
autotest Documentation

Release 0.16.3-30-ga33d

Autotest Team

Aug 29, 2017

1	Autotest Documentation	3
1.1	General Information	3
1.2	Local (Former Client)	20
1.3	Remote (Former Server)	41
1.4	Frontend	70
1.5	System Administration	124
1.6	Scheduler	144
1.7	Developer	153
2	client Package	193
2.1	autotest_local Module	193
2.2	base_sysinfo Module	193
2.3	base_utils Module	194
2.4	bkr_proxy Module	198
2.5	bkr_xml Module	200
2.6	client_logging_config Module	201
2.7	cmdparser Module	201
2.8	common Module	202
2.9	config Module	202
2.10	cpuset Module	202
2.11	fsdev_disks Module	204
2.12	fsdev_mgr Module	206
2.13	fsinfo Module	206
2.14	harness Module	207
2.15	harness_autoserv Module	208
2.16	harness_beaker Module	208
2.17	harness_simple Module	209
2.18	harness_standalone Module	210
2.19	job Module	210
2.20	kernel Module	213
2.21	kernel_config Module	216
2.22	kernel_versions Module	216
2.23	kernelexpand Module	217
2.24	kvm_control Module	217
2.25	local_host Module	218
2.26	lv_utils Module	218

2.27	optparser Module	219
2.28	os_dep Module	219
2.29	parallel Module	223
2.30	partition Module	223
2.31	profiler Module	228
2.32	setup Module	228
2.33	setup_job Module	228
2.34	setup_modules Module	229
2.35	sysinfo Module	229
2.36	test Module	229
2.37	test_config Module	230
2.38	utils Module	230
2.39	xen Module	231
2.40	Subpackages	231
3	frontend Package	363
3.1	Subpackages	363
4	Indices and tables	365
	Python Module Index	367

Autotest is a framework for fully automated testing. It is designed primarily to test the Linux kernel, though it is useful for many other purposes such as qualifying new hardware, virtualization testing and other general user space program testing under linux platforms. It's an open-source project under the GPL and is used and developed by a number of organizations, including Google, IBM, Red Hat, and many others.

Please check Avocado, a next generation test automation framework being developed by several of the original Autotest team members: <http://avocado-framework.github.io/>

General Information

Contact information

- [Autotest mailing list](#)
- Autotest IRC channel: `irc.oftc.net #autotest`

Who uses autotest?

An active community of users is fundamental to sustain an open source project. Currently, there are quite a few projects that use autotest as their main test automation platform:

- Fedora's [AutoQA](#)
- [Chrome OS](#)
- [KVM kernel based virtual machine](#)
- Goobuntu (internal customized version of ubuntu for google)

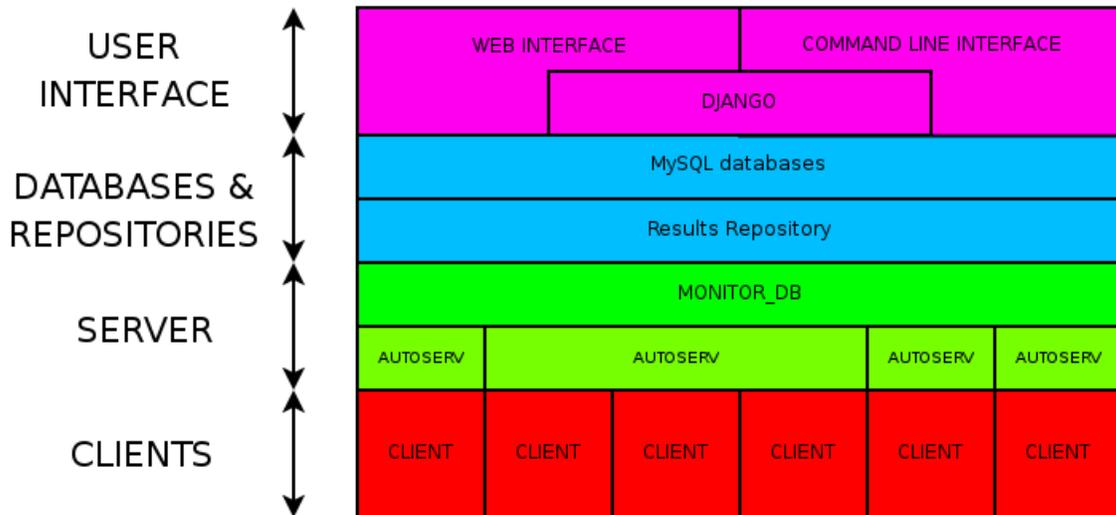
Besides these projects, other people from several different companies/affiliations use autotest for their test automation needs.

Autotest structure overview

This document intends to be a high level overview of the autotest project structure. We try to be brief and show the high level diagrams.

Simplified block diagram

For the sake of clarity, some things are simplified here, but it gives you a good idea of the overall layout.



Web interface and command line interface

The web interface and the command line interface are complementary ways to interact with autotest and create jobs. Both were designed to have the same functionality, to add to the user's convenience. The interfaces allows you to:

- Manage jobs? - create, monitor, abort, etc.
- Manage client hosts
- Look at results.

The frontends will inject jobs into the server by creating records in a mysql database.

Server

The server consists of three main parts:

- A mysql database that holds information on all jobs, clients (test machines), users and tests.
- The dispatcher (monitor_db) - chooses the jobs from the database to run. It's input is the database, pretty much all it does is start autoserv processes to service requests.
 - There is normally one dispatcher process per machine
 - Client side jobs are run asynchronously (as client machines become available)
 - Server side jobs are run synchronously (ie we wait for all clients before commencing)
- Autoserv: the server manages clients via autoserv processes - there will be one autoserv process per running job?. Each autoserv process:
 - controls and monitors one or more clients
 - verifies clients are working properly, and if it fails verification, attempts to repair it
 - manages the execution of a job?
 - updates the autotest software on each client before commencing work.

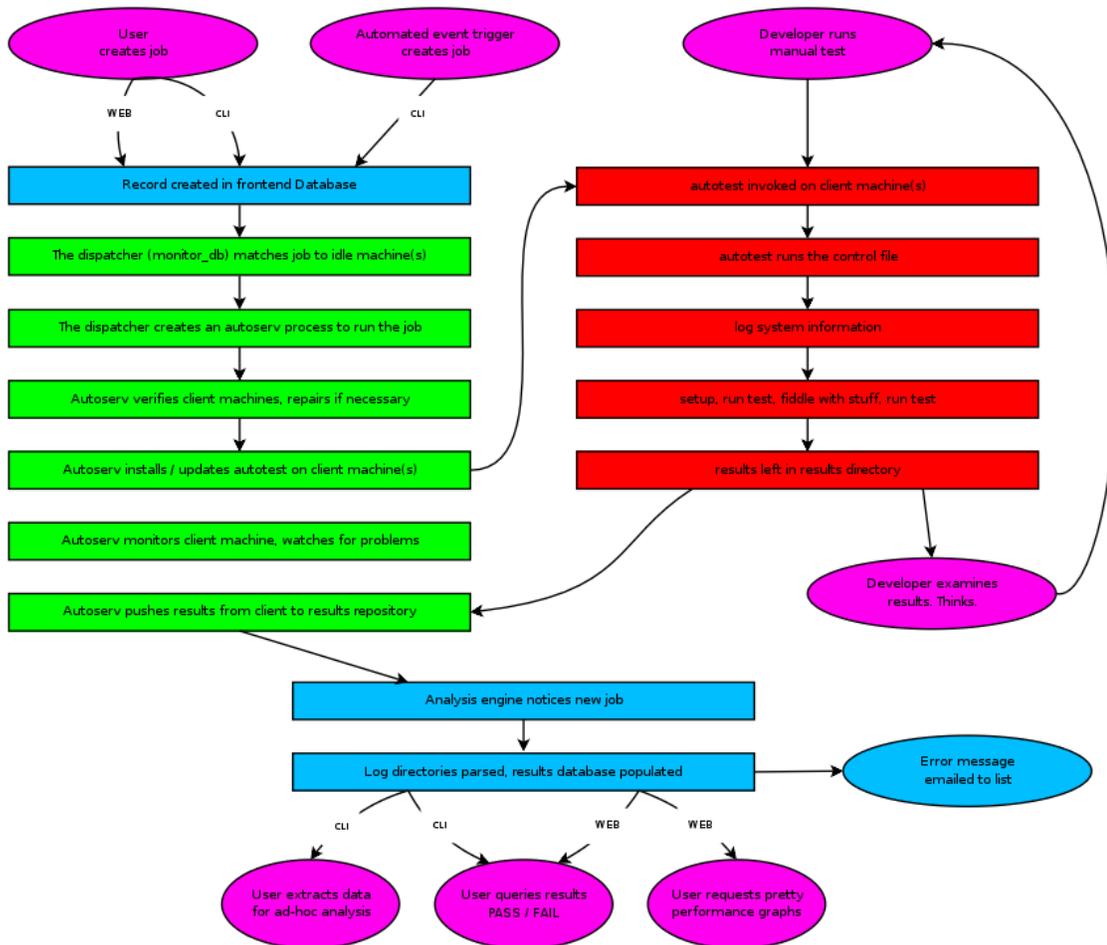
The mysql database can live on a different machine than the dispatcher. There can be multiple dispatchers to spread the workload, though each can service a few thousand clients, so this is not normally necessary.

Client

The client does most of the work of running a job?; this can be invoked:

- manually - from `client/autotest-local <control_file_name>`
- via the server

A typical job workflow is as follows:



Results repository

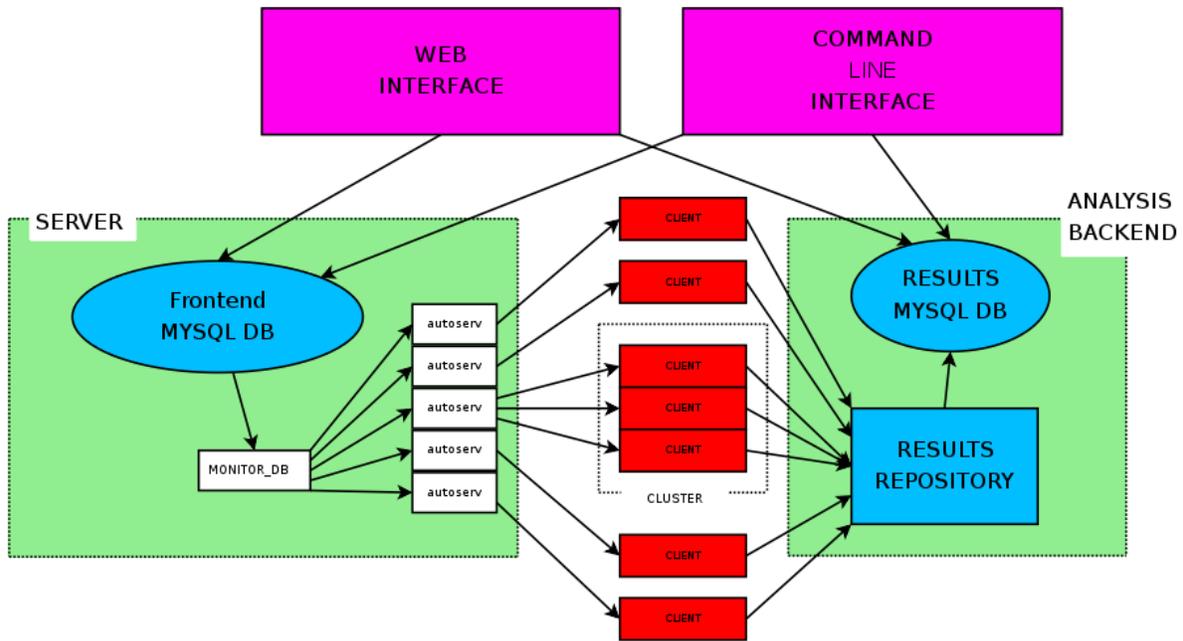
A directory tree of all the results. Each job has a well formatted *directory structure*

Results MySQL DB

A simple mysql database containing the jobs, test results, and performance metrics for each test

Overall structure

With all the parts of the code briefly commented, it's easier to understand the overall structure diagram:



Autotest White Paper

Abstract

This paper describes the motivation for, and design goals of, the autotest and test.kernel.org projects. Autotest is a framework for fully automated testing, that is designed primarily to test the Linux kernel, though is useful for many other functions too. Test.kernel.org is a framework for communicating, sharing, and analysing test results.

In a traditional corporate systems software development environment, there is normally a large test team responsible for assuring the quality of the final product. Open source projects do not have that luxury, and we need to find another way to run testing. We feel that the only realistic way to achieve the goal is to fully automate the test process, and drastically reduce the need for human staffing. It turns out that this also solves several other critical problems with testing.

- Consistency - it's much easier to guarantee the tests are run the same way as last time.
- Knowledge capture - the knowledge of how to run testing is not held in one person, but within a system.
- Sharing - you can easily share tests with vendors, partners, and across a wide community.
- Reproducibility - they say 90% of fixing a bug is to get an easily reproducible test case.

Testing is not about running tests ... testing is about finding and fixing bugs. We have to:

- Run the tests
- Find a bug
- Classify the bug

- Hand the bugs off to a developer
- Developer investigates bug (cyclical)
- Developer tests some proposed fix (cyclical)
- Fix checked in
- New release issued to test team.

So many test systems I see are oriented only around the first two (or even one!) steps. This is massively inefficient - so often I see developers writing a simple testcase to reproduce what happens in a more complex test, or proprietary application, and then these are thrown away. If we started with open tests that we could freely and easily share, much effort and time would be saved. This is not just about the cost of the people's time, salaries and machine resources. It's about the opportunity cost of stalling a release, which is massively greater - these problems are often single-threading the critical path.

We want bug identification, investigation, and fixing to be done earlier in the cycle. This allows multiple debugging efforts to be done in parallel, without affecting others, as well as many other advantages, such as the problem still being fresh in the developers mind, and not interacting with other later changes. This means running tests on multiple codebases (development trees), with high frequency - how can we scale to this? Fully automated testing. Machines are cheap, people are expensive - this is the reality of the modern age. For Linux, the problem is compounded by the staggering diversity of hardware and kernel configurations that we support.

Moreover, a test system is not just about simple functional tests; we should also identify performance regressions and trends, adding statistical analysis. A broad spectrum of tests are necessary - boot testing, regression, function, performance, and stress testing; from disk intensive to compute intensive to network intensive loads. A fully automated test harness also empowers other techniques that are impractical when testing manually, in order to make debugging and problem identification easier, e.g. automated binary chop search amongst thousands of patches to weed out dysfunctional changes.

It's critical that when operating in an open community, we can share and compare test results - that necessitates consistency of results formats. The easiest way to achieve this is to share one common test harness.

Introduction

It is critical for any project to maintain a high level of software quality, and consistent interfaces to other software that it uses or uses it. There are several methods for increasing quality, but none of these works in isolation, we need a combination of:

- skilled developers carefully developing high quality code,
- static code analysis,
- regular and rigorous code review,
- functional tests for new features,
- regression testing,
- performance testing, and
- stress testing.

Whilst testing will never catch all bugs, it will improve the overall quality of the finished product. Improved code quality results in a better experience not only for users, but also for developers, allowing them to focus on their own code. Even simple compile errors hinder developers.

In this paper we will look at the problem of automated testing, the current state of it, and our views for its future. Then we will take a case study of the test.kernel.org automated test system. We will examine a key test component, the client harness, in more detail, and describe the Autotest test harness project. Finally we will conclude with our vision of the future and a summary.

Automated Testing

It is obvious that testing is critical, what is perhaps not so obvious is the utility of regular testing at all stages of development. There are two main things we're trying to achieve here, parallelism of work, and catching the bugs as quickly as possible. These are critical as:

- it prevents replication of the bad code into other code bases,
- fewer users are exposed to the bug,
- the code is still fresh in the authors mind,
- the change is less likely to interact with subsequent changes, and
- the code is easy to remove should that be required.

In a perfect world all contributions would be widely tested before being applied; however, as most developers do not have access to a large range of hardware this is impractical. More reasonably we want to ensure that any code change is tested before being introduced into the mainline tree, and fixed or removed before most people will ever see it. In the case of Linux, Andrew Morton's -mm tree (the de facto development tree) and other subsystem specific trees are good testing grounds for this purpose.

Test early, test often!

The open source development model and Linux in particular introduces some particular challenges. Open-source projects generally suffer from the lack of a mandate to test submissions and the fact that there is no easy funding model for regular testing. Linux is particularly hard hit as it has a constantly high rate of change, compounded with the staggering diversity of the hardware on which it runs. It is completely infeasible to do this kind of testing without extensive automation.

There is hope; machine-power is significantly cheaper than man-power in the general case. Given a large quantity of testers with diverse hardware it should be possible to cover a useful subset of the possible combinations. Linux as a project has plenty of people and hardware; what is needed is a framework to coordinate this effort.

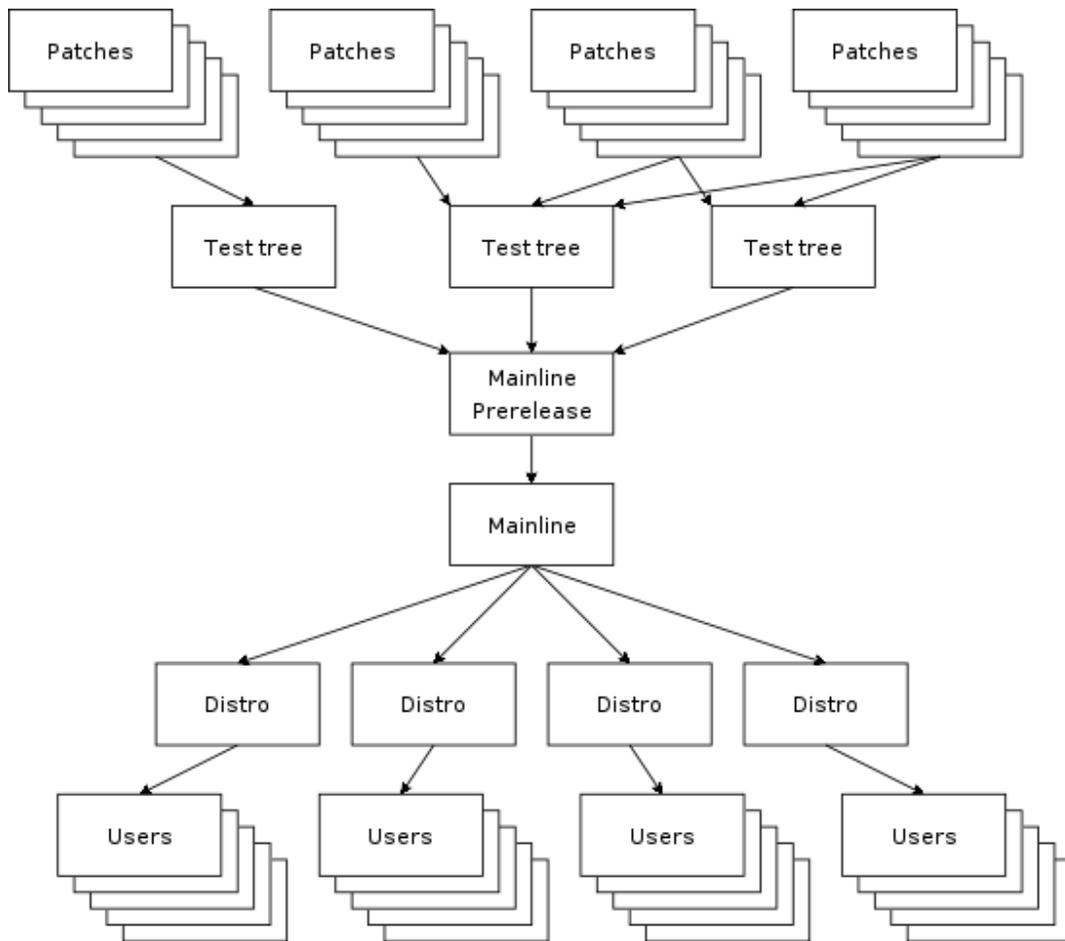
The Testing Problem

As we can see from the diagram above Linux's development model forms an hourglass starting highly distributed, with contributions being concentrated in maintainer trees before merging into the development releases (the -mm tree) and then into mainline itself. It is vital to catch problems here in the neck of the hourglass, before they spread out to the distros – even once a contribution hits mainline it is has not yet reached the general user population, most of whom are running distro kernels which often lag mainline by many months.

In the Linux development model, each actual change is usually small and attribution for each change is known making it easy to track the author once a problem is identified. It is clear that the earlier in the process we can identify there is a problem, the less the impact the change will have, and the more targeted we can be in reporting and fixing the problem.

Whilst contributing untested code is discouraged we cannot expect lone developers to be able to do much more than basic functional testing, they are unlikely to have access to a wide range of systems. As a result, there is an opportunity for others to run a variety of tests on incoming changes before they are widely distributed. Where problems are identified and flagged, the community has been effective at getting the change rejected or corrected.

By making it easier to test code, we can encourage developers to run the tests before ever submitting the patch; currently such early testing is often not extensive or rigorous, where it is performed at all. Much developer effort is being wasted on bugs that are found later in the cycle when it is significantly less efficient to fix them.



The State of the Union

It is clear that a significant amount of testing resource is being applied by a variety of parties, however most of the current testing effort goes on after the code has forked from mainline. The distribution vendors test the code that they integrate into their releases, hardware vendors are testing alpha or beta releases of those distros with their hardware. Independent Software Vendors (ISVs) are often even later in the cycle, first testing beta or even after distro release. Whilst integration testing is always valuable, this is far too late to be doing primary testing, and makes it extremely difficult and inefficient to fix problems that are found. Moreover, neither the tests that are run, nor the results of this testing are easily shared and communicated to the wider community.

There is currently a large delay between a mainline kernel releasing and that kernel being accepted and released by the distros, embedded product companies and other derivatives of Linux. If we can improve the code quality of the mainline tree by putting more effort into testing mainline earlier, it seems reasonable to assume that those ‘customers’ of Linux would update from the mainline tree more often. This will result in less time being wasted porting changes backwards and forwards between releases, and a more efficient and tightly integrated Linux community.

What Should we be Doing?

Linux’s constant evolutionary approach to software development fits well with a wide-ranging, high-frequency regression testing regime. The ‘release early, release often’ development philosophy provides us with a constant stream of test candidates; for example the -git snapshots which are produced twice daily, and Andrew Morton’s collecting of the specialized maintainer trees into a bleeding-edge -mm development tree.

In an ideal world we would be regression testing at least daily snapshots of all development trees, the -mm tree and mainline on all possible combinations of hardware; feeding the results back to the owners of the trees and the authors of the changes. This would enable problems to be identified as early as possible in the concentration process and get the offending change updated or rejected. The test.kernel.org testing project provides a preview of what is possible, providing some limited testing of the mainline and development trees, and is discussed more fully later.

Just running the tests is not sufficient, all this does is produce large swaths of data for humans to wade through; we need to analyse the results to engender meaning, and isolate any problems identified.

Regression tests are relatively easy to analyse, they generate a clean pass or fail; however, even these can fail intermittently. Performance tests are harder to analyse, a result of 10 has no particular meaning without a baseline to compare it against. Moreover, performance tests are not 100% consistent, so taking a single sample is not sufficient, we need to capture a number of runs and do simple statistical analysis on the results in order to determine if any differences are statistically significant or not. It is also critically important to try to distinguish failures of the machine or harness from failures of the code under test.

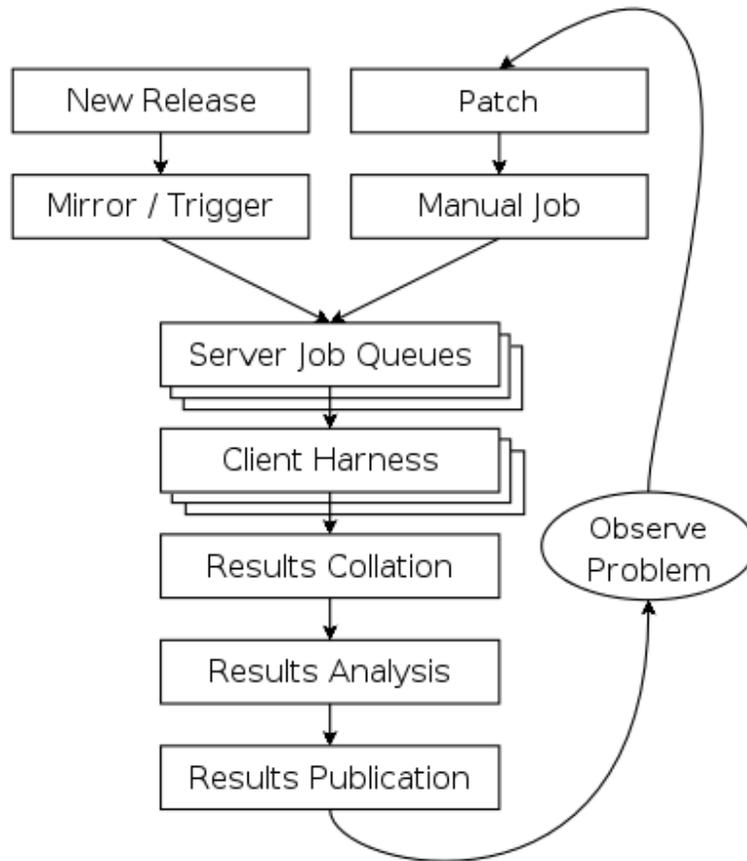
Case Study: test.kernel.org

We have tried to take the first steps towards the automated testing goals we have outlined above with the testing system that generates the test.kernel.org website. Whilst it is still far from what we would like to achieve, it is a good example of what can be produced utilising time on an existing in house system sharing and testing harness and a shared results repository.

New kernel releases are picked up automatically within a few minutes of release, and a predefined set of tests are run across them by a proprietary IBM® system called ABAT, which includes a client harness called autobench. The results of these tests are then collated, and pushed to the TKO server, where they are analysed and the results published on the TKO website.

Whilst all of the test code is not currently open, the results of the testing are, which provides a valuable service to the community, indicating (at least at a gross level) a feel for the viability of that release across a range of existing machines, and the identification of some specific problems. Feedback is in the order of hours from release to results publication.

How it Works



The TKO system is architected as show in the figure above. Its is made up of a number of distinct parts, each described below:

The mirror / trigger engine: test execution is keyed from kernel releases; by any -mm tree release (2.6.16-rc1-mm1), git release (2.6.17-rc1-git10), release candidate (2.6.17-rc1), stable release (2.6.16) or stable patch release (2.6.16.1). A simple rsync local mirror is leveraged to obtain these images as soon as they are available. At the completion of the mirroring process any newly downloaded image is identified and those which represent new kernels trigger testing of that image.

Server Job Queues: for each new kernel, a predefined set of test jobs are created in the server job queues. These are interspersed with other user jobs, and are run when time is available on the test machines. IBM's ABAT server software currently fulfils this function, but a simple queueing system could serve for the needs of this project.

Client Harness: when the test system is available, the control file for that test is passed to the client harness. This is responsible for setting up the machine with appropriate kernel version, running the tests, and pushing the results to a local repository. Currently this function is served by autobench. It is here that our efforts are currently focused with the Autotest client replacement project which we will discuss in detail later.

Results Collation: results from relevant jobs are gathered asynchronously as the tests complete and they are pushed out to test.kernel.org. A reasonably sized subset of the result data is pushed, mostly this involves stripping the kernel binaries and system information dumps.

Results Analysis: once uploaded the results analysis engine runs over all existing jobs and extracts the relevant status; this is then summarised on a per release basis to produce both overall red, amber and green status for each

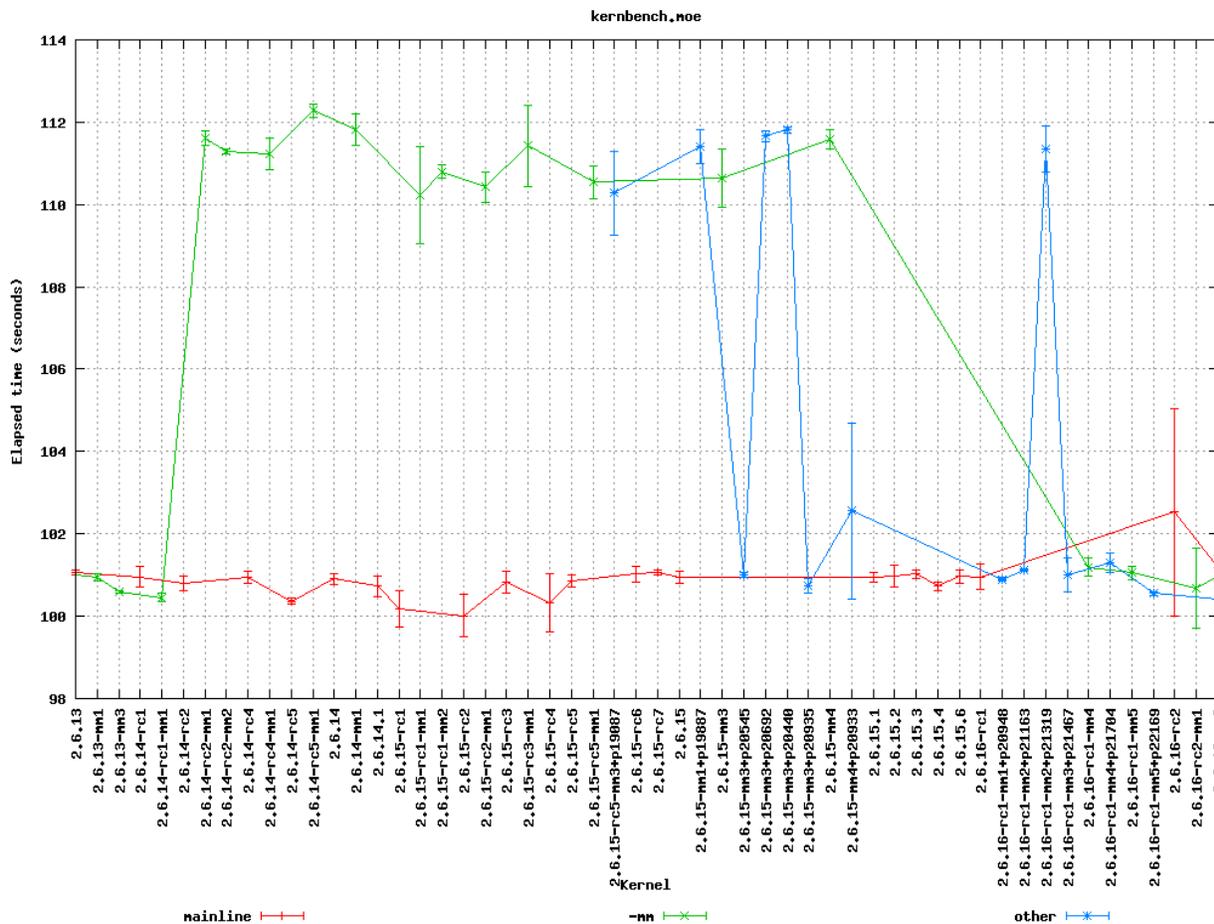
release/machine combination. Performance data is also analysed, in order to produce historical performance graphs for a selection of benchmarks.

Results Publication: results are made available automatically on the TKO web site. However, this is currently a ‘polled’ model; no automatic action is taken in the face of either test failures or if performance regressions are detected, it relies on developers to monitor the site. These failures should be actively pushed back to the community via an appropriate publication mechanism (such as email, with links back to more detailed data).

Observed problems: When a problem (functional or performance) is observed by a developer monitoring the analysed and published results, this is manually communicated back to the development community (normally via email). This often results in additional patches to test, which can be manually injected into the job queues via a simple script, but currently only by an IBM engineer. These then automatically flow through with the regular releases, right through to publication on the matrix and performance graphs allowing comparison with those releases.

TKO in Action

The regular compile and boot testing frequently shakes out bugs as the patch that carried them enters the -mm tree. By testing multiple architectures, physical configurations, and kernel configurations we often catch untested combinations and are able to report them to the patch author. Most often these are compile failures, or boot failures, but several performance regressions have also been identified.



As a direct example, recently the performance of highly parallel workloads dropped off significantly on some types of systems, specifically with the -mm tree. This was clearly indicated by a drop off in the kernbench performance figures. In the graph above we can see the sudden increase in elapsed time to a new plateau with 2.6.14-rc2-mm1.

Note the vertical error bars for each data point – doing multiple test runs inside the same job allows us to calculate error margins, and clearly display them.

Once the problem was identified some further analysis narrowed the bug to a small number of scheduler patches which were then also tested; these appear as the blue line (‘other’ releases) in the graph. Once the regression was identified the patch owner was then contacted, several iterations of updated fixes were then produced and tested before a corrected patch was applied. This can be seen in the figures for 2.6.16-rc1-mm4.

The key thing to note here is that the regression never made it to the mainline kernel let alone into a released distro kernel; user exposure was prevented. Early testing ensured that the developer was still available and retained context on the change.

Summary

The current system is providing regular and useful testing feedback on new releases and providing ongoing trend analysis against historical releases. It is providing the results of this testing in a public framework available to all developers with a reasonable turn round time from release. It is also helping developers by testing on rarer hardware combinations to which they have no access and cannot test.

However, the system is not without its problems. The underlying tests are run on a in-house testing framework (ABAT) which is currently not in the public domain; this prevents easy transport of these tests to other testers. As a result there is only one contributor to the result set at this time, IBM. Whilst the whole stack needs to be made open, we explain in the next section why we have chosen to start first with the client test harness.

The tests themselves are very limited, covering a subset of the kernel. They are run on a small number of machines, each with a few, fixed configurations. There are more tests which should be run but lack of developer input and lack of hardware resources on which to test prevent significant expansion.

The results analysis also does not communicate data back as effectively as it could to the community – problems (especially performance regressions) are not as clearly isolated as they could be, and notification is not as prompt and clear as it could be. More data ‘folding’ needs to be done as we analyse across a multi-dimensional space of kernel version, kernel configuration, machine type, toolchain, and tests.

Client Harnesses

As we have seen, any system which will provide the required level of testing needs to form a highly distributed system, and be able to run across a large test system base. This will necessitate a highly flexible client test harness; a key component of such a system. We have used our experiences with the IBM autobench client, and the TKO analysis system to define requirements for such a client. This section will discuss client harnesses in general and lead on to a discussion of the Autotest project’s new test harness.

We chose to attack the problem of the client harness first as it seems to be the most pressing issue. With this solved, we can share not only results, but the tests themselves more easily, and empower a wide range of individuals and corporations to run tests easily, and share the results. By defining a consistent results format, we can enable automated collation and analysis of huge amounts of data.

Requirements / Design Goals

A viable client harness must be operable stand-alone or under an external scheduler infrastructure. Corporations already have significant resources invested in bespoke testing harnesses which they are not going to be willing to waste; the client needs to be able to plug into those, and timeshare resources with them. On the other hand, some testers and developers will have a single machine and want something simple they can install and use. This bimodal flexibility is particularly relevant where we want to be able to pass a failing test back to a patch author, and have them reproduce the problem.

The client harness must be modular, with a clean internal infrastructure with simple, well defined APIs. It is critical that there is clear separation between tests, and between tests and the core, such that adding a new test cannot break existing tests.

The client must be simple to use for newcomers, and yet provide a powerful syntax for complex testing if necessary. Tests across multiple machines, rebooting, loops, and parallelism all need to be supported.

We want distributed scalable maintainership, the core being maintained by a core team and the tests by the contributors. It must be able to reuse the effort that has gone into developing existing tests, by providing a simple way to encapsulate them. Whilst open tests are obviously superior, we also need to allow the running of proprietary tests which cannot be contributed to the central repository.

There must be a low knowledge barrier to entry for development, in order to encourage a wide variety of new developers to start contributing. In particular, we desire it to be easy to write new tests and profilers, abstracting the complexity into the core as much as possible.

We require a high level of maintainability. We want a consistent language throughout, one which is powerful and yet easy to understand when returning to the code later, not only by the author, but also by other developers.

The client must be robust, and produce consistent results. Error handling is critical – tests that do not produce reliable results are useless. Developers will never add sufficient error checking into scripts, we must have a system which fails on any error unless you take affirmative action. Where possible it should isolate hardware or harness failures from failures of the code under test; if something goes wrong in initialisation or during a test we need to know and reject that test result.

Finally, we want a consistent results architecture – it is no use to run thousands of tests if we cannot understand or parse the results. On such a scale such analysis must be fully automatable. Any results structure needs to be consistent across tests and across machines, even if the tests are being run by a wide diversity of testers.

What Tests are Needed?

As we mentioned previously, the current published automated testing is very limited in its scope. We need very broad testing coverage if we are going to catch a high proportion of problems before they reach the user population, and need those tests to be freely sharable to maximise test coverage.

Most of the current testing is performed in order to verify that the machine and OS stack is fit for a particular workload. The real workload is often difficult to set up, may require proprietary software, and is overly complex and does not give sufficiently consistent reproducible results, so use is made of a simplified simulation of that workload encapsulated within a test. This has the advantage of allowing these simulated workloads to be shared. We need tests in all of the areas below:

Build tests simply check that the kernel will build. Given the massive diversity of different architectures to build for, different configuration options to build for, and different toolchains to build with, this is an extensive problem. We need to check for warnings, as well as errors.

Static verification tests run static analysis across the code with tools like sparse, lint, and the Stanford checker, in the hope of finding bugs in the code without having to actually execute it.

Inbuilt debugging options (e.g. `CONFIG_DEBUG_PAGEALLOC`, `CONFIG_DEBUG_SLAB`) and fault insertion routines (e.g. fail every 100th memory allocation, fake a disk error occasionally) offer the opportunity to allow the kernel to test itself. These need to be a separated set of test runs from the normal functional and performance tests, though they may reuse the same tests.

Functional or unit tests are designed to exercise one specific piece of functionality. They are used to test that piece in isolation to ensure it meets some specification for its expected operation. Examples of this kind of test include LTP and Crashme.

Performance tests verify the relative performance of a particular workload on a specific system. They are used to produce comparisons between tests to either identify performance changes, or confirm none is present. Examples

of these include: CPU performance with Kernbench and AIM7/ream; disk performance with bonnie, tbench and iobench; and network performance with netperf.

Stress tests are used to identify system behaviour when pushed to the very limits of its capabilities. For example a kernel compile executed completely in parallel creates a compile process for each file. Examples of this kind of test include kernbench (configured appropriately), and deliberately running under heavy memory pressure such as running with a small physical memory.

Profiling and debugging is another key area. If we can identify a performance regression, or some types of functional regression, it is important for us to be able to gather data about what the system was doing at the time in order to diagnose it. Profilers range from statistical tools like readprofile and lockmeter to monitoring tools like vmstat and sar. Debug tools might range from dumping out small pieces of information to full blown crashdumps.

Existing Client Harnesses

There are a number of pre-existing test harnesses in use by testers in the community. Each has its features and problems, we touch on a few of them below.

IBM autobench is a fairly fully featured client harness, it is completely written in a combination of shell and perl. It has support for tests containing kernel builds and system boots. However, error handling is very complex and must be explicitly added in all cases, but does encapsulate the success or failure state of the test. The use of multiple different languages may have been very efficient for the original author, but greatly increases the maintenance overheads. Whilst it does support running multiple tests in parallel, loops within the job control file are not supported nor is any complex 'programming'.

OSDL STP The Open Systems Development Lab (OSDL) has the Scalable Test Platform (STP). This is a fully integrated testing environment with both a server harness and client wrapper. The client wrapper here is very simple consisting of a number of shell support functions. Support for reboot is minimal and kernel installation is not part of the client. There is no inbuilt handling of the meaning of results. Error checking is down to the test writer; as this is shell it needs to be explicit else no checking is performed. It can operate in isolation and results are emailable, reboot is currently being added.

LTP (<http://ltp.sourceforge.net/>) The Linux Test Project is a functional / regression test suite. It contains approximately 2900 small regression tests which are applied to the system running LTP. There is no support for building kernels or booting them, performance testing or profiling. Whilst it contains a lot of useful tests, it is not a general heavy weight testing client.

A number of other testing environments currently exist, most appear to suffer from the same basic issues, they evolved from the simplest possible interface (a script) into a test suite; they were not designed to meet the level of requirements we have identified and specified.

All of those we have reviewed seem to have a number of key failings. Firstly, most lack most lack bottom up error handling. Where support exists it must be handled explicitly, testers never will think of everything. Secondly, most lack consistent machine parsable results. There is often no consistent way to tell if a test passes, let alone get any details from it. Lastly, due to their evolved nature they are not easy to understand nor to maintain. Fortunately it should be reasonably easy to wrap tests such as LTP, or to port tests from STP and autobench.

Autotest - a Powerful Open Client

The Autotest open client is an attempt to address the issues we have identified. The aim is to produce a client which is open source, implicitly handles errors, produces consistent results, is easily installable, simple to maintain and runs either standalone or within any server harness.

Autotest is an all new client harness implementation. It is completely written in Python; chosen for a number of reasons, it has a simple, clean and consistent syntax, it is object oriented from inception, and it has very powerful error and exception handling. Whilst no language is perfect, it meets the key design goals well, and it is open source and widely supported.

As we have already indicated, there are a number of existing client harnesses; some are even open-source and therefore a possible basis for a new client. Starting from scratch is a bold step, but we believe that the benefits from a designed approach outweigh the effort required initially to get to a workable position. Moreover, much of the existing collection of tests can easily be imported or wrapped.

Another key goal is the portability of the tests and the results; we want to be able to run tests anywhere and to contribute those test results back. The use of a common programming language, one with a strict syntax and semantics should make the harness and its contained tests very portable. Good design of the harness and results specifications should help to maintain portable results.

The autotest Test Harness

Autotest utilises an executable control file to represent and drives the users job. This control file is an executable fragment of Python and may contain any valid Python constructs, allowing the simple representation of loops and conditionals. Surrounding this control file is the Autotest harness, which is a set of support functions and classes to simplify execution of tests and allow control over the job.

The key component is the job object which represents the executing job, provides access to the test environment, and provides the framework to the job. It is responsible for the creation of the results directory, for ensuring the job output is recorded, and for any interactions with any server harness. Below is a trivial example of a control file:

```
job.runtest('test1', 'kernbench', 2, 5)
```

One key benefit of the use of a real programming language is the ability to use the full range of its control structures in the example below we use an iterator:

```
for i in range(0, 5):
    job.runtest('test%d' % i, 'kernbench',
                2, 5)
```

Obviously as we are interested in testing Linux, support for building, installing and booting kernels is key. When using this feature, we need a little added complexity to cope with the interruption to control flow caused by the system reboot. This is handled using a phase stepper which maintains flow across execution interruptions, below is an example of such a job, combining booting with iteration:

```
def step_init():
    step_test(1)

def step_test(iteration):
    if (iteration < 5):
        job.next_step([step_test,
                       iteration + 1])

    print "boot: %d" % iteration

    kernel = job.distro_kernel()
    kernel.boot()
```

Tests are represented by the test object; each test added to Autotest will be a subclass of this. This allows all tests to share behaviour, such as creating a consistent location and layout for the results, and recording the result of the test in a computer readable form. Below is the class definition for the kernbench benchmark. As we can see it is a subclass of test, and as such benefits from its management of the results directory hierarchy.

```
import test
from autotest_utils import *
```

```

class kernbench(test):

    def setup(self,
              iterations = 1,
              threads = 2 * count_cpus(),
              kernelver = '/usr/local/src/linux-2.6.14.tar.bz2',
              config = os.environ['AUTODIRBIN'] + "/tests/kernbench/config"):

        print "kernbench -j %d -i %d -c %s -k %s" % (threads, iterations, config, ↵
↵kernelver)

        self.iterations = iterations
        self.threads = threads
        self.kernelver = kernelver
        self.config = config

        top_dir = job.tmpdir+'/kernbench'
        kernel = job.kernel(top_dir, kernelver)
        kernel.config([config])

    def execute(self):
        testkernel.build_timed(threads)          # warmup run
        for i in range(1, iterations+1):
            testkernel.build_timed(threads, '../log/time.%d' % i)

        os.chdir(top_dir + '/log')
        system("grep elapsed time.* > time")

```

Summary

We feel that Autotest is much more powerful and robust design than the other client harnesses available, and will produce more consistent results. Adding tests and profilers is simple, with a low barrier to entry, and they are easy to understand and maintain.

Much of the power and flexibility of Autotest stems from the decision to have a user-defined control file, and for that file to be written in a powerful scripting language. Whilst this was more difficult to implement, the interface the user sees is still simple. If the user wishes to repeat tests, run tests in parallel for stress, or even write a bisection search for a problem inside the control file, that is easy to do.

The Autotest client can be used either as standalone, or easily linked into any scheduling backend, from a simple queueing system to a huge corporate scheduling and allocation engine. This allows us to leverage the resources of larger players, and yet easily allow individual developers to reproduce and debug problems that were found in the lab of a large corporation.

Each test is a self-contained modular package. Users are strongly encouraged to create open-source tests (or wrap existing tests) and contribute those to the main test repository on test.kernel.org (see the Autotest wiki for details). However, private tests and repositories are also allowed, for maximum flexibility. The modularity of the tests means that different maintainers can own and maintain each test, separate from the core harness. We feel this is critical to the flexibility and scalability of the project.

We currently plan to support the Autotest client across the range of architectures and across the main distros. There is no plans to support other operating systems, as it would add unnecessary complexity to the project. The Autotest project is released under the GNU Public License.

Future

We need a broader spectrum of tests added to the Autotest project. Whilst the initial goal is to replace autobench for the published data on test.kernel.org, this is only a first step – there are a much wider range of tests that could and should be run. There is a wide body of tests already available that could be wrapped and corralled under the Autotest client.

We need to encourage multiple different entities to contribute and share testing data for maximum effect. This has been stalled waiting on the Autotest project, which is now nearing release, so that we can have a consistent data format to share and analyse. There will be problems to tackle with quality and consistency of data that comes from a wide range of sources.

Better analysis of the test results is needed. Whilst the simple red/yellow/green grid on test.kernel.org and simple gnuplot graphs are surprisingly effective for so little effort, much more could be done. As we run more tests, it will become increasingly important to summarise and fold the data in different ways in order to make it digestible and useful.

Testing cannot be an island unto itself – not only must we identify problems, we must communicate those problems effectively and efficiently back to the development community, provide them with more information upon request, and be able to help test attempted fixes. We must also track issues identified to closure.

There is great potential to automate beyond just identifying a problem. An intelligent automation system should be able to further narrow down the problem to an individual patch (by bisection search, for example, which is $O(\log 2)$ number of patches). It could drill down into a problem by running more detailed sets of performance tests, or repeating a failed test several times to see if a failure was intermittent or consistent. Tests could be selected automatically based on the area of code the patch touches, correlated with known code coverage data for particular tests.

Summary

We are both kernel developers, who started the both test.kernel.org and Autotest projects out of a frustration with the current tools available for testing, and for fully automated testing in particular. We are now seeing a wider range of individuals and corporations showing interest in both the test.kernel.org and Autotest projects, and have high hopes for their future.

In short we need:

- more automated testing, run at frequent intervals,
- those results need to be published consistently and cohesively,
- to analyse the results carefully,
- better tests, and to share them, and
- a powerful, open source, test harness that is easy to add tests to.

There are several important areas where interested people can help contribute to the project:

- run a diversity of tests across a broad range of hardware,
- contribute those results back to test.kernel.org,
- write new tests and profilers, contribute those back, and
- for the kernel developers ... fix the bugs!!!

An intelligent system can not only improve code quality, but also free developers to do more creative work.

Acknowledgements

We would like to thank OSU for the donation of the server and disk space which supports the test.kernel.org site.

We would like to thank Mel Gorman for his input to and review of drafts of this paper.

Autotest Development Community Size

After re-consideration about the subject, in April 2012 we have rewritten the entire autotest tree history. Autotest was a project kept on svn for about 6 years, and for a long time there was an unofficial git-svn mirror, that after we adopted git as the official reference, we just kept that mirror.

Obviously this does not play well with the traditional tools to verify stats on git, so that's why we decided to rewrite. Now you can see the individual authors that contributed since the inception of the project:

```
$ git shortlog -s | wc -l
202
```

And all other fun git statistics, such as the number of organizations that contributed resources to some extent to the project

```
$ git shortlog -se | sed -e 's/.*@//g' -e 's/\W*$//g' | sort | uniq | grep -v "<"
alien8.de
amd.com
bl-systems.de
br.ibm.com
canonical.com
chromium.org
cn.fujitsu.com
cn.ibm.com
digium.com
gelato.unsw.edu.au
gmail.com
google.com
hp.com
ifup.org
inf.u-szeged.hu
in.ibm.com
intel.com
intra2net.com
kerlabs.com
linux.vnet.ibm.com
mvista.com
nokia.com
openvz.org
oracle.com
osdl.org
oss.ntt.co.jp
place.org
raisama.net
redhat.com
samba.org
secretlab.ca
shadowen.org
stanford.edu
stec-inc.com
suse.com
```

```
suse.cz
suse.de
twitter.com
uk.ibm.com
us.ibm.com
windriver.com
xenotime.net
```

As not all of them are strictly institutions, and there are different domains from the same root company, we can estimate about 30 institutions.

Have fun with your git stats, enjoy!

This section has assorted information and some past, informative presentations and articles about autotest architecture and goals, such as [This article on autotest by John Admanski](#) and [This presentation from OLS 2009](#).

Local (Former Client)

Autotest Client Quick Start

The autotest client has few requirements. Make sure you have python 2.4 or later installed. Also, it is a good idea to try things in a VM or test machine you don't care about, for safety.

Download the client wherever you see fit:

```
git clone --recursive git://github.com/autotest/autotest.git
cd autotest
```

Run some simple tests, such our `sleeptest`, which only sleeps for a given amount of seconds (our favorite autotest sanity testing). From the autotest directory (i.e. `/usr/local/autotest/client`):

```
client/autotest-local --verbose run sleeptest
```

To run any individual test:

```
client/autotest-local run <testname>
```

You can also run tests by providing the control file

```
client/autotest-local client/tests/sleeptest/control
```

Some tests may require that you run them as root. For example, if you try to run the `rtc` test as normal user, you will get `/dev/rtc0: Permission denied error` in your test result. So you must run the test as root.

In case you run the client as root, then switch back to a regular user, some important directories will be owned by root and the next run will fail. If that happens, you can remove the directories:

```
sudo rm -rf client/tmp
sudo rm -rf client/results
```

There are sample control files inside the `client/samples` directory, useful for learning from. The `kbuild_and_tests/control` file in there will download a kernel, compile it, then reboot the machine into it.

Execute it as root:

```
client/autotest-local --verbose client/samples/kbuild_and_tests/control
```

WARNING - do it on a test machine, or in a VM, so you don't mess up your existing system boot configuration

Client Control files

The key defining component of a job is its *control file*; this file defines all aspects of a jobs life cycle. The control file is a Python script which directly drives execution of the tests in question.

Simple Jobs

You are automatically supplied with a *job object* which drives the job and supplies services to the control file. A control file can be as simple as:

```
job.run_test('kernbench')
```

The only mandatory argument is the name of the test. There are lots of examples; each test has a sample control file under `tests/<testname>/control`

If you're sitting in the top level of the Autotest client, you can run the control file like this:

```
$ client/autotest-local <control_file_name>
```

You can also supply specific arguments to the test

```
job.run_test('kernbench', iterations=2, threads=5)
```

- First parameter is the test name.
- The others are arguments to the test. Most tests will run with no arguments if you want the defaults.

If you would like to specify a tag for the results directory for a particular test:

```
job.run_test('kernbench', iterations=2, threads=5, tag='mine')
```

Will create a results directory “kernbench.mine” instead of the default “kernbench”. This is particularly important when writing more complex control files that may run the same test multiple times, in order to properly separate the results of each of the test runs they will need a unique tag.

External tests

Sometimes when you are developing a test it's useful to have it packaged somewhere so your control file can download it, uncompress it and run it. The convention for packaging test is on *External Tests*. Make sure you read that session before you try to package and run your own external tests.

Flow control

One of the benefits of the use of a true programming language for the control script is the ability to use all of its structural and error checking features. Here we use a loop to run kernbench with different threading levels.

```
for t in [8, 16, 32]:
    job.run_test('kernbench', iterations=2, threads=t, tag='%d' % t)
```

System information collection

After every reboot and after every test, Autotest will collect a variety of standard pieces of system information made up of specific files grabbed from the filesystem (e.g. `/proc/meminfo`) and the output of various commands (e.g. `uname -a`). You can see this output in the results directories, under `sysinfo/` (for per-reboot data) and `<testname>/sysinfo` (for pre-test data).

For a full list of what's collected by default you can take a look at `client/bin/base_sysinfo.py`; however, there also exists a mechanism for adding extra files and commands to the system info collection inside your control files. To add a custom file to the log collection you can call:

```
job.add_sysinfo_file("/proc/vmstat")
```

This would collect the contents of `/proc/vmstat` after every reboot. To collect it on every test you can use the optional `on_every_test` parameter, like so:

```
job.add_sysinfo_file("/proc/vmstat", on_every_test=True)
```

There exists a similar method for adding a new command to the sysinfo collection:

```
job.add_sysinfo_command("lspci -v", logfile="lspci.txt")
```

This will run `lspci -v` through the shell on every reboot, logging the output in `lspci.txt`. The `logfile` parameter is optional; if you do not specify it, the logfile will default to the command text with all whitespace replaced with underscores (e.g. in this case it would use `lspci_ -v` as the filename). This method also takes an `on_every_test` parameter that can be used to run the collection after every test instead of every reboot.

Using the profilers facility

You can enable one or more profilers to run during the test. Simply add them before the tests, and remove them afterwards, e.g.:

```
job.profilers.add('oprofile')
job.run_test('sleeptest')
job.profilers.delete('oprofile')
```

If you run multiple tests like this:

```
job.profilers.add('oprofile')
job.run_test('kernbench')
job.run_test('dbench')
job.profilers.delete('oprofile')
```

It will create separate profiling output for each test - generally we do a separate profiling run inside each test, so as not to perturb the performance results. Profiling output will appear under `<testname>/profiling` in the results directory.

Again, there are examples for all profilers in `profilers/<profiler-name>/control`.

Creating filesystems

We have support built in for creating filesystems. Suppose you wanted to run the `fsx` test against a few different filesystems:

```

# Uncomment this line, and replace the device with something sensible
# for you ...
# fs = job.filesystem('/dev/hda2', job.tmpdir)

for fstype in ('ext2', 'ext3'):
    fs.mkfs(fstype)
    fs.mount()
    try:
        job.run_test('fsx', job.tmpdir, tag=fstype)
    finally:
        fs.unmount()

```

or if we want to show off and get really fancy, we could mount EXT3 with a bunch of different options, and see how the performance compares across them:

```

fs = job.filesystem('/dev/sda3', job.tmpdir)

iters=10

for fstype, mountopts, tag in (('ext2', '', 'ext2'),
                              ('ext3', '-o data=writeback', 'ext3writeback'),
                              ('ext3', '-o data=ordered', 'ext3ordered'),
                              ('ext3', '-o data=journal', 'ext3journal')):
    fs.mkfs(fstype)
    fs.mount(args=mountopts)
    try:
        job.run_test('fsx', job.tmpdir, tag=tag)
        job.run_test('iozone', job.tmpdir, iterations=iters, tag=tag)
        job.run_test('dbench', iterations=iters, dir=job.tmpdir, tag=tag)
        job.run_test('tiobench', dir=job.tmpdir, tag=tag)
    finally:
        fs.unmount()

```

Rebooting during a job

Where a job needs to cause a system reboot such as when booting a newly built kernel, there is necessarily an interruption to the control script execution. The job harness therefore also provides a phased or step based interaction model.

```

def step_init():
    job.next_step([step_test])
    testkernel = job.kernel('2.6.18')
    testkernel.config('http://mbligh.org/config/opteron2')
    testkernel.build()
    testkernel.boot()           # does autotest by default

def step_test():
    job.run_test('kernbench', iterations=2, threads=5)
    job.run_test('dbench', iterations=5)

```

By defining a `step_init` this control script has indicated it is using step mode. This triggers automatic management of the step state across breaks in execution (such as a reboot) maintaining forward flow.

It is important to note that the step engine is not meant to work from the scope of the tests, that is, inside a test module (`job.run_test()`, from the control file perspective). The reboots and step engine are only meant to be used from the control file level, since a lot of precautions are taken when running test code, such as shielding autotest from rogue

exceptions thrown during test code, as well as executing test code on a subprocess, where it is less likely to break autotest and we can kill that subprocess if it reaches a timeout.

So this code inside a control file is correct:

```
def step_init():
    job.next_step([step_test])
    testkernel = job.kernel('testkernel.rpm')
    testkernel.install()
    testkernel.boot()

def step_test():
    job.run_test('ltp')
```

This code, inside a test module, isn't:

```
class kerneltest(test.test):
    def execute(self):
        testkernel = job.kernel('testkernel.rpm')
        testkernel.boot()
```

In broad brush, when using the step engine, the control file is not simply executed once, but repeatedly executed until it indicates the job is complete. In a stand-alone context we would expect to re-start execution automatically on boot when a control file exists, in a managed environment the managing server would perform the same role.

Obviously looping is more difficult in the face of phase based execution. The state maintained by the stepping engine is such, that we can implement a boot based loop using step parameters.

```
def step_init():
    step_test(1)

def step_test(iteration):
    if (iteration < 5):
        job.next_step([step_test, iteration + 1])

    print "boot: %d" % iteration

    job.run_test('kernbench', tag="%d" % i)
    job.reboot()
```

Running multiple tests in parallel

The job object also provides a parallel method for running multiple tasks at the same time. The method takes a variable number of arguments, each representing a different task to be run in parallel. Each argument should be a list, where the first item on the list is a function to be called and all the remaining elements are arguments that will be passed to the function when it is called.

```
def first_task():
    job.run_test('kernbench')

def second_task():
    job.run_test('dbench')

job.parallel([first_task], [second_task])
```

This control file will run both *kernbench* and *dbench* at the same time. Alternatively, this could've been written as:

```
job.parallel([job.run_test, 'kernbench'], [job.run_test, 'dbench'])
```

However, if you want to do something more complex in your tasks than call a single function then you'll have to define your own functions to do it, as in the first example.

The parallel jobs are run through fork, so each task will be running in its own address space and you don't need to worry about performing any process-local synchronization between your separate tasks. However, these processes will still be running on the same machine and so still need to make certain that these tasks don't crash into each other while accessing shared resources (e.g. the filesystem). This means no rebooting during parallel tasks, and if you're running the same test in different tasks, you must be sure to give each task a unique tag

Control file specification

This document will go over what is required to be in a control file for it to be accepted into git. The goal of this is to have control files that contain all necessary information for the frontend/the user to ascertain what the test does and in what ways it can be modified.

Naming your control files

Control files should always start with **control.XXXXX**, where **XXXXX** is up to you and the code reviewer, the idea is for it to be short sweet and descriptive. For example, for 500 iterations of hard reboot test a decent name would be `control.hard500`.

Variables

An overview of variables that should be considered required in any control file submitted to our repo.

Name	Description
* AUTHOR	Contact information for the person or group that wrote the test
DEPENDENCIES	What the test requires to run. Comma delimited list e.g. 'CONSOLE'
* DOC	Description of the test including what arguments can be passed to it
EXPERIMENTAL	If this is set to True production servers will ignore the test
* NAME	The name of the test for the frontend
RUN_VERIFY	Whether or not the scheduler should run the verify stage, default True
SYNC_COUNT	Accepts a number >=1 (1 being the default)
* TIME	How long the test runs SHORT < 15m, MEDIUM < 4 hours, LONG > 4 hour
TEST_CLASS	This describes the class for your the test belongs in. e.g. Kernel, Hardware
TEST_CATEGORY	This describes the category for your tests e.g. Stress, Functional
* TEST_TYPE	What type of test: client, server

* Are required for test to be considered valid

If you'd like to verify that your control file defines these variables correctly, try the `utils/check_control_file_vars.py` utility.

AUTHOR (Required)

The name of either a group or a person who will be able to answer questions pertaining to the test should the development team not be able to fix bugs. **With email address included**

DEPENDENCIES (Optional, default: None)

Dependencies are a way to describe what type of hardware you need to find to run a test on. Dependencies are just a fancy way of saying if this machine has this label on it then it is eligible for this test.

An example usecase for this would be if you need to run a test on a device that has bluetooth you would add the following to your control file:

```
DEPENDENCY = "Bluetooth"
```

Where `Bluetooth` is the exact label that was created in Autotest and has been added to a machine in Autotest either via the CLI or the Django admin interface.

DOC (Required)

The doc string should be fairly verbose describing what is required for the test to be run successfully and any modifications that can be done to the test. Any arguments in your `def execute()` that can change the behavior of the test need to be listed here with their defaults and a description of what they do.

EXPERIMENTAL (Optional, default: False)

If this field is set the test import process for the frontend will ignore these tests for production Autotest servers. This is useful for gettings tests checked in and tested in development servers without having to worry about them sneaking into production servers.

NAME (Required)

The name that the frontend will display, this is useful when you have multiple control files for the same test but with slight variations.

RUN_VERIFY (Optional, default: True)

It is used to have the scheduler not run verify on a particular job when it is scheduling it.

SYNC_COUNT (Optional, default: 1)

It accepts a number ≥ 1 (1 being the default). If it's 1, then it's a async test. If it's > 1 it's sync.

For example, if I have a test that requires exactly two machines `SYNC_COUNT = 2`. The scheduler will then find the maximum amount of machines from the job submitted that will run that fit the `SYNC_COUNT` evenly.

For example, if I submit a job with 23 machines, 22 machines will run the test in that job and one will fail to run because it doesn't have a pair.

TIME (Required)

How long the test generally takes to run. This does not include the autotest setup time but just your individual test's time.

TIME	Description
SHORT	Test runs for a maximum of 15 minutes
MEDIUM	Test runs for less four hours
LONG	Test runs for longer four hours

TEST_CATEGORY (Required)

This is used to define the category your tests are a part of.

Examples of categories:

- Functional
- Stress

TEST_CLASS (Required)

This****describes the class type of tests. This is useful if you have different different types of tests you want to filter on. If a test is added with a TEST_CLASS that does not exist the frontend should add that class.

Example tests classes

- Kernel
- Hardware

TEST_TYPE (Required)

This will tell the frontend what type of test it is. Valid values are **server** and **client**. Although `server_async` jobs are also a type of job in correlation with `SYNC_COUNT` this is taken care of.

Example

```

TIME = 'MEDIUM'
AUTHOR = 'Scott Zawalski ( scott@xxx.com )'
TEST_CLASS = 'Hardware'
TEST_CATEGORY = 'Functional'
NAME = 'Hard Reboot'
SYNC_COUNT = 1
TEST_TYPE = 'server'
TEST_CLASS = 'Hardware'
DEPENDCIES = 'POWER, CONSOLE'

DOC = """
Tests the reliability of platforms when rebooted. This test allows
you to do a hard reboot or a software reboot.

Args:
type: can be "soft" or "hard", default is "hard"
e.g. job.run_test('reboot', machine, type="soft")
This control file does a HARD reboot
"""

def run(machine):

```

```
job.run_test('reboot', machine, type="hard")
parallel_simple(run, machines)
```

Test modules development

Tests should be self-contained modular units, encompassing everything needed to run the test (apart from calls back into the core harness)

Tests should:

- Run across multiple hardware architectures
- Run on multiple distros
- Have a maintainer
- Provide simple examples for default running
- Not modify anything outside of their own directories, or provided scratch areas.

Adding tests to autotest

Adding a test is probably the easiest development activity to do.

Each test is completely contained in it's own subdirectory (either in `client/tests` for client-side tests or `server/tests` for server-side tests) - the normal components are

- An example control file, e.g. `tests/mytest/control`.
- A test wrapper, e.g. `tests/mytest/mytest.py`.
- Some source code for the test, if it's not all done in just the Python script.

Start by taking a look over an existing test, e.g. `tests/dbench`. First, note that the name of the subdirectory - `tests/dbench`, the test wrapper - `dbench.py` and the name of the class inside the test wrapper - `dbench`, all match. Make sure you do this in your new test too.

The control file is trivial:

```
job.run_test('dbench')
```

That just takes the default arguments to run `dbench` - mostly, we try to provide sensible default settings to get you up and running easily, then you can override most things later.

There's a tarball for the source code - `dbench-3.04.tar.gz` - this will get extracted under `src/` later. Most of what you're going to have to do is in the Python wrapper. Look at `dbench.py` - you'll see it inherits from the main test class, and defines a version (more on that later). You'll see four functions:

- `initialize()` - This is run before everything, every time the test is run.
- `setup()` - This is run when you first use the test, and normally is used to compile the source code.
- `run_once()` - This is called by `job.run_test` N times, where N is controlled by the `iterations` parameter to `run_test` (defaulting to one). It also gets called an additional time with profilers turned on, if you have any profilers enabled.
- `postprocess_iteration()` - This processes any results generated by the test iteration, and writes them out into a `keyval`. It's generally not called for the profiling iteration, as that may have different performance.

The test will result in a *PASS*, unless you throw an exception, in which case it will *FAIL* (`error.TestFail?`), *WARN* (`error.TestWarn?`) or *ERROR* (anything else). Most things that go wrong in Python will throw an exception for you, so you don't normally have to worry about this much - you can check extra things and throw an exception if you need. Now let's look at those functions in some more detail.

setup

This is the one-off setup function for this test. It won't run again unless you change the version number (so be sure to bump that if you change the source code). In this case it'll extract *dbench-3.04.tar.gz* into `src/`, and compile it for us. Look at the first few lines:

```
# http://samba.org/ftp/tridge/dbench/dbench-3.04.tar.gz
def setup(self, tarball='dbench-3.04.tar.gz'):
    tarball = utils.unmap_url(self.bindir, tarball, self.tmpdir)
```

A comment saying where we got the source from. The function header - defines what the default tarball to use for the source code is (you can override this with a different *dbench* version from the control file if you wanted to, but that's highly unusual). Lastly there's some magic with `unmap_url` - that's just incase you overrode it with a `url` - it'll download it for you, and return the local path ... just copy that bit.

```
utils.extract_tarball_to_dir(tarball, self.srcdir)
os.chdir(self.srcdir)
utils.system('./configure')
utils.system('make')
```

OK, so this just extracts the tarball into `self.srcdir` (pre-setup for you to be `src/` under the test), `cd`'s into that `src` dir, and runs `./configure; make` just as you would for most standard compilations.

Note: We use the local `system()` wrapper, not `os.system()` - this will

automatically throw an exception if the return code isn't 0, etc.

Apart from compiling a package from the source, you have an option to use the client `system`'s software manager to install a package using the `software_manager` module.

Here is how you do it:

```
from autotest.client.shared import software_manager
backend=software_manager.SoftwareManager()
backend.install('<package_name>')
```

That's all!

run_once

This actually executes the test. The core of what it's doing is just:

```
self.results.append(utils.system_output(cmd))
```

Which says "run `dbench` and add the output to `self.results`". We need to record the output so that we can process it after the test runs in `postprocess`.

postprocess_iteration

For performance benchmarks, we want to produce a keyval file of **key=value** pairs, describing how well the benchmark ran. The key is just a string, and the value is a floating point or integer number. For `dbench`, we produce just two performance metrics - “throughput” and “nprocs”. The function is called once per iteration (except for the optional profiling iteration), and we end up with a file that looks like this:

```
throughput = 217
nprocs = 4

throughput = 220
nprocs = 4

throughput = 215
nprocs = 4
```

Note that the above was from a run with three iterations - we ran the benchmark 3 times, and thus print three sets of results. Each set is separated by a blank line.

Additional test methods

These methods aren’t implemented in the `dbench` test, but they can be implemented if you need to take advantage of them.

warmup

For performance tests that need to conduct any pre-test priming to make the results valid. This is called by `job.run_test` before running the test itself, but after all the setup.

cleanup

Used for any post-test cleanup. If test may have left the machine in a broken state, or your initialize made a large mess (e.g. used up most of the disk space creating test files) that could cause problems with subsequent tests then it’s probably a good idea to write a cleanup that undoes this. It always gets called, regardless of the success or failure of the test execution.

execute

Used for executing the test, by calling `warmup`, `run_once` and `postprocess`. The base test class provides an implementation that already supports profilers and multiple test iterations, but if you need to change this behavior you can override the default implementation with your own.

Note: If you want to properly support multi-iteration tests and/or profiling

runs, you must provide that support yourself in your custom `execute` implementation.

Adding your own test

Now just create a new subdirectory under tests, and add your own control file, source code, and wrapper. It's probably easiest to just copy `dbench.py` to `mytest.py`, and edit it - remember to change the name of the class at the top though.

If you have any problems, or questions, drop an email to the [Autotest mailing list](#)), and we'll help you out.

Using and developing job profilers

Adding a profiler is much like adding a test. Each profiler is completely contained in it's own subdirectory (under `client/profilers` or if you just checked out the client - under `profilers/`) - the normal components are:

- An example control file, e.g. `profilers/myprofiler/control`.
- A profiler wrapper, e.g. `profilers/myprofiler.py`.
- Some source code for the profiler (if it's not all done in just the Python script)

Start by taking a look over an existing profiler. I'll pick `readprofile`, though it's not the simplest one, as it shows all the things you might need. Be aware this one will only work if you have `readprofile` support compiled into the kernel.

The control file is trivial, just

```
job.profilers.add('readprofile')
job.run_test('sleeptest', 1)
job.profilers.delete('readprofile')
```

That just says "please use `readprofile` for the following tests". You can call `profilers.add` multiple times if you want multiple profilers at once. Then we generally just use `sleeptest` to do a touch test of profilers - it just sleeps for N seconds (1 in this case).

There's a tarball for the source code - `util-linux-2.12r.tar.bz2` - this will get extracted under `src/` later. Most of what you're going to have to do is in the python wrapper. Look at `readprofile.py` - you'll see it inherits from the main profiler class, and defines a version (more on that later). You'll see several functions:

- `setup()` - This is run when you first use the profiler, and normally is used to compile the source code.
- `intialize()` - This is run whenever you import the profiler.
- `start()` - Starts profiling.
- `stop()` - Stops profiling.
- `report()` - Run a report on the profiler data.

Now let's look at those functions in some more detail.

Setup

This is the one-off setup function for this test. It won't run again unless you change the version number (so be sure to bump that if you change the source code). In this case it'll extract `util-linux-2.12r.tar.bz2` into `src/`, and compile it for us. Look at the first few lines:

```
# http://www.kernel.org/pub/linux/utils/util-linux/util-linux-2.12r.tar.bz2
def setup(self, tarball = 'util-linux-2.12r.tar.bz2'):
    self.tarball = unmap_url(self.bindir, tarball, self.tmpdir)
    extract_tarball_to_dir(self.tarball, self.srcdir)
```

A comment saying where we got the source from. The function header - defines what the default tarball to use for the source code is (you can override this with a different version from the control file if you wanted to, but that's highly unusual). Lastly there's some magic with `unmap_url` - that's just in case you overrode it with a URL - it'll download it for you, and return the local path just copy that bit.

```
os.chdir(self.srcdir)
system('./configure')
os.chdir('sys-utils')
system('make readprofile')
```

OK, so this just extracts the tarball into `self.srcdir` (pre-setup for you to be `src/` under the profiler), `cd`'s into that `src` dir, and runs `./configure` and then just makes the `readprofile` component (util-linux also contains a bunch of other stuff we don't need) - just as you would for most standard compilations. Note that we use the local `system()` wrapper, not `os.system()` - this will automatically throw an exception if the return code isn't 0, etc.

Initialize

```
def initialize(self):
    try:
        system('grep -iq " profile=" /proc/cmdline')
    except:
        raise CmdError, 'readprofile not enabled'

    self.cmd = self.srcdir + '/sys-utils/readprofile'
```

This runs whenever we import this profiler - it just checks that `readprofile` is enabled, else it won't work properly.

Start

```
def start(self, test):
    system(self.cmd + ' -r')
```

Start the profiler! Just run `readprofile -r`.

Stop

```
def stop(self, test):
    # There's no real way to stop readprofile, so we stash the
    # raw data at this point instead. BAD EXAMPLE TO COPY! ;- )
    self.rawprofile = test.profdir + '/profile.raw'
    print "STOP"
    shutil.copyfile('/proc/profile', self.rawprofile)
```

Normally you'd just run `readprofile --stop`, except this profiler doesn't seem to have that. We want to do the lightest-weight thing possible, in case there are multiple profilers running, and we don't want them to interfere with each other.

Report

```
def report(self, test):
    args = ' -n'
    args += ' -m ' + get_systemmap()
```

```

args += ' -p ' + self.rawprofile
cmd = self.cmd + ' ' + args
txtprofile = test.profdir + '/profile.text'
system(cmd + ' | sort -nr > ' + txtprofile)
system('bzip2 ' + self.rawprofile)

```

This just converts it into text. We need to find this kernel's `System.map` etc (for which there's a helper), and then produce the results in a useful form (in this case, a text file). Note that we're passed the test object, so we can store the results under the `profiling/` subdirectory of the test's output by using the `test.profdir` which has been set up automatically for you.

Adding your own profiler

Now just create a new subdirectory under `profilers`, and add your own control file, source code, and wrapper. It's probably easiest to just copy `readprofile.py` to `mytest.py`, and edit it - remember to change the name of the class at the top though.

If you have any problems, or questions, drop an email to the [Autotest mailing list](#), and we'll help you out.

Linux distribution detection

Autotest has a facility that lets tests determine quite precisely the distribution they're running on.

This is done through the implementation and registration of probe classes.

Those probe classes can check for given characteristics of the running operating system, such as the existence of a release file, its contents or even the existence of a binary that is exclusive to a distribution (such as package managers).

Quickly detecting the Linux distribution

The `autotest.client.shared.distro` module provides many APIs, but the simplest one to use is the `detect()`.

Its usage is quite straightforward:

```

from autotest.client.shared import distro
detected_distro = distro.detect()

```

The returned `distro` can be the result of a probe validating the distribution detection, or the not so useful `UNKNOWN_DISTRO`.

To access the relevant data on a `LinuxDistro`, simply use the attributes:

- name
- version
- release
- arch

Example:

```

>>> detected_distro = distro.detect()
>>> print detected_distro.name
redhat

```

The unknown Linux distribution

When the detection mechanism can't precisely detect the Linux distribution, it will still return a *LinuxDistro* instance, but a special one that contains special values for its name, version, etc.

```
autotest.client.shared.distro.UNKNOWN_DISTRO = <LinuxDistro: name=unknown, version=0, release=0, arch=unknown>
```

The distribution that is used when the exact one could not be found

Writing a Linux distribution probe

The easiest way to write a probe for your target Linux distribution is to make use of the features of the *Probe* class.

Even if you do plan to use the features documented here, keep in mind that all probes should inherit from *Probe* and provide a basic interface.

Checking the distribution name only

The most trivial probe is one that checks the existence of a file and returns the distribution name:

```
class RedHatProbe(Probe):
    CHECK_FILE = '/etc/redhat-release'
    CHECK_FILE_DISTRO_NAME = 'redhat'
```

To make use of a probe, it's necessary to register it:

```
from autotest.client.shared import distro
distro.register_probe(RedHatProbe)
```

And that's it. This is a valid example, but will give you nothing but the distro name.

You should usually aim for more information, such as the version numbers.

Checking the distribution name and version numbers

If you want to also detect the distro version numbers (and you should), then it's possible to use the *Probe*.*CHECK_VERSION_REGEX* feature of the *Probe* class.

Probe.**CHECK_VERSION_REGEX = None**

A regular expression that will be run on the file pointed to by *CHECK_FILE_EXISTS*

If your regex has two or more groups, that is, it will look for and save references to two or more string, it will consider the second group to be the *LinuxDistro*.*release* number.

Probe Scores

To increase the accuracy of the probe results, it's possible to register a score for a probe. If a probe wants to, it can register a score for itself.

Probes that return a score will be given priority over probes that don't.

The score should be based on the number of checks that ran during the probe to account for its accuracy.

Probes should not be given a higher score because their checks look more precise than everyone else's.

Registering your own probes

Not only the probes that ship with Autotest can be used, but your custom probe classes can be added to the detection system.

To do that simply call the function `register_probe()`:

```
autotest.client.shared.distro.register_probe(probe_class)
    Register a probe to be run during autodetection
```

Now, remember that for things to happen smoothly your registered probe must be a subclass of `Probe`.

API Reference

LinuxDistro

class `autotest.client.shared.distro.LinuxDistro` (*name, version, release, arch*)
Simple collection of information for a Linux Distribution

Probe

class `autotest.client.shared.distro.Probe`
Probes the machine and does it best to confirm it's the right distro

CHECK_FILE = None

Points to a file that can determine if this machine is running a given Linux Distribution. This servers a first check that enables the extra checks to carry on.

CHECK_FILE_CONTAINS = None

Sets the content that should be checked on the file pointed to by `CHECK_FILE_EXISTS`. Leave it set to *None* (its default) to check only if the file exists, and not check its contents

CHECK_FILE_DISTRO_NAME = None

The name of the Linux Distribution to be returned if the file defined by `CHECK_FILE_EXISTS` exist.

CHECK_VERSION_REGEX = None

A regular expression that will be run on the file pointed to by `CHECK_FILE_EXISTS`

check_name_for_file()

Checks if this class will look for a file and return a distro

The conditions that must be true include the file that identifies the distro file being set (`CHECK_FILE`) and the name of the distro to be returned (`CHECK_FILE_DISTRO_NAME`)

check_name_for_file_contains()

Checks if this class will look for text on a file and return a distro

The conditions that must be true include the file that identifies the distro file being set (`CHECK_FILE`), the text to look for inside the distro file (`CHECK_FILE_CONTAINS`) and the name of the distro to be returned (`CHECK_FILE_DISTRO_NAME`)

check_release()

Checks if this has the conditions met to look for the release number

check_version()

Checks if this class will look for a regex in file and return a distro

get_distro()

Returns the `LinuxDistro` this probe detected

name_for_file()
Get the distro name if the *CHECK_FILE* is set and exists

name_for_file_contains()
Get the distro if the *CHECK_FILE* is set and has content

release()
Returns the release of the distro

version()
Returns the version of the distro

register_probe()

`autotest.client.shared.distro.register_probe(probe_class)`
Register a probe to be run during autodetection

detect()

`autotest.client.shared.distro.detect()`
Attempts to detect the Linux Distribution running on this machine

Returns the detected *LinuxDistro* or *UNKNOWN_DISTRO*

Return type *LinuxDistro*

External downloadable tests

As well as executing built-in tests it is possible to execute external tests. This allows non-standard tests to be constructed and executed without any requirement to modify the installed Autotest client.

Executing Tests

A downloadable test is triggered and run in the standard way via the `run_test` method, but specifying a URL to a tarball of the test:

```
job.run_test('http://www.example.com/~someone/somewhere/test.tar.bz2')
```

This will download, install, and execute the test as if it were built-in.

Constructing external downloadable tests

External downloadable tests consist of a bzip'ed tarball of the *contents* of a test directory. Things that need to match:

1. The name of the tarball, i.g. `my_test.tar.bz2`
2. The name of the primary Python file, i.g. `my_test.py`
3. The name of the test class itself, i.e. `class my_test(test.test):`

Example:

```
$ cat example_test/my_test.py
from autotest_lib.client.bin import test

class my_test(test.test):
    version = 1

    def initialize(self):
        print "INIT"

    def run_once(self):
        print "RUN"

$ tar -C example_test -jcvf my_test.tar.bz2 .
./
./my_test.py
```

Note: You should not pack “example_test” directory but the **contents** of it. Files must be at the *root* of the archive.

Keyval files in Autotest

There are several “keyval” files in the `results` directory. These take the simple form

```
key1=value1
key2=value2
```

Below we describe what information is in which file.

Job level keyval

This file contains high level information about the job such as when it was queued, started, finished, the username of the submitter, and what machines are involved.

Synchronous multi-machine jobs

When running a multi-machine job synchronously, you will end up with multiple “job level” keyval files; at the very least, one upper-level keyval file in the root results directory, and one in each machine subdirectory. In the results database each machine will be interpreted as a separate set of results, with the total job keyval data being composed of data from the “uppermost” of the keyval files (i.e. the single job level keyval in the root dir). The single exception to this is the `hostname` field - this is taken from the machine directory.

Test level keyval

This file contains the version of the test, and some per-test system information (parsed from the `sysinfo` dir) so that we can load it up into the database easily.

Results level keyval

This file contains performance information for a test. Maybe something like

```
throughput=100
latency=12
```

If we ran multiple iterations of a test, there will be repeated keyvals in there, separated by a blank line:

```
throughput=101
latency=12.9

throughput=100
latency=11.2

throughput=96
latency=13.1
```

Diagnosing failures in your results

This document will describe how to go about triaging your Autotest results and finding out what went wrong.

Basics

A lot of times when tests fail there are a number of things that could have come into play. Below are a few things that should be considered.

- Baseline
- What changed between tests
- Look at the raw results

Having a baseline is an absolute **must**:

- Have you run these tests on this particular system before?
- Did it pass without any issues?

These are questions you should be asking yourself. If you do not have a baseline that is the first thing to establish. It really is as simple as running a job and making note of the results.

A lot of the time that people have tests fail they do not consider what changed in between tests. Any change what so ever is important to make note of. From something big like, did I change the kernel? To something small like did I move my system to a different area which may have impacted the cooling of the system?

Lastly if nothing has changed and you have established a baseline for your machines it is time to delve into the results.

Looking at raw results

There are a few key areas worth looking at when evaluating what could have went wrong with your job. From the *View Job* tab click on *raw results log*. Here you will be presented with a directory structure that represents your job flat files. If you created a job with multiple machines there will be individual directories for each machine. Navigate to the machine you want to investigate.

The debug directory

All tests run including the main Autotest job will have a `debug` directory. Here you will find the majority of the information you need to diagnose issues with tests.

The following files in `debug` directory will give you insight into what Autotest was doing at the time:

```
debug/
- build_log.gz
- client.DEBUG
- client.ERROR
- client.INFO
- client.WARNING
```

If you have console support (via `conmux`) you should also take a look at `conmux.log`.

If at any point Autotest produced a stacktrace, `*.ERROR` will most likely contain this information. That is a good place to start if the test run failed and you want to see if Autotest itself as at fault for the problem.

If both of these files are clean next we go to the `<hostname>/test/` directory.

Example investigation

This example was created on host without `time` utility, I tried to launch `kernbench` (output reduced):

```
# client/autotest-local --verbose run kernbench
10:01:59 INFO | Writing results to /usr/local/autotest/client/results/default
...
10:03:19 DEBUG| Running 'gzip -9 '/usr/local/autotest/client/results/default/
↳kernbench/debug/build_log''
10:03:19 ERROR| Exception escaping from test:
Traceback (most recent call last):
  File "/usr/local/autotest/client/shared/test.py", line 398, in _exec
    *args, **dargs)
  File "/usr/local/autotest/client/shared/test.py", line 823, in _call_test_function
    return func(*args, **dargs)
  File "/usr/local/autotest/client/shared/test.py", line 738, in _cherry_pick_call
    return func(*p_args, **p_dargs)
  File "/usr/local/autotest/client/tests/kernbench/kernbench.py", line 53, in warmup
    self.kernel.build_timed(self.threads, output=logfile) # warmup run
  File "/usr/local/autotest/client/kernel.py", line 377, in build_timed
    utils.system(build_string)
  File "/usr/local/autotest/client/shared/utils.py", line 1232, in system
    verbose=verbose).exit_status
  File "/usr/local/autotest/client/shared/utils.py", line 918, in run
    "Command returned non-zero exit status")
CmdError: Command </usr/bin/time -o /dev/null make -j 4 vmlinux > /usr/local/
↳autotest/client/results/default/kernbench/debug/build_log 2>&1> failed, rc=127,
↳Command returned non-zero exit status
* Command:
/usr/bin/time -o /dev/null make -j 4 vmlinux >
/usr/local/autotest/client/results/default/kernbench/debug/build_log 2>&1
Exit status: 127
Duration: 0.00197100639343
```

Here we are investigating why `kernbench` failed. The first place we want to look at is the `debug` directory. There we see the following files:

```
# tree -s debug/
debug/
- [          79] build_log.gz
- [        1345] client.DEBUG
- [           0] client.ERROR
```

```
- [          511] client.INFO
- [           0] client.WARNING
```

As it failed during build phase I am going to look at build_log:

```
$ cat build_log
/bin/bash: /usr/bin/time: No such file or directory
```

Well, that is true as:

```
[user@a5 debug]# which time
/usr/bin/which: no time in (/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/
↪bin:/root/bin)
[user@a5 debug]# ls /usr/bin/time
ls: cannot access /usr/bin/time: No such file or directory
```

In general test diagnoses should be that straight forward. Obvious this can not cover all cases.

The sysinfo directory

The sysinfo directory is exactly what it sounds like. A directory that contains as much information as possible that can be gathered from the machine:

```
# tree sysinfo/
sysinfo/
- df
- dmesg.gz
- messages.gz
- reboot_current -> ../../sysinfo
```

In general this directory is your second bet for finding issues. Most files are self explanatory, you should always examine dmesg to make sure your boot was clean. Then depending on what test you were running that failed examine files that will give you insight to that particular piece of hardware.

Manually running a job on a machine that is causing problems

A lot of times you will run into the case that all of your machines but two or three pass. While you may be able to figure out why most of them failed by looking at files it is sometimes advantageous to run the Autotest process individually on the problem machines.

Log-in to the machine and change to /home/autotest, there you will find the installation that the server put on this particular system.

The last control file of the job that was run is also available to you - control.autoserv.

To start the job over again run the following:

```
[root@udc autotest]# bin/autotest control.autoserv
```

This is exactly how the autotest server starts jobs on client machines.

If you have a large control file that runs multiple tests and you are only interested in one or two of them you can safely edit this file and remove any tests that you know work for sure. A lot of the time failures can be diagnosed by babysitting a machine and seeing what else is going on with general diagnostic on a machine.

Remote (Former Server)

Autotest Remote (Autoserv)

Autoserv is a framework for “automating machine control”

Autoserv’s purpose is to control machines, it can:

- power cycle
- install kernels
- modify bootloader entries
- run arbitrary commands
- run Autotest Local (client) tests
- transfer files

A machine can be:

- local
- remote (through ssh and conmux)
- virtual (through kvm)

Control Files

In a way similar to Autotest, Autoserv uses control files. Those control files use different commands than the Autotest ones but like the Autotest ones they are processed by the python interpreter so they contain functions provided by Autoserv and can contain python statements.

Here is an example control file that installs a .deb packaged kernel on a remote host controlled through ssh. If this file is placed in the `server/` directory and named “`example.control`”, it can be executed as `./autotest-remote example.control` from within the `server/` directory:

```
remote_host= hosts.SSHHost ("192.168.1.1")

print remote_host.run("uname -a").stdout

kernel= deb_kernel.DEBKernel()
kernel.get ("/var/local/linux-2.6.22.deb")

print kernel.get_version()
print kernel.get_image_name()
print kernel.get_initrd_name()

kernel.install(remote_host)

remote_host.reboot()

print remote_host.run("uname -a").stdout
```

Hosts

“Host” objects are the work horses of Autoserv control files. There are Host objects for machines controlled through ssh, through conmux or virtual machines. The structure of the code was planned so that support for other types

of hosts can be added if necessary. If you add support for another type of host, make sure to add that host to the `server/hosts/__init__.py` file.

Main Host Methods

Here are the most commonly used Host methods. Every type of host should implement these and support at least the options listed. Specific hosts may support more commands or more options. For information on these, see the associated source file for the host type in the `server/hosts/` subdirectory of Autotest. This listing is not a substitute for the source code function headers of those files, it's only a short summary. In particular, have a look at the `server/hosts/ssh_host.py` file.

- `run(command)`
- `reboot()`
- `get_file(source, dest)`
- `send_file(source, dest)`
- `get_tmp_dir()`
- `is_up()`
- `wait_up(timeout)`
- `wait_down(timeout)`
- `get_num_cpu()`

CmdResult Objects

The return value from a `run()` call is a `CmdResult` object. This object contains information about the command and its execution. It is defined and its documentation is in the file `server/hosts/base_classes.py`. `CmdResult` objects can be printed and they will output all their information. Each field can also be accessed separately. The list of fields is:

- `command`: String containing the command line itself
- `exit_status`: Integer exit code of the process
- `stdout`: String containing stdout of the process
- `stderr`: String containing stderr of the process
- `duration`: Elapsed wall clock time running the process
- `aborted`: Signal that caused the command to terminate (0 if none)

Main types of Host

SSHHost

`SSHHost` is probably the most common and most useful type of host. It represents a remote machine controlled through an ssh session. It supports all the base methods for hosts and features a `run()` function that supports timeouts. `SSHHost` uses `ssh` to run commands and `scp` to transfer files.

In order to use an `SSHHost` the remote machine must be configured for password-less login, for example through public key authentication. An `SSHHost` object is built by specifying a host name and, optionally, a user name and port number.

ConmuxSSHHost

ConmuxSSHHost is an extension of SSHHost. It is for machines that use Conmux (HOWTO). These support hard reset through the `hardreset()` method.

SiteHost

Site host is an empty class that is there to add site-specific methods or attributes to all types of hosts. It is defined in the file `server/hosts/site_host.py` but this file may be left empty, as it is, or removed altogether. Things that come to mind for this class are functions for flashing a BIOS, determining hardware versions or other operations that are too specific to be of general use. Naturally, control files that use these functions cannot really be distributed but at least they can use the generic host types like SSHHost without directly modifying those.

KVMGuest

KVMGuest represents a KVM virtual machine on which you can run programs. It must be bound to another host, the machine that actually runs the hypervisor. A KVMGuest is very similar to an SSHHost but it also supports “hard reset” through the `hardreset()` method (implemented in Guest) which commands the hypervisor to reset the guest. Please see the KVM section for more information on KVM and KVM guests.

LocalHost

Early versions of Autoserv represented the local machine (the one Autoserv runs on) as part of the Host hierarchy. This is no longer the case however because it was felt that some of the Host operations did not make sense on the local machine (`wait_down()` for example).

Bootloader

Boottool is a Perl script to query and modify boot loader entries. Autoserv provides the Bootloader class, a wrapper around boottool. Autoserv copies the boottool script automatically to a temporary directory the first time it is needed. Please see the `server/hosts/bootloader.py` file for information on all supported methods. The most important one is `add_kernel()`.

When adding a kernel, boottool’s default behavior is to reuse the command line of the first kernel entry already present in the bootloader configuration and use it to deduce the options to specify for the new entry.

InstallableObject

An InstallableObject represents a software package that can be installed on a host. It is characterized by two methods:

- `get(location)`
- `install(host)`

`get()` is responsible for fetching the source material for the software package. It can take many types of arguments as the location:

- a local file or directory
- a URL (http or ftp)
- a python file-like object

- if the argument doesn't look like any of the above, `get()` will assume that it is a string that represents the content itself

`get()` will store the content in a temporary folder on the host. This way, it can be fetched once and installed on many hosts. `install()` will install the software package on a host, typically in a temporary directory.

Autotest Support

Autoserv includes specific support for Autotest. It can install Autotest on a Host, run an Autotest control file and fetch the results back to the server. This is done through the Autotest and Run classes in `server/autotest.py`. The Autotest object is an `InstallableObject`. To use it, you have to:

- specify the source material via `get()` The Autotest object is special in this regard. If you do not specify any source, it will use the Autotest svn repository to fetch the software. This will be done on the target Host.
- `install()` it on a host When installing itself, Autotest will look for a `/etc/autotest.conf` file on the target host with a format similar to the following:

```
autodir=/usr/local/autotest/
```

- `run()` a control file The `run()` syntax is the following: `run(control_file, results_dir, host)` The `control_file` argument supports the same types of value as the `get()` method of `InstallableObject` (they use the same function behind the scenes)

Here is an example Autoserv control file to run an Autotest job, the results will be transferred to the "job_results" directory on the server (the machine Autoserv is running on).

```
remote_host= hosts.SSHHost ("192.168.1.1")

at= autotest.Autotest()
at.get("/var/local/autotest/client")
at.install(remote_host)

control_file= """
job.profilers.add("oprofile", events= ["CPU_CLK_UNHALTED:8000"])
job.run_test("linus_stress")
"""

results_dir= "job_results"

at.run(control_file, results_dir, remote_host)
```

Kernel Objects

Kernel objects are another type of `InstallableObject`s. Support is planned for kernels compiled from source and binary kernels packaged as `.rpm` and `.deb`. At the moment (Autotest revision 626), only `.deb` kernels are implemented. Some support for kernels from source is already in Autotest. Kernels support the following methods:

- **get(location)** customary `InstallableObject` method
- `install(host, extra arguments to boottool)` When a kernel is installed on a host, it will use `boottool` to make itself the default kernel to boot. If you want to specify additional arguments, you can do so and they will be passed to the `add_kernel()` method of the boot loader.
- `get_version()`
- `get_image_name()`

- `get_initrd_name()`

As always, see the source file function headers for complete details, for example see the file `server/deb_kernel.PY`

DEBKernels have an additional method, `extract(host)`. This method will extract the content the package to a temporary directory on the specified Host. This is not a step of the install process, it is if you want to access the content of the package without installing it. A possible usage of that function is with `kvm` and `qemu`'s `-kernel` option.

Here is an example Autoserv control file to install a kernel:

```
rh= hosts.SSHHost("192.168.1.1")

print rh.run("uname -a").stdout

kernel= deb_kernel.DEBKernel()
kernel.get("/var/local/linux-2.6.22.deb")

kernel.install(rh)

rh.reboot()

print rh.run("uname -a").stdout
```

A similar example using an RPM kernel and allowing the hosts to be specified from the autoserv commandline (autoserv `-m host1,host2 install-rpm`, for example):

```
if not machines:
    raise "Specify the machines to run on via the -m flag"

hosts = [hosts.SSHHost(h) for h in machines]

kernel = rpm_kernel.RPMKernel()
kernel.get('/stuff/kernels/kernel-smp-2.6.18.x86_64.rpm')

for host in hosts:
    print host.run("uname -a").stdout
    kernel.install(host, default=True)
    host.reboot()
    print host.run("uname -a").stdout

print "Done."
```

KVM Support

As stated previously, Autoserv supports controlling virtual machines. The object model has been designed so that various types of “virtual machine monitors”/hypervisors can be supported. At the moment (Autotest revision 626), only **KVM** support is included. In order to use KVM you must do the following:

1. create a Host, this will be machine that runs the hypervisor
2. create the KVM object, specify the source material for it via `get()`, and install it on that host The `KVM InstallableObject` is special in the sense that once it is installed on a Host, it is bound to that Host. This is because some status is maintained in the KVM object about the virtual machines that are running.
3. create `KVMGuest` objects, you have to specify, among other things, the KVM object created above
4. use the `KVMGuest` object like any other type of Host to run commands, change kernel, run Autotest, ...

Please see the files `server/kvm.py` and `server/hosts/kvm_guest.py` for more information on the parameters required, in particular, have a look at the function headers of `KVM.install()` and the `KVMGuest` constructor.

Here is an example Autoserv control file to do the above. Line 5 includes a list comprehension to create the required address list, remember that the control files are python.

```
remote_host= hosts.SSHHost("192.168.1.1")

kvm_on_remote_host= kvm.KVM(remote_host)
kvm_on_remote_host.get("/var/local/src/kvm-33.tar.gz")
addresses= [{"mac": "02:00:00:00:00:%02x" % (num,), "ip" : "192.168.2.%d" % (num,)}_
↳for num in range(1, 32)]
kvm_on_remote_host.install(addresses)

qemu_options= "-m 256 -hda /var/local/vdisk.img -snapshot"
g= hosts.KVMGuest(kvm_on_remote_host, qemu_options)
g.wait_up()

print g.run('uname -a').stdout.strip()
```

Compiling Options

You have to specify the source package for `kvm`, this should be an archive from http://sourceforge.net/project/showfiles.php?group_id=180599. When the `KVM` object is installed you have the control over two options: `build` (default `True`) and `insert_modules` (default `True`).

If `build` is `True`, Autoserv will execute `configure` and `make` to build the `KVM` client and kernel modules from the source material. `make install` will never be performed, to avoid disturbing an already present install of `kvm` on the system. In order for the build to succeed, the kernel source has to be present (`/lib/modules/${uname -r}/build` points to the appropriate directory). If `build` is `False`, `configure` and `make` should have been executed already and the binaries should be present in the source directory that was specified to `get()` (in step 2). You can also re-archive (`tar`) the source directories after building `kvm` if you wish and specify an archive to `get()`.

If `insert_modules` is `True`, Autoserv will first remove the `kvm` modules if they are present and insert the ones from the source material (that might have just been compiled or might have been already compiled, depending on the `build` option) when doing the `install()`. When the `KVM` object is deleted, it will also remove the modules from the kernel. At the moment, Autoserv will check for the appropriate type of kernel modules to insert, `kvm-amd` or `kvm-intel`. It will not check if `qemu` or `qemu-system-x86_64` should be used however, it always uses the latter. If `insert_modules` is `False`, the running kernel is assumed to already have `kvm` support and nothing will be done concerning the modules.

In short:

- If your kernel already includes appropriate `kvm` support, run `install(addresses, build=True, insert_modules=False)` or `install(addresses, build=False, insert_modules=False)` depending on whether you have the source for the running kernel. If `kvm` kernel support is compiled as modules, make sure that they are loaded before instantiating a `KVMGuest`, possibly using a command like this `remote_host.run("modprobe kvm-intel")` in your control file.
- If the kernel source will be present on the host, run `install(addresses, build=True, insert_modules=True)`
- Otherwise, compile the `kvm` sources on the server or another machine before running Autoserv and run `install(addresses, build=False, insert_modules=True)`

Kernel Considerations

Here are some kernel configuration options that might be relevant when you build your kernels.

Host Kernel

`CONFIG_HPET_EMULATE_RTC`, from the [kvm faq](#): I get “rtc interrupts lost” messages, and the guest is very slow
`KVM`, `KVM_AMD`, `KVM_INTEL`, if your kernel is recent enough and you want to have kvm support from the kernel

Guest Kernel

There are no specific needs for the guest kernel, so long as it can run under qemu, it is OK. Qemu emulates an IDE hard disk. Many distribution kernels use `ide` and `ide_generic` drivers so sticking with those instead of the newer `libata` potentially avoids device name changes from `/dev/hda` to `/dev/sda`. These can be compiled as modules, in which case an `initrd` will be needed. There is no real need for that however, compiling in the IDE drivers avoids the need for an `initrd`, this will ease the use of the `qemu -kernel` option.

Disk Image Considerations

The disk image must be specified as a `qemu` option, as in the example above:

```
qemu_options= "-m 256 -hda /var/local/vdisk.img -snapshot"
g= hosts.KVMGuest(kvm_on_remote_host, qemu_options)
```

Here `/var/local/vdisk.img` is the disk image and `-snapshot` instructs `qemu` not to modify the disk image, changes are discarded after the virtual machine terminates. Please refer to the [QEMU Documentation](#) for more information on the options you can pass to `qemu`.

IP Address Configuration

A few things have to be considered for the guest disk image. The most important one is specified in the `kvm.py:install()` documentation: “The virtual machine os must therefore be configured to configure its network with the ip corresponding to the mac”. `Autoserv` can only control the mac address of the virtual machine through `qemu` but it will attempt to contact it by its ip. You specify the mac-ip mapping in the `install()` function but you also have to make sure that when the virtual machine boots it acquires/uses the right ip. If you only want to spawn one virtual machine at a time you can set the ip statically on the guest disk image. If on the other hand you want to spawn many guests from the same disk image, you can assign ip’s from a properly configured dhcp server or you can have the os of the virtual machine choose an ip based on its mac. One way to do this with Debian compatible GNU/Linux distributions is through the `/etc/network/interfaces` file with a content similar to the following:

```
auto eth0
mapping eth0
    script /usr/local/bin/get-mac-address.sh
    map 02:00:00:00:00:01 vhost1
    map 02:00:00:00:00:02 vhost2

iface vhost1 inet static
    address 10.0.2.1
    netmask 255.0.0.0
    gateway 10.0.0.1
```

```
iface vhost2 inet static
    address 10.0.2.2
    netmask 255.0.0.0
    gateway 10.0.0.1
```

The file `/usr/local/bin/get-mac-address.sh` is the following:

```
#!/bin/sh

set -e

export LANG=C

iface="$1"
mac=$(/sbin/ifconfig "$iface" | sed -n -e '/^.*HWaddr \([[:xdigit:]]*\).*/{s//\1/;y/
↪ABCDEF/abcdef/;p;q;}')
which=""

while read testmac scheme; do
    if [ "$which" ]; then continue; fi
    if [ "$mac" = "$(echo "$testmac" | sed -e 'y/ABCDEF/abcdef/')' ]; then which="
↪$scheme"; fi
done

if [ "$which" ]; then echo $which; exit 0; fi
exit 1
```

The `/etc/network/interfaces` file is repetitive and tedious to write, instead it can be generated with the following python script. Make sure to adjust the values for `map_entry`, `host_entry`, `first_value` and `last_value`:

```
#!/usr/bin/python

header= """# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
mapping eth0
    script /usr/local/bin/get-mac-address.sh"""

map_entry= "        map 00:1a:11:00:00:%02x vhost%d"

host_entry= """iface vhost%d inet static
    address 10.0.2.%d
    netmask 255.0.0.0
    gateway 10.0.0.1"""

print header

first_value= 1
last_value= 16

for i in range(first_value, last_value + 1):
```

```

    print map_entry % (i, i,)

print ""

for i in range(first_value, last_value + 1):
    print host_entry % (i, i,)

```

SSH Authentication

Since a guest is accessed a lot like a SSHHost, it must also be configured for password-less login, for example through public key authentication.

Serial Console

Although this is not necessary for Autoserv itself, it is almost essential to be able to start the guest image with qemu manually, for example to do the initial setup. Qemu can emulate the display from a video card but it can also emulate a serial port. In order for this to be useful, the guest image must be setup appropriately:

- in the grub config (/boot/grub/menu.lst), if you use grub, to display the boot menu

```

serial --unit=0 --speed=9600 --word=8 --parity=no --stop=1
terminal --timeout=3 serial console

```

- in the kernel boot options, for boot and syslog output to the console

```

console=tty0 console=ttyS0,9600

```

- have a getty bound to the console for login, in /etc/inittab

```

T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100

```

Running Autotest In a Guest

Here is an example Autoserv control file to run an Autotest job inside a guest (virtual machine). This control file is special because it also runs OProfile on the host to collect some profiling information about the host system while the guest is running. This uses the system installation of oprofile, it must therefore be properly installed and configured on the host. The output of `oprofile` is saved in the results directory of the job that is run on the guest.

Here, a single address mapping is specified to `kvm`, since only one guest will be spawned. We tried running `oprofile` inside a `kvm` guest, without success, therefore it is not enabled. Finally, the options to `opcontrol --setup` should be adjusted if you know that `vmlinux` is present on the host system.

```

remote_host= hosts.SSHHost ("192.168.1.1")

kvm_on_remote_host= kvm.KVM(remote_host)

kvm_on_remote_host.get ("/var/local/src/kvm-compiled.tar.gz")
addresses= [{"mac": "02:00:00:00:00:01" , "ip" : "10.0.0.1"}]
kvm_on_remote_host.install(addresses, build=False, insert_modules=False)

qemu_options= "-m 256 -hda /var/local/vdisk.img -snapshot"
g1= hosts.KVMGuest(kvm_on_remote_host, qemu_options)
g1.wait_up()

```

```
at= autotest.Autotest()
at.get("/home/foo/autotest/client")
at.install(g1)

control_file= ""
#~ job.profilers.add("oprofile", events= ["CPU_CLK_UNHALTED:8000"])
job.run_test("linus_stress")
"""

results_dir= "g1_results"

# -- start oprofile
remote_host.run("opcontrol --shutdown")
remote_host.run("opcontrol --reset")
remote_host.run("opcontrol --setup "
    # "--vmlinux /lib/modules/${uname -r}/build/vmlinux "
    "--no-vmlinux "
    "--event CPU_CLK_UNHALTED:8000")
remote_host.run("opcontrol --start")
# --

at.run(control_file, results_dir, g1)

# -- stop oprofile
remote_host.run("opcontrol --stop")
tmpdir= remote_host.get_tmp_dir()
remote_host.run('opreport -l &> "%s" % (sh_escape(os.path.join(tmpdir, "report")),)')
remote_host.get_file(os.path.join(tmpdir, "report"), os.path.join(results_dir, "host_
↳oprofile"))
# --
```

Changing the Guest Kernel

“Usual” Way

The kvm virtual machine uses a bootloader, it can be rebooted and kvm will keep running, therefore, you can install a different kernel on a guest just like on a regular host:

```
remote_host= hosts.SSHHost("192.168.1.1")

kvm_on_remote_host= kvm.KVM(remote_host)
kvm_on_remote_host.get("/var/local/src/kvm-compiled.tar.gz")
addresses= [{"mac": "02:00:00:00:00:01", "ip": "10.0.0.1"}]
kvm_on_remote_host.install(addresses, build=False, insert_modules=False)

qemu_options= "-m 256 -hda /var/local/vdisk.img -snapshot"
g1= hosts.KVMGuest(kvm_on_remote_host, qemu_options)
g1.wait_up()

print g1.run("uname -a").stdout

kernel= deb_kernel.DEBKernel()
kernel.get("/home/foo/linux-2.6.21.3-6_2.6.21.3-6_amd64.deb")

kernel.install(g1)
```

```
g1.reboot()

print g1.run("uname -a").stdout
```

“QEMU” Way

It is also possible to use the `qemu -kernel`, `-append` and `-initrd` options. These options allow you to specify the guest kernel as a kernel image on the host’s hard disk.

This is a situation where `DEBKernel`’s `extract()` method is useful because it can extract the kernel image from the archive on the host, without installing it uselessly. However, `.deb` kernel images do not contain an `initrd`. The `initrd`, if needed, is generated after installing the package with a tool like `update-initramfs`. The tools `update-initramfs`, `mkinitramfs` or `mkinitrd` are all designed to work with an installed kernel, it is therefore very inconvenient to generate an `initrd` image for a `.deb` packaged kernel without installing it. The best alternative is to configure the guest kernel so that it doesn’t need an `initrd`, this is easy to achieve for a `qemu` virtual machine, it is discussed in the section *Guest Kernel*. On the other hand, if you already have a kernel and its `initrd`, you can also transfer them to the host with `send_file()` and then use those.

An important thing to note is that even though the kernel image (and possibly the `initrd`) are loaded from the host’s hard disk, the modules must still be present on the guest’s hard disk image. Practically, if your kernel needs modules, you can install them by manually starting `qemu` (without the `-snapshot` option) with the desired disk image and installing a kernel (via a `.deb` if you want) for the same version and a similar configuration as the one you intend to use with `-kernel`. You can also keep the `-snapshot` option and use the `commit` command in the `qemu` monitor.

Here’s an example control file that uses the `qemu -kernel` option. It gets the kernel image from a `.deb`, it is a kernel configured not to need an `initrd`:

```
remote_host= hosts.SSHHost("192.168.1.1")

kvm_on_remote_host= kvm.KVM(remote_host)
kvm_on_remote_host.get("/var/local/src/kvm-compiled.tar.gz")
addresses= [{"mac": "02:00:00:00:00:01", "ip": "10.0.0.1"}]
kvm_on_remote_host.install(addresses, build=False, insert_modules=False)

kernel= deb_kernel.DEBKernel()
kernel.get("/home/foo/linux-2.6.21.3-6_2.6.21.3-6_amd64-noNeedForInitrd.deb")
kernel_dir= kernel.extract(remote_host)

qemu_options= '-m 256 -hda /var/local/vdisk.img -snapshot -kernel "%s" -append "%s" '
↳% (sh_escape(os.path.join(kernel_dir, kernel.get_image_name()[1:])), sh_escape(
↳"root=/dev/hda1 ro console=tty0 console=ttyS0,9600"),)

g1= hosts.KVMGuest(kvm_on_remote_host, qemu_options)
g1.wait_up()

print g1.run("uname -a").stdout
```

Parallel commands

Autoserv control files can run commands in parallel via the `parallel()` and `parallel_simple()` functions from `subcommand.py`. This is useful to control many machines at the same time and run client-server tests. Here is an example that runs the Autoserv `netperf2` test, which is a network benchmark. This example runs the benchmark between a `kvm` guest running on one host and another (physical) host. This control file also has some code to check

that a specific kernel version is installed on these hosts and install it otherwise. This is not necessary to the netperf2 test or to parallel commands but it is done here to have a known configuration for the benchmarks.

```
def check_kernel(host, version, package):
    if host.run("uname -r").stdout.strip() != version:
        package.install(host)
        host.reboot()

def install_kvm(kvm_on_host_var_name, host, source, addresses):
    exec ("global %(var_name)s\n"
         "%(var_name)s= kvm.KVM(host)\n"
         "%(var_name)s.get(source)\n"
         "%(var_name)s.install(addresses)\n" % {"var_name": kvm_on_host_var_name})

remote_host1= hosts.SSHHost("192.168.1.1")
remote_host2= hosts.SSHHost("192.168.1.2")

kernel= deb_kernel.DEBKernel()
kernel.get("/var/local/linux-2.6.21.3-3_2.6.21.3-3_amd64.deb")

host1_command= subcommand(check_kernel, [remote_host1, "2.6.21.3-3", kernel])
host2_command= subcommand(check_kernel, [remote_host2, "2.6.21.3-3", kernel])

parallel([host1_command, host2_command])

install_kvm("kvm_on_remote_host1", remote_host1, "/var/local/src/kvm-33.tar.gz", [{
    ↪"mac": "02:00:00:00:00:01", "ip" : "10.0.0.1"}])

qemu_options= "-m 256 -hda /var/local/vdisk.img -snapshot"
gserver= hosts.KVMGuest(kvm_on_remote_host1, qemu_options)
gserver.wait_up()

at= autotest.Autotest()
at.get("/home/foo/autotest/client")
at.install(gserver)
at.install(remote_host2)

server_results_dir= "results-netperf-guest-to-host-far-server"
client_results_dir= "results-netperf-guest-to-host-far-client"

server_control_file= 'job.run_test("netperf2", "%s", "%s", "server", tag="server")' % ↵
    ↪(sh_escape(gserver.hostname), sh_escape(remote_host2.hostname),)
client_control_file= 'job.run_test("netperf2", "%s", "%s", "client", tag="client")' % ↵
    ↪(sh_escape(gserver.hostname), sh_escape(remote_host2.hostname),)

server_command= subcommand(at.run, [server_control_file, server_results_dir, gserver])
client_command= subcommand(at.run, [client_control_file, client_results_dir, remote_
    ↪host2])

parallel([server_command, client_command])
```

Autotest Server Quick Start

You can use the autoserv program located in the server directory of the Autotest tree to run tests on one or more remote machines. The machines must be configured so that you can ssh to them without being prompted for a password.

A simple example is running the sleeptest on a remote machine. Say you have two machines: On one you have

installed the Autotest code (which will be referred to as the server), and the other is a machine named mack (which will be referred to as the client).

Then you can run sleeptest on the client. Go to the top of the autotest tree:

```
server/autotest-remote -m mack -c client/tests/sleeptest/control
```

This will result in quite a bit of activity on the screen. Perhaps we log too much, but you will definitely know that something is happening. After some time the output should stop and if all went well you will see that the results directory is now full of files and directories. Before explaining that, first lets dissect the command above. The “-m” option is followed by a comma delimited list of machine names (clients) on which you wish to run your test. The “-c” option tells autoserv that this is a client side test you are running. And the last argument is the control file you wish to execute (in this case the sleeptest control file).

The results directory will generally contain a copy of the control file that is run (named control.srv). There will also be a keyval file and a status.log file. In addition there will be a debug/ directory, and a sysinfo/ directory along with a directory for each client machine (in this case a mack/ directory). The results of the test are located in the directories named for each client.

A server side control file allows the possibility of running a test that involves two or more machines interacting. An example of a server side multi-machine control file is server/tests/netperf2/control.srv. This control file requires 2 or more client machines to run. An example of how to use autoserv follows

```
server/autotest-remote -m mack,nack -s server/tests/netperf2/control.srv
```

In this example we are again running the command from the results/ directory. Here we see the “-s” option which specifies this as a server side control file. We have specified two machines using the “-m” option (mack and nack). The command should produce a flurry of activity. Afterwards you can explore the contents of the results directory to see the results. Of special note will be the contents of the mack/netperf2/results/keyval and nack/netperf2/results/keyval files. One of these files will list various performance metrics acquired by the netperf test.

Autoserv Client Install

When you install an Autotest client from a server side control file, either manually using `Autotest.install` or automatically when running a client control file using `autoserv`, `autoserv` has to determine a location on the remote host to install the client.

If you need the client installed in a specific location then the most direct solution is to pass in an `autodir` parameter to `Autotest.install` since this will disable any automatic determination and just use the provided path. However in the case that this is not possible or practical then the following sources are checked for a path and the first one found is used:

1. The result of calling `Host.get_autodir` if it returns a value
2. The dirname of the target of the `/etc/autotest.conf` symlink on the remote machine
3. `/usr/local/autotest` if it exists on the remote machine
4. `/home/autotest` if it exists on the remote machine
5. `/usr/local/autotest` even if it doesn't exist

Note that an Autotest client install will itself call `Host.set_autodir` to set it to the install location it ended up using.

Autotest server interaction with clients

Tests can be run on standalone machines, or in a server-client mode.

The server interaction is simple:

- Copy the control file across
- Execute the control file repeatedly until it completes
- Client notifies server of any reboot for monitoring
- Upon completion of control script, server pulls results back (not client push)

All interaction with the server harness will be via the *harness* object. This object provides for a per harness interface. A null interface will be provided for standalone use.

Writing server-side control files

Start with the client-side files. It's amazing how much stuff you can do with them (including reboots, etc). The client-side harness will communicate back with the server, and monitor status, etc.

However, if you want to do more powerful things, like control a complex test across a cluster, you'll probably want to use server-side control files. Read *Autotest Structure* on how the server works first, this will help explain things ...

Server-side control files have the same philosophy as the client-side files, but run on the server, so it's still a Python script, with all the flexibility that gives you. You should generally name server-side control files ending in `.srv` - that makes it a lot easier to recognize server-side control files at a glance.

You run a server-side control file by doing

```
server/autoserv -m <machine,machine,...> mycontrolfile.srv
```

We strip out the `-m` parameter, break up the comma-separated list, and put that into your namespace as a list called `"machines"`. Any extra arguments besides the control file name will appear as a list called `"args"`.

A basic control file

A simple one might do something like this:

```
host = hosts.create_host(machines[0])

print host.run("uname -a").stdout
host.reboot()
print host.run("uname -a").stdout
```

Firstly we create a `"host"` object from the machine name. That has lots of magic helpers for you, and is how you get most stuff done on the client.

After, the control file runs `"uname -a"` on the remote host, printing the output of the command. It then reboots the machine, and re-runs the `"uname -a"` command. So you will see what kernel was running on the machine when the test started, and then you will see whatever the default kernel is once the machine is rebooting, ending up with output like:

```
KERNEL VERSION AT START OF TEST
DEFAULT KERNEL VERSION
```

Running some server-side tests

Okay, so now we want to run some actual tests. The easiest kind of test to run from the server is a server-side test (i.e. something in `server/tests` or `server/site_tests`). You run it just like you would run a client-side test from a client-side

control file - with `job.run_test`. So you can run a simple sleeptest with:

```
job.run_test("sleeptest")
```

This will run sleeptest. However, it's important to remember that when you run a server-side test then it runs on the server, not on the list of machines you pass in on the `autoserv` command line. For something like a simple sleep test this doesn't really matter, but in general your test will need to manually do the setup required to run command remotely; either by creating its own host object with `create_host`, or by accepting a host object as a parameter.

Running some client-side tests

OK, so when it comes to running server-side tests we mentioned that you have to make sure your test runs all of its commands through a host object. But if all your test needs to do is run a bunch of local commands, that can make things a lot uglier; it would be easier to just run the test directly on the test machine, like you do with a client-side test.

Fortunately, just using a server-side control file it doesn't mean that you have to use server-side tests; you can write client-side tests like you normally would and still use a control file from the server-side to do whatever setup you need to do, then launch the tests on the remote machine using the Autotest client.

So, supposing we want to run some client-side tests on a remote machine. What you then need to do is:

- create a host object with `hosts.create_host`
- create an Autotest object with `autotest.Autotest`, on the remote host
- run a client-side control file on the remote host with `run` (or use the `run_test` helper for the simple case of running a single test)

You can do this like so:

```
host = hosts.create_host(machines[0])
at = autotest_remote.Autotest()(host)
at.run_test('kernbench', iterations=2)
```

This will create a host object, create an Autotest object against that host, and then run the client-side kernbench test on the remote host, using Autotest. If Autotest is not installed on the remote machine, using `at.run_test` (or `at.run`) would automatically install it first. Alternatively, if you need to explicitly control when the installation of Autotest happens you can call `at.install`.

For an example of how to use `run` instead of `run_test`, see:

```
host = hosts.create_host(machines[0])
at = autotest_remote.Autotest(host)
control = """\
job.run_test('kernbench', iterations=5)
job.run_test('dbench', iterations=5)
job.run_test('tbench', iterations=5)
job.run_test('bonnie', iterations=5)
job.run_test('iozone', iterations=5)
job.run_test('fsx')
"""
at.run(control)
```

This will produce the same effect as if you installed an Autotest client on the remote machine, created a control file like the one stored in the `control` variable, and then ran it directly with the `bin/autotest` script.

Running other existing server control files

So, sometimes instead of just running a specific test you actually have a pre-existing suite of tests you want to run. For example, suppose you have a control file for running a standard suite of fast-running performance tests that you want to incorporate into a new control file you're building. You could just look at what tests the existing suite runs and run them yourself from your new control file, but not only is that a tedious bunch of cut-and-paste work, it also means that if the "standard" suite changes you now have to go and update your new script as well.

Instead of doing that, we can just make use of the `job.run_control` method. This allows you to just run a control file directly from another control file by passing in a file name. So for example, if on your server installation you have a `test_suites/std_quick_tests` control file, you can execute it from a new one quite simply as:

```
job.run_control('test_suites/std_quick_tests')
```

The path you pass is relative to the Autotest directory (i.e. `job.autodir`). Similarly, if you wanted to run the standard sleep test control file you could do it with:

```
job.run_control('server/tests/sleeptest/control')
```

Note that variables from your current execution environment will not leak into the environment of the executed control file, and vice versa. So you cannot pass "parameters" into a control file by just setting a global variable that the executed control file then reads, and you cannot pass back results to assigning a global in the executed control file. However, this doesn't mean that the two execution environments are completely isolated; in particular, the job instance used by the executing file is the same one used by the executed file. However, as a general rule control files should avoid developing interdependencies by modifying the job object to pass information back and forth.

Using more than one machine at once

So far all the examples that have run on the remote machine have done so using `hosts.create_host(machines[0])` to create a `Host` object. However, while this is okay for just trying things out it's not a good way to write a "real" control file; if you run `autoserv` with a list of machines, you'll only ever run tests on the first one!

Now, the most obvious thing to do would be to just wrap your `machines[0]` in a for loop, but this isn't going to work very well if you run something against a hundred machines – it's going to do the runs sequentially, with 99 of the machines sitting around doing nothing at any particular point in time. Instead what you want to do is run things in parallel, like so:

```
def run(machine):
    host = hosts.create_host(machine)
    at = autotest_remote.Autotest(host)
    at.run_test('kernbench', iterations=5)

commands = [subcommand(run, args=[machine], subdir=machine) for machine in machines]
parallel(commands)
```

What this does is actually simpler than it looks; first, it defines a runs `kernbench` on one machine. Then, it defines a list of subcommands, one for each machine. Finally, it uses `parallel` to run all these commands (in parallel, via `fork`).

If you're familiar with `job.parallel` on the client, this is somewhat similar, but more powerful. The `job.parallel` method represents subcommands as a list, with the first item being a function run and the remainder being arguments to pass to it. The subcommand object is similar, taking a function and a list of args to pass to it.

In addition, `subcommand` also takes a very useful `subdir` argument to allow us to avoid mashing together all the results from each machine in the same results directory. If you specify `subdir` to a subcommand, the forked subcommand will run inside of `subdir` (creating it if it exists). So you will end up with three separate `kernbench` results in three separate machine subdirectories.

It's important to keep in mind that the final test results parser really only works well with results directories that are associated directly with a single machine, so when using parallel to do separate runs on individual machines you pretty much always want to specify a `subdir=machine` argument to your subcommands.

In fact, for this very specific case (running the exact same function on N machines) we have a special helper method, `job.parallel_simple`, doesn't require as much setup. You could replace the above code with the simpler:

```
def run(machine):
    host = hosts.create_host(machine)
    at = autotest_remote.Autotest(host)
    at.run_test('kernbench', iterations=5)

job.parallel_simple(run, machines)
```

Synchronous vs Asynchronous jobs

If you run control files through the frontend, it needs to know how you want them to be run.

Let's say there's 6 clients we're controlling. We could either run asynchronously, with a separate autoserv instance controlling each machine. If you do this, it will kick off separate autoserv instances as each machine becomes available. We ask for this by specifying `SYNC_COUNT=1`

```
autoserv control_file -m machine1
autoserv control_file -m machine2
autoserv control_file -m machine3
autoserv control_file -m machine4
autoserv control_file -m machine5
autoserv control_file -m machine6
```

Or we can run synchronously. If you do that, we'll wait for **all** the machines you asked for before starting the job, and do something like this:

```
autoserv control_file -m machine1,machine2,machine3,machine4,machine5,machine6
```

Often we only need to pair up machines (say 1 client and 1 server to run a network test). But we don't want to wait for all 6 machines to be available; as soon as we have 2 ready, we might as well kick those off. We can use `SYNC_COUNT` to specify how many we need at a time, in this case `SYNC_COUNT=2`. We'll end up doing something like this:

```
autoserv control_file -m machine1,machine2
autoserv control_file -m machine3,machine4
autoserv control_file -m machine5,machine6
```

Installing kernels from a server-side control file

So, if you've written a client-side control file for installing a kernel, you're probably familiar with code that looks something like:

```
testkernel = job.kernel('/usr/local/mykernel.rpm')
testkernel.install()
testkernel.boot()
```

This will install a client on the local machine. Well, we've also seen that in a server-side control file, unless you use a Host object to run commands then your operations run on the server, not your test machine(s). So just trying to use the same code won't work.

However, we've already seen that you can use an Autotest object to run arbitrary client-side control files on a remote machine. So you can instead use some code like this:

```
kernel_install_control = """
def step_init():
    job.next_step([step_test])
    testkernel = job.kernel('/usr/local/mykernel.rpm')

    testkernel.install()
    testkernel.boot()

def step_test():
    pass
"""

def install_kernel(machine):
    host = hosts.create_host(machine)
    at = autotest_remote.Autotest(host)
    at.run(kernel_install_control, host=host)
job.parallel_simple(install_kernel, machines)
```

This will install `/usr/local/mykernel.rpm` on all the machines you're running your test on, all in parallel. You can then follow up this code in your control file with the code to run your actual tests.

The Host classes

There are six main classes in the Host hierarchy, with two concrete classes that can be instantiated; one that uses the OpenSSH binary for executing commands on a remote machine, and one that uses the Paramiko module to do the same. The specific classes are:

- `Host` - the top-level abstract base class, contains definitions for most of the standard Host methods, as well as implementations for some of the high-level helper methods.
- `RemoteHost` - a subclass of `Host` that also adds some options specific to “remote” machines, such as having a hostname, as well as providing generic reboot and crashinfo implementations.
- `SiteHost` - a subclass of `RemoteHost` that allows you to hook site-specific implementation behavior into your `Host` classes. This may not even be defined (in which case we automatically default to providing a empty definition) but can be used to insert hooks into any methods you need. An example of such a use would be adding a `machine_install` implementation that takes advantage of your local installer infrastructure and so isn't suitable for inclusion into the core classes.
- `AbstractSSHHost` - a subclass of `SiteHost`, this provides most of the remaining implementation needed for using ssh-based interaction with a remote machine such as the ability to copy files to and from the remote machine as well as an implementation of the various `wait_*` methods
- `SSHHost` - one of the concrete subclasses of `AbstractSSHHost`, this class can be directly instantiated. It provides an implementation of `Host.run` based around using an external `ssh` binary (generally assumed to be OpenSSH). This is also currently the default implementation used if you're using the factory to create the method rather than creating `Host` instance directly.
- `ParamikoHost` - the other concrete subclass of `AbstractSSHHost`. This class provides a lower-overhead, better-integrated alternative to the `SSHHost` implementation, with some caveats. In order to use this class directly you'll need to explicitly create an instance of the class, or use custom hooks into the host factory. Note that using this class also requires that you have the `paramiko` library installed, as this module is not included in the Python standard library.

Creating instances of Host classes

The concrete host subclasses (`SSHHost`, `ParamikoHost`) can both be instantiated directly, by just creating an instance. Both classes accept `hostname`, `user` (defaults to `root`), `port` (defaults to `22`) and `password` (nothing by default, and ignored if connecting using ssh keys). So the simplest way to create a host is just with a piece of code such as:

```
from autotest_lib.server.hosts import paramiko_host

host = paramiko_host.ParamikoHost("remotemachine")
```

However, there are several disadvantages to this method. First, it ties you to a specific SSH implementation (which you may or may not care about). Second, it loses out on support for the extra mixin Host classes that Autotest provides. So the preferred method for creating a host object is:

```
from autotest_lib.server import hosts

host = hosts.create_host("remotemachine")
```

The `create_host` function passes on any extra arguments to the core host classes, so you can still pass in `user`, `port` and `password` options. It also accepts additional boolean parameters, `auto_monitor` and `netconsole`.

If you use `create_host` to build up your instances, it also mixes in some extra monitoring classes provided by Autotest. Specifically, it mixes in `SerialHost` and/or `LogfileMonitorMixin`, depending on what services are available on the remote machine. Both of these classes provide automatic capturing and monitoring of the machine (via `SerialHost` if the machine has a serial console available via `conmux`, via monitoring of `/var/log/kern.log` and `/var/log/messages` otherwise). If `netconsole=True` (it defaults to `False`) then we will also enable and monitor the network console; this is disabled by default because network console can interact badly with some network drivers and hang machines on shutdown.

If for some reason you want this monitoring disabled (e.g. it's too heavyweight, or you already have some monitoring of the host via alternate machines) then it can still be disabled by setting `auto_monitor=False`. This allows you to still use `create_host` to automatically select the appropriate host class; by default this still just uses `SSHHost`, but in the future it may change. Or, your server may be using custom site hooks into `create_host` which already change this behavior anyway.

Custom hooks in create_host

You can optionally define a `site_factory.py` module with a `postprocess_classes` function. This takes as its first parameter a list of classes that will be mixed together to create the host instance, and then a complete copy of the args passed to `create_host`. This function can then modify the list of classes (in place) to customize what is actually mixed together. For example if you wanted to default to `ParamikoHost` instead of `SSHHost` at your site you could define a site function:

```
from autotest_lib.server.hosts import ssh_host, paramiko_host

def postprocess_classes(classes, **args):
    if ssh_host.SSHHost in classes:
        classes[classes.index(ssh_host.SSHHost)] = paramiko_host.ParamikoHost
```

This will change the factory to use `ParamikoHost` by default instead. Or you could do other changes, for example disabling `SerialHost` completely by removing it from the list of classes. Or you could do something even more complex, like using `ParamikoHost` if a host supports it and falling back to `SSHHost` otherwise. Adding additional args to `postprocess_classes` is also an option, to add more user-controllable host creation, but keep in mind that such extensions can then only be used in site-specific files and tests.

Paramiko vs OpenSSH

Why do we provide two methods of connecting via ssh at all? Well, there are a few advantages and disadvantages to both.

Why openssh?

If we use openssh then we generally have more portability and better integration with the users configuration (via `ssh_config`). This is also more configurable in general, from an external point of view, since a user can customize ssh behavior somewhat just by tweaking `~/.ssh/config`

So why paramiko?

However, there are also limitations that come up with openssh. It mostly operates as a black box; all we can do to detect network- or ssh-level issues is to watch for a 255 exit code from ssh, and to attempt to break things down into authentication issues versus various connection issues we have to try and parse the output of the program itself, output which may be mixed in with the output of the remote command.

There can also be performance issues when openssh is in use, due to the large number of processes that can end up being spawned to run ssh commands; even if most of this memory is cached and shared the memory costs start to pile up. Additionally the cost of creating new connections for every single ssh command can start to pile up.

Paramiko alleviates these problems by moving the ssh handler in-process as a python library, and taking advantage of the multi-session support in SSH protocol 2 to run multiple commands over a single persistent connection. However, it has the cost of requiring that you use a protocol 2 sshd on the remote machine, and requires installing the paramiko library. It also has much weaker support for `ssh_config`, with some support for finding keyfiles (via `IdentityFile`?) and nothing else.

Setting up ParamikoHost

There are two main issues you need to resolve to use ParamikoHost, 1) installing paramiko and 2) making sure you have support for protocol 2 connections.

Point one is fairly straightforward, just refer to one of the bullet points in *autotest server install* that explains how to install paramiko.

Point two is a bit more complex. There's a fairly good chance your infrastructure already supports protocol 2, since it's been around for quite a long time now and is generally considered to be the standard. To test it, just try connecting to a machine via ssh using the `-o Protocol=2` option; if it succeeds then ParamikoHost should just work once the point one is taken care of. If it fails with an error message about protocol major version numbers differing, then you're in trouble; you'll need to reconfigure sshd on your remote machines to support protocol 2, and if you're using key-based authentication you'll need to add support for protocol 2 keys as well. If these configuration changes are not practical (either for technical or organizational reasons) then you'll simply have to forgo the use of ParamikoHost.

Standard Methods

The Host classes provide a collection of standard methods for running commands on remote machines, copying files to and from them, and rebooting them (for remote machines).

Host.run

This method can be used to run commands on a host via an interface like that of the run function in the utils module. It returns a `CmdResult?` object just like `utils.run`, and supports the `ignore_status`, `timeout` and `std*_tee` methods with the same semantics.

Host.send_file, Host.get_file

These methods allow you to copy file(s) and/or directory(s) to a remote machine. You can provide a single path (or a list of paths) as a source and a destination path to copy to, with `send_file` for destinations on the host and `get_file` for sources on the host. The pathname semantics are intended to mirror those of `rsync` so that you can specify “the contents of a directory” by terminating the path with a `/`.

Host.reboot, Host.reboot_setup, Host.reboot_followup, Host.wait_up, Host.wait_down

The reboot method allows you to reboot a machine with a few different options for customizing the boot:

- `timeout` - allows you to specify a custom timeout in seconds. Used when you want reboot to automatically wait for the machine to restart (the default). If the reboot takes longer than `timeout` seconds to come back after shutting down then an exception will be thrown.
- `label` - the kernel label, used to specify what kernel to boot into. Defaults to `host.LAST_BOOT_TAG` which will reboot into whatever kernel the host was last booted into by Autotest (or the default kernel if Autotest has not yet booted the machine in the job).
- `kernel_args` - a string of extra kernel args to add to the kernel being booted, defaults to none (which means no extra args will be added)
- `wait` - a boolean indicating if reboot should wait for the machine to restart after starting the boot, defaults to true. If you set this to False then if you try to run commands against the Host it'll just time out and fail, and the `reboot_followup` method won't be called.
- `fastsync` - if True (default is False) don't try to sync and wait for the machine to shut down cleanly, just shut down. This is useful if a faster shutdown is more important than data integrity.
- `reboot_cmd` - an optional string that lets you specify your own custom command to reboot the machine. This is useful if you want to specifically crank up (or turn down) the harshness of the shutdown command.

In addition to `reboot`, there are two hooks (`reboot_start` and `reboot_followup`) that are called before and after the reboot is run. This allows you to define mixins (like `SerialHost` and some other classes we'll mention later) that can hook into the reboot process without having to implement their own `reboot`.

Finally, there are `wait_down` and `wait_up` methods, specifically for waiting for a rebooting machine to shut down or come up. If you use the `reboot` method these should generally be only used internally, but you can use them yourself directly if you need more custom control of the powering up and/or down of the machine.

Synchronize clients in multi machine (server) tests

Synchronization is useful when is started server part test which starts client part test on multiple hosts, then is sometimes needed to synchronize state or data between client part tests. By this reason was created class **Barrier** and class **Syncdata**. Both classes are placed in `autotest/client/shared`.

class Barrier

Barrier allows only state synchronization. Both clients start:

```
job.barrier(host_name, tag, timeout)
```

Where:

host_name Host identifier (host_ip | host_name[#optional_tag]).

tag Identifier of barrier.

timeout Timeout for barrier.

Usage:

```
b = job.barrier(ME, 'server-up', 120) # Create barrier object
b.rendezvous(CLIENT, SERVER)        # Block test(thread) until barrier is reached
                                     # by all sides or barrier timeouted.
```

Where **ME** depends where is this code started. It could be **CLIENT** or **SERVER**. The same code is started all hosts which waits for barrier.

Communication:

```
MASTER                CLIENT1                CLIENT2
<-----TAG C1----->
-----wait----->
      [...]
<-----TAG C2----->
-----wait----->
      [...]
-----ping----->
<-----pong----->
-----ping----->
<-----pong----->
      ----- BARRIER conditions MET -----
-----rlse----->
-----rlse----->
```

Master side creates socket server. Client side connects to this server and communicate through them. During waiting, the barrier checks if all sides which wait for barrier are alive. For the checking barrier uses ping-pong messages.

class SyncData

SyncData class allows synchronization of state and data but it not check liveness of synchronized nodes. When one node dies after sending his data, others nodes know nothing about death of node. Information about death is logged to log. SyncData class could be use instead class Barrier.

```
SyncData(master_id, hostid, hosts, session_id, sync_server)
```

Where:

master_id master host identifier. This host has or create sync_server and others connect to them.

hostid host identifier.

hosts list of all host which should exchange data.

session_id session_id identifies data synchronization. Session_id must be unique.

sync_server If sync_server is None then master create new sync_server for synchronization.

Usage:

```
from autotest.client.shared.syncdata import SyncData

master_id = MASTER
sync = SyncData(master_id, hostid, hosts,
                session_id, tag)

data = sync.sync(data, timeout, session_id) # sync could be run in different threads
                                           # with different session_id_
↳simultaneously.                          # session_id there override session_id_
↳defined in                               # class definition. session_id could be_
↳None.

data_hostid2 = data[hostid2]              # data = {hostid1: data1, hostid2: data2}
```

sync return dictionary with data from all clients.

Communication:

```
MASTER          CLIENT1          CLIENT2
if not listen_server -> create

<-----session_id/hosts/timeout----->
<-----data1----->
[... ]
<-----session_id/hosts/timeout----->
<-----data2----->
-----{hostid1: data1, hostid2: data2}----->
<-----BYE----->
-----{hostid1: data1, hostid2: data2}----->
<-----BYE----->
```

Server waits for data from all clients and then sends data to all clients.

Autoserv message logging specification

1. All output for the job, and any tests in it should go in debug/
2. All output within a parallel_simple() subcommand should *also* go in \$hostname/debug (for parallel_simple() over hostnames)
3. All output during any test should *also* go in \$testname/debug/
4. We should not buffer beyond one message
5. All lines in the output should be tagged with the logging prefix (for multi-line messages, that means one tag per line, so grep works)
 - the prefix is “[m/d H:M:S level module]”, i.e. “[06/08 16:39:17 DEBUG utils]”
6. All output from subcommands is logged, by default at DEBUG level for stdout and ERROR level for stderr
7. All print statements to stdout/stderr get logged with levels DEBUG and ERROR respectively. Ideally we’d like to convert all print statements into logging calls but that probably won’t happen any time soon.
8. In each debug/ directory, there are two log files kept:

- All debug level messages and above in `autoserv.stdout`
- All error level messages and above in `autoserv.stderr`

Conmux - Console Multiplexor

Conmux is a console multiplexor. It can:

- Connect to a serial console or network console
- Allow multiple users to connect to the console session at once, and share that session
- Control power strips etc (via expect scripts) - these are abstracted through commands like “`~$hardreset`”

Manual usage:

```
console <machinename>
```

Conmux HOWTO - A walkthrough for setting up a conmux server and creating console configurations

[Original Documentation](#)

Installing a Conmux Server

This document will explain how to install a conmux server starting from the Autotest codebase. A rudimentary configuration for an example console will also be provided

Installing the conmux server

This assumes that you already have a freshly sync'd version of Autotest as defined in: [Downloading The Source](#) or that you are using one of the release tarballs. A lot of this is covered in the [autotest/conmux/INSTALL](#) file.

Required perl modules:

- `IO::Multiplex`;
 - Debian/Ubuntu? Packages: `libio-multiplex-perl`
 - Fedora Packages: `perl-IO-Multiplex`

Installing `IO::Multiplex` via CPAN:

```
perl -MCPAN -e 'install IO::Multiplex'
```

Building

This section describes how to get the conmux system in to the place you want it installed on your system. The default location is `/usr/local/conmux`

To make and install this package to the default location

```
make install
```

To an alternative location:

```
make PREFIX=/usr/alt/conmux install
```

To build for a specified prefix, but installed into a temporary tree:

```
make PREFIX=/usr/alt/conmux BUILD=build/location install
```

Console configuration

This will walk through some configurations for consoles in conmux. Each configuration has a listener, payload and optionally one or more panel commands. Configuration is provided via a per console configuration file.

- All configurations are stored in `BASE_INSTALL/etc` with a `.cf` extension (e.g. `dudicus.cf`)

listener:

```
**listener server/name** defines the name of this console port as it appears in the registry.
```

payload:

```
**socket name title host:port** defines a console payload connected to a tcp socket on the network. name defines this payload within the multiplexor, title is announced to the connecting clients.
```

```
**application name title cmd** defines a console payload which is accessed by running a specific command. name defines this payload within the multiplexor, title is announced to the connecting clients.
```

command panel:

```
**command panel message cmd** defines a panel command for the preceding payload, triggered when panel is typed at the command prompt. message is announced to the user community. cmd will be actually executed.
```

Example Config

A conmux configuration using a socket to connect to the console

```
listener localhost/dudicus
socket console 'dudicus' '192.168.0.3:23'
```

Example with an application:

A very basic example of starting an application (which could be any application including ones that connect to a proprietary protocol). This is more just to show how this feature would be used.

```
listener localhost/cat
application console 'cat' '/bin/cat'
```

Not that in the above examples the listener is set to `localhost`. That states that the localhost is where the consoles are started and where the `conmux_registry` exists. If you are running lots of consoles you may want to have one central registry and a number of different machines providing access to them if that were the case you would want to set localhost to the hostname where the conmux registry is running.

Conmux configuration with hardreset

Adding a hardreset command, if you aren't familiar with the Autotest Hardreset please refer to that for terminology. There are a number of different expect scripts/python pexpect scripts available in conmux/lib/drivers (on the installed server) each one of these connects to an RPM in their own way. A unified solution is being worked on but it is low priority. Basically the customer needs to give you the information required as outlined in the hardreset documentation and then you identify which script to use by connecting to the RPM and looking for brandings like SENTRY or CITRIX etc.

```
listener localhost/dudicus
socket console 'dudicus' '192.168.0.3:23'
command 'hardreset' 'initiated a hard reset' 'reboot-cyclades 192.168.0.12 48 user_
↳password 5'
```

Conmux doesn't really care what it is calling here it is just a program with parameters, to understand how to use the reboot-cyclades driver you need to actually open up the file and read it.

Generic command Below is an example of a generic command. Commands are issued using the ~\$ escape sequence and then the command name. An example of a useful command would be one to show the configuration of the console you are connected to:

Add the following to your config.cf file:

```
"command 'config' 'Show conmux configuration' 'cat /home/conmux/etc/dudicus.cf'
```

Example output:

```
[/usr/local/conmux/bin]$. /console netcat
Connected to netcat [channel transition] (~$quit to exit)

Command(netcat)> config
(user:me) Show conmux configuration
listener localhost/netcat
socket console 'netcat' 'localhost:13467'
command 'config' 'Show conmux configuration' 'cat /usr/local/conmux/etc/netcat.cf'
```

Starting the Conmux Server

Conmux comes with a bash script that will do the following

- Start the conmux registry
- Start all configurations in BASE_INSTALL/etc that end with .cf prefixes
- Restart consoles that died since the last start command
- Restart consoles whose configuration has changed since the last start command
- Log console output in BASE_INSTALL/log

To start the conmux registry and all the consoles issue the following command

```
BASE_INSTALL/sbin/start
```

Example output:

```
/usr/local/conmux/sbin/start
starting registry ...
```

```
starting CONSOLE1 ...
starting CONSOLE2 ...
```

Mock Console Setup using nc

After following all of the above this section provides a concrete example for users who do not currently have access to any console hardware. In this section a configuration will be setup for a console on localhost. Netcat will be used on the machine to listen to the port for a connection so that an actual console connection can be created.

The configuration:

etc/netcat.cf

```
listener localhost/netcat
socket console 'netcat' 'localhost:13467'
command 'config' 'Show conmux configuration' 'cat /usr/local/conmux/etc/netcat.cf'
```

Start netcat in a different terminal listening on port 13467

```
nc -l -p 13467
```

Start your conmux server

```
BASE_INSTALL/sbin/start
```

Now connect to the console:

```
BASE_INSTALL/bin/console netcat
```

Output should be similar to:

```
/usr/local/conmux/bin]$. /console netcat
Connected to netcat [channel connected] (~$quit to exit)
```

If you start typing in here you will notice in the terminal where netcat is running what you typed and vice versa.

You can also issue the *config* command by using *~\$* and inputting *config*

Conmux - Original Documentation

conmux, the console multiplexor is a system designed to abstract the concept of a console. That is to provide a virtualised machine interface, including access to the console and the 'switches' on the front panel; the */dev/console* stream and the reset button. It creates the concept of a virtual console server for multiple consoles and provides access to and sharing of consoles connected to it.

There are two main motivations for wanting to do this. Firstly, we have many different machine types with vastly differing access methodologies for their consoles and for control functions (VCS, HMC, Annex) and we neither want to know what they are nor how they function. Secondly, most console sources are single access only and we would like to be able to share the console data between many consumers including users. Basic Usage

The main interface to the consoles is via the console program. This connects us to the console server for the machine and allows us to interact with it, including issuing out-of-band commands to control the machine.

```
$ console <host>/<console>
```

In the example below we indicate that the console we require is located on the virtual console server `consoles.here.com` and the specific console is `elm3b70`.

```
$ console consoles.here.com/elm3b70
Connected to elm3b70 console (~$quit to exit) Debian GNU/Linux 3.1 elm3b70 ttyS0
elm3b70 login:
```

Once connected we can interact normally with the console stream. To perform front pannel operation such as performing an hard reset we switch to command mode. This is achieved using the escape sequence `~$`. Note the prompt `Command>`

```
elm3b70 login: ~$
Command> quit
Connection closed $
```

Command Summary

The following commands are generally available:

Com-mand	Description
quit	quit this console session, note that this disconnects us from the session it does not affect the integrity of the session itself.
hardreset	force a hard reset on the machine, this may be a simple reset or a power off/on sequence whatever is required by this system.

Architecture

The `conmux` provides a virtual console multiplexor system reminiscent of an Annex terminal server. You refer to the `conmux` server and lines, unlike an Annex lines are referred to by mnemonic names. Above we referred to the console for `elm3b70` ‘connected to’ the server `consoles.here.com`. A virtual console server consists of a number of server processes. One `conmux-registry` server, several `conmux` servers and optionally several helper processes.

`conmux-registry`: a server is defined by the server registry. This maintains the mnemonic name to current server location relation. When a client wishes to attach to a console on a server, the registry is first queried to locate the server currently handling that console.

`conmux`: for each connected console there is a corresponding console multiplexor. This process is responsible for maintaining the connection to the console and for redistributing the output to the various connected clients. It is also responsible for handling “panel” commands from the client channels.

`autoboot-helper`: an example helper which aids systems which are not capable of an automatic reboot. It connects to a console and watches for tell-tale reboot activity, performing a “panel” `hardreset` when required. This provides the impression of seamless reboot for systems which this does not work. Configuration `conmux-registry`

Configuration of this service is very simple. Supplying the default registry port (normally 63000) and the location for the persistant registry database. `conmux`

Configuration of each `conmux` is complex. Each has a listener, payload and optionally one or more panel commands. Configuration is provided via a per console configuration file. This file consists of lines defining each element:

```
listener <server>/<name>: defines the name of this console port as it appears in the ↵
↵registry.

socket <name> <title> <host>:<port>: defines a console payload connected to a tcp ↵
↵socket on the network. name defines this payload within the multiplexor, title is ↵
↵announced to the connecting clients.
```

```
application <name> <title> <cmd>: defines a console payload which is accessed by
↳running a specific command. name defines this payload within the multiplexor, title
↳is announced to the connecting clients.
```

```
command <panel> <message> <cmd>: defines a panel command for the preceeding payload,
↳triggerd when panel is typed at the command prompt. message is announced to the
↳user community. cmd will be actually executed.
```

For example here is the configuration for a NUMA-Q system which is rebooted using a remote VCS console and for which the real console channel is on an Annex terminal server:

```
listener localhost/elm3b130
socket console 'elm3b130 console' console.server.here.com:2040
command 'hardreset' 'initated a hard reset' \ './reboot-numaq vcs 1.2.3.4 elm3b130
↳12346 Administrator password'
```

ACL Behavior Reference

The following is a reference for the actions that ACLs restrict.

Hosts

- Users must be in some ACL with a host to modify or delete the host and to add the host to an ACL group.

Jobs

- For jobs scheduled against individual hosts, the user must be in some ACL with the host.
- The owner of a job may abort the job. Any other user with ACL access to a host can abort that host for any job, *unless* the host is in the 'Everyone' ACL.

ACL Groups

- To add or remove users/hosts in an ACL, the user must be a member of that ACL.
- The 'Everyone' ACL cannot be modified or deleted.
- When a host is added to an ACL other than 'Everyone', it is automatically removed from 'Everyone'. As long as it is a member of some other ACL it will always be automatically removed from 'Everyone'.
- When a host is removed from all ACL, it is automatically added to 'Everyone'.

Superusers

Superusers can bypass most of these restrictions. The only thing a superuser cannot do is delete the 'Everyone' group. To create a superuser, run the script at `<autotest_root>/frontend/make_superuser.py`, with the username as a command-line parameter.

Frontend

Autotest Command Line Interface

Autotest provides a set of commands that can be used to manage the autotest database, as well as schedule and manage jobs.

The commands are in the `./cli` directory.

The main command is called 'autotest-rpc-client'. The general syntax is:

```
autotest-rpc-client <topic> <action> <items> [options]
```

Where:

- topic is one of: acl, host, job, label or user
- action is one of: create, delete, list, stat, mod, add, rm. Not all the actions are available for all topics.

Topic References

The references for the different topics are available for acl?, label?, host?, user?, test? and job? management

Common options

The options common to all commands are:

- `help`: displays the options specific to the topic and/or action. It can be used as:
 - `autotest-rpc-client help`
 - `autotest-rpc-client <topic> help`
 - `autotest-rpc-client <topic> <action> help`
- `-w|--web`: specifies the autotest server to use (see below).
- `--parse`: formats the output in colon separated key=values pairs.
- `--kill-on-failure`: stops processing the arguments at the first failure. Default is to continue and displays the failures at the end.
- `-v|--verbose`: Displays more information.

Server Access

By default, the commands access the server at: `http://autotest`. This can be overwritten by setting the `AUTOTEST_WEB` environment variable or using the `-w|--web` option using only the hostname. The order of priority is:

1. the command line option,
2. the `AUTOTEST_WEB` environment variable
3. the default 'autotest' server.

Wildcard

The `list` action accepts the `*` wildcard at the end of a filter to match all items starting with a pattern. It may be necessary to escape it to avoid the `*` to be interpreted by the shell.

```
# autotest-rpc-client host list host1\*
Host      Status Locked Platform Labels
host1     Ready  False
host12    Ready  False
host13    Ready  False
host14    Ready  False
host15    Ready  False
```

File List Format

Several options can take a file as an argument. The file can contain space- **or** comma-separated list of items e.g.,

```
# cat file_list
host0
host1
host2,host3
host4 host5
```

Note the `host1, host2` (comma **and** space) is not a valid syntax

Access Control List Management - `autotest-rpc-client acl`

The following actions are available to manage the ACLs:

```
# autotest-rpc-client acl help
usage: autotest-rpc-client acl [create|delete|list|add|rm] [options] <acls>
```

Creating an ACL

```
# autotest-rpc-client acl create help
usage: autotest-rpc-client acl create [options] <acls>

options:
  -h, --help          show this help message and exit
  -g, --debug         Print debugging information
  --kill-on-failure   Stop at the first failure
  --parse             Print the output using colon separated key=value
                    fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                    Specify the autotest server to talk to
  -d DESC, --desc=DESC Creates the ACL with the DESCRIPTION
```

Only one ACL can be create at a time. You must specify the ACL name and its description:

```
# autotest-rpc-client acl create my_acl -d "For testing" -w autotest-dev
Created ACL:
    my_acl
```

Deleting an ACL

```
# autotest-rpc-client acl delete help
usage: autotest-rpc-client acl delete [options] <acls>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -a ACL_FLIST, --alist=ACL_FLIST
                       File listing the ACLs
```

You can delete multiple ACLs at a time. They can be specified on the command line or in a file, using the `-a|--alist` option.

```
autotest-rpc-client acl delete my_acl,my_acl_2
Deleted ACLs:
    my_acl, my_acl_2
```

Listing ACLs

```
# autotest-rpc-client acl list help
usage: autotest-rpc-client acl list [options] <acls>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -a ACL_FLIST, --alist=ACL_FLIST
                       File listing the ACLs
  -u USER, --user=USER List ACLs containing USER
  -m MACHINE, --machine=MACHINE
                       List ACLs containing MACHINE
```

You can list all the ACLs, or filter on specific ACLs, users or machines (exclusively). The `--verbose` option provides the list of users and hosts belonging to the ACLs.

```
# autotest-rpc-client acl list -w autotest-dev
Name                Description
Everyone
reserved-qual       Qualification machines
benchmarking_group  Benchmark machines
my_acl               For testing

# autotest-rpc-client acl list -v -w autotest-dev
Name                Description
```

```

Everyone
Hosts:
    qual0, qual1, qual2, qual3, qual4, host0, host1, host2, host3, host4
    bench0, bench1, bench2, bench3, bench4, test0
Users:
    user0, user1, user2, user3, user4

reserved-qual          Qualification machines
Hosts:
    qual0, qual1, qual2, qual3, qual4
Users:
    user0

benchmarking_group     Benchmark machines
Hosts:
    bench0, bench1, bench2, bench3, bench4
Users:
    user1, user2

my_acl                 For testing

# autotest-rpc-client acl list -w autotest-dev -u user0
Name          Description
Everyone
reserved-qual  Qualification machines

# autotest-rpc-client acl list -w autotest-dev -m bench0 -v
Name          Description
Everyone
benchmarking_group  Benchmark machines
Hosts:
    bench0, bench1, bench2, bench3, bench4
Users:
    user1, user2

```

Adding Hosts or Users to an ACL

```

# autotest-rpc-client acl add help
usage: autotest-rpc-client acl add [options] <acls>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -a ACL_FLIST, --alist=ACL_FLIST
                       File listing the ACLs
  -u USER, --user=USER Add USER(s) to the ACL

```

```
--ulist=USER      File containing users to add to the ACL
-m MACHINE, --machine=MACHINE
                  Add MACHINE(s) to the ACL
--mlist=MACHINE   File containing machines to add to the ACL
```

You must specify at least one ACL and one machine or user.

```
# autotest-rpc-client acl add my_acl -u user0,user1 -v -w autotest-dev
Added to ACL my_acl user:
    user0, user1

# cat machine_list
host0 host1
host2
host3,host4

# autotest-rpc-client acl add my_acl --mlist machine_list -w autotest-dev
Added to ACL my_acl hosts:
    host0, host1, host2, host3, host4

# autotest-rpc-client acl list -w autotest-dev -v my*
Name    Description
my_acl  For testing
Hosts:
    host0, host1, host2, host3, host4
Users:
    user0, user1
```

Note the usage of wildcard to specify the ACL in the last example: my*

Removing Hosts or Users from an ACL

```
# autotest-rpc-client acl rm help
usage: autotest-rpc-client acl rm [options] <acls>

options:
  -h, --help          show this help message and exit
  -g, --debug         Print debugging information
  --kill-on-failure   Stop at the first failure
  --parse             Print the output using colon separated key=value
                    fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                    Specify the autotest server to talk to
  -a ACL_FLIST, --alist=ACL_FLIST
                    File listing the ACLs
  -u USER, --user=USER Remove USER(s) from the ACL
  --ulist=USER       File containing users to remove from the ACL
  -m MACHINE, --machine=MACHINE
                    Remove MACHINE(s) from the ACL
  --mlist=MACHINE    File containing machines to remove from the ACL
```

The options are the same than for adding hosts or users. You must specify at least one ACL and one machine or user.

```
# autotest-rpc-client acl rm my_acl -m host3 -w autotest-dev
Removed from ACL my_acl host:
```

```

host3
# autotest-rpc-client acl rm my_acl -u user0 -v -w autotest-dev
Removed from ACL my_acl user:
    user0

# autotest-rpc-client acl list -w autotest-dev -v my_*
Name      Description
my_acl    For testing
Hosts:
    host0, host1, host2, host4
Users:
    user1

# autotest-rpc-client acl delete my_acl -w autotest-dev
Deleted ACL:
    my_acl

```

Possible errors and troubleshooting

In case of error, add the `-v` option to gather more information.

Duplicate ACL:

```

# autotest-rpc-client acl create my_acl -d "For testing" -w autotest-dev
Operation add_acl_group failed for: my_acl

# autotest-rpc-client acl create my_acl -d "For testing" -w autotest-dev -v
Operation add_acl_group failed for: my_acl
    ValidationError: {'name': 'This value must be unique (my_acl)'}

```

Adding an unknown user or host:

```

# autotest-rpc-client acl add my_acl -u foo
Operation acl_group_add_users failed for: my_acl (foo)

# autotest-rpc-client acl add my_acl -u foo -v
Operation acl_group_add_users failed for: my_acl (foo)
    DoesNotExist: User matching query does not exist.

```

Removing an ACL requires that you are part of this ACL:

```

# autotest-rpc-client acl delete my_acl -w autotest-dev
Operation delete_acl_group failed for: my_acl

# autotest-rpc-client acl delete my_acl -w autotest-dev -v
Operation delete_acl_group failed for: my_acl
    AclAccessViolation: You do not have access to my_acl

# Adding yourself to the ACL:
# autotest-rpc-client acl add -u mylogin my_acl -w autotest-dev
Added to ACL my_acl user:
    mylogin

# autotest-rpc-client acl delete my_acl -w autotest-dev
Deleted ACL:
    my_acl

```

Host Management - autotest-rpc-client host

NOTE: THIS IS ONLY PARTIALLY DONE.

The following actions are available to manage hosts:

```
# autotest-rpc-client host help
Usage: autotest-rpc-client host [create|delete|list|stat|mod|jobs] [options] <hosts>

Options:
  -h, --help           show this help message and exit
  -g, --debug          Print debugging information
  --kill-on-failure    Stop at the first failure
  --parse              Print the output using colon separated key=value
                      fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                      Specify the autotest server to talk to
  -M MACHINE_FLIST, --mlist=MACHINE_FLIST
                      File listing the machines
```

Creating a Host

```
# autotest-rpc-client host create help
usage: autotest-rpc-client host create [options] <hosts>

options:
  -h, --help           show this help message and exit
  -g, --debug          Print debugging information
  --kill-on-failure    Stop at the first failure
  --parse              Print the output using colon separated key=value
                      fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                      Specify the autotest server to talk to
  --mlist=MACHINE_FLIST
                      File listing the machines
  -l, --lock           Create the hosts as locked
  -u, --unlock         Create the hosts as unlocked (default)
  -t PLATFORM, --platform=PLATFORM
                      Sets the platform label
  -b LABELS, --labels=LABELS
                      Comma separated list of labels
  --blist=LABEL_FLIST File listing the labels
  -a ACLS, --acls=ACLS Comma separated list of ACLs
  --alist=ACL_FLIST   File listing the acls
```

Multiple hosts can be created with one command. The hostname(s) can be specified on the command line or in a file using the `--mlist` option.

You can specify the platform type, labels and ACLs for all the newly added hosts. If you want the hosts to be locked, specify `--locked` flag. The scheduler will not assign jobs to a locked host.

```
# cat /tmp/my_machines
host0
host1
```

```
# Create 2 hosts, locked and add them to the my_acl ACL.
# autotest-rpc-client host create --mlist /tmp/my_machines -a my_acl -l
Added hosts:
    host0, host1
```

Deleting a Host

```
# autotest-rpc-client host delete help
usage: autotest-rpc-client host delete [options] <hosts>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  --mlist=MACHINE_FLIST
                       File listing the machines
```

Multiple hosts can be deleted with one CLI. The hostname(s) can be specified on the command line or in a file using the `--mlist` option.

```
# The list can be comma or space separated.
# autotest-rpc-client host delete host1,host0 host2
Deleted hosts:
    host0, host1, host2
```

Listing Hosts

```
# autotest-rpc-client host list help
Usage: autotest-rpc-client host list [options] <hosts>

Options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -M MACHINE_FLIST, --mlist=MACHINE_FLIST
                       File listing the machines
  -b LABEL, --label=LABEL
                       Only list hosts with this label
  -s STATUS, --status=STATUS
                       Only list hosts with this status
  -a ACL, --acl=ACL     Only list hosts within this ACL
  -u USER, --user=USER Only list hosts available to this user
```

You can which host(s) you want to display using a combination of options and wildcards.

```
# List all the hosts
# autotest-rpc-client host list
Host    Status  Locked  Platform  Labels
host1   Ready   True      label1
host0   Ready   True      label0
mach0   Ready   True
mach1   Ready   True

# Only hosts starting with ho
# autotest-rpc-client host list ho\*
Host    Status  Locked  Platform  Labels
host1   Ready   True      label1
host0   Ready   True      label0

# Only hosts having the label0 label
# autotest-rpc-client host list -b label0
Host    Status  Locked  Platform  Labels
host0   Ready   True      label0

# Only hosts having a label starting with lab
# autotest-rpc-client host list -b lab\*
Host    Status  Locked  Platform  Labels
host1   Ready   True      label1
host0   Ready   True      label0

# Only hosts starting with ho and having a label starting with la
# autotest-rpc-client host list -b la\* ho\*
Host    Status  Locked  Platform  Labels
host1   Ready   True      label1
host0   Ready   True      label0
```

Getting Hosts Status

```
# autotest-rpc-client host stat help
Usage: autotest-rpc-client host stat [options] <hosts>

Options:
  -h, --help           show this help message and exit
  -g, --debug          Print debugging information
  --kill-on-failure    Stop at the first failure
  --parse              Print the output using colon separated key=value
                      fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                      Specify the autotest server to talk to
  -M MACHINE_FLIST, --mlist=MACHINE_FLIST
```

To display host information:

```
# autotest-rpc-client host stat host0
-----
Host: host0
Platform: x386
Status: Repair Failed
Locked: False
Locked by: None
```

```
Locked time: None
Protection: Repair filesystem only
```

```
ACLs:
```

```
Id  Name
110 ac10
136 ac11
```

```
Labels:
```

```
Id  Name
392 standard_config
428 my_machines
```

Modifying Hosts Status

```
# autotest-rpc-client host mod help
Usage: autotest-rpc-client host mod [options] <hosts>

Options:
  -h, --help                show this help message and exit
  -g, --debug               Print debugging information
  --kill-on-failure         Stop at the first failure
  --parse                   Print the output using colon separated key=value
                           fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                           Specify the autotest server to talk to
  -M MACHINE_FLIST, --mlist=MACHINE_FLIST
                           File listing the machines
  -y, --ready               Mark this host ready
  -d, --dead                Mark this host dead
  -l, --lock                Lock hosts
  -u, --unlock              Unlock hosts
  -p PROTECTION, --protection=PROTECTION
                           Set the protection level on a host. Must be one of:
                           "Repair filesystem only", "No protection", or "Do not
                           repair"
```

You can change the various states of the machines:

```
# Lock all ho* hosts:
# autotest-rpc-client host mod -l ho*
Locked hosts:
    host0, host1

# Hosts have been repaired, put them back in the pool:
# autotest-rpc-client host mod --ready host0
Set status to Ready for host:
    host0
```

Job Management - autotest-rpc-client job

The following actions are used to manage jobs:

```
# autotest-rpc-client job help
usage: autotest-rpc-client job [create|list|stat|abort] [options] <job_ids>

options:
  -h, --help                show this help message and exit
  -g, --debug               Print debugging information
  --kill-on-failure         Stop at the first failure
  --parse                   Print the output using colon separated key=value
                           fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                           Specify the autotest server to talk to
```

Creating a Job

```
# autotest-rpc-client job create help
usage: autotest-rpc-client job create [options] job_name

options:
  -h, --help                show this help message and exit
  -g, --debug               Print debugging information
  --kill-on-failure         Stop at the first failure
  --parse                   Print the output using colon separated key=value
                           fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                           Specify the autotest server to talk to
  -p PRIORITY, --priority=PRIORITY
                           Job priority (low, medium, high, urgent),
                           default=medium
  -y, --synchronous        Make the job synchronous
  -c, --container           Run this client job in a container
  -f FILE, --control-file=FILE
                           use this control file
  -s, --server              This is server-side job
  -t TESTS, --tests=TESTS
                           Run a job with these tests
  -k KERNEL, --kernel=KERNEL
                           Install kernel from this URL before beginning job
  -m MACHINE, --machine=MACHINE
                           List of machines to run on (hostnames or n*label)
  -M MACHINE_FLIST, --mlist=MACHINE_FLIST
                           File listing machines to use
```

You can only create one job at a time. The job will be assigned the name `job_name` and will be run on the machine(s) specified using the `-m|--machine|-M|--mlist` options.

The machines can be specified using their hostnames or if you are just interested in a specific group of machines, you can use any arbitrary label you have defined, both platform and non-platform.

The syntax for those is: `n*label` to run on `n` machines of type `label` e.g., `2*Xeon,3*lab1,hostprovisioning`. You can omit `n` if `n` equals 1.

The options are:

- `-p|--priority` sets the job scheduling priority to Low, Medium (default), High or Urgent.

- `-s|--server` specifies if the job is a server job, or a client job (default). A server job must specify a control file using the `--control-file` option.
- `-y|--synchronous` specifies if the job is synchronous or asynchronous (default).
- `-k|--kernel=<file>` specifies the URL of a kernel to install before running the test(s).
- `-c|--container` runs the test(s) in a container. This is only valid for client-side jobs.

The tests can be specified in 2 mutually exclusive ways:

- `-f|--control-file=FILE` will run the job described in the control file FILE,
- `-t|--tests=a,b,c` will create a control file to run the tests a, b, and c.

One of these 2 options must be present.

The control file must be specified if your job is:

- synchronous, or
- a server-side job.

The `--control-file` option cannot be used with:

- the `--kernel` option.
- the `--container` option.

If you want to do any of those, code it in the control file itself.

You can find the list of existing tests using `autotest-rpc-client test list`.

```
# Create a job my_test using known tests on host0:
# autotest-rpc-client job create --test dbench,kernbench -m host0 my_test
Created job:
    my_test (id 6749)

# Create a server job using a custom control file on host0:
# cat ./control
job.run_test('sleeptest')

# autotest-rpc-client job create --server -f ./control -m host0 my_test_ctrl_file
Created job:
    my_test_ctrl_file (id 6751)

# Create a job on 2 Xeon machines, 3 Athlon and 1 x286:
# Find the platform labels:
# autotest-rpc-client label list -t
Name    Valid
Xeon    True
Athlon  True
x286    True

# autotest-rpc-client job create --test kernbench -m 2*Xeon,3*Athlon,*x286, test_on_
↪meta_hosts
Created job:
    test_on_meta_hosts (id 6761)
```

Listing Jobs

```
# autotest-rpc-client job list help
usage: autotest-rpc-client job list [options] <job_ids>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -a, --all             List jobs for all users.
  -r, --running        List only running jobs
  -u USER, --user=USER List jobs for given user
```

You can list all the jobs, or filter on specific users, IDs or job names. You can use the * wildcard for the job_name filter.

```
# List all my jobs
# autotest-rpc-client job list
Id   Owner  Name                Status Counts
3590 user0  Thourough test     Aborted:31, Completed:128, Failed:74
6626 user0  Job                 Completed:1
6634 user0  Job name with spaces Aborted:1
6749 user0  my_test             Queued:1
6751 user0  my_test_ctrl_file  Queued:1

# List all jobs starting with 'my'
# autotest-rpc-client job list my*
Id   Owner  Name                Status Counts
1646 user1  myjob               Completed:2
2702 user2  mytestburnin3      Aborted:1
6749 user0  my_test             Queued:1
6751 user0  my_test_ctrl_file  Queued:1
```

Getting Jobs Status

```
# autotest-rpc-client job stat help
usage: autotest-rpc-client job stat [options] <job_ids>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -f, --control-file    Display the control file
```

At least one job ID or name must be specified. The * wildcard can be used for the job name but **not** for the job ID.

```
# Get status of the previously queued jobs. Note the hostname in this output:
# autotest-rpc-client job stat my_test\*
Id   Name                Priority  Status Counts  Host Status
6749 my_test              Medium   Queued:1      Queued:host0
6751 my_test_ctrl_file    Medium   Queued:1      Queued:host0

# The stats on a meta host job will show the hostname once the scheduler mapped the
↳platform label to available hosts:

# autotest-rpc-client job stat 6761
Id   Name                Priority  Status Counts  Host Status
6761 test_on_meta_hosts    Medium   Queued:4, Running:1  Running:host42
```

Aborting Jobs

```
# autotest-rpc-client job abort help
usage: autotest-rpc-client job abort [options] <job_ids>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
```

You must specify at least one job ID. You cannot use the job name.

```
# autotest-rpc-client job abort 6749,6751 6761
Aborted jobs:
    6749, 6751, 6761
```

Label Management - autotest-rpc-client label

The following actions are available to manage the labels:

```
# autotest-rpc-client label help
usage: autotest-rpc-client label [create|delete|list|add|remove] [options] <labels>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -B LABEL_FLIST, --blist=LABEL_FLIST
                       File listing the labels
```

Creating a label

```
# autotest-rpc-client label create help
usage: autotest-rpc-client label create [options] <labels>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -B LABEL_FLIST, --blist=LABEL_FLIST
                       File listing the labels
  -t, --platform       To create this label as a platform
```

You can create multiple labels at a time. They can be specified on the command line or in a file, using the `-B|--blist` option.

```
# autotest-rpc-client label create my_label
Created label:
    my_label
# autotest-rpc-client label create label0 label1
Created label:
    label0, label1
```

Deleting a label

```
# autotest-rpc-client label delete help
usage: autotest-rpc-client label delete [options] <labels>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -B LABEL_FLIST, --blist=LABEL_FLIST
                       File listing the labels
```

You can delete multiple labels at a time. They can be specified on the command line or in a file, using the `-b|--blist` option.

```
# autotest-rpc-client label delete label0,label1
Deleted labels:
    label0, label1
```

Listing labels

```
# autotest-rpc-client label list help
usage: autotest-rpc-client label list [options] <labels>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -B LABEL_FLIST, --blist=LABEL_FLIST
                       File listing the labels
  -t, --platform-only  Display only platform labels
  -d, --valid-only     Display only valid labels
  -a, --all             Display both normal & platform labels
  -m MACHINE, --machine=MACHINE
                       List LABELs of MACHINE
```

You can list all the labels, or filter on specific labels or machines (exclusively).

```
# Show all labels
# autotest-rpc-client label list
Name    Valid
label0  True
label1  True

# Display labels that host host0 is tagged with
# autotest-rpc-client label list label0 -m host0
Name    Valid
label0  True
```

Adding Hosts to a Label

```
# autotest-rpc-client label add help
usage: autotest-rpc-client label add [options] <labels>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -B LABEL_FLIST, --blist=LABEL_FLIST
                       File listing the labels
  -m MACHINE, --machine=MACHINE
                       Add MACHINE(s) to the LABEL
  -M MACHINE_FLIST, --mlist=MACHINE_FLIST
                       File containing machines to add to the LABEL
```

You must specify at least one label and one machine.

```
# Add hosts host0 and host1 to 'my_label'
# autotest-rpc-client label add my_label -m host0,host1
Added to label my_label hosts:
    host0, host1
```

Removing Hosts from a Label

```
# autotest-rpc-client label remove help
usage: autotest-rpc-client label remove [options] <labels>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -B LABEL_FLIST, --blist=LABEL_FLIST
                       File listing the labels
  -m MACHINE, --machine=MACHINE
                       Remove MACHINE(s) from the LABEL
  -M MACHINE_FLIST, --mlist=MACHINE_FLIST
                       File containing machines to remove from the LABEL
```

The options are the same than for adding hosts. You must specify at least one label and one machine.

```
# cat my_machines
host0
host1,host2
# autotest-rpc-client label rm my_label --mlist my_machines
Removed from label my_label hosts:
    host0, host1, host2

# Completely delete the LABEL.
# autotest-rpc-client label delete my_label
Deleted label:
    my_label
```

Possible errors and troubleshooting

```
Duplicate label: {{{# autotest-rpc-client label create my\_label Operation add\_label
failed:

ValidationError?: {'name': 'This value must be unique (my\_label)'}
}}}
```

Adding an unknown host:

```
# autotest-rpc-client label add my_label -m host20,host21
Operation label_add_hosts failed:
  DoesNotExist: Host matching query does not exist. (my_label (host20,host21))}}
```

Test Management - autotest-rpc-client test

The following actions are available to manage the tests:

```
# autotest-rpc-client test help
usage: autotest-rpc-client test list [options] [tests]

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -T TEST_FLIST, --tlist=TEST_FLIST
                       File listing the tests
```

Listing Tests

```
# autotest-rpc-client test list help
usage: autotest-rpc-client test list [options] [tests]

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse               Print the output using colon separated key=value
                       fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                       Specify the autotest server to talk to
  -T TEST_FLIST, --tlist=TEST_FLIST
                       File listing the tests
  -d, --description    Display the test descriptions
```

You can list all the tests, or specify a few you'd like information on.

```
# autotest-rpc-client test list
Name      Test Type  Test Class
sleeptest Client     Canned Test Sets
dbench    Client     Canned Test Sets
Kernbench Client     Canned Test Sets

# Specifying some test names, with descriptions:
# autotest-rpc-client test list Kernbench,dbench -d
Name      Test Type  Test Class      Description
Kernbench Client     Canned Test Sets  unknown
dbench    Client     Canned Test Sets  dbench is one of our standard kernel stress_
↳ tests. It produces filesystem
```

load like netbench originally did, but involves no network system calls. Its results include throughput rates, which can be used **for** performance analysis.

More information on dbench can be found here:
<http://samba.org/ftp/tridge/dbench/README>

User Management - autotest-rpc-client user

The following actions are available to manage users:

```
# autotest-rpc-client user help
usage: autotest-rpc-client user list [options] <users>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse              Print the output using colon separated key=value
                      fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                      Specify the autotest server to talk to
  -u USER_FLIST, --ulist=USER_FLIST
                      File listing the users
```

Listing users

```
# autotest-rpc-client user list help
usage: autotest-rpc-client user list [options] <users>

options:
  -h, --help            show this help message and exit
  -g, --debug           Print debugging information
  --kill-on-failure     Stop at the first failure
  --parse              Print the output using colon separated key=value
                      fields
  -v, --verbose
  -w WEB_SERVER, --web=WEB_SERVER
                      Specify the autotest server to talk to
  -u USER_FLIST, --ulist=USER_FLIST
                      File listing the users
  -a ACL, --acl=ACL     Only list users within this ACL
  -l ACCESS_LEVEL, --access_level=ACCESS_LEVEL
                      Only list users at this access level
```

You can list all the users or filter on specific users, ACLs or access levels. You can use wildcards for those options. The verbose option displays the access level.

```
# Show all users
# autotest-rpc-client user list
Login
user0
user1
```

```

me_too
you_as_well

# Show all users starting with u
# autotest-rpc-client user list u\* -v
Id  Login          Access Level
3   user0          0
7   user1          1

# Show all users starting with u and access level 0.
# autotest-rpc-client user list u\* -v -l 0
Id  Login          Access Level
3   user0          0

# Show all users belonging to the ACL acl0
# autotest-rpc-client user list -a acl0
Login
user1
metoo

```

Frontend Database (autotest_web)

The AFE frontend and the scheduler both work from the “autotest_web” database.

- **Test:** a test than can be run as part of a job. Each row corresponds to a control file, most often found at (clientserver)/tests/<test name>/control, but not always.
- **User:** a user of the system.
- **Host:** a machine on which tests can be run.
- **AcIGroup:** access control groups. Each group is in a many-to-many relationship with users and hosts and gives users in that group permission to run jobs on hosts in the same group.
- **Label:** a label describing a type of host, such as “intel” or “regression_testing_machines”. These help users schedule jobs on particular groups of machines.
- **Job:** a logical job consists of a set of hosts and a control file to run on those hosts. It can be tracked throughout the system by its ID. A row in this table contains the control file for the job and information about how it should be run.
- **HostQueueEntry:** this table provides a many-to-many relationship between jobs and hosts. It is used to keep track of the hosts on which a job is scheduled to run, and by the scheduler to keep track of the progress of those runs. It can also represent a “metahost” for a job, which indicates that a job is scheduled to run on any machine from a particular label.
- **IneligibleHostQueue:** this table also provides a many-to-many relationship between jobs and hosts. It is used to indicate which hosts a job has already been scheduled against and is used by the scheduler in assigning metahosts.

Understanding the TKO Results Database

This page will (hopefully) help you understand how results are structured in the Autotest results database, and how you can best structure results for your test.

Structure of test results

The core results entity produced when you run a tests is a **Test Result**. (The DB model name is simply “Test”, but “Test Result” is more clear, so I’m going to use that term here.) Each Test Result has a number of fields, most importantly the name of that test that ran and the status of the test outcome. Test Results also include timestamps and links to a few related objects, including the kernel and machine on which the test ran, and the job that ran the test. Each of these objects includes other fields - see *TKO database* for the full list.

Each Test Result can also have any number of **Test Attributes**, each of which is a key-value pair of strings. Note that some Test Attributes are included with each test automatically, including information on test parameters and machine sysinfo.

Furthermore, each Test Result can have any number of **Iterations**, indexed from zero. These are primarily for use by performance tests.

- Each Iteration can have any number of **Iteration Attributes**, each of which is a key-value pair of strings.
- Each Iteration can also have any number of **Iteration Results**, each of which is a key-value pair with floating-point values (and string keys, as usual). This is the only way to record numerical data for a test. It is used for all performance tests.

Note that, despite the names, both of these kinds of iteration keyvals are intended to describe results-oriented information. The only difference is that one holds string-valued results while the other holds numerical results. Neither type of iteration keyval is intended to hold information about how the test ran (such as test parameters). By design, all iterations within a test should run the exact same way. The only intended purpose of iterations is to gather more samples for statistical purposes. If you want to run a test multiple times varying parameters, you should create multiple Test Results (see below).

To summarize:

- Job
 - Test Results
 - * Test Attributes (string key -> string value)
 - * Iterations (indexed from 0)
 - Iteration Attributes (string key -> string value)
 - Iteration Results (string key -> float value)

How are test results created?

Each call to `job.run_test()` implicitly creates one Test Result. The status of the Test Result is determined by what, if any, exception was raised (and escaped) during test execution. Any calls to record keyvals within the test will be associated with the Test Result for that call to `run_test()`.

If you want to create many Test Result objects, you must have code to call `job.run_test()` many times. This code must reside in the control file, or in a library called by the control file, but not within the test class itself (since everything in the test class executes within a call to `run_test()`).

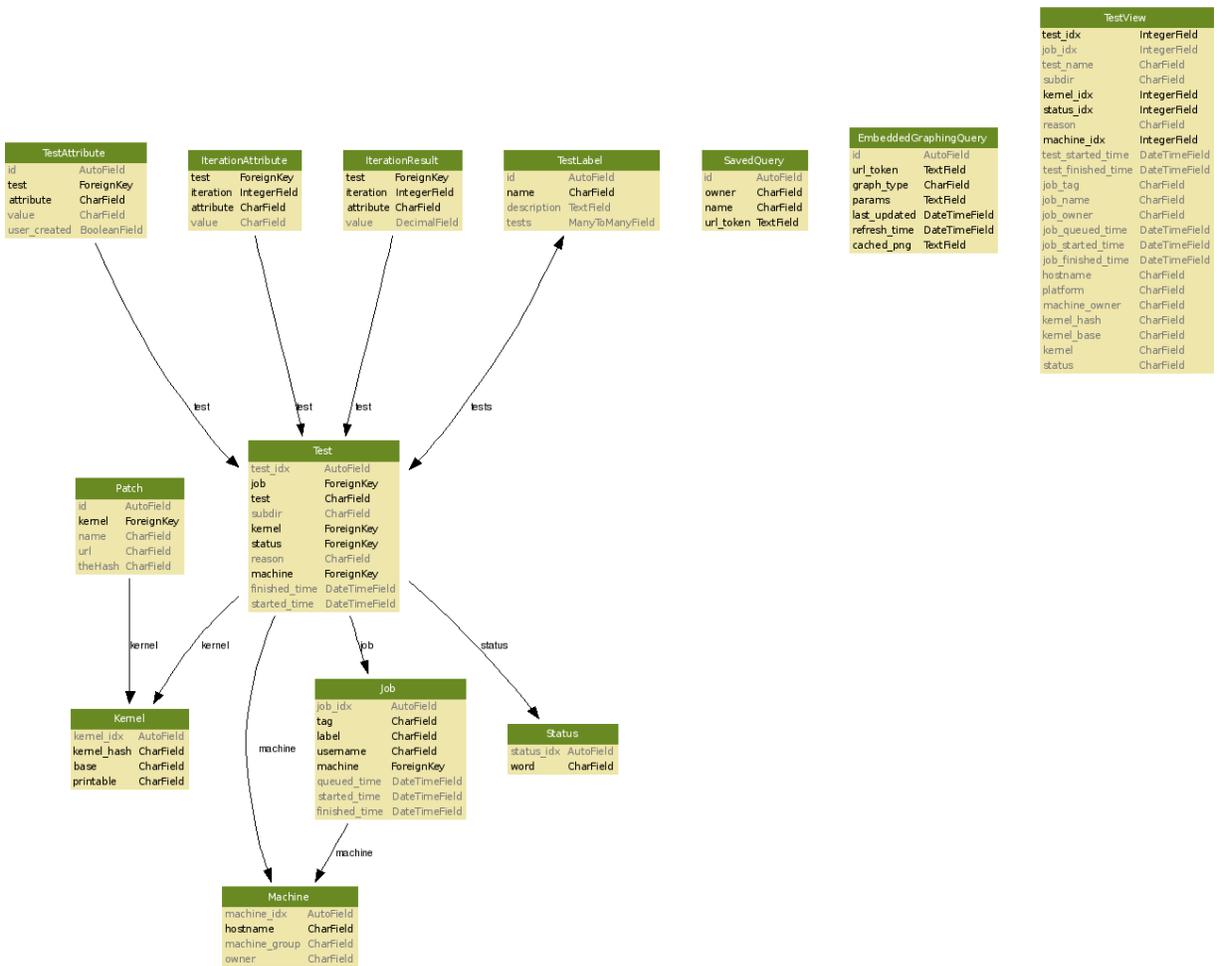
A new issue arises when running the same test multiple times within a job. This will generate many Test Results with the same test name, but there must be a unique identifier for each Test Result (other than the database ID). This brings another Test Result field into play – `subdir`, the subdirectory containing the result files for that Test Result. `subdir` is normally equal to the test name, but this field must be unique among all Test Results for a job. When running a test multiple times, unique `subdir`’s are usually achieved by passing a unique `tag` with each call to `job.run_test()` for a particular test. The `subdir` then becomes `$test_name.$tag`.

Further reading

- AutotestApi explains how each of these keyvals can be recorded by test code using Test APIs.
- *TkoDatabase* illustrates the database schema. Note that it does not map directly onto these concepts. In particular, there's no table for iterations themselves, only the iteration keyvals. The existence of iterations themselves is implicit.
- *Keyval <../local/Keyval>* explains the placement and format of keyval files within the results directories. These are written by Autoserv and read by the Parser to fill in the database.

TKO results database

The TKO results database holds results of test runs. The parser puts data into it and the TKO web interface allows users to view data from it.



- The `tests` table is the core of the DB and contains a row for each test run.
- A job in the `jobs` table corresponds to a single execution instance of autoserv. Each job can have many tests.
- The `test_attributes`, `iteration_attributes`, and `iteration_result` hold keyval information about tests.

- The `status` table is simply an enumeration of tests status values, i.e. completed, failed, aborted, etc.
- The `kernels` and `patches` tables hold kernel information for kernels against which tests are run.
- The `machines` tables holds information on machines on which tests are run.

MySQL replication

Introduction

If you're a *heavy* user of Autotest and its reporting/graphing functionality its possible that you've experienced slow downs that database slave(s) could mitigate. There are lots of guides on the internet for doing MySQL replication. This presents just one possible way to set it up.

Notes on replication:

- Only read-only operations can go through the slave. At the moment, only the new TKO interface supports splitting read-only and read-write traffic up between servers.
- MySQL replicates by replaying SQL statements. This means that it is possible to construct SQL statements that will execute non-deterministically on replicas. None of the commands Autotest runs *should* have this problem, but you need to know it's possible. This also means that you might want to verify the consistency of the slave database once in a while.
- MySQL replication happens in one thread. In highly parallelizable, write heavy workloads, the slave will probably fall behind. In practice this is pretty much never an issue.
- ...there's lots of other caveats. If you're still reading, you might want to check out <http://oreilly.com/catalog/9780596101718/>

Preparing the Master

First of all, you're going to need to set up the binary log. All queries which might affect the database (i.e. not SELECTs) will be written to this log. Replication threads will then read the file and send updates to the database slaves. Because it's in a file, this also means that if a slave goes off line for a while (under the limit we'll set in a moment), it can easily re-sync later.

Open the `/etc/mysql/my.cnf` file with root permissions (so probably with `sudo`).

Uncomment out (or add) the following lines in the `[mysqld]` section of the file.

```
server-id          = SOMETHING_UNIQUE
log_bin           = /var/log/mysql/mysql-bin.log
expire_logs_days  = 10
max_binlog_size   = 100M
```

The `server-id` needs to be an *unique* 32 bit int but otherwise doesn't matter. The `log_bin` says to use binary logging and specifies the prefix used for log files. The log files are rotated when they become `max_binlog_size` and are kept for `expire_logs_days` days.

Restart the mysql server and log into the prompt with the `mysql` command. Now create a user for replication:

```
GRANT REPLICATION SLAVE ON *.* TO 'slave_user'@'%' IDENTIFIED BY 'some_password';
FLUSH PRIVILEGES;
```

Creating a Snapshot

MySQL has a built in command to sync a slave to a master without any existing data, but this isn't useful in a production environment because it locks all the tables on the master for an extended period of time. The following is a good compromise of downtime (it'll lock things for a couple minutes) and ease of use. If you can't have any down time, consult other resources and good luck. :-)

The following command will dump all databases to a file called /tmp/backup.sql. It uses extended inserts which cuts down on the file size, but makes the file (a bit) less portable. The `--master-data` tells it to write what the current bin-log location is to the beginning of the file and causes the database to be read-only locked during the duration.

```
mysqldump -uroot -p --all-databases --master-data --extended-insert > /tmp/backup.sql
```

Setting up the Slave

On the database slave, simply copy over the SQL dump you created in the last step and (assuming the dump is in /tmp/backup.sql):

```
mysql -uroot -p < /tmp/backup.sql
```

Now edit your `/etc/mysql/my.cnf`. Add the following lines under the `[mysqld]` section:

```
server-id          = SOMETHING_UNIQUE
log_bin           = /var/log/mysql/mysql-bin.log
expire_logs_days  = 10
max_binlog_size   = 100M
read_only         = 1
```

The `read_only` parameter makes it so that only DB slave processes and those with SUPER access can modify the database. The `log_bin` turns on the binary logging so that other servers can be chained off of this replica.

If you're using a debian based distro, you'll need to copy over the login data from the `/etc/mysql/debian.cnf` of the master to the slave.

Stop and start mysql.

```
sudo /etc/init.d/mysql stop
sudo /etc/init.d/mysql start
```

Out of the SQL dump we loaded earlier, get the master position via

```
grep 'CHANGE MASTER' /tmp/backup.sql | head -n1
```

Open up a mysql root prompt and run the following command (modified for your local setup). After that, start the slave thread and show the current status.

```
CHANGE MASTER TO MASTER_HOST='some.host.com', MASTER_USER='slave_user', MASTER_
↪PASSWORD='some_password', MASTER_LOG_FILE='from the output above', MASTER_LOG_
↪POS=ditto;
START SLAVE;
SHOW SLAVE STATUS\G;
```

On your database master, you can run `SHOW MASTER STATUS;` and verify that the slave is up to date (or is currently catching up).

RPC Server

The Autotest RPC Server, also known as the frontend, is a Django based application that provides:

- The Database Objects (defined by Django `Models`)
- A remoting interface using the JSON-RPC protocol
- The `Administration Web Interface` that Django gives us for free

We'll start by taking a look at the Database the Models and the database structure that they generate.

Models

The Database Models play a major role in the RPC server. The most important things they do:

- Define and create the database structure on the Autotest Relational Database
- Provide a object like uniform API for the Database entries

Note: For historical reasons, the RPC server is composed of two different applications, AFE and TKO. Because of that, the models are also defined in two different modules.

These may soon be united into a single application, specially their model definition. For now, keep in mind that the model you are looking for may be in one of two different places.

Model Logic

Autotest extends the base Django Database models with some custom logic.

`ModelWithInvalid`

AFE Models

AFE stands for Autotest Front End. It's an application that provides access to the core of Autotest definitions, such as Hosts, Tests, Jobs, etc.

For the classes that inherit from `django.db.models.Model` some of the attributes documented here are instances from one of the many `django.db.models.fields` classes and will be mapped into a field on the relational database.

`AtomicGroup`

`Job`

`Label`

`Drone`

`DroneSet`

User

Host

HostAttribute

Test

TestParameter

Profiler

AclGroup

Kernel

ParameterizedJob

ParameterizedJobProfiler

ParameterizedJobProfilerParameter

ParameterizedJobParameter

Job

AFE Exceptions

Besides persistence, Models also provide some logic. And as such, some custom error conditions exist.

TKO Models

TKO is the autotest application dedicated to storing and querying test results.

Machine

Kernel

Patch

Status

Job

JobKeyval

Test

RPC Interface

Functions exposed over the RPC interface.

Note: For historical reasons, the RPC server is composed of two different applications, AFE and TKO.

AFE RPC Interface

Custom RPC Scripts

This is a brief outline of how to use the TKO RPC interface to write custom results analysis scripts in Python. Using the AFE RPC interface is very similar.

Basically:

- make your script any place in the client with a common.py
- to import the rpc stuff you need do:

```
import common # pylint: disable=W0611
from autotest_lib.cli import rpc
```

- to create the object you need for making the rpc calls use “comm = rpc.tko_comm()”; you can pass in a host name if you want to point at something other than what’s in the global_config.ini file in your client.
- you can get the test detail with code like:

```
test_views = comm.run("get_detailed_test_views", ...filters go here...)
```

The filters are basically django filters. I won’t go into much detail here, the obvious ones you’d want to use are:

- job_tag__startswith - set it to something like “1234-” to get data on job 1234
- hostname - if you want data for a specific hostname, set this
- test_name - if you want data for a specific test name, set this

So you could do something like:

```
test_views = comm.run("get_detailed_test_views", job_tag__startswith=
↪ "1234-", hostname="myhost")
```

The test_views returned by that call is a list of dictionaries, one dictionary for each test returned by the call. The main keys you’re concerned with will be “attributes” and “iterations”.

attributes is a dictionary of all the test level keyvals - you can see stuff like “sysinfo-uname” here.

iterations is a list of dictionaries, one for each iteration. Each dictionary has two entries; an “attr” one, which is a dictionary of all the key{attr}=value attributes in the test, and a “perf” one, which is a dictionary of all the key{perf}=value attributes.

And...that’s basically how you access all that info. You make that call and get a big list of dictionaries. Oh, and avoid calling it without filters; trying to pull down data for every single test can be a bad idea (depending on the size of your database).

Policy for changing the frontend(AFE) and TKO RPC protocols

Try to make any RPC protocol change so that it's backwards compatible. If there are good reasons not to make it backwards compatible then the following procedure has to be followed:

- initial code changes have to be backwards compatible (so we end up supporting both old and the new RPC API); existent RPC users in the autotest code base have be already changed to use the new API
- to give enough time for external RPC users, an announcement about this RPC change should go on the public mailing list
- after at least a month since the RPC API change announcement the support for the old RPC API can be removed from the code

Web Frontend HOWTO

The Autotest web frontend can be used for

- browsing existing jobs
- viewing job details and getting to job results and log files
- submitting new jobs
- tracking hosts' statuses
- managing (browsing, creating, modifying, and deleting) hosts, labels, profilers, and ACL groups

When you first bring up the frontend, you'll see something like this:

Autotest Frontend - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8000/afe/server/client/autotest.AfeClient/AfeCler

Autotest Frontend | Admin | Results (Old TKO) | Documentation | Feeds: Completed, Failed

Job List View Job Create Job Host List View Host Refresh

Queued Jobs Running Jobs Finished Jobs All Jobs

View jobs for user: alincoln

Select: all, visible, none Actions

Select	ID	Owner	Name	Priority	Client/Server	Created	Status
<input type="checkbox"/>	5	alincoln	another server job	Medium	Server	2008-01-10 17:17	3 Queued
<input type="checkbox"/>	4	alincoln	very important job	Urgent	Client	2008-01-10 17:17	7 Queued
<input type="checkbox"/>	3	alincoln	server job	Low	Server	2008-01-10 17:16	3 Queued
<input type="checkbox"/>	2	alincoln	my job	Medium	Client	2008-01-10 17:15	5 Queued
<input type="checkbox"/>	1	alincoln	run some tests	Medium	Client	2008-01-10 15:51	2 Failed 2 Completed 1 Aborted

Done

Job List

The interface initially shows the Job List tab, which allows you to browse existing jobs. The four links at the top filter jobs by status - you can view only queued, running, or finished (which includes completed and aborted) jobs, or view them all (the default). You can also filter by job owner and job name. The initial view shows all jobs owned by you. Most recently submitted jobs are displayed first.

The Hosts column shows how many hosts in each job are currently in each state (see JobAndHostStatuses). You can use the Refresh button at the top to refresh the list (it won't refresh itself). Clicking on a job in the list brings up the View Job tab for the selected job.

You can select multiple jobs with the checkboxes on the left, or using the links at the top of the table. You can then using the "Actions" menu to operate on many jobs at once. Currently, this only allows you to abort jobs.

View Job

The View Job tab shows details about a single job along with results and a link to log files.

The box at the top allows you to manually fetch a job by ID. The page displays basic info about the job, an "Abort Job" button if the job has not completed, and a "Clone Job" button to create a new job modeled after the current job. Clone job will present three options:

- Reuse any similar hosts - if the original job use "run on any" hosts, the new job will do the same, so that it could get assigned different hosts.
- Reuse same specific hosts - the exact same set of hosts will be used, even if the original job specific "run on any" hosts.
- Use failed and aborted hosts - uses the hosts have have been aborted, or have failed the job in some way

Below this, the full contents of the job's control file are displayed, follow by job results. This consists of an embedded TKO spreadsheet for the job with three links above:

- open in new window - to open the old TKO interface for the job.
- new results interface - to open the new TKO interface for the job.
- raw results logs - to bring up a listing of the job results directory. This is often useful for debugging when things go wrong.

Finally, the table at the bottom shows all hosts on which the job was scheduled and the current status of the job on each host (see JobAndHostStatuses). Links are provided to jump directly to the status log and debug logs for each host. In addition, you can select individual hosts and abort them with the Actions menu. You can clone the job on the selected hosts from the Actions menu as well. Selecting no hosts and choosing "Clone job on selected hosts" will clone the job without adding any hosts.

Create Job

This tab allows you to create and submit a new job.

Create job parameters

- **Job name** can be any string.
- **Priority** affects how your job will be placed in the queue; higher priority jobs with preempt lower priority ones that have not yet started when the jobs are scheduled on the same machine.

Autotest Frontend - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8000/afe/server/client/autotest.AfeClient/AfeCler

Google

Frontend | [Admin](#) | [Results \(Old TKO\)](#) | [Documentation](#) | Feeds: [Completed](#), [Failed](#)

Job List **View Job** Create Job Host List View Host Refresh

Fetch job by ID: Go

Job: run some tests (1-alincoln)

Clone job

Label: run some tests
Owner: alincoln
Priority: Medium
Created: 2008-01-10 15:51:32
Timeout: 72 hours
Email List:
Status: 2 Failed, 2 Completed, 1 Aborted
Client control file (asynchronous):

```
def step_init():
    job.next_step([step_test])
    testkernel = job.kernel('2.6.18-rc3-mm1')

    testkernel.install()
    testkernel.boot()

def step_test():
    job.run_test('aiostress')

    job.run_test('bonnie')

def client():
    job.run_test('netperf2', '127.0.0.1', '127.0.0.1', 'client', tag='client')

def server():
    job.run_test('netperf2', '127.0.0.1', '127.0.0.1', 'server', tag='server')

job.parallel([server], [client])
```

Full results ([open in new window](#)) ([new results interface](#)) ([raw results logs](#))

Done

Autotest Frontend - Mozilla Firefox
_ □ ×

File Edit View History Bookmarks Tools Help
⚙

← → ↻ 🏠
🔍 Google

Job List View Job **Create Job** Host List View Host Refresh

Job name:

Priority:

Kernel:

Timeout (hours):

Email List:

Skip verify:

Tests: Test type:

Test	
<input type="checkbox"/> aiostress	
<input type="checkbox"/> bonnie	
<input type="checkbox"/> fsx	
<input type="checkbox"/> kernbench	
<input type="checkbox"/> memtest86	
<input type="checkbox"/> netperf2	
<input type="checkbox"/> sleeptest	

bonnie (Kernel / Functional)

Written by: Martin Bligh <mbligh@google.com>

Type: Client Asynchronous

Time: SHORT (less than 15 minutes)

Bonnie is a benchmark which measures the performance of Unix file system operations. Bonnie is concerned with identifying bottlenecks; the name is a tribute to Bonnie Raitt, who knows how to use one.

For more info, see <http://www.textuality.com/bonnie/>

This benchmark configuration run generates sustained write traffic of 35-50MB/s of .1MB writes to just one disk. It appears to have a sequential and a random workload. It gives profile measurements for: throughput, %CPU rand seeks per second. Not sure if the the CPU numbers are trustworthy.

Profilers:

[View control file](#)

One-time host(s):

Run on any

Number:

Available hosts: *(Click on a host to add it)*

Hostname:

Platform:

Label:

Status:

Locked:

ACL accessible only ACL accessible only

<< < 1-7 of 7 Next Last
First Previous > >>

Hostname	Platform	Other labels	Status	Locked
berlin	x86_64		Ready	No
fortwayne	i386		Ready	No
london	i386		Ready	No
odessa	ppc64		Ready	No
sanfrancisco	x86_64		Ready	No
washington	x86_64		Ready	No
waterloo	i386		Ready	No

Selected hosts: *(Click on a host to remove it)*

<< < 0-0 of 0 Next Last
First Previous > >>

Hostname	Platform	Other labels	Status	Locked
<<	<	0-0 of 0	Next	Last
First	Previous		>	>>

1.4. Frontend
101

- The **kernel** field allows you to specify a kernel to install on the test machine before testing; leaving this field blank will leave out the kernel install step. You can specify a URL pointing to a kernel source tarball or a .rpm or .deb package. Site-specific extensions are also possible.
- **Timeout** specifies the hours after job creation until the scheduler will automatically abort the job if it hasn't yet completed.
- **Max runtime** specifies the hours after the job starts running (Autoserv is executed) until the scheduler will automatically abort the job if it hasn't yet completed.
- **Email List** can contain a comma- or space-separated list of email addresses which will be notified upon job completion.
- If **Skip verify** is checked, hosts won't be verified before the job is run. This is useful for machine reinstalls among other things.
- **Reboot before** determines whether hosts will be rebooted before the job runs. *If dirty* means the host will be rebooted if it hasn't been rebooted since being added, being locked, or having the last job run.
- **Reboot after** determines whether hosts will be rebooted after the job runs. *If all tests passed* means the host won't be rebooted if any test within the job failed.
- If **Include failed repair results** is checked, when a machine fails repair, "repair" and "verify" test entries will show up in TKO for that machine, along with a SERVER_JOB entry. If unchecked, nothing at all will show up in TKO for the failed machine.
- The **Tests** section contains a table allowing you to select a set of client- or server-side tests to run. You can click on any test to view its description. Your test selections, along with the kernel field, are used to build the job's control file.
- **Profilers** shows available profilers than can be enabled for your job.
- Clicking **View control file** will display a box that shows the control file being constructed from your choices. You may edit the control file by hand by clicking **Edit control file**. This will make the control file field editable, but disables the kernel input and all test selector. If you want to go back and change your selections in these inputs, you'll need to revert your kernel changes. When editing a control file, you have two additional options. You shouldn't have to edit these unless you know what you're doing.
 - **Client or Server** - whether the control file should run on the client-side or the server-side.
 - **Synchronous** - if checked, the job will wait for all machines to be ready and then run all machines in a single autoserv instance. This is usually only necessary for multi-machine tests.
- The **Available hosts** and **Selected hosts** tables allow you to select hosts on which to run the job. Individual hosts can be selected and deselected by clicking on them. The filters at the top of the Available hosts table can be used to narrow your selection, just like in the Hosts tab. "Select visible" adds all hosts currently visible in the Available hosts table. "Select all" adds all hosts currently matching the filters.
 - The **Run on any** box allows you to request that the job be run on any machines from a given platform or label. The machines will be automatically selected from the set of available machines when the job is run.
 - The **One-time host(s)** box allows you to enter a hostname (or space-separated list of hostnames) that will be added to the database just for the job, without leaving the machine available for other jobs.
- Finally, the **Submit Job** button will attempt to submit your job, and any errors will show up in red.

Host List

This tab allows you to browse all hosts in the system.

The table can be searched and filtered using the boxes at the top. Clicking on a host brings you to the "View Host" tab for that host.

Autotest Frontend - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8000/afe/server/client/autotest.AfeClient/AfeClient

Autotest Frontend | [Admin](#) | [Results \(Old TKO\)](#) | [Documentation](#) | Feeds: [Completed](#), [Failed](#)

Job List View Job Create Job **Host List** View Host Refresh

Hostname

Platform

Label

Status

Locked

ACL accessible only ACL accessible only

<< First < Previous 1-7 of 7 Next > Last >>

Hostname	Platform	Other labels	Status	Locked
berlin	x86_64		Ready	No
fortwayne	i386		Ready	No
london	i386		Ready	No
odessa	ppc64		Ready	No
sanfrancisco	x86_64		Ready	No
washington	x86_64		Ready	No
waterloo	i386		Ready	No

<< First < Previous 1-7 of 7 Next > Last >>

Done

Additionally, you can force hosts to go into Verify by selecting them and choosing “Reverify hosts” from the Actions menu.

View Host

Autotest Frontend - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8000/afe/server/client/autotest.AfeClient/AfeClient

Autotest Frontend | Admin | Results (Old TKO) | Documentation | Feeds: Completed, Failed

Job List View Job Create Job Host List View Host Refresh

Fetch host: Go

Host odessa

Platform: ppc64
 Other labels:
 Status: Ready
 Locked: No

[View Verify/Repair Logs](#)

Jobs for this host:

<< First < Previous 1-3 of 3 Next > Last >>

Job ID	Job Owner	Job Name	Status
4	alincoln	very important job	Queued
3	alincoln	server job	Queued
1	alincoln	run some tests	Aborted

<< First < Previous 1-3 of 3 Next > Last >>

Done

This tab shows detailed information for a particular host including a list of all jobs queued, running and previously run on that host. It additionally provides a link to the scheduler’s verify/repair logs for the host.

User preferences

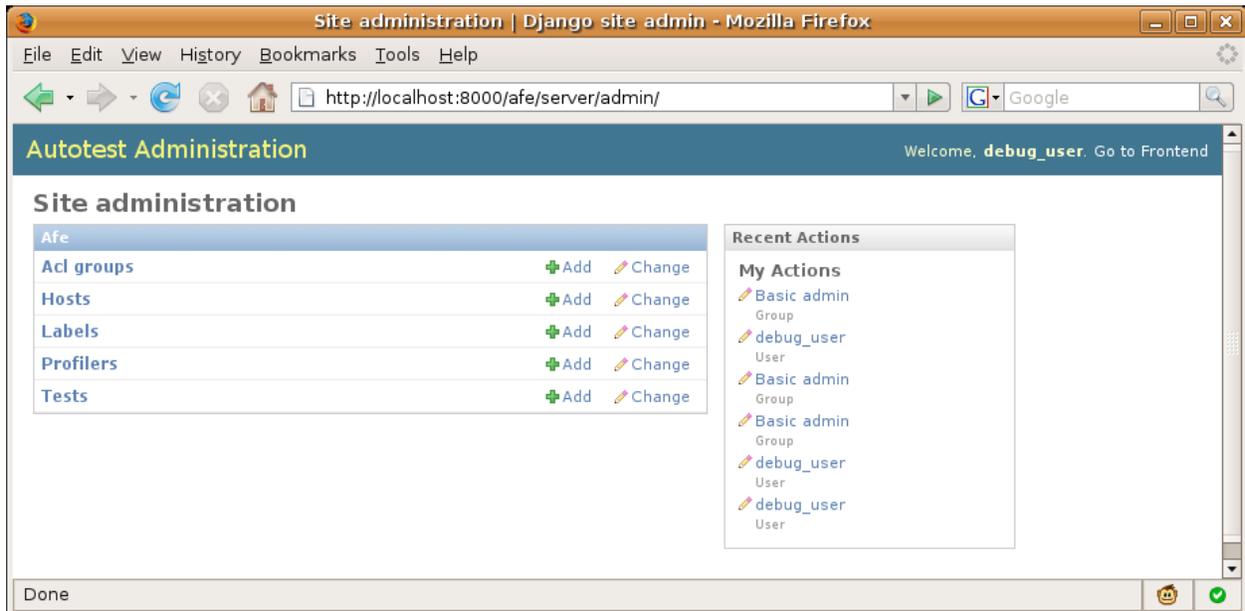
The user preferences tab allows you to set defaults for creating jobs. See [WebFrontendHowTo#Createjobparameters](#).

- **Reboot before** and **Reboot after** control default values for the corresponding options on the Create Job page.
- **Show experimental tests** will make the Create Job page show tests that are marked as “experimental” in the control file.

Admin interface

Clicking the “Admin” link in the upper right corner takes you to the admin interface for managing hosts, labels, profilers and ACL groups. Tests may be managed through the admin interface as well, but the preferred server setup is

to use `utils/test_importer.py` to automatically populate the DB with information from the test control files themselves (see `ControlRequirements` and `utils/test_importer.py --help`).



This is the built-in Django admin system. Here you can browse, create, modify, and delete objects. The link in the upper right corner takes you back to the frontend. The different objects types appear on the Admin index page. Clicking on any object type takes you to a list of that object type.

The list can be sorted, searched, and filtered. The link at the top right allows you to create a new object, and clicking on any object takes you to the edit page for that object.

From this page you can fill in the information in the fields and click “Save” at the lower right corner to add or edit the object. You can also delete the object using the link at the lower left corner.

For help on the meanings of different fields, see the database documentation.

Web Frontend Roadmap

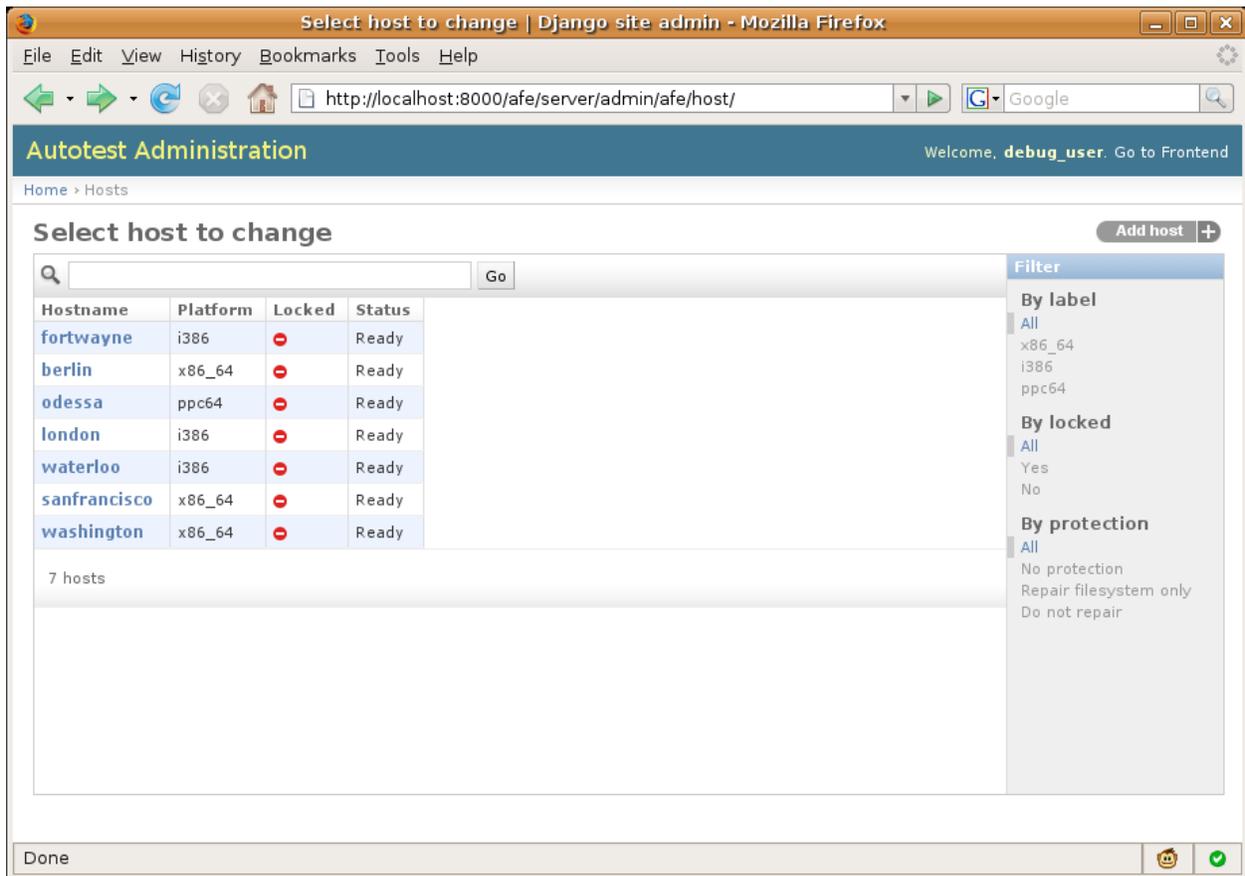
There are currently two completely separate projects with Autotest that might be called web frontends:

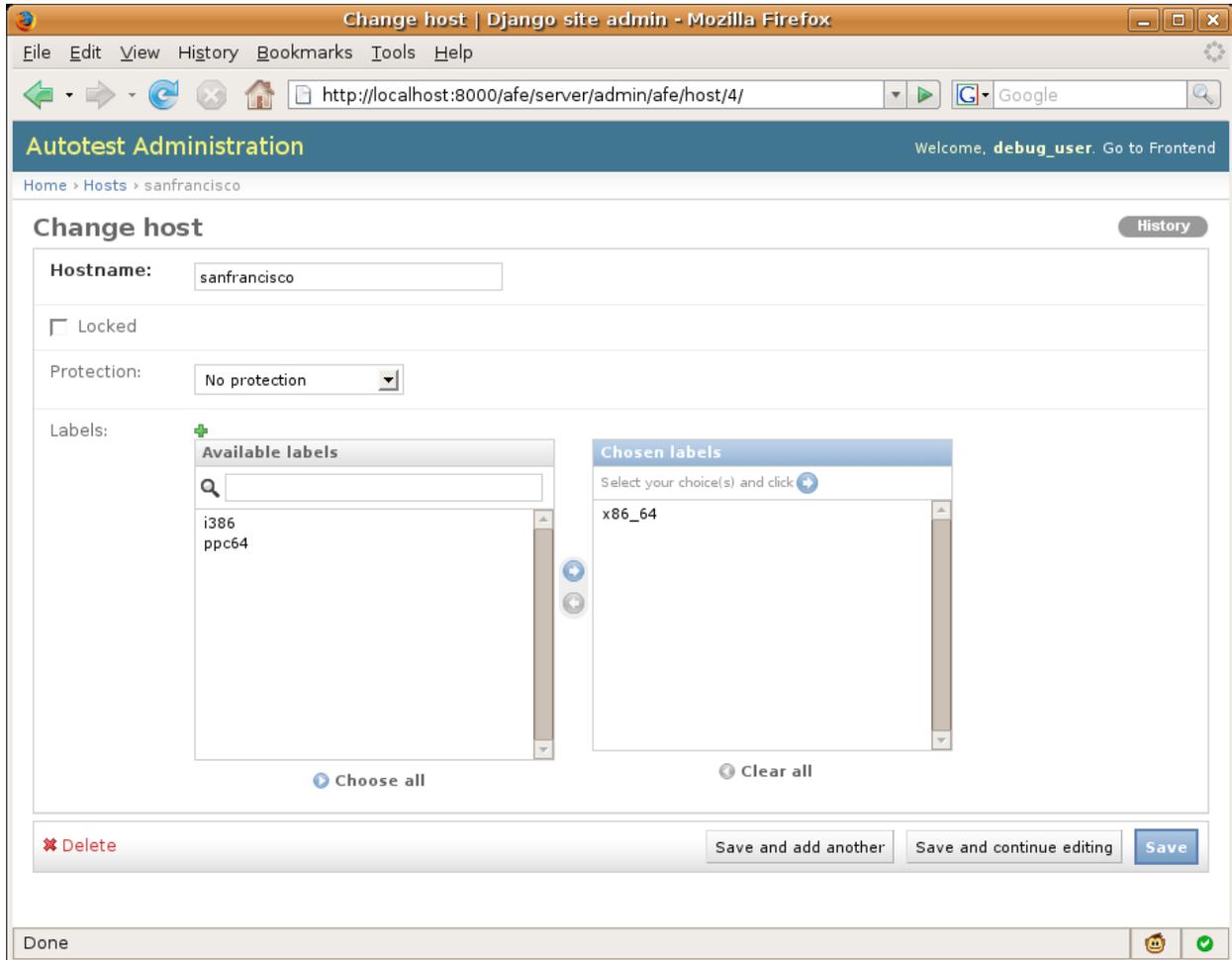
- the Autotest Frontend or AFE project is a GUI for managing jobs and hosts, including creation of new jobs and tracking queued and running jobs. It lives under the “frontend” directory. This is frequently referred to as simply “the web frontend”.
- the TKO project is a GUI for results reporting. It allows the user to view summarized test results across many jobs, filtered and grouped by various categories. It lives under the “new_tko” directory.

AFE

There are a few medium-sized features we’d like to complete:

- Implement complete ACL support – *partially done* ACL support is barely implemented right now – ACL-inaccessible hosts are hidden from the user in the GUI host list, and meta-hosts are blocked from being scheduled on inaccessible hosts. We need to add proper support for blocking the scheduling of inaccessible hosts, including support for superusers. We need ACL protection for aborting jobs and for modifying hosts.





- Creating jobs using previous jobs as templates – **done** When the “Requeue job” is clicked, instead of immediately creating a new job, the user will be taken to the “Create Job” tab. All the info from the old job will be filled in. The user will then have the option of making changes before submitting the new job.
- Easier management of many items (jobs + host) – **done** Currently, to abort many jobs, the user must click each job individually to go to its job detail page and then click the “Abort job” button. We’d like to allow the user to select many jobs in the job list page and then abort them all at once. Similar functionality could be used on the host list page to, for instance, send many hosts into repair.
- Better linking directly to raw logs (job + host logs) – **done** Jobs are often triaged by looking at the raw results logs. The only link to these from the frontend is the one “raw results logs” link on the job detail page, which takes the user to the root results directory for the job. The host queue entries table on the job detail tab should contain links to the debug logs for each host, and the host detail page should link to the host log for each host.
- Parsing and using information from control files - **done** The frontend should be able to parse information such as test types and descriptions from control files and put this information into the database. The frontend should display or use this information as appropriate. Most of it is already used or displayed, but some of this could be improved, such as the display of test descriptions (currently done with tooltips).

Larger features we’d like to have include:

- Host management features We’d like the Autotest frontend to have more powerful features for managing a large pool of hosts, including tracking of machine health and better support for machine repairs.
- Port admin interface to GWT Addition, modification and deletion of hosts, labels and tests is currently done through the Django admin interface. We’d like to port this functionality to GWT so that we can better customize it and integrate it with the rest of the frontend.

TKO

See `TkoWebRequirements` for reference.

Stage 1

done Basic spreadsheet view including all features of old TKO interface (or equivalent newer versions)

- SQL filtering conditions
- Row and column field selection
- Left-click default drilldown (single test cells go straight to logs instead of test detail view)
- Floating headers

Stage 2

done Enhanced spreadsheet features

- Right-click menu with drilldown options (and table-wide actions menu at top)
- Multiple cell selection
- Test labels

Stage 3

done Table view

- Column selection + ordering
- Grouping feature
- Left- and right-click actions
- Sorting
- Job triage options from spreadsheet view

Stage 4

never got implemented User-friendly filtering

- Filter widgets mode
- Filtering widgets for all fields
- Conversion to SQL with custom editing allowed

Longer term

Plotting functionality and test detail view **both done**

Configuring hosts on the Autotest server

How to configure your hosts in the Autotest service.

Hosts

Hosts must be added to the Autotest system before they can be used to run tests. Hosts can be added through the *one-time hosts* interface, but for repeated tests it's better to add them to the system properly. Hosts can be added through the admin interface (WebFrontendHowTo) or the CLI (CLIHowTo). Host options include:

- **hostname** – this is how the host will be identified in the frontend and CLI and how Autotest will attempt to connect to the host.
- **locked** – when a host is locked, no jobs will be scheduled on the host. Existing jobs will continue to completion.
- **protection** – see HostProtections.

Labels

Labels can be applied to machines to indicate arbitrary features of machines. The most common usage of labels is to indicate a machine's platform, but they can also be used to indicate machine capabilities or anything else the user likes. Labels are displayed in the frontend but also play an important role in AdvancedJobScheduling.

- **name** – this is how the label will be identified in the frontend and CLI
- **kernel_config** – **deprecated** this field is generally unused and should be removed

- **platform** – true if this label indicates a platform type. This option affects web frontend display only and has no effect on scheduling.
- **only_if_needed** – see `AdvancedJobScheduling#Onlyifneededlabels`.

ACLs

Access Control Lists restrict which users can perform certain actions on machines. They are primarily used to prevent other users from running jobs on a particular user's machines. See `ACLBehavior` for details on what ACLs control and how they work.

Each ACL is associated with some group of users and some group of machines. A user has ACL access to a machine if she is in any ACL group with that machine. By default, all users and machines are in the "Everyone" ACL, which essentially makes a machine publicly shared in the system.

Any user can create a new ACL using web frontend (`WebFrontendHowTo`) or CLI (`CLIFHowTo`).

Atomic Groups

See `AdvancedJobScheduling#AtomicGroups`

Setting a Graphing Filter



The screenshot shows a web interface for setting graphing filters. At the top, there are two radio buttons: "all of" (selected) and "any of". Below this is a dropdown menu currently showing "Test index" with a downward arrow. To the right of the dropdown is an empty text input field. To the right of the input field is a blue link "[X]". Below the input field is another blue link "[Add Filter]". At the bottom, there is a blue link "View Filter String" and a button "Edit Filter String".

These filters manipulate the data displayed and analyzed in your plots on the graphing interface. The [X] link next to each filter removes the associated filter from the list (or clears it, if there is only one), while the [Add Filter] link adds a new filter to the end of the list.

Interface Options

- **all of / any of**: Specify whether you want the data to satisfy all or any of the filters you listed.
- **database column (drop-down)**: Select the database column you are going to be filtering on. See *Graphing Fields* `<../frontend/Web/GraphingFilters>` for more details.
- **condition (textbox)**: Specify the condition you want to use for the database column you specified. You may enter any condition that is valid in a SQL WHERE clause. Examples:
 - = 12345
 - LIKE 'kernbench%'
 - REGEXP 'bad-dimm0[^0-9]*'

Filter String Viewer

In addition to the controls above, there is a viewer area in which you can see the SQL WHERE clause that the frontend is building. Click View Filter String to expand the textarea to show the clause. You may also click “Edit Filter String” to edit the WHERE clause yourself. You may use any of the fields specified in *GraphingDatabaseFields* <../frontend/Web/GraphingFilters>.

Preconfigured Graphing Queries

It is possible to build a preconfigured query and keep it on the server. These preconfigs will appear on the graphing interface under the **Preconfigured** control. Preconfig files are key:value pairs separated by lines that build the query on the frontend. See [[MetricsPlot]] and [[MachineQualHistograms]].

The two frontends have different preconfig formats:

- [[MetricsPreconfigs]]
- [[QualPreconfigs]]

Using the Metrics Plot Frontend

The **Metrics plot frontend** is able to generate a line or bar chart of most TKO database fields against aggregated values of most other TKO database fields. This is usually used to create plots of performance data versus some machine property, such as kernel version or BIOS revision.

Using the Interface

Interface Options

- **Graph Type:** Set to “Metrics Plot” to show this interface.
- **Preconfigured:** Select a preconfigured graphing query. Use this to automatically populate the fields in the interface to a preconfigured example. You may then submit the query for plotting as is, or edit the fields to modify the query. See Graphing Pre Configs to more information about preconfigured queries.
- **Plot:** Select whether you want a line plot or a bar chart.
- **X-axis values:** Select the values to place across the x-axis of the plot. For example, selecting “Kernel” create a plot against different kernel versions across the x-axis. See GraphingDatabaseFields for details about the different options. In addition to the options listed there, **X-axis values** also accepts “(Single Point)” as an input, which will plot all values on a single point on the x-axis; this is more applicable for bar charts than for line plots.
- **Global filters:** Set the filters to apply across all series of the plot. See GraphingFilters for more information on setting a filter.
- **Series:** Set each series that you would like to display. Clicking the [Add Series] link adds a series to the list. Each series has its own Delete Series link, which will remove the series from the list. If there is only one series and it is deleted, it will instead be reset.
 - **Name:** The name you want to give the series. It will be displayed as the title of its respective subplot if you requested multiple subplots, or as a label in the legend otherwise.
 - **Values:** The values you want to aggregate to plot on the y-axis. Typically, this is “Performance Keyval (Value)” to aggregate performance data.



[Spreadsheet](#) [Table](#) **[Graphing](#)** [Test details](#)
Refresh
Saved queries... ▾

Graph Type: Metrics Plot ▾

Preconfigured: ▾

Plot: Line ▾

X-axis values: (Single Point) ▾

Global filters:

all of any of
Test Index ▾ _____ [X]
[\[Add Filter\]](#)
[View Filter String](#) Edit Filter String

Series:

Name: _____ Invert y-axis
 Values: Test Index ▾
 Aggregation: AVG ▾ error bars
 Filters:
 all of any of
Test Index ▾ _____ [X]
[\[Add Filter\]](#)
[View Filter String](#) Edit Filter String
[Delete Series](#)

[\[Add Series\]](#)

Normalize to:

No normalization (multiple subplots)
 No normalization (single plot)
 Specified series: _____ ▾
 First data point
 Specified X-axis value: _____

Graph

- **Aggregation:** The type of aggregation you want to do on the data returned for each x-axis point. For example, specifying “AVG” will plot the average of the value you selected above for each point on the x axis.
- **error bars:** If the **Aggregation** is “AVG”, you may check this box to show the standard deviations of each point as error bars.
- **Filters:** Set the filters you want to apply to this particular series. See `GraphingFilters` for more information on setting a filter.
- **Invert y-axis:** Check this box if you want higher numbers towards the bottom of the y-axis for this series.
- **Normalize to:** Set the normalization you want to use on this plot.
 - **No normalization (multiple subplots):** Do not normalize the data, and display each series on a separate subplot. Note that this option is only available for Line plots.
 - **No normalization (single plot):** Do not normalize the data, and display all series on a single plot. This is the default option.
 - **Specified series:** Graph all series as percent changes from a particular series. That is, for each point on each series, plot the percent different of the y-value from the y-value of the specified series at their corresponding x-value. The series that you normalize against will not be plotted (since all values will be 0). If the series you normalize against does not have data for some x-values, those values will not be plotted.
 - **First data point:** Graph all series, renormalized to the first valid data point in each series.
 - **Specified X-axis value:** Graph all series, renormalized to the data point at the specified x-axis value for each series. This is similar to the above option, but rescales the y-axis for a point other than the first data point. You must enter the exact name of the x-axis value.

Interacting with the Graph

The four main actions you can do on the graph are:

- **Hover:** Hovering the cursor over a point or bar shows a tooltip displaying the series that the point or bar is from, and the x- and y-values for that data.
- **Click:** Clicking on a point or bar opens a drill-down dialog. The dialog shows a sorted list of all the y-values that were aggregated to form the point or bar. Clicking on any particular line in that list jumps to the **Test detail view** describing the test that generated that line of data.
- **Embed:** Clicking the [Link to this Graph] link at the bottom-right of the generated plot displays an HTML snippet you can paste into a webpage to embed the graph. The embedded graph updates with live data at a specified refresh rate (as the `max_age` URL parameter, which is in minutes), and show an indication of the last time it was updated. Clicking on the embedded graph links to the **Metrics plot frontend**, automatically populated with the query that will generate the graph. See `AutotestReportingApi` for a more powerful way to embed graphs in your pages.
- **Save:** The graph is delivered as a PNG image, so you can simply right-click it and save it if you want a snapshot of the graph at a certain point in time.

Metrics Preconfigs

Metrics preconfigs should be put in `<autotest_dir>/new_tko/tko/preconfigs/metrics/`

The parameters are:

- **plot:** Line or Bar

- **xAxis**: Database column name for the **X-axis values** control. See GraphingDatabaseFields.
- **globalFilter[i][db]**: Database column name for the i^{th} global filter (start at 0). See GraphingDatabaseFields.
- **globalFilter[i][condition]**: Condition field for the i^{th} global filter (start at 0).
- **globalFilter_all**: This controls if you have “all of” or “any of” selected as the filter combination operation for the global filters. Set to `true` for “all of”, and `false` for “any of”.
- **name[j]**: The name of the j^{th} series.
- **values[j]**: The database column name that should be plotted on the y-axis for the j^{th} series. See GraphingDatabaseFields.
- **aggregation[j]**: The aggregation to be applied to the data of the j^{th} series. Available aggregations are:
 - AVG
 - COUNT (DISTINCT)
 - MIN
 - MAX
- **errorBars[j]**: Sets if the error bars should be shown for the j^{th} series, if the aggregation is AVG. Set to `true` to show error bars, `false` to keep them hidden.
- **seriesFilters[j][k][db]**: Database column name for the k^{th} filter of the j^{th} series. See GraphingDatabaseFields.
- **seriesFilters[j][k][condition]**: Condition field for the k^{th} filter of the j^{th} series.
- **seriesFilters[j]_all**: This controls if you have “all of” or “any of” selected as the filter combination operation for the filters on the j^{th} series. Set to `true` for “all of”, and `false` for “any of”.

Example:

```

plot: Line
xAxis: kernel
globalFilter[0][db]: hostname
globalFilter[0][condition]: = 'my_test_host'
globalFilter_all: true
name[0]: dbench (throughput)
values[0]: iteration_value
aggregation[0]: AVG
errorBars[0]: true
seriesFilters[0][0][db]: iteration_key
seriesFilters[0][0][condition]: = 'throughput'
seriesFilters[0][1][db]: test_name
seriesFilters[0][1][condition]: = 'dbench'
seriesFilters[0]_all: true
name[1]: unixbench (score)
values[1]: iteration_value
aggregation[1]: AVG
errorBars[1]: true
seriesFilters[1][0][db]: iteration_key
seriesFilters[1][0][condition]: = 'score'
seriesFilters[1][1][db]: test_name
seriesFilters[1][1][condition]: = 'unixbench'
seriesFilters[1]_all: true
    
```

Machine Qualification Preconfigs

Machine qualification preconfigs should be put in `<autotest_dir>/new_tko/tko/preconfigs/qual/`

The parameters are:

- **globalFilter[i][db]**: Database column name for the i^{th} global filter (start at 0). See `GraphingDatabaseFields`.
- **globalFilter[i][condition]**: Condition field for the i^{th} global filter (start at 0).
- **globalFilter_all**: This controls if you have “all of” or “any of” selected as the filter combination operation for the global filters. Set to `true` for “all of”, and `false` for “any of”.
- **testFilter[j][db]**: Database column name for the j^{th} test set filter (start at 0). See `GraphingDatabaseFields`.
- **testFilter[j][condition]**: Condition field for the j^{th} test set filter (start at 0).
- **testFilter_all**: This controls if you have “all of” or “any of” selected as the filter combination operation for the test set filters. Set to `true` for “all of”, and `false` for “any of”.
- **interval**: Sizes of the bins in the histogram.

Example:

```
globalFilter[0][db]: hostname
globalFilter[0][condition]: LIKE 'my_host_names%'
globalFilter[1][db]: hostname
globalFilter[1][condition]: LIKE 'my_other_host_names%'
globalFilter_all: false
testFilter[0][db]: test_name
testFilter[0][condition]: = 'my_test_name'
testFilter_all: true
interval: 10
```

TKO Web Interface Requirements

The TKO web interface is a system to generate customizable reports summarizing test results across many jobs. Whereas AFE focuses on displaying execution status of individual jobs, TKO focuses on displaying pass/fail results for individual tests. It has options for filtering out various subsets of test results, grouping test results along various dimensions, and displaying the results in different ways.

The new TKO UI will be a dynamic web application broadly resembling AFE. Like AFE, the interface will be divided into tabs.

Overview

- There will be four main tabs: **spreadsheet view**, **table view**, **plotting view**, and **test details**.
- To the right of these tabs will be a refresh button, followed by a “**Saved queries**” drop-down box. This box will allow the user to save a particular view, including which tab is being viewed, the filtering conditions, and any parameters configuring the display. The box will display a list of saved queries for the user as well as an option to save a new query. Queries will have history support (see below), so they can be shared via URLs (i.e. something like http://myautotestserver/tko/#saved_query_1234).
- To the right of the saved queries will be a “**Download CSV**” link.
- The interface will have full **history support**. This will include changing the browser title when changing certain view parameters. This provides two benefits:
 - users can share reports by copy-pasting URLs.

- browser history will serve as a useful way to navigate among recent queries.

Filtering conditions

- All TKO activities involving filtering down to some subset of all recorded test data. All views will share a common interface for specifying these conditions. There will be two ways to specify these conditions: via **filtering widgets** for each field, or via a single **custom SQL** text area. The custom SQL text area is the analogue of the condition text box in the old TKO interface.
- The UI will default to filtering widgets, with a button to go to custom SQL mode. When switching to custom SQL, the current widget selections will be converted to SQL. The widgets will be replaced with a single text area, in which the user can then edit the SQL condition. She may also click the button to start with and write a SQL condition from scratch. Edited SQL can not be converted back to widgets – changes will have to be reverted. This is analogous to the “Edit control file” button in AFE.
- Filtering widgets mode will initially display a drop-down box of fields on which to filter. This list includes **hostname, host keyval, host labels, job name, job tag, failure reason, test keyval, test labels, test name, test status, time queued, time started, time completed, user**.
- Selecting a field from the drop-down will display a selection widget for that field. The widget varies with the field. For most fields, there will be a **pair of list boxes** displaying the available and selected values for the field. For some fields, there will be an alternative option to enter a **regex** to match. Some fields may be completely different (i.e. time fields will allow the user to define ranges via start and end times, with calendar- and clock-like helper widgets available).
- To the right of each filter widget will be “+” and “-” buttons, allowing the user to **add another filter** and **delete the given filter**, respectively.

Spreadsheet view

- This view is the future version of what the existing TKO interface does. It allows the user to group by two fields, one for row headers and one for column headers. It then displays counts of passed test runs and all test runs within each grouping.
- **Incomplete (queued and running) tests** are included in the spreadsheet, unless filtered out.
- At the top, below the filtering area, will be a drop-down box to select the **row** and **column** grouping fields. This is just like the old TKO interface. Below each box will be a “**Customize rows/columns...**” link, which will expand to allowing the user to do two things:
 - select multiple fields for row or column headers to create **composite headers** (and customize the field ordering)
 - customize ordering of row and column values.
- Just above the spreadsheet will be a drop-down box with **table-wide actions**. It will resemble the right-click context menus (see below).
- The displayed spreadsheet will look similar to how it does today, but will have **floating row and column headers**, much like Excel or Google Spreadsheets.
- Left-clicking on a cell will perform a *default drilldown* operation as it does in the old interface.
- Right-clicking on a cell will bring up a **context menu**.
 - Cells with multiple test runs will have a number of **drill down options** first, showing different combinations of row-column fields to drilldown to.
 - Cells with a single test run will have a single option at the top to **view test details** (this is the default drilldown option). This will bring the user to the test details tab.

- All cells will have an option to **switch to table view**, to **triage failures** (see below), and to **apply or remove a label**. Apply/remove label will bring up a small dialog allowing the user to select which label to use.
- **Row and column headers** will act like cells with multiple test runs.
- Ctrl-left-clicking on a cell will select (or deselect) the cell. Multiple cells can be selected and then right-clicking can be used to act on all selected cells.

Table view

- This view will display individual test runs as rows within a table. The columns and sorting can be customized. It also has the capability to group and show counts.
- Below the filtering area at the top will be a selection widget allowing the user to **select and order the columns displayed**.
- Below the column selection will be a check box to **“Group by these columns and show counts”**. When this is selected, results will be grouped by all selected columns and each row will show the count of test runs within that group.
- Clicking on a column header will **sort** the table on that column.
- Left-clicking on a row will bring the user to the test details tab. Right-clicking on a row will bring up a menu allowing the user to go to test details or to apply/remove labels.
- Left-clicking on a grouped row will drilldown to an ungrouped table view. Right-clicking will bring up a menu allowing drilldown or apply/remove labels.
- **Job triage** view is a particular table view. It is a grouped table view, with columns for job tag, test name, and failure reason. It is sorted by these columns in this order, and finally by counts descending. This view is particularly useful for triaging failures among many test runs and is therefore accessible via shortcuts from spreadsheet view.

Plotting view

- Detailed requirements for the plotting view have yet to be determined.

Test details

- This view will display detailed information for a single test run. All of the fields for a test will be displayed, including all hosts on which a test ran and their attributes and all test and iteration keyvals. Key **log files** will also be readily accessible in expandable boxes, including status.log, autoserv.stdout, autoserv.stderr, and client.log.*.

New UI user requirements

Use cases

- **Job tracking** - viewing a spreadsheet of tests vs machines for a given job, with cells showing status of each test on each machine (queued, running, passed, failed, etc.). Tests can be sorted in the order in which they ran. Results logs are easily accessible. *This is mostly available in the old interface. The addition of queued/running tests will be the biggest addition. Sorting tests in running order is not as simple as it seems (control files aren't guaranteed to be deterministic, for example). We have ideas about how to solve that but we've deferred it for now.*

- **Job triage** - viewing a summary of failure reasons for a job. The view should display a list of unique failure reasons for each test (including job failures) with information on the frequency of each failure reason. It should be easy to view the list of machines that failed for each reason with links to detailed log files. *See “job triage” feature.*
- **Kernel test status** - viewing a spreadsheet of kernel versions vs tests for a set of “official kernel test” jobs, with cells showing success rates. User can select which kernel versions to include. It should be easy to:
 - group headers for kernel versions, so that the user can compare multiple release candidates within multiple kernel versions
 - drill down to see machine architecture vs tests for a particular kernel version, to assist in triaging architecture-specific failures
 - drill down to see failure reasons for failures of a particular test on a particular kernel. As with job triage, this should make it easy to drill down to machine lists for each failure reason. *Test labels solve the “official kernel tests” problem. Filter widgets will ease selection of included kernels. Grouping headers by kernel version will***not***be supported for now (this is not to be confused with composite headers, which combines two different fields). Different drill downs are supported via context menus.*
- **Test series** - user has a pool of machines and runs a test on all machines. Machines that fail are triaged and the tests is rerun on them, and so on until all machines pass. User should be able to view status of last run test within the series for each machine. Triage of failed machines should be easy, as in **Job triage**. Additionally, user can see state of non-passed machines - failed awaiting triage, triaged awaiting re-test, re-test queued/running, etc. *Test labels should support this workflow. It will still require a fair bit of work on the part of the user, but we felt this was a necessary tradeoff in order to avoid putting too much specialized complexity in the frontend. Multiple selection should allow fairly powerful label usage, which, in combination with saved queries and filter widgets, should ease the pain greatly.*
- **Machine utilization** - viewing a chronological history of all tests (and verifies/repairs?) run on a particular machine. Test/verify/repair outcome information is displayed, making it easy to track down when a certain test started failing or when machine verification first failed. Detailed logs are easily accessible. *Table view should provide this basic feature. The main lacking aspect is inclusion of verify and repair info. This is certainly doable but requires further discussion.*
- **Performance graphs** - plotting performance data vs. kernel version for many iterations of a particular test on a particular machine. *This, along with the other plotting use cases below, are not being addressed now.*
- **Machine qualification graphs** - plotting a histogram of percentage of tests passed on each machine, with bars clickable to view list of machines in each bucket.
- **Utilization graphs** - plotting machine utilization as a percentage of time vs. machine, over a given span of time.
- **Generic keyval graphs** - user selects a set of kernels, a set of machines, and a set of tests. In a single graph, all keyvals are plotted together (normalized) vs. kernel version. The ordering of kernels is completely user-definable. Data points link back to results logs.
- **Kernel benchmark comparisons** - plotting a set of benchmark values for a pair of kernels together, to compare the two versions.
- **Job set comparisons** - plotting a set of benchmark values for two sets of jobs together.

Specific feature requests

- Clicking on a kernel brings up a tests vs. status spreadsheet filtered for that particular kernel (*possible with drilldown options*) *This is a easy shortcut for bringing up a particular report.*
- Reason values displayed in table or one click away (*job triage view*) *When triaging a job or jobs with many failures, there needs to be a easy way to view a summary of the reasons for failures (from the DB “reason”*

field). Similar reasons should be grouped together and it should be easy to see which hosts failed with which reasons.

- Include tests that are queued or running in TKO display (included) Right now TKO only shows tests that have completed. It should also display queued and running tests so the user can get a full picture of a job from a single report.
- Preserve and display query history (included as browser history) The UI should present a list of the last few (or many) spreadsheet queries executed, including drilldown history. The user should be able to click to go back to a previous query.
- Filtering on a list of kernels/jobs to match (filter widgets) The user should be able to easily specify a list of kernels and filter down to tests run on any of those kernels. Likewise for filtering to a list of jobs.
- Kernels must sort in chronological order (not addressed; this is a very particular request which we may address with specialized code) Most fields simply sort alphanumerically, but kernels must sort specially so that they come out in chronological order.
- Clicking on a kernel brings up a list of failed machines (context menus) This is another easy shortcut for bringing up a particular report.
- Ability to have more than one grouping field for rows or columns (aka “composite headers” or “multiple headers”) (included) For example, the user might specify two fields for row grouping and the resulting spreadsheet would have a row for each combination of values from the two fields.
- Grouping on custom expressions (not included; potential future addition) Instead of simply specifying a field to group on, the user could specify a custom SQL-like expression.
- More powerful filtering by machine labels (should be possible with appropriate usage of machine labels) The user should be able to filter on machine types both very specifically (i.e. Intel Pentium D 1GB RAM) and very generally (i.e. all Intel).
- Easy way to keep track of where the user is in a large table (when row and column headers are no longer visible) (floating headers) When browsing a large table, after scrolling to the right and down, the row and column headers are no longer visible and the user may have no way to know what values a particular cell corresponds to.
- Machine-centric view showing utilization of a particular machine over time (see use case; graphical timeline not included) This view would show a list of things that have been run on the machine in chronological order, so the user could get some idea of how the machine’s been utilized. The ability to view percentage of time in use would be good. A graphical timeline sort of view would also be good.
- CSV data export (included) The user should always be able to download the currently displayed data in CSV format.
- Invalidation of jobs (solved with machine labels) The user should be able to mark jobs (perhaps even individual tests) as invalid and have them excluded from TKO reporting.
- Powerful and flexible filtering (included) Selections can be specified by choosing from a list, by regexp matching or by entering raw SQL expressions
- Automatic bug filing (not included) When triaging failures, the user can click a button to create a new bug in a bugtracking system and have job and failure information automatically bundled up and attached to the bug.
- Filtering on keyvals (included) Users should be able to filter on any keyval when filtering results

Autotest Reporting API

The Autotest Reporting API allows you to embed TKO spreadsheets, tables and graphs into your own HTML pages. This can be used to create powerful, customizable dashboards based on Autotest results.

Currently, only graphs are supported. Spreadsheets and tables are coming soon.

Setup

In order to use the Autotest Reporting API, your HTML page needs to load the Autotest Reporting API Javascript library and then call it to create widgets. Here's a simple skeleton:

```
<!DOCTYPE html>
<head>
  <script type="text/javascript" src="http://your-autotest-server/embedded-tko/
↪autotest.EmbeddedTkoClient.nocache.js">
  <script type="text/javascript">
    function initialize() {
      Autotest.initialize("http://your-autotest-server");

      // code to setup widgets goes here.  for example:
      var plot = Autotest.createMetricsPlot(document.getElementById("plot_canvas"));
      plot.refresh(...); // see below
    }
  </script>
</head>

<body onload="initialize()">
  <!-- document outline goes here.  for example: -->
  <div id="plot_canvas"></div>
</body>
```

The first script tag loads the Autotest Reporting API library. The `initialize()` function then calls `Autotest.initialize()`, which tells the library where to find the Autotest server running the TKO web interface. Finally, it can proceed to call `Autotest.create*` methods to create widgets. All `Autotest.create*` methods accept a DOM Element to which they will attach themselves.

Graphing

You can create a `MetricsPlot` widget using `Autotest.createMetricsPlot(parentElement)`. `MetricsPlot` widgets have one method, `refresh(parameters)`. This interface will be changing soon so it won't be documented in detail; please see the example in `frontend/client/src/autotest/public/EmbeddedTkoClientTest` or ask showard if you would like to use it and have questions.

Autotest Web Frontend Implementation details

Here we outline the building blocks and implementation details of the autotest web interface.

Overview

Here's a broad overview of how the system fits together:

[[FrontendImplementationDetails/frontend_overview.png]]

- The **Django RPC server** is an RPC server, written using the Django framework. It functions as a web server, accepting RPCs as HTTP POST requests, querying the MySQL database as necessary, and returning results. In a production environment, it runs within Apache using `mod_python`
 - The AFE server code lives under `frontend/afe` and uses the `autotest_web` database.

- The TKO server code lives under `new_tko/tko` and uses the `tko` database.
- In both servers, the RPC entry points are defined in `rpc_interface.py`.
- All RPC POST requests go to a single URL, `(afe|new_tko)/server/rpc/`. They get dispatched to RPC methods by the code in `rpc_handler.py`. See [Django documentation](#) for an explanation of how HTTP requests get mapped to Python code using URLconfs.
- Database models live in `models.py`. See [Django documentation](#) for an explanation of models.
- **RPC calls** and responses are encoded according to the JSON-RPC protocol.
 - JSON is a simple data representation format based on Javascript. See <http://json.org>.
 - JSON-RPC is a very simple standard for representing RPC calls and responses in JSON. See <http://jsonrpc.org>.
 - RPCs are made by sending a POST request to the server with the POST data containing the JSON-encoded request. The response text is a JSON-encoded response.
 - * On the server, the code for serializing JSON lives at `frontend/afe/simplejson`. The code for forming and dispatching JSON-RPC requests lives at `frontend/afe/json_rpc`.
 - * The CLI uses the same code for serializing JSON-RPC.
 - * The GWT client uses GWT's builtin JSON library for serializing JSON. The code for handling JSON-RPC requests is in `autotest.common.JsonRpcProxy` and friends.
- The **GWT client** is a browser-based client for AFE and TKO (technically, there are two separate clients). It's written using Google Web Toolkit (GWT), a framework for writing browser apps in Java and having them compiled to Javascript. See <http://code.google.com/webtoolkit>.
 - More details...
- The **CLI** is a command-line Python application that makes calls to the RPC server. It lives under the `cli` directory. `cli/autotest-rpc-client` is the main entry point.

Host Protection Levels

Host protection levels are used to protect particular hosts from actions that occur during the verify and repair phases. These can be set using the CLI or the frontend admin interface. They are defined in `client/common_lib/host_protections.py` and contained in the `protection` field of the `hosts` table in the `autotest_web` database.

- **No protection** – anything can be done to this host.
- **Repair software only** – any software problem can be fixed, including a full machine reinstall.
- **Repair filesystem only** – the filesystem can be cleaned out, but not system reconfiguration or reinstall can occur.
- **Do not repair** – do not attempt any repair on the machine.
- **Do not verify** – do not verify or repair the machine (the machine will be assumed to be in working order).

Specifying kernels in the Job Creation Interface

Autotest has a system to expand Linux kernel versions to actually downloadable source trees, or even installable distro packages, that can be used in job creation interfaces, such as CLI and web interfaces. At the moment, we support the following release schemas:

- Upstream versions. You can specify an upstream version, that will expand to an URL pointing to a tarball inside the kernel.org mirror you have specified. The script/library `client/kernalexpend.py` has this functionality implement, and lets you test it which versions can be actually expanded:

```
$ client/kernalexpend.py 3.2.1
http://www.kernel.org/pub/linux/kernel/v3.x/linux-3.2.1.tar.bz2
```

We still don't allow you to specify an arbitrary distro package version for autotest to download, for example:

```
$ client/kernalexpend.py 3.3.4-5.fc17.x86_64
Kernel '3.3.4-5.fc17.x86_64' not found. Please verify if your version number is_
↪correct.
```

- Direct URLs pointing to rpm and deb packages containing the kernel. Example:

```
http://example.com/kernel-3.3.1.rpm
http://example.com/kernel-3.5-rc2.deb
```

You can specify multiple versions separating them with a comma or space.

Obviously, we'd like to cleanly support other ways of specifying kernels in the job creation interface, so this makes the complicated logic transparent to users, but we're not there yet. Please open an issue requesting for a given method and we'll consider it carefully.

Using the Machine Qualification Histogram Frontend

The **Machine qualification histogram frontend** is able to generate a histogram of test pass rates for a specified set of tests and machines. The histogram shows bins of configurable size for pass rates between 0 and 100, exclusive, as well as special bins for 0% and 100% pass rates. There is also an "N/A" bin, which shows the machines that did not run any of the tests that you specified to analyze.

[[MachineQualHistograms/machine_qual_interface.png]]

Using the Interface

Interface Options

- **Graph Type:** Set to "Machine Qualification Histogram" to show this interface.
- **Preconfigured:** Select a preconfigured graphing query. Use this to automatically populate the fields in the interface to a preconfigured example. You may then submit the query for plotting as is, or edit the fields to modify the query. See `Graphing Pre Configs` for more information about preconfigured queries.
- **Global filters:** Set the filters on the machines you would like to see. Any machine that satisfies the filter will be plotted in the histogram in some way. See `GraphingFilters` for more information on setting a filter.
- **Test set filters:** Set the filters on the tests that you want to analyze. The pass rates for what you enter in this filter will be plotted on the histogram. If a machine satisfies the **Global filters** above but has not run any tests that satisfy the **Test set filters**, it will appear in the "N/A" bin. See `GraphingFilters` for more information on setting a filter.
- **Interval:** Configure the size of each bin. For example, an interval of 5 means that the bins should be 0%-5%, 5%-10%, etc.

Interacting with the Graph

The four main actions you can do on the graph are:

- **Hover:** Hovering the cursor over a bar shows a tooltip displaying the boundaries of the bin and the number of machines in that bin.
- **Click:** Clicking on a bar jumps to the **Table view**, automatically configured to show the specific machines and pass rates in that bin.
- **Embed:** Clicking the [Link to this Graph] link at the bottom-right of the generated plot displays an HTML snippet you can paste into a webpage to embed the graph. The embedded graph updates with live data at a specified refresh rate (as the `max_age` URL parameter, which is in minutes), and show an indication of the last time it was updated. Clicking on the embedded graph links to the **Machine qualification histogram frontend**, automatically populated with the query that will generate the graph.
- **Save:** The graph is delivered as a PNG image, so you can simply right-click it and save it if you want a snapshot of the graph at a certain point in time.

Existing Graphing Scripts Frontend

The **Existing graphing scripts frontend** is a graphical frontend to some existing graphing CGI scripts in TKO.



Spreadsheet
Table
Graphing
Test details
Refresh
Saved queries... ▾

Graph Type: Existing Graphs ▾

Normalize Performance (allows multiple benchmarks on one graph)

Hostname:

Benchmark: (Please select a hostname first) ▾

Kernel:

Interface Options

- **Normalize Performance:** This checkbox allows you to normalize the performance numbers to percent differences instead of absolute numbers. Checking this option also allows you to select more than one benchmark at a time in the **Benchmark** control.
- **Hostname:** Name of the machine you want to analyze. As you begin typing, this textbox will show suggested completions based on all the hosts present in your TKO database.

- **Benchmark:** This control will either be a drop-down box or a multiple-select box, depending on if **Normalize Performance** is checked or not. Select the benchmarks you want to analyze here. Only *kernbench*, *dbench*, *tbench*, *unixbench*, and *iozone* are supported.
- **Kernel:** Specify the kernels that you want to have appear on the x-axis, or `all` for all versions with data matching the hostname and benchmark specifications above.

System Administration

Installing an Autotest server (Ubuntu/Debian version)

Install script

We have developed a script to automate the steps described below on a Ubuntu 12.04/12.10 server. So if you want to save yourself some time, please check the *Installing Server/Scheduler/WebUI notes*.

If you want to do it all yourself, we opted by keeping the documentation herem and we'll do the best to update it. However, we're always working on streamlining this process, so it might be possible that this can get out of sync.

If you find any step that might be outdated, please let us know, and we'll fix it.

Server/Scheduler/Web UI Installation Steps

Install required packages

Autotest is a complex project and requires a number of dependencies to be installed.

Note: Currently autotest is compatible with Django 1.5, so if your distribution has anything lower or higher than this version, you will have problems and are advised to use a compatible version.

We have automated this step on recent Ubuntu (12.04/12.10), although it should work on Debian too:

```
sudo /usr/local/autotest/installation_support/autotest-install-packages-deps
```

If you want to install it manually here it goes. Keep in mind this can be outdated, if so we kindly ask your help with keeping it up to date.

Install utility packages:

```
apt-get install -y unzip wget gnuplot makepasswd
```

Install webserver related packages (and Django):

```
apt-get install -y apache2-mpm-prefork libapache2-mod-wsgi python-django
```

Install database related packages:

```
apt-get install -y mysql-server python-mysqldb
```

Install java in order to compile the web interface, and git for cloning the autotest source code repository:

```
apt-get install git openjdk-7-jre-headless
```

Also, you'll need to install a bunch of auxiliary external packages

```
apt-get install python-imaging python-crypto python-paramiko python-httplib2 python-
↳numpy python-matplotlib python-setuptools python-simplejson
```

Important notes

Important: For this entire documentation, we will assume that you'll install autotest under /usr/local/autotest. If you use a different path, please change /usr/local/autotest accordingly. Please that you may have some issues with apache configuration if you don't choose /usr/local/autotest.

Important: We will also assume that you have created an autotest user on your box, that you'll use to perform most of the instructions after the point you have created it. Most of the instructions will use autotest unless otherwise noted.

Creating the autotest user

As root:

```
useradd autotest
passwd autotest [type in new password]
```

Cloning autotest

You can then clone the autotest repo (as root):

```
cd /usr/local
git clone --recursive git://github.com/autotest/autotest.git
chown -R autotest:autotest autotest
```

Log out, re-log as autotest, and then proceed.

Setup MySQL

Please check the shared *Configuring Autotest Server Database notes*

Install extra packages

Run the build script to install necessary external packages. If you ran the package install script, you should have all you could get from your system packages and it would download only GWT. As autotest:

```
/usr/local/autotest/utils/buildexternals.py
```

Always re-run this after a git pull if you notice it has changed, new dependencies may have been added. This is safe to rerun as many times as you want. It will only fetch and build what it doesn't already have. It's important to note that the autotest scheduler will also try to run buildexternals.py whenever it's executed in order to make sure every piece of software has the right versions.

NOTE: Set the HTTP_PROXY environment variable to <http://proxy:3128/> before running the above if your site requires a proxy to fetch urls.

Update Apache config

If the only thing you want to do with Apache is run Autotest, you can use the premade Apache conf:

Ubuntu 12.04

```
sudo rm /etc/apache2/sites-enabled/000-default
sudo ln -s /etc/apache2/mods-available/version.load /etc/apache2/mods-enabled/
sudo ln -s /usr/local/autotest/apache/conf /etc/apache2/autotest.d
sudo ln -s /usr/local/autotest/apache/apache-conf /etc/apache2/sites-enabled/001-
↳autotest
sudo ln -s /usr/local/autotest/apache/apache-web-conf /etc/apache2/sites-enabled/002-
↳autotest
```

Ubuntu 12.10 - The version plugin now is compiled into apache, so it can't be enabled, otherwise you will have trouble.

```
sudo rm /etc/apache2/sites-enabled/000-default
sudo ln -s /usr/local/autotest/apache/conf /etc/apache2/autotest.d
sudo ln -s /usr/local/autotest/apache/apache-conf /etc/apache2/sites-enabled/001-
↳autotest
sudo ln -s /usr/local/autotest/apache/apache-web-conf /etc/apache2/sites-enabled/002-
↳autotest
```

You will have to comment the line

```
WSGISocketPrefix run/wsgi
```

In */usr/local/autotest/apache/conf/django-directives*, as we found out that WSGI configuration varies among distros, and the version shipped with Ubuntu 12.04 is not compatible with this directive.

Also, you'll need to enable rewrite mod rules, which you can do by

```
a2enmod rewrite
```

Then, update your apache2 service

```
update-rc.d apache2 defaults
```

If you want to do other things on the Apache server as well, you'll need to insert the following line into your Apache conf, under the appropriate `VirtualHost` section:

```
Include "/usr/local/autotest/apache/apache-conf"
Include "/usr/local/autotest/apache/apache-web-conf"
```

And make sure the rewrite mod is enabled, as well as the autotest config file directory is properly linked:

```
sudo ln -s /etc/apache2/mods-available/version.load /etc/apache2/mods-enabled/
sudo ln -s /usr/local/autotest/apache/conf /etc/apache2/autotest.d
```

Note: You will have to enable `mod_env` on SuSE based distro's for the all-directives to load properly when apache is started.

Update Autotest config files

Important: Edit the following files to match the database passwords you set earlier during session #Set_up_MySQL, as autotest, more specifically, `MYSQL_AUTOTEST_PASS`.

```
/usr/local/autotest/global_config.ini
/usr/local/autotest/shadow_config.ini
```

Important: Please, do *not* change this field

```
[AUTOTEST_WEB]
# Machine that hosts the database
host: localhost
```

As we are doing the setup on the same machine where mysql is running, so *please, pretty please* don't change it otherwise you will have trouble moving forward.

Things that you usually want to change on *global_config.ini*:

Section AUTOTEST_WEB

```
# DB password. You must set a different password than the default
password: please_set_this_password
```

Section SCHEDULER

```
# Where to send emails with scheduler failures to
# (usually an administrator of the autotest setup)
notify_email:
# Where the emails seem to come from (usually a noreply bogus address)
notify_email_from:
```

Section SERVER

```
# Use custom SMTP server
# If none provided, will try to use MTA installed on the box
smtp_server:
# Use custom SMTP server
# If none provided, will use the default SMTP port
smtp_port:
# Use custom SMTP user
# If none provided, no authentication will be used
smtp_user:
# Use SMTP password
# It only makes sense if SMTP user is set
smtp_password:
```

Run DB migrations to set up DB schemas and initial data

Important: If you set up your database using autotest-database-turnkey, this step can be safely skipped.

During the time span of the project, the autotest database went through design changes. In order to make it able for people running older versions to upgrade their databases, we have the concept of migration. Migration is nothing but starting from the initial database design until the latest one used by this specific version of the application. As autotest:

```
/usr/local/autotest/database/migrate.py --database=AUTOTEST_WEB sync
```

Run Django's syncdb

Important: If you set up your database using autotest-database-turnkey, this step can be safely skipped.

You have to run syncdb twice, due to peculiarities of the way syncdb works on Django. As autotest:

```
/usr/local/autotest/frontend/manage.py syncdb
/usr/local/autotest/frontend/manage.py syncdb
```

Compile the GWT web frontends

Compile the Autotest web application and TKO frontend. As autotest:

```
/usr/local/autotest/utils/compile_gwt_clients.py -a
```

You will need to re-compile after any changes/syncs of the frontend/client pages.

Fix permissions

Make everything in the `/usr/local/autotest` directory world-readable, for Apache's sake:

```
chmod -R o+r /usr/local/autotest
find /usr/local/autotest/ -type d | xargs chmod o+x
```

Restart apache

```
sudo apache2ctl restart
```

Test the server frontend

You should be able to access the web frontend at <http://localhost/afe/>, or <http://your.server.fully.qualified.name.or.ip/afe/>

Start the scheduler

Executing using SysV init scripts

To start the scheduler on reboot, you can setup init.d.

```
sudo cp /usr/local/autotest/utils/autotest.init /etc/init.d/autotestd
sudo update-rc.d /etc/init.d/autotestd defaults
```

Then, you can reboot and you will see `autotest-scheduler-watcher` and `autotest-scheduler` processes running.

Executing using systemd (Debian Unstable)

If you're using systemd, we ship a systemd service file. Copy the service file to systemd service directory. As root or using `sudo`:

```
sudo cp /usr/local/autotest/utils/autotestd.service /etc/systemd/system/
```

Make systemd aware of it:

```
sudo systemctl daemon-reload
```

Start the service:

```
sudo systemctl start autotestd.service
```

Check its status:

```
autotestd.service - Autotest scheduler
  Loaded: loaded (/etc/systemd/system/autotestd.service)
  Active: active (running) since Wed, 25 May 2011 16:13:31 -0300; 57s ago
  Main PID: 1962 (autotest-schedu)
  CGroup: name=systemd:/system/autotestd.service
          1962 /usr/bin/python -u /usr/local/autotest/scheduler/autotest-
↪scheduler-watcher
          1963 /usr/bin/python -u /usr/local/autotest/scheduler/autotest-
↪scheduler /usr/local/autotest/results
```

Executing manually using screen (not recommended)

You can execute the babysitter scripter through, let's say, nohup or screen. It is important to remember that by design, it's better to create an 'autotest' user that can run the scheduler and communicate with the machines through ssh. As root:

```
yum install screen
```

As autotest:

```
screen
/usr/local/autotest/scheduler/autotest-scheduler-watcher
```

You can even close the terminal window with screen running, it will keep the babysitter process alive. In order to troubleshoot problems, you can pick up the log file that autotest-scheduler-watcher prints and follow it with tail. This way you might know what happened with a particular scheduler instance.

Client Installation Steps

Clients are managed in the tab hosts of the web frontend. It is important that you can log onto your clients from your server using ssh *without* requiring a password.

[[remote-connection.png]]

Setup password-less ssh connection from the server to this host (client)

As autotest, on the server, create a RSA key in the following way:

```
ssh-keygen -t rsa
```

Then, still on the server, and as autotest, copy it to the host:

```
ssh-copy-id root@your.host.name
```

Import tests data into the database

You can import all the available tests inside the autotest client dir by running the test importer script as autotest:

```
/usr/local/autotest/utils/test_importer.py -A
```

If you did clone the autotest repo with `--recursive`, the virt test will be among the imported tests.

Troubleshooting your server

You can refer to the *Autotest Troubleshooting Documentation* documentation for some commonly reported problems and their root causes.

Virt Test specific configuration

Please refer to the shared *Autotest Virt Documentation*

See also

- *The Parser* is used to import results into TKO
- *The Web Frontend Docs* talks about using the frontend
- *The Web Frontend Development* talks about setting up for frontend development work - you do not want to develop through Apache!

Installing an Autotest server (Red Hat version)

Install script

We have developed a script to automate the steps described below on a (Fedora 16/17/RHEL6.2) server. So if you want to save yourself some time, please check the *Installing Server/Scheduler/WebUI notes*.

If you want to do it all yourself, we opted by keeping the documentation herem and we'll do the best to update it. However, we're always working on streamlining this process, so it might be possible that this can get out of sync.

If you find any step that might be outdated, please let us know, and we'll fix it.

Server/Scheduler/Web UI Installation Steps

Install required packages

We have automated this step on recent Fedora (17, 18) and RHEL 6, although it should work on Debian too:

```
sudo /usr/local/autotest/installation_support/autotest-install-packages-deps
```

If you want to install it manually here it goes. Keep in mind this can be outdated, if so we kindly ask your help with keeping it up to date.

Note: Currently autotest is compatible with Django 1.5, so if your distribution has anything lower or higher than this version, you will have problems and are advised to use a compatible version.

If the distro you are running has Django 1.5 packaged, you can install the django that your distro ships:

```
yum install Django
```

Otherwise, it's best to leave to `build_externals.py` the task of installing it. Other needed packages:

```
yum install git make wget python-devel unzip
yum install httpd mod_wsgi mysql-server MySQL-python gnuplot python-crypto python-
↳paramiko java-1.6.0-openjdk-devel python-httplib2
yum install numpy python-matplotlib libpng-devel freetype-devel python-imaging
```

And our `aexpect` package, that can be installed from our COPR repo. Instructions to add the repo can be found on:

<https://copr.fedoraproject.org/coprs/lmr/Autotest/>

With the repo enabled, you can go on to install:

```
yum install aexpect
```

Alternatively, you can simply install it from pip:

```
pip install aexpect
```

Important notes

Important: For this entire documentation, we will assume that you'll install autotest under `/usr/local/autotest`. If you use a different path, please change `/usr/local/autotest` accordingly. Please that you may have some issues with apache configuration if you don't choose `/usr/local/autotest`.

Important: We will also assume that you have created an autotest user on your box, that you'll use to perform most of the instructions after the point you have created it. Most of the instructions will use autotest unless otherwise noted.

Creating the autotest user

As root:

```
useradd autotest
passwd autotest [type in new password]
```

Cloning autotest

You can then clone the autotest repo (as root):

```
cd /usr/local
git clone --recursive git://github.com/autotest/autotest.git
chown -R autotest:autotest autotest
```

Log out, re-log as autotest, and then proceed.

Setup MySQL

Please check the shared *Configuring Autotest Server Database notes*

Install extra packages

Run the build script to install necessary external packages. If you ran the package install script, you should have all you could get from your system packages and it would download only GWT. As autotest:

```
/usr/local/autotest/utils/buildexternals.py
```

Always re-run this after a git pull if you notice it has changed, new dependencies may have been added. This is safe to rerun as many times as you want. It will only fetch and build what it doesn't already have. It's important to note that the autotest scheduler will also try to run `buildexternals.py` whenever it's executed in order to make sure every piece of software has the right versions.

Important: Set the `HTTP_PROXY` environment variable to `http://proxy:3128/` before running the above if your site requires a proxy to fetch urls.

Update Apache config

As root:

```
ln -s /usr/local/autotest/apache/conf /etc/httpd/autotest.d
ln -s /usr/local/autotest/apache/apache-conf /etc/httpd/conf.d/z_autotest.conf
ln -s /usr/local/autotest/apache/apache-web-conf /etc/httpd/conf.d/z_autotest-web.conf
```

Test your configuration (now with all autotest directives) by running (as root):

```
service httpd configtest
```

Now make sure apache will be started on the next boot. If you are running on a pre-systemd OS, such as RHEL6, you can enable do it this way:

```
chkconfig --level 2345 httpd on
```

On a systemd OS (Fedora >= 16), you could do it this way:

```
systemctl enable httpd.service
```

Update Autotest config files

Important: Edit the following files to match the database passwords you set earlier during session #Set_up_MySQL, as autotest, more specifically, `MYSQL_AUTOTEST_PASS`.

```
/usr/local/autotest/global_config.ini
/usr/local/autotest/shadow_config.ini
```

Important: Please, do *not* change this field

```
[AUTOTEST_WEB]
# Machine that hosts the database
host: localhost
```

As we are doing the setup on the same machine where mysql is running, so *please, pretty please* don't change it otherwise you will have trouble moving forward.

Things that you usually want to change on `global_config.ini`:

Section `AUTOTEST_WEB`

```
# DB password. You must set a different password than the default
password: please_set_this_password
```

Section SCHEDULER

```
# Where to send emails with scheduler failures to
# (usually an administrator of the autotest setup)
notify_email:
# Where the emails seem to come from (usually a noreply bogus address)
notify_email_from:
```

Section SERVER

```
# Use custom SMTP server
# If none provided, will try to use MTA installed on the box
smtp_server:
# Use custom SMTP server
# If none provided, will use the default SMTP port
smtp_port:
# Use custom SMTP user
# If none provided, no authentication will be used
smtp_user:
# Use SMTP password
# It only makes sense if SMTP user is set
smtp_password:
```

Run DB migrations to set up DB schemas and initial data

Important: If you set up your database using autotest-database-turnkey, this step can be safely skipped.

During the time span of the project, the autotest database went through design changes. In order to make it able for people running older versions to upgrade their databases, we have the concept of migration. Migration is nothing but starting from the initial database design until the latest one used by this specific version of the application. As autotest:

```
/usr/local/autotest/database/migrate.py --database=AUTOTEST_WEB sync
```

Run Django's syncdb

Important: If you set up your database using autotest-database-turnkey, this step can be safely skipped.

You have to run syncdb twice, due to peculiarities of the way syncdb works on Django. As autotest:

```
/usr/local/autotest/frontend/manage.py syncdb
/usr/local/autotest/frontend/manage.py syncdb
```

Compile the GWT web frontends

Compile the Autotest web application and TKO frontend. As autotest:

```
/usr/local/autotest/utils/compile_gwt_clients.py -a
```

You will need to re-compile after any changes/syncs of the frontend/client pages.

SELinux Issues

You may encounter issues with SELinux not allowing a section of the web UI to work when running in Enforcing Mode. In order to fix this, you can run the following commands to allow execution of the cgi scripts on your server.

As root:

```
semanage fcontext -a -t httpd_sys_script_exec_t '/usr/local/autotest/tko(/.*cgi)?'  
restorecon -Rvv /usr/local/autotest
```

Note: If you are having weird problems with installing autotest, you might want to turn off SELinux to see if the problem is related to it, and then think of a sensible solution to resolve it.

Restart Apache

As root:

```
/sbin/service httpd restart
```

Test the server frontend

You should be able to access the web frontend at <http://localhost/afe/>, or <http://your.server.fully.qualified.name.or.ip/afe/>

Start the scheduler

Executing using old SysV init scripts (RHEL6 and Fedora <= 14)

As root:

```
cp /usr/local/autotest/utils/autotest-rh.init /etc/init.d/autotestd  
chkconfig --add /etc/init.d/autotestd  
service autotestd start
```

Executing using systemd (Fedora >= 15)

Copy the service file to systemd service directory. As root or using sudo:

```
sudo cp /usr/local/autotest/utils/autotestd.service /etc/systemd/system/
```

Make systemd aware of it:

```
sudo systemctl daemon-reload
```

Start the service:

```
sudo systemctl start autotestd.service
```

Check its status:

```

autotest.service - Autotest scheduler
   Loaded: loaded (/etc/systemd/system/autotest.service)
   Active: active (running) since Wed, 25 May 2011 16:13:31 -0300; 57s ago
   Main PID: 1962 (autotest-schedu)
   CGroup: name=systemd:/system/autotest.service
            1962 /usr/bin/python -u /usr/local/autotest/scheduler/autotest-
↳ scheduler-watcher
            1963 /usr/bin/python -u /usr/local/autotest/scheduler/autotest-
↳ scheduler /usr/local/autotest/results

```

Executing manually using screen (not recommended)

You can execute the babysitter scripter through, let's say, nohup or screen. It is important to remember that by design, it's better to create an 'autotest' user that can run the scheduler and communicate with the machines through ssh. As root:

```
yum install screen
```

As autotest:

```

screen
/usr/local/autotest/scheduler/autotest-scheduler-watcher

```

You can even close the terminal window with screen running, it will keep the babysitter process alive. In order to troubleshoot problems, you can pick up the log file that autotest-scheduler-watcher prints and follow it with tail. This way you might know what happened with a particular scheduler instance.

Client Installation Steps

Clients are managed in the tab hosts of the web frontend. It is important that you can log onto your clients from your server using ssh *without* requiring a password.

Setup password-less ssh connection from the server to this host (client)

As autotest, on the server, create a RSA key in the following way:

```
ssh-keygen -t rsa
```

Then, still on the server, and as autotest, copy it to the host:

```
ssh-copy-id root@your.host.name
```

Import tests data into the database

You can import all the available tests inside the autotest client dir by running the test importer script as autotest:

```
/usr/local/autotest/utis/test_importer.py -A
```

If you did clone the autotest repo with `-recursive`, the virt test will be among the imported tests.

Troubleshooting your server

You can refer to the *Autotest Troubleshooting Documentation* <../sysadmin/AutotestServerTroubleshooting> documentation for some commonly reported problems and their root causes.

Virt Test specific configuration

Please refer to the shared *Autotest Virt Documentation* <../sysadmin/AutotestServerVirt>

See also

- *The Parser* <../scheduler/Parse> is used to import results into TKO
- *The Web Frontend Docs* <../sysadmin/WebFrontendHowTo> talks about using the frontend
- *The Web Frontend Development Docs* <../developer/WebFrontendDevelopment> talks about setting up for frontend development work - you do not want to develop through Apache!

Autotest Server Install - Set up MySQL

Let's say you have mysql installed and unconfigured, and that you have chosen a password, that we'll call MYSQL_ROOT_PASS and a password for the autotest user, that we'll call MYSQL_AUTOTEST_PASS. The autotest-server-install.sh script will set them to the same value, but if you are doing things manually, you are free to choose.

Make sure that mysql daemon is up and starts on each boot. As root:

```
/sbin/service mysqld restart
chkconfig mysqld on
```

The next step is automated through the script autotest-database-turnkey, so if you want to use it, the process should be as simple as:

```
/usr/local/autotest/installation_support/autotest-database-turnkey --check-
↪credentials --root-password MYSQL_ROOT_PASS -p MYSQL_AUTOTEST_PASS
```

If you want to do it manually, provide mysql server with password by running the following command (as autotest or root, you choose):

```
mysqladmin -u root password MYSQL_ROOT_PASS
```

Now, to get a mysql query prompt, type

```
mysql -u root -p
```

The following commands will set up mysql with a read-only user called nobody and a user with full permissions called autotest with a password MYSQL_AUTOTEST_PASS, and must be typed on mysql's query prompt:

```
create database autotest_web;
grant all privileges on autotest_web.* TO 'autotest'@'localhost' identified by 'MYSQL_
↪AUTOTEST_PASS';
grant SELECT on autotest_web.* TO 'nobody'@'%';
grant SELECT on autotest_web.* TO 'nobody'@'localhost';
create database tko;
grant all privileges on tko.* TO 'autotest'@'localhost' identified by 'MYSQL_AUTOTEST_
↪PASS';
```

```
grant SELECT on tko.* TO 'nobody'@'%';
grant SELECT on tko.* TO 'nobody'@'localhost';
```

If you use *safesync* for migrating the databases you will want to grant access to the test database. Note that this is entirely optional.

```
GRANT ALL ON test_autotest_web.* TO 'autotest'@'localhost' identified by 'MYSQL_
↳AUTOTEST_PASS';
```

If you want mysql available to hosts other than the localhost, you'll then want to comment out the `bind-address = 127.0.0.1` line in the `/etc/mysql/my.cnf`.

In addition, you may want to increase the `set-variable = max_connections` to something like 6000, if you're running on a substantial server. If you experience scalability issues, you may want to log slow queries for debugging purposes. This is done with the following lines:

```
log_slow_queries = /var/log/mysql/mysql-slow.log # Log location
long_query_time = 30 # Time in seconds before we consider it slow
```

Advanced setups may wish to use *MySQL Replication*

Autotest Server/Scheduler/WebUI Install script

We have developed a script to automate the install steps for the autotest server, scheduler and web UI on a (Fedora 16/17/RHEL6/Ubuntu) server. Debian should also work, but it was not tested.

The recommended installation procedure is:

1. Make sure you have a freshly installed system that we support (a VM, for example).
2. Pick this script straight from github

```
curl -OL https://raw.github.com/autotest/autotest/master/contrib/install-autotest-
↳server.sh
```

Debian/Ubuntu: don't forget to first install curl with `apt-get install curl`.

Then make it executable and execute it:

```
chmod +x install-autotest-server.sh
./install-autotest-server.sh
```

The command above will show you the script options. Usually you'll want to provide the options `-u` for the autotest user password, and `-d` for the autotest database password. The script is going to set all passwords, permissions and dependency installing, and it should log every step of the way, reporting a log file that you can look at.

```
# ./install-autotest-server.sh -u password -d password
15:59:21 INFO | Installing the Autotest server
15:59:21 INFO | A log of operation is kept in /tmp/install-autotest-server-07-23-2013-
↳15-59-21.log
15:59:21 INFO | Install started at: Tue Jul 23 15:59:21 BRT 2013
15:59:21 INFO | /usr/local free 37G
15:59:21 INFO | /var free 37G
15:59:21 INFO | Installing git packages
...
```

Hopefully at the end the script will report a URL that you can use to access your newly installed server. The script should also take care of importing existing control files, so they appear right away in the server.

Autotest Server Troubleshooting

Here we have some common problems in the server/scheduler/web UI and solutions for them. Also, we have info on log files you can look after.

Checking scheduler logs

You can find them in the autotest logs directory. As autotest or root:

```
tail -f /usr/local/autotest/logs/scheduler-[timestamp].log
```

Status is queing

The scheduler is not running. You are strongly advised to use the init scripts mentioned in the AutotestServerInstall or AutotestServerInstallRedHat documentation. If you are using them, restarting the scheduler should be simple:

```
service autotestd start
```

Status is pending

Usually it is a result of scheduler crash due to lack of disk space on Autotest server, so you might want to check that.

Setting up an Autotest Drone (Results Server)

After completing this document you should have at the very minimum two servers setup. The Autotest system you had setup initially and another system for storing the results of job runs. This document assumes that you have a working Autotest server as described in: Autotest Server Install.

Benefits of setting up a results server

- Offload all jobs to one central location that is only used for storing the results.
- Offload the main autotest server from having to also store results copied back to it.
- Off site copy of results.

The benefits of setting up a results server are most apparent when you have Autotest running jobs on multiple drones.

Global Configuration Variables

In the global_config.ini SCHEDULER section there are some variables you can use to tell Autotest where to archive results:

```
[SCHEDULER]
results_host: localhost
results_host_installation_directory:
```

- *results_host* defines the host where results should be offloaded. This is typically localhost and basically tells Autotest not to copy files anywhere else after a job completes.
- *results_host_installation_directory* is used to specify a custom directory if it is required. By default it uses whatever the Autotest server uses on the scheduler commandline. Most people will want to leave this at default.

Our drone system in general allows for more flexibility using “special variables” that do not exist in the default `global_config.ini` but can be used to change the behavior of the system. Below will be an example of using the `HOSTNAME_username` directive to make all results collection be done as a user I specify.

Updated [SCHEDULER] configuration

```
[SCHEDULER]
max_processes_per_drone: 1000
max_jobs_started_per_cycle: 100
max_parse_processes: 5
max_transfer_processes: 50
drones: localhost
drone_installation_directory: /usr/local/autotest
results_host: dumpster
results_host_installation_directory:
dumpster_username: offloader**
secs_to_wait_for_atomic_group_hosts: 600
reverify_period_minutes: 0
```

With the above settings, all jobs from all drones (including a regular localhost drone) will be copied to hostname *dumpster* using username *offloader*. The username setting is using the aforementioned special variable. If I did not use *dumpster_username* the results server would have data copied to it as the user the autoserv process is run under (Which in most cases would be autotest).

- Make sure you keep the `global_config.ini` files in sync

throughout your whole Autotest system otherwise you may experience very strange issues.

Software Required on the Results Server

A results server requires all the same software a Drone requires or a local Autotest server without MySQL. You will need a full Autotest installation on the system. If you are not doing anything special to synchronize all of your Autotest Server Systems then you can simply `rsync` your current server Autotest directory to your Results server.

Example Rsync command:

```
rsync -av /usr/local/autotest dudicus:/usr/local/autotest
```

How the two installations are kept in sync is the job of the system administrator we do not attempt to solve this problem.

Start/Restart the Scheduler

Once you have the following steps complete restart the scheduler and you will be running with a results server

- Your global configuration has been updated
- You’ve installed all required software on the results server
- An updated `global_config.ini` as described above is on all of your Autotest System Servers.

Restart your scheduler and run a few jobs to make sure files are showing up.

Results will show up in your autotest directory under results. For example `/usr/local/autotest/results/`

Tips and Tricks

- Often times corporate accounts are weighed down with other authentication methods like LDAP that can make transfers very slow. Try setting up a local account that uses your autotest users ssh key.
- SSH connections are dropped when a large job completes: Modify the following variable in your `/etc/ssh/config`: **MaxStartups XXXX**. This will allow half complete connections to wait around until your system is available to process all of the connections.

System Administration Tips and Tricks

This page is for random system administration tips that don't fit elsewhere. Over time as these gather we can organize them better.

Message of the Day

If you create a file `motd.txt` in the root Autotest directory, its contents will be displayed at the top-right corner of AFE and TKO. You can include HTML. AFE and TKO will refresh the MOTD automatically every so often.

Virt Test specific configuration

Important server configuration for virt-test

The way the virt control file is organized right now requires the user to change one value on `global_config.ini` file, that should be at the top of the autotest tree.

As autotest, please change the following configuration value from what's default to make it look like this:

```
[PACKAGES]
serve_packages_from_autoserv: False
```

By default, the above value is True. To make a long story short, changing this value will make autoserv to copy all tests to the server before trying to execute the control file, and this is necessary for the kvm control file to run. Also, we need the other tests present to run autotest tests inside guests.

Update virt test config files

Run `/usr/local/autotest/client/tests/virt/qemu/get_started.py` as autotest to be guided through the process of setting up the autotest config files. Edit the files to suit your testing needs.

The server is now ready to use. Please check out the following sections to learn how to configure remote hosts and execute the KVM test suite.

Analyze virt job execution results

The results interface provided by autotest allowing SQL query based filtering, usable display of logs and test attributes and graphing capabilities.

However, any autotest job also produces a detailed, formatted html report (**job_report.html**) per remote host tested in addition to standard autotest logs, where kvm-autotest results data is nicely organized. The html reports are stored in the job main results directory (accessible via *raw results logs* link).

Setting up a distributed Autotest production environment

This document aims to discuss how to setup a distributed autotest environment.

The problem

The standard Autotest production environment uses a single server to do many things:

- Run MySQL for the frontend and results databases
- Run Apache for the AFE and TKO web interfaces
- Run a scheduler to coordinate job executions
- Run many Autoserv processes to execute tests on remote machines
- Store all results in a single results repository directory

As the size of an Autotest server grows, and in particular as the number of concurrent machines under test grows, this single-server setup can run into scalability limitations quickly. In order to allow continued growth of an Autotest production environment, the Autotest system supports breaking out these roles onto different machines. Once properly configured, the difference should be nearly invisible to users.

MySQL and Apache

Autotest has always been capable of using a remote database server -the `global_config.ini` file contains parameters for database hostname. The web interfaces are almost exclusively dependent on the database, so they too are fairly simple to break out.

Scheduler, Autoserv and the Results Repository

The main complexity in a distributed setup arises in the scheduler. The scheduler is responsible for reading the database, executing Autoserv processes, and gathering the results into a central location. So the scheduler must be capable of executing Autoserv processes on remote machines and transferring the results files to a separate results repository machine. This behavior is achieved through the following `global_config` parameters:

- **“drones”**: a **“drone” is a machine that will be used to execute** Autoserv processes. This parameter should be a comma-separated list of hostnames for machines to be used as drones.
- **“results_host”**: the hostname of the machine to use as the results repository.

Any machine used as a drone or results repository must be set up for passwordless SSH from the scheduler, just as for machines under test. In addition, these hosts must have the results directory created with read/write permissions for the SSH user (the results directory is passed to the scheduler on the command line). They must also have Autotest installed at the location given in the `“drone_installation_directory”` `global_config` option. This may be the same as the results directory. Finally, since the parser will run on the drones, they must have TKO database parameters properly configured in `global_config.ini`.

Note that `“localhost”` is a valid hostname for either option, and when using localhost, SSH is not required to be set up. For a single-server setup, both options would simply be set to `“localhost”`.

See `GlobalConfig` for more options that can be used.

Viewing results files from the web

With the above setup, your jobs will execute successfully, but viewing results through the web remains a challenge because the logs may not reside on the same machine as Apache. For this reason, both AFE and TKO perform all log retrieval through the “tko/retrieve_logs.cgi” script. This script reads the global_config options above, as well as a third:

- “**archive_host**” (optional): an additional hostname to check for results files when they cannot be found elsewhere. System administrators may manually move results off of the main results repository to this machine.

“retrieve_logs.cgi” attempts to fetch the requested log file from the results repository, then from each drone, and finally from the archive host, until it succeeds. If it succeeds, it redirects the user to the appropriate host. For this to work properly, all drones, the results repository host, and the optional archive host must **all** be running Apache with the results directory mapped to “/results”.

Recommendations

So now you know how to configure a distributed Autotest environment. But how do you figure out what distribution of components is necessary? Here are a few general tips:

- The most important thing to do is to run the Autoserv processes on a different machine than MySQL. These components are usually the two biggest resource hogs. Each Autoserv process should not be too resource-intensive, but since there will be at least one process per host under test, there can be a huge number of Autoserv processes running concurrently.
- Since the web interfaces and the scheduler depend heavily on the database, it can be beneficial to run Apache and the Scheduler on the same machine as MySQL. Since Apache and the Scheduler are not very resource intensive, this is generally not a performance problem.
- The drones will often end up being the bottleneck in a large system, and the Autoserv processes will most likely be IO-bound. Therefore, configuring drones with performance-enhancing RAID setups can provide a dramatic increase in system capacity.
- For system reliability, it is often beneficial to isolate drones for running Autoserv processes only. Large numbers of Autoserv processes are the most likely components to crash the system. With dedicated drones, a machine crash due to Autoserv will not affect the web interfaces, and if multiple drones are being used, jobs can continue to run uninterrupted on other drones.

Using the autotest package management with autoserv

This document will go over how to setup your Global Configuration to use your Autotest server as a packaging repository. After that there will be a section going over how to add another separate machine as a remote repository for packages.

By setting up packaging in Autotest you can reduce the amount of files transferred to clients running jobs which generally decreases the amount of setup time Autotest has to do for clients.

Setting up your Autotest server as a packaging repository

This section assumes you already have AFE and TKO running properly as outlined in the Autotest Server Install documentation, if this isn't the case it is left up to the reader to ensure Apache is running and able to serve files out of the directory they reference in the fetch_locations below.

In order for packaging to work we need to add the following section to our global config.

```
[PACKAGES]
fetch_location: http://your_autotest_server/packages/builtin, http://your_autotest_
↪server/packages/custom
upload_location: /usr/local/autotest/packages/builtin
custom_upload_location: /usr/local/autotest/packages/custom/
custom_download_location: /usr/local/autotest/custom_packages
```

Explanation:

fetch_location: is what the client uses when downloading tests. The order that these are listed are the order they are used by the Autotest client. We have an entry for both custom and builtin tests since Autotest doesn't directly discern between custom packages and builtin packages. We keep them separate so we have to list both locations. It is up to you to keep these separate but we prefer to do this for clarity.

upload_location: /usr/local/autotest/utls/packager.py uses this location to determine where it needs to upload files. For example when you run packager.py upload -all all tests profilers and dependencies in your tree will be archived and copied either via scp or cp (depending on if it is local or remote)

custom_upload_location: This is for custom tests and kernels uploaded through the frontened or via the command line.

custom_download_location: This is the location where Autotest puts packages users upload through the frontend before it is uploaded to your http repository.

Adding a SSH/HTTP Repository

For a remote repository we use SSH and HTTP. SSH For transferring files to the machine and http to serve the tests to the clients running jobs. We chose HTTP for lower overhead transfers (for files that are extremely large).

This step assumes the user is familiar with setting up Apache (At the very least barebones to serve files) and keyless SSH.

Requirements:

- Passwordless SSH for the user defined http repo below
- Apache setup to serve files out of the directory specified below (In this case /var/www/packages/builtin)

Using the above PACKAGES section we add in three new pieces of information

- fetch_location: http://your_http_repo_hostname/packages/builtin, http://your_http_repo_hostname/packages/custom
- upload_location: ssh://root@your_http_repo_hostname/var/www/packages/builtin
- custom_upload_location: ssh://root@your_http_repo_hostname/var/www/packages/custom

```
[PACKAGES]
fetch_location: http://your_http_repo_hostname/packages/builtin, http://your_http_
↪repo_hostname/packages/custom, http://your_autotest_server/packages/builtin, http://
↪your_autotest_server/packages/custom
upload_location: /usr/local/autotest/packages/builtin, ssh://root@your_http_repo_
↪hostname/var/www/packages/builtin
custom_upload_location: /usr/local/autotest/packages/custom/, ssh://root@your_http_
↪repo_hostname/var/www/packages/custom
custom_download_location: /usr/local/autotest/custom_packages
```

Scheduler

Scheduler specification

Basic flow

Results files

- The scheduler always creates a “job directory”, results/<job tag>
- For asynchronous jobs, the scheduler creates a results/<job tag>/<hostname> directory for each host and runs one instance of autoserv for each host with these per-host directories as results directories.
- For synchronous jobs, the scheduler creates a results/<job tag>/groupN directory for each group of hosts formed, as defined by the job’s sync_count. N is a numeric index starting at zero. The scheduler runs an instance of autoserv for each group of machines with these per-group directories as results directories.

Metahosts always get queue.log.<id> files created in the job directory (results/<job tag>). These logs contain a single line for each time a metahost is assigned a new host or cleared of its host. Each line includes a timestamp.

Verify/repair/cleanup information is handled like so:

- During execution of verify/repair/cleanup, Autoserv output is directed to a temporary file under the results/drone_tmp directory.
- When Autoserv completes, this file is copied to the host logs directory under results/hosts/<hostname>.
- If the task fails and causes job failure, the log is also copied to the execution results directory (results/<job tag>/<hostname or groupN>). This happens if:
 - The task was a pre-job cleanup or verify
 - The task failed
 - The correspond queue entry was scheduled for a particular host, not a metahost (for metahosts, the queue entry would simply choose a new host, so it wouldn’t make sense to include the verify failure as part of the job).

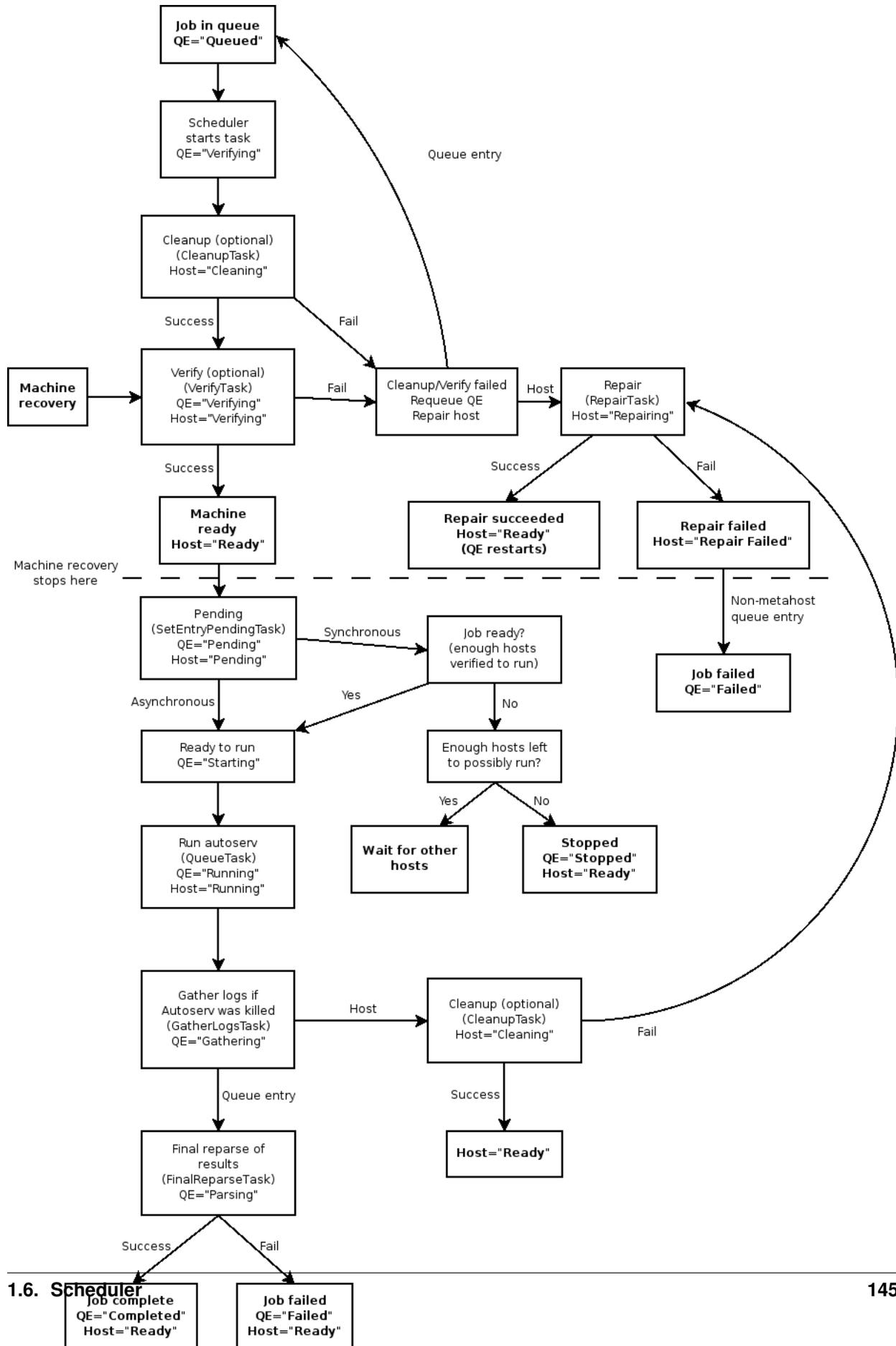
If the subsequent repair succeeds, the log file is removed and the job is restarted.

The scheduler only creates a .machines file for asynchronous multi-machine jobs. It creates this file on the fly by appending each hostname to this file immediately before running the main autoserv process on that host. For synchronous jobs, autoserv creates the .machines file itself.

Distributed implementation

In order to support distributed setups (see DistributedServerSetup), the scheduler performs much of its work through the drone_manager module. A “drone” is a machine on which Autoserv is executed, which is not necessarily the machine on which the server is running. Here is a guide to this implementation:

- Overview
 - All OS-dependent calls in the scheduler have been extracted into an interface on the drone_manager.DroneManager? class. This includes filesystem access and process execution, killing and monitoring.
 - DroneManager? methods queue up actions to perform on drones.
 - The scheduler calls DroneManager?.refresh() at the beginning of each tick, which connects to each drone and gathers information on running processes.



1.6. Scheduler

- The scheduler calls `DroneManager?.execute_actions()` at the end of each tick, which connects to each drone and executes all queued actions.
- `DroneUtility?`
 - The `drone_utility.DroneUtility?` class contains implementations of all the OS-dependent actions.
 - The `drone_utility.MethodCall?` class abstracts a call to a method on `DroneUtility?`.
 - `DroneUtility?.execute_calls()` accepts a list of `MethodCall?` objects and returns a list of results, along with any warnings that were generated.
 - The `drone_utility` module is executable as a script. It accepts a filename on the command line and reads a list of `MethodCall?` objects from that file in pickled format. This implements a simple batched RPC mechanism for `DroneUtility?`.
- Drone objects
 - The `drones` module provides implementations of the `drones._AbstractDrone` interface. `AbstractDrone?` allows the client to queue up method calls to a `DroneUtility?` instance and execute them on the drone machine. There are two implementations:
 - * a `_LocalDrone` class which simply imports `drone_utility` and calls methods directly, and
 - * a `_RemoteDrone` class which executes `drone_utility` on a remote host using the `server.hosts.ssh_host.SSHHost` class. It pickles the call list into a file, sends the file to the remote host, and executes `drone_utility` remotely on that file.
 - The `drones.get_drone(hostname)` factory method is used to retrieve a drone object.
- `DroneManager?`
 - `DroneManager?` maintains a list of drone objects, one for each drone as well as one for the results repository host. Methods on `DroneManager?` are implemented by queuing up method calls on the appropriate drone objects. `DroneManager?.execute_actions()` then executes all queued calls for each drone in turn.
 - `DroneManager?` also contains limited handling for dead drones.

See Also

- `SchedulerAutoservInteractions`

Job and Host Statuses

Job Statuses

- **Queued** – the job is waiting for machines to become ready and/or accessible, or the scheduler has simply not picked up the job yet. A job can go back to this state from **Verifying** when a machine fails verify and goes to repair.
- **Verifying** – the job is going through pre-job cleanup and/or verification. See host statuses **Cleaning** and **Verifying**. This is controlled by the job options *reboot before* and *skip verify* and well as by *Host Protections* (namely *Do not verify*).
- **Pending** – the job is ready to run on this host but is waiting for other hosts because it's a synchronous job.
- **Starting** – the job is about to start. Jobs should only stay in this state when the system is at its capacity limit.
- **Running** – the job is running (Autoserv is actively running on the server).

- **Gathering** – after Autoserv is aborted (or otherwise unexpectedly killed), a job will enter this state to gather uncollected logs and crash information from the machine under test. This stage will also wait several hours for the machine to come back if it went down.
- **Parsing** – the parser is running a final reparse of job results. This stage should be very brief unless the system is under heavy load, in which case parses are throttled by the results database.
- **Completed** – the job is over and Autoserv completed successfully (note that *functional tests* may have failed, but the *job* ran all tests without error).
- **Failed** – the job is over and Autoserv exited with some failure.
- **Aborted** – the job is over and was aborted.

Host Statuses

- **Ready** – the host is idle and ready to run.
- **Cleaning** – the host is running pre-job, post-job, or post-abort cleanup (see job options *reboot before* and *reboot after*). The cleanup phase includes rebooting the host and, optionally, site-specific cleanup tasks.
- **Verifying** – the host is running pre-job or post-abort verify (see job option *skip verify*). The verify phase checks for basic connectivity, disk space requirements, and, optionally, site-specific conditions.
- **Repairing** – the host is undergoing attempted repair; this includes rebooting, waiting for the host to come up, clearing off disks, and, optionally, site-specific extensions. This is controlled by *Host Protections*.
- **Pending** – see the job state **Pending**.
- **Running** – the host is being held for a running job. This includes time that Autoserv is actually running (job state **Running**) as well as the job **Gathering** phase.
- **Repair Failed** – the host failed repair and it currently assumed to be in an unusable state. Scheduling a new job against this host will reset it to the **Ready** state.

See also

- The flowchart at *SchedulerSpecification* illustrates how hosts and jobs move through these states.
- *Web Frontend Howto* documents the above-mentioned job options.

Advanced Job Scheduling

This page documents some of the more advanced things that the scheduler is capable of.

Metahost entries (“Run on any...”)

Jobs can be scheduled to run against any host with a particular label. This is used through the frontend with the “Run on any...” box (for example, “run on any x86”). Such entries are called *metahost* entries. Metahost entries will be assigned to eligible and ready hosts dynamically by the scheduler.

“Only if needed” labels

If a label is marked *only if needed*, any host with that label is not eligible for assignment to metahost entries unless

- the job’s *dependency labels* includes that label, or

- the metahost is against that particular label.

Note that such hosts can still be used by any job if selected explicitly (i.e. not through a metahost).

Atomic Groups

An *atomic group* is a group of machines that must be scheduled together for a job. Regular jobs cannot be scheduled against hosts within these groups; they must be used together.

Atomic groups are created in the admin interface to specify classes of atomic groups of machines (for example, “x86-64 rack” might be an atomic group). Labels can then be marked as instances of a particular atomic group; in this case, a label would include all machines for a particular group (for instance, “x86-64 rack #1”). Finally, machines can be added to these labels to form the actual groups.

Example

As an example, assume you have twenty hosts, ten x86-64 and ten i386. You wish to run a test that requires a rack of five machines. You might do the following:

- Create two atomic groups, “x86-64 rack” and “i386 rack”.
- Create four labels: “x86-64 rack #1” and “x86-64 rack #2” are both labels with atomic group type “x86-64 rack”, and likewise for i386.
- Assign five x86-64 hosts to the “x86-64 rack #1” label, and the remaining five to the “x86-64 rack #2” label. Likewise for i386.

Now, you could run a job with `synch count = 5`, and specify that you want to run against one atomic group of type “x86-64 rack” and one of type “i386 rack”. The scheduler will dynamically pick a rack of each type that is ready to run the job. Users trying to schedule regular jobs against hosts within these groups will be unable to do so; they will remain reserved for jobs intended for the entire group.

Variable host counts

Some tests can run against a variable number of machines, and you may wish to run such a test against all the ready machines within an atomic group, within some bounds. The scheduler can do this for you – at job run time, it will verify all machines in the group and use all the ones that are ready. The following constraints are available:

- The “max number of machines” attribute on the Atomic Group specifies the maximum number of machines to use at once.
- The job’s “synch count” attribute specifies the minimum number of machines to use from the group. If fewer than this number are ready, the job will be unable to run. Note that this behavior is unique to jobs run against atomic groups – normally, synch count specifies the exact number of machines to use, but with an atomic group, the scheduler will use as many machines as are ready (up to the maximum).

Autotest Scheduler Roadmap

For the most part, the scheduler is now stable and its feature set is satisfactory. There are a few features we’ll be adding soon:

- Maximum running job count - **done**
- Job timeouts **done**
- User-friendly status messages **done**

- Better automated repairs
- Multiple scheduler support (distributed execution) **done**

General Overview

The purpose of the parser is to take one or more directories of test results and convert them into summary test results in the TKO database to be available for more generic queries. The parser is primarily only concerned with the status log for a test, since this is where the summary of job and test passes (or failures) is stored, however it also makes use of other result data (e.g. keyval entries) to help annotate the test results with relevant information such as the kernel version used for each test.

The parser is usable as a standalone script so that it can be run by hand on complete results, however it is also importable and usable as an in-process python library to allow for continuous parsing of partial results without having to continually launch new processes and perform a full re-parse every time new data is generated.

Versioning

We need to always be able to parse existing log data, while at the same time providing for the capability in the future to change the logging format to provide new capabilities and data. These types of changes will generally require parser changes, and although in the ideal case we could extend the parser in such a way that it can still parse both new and old data, this may not always be possible (or may significantly increase the difficulty of making the required changes). The implementation of this specification is an example of this.

The version of the status log format should be written out by autoserv (or whatever application is being used to generate job results) into the job keyval files as the variable `status_version`. If the keyval is unspecified then this implies version 0, the pre-specification version of the parser, while the parser specified by this document is version 1. Once the version is determined the results data should then be fed into the appropriate parser library and pushed into the database.

In the long term, it may also be desirable to specify some form of intermediate output that the parsers will produce to help isolate them from changes in the backing database; the current approach of writing out data manually will still make it difficult to change the schema as every parser version would have to be changed, not just the “current” version. However, at this time the only two versions in existence will be writing data out to the same schema so putting in the development time to build an intermediate output format would provide no immediate benefits.

Work Required to Implement this Specification

This specification represents a description of how the parser ideally **SHOULD** work, rather than a description of how it currently does work. However, this specification can be implemented incrementally, requiring the following work:

1. Change Autotest to properly write out full kernel information during the `reboot.verify`. The current code does output a kernel version, but this does not handle cases where you are building kernels with custom patches.
2. Change Autotest to write the `status_version` entry out to the results keyval files.
3. Build a parser class that uses the library approach described in this specification (a stateful parser object, separate out the reading of files from the parser itself, allow it to be used in standalone and in-process manner) but based on the existing parsing algorithm rather than the new one proposed by this specification.
4. Change autoserv to perform continuous parsing using the library version of the parser.
5. Implement a new parser class that uses the algorithm described in this specification.
6. Change the parser to auto-select either the new parser or the legacy parser based on the value of `status_version` (0=legacy, 1=new) in the results.

Once these steps are complete, a next possible step might be to move the actual parsing of data (or at least the writing of parser data into the database) back *out* of autoserv and into a separate process; however, this separate process would be a single daemon shared between all instances of autoserv on a machine, instead of the current model where a parser process is launched every time the results are parsed. This would avoid the current problem where a large number of database connections are consumed by the parsing tasks.

Library Design

The base of the parser will be a stateful object designed for parsing the results of a single job (i.e. a single-machine job, or one machine of a multi-machine asynchronous job). It will in no way be responsible for accessing the results directory; this will be the job of the calling code. This should make it easier to embed the parser into autoserv itself. It should also isolate the parser from the details of how watching for new data is being performed.

Given the results directory of a completed test, the parser can find all of the information it needs in the following places:

- status.log - the actual status logs come from here, this is the core of what the parser needs
- keyval - most of the job data comes from here, specifically:
 - username - the user who ran the job
 - label - the label of the job
 - machine - the hostname that this specific job was run on
 - job_queued, job_started, job_finished - timestamps from when the job was queued, started and finished
 - owner - the owner of the test machine
- <subdir>/keyval - some additional test meta-data comes from here, namely:
 - version - the version number of the test
- <subdir>/results/keyval - optional test data regarding iterations comes from here

When being used as a standalone process the parser will need to be able to access this data and so provides functions for retrieving it. It also provides a main() function that allows you to run the parser:

- on a single machine results directory (i.e. a single-machine job, or a single machine of a multi-machine job)
- on a multi-machine results directory
- on a top-level results directory, parsing all the results of an entire results repository

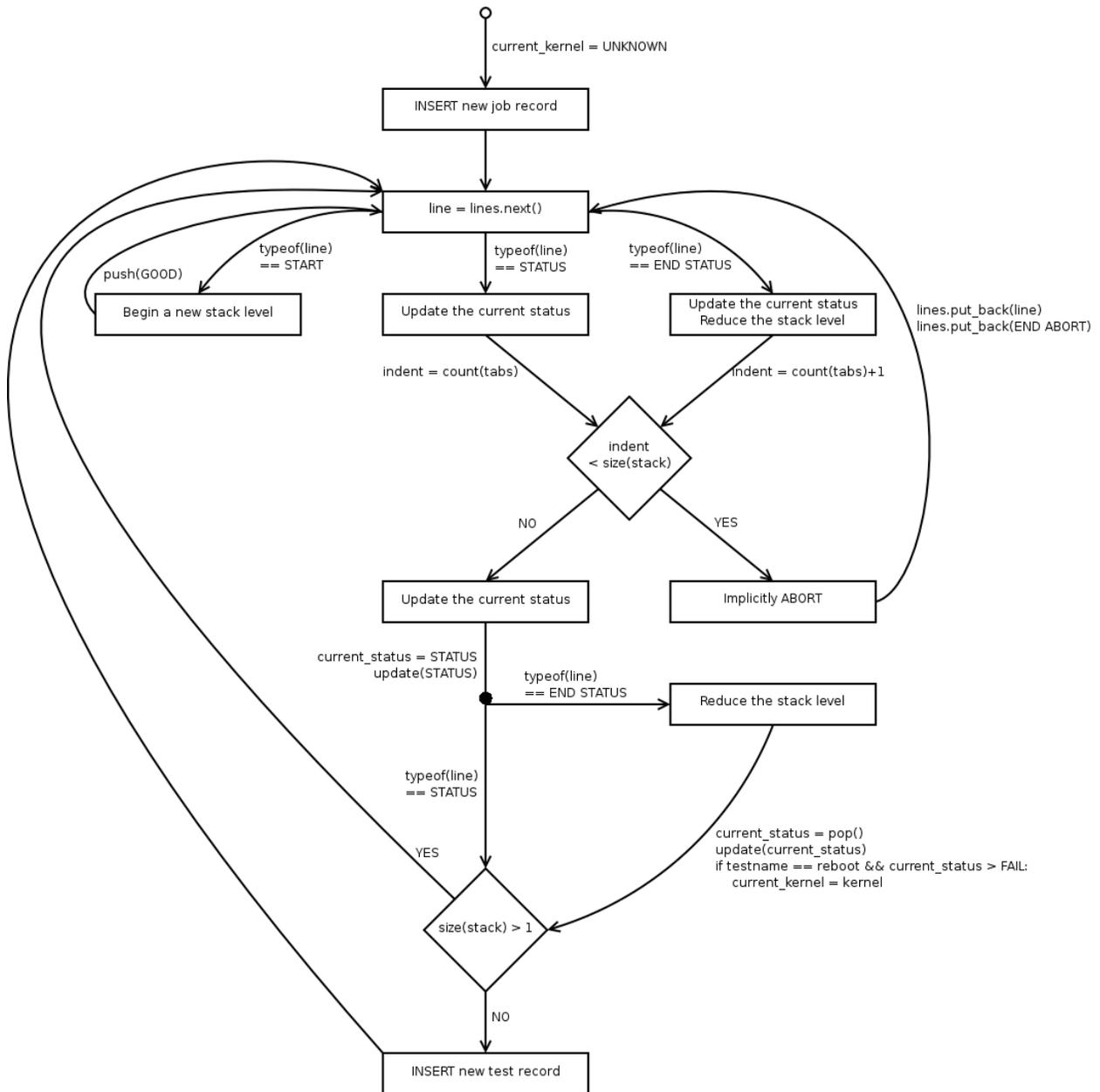
Parser Algorithm

The general algorithm of the parser is most easily summarized by the following diagram:

For tracking the “current” status, the parser has to use a stack of statuses. Manipulations of this stack are included in some of the transitions in the diagram, with the following operations:

- push(status) pushes status onto the stack
- pop() pops the top status off of the stack
- update(status) replaces the top of the stack with status if and only if status is “worse” than whatever is already on top

The update operation uses the concept of some statuses being “worse” than others. The idea behind this is that if a bunch of tests are being run as part of a single, cohesive group (or a single test produces multiple status lines of output) then the results should be combined in such a way that negative results poison the results of the entire set. So if some



results in the group are GOOD and some are FAIL, then the entire group should be considered a FAIL. The expected set of statuses is, from “best” to “worst”:

- GOOD - the operation was successful
- WARN - something suspicious has happened, but not a clear failure
- FAIL - the test has failed
- ABORT - something catastrophic has happened, and the entire job is terminating

Conceptually, the parser operates on a stream of lines. In a standalone parser process where it just performs a full re-parse and then exits the parser will simply operate on the results of `file.readlines` in a single shot. However, it should be just as easily usable in an in-process, continuous parsing fashion where it is fed status lines as they are generated and maintains its state until the application (e.g. `autoserv`) indicates that the job is finished and there are no more results.

Database Handling

There already exists code in `tko/db.py` for performing database lookups, inserts and deletes on the relevant objects as well as for looking up the appropriate authentication information in the Autotest configuration, so the parser will simply make use of this. The insertion of parsed results will **not** be performed in a transactional fashion in order to facilitate continuous parsing. The expected data flow is simply:

1. Delete any existing results job & test data.
2. Insert job entry.
3. Insert test entries as tests complete in the status log.

If a transaction mechanism needs to be implemented on top of this then that should be straightforward to do manually.

TKO parse documentation

```
usage: parse [options]

options:
  -h, --help  show this help message and exit
  -m          Send mail for FAILED tests
  -r          Reparse the results of a job
  -o          one: parse a single results directory
  -l LEVEL   levels of subdirectories to include in job name
```

Typical usages:

To populate the database with ALL results.

```
tko/parse $AUTODIR/results
```

To recreate the database (from some corruption, etc). First drop all the tables, and recreate them. Then run:

```
tko/parse -r $AUTODIR/results
```

To reparse a single job’s results

```
tko/parse -r -o $AUTODIR/results/666-me
```

To reparse a single machine’s results within a job:

```
tko/parse -r -l 2 -o $AUTODIR/results/666-me/machine1
```

The `-l2` here makes it create the job as “666-me/machine1” instead of “machine1”, which is normally what we want. It just says “take the last 2 elements of the path, not the last one”.

Developer

Downloading the Source

The main source is maintained on git and may be cloned as below:

```
git clone git://github.com/autotest/autotest.git
```

If you want to learn how to use git as an effective contribution tool, consider reading *GitWorkflow*.

Autotest’s Directory Structure

- **client**: The autotest client. When using the autotest server, the entire client dir is deployed to the machine under test.
 - **shared**: All the files common to both autotest server and the client are in this directory. It needs to be here, rather than in the top level, because only `/client` is copied to machines under tests. If you add new modules to the shared library. Your library will then be importable as `autotest.client.shared.mylibname`.
 - **bin**: The autotest core python files are all here. Also, any libraries not shared with the server are here.
 - **tools**: All executables besides autotest itself are here. This includes helpers like `boottool`.
 - **tests**: All the tests go here. Each test should be in a directory, which we’ll call `test_name`. There should also be a `test_name.py` file in that directory, which is the actual test. In addition, a file named `control` should also be in that directory to run the test with default paramaters. All other files the test depends on (and optionally other control files) should be in this directory as well.
 - **site_tests**: Same as above but for Internal client side tests.
 - **profilers**: Profilers are here. Profilers run during tests and are not tied to any one test.
- **conmux**: This has `conmux`, which is a console multiplexer. This allows multiple people to share serial concentrators and power strips. Several different types of concentrators and strips are supported, and new ones can be added by writing simple expect scripts.
- **Documentation**: This wiki is generally more up to date, but there are some old diagrams here.
- **mirror**: This is used for mirroring kernels from `kernel.org`.
- **queue**: This is an empty directory used for the file-system based queueing system.
- **results**: This is an empty directory where results can sit.
- **scheduler**: The scheduler lives here. The scheduler spawns `autoserv` instances to test new kernels.
- **server**: The autotest server (sometimes called `autoserv`). Unlike the client, all the python files are just in the root dir. (Should we move them?)
 - **doc**: Some documentation files. Unfortunately, these are largely out of date. The wiki’s your best bet for documentation.

- **hosts**: This contains all the host classes. SSHHost is what most users will be using.
- **tests** and **site_tests**: These are the same as in the client.
- **tko**: This is the web-based reporting backend for test.kernel.org
- **ui**: A script for generating control files.

Where should I put the files I'm adding?

Is this a generic module that will be useful on on both the client and the server? Then put it in client/shared. Or, if this module is providing site-specific functions for use on your local server, add the name to the libraries variable in client/shared/site_libraries.

Are you adding code to the client? Then put it in client. Remember that this code will only be accessible from other client code (and client-side tests), not from server code. Even though the server has a copy of the client, it generally avoids reaching into the client to import code (except for a few special cases). If you want to use your client code from the server as well then put it in the shared library, not on the client.

Are you adding code on the server? If it's a new kind of host, add it in server/hosts. Be sure to add an import for you new kind of host to server/hosts/___init.py___, since the server code will import host classes by pulling in the whole host package, rather than importing classes from specific submodules.

Are you adding tests? Public client side tests should be added in client/tests/<name>. Private client side tests go in client/site_tests/<name>. Server-side tests should go into server/tests/<name> and again for private server side tests server/site_tests/<name>.

Autotest Code Submission Check List

This document describes to contributors what we are looking for when we go through submitted patches. Please try to follow this as much as possible to save both the person reviewing your code as well as yourself some extra time.

Github Pull Requests

In order to keep code review in one place, making the work of our maintainers easier, we decided to make pull requests the primary means to contributing to all projects inside the autotest umbrella.

That means it is highly preferable to send pull requests, rather than patches to the mailing list. If you feel strongly against using pull requests, we'll take your patches, but please consider using the recommended method, as it is considered nicer with the maintainers.

This [documentation on github pull requests](#) is complete and up to date, it'll work you through all details necessary. The bottom line is that you'll fork virt-test or autotest_remote_unittest, create a working branch, push changes to this branch and then go to the web interface to create the request.

Subscribe to the mailing list

That's important. See the link in [the contact info documentation](#). Even though we don't use the mailing list for patch review, we still discuss RFCs and send announcements to it.

Running Unit tests

Regardless of what you change it is recommended that you not only add unittests but also run the unittest suite of each project to be sure any changes you made did not break anything. In order to install all the deps required for unittests, please check *the unittest suite docs*.

Example (autotest):

```
[foo@bar autotest]$ utils/unittest_suite.py --full
Number of test modules found: 65
[... lots of output ...]
All passed!
```

Example (virt-test):

```
[foo@bar virt-test]$ tools/run_unittests.py --full
Number of test modules found: 22
[... lots of output ...]
All passed!
```

Running pylint

Another tool we use to insure the correctness of our code is pylint. Due to the way imports have been implemented in the autotest code base a special wrapper is required to run pylint.

The file is located in `utils/run_pylint.py`. The `virt-test` version is in `tools/run_pylint.py`.

Simply run the command from your code directory and the rest is taken care of.

Example of running on a source file with warnings:

```
[lmr@freedom autotest]$ utils/run_pylint.py -q client/job.py
```

Good. Same process, now with an error I introduced:

```
[lmr@freedom autotest]$ utils/run_pylint.py -q client/job.py
***** Module client.job
E0602:175,14:base_client_job._pre_record_init: Undefined variable 'bar'
```

Here is the error, an undefined variable:

```
[lmr@freedom autotest]$ git diff
diff --git a/client/job.py b/client/job.py
index c5e362b..8d335b4 100644
--- a/client/job.py
+++ b/client/job.py
@@ -172,6 +172,7 @@ class base_client_job(base_job.base_job):
     As of now self.record() needs self.resultdir, self._group_level,
     self.harness and of course self._logger.
     """
+
+     foo = bar
     if not options.cont:
         self._cleanup_debugdir_files()
         self._cleanup_results_dir()
```

So, pylint is a valuable ally here, use it!

Running reindent.py

Yet another tool that we use to fix indentation inconsistencies (important thing to notice when you're doing python code) is `utils/reindent.py` (autotest) or `tools/reindent.py` (virt-test). You can use the script giving your files as an argument, so it will prune trailing whitespaces from lines and fix incorrect indentation.

Breaking up changes

- Submit a separate patch for each logical change (if your description includes “add this, fix that, remove three other unrelated things”; probably bad).
- Put a summary line at the very top of the commit message, explaining briefly what has changed and where.
- Put cleanups in separate patches than functional changes.
- *Please* set up your git environment properly, and always sign your patches using `commit -s`.

Patch Descriptions

Patch descriptions need to be as verbose as possible. Some of these points are obvious but still worth mentioning. Describe:

- The motivation for the change - what problem are you trying to fix?
- High level description / design approach of how your change works (for non-trivial changes)
- Implementation details
- Testing results

Follow control file specification

All tests must follow the control file specification Refer to the *Control Requirements section*. In virt-test, you don't usually need to write control files, so feel free to skip this if you're developing virt-test.

Follow Coding Style

Autotest and virt-test (mostly) follow PEP8, but it's always good to take a look at [the coding style documentation](#).

Git Setup

Please make sure you have git properly setup. We have a fairly brief and descriptive document explaining how to get the basics [setup here](#). In particular, tend to stick to one version of your written name, so all your contributions appear under a same name on git shortlog. For example:

John Doe Silverman

or

John D. Silverman

Please *do choose* between one of them when sending patches, for consistency.

Example Patch

This is a good example of a patch with a descriptive commit message.

```
commit 37fe66bb2f6d0b489d70426ed4a78953083c7e46
Author: Nishanth Aravamudan <nacc@linux.vnet.ibm.com>
Date: Thu Apr 26 03:38:44 2012 +0000

    conmux/ivm: use immediate reboot rather than delayed

    Delayed reboots use EPOW, which does not always result in a shutdown of
    the LPAR. Use the more sever immediate shutdown, to ensure the LPAR goes
    down. This matches the HMC code.

    Signed-off-by: Nishanth Aravamudan <nacc@us.ibm.com>
```

How to use git to contribute patches to autotest

Git is a powerful revision control system designed to make contributing to open source projects simple. Here's how you can contribute to autotest easily using git:

1) Make sure you have configured git to automatically create your signature on the commits you make inside your local tree. The following is an example script to do it, just edit replacing your name, email and choosing all aliases you want. Needless to say that once you run it, the configs are persistent (written to the git config files), so you only need to do this once.

```
#!/bin/bash
# personalize these with your own name and email address
git config --global user.name "John Doe"
git config --global user.email "john.doe@foo.com"

# colorize output (Git 1.5.5 or later)
git config --global color.ui auto

# colorize output (prior to Git 1.5.5)
git config --global color.status auto
git config --global color.diff auto
git config --global color.branch auto

# and from 1.5.4 onwards, this will work:
git config --global color.interactive auto

# user-friendly paging of some commands which don't use the pager by default
# (other commands like "git log" already does)
# to override pass --no-pager or GIT_PAGER=cat
git config --global pager.status true
git config --global pager.show-branch true

# shortcut aliases
git config --global alias.st status
git config --global alias.ci commit
git config --global alias.co checkout

# this so I can submit patches using git send-email
git config --global sendemail.smtpserver [your-smtp-server]
git config --global sendemail.aliasesfile ~/.gitaliases
git config --global sendemail.aliasfiletype mailrc
```

```
# shortcut aliases for submitting patches for Git itself
# refer to the "See also" section below for additional information
echo "alias autotest autotest-kernel@redhat.com" >> ~/.gitaliases

# another feature that will be available in 1.5.4 onwards
# this is useful when you use topic branches for grouping together logically related
↪changes
git config --global format.numbered auto

# turn on new 1.5 features which break backwards compatibility
git config --global core.legacyheaders false
git config --global repack.usedeltabaseoffset true
```

2. git clone the autotest git mirror repo:

```
git clone git://github.com/autotest/autotest.git
cd autotest
```

3. create a branch for the change you're going to make

```
git branch [branch-name]
git checkout [branch-name]
```

4) Make your changes in the code. For every change, you can make a git commit. For folks used to other paradigms of version control, don't worry too much, just have in mind that git trees usually are independent, and you can commit changes on your local tree. Those commits can then be generated in the form of patches, that can be conveniently sent to the maintainers of the upstream project. To commit you use:

```
git commit -as
```

If you have executed the git configuration, you'll see that there is already a Signed-off-by: with your name and e-mail, sweet, isn't it? Save and there you have your commit.

5) A alternative configuration is helpful for some guys who are using thunderbird, Zimbra or something like that to filter mail subject contains "[Autotest]" patches:

```
git config format.subjectprefix Autotest][PATCH
```

And then if you run 'git format-patch' later, you will get a patch with "[Autotest][PATCH]" mail's subject prefix.

6. When you want to generate the patches, it's as easy as doing a:

```
git format-patch master
```

It will generate all the differences between your branch and the master branch. You can also generate a certain number of patches arbitrarily from any branch. Let's say you want to pick the last 2 commits you made and create patch files out of it:

```
git format-patch -2 --cover-letter
```

This will generate 2 patches that also happen to be in a unix mailbox format that can be sent to the mailing list using git send-email ;)

7) Edit your cover letter (patch number 0 generated) with the info you'd like to include in the patchset.

8. Then you can send the patches with git send-email:

```
git send-email patch1.patch patch2.patch... patchN.patch --to address@foo.org --cc_
↳address@bar.org
```

Note that the aliases you defined on your configuration will allow you do do stuff like this:

```
git send-email patch1.patch --to autotest
```

So that autotest is expanded to the actual mailing list address.

Life cycle of an idea in autotest

If you are wondering how to propose an idea and work through its completion (feature making its way to a stable release), here is a small schema of how ideas transition to working code in the autotest developer community:

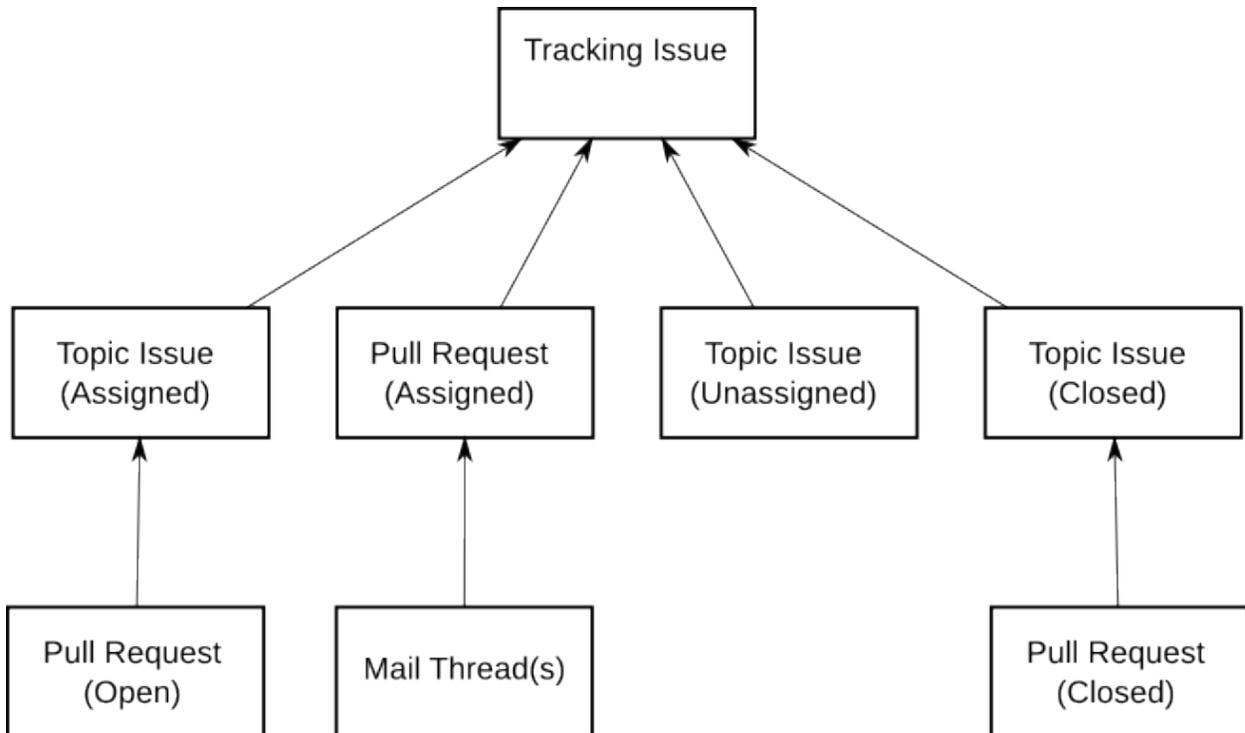
1. RFC email to the mailing list
2. Allow 2-3 days for feedback. RFC's often have a lower priority than bugs and usage problems.
3. Open github issues according to results of discussion
4. Create patchsets that implement solutions to the github issues
5. Review, fix comments, resend, until the patches are deemed good by the maintainers
6. Patchsets go to the next branch
7. Next branch gets tested/scrutinized by automated scripts
8. If needed, more bugfix iterations to fix the problems
9. next gets merged to master
10. master at some point is tagged as a stable autotest release

Although it seems convoluted, no one is stopping you from starting to design and implement your feature, and sending it straight away to github/mailling list (start on step 4). The maintainers will have to analyze and make judgement calls of whether the feature fits the current state of project, reason why it is more advisable to check on the feasibility before starting to spend too much energy implementing things.

You can see what to verify before sending patches in [the submission checklist page](#), and if you are new to git, you can read [the git workflow page](#).

Workflow Details

- Tracking issues do not take the place of high-level mailing-list discussoins and/or the *RFC process*. They are *only* intended to help coordinate simultaneous development on a specific topic.
- Tracking issues provide an automatically updated centralized location tracking a collection of related topic issues and pull requests.
- Anybody with access to open normal github issues and pull requests is able to link them to one or more tracking issues.
- No discussion should be posted to tracking issues directly. All discussion should happen within the topic-issues and pull requests.



Topic Issues

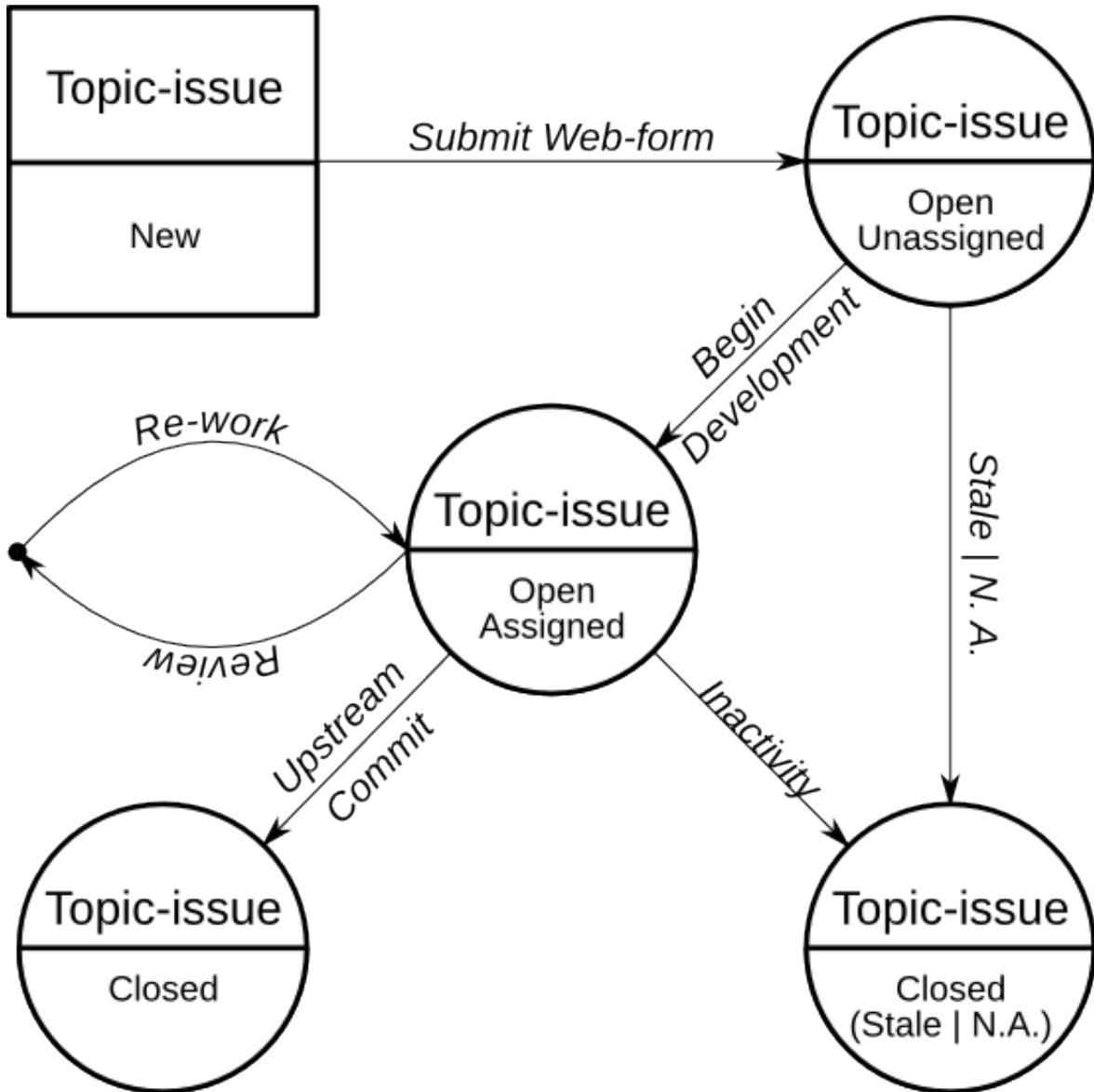
- For each proposed feature or enhancement, an issue is opened (Topic Issue)
- Topic issues summarize the proposed test/enhancement and provide a place for discussion.
- Topic issues are labelled with “*future*” and topic-specific label(s) such as “*virt-libvirt*”, “*client*”, *etc.*
- The topic issue is then linked to the tracking issue by mentioning it’s number. For example: “**Linking to tracking issue #9959**”
- Code cannot be posted to a topic issue directly. (see *Pull Requests* and *Mail List Publishing* below)

Topic Issue States

- **Open and unassigned:** Anybody may take ownership and begin working on this topic, and/or contribute to the discussion.
- **Open and assigned:** Someone is actively working on code for this topic. To avoid conflicts, other contributors will need to coordinate with the assigned person/team.
- **closed:** Code is finished and has been committed to the project. The issue may be re-opened under some circumstances. For example, if a major bug is discovered, and the code is removed from upstream.
- **closed stale:** Open or Open/Assigned issues with no code posted within several months.

Pull Requests

- [Pull requests](#) are a github-based tool where a developer makes a request that one of their topic-branches be merged with the upstream branch. Pull Requests may not be opened unless there are code changes available to push.



- All Pull Requests are also github issues. Comments can be posted, including comments in-line with the code.
- Sending the full patch-set to the Mailing list is not necessary. However a note to the list containing a link and summary are appreciated.
- Pull Requests are linked to tracking issues in exactly the same way as topic-issues. Simply mention it's number. For example: “**Linking to tracking issue #9959**“
- If multiple pull requests are required for a single topic, then an intermediate topic issue should be opened and linked to the tracking- issue. The pull requests may then all be linked to the intermediate topic issue.

Pull Request Updates

- Updates made by the author to a topic-branch (then pushed up to github) will automatically update the Pull Request.
- If other developers want to contribute to a pull request, the process is identical, except when submitting. In this case, the target branch should be the original author's forked branch instead of upstream.
- The original author may then review the changes, and if accepted they will automatically be merged in with the main pull request.
- **Utilizing this method is critical, since it preserves the state of the issue and keeps the tracking issue from becoming cluttered.**

Mail List Publishing

- Utilizing *git send-email*, patches may be sent to the mailing list. However, revisions require re-sending the entire patch-set. This works well for small, simple patches.
- In order to track proposed and under-development mailing list patch work, please also open a github *Topic Issues*. The patches should be referenced in the topic issue by pasting a http-link from [the mailing list archive](#)).
- Mailing list patches for anything reasonably complicated must be split up logically and use of a cover-letter is highly encouraged (see *git setup/usage*).
- Discussion regarding mailings list patches should occur on the mailing list. The github topic issue is simply used for tracking purposes.

Autotest Test API

This is a review of the available autotest test API.

Control files

A control file is just python code, and therefore should follow the Autotest python style. The control file ultimately defines the test. In fact the entire test can be coded in the control file. However if this leads to a very complicated control file, it is generally recommended that most of the test code logic be placed in a python module that the control file runs (via the job object).

A control file should define at the very top a set of variables. These are:

- AUTHOR
- TIME
- NAME

- TEST_CATEGORY
- TEST_CLASS
- TEST_TYPE
- SYNC_COUNT
- DOC

All except SYNC_COUNT are set to a string. SYNC_COUNT is a number which has relevance for the scheduling of multi-machine server side jobs. In addition you can define the variable EXPERIMENTAL (either True or False). By default it is False, but when set to True, will control whether the job shows up in the web frontend by default.

Unlike python test code, it is not imported, but rather is executed directly with the exec() method in the context of certain global and local symbols. One of the symbols that your control file can assume exists is *job*. The *job* object has a number of methods that you will most probably use in your control file. The most common are

- *job.run_test(test_object, tag, iterations, constraints, **dargs)******
- *job.parallel_simple(run_method, machine_list)*
- *job.record(status_code, subdir, operation, status)*

In addition, the control file has access to *machines* which is a list of the machines that were passed to the autoserv executable.

Client side tests

A client side test runs entirely on the client (or host machine). Essentially the entire client subdirectory of Autotest is installed on the host machine at the beginning of the test. And so the client control file through the *job.run_test()* method can execute code contained in a test class. A test class is code that is generally located in either a subdirectory of *client/tests/* or *client/site_tests/*. A test class always is a subclass of *autotest_lib.client.bin.test.test*. You then must provide an override for the *run_once()* method in your class. You must also define the class variable *version*. The *run_once* method can accept any arguments you desire. These are passed in as the ***dargs***in the ***job.run_test()* method in the control file.**

In addition to *run_once()* you may optionally override the following methods

- *initialize()*
- *setup()*
- *warmup()*
- *run_once()*
- *postprocess()*

The *initialize* is called first every time the test is run. The *setup* method is called once if the test version changed. Then the *warmup* is called once. After this *run_once* is called *iterations* times. Finally *postprocess* is called. The arguments that each method take are arbitrary. The ***dargs***from ***run_test()* are simply passed through. The exception being *postprocess* which takes no arguments (other than self of course).**

The *autotest_lib.client.bin.test.test* class also defines various useful variables. These are

- *job*: the job object from the control file
- *autodir*: the autotest directory
- *outputdir*: the output directory
- *resultsdir*: the results directory
- *profdir*: the profiling directory

- *debugdir*: the debugging directory
- *bindir*: The bin directory
- *srcdir*: the src directory
- *tmpdir*: the tmp directory

In addition the test object has a handful of very useful methods

- *write_test_keyval(attr_dict)*
- *write_perf_keyval(perf_dict)*
- *write_attr_keyval(attr_dict)*
- *write_iteration_keyval(attr_dict, perf_dict)*

The test keyvals are key/attribute pairs that are associated with the test. You supply a dictionary, and these will be recorded in a test level keyvals file as well as in the results (tko) database. The iteration keyvals can be either performance metrics (a number) or an attribute (a string). They can be recorded for each iteration, and you can either record one, the other, or both with the latter three methods above.

In addition the test class at the end of each iteration will evaluate any constraints that have been passed into the test via the `job.run_test()` command. The constraints variable is a list of strings, where each string makes an assertion regarding an iteration keyval. These are evaluated, and failures are recorded. An example constraints might be: `constraints = ['throughput > 6500', 'test_version == 2']`

Generally a typical client side test will make use of code contained in the standard python libraries, as well as the various utilities located in `autotest_lib.client.bin.utils`.

Server side tests

In a typical server side test, the autotest client is not installed on the host machines. Rather the server keeps host objects that represent an ssh connection to the host machine, and through which the server can execute code on the clients. A host object is generally created in the following way

```
host = hosts.create_host(machine)
```

The `hosts` module is one of those symbols that you can safely assume is present in your server control file. The machine is a machine name, and is generally one of the list `machines` which is also assumed to be accessible from your control file.

A typical server control file might look like

```
def run(machine):
    host = hosts.create_host(machine)
    ...

job.parallel_simple(run, machines)
```

In the above code, the `job.parallel_simple()` takes the list of `machines` and a method, and executes that method for each member of `machines`. The first line of the `run` method creates a `host` object that the server can use to execute commands (via ssh) on the client. A `host` object has various member variables:

- *hostname*
- *autodir*
- *ip*
- *user*

- *port*
- *password*
- *env*
- *serverdir*

Running code on a client can be done via the host object. Typical methods of a *host* object are:

- *run(cmd)*
- *run_output(cmd, *args, **dargs)******
- *reboot()*
- *sysrq_reboot()*
- *get_file(src, dest, delete_dest=False)*
- *send_file(src, dest, delete_dest=False)*
- *get_tmp_dir()*
- *is_up()*
- *is_shutting_down()*
- *wait_up(timeout=None)*
- *wait_down(timeout=None)*
- *ssh_ping(timeout=60)*

A large number of other methods are available and are scattered throughout the code in `server/hosts/`. The host object that is created by the `hosts.create_host()` method is a mix-in of various host behaviours that are defined in the `server/hosts` directory. However the most common are defined above.

In addition to methods on host, we can run client code via our server control file using an Autotest object. In order to use the autotest module you must import it from `autotest_lib.server`. A typical usage is

```
from autotest_lib.server import autotest

control_file = """job.run_test('sleepitest')"""

def run(machine):
    host = hosts.create_host(machine)
    at = autotest.Autotest(host)
    at.run(control_file, machine)

job.parallel_simple(run, machines)
```

The *autotest* object will (as part of its instantiation) install the autotest client on the host. Then we can use the *run* method to run code on the client. The first argument is a string. We could have just as easily written

```
at.run(open("some control file").read(), machine)
```

as well.

Multi-machine server side tests

The power of server side tests, is their ability to run different code on multiple machines simultaneously, and to control their interactions. The easiest way to describe a multi-machine test is to look at a real example of one. The following control file is located in `server/tests/netperf2/control.srv`

```
AUTHOR = "mbligh@google.com (Martin Bligh) and bboe@google.com (Bryce Boe)"
TIME = "SHORT"
NAME = "Netperf Multi-machine"
TEST_CATEGORY = "Stress"
TEST_CLASS = 'Hardware'
TEST_TYPE = "Server"
SYNC_COUNT = 2
DOC = ""
...
"""

from autotest_lib.server import utils, autotest

def run(pair):
    server = hosts.create_host(pair[0])
    client = hosts.create_host(pair[1])

    server_at = autotest.Autotest(server)
    client_at = autotest.Autotest(client)

    template = ''.join(["job.run_test('netperf2', server_ip='%s', client_ip=",
                        "'%s', role='%s', test='TCP_STREAM', test_time=10,",
                        "stream_list=[1,10]"))

    server_control_file = template % (server.ip, client.ip, 'server')
    client_control_file = template % (server.ip, client.ip, 'client')

    server_command = subcommand(server_at.run,
                                [server_control_file, server.hostname])
    client_command = subcommand(client_at.run,
                                [client_control_file, client.hostname])

    parallel([server_command, client_command])

# grab the pairs (and failures)
(pairs, failures) = utils.form_ntuples_from_machines(machines, 2)

for failure in failures:
    job.record("FAIL", failure[0], "netperf2", failure[1])

# now run through each pair and run
job.parallel_simple(run, pairs, log=False)
```

The top of the file contains the usual control variables. The most important one is `SYNC_COUNT`. This test is a 2 machine test. The first code that runs, is the line

```
(pairs, failures) = utils.form_ntuples_from_machines(machines, 2)
```

This code uses a method from `autotest_lib.server.utils` which given the full collection of `machines`, forms a list of `pairs` of machines, and a list of ‘failures’. These failures will, in this case, be at most a single machine (odd man out). The next line merely uses the `job` object to record a failure for each of the failures. After this, we call `job.parallel_simple()`

passing in the run function and the list of pairs.

The run function defined above takes a pair (recall the function referenced in *job.parallel_simple()* takes a single element from the list that is passed in. In this case it is a single pair). We then create a host object for each of the machines in the pair. Then we create an autotest object for each host. A control file string is then constructed for each of the machines. In this test one host acts as a client, while the other acts as a server in a network test between the two hosts. So in this test server does not refer to the autotest server, but rather to one of the autotest clients running this two machine test.

The next three lines are new. The *subcommand* class, and the *parallel* method are defined in *autotest_lib.server* and are assumed to be part of the control files namespace. The constructor to subcommand requires a method, and list of args to pass to that method

```
server_command = subcommand(server_at.run, [server_control_file, server.hostname])
```

Here the method is the *run* method of one of the autotest objects created earlier, and we are passing that method the *server_control_file*, and the hostname. We form the two subcommands (one for the netperf test server and the other for the netperf test client). We pass these both to the *parallel()* method as a list. This method executes both subcommands simultaneously.

The server netperf2 test whose control file is described above, makes use of the client side netperf2.py test file. This is located in *client/tests/netperf2/netperf2.py*. This code is resident on the host machines by virtue of the creation of the autotest objects. If you take a look at the *run_once* method of the netperf2 class, you will see how it is that we synchronize the running of the client and server sides of the netperf2 test. The relevant code is

```
...
server_tag = server_ip + '#netperf-server'
client_tag = client_ip + '#netperf-client'
all = [server_tag, client_tag]
...
if role == 'server':
    ...
    self.job.barrier(server_tag, 'start_%d' % num_streams, 600).rendevous(*all)
    ...
else if role == 'client':
    ...
    self.job.barrier(client_tag, 'start_%d' % num_streams, 600).rendevous(*all)
    ...
```

The above demonstrates how we can synchronize clients. In the above we register two tags (one for each of two roles). Recall that one of the hosts is running as the client, while the other is running as the server. We then form a list of the two tags. The next code segment is important. If we are the server, we employ the job object that every test has a reference to, and use it to construct a barrier object using the *server_tag*. This says we are registering at the barrier using the *server_tag* as our tag, and additionally we pass in 600 seconds as our timeout. The second argument is a logging string. We then call the *rendevous* method of the barrier object (yes it is mis-spelled in the code) and pass in **all*. This says that we will wait until all the tags in the *all* list register. The client side of the code does the complementary thing. The *rendevous* method blocks until both the *server_tag* and the *client_tag* register. Using these barriers, we can sync the client and server.

Submission common problems

These are quick notes to help you fix common problems autotest/virt-test code submissions usually have. Please read this and keep it in mind when writing code for these projects:

Gratuitous use of shell script inside a python program

While we understand that sometimes the contributions in question are adaptations of existing shell scripts, we ask you to avoid needlessly use shell script constructs that can be easily replaced by standard python API. Common cases:

1. Use of `rm`, when you can use `os.remove()`, and `rm -rf` when you can use `shutil.rmtree`.

Please don't

```
os.system('rm /tmp/foo')
```

Do

```
os.remove('/tmp/foo')
```

Please don't

```
os.system('rm -rf /tmp/foo')
```

Do

```
shutil.rmtree('/tmp/foo')
```

2. Use of `cat` when you want to write contents to a file

Please, really, don't

```
cmd = """cat << EOF > %s
Hey, this is a multiline text
to %
EOF""" % (some_file, some_string)
commands.getstatusoutput(cmd)
```

Do

```
content = """
Hey, this is a multiline text
to %s
""" % some_string
some_file_obj = open(some_file, 'w')
some_file_obj.write(content)
some_file_obj.close()
```

Use of the commands API, or `os.system`

Autotest already provides utility methods that are preferable over `os.system` or `commands.getstatus()` and the likes. The APIs are called `utils.system`, `utils.run`, `utils.system_output`. They raise exceptions in case of a return code `!=0`, so keep this in mind (either you pass `ignore_status=True` or trap an exception in case you want something different other than letting this exception coalesce and fail your test).

```
from autotest.client.shared import error
from autotest.client import utils

# Raises exception, use with error.context
error.context('Disabling firewall')
utils.system('iptables -F')
```

```
# If you just want the output
output = utils.system_output('dmidecode')

# Gives a cmdresult object, that has .stdout, .stderr attributes
cmdresult = utils.run('lspci')
if not "MYDEVICE" in cmdresult.stdout():
    raise error.TestError("Special device not found")
```

Use of backslashes

In general the use of backslashes is really ugly, and it can be avoided pretty much every time. Please don't use

```
long_cmd = "foo & bar | foo & bar | foo & bar | foo & bar | foo & bar \
           foo & bar"
```

instead, use

```
long_cmd = ("foo & bar | foo & bar | foo & bar | foo & bar | foo & bar "
           "foo & bar")
```

So, parentheses can avoid the use of backslashes in long lines and commands.

Use of constructs that appeared in versions of python > 2.4

Autotest projects use strictly python 2.4, so you can't use constructs that appeared in newer versions of python, some examples:

```
try:
    foo()
except BarError as details: # except ExceptionClass as variable was introduced after_
    ↪2.4
    baz
```

```
try:
    foo()
except BarError, details: # correct, 2.4 compliant syntax
    baz()
finally: # This is the problem, try/except/finally blocks were introduced after 2.4
    gnu()
```

So, when in doubt, consult the python documentation before sending the patch.

Unconditional import of external python libraries

Sometimes, for a tiny feature inside the test suite, people import an external, lesser known python library, on a very central and prominent part of the framework.

Please, don't do it. You are breaking other people's workflow and that is bad.

The correct way of doing this is conditionally importing the library, setting a top level variable that indicates whether the feature is active in the system (that is, the library can be imported), and when calling the specific feature, check the top level variable to see if the feature could be found. If it couldn't, you fail the test, most probably by throwing an `autotest.client.shared.error.TestNAError`.

So, instead of doing:

```
import platinumlib
...
platinumlib.destroy_all()
```

You will do:

```
PLATINUMLIB_ENABLED = True
try:
    import platinumlib
except ImportError:
    PLATINUMLIB_ENABLED = False
...
if not PLATINUMLIB_ENABLED:
    raise error.TestNAError('Platinum lib is not installed. '
                             'You need to install the package '
                             'python-platinumlib for this test '
                             'to work.')
platinumlib.destroy_all()
```

Any patch that carelessly sticks external library imports in central libraries of virt-test for optional features will be downright rejected.

Autotest requirements

Make it simple to use

- Make the system as user-friendly as possible, whilst still allowing power users (defaults with overrides!)
- Provide web front-ends where possible.
- Capture the “magic” knowledge of how to complex or fiddly operations within the harness, not in a person.
- Low barrier to entry for use and development

Gather as much information as possible

- Collect stdout and stderr. Break them out per test.
- Collect dmesg, and serial console where available. Fall back to netconsole where not.
- Gather profiling data from oprofile, vmstat etc.
- On a hang, gather alt+sysrq+t, etc.
- Monitor the machine via ssh and ICMP ping for it going down

Allow the developers to DEBUG the test failures

- Allow them to rerun the exact same test by hand easily.
- Keep the tests as simple as possible.
- Provide tracebacks on a failure
- Provide a flexible control file format that allows developers to do custom modifications easily.

Support all types of testing

- Allow tests to run in parallel
- Provide reproducible performance benchmarks

- Allow multiple iterations to be done cleanly for performance testing.
- Support filesystem tests (mkfs, mount, umount, fsck, etc)
- Provide test grouping into single units (build, filesystem, etc).
- Support multi-machine testing and provide synchronization barriers
- Support virtualized machines (Containers, KVM, Xen)

An OPEN harness

- Allow us to interact with vendors by sharing tests and problem scenarios easily
- Allow us to interact with the open source community by sharing tests and problem scenarios easily
- Encourage others to contribute to development.
- Also cleanly support proprietary tests where necessary, and code extensions.

Robust operation

- Allow reinstall of machines from scratch
- Support power cycle on failure

Scheduling and automation

- Provide one job queue per machine, or machine group
- Collect results to a central repository
- Automatically watch for new software releases, and kick off any job based on that.

Provide back-end analysis

- Suck all the results into a simple, well formatted database.
- Give a clear PASS/FAIL indication from the client test
- Allow arbitrary key-value pairs per test iteration
- Provide clear display of which tests passed on which machines.
- Graph performance results over time, indicating errors, etc.
- Compare two releases for statistically significant performance differences.

Autotest Design Goals

- Open source - share tests and problem reproductions
- Make it simple to write control files, and yet a powerful language
- Make it simple to write tests and profilers - encourage people to contribute
- Client is standalone, or will plug into any server/scheduler harness
 - Some people just have single machine, want simple set up.
 - Corporations each have their own scheduling harness, difficult to change
 - Very little interaction is needed, simple API
 - Simple handoff from full automated mesh to individual developer
- Maintainable
 - Written in one language, that is powerful, and yet easy to maintain

- Infrastructure that is easily understandable, and allows wide variety of people to contribute
- Modular - the basic infrastructure is cleanly separated with well defined APIs.
 - Easy writing of new tests, profilers, bootloaders, etc
 - New non-core changes (eg new test) doesn't break other things.
 - Lower barrier to entry for new developers.
 - Distributed/scalable maintainership - code controlled by different people.
- Core is small, with few developers
 - This isn't a super-hard problem.
 - Most of the intelligence is in sub-modules (eg the tests).
- Error handling.
 - Tests that don't produce reliable results are useless in an automated world.
 - Developers don't write error checking easily - need 'encouragement'.

Modules

- Core - ties everything together
- Tests - execute each tests. many, many separate tests modules.
 - eg kernbench, aim7, dbench, bonnie, etc.
- Profilers - gather information on a test whilst it's running, or before/after.
 - eg readprofile, oprofile, schedstats, gcov, /proc info
- Results Analysis - Compare equivalent sets of test results. Per test / profiler.
- Kernel build - build kernels, with different patches, configs, etc.
 - Will need different variations to cope with mainline, distro kernels, etc.
- Bootloader handling - Install new kernels, and reboot to them, pass parameters, etc
 - eg. Grub, Lilo, yaboot, zlilo, etc

Key differences

Here are some of the key changes from previous systems I have seen / used / written:

- The job control file is a script. This allows flexibility and power.
- The client is standalone, so we can easily reproduce problems without the server.
- Code and tests are modular - you can allow loser control over tests than the core.
- Code is GPL.

Autotest Maintenance Docs

This document was written to increase the [Bus Factor](#) of the autotest project. Jokes aside, distributing tasks makes the project more maintainable, given that the load is spread across individuals.

So, these are the activities of a project maintainer, according to the current project conventions:

1. Patch review / Update of development branch
2. Sync of the development / master branches
3. Policy definition and enforcement

Let's talk about each one of them.

Quick primer to pull request maintenance

We will talk about all that on the following topics, but we have a little video, part of our autotest weekly hangout, where I speak about maintenance. It might be useful to watch it, then read the rest of the document.

<https://www.youtube.com/watch?v=EzB4fYX5i4s>

The actual maintenance talk is between 37:00 - 49:40.

Patch reviewing and devel branch update

We strive to keep a model similar to the one described [in this link](#) which boils down to:

1. Have a master branch, which is always supposed to be stable
2. Have a next branch, which is the integration branch
3. When the master branch is updated, by definition, this is a stable release

In the case of the autotest project (the framework project) the only exception is that we define what is a release in terms of desired functionality, so there might be many syncs next-master before a stable release can be called upon.

On sub projects, such as virt tests, we adopt the model as is, every next-master sync means a stable release, that we tag with a timestamp in ISO 8601 format. So, given that this document is the reference document for all projects under the autotest umbrella, please keep in mind those little differences.

Very well. Autotest currently uses [github](#) as the project infrastructure provider. In the past, we used our own hosted solutions, which were useful at one point, but then became too burdensome to maintain them. Github has a functionality called [Pull requests](#) that pretty much presents a patch set in a graphical, rich way, and allows people that have github accounts to comment on the patches.

If you're not familiar with the process, please read the docs pointed out above. Now, the caveat here is that we don't use the pull request functionality of automatically merge the code to the branch against the code is being developed against. This is because we have checker scripts used to verify the code being submitted for:

1. Syntax errors
2. Code that breaks existing unittests
3. Permission problems (like an executable script without executable permissions)
4. Trailing whitespace/inconsistent indentation problems

Like it or not, keeping the code clean with regards to these problems is project policy, and tends to make our life better in the long term. So here are the tools that we hope will make your life easier:

Autotest

Pre-Reqs

These tools assume you have a number of dependency packages installed to your box to run all these effectly, such as pylint, for static checking, Django libs to run autotest DB unittests, so on and so forth. So you may go to [this link](#) for instructions on how to install them.

Tools

utils/check_patch.py - This tool is supposed to help you to verify whether a code from a pull request has no obvious, small problems. It'll:

1. Create a new branch from next (our reference devel branch)
2. Apply the code in the form of a patch
3. Verify if all changed/created files have no syntax problem (run_pylint.py with -q flag)
4. Verify if any changed/created files have no indentation/trailing whitespace problems
5. Verify if any changed/created files have a unittest, in which case it'll execute the unittest and report results

If any problems are found, it will return exit code != 0 and ask you to fix the problems. In this case, you can point out the code submitter of the problems and ask him/her to fix them. In order to check a given pull request, say:

<https://github.com/autotest/autotest/pull/619>

You'll just execute:

```
utils/check_patch.py -g 619
```

And that'd be it. This script has also another important function - It is a full tree checker, useful to check your own code. Just execute:

```
utils/check_patch.py --full --yes
```

And it'll scan through all files and point you all problems found.

utils/unittest_suite.py - Runs all unittests. Ideally the output of it should be like:

```
utils/unittest_suite.py --full
Number of test modules found: 81
autotest.client.kernel_versions_unittest: PASS
autotest.tko.utils_unittest: PASS
autotest.mirror.database_unittest: PASS
autotest.scheduler.gc_stats_unittest: PASS
autotest.client.shared.settings_unittest: PASS
autotest.client.shared.control_data_unittest: PASS
autotest.database_legacy.db_utils_unittest: PASS
...
All passed!
```

If it is not, please check out the errors.

Virt-Test

tools/check_patch.py - Exactly the same as utils/check_patch.py from autotest, the difference is the path, really.

tools/run_unittests.py - Exactly the same as the autotest version, only the path is different.

Applying the code that was reviewed and looks ready for inclusion

You'll:

1. Apply the code using the `check_patch` script. The execution should come clean.
2. `git checkout next`
3. `git merge github-[pull request number]` that was created by the script
4. `git push`

That's it. Alternatively, you can use GitHub tools to perform branch merging, such as hitting the green button, or pulling from the branch manually. As long as you've done your due diligence, it's all fine.

Policy enforcement

There are a number of common mistakes made by people submitting patches to autotest and offspring projects, more frequent when the contributions are test modules. So when you find such mistakes, please politely help them localize their mistakes and refer them to [this link on test coding style](#).

Other than that, trying to give the best of your attention on a patch review is always important.

Non fast forward updates

Sometimes we need to update the development branch in a non fast forward way. This is fine, considering the dev branch is not supposed to be fast forward, however, in order to ease the work of your fellow maintainers, some care has to be taken (we should keep those updates to a minimum). The main use case for non fast forward update is when there's a patch that introduced a regression, and we have to either fix the patch or drop it from next.

In case you have to do it, please make an announcement on the mailing list about it, explaining the reasons underlying the move.

Sync of the development branches

The development branch should pass through regular QA in order to capture regressions in the code that is getting added to the projects. The current tests comprise:

1. Job runs on a sever that is updated every day with the latest contents of the development branch
2. Unittests on a recent dev platform (F18, Ubuntu 12.04)
3. Static checking on an older system with python 2.4 (such as RHEL5)

So, there are 2 possibilities:

1. The development branch passes all tests, then it is considered apt to release. The merge could've happen right away.
2. The tests fail. The bad commit should be either fixed straight away, or yanked from the branch.

More details about this step should be written at a later point.

Becoming a Maintainer

Besides the ability to commit code directly to the `next` branch, and being an authority over some aspect of the tree, there is little other difference with working as a public contributor. That is to say, a maintainer has exactly the same

expectations as a contributors, but with the addition of a few more responsibilities. With that in mind, whether you are nominated or request maintainer access, here is a *guideline* for the minimum requirements:

1. X Code submissions per month.
2. Y Community-code submission reviews per month.
3. Z days elapsed since first code submission.

In general becoming a maintainer follows the following workflow:

1. Candidate is nominated, or pledges to a current maintainer.
2. Data from above is presented to Maintainer council for relevant project aspect (i.e. autotest, virt-test/libvirt, qemu, etc.).
3. Maintainer council reviews data and discusses candidate.
4. Feedback is provided to candidate on decision and/or areas needing improvement.

If the Maintainer Council approves the request:

1. Access is granted.
2. Community announcement delivered.
3. MAINTAINERS document(s) updated.
4. Requirements and expectations (re-)communicated.

Global Configuration

The global configuration is responsible for configuring many different aspects of the autotest programs. The client, server, scheduler, some portions of the frontend as well as other stand alone scripts require this file to get specific information about your setup. Below is a list of sections and in each section the options available in the configuration are described.

If you are making a stand alone checkout of the autotest client, it will warn you that you might want to create a default config file. If you want to do so, create a `global_config.ini` file inside the client directory with the documented keys on this page, it will look something like this:

```
[CLIENT]
drop_caches: True
drop_caches_between_iterations: True
```

For the other autotest programs, it's necessary that you have `global_config.ini` set on a proper location.

CLIENT

This section describes the global config [CLIENT] section.

Key	Description
drop_caches	If the autotest client will drop the memory cache for the client machine between test executions
drop_caches_between_iterations	If the autotest client will drop the memory cache for the client machine between test iterations executions
output_dir	Specify an alternate location to store the test results.

COMMON

This section describes the global config [COMMON] section.

Key	Description
autotest_top_path	The path for the toplevel autotest directory, defaults to /usr/local/autotest, might vary among distributors.

AUTOTEST_WEB

Parameters for configuring the frontend and scheduler database connections

Key	Description
host	The host name where the database is located
database	The name of the database
db_type	The type of database running (mysql, sqlite)
user	Username to connect to the database
password	Username to connect to the database
job_timeout_default	Default timeout (in hours) for new jobs. If the job gets schedule but it doesn't get to run, it'll be aborted without it running at all if this timeout is reached.
job_max_runtime	Default timeout (in hours) for running jobs. If job gets to run, but it doesn't finish during this timeout, it'll be aborted.
base_url	URL to your Autotest server's AFE interface. You only need this option if the URL is something other than http://protect\T1\textdollarhostname/afe/ , where \$hostname is the "hostname" value from the SERVER section.
template_debug_mode	Whether to enable django template debug mode. If this is set to True, all django errors will be wrapped in a nice debug page with detailed environment and stack trace info. Turned off by default.
sql_debug_mode	Whether to enable django SQL debug mode. If this is set to True, all queries performed by the Object Relational Mapper subsystem will be printed, which means the scheduler logs will contains all the queries executed. This is too much verbosity for 'production' systems, hence turned off by default.

Retry configuration

The db.py API for connecting to the TKO database includes support for automatically reconnecting and retrying queries when they fail due to OperationalErrors (assuming this is possible, i.e. when autocommit is in use).

Key	Description
query_timeout	Maximum number of seconds to wait before no giving up and no longer retrying
min_retry_delay	The minimum number of seconds to wait after an OperationalError before reconnecting and retrying
max_retry_delay	The maximum number of seconds to wait after an OperationalError before reconnecting and retrying

Graph configuration

Configuration parameters for the TKO graphing interface

Key	Description
graph_cache_creation_timeout_minutes	How frequently cached images for embedded graphing queries will be updated

AUTOSERV

Key	Description
client_autodir_paths	A comma-delimited list of paths where autoserv will attempt to install clients onto test machines
ssh_engine	Autotest has 2 implementations of SSH based hosts, the default (raw_ssh), and another one based on the python SSH library paramiko (paramiko). You can change the default 'raw_ssh' to 'paramiko' if you want to.
enable_master_ssh	Enable OpenSSH connection sharing. Only useful if ssh_engine is 'raw_ssh'
require_atfork_module	Fix problems originated from logging + threading inside autotest. Specially useful when ssh_engine is 'paramiko'
use_sshagent_with_paramiko	Set to False to disable ssh-agent usage with paramiko

SERVER

Key	Description
hostname	The hostname of the server running the Autotest web interface.

INSTALL_SERVER

Code to interact with a provisioning system, to make it install clients.

Key	Description
type	Type of install server we talk to. Default: cobbler
xmlrpc_url	RPC server URL for your install server. Example: http://foo.com/cobbler_api
xmlrpc_user	XMLRPC user, in case the server requires authentication
xmlrpc_password	XMLRPC password, in case the server requires authentication

SCHEDULER

This section describes the [SCHEDULER] section of the global configuration.

Key	Description
notify_email	Email address to receive warning and error messages from the scheduler
notify_email_from	Email address from which to send scheduler messages; defaults to the user running the scheduler
notify_email_statuses	When a host in a job reaches one of these statuses, send email to the email_list field of that job. If empty, email will only be sent when the whole job completes.
max_processes_per_driver	Maximum number of running Autoserv processes at once on a single server
max_jobs_started_per_driver	Maximum number of Autoserv processes started within one scheduler cycle
max_parse_processes	Maximum number of parser processes running at once
tick_pause_sec	The pause (in seconds) between the end of a tick and the beginning of the next tick
clean_interval_minutes	Time (in minutes) between database sweeps to abort timed-out jobs
synch_job_start_timeout_minutes	Time (in minutes) after which a synchronous job that has not yet started running will be aborted)
results_host	A host to offload results to via rsync/scp Default: localhost
results_host_installation_directory	If you installed your results_host in a different location than the standard directory/autotest, this often will be blank

Distributed execution parameters

The following parameters only need to be changed in a *Distributed Server Setup*.

drones	List of hostnames to act as drones (machines that run Autoserv)
drone_installation_directory	Directory in which Autotest is installed on drones, from which Autoserv will be run
results_host	Hostname to copy results to after job completion
max_transfer_processes	Maximum number of rsync/scp transfers to the results repository at once.

The following are optional parameters that can be used in a *Distributed Server Setup*.

archive_host	An additional hostname to check for results files when they cannot be found elsewhere after a user requests logs through the web interface
\$host-name_disabled	If set to 1, the drone \$hostname will be disabled – no new jobs will run, but existed jobs will be seen to completion
\$host-name_max_processes	Overrides max_processes_per_drone for a particular drone

HOSTS

This section describes the [HOSTS] section of the global configuration.

Key	Description
wait_up_processes	A comma-delimited list of processes that Host.wait_up expects to find one of before it considers the host “up”
default_protection	Default level of protection to put on new hosts. See <i>HostProtections</i>

PACKAGES

This section describes the [PACKAGES] section of the global configuration.

Key	Description
fetch_location	http://myserver.blah.com
upload_location	/usr/local/autotest/packages
serve_packages_from_autoserv	If set to True, autoserv will act as a last-resort package repository, allowing you to use the packaging system without setting up HTTP repositories. This defaults to True, but in large-scale production setups where you expect to run a large number of simultaneous autoserv processes you may want to disable this as autoserv builds up the package tarballs on-demand and so this is significantly more expensive than serving static packages over HTTP.

Adding site-specific extensions

If you need to extend the Autotest code in a way that isn’t usable by the main project, then you’ll probably want to do so in a way that doesn’t unduly complicate merging your local, extended code with the official project code. In general this means that you want to pull any site-specific code into separate files, and have the main code call into the extension in an optional way.

For site-specific tests this is not a problem. Each test should be self-contained in its own directory and so you should be able to add new tests without any other changes to Autotest at all. There may occasionally be a conflict if a new test is added to the project that conflicts with a private name you’re already using, but this will should not be overly common and is easily fixed by renaming.

For adding site-specific common libraries, this is also not a big problem. Add your module to the client/common_lib directory but add the name of your module to client/common_lib/site_libraries.py instead of directly to client/common_lib/___init___.py. This will create a small conflict as your local client/common_lib/site_libraries.py will differ from the official one, however since the official one should never really be changing, merging should never be a problem. However, remember that any code that imports these site-specific libraries has itself become site-specific.

In any other cases where you have to modify the core Autotest code, you'll have to make an effort to separate out your extensions from the main body of code. Assuming your extension is being done in a file x.py, the easiest way to extend it is to add a new module site_x.py that contains your site specific-code, and then add code to x.py that imports site_x and makes the appropriate calls.

Now, you'll want to be able to push out these calls to site_x into the official code so that you don't have to constantly merge around them. That means you'll still have to be careful about how you use site_x. In particular:

1. the import of site_x has to be done in such a way the code still works properly when site_x doesn't exist
2. the coupling between x and site_x should be as minimal as possible (to reduce the chances that other people's changes to x inadvertently break site_x)

As an example, look at the use of site_kernel in client/bin/kernel.py. It supports point 1 by pulling in a function from site_kernel, and if the import of site_kernel fails, it provides a default implementation of the function it is trying to import. It supports point 2 by only inserting a single call into auto_kernel stage, one with very clear and simple semantics (i.e. perform some optional, site-specific munging of path names before using them).

Adding site-specific extensions to the CLI

If you need to change the default behavior of some autotest-rpc-client commands, you can create a cli/site_<topic>.py file to subclass some of the classes from cli/<topic>.py.

The following example would prevent the creation of platform labels:

```
import inspect, new, sys

from autotest_lib.cli import topic_common, label

class site_label(label.label):
    pass

class site_label_create(label.label_create):
    """Disable the platform option
    autotest-rpc-client label create <labels>|--blist <file>"""
    def __init__(self):
        super(site_label_create, self).__init__()
        self.parser.remove_option("--platform")

    def parse(self):
        (options, leftover) = super(site_label_create, self).parse()
        self.is_platform = False
        return (options, leftover)

# The following boiler plate code should be added at the end to create
# all the other site_<topic>_<action> classes that do not modify their
# <topic>_<action> super class.
```

```
# Any classes we don't override in label should be copied automatically
for cls in [getattr(label, n) for n in dir(label) if not n.startswith("_")]:
    if not inspect.isclass(cls):
        continue
    cls_name = cls.__name__
    site_cls_name = 'site_' + cls_name
    if hasattr(sys.modules[__name__], site_cls_name):
        continue
    bases = (site_label, cls)
    members = {'__doc__': cls.__doc__}
    site_cls = new.classobj(site_cls_name, bases, members)
    setattr(sys.modules[__name__], site_cls_name, site_cls)
```

Autotest status file specification

General Structure

The status file is a variably indented human readable text file format storing the results or various steps done while running an Autotest job (ex. reboot start/end, autotest client install, test run/end, etc). The file is organized by lines and columns, where columns are separated by TABs. Each line has at least 3 columns:

```
<command><TAB><subdir><TAB><testname><TAB>...optional content
```

Note: there must be a trailing <TAB> after the last column on any line

Before the <command> there can be a number of <TAB> characters (also known as the indentation level).

Formal syntax and semantics specification

The formal definition of the file can be written like this (assuming the job was not aborted and thus the result file is complete):

```
<line>
<line>
...
EOF
```

Where:

<line> := [<status-line>|<info-line>|<group>] # inside a group we can have status lines, info lines or other groups

<status-line> := [<abort-line>|<alert-line>|<error-line>|<fail-line>|<good-line>|<warn-line>]

<abort-line> := "ABORT<TAB><subdir-testname><optional-fields>\n"

<alert-line> := "ALERT<TAB><subdir-testname><optional-fields>\n"

<error-line> := "ERROR<TAB><subdir-testname><optional-fields>\n"

<fail-line> := "FAIL<TAB><subdir-testname><optional-fields>\n"

<good-line> := "GOOD<TAB><subdir-testname><optional-fields>\n"

<warn-line> := "WARN<TAB><subdir-testname><optional-fields>\n"

<info-line> := "INFO<TAB><subdir-testname><optional-fields>\n"

<subdir-testname> := [<none-subdir-testname>|<valid-subdir-testname>]

<none-subdir-testname> := “—<TAB>—<TAB>”

<valid-subdir-testname> := “<subdir><TAB><valid-testname><TAB>”

<subdir> := | arbitrary string of characters that does not contain <TAB>?

<testname> := arbitrary string of characters that does not contain <TAB> and is not equal to “—“

<optional-fields> := [”“|”<optional-fields-elements><reason><TAB>”] # optional fields can either be empty or if not must have a reason at the end which is not key=value syntax

<reason> := string description of a success/failure reason, does not contain <TAB>

<optional-fields-elements> := [”“|<optional-field-element>] # we may have a reason but no other optional field

<optional-field-element> := “<optional-field-name>=<optional-field-value><TAB><optional-fields-elements>” # the optional fields to the left of the reason field must be of key=value syntax

<optional-field-name> := string of characters that do not contain “=” or <TAB>

<optional-field-value> := string of characters that do not contain <TAB>

<group> := “<start-line><group-contents><end-line>”

<start-line> := “START<TAB><subdir-testname><optional-fields>\n”

<end-line> := “<end-command><TAB><subdir-testname><optional-fields>\n”

<end-command> := [”END ABORT”|”END FAIL”|”END GOOD”]

<group-contents> := [”“|<group-line>] # a group can be empty

<group-line> := “<TAB><line>”

Definitions:

- a job group is a group with testname “SERVER_JOB” or “CLIENT_JOB”
- a test group it’s a group with testname != “—” that is not a job group
- a base test group is a test group that may be included in a job group but is not included in any test group

The formal syntax definition cannot express semantical constraints on the contents of the file. These are:

- inside a base test group all valid (that is all values except the “—” ones) <testname> columns of any line must be equal to the base test group <testname> (that is, there are no sub-tests, once a base test group has started everything inside is relevant for that test)
- a job group is present only once in a result file (ie you can’t have multiple job groups with the same <testname>)
- it’s invalid to have 2 or more test groups with the same <testname> unless one of them includes all the others
- the next same indentation level END line after a START line shall have the same <testname> as its corresponding START line or have “—” <testname>
- it’s invalid to have a status-line with “—” subdir and testname while not being inside a base test group
- it’s invalid for a <status-line> inside a job group but not inside a base test group to have the same <testname> as an active job group <testname> unless it’s the inner most job group

Parsing Behaviour

A violation of the syntactical and semantical constraints shall result in behaviour as if the next lines in the input buffer after the faulty line are just a sequence of END ABORT lines ending all the active (started but not ended) groups having subdir/testname corresponding to the group they end.

<status-line> parsing:

- if the line has a valid subdir and we are inside a base test group then we update the base test group's subdir
- if there is no current base test group and if the status line <testname> does not refer to an active job group it will behave as if the input buffer has a test group START/END lines with the status line testname, subdir, reason, finished time (from the timestamp optional field)
- if there is no current base test group and the status line <testname> is equal to an active job group <testname> it will update the status of that job group if the <status-line> status is worse in which case if there is a reason field it will be used to update the current reasons of the referred job group
- if the status line is inside a base test group it will update that group's current status if the new status is worse the the old one and finished time (based on the optional timestamp field); if it has updated the status and if it has a reason field it will be used to update the current reasons of the base test group

A <info-line> parsing can be used to update the current kernel version if there is such an optional field. The current kernel version is a parser wide state variable that crosses group boundaries. Can't there be multiple clients registering INFO for various kernels they boot in the server server side results file??

When parsing a <end-line> besides ending the current group:

- the status of the END line (determined by the word after the "END " part) will be used to update the current group status
- if the previous group is a test group with an invalid (ie "—") subdir update the subdir of the previous group with the current group subdir
- the finished time of the current group is updated with the timestamp of the END line
- if the end line is for a reboot operation then current kernel version is updated with the version from this line
- if this is the end of a base test group it will be recorded in the db with the state, subdir, testname, reasons, finished_timestamp, current kernel version

Autotest job results specification

On the client machine, results are stored under `$AUTODIR/results/$JOBNAME/...`, where `$JOBNAME` is *default* unless you specify otherwise.

Single machine job output format

The results to each job should conform to:

`$AUTODIR/results/default/$JOBNAME/...`

- debug/
- build<.*****tag*****>/
 - src/
 - build/
 - patches/
 - config/
 - debug/
 - summary
- testname<.*****tag*****>/
 - results/

- profiling/
- debug/
- tmp/
- summary
- sysinfo/
- control (*the control script*)
- summary

Format of status file

There are two copies of the status file, one written by the server as we go called “status.log”), and another copied back from the client (if it doesn’t crash) called “status”. Both have the same format specification. You can read more about the status file format at [StatusFileSpecification](#).

Multi-machine tests

When collating the results together for a multi-machine test, the results should be formatted with one subdirectory for each machine in the test, which should contain the job layout above.

There should be a .machines file in the top level that indicates to the parser that this a multi-machine job, and lists the correct directories to parse.

There are two ways a multi-machine job can be run:

- For synchronous jobs, the scheduler kicks off one copy of autoserv, with multiple machines passed with “-m” option. In this case, it’s autoserv’s responsibility to create the .machines file. This should be appended to, one machine at a time, as the main part of the job is kicked off.
- For asynchronous jobs, the scheduler kicks off one copy of autoserv per machine. In this case it is the *scheduler’s* responsibility to create the .machines file - we can’t do it from autoserv, as we didn’t know there were multiple machines.

Scheduler behavior

Results directories and autoserv execution:

- The scheduler always created a job directory, results/<job tag>
- For synchronous jobs, the scheduler runs a single instance of autoserv with all machines and with the job directory as the results directory.
- For asynchronous single-machine jobs, the scheduler runs a single instance of autoserv with that machine and with the job directory as the results directory.
- For asynchronous multi-machine jobs, the scheduler creates a results/<job tag>/<hostname> directory for each host and runs one instance of autoserv for each host with those directories as results directories.

Metahosts always get queue.log.<id> files created in the job directory (results/<job tag>). These logs contain a single line for each time a meta-host is assigned a new host or cleared of its host.

Verify information is handled like so:

- Verify logs from autoserv are always directed to a temporary directory using the -r option to autoserv.
- Verify stdout is also directed to a host log at results/hosts/<hostname>.

- On verify success, the contents of the temporary directory are moved to results/<job tag>/<hostname>, UNLESS it was an asynchronous single-machine job, in which case the contents are moved to results/<job tag>.
- On verify failure for a non-metahost, the contents are copied as for success.
- On verify failure for a metahost, the contents of the temporary directory are deleted. They are never placed in the job directory. The only place to find them is in the host log.

The scheduler only creates a .machines file for asynchronous multi-machine jobs. It creates this file on the fly by appending each hostname to this file right before running the main autoserv process on that host.

Documentation

There are two different ways to view the test API documentation.

The more complete (for now) way is to use Pydoc. The less complete (but new) way is to generate the HTML documentation.

Pydoc

Set your Python path to one directory before your autotest path, then start the pydoc web server on a port of your choosing.

For example, if your autotest installation is in /usr/local/autotest, then:

```
$ export PYTHONPATH=/usr/local
$ pydoc -p 8888
```

Now use a browser to visit [<http://localhost:8888>](http://localhost:8888).

This will show all of the Python modules on your system. Click on the autotest entry. Explore from there.

Generate the HTML API documentation

The new approach (still in progress), is to generate the API docs as html. The HTML docs are nicer looking than the Pydoc webserver ones, but are not yet as complete.

Here [is an example](http://justinlift.fedorapeople.org/autotest_docs/), generated on 6th Aug 2013.

Instructions to generate your own, known to work on Fedora 19:

```
$ sudo yum -y install MySQL-python python-django python-sphinx
$ cd /usr/local/autotest
$ python setup.py build_doc
running build_doc
Running Sphinx v1.1.3
loading pickled environment... done
building [html]: targets for 0 source files that are out of date
updating environment: 0 added, 4 changed, 0 removed
Traceback (most recent call last):istrot_detection
  File "/usr/lib/python2.7/site-packages/sphinx/ext/autodoc.py", line 321, in import_
↪object
    __import__(self.modname)
ImportError: No module named Probe
reading sources... [100%] frontend/tko_models
/usr/local/autotest/documentation/source/client/distro_detection.rst:91: WARNING: ↪
↪autodoc can't import/find data 'Probe.CHECK_VERSION_REGEX', it reported error: "No ↪
↪module named Probe", please check your spelling and sys.path
```

```
looking for now-outdated files... none found
pickling environment... done
checking consistency... done
preparing documents... done
writing output... [100%] index
writing additional files... (4 module code pages) _modules/index
  genindex py-modindex search
copying static files... done
dumping search index... done
dumping object inventory... done
build succeeded, 1 warning.
```

The generated docs should now be in `/usr/local/autotest/build/sphinx/html/`.

Autotest Unittest suite

The unittest suite module is the main entry point used to run all the autotest unit tests. It is important to keep this module running on the autotest code base to ensure we are not breaking the test coverage we already got.

Setting up dependencies

This documentation was written for a F18 development box, if you are running other OS to develop autotest, feel free to add the relevant bits for your distro.

First, install all dependencies:

```
sudo installation_support/autotest-install-packages-deps
```

Now, grab gwt for the dependencies (gwt isn't packaged right now):

```
utils/build_externals.py
```

To run the 'short' version of the unittests, just do a:

```
utils/unittest_suite.py
```

If you want to run the entire set of unittests, you have to pass the flag `-full`:

```
utils/unittest_suite.py --full
```

Web Frontend Development

When we run a production Autotest server, we run the Django server through Apache and serve a compiled version of the GWT client. For development, however, this is far too painful, and we go through a completely different setup.

Basic setup

Steps below assume that you have basic software setup. Make sure you run beforehand: `installation_support/autotest-install-package-deps` and `installation_support/autotest-database-turnkey`. On a new environment good validation step is to run unit tests before proceeding.

Django server development

You can read more about Django development at their [documentation site](#), but here's the short version.

Without Eclipse

- Running `manage.py runserver` will start a development server on <http://localhost:8000>. This server automatically reloads files when you change them. You can also view `stdout/stderr` from your Django code right in the console. There's not a whole lot you can do from your browser with this server by itself, since the only interface to it is through RPCs.
- `manage.py test` will run the server test suite (implemented in `frontend/afe/test.py`). This includes running `pylint` on all files in `frontend/afe/` (checking for errors only), running doctests found in the code, and running the extended doctests in `frontend/afe/doctests`. This suite is pretty good at catching errors, and you should definitely make sure it passes before submitting patches (and please add to it if you add new features). Note you may need to install `pylint` (Ubuntu package `python2.4-pylint`).
- On that note, `frontend/afe/doctests/rpc_test.txt` is also the best documentation of the RPC interface to the server, so it's a pretty good place to start in understanding what's going on. It's purposely written to be readable as documentation, so it doesn't contain tests for all corner cases (such as error cases). Such tests should be written eventually, but they don't exist now, and if you write some, please place them in a separate file so as to keep `rpc_test.txt` readable.
- You can test the RPC interface out by hand from a Python interpreter:

```
>>> import common # pylint: disable=W0611
>>> from frontend.afe import rpc_client_lib
>>> proxy = rpc_client_lib.get_proxy('http://localhost:8000/afe/server/rpc/',
↳ headers={})
>>> proxy.get_tests(name='sleeptest')
[{'description': u'Just a sleep test.', u'test_type': u'Client', u'test_class': u
↳ 'Kernel', u'path': u'client/tests/sleeptest/control', u'id': 1, u'name': u
↳ 'sleeptest'}]
```

With Eclipse

- First make sure that you have Eclipse working with PyDev (<http://pydev.org/index.html>)
- In Eclipse create django project wrapping frontend;
- File>New>Other...>PyDev>PyDev Django Project; click Next
- Project Contents, uncheck Use default and specify directory `autotest/frontend`, Next few times to set all properties
- Now you can use Debug As>PyDev: Django that will start your server in debug mode; You can use standard Eclipse facilities: breakpoints, watches, etc

Note that in both cases when django app is running you can use the admin interface locally by navigating to <http://localhost:8000/afe/server/admin/>; This allows to easily add some test data, examine existing records etc. Note that static files are not served properly so it is a big ugly but usable.

GWT client development

Again, the full scoop can be found in the [GWT Developer Guide](#), but here's the short version:

Without Eclipse

- `frontend/client/AfeClient-shell` runs a GWT development shell. This runs the client in a JRE in a modified browser widget. It will connect to the Django server and operate just like the production setup, but it's all running as a normal Java program and it compiles on-demand, so you'll never need to compile, you can use your favorite Java debugger, etc.
- Exception tracebacks are viewable in the console window, and you can print information to this console using `GWT.log()`.
- Hitting reload in the browser window will pull in and recompile any changes to the Java code.

With Eclipse

- First download and install GWT and Eclipse plug in and make sure all is working by running sample GWT app (<https://developers.google.com/web-toolkit/usingeclipse>)
- Change the settings in autotest `global_config.ini` file by turning on `sql_debug_mode: True` (section [AUTOTEST_WEB]); This will run frontend application in debug mode and forward calls to GWT running in debug mode (in addition to printing sql statements as name implies).
- Start the django app as described above by running `manage.py runserver` in frontend directory on default port 8000
- The `frontend/client/` directory contains `.project` and `.classpath` files for Eclipse, so you should be able to import the project using File->Import...->Existing Project into Workspace.
- Double check the project properties:
- Google->Web Application 'This project has a WAR directory' should be unchecked
- Google->Web Toolkit 'Use Google Web Toolkit' should be checked and project connected to appropriate GWT
- Java Build Path->Libraries tab: remove existing (probably bogus) gwt jar files references and click Add Library-> choose Google Web Toolkit
- Create a run configuration
- Choose 'Debug Configurations...' from the menu
- Click New under (Google) Web Application, give it a name, e.g. AfeFrontEnd
- Main tab: Project AfeClient; Main class: `com.google.gwt.dev.GWTShell` (default)
- GWT tab: URL: `autotest.AfeClient/AfeClient.html`
- Common tab: optionally set Display in favorites menu
- Start debugging AfeFrontEnd configuration
- Open in a browser url: `127.0.0.1:8000/afe/server/autotest.AfeClient/AfeClient.html?gwt.codesvr=127.0.0.1:9997`
Note is is important to use 8000 (django port) and not 8888 GWT port
- At this point you can use normal debugging facilities of Eclipse: set breakpoints, watches, etc
- Note that `frontend/client/AfeClient.launch` is not working at the moment and needs to be updated

See Also

- *AutotestServerInstall* <../sysadmin/AutotestServerInstall>

Using the Autotest Mock Library for unit testing

To aid with unit testing, we've implemented a very useful mocking and stubbing library under `client/shared/test_utils/mock.py`. This library can help you with

- safety stubbing out attributes of modules, classes, or instances, and restoring them when the test completes
- creating mock functions and objects to substitute for real function and class instances
- verifying that code under test interacts with external functions and objects in a certain way, without actually depending on external objects

Setting up to use the code

```
from autotest.client.shared.test_utils import mock
```

You'll often need a `mock_god` instance as we'll see later. This is best done in your `setUp` method:

```
class MyTest(unittest.TestCase):
    def setUp(self):
        self.god = mock.mock_god()
```

As we'll also see later, you'll often want to call `mock_god.unstub_all()` in your `tearDown` method, so I'll include that here too:

```
def tearDown(self):
    self.god.unstub_all()
```

Stubbing out attributes

Say we want to make `os.path.exists()` always return `True` for a test. First, we can create a mock function:

```
mock_exists = mock.mock_function('os.path.exists', default_return_val=True)
```

This returns a function (actually it's a callable object, but no matter) that will accept any arguments and always returns `True`. The function name passed in (`'os.path.exists'`) is used only for error messages and can be anything you find helpful. Next, we want to stub out `os.path.exists` with our new function:

```
self.god.stub_with(os.path, 'exists', mock_exists)
```

Now you can call the code under test, and when it calls `os.path.exists` it'll actually be calling your mock function. Note that `stub_with` can accept any object to use as a stub – it doesn't have to be a `mock_function`. You could define your own function to do actual work, but that's rarely necessary.

Calling `self.god.unstub_all()` will restore `os.path.exists` to its original value. **You must remember to always do this at the end of your test.** Even if your test never needs it to be unstubbed, your test may be combined with others in a single test run, and you could mess up those other tests if you don't clean up your stubs. The best way to do this is to **always call “`unstub_all()`” in your “`tearDown`” method** if you're using stubbing.

Stubbing methods on classes

The above approach won't work for stubbing out methods on classes (not instances, but the classes themselves). You'll need to use the trick of wrapping the mock function in `staticmethod()`:

```
self.god.stub_with(MyClass, 'my_method', staticmethod(mock_method))
```

Verifying external interactions of code under test

The above trick is nice, but what if you need to ensure the code under tests calls your mock functions in a certain way? For that, you can use `mock_god.create_mock_function`.

```
mock_exists = self.god.create_mock_function('os.path.exists')
self.god.stub_with(os.path, 'exists', mock_exists)
# note that stub_function() would be more convenient here - see below
```

How is this different from the above? Mock functions created using `mock_god.create_mock_function` follow the *expect/verify* model. The basic outline of this is as follows:

- Create your mock functions.
- Set up the expected call sequence on those functions.
- Run the code under test.
- Verify that the mock functions were called as expected.

Let's look at an example, following from the snippet above:

```
# return True the first time it's called
os.path.expect_call('/my/directory').and_return(True)
# return False the next time it's called
os.path.expect_call('/another/directory').and_return(False)
# run the code under test
function_under_test()
# ensure the code under test made the calls we expected
self.god.check_playback()
```

This tells the mock god to expect a call to `os.path.exists` with the argument `'/my/directory'` and then with `'/another/directory'`. If the code under tests makes these calls in this order, it will get the specified return values and `check_playback()` will return without error. `check_playback()` will raise an exception if any of the following occurred:

- a mock function was called with the wrong arguments
- a mock function was called when it wasn't supposed to be
- a mock function was not called when it was expected to be

Note that order must be consistent across all mock functions (remember god knows all)

Constructing mock class instances

Frequently our code under test will expect an object to be passed in, and we'll want to mock out every method on that object. In that case we can use `mock_god.create_mock_class`:

```
mock_data_source = self.god.create_mock_class(DataSource, 'mock_data_source')
mock_data_source.get_data.expect_call().and_return('some data') # method taking no_
↳parameters
mock_data_source.put_data.expect_call(1) # void method
function_under_test(mock_data_source)
self.god.check_playback()
```

This code creates a mock instance of `DataSource`. On the mock instance, *all public methods* of `DataSource` will be replaced with mock functions on which you can use the expect/verify model, just like functions created with `create_mock_function`. The second argument to `create_mock_class` can be any name; it's just used in the debug output.

Isolating a method from other methods on the same instance

You may find yourself needing to test a method of a class instance and wanting to mock out every other method of that instance. `mock_god.mock_up()` provides a convenient way to do this:

```
# construct a real DataSource
data_source = DataSource()
# replace every method with a mock function
self.god.mock_up(data_source, "data_source")
data_source.get_data.expect_call().and_return('data')
data_source.put_data.expect_call('more data')
# run a real method on the instance
data_source.do_data_manipulation.run_original_function()
# do_data_manipulation() calls get_data() and put_data()
self.god.check_playback()
```

Unlike `create_mock_class`, `mock_up` takes an existing instance and replaces all methods (that don't start with `'_'`) with mock functions, while retaining the ability to run the original functions through `run_original_function()`. Unlike `create_mock_class` it will mock up functions for “protected” (starting with `'_'`) methods.

Verifying class creation within code under test

What if your code under test instantiates and uses a class, and you want to mock out that class but never have access to it? In this case you can stub out the class itself using `mock_god.create_mock_class_obj`. I'll use `subprocess.Popen` as an example:

```
MockPopen = self.god.create_mock_class_obj(subprocess.Popen)
self.god.stub_with(subprocess, 'Popen', MockPopen)
# expect creation of a Popen object
proc = subprocess.Popen.expect_new('some command', shell=True)
# expect a call on the created Popen object
proc.poll.expect_call().and_return(0)
# code under test creates a subprocess.Popen object and uses it
function_under_test()
self.god.check_playback()
```

Convenient shortcuts for stubbing

`stub_function` automatically stubs out a function with a mock function created using `mock_god.create_mock_function`, so that you can use the expect/verify model on it.

```
self.god.stub_function(os.path, 'exists')
# this is equivalent to:
mock_exists = self.god.create_mock_function('exists')
self.god.stub_with(os.path, 'exists', mock_exists)
```

`stub_class_method` does the same thing, but wraps the mock function in `staticmethod()` and thus is suitable for class methods.

```
self.god.stub_class_method(MyClass, 'my_method')
# this is equivalent to:
mock_method = self.god.create_mock_function('my_method')
self.god.stub_with(MyClass, 'my_method', staticmethod(mock_method))
```

Stubbing out builtins

Often we'll want to stub out a builtin function like `open()`. We've found that the best way to do this is to set an attribute on the module under test, rather than try to mess with `__builtins__` or anything, as that can mess up other code (such as test infrastructure code).

```
self.god.stub_function(module_under_test, 'open')
# note we're using StringIO to fake a file object
module_under_test.open.expect_call('/some/path', 'r').and_return(StringIO.StringIO(
    ↪'file text'))

module_under_test.function_under_test() # tries to call builtin open
self.god.check_playback()
```

autotest_local Module

```
class autotest.client.autotest_local.AutotestLocalApp
    Autotest local app runs tests locally

    Point it to a control file and let it rock

    main()
    parse_cmdline()
    usage()
```

base_sysinfo Module

```
class autotest.client.base_sysinfo.base_sysinfo(job_resultsdir)
    Bases: object

    deserialize(serialized)
    log_after_each_iteration(*args, **dargs)
    log_after_each_test(*args, **dargs)
    log_before_each_iteration(*args, **dargs)
    log_before_each_test(*args, **dargs)
    log_per_reboot_data(*args, **dargs)
    log_test_keyvals(test_sysinfodir)
        Logging hook called by log_after_each_test to collect keyval entries to be written in the test keyval.
    serialize()
```

class `autotest.client.base_sysinfo.command` (*cmd*, *logf=None*, *log_in_keyval=False*, *compress_log=False*)

Bases: `autotest.client.base_sysinfo.loggable`

run (*logdir*)

class `autotest.client.base_sysinfo.logfile` (*path*, *logf=None*, *log_in_keyval=False*)

Bases: `autotest.client.base_sysinfo.loggable`

run (*logdir*)

class `autotest.client.base_sysinfo.loggable` (*logf*, *log_in_keyval*)

Bases: `object`

Abstract class for representing all things “loggable” by sysinfo.

readline (*logdir*)

base_utils Module

DO NOT import this file directly - import `client/bin/utils.py`, which will mix this in

Convenience functions for use by tests or whomever.

Note that this file is mixed in by `utils.py` - note very carefully the precedence order defined there

`autotest.client.base_utils.append_path` (*oldpath*, *newpath*)

append newpath to oldpath

`autotest.client.base_utils.avgtime_print` (*dir*)

Calculate some benchmarking statistics. Input is a directory containing a file called ‘time’. File contains one-per-line results of `/usr/bin/time`. Output is average Elapsed, User, and System time in seconds, and average CPU percentage.

`autotest.client.base_utils.cat_file_to_cmd` (*file*, *command*, *ignore_status=0*, *return_output=False*)

equivalent to ‘`cat file | command`’ but knows to use `zcat` or `bzcat` if appropriate

`autotest.client.base_utils.check_for_kernel_feature` (*feature*)

`autotest.client.base_utils.check_glibc_ver` (*ver*)

`autotest.client.base_utils.check_kernel_ver` (*ver*)

`autotest.client.base_utils.count_cpus` ()

Total number of online CPUs in the local machine

`autotest.client.base_utils.count_total_cpus` ()

Total number of (online+offline) CPUs in the local machine

`autotest.client.base_utils.cpu_has_flags` (*flags*)

Check if a list of flags are available on current CPU info

Parameters *flags* (*list*) – A list of cpu flags that must exist on the current CPU.

Returns *bool* True if all the flags were found or False if not

Return type *list*

`autotest.client.base_utils.cpu_online_map` ()

Check out the available cpu online map

`autotest.client.base_utils.difflist` (*list1*, *list2*)

returns items in *list2* that are not in *list1*

`autotest.client.base_utils.disk_block_size` (*path*)
Return the disk block size, in bytes

`autotest.client.base_utils.dump_object` (*object*)
Dump an object's attributes and methods
kind of like `dir()`

`autotest.client.base_utils.environ` (*env_key*)
return the requested environment variable, or "" if unset

`autotest.client.base_utils.extract_all_time_results` (*results_string*)
Extract user, system, and elapsed times into a list of tuples

`autotest.client.base_utils.extract_tarball` (*tarball*)
Returns the directory extracted by the tarball.

`autotest.client.base_utils.extract_tarball_to_dir` (*tarball, dir*)
Extract a tarball to a specified directory name instead of whatever the top level of a tarball is - useful for versioned directory names, etc

`autotest.client.base_utils.file_contains_pattern` (*file, pattern*)
Return true if file contains the specified egrep pattern

`autotest.client.base_utils.force_copy` (*src, dest*)
Replace dest with a new copy of src, even if it exists

`autotest.client.base_utils.force_link` (*src, dest*)
Link src to dest, overwriting it if it exists

`autotest.client.base_utils.freespace` (*path*)
Return the disk free space, in bytes

`autotest.client.base_utils.get_cc` ()

`autotest.client.base_utils.get_cpu_arch` ()
Work out which CPU architecture we're running on

`autotest.client.base_utils.get_cpu_family` ()

`autotest.client.base_utils.get_cpu_info` ()
Reads `/proc/cpuinfo` and returns a list of file lines
Returns *list* of lines from `/proc/cpuinfo` file
Return type *list*

`autotest.client.base_utils.get_cpu_stat` (*key*)
Get load per cpu from `/proc/stat` :return: list of values of CPU times

`autotest.client.base_utils.get_cpu_vendor` ()

`autotest.client.base_utils.get_cpu_vendor_name` ()
Get the current cpu vendor name
Returns string 'intel' or 'amd' or 'power7' depending on the current CPU architecture.
Return type *string*

`autotest.client.base_utils.get_current_kernel_arch` ()
Get the machine architecture

`autotest.client.base_utils.get_disks` ()

`autotest.client.base_utils.get_file_arch` (*filename*)

`autotest.client.base_utils.get_hwclock_seconds(utc=True)`

Return the hardware clock in seconds as a floating point value. Use Coordinated Universal Time if `utc` is `True`, local time otherwise. Raise a `ValueError` if unable to read the hardware clock.

`autotest.client.base_utils.get_loaded_modules()`

`autotest.client.base_utils.get_modules_dir()`

Return the modules dir for the running kernel version

`autotest.client.base_utils.get_os_vendor()`

Try to guess what's the os vendor.

`autotest.client.base_utils.get_submodules(module_name)`

Get all submodules of the module.

Parameters `module_name` (*str*) – Name of module to search for

Returns List of the submodules

Return type *list*

`autotest.client.base_utils.get_systemmap()`

Return the full path to `System.map`

Ahem. This is crap. Pray harder. Bad Martin.

`autotest.client.base_utils.get_uptime()`

Returns return the uptime of system in secs in float in error case return 'None'

`autotest.client.base_utils.get_vmlinux()`

Return the full path to `vmlinux`

Ahem. This is crap. Pray harder. Bad Martin.

`autotest.client.base_utils.grep(pattern, file)`

This is mainly to fix the return code inversion from `grep` Also handles compressed files.

returns 1 if the pattern is present in the file, 0 if not.

`autotest.client.base_utils.hash_file(filename, size=None, method='md5')`

Calculate the hash of filename. If `size` is not `None`, limit to first `size` bytes. Throw exception if something is wrong with filename. Can be also implemented with bash one-liner (assuming `size%1024==0`): `dd if=filename bs=1024 count=size/1024 | sha1sum -`

Parameters

- **filename** – Path of the file that will have its hash calculated.
- **method** – Method used to calculate the hash. Supported methods: * `md5` * `sha1`

Returns Hash of the file, if something goes wrong, return `None`.

`autotest.client.base_utils.human_format(number)`

`autotest.client.base_utils.list_grep(list, pattern)`

True if any item in list matches the specified pattern.

`autotest.client.base_utils.load_module(module_name)`

`autotest.client.base_utils.loaded_module_info(module_name)`

Get loaded module details: Size and Submodules.

Parameters `module_name` (*str*) – Name of module to search for

Returns Dictionary of module info, name, size, submodules if present

Return type dict

`autotest.client.base_utils.locate` (*pattern*, *root*='/home/docs/checkouts/readthedocs.org/user_builds/autotest/checkouts')

`autotest.client.base_utils.module_is_loaded` (*module_name*)

Is module loaded

Parameters `module_name` (*str*) – Name of module to search for

Returns True is module is loaded

Return type bool

`autotest.client.base_utils.parse_lsmod_for_module` (*l_raw*, *module_name*, *escape*=True)

Use a regexp to parse raw lsmod output and get module information :param l_raw: raw output of lsmod :type l_raw: str :param module_name: Name of module to search for :type module_name: str :param escape: Escape regexp tokens in module_name, default True :type escape: bool :return: Dictionary of module info, name, size, submodules if present :rtype: dict

`autotest.client.base_utils.pickle_load` (*filename*)

`autotest.client.base_utils.ping_default_gateway` ()

Ping the default gateway.

`autotest.client.base_utils.prepend_path` (*newpath*, *oldpath*)

prepend newpath to oldpath

`autotest.client.base_utils.print_to_tty` (*string*)

Output string straight to the tty

`autotest.client.base_utils.process_is_alive` (*name_pattern*)

'pgrep name' misses all python processes and also long process names. 'pgrep -f name' gets all shell commands with name in args. So look only for command whose initial pathname ends with name. Name itself is an egrep pattern, so it can use | etc for variations.

`autotest.client.base_utils.running_config` ()

Return path of config file of the currently running kernel

`autotest.client.base_utils.running_os_full_version` ()

`autotest.client.base_utils.running_os_ident` ()

`autotest.client.base_utils.running_os_release` ()

`autotest.client.base_utils.set_power_state` (*state*)

Set the system power state to 'state'.

`autotest.client.base_utils.set_wake_alarm` (*alarm_time*)

Set the hardware RTC-based wake alarm to 'alarm_time'.

`autotest.client.base_utils.standby` ()

Power-on suspend (S1)

`autotest.client.base_utils.suspend_to_disk` ()

Suspend the system to disk (S4)

`autotest.client.base_utils.suspend_to_ram` ()

Suspend the system to RAM (S3)

`autotest.client.base_utils.sysctl` (*key*, *value*=None)

Generic implementation of sysctl, to read and write.

Parameters

- **key** – A location under /proc/sys

- **value** – If not None, a value to write into the sysctl.

Returns The single-line sysctl value as a string.

`autotest.client.base_utils.sysctl_kernel` (*key, value=None*)
 (Very) partial implementation of sysctl, for kernel params

`autotest.client.base_utils.to_seconds` (*time_string*)
 Converts a string in M+:SS.SS format to S+.SS

`autotest.client.base_utils.unload_module` (*module_name*)
 Removes a module. Handles dependencies. If even then it's not possible to remove one of the modules, it will throw an `CmdError` exception.

Parameters `module_name` (*str*) – Name of the module we want to remove.

`autotest.client.base_utils.unmap_url_cache` (*cachedir, url, expected_hash, method='md5'*)
 Downloads a file from a URL to a cache directory. If the file is already at the expected position and has the expected hash, let's not download it again.

Parameters

- **cachedir** – Directory that might hold a copy of the file we want to download.
- **url** – URL for the file we want to download.
- **expected_hash** – Hash string that we expect the file downloaded to have.
- **method** – Method used to calculate the hash string (md5, sha1).

`autotest.client.base_utils.where_art_thy_filehandles` ()
 Dump the current list of filehandles

bkr_proxy Module

`bkr_proxy` - class used to talk to beaker

class `autotest.client.bkr_proxy.BkrProxy` (*recipe_id, labc_url=None*)

Bases: `object`

`get_recipe` ()

`recipe_abort` ()

`recipe_stop` ()

`recipe_upload_file` (*localfile, remotepath=''*)

`result_upload_file` (*task_id, result_id, localfile, remotepath=''*)

`task_abort` (*task_id*)

`task_result` (*task_id, result_type, result_path, result_score, result_summary*)

`task_start` (*task_id, kill_time=0*)

`task_stop` (*task_id*)

`task_upload_file` (*task_id, localfile, remotepath=''*)

`update_watchdog` (*task_id, kill_time*)

exception `autotest.client.bkr_proxy.BkrProxyException` (*text*)

Bases: `exceptions.Exception`

`autotest.client.bkr_proxy.copy_data` (*data, dest, header=None, use_put=None*)

Copy data to a destination

To aid in debugging, copy a file locally to verify the contents. Attempts to write the same data that would otherwise be sent remotely.

Parameters

- **data** – data string to copy
- **dest** – destination path
- **header** – header info item to return
- **use_put** – dictionary of items for PUT method

Returns nothing or header info if requested

`autotest.client.bkr_proxy.copy_local` (*data, dest, use_put=None*)

Copy data locally to a file

To aid in debugging, copy a file locally to verify the contents. Attempts to write the same data that would otherwise be sent remotely.

Parameters

- **data** – encoded data string to copy locally
- **dest** – local file path
- **use_put** – chooses to write in binary or text

Returns nothing

`autotest.client.bkr_proxy.copy_remote` (*data, dest, use_put=None*)

Copy data to a remote server using http calls POST or PUT

Using http POST and PUT methods, copy data over http. To use PUT method, provide a dictionary of values to be populated in the Content-Range and Content-Length headers. Otherwise default is to use POST method.

Traps on HTTPError 500 and 400

Parameters

- **data** – encoded data string to copy remotely
- **dest** – remote server URL
- **use_put** – dictionary of items if using PUT method

Returns html header info for post processing

`autotest.client.bkr_proxy.make_path_bkr_cache` (*r*)

Converts a recipe id into an internal path for cache'ing recipe

Parameters **r** – recipe id

Returns a path to the internal recipe cache file

`autotest.client.bkr_proxy.make_path_cmdlog` (*r*)

Converts a recipe id into an internal path for logging purposes

Parameters **r** – recipe id

Returns a path to the internal command log

`autotest.client.bkr_proxy.make_path_log(r, t=None, i=None)`

Converts id into a beaker path to log file

Given a recipe id, a task id, and/or result id, translate them into the proper beaker path to the log file. Depending on which log file is needed, provide the appropriate params. Note the dependency, a result id needs a task id and recipe id, while a task id needs a recipe id.

Parameters

- **r** – recipe id
- **t** – task id
- **i** – result id

Returns a beaker path of the task's result file

`autotest.client.bkr_proxy.make_path_recipe(r)`

Converts a recipe id into a beaker path

Parameters **r** – recipe id

Returns a beaker path to the recipe id

`autotest.client.bkr_proxy.make_path_result(r, t)`

Converts task id into a beaker path to result file

Given a recipe id and a task id, translate them into the proper beaker path to the result file.

Parameters

- **r** – recipe id
- **t** – task id

Returns a beaker path of the task's result file

`autotest.client.bkr_proxy.make_path_status(r, t=None)`

Converts id into a beaker path to status file

Given a recipe id and/or a task id, translate them into the proper beaker path to the status file. Recipe only, returns the path to the recipe's status, whereas including a task returns the path to the task's status.

Parameters

- **r** – recipe id
- **t** – task id

Returns a beaker path of the recipe's/task's status file

`autotest.client.bkr_proxy.make_path_watchdog(r)`

Converts a recipe id into a beaker path for the watchdog

Parameters **r** – recipe id

Returns a beaker path of the recipe's watchdog file

bkr_xml Module

module to parse beaker xml recipe

```

class autotest.client.bkr_xml.BeakerXMLParser
    Bases: object

    Handles parsing of beaker job xml

    handle_recipe (recipe_node)

    handle_recipes (recipe_nodes)

    handle_task (recipe, task_node)

    handle_task_param (task, param_node)

    handle_task_params (task, param_nodes)

    handle_tasks (recipe, task_nodes)

    parse_from_file (file_name)

    parse_xml (xml)
        Returns dict, mapping hostname to recipe

class autotest.client.bkr_xml.Recipe
    Bases: object

class autotest.client.bkr_xml.Task
    Bases: object

    Simple record to store task properties

    get_param (key, default=None)

autotest.client.bkr_xml.xml_attr (node, key, default=None)

autotest.client.bkr_xml.xml_get_nodes (node, tag)

```

client_logging_config Module

```

class autotest.client.client_logging_config.ClientLoggingConfig (use_console=True)
    Bases: autotest.client.shared.logging_config.LoggingConfig

    add_debug_file_handlers (log_dir, log_name=None)

    configure_logging (results_dir=None, verbose=False)

```

cmdparser Module

Autotest command parser

copyright Don Zickus <dzickus@redhat.com> 2011

```

class autotest.client.cmdparser.CmdParserLoggingConfig (use_console=True)
    Bases: autotest.client.shared.logging_config.LoggingConfig

    Used with the sole purpose of providing convenient logging setup for the KVM test auxiliary programs.

    configure_logging (results_dir=None, verbose=False)

class autotest.client.cmdparser.CommandParser
    Bases: object

    A client-side command wrapper for the autotest client.

```

```
COMMAND_LIST = ['help', 'list', 'run', 'fetch', 'bootstrap']
```

bootstrap (*args, options*)

Bootstrap autotest by fetching the control file first and pass it back

Currently this relies on a harness to retrieve the file

fetch (*args, options*)

fetch a remote control file or packages

classmethod help ()

List the commands and their usage strings.

:param args is not used here.

classmethod list_tests ()

List the available tests for users to choose from

parse_args (*args, options*)

Process a client side command.

Parameters args – Command line args.

run (*args, options*)

Wrap args with a path and send it back to autotest.

common Module

config Module

The Job Configuration

The job configuration, holding configuration variable supplied to the job.

The config should be viewed as a hierarchical namespace. The elements of the hierarchy are separated by periods (.) and where multiple words are required at a level they should be separated by underscores (_). Please no StudyCaps.

For example: boot.default_args

```
class autotest.client.config.config(job)
```

Bases: `object`

The BASIC job configuration

Properties:

job The job object for this job

config The job configuration dictionary

get (*name*)

set (*name, value*)

cpuset Module

```
autotest.client.cpuset.abbrev_list(vals)
```

Condense unsigned (0,4,5,6,7,10) to '0,4-7,10'.

```
autotest.client.cpuset.all_drive_names()
```

```

autotest.client.cpuset.avail_mbytes (parent='')
autotest.client.cpuset.available_exclusive_mem_nodes (parent_container)
autotest.client.cpuset.container_bytes (name)
autotest.client.cpuset.container_exists (name)
autotest.client.cpuset.container_mbytes (name)
autotest.client.cpuset.cpus_path (container_name)
autotest.client.cpuset.cpuset_attr (container_name, attr)
autotest.client.cpuset.create_container_directly (name, mbytes, cpus)
autotest.client.cpuset.create_container_via_memcg (name, parent, bytes, cpus)
autotest.client.cpuset.create_container_with_mbytes_and_specific_cpus (name,
                                                                    mbytes,
                                                                    cpus=None,
                                                                    root='',
                                                                    io={},
                                                                    move_in=True,
                                                                    time-
                                                                    out=0)

```

Create a cpuset container and move job's current pid into it Allocate the list "cpus" of cpus to that container

Parameters

- **name** – arbitrary string tag
- **mbytes** – requested memory for job in megabytes
- **(None)** (*cpus*) – list of cpu indices to associate with the cpuset defaults to all cpus avail with given root
- **root** – the parent cpuset to nest this new set within, "" unnested top-level container
- **io** – arguments for proportional IO containers
- **(True)** (*move_in*) – Move current process into the new container now.
- **(must be 0)** (*timeout*) – persist until explicitly deleted.

```

autotest.client.cpuset.create_container_with_specific_mems_cpus (name, mems,
                                                                    cpus)
autotest.client.cpuset.delete_leftover_test_containers ()
autotest.client.cpuset.discover_container_style ()
autotest.client.cpuset.full_path (container_name)
autotest.client.cpuset.get_boot_numa ()
autotest.client.cpuset.get_cpus (container_name)
autotest.client.cpuset.get_mem_nodes (container_name)
autotest.client.cpuset.get_tasks (container_name)
autotest.client.cpuset.inner_containers_of (parent)
autotest.client.cpuset.io_attr (container_name, attr)
autotest.client.cpuset.mbytes_per_mem_node ()
autotest.client.cpuset.memory_path (container_name)

```

```
autotest.client.cpuset.mems_path(container_name)
autotest.client.cpuset.move_self_into_container(name)
autotest.client.cpuset.move_tasks_into_container(name, tasks)
autotest.client.cpuset.my_available_exclusive_mem_nodes()
autotest.client.cpuset.my_container_name()
autotest.client.cpuset.my_lock(lockname)
autotest.client.cpuset.my_mem_nodes()
autotest.client.cpuset.my_unlock(lockfile)
autotest.client.cpuset.need_fake_numa()
autotest.client.cpuset.need_mem_containers()
autotest.client.cpuset.node_avail_kbytes(node)
autotest.client.cpuset.nodes_avail_mbytes(nodes)
autotest.client.cpuset.rangelist_to_set(rangelist)
autotest.client.cpuset.release_container(container_name=None)
autotest.client.cpuset.remove_empty_prio_classes(prios)
autotest.client.cpuset.set_io_controls(container_name, disks=[], ioprio_classes=[2],
                                       io_shares=[95], io_limits=[0])
autotest.client.cpuset.tasks_path(container_name)
autotest.client.cpuset.unpath(container_path)
```

fsdev_disks Module

```
autotest.client.fsdev_disks.finish_fsdev(force_cleanup=False)
```

This method can be called from the test file to optionally restore all the drives used by the test to a standard ext2 format. Note that if `use_fsdev_lib()` was invoked with `'reinit_disks'` not set to `True`, this method does nothing. Note also that only fsdev “server-side” dynamic control files should ever set `force_cleanup` to `True`.

```
class autotest.client.fsdev_disks.fsdev_disks(job)
```

Disk drive handling class used for file system development

```
config_sched_tunables(desc_file)
```

```
get_fsdev_mgr()
```

```
load_sched_tunable_values(val_file)
```

```
set_sched_tunables(disks)
```

Given a list of disks in the format returned by `get_disk_list()` above, set the I/O scheduler values on all the disks to the values loaded earlier by `load_sched_tunables()`.

```
set_tunable(disk, name, path, val)
```

Given a disk name, a path to a tunable value under `_TUNE_PATH` and the new value for the parameter, set the value and verify that the value has been successfully set.

```
autotest.client.fsdev_disks.get_disk_list(std_mounts_only=True, get_all_disks=False)
```

Get a list of dictionaries with information about disks on this system.

Parameters

- **std_mounts_only** – Whether the function should return only disks that have a mount point defined (True) or even devices that doesn't (False).
- **get_all_disks** – Whether the function should return only partitioned disks (False) or return every disk, regardless of being partitioned or not (True).

Returns List of dictionaries with disk information (see more below).

The 'disk_list' array returned by `get_disk_list()` has an entry for each disk drive we find on the box. Each of these entries is a map with the following 3 string values:

'device' disk device name (i.e. the part after /dev/) 'mountpt' disk mount path 'tunable' disk name for setting scheduler tunables (/sys/block/sd??)

The last value is an integer that indicates the current mount status of the drive:

'mounted' 0 = not currently mounted

1 = mounted r/w on the expected path

-1 = mounted readonly or at an unexpected path

When the 'std_mounts_only' argument is True we don't include drives mounted on 'unusual' mount points in the result. If a given device is partitioned, it will return all partitions that exist on it. If it's not, it will return the device itself (ie, if there are /dev/sdb1 and /dev/sdb2, those will be returned but not /dev/sdb. if there is only a /dev/sdc, that one will be returned).

`autotest.client.fsdev_disks.match_fs (disk, dev_path, fs_type, fs_makeopt)`

Matches the user provided fs_type and fs_makeopt with the current disk.

`autotest.client.fsdev_disks.mkfs_all_disks (job, disk_list, fs_type, fs_makeopt, fs_mnt_opt)`

Prepare all the drives in 'disk_list' for testing. For each disk this means unmounting any mount points that use the disk, running mkfs with 'fs_type' as the file system type and 'fs_makeopt' as the 'mkfs' options, and finally remounting the freshly formatted drive using the flags in 'fs_mnt_opt'.

`autotest.client.fsdev_disks.prepare_disks (job, fs_desc, disk1_only=False, disk_list=None)`

Prepare drive(s) to contain the file system type / options given in the description line 'fs_desc'. When 'disk_list' is not None, we prepare all the drives in that list; otherwise we pick the first available data drive (which is usually hdc3) and prepare just that one drive.

Args:

fs_desc: A partition.FsOptions instance describing the test -OR- a

legacy string describing the same in '/' separated format: 'fstype / mkfs opts / mount opts / short name'.

disk1_only: Boolean, defaults to False. If True, only test the first disk.

disk_list: A list of disks to prepare. If None is given we default to asking `get_disk_list()`.

Returns: (mount path of the first disk, short name of the test, list of disks) OR (None, '', None) if no fs_desc was given.

`autotest.client.fsdev_disks.prepare_fsdev (job)`

Called from the test file to get the necessary drive(s) ready; return a pair of values: the absolute path to the first drive's mount point plus the complete disk list (which is useful for tests that need to use more than one drive).

`autotest.client.fsdev_disks.restore_disks (job, restore=False, disk_list=None)`

Restore ext2 on the drives in 'disk_list' if 'restore' is True; when disk_list is None, we do nothing.

`autotest.client.fsdev_disks.use_fsdev_lib (fs_desc, disk1_only, reinit_disks)`

Called from the control file to indicate that fsdev is to be used.

`autotest.client.fsdev_disks.wipe_disks` (*job, disk_list*)

Wipe all of the drives in 'disk_list' using the 'wipe' functionality in the filesystem class.

fsdev_mgr Module

This module defines the BaseFsdevManager Class which provides an implementation of the 'fsdev' helper API; site specific extensions to any of these methods should inherit this class.

class `autotest.client.fsdev_mgr.BaseFsdevManager`

Bases: `object`

check_mount_point (*part_name, mount_point*)

Parameters

- **part_name** – A partition name such as 'sda3' or similar.
- **mount_point** – A mount point such as '/usr/local' or an empty string if no mount point is known.

Returns The expected mount point for part_name or a false value (None or '') if the client should not mount this partition.

include_partition (*part_name*)

map_drive_name (*part_name*)

use_partition (*part_name*)

Parameters **part_name** – A partition name such as 'sda3' or similar.

Returns bool, should we use this partition for testing?

class `autotest.client.fsdev_mgr.FsdevManager`

Bases: `autotest.client.fsdev_mgr.BaseFsdevManager`

`autotest.client.fsdev_mgr.SiteFsdevManager`

alias of `BaseFsdevManager`

fsinfo Module

This module gives the mkfs creation options for an existing filesystem.

tune2fs or xfs_growfs is called according to the filesystem. The results, filesystem tunables, are parsed and mapped to corresponding mkfs options.

`autotest.client.fsinfo.compare_features` (*needed_feature, current_feature*)

Compare two ext* feature lists.

`autotest.client.fsinfo.convert_conf_opt` (*default_opt*)

`autotest.client.fsinfo.ext_mkfs_options` (*tune2fs_dict, mkfs_option*)

Map the tune2fs options to mkfs options.

`autotest.client.fsinfo.ext_tunables` (*dev*)

Call tune2fs -l and parse the result.

`autotest.client.fsinfo.match_ext_options` (*fs_type, dev, needed_options*)

Compare the current ext* filesystem tunables with needed ones.

`autotest.client.fsinfo.match_mkfs_option` (*fs_type, dev, needed_options*)
Compare the current filesystem tunables with needed ones.

`autotest.client.fsinfo.match_xfs_options` (*dev, needed_options*)
Compare the current ext* filesystem tunables with needed ones.

`autotest.client.fsinfo.merge_ext_features` (*conf_feature, user_feature*)

`autotest.client.fsinfo.opt_string2dict` (*opt_string*)
Breaks the mkfs.ext* option string into dictionary.

`autotest.client.fsinfo.parse_mke2fs_conf` (*fs_type, conf_file='/etc/mke2fs.conf'*)
Parses mke2fs config file for default settings.

`autotest.client.fsinfo.xfs_mkfs_options` (*tune2fs_dict, mkfs_option*)
Maps filesystem tunables to their corresponding mkfs options.

`autotest.client.fsinfo.xfs_tunables` (*dev*)
Call `xfs_grow -n` to get filesystem tunables.

harness Module

The harness interface

The interface between the client and the server when hosted.

class `autotest.client.harness.harness` (*job*)

Bases: `object`

The NULL server harness

Properties:

job The job object for this job

run_abort ()

A run within this job is aborting. It all went wrong

run_complete ()

A run within this job is completing (all done)

run_pause ()

A run within this job is completing (expect continue)

run_reboot ()

A run within this job is performing a reboot (expect continue following reboot)

run_start ()

A run within this job is starting

run_test_complete ()

A test run by this job is complete. Note that if multiple tests are run in parallel, this will only be called when all of the parallel runs complete.

setup (*job*)

job The job object for this job

test_status (*status, tag*)

A test within this job is completing

test_status_detail (*code, subdir, operation, status, tag, optional_fields*)

A test within this job is completing (detail)

`autotest.client.harness.select` (*which, job, harness_args*)

harness_autoserv Module

class `autotest.client.harness_autoserv.AutoservFetcher` (*package_manager, job_harness*)
 Bases: `autotest.client.shared.base_packages.RepositoryFetcher`
fetch_pkg_file (*filename, dest_path*)

class `autotest.client.harness_autoserv.harness_autoserv` (*job, harness_args*)
 Bases: `autotest.client.harness.harness`

The server harness for running from autoserv

Properties:

job The job object for this job

fetch_package (*pkg_name, dest_path*)
 Request a package from the remote autoserv.

Parameters

- **pkg_name** – The name of the package, as generally used by the `client.shared.packages` infrastructure.
- **dest_path** – The path the package should be copied to.

run_start ()

run_test_complete ()
 A test run by this job is complete, signal it to autoserv and wait for it to signal to continue

test_status (*status, tag*)
 A test within this job is completing

harness_beaker Module

The harness interface The interface between the client and beaker lab controller.

exception `autotest.client.harness_beaker.HarnessException` (*text*)
 Bases: `exceptions.Exception`

`autotest.client.harness_beaker.get_beaker_code` (*at_code*)

class `autotest.client.harness_beaker.harness_beaker` (*job, harness_args*)
 Bases: `autotest.client.harness.harness`

bootstrap (*fetchdir*)
 How to kickstart autotest when you have no control file? You download the beaker XML, convert it to a control file and pass it back to autotest. Much like bootstrapping.. :-)

convert_task_to_control (*fetchdir, control, task*)
 Tasks are really just: `# yum install $TEST # cd /mnt/tests/$TEST # make run`
 Convert that into a test module with a control file

find_recipe (*recipes_dict*)

get_processed_tests ()

get_recipe_from_LC ()
get_test_name (*task*)
init_recipe_from_beaker ()
init_task_params (*task*)
kill_watchdog ()
parse_args (*args, is_bootstrap*)
parse_quickcmd (*args*)
run_abort ()
 A run within this job is aborting. It all went wrong
run_complete ()
 A run within this job is completing (all done)
run_pause ()
 A run within this job is completing (expect continue)
run_reboot ()
 A run within this job is performing a reboot (expect continue following reboot)
run_start ()
 A run within this job is starting
run_test_complete ()
 A test run by this job is complete. Note that if multiple tests are run in parallel, this will only be called when all of the parallel runs complete.
setupInitSymlink ()
start_watchdog (*heartbeat*)
tear_down ()
 called from complete and abort. clean up and shutdown
test_status (*status, tag*)
 A test within this job is completing
test_status_detail (*code, subdir, operation, status, tag, optional_fields*)
 A test within this job is completing (detail)
upload_recipe_files ()
upload_result_files (*task_id, resultid, subdir*)
upload_task_files (*task_id, subdir*)
watchdog_loop (*heartbeat*)
write_processed_tests (*subdir, t_id='0'*)

harness_simple Module

The simple harness interface

class `autotest.client.harness_simple.harness_simple` (*job, harness_args*)
 Bases: `autotest.client.harness.harness`

The simple server harness

Properties:

job The job object for this job

test_status (*status, tag*)

A test within this job is completing

harness_standalone Module

The standalone harness interface

The default interface as required for the standalone reboot helper.

class `autotest.client.harness_standalone.harness_standalone` (*job, harness_args*)
 Bases: `autotest.client.harness.harness`

The standalone server harness

Properties:

job The job object for this job

job Module

The main job wrapper

This is the core infrastructure.

Copyright Andy Whitcroft, Martin J. Bligh 2006

exception `autotest.client.job.NotAvailableError`
 Bases: `autotest.client.shared.error.AutotestError`

exception `autotest.client.job.StepError`
 Bases: `autotest.client.shared.error.AutotestError`

class `autotest.client.job.base_client_job` (*control, options, drop_caches=True, extra_copy_cmdline=None*)
 Bases: `autotest.client.shared.base_job.base_job`

The client-side concrete implementation of `base_job`.

Optional properties provided by this implementation: - control - bootloader - harness

add_repository (*repo_urls*)
 Adds the repository locations to the job so that packages can be fetched from them when needed. The repository list needs to be a string list Ex: `job.add_repository(['http://blah1', 'http://blah2'])`

add_sysinfo_command (*command, logfile=None, on_every_test=False*)

add_sysinfo_logfile (*file, on_every_test=False*)

barrier (**args, **kws*)
 Create a barrier object

complete (*status*)
 Write pending TAP reports, clean up, and exit

config_get (*name*)

config_set (*name, value*)

control_get ()

control_set (*control*)

cpu_count ()

disable_external_logging ()

disable_warnings (*warning_type*)

enable_external_logging ()

enable_warnings (*warning_type*)

end_reboot (*subdir, kernel, patches, running_id=None*)

end_reboot_and_verify (*expected_when, expected_id, subdir, type='src', patches=[]*)

Check the passed kernel identifier against the command line and the running kernel, abort the job on mismatch.

filesystem (**args, **dargs*)

Same as partition

Deprecated Use partition method instead

handle_persistent_option (*options, option_name*)

Select option from command line or persistent state. Store selected option to allow standalone client to continue after reboot with previously selected options. Priority: 1. explicitly specified via command line 2. stored in state file (if continuing job '-c') 3. default is None

harness_select (*which, harness_args*)

install_pkg (*name, pkg_type, install_dir*)

This method is a simple wrapper around the actual package installation method in the Packager class. This is used internally by the profilers, deps and tests code.

Parameters

- **name** – name of the package (ex: sleeptest, dbench etc.)
- **pkg_type** – Type of the package (ex: test, dep etc.)
- **install_dir** – The directory in which the source is actually

untarred into. (ex: client/profilers/<name> for profilers)

kernel (*base_tree, results_dir='', tmp_dir='', leave=False*)

Summon a kernel object

monitor_disk_usage (*max_rate*)

Signal that the job should monitor disk space usage on / and generate a warning if a test uses up disk space at a rate exceeding 'max_rate'.

Parameters:

max_rate - the maximum allowed rate of disk consumption during a test, in MB/hour, or 0 to indicate no limit.

next_step (*fn, *args, **dargs*)

Create a new step and place it after any steps added while running the current step but before any steps added in previous steps

next_step_append (*fn, *args, **dargs*)

Define the next step and place it at the end

next_step_prepend (*fn, *args, **dargs*)

Insert a new step, executing first

noop (*text*)

parallel (**args, **dargs*)

Run tasks in parallel

partition (*device, loop_size=0, mountpoint=None*)

Work with a machine partition

param device e.g. /dev/sda2, /dev/sdb1 etc...

param mountpoint Specify a directory to mount to. If not specified autotest tmp directory will be used.

param loop_size Size of loopback device (in MB). Defaults to 0.

return A L{client.partition.partition} object

quit ()

reboot (*tag=<object object>*)

reboot_setup ()

relative_path (*path*)

Return a patch relative to the job results directory

require_gcc ()

Test whether gcc is installed on the machine.

run_group (*function, tag=None, **dargs*)

Run a function nested within a group level.

Parameters

- **function** – Callable to run.
- **tag** – An optional tag name for the group. If None (default)

function.__name__ will be used. :param dargs: Named arguments for the function.

run_test (**args, **dargs*)

Summon a test object and run it.

:param url A url that identifies the test to run. :param tag An optional keyword argument that will be added to the test and subdir name. :param subdir_tag An optional keyword argument that will be added to the subdir name.

Returns True if the test passes, False otherwise.

run_test_detail (**args, **dargs*)

Summon a test object and run it, returning test status.

:param url A url that identifies the test to run. :param tag An optional keyword argument that will be added to the test and subdir name. :param subdir_tag An optional keyword argument that will be added to the subdir name.

Returns Test status

See client/shared/error.py, exit_status

setup_dep (*deps*)

Set up the dependencies for this test. deps is a list of libraries required for this test.

setup_dirs (*results_dir, tmp_dir*)

start_reboot ()

step_engine ()

The multi-run engine used when the control file defines step_init.

Does the next step.

xen (*base_tree, results_dir='', tmp_dir='', leave=False, kjob=None*)

Summon a xen object

class `autotest.client.job.disk_usage_monitor` (*logging_func, device, max_mb_per_hour*)

start ()

stop ()

classmethod watch (**monitor_args, **monitor_dargs*)

Generic decorator to wrap a function call with the standard create-monitor -> start -> call -> stop idiom.

class `autotest.client.job.job` (*control, options, drop_caches=True, extra_copy_cmdline=None*)

Bases: `autotest.client.job.base_client_job`

`autotest.client.job.runjob` (*control, drop_caches, options*)

Run a job using the given control file.

This is the main interface to this module.

See `base_job.__init__` for parameter info.

`autotest.client.job.site_job`

alias of `base_client_job`

class `autotest.client.job.status_indenter` (*job*)

Bases: `autotest.client.shared.base_job.status_indenter`

Provide a status indenter that is backed by `job._record_prefix`.

decrement ()

increment ()

indent

kernel Module

class `autotest.client.kernel.BootableKernel` (*job*)

Bases: `object`

add_to_bootloader (*args=''*)

`autotest.client.kernel.auto_kernel` (*job, path, subdir, tmp_dir, build_dir, leave=False*)

Create a kernel object, dynamically selecting the appropriate class to use based on the path provided.

class `autotest.client.kernel.kernel` (*job, base_tree, subdir, tmp_dir, build_dir, leave=False*)

Bases: `autotest.client.kernel.BootableKernel`

Class for compiling kernels.

Data for the object includes the src files used to create the kernel, patches applied, config (base + changes), the build directory itself, and logged output

Properties:

job Backpointer to the job object we're part of

autodir Path to the top level autotest dir (see `global_config.ini`, session COMMON/autotest_top_path)

src_dir <tmp_dir>/src/

build_dir <tmp_dir>/linux/

config_dir <results_dir>/config/

log_dir <results_dir>/debug/

results_dir <results_dir>/results/

apply_patches (*local_patches*)
 apply the list of patches, in order

autodir = ''

boot (*args*='', *ident*=True)
 install and boot this kernel, do not care how just make it happen.

build (**args*, ***dargs*)

build_timed (*threads*, *timefile*='/dev/null', *make_opts*='', *output*='/dev/null')
 time the building of the kernel

clean (**args*, ***dargs*)

config (**args*, ***dargs*)

extract (**args*, ***dargs*)

extraversion (*tag*, *append*=True)

get_kernel_build_arch (*arch*=None)
 Work out the current kernel architecture (as a kernel arch)

get_kernel_build_ident ()

get_kernel_build_release ()

get_kernel_build_ver ()
 Check Makefile and .config to return kernel version

get_kernel_tree (*base_tree*)
 Extract/link base_tree to self.build_dir

get_patches (*patches*)
 fetch the patches to the local src_dir

install (**args*, ***dargs*)

kernelexpand (*kernel*)

mkinitrd (**args*, ***dargs*)

patch (**args*, ***dargs*)

pickle_dump (*filename*)
 dump a pickle of ourself out to the specified filename
 we can't pickle the backreference to job (it contains fd's), nor would we want to. Same for logfile (fd's).

set_build_image (*image*)

set_build_target (*build_target*)

set_cross_cc (*target_arch=None, cross_compile=None, build_target='bzImage'*)

Set up to cross-compile. This is broken. We need to work out what the default compile produces, and if not, THEN set the cross compiler.

`autotest.client.kernel.preprocess_path` (*path*)

class `autotest.client.kernel.rpm_kernel` (*job, rpm_package, subdir*)

Bases: `autotest.client.kernel.BootableKernel`

Class for installing a binary rpm kernel package

boot (*args='', ident=True*)

install and boot this kernel

build (**args, **dargs*)

Dummy function, binary kernel so nothing to build.

install (**args, **dargs*)

kernel_string = `'/boot/vmlinuz'`

class `autotest.client.kernel.rpm_kernel_suse` (*job, rpm_package, subdir*)

Bases: `autotest.client.kernel.rpm_kernel`

Class for installing openSUSE/SLE rpm kernel package

add_to_bootloader (*args=''*)

Set parameters of this kernel in bootloader

install ()

kernel_string = `'/boot/vmlinuz'`

`autotest.client.kernel.rpm_kernel_vendor` (*job, rpm_package, subdir*)

class `autotest.client.kernel.srpm_kernel` (*job, rpm_package, subdir*)

Bases: `autotest.client.kernel.kernel`

apply_patches (*local_patches*)

binrpm_pattern = `<_sre.SRE_Pattern object>`

boot (*args=''*)

build (*tag='autotest'*)

config (**args, **kwargs*)

consume_one_config (*config_option*)

finish_init ()

install (*tag='autotest'*)

prefix = `'/root/rpmbuild'`

prep (*tag='autotest'*)

setup_source ()

update_spec (*tag*)

update_spec_line (*line, outspec, tag*)

class `autotest.client.kernel.srpm_kernel_suse` (*job, rpm_package, subdir*)

Bases: `autotest.client.kernel.srpm_kernel`

finish_init ()

`prefix = '/usr/src/packages'`

`setup_source ()`

`update_spec_line (line, outspec, tag)`

`autotest.client.kernel.srpm_kernel_vendor (job, rpm_package, subdir)`

`autotest.client.kernel.tee_output_logdir_mark (fn)`

kernel_config Module

`autotest.client.kernel_config.apply_overrides (orig_file, changes_file, output_file)`

`autotest.client.kernel_config.config_by_name (name, s)`

`autotest.client.kernel_config.diff_configs (old, new)`

`autotest.client.kernel_config.feature_enabled (feature, config)`

Verify whether a given kernel option is enabled.

Parameters

- **feature** – Kernel feature, such as “CONFIG_DEFAULT_UIMAGE”.
- **config** – Config file path, such as /tmp/config.

class `autotest.client.kernel_config.kernel_config (job, build_dir, config_dir, orig_file, overrides, defconfig=False, name=None, make=None)`

Bases: `object`

Build directory must be ready before init'ing config.

Stages:

1. Get original config file
2. Apply overrides
3. **Do 'make oldconfig' to update it to current source code** (gets done implicitly during the process)

You may specify the defconfig within the tree to build, or a custom config file you want, or None, to get machine's default config file from the repo.

config_record (name)

Copy the current .config file to the config.<name>[.<n>]

update_config (old_config, new_config=None)

`autotest.client.kernel_config.modules_needed (config)`

kernel_versions Module

`autotest.client.kernel_versions.is_release_candidate (version)`

`autotest.client.kernel_versions.is_released_kernel (version)`

`autotest.client.kernel_versions.version_choose_config (version, candidates)`

`autotest.client.kernel_versions.version_encode (version)`

`autotest.client.kernel_versions.version_len (version)`

`autotest.client.kernel_versions.version_limit` (*version, n*)

kernelexpand Module

Program and API used to expand kernel versions, trying to match them with the URL of the correspondent package on kernel.org or a mirror. Example:

```
$ ./kernelexpand.py 3.1 http://www.kernel.org/pub/linux/kernel/v3.x/linux-3.1.tar.bz2
```

author Andy Whitcroft (apw@shadowen.org)

copyright IBM 2008

license GPL v2

see Inspired by kernelexpand by Martin J. Bligh, 2003

`autotest.client.kernelexpand.decompose_kernel` (*kernel*)

`autotest.client.kernelexpand.decompose_kernel_2x_once` (*kernel*)

Generate the parameters for the patches (2.X version):

full => full kernel name base => all but the matches suffix minor => 2.n.m major => 2.n minor-prev => 2.n.m-1

Parameters `kernel` – String representing a kernel version to be expanded.

`autotest.client.kernelexpand.decompose_kernel_post_2x_once` (*kernel*)

Generate the parameters for the patches (post 2.X version):

full => full kernel name base => all but the matches suffix minor => o.n.m major => o.n minor-prev => o.n.m-1

Parameters `kernel` – String representing a kernel version to be expanded.

`autotest.client.kernelexpand.expand_classic` (*kernel, mirrors*)

`autotest.client.kernelexpand.get_mappings_2x` ()

`autotest.client.kernelexpand.get_mappings_post_2x` ()

`autotest.client.kernelexpand.mirror_kernel_components` (*mirrors, components*)

`autotest.client.kernelexpand.select_kernel_components` (*components*)

`autotest.client.kernelexpand.url_accessible` (*url*)

kvm_control Module

Utilities useful to client control files that test KVM.

`autotest.client.kvm_control.get_kvm_arch` ()

Get the kvm kernel module to be loaded based on the CPU architecture

Raises `error.TestError` if no vendor name or cpu flags are found

Returns 'kvm_amd' or 'kvm_intel' or 'kvm_power7'

Return type *string*

`autotest.client.kvm_control.load_kvm` ()

Loads the appropriate KVM kernel modules depending on the current CPU architecture

Returns 0 on success or 1 on failure

Return type *int*

`autotest.client.kvm_control.unload_kvm()`
Unloads the current KVM kernel modules (if loaded)

Returns 0 on success or 1 on failure

Return type *int*

local_host Module

This file contains the implementation of a host object for the local machine.

class `autotest.client.local_host.LocalHost (*args, **dargs)`
Bases: `autotest.client.shared.hosts.base_classes.Host`

list_files_glob (*path_glob*)

Get a list of files on a remote host given a glob pattern path.

run (*command, timeout=3600, ignore_status=False, stdout_tee=<object object>, stderr_tee=<object object>, stdin=None, args=()*)

See `shared.hosts.Host.run()`

symlink_closure (*paths*)

Given a sequence of path strings, return the set of all paths that can be reached from the initial set by following symlinks.

Parameters *paths* – sequence of path strings.

Returns a sequence of path strings that are all the unique paths that can be reached from the given ones after following symlinks.

wait_up (*timeout=None*)

lv_utils Module

Utility for taking shapshots from existing logical volumes or creates such.

author Plamen Dimitrov

copyright Intra2net AG 2012

license GPL v2

param vg_name Name of the volume group.

param lv_name Name of the logical volume.

param lv_size Size of the logical volume as string in the form “#G” (for example 30G).

param lv_snapshot_name Name of the snapshot with origin the logical volume.

param lv_snapshot_size Size of the snapshot with origin the logical volume also as “#G”.

param ramdisk_vg_size Size of the ramdisk virtual group.

param ramdisk_basedir Base directory for the ramdisk sparse file.

param ramdisk_sparse_filename Name of the ramdisk sparse file.

Sample ramdisk params: - ramdisk_vg_size = “40000” - ramdisk_basedir = “/tmp” - ramdisk_sparse_filename = “virtual_hdd”

Sample general params: - vg_name='autotest_vg', - lv_name='autotest_lv', - lv_size='1G', - lv_snapshot_name='autotest_sn', - lv_snapshot_size='1G' The ramdisk volume group size is in MB.

`autotest.client.lv_utils.lv_check(vg_name, lv_name)`

Check whether provided logical volume exists.

`autotest.client.lv_utils.lv_list(vg_name)`

`autotest.client.lv_utils.lv_list_all()`

List available group volumes.

`autotest.client.lv_utils.thin_lv_create(vg_name, thinpool_name='lvthinpool', thinpool_size='1.5G', thinlv_name='lvthin', thinlv_size='1G')`

Create a thin volume from given volume group.

Parameters

- **vg_name** – An exist volume group
- **thinpool_name** – The name of thin pool
- **thinpool_size** – The size of thin pool to be created
- **thinlv_name** – The name of thin volume
- **thinlv_size** – The size of thin volume

`autotest.client.lv_utils.vg_check(vg_name)`

Check whether provided volume group exists.

`autotest.client.lv_utils.vg_list()`

List available volume groups.

`autotest.client.lv_utils.vg_ramdisk_cleanup(ramdisk_filename=None, vg_ramdisk_dir=None, vg_name=None, loop_device=None, use_tmpps=True)`

Inline cleanup function in case of test error.

optparser Module

Autotest client/local option parser

class `autotest.client.optparser.AutotestLocalOptionParser`

Bases: `optparse.OptionParser`

Default autotest option parser

os_dep Module

class `autotest.client.os_dep.Ldconfig`

Bases: `object`

class `DirEntry(path, flag, ino, dev)`

Bases: `object`

`Ldconfig.LD_SO_CONF = '/etc/ld.so.conf'`

Ldconfig.**MAX_RECURSION_DEPTH** = 20

Ldconfig.**ldconfig** (*ld_so_conf_filename*='/etc/ld.so.conf', *extra_dirs*=('/lib', '/usr/lib', '/lib64',
 '/usr/lib64', '/lib/tls', '/usr/lib/tls', '/lib64/tls', '/usr/lib64/tls'))

Read and parse /etc/ld.so.conf to generate a list of directories that ldconfig would search. Pre-seed the search directory list with ('/lib', '/usr/lib', '/lib64', '/usr/lib64')

Parameters

- **ld_so_conf_filename** (*str*) – path to /etc/ld.so.conf
- **extra_dirs** (*iterable*) –

Returns iterator over the directories found

Return type iterable

Ldconfig.**parse_conf** (*filename*='/etc/ld.so.conf', *recursion*=0)

autotest.client.os_dep.**command** (*target*, **args*, ***kwargs*)

Find a program by searching in the environment path and in common binary paths.

check both if it is a file and executable *which* always returns the abspath return "" if failure because "" is well-defined NULL path, so it is better than None or ValueError

Parameters

- **program** (*str*) – command name or path to command
- **extra_dirs** (*iterable*) – iterable of extra paths to search

Returns abspath of command if found

Return type str

Raises **ValueError** – when program not found

autotest.client.os_dep.**commands** (**cmds*)

autotest.client.os_dep.**exception_when_false_wrapper** (*func*, *exception_class*,
 value_error_message_template)

Wrap a function to raise an exception when the return value is not True.

Parameters

- **func** (*function*) – function to wrap
- **exception_class** (*Exception*) – exception class to raise
- **value_error_message_template** (*str*) – string to pass to exception

Returns wrapped function

Return type function

Raises **exception_class** – when func returns not true

autotest.client.os_dep.**generate_bin_search_paths** (*program*, *extra_dirs*)

Generate full paths of potential locations of a given binary file based on COMMON_BIN_PATHS.

Use the environment variable \$PATH seed the list of search directories.

Parameters

- **program** (*str*) – library filename to join with all search directories
- **extra_dirs** (*str*) – extra directories to append to the directory search list

Returns iterator over all generated paths

Return type `iter`

`autotest.client.os_dep.generate_include_search_paths` (*hdr*, *extra_dirs*)

Generate full paths of potential locations of a given header file based on `COMMON_HEADER_PATHS`.

Parameters

- **hdr** (*str*) – header filename to join with all search directories
- **extra_dirs** (*iterable*) – extra directories to append to the directory search list

Returns iterator over all generated paths

Return type `iterable`

`autotest.client.os_dep.generate_library_search_paths` (*lib*, *extra_dirs*=('/lib',
'/usr/lib', '/lib64', '/usr/lib64',
'/lib/tls', '/usr/lib/tls',
'/lib64/tls', '/usr/lib64/tls'),
ld_so_conf_filename='/etc/ld.so.conf')

Generate full paths of potential locations of a given library file based on `COMMON_LIB_PATHS`.

Parameters

- **lib** (*str*) – library filename to join with all search directories
- **extra_dirs** (*iterable*) – extra directories to append to the directory search list
- **ld_so_conf_filename** (*str*) – location of `/etc/ld.so.conf` to parse to find all system library locations

Returns iterator over all generated paths

Return type `iterable`

`autotest.client.os_dep.header` (*target*, **args*, ***kwargs*)

Find a header file by searching in the common include search paths, (`'/usr/include'`, `'/usr/local/include'`)

Check both if the header is a file and readable.

Parameters

- **hdr** (*str*) – header file or path to header file, e.g. `stdio.h`
- **extra_dirs** (*iterable*) – iterable of extra paths to search

Returns abspath of header if found

Return type `str`

Raises `ValueError` – when header is not found

`autotest.client.os_dep.headers` (**hdrs*)

`autotest.client.os_dep.is_file_and_readable` (*pth*)

Parameters *pth* – path to check

Returns true if the path is a file and `R_OK`

Return type `bool`

`autotest.client.os_dep.is_file_and_rx` (*pth*)

Parameters *pth* – path to check

Returns true if the path is a file and `R_OK` & `X_OK`

Return type `bool`

`autotest.client.os_dep.libraries` (*libs)

`autotest.client.os_dep.library` (target, *args, **kwargs)

Find a library file by parsing `/etc/ld.so.conf` and also searching in the common library search paths, (`'/lib'`, `'/usr/lib'`, `'/lib64'`, `'/usr/lib64'`, `'/lib/tls'`, `'/usr/lib/tls'`, `'/lib64/tls'`, `'/usr/lib64/tls'`)

Check both if the library is a file and readable.

Parameters

- **lib** (*str*) – library file or path to library file, e.g. `libc.so.6`
- **extra_dirs** (*iterable*) – iterable of extra paths to search

Returns abspath of library if found

Return type `str`

Raises `ValueError` – when library is not found

`autotest.client.os_dep.make_path_searcher` (path_generator, target_predicate, target_normalizer, extra_paths, **kwargs)

Universal search function generator using lazy evaluation.

Generate a function that will iterate over all the paths from `path_generator` using `target_predicate` to filter matching paths. Each matching path is then normalized by `target_predicate`. Only the first match is returned.

Parameters

- **path_generator** (*iterator*) – all paths to test with `target_predicate`
- **target_predicate** (*function*) – boolean function that tests a given path
- **target_normalizer** (*function*) – function that transforms a matching path to some normalized form
- **extra_paths** (*iterator*) – extra paths to pass to the `path_generator`

Returns the path searching function

Return type `function`

`autotest.client.os_dep.path_joiner` (target, search_paths)

Create a generator that joins target to each search path

Parameters

- **target** (*str*) – filename to join to each search path
- **search_paths** (*iterator*) – iterator over all the search paths

Returns iterator over all the joined paths

Return type `iterator`

`autotest.client.os_dep.unique_not_false_list` (arg_paths)

`autotest.client.os_dep.which` (target, extra_dirs=(`'/usr/libexec'`, `'/usr/local/sbin'`, `'/usr/local/bin'`, `'/usr/sbin'`, `'/usr/bin'`, `'/sbin'`, `'/bin'`))

Find a program by searching in the environment path and in common binary paths.

check both if it is a file and executable *which* always returns the abspath return "" if failure because "" is well-defined NULL path, so it is better than None or ValueError

Parameters

- **program** (*str*) – command name or path to command
- **extra_dirs** (*iterble*) – iterable of extra paths to search

Returns abspath of command if found, else ''

Return type `str`

`autotest.client.os_dep.which_header(target, extra_dirs=frozenset([]))`

Find a header file by searching in the common include search paths, ('/usr/include', '/usr/local/include')

Check both if the header is a file and readable.

Parameters

- **hdr** (`str`) – header file or path to header file, e.g. `stdio.h`
- **extra_dirs** (`iterable`) – iterable of extra paths to search

Returns abspath of header if found, else ''

Return type `str`

`autotest.client.os_dep.which_library(target, extra_dirs=('/lib', '/usr/lib', '/lib64',
'/usr/lib64', '/lib/tls', '/usr/lib/tls', '/lib64/tls',
'/usr/lib64/tls'))`

Find a library file by parsing `/etc/ld.so.conf` and also searching in the common library search paths, ('/lib', '/usr/lib', '/lib64', '/usr/lib64', '/lib/tls', '/usr/lib/tls', '/lib64/tls', '/usr/lib64/tls')

Check both if the library is a file and readable.

Parameters

- **lib** (`str`) – library file or path to library file, e.g. `libc.so.6`
- **extra_dirs** (`iterable`) – iterable of extra paths to search

Returns abspath of library if found, else ''

Return type `str`

parallel Module

Parallel execution management

`autotest.client.parallel.fork_nuke_subprocess(tmp, pid)`

`autotest.client.parallel.fork_start(tmp, l)`

`autotest.client.parallel.fork_waitfor(tmp, pid)`

`autotest.client.parallel.fork_waitfor_timed(tmp, pid, timeout)`

Waits for pid until it terminates or timeout expires. If timeout expires, test subprocess is killed.

partition Module

APIs to write tests and control files that handle partition creation, deletion and formatting.

copyright Google 2006-2008

author Martin Bligh (mbligh@google.com)

class `autotest.client.partition.FsOptions` (`fstype`, `fs_tag`, `mkfs_flags=None`,
`mount_options=None`)

Bases: `object`

A class encapsulating a filesystem test's parameters.

fs_tag

fstype

mkfs_flags

mount_options

`autotest.client.partition.filesystems()`

Return a list of all available filesystems

`autotest.client.partition.filter_partition_list(partitions, devnames)`

Pick and choose which partition to keep.

`filter_partition_list` accepts a list of partition objects and a list of strings. If a partition has the device name of the strings it is returned in a list.

Parameters

- **partitions** – A list of L{partition} objects
- **devnames** – A list of devnames of the form '/dev/hdc3' that specifies which partitions to include in the returned list.

Returns A list of L{partition} objects specified by devnames, in the order devnames specified

`autotest.client.partition.get_iosched_path(device_name, component)`

`autotest.client.partition.get_mount_info(partition_list)`

Picks up mount point information about the machine mounts. By default, we try to associate mount points with UUIDs, because in newer distros the partitions are uniquely identified using them.

`autotest.client.partition.get_partition_list(job, min_blocks=0, filter_func=None, exclude_swap=True, open_func=<built-in function open>)`

Get a list of partition objects for all disk partitions on the system.

Loopback devices and unnumbered (whole disk) devices are always excluded.

Parameters

- **job** – The job instance to pass to the partition object constructor.
- **min_blocks** – The minimum number of blocks for a partition to be considered.
- **filter_func** – A callable that returns True if a partition is desired. It will be passed one parameter: The partition name (hdc3, etc.). Some useful filter functions are already defined in this module.
- **exclude_swap** – If True any partition actively in use as a swap device will be excluded.
- **__open** – Reserved for unit testing.

Returns A list of L{partition} objects.

`autotest.client.partition.get_unmounted_partition_list(root_part, job=None, min_blocks=0, filter_func=None, exclude_swap=True, open_func=<built-in function open>)`

Return a list of partition objects that are not mounted.

Parameters

- **root_part** – The root device name (without the `/dev/` prefix, example `'hda2'`) that will be filtered from the partition list.

Reasoning: in Linux `/proc/mounts` will never directly mention the root partition as being mounted on `/` instead it will say that `/dev/root` is mounted on `/`. Thus require this argument to filter out the `root_part` from the ones checked to be mounted.

- **min_blocks**, **filter_func**, **exclude_swap**, **open_func** (*job,*) – Forwarded to `get_partition_list()`.

Returns List of `L{partition}` objects that are not mounted.

`autotest.client.partition.is_linux_fs_type(device)`

Checks if specified partition is type 83

Parameters **device** – the device, e.g. `/dev/sda3`

Returns False if the supplied partition name is not type 83 linux, True otherwise

`autotest.client.partition.is_valid_disk(device)`

Checks if a disk is valid

Parameters **device** – e.g. `/dev/sda`, `/dev/hda`

`autotest.client.partition.is_valid_partition(device)`

Checks if a partition is valid

Parameters **device** – e.g. `/dev/sda1`, `/dev/hda1`

`autotest.client.partition.list_mount_devices()`

`autotest.client.partition.list_mount_points()`

`autotest.client.partition.parallel(partitions, method_name, *args, **dargs)`

Run a partition method (with appropriate arguments) in parallel, across a list of partition objects

class `autotest.client.partition.partition(job, device, loop_size=0, mountpoint=None)`

Bases: `object`

Class for handling partitions and filesystems

fsck (*args='-fy', record=True*)

Run filesystem check

Parameters **args** – arguments to filesystem check tool. Default is `"-n"` which works on most tools.

get_fsck_exec ()

Return the proper mkfs executable based on self.fstype

get_io_scheduler (*device_name*)

get_io_scheduler_list (*device_name*)

get_mountpoint (*open_func=<built-in function open>, filename=None*)

Find the mount point of this partition object.

Parameters

- **open_func** – the function to use for opening the file containing the mounted partitions information
- **filename** – where to look for the mounted partitions information (default `None` which means it will search `/proc/mounts` and/or `/etc/mtab`)

Returns a string with the mount point of the partition or `None` if not mounted

mkfs (*fstype=None, args='', record=True*)
 Format a partition to filesystem type

Parameters

- **fstype** – the filesystem type, e.g.. “ext3”, “ext2”
- **args** – arguments to be passed to mkfs command.
- **record** – if set, output result of mkfs operation to autotest output

mkfs_exec (*fstype*)
 Return the proper mkfs executable based on fs

mount (*mountpoint=None, fstype=None, args='', record=True*)
 Mount this partition to a mount point

Parameters

- **mountpoint** – If you have not provided a mountpoint to partition object or want to use a different one, you may specify it here.
- **fstype** – Filesystem type. If not provided partition object value will be used.
- **args** – Arguments to be passed to “mount” command.
- **record** – If True, output result of mount operation to autotest output.

run_test (*test, **dargs*)

run_test_on_partition (*test, mountpoint_func, **dargs*)
 Executes a test fs-style (umount,mkfs,mount,test)

Here we unmarshal the args to set up tags before running the test. Tests are also run by first unmounting, mkfsing and then mounting before executing the test.

Parameters

- **test** – name of test to run
- **mountpoint_func** – function to return mount point string

set_fs_options (*fs_options*)
 Set filesystem options

param fs_options A L{FsOptions} object

set_io_scheduler (*device_name, name*)

setup_before_test (*mountpoint_func*)

Prepare a partition for running a test. Unmounts any filesystem that’s currently mounted on the partition, makes a new filesystem (according to this partition’s filesystem options) and mounts it where directed by mountpoint_func.

Parameters mountpoint_func – A callable that returns a path as a string, given a partition instance.

umount (*ignore_status=False, record=True*)
 Umount this partition.

It’s easier said than done to umount a partition. We need to lock the mtab file to make sure we don’t have any locking problems if we are unmounting in parallel.

If there turns out to be a problem with the simple umount we end up calling amount_force to get more aggressive.

Parameters

- **ignore_status** – should we notice the umount status
- **record** – if True, output result of umount operation to autotest output

unmount_force()

Kill all other jobs accessing this partition. Use fuser and ps to find all mounts on this mountpoint and unmount them.

Returns true for success or false for any errors

wipe()

Delete all files of a given partition filesystem.

`autotest.client.partition.partitionname_to_device(part)`

Converts a partition name to its associated device

`autotest.client.partition.run_test_on_partitions(job, test, partitions, mountpoint_func, tag, fs_opt, do_fsck=True, **dargs)`

Run a test that requires multiple partitions. Filesystems will be made on the partitions and mounted, then the test will run, then the filesystems will be unmounted and optionally fsck'd.

Parameters

- **job** – A job instance to run the test
- **test** – A string containing the name of the test
- **partitions** – A list of partition objects, these are passed to the test as `partitions=`
- **mountpoint_func** – A callable that returns a mountpoint given a partition instance
- **tag** – A string tag to make this test unique (Required for control files that make multiple calls to this routine with the same value of 'test'.)
- **fs_opt** – An FsOptions instance that describes what filesystem to make
- **do_fsck** – include fsck in post-test partition cleanup.
- **dargs** – Dictionary of arguments to be passed to `job.run_test()` and eventually the test

`autotest.client.partition.unmount_partition(device)`

Unmount a mounted partition

Parameters **device** – e.g. `/dev/sda1`, `/dev/hda1`

class `autotest.client.partition.virtual_partition(file_img, file_size)`

Handles block device emulation using file images of disks. It's important to note that this API can be used only if we have the following programs present on the client machine:

- `sfdisk`
- `losetup`
- `kpartx`

destroy()

Removes the virtual partition from `/dev/mapper`, detaches the image file from the loopback device and removes the image file.

`autotest.client.partition.wipe_filesystem(job, mountpoint)`

profiler Module

class `autotest.client.profiler.profiler` (*job*)

initialize (**args, **dargs*)

preserve_srcdir = `False`

report (*test*)

setup (**args, **dargs*)

start (*test*)

stop (*test*)

supports_reboot = `False`

setup Module

`autotest.client.setup.get_data_files` ()

`autotest.client.setup.get_filelist` ()

`autotest.client.setup.get_package_data` ()

`autotest.client.setup.get_package_dir` ()

`autotest.client.setup.get_packages` ()

`autotest.client.setup.get_scripts` ()

`autotest.client.setup.run` ()

setup_job Module

`autotest.client.setup_job.init_test` (*options, testdir*)

Instantiate a client test object from a given test directory.

:param options Command line options passed in to instantiate a `setup_job` which associates with this test.

:param testdir The test directory. **:return:** A test object or `None` if failed to instantiate.

`autotest.client.setup_job.load_all_client_tests` (*options*)

Load and instantiate all client tests.

This function is inspired from `runtest()` on `client/shared/test.py`.

Parameters options – an object passed in from command line `OptionParser`. See all options defined on `client/autotest`.

Returns a tuple containing the list of all instantiated tests and a list of tests that failed to instantiate.

class `autotest.client.setup_job.setup_job` (*options*)

Bases: `autotest.client.job.job`

`setup_job` is a job which runs client test `setup()` method at server side.

This job is used to pre-setup client tests when development toolchain is not available at client.

`autotest.client.setup_job.setup_test` (*client_test*)
Direct invoke test.setup() method.

Returns A boolean to represent success or not.

`autotest.client.setup_job.setup_tests` (*options*)
Load and instantiate all client tests.

This function is inspired from `runtest()` on `client/shared/test.py`.

Parameters *options* – an object passed in from command line `OptionParser`. See all options defined on `client/autotest`.

setup_modules Module

Module used to create the autotest namespace for single dir use case.

Autotest programs can be used and developed without requiring it to be installed system-wide. In order for the code to see the library namespace:

```
from autotest.client.shared import error from autotest.server import hosts ...
```

Without system wide install, we need some hacks, that are performed here.

author John Admanski (jadmanski@google.com)

`autotest.client.setup_modules.import_module` (*module, from_where*)
Equivalent to ‘`from from_where import module`’.

Parameters

- **module** – Module name.
- **from_where** – Package from where the module is being imported.

Returns The corresponding module.

`autotest.client.setup_modules.setup` (*base_path, root_module_name='autotest'*)
Setup a library namespace, with the appropriate top root module name.

Perform all the necessary setup so that all the packages at ‘`base_path`’ can be imported via “`import root_module_name.package`”.

Parameters

- **base_path** – Base path for the module.
- **root_module_name** – Top level name for the module.

sysinfo Module

test Module

`autotest.client.test.runtest` (*job, url, tag, args, dargs*)

class `autotest.client.test.test` (*job, bindir, outputdir*)
Bases: `autotest.client.shared.test.base_test`
`configure_crash_handler()`

Configure the crash handler by:

- Setting up core size to unlimited
- Putting an appropriate crash handler on `/proc/sys/kernel/core_pattern`
- Create files that the crash handler will use to figure which tests are active at a given moment

The crash handler will pick up the core file and write it to `self.debugdir`, and perform analysis on it to generate a report. The program also outputs some results to `syslog`.

If multiple tests are running, an attempt to verify if we still have the old PID on the system process table to determine whether it is a parent of the current test execution. If we can't determine it, the core file and the report file will be copied to all test debug dirs.

crash_handler_report ()

If core dumps are found on the `debugdir` after the execution of the test, let the user know.

test_config Module

Wrapper around `ConfigParser` to manage testcases configuration.

author `rsalveti@linux.vnet.ibm.com` (Ricardo Salveti de Araujo)

class `autotest.client.test_config.config_loader` (*cfg, tmpdir='/tmp', raise_errors=False*)

Base class of the configuration parser

check (*section*)

Check if the config file has valid values

check_parameter (*param_type, parameter*)

Check if a option has a valid value

get (*section, option, default=None*)

Get the value of a option.

Section of the config file and the option name. You can pass a default value if the option doesn't exist.

Parameters

- **section** – Configuration file section.
- **option** – Option we're looking after.

Default In case the option is not available and `raise_errors` is set to `False`, return the default.

remove (*section, option*)

Remove an option.

save ()

Save the configuration file with all modifications

set (*section, option, value*)

Set an option.

This change is not persistent unless saved with `'save()'`.

utils Module

Convenience functions for use by tests or whomever.

NOTE: this is a mixin library that pulls in functions from several places Note carefully what the precedence order is There's no really good way to do this, as this isn't a class we can do inheritance with, just a collection of static methods.

xen Module

```
class autotest.client.xen.xen(job, base_tree, results_dir, tmp_dir, build_dir, leave=False,
                             kjob=None)
    Bases: autotest.client.kernel.kernel

    add_to_bootloader(tag='autotest', args='')
        add this kernel to bootloader, taking an optional parameter of space separated parameters e.g.: kernel.add_to_bootloader('mykernel', 'ro acpi=off')

    build(make_opts='', logfile='', extraversion='autotest')
        build xen

        make_opts additional options to make, if any

    build_timed(*args, **kws)

    config(config_file, config_list=None)

    fix_up_xen_kernel_makefile(kernel_dir)
        Fix up broken EXTRAVERSION in xen-ified Linux kernel Makefile

    get_xen_build_ver()
        Check Makefile and .config to return kernel version

    get_xen_kernel_build_ver()
        Check xen buildconfig for current kernel version

    install(tag='', prefix='', extraversion='autotest')
        make install in the kernel tree

    log(msg)
```

Subpackages

net Package

basic_machine Module

common Module

net_tc Module

Convenience methods for use to manipulate traffic control settings.

see <http://linux.die.net/man/8/tc> for details about traffic controls in linux.

Example

```
try: import autotest.common as common # pylint: disable=W0611
```

except ImportError:

```
import common # pylint: disable=W0611
```

```

from autotest.client.net.net_tc import * from autotest.client.net.net_utils import *
class mock_netif(object):
    def __init__(self, name): self._name = name
    def get_name(self): return self._name
netem_qdisc = netem() netem_qdisc.add_param('loss 100%')
ack_filter = u32filter() ack_filter.add_rule('match ip protocol 6 0xff') ack_filter.add_rule('match u8 0x10 0x10
at nexthdr+13') ack_filter.set_dest_qdisc(netem_qdisc)
root_qdisc = prio() root_qdisc.get_class(2).set_leaf_qdisc(netem_qdisc) root_qdisc.add_filter(ack_filter)
lo_if = mock_netif('lo')
root_qdisc.setup(lo_if)
# run test here ... root_qdisc.restore(lo_if)
class autotest.client.net.net_tc.classful_qdisc(handle)
    Bases: autotest.client.net.net_tc.qdisc
    add_class(child_class)
    add_filter(filter)
    classful = True
    restore(netif)
    setup(netif)
class autotest.client.net.net_tc.classless_qdisc(handle)
    Bases: autotest.client.net.net_tc.qdisc
    classful = False
class autotest.client.net.net_tc.netem(handle=300)
    Bases: autotest.client.net.net_tc.classless_qdisc
    add_param(param)
    name = 'netem'
    setup(netif)
autotest.client.net.net_tc.new_handle()
class autotest.client.net.net_tc.pfifo(handle=200)
    Bases: autotest.client.net.net_tc.classless_qdisc
    name = 'pfifo'
    setup(netif)
class autotest.client.net.net_tc.prio(handle=100, bands=3)
    Bases: autotest.client.net.net_tc.classful_qdisc
    get_class(band)
    name = 'prio'
    setup(netif)
class autotest.client.net.net_tc.qdisc(handle)
    Bases: object

```

```

get_handle ()
get_parent_class ()
id ()
restore (netif)
set_parent_class (parent_class)
setup (netif)
tc_cmd (tc_conf)

```

class autotest.client.net.net_tc.**tcclass** (*handle, minor, leaf_qdisc=None*)

Bases: `object`

```

add_child (child_class)
get_leaf_qdisc ()
get_minor ()
get_parent_class ()
id ()
restore (netif)
set_leaf_qdisc (leaf_qdisc)
set_parent_class (parent_class)
setup (netif)

```

class autotest.client.net.net_tc.**tcfilter**

Bases: `object`

```

conf_command = 'cmd'
conf_device = 'dev'
conf_flowid = 'flowid'
conf_name = 'name'
conf_params = 'params'
conf_parent = 'parent'
conf_priority = 'priority'
conf_protocol = 'protocol'
conf_qdiscid = 'qdiscid'
conf_rules = 'cmd'
conf_type = 'filtertype'
get_dest_qdisc ()
get_handle ()
get_parent_qdisc ()
get_priority ()
get_protocol ()
restore (netif)

```

```

set_dest_qdisc (dest_qdisc)
set_handle (handle)
set_parent_qdisc (parent_qdisc)
set_priority (priority)
set_protocol (protocol)
setup (netif)
tc_cmd (tc_conf)

```

```

class autotest.client.net.net_tc.u32filter
  Bases: autotest.client.net.net_tc.tcfilter
add_rule (rule)
filtertype = 'u32'
restore (netif)
setup (netif)

```

net_utils Module

Convenience functions for use by network tests or whomever.

This library is to release in the public repository.

```
autotest.client.net.net_utils.bond()
```

```

class autotest.client.net.net_utils.bonding
  Bases: object

```

This class implements bonding interface abstraction.

```

AB_MODE = 1
AD_MODE = 2
NO_MODE = 0
disable()
enable()
get_active_interfaces()
get_mii_status()
get_mode()
get_slave_interfaces()
is_bondable()
is_enabled()
wait_for_state_change()

```

Wait for bonding state change.

Wait up to 90 seconds to successfully ping the gateway. This is to know when LACP state change has converged. (0 seconds is 3x lacp timeout, use by protocol)

class `autotest.client.net.net_utils.ethernet`

Bases: `object`

Provide ethernet packet manipulation methods.

CHECKSUM_LEN = 4

ETH_LLDP_DST_MAC = '01:80:C2:00:00:0E'

ETH_PACKET_MAX_SIZE = 1518

ETH_PACKET_MIN_SIZE = 64

ETH_TYPE_8021Q = 33024

ETH_TYPE_ARP = 2054

ETH_TYPE_CDP = 8192

ETH_TYPE_IP = 2048

ETH_TYPE_IP6 = 34525

ETH_TYPE_LLDP = 35020

ETH_TYPE_LOOPBACK = 36864

FRAME_KEY_DST_MAC = 'dst'

FRAME_KEY_PAYLOAD = 'payload'

FRAME_KEY_PROTO = 'proto'

FRAME_KEY_SRC_MAC = 'src'

HDR_LEN = 14

static `mac_binary_to_string` (*hwaddr*)

Converts a MAC address byte string to text string.

Converts a MAC byte string 'xxxxxxxxxx' to a text string 'aa:aa:aa:aa:aa:aa'

Args: *hwaddr*: a byte string containing the MAC address to convert.

Returns: A text string.

static `mac_string_to_binary` (*hwaddr*)

Converts a MAC address text string to byte string.

Converts a MAC text string from a text string 'aa:aa:aa:aa:aa:aa' to a byte string 'xxxxxxxxxx'

Args: *hwaddr*: a text string containing the MAC address to convert.

Returns: A byte string.

static `pack` (*dst, src, protocol, payload*)

Pack a frame in a byte string.

Args: *dst*: destination mac in byte string format *src*: src mac address in byte string format *protocol*: short in network byte order *payload*: byte string payload data

Returns: An ethernet frame with header and payload in a byte string.

static `unpack` (*raw_frame*)

Unpack a raw ethernet frame.

Returns:

None on error

```
    { 'dst' [byte string,] 'src' : byte string, 'proto' : short in host byte order, 'payload' : byte string
    }
```

```
autotest.client.net.net_utils.ethernet_packet()
```

```
autotest.client.net.net_utils.netif(name)
```

```
autotest.client.net.net_utils.network()
```

```
class autotest.client.net.net_utils.network_interface(name)
```

```
    Bases: object
```

```
    DISABLE = False
```

```
    ENABLE = True
```

```
    add_maddr(maddr)
```

```
    del_maddr(maddr)
```

```
    disable_loopback()
```

```
    disable_promisc()
```

```
    down()
```

```
    enable_loopback()
```

```
    enable_promisc()
```

```
    exists()
```

```
    flush()
```

```
    get_advertised_link_modes()
```

```
    get_carrier()
```

```
    get_driver()
```

```
    get_hwaddr()
```

```
    get_ipaddr()
```

```
    get_name()
```

```
    get_speed()
```

```
    get_stats()
```

```
    get_stats_diff(orig_stats)
```

```
    get_supported_link_modes()
```

```
    get_wakeon()
```

```
    is_autoneg_advertised()
```

```
    is_autoneg_on()
```

```
    is_down()
```

```
    is_full_duplex()
```

```
    is_loopback_enabled()
```

```
    is_pause_autoneg_on()
```

```
    is_rx_pause_on()
```

```

is_rx_summing_on ()
is_scatter_gather_on ()
is_tso_on ()
is_tx_pause_on ()
is_tx_summing_on ()
parse_ethtool (field, match, option='', next_field='')
recv (len)
restore ()
send (buf)
set_hwaddr (hwaddr)
set_ipaddr (ipaddr)
up ()
wait_for_carrier (timeout=60)

```

class autotest.client.net.net_utils.**network_utils**

Bases: `object`

disable_ip_local_loopback (*ignore_status=False*)

enable_ip_local_loopback (*ignore_status=False*)

get_ip_local (*query_ip, netmask='24'*)

Get ip address in local system which can communicate with `query_ip`.

Parameters `query_ip` – IP of client which wants to communicate with autotest machine.

Returns IP address which can communicate with `query_ip`

list ()

process_mpstat (*mpstat_out, sample_count, loud=True*)

Parses mpstat output of the following two forms: 02:10:17 0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 100.00
1012.87 02:10:13 PM 0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 100.00 1019.00

reset (*ignore_status=False*)

start (*ignore_status=False*)

stop (*ignore_status=False*)

class autotest.client.net.net_utils.**raw_socket** (*iface_name*)

Bases: `object`

This class implements an raw socket abstraction.

ETH_P_ALL = 3

SOCKET_TIMEOUT = 1

close ()

Close the raw socket

open (*protocol=None*)

Opens the raw socket to send and receive.

Args: `protocol` : short in host byte order. None if ALL

recv (*timeout*)

Synchronous receive.

Receives one packet from the interface and returns its content in a string. Wait up to timeout for the packet if timeout is not 0. This function filters out all the packets that are less than the minimum ethernet packet size (60+crc).

Args:

timeout: max time in seconds to wait for the read to complete. '0', wait for ever until a valid packet is received

Returns:

packet: None no packet was received a binary string containing the received packet.

time_left: amount of time left in timeout

recv_from (*dst_mac, src_mac, protocol*)

Receive an ethernet frame that matches the dst, src and proto.

Filters all received packet to find a matching one, then unpack it and present it to the caller as a frame.

Waits up to self._socket_timeout for a matching frame before returning.

Args: dst_mac: 'byte string'. None do not use in filter. src_mac: 'byte string'. None do not use in filter. protocol: short in host byte order. None do not use in filter.

Returns:

ethernet frame: { 'dst' [byte string,
 'src' : byte string, 'proto' : short in host byte order, 'payload' : byte string
 }

send (*packet*)

Send an ethernet packet.

send_to (*dst_mac, src_mac, protocol, payload*)

Send an ethernet frame.

Send an ethernet frame, formating the header.

Args: dst_mac: 'byte string' src_mac: 'byte string' protocol: short in host byte order payload: 'byte string'

set_socket_timeout (*timeout*)

Set the timeout use by recv_from.

Args: timeout: time in seconds

socket ()

socket_timeout ()

Get the timeout use by recv_from

net_utils_mock Module

Set of Mocks and stubs for network utilities unit tests.

Implement a set of mocks and stubs use to implement unit tests for the network libraries.

class autotest.client.net.net_utils_mock.netif_stub (*iface, cls, name, *args, **kwargs*)

Bases: *autotest.client.shared.test_utils.mock.mock_class*

wait_for_carrier (*timeout*)

autotest.client.net.net_utils_mock.**netutils_netif** (*iface*)

class autotest.client.net.net_utils_mock.**network_interface_mock** (*iface='some_name', test_init=False*)

Bases: *autotest.client.net.net_utils.network_interface*

get_driver ()

get_ipaddr ()

is_down ()

is_loopback_enabled ()

wait_for_carrier (*timeout=1*)

autotest.client.net.net_utils_mock.**os_open** (**args, **kwarg*)

class autotest.client.net.net_utils_mock.**os_stub** (*symbol, **kwargs*)

Bases: *autotest.client.shared.test_utils.mock.mock_function*

open (**args, **kwargs*)

read (**args, **kwargs*)

readval = ''

class autotest.client.net.net_utils_mock.**socket_stub** (*iface, cls, name, *args, **kwargs*)

Bases: *autotest.client.shared.test_utils.mock.mock_class*

Class use to mock sockets.

bind (*arg*)

close ()

recv (*size*)

send (*buf*)

settimeout (*timeout*)

socket (*family, type*)

profilers Package

profilers Package

class autotest.client.profilers.**profilers** (*job*)

Bases: *autotest.client.shared.profiler_manager.profiler_manager*

load_profiler (*profiler, args, dargs*)

Subpackages

blktrace Package

blktrace Module

Autotest profiler for blktrace blktrace - generate traces of the i/o traffic on block devices

```
class autotest.clientprofilers.blktrace.blktrace.blktrace (job)
    Bases: autotest.client.profiler.profiler

    get_device (test)
    initialize (**dargs)
    report (test)
    setup (tarball='blktrace.tar.bz2', **dargs)
    start (test)
    stop (test)
    version = 2
```

catprofile Package

catprofile Module

Sets up a subprocess to cat a file on a specified interval

Defaults options: `job.profilers.add('catprofile', ['/proc/meminfo', '/proc/uptime'],
outfile=monitor, interval=1)`

```
class autotest.clientprofilers.catprofile.catprofile.catprofile (job)
    Bases: autotest.client.profiler.profiler

    initialize (filenames=['/proc/meminfo', '/proc/slabinfo'], outfile='monitor', interval=1, **dargs)
    report (test)
    start (test)
    stop (test)
    version = 1
```

cmdprofile Package

cmdprofile Module

Sets up a subprocess to run any generic command in the background every few seconds (by default the interval is 60 secs)

```
class autotest.clientprofilers.cmdprofile.cmdprofile.cmdprofile (job)
    Bases: autotest.client.profiler.profiler

    initialize (cmds=['ps'], interval=60, outputfile='cmdprofile', outputfiles=None, **dargs)
    start (test)
    stop (test)
    supports_reboot = True
    version = 2
```

cpistat Package

cpistat Module

Uses perf_events to count cycles and instructions

Defaults options: job.profilers.add('cpistat', interval=1)

class autotest.client.profilers.cpistat.cpistat.**cpistat** (*job*)

Bases: *autotest.client.profiler.profiler*

initialize (*interval=1, **dargs*)

start (*test*)

stop (*test*)

version = 1

ftrace Package

ftrace Module

Function tracer profiler for autotest.

author David Sharp (dhsharp@google.com)

class autotest.client.profilers.ftrace.ftrace.**ftrace** (*job*)

Bases: *autotest.client.profiler.profiler*

ftrace profiler for autotest. It builds ftrace from source and runs trace-cmd with configurable parameters.

@see: [git://git.kernel.org/pub/scm/linux/kernel/git/rostedt/trace-cmd.git](http://git.kernel.org/pub/scm/linux/kernel/git/rostedt/trace-cmd.git)

initialize (*tracepoints, buffer_size_kb=1408, **kwargs*)

Initialize ftrace profiler.

Parameters

- **tracepoints** – List containing a mix of tracepoint names and (tracepoint name, filter) tuples. Tracepoint names are as accepted by trace-cmd -e, eg “syscalls”, or “syscalls:sys_enter_read”. Filters are as accepted by trace-cmd -f, eg “((sig >= 10 && sig < 15) || sig == 17)”
- **buffer_size_kb** – Set the size of the ring buffer (per cpu).

static join_command (*cmd*)

Shell escape the command for BgJob. grmbl.

Parameters cmd – Command list.

mountpoint = '/sys/kernel/debug'

setup (*tarball='trace-cmd.tar.bz2', **kwargs*)

Build and install trace-cmd from source.

The tarball was obtained by checking the git repo at 09-14-2010, removing the Documentation and the .git folders, and compressing it.

Parameters

- **tarball** – Path to trace-cmd tarball.

- ****kwargs** – Dictionary with additional parameters.

start (*test*)

Start ftrace profiler

Parameters test – Autotest test in which the profiler will operate on.

stop (*test*)

Stop ftrace profiler.

Parameters test – Autotest test in which the profiler will operate on.

tracing_dir = '/sys/kernel/debug/tracing'

version = 1

inotify Package

inotify Module

inotify logs filesystem activity that may be directly or indirectly caused by the test that is running. It requires the inotify-tools package, more specifically, the inotifywait tool.

Heavily inspired / shamelessly copied from the kvm_stat profiler.

copyright Red Hat 2013

author Cleber Rosa <cleber@redhat.com>

class `autotest.clientprofilers.inotify.inotify.inotify` (*job*)

Bases: `autotest.client.profiler.profiler`

Profiler based on inotifywait from inotify-tools

initialize (*paths=[]*)

report (*test*)

start (*test*)

stop (*test*)

version = 1

iostat Package

iostat Module

Run iostat with a default interval of 1 second.

class `autotest.clientprofilers.iostat.iostat.iostat` (*job*)

Bases: `autotest.client.profiler.profiler`

initialize (*interval=1, options='', **dargs*)

report (*test*)

start (*test*)

stop (*test*)

version = 2

kvm_stat Package

kvm_stat Module

kvm_stat prints statistics generated by the kvm module. It depends on debugfs. If no debugfs is mounted, the profiler will try to mount it so it's possible to proceed.

copyright Red Hat 2010

author Lucas Meneghel Rodrigues (lmr@redhat.com)

class `autotest.clientprofilers.kvm_stat.kvm_stat.kvm_stat` (*job*)

Bases: `autotest.client.profiler.profiler`

kvm_stat based profiler. Consists on executing `kvm_stat -l` during a given test execution, redirecting its output to a file on the profile dir.

initialize (***dargs*)

Gets path of `kvm_stat` and verifies if `debugfs` needs to be mounted.

report (*test*)

Report function. Does nothing as there's no postprocessing needed.

Parameters *test* – Autotest test on which this profiler will operate on.

start (*test*)

Starts `kvm_stat` subprocess.

Parameters *test* – Autotest test on which this profiler will operate on.

stop (*test*)

Stops profiler execution by sending a `SIGTERM` to `kvm_stat` process.

Parameters *test* – Autotest test on which this profiler will operate on.

version = 1

lockmeter Package

lockmeter Module

Lockstat is the basic tool used to control the kernel's Lockmeter functionality: e.g., turning the kernel's data gathering on or off, and retrieving that data from the kernel so that Lockstat can massage it and produce printed reports. See <http://oss.sgi.com/projects/lockmeter> for details.

NOTE: if you get compile errors from `config.h`, referring you to a FAQ, you might need to do `'cat < /dev/null > /usr/include/linux/config.h'`. But read the FAQ first.

class `autotest.client.profilers.lockmeter.lockmeter.lockmeter` (*job*)

Bases: `autotest.client.profiler.profiler`

initialize (***dargs*)

report (*test*)

setup (*tarball*='lockstat-1.4.11.tar.bz2')

start (*test*)

stop (*test*)

version = 1

lttng Package

lttng Module

Trace kernel events with Linux Tracing Toolkit (lttng). You need to install the lttng patched kernel in order to use the profiler.

Examples:

```
job.profilers.add('lttng', tracepoints = None): enable all trace points.
job.profilers.add('lttng', tracepoints = []): disable all trace points.
job.profilers.add('lttng', tracepoints = ['kernel_arch_syscall_entry',
                                          'kernel_arch_syscall_exit'])
```

will only trace syscall events. Take a look at `/proc/ltt` for the list of the tracing events currently supported by lttng and their output formats.

To view the collected traces, copy `results/your-test/profiler/lttng` to a machine that has Linux Tracing Toolkit Viewer (lttv) installed:

```
test$ scp -r results/your-test/profiler/lttng user@localmachine:/home/tmp/
```

Then you can examine the traces either in text mode or in GUI:

```
localmachine$ lttv -m textDump -t /home/tmp/lttng
```

or

```
localmachine$ lttv-gui -t /home/tmp/lttng &
```

```
class autotest.client.profilers.lttng.lttng.lttng (job)
    Bases: autotest.client.profiler.profiler
    initialize (outputsize=1048576, tracepoints=None, **dargs)
    setup (tarball='ltt-control-0.51-12082008.tar.gz', **dargs)
    start (test)
    stop (test)
    version = 1
```

mpstat Package

mpstat Module

Sets up a subprocess to run mpstat on a specified interval, default 1 second

```
class autotest.client.profilers.mpstat.mpstat.mpstat (job)
    Bases: autotest.client.profiler.profiler
    initialize (interval=1, **dargs)
    report (test)
    start (test)
    stop (test)
```

```
version = 1
```

oprofile Package

oprofile Module

OProfile is a system-wide profiler for Linux systems, capable of profiling all running code at low overhead. OProfile is released under the GNU GPL.

It consists of a kernel driver and a daemon for collecting sample data, and several post-profiling tools for turning data into information.

More Info: <http://oprofile.sourceforge.net/> Will need some libraries to compile. Do 'apt-get build-dep oprofile'

```
class autotest.clientprofilers.oprofile.oprofile.oprofile (job)
  Bases: autotest.client.profiler.profiler

  initialize (vmlinux=None, events=[], others=None, local=None, **dargs)

  report (test)

  setup (tarball='oprofile-0.9.4.tar.bz2', local=None, *args, **dargs)

  setup_done = False

  start (test)

  stop (test)

  version = 7
```

perf Package

perf Module

perf is a tool included in the linux kernel tree that supports functionality similar to oprofile and more.

@see: <http://lwn.net/Articles/310260/>

```
class autotest.clientprofilers.perf.perf.perf (job)
  Bases: autotest.client.profiler.profiler

  initialize (events=['cycles', 'instructions'], trace=False, **dargs)

  report (test)

  start (test)

  stop (test)

  version = 1
```

powertop Package

powertop Module

What's eating the battery life of my laptop? Why isn't it many more hours? Which software component causes the most power to be burned? These are important questions without a good answer... until now.

```
class autotest.clientprofilers.powertop.powertop.powertop (job)
    Bases: autotest.client.profiler.profiler

    preserve_srcdir = True

    report (test)

    setup (*args, **dargs)

    start (test)

    stop (test)

    version = 1
```

readprofile Package

readprofile Module

readprofile - a tool to read kernel profiling information

The readprofile command uses the /proc/profile information to print ascii data on standard output. The output is organized in three columns: the first is the number of clock ticks, the second is the name of the C function in the kernel where those many ticks occurred, and the third is the normalized ‘load’ of the procedure, calculated as a ratio between the number of ticks and the length of the procedure. The output is filled with blanks to ease readability.

```
class autotest.clientprofilers.readprofile.readprofile.readprofile (job)
    Bases: autotest.client.profiler.profiler

    initialize (**dargs)

    report (test)

    setup (tarball='util-linux-2.12r.tar.bz2')

    start (test)

    stop (test)

    version = 1
```

sar Package

sar Module

Sets up a subprocess to run sar from the sysstat suite

Default options: sar -A -f

```
class autotest.clientprofilers.sar.sar.sar (job)
    Bases: autotest.client.profiler.profiler
```

The sar command writes to standard output the contents of selected cumulative activity counters in the operating system. This profiler executes sar and redirects its output in a file located in the profiler results dir.

```
initialize (interval=1, **dargs)
```

Set sar interval and verify what flags the installed sar supports.

Parameters interval – Interval used by sar to produce system data.

report (*test*)

Report function. Convert the binary sar data to text.

Parameters **test** – Autotest test on which this profiler will operate on.

start (*test*)

Starts sar subprocess.

Parameters **test** – Autotest test on which this profiler will operate on.

stop (*test*)

Stops profiler execution by sending a SIGTERM to sar process.

Parameters **test** – Autotest test on which this profiler will operate on.

version = 1

systemtap Package

systemtap Module

Autotest systemtap profiler.

class `autotest.clientprofilers.systemtap.systemtap.systemtap` **systemtap** (*job*)

Bases: `autotest.client.profiler.profiler`

Tracing test process using systemtap tools.

initialize (***dargs*)

report (*test*)

start (*test*)

stop (*test*)

version = 1

vmstat Package

vmstat Module

Runs vmstat X where X is the interval in seconds

Defaults options: `job.profilers.add('vmstat', interval=1)`

class `autotest.client.profilers.vmstat.vmstat` **vmstat** (*job*)

Bases: `autotest.client.profiler.profiler`

initialize (*interval=1, **dargs*)

report (*test*)

start (*test*)

stop (*test*)

version = 1

shared Package

autotemp Module

Autotest tempfile wrapper for mkstemp (known as tempfile here) and mkdtemp (known as tempdir).

This wrapper provides a mechanism to clean up temporary files/dirs once they are no longer need.

Files/Dirs will have a `unique_id` prepended to the suffix and a `_autotmp_` tag appended to the prefix.

It is required that the `unique_id` param is supplied when a temp dir/file is created.

```
class autotest.client.shared.autotemp.tempdir (suffix='', unique_id=None, prefix='', dir=None)
```

Bases: `object`

A wrapper for `tempfile.mkdtemp`

@var name: The name of the temporary dir. :return: A `tempdir` object example usage:

```
b = autotemp.tempdir(unique_id='exemdir') b.name # your directory b.clean() # clean up after yourself
```

clean()

Remove the temporary dir that was created. This is also called by the destructor.

```
class autotest.client.shared.autotemp.tempfile (unique_id, suffix='', prefix='', dir=None, text=False)
```

Bases: `object`

A wrapper for `tempfile.mkstemp`

Parameters `unique_id` – required, a unique string to help identify what part of code created the tempfile.

@var name: The name of the temporary file. @var fd: the file descriptor of the temporary file that was created. :return: a `tempfile` object example usage:

```
t = autotemp.tempfile(unique_id='fig') t.name # name of file t.fd # file descriptor t.fo # file object t.clean() # clean up after yourself
```

clean()

Remove the temporary file that was created. This is also called by the destructor.

barrier Module

base_barrier Module

```
exception autotest.client.shared.base_barrier.BarrierAbortError
```

Bases: `autotest.client.shared.error.BarrierError`

Special `BarrierError` raised when an explicit abort is requested.

```
class autotest.client.shared.base_barrier.barrier (hostid, tag, timeout=None, port=None, listen_server=None)
```

Bases: `object`

Multi-machine barrier support.

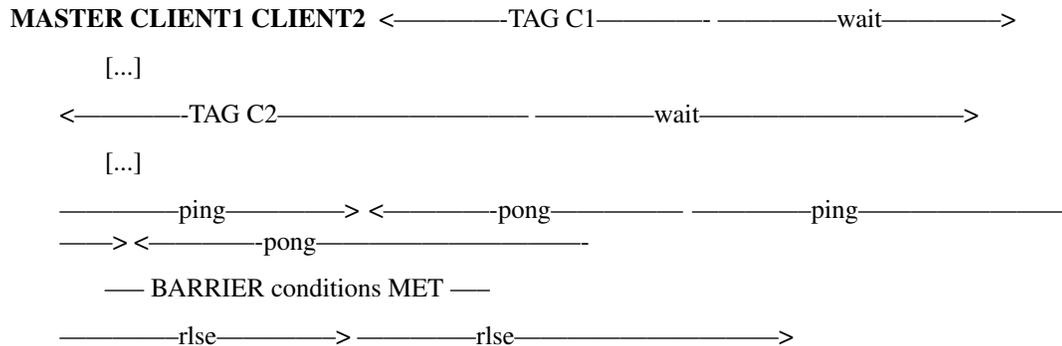
Provides multi-machine barrier mechanism. Execution stops until all members arrive at the barrier.

Implementation Details:

When a barrier is forming the master node (first in sort order) in the set accepts connections from each member of the set. As they arrive they indicate the barrier they are joining and their identifier (their hostname or IP address and optional tag). They are then asked to wait. When all members are present the master node then checks that each member is still responding via a ping/pong exchange. If this is successful then everyone has checked in at the barrier. We then tell everyone they may continue via a rlse message.

Where the master is not the first to reach the barrier the client connects will fail. Client will retry until they either succeed in connecting to master or the overall timeout is exceeded.

As an example here is the exchange for a three node barrier called 'TAG'



Note that once the last client has responded to pong the barrier is implicitly deemed satisfied, they have all acknowledged their presence. If we fail to send any of the rlse messages the barrier is still a success, the failed host has effectively broken 'right at the beginning' of the post barrier execution window.

In addition, there is another rendezvous, that makes each slave a server and the master a client. The connection process and usage is still the same but allows barriers from machines that only have a one-way connection initiation. This is called rendezvous_servers.

For example:

```

if ME == SERVER: server start
b = job.barrier(ME, 'server-up', 120) b.rendezvous(CLIENT, SERVER)

if ME == CLIENT: client run
b = job.barrier(ME, 'test-complete', 3600) b.rendezvous(CLIENT, SERVER)

if ME == SERVER: server stop
    
```

Any client can also request an abort of the job by setting abort=True in the rendezvous arguments.

rendezvous (*hosts, **dargs)

rendezvous_servers (masterid, *hosts, **dargs)

autotest.client.shared.base_barrier.get_host_from_id(hostid)

class autotest.client.shared.base_barrier.listen_server(address='', port=11922)

Bases: object

Manages a listening socket for barrier.

Can be used to run multiple barrier instances with the same listening socket (if they were going to listen on the same port).

Attributes:

Attr address Address to bind to (string).

Attr port Port to bind to.

Attr socket Listening socket object.

close()
Close the listening socket.

base_check_version Module

class autotest.client.shared.base_check_version.**base_check_python_version**

PYTHON_BIN_GLOB_STRINGS = ['/usr/bin/python2*', '/usr/local/bin/python2*']

extract_version(*path*)

find_desired_python()
Returns the path of the desired python interpreter.

restart()

base_job Module

class autotest.client.shared.base_job.**TAPReport**(*enable*, *resultdir=None*,
global_filename='status')

Bases: `object`

Deal with TAP reporting for the Autotest client.

job_statuses = {'END GOOD': True, 'GOOD': True, 'NOSTATUS': False, 'WARN': False, 'START': True, 'ERROR': True}

record(*log_entry*, *indent*, *log_files*)
Append a job-level status event to self._reports_container. All events will be written to TAP log files at the end of the test run. Otherwise, it's impossible to determine the TAP plan.

Parameters

- **log_entry** – A string status code describing the type of status entry being recorded. It must pass `log.is_valid_status` to be considered valid.
- **indent** – Level of the `log_entry` to determine the operation if `log_entry.operation` is not given.
- **log_files** – List of full path of files the TAP report will be written to at the end of the test.

record_keyval(*path*, *dictionary*, *type_tag=None*)
Append a key-value pairs of dictionary to self._keyval_container in TAP format. Once finished write out the `keyval.tap` file to the file system.

If `type_tag` is `None`, then the key must be composed of alphanumeric characters (or dashes + underscores). However, if `type-tag` is not null then the keys must also have "{`type_tag`}" as a suffix. At the moment the only valid values of `type_tag` are "attr" and "perf".

Parameters

- **path** – The full path of the `keyval.tap` file to be created
- **dictionary** – The keys and values.
- **type_tag** – The type of the values

classmethod **tap_ok**(*success*, *counter*, *message*)
return a TAP message string.

Parameters

- **success** – True for positive message string.
- **counter** – number of TAP line in plan.
- **message** – additional message to report in TAP line.

write()

Write the TAP reports to files.

class `autotest.client.shared.base_job.base_job(*args, **dargs)`

Bases: `object`

An abstract base class for the various autotest job classes.

Property autodir The top level autotest directory.

Property clientdir The autotest client directory.

Property serverdir The autotest server directory. [OPTIONAL]

Property resultdir The directory where results should be written out. [WRITABLE]

Property pkgdir The job packages directory. [WRITABLE]

Property tmpdir The job temporary directory. [WRITABLE]

Property testdir The job test directory. [WRITABLE]

Property customtestdir The custom test directory. [WRITABLE]

Property site_testdir The job site test directory. [WRITABLE]

Property bindir The client bin/ directory.

Property configdir The client config/ directory.

Property profdir The client profilers/ directory.

Property toolmdir The client tools/ directory.

Property conmuxdir The conmux directory. [OPTIONAL]

Property control A path to the control file to be executed. [OPTIONAL]

Property hosts A set of all live Host objects currently in use by the job. Code running in the context of a local client can safely assume that this set contains only a single entry.

Property machines A list of the machine names associated with the job.

Property user The user executing the job.

Property tag A tag identifying the job. Often used by the scheduler to give a name of the form NUMBER-USERNAME/HOSTNAME.

Property args A list of additional miscellaneous command-line arguments provided when starting the job.

Property last_boot_tag The label of the kernel from the last reboot. [OPTIONAL,PERSISTENT]

Property automatic_test_tag A string which, if set, will be automatically added to the test name when running tests.

Property default_profile_only A boolean indicating the default value of `profile_only` used by `test.execute`. [PERSISTENT]

Property drop_caches A boolean indicating if caches should be dropped before each test is executed.

Property drop_caches_between_iterations A boolean indicating if caches should be dropped before each test iteration is executed.

Property run_test_cleanup A boolean indicating if test.cleanup should be run by default after a test completes, if the run_cleanup argument is not specified. [PERSISTENT]

Property num_tests_run The number of tests run during the job. [OPTIONAL]

Property num_tests_failed The number of tests failed during the job. [OPTIONAL]

Property bootloader An instance of the boottool class. May not be available on job instances where access to the bootloader is not available (e.g. on the server running a server job). [OPTIONAL]

Property harness An instance of the client test harness. Only available in contexts where client test execution happens. [OPTIONAL]

Property logging An instance of the logging manager associated with the job.

Property profilers An instance of the profiler manager associated with the job.

Property sysinfo An instance of the sysinfo object. Only available in contexts where it's possible to collect sysinfo.

Property warning_manager A class for managing which types of WARN messages should be logged and which should be suppressed. [OPTIONAL]

Property warning_loggers A set of readable streams that will be monitored for WARN messages to be logged. [OPTIONAL]

Abstract methods:

`_find_base_directories` [CLASSMETHOD] Returns the location of autotest, clientdir and serverdir

`_find_resultdir` Returns the location of resultdir. Gets a copy of any parameters passed into `base_job.__init__`. Can return None to indicate that no resultdir is to be used.

`_get_status_logger` Returns a status_logger instance for recording job status logs.

`autotest`

`automatic_test_tag`

`bindir`

`clientdir`

`configdir`

`conmuxdir`

`customtestdir`

`default_profile_only`

`get_state` (*name*, *default*=<object object>)

Returns the value associated with a particular name.

Parameters

- **name** – The name the value was saved with.
- **default** – A default value to return if no state is currently associated with var.

Returns A deep copy of the value associated with name. Note that this explicitly returns a deep copy to avoid problems with mutable values; mutations are not persisted or shared.

Raises `KeyError` when no state is associated with var and a default value is not provided.

`last_boot_tag`

`pkgdir`

`pop_execution_context ()`

Reverse the effects of the previous `push_execution_context` call.

Raises `IndexError` when the stack of contexts is empty.

`profdir`

`push_execution_context (resultdir)`

Save off the current context of the job and change to the given one.

In practice method just changes the `resultdir`, but it may become more extensive in the future. The expected use case is for when a child job needs to be executed in some sort of nested context (for example the way `parallel_simple` does). The original context can be restored with a `pop_execution_context` call.

Parameters `resultdir` – The new `resultdir`, relative to the current one.

`record (status_code, subdir, operation, status='', optional_fields=None)`

Record a job-level status event.

Logs an event noteworthy to the Autotest job as a whole. Messages will be written into a global status log file, as well as a subdir-local status log file (if `subdir` is specified).

Parameters

- **status_code** – A string status code describing the type of status entry being recorded. It must pass `log.is_valid_status` to be considered valid.
- **subdir** – A specific results subdirectory this also applies to, or `None`. If not `None` the subdirectory must exist.
- **operation** – A string describing the operation that was run.
- **status** – An optional human-readable message describing the status entry, for example an error message or “completed successfully”.
- **optional_fields** – An optional dictionary of additional named fields to be included with the status message. Every time `timestamp` and `localtime` entries are generated with the current time and added to this dictionary.

`record_entry (entry, log_in_subdir=True)`

Record a job-level status event, using a `status_log_entry`.

This is the same as `self.record` but using an existing status log entry object rather than constructing one for you.

Parameters

- **entry** – A `status_log_entry` object
- **log_in_subdir** – A boolean that indicates (when true) that subdir logs should be written into the subdirectory status log file.

`resultdir`

`run_test_cleanup`

`serverdir`

`set_state (name, value)`

Saves the value given with the provided name.

Parameters

- **name** – The name the value should be saved with.
- **value** – The value to save.

site_testdir

tag

testdir

tmpdir

toolsdir

use_sequence_number

class `autotest.client.shared.base_job.job_directory` (*path*, *is_writable=False*)

Bases: `object`

Represents a job.*dir directory.

exception `JobDirectoryException`

Bases: `autotest.client.shared.error.AutotestError`

Generic job_directory exception superclass.

exception `job_directory.MissingDirectoryException` (*path*)

Bases: `autotest.client.shared.base_job.JobDirectoryException`

Raised when a directory required by the job does not exist.

exception `job_directory.UncreatableDirectoryException` (*path*, *error*)

Bases: `autotest.client.shared.base_job.JobDirectoryException`

Raised when a directory required by the job is missing and cannot be created.

exception `job_directory.UnwritableDirectoryException` (*path*)

Bases: `autotest.client.shared.base_job.JobDirectoryException`

Raised when a writable directory required by the job exists but is not writable.

static `job_directory.property_factory` (*attribute*)

Create a job.*dir -> job.*dir.path property accessor.

Parameters **attribute** – A string with the name of the attribute this is exposed as. ‘_’+attribute must then be attribute that holds either None or a job_directory-like object

Returns A read-only property object that exposes a job_directory path

class `autotest.client.shared.base_job.job_state`

Bases: `object`

A class for managing explicit job and user state, optionally persistent.

The class allows you to save state by name (like a dictionary). Any state stored in this class should be picklable and deep copyable. While this is not enforced it is recommended that only valid python identifiers be used as names. Additionally, the namespace ‘stateful_property’ is used for storing the valued associated with properties constructed using the property_factory method.

NO_DEFAULT = <object object>

PICKLE_PROTOCOL = 2

discard (**args*, ***dargs*)

If namespace.name is a defined value, deletes it.

Parameters

- **namespace** (*string*) – The namespace that the property should be stored in.
- **name** (*string*) – The name the value was saved with.

discard_namespace (**args, **dargs*)
Delete all defined namespace.* names.

Parameters **namespace** (*string*) – The namespace to be cleared.

get (**args, **dargs*)
Returns the value associated with a particular name.

Parameters

- **namespace** (*string*) – The namespace that the property should be stored in.
- **name** (*string*) – The name the value was saved with.
- **default** (*object*) – A default value to return if no state is currently associated with var.

Returns A deep copy of the value associated with name. Note that this explicitly returns a deep copy to avoid problems with mutable values; mutations are not persisted or shared.

Raises `KeyError` raised when no state is associated with var and a default value is not provided.

has (**args, **dargs*)
Return a boolean indicating if namespace.name is defined.

Parameters

- **namespace** (*string*) – The namespace that the property should be stored in.
- **name** (*string*) – The name the value was saved with.

Returns True if the given name is defined in the given namespace and False otherwise.

Return type `bool`

static property_factory (*state_attribute, property_attribute, default, namespace='global_properties'*)
Create a property object for an attribute using self.get and self.set.

Parameters

- **state_attribute** – A string with the name of the attribute on job that contains the job_state instance.
- **property_attribute** – A string with the name of the attribute this property is exposed as.
- **default** – A default value that should be used for this property if it is not set.
- **namespace** – The namespace to store the attribute value in.

Returns A read-write property object that performs self.get calls to read the value and self.set calls to set it.

read_from_file (*file_path, merge=True*)
Read in any state from the file at file_path.

When `merge=True`, any state specified only in-memory will be preserved. Any state specified on-disk will be set in-memory, even if an in-memory setting already exists.

Parameters

- **file_path** (*string*) – The path where the state should be read from. It must exist but it can be empty.
- **merge** (*bool*) – If true, merge the on-disk state with the in-memory state. If false, replace the in-memory state with the on-disk state.

Warning: This method is intentionally concurrency-unsafe. It makes no attempt to control concurrent access to the file at `file_path`.

set (**args, **dargs*)

Saves the value given with the provided name.

Parameters

- **namespace** (*string*) – The namespace that the property should be stored in.
- **name** (*string*) – The name the value was saved with.
- **value** – The value to save.

set_backing_file (*file_path*)

Change the path used as the backing file for the persistent state.

When a new backing file is specified if a file already exists then its contents will be added into the current state, with conflicts between the file and memory being resolved in favor of the file contents. The file will then be kept in sync with the (combined) in-memory state. The syncing can be disabled by setting this to `None`.

Parameters **file_path** (*string*) – A path on the filesystem that can be read from and written to, or `None` to turn off the backing store.

write_to_file (*file_path*)

Write out the current state to the given path.

Warning: This method is intentionally concurrency-unsafe. It makes no attempt to control concurrent access to the file at `file_path`.

Parameters **file_path** (*string*) – The path where the state should be written out to. Must be writable.

class `autotest.client.shared.base_job.status_indenter`

Bases: `object`

Abstract interface that a status log indenter should use.

decrement ()

Decrease indentation by one level.

increment ()

Increase indentation by one level.

indent

class `autotest.client.shared.base_job.status_log_entry` (*status_code, subdir, operation, message, fields, timestamp=None*)

Bases: `object`

Represents a single status log entry.

BAD_CHAR_REGEX = `<_sre.SRE_Pattern object>`

LOCALTIME_FIELD = `'localtime'`

RENDERED_NONE_VALUE = `'—'`

TIMESTAMP_FIELD = 'timestamp'

is_end()

Indicates if this status log is the end of a nested block.

Returns A boolean indicating if this entry ends a nested block.

is_start()

Indicates if this status log is the start of a new nested block.

Returns A boolean indicating if this entry starts a new nested block.

classmethod parse (*line*)

Parse a status log entry from a text string.

This method is the inverse of render; it should always be true that parse(entry.render()) produces a new status_log_entry equivalent to entry.

Returns A new status_log_entry instance with fields extracted from the given status line. If the line is an extra message line then None is returned.

render()

Render the status log entry into a text string.

Returns A text string suitable for writing into a status log file.

```
class autotest.client.shared.base_job.status_logger (job, indenter,
                                                    global_filename='status',
                                                    subdir_filename='status',
                                                    record_hook=None,
                                                    tap_writer=None)
```

Bases: `object`

Represents a status log file. Responsible for translating messages into on-disk status log lines.

Property global_filename The filename to write top-level logs to.

Property subdir_filename The filename to write subdir-level logs to.

record_entry (*log_entry*, *log_in_subdir=True*)

Record a status_log_entry into the appropriate status log files.

Parameters

- **log_entry** – A status_log_entry instance to be recorded into the status logs.
- **log_in_subdir** – A boolean that indicates (when true) that subdir logs should be written into the subdirectory status log file.

render_entry (*log_entry*)

Render a status_log_entry as it would be written to a log file.

Parameters log_entry – A status_log_entry instance to be rendered.

Returns The status log entry, rendered as it would be written to the logs (including indentation).

autotest.client.shared.base_job.**with_backing_file** (*method*)

A decorator to perform a lock-read-***-write-unlock cycle.

When applied to a method, this decorator will automatically wrap calls to the method in a lock-and-read before the call followed by a write-and-unlock. Any operation that is reading or writing state should be decorated with this method to ensure that backing file state is consistently maintained.

autotest.client.shared.base_job.**with_backing_lock** (*method*)

A decorator to perform a lock-***-unlock cycle.

When applied to a method, this decorator will automatically wrap calls to the method in a backing file lock and before the call followed by a backing file unlock.

base_packages Module

This module defines the BasePackageManager Class which provides an implementation of the packaging system API providing methods to fetch, upload and remove packages. Site specific extensions to any of these methods should inherit this class.

```
class autotest.client.shared.base_packages.BasePackageManager (pkgmgr_dir, host-
name=None,
repo_urls=None,
upload_paths=None,
do_locking=True,
run_function=<function
run>,
run_function_args=[],
run_function_dargs={})
```

Bases: `object`

add_repository (*repo*)

compare_checksum (*pkg_path*, *repo_url*)

Calculate the checksum of the file specified in `pkg_path` and compare it with the checksum in the checksum file. Return True if both match else return False. :param `pkg_path`: The full path to the package file for which the checksum is being compared :param `repo_url`: The URL to fetch the checksum from

compute_checksum (*pkg_path*)

Compute the MD5 checksum for the package file and return it. `pkg_path` : The complete path for the package file

fetch_pkg (*pkg_name*, *dest_path*, *repo_url=None*, *use_checksum=False*, *install=False*)

Fetch the package into `dest_dir` from `repo_url`. By default `repo_url` is None and the package is looked in all the repositories specified. Otherwise it fetches it from the specific `repo_url`. `pkg_name` : name of the package (ex: `test-sleeptest.tar.bz2`,

`dep-gcc.tar.bz2`, `kernel.1-1.rpm`)

`repo_url` : the URL of the repository where the package is located. `dest_path` : complete path of where the package will be fetched to. `use_checksum` : This is set to False to fetch the packages.checksum file

so that the checksum comparison is bypassed for the checksum file itself. This is used internally by the packaging system. It should be ignored by external callers of this method who use it to fetch custom packages.

install [`install_path` has unique name and destination requirements] that vary based on the fetcher that is used. So call them here as opposed to `install_pkg`.

get_fetcher (*url*)

get_mirror_list (*repo_urls*)

Stub function for site specific mirrors.

Returns: Priority ordered list

get_package_name (*url*, *pkg_type*)

Extract the group and test name for the url. This method is currently used only for tests.

static get_tarball_name (*name, pkg_type*)

Converts a package name and type into a tarball name.

Parameters

- **name** – The name of the package
- **pkg_type** – The type of the package

Returns A tarball filename for that specific type of package

install_pkg (*name, pkg_type, fetch_dir, install_dir, preserve_install_dir=False, repo_url=None*)

Remove `install_dir` if it already exists and then recreate it unless `preserve_install_dir` is specified as `True`. Fetch the package into the `pkg_dir`. Untar the package into `install_dir`. The assumption is that packages are of the form : `<pkg_type>.<pkg_name>.tar.bz2` `name` : name of the package `type` : type of the package `fetch_dir` : The directory into which the package tarball will be

fetches to.

`install_dir` : the directory where the package files will be untarred to `repo_url` : the url of the repository to fetch the package from.

static parse_tarball_name (*tarball_name*)

Converts a package tarball name into a package name and type.

Parameters `tarball_name` – The filename of the tarball

Returns (`name, pkg_type`) where `name` is the package name and `pkg_type` is the package type.

remove_checksum (*pkg_name*)

Remove the checksum of the package from the packages checksum file. This method is called whenever a package is removed from the repositories in order to clean its corresponding checksum. `pkg_name` : The name of the package to be removed

remove_pkg (*pkg_name, remove_path=None, remove_checksum=False*)

Remove the package from the specified `remove_path` `pkg_name` : name of the package (ex: `test-sleeptest.tar.bz2`,

`dep-gcc.tar.bz2`)

`remove_path` : the location to remove the package from.

remove_pkg_file (*filename, pkg_dir*)

Remove the file named `filename` from `pkg_dir`

repo_check (*repo*)

Check to make sure the repo is in a sane state: ensure we have at least XX amount of free space. Make sure we can write to the repo

tar_package (*pkg_name, src_dir, dest_dir, include_string=None, exclude_string=None*)

Create a tar.bz2 file with the name '`pkg_name`' say `test-blah.tar.bz2`.

Includes the files specified in `include_string`, and excludes the files specified on the `exclude_string`, while tarring the source. Returns the destination tarball path.

Parameters

- **pkg_name** – Package name.
- **src_dir** – Directory that contains the data to be packaged.
- **dest_dir** – Directory that will hold the destination tarball.
- **include_string** – Pattern that represents the files that will be added to the tar package.

- **exclude_string** – Pattern that represents the files that should be excluded from the tar package. It could be either a string or a list.

untar_pkg (*tarball_path, dest_dir*)

Untar the package present in the *tarball_path* and put a ".checksum" file in the *dest_dir* containing the checksum of the tarball. This method assumes that the package to be untarred is of the form <name>.tar.bz2

untar_required (*tarball_path, dest_dir*)

Compare the checksum of the *tarball_path* with the .checksum file in the *dest_dir* and return False if it matches. The untar of the package happens only if the checksums do not match.

update_checksum (*pkg_path*)

Update the checksum of the package in the packages' checksum file. This method is called whenever a package is fetched just to be sure that the checksums in the local file are the latest. *pkg_path* : The complete path to the package file.

upkeep (*custom_repos=None*)

Clean up custom upload/download areas

upload_pkg (*pkg_path, upload_path=None, update_checksum=False, timeout=300*)

upload_pkg_dir (*dir_path, upload_path*)

Upload a full directory. Depending on the upload path, the appropriate method for that protocol is called. Currently this copies the whole tmp package directory to the target directory. This assumes that the web server is running on the same machine where the method is being called from. The *upload_path*'s files are basically served by that web server.

upload_pkg_file (*file_path, upload_path*)

Upload a single file. Depending on the upload path, the appropriate method for that protocol is called. Currently this simply copies the file to the target directory (but can be extended for other protocols) This assumes that the web server is running on the same machine where the method is being called from. The *upload_path*'s files are basically served by that web server.

upload_pkg_parallel (*pkg_path, upload_path, update_checksum=False*)

Uploads to a specified *upload_path* or to all the repos. Also uploads the checksum file to all the repos. *pkg_path* : The complete path to the package file *upload_path* : the absolute path where the files are copied to.

if set to 'None' assumes 'all' repos

update_checksum [If set to False, the checksum file is not] going to be updated which happens by default.

This is necessary for custom packages (like custom kernels and custom tests) that get uploaded which do not need to be part of the checksum file and bloat it.

class `autotest.client.shared.base_packages.GitFetcher` (*package_manager, repository_url*)

Bases: `autotest.client.shared.base_packages.RepositoryFetcher`

A git based repository fetcher

fetch_pkg_file (*filename, dest_path*)

Fetch a package file and save it to the given destination path

git is an SCM, you can download the test directly. No need to fetch a bz2'd tarball file. However 'filename' is <type>-<name>.tar.bz2 break this up and only fetch <name>.

Parameters

- **filename** (*string*) – The filename of the package file to fetch.
- **dest_path** (*string*) – Destination path to download the file to.

`git_archive_cmd_pattern = 'git archive --remote=%s -o %s %s'`

`install_pkg_post (filename, fetch_dir, install_dir, preserve_install_dir=False)`

class `autotest.client.shared.base_packages.HttpFetcher` (*package_manager, repository_url*)

Bases: `autotest.client.shared.base_packages.RepositoryFetcher`

Repository Fetcher using HTTP

`fetch_pkg_file (filename, dest_path)`

Fetch a package file from a package repository.

Parameters

- `filename` (*string*) – The filename of the package file to fetch.
- `dest_path` (*string*) – Destination path to download the file to.

Raises `PackageFetchError` – if the fetch failed

`wget_cmd_pattern = 'wget --connect-timeout=15 -nv %s -O %s'`

class `autotest.client.shared.base_packages.LocalFilesystemFetcher` (*package_manager, repository_url*)

Bases: `autotest.client.shared.base_packages.RepositoryFetcher`

`fetch_pkg_file (filename, dest_path)`

class `autotest.client.shared.base_packages.RepositoryFetcher` (*package_manager, repository_url*)

Bases: `object`

Base class with common functionality for repository fetchers

`fetch_pkg_file (filename, dest_path)`

Fetch a package file from a package repository.

Parameters

- `filename` (*string*) – The filename of the package file to fetch.
- `dest_path` (*string*) – Destination path to download the file to.

Raises `PackageFetchError` – if the fetch failed

`install_pkg_post (filename, fetch_dir, install_dir, preserve_install_dir=False)`

Fetcher specific post install

Parameters

- `filename` (*string*) – The filename of the package to install
- `fetch_dir` (*string*) – The fetched path of the package
- `install_dir` (*string*) – The path to install the package to

@preserve_install_dir: Preserve the install directory

`install_pkg_setup (name, fetch_dir, install)`

Install setup for a package based on fetcher type.

Parameters

- `name` (*string*) – The filename to be munged
- `fetch_dir` (*string*) – The destination path to be munged

- **install** (*boolean*) – Whether this is be called from the install path or not

Returns tuple with (name, fetch_dir)

url = None

`autotest.client.shared.base_packages.check_diskspace(repo, min_free=None)`

Check if the remote directory over at the pkg repo has available diskspace

If the amount of free space is not supplied, it is taken from the global configuration file, section [PACKAGES], key 'minimum_free_space'. The unit used are in SI, that is, 1 GB = 10**9 bytes.

Parameters **repo** (*string*) – a remote package repo URL

Param **min_free** minimum amount of free space, in GB (10**9 bytes)

Raises

- **error.RepoUnknownError** – general repository error condition
- **error.RepoDiskFullError** – repository does not have at least the requested amount of free disk space.

`autotest.client.shared.base_packages.check_write(repo)`

Checks that the remote repository directory is writable

Parameters **repo** (*string*) – a remote package repo URL

Raises **error.RepoWriteError** – repository write error

`autotest.client.shared.base_packages.create_directory(repo)`

Create a directory over at the remote repository

Parameters **repo** (*string*) – the repo URL containing the remote directory path

Returns a CmdResult object or None

`autotest.client.shared.base_packages.has_pbzip2()`

Check if parallel bzip2 is available on this system.

Returns True if pbzip2 is available, False otherwise

`autotest.client.shared.base_packages.parse_ssh_path(repo)`

Parse an SSH url

Parameters **repo** (*string*) – a repo uri like ssh://xx@xx/path/to/

Returns tuple with (host, remote_path)

`autotest.client.shared.base_packages.repo_run_command(repo, cmd, ignore_status=False, cd=True)`

Run a command relative to the repo path

This is basically a `utils.run()` wrapper that sets itself in a repo directory if it is appropriate, so parameters such as `cmd` and `ignore_status` are passed along to it.

Parameters

- **repo** (*string*) – a repository url
- **cmd** (*string*) – the command to be executed. This is passed along to `utils.run()`
- **ignore_status** (*boolean*) – do not raise an exception, no matter what the exit code of the command is.
- **cd** (*boolean*) – wether to change the working directory to the repo directory before running the specified command.

Returns a `CmdResult` object or `None`

Raises `CmdError` – the exit code of the command execution was not 0

`autotest.client.shared.base_packages.trim_custom_directories` (*repo*,
older_than_days=None)

Remove old files from the remote repo directory

The age of the files, if not provided by the `older_than_days` parameter is taken from the global configuration file, at section [PACKAGES], configuration item 'custom_max_age'.

Parameters `repo` (*string*) – a remote package repo URL

base_syncdata Module

class `autotest.client.shared.base_syncdata.SessionData` (*hosts*, *timeout*)

Bases: `object`

`close()`

`is_finished()`

`set_finish()`

`timeout()`

class `autotest.client.shared.base_syncdata.SyncData` (*masterid*, *hostid*, *hosts*, *session_id=None*, *listen_server=None*,
port=13234, *tmpdir=None*)

Bases: `object`

Provides data synchronization between hosts.

Transferred data is pickled and sent to all destination points. If there is no listen server it will create a new one. If multiple hosts wants to communicate with each other, then communications are identified by `session_id`.

`close()`

`single_sync` (*data=None*, *timeout=60*, *session_id=None*)

`sync` (*data=None*, *timeout=60*, *session_id=None*)

Synchronize data between hosts.

`timeout()`

class `autotest.client.shared.base_syncdata.SyncListenServer` (*address=''*,
port=13234, *tm-*
pdir=None)

Bases: `object`

`close()`

Close `SyncListenServer` thread.

Close all open connection with clients and listen server.

class `autotest.client.shared.base_syncdata.TempDir` (*tmpdir=None*)

Bases: `autotest.client.shared.autotemp.tempdir`

`TempDir` class is `tempdir` for predefined `tmpdir`.

`clean()`

Should not delete predefined `tmpdir`.

`autotest.client.shared.base_syncdata.net_recv_object` (*sock*, *timeout=60*)

Receive python object over network.

Parameters

- **ip_addr** – ipaddress of waiter for data.
- **obj** – object to send

Returns object from network

`autotest.client.shared.base_syncdata.net_send_object(sock, obj)`
Send python object over network.

Parameters

- **ip_addr** – ipaddress of waiter for data.
- **obj** – object to send

boottool Module

boottool client-side module.

This module provides an API for client side tests that need to manipulate boot entries. It's based on the rewrite of boottool, now python and grubby based. It aims to be keep API compatibility with the older version, except from XEN support which has been removed. We'll gladly accept patches that provide full coverage for this mode/feature.

Copyright 2009 Google Inc. Copyright 2012 Red Hat, Inc.

Released under the GPL v2

class `autotest.client.shared.boottool.boottool` (*path=None*)

Bases: `autotest.client.tools.boottool.Grubby`

Client site side boottool wrapper.

Inherits all functionality from boottool(.py) CLI app (lazily).

check_version Module

class `autotest.client.shared.check_version.check_python_version`

Bases: `autotest.client.shared.check_version.site_check_python_version`,
`autotest.client.shared.base_check_version.base_check_python_version`

class `autotest.client.shared.check_version.site_check_python_version`

common Module

control_data Module

class `autotest.client.shared.control_data.ControlData` (*vars*, *path*,
raise_warnings=False)

Bases: `object`

set_attr (*attr, val, raise_warnings=False*)

set_author (*val*)

set_dependencies (*val*)

set_doc (*val*)

set_experimental (*val*)

set_name (*val*)
set_run_verify (*val*)
set_sync_count (*val*)
set_test_category (*val*)
set_test_class (*val*)
set_test_parameters (*val*)
set_test_type (*val*)
set_time (*val*)

exception `autotest.client.shared.control_data.ControlVariableException`

Bases: `exceptions.Exception`

`autotest.client.shared.control_data.parse_control` (*path*, *raise_warnings=False*)

distro Module

This module provides the client facilities to detect the Linux Distribution it's running under.

This is a replacement for the `get_os_vendor()` function from the `utils` module.

class `autotest.client.shared.distro.LinuxDistro` (*name*, *version*, *release*, *arch*)

Bases: `object`

Simple collection of information for a Linux Distribution

class `autotest.client.shared.distro.Probe`

Bases: `object`

Probes the machine and does it best to confirm it's the right distro

CHECK_FILE = None

Points to a file that can determine if this machine is running a given Linux Distribution. This servers a first check that enables the extra checks to carry on.

CHECK_FILE_CONTAINS = None

Sets the content that should be checked on the file pointed to by `CHECK_FILE_EXISTS`. Leave it set to *None* (its default) to check only if the file exists, and not check its contents

CHECK_FILE_DISTRO_NAME = None

The name of the Linux Distribution to be returned if the file defined by `CHECK_FILE_EXISTS` exist.

CHECK_VERSION_REGEX = None

A regular expression that will be run on the file pointed to by `CHECK_FILE_EXISTS`

check_name_for_file ()

Checks if this class will look for a file and return a distro

The conditions that must be true include the file that identifies the distro file being set (`CHECK_FILE`) and the name of the distro to be returned (`CHECK_FILE_DISTRO_NAME`)

check_name_for_file_contains ()

Checks if this class will look for text on a file and return a distro

The conditions that must be true include the file that identifies the distro file being set (`CHECK_FILE`), the text to look for inside the distro file (`CHECK_FILE_CONTAINS`) and the name of the distro to be returned (`CHECK_FILE_DISTRO_NAME`)

check_release()
 Checks if this has the conditions met to look for the release number

check_version()
 Checks if this class will look for a regex in file and return a distro

get_distro()
 Returns the *LinuxDistro* this probe detected

name_for_file()
 Get the distro name if the *CHECK_FILE* is set and exists

name_for_file_contains()
 Get the distro if the *CHECK_FILE* is set and has content

release()
 Returns the release of the distro

version()
 Returns the version of the distro

`autotest.client.shared.distro.register_probe(probe_class)`
 Register a probe to be run during autodetection

`autotest.client.shared.distro.detect()`
 Attempts to detect the Linux Distribution running on this machine

Returns the detected *LinuxDistro* or *UNKNOWN_DISTRO*

Return type *LinuxDistro*

distro_def Module

This module defines a structure and portable format for relevant information on Linux Distributions in such a way that information about known distros can be packed and distributed.

Please note that this module deals with Linux Distributions not necessarily installed on the running system.

`autotest.client.shared.distro_def.save(linux_distro, path)`
 Saves the *linux_distro* to an external file format

Parameters

- **linux_distro** (*DistroDef*) – an *DistroDef* instance
- **path** (*str*) – the location for the output file

Returns None

`autotest.client.shared.distro_def.load(path)`
 Loads the distro from an external file

Parameters **path** (*str*) – the location for the input file

Returns an *DistroDef* instance

Return type *DistroDef*

`autotest.client.shared.distro_def.load_from_tree(name, version, release, arch, package_type, path)`
 Loads a *DistroDef* from an installable tree

Parameters

- **name** (*str*) – a short name that precisely distinguishes this Linux Distribution among all others.
- **version** (*str*) – the major version of the distribution. Usually this is a single number that denotes a large development cycle and support file.
- **release** (*str*) – the release or minor version of the distribution. Usually this is also a single number, that is often omitted or starts with a 0 when the major version is initially release. It's often associated with a shorter development cycle that contains incremental a collection of improvements and fixes.
- **arch** (*str*) – the main target for this Linux Distribution. It's common for some architectures to ship with packages for previous and still compatible architectures, such as it's the case with Intel/AMD 64 bit architecture that support 32 bit code. In cases like this, this should be set to the 64 bit architecture name.
- **package_type** (*str*) – one of the available package info loader types
- **path** (*str*) – top level directory of the distro installation tree files

class `autotest.client.shared.distro_def.SoftwarePackage` (*name, version, release, checksum, arch*)

Bases: `object`

Definition of relevant information on a software package

class `autotest.client.shared.distro_def.DistroDef` (*name, version, release, arch*)

Bases: `autotest.client.shared.distro.LinuxDistro`

More complete information on a given Linux Distribution

software_packages = None

All the software packages that ship with this Linux distro

software_packages_type = None

A simple text that denotes the software type that makes this distro

`autotest.client.shared.distro_def.DISTRO_PKG_INFO_LOADERS = {'deb': <class 'autotest.client.shared.distro_>`
the type of distro that will determine what loader will be used

enum Module

Generic enumeration support.

class `autotest.client.shared.enum.Enum` (**names, **kwargs*)

Bases: `object`

Utility class to implement Enum-like functionality.

```
>>> e = Enum('String one', 'String two')
>>> e.STRING_ONE
0
>>> e.STRING_TWO
1
>>> e.choices()
[(0, 'String one'), (1, 'String two')]
>>> e.get_value('String one')
0
>>> e.get_string(0)
'String one'
```

```
>>> e = Enum('Hello', 'Goodbye', string_values=True)
>>> e.HELLO, e.GOODBYE
('Hello', 'Goodbye')
```

```
>>> e = Enum('One', 'Two', start_value=1)
>>> e.ONE
1
>>> e.TWO
2
```

choices ()

Return choice list suitable for Django model choices.

static get_attr_name (string)

get_string (value)

Given a value, get the string name for it.

get_value (name)

Convert a string name to it's corresponding value. If a value is passed in, it is returned.

error Module

Internal global error types

`autotest.client.shared.error.format_error ()`

`autotest.client.shared.error.context_aware (fn)`

A decorator that must be applied to functions that call context().

`autotest.client.shared.error.context (s=' ', log=None)`

Set the context for the currently executing function and optionally log it.

Parameters

- **s** – A string. If not provided, the context for the current function will be cleared.
- **log** – A logging function to pass the context message to. If None, no function will be called.

`autotest.client.shared.error.get_context ()`

Return the current context (or None if none is defined).

`autotest.client.shared.error.exception_context (e)`

Return the context of a given exception (or None if none is defined).

exception `autotest.client.shared.error.AutoservHostIsShuttingDownError`

Bases: `autotest.client.shared.error.AutoservHostError`

Host is shutting down

exception `autotest.client.shared.error.AutoservShutdownError`

Bases: `autotest.client.shared.error.AutoservRebootError`

Error occurred during shutdown of machine

exception `autotest.client.shared.error.AutoservHardwareRepairRequiredError`

Bases: `autotest.client.shared.error.AutoservError`

Exception class raised during repairs to indicate that a hardware repair is going to be necessary.

- exception** `autotest.client.shared.error.RepoWriteError`
 Bases: `autotest.client.shared.error.PackagingError`
 Raised when packager cannot write to a repo's destination
- exception** `autotest.client.shared.error.AutoservUnsupportedError`
 Bases: `autotest.client.shared.error.AutoservError`
 Error raised when you try to use an unsupported optional feature
- exception** `autotest.client.shared.error.CmdError` (`command`, `result_obj`, `additional_text=None`)
 Bases: `autotest.client.shared.error.TestError`
 Indicates that a command failed, is fatal to the test unless caught.
- exception** `autotest.client.shared.error.AutotestError`
 Bases: `exceptions.Exception`
 The parent of all errors deliberately thrown within the client code.
- exception** `autotest.client.shared.error.RepoDiskFullError`
 Bases: `autotest.client.shared.error.PackagingError`
 Raised when the destination for packages is full
- exception** `autotest.client.shared.error.AutoservRebootError`
 Bases: `autotest.client.shared.error.AutoservError`
 Error occurred while rebooting a machine
- exception** `autotest.client.shared.error.TestWarn`
 Bases: `autotest.client.shared.error.TestBaseException`
 Indicates that bad things (may) have happened, but not an explicit failure.
exit_status = 'WARN'
- exception** `autotest.client.shared.error.PackageInstallError`
 Bases: `autotest.client.shared.error.PackagingError`
 Raised when there is an error installing the package
- exception** `autotest.client.shared.error.HostInstallProfileError`
 Bases: `autotest.client.shared.error.JobError`
 Indicates the machine failed to have a profile assigned.
- exception** `autotest.client.shared.error.PackageError`
 Bases: `autotest.client.shared.error.TestError`
 Indicates an error trying to perform a package operation.
- exception** `autotest.client.shared.error.AutotestHostRunError` (`description`, `result_obj`)
 Bases: `autotest.client.shared.error.HostRunErrorMixin`, `autotest.client.shared.error.AutotestError`
- exception** `autotest.client.shared.error.UnhandledTestFail` (`unhandled_exception`)
 Bases: `autotest.client.shared.error.TestFail`
 Indicates an unhandled fail in a test.
- exception** `autotest.client.shared.error.BarrierAbortError`
 Bases: `autotest.client.shared.error.BarrierError`
 Indicate that the barrier was explicitly aborted by a member.

exception `autotest.client.shared.error.AutoservSubcommandError` (*func, exit_code*)
Bases: `autotest.client.shared.error.AutoservError`

Indicates an error while executing a (forked) subcommand

exception `autotest.client.shared.error.NetCommunicationError`
Bases: `autotest.client.shared.error.JobError`

Indicate that network communication was broken.

exception `autotest.client.shared.error.PackageRemoveError`
Bases: `autotest.client.shared.error.PackagingError`

Raised when there is an error removing the package

exception `autotest.client.shared.error.UnhandledTestError` (*unhandled_exception*)
Bases: `autotest.client.shared.error.TestError`

Indicates an unhandled error in a test.

exception `autotest.client.shared.error.DataSyncError`
Bases: `autotest.client.shared.error.NetCommunicationError`

Indicates problem during synchronization data over network.

exception `autotest.client.shared.error.AutoservHostError`
Bases: `autotest.client.shared.error.AutoservError`

Error reaching a host

exception `autotest.client.shared.error.TestBaseException`
Bases: `autotest.client.shared.error.AutotestError`

The parent of all test exceptions.

exit_status = 'NEVER_RAISE_THIS'

exception `autotest.client.shared.error.TestNAError`
Bases: `autotest.client.shared.error.TestBaseException`

Indicates that the test is Not Applicable. Should be thrown when various conditions are such that the test is inappropriate.

exit_status = 'TEST_NA'

exception `autotest.client.shared.error.AutoservHardwareHostError`
Bases: `autotest.client.shared.error.AutoservHostError`

Found hardware problems with the host

exception `autotest.client.shared.error.AutoservError`
Bases: `exceptions.Exception`

exception `autotest.client.shared.error.AutoservSSTimeout`
Bases: `autotest.client.shared.error.AutoservError`

SSH experienced a connection timeout

exception `autotest.client.shared.error.InstallError`
Bases: `autotest.client.shared.error.JobError`

Indicates an installation error which Terminates and fails the job.

exception `autotest.client.shared.error.AutoservDiskFullHostError` (*path, want_gb, free_space_gb*)
Bases: `autotest.client.shared.error.AutoservHostError`

Not enough free disk space on host

exception `autotest.client.shared.error.AutoservInstallError`

Bases: `autotest.client.shared.error.AutoservError`

Error occurred while installing autotest on a host

exception `autotest.client.shared.error.TestError`

Bases: `autotest.client.shared.error.TestBaseException`

Indicates that something went wrong with the test harness itself.

exit_status = 'ERROR'

exception `autotest.client.shared.error.AutoservVirtError`

Bases: `autotest.client.shared.error.AutoservError`

Virtualization related error

exception `autotest.client.shared.error.BarrierError`

Bases: `autotest.client.shared.error.JobError`

Indicates an error happened during a barrier operation.

exception `autotest.client.shared.error.AutotestRunError`

Bases: `autotest.client.shared.error.AutotestError`

Indicates a problem running server side control files.

exception `autotest.client.shared.error.RepoError`

Bases: `autotest.client.shared.error.PackagingError`

Raised when a repo isn't working in some way

exception `autotest.client.shared.error.PackagingError`

Bases: `autotest.client.shared.error.AutotestError`

Abstract error class for all packaging related errors.

exception `autotest.client.shared.error.RepoUnknownError`

Bases: `autotest.client.shared.error.PackagingError`

Raised when packager cannot write to a repo's destination

exception `autotest.client.shared.error.UnhandledJobError` (*unhandled_exception*)

Bases: `autotest.client.shared.error.JobError`

Indicates an unhandled error in a job.

exception `autotest.client.shared.error.TestFail`

Bases: `autotest.client.shared.error.TestBaseException`

Indicates that the test failed, but the job will not continue.

exit_status = 'FAIL'

exception `autotest.client.shared.error.JobError`

Bases: `autotest.client.shared.error.AutotestError`

Indicates an error which terminates and fails the whole job (ABORT).

exception `autotest.client.shared.error.AutoservRunError` (*description, result_obj*)

Bases: `autotest.client.shared.error.HostRunErrorMixin`, `autotest.client.shared.error.AutoservError`

exception `autotest.client.shared.error.PackageFetchError`

Bases: `autotest.client.shared.error.PackagingError`

Raised when there is an error fetching the package

exception `autotest.client.shared.error.PackageUploadError`

Bases: `autotest.client.shared.error.PackagingError`

Raised when there is an error uploading the package

exception `autotest.client.shared.error.AutoservHardwareRepairRequestedError`

Bases: `autotest.client.shared.error.AutoservError`

Exception class raised from `Host.repair_full()` (or overrides) when software repair fails but it successfully managed to request a hardware repair (by notifying the staff, sending mail, etc)

exception `autotest.client.shared.error.HostRunErrorMixin` (*description, result_obj*)

Bases: `exceptions.Exception`

Indicates a problem in the `host run()` function raised from client code. Should always be constructed with a tuple of two args (error description (str), run result object). This is a common class mixed in to create the client and server side versions of it.

exception `autotest.client.shared.error.HarnessError`

Bases: `autotest.client.shared.error.JobError`

Indicates problem with the harness.

exception `autotest.client.shared.error.AutoservNotMountedHostError`

Bases: `autotest.client.shared.error.AutoservHostError`

Found unmounted partitions that should be mounted

exception `autotest.client.shared.error.AutoservSshPermissionDeniedError` (*description, re-sult_obj*)

Bases: `autotest.client.shared.error.AutoservRunError`

Indicates that a SSH permission denied error was encountered.

exception `autotest.client.shared.error.HostInstallTimeoutError`

Bases: `autotest.client.shared.error.JobError`

Indicates the machine failed to be installed after the predetermined timeout.

exception `autotest.client.shared.error.AutoservSshPingHostError`

Bases: `autotest.client.shared.error.AutoservHostError`

SSH ping failed

exception `autotest.client.shared.error.AutotestTimeoutError`

Bases: `autotest.client.shared.error.AutotestError`

This exception is raised when an autotest test exceeds the timeout parameter passed to `run_timed_test` and is killed.

git Module

Code that helps to deal with content from git repositories

class `autotest.client.shared.git.GitRepoHelper` (*uri, branch='master', lbranch=None, commit=None, destination_dir=None, base_uri=None*)

Bases: `object`

Helps to deal with git repos, mostly fetching content from a repo

checkout (*branch=None, commit=None*)

Performs a git checkout for a given branch and start point (commit)

Parameters

- **branch** – Remote branch name.
- **commit** – Specific commit hash.

execute ()

Performs all steps necessary to initialize and download a git repo.

This includes the init, fetch and checkout steps in one single utility method.

fetch (*uri*)

Performs a git fetch from the remote repo

get_top_commit ()

Returns the topmost commit id for the current branch.

Returns Commit id.

get_top_tag ()

Returns the topmost tag for the current branch.

Returns Tag.

git_cmd (*cmd, ignore_status=False*)

Wraps git commands.

Parameters

- **cmd** – Command to be executed.
- **ignore_status** – Whether we should suppress error.CmdError exceptions if the command did return exit code !=0 (True), or not suppress them (False).

init ()

Initializes a directory for receiving a verbatim copy of git repo

This creates a directory if necessary, and either resets or inits the repo

`autotest.client.shared.git.get_repo` (*uri, branch='master', lbranch=None, commit=None, destination_dir=None, base_uri=None*)

Utility function that retrieves a given git code repository.

Parameters

- **uri** (*string*) – git repository url
- **branch** (*string*) – git remote branch
- **destination_dir** (*string*) – path of a dir where to save downloaded code
- **commit** (*string*) – specific commit to download
- **lbranch** (*string*) – git local branch name, if different from remote
- **uri** – a closer, usually local, git repository url from where to fetch content first from

host_protections Module

host_queue_entry_states Module

This module contains the status enums for use by HostQueueEntry in the database. It is a stand alone module as these status strings are needed from various disconnected pieces of code that should not depend on everything that autotest.frontend.afe.models depends on such as RPC clients.

iscsi Module

Basic iscsi support for Linux host with the help of commands iscsiadm and tgtadm.

This include the basic operates such as login and get device name by target name. And it can support the real iscsi access and emulated iscsi in localhost then access it.

class autotest.client.shared.iscsi.**Iscsi** (*params, root_dir='/tmp'*)

Bases: object

Basic iscsi support class. Will handle the emulated iscsi export and access to both real iscsi and emulated iscsi device.

cleanup ()

Clean up env after iscsi used.

delete_target ()

Delete target from host.

export_target ()

Export target in localhost for emulated iscsi

get_device_name ()

Get device name from the target name.

get_target_id ()

Get target id from image name. Only works for emulated iscsi device

logged_in ()

Check if the session is login or not.

login ()

Login session for both real iscsi device and emulated iscsi. Include env check and setup.

logout ()

Logout from target.

portal_visible ()

Check if the portal can be found or not.

autotest.client.shared.iscsi.**iscsi_discover** (*portal_ip*)

Query from iscsi server for available targets

Parameters *portal_ip* – Ip for iscsi server

autotest.client.shared.iscsi.**iscsi_get_nodes** ()

Get the iscsi nodes

autotest.client.shared.iscsi.**iscsi_get_sessions** ()

Get the iscsi sessions activated

autotest.client.shared.iscsi.**iscsi_login** (*target_name*)

Login to a target with the target name

Parameters `target_name` – Name of the target

`autotest.client.shared.iscsi.iscsi_logout` (*target_name=None*)

Logout from a target. If the target name is not set then logout all targets.

Params `target_name` Name of the target.

iso9660 Module

Basic ISO9660 file-system support.

This code does not attempt (so far) to implement code that knows about ISO9660 internal structure. Instead, it uses commonly available support either in userspace tools or on the Linux kernel itself (via mount).

`autotest.client.shared.iso9660.iso9660` (*path*)

Checks the available tools on a system and chooses class accordingly

This is a convenience function, that will pick the first available iso9660 capable tool.

Parameters `path` (*str*) – path to an iso9660 image file

Returns an instance of any iso9660 capable tool

Return type *Iso9660IsoInfo*, *Iso9660IsoRead*, *Iso9660Mount* or *None*

class `autotest.client.shared.iso9660.Iso9660IsoInfo` (*path*)

Bases: `autotest.client.shared.iso9660.BaseIso9660`

Represents a ISO9660 filesystem

This implementation is based on the cdrkit's isoinfo tool

read (*path*)

class `autotest.client.shared.iso9660.Iso9660IsoRead` (*path*)

Bases: `autotest.client.shared.iso9660.BaseIso9660`

Represents a ISO9660 filesystem

This implementation is based on the libcdio's iso-read tool

close ()

copy (*src*, *dst*)

read (*path*)

class `autotest.client.shared.iso9660.Iso9660Mount` (*path*)

Bases: `autotest.client.shared.iso9660.BaseIso9660`

Represents a mounted ISO9660 filesystem.

close ()

Perform umount operation on the temporary dir

Return type *None*

copy (*src*, *dst*)

Parameters

- **src** (*str*) – source
- **dst** (*str*) – destination

Return type *None*

read (*path*)

Read data from path

Parameters **path** (*str*) – path to read data

Returns data content

Return type *str*

jsontemplate Module

Python implementation of json-template.

JSON Template is a minimal and powerful templating language for transforming a JSON dictionary to arbitrary text.

To use this module, you will typically use the Template constructor, and catch various exceptions thrown. You may also want to use the FromFile/FromString methods, which allow Template constructor options to be embedded in the template string itself.

Other functions are exposed for tools which may want to process templates.

exception `autotest.client.shared.jsontemplate.Error`

Bases: `exceptions.Exception`

Base class for all exceptions in this module.

Thus you can “except `jsontemplate.Error`: to catch all exceptions thrown by this module.

exception `autotest.client.shared.jsontemplate.CompilationError`

Bases: `autotest.client.shared.jsontemplate.Error`

Base class for errors that happen during the compilation stage.

exception `autotest.client.shared.jsontemplate.EvaluationError` (*msg*, *original_exception=None*)

Bases: `autotest.client.shared.jsontemplate.Error`

Base class for errors that happen when expanding the template.

This class of errors generally involve the data dictionary or the execution of the formatters.

exception `autotest.client.shared.jsontemplate.BadFormatter`

Bases: `autotest.client.shared.jsontemplate.CompilationError`

A bad formatter was specified, e.g. `{variable|BAD}`

exception `autotest.client.shared.jsontemplate.BadPredicate`

Bases: `autotest.client.shared.jsontemplate.CompilationError`

A bad predicate was specified, e.g. `{.BAD?}`

exception `autotest.client.shared.jsontemplate.MissingFormatter`

Bases: `autotest.client.shared.jsontemplate.CompilationError`

Raised when formatters are required, and a variable is missing a formatter.

exception `autotest.client.shared.jsontemplate.ConfigurationError`

Bases: `autotest.client.shared.jsontemplate.CompilationError`

Raised when the Template options are invalid and it can’t even be compiled.

exception `autotest.client.shared.jsontemplate.TemplateSyntaxError`

Bases: `autotest.client.shared.jsontemplate.CompilationError`

Syntax error in the template text.

exception `autotest.client.shared.jsontemplate.UndefinedVariable` (*msg*, *original_exception=None*)

Bases: `autotest.client.shared.jsontemplate.EvaluationError`

The template contains a variable not defined by the data dictionary.

`autotest.client.shared.jsontemplate.CompileTemplate` (*template_str*, *builder=None*, *meta='{}'*, *format_char='\'*, *more_formatters=<function <lambda>>*, *more_predicates=<function <lambda>>*, *default_formatter='str'*)

Compile the template string, calling methods on the 'program builder'.

Args:

template_str: The template string. It should not have any compilation options in the header – those are parsed by FromString/FromFile

builder: The interface of `_ProgramBuilder` isn't fixed. Use at your own risk.

meta: The metacharacters to use, e.g. '{}', '[]'.

more_formatters:

Something that can map format strings to formatter functions. One of:

- A plain dictionary of names -> functions e.g. {'html': cgi.escape}
- A higher-order function which takes format strings and returns formatter functions. Useful for when formatters have parsed arguments.
- A FunctionRegistry instance for the most control. This allows formatters which takes contexts as well.

more_predicates: Like `more_formatters`, but for predicates.

default_formatter: The formatter to use for substitutions that are missing a formatter. The 'str' formatter the "default default" – it just tries to convert the context value to a string in some unspecified manner.

Returns: The compiled program (obtained from the builder)

Raises: The various subclasses of `CompilationError`. For example, if `default_formatter=None`, and a variable is missing a formatter, then `MissingFormatter` is raised.

This function is public so it can be used by other tools, e.g. a syntax checking tool run before submitting a template to source control.

`autotest.client.shared.jsontemplate.FromString` (*s*, *more_formatters=<function <lambda>>*, *_constructor=None*)

Like `FromFile`, but takes a string.

`autotest.client.shared.jsontemplate.FromFile` (*f*, *more_formatters=<function <lambda>>*, *_constructor=None*)

Parse a template from a file, using a simple file format.

This is useful when you want to include template options in a data file, rather than in the source code.

The format is similar to HTTP or E-mail headers. The first lines of the file can specify template options, such as the metacharacters to use. One blank line must separate the options from the template body.

Example:

default-formatter: none meta: {{{}} format-char: : <blank line required> Template goes here: {{{variable:html}}}

Args: f: A file handle to read from. Caller is responsible for opening and closing it.

class `autotest.client.shared.jsontemplate.Template` (*template_str*, *builder=None*, *undefined_str=None*, ***compile_options*)

Bases: `object`

Represents a compiled template.

Like many template systems, the template string is compiled into a program, and then it can be expanded any number of times. For example, in a web app, you can compile the templates once at server startup, and use the `expand()` method at request handling time. `expand()` uses the compiled representation.

There are various options for controlling parsing – see `CompileTemplate`. Don't go crazy with metacharacters. `{}`, `[]`, `{{}}` or `<>` should cover nearly any circumstance, e.g. generating HTML, CSS XML, JavaScript, C programs, text files, etc.

expand (**args*, ***kwargs*)

Expands the template with the given data dictionary, returning a string.

This is a small wrapper around `render()`, and is the most convenient interface.

Args: The JSON data dictionary. Like the builtin `dict()` constructor, it can take a single dictionary as a positional argument, or arbitrary keyword arguments.

Returns: The return value could be a `str()` or `unicode()` instance, depending on the the type of the template string passed in, and what the types the strings in the dictionary are.

render (*data_dict*, *callback*)

Low level method to expands the template piece by piece.

Args: *data_dict*: The JSON data dictionary. *callback*: A callback which should be called with each expanded token.

Example: You can pass 'f.write' as the callback to write directly to a file handle.

tokenstream (*data_dict*)

Yields a list of tokens resulting from expansion.

This may be useful for WSGI apps. NOTE: In the current implementation, the entire expanded template must be stored memory.

NOTE: This is a generator, but JavaScript doesn't have generators.

`autotest.client.shared.jsontemplate.expand` (*template_str*, *dictionary*, ***kwargs*)

Free function to expands a template string with a data dictionary.

This is useful for cases where you don't care about saving the result of compilation (similar to `re.match('.', s)` vs `DOT_STAR.match(s)`)

kernel_versions Module

`autotest.client.shared.kernel_versions.is_release_candidate` (*version*)

`autotest.client.shared.kernel_versions.is_released_kernel` (*version*)

`autotest.client.shared.kernel_versions.version_choose_config` (*version*, *candidates*)

`autotest.client.shared.kernel_versions.version_encode` (*version*)

`autotest.client.shared.kernel_versions.version_len` (*version*)

`autotest.client.shared.kernel_versions.version_limit` (*version, n*)

log Module

`autotest.client.shared.log.is_failure` (*status*)

`autotest.client.shared.log.is_valid_status` (*status*)

`autotest.client.shared.log.log_and_ignore_errors` (*msg*)

A decorator for wrapping functions in a ‘log exception and ignore’ try-except block.

`autotest.client.shared.log.record` (*fn*)

Generic method decorator for logging calls under the assumption that return=GOOD, exception=FAIL. The method determines parameters as:

```
subdir = self.subdir if it exists, or None
operation = "class name"."method name"
status = None on GOOD, str(exception) on FAIL
```

The object using this method must have a job attribute for the logging to actually occur, otherwise the logging will silently fail.

Logging can explicitly be disabled for a call by passing a `logged=False` parameter

logging_config Module

class `autotest.client.shared.logging_config.AllowBelowSeverity` (*level*)

Bases: `logging.Filter`

Allows only records less severe than a given level (the opposite of what the normal logging level filtering does).

filter (*record*)

class `autotest.client.shared.logging_config.LoggingConfig` (*use_console=True*)

Bases: `object`

add_console_handlers ()

add_debug_file_handlers (*log_dir, log_name=None*)

add_file_handler (*file_path, level=10, log_dir=None*)

add_stream_handler (*stream, level=10*)

configure_logging (*use_console=True, verbose=False*)

console_formatter = <logging.Formatter object>

file_formatter = <logging.Formatter object>

classmethod `get_autotest_root` ()

classmethod `get_server_log_dir` ()

classmethod `get_timestamped_log_name` (*base_name*)

global_level = 10

stderr_level = 40

stdout_level = 20

class `autotest.client.shared.logging_config.TestingConfig` (*use_console=True*)

Bases: `autotest.client.shared.logging_config.LoggingConfig`

```
add_file_handler (*args, **kwargs)
add_stream_handler (*args, **kwargs)
configure_logging (**kwargs)
```

logging_manager Module

```
class autotest.client.shared.logging_manager.FdRedirectionLoggingManager
```

Bases: *autotest.client.shared.logging_manager.LoggingManager*

A simple extension of `LoggingManager` to use `FdRedirectionStreamManagers`, so that managed streams have their underlying FDs redirected.

STREAM_MANAGER_CLASS

alias of `_FdRedirectionStreamManager`

```
start_logging ()
```

```
undo_redirect ()
```

```
class autotest.client.shared.logging_manager.LoggingFile (prefix='', level=10, logger=<logging.RootLogger object>)
```

Bases: `object`

File-like object that will receive messages pass them to the logging infrastructure in an appropriate way.

```
flush ()
```

```
isatty ()
```

```
write (data)
```

” Writes data only if it constitutes a whole line. If it’s not the case, store it in a buffer and wait until we have a complete line. :param data - Raw data (a string) that will be processed.

```
writelines (lines)
```

” Writes iterable of lines

Parameters lines – An iterable of strings that will be processed.

```
class autotest.client.shared.logging_manager.LoggingManager
```

Bases: `object`

Manages a stack of logging configurations, allowing clients to conveniently add and remove logging destinations. Also keeps a list of `StreamManagers` to easily direct streams into the logging module.

STREAM_MANAGER_CLASS

alias of `_StreamManager`

```
logging_config_object = None
```

```
manage_stderr ()
```

```
manage_stdout ()
```

```
manage_stream (stream, level, stream_setter)
```

Tells this manager to manage the given stream. All data written to the stream will be directed to the logging module instead. Must be called before `start_logging()`.

Parameters

- **stream** – stream to manage
- **level** – level to log data written to this stream

- **stream_setter** – function to set the stream to a new object

redirect (*filename*)

Redirect output to the specified file

redirect_to_stream (*stream*)

Redirect output to the given stream

restore ()

Same as `undo_redirect()`. For backwards compatibility with `fd_stack`.

start_logging ()

Begin capturing output to the logging module.

stop_logging ()

Restore output to its original state.

tee_redirect (*filename*, *level=None*)

Tee output to the specified file

tee_redirect_debug_dir (*debug_dir*, *log_name=None*, *tag=None*)

Tee output to a full new set of debug logs in the given directory.

tee_redirect_to_stream (*stream*)

Tee output to the given stream

undo_redirect ()

Undo the last redirection (that hasn't yet been undone).

If any subprocesses have been launched since the redirection was performed, they must have ended by the time this is called. Otherwise, this will hang waiting for the logging subprocess to end.

```
class autotest.client.shared.logging_manager.SortingLogFile (prefix='',
                                                         level_list=[('ERROR',
                                                         40), ('WARN',
                                                         30), ('INFO',
                                                         20), ('DE-
                                                         BUG', 10)], log-
                                                         ger=<logging.RootLogger
                                                         object>)
```

Bases: `autotest.client.shared.logging_manager.LoggingFile`

File-like object that will receive messages and pass them to the logging infrastructure. It decides where to pass each line by applying a regex to it and seeing which level it matched.

```
autotest.client.shared.logging_manager.configure_logging (logging_config,
                                                         **kwargs)
```

Configure the logging module using the specific configuration object, which should be an instance of `logging_config.LoggingConfig` (usually of a subclass). Any keyword args will be passed to the object's `configure_logging()` method.

Every entry point should call this method at application startup.

```
autotest.client.shared.logging_manager.do_not_report_as_logging_caller (func)
```

Decorator to annotate functions we will tell logging not to log.

```
autotest.client.shared.logging_manager.get_logging_manager (manage_stdout_and_stderr=False,
                                                         redirect_fds=False)
```

Create a `LoggingManager` that's managing `sys.stdout` and `sys.stderr`.

Every entry point that wants to capture `stdout/stderr` and/or use `LoggingManager` to manage a stack of destinations should call this method at application startup.

magic Module

Library used to determine a file MIME type by its magic number, it doesn't have any external dependencies. Based on work of Jason Petrone (jp_py@jsnp.net), adapted to autotest.

Command Line Usage: Running as `'python magic.py file_path'` will print a mime string (or just a description) of the file present on `file_path`.

API Usage: `magic.guess_type(file_path)` - Returns a description of what the file on path 'file' contains. This function name was chosen due to a similar function on python standard library 'mimetypes'.

@license: GPL v2 :copyright: Jason Petrone (jp_py@jsnp.net) 2000 :copyright: Lucas Meneghel Rodrigues (lmr@redhat.com) 2010 @see: <http://www.jsnp.net/code/magic.py>

class `autotest.client.shared.magic.MagicLoggingConfig` (*use_console=True*)

Bases: `autotest.client.shared.logging_config.LoggingConfig`

configure_logging (*results_dir=None, verbose=False*)

class `autotest.client.shared.magic.MagicTest` (*offset, t, op, value, msg, mask=None*)

Bases: `object`

Compile a magic database entry so it can be compared with data read from files.

compare (*data*)

Compare data read from the file with the expected data for this particular mime type register.

Parameters *data* – Data read from the file.

test (*data*)

Compare data read from file with self.value if operator is '='.

Parameters *data* – Data read from the file.

Returns None if no match between data and expected value string. Else, print matching mime type information.

`autotest.client.shared.magic.guess_type` (*filename*)

Guess the mimetype of a file based on its filename.

Parameters *filename* – File name.

Returns Mimetype string or description, when appropriate mime not available.

mail Module

Notification email library.

Aims to replace a bunch of different email module wrappers previously used.

class `autotest.client.shared.mail.EmailNotificationManager` (*module='scheduler'*)

Bases: `object`

Email notification facility, for use in things like the autotest scheduler.

This facility can use values defined in the autotest settings (`global_config.ini`) to conveniently send notification emails to the admin of an autotest module.

enqueue_admin (*subject, message*)

Enqueue an email to the test grid admin.

enqueue_exception_admin (*reason*)

Enqueue an email containing an exception to the test grid admin.

send (*to_string*, *subject*, *body*)

Send emails to the addresses listed in *to_string*.

to_string is split into a list which can be delimited by any of: ‘;’, ‘,’ or any whitespace

send_admin (*subject*, *body*)

Send an email to this grid admin.

send_queued_admin ()

Send all queued emails to the test grid admin.

set_module (*module*)

Change the name of the module we’re notifying for.

`autotest.client.shared.mail.send` (*from_address*, *to_addresses*, *cc_addresses*, *subject*, *body*,
smtp_info, *html=None*)

Send out an email.

Args: *from_address*: The email address to put in the “From:” field. *to_addresses*: Either a single string or an iterable of

strings to put in the “To:” field of the email.

cc_addresses: Either a single string of an iterable of strings to put in the “Cc:” field of the email.

subject: The email subject. *body*: The body of the email. there’s no special

handling of encoding here, so it’s safest to stick to 7-bit ASCII text.

smtp_info: Dictionary with SMTP info. *html*: Optional HTML content of the message.

mock Module

```
class autotest.client.shared.mock.Mock (spec=None,          side_effect=None,          re-
                                     return_value=sentinel.DEFAULT,          wraps=None,
                                     name=None,          spec_set=None,          par-
                                     ent=None,          _spec_state=None,          _new_name='',
                                     _new_parent=None, **kwargs)
```

Bases: `autotest.client.shared.mock.CallableMixin`, `autotest.client.shared.mock.NonCallableMock`

Create a new *Mock* object. *Mock* takes several optional arguments that specify the behaviour of the *Mock* object:

- *spec*: This can be either a list of strings or an existing object (a class or instance) that acts as the specification for the mock object. If you pass in an object then a list of strings is formed by calling `dir` on the object (excluding unsupported magic attributes and methods). Accessing any attribute not in this list will raise an *AttributeError*.

If *spec* is an object (rather than a list of strings) then `mock.__class__` returns the class of the *spec* object. This allows mocks to pass *isinstance* tests.

- *spec_set*: A stricter variant of *spec*. If used, attempting to *set* or get an attribute on the mock that isn’t on the object passed as *spec_set* will raise an *AttributeError*.
- *side_effect*: A function to be called whenever the *Mock* is called. See the *side_effect* attribute. Useful for raising exceptions or dynamically changing return values. The function is called with the same arguments as the mock, and unless it returns *DEFAULT*, the return value of this function is used as the return value.

Alternatively *side_effect* can be an exception class or instance. In this case the exception will be raised when the mock is called.

If *side_effect* is an iterable then each call to the mock will return the next value from the iterable. If any of the members of the iterable are exceptions they will be raised instead of returned.

- return_value*: The value returned when the mock is called. By default this is a new Mock (created on first access). See the *return_value* attribute.
- wraps*: Item for the mock object to wrap. If *wraps* is not None then calling the Mock will pass the call through to the wrapped object (returning the real result). Attribute access on the mock will return a Mock object that wraps the corresponding attribute of the wrapped object (so attempting to access an attribute that doesn't exist will raise an *AttributeError*).

If the mock has an explicit *return_value* set then calls are not passed to the wrapped object and the *return_value* is returned instead.

- name*: If the mock has a name then it will be used in the repr of the mock. This can be useful for debugging. The name is propagated to child mocks.

Mocks can also be called with arbitrary keyword arguments. These will be used to set attributes on the mock after it is created.

```
class autotest.client.shared.mock.MagicMock(*args, **kw)
Bases: autotest.client.shared.mock.MagicMixin, autotest.client.shared.mock.Mock
```

MagicMock is a subclass of Mock with default implementations of most of the magic methods. You can use MagicMock without having to configure the magic methods yourself.

If you use the *spec* or *spec_set* arguments then *only* magic methods that exist in the spec will be created.

Attributes and the return value of a *MagicMock* will also be *MagicMocks*.

```
mock_add_spec(spec, spec_set=False)
```

Add a spec to a mock. *spec* can either be an object or a list of strings. Only attributes on the *spec* can be fetched as attributes from the mock.

If *spec_set* is True then only attributes on the spec can be set.

```
autotest.client.shared.mock.patch(target, new=sentinel.DEFAULT, spec=None, create=False,
                                  spec_set=None, autospec=None, new_callable=None,
                                  **kwargs)
```

patch acts as a function decorator, class decorator or a context manager. Inside the body of the function or with statement, the *target* is patched with a *new* object. When the function/with statement exits the patch is undone.

If *new* is omitted, then the target is replaced with a *MagicMock*. If *patch* is used as a decorator and *new* is omitted, the created mock is passed in as an extra argument to the decorated function. If *patch* is used as a context manager the created mock is returned by the context manager.

target should be a string in the form '*package.module.ClassName*'. The *target* is imported and the specified object replaced with the *new* object, so the *target* must be importable from the environment you are calling *patch* from. The target is imported when the decorated function is executed, not at decoration time.

The *spec* and *spec_set* keyword arguments are passed to the *MagicMock* if patch is creating one for you.

In addition you can pass *spec=True* or *spec_set=True*, which causes patch to pass in the object being mocked as the *spec/spec_set* object.

new_callable allows you to specify a different class, or callable object, that will be called to create the *new* object. By default *MagicMock* is used.

A more powerful form of *spec* is *autospec*. If you set *autospec=True* then the mock will be created with a spec from the object being replaced. All attributes of the mock will also have the spec of the corresponding attribute of the object being replaced. Methods and functions being mocked will have their arguments checked and will

raise a *TypeError* if they are called with the wrong signature. For mocks replacing a class, their return value (the ‘instance’) will have the same spec as the class.

Instead of *autospec=True* you can pass *autospec=some_object* to use an arbitrary object as the spec instead of the one being replaced.

By default *patch* will fail to replace attributes that don’t exist. If you pass in *create=True*, and the attribute doesn’t exist, *patch* will create the attribute for you when the patched function is called, and delete it again afterwards. This is useful for writing tests against attributes that your production code creates at runtime. It is off by default because it can be dangerous. With it switched on you can write passing tests against APIs that don’t actually exist!

Patch can be used as a *TestCase* class decorator. It works by decorating each test method in the class. This reduces the boilerplate code when your test methods share a common patchings set. *patch* finds tests by looking for method names that start with *patch.TEST_PREFIX*. By default this is *test*, which matches the way *unittest* finds tests. You can specify an alternative prefix by setting *patch.TEST_PREFIX*.

Patch can be used as a context manager, with the *with* statement. Here the patching applies to the indented block after the *with* statement. If you use “as” then the patched object will be bound to the name after the “as”; very useful if *patch* is creating a mock object for you.

patch takes arbitrary keyword arguments. These will be passed to the *Mock* (or *new_callable*) on construction.

patch.dict(...), *patch.multiple(...)* and *patch.object(...)* are available for alternate use-cases.

`autotest.client.shared.mock.call`

A tuple for holding the results of a call to a mock, either in the form *(args, kwargs)* or *(name, args, kwargs)*.

If *args* or *kwargs* are empty then a call tuple will compare equal to a tuple without those values. This makes comparisons less verbose:

```
_Call(('name', (), {})) == ('name',)
_Call(('name', (1,)), {}) == ('name', (1,))
_Call((), {'a': 'b'}) == ({'a': 'b'},)
```

The *_Call* object provides a useful shortcut for comparing with *call*:

```
_Call(((1, 2), {'a': 3})) == call(1, 2, a=3)
_Call(('foo', (1, 2), {'a': 3})) == call.foo(1, 2, a=3)
```

If the *_Call* has no name then it will match any name.

`autotest.client.shared.mock.create_autospec` (*spec, spec_set=False, instance=False, _parent=None, _name=None, **kwargs*)

Create a mock object using another object as a spec. Attributes on the mock will use the corresponding attribute on the *spec* object as their spec.

Functions or methods being mocked will have their arguments checked to check that they are called with the correct signature.

If *spec_set* is *True* then attempting to set attributes that don’t exist on the spec object will raise an *AttributeError*.

If a class is used as a spec then the return value of the mock (the instance of the class) will have the same spec. You can use a class as the spec for an instance object by passing *instance=True*. The returned mock will only be callable if instances of the mock are callable.

create_autospec also takes arbitrary keyword arguments that are passed to the constructor of the created mock.

```
class autotest.client.shared.mock.NonCallableMock (spec=None, wraps=None,
name=None, spec_set=None, parent=None, _spec_state=None,
_new_name='', _new_parent=None,
**kwargs)
```

Bases: autotest.client.shared.mock.Base

A non-callable version of *Mock*

assert_any_call (*args, **kwargs)

assert the mock has been called with the specified arguments.

The assert passes if the mock has *ever* been called, unlike *assert_called_with* and *assert_called_once_with* that only pass if the call is the most recent one.

assert_called_once_with (_mock_self, *args, **kwargs)

assert that the mock was called exactly once and with the specified arguments.

assert_called_with (_mock_self, *args, **kwargs)

assert that the mock was called with the specified arguments.

Raises an AssertionError if the args and keyword args passed in are different to the last call to the mock.

assert_has_calls (calls, any_order=False)

assert the mock has been called with the specified calls. The *mock_calls* list is checked for the calls.

If *any_order* is False (the default) then the calls must be sequential. There can be extra calls before or after the specified calls.

If *any_order* is True then the calls can be in any order, but they must all appear in *mock_calls*.

attach_mock (mock, attribute)

Attach a mock as an attribute of this one, replacing its name and parent. Calls to the attached mock will be recorded in the *method_calls* and *mock_calls* attributes of this one.

call_args

call_args_list

call_count

called

configure_mock (**kwargs)

Set attributes on the mock through keyword arguments.

Attributes plus return values and side effects can be set on child mocks using standard dot notation and unpacking a dictionary in the method call:

```
>>> attrs = {'method.return_value': 3, 'other.side_effect': KeyError}
>>> mock.configure_mock(**attrs)
```

mock_add_spec (spec, spec_set=False)

Add a spec to a mock. *spec* can either be an object or a list of strings. Only attributes on the *spec* can be fetched as attributes from the mock.

If *spec_set* is True then only attributes on the spec can be set.

mock_calls

reset_mock ()

Restore the mock object to its initial state.

return_value

side_effect

class `autotest.client.shared.mock.NonCallableMagicMock` (**args, **kw*)
 Bases: `autotest.client.shared.mock.MagicMixin`, `autotest.client.shared.mock.NonCallableMock`

A version of *MagicMock* that isn't callable.

mock_add_spec (*spec, spec_set=False*)
 Add a spec to a mock. *spec* can either be an object or a list of strings. Only attributes on the *spec* can be fetched as attributes from the mock.

If *spec_set* is True then only attributes on the spec can be set.

`autotest.client.shared.mock.mock_open` (*mock=None, read_data=''*)
 A helper function to create a mock to replace the use of *open*. It works for *open* called directly or used as a context manager.

The *mock* argument is the mock object to configure. If *None* (the default) then a *MagicMock* will be created for you, with the API limited to methods or attributes available on standard file handles.

read_data is a string for the *read* method of the file handle to return. This is an empty string by default.

class `autotest.client.shared.mock.PropertyMock` (*spec=None, side_effect=None, return_value=sentinel.DEFAULT, wraps=None, name=None, spec_set=None, parent=None, _spec_state=None, _new_name='', _new_parent=None, **kwargs*)

Bases: `autotest.client.shared.mock.Mock`

A mock intended to be used as a property, or other descriptor, on a class. *PropertyMock* provides `__get__` and `__set__` methods so you can specify a return value when it is fetched.

Fetching a *PropertyMock* instance from an object calls the mock, with no args. Setting it calls the mock with the value being set.

openvswitch Module

class `autotest.client.shared.openvswitch.OpenVSwitch` (*tmpdir, db_path=None, db_socket=None, db_pidfile=None, ovs_pidfile=None, schema=None, db_in_stall_prefix=None*)

Bases: `autotest.client.shared.openvswitch.OpenVSwitchSystem`

OpenVSwitch class.

clean ()

init_db ()

init_new ()
 Create new dbfile without any configuration.

start_ovs_vswitchd ()

class `autotest.client.shared.openvswitch.OpenVSwitchControl`

Bases: `object`

Class select the best matches control class for installed version of OpenVSwitch.

OpenVSwitich parameters are described in man ovs-vswitchd.conf.db

add_br (*br_name*)

add_port (*br_name, port_name*)

add_port_tag (*port_name, tag*)

add_port_trunk (*port_name, trunk*)

br_exist (*br_name*)

check_port_in_br (*br_name, port_name*)

static convert_version_to_int (*version*)

Parameters **version** – (int) Converted from version string 1.4.0 => int 140

del_br (*br_name*)

del_port (*br_name, port_name*)

classmethod get_version ()

Get version of installed OpenVSwitich.

Returns Version of OpenVSwitich.

list_br ()

set_vlanmode (*port_name, vlan_mode*)

status ()

class autotest.client.shared.openvswitch.**OpenVSwitchControlCli**

Bases: *autotest.client.shared.openvswitch.OpenVSwitchControl*, *autotest.client.shared.utils.VersionableClass*

Class select the best matches control class for installed version of OpenVSwitch.

class autotest.client.shared.openvswitch.**OpenVSwitchControlCli_140**

Bases: *autotest.client.shared.openvswitch.OpenVSwitchControlCli*, *autotest.client.shared.utils.VersionableClass*

Don't use this class directly. This class is automatically selected by OpenVSwitchControl.

add_br (*br_name*)

add_fake_br (*br_name, parent, vlan*)

add_port (*br_name, port_name*)

add_port_tag (*port_name, tag*)

add_port_trunk (*port_name, trunk*)

Parameters **trunk** – list of vlans id.

br_exist (*br_name*)

del_br (*br_name*)

del_port (*br_name, port_name*)

classmethod is_right_version (*version*)

Check condition for select control class.

Parameters **version** – version of OpenVSwitich

list_br ()

list_ports (*br_name*)

ovs_vsctl (*parms, ignore_status=False*)

port_to_br (*port_name*)

Return bridge which contain port.

Parameters **port_name** – Name of port.

Returns Bridge name or None if there is no bridge which contain port.

set_vlanmode (*port_name, vlan_mode*)

status ()

class `autotest.client.shared.openvswitch.OpenVSwitchControlDB`

Bases: `autotest.client.shared.openvswitch.OpenVSwitchControl`, `autotest.client.shared.utils.VersionableClass`

Class select the best matches control class for installed version of OpenVSwitch.

class `autotest.client.shared.openvswitch.OpenVSwitchControlDB_140`

Bases: `autotest.client.shared.openvswitch.OpenVSwitchControlDB`, `autotest.client.shared.utils.VersionableClass`

Don't use this class directly. This class is automatically selected by OpenVSwitchControl.

classmethod **is_right_version** (*version*)

Check condition for select control class.

Parameters **version** – version of OpenVSwitch

class `autotest.client.shared.openvswitch.OpenVSwitchSystem` (*db_path=None, db_socket=None, db_pidfile=None, ovs_pidfile=None, dbschema=None, install_prefix=None*)

Bases: `autotest.client.shared.openvswitch.OpenVSwitchControlCli`, `autotest.client.shared.openvswitch.OpenVSwitchControlDB`

OpenVSwitch class.

check ()

check_db_daemon ()

Check if OVS daemon is started correctly.

check_db_file ()

Check if db_file exists.

check_db_socket ()

Check if db socket exists.

check_switch_daemon ()

Check if OVS daemon is started correctly.

clean ()

Empty cleanup function

init_system ()

Create new dbfile without any configuration.

is_installed ()

Check if OpenVSwitch is already installed in system on default places.

Returns Version of OpenVSwitch.

```
class autotest.client.shared.openvswitch.ServiceManager
    Bases: autotest.client.shared.openvswitch.ServiceManagerInterface

class autotest.client.shared.openvswitch.ServiceManagerInterface
    Bases: autotest.client.shared.utils.VersionableClass

    classmethod get_version ()
        Get version of ServiceManager. :return: Version of ServiceManager.

    restart (service_name)

    start (service_name)

    status (service_name)

    stop (service_name)

class autotest.client.shared.openvswitch.ServiceManagerSystemD
    Bases: autotest.client.shared.openvswitch.ServiceManagerInterface, autotest.client.shared.utils.VersionableClass

    classmethod is_right_version (version)

    restart (service_name)

    start (service_name)

    status (service_name)

    stop (service_name)

class autotest.client.shared.openvswitch.ServiceManagerSysvinit
    Bases: autotest.client.shared.openvswitch.ServiceManagerInterface, autotest.client.shared.utils.VersionableClass

    classmethod is_right_version (version)

    restart (service_name)

    start (service_name)

    stop (service_name)
```

packages Module

```
class autotest.client.shared.packages.PackageManager (pkgmgr_dir, host-
                                                    name=None, repo_urls=None,
                                                    upload_paths=None,
                                                    do_locking=True,
                                                    run_function=<function
                                                    run>, run_function_args=[],
                                                    run_function_dargs={})
    Bases: autotest.client.shared.base_packages.BasePackageManager
```

pidfile Module

```
class autotest.client.shared.pidfile.PidFileManager (label, results_dir)
    Bases: object

    close_file (exit_code, signal_code=0)
```

`open_file()`

profiler_manager Module

exception `autotest.client.shared.profiler_manager.ProfilerNotPresentError` (*name*,
**args*,
***dargs*)

Bases: `autotest.client.shared.error.JobError`

class `autotest.client.shared.profiler_manager.profiler_manager` (*job*)
 Bases: `object`

active ()
 Returns True if profilers are present and started, False otherwise

add (*profiler*, **args*, ***dargs*)
 Add a profiler

before_start (*test*)
 Override to do any setup needed before actually starting the profilers (this function is called before calling `test.before_run_once()` and `profilers.start()` in a profiled run).

current_profilers ()
 Returns a set of the currently enabled profilers

delete (*profiler*)
 Remove a profiler

load_profiler (*profiler*, *args*, *dargs*)
 Given a name and args, loads a profiler, initializes it with the required arguments, and returns an instance of it. Raises a `ProfilerNotPresentError` if the module isn't found.

only ()
 Returns True if job is supposed to be run only with profiling turned on, False otherwise

present ()
 Indicates if any profilers are enabled

report (*test*)
 Report on all enabled profilers

set_only (*value*)
 Changes the flag which determines whether or not the job is to be run without profilers at all

start (*test*)
 Start all enabled profilers

stop (*test*)
 Stop all enabled profilers

progressbar Module

Basic text progress bar without fancy curses features

class `autotest.client.shared.progressbar.ProgressBar` (*minimum=0*, *maximum=100*,
width=77, *title=''*)

Displays interactively the progress of a given task

Inspired/adapted from [code.activestate.com recipe #168639](http://code.activestate.com/recipe/168639/)

DEFAULT_WIDTH = 77

get_screen_text ()
Builds the actual progress bar text

increment (*increment*, *update_screen=True*)
Increments the current amount value

update (*amount*, *update_screen=True*)
Performs sanity checks and update the current amount

update_screen ()
Prints the updated text to the screen

report Module

Module used to parse the autotest job status file and generate a JSON file.

Optionally, we can also generate reports (HTML)

exception `autotest.client.shared.report.InvalidAutotestResultDirError` (*directory*)
Bases: `exceptions.Exception`

exception `autotest.client.shared.report.InvalidOutputDirError` (*directory*)
Bases: `exceptions.Exception`

class `autotest.client.shared.report.ReportLoggingConfig` (*use_console=True*)
Bases: `autotest.client.shared.logging_config.LoggingConfig`

Used with the sole purpose of providing convenient logging setup for this program.

configure_logging (*results_dir=None*, *verbose=False*)

class `autotest.client.shared.report.ReportOptionParser`
Bases: `optparse.OptionParser`

`autotest.client.shared.report.generate_html_report` (*results_dir*, *relative_links=True*)
Render a job report HTML.

All CSS and javascript are inlined, for more convenience.

Parameters `results_dir` – Path to the results directory.

`autotest.client.shared.report.generate_json_file` (*results_dir*, *relative_links=True*)
Generate a JSON file with autotest job summary on a given results directory

Parameters `results_dir` – Path to the results directory.

`autotest.client.shared.report.get_info_file` (*filename*)
Gets the contents of an autotest info file.

It also and highlights the file contents with possible problems.

Parameters `filename` – Info file path.

`autotest.client.shared.report.parse_results_dir` (*results_dir*, *relative_links=True*)
Parse a top level status file and produce a dictionary with job data.

Parameters `dirname` – Autotest results directory path

Returns Dictionary with job data.

`autotest.client.shared.report.write_html_report` (*results_dir*, *report_path=None*, *encoding='utf8'*)

Write an HTML file at `report_path`, with job data summary.

If no `report_path` specified, generate one at `results_dir/job_report.html`.

Parameters

- **results_dir** – Directory with test results.
- **report_path** – Path to a report file (optional).
- **encoding** – Encoding for output (optional).

service Module

`autotest.client.shared.service.ServiceManager` (*run=<function run>*)

Detect which init program is being used, init or systemd and return a class has methods to start/stop services.

Get the system service manager `service_manager = ServiceManager()`

Starting service/unit “ssh” `service_manager.start(“ssh”)`

Getting a list of available units `units = service_manager.list()`

Disabling and stopping a list of services `services_to_disable = [‘ntpd’, ‘httpd’]` for `s` in `services_to_disable`:

`service_manager.disable(s) service_manager.stop(s)`

Returns `SysVInitServiceManager` or `SystemdServiceManager`

Return type `_GenericServiceManager`

`autotest.client.shared.service.SpecificServiceManager` (*service_name, run=<function run>*)

Get the specific service manager for sshd `sshd = SpecificServiceManager(“sshd”) sshd.start() sshd.stop() sshd.reload() sshd.restart() sshd.condrestart() sshd.status() sshd.enable() sshd.disable() sshd.is_enabled()`

Parameters **service_name** (*str*) – systemd unit or init.d service to manager

Returns `SpecificServiceManager` that has start/stop methods

Return type `_SpecificServiceManager`

`autotest.client.shared.service.convert_systemd_target_to_runlevel` (*target*)

Convert systemd target to runlevel.

Parameters **target** (*str*) – systemd target

Returns `sys_v` runlevel

Return type *str*

Raises **ValueError** – when systemd target is unknown

`autotest.client.shared.service.convert_sysv_runlevel` (*level*)

Convert runlevel to systemd target.

Parameters **level** (*str or int*) – `sys_v` runlevel

Returns systemd target

Return type *str*

Raises **ValueError** – when runlevel is unknown

`autotest.client.shared.service.get_name_of_init` (*run=<function run>*)

Determine what executable is PID 1, aka init by checking `/proc/1/exe` This init detection will only run once and cache the return value.

Returns executable name for PID 1, aka init

Return type *str*

`autotest.client.shared.service.sys_v_init_command_generator` (*command*)
 Generate lists of command arguments for sys_v style inits.

Parameters `command` (*str*) – start,stop,restart, etc.

Returns list of commands to pass to `utils.run` or similar function

Return type *list*

`autotest.client.shared.service.sys_v_init_result_parser` (*command*)
 Parse results from sys_v style commands.

Parameters `command` (*str.*) – command.

Returns different from the command.

`command` is status: return true if service is running. `command` is `is_enabled`: return true if service is enabled.
`command` is list: return a dict from service name to status. `command` is others: return true if operate success.

`autotest.client.shared.service.systemd_command_generator` (*command*)

Generate list of command line argument strings for `systemctl`. One argument per string for compatibility

Popen
 WARNING: If `systemctl` detects that it is running on a tty it will use color, pipe to `$PAGER`, change column sizes and not truncate unit names. Use `--no-pager` to suppress pager output, or set `PAGER=cat` in the environment. You may need to take other steps to suppress color output. See https://bugzilla.redhat.com/show_bug.cgi?id=713567

Parameters `command` (*str*) – start,stop,restart, etc.

Returns list of command and arguments to pass to `utils.run` or similar functions

Return type *list*

`autotest.client.shared.service.systemd_result_parser` (*command*)
 Parse results from `systemd` style commands.

Parameters `command` (*str.*) – command.

Returns different from the command.

`command` is status: return true if service is running. `command` is `is_enabled`: return true if service is enabled.
`command` is list: return a dict from service name to status. `command` is others: return true if operate success.

settings Module

A singleton class for accessing global config values.

provides access to global configuration file.

class `autotest.client.shared.settings.Settings`

Bases: `object`

`check_stand_alone_client_run` ()

`config` = `None`

`config_file` = `'/home/docs/checkouts/readthedocs.org/user_builds/autotest/checkouts/stable/global_config.ini'`

`get_section_values` (*sections*)

Return a config parser object containing a single section of the global configuration, that can be later written to a file object.

Parameters `section` – Tuple with sections we want to turn into a config parser object.

Returns `ConfigParser`() object containing all the contents of sections.

get_value (*section, key, type=<type 'str'>, default=<object object>, allow_blank=False*)

merge_configs (*shadow_config*)

override_value (*section, key, new_value*)

Override a value from the config file with a new value.

parse_config_file ()

reset_values ()

Reset all values to those found in the config files (undoes all overrides).

running_stand_alone_client = False

set_config_files (*config_file='/home/docs/checkouts/readthedocs.org/user_builds/autotest/checkouts/stable/global_config'*
shadow_file='/home/docs/checkouts/readthedocs.org/user_builds/autotest/checkouts/stable/shadow_config')

shadow_file = '/home/docs/checkouts/readthedocs.org/user_builds/autotest/checkouts/stable/shadow_config.ini'

exception `autotest.client.shared.settings.SettingsError`

Bases: `autotest.client.shared.error.AutotestError`

exception `autotest.client.shared.settings.SettingsValueError`

Bases: `autotest.client.shared.settings.SettingsError`

software_manager Module

Software package management library.

This is an abstraction layer on top of the existing distributions high level package managers. It supports package operations useful for testing purposes, and multiple high level package managers (here called backends). If you want to make this lib to support your particular package manager/distro, please implement the given backend class.

author Higor Vieira Alves (halves@br.ibm.com)

author Lucas Meneghel Rodrigues (lmr@redhat.com)

author Ramon de Carvalho Valle (rcvalle@br.ibm.com)

copyright IBM 2008-2009

copyright Red Hat 2009-2010

class `autotest.client.shared.software_manager.AptBackend`

Bases: `autotest.client.shared.software_manager.DpkgBackend`

Implements the apt backend for software manager.

Set of operations for the apt package manager, commonly found on Debian and Debian based distributions, such as Ubuntu Linux.

add_repo (*repo*)

Add an apt repository.

Parameters **repo** – Repository string. Example: 'deb <http://archive.ubuntu.com/ubuntu/> maverick universe'

install (*name*)

Installs package [name].

Parameters **name** – Package name.

provides (*path*)

Return a list of packages that provide [path].

Parameters `path` – File path.

remove (*name*)

Remove package [name].

Parameters `name` – Package name.

remove_repo (*repo*)

Remove an apt repository.

Parameters `repo` – Repository string. Example: ‘deb <http://archive.ubuntu.com/ubuntu/> maverick universe’

upgrade (*name=None*)

Upgrade all packages of the system with eventual new versions.

Optionally, upgrade individual packages.

Parameters `name` (*str*) – optional parameter wildcard spec to upgrade

class `autotest.client.shared.software_manager.BaseBackend`

Bases: `object`

This class implements all common methods among backends.

install_what_provides (*path*)

Installs package that provides [path].

Parameters `path` – Path to file.

class `autotest.client.shared.software_manager.DpkgBackend`

Bases: `autotest.client.shared.software_manager.BaseBackend`

This class implements operations executed with the dpkg package manager.

dpkg is a lower level package manager, used by higher level managers such as apt and aptitude.

INSTALLED_OUTPUT = ‘install ok installed’

PACKAGE_TYPE = ‘deb’

check_installed (*name*)

list_all ()

List all packages available in the system.

list_files (*package*)

List files installed by package [package].

Parameters `package` – Package name.

Returns List of paths installed by package.

class `autotest.client.shared.software_manager.RpmBackend`

Bases: `autotest.client.shared.software_manager.BaseBackend`

This class implements operations executed with the rpm package manager.

rpm is a lower level package manager, used by higher level managers such as yum and zypper.

PACKAGE_TYPE = ‘rpm’

SOFTWARE_COMPONENT_QRY = ‘rpm %{NAME} %{VERSION} %{RELEASE} %{SIGMD5} %{ARCH}’

check_installed (*name, version=None, arch=None*)

Check if package [name] is installed.

Parameters

- **name** – Package name.
- **version** – Package version.
- **arch** – Package architecture.

list_all (*software_components=True*)

List all installed packages.

Parameters **software_components** – log in a format suitable for the SoftwareComponent schema

list_files (*name*)

List files installed on the system by package [name].

Parameters **name** – Package name.

class `autotest.client.shared.software_manager.SoftwareManager`

Bases: `object`

Package management abstraction layer.

It supports a set of common package operations for testing purposes, and it uses the concept of a backend, a helper class that implements the set of operations of a given package management tool.

class `autotest.client.shared.software_manager.SoftwareManagerLoggingConfig` (*use_console=True*)

Bases: `autotest.client.shared.logging_config.LoggingConfig`

Used with the sole purpose of providing logging setup for this program.

configure_logging (*results_dir=None, verbose=False*)

class `autotest.client.shared.software_manager.SystemInspector`

Bases: `object`

System inspector class.

This may grow up to include more complete reports of operating system and machine properties.

get_package_management ()

Determine the supported package management systems present on the system. If more than one package management system installed, try to find the best supported system.

class `autotest.client.shared.software_manager.YumBackend`

Bases: `autotest.client.shared.software_manager.RpmBackend`

Implements the yum backend for software manager.

Set of operations for the yum package manager, commonly found on Yellow Dog Linux and Red Hat based distributions, such as Fedora and Red Hat Enterprise Linux.

add_repo (*url*)

Adds package repository located on [url].

Parameters **url** – Universal Resource Locator of the repository.

install (*name*)

Installs package [name]. Handles local installs.

provides (*name*)

Returns a list of packages that provides a given capability.

Parameters **name** – Capability name (eg, 'foo').

remove (*name*)

Removes package [name].

Parameters name – Package name (eg. ‘ipython’).

remove_repo (*url*)

Removes package repository located on [url].

Parameters url – Universal Resource Locator of the repository.

upgrade (*name=None*)

Upgrade all available packages.

Optionally, upgrade individual packages.

Parameters name (*str*) – optional parameter wildcard spec to upgrade

class `autotest.client.shared.software_manager.ZypperBackend`

Bases: `autotest.client.shared.software_manager.RpmBackend`

Implements the zypper backend for software manager.

Set of operations for the zypper package manager, found on SUSE Linux.

add_repo (*url*)

Adds repository [url].

Parameters url – URL for the package repository.

install (*name*)

Installs package [name]. Handles local installs.

Parameters name – Package Name.

provides (*name*)

Searches for what provides a given file.

Parameters name – File path.

remove (*name*)

Removes package [name].

remove_repo (*url*)

Removes repository [url].

Parameters url – URL for the package repository.

upgrade (*name=None*)

Upgrades all packages of the system.

Optionally, upgrade individual packages.

Parameters name (*str*) – Optional parameter wildcard spec to upgrade

`autotest.client.shared.software_manager.install_distro_packages` (*distro_pkg_map*,
interactive=False)

Installs packages for the currently running distribution

This utility function checks if the currently running distro is a key in the `distro_pkg_map` dictionary, and if there is a list of packages set as its value.

If these conditions match, the packages will be installed using the software manager interface, thus the native packaging system if the currently running distro.

Parameters distro_pkg_map (*dict*) – mapping of distro name, as returned by `utils.get_os_vendor()`, to a list of package names

Returns True if any packages were actually installed, False otherwise

ssh_key Module**syncdata Module****test Module****class** autotest.client.shared.test.**Subtest**Bases: `object`

Collect result of subtest of main test.

clean()

Check if cleanup is defined.

For makes test fatal add before implementation of test method decorator `@subtest_nocleanup`**decorated()****failed = 0****classmethod** `get_full_text_result` (*format_func=None*)**Returns** string with text form of result**classmethod** `get_result` ()**Returns**

Result of subtests. Format:

`tuple(pass/fail,function_name,call_arguments)`**classmethod** `get_text_result` (*format_func=None*)**Returns** string with text form of result**classmethod** `has_failed` ()**Returns** If any of subtest not pass return True.**classmethod** `log_append` (*msg*)Add `log_append` to result output.**Parameters** `msg` – Test of `log_append`**passed = 0****result = []****static** `result_to_string` (*result*)

Format of result dict.

result = {

```

    'result' : "PASS" / "FAIL", 'name' : class name, 'args' : test's args, 'kargs' : test's kargs,
    'output' : return of test function,

```

}**Parameters** `result` – Result of test.**static** `result_to_string_debug` (*result*)**Parameters** `result` – Result of test.

runsubtest (*url*, **args*, ***dargs*)

Execute another autotest test from inside the current test's scope.

Parameters

- **test** – Parent test.
- **url** – Url of new test.
- **tag** – Tag added to test name.
- **args** – Args for subtest.
- **dargs** – Dictionary with args for subtest.

@iterations: Number of subtest iterations. @profile_only: If true execute one profiled run.

test ()

Check if test is defined.

For makes test fatal add before implementation of test method decorator @subtest_fatal

class autotest.client.shared.test.**base_test** (*job*, *bindir*, *outputdir*)

Bases: `object`

after_run_once ()

Called after every run_once (including from a profiled run when it's called after stopping the profilers).

analyze_perf_constraints (*constraints*)

assert_ (*expr*, *msg*='Assertion failed.')

before_run_once ()

Override in tests that need it, will be called before any run_once() call including the profiling run (when it's called before starting the profilers).

cleanup ()

configure_crash_handler ()

crash_handler_report ()

drop_caches_between_iterations ()

execute (*iterations*=None, *test_length*=None, *profile_only*=None, *_get_time*=<built-in function time>, *postprocess_profiled_run*=None, *constraints*=(), **args*, ***dargs*)

This is the basic execute method for the tests inherited from base_test. If you want to implement a benchmark test, it's better to implement the run_once function, to cope with the profiling infrastructure. For other tests, you can just override the default implementation.

Parameters

- **test_length** – The minimum test length in seconds. We'll run the run_once function for a number of times large enough to cover the minimum test length.
- **iterations** – A number of iterations that we'll run the run_once function. This parameter is incompatible with test_length and will be silently ignored if you specify both.
- **profile_only** – If true run X iterations with profilers enabled. If false run X iterations and one with profiling if profiles are enabled. If None, default to the value of job.default_profile_only.
- **_get_time** – [time.time] Used for unit test time injection.
- **postprocess_profiled_run** – Run the postprocessing for the profiled run.

initialize ()

network_destabilizing = False

postprocess ()

postprocess_iteration ()

preserve_srcdir = False

process_failed_constraints ()

register_after_iteration_hook (iteration_hook)

This is how we expect test writers to register an after_iteration_hook. This adds the method to the list of hooks which are executed after each iteration.

Parameters iteration_hook – Method to run after each iteration. A valid hook accepts a single argument which is the test object.

register_before_iteration_hook (iteration_hook)

This is how we expect test writers to register a before_iteration_hook. This adds the method to the list of hooks which are executed before each iteration.

Parameters iteration_hook – Method to run before each iteration. A valid hook accepts a single argument which is the test object.

run_once_profiling (postprocess_profiled_run, *args, **dargs)

setup ()

warmup (*args, **dargs)

write_attr_keyval (attr_dict)

write_iteration_keyval (attr_dict, perf_dict, tap_report=None)

write_perf_keyval (perf_dict)

write_test_keyval (attr_dict)

autotest.client.shared.test.runtest (job, url, tag, args, dargs, local_namespace={}, global_namespace={}, before_test_hook=None, after_test_hook=None, before_iteration_hook=None, after_iteration_hook=None)

autotest.client.shared.test.subtest_fatal (function)

Decorator which mark test critical. If subtest fails the whole test ends.

autotest.client.shared.test.subtest_nocleanup (function)

Decorator used to disable cleanup function.

utils Module

Convenience functions for use by tests or whomever.

NOTE: this is a mixin library that pulls in functions from several places Note carefully what the precedence order is

There's no really good way to do this, as this isn't a class we can do inheritance with, just a collection of static methods.

class autotest.client.shared.utils.AsyncJob (command, stdout_tee=None, stderr_tee=None, verbose=True, stdin=None, stderr_level=40, kill_func=None, close_fds=False)

Bases: *autotest.client.shared.utils.BgJob*

cleanup ()

`get_stderr()`

`get_stdout()`

`output_prepare(stdout_file=None, stderr_file=None)`

`process_output(stdout=True, final_read=False)`

`wait_for(timeout=None)`

Wait for the process to finish. When timeout is provided, process is safely destroyed after timeout. :param timeout: Acceptable timeout :return: results of this command

class `autotest.client.shared.utils.BgJob` (*command, stdout_tee=None, stderr_tee=None, verbose=True, stdin=None, stderr_level=40, close_fds=False*)

Bases: `object`

`cleanup()`

`output_prepare(stdout_file=None, stderr_file=None)`

`process_output(stdout=True, final_read=False)`

output_prepare must be called prior to calling this

class `autotest.client.shared.utils.CmdResult` (*command='', stdout='', stderr='', exit_status=None, duration=0*)

Bases: `object`

Command execution result.

command: String containing the command line itself exit_status: Integer exit code of the process stdout: String containing stdout of the process stderr: String containing stderr of the process duration: Elapsed wall clock time running the process

class `autotest.client.shared.utils.FileFieldMonitor` (*status_file, data_to_read, mode_diff, continuously=False, contlogging=False, separator=' + ', time_step=0.1*)

Bases: `object`

Monitors the information from the file and reports it's values.

It gather the information at start and stop of the measurement or continuously during the measurement.

class `Monitor` (*master*)

Bases: `threading.Thread`

Internal monitor class to ensure continuous monitor of monitored file.

`run()`

Start monitor in thread mode

`FileFieldMonitor.get_status()`

Returns Status of monitored process average value, time of test and array of monitored values and time step of continuous run.

`FileFieldMonitor.start()`

Start value monitor.

`FileFieldMonitor.stop()`

Stop value monitor.

class `autotest.client.shared.utils.ForAll`

Bases: `list`

class `autotest.client.shared.utils.ForAllP`

Bases: `list`

Parallel version of `ForAll`

class `autotest.client.shared.utils.ForAllPSE`

Bases: `list`

Parallel version of and suppress exception.

class `autotest.client.shared.utils.InterruptedThread` (*target, args=(), kwargs={}*)

Bases: `threading.Thread`

Run a function in a background thread.

join (*timeout=None, suppress_exception=False*)

Join the thread. If target raised an exception, re-raise it. Otherwise, return the value returned by target.

Parameters

- **timeout** – Timeout value to pass to `threading.Thread.join()`.
- **suppress_exception** – If True, don't re-raise the exception.

run ()

Run target (passed to the constructor). No point in calling this function directly. Call `start()` to make this function run in a new thread.

class `autotest.client.shared.utils.Statistic`

Bases: `object`

Class to display and collect average, max and min values of a given data set.

get_average ()

get_max ()

get_min ()

record (*value*)

Record new value to statistic.

class `autotest.client.shared.utils.SystemLoad` (*pids, advanced=False, time_step=0.1, cpu_cont=False, use_log=False*)

Bases: `object`

Get system and/or process values and return average value of load.

dump (*pids=[]*)

Get the status of monitoring. :param pids: List of PIDs you intend to control. Use `pids=[]` to control all defined PIDs.

return

tuple([cpu load], [memory load]):

(((PID1, (PID1_cpu_meas)), (PID2, (PID2_cpu_meas)), ...), [(PID1, (PID1_mem_meas)), (PID2, (PID2_mem_meas)), ...])

PID1_cpu_meas: `average_values[], test_time, cont_meas_values[[]], time_step`

PID1_mem_meas: `average_values[], test_time, cont_meas_values[[]], time_step`

where `average_values[]` are the measured values (`mem_free,swap,...`) which are described in `SystemLoad.__init__()-FileFieldMonitor`. `cont_meas_values[[]]` is a list of `average_values` in the sampling times.

get_cpu_status_string (*pids=[]*)

Convert status to string array. :param pids: List of PIDs you intend to control. Use pids=[] to control all defined PIDs.

Returns String format to table.

get_mem_status_string (*pids=[]*)

Convert status to string array. :param pids: List of PIDs you intend to control. Use pids=[] to control all defined PIDs.

Returns String format to table.

start (*pids=[]*)

Start monitoring of the process system usage. :param pids: List of PIDs you intend to control. Use pids=[] to control

all defined PIDs.

stop (*pids=[]*)

Stop monitoring of the process system usage. :param pids: List of PIDs you intend to control. Use pids=[] to control

all defined PIDs.

class autotest.client.shared.utils.**VersionableClass**

Bases: `object`

VersionableClass provides class hierarchy which automatically select right version of class. Class manipulation is used for this reason. By this reason is: Advantage) Only one version is working in one process. Class is changed in whole process. Disadvantage) Only one version is working in one process.

Example of usage (in `utils_unittest`):

```
class FooC(object):
    pass

#Not implemented get_version -> not used for versioning.
class VCP(FooC, VersionableClass):
    def __new__(cls, *args, **kwargs):
        VCP.master_class = VCP
        return super(VCP, cls).__new__(cls, *args, **kwargs)

    def foo(self):
        pass

class VC2(VCP, VersionableClass):
    @staticmethod
    def get_version():
        return "get_version_from_system"

    @classmethod
    def is_right_version(cls, version):
        if version is not None:
            if "version is satisfied":
                return True
        return False

    def func1(self):
```

```

        print "func1"

    def func2(self):
        print "func2"

# get_version could be inherited.
class VC3(VC2, VersionableClass):
    @classmethod
    def is_right_version(cls, version):
        if version is not None:
            if "version+1 is satisfied":
                return True
            return False

    def func2(self):
        print "func2_2"

class M(VCP):
    pass

m = M() # <- When class is constructed the right version is
        # automatically selected. In this case VC3 is selected.
m.func2() # call VC3.func2(m)
m.func1() # call VC2.func1(m)
m.foo() # call VC1.foo(m)

# When controlled "program" version is changed then is necessary call
check_repair_versions or recreate object.

m.check_repair_versions()

# priority of class. (change place where is method searched first in group
# of verisonable class.)

class PP(VersionableClass):
    def __new__(cls, *args, **kargs):
        PP.master_class = PP
        return super(PP, cls).__new__(cls, *args, **kargs)

class PP2(PP, VersionableClass):
    @staticmethod
    def get_version():
        return "get_version_from_system"

    @classmethod
    def is_right_version(cls, version):
        if version is not None:
            if "version is satisfied":
                return True
            return False

    def func1(self):
        print "PP func1"

class N(VCP, PP):
    pass

n = N()

```

```
n.func1() # -> "func2"

n.set_priority_class(PP, [VCP, PP])

n.func1() # -> "PP func1"
```

Necessary for using: 1) Subclass of versionable class must have implemented class methods `get_version` and `is_right_version`. These two methods are necessary for correct version section. Class without this method will be never chosen like suitable class.

2) Every class derived from `master_class` have to add to class definition inheritance from `VersionableClass`. Direct inheritance from `Versionable Class` is use like a mark for manipulation with `VersionableClass`.

3) Master of `VersionableClass` have to defined class variable `cls.master_class`.

classmethod `check_repair_versions` (*master_classes=None*)

Check version of versionable class and if version not match repair version to correct version.

Parameters `master_classes` (*list.*) – Check and repair only `master_class`.

classmethod `get_version` ()

Get version of installed OpenVSwitich. Must be re-implemented for in child class.

Returns Version or None when `get_version` is unsuccessful.

classmethod `is_right_version` (*version*)

Check condition for select control class. Function must be re-implemented in new `OpenVSwitchControl` class. Must be re-implemented for in child class.

Parameters `version` – version of OpenVSwitich

classmethod `set_priority_class` (*prioritized_class, group_classes*)

Set class priority. Limited only for change bases class priority inside one subclass. `__bases__` after that continue to another class.

```
autotest.client.shared.utils.archive_as_tarball(source_dir, dest_dir, tar-
ball_name=None, compression='bz2',
verbose=True)
```

Saves the given source directory to the given destination as a tarball

If the name of the archive is omitted, it will be taken from the `source_dir`. If it is an absolute path, `dest_dir` will be ignored. But, if both the destination directory and tarball anem is given, and the latter is not an absolute path, they will be combined.

For archiving directory `/tmp` in `/net/server/backup` as file `tmp.tar.bz2`, simply use:

```
>>> utils.archive_as_tarball('/tmp', '/net/server/backup')
```

To save the file it with a different name, say `host1-tmp.tar.bz2` and save it under `/net/server/backup`, use:

```
>>> utils.archive_as_tarball('/tmp', '/net/server/backup',
                             'host1-tmp')
```

To save with gzip compression instead (resulting in the file `/net/server/backup/host1-tmp.tar.gz`), use:

```
>>> utils.archive_as_tarball('/tmp', '/net/server/backup',
                             'host1-tmp', 'gz')
```

`autotest.client.shared.utils.args_to_dict` (*args*)

Convert autoserv extra arguments in the form of key=val or key:val to a dictionary. Each argument key is converted to lowercase dictionary key.

Args: *args* - list of autoserv extra arguments.

Returns: dictionary

`autotest.client.shared.utils.ask` (*question*, *auto=False*)

Raw input with a prompt that emulates logging.

Parameters

- **question** – Question to be asked
- **auto** – Whether to return “y” instead of asking the question

`autotest.client.shared.utils.aton` (*sr*)

Transform a string to a number(include float and int). If the string is not in the form of number, just return false.

Parameters *sr* – string to transform

Returns float, int or False for failed transform

`autotest.client.shared.utils.bitlist_to_string` (*data*)

Transform from bit list to ASCII string.

Parameters *data* – Bit list to be transformed

`autotest.client.shared.utils.close_log_file` (*filename*)

`autotest.client.shared.utils.compare_versions` (*ver1*, *ver2*)

Version number comparison between ver1 and ver2 strings.

```
>>> compare_tuple("1", "2")
-1
>>> compare_tuple("foo-1.1", "foo-1.2")
-1
>>> compare_tuple("1.2", "1.2a")
-1
>>> compare_tuple("1.2b", "1.2a")
1
>>> compare_tuple("1.3.5.3a", "1.3.5.3b")
-1
```

Args: *ver1*: version string *ver2*: version string

Returns:

int: 1 if ver1 > ver2

0 if ver1 == ver2

-1 if ver1 < ver2

`autotest.client.shared.utils.configure` (*extra=None*, *configure='./configure'*)

Run configure passing in the correct host, build, and target options.

Parameters

- **extra** – extra command line arguments to pass to configure
- **configure** – which configure script to use

`autotest.client.shared.utils.convert_data_size` (*size*, *default_suffix='B'*)

Convert data size from human readable units to an int of arbitrary size.

Parameters

- **size** – Human readable data size representation (string).
- **default_suffix** – Default suffix used to represent data.

Returns Int with data size in the appropriate order of magnitude.

`autotest.client.shared.utils.convert_ipv4_to_ipv6` (*ipv4*)

Translates a passed in string of an ipv4 address to an ipv6 address.

Parameters **ipv4** – a string of an ipv4 address

`autotest.client.shared.utils.cpu_affinity_by_task` (*pid*, *vcpu_pid*)

This function returns the allowed cpus from the proc entry for each vcpu's through its task id for a pid(of a VM)

`autotest.client.shared.utils.create_subnet_mask` (*bits*)

`autotest.client.shared.utils.create_x509_dir` (*path*, *cacert_subj*, *server_subj*, *passphrase*,
secure=False, *bits=1024*, *days=1095*)

Creates directory with freshly generated: ca-cart.pem, ca-key.pem, server-cert.pem, server-key.pem,

Parameters

- **path** – defines path to directory which will be created
- **cacert_subj** – ca-cert.pem subject

:param server_key.csr subject :param passphrase - passphrase to ca-key.pem :param secure = False - defines if the server-key.pem will use a passphrase :param bits = 1024: bit length of keys :param days = 1095: cert expiration

Raises

- **ValueError** – openssl not found or rc != 0
- **OSError** – if os.makedirs() fails

`autotest.client.shared.utils.delete_pid_file_if_exists` (*program_name*,
pid_files_dir=None)

Tries to remove <program_name>.pid from the main autotest directory.

`autotest.client.shared.utils.deprecated` (*func*)

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

`autotest.client.shared.utils.display_data_size` (*size*)

Display data size in human readable units.

Parameters **size** (*int*) – Data size, in Bytes.

Returns Human readable string with data size.

`autotest.client.shared.utils.etraceback` (*prep*, *exc_info*)

Enhanced Traceback formats traceback into lines “prep: line

name: line”

param prep desired line preposition

param exc_info sys.exc_info of the exception

return string which contains beautifully formatted exception

`autotest.client.shared.utils.find_command(cmd)`

Try to find a command in the PATH, paranoid version.

Parameters `cmd` – Command to be found.

Raise `ValueError` in case the command was not found.

`autotest.client.shared.utils.find_free_port(start_port, end_port, address='localhost')`

Return a host free port in the range [start_port, end_port].

Parameters

- **start_port** – First port that will be checked.
- **end_port** – Port immediately after the last one that will be checked.

`autotest.client.shared.utils.find_free_ports(start_port, end_port, count, address='localhost')`

Return count of host free ports in the range [start_port, end_port].

@count: Initial number of ports known to be free in the range. :param start_port: First port that will be checked. :param end_port: Port immediately after the last one that will be checked.

`autotest.client.shared.utils.find_substring(string, pattern1, pattern2=None)`

Return the match of pattern1 in string. Or return the match of pattern2 if pattern1 is not matched.

@string: string @pattern1: first pattern want to match in string, must set. @pattern2: second pattern, it will be used if pattern1 not match, optional.

Return: Match substring or None

`autotest.client.shared.utils.format_ip_with_mask(ip, mask_bits)`

`autotest.client.shared.utils.format_str_for_message(msg_str)`

Format msg_str so that it can be appended to a message. If msg_str consists of one line, prefix it with a space. If msg_str consists of multiple lines, prefix it with a newline.

Parameters `msg_str` – string that will be formatted.

`autotest.client.shared.utils.generate_random_id()`

Return a random string suitable for use as a qemu id.

`autotest.client.shared.utils.generate_random_string(length, ignore_str='!\"#$%&\'()*+,-./;<=>?@[\\]^_`{|}~', convert_str='')`

Return a random string using alphanumeric characters.

Parameters

- **length** – Length of the string that will be generated.
- **ignore_str** – Characters that will not include in generated string.
- **convert_str** – Characters that need to be escaped (prepend “”).

Returns The generated random string.

`autotest.client.shared.utils.generate_tmp_file_name(file_name, ext=None, directory='/tmp/')`

Returns a temporary file name. The file is not created.

`autotest.client.shared.utils.get_arch(run_function=<function run>)`

Get the hardware architecture of the machine. run_function is used to execute the commands. It defaults to `utils.run()` but a custom method (if provided) should be of the same schema as `utils.run`. It should return a `CmdResult` object and throw a `CmdError` exception.

`autotest.client.shared.utils.get_archive_tarball_name` (*source_dir*, *tarball_name*, *compression*)

Get the name for a tarball file, based on source, name and compression

`autotest.client.shared.utils.get_children_pids` (*ppid*)

Get all PIDs of children/threads of parent ppid param ppid: parent PID return: list of PIDs of all children/threads of ppid

`autotest.client.shared.utils.get_cpu_percentage` (*function*, **args*, ***dargs*)

Returns a tuple containing the CPU% and return value from function call.

This function calculates the usage time by taking the difference of the user and system times both before and after the function call.

`autotest.client.shared.utils.get_field` (*data*, *param*, *linestart*=' ', *sep*=' ')

Parse data from string. :param data: Data to parse.

example:

data: cpu 324 345 34 5 345 cpu0 34 11 34 34 33 ^^^^ start of line params 0 1 2 3 4

Parameters

- **param** – Position of parameter after linestart marker.
- **linestart** – String to which start line with parameters.
- **sep** – Separator between parameters regular expression.

`autotest.client.shared.utils.get_file` (*src*, *dest*, *permissions*=None)

Get a file from src, which can be local or a remote URL

`autotest.client.shared.utils.get_full_pci_id` (*pci_id*)

Get full PCI ID of pci_id.

Parameters **pci_id** – PCI ID of a device.

`autotest.client.shared.utils.get_hash_from_file` (*hash_path*, *dvd_basename*)

Get the a hash from a given DVD image from a hash file (Hash files are usually named MD5SUM or SHA1SUM and are located inside the download directories of the DVDs)

Parameters

- **hash_path** – Local path to a hash file.
- **cd_image** – Basename of a CD image

`autotest.client.shared.utils.get_ip_local_port_range` ()

`autotest.client.shared.utils.get_num_logical_cpus_per_socket` (*run_function*=<*function* *run*>)

Get the number of cores (including hyperthreading) per cpu. run_function is used to execute the commands. It defaults to utils.run() but a custom method (if provided) should be of the same schema as utils.run. It should return a CmdResult object and throw a CmdError exception.

`autotest.client.shared.utils.get_path` (*base_path*, *user_path*)

Translate a user specified path to a real path. If user_path is relative, append it to base_path. If user_path is absolute, return it as is.

Parameters

- **base_path** – The base path of relative user specified paths.
- **user_path** – The user specified path.

`autotest.client.shared.utils.get_pid_cpu(pid)`

Get the process used cpus.

Parameters `pid` – process id

Returns A list include all cpus the process used

Return type *list*

`autotest.client.shared.utils.get_pid_from_file(program_name, pid_files_dir=None)`

Reads the pid from `<program_name>.pid` in the autotest directory.

:param `program_name` the name of the program :return: the pid if the file exists, None otherwise.

`autotest.client.shared.utils.get_pid_path(program_name, pid_files_dir=None)`

`autotest.client.shared.utils.get_process_name(pid)`

Get process name from PID. :param `pid`: PID of process.

`autotest.client.shared.utils.get_relative_path(path, reference)`

Given 2 absolute paths “path” and “reference”, compute the path of “path” as relative to the directory “reference”.

:param `path` the absolute path to convert to a relative path :param `reference` an absolute directory path to which the relative

`path` will be computed

`autotest.client.shared.utils.get_stderr_level(stderr_is_expected)`

`autotest.client.shared.utils.get_stream_tee_file(stream, level, prefix='')`

`autotest.client.shared.utils.get_thread_cpu(thread)`

Get the light weight process(thread) used cpus.

Parameters `thread` (*string*) – thread checked

Returns A list include all cpus the thread used

Return type *list*

`autotest.client.shared.utils.get_unique_name(check, prefix='', suffix='', length=None, skip=None)`

Get unique name according to check function, use only 1000 iterations. :param `cmp`: Function called to discover name uniqueness :param `prefix`: Name prefix :param `suffix`: Name suffix :param `length`: Length of random string, when None use numbers (0,1,2) :param `skip`: skip n numbers (only when `length=None`)

Raises `StopIteration` – In case no unique name obtained in 1000 iterations

Returns Unique name according to check function

`autotest.client.shared.utils.get_unused_port()`

Finds a semi-random available port. A race condition is still possible after the port number is returned, if another process happens to bind it.

Returns: A port number that is unused on both TCP and UDP.

`autotest.client.shared.utils.get_vendor_from_pci_id(pci_id)`

Check out the device vendor ID according to `pci_id`.

Parameters `pci_id` – PCI ID of a device.

`autotest.client.shared.utils.hash(type, input=None)`

Returns an hash object of type md5 or sha1. This function is implemented in order to encapsulate hash objects in a way that is compatible with python 2.4 and python 2.6 without warnings.

Note that even though python 2.6 hashlib supports hash types other than md5 and sha1, we are artificially limiting the input values in order to make the function to behave exactly the same among both python implementations.

Parameters `input` – Optional input string that will be used to update the hash.

`autotest.client.shared.utils.import_site_class` (*path, module, classname, baseclass, modulefile=None*)

Try to import site specific class from site specific file if it exists

Args: `path`: full filename of the source file calling this (ie `__file__`) `module`: full module name `classname`: class name to be loaded from site file `baseclass`: base class object to return when no site file present or

to mixin when site class exists but is not inherited from baseclass

`modulefile`: module filename

Returns: `baseclass` if site specific class does not exist, the site specific class if it exists and is inherited from baseclass or a mixin of the site specific class and baseclass when the site specific class exists and is not inherited from baseclass

Raises: `ImportError` if the site file exists but imports fails

`autotest.client.shared.utils.import_site_function` (*path, module, funcname, dummy, modulefile=None*)

Try to import site specific function from site specific file if it exists

Args: `path`: full filename of the source file calling this (ie `__file__`) `module`: full module name `funcname`: function name to be imported from site file `dummy`: dummy function to return in case there is no function to import `modulefile`: module filename

Returns: site specific function object or dummy

Raises: `ImportError` if the site file exists but imports fails

`autotest.client.shared.utils.import_site_module` (*path, module, dummy=None, modulefile=None*)

Try to import the site specific module if it exists.

:param `path` full filename of the source file calling this (ie `__file__`) :param `module` full module name :param `dummy` dummy value to return in case there is no symbol to import :param `modulefile` module filename

Returns site specific module or dummy

:raise `ImportError` if the site file exists but imports fails

`autotest.client.shared.utils.import_site_symbol` (*path, module, name, dummy=None, modulefile=None*)

Try to import site specific symbol from site specific file if it exists

:param `path` full filename of the source file calling this (ie `__file__`) :param `module` full module name :param `name` symbol name to be imported from the site file :param `dummy` dummy value to return in case there is no symbol to import :param `modulefile` module filename

Returns site specific symbol or dummy

:raise `ImportError` if the site file exists but imports fails

`autotest.client.shared.utils.interactive_download` (*url, output_file, title='', chunk_size=102400*)

Interactively downloads a given file url to a given output file

Parameters

- `url` (*string*) – URL for the file to be download

- **output_file** (*string*) – file name or absolute path on which to save the file to
- **title** (*string*) – optional title to go along the progress bar
- **chunk_size** (*integer*) – amount of data to read at a time

`autotest.client.shared.utils.ip_to_long(ip)`

`autotest.client.shared.utils.is_mounted(src, mount_point, fstype, perm=None, verbose=True, fstype_mtab=None)`

Check mount status from /etc/mstab

Parameters

- **src** (*string*) – mount source
- **mount_point** (*string*) – mount point
- **fstype** (*string*) – file system type
- **perm** (*string*) – mount permission
- **fstype_mtab** (*str*) – file system type in mtab could be different

Returns if the src is mounted as expect

Return type Boolean

`autotest.client.shared.utils.is_port_free(port, address)`

Return True if the given port is available for use.

Parameters **port** – Port number

`autotest.client.shared.utils.is_url(path)`

Return true if path looks like a URL

`autotest.client.shared.utils.join_bg_jobs(bg_jobs, timeout=None)`

Joins the bg_jobs with the current thread.

Returns the same list of bg_jobs objects that was passed in.

`autotest.client.shared.utils.kill_process_tree(pid, sig=9)`

Signal a process and all of its children.

If the process does not exist – return.

Parameters

- **pid** – The pid of the process to signal.
- **sig** – The signal to send to the processes.

`autotest.client.shared.utils.lock_file(filename, mode=2)`

`autotest.client.shared.utils.log_last_traceback(msg=None, log=<function error>)`

Writes last traceback into specified log. :param msg: Override the default message. ["Original traceback"]
:param log: Where to log the traceback [logging.error]

`autotest.client.shared.utils.log_line(filename, line)`

Write a line to a file. ‘

‘ is appended to the line.

param filename Path of file to write to, either absolute or relative to the dir set by `set_log_file_dir()`.

param line Line to write.

`autotest.client.shared.utils.long_to_ip(number)`

`autotest.client.shared.utils.make(extra='', make='make', timeout=None, ignore_status=False)`

Run make, adding MAKEOPTS to the list of options.

Parameters `extra` – extra command line arguments to pass to make.

`autotest.client.shared.utils.matrix_to_string(matrix, header=None)`

Return a pretty, aligned string representation of a nxm matrix.

This representation can be used to print any tabular data, such as database results. It works by scanning the lengths of each element in each column, and determining the format string dynamically.

Parameters

- `matrix` – Matrix representation (list with n rows of m elements).
- `header` – Optional tuple or list with header elements to be displayed.

`autotest.client.shared.utils.merge_trees(src, dest)`

Merges a source directory tree at 'src' into a destination tree at 'dest'. If a path is a file in both trees than the file in the source tree is APPENDED to the one in the destination tree. If a path is a directory in both trees then the directories are recursively merged with this function. In any other case, the function will skip the paths that cannot be merged (instead of failing).

`autotest.client.shared.utils.mount(src, mount_point, fstype, perm=None, verbose=True, fstype_mtab=None)`

Mount the src into mount_point of the host.

Src mount source

Mount_point mount point

Fstype file system type

Perm mount permission

Parameters `fstype_mtab` (*str*) – file system type in mtab could be different

`autotest.client.shared.utils.normalize_hostname(alias)`

`autotest.client.shared.utils.nuke_pid(pid, signal_queue=(15, 9))`

`autotest.client.shared.utils.nuke_subprocess(subproc)`

`autotest.client.shared.utils.open_write_close(filename, data)`

`autotest.client.shared.utils.parallel(targets)`

Run multiple functions in parallel.

Parameters `targets` – A sequence of tuples or functions. If it's a sequence of tuples, each tuple will be interpreted as (target, args, kwargs) or (target, args) or (target,) depending on its length. If it's a sequence of functions, the functions will be called without arguments.

Returns A list of the values returned by the functions called.

`autotest.client.shared.utils.pid_exists(pid)`

Return True if a given PID exists.

Parameters `pid` – Process ID number.

`autotest.client.shared.utils.pid_is_alive(pid)`

True if process pid exists and is not yet stuck in Zombie state. Zombies are impossible to move between cgroups, etc. pid can be integer, or text of integer.

`autotest.client.shared.utils.process_or_children_is_defunct(ppid)`

Verify if any processes from PPID is defunct.

Attempt to verify if parent process and any children from PPID is defunct (zombie) or not. :param ppid: The parent PID of the process to verify.

`autotest.client.shared.utils.program_is_alive(program_name, pid_files_dir=None)`

Checks if the process is alive and not in Zombie state.

:param program_name the name of the program :return: True if still alive, False otherwise

`autotest.client.shared.utils.read_file(filename)`

`autotest.client.shared.utils.read_keyval(path)`

Read a key-value pair format file into a dictionary, and return it. Takes either a filename or directory name as input. If it's a directory name, we assume you want the file to be called keyval.

`autotest.client.shared.utils.read_one_line(filename)`

`autotest.client.shared.utils.run(command, timeout=None, ignore_status=False, stdout_tee=None, stderr_tee=None, verbose=True, stdin=None, stderr_is_expected=None, args=())`

Run a command on the host.

Parameters

- **command** – the command line string.
- **timeout** – time limit in seconds before attempting to kill the running process. The `run()` function will take a few seconds longer than ‘timeout’ to complete if it has to kill the process.
- **ignore_status** – do not raise an exception, no matter what the exit code of the command is.
- **stdout_tee** – optional file-like object to which stdout data will be written as it is generated (data will still be stored in `result.stdout`).
- **stderr_tee** – likewise for stderr.
- **verbose** – if True, log the command being run.
- **stdin** – stdin to pass to the executed process (can be a file descriptor, a file object of a real file or a string).
- **args** – sequence of strings of arguments to be given to the command inside ” quotes after they have been escaped for that; each element in the sequence will be given as a separate command argument

Returns a `CmdResult` object

Raises `CmdError` – the exit code of the command execution was not 0

`autotest.client.shared.utils.run_bg(*args, **dargs)`

Function deprecated. Please use `BgJob` class instead.

`autotest.client.shared.utils.run_parallel(commands, timeout=None, ignore_status=False, stdout_tee=None, stderr_tee=None)`

Behaves the same as `run()` with the following exceptions:

- `commands` is a list of commands to run in parallel.
- `ignore_status` toggles whether or not an exception should be raised on any error.

Returns a list of CmdResult objects

`class autotest.client.shared.utils.run_randomly(run_sequentially=False)`

`add(*args, **dargs)`

`run(fn)`

`autotest.client.shared.utils.safe_kill(pid, signal)`

Attempt to send a signal to a given process that may or may not exist.

Parameters `signal` – Signal number.

`autotest.client.shared.utils.safe_rmdir(path, timeout=10)`

Try to remove a directory safely, even on NFS filesystems.

Sometimes, when running an autotest client test on an NFS filesystem, when not all filedescriptors are closed, NFS will create some temporary files, that will make `shutil.rmtree` to fail with error 39 (directory not empty). So let's keep trying for a reasonable amount of time before giving up.

Parameters

- `path` (*string*) – Path to a directory to be removed.
- `timeout` (*int*) – Time that the function will try to remove the dir before giving up (seconds)

Raises `OSError`, with `errno 39` in case after the timeout `shutil.rmtree` could not successfully complete. If any attempt to `rmtree` fails with `errno` different than 39, that exception will be just raised.

`autotest.client.shared.utils.selinux_enforcing()`

Returns True if SELinux is in enforcing mode, False if permissive/disabled

`autotest.client.shared.utils.set_ip_local_port_range(lower, upper)`

`autotest.client.shared.utils.set_log_file_dir(directory)`

Set the base directory for log files created by `log_line()`.

Parameters `dir` – Directory for log files.

`autotest.client.shared.utils.sh_escape(command)`

Escape special characters from a command so that it can be passed as a double quoted (" ") string in a (ba)sh command.

Args: `command`: the command string to escape.

Returns: The escaped command string. The required enclosing double quotes are NOT added and so should be added at some point by the caller.

See also: <http://www.tldp.org/LDP/abs/html/escapingsection.html>

`autotest.client.shared.utils.signal_pid(pid, sig)`

Sends a signal to a process id. Returns True if the process terminated successfully, False otherwise.

`autotest.client.shared.utils.signal_program(program_name, sig=15, pid_files_dir=None)`

Sends a signal to the process listed in `<program_name>.pid`

:param `program_name` the name of the program :param `sig` signal to send

`autotest.client.shared.utils.string_to_bitlist(data)`

Transform from ASCII string to bit list.

Parameters `data` – String to be transformed

`autotest.client.shared.utils.strip_console_codes` (*output*)

Remove the Linux console escape and control sequences from the console output. Make the output readable and can be used for result check. Now only remove some basic console codes using during boot up.

Parameters `output` (*string*) – The output from Linux console

Returns the string without any special codes

Return type *string*

`autotest.client.shared.utils.strip_unicode` (*input*)

`autotest.client.shared.utils.system` (*command*, *timeout=None*, *ignore_status=False*, *verbose=True*)

Run a command

Parameters

- **timeout** – timeout in seconds
- **ignore_status** – if `ignore_status=False`, throw an exception if the command's exit code is non-zero if `ignore_status=True`, return the exit code.
- **verbose** – if `True`, log the command being run.

Returns exit status of command (note, this will always be zero unless `ignore_status=True`)

`autotest.client.shared.utils.system_output` (*command*, *timeout=None*, *ignore_status=False*, *retain_output=False*, *args=()*, *verbose=True*)

Run a command and return the stdout output.

Parameters

- **command** – command string to execute.
- **timeout** – time limit in seconds before attempting to kill the running process. The function will take a few seconds longer than 'timeout' to complete if it has to kill the process.
- **ignore_status** – do not raise an exception, no matter what the exit code of the command is.
- **retain_output** – set to `True` to make stdout/stderr of the command output to be also sent to the logging system
- **args** – sequence of strings of arguments to be given to the command inside " quotes after they have been escaped for that; each element in the sequence will be given as a separate command argument
- **verbose** – if `True`, log the command being run.

Returns a string with the stdout output of the command.

`autotest.client.shared.utils.system_output_parallel` (*commands*, *timeout=None*, *ignore_status=False*, *retain_output=False*)

`autotest.client.shared.utils.system_parallel` (*commands*, *timeout=None*, *ignore_status=False*)

This function returns a list of exit statuses for the respective list of commands.

`autotest.client.shared.utils.umount` (*src*, *mount_point*, *fstype*, *verbose=True*, *fstype_mtab=None*)

Unmount the `src` mounted in `mount_point`.

Src mount source

Mount_point mount point

Type file system type

Parameters **fstype_mtab** (*str*) – file system type in mtab could be different

`autotest.client.shared.utils.unique` (*llist*)

Return a list of the elements in list, but without duplicates.

Parameters **list** – List with values.

Returns List with non duplicate elements.

`autotest.client.shared.utils.unlock_file` (*lockfile*)

`autotest.client.shared.utils.unmap_url` (*srcdir, src, destdir='.'*)

Receives either a path to a local file or a URL. returns either the path to the local file, or the fetched URL

unmap_url('/usr/src', 'foo.tar', '/tmp') = '/usr/src/foo.tar'

unmap_url('/usr/src', 'http://site/file', '/tmp') = '/tmp/file' (after retrieving it)

`autotest.client.shared.utils.update_version` (*srcdir, preserve_srcdir, new_version, install, *args, **dargs*)

Make sure srcdir is version new_version

If not, delete it and install() the new version.

In the preserve_srcdir case, we just check it's up to date, and if not, we rerun install, without removing srcdir

`autotest.client.shared.utils.urlopen` (*url, data=None, timeout=5*)

Wrapper to urllib2.urlopen with timeout addition.

`autotest.client.shared.utils.urlretrieve` (*url, filename, data=None, timeout=300*)

Retrieve a file from given url.

`autotest.client.shared.utils.verify_running_as_root` ()

Verifies whether we're running under UID 0 (root).

Raise error.TestNAError

`autotest.client.shared.utils.wait_for` (*func, timeout, first=0.0, step=1.0, text=None*)

If func() evaluates to True before timeout expires, return the value of func(). Otherwise return None.

@brief: Wait until func() evaluates to True.

Parameters

- **timeout** – Timeout in seconds
- **first** – Time to sleep before first attempt
- **steps** – Time to sleep between attempts in seconds
- **text** – Text to print while waiting, for debug purposes

`autotest.client.shared.utils.write_keyval` (*path, dictionary, type_tag=None, tap_report=None*)

Write a key-value pair format file out to a file. This uses append mode to open the file, so existing text will not be overwritten or reparsed.

If type_tag is None, then the key must be composed of alphanumeric characters (or dashes+underscores). However, if type-tag is not null then the keys must also have "{type_tag}" as a suffix. At the moment the only valid values of type_tag are "attr" and "perf".

Parameters

- **path** – full path of the file to be written
- **dictionary** – the items to write
- **type_tag** – see text above

`autotest.client.shared.utils.write_one_line(filename, line)`

`autotest.client.shared.utils.write_pid(program_name, pid_files_dir=None)`

Try to drop <program_name>.pid in the main autotest directory.

Args: program_name: prefix for file name

utils_cgroup Module

Helpers for cgroup testing.

copyright 2011 Red Hat Inc.

author Lukas Doktor <ldoktor@redhat.com>

class `autotest.client.shared.utils_cgroup.Cgroup(module, _client)`

Bases: `object`

Cgroup handling class.

cgclassify_cgroup (*pid, cgroup*)

Classify pid into cgroup

Parameters

- **pid** – pid of the process
- **cgroup** – cgroup name

cgdelete_all_cgroups ()

Delete all cgroups in the module

cgdelete_cgroup (*cgroup, recursive=False*)

Delete desired cgroup.

Params **cgroup** desired cgroup

:params force: If true, sub cgroup can be deleted with parent cgroup

cgexec (*cgroup, cmd, args=''*)

Execute command in desired cgroup

Param **cgroup**: Desired cgroup

Param **cmd**: Executed command

Param **args**: Executed command's parameters

cgset_property (*prop, value, pwd=None, check=True, checkprop=None*)

Sets the property value by cgset command

Param **prop**: property name (file)

Param **value**: desired value

Parameters

- **pwd** – cgroup directory
- **check** – check the value after setup / override checking value

- **checkprop** – override prop when checking the value

get_cgroup_index (*cgroup*)

Get cgroup's index in cgroups

Param cgroup: cgroup name

Returns index of cgroup

get_cgroup_name (*pwd=None*)

Get cgroup's name

Param pwd: cgroup name

Returns cgroup's name

get_pids (*pwd=None*)

Get all pids in cgroup

Params pwd: cgroup directory

Returns all pids(list)

get_property (*prop, pwd=None*)

Gets the property value :param prop: property name (file) :param pwd: cgroup directory :return: [] values or None when FAILED

initialize (*modules*)

Initializes object for use.

Parameters modules – Array of all available cgroup modules.

is_cgroup (*pid, pwd*)

Checks if the 'pid' process is in 'pwd' cgroup :param pid: pid of the process :param pwd: cgroup directory :return: 0 when is 'pwd' member

is_root_cgroup (*pid*)

Checks if the 'pid' process is in root cgroup (WO cgroup) :param pid: pid of the process :return: 0 when is 'root' member

mk_cgroup (*pwd=None, cgroup=None*)

Creates new temporary cgroup :param pwd: where to create this group (default: self.root) :param cgroup: desired cgroup name :return: last cgroup index

mk_cgroup_cgcreate (*pwd=None, cgroup=None*)

Make a cgroup by cgcreate command

Params cgroup: Maked cgroup name

Returns last cgroup index

refresh_cgroups ()

Refresh all cgroups path.

rm_cgroup (*pwd*)

Removes cgroup.

Parameters pwd – cgroup directory.

set_cgroup (*pid, pwd=None*)

Sets cgroup membership :param pid: pid of the process :param pwd: cgroup directory

set_property (*prop, value, pwd=None, check=True, checkprop=None*)

Sets the property value :param prop: property name (file) :param value: desired value :param pwd: cgroup

directory :param check: check the value after setup / override checking value :param checkprop: override prop when checking the value

set_property_h (*prop, value, pwd=None, check=True, checkprop=None*)

Sets the one-line property value concerning the K,M,G postfix :param prop: property name (file) :param value: desired value :param pwd: cgroup directory :param check: check the value after setup / override checking value :param checkprop: override prop when checking the value

set_root_cgroup (*pid*)

Resets the cgroup membership (sets to root) :param pid: pid of the process :return: 0 when PASSED

smoke_test ()

Smoke test Module independent basic tests

test (*cmd*)

Executes cgroup_client.py with cmd parameter.

Parameters **cmd** – command to be executed

Returns subprocess.Popen() process

class autotest.client.shared.utils_cgroup.**CgroupModules** (*mountdir=None*)

Bases: `object`

Handles the list of different cgroup filesystems.

get_pwd (*module*)

Returns the mount directory of 'module' :param module: desired module (memory, ...) :return: mount directory of 'module' or None

init (*_modules*)

Checks the mounted modules and if necessary mounts them into tmp mountdir.

Parameters **_modules** – Desired modules.'memory', 'cpu,cpuset'...

Returns Number of initialized modules.

autotest.client.shared.utils_cgroup.**all_cgroup_delete** ()

Clear all cgroups in system

autotest.client.shared.utils_cgroup.**cgconfig_condrestart** ()

Condrestart cgconfig service

autotest.client.shared.utils_cgroup.**cgconfig_exists** ()

Check if cgconfig is available on the host or perhaps systemd is used

autotest.client.shared.utils_cgroup.**cgconfig_is_running** ()

Check cgconfig service status

autotest.client.shared.utils_cgroup.**cgconfig_restart** ()

Restart cgconfig service

autotest.client.shared.utils_cgroup.**cgconfig_start** ()

Stop cgconfig service

autotest.client.shared.utils_cgroup.**cgconfig_stop** ()

Start cgconfig service

autotest.client.shared.utils_cgroup.**get_all_controllers** ()

Get all controllers used in system

Returns all used controllers(controller_list)

`autotest.client.shared.utils_cgroup.get_cgroup_mountpoint(controller)`

Get desired controller's mountpoint

@controller: Desired controller :return: controller's mountpoint

`autotest.client.shared.utils_cgroup.get_load_per_cpu(_stats=None)`

Gather load per cpu from /proc/stat :param _stats: previous values :return: list of diff/absolute values of CPU times [SUM, CPU1, CPU2, ...]

`autotest.client.shared.utils_cgroup.resolve_task_cgroup_path(pid, controller)`

Resolving cgroup mount path of a particular task

Params pid : process id of a task for which the cgroup path required

Params controller: takes one of the controller names in controller list

Returns resolved path for cgroup controllers of a given pid

`autotest.client.shared.utils_cgroup.service_cgconfig_control(action)`

Cgconfig control by action.

If cmd executes successfully, return True, otherwise return False. If the action is status, return True when it's running, otherwise return False. To check if the cgconfig stuff is available, use action "exists".

@ param action: start|stop|status|restart|condrestart

utils_koji Module

class `autotest.client.shared.utils_koji.KojiClient` (*cmd=None*)

Bases: `object`

Establishes a connection with the build system, either koji or brew.

This class provides convenience methods to retrieve information on packages and the packages themselves hosted on the build system. Packages should be specified in the KojiPkgSpec syntax.

CMD_LOOKUP_ORDER = ['/usr/bin/brew', '/usr/bin/koji']

CONFIG_MAP = {'/usr/bin/brew': '/etc/brewkoji.conf', '/usr/bin/koji': '/etc/koji.conf'}

get_default_command()

Looks up for koji or brew "binaries" on the system

Systems with plain koji usually don't have a brew cmd, while systems with koji, have *both* koji and brew utilities. So we look for brew first, and if found, we consider that the system is configured for brew. If not, we consider this is a system with plain koji.

Returns either koji or brew command line executable path, or None

get_pkg_base_url()

Gets the base url for packages in Koji

get_pkg_info(pkg)

Returns information from Koji on the package

Parameters pkg (`KojiPkgSpec`) – information about the package, as a `KojiPkgSpec` instance

Returns information from Koji about the specified package

get_pkg_rpm_file_names(pkg, arch=None)

Gets the file names for the RPM packages specified in pkg

Parameters

- **pkg** (`KojiPkgSpec`) – a package specification
- **arch** (`string`) – packages built for this architecture, but also including architecture independent (noarch) packages

get_pkg_rpm_info (`pkg, arch=None`)

Returns a list of information on the RPM packages found on koji

Parameters

- **pkg** (`KojiPkgSpec`) – a package specification
- **arch** (`string`) – packages built for this architecture, but also including architecture independent (noarch) packages

get_pkg_rpm_names (`pkg, arch=None`)

Gets the names for the RPM packages specified in pkg

Parameters

- **pkg** (`KojiPkgSpec`) – a package specification
- **arch** (`string`) – packages built for this architecture, but also including architecture independent (noarch) packages

get_pkg_urls (`pkg, arch=None`)

Gets the urls for the packages specified in pkg

Parameters

- **pkg** (`KojiPkgSpec`) – a package specification
- **arch** (`string`) – packages built for this architecture, but also including architecture independent (noarch) packages

get_pkgs (`pkg, dst_dir, arch=None`)

Download the packages

Parameters

- **pkg** (`KojiPkgSpec`) – a package specification
- **dst_dir** (`string`) – the destination directory, where the downloaded packages will be saved on
- **arch** (`string`) – packages built for this architecture, but also including architecture independent (noarch) packages

get_scratch_base_url ()

Gets the base url for scratch builds in Koji

get_scratch_pkg_urls (`pkg, arch=None`)

Gets the urls for the scratch packages specified in pkg

Parameters

- **pkg** (`KojiScratchPkgSpec`) – a scratch package specification
- **arch** (`string`) – packages built for this architecture, but also including architecture independent (noarch) packages

get_scratch_pkgs (`pkg, dst_dir, arch=None`)

Download the packages from a scratch build

Parameters

- **pkg** (`KojiScratchPkgSpec`) – a scratch package specification

- **dst_dir** (*string*) – the destination directory, where the downloaded packages will be saved on
- **arch** (*string*) – packages built for this architecture, but also including architecture independent (noarch) packages

get_session_options ()

Filter only options necessary for setting up a cobbler client session

Returns only the options used for session setup

is_command_valid ()

Checks if the currently set koji command is valid

Returns True or False

is_config_valid ()

Checks if the currently set koji configuration is valid

Returns True or False

is_pkg_spec_build_valid (*pkg*)

Checks if build is valid on Koji

Parameters **pkg** – a Pkg instance

is_pkg_spec_tag_valid (*pkg*)

Checks if tag is valid on Koji

Parameters **pkg** (*KojiPkgSpec*) – a package specification

is_pkg_valid (*pkg*)

Checks if this package is altogether valid on Koji

This verifies if the build or tag specified in the package specification actually exist on the Koji server

Returns True or False

read_config (*check_is_valid=True*)

Reads options from the Koji configuration file

By default it checks if the koji configuration is valid

Parameters **check_valid** (*boolean*) – whether to include a check on the configuration

Raise ValueError

Returns None

class autotest.client.shared.utils_koji.**KojiDirIndexParser**

Bases: *HTMLParser.HTMLParser*

Parser for HTML directory index pages, specialized to look for RPM links

handle_starttag (*tag, attrs*)

Handle tags during the parsing

This just looks for links ('a' tags) for files ending in .rpm

class autotest.client.shared.utils_koji.**KojiPkgSpec** (*text='', tag=None, build=None, package=None, subpackages=[]*)

Bases: *object*

A package specification syntax parser for Koji

This holds information on either tag or build, and packages to be fetched from koji and possibly installed (features external do this class).

New objects can be created either by providing information in the textual format or by using the actual parameters for tag, build, package and sub- packages. The textual format is useful for command line interfaces and configuration files, while using parameters is better for using this in a programatic fashion.

The following sets of examples are interchangeable. Specifying all packages part of build number 1000:

```
>>> from kvm_utils import KojiPkgSpec
>>> pkg = KojiPkgSpec('1000')
```

```
>>> pkg = KojiPkgSpec(build=1000)
```

Specifying only a subset of packages of build number 1000:

```
>>> pkg = KojiPkgSpec('1000:kernel, kernel-devel')
```

```
>>> pkg = KojiPkgSpec(build=1000,
                      subpackages=['kernel', 'kernel-devel'])
```

Specifying the latest build for the 'kernel' package tagged with 'dist-f14':

```
>>> pkg = KojiPkgSpec('dist-f14:kernel')
```

```
>>> pkg = KojiPkgSpec(tag='dist-f14', package='kernel')
```

Specifying the 'kernel' package using the default tag:

```
>>> kvm_utils.set_default_koji_tag('dist-f14')
>>> pkg = KojiPkgSpec('kernel')
```

```
>>> pkg = KojiPkgSpec(package='kernel')
```

Specifying the 'kernel' package using the default tag:

```
>>> kvm_utils.set_default_koji_tag('dist-f14')
>>> pkg = KojiPkgSpec('kernel')
```

```
>>> pkg = KojiPkgSpec(package='kernel')
```

If you do not specify a default tag, and give a package name without an explicit tag, your package specification is considered invalid:

```
>>> print kvm_utils.get_default_koji_tag()
None
>>> print kvm_utils.KojiPkgSpec('kernel').is_valid()
False
```

```
>>> print kvm_utils.KojiPkgSpec(package='kernel').is_valid()
False
```

SEP = ':'

describe()

Describe this package specification, in a human friendly way

Returns package specification description

describe_invalid()

Describes why this is not valid, in a human friendly way

is_valid()

Checks if this package specification is valid.

Being valid means that it has enough and not conflicting information. It does not validate that the packages specified actually exist on the Koji server.

Returns True or False

parse(text)

Parses a textual representation of a package specification

Parameters **text** (*string*) – textual representation of a package in koji

to_text()

Return the textual representation of this package spec

The output should be consumable by parse() and produce the same package specification.

We find that it's acceptable to put the currently set default tag as the package explicit tag in the textual definition for completeness.

Returns package specification in a textual representation

```
class autotest.client.shared.utils_koji.KojiScratchPkgSpec (text='', user=None,
                                                         task=None, subpack-
                                                         ages=[])
```

Bases: `object`

A package specification syntax parser for Koji scratch builds

This holds information on user, task and subpackages to be fetched from koji and possibly installed (features external do this class).

New objects can be created either by providing information in the textual format or by using the actual parameters for user, task and subpackages. The textual format is useful for command line interfaces and configuration files, while using parameters is better for using this in a programmatic fashion.

This package definition has a special behaviour: if no subpackages are specified, all packages of the chosen architecture (plus noarch packages) will match.

The following sets of examples are interchangeable. Specifying all packages from a scratch build (whose task id is 1000) sent by user jdoe:

```
>>> from kvm_utils import KojiScratchPkgSpec
>>> pkg = KojiScratchPkgSpec('jdoe:1000')
```

```
>>> pkg = KojiScratchPkgSpec(user=jdoe, task=1000)
```

Specifying some packages from a scratch build whose task id is 1000, sent by user jdoe:

```
>>> pkg = KojiScratchPkgSpec('jdoe:1000:kernel, kernel-devel')
```

```
>>> pkg = KojiScratchPkgSpec(user=jdoe, task=1000,
                             subpackages=['kernel', 'kernel-devel'])
```

SEP = ':'

parse(text)

Parses a textual representation of a package specification

Parameters `text` (*string*) – textual representation of a package in koji

class `autotest.client.shared.utils_koji.RPMFileNameInfo` (*filename*)

Simple parser for RPM based on information present on the filename itself

get_arch ()

Returns just the architecture as present on the RPM filename

get_filename_without_arch ()

Returns the filename without the architecture

This also excludes the RPM suffix, that is, removes the leading arch and RPM suffix.

get_filename_without_suffix ()

Returns the filename without the default RPM suffix

get_nvr_info ()

Returns a dictionary with the name, version and release components

If koji is not installed, this returns None

`autotest.client.shared.utils_koji.get_default_koji_tag` ()

`autotest.client.shared.utils_koji.set_default_koji_tag` (*tag*)

Sets the default tag that will be used

utils_memory Module

`autotest.client.shared.utils_memory.drop_caches` ()

Writes back all dirty pages to disk and clears all the caches.

`autotest.client.shared.utils_memory.freememtotal` ()

`autotest.client.shared.utils_memory.get_buddy_info` (*chunk_sizes*, *nodes='all'*,
zones='all')

Get the fragmentation status of the host. It use the same method to get the page size in buddyinfo. $2^{\text{chunk_size}} * \text{page_size}$ The `chunk_sizes` can be string make up by all orders that you want to check splited with blank or a mathematical expression with '>', '<' or '='. For example: The input of `chunk_size` could be: "0 2 4" And the return will be: {'0': 3, '2': 286, '4': 687} if you are using expression: ">=9" the return will be: {'9': 63, '10': 225}

Parameters

- **chunk_size** (*string*) – The order number shows in buddyinfo. This is not the real page size.
- **nodes** (*string*) – The numa node that you want to check. Default value is all
- **zones** (*string*) – The memory zone that you want to check. Default value is all

Returns A dict using the `chunk_size` as the keys

Return type dict

`autotest.client.shared.utils_memory.get_huge_page_size` ()

`autotest.client.shared.utils_memory.get_num_huge_pages` ()

`autotest.client.shared.utils_memory.memtotal` ()

`autotest.client.shared.utils_memory.node_size` ()

`autotest.client.shared.utils_memory.numa_nodes` ()

`autotest.client.shared.utils_memory.read_from_meminfo` (*key*)

`autotest.client.shared.utils_memory.read_from_numa_maps(pid, key)`

Get the process numa related info from numa_maps. This function only use to get the numbers like anon=1.

Parameters

- **pid** (*String*) – Process id
- **key** (*String*) – The item you want to check from numa_maps

Returns A dict using the address as the keys

Return type `dict`

`autotest.client.shared.utils_memory.read_from_smaps(pid, key)`

Get specific item value from the smaps of a process include all sections.

Parameters

- **pid** (*String*) – Process id
- **key** (*String*) – The item you want to check from smaps

Returns The value of the item in kb

Return type `int`

`autotest.client.shared.utils_memory.read_from_vmstat(key)`

Get specific item value from vmstat

Parameters **key** (*String*) – The item you want to check from vmstat

Returns The value of the item

Return type `int`

`autotest.client.shared.utils_memory.rounded_memtotal()`

`autotest.client.shared.utils_memory.set_num_huge_pages(num)`

version Module

Based on work from Douglas Creager <dcreager@dcreager.net>

Gets the current version number. If possible, this is the output of “git describe”, modified to conform to the versioning scheme that setup tools uses. If “git describe” returns an error (most likely because we’re in an unpacked copy of a release tarball, rather than in a git working copy), then we fall back on reading the contents of the RELEASE-VERSION file.

To use this script, simply import it your setup.py file, and use the results of get_version() as your package version:

```
from autotest.client.shared import version
```

```
setup( version=get_version(), . . .
```

```
)
```

This will automatically update the RELEASE-VERSION file, if necessary. Note that the RELEASE-VERSION file should *not* be checked into git; please add it to your top-level .gitignore file.

You’ll probably want to distribute the RELEASE-VERSION file in your sdist tarballs; to do this, just create a MANIFEST.in file that contains the following line:

```
include RELEASE-VERSION
```

```
autotest.client.shared.version.get_version(abbrev=4)
```

Subpackages

backports Package

backports Package

This module contains backported functions that are not present on Python 2.4 but are standard in more recent versions.

`autotest.client.shared.backports.all` (*iterable*)

From <http://stackoverflow.com/questions/3785433/python-backports-for-some-methods> :codeauthor: Tim Pietzcker <http://stackoverflow.com/users/20670/tim-pietzcker> licensed under cc-wiki with attribution required

`autotest.client.shared.backports.any` (*iterable*)

From <http://stackoverflow.com/questions/3785433/python-backports-for-some-methods> :codeauthor: Tim Pietzcker <http://stackoverflow.com/users/20670/tim-pietzcker> licensed under cc-wiki with attribution required

`autotest.client.shared.backports.bin` (*number*)

Adapted from <http://code.activestate.com/recipes/576847/> :codeauthor: Vishal Sapre :license: MIT

A foolishly simple look-up method of getting binary string from an integer This happens to be faster than all other ways!!!

`autotest.client.shared.backports.next` (**args*)

Retrieve the next item from the iterator by calling its next() method. If default is given, it is returned if the iterator is exhausted, otherwise StopIteration is raised. New in version 2.6.

Parameters

- **iterator** (*iterator*) – the iterator
- **default** (*object*) – the value to return if the iterator raises StopIteration

Returns The object returned by iterator.next()

Return type object

Subpackages

collections Package

collections Package

OrderedDict Module

Backport of OrderedDict() class that runs on Python 2.4, 2.5, 2.6, 2.7 and pypy. Passes Python2.7's test suite and incorporates all the latest updates.

Obtained from: <http://code.activestate.com/recipes/576693-ordered-dictionary-for-py24/>

class `autotest.client.shared.backports.collections.OrderedDict`. **OrderedDict** (**args*, ***kwds*)

Bases: `dict`

Dictionary that remembers insertion order

<http://code.activestate.com/recipes/576693-ordered-dictionary-for-py24/> :codeauthor: Raymond Hettinger :license: MIT

clear () → None. Remove all items from od.

copy () → a shallow copy of od

classmethod fromkeys (S[, v]) → New ordered dictionary with keys from S and values equal to v (which defaults to None).

items () → list of (key, value) pairs in od

iteritems ()
od.iteritems → an iterator over the (key, value) items in od

iterkeys () → an iterator over the keys in od

itervalues ()
od.itervalues → an iterator over the values in od

keys () → list of keys in od

pop (k[, d]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem () → (k, v), return and remove a (key, value) pair.
Pairs are returned in LIFO order if last is true or FIFO order if false.

setdefault (k[, d]) → od.get(k,d), also set od[k]=d if k not in od

update (E, **F) → None. Update od from dict/iterable E and F.
If E is a dict instance, does: for k in E: od[k] = E[k] If E has a .keys() method, does: for k in E.keys(): od[k] = E[k] Or if E is an iterable of items, does: for k, v in E: od[k] = v In either case, this is followed by: for k, v in F.items(): od[k] = v

values () → list of values in od

viewitems () → a set-like object providing a view on od's items

viewkeys () → a set-like object providing a view on od's keys

viewvalues () → an object providing a view on od's values

defaultdict Module

Backport of the defaultdict module, obtained from: <http://code.activestate.com/recipes/523034-emulate-collectionsdefaultdict/>

```
class autotest.client.shared.backports.collections.defaultdict.defaultdict (default_factory=None,
                                                                    *a,
                                                                    **kw)
```

Bases: dict

collections.defaultdict is a handy shortcut added in Python 2.5 which can be emulated in older versions of Python. This recipe tries to backport defaultdict exactly and aims to be safe to subclass and extend without worrying if the base class is in C or is being emulated.

<http://code.activestate.com/recipes/523034-emulate-collectionsdefaultdict/> :codeauthor: Jason Kirtland :license: PSF

Changes: * replaced self.items() with self.iteritems() to fix Pickle bug as recommended by Aaron Lav * reformatted with autopep8

copy ()

namedtuple Module

This module contains a backport for `collections.namedtuple` obtained from <http://code.activestate.com/recipes/500261-named-tuples/>

`autotest.client.shared.backports.collections.namedtuple.namedtuple` (*typename*,
field_names,
verbose=False,
rename=False)

Returns a new subclass of tuple with named fields.

```

>>> Point = namedtuple('Point', 'x y')
>>> Point.__doc__           # docstring for the new class
'Point(x, y)'
>>> p = Point(11, y=22)     # instantiate with positional args or keywords
>>> p[0] + p[1]            # indexable like a plain tuple
33
>>> x, y = p               # unpack like a regular tuple
>>> x, y
(11, 22)
>>> p.x + p.y              # fields also accessible by name
33
>>> d = p._asdict()        # convert to a dictionary
>>> d['x']
11
>>> Point(**d)             # convert from a dictionary
Point(x=11, y=22)
>>> p._replace(x=100)     # _replace() is like str.replace() but
↳ targets named fields
Point(x=100, y=22)

```

<http://code.activestate.com/recipes/500261-named-tuples/> :codeauthor: Raymond Hettinger :license: PSF

Changes: * autopep8 reformatting

simplejson Package

simplejson Package

decoder Module

encoder Module

ordered_dict Module

scanner Module

tool Module

hosts Package

hosts Package

This is a convenience module to import all available types of hosts.

Implementation details: You should 'import hosts' instead of importing every available host module.

base_classes Module

This module defines the base classes for the Host hierarchy.

Implementation details: You should import the "hosts" package instead of importing each type of host.

Host: a machine on which you can run programs

class `autotest.client.shared.hosts.base_classes.Host` (*args, **dargs)

Bases: `object`

This class represents a machine on which you can run programs.

It may be a local machine, the one autoserv is running on, a remote machine or a virtual machine.

Implementation details: This is an abstract class, leaf subclasses must implement the methods listed here. You must not instantiate this class but should instantiate one of those leaf subclasses.

When overriding methods that raise `NotImplementedError`, the leaf class is fully responsible for the implementation and should not chain calls to super. When overriding methods that are a NOP in Host, the subclass should chain calls to super(). The criteria for fitting a new method into one category or the other should be:

- 1.If two separate generic implementations could reasonably be concatenated, then the abstract implementation should pass and subclasses should chain calls to super.
- 2.If only one class could reasonably perform the stated function (e.g. two separate `run()` implementations cannot both be executed) then the method should raise `NotImplementedError` in Host, and the implementor should NOT chain calls to super, to ensure that only one implementation ever gets executed.

DEFAULT_REBOOT_TIMEOUT = 1800

HARDWARE_REPAIR_REQUEST_THRESHOLD = 4

HOURS_TO_WAIT_FOR_RECOVERY = 2.5

WAIT_DOWN_REBOOT_TIMEOUT = 840

WAIT_DOWN_REBOOT_WARNING = 540

check_diskspace (*path*, *gb*)

Raises an error if path does not have at least gb GB free.

:param path The path to check for free disk space. :param gb A floating point number to compare with a granularity

of 1 MB.

1000 based SI units are used.

:raise `AutoservDiskFullHostError` if path has less than gb GB free.

check_partitions (*root_part*, *filter_func=None*)

Compare the contents of `/proc/partitions` with those of `/proc/mounts` and raise exception in case unmounted partitions are found

root_part: in Linux `/proc/mounts` will never directly mention the root partition as being mounted on `/` instead it will say that `/dev/root` is mounted on `/`. Thus require this argument to filter out the *root_part* from the ones checked to be mounted

`filter_func`: unary predicate for additional filtering out of partitions required to be mounted

Raise: `error.AutoservHostError` if unfiltered unmounted partition found

cleanup ()

cleanup_kernels (*boot_dir='/boot'*)

Remove any kernel image and associated files (`vmlinux`, `system.map`, `modules`) for any image found in the boot directory that is not referenced by entries in the bootloader configuration.

Parameters `boot_dir` – boot directory path string, default `'/boot'`

close ()

disable_ipfilters ()

Allow all network packets in and out of the host.

enable_ipfilters ()

Re-enable the IP filters disabled from `disable_ipfilters()`

erase_dir_contents (*path, ignore_status=True, timeout=3600*)

Empty a given directory path contents.

get_arch ()

Get the hardware architecture of the remote machine.

get_autodir ()

get_boot_id (*timeout=60*)

Get a unique ID associated with the current boot.

Should return a string with the semantics such that two separate calls to `Host.get_boot_id()` return the same string if the host did not reboot between the two calls, and two different strings if it has rebooted at least once between the two calls.

:param timeout The number of seconds to wait before timing out.

Returns A string unique to this boot or `None` if not available.

get_cmdline ()

Get the kernel command line of the remote machine.

get_file (*source, dest, delete_dest=False*)

get_kernel_ver ()

Get the kernel version of the remote machine.

get_meminfo ()

Get the kernel memory info (`/proc/meminfo`) of the remote machine and return a dictionary mapping the various statistics.

get_num_cpu ()

Get the number of CPUs in the host according to `/proc/cpuinfo`.

get_open_func (*use_cache=True*)

Defines and returns a function that may be used instead of built-in `open()` to open and read files. The returned function is implemented by using `self.run('cat <file>')` and may cache the results for the same filename.

:param use_cache Cache results of `self.run('cat <filename>')` for the same filename

Returns a function that can be used instead of built-in `open()`

get_tmp_dir ()

get_wait_up_processes ()

Gets the list of local processes to wait for in wait_up.

install (*installableObject*)

is_shutting_down ()

Indicates is a machine is currently shutting down.

is_up ()

job = None

list_files_glob (*glob*)

Get a list of files on a remote host given a glob pattern path.

log_kernel ()

Helper method for logging kernel information into the status logs. Intended for cases where the “current” kernel is not really defined and we want to explicitly log it. Does nothing if this host isn’t actually associated with a job.

log_reboot (*reboot_func*)

Decorator for wrapping a reboot in a group for status logging purposes. The *reboot_func* parameter should be an actual function that carries out the reboot.

machine_install ()

path_exists (*path*)

Determine if path exists on the remote machine.

reboot ()

reboot_followup (**args, **dargs*)

reboot_setup (**args, **dargs*)

record (**args, **dargs*)

Helper method for recording status logs against *Host.job* that silently becomes a NOP if *Host.job* is not available. The *args* and *dargs* are passed on to *Host.job.record* unchanged.

repair_filesystem_only ()

perform file system repairs only

repair_full ()

repair_full_disk (*mountpoint*)

repair_software_only ()

perform software repairs only

repair_with_protection (*protection_level*)

Perform the maximal amount of repair within the specified protection level.

Parameters protection_level – the protection level to use for limiting repairs, a *host_protections.Protection*

request_hardware_repair ()

Should somehow request (send a mail?) for hardware repairs on this machine. The implementation can either return by raising the special error *AutoservHardwareRepairRequestedError* exception or can try to wait until the machine is repaired and then return normally.

run (*command, timeout=3600, ignore_status=False, stdout_tee=<object object>, stderr_tee=<object object>, stdin=None, args=()*)

Run a command on this host.

Parameters

- **command** – the command line string
- **timeout** – time limit in seconds before attempting to kill the running process. The `run()` function will take a few seconds longer than ‘timeout’ to complete if it has to kill the process.
- **ignore_status** – do not raise an exception, no matter what the exit code of the command is.
- **stdout_tee/stderr_tee** – where to tee the stdout/stderr
- **stdin** – stdin to pass (a string) to the executed command
- **args** – sequence of strings to pass as arguments to command by quoting them in ” and escaping their contents if necessary

Returns a `utils.CmdResult` object

Raises `AutotestHostRunError` – the exit code of the command execution was not 0 and `ignore_status` was not enabled

run_output (*command*, **args*, ***dargs*)

send_file (*source*, *dest*, *delete_dest=False*)

set_autodir ()

setup ()

start_loggers ()

Called to start continuous host logging.

stop_loggers ()

Called to stop continuous host logging.

symlink_closure (*paths*)

Given a sequence of path strings, return the set of all paths that can be reached from the initial set by following symlinks.

Parameters *paths* – sequence of path strings.

Returns a sequence of path strings that are all the unique paths that can be reached from the given ones after following symlinks.

sysrq_reboot ()

verify ()

verify_connectivity ()

verify_hardware ()

verify_software ()

wait_down (*timeout=None*, *warning_timer=None*, *old_boot_id=None*)

wait_for_restart (*timeout=1800*, *down_timeout=840*, *down_warning=540*, *log_failure=True*, *old_boot_id=None*, ***dargs*)

Wait for the host to come back from a reboot. This is a generic implementation based entirely on `wait_up` and `wait_down`.

wait_up (*timeout=None*)

common Module

test_utils Package

config_change_validation Module

Module for testing config file changes.

author Kristof Katus and Plamen Dimitrov

copyright Intra2net AG 2012

@license: GPL v2

autotest.client.shared.test_utils.config_change_validation.**assert_config_change** (*actual_result*,
ex-pected_result)

Wrapper of the upper method returning boolean true if no config changes were detected.

autotest.client.shared.test_utils.config_change_validation.**assert_config_change_dict** (*actual_r*,
ex-pected_

Calculates unexpected line changes.

The arguments *actual_result* and *expected_results* are of the same data structure type: Dict[file_path] -> (adds, removes), where adds = [added_line, ...] and removes = [removed_line, ...].

The return value has the following structure: Dict[file_path] -> (unexpected_adds,
not_present_adds, unexpected_removes, not_present_removes)

autotest.client.shared.test_utils.config_change_validation.**del_temp_file_copies** (*file_paths*)
Deletes all the provided files

autotest.client.shared.test_utils.config_change_validation.**extract_config_changes** (*file_paths*,
com-pared_file_p

Extracts diff information based on the new and temporarily saved old config files

Returns a dictionary of file path and corresponding diff information key-value pairs.

autotest.client.shared.test_utils.config_change_validation.**get_temp_file_path** (*file_path*)
Generates a temporary filename

autotest.client.shared.test_utils.config_change_validation.**make_temp_file_copies** (*file_paths*)
Creates temporary copies of the provided files

autotest.client.shared.test_utils.config_change_validation.**parse_unified_diff_output** (*lines*)
Parses the unified diff output of two files

Returns a pair of adds and removes, where each is a list of trimmed lines

autotest.client.shared.test_utils.config_change_validation.**print_change_diffs** (*change_diffs*)
Pretty prints the output of the evaluate_config_changes function

functools_24 Module

autotest.client.shared.test_utils.functools_24.**compose** (**args*)

autotest.client.shared.test_utils.functools_24.**fastcut** (**sargs*, ***skw*)

mock Module

exception `autotest.client.shared.test_utils.mock.CheckPlaybackError`

Bases: `exceptions.Exception`

Raised when mock playback does not match recorded calls.

class `autotest.client.shared.test_utils.mock.SaveDataAfterCloseStringIO (buf='')`

Bases: `StringIO.StringIO`

Saves the contents in a `final_data` property when `close()` is called.

Useful as a mock output file object to test both that the file was closed and what was written.

Properties:

final_data: Set to the `StringIO`'s `getvalue()` data when `close()` is called. None if `close()` has not been called.

`close()`

`final_data = None`

exception `autotest.client.shared.test_utils.mock.StubNotFoundError`

Bases: `exceptions.Exception`

Raised when god is asked to unstub an attribute that was not stubbed

class `autotest.client.shared.test_utils.mock.anything_comparator`

Bases: `autotest.client.shared.test_utils.mock.argument_comparator`

`is_satisfied_by (parameter)`

class `autotest.client.shared.test_utils.mock.argument_comparator`

Bases: `object`

`is_satisfied_by (parameter)`

class `autotest.client.shared.test_utils.mock.base_mapping (symbol, return_obj, *args, **dargs)`

Bases: `object`

`match (*args, **dargs)`

class `autotest.client.shared.test_utils.mock.equality_comparator (value)`

Bases: `autotest.client.shared.test_utils.mock.argument_comparator`

`is_satisfied_by (parameter)`

class `autotest.client.shared.test_utils.mock.function_any_args_mapping (symbol, re-
turn_val,
*args,
**dargs)`

Bases: `autotest.client.shared.test_utils.mock.function_mapping`

A mock function mapping that doesn't verify its arguments.

`match (*args, **dargs)`

class `autotest.client.shared.test_utils.mock.function_mapping (symbol, return_val,
*args, **dargs)`

Bases: `autotest.client.shared.test_utils.mock.base_mapping`

`and_raises (error)`

and_return (*return_obj*)

class `autotest.client.shared.test_utils.mock.is_instance_comparator` (*cls*)
 Bases: `autotest.client.shared.test_utils.mock.argument_comparator`

is_satisfied_by (*parameter*)

class `autotest.client.shared.test_utils.mock.is_string_comparator`
 Bases: `autotest.client.shared.test_utils.mock.argument_comparator`

is_satisfied_by (*parameter*)

class `autotest.client.shared.test_utils.mock.mask_function` (*symbol*, *original_function*, *default_return_val=None*, *record=None*, *playback=None*)
 Bases: `autotest.client.shared.test_utils.mock.mock_function`

run_original_function (**args*, ***dargs*)

class `autotest.client.shared.test_utils.mock.mock_class` (*cls*, *name*, *default_ret_val=None*, *record=None*, *playback=None*)
 Bases: `object`

class `autotest.client.shared.test_utils.mock.mock_function` (*symbol*, *default_return_val=None*, *record=None*, *playback=None*)
 Bases: `object`

expect_any_call ()

Like `expect_call` but don't give a hoot what arguments are passed.

expect_call (**args*, ***dargs*)

class `autotest.client.shared.test_utils.mock.mock_god` (*debug=False*, *fail_fast=True*, *ut=None*)
 Bases: `object`

NONEXISTENT_ATTRIBUTE = <object object>

check_playback ()

Report any errors that were encountered during calls to `__method_playback()`.

create_mock_class (*cls*, *name*, *default_ret_val=None*)

Given something that defines a namespace *cls* (class, object, module), and a (hopefully unique) *name*, will create a `mock_class` object with that name and that possesses all the public attributes of *cls*. *default_ret_val* sets the `default_ret_val` on all methods of the *cls* mock.

create_mock_class_obj (*cls*, *name*, *default_ret_val=None*)

create_mock_function (*symbol*, *default_return_val=None*)

create a `mock_function` with name *symbol* and default return value of *default_ret_val*.

mock_io ()

Mocks and saves the stdout & stderr output

mock_up (*obj*, *name*, *default_ret_val=None*)

Given an object (class instance or module) and a registration name, then replace all its methods with mock function objects (passing the original functions to the mock functions).

set_fail_fast (*fail_fast*)

stub_class (*namespace, symbol*)

stub_class_method (*cls, symbol*)

stub_function (*namespace, symbol*)

stub_function_to_return (*namespace, symbol, object_to_return*)

Stub out a function with one that always returns a fixed value.

:param namespace The namespace containing the function to stub out. :param symbol The attribute within the namespace to stub out. :param object_to_return The value that the stub should return whenever

it is called.

stub_with (*namespace, symbol, new_attribute*)

unmock_io ()

Restores the stdout & stderr, and returns both output strings

unstub (*namespace, symbol*)

unstub_all ()

class `autotest.client.shared.test_utils.mock.regex_comparator` (*pattern, flags=0*)

Bases: `autotest.client.shared.test_utils.mock.argument_comparator`

is_satisfied_by (*parameter*)

unittest Module

Python unit testing framework, based on Erich Gamma's JUnit and Kent Beck's Smalltalk testing framework.

This module contains the core framework classes that form the basis of specific test cases and suites (TestCase, TestSuite etc.), and also a text-based utility class for running the tests and reporting the results

(TextTestRunner).

Simple usage:

```
import unittest
```

```
class IntegerArithmeticTestCase(unittest.TestCase):
```

```
    def testAdd(self): ## test method names begin 'test*' self.assertEqual((1 + 2), 3)
        self.assertEqual(0 + 1, 1)
```

```
    def testMultiply(self): self.assertEqual((0 * 10), 0) self.assertEqual((5 * 8), 40)
```

```
if __name__ == '__main__': unittest.main()
```

Further information is available in the bundled documentation, and from

<http://docs.python.org/library/unittest.html>

Copyright (c) 1999-2003 Steve Purcell Copyright (c) 2003-2009 Python Software Foundation Copyright (c) 2009 Garrett Cooper This module is free software, and you may redistribute it and/or modify it under the same terms as Python itself, so long as this copyright message and disclaimer are retained in their original form.

IN NO EVENT SHALL THE AUTHOR BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS CODE, EVEN IF THE AUTHOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHOR SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE CODE PROVIDED HEREUNDER IS ON AN “AS IS” BASIS, AND THERE IS NO OBLIGATION WHATSOEVER TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Garrett: This module was backported using source from r71263 with fixes noted in Issue 5771.

class `autotest.client.shared.test_utils.unittest.TestResult`

Bases: `object`

Holder for test result information.

Test results are automatically managed by the `TestCase` and `TestSuite` classes, and do not need to be explicitly manipulated by writers of tests.

Each instance holds the total number of tests run, and collections of failures and errors that occurred among those test runs. The collections contain tuples of (`testcase`, `exceptioninfo`), where `exceptioninfo` is the formatted traceback of the error that occurred.

addError (*test*, *err*)

Called when an error has occurred. ‘err’ is a tuple of values as returned by `sys.exc_info()`.

addExpectedFailure (*test*, *err*)

Called when an expected failure/error occurred.

addFailure (*test*, *err*)

Called when an error has occurred. ‘err’ is a tuple of values as returned by `sys.exc_info()`.

addSkip (*test*, *reason*)

Called when a test is skipped.

addSuccess (*test*)

Called when a test has completed successfully

addUnexpectedSuccess (*test*)

Called when a test was expected to fail, but succeed.

startTest (*test*)

Called when the given test is about to be run

stop ()

Indicates that the tests should be aborted

stopTest (*test*)

Called when the given test has been run

wasSuccessful ()

Tells whether or not this result was a success

class `autotest.client.shared.test_utils.unittest.TestCase` (*methodName*=‘runTest’)

Bases: `object`

A class whose instances are single test cases.

By default, the test code itself should be placed in a method named ‘runTest’.

If the fixture may be used for many test cases, create as many test methods as are needed. When instantiating such a `TestCase` subclass, specify in the constructor arguments the name of the test method that the instance is to execute.

Test authors should subclass `TestCase` for their own tests. Construction and deconstruction of the test’s environment (‘fixture’) can be implemented by overriding the ‘setUp’ and ‘tearDown’ methods respectively.

If it is necessary to override the `__init__` method, the base class `__init__` method must always be called. It is important that subclasses should not change the signature of their `__init__` method, since instances of the classes are instantiated automatically by parts of the framework in order to be run.

addTypeEqualityFunc (*typeobj, function*)

Add a type specific `assertEqual` style function to compare a type.

This method is for use by `TestCase` subclasses that need to register their own type equality functions to provide nicer error messages.

Args:

typeobj: The data type to call this function on when both values are of the same type in `assertEqual()`.

function: The callable taking two arguments and an optional `msg=` argument that raises `self.failureException` with a useful error message when the two arguments are not equal.

assertAlmostEqual (*first, second, places=7, msg=None*)

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

assertAlmostEquals (*first, second, places=7, msg=None*)

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

assertDictContainsSubset (*expected, actual, msg=None*)

Checks whether `actual` is a superset of `expected`.

assertDictEqual (*d1, d2, msg=None*)

assertEqual (*first, second, msg=None*)

Fail if the two objects are unequal as determined by the `'=='` operator.

assertEquals (*first, second, msg=None*)

Fail if the two objects are unequal as determined by the `'=='` operator.

assertFalse (*expr, msg=None*)

Fail the test if the expression is true.

assertGreater (*a, b, msg=None*)

Just like `self.assertTrue(a > b)`, but with a nicer default message.

assertGreaterEqual (*a, b, msg=None*)

Just like `self.assertTrue(a >= b)`, but with a nicer default message.

assertIn (*member, container, msg=None*)

Just like `self.assertTrue(a in b)`, but with a nicer default message.

assertIs (*expr1, expr2, msg=None*)

Just like `self.assertTrue(a is b)`, but with a nicer default message.

assertIsNone (*obj, msg=None*)

Same as `self.assertTrue(obj is None)`, with a nicer default message.

assertIsNot (*expr1, expr2, msg=None*)

Just like `self.assertTrue(a is not b)`, but with a nicer default message.

assertIsNotNone (*obj, msg=None*)

Included for symmetry with `assertIsNone`.

assertLess (*a, b, msg=None*)

Just like `self.assertTrue(a < b)`, but with a nicer default message.

assertLessEqual (*a, b, msg=None*)

Just like `self.assertTrue(a <= b)`, but with a nicer default message.

assertListEqual (*list1, list2, msg=None*)

A list-specific equality assertion.

Args: `list1`: The first list to compare. `list2`: The second list to compare. `msg`: Optional message to use on failure instead of a list of differences.

assertMultiLineEqual (*first, second, msg=None*)

Assert that two multi-line strings are equal.

assertNotAlmostEqual (*first, second, places=7, msg=None*)

Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

assertNotAlmostEquals (*first, second, places=7, msg=None*)

Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

assertNotEqual (*first, second, msg=None*)

Fail if the two objects are equal as determined by the `'=='` operator.

assertNotEquals (*first, second, msg=None*)

Fail if the two objects are equal as determined by the `'=='` operator.

assertNotIn (*member, container, msg=None*)

Just like `self.assertTrue(a not in b)`, but with a nicer default message.

assertRaises (*excClass, callableObj=None, *args, **kwargs*)

Fail unless an exception of class `excClass` is thrown by `callableObj` when invoked with arguments `args` and keyword arguments `kwargs`. If a different type of exception is thrown, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with `callableObj` omitted or `None`, will return a context object used like this:

```
with self.assertRaises(some_error_class):
    do_something()
```

assertRaisesRegexp (*expected_exception, expected_regexp, callable_obj=None, *args, **kwargs*)

Asserts that the message in a raised exception matches a regexp.

Args: `expected_exception`: Exception class expected to be raised. `expected_regexp`: Regexp (re pattern object or string) expected

to be found in error message.

`callable_obj`: Function to be called. `args`: Extra args. `kwargs`: Extra kwargs.

assertRegexpMatches (*text, expected_regex, msg=None*)

assertSameElements (*expected_seq, actual_seq, msg=None*)

An unordered sequence specific comparison.

Raises with an error message listing which elements of `expected_seq` are missing from `actual_seq` and vice versa if any.

assertSequenceEqual (*seq1, seq2, msg=None, seq_type=None*)

An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Args: `seq1`: The first sequence to compare. `seq2`: The second sequence to compare. `seq_type`: The expected datatype of the sequences, or `None` if no

datatype should be enforced.

msg: Optional message to use on failure instead of a list of differences.

assertSetEqual (*set1, set2, msg=None*)

A set-specific equality assertion.

Args: `set1`: The first set to compare. `set2`: The second set to compare. `msg`: Optional message to use on failure instead of a list of

differences.

For more general containership equality, `assertSameElements` will work with things other than sets. This uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue (*expr, msg=None*)

Fail the test unless the expression is true.

assertTupleEqual (*tuple1, tuple2, msg=None*)

A tuple-specific equality assertion.

Args: `tuple1`: The first tuple to compare. `tuple2`: The second tuple to compare. `msg`: Optional message to use on failure instead of a list of

differences.

assert_ (*expr, msg=None*)

Fail the test unless the expression is true.

countTestCases ()

debug ()

Run the test without collecting errors in a `TestResult`

defaultTestResult ()

fail (*msg=None*)

Fail immediately, with the given message.

failIf (**args, **kwargs*)

failIfAlmostEqual (**args, **kwargs*)

failIfEqual (**args, **kwargs*)

failUnless (**args, **kwargs*)

failUnlessAlmostEqual (*args, **kwargs)

failUnlessEqual (*args, **kwargs)

failUnlessRaises (*args, **kwargs)

failureException

alias of AssertionError

id ()

longMessage = False

run (result=None)

setUp ()

Hook method for setting up the test fixture before exercising it.

shortDescription ()

Returns both the test method name and first line of its docstring.

If no docstring is given, only returns the method name.

This method overrides unittest.TestCase.shortDescription(), which only returns the first line of the docstring, obscuring the name of the test upon failure.

skipTest (reason)

Skip this test.

tearDown ()

Hook method for deconstructing the test fixture after testing it.

class autotest.client.shared.test_utils.unittest.**TestSuite** (tests=())

Bases: object

A test suite is a composite test consisting of a number of TestCases.

For use, create an instance of TestSuite, then add test case instances. When all tests have been added, the suite can be passed to a test runner, such as TextTestRunner. It will run the individual test cases in the order in which they were added, aggregating the results. When subclassing, do not forget to call the base class constructor.

addTest (test)

addTests (tests)

countTestCases ()

debug ()

Run the tests without collecting errors in a TestResult

run (result)

class autotest.client.shared.test_utils.unittest.**ClassTestSuite** (tests,

class_collected_from)

Bases: autotest.client.shared.test_utils.unittest.TestSuite

Suite of tests derived from a single TestCase class.

id ()

run (result)

shortDescription ()

class `autotest.client.shared.test_utils.unittest.TextTestRunner` (*stream=<open file '<stderr>', mode 'w', descriptions=1, verbosity=1*)

Bases: `object`

A test runner class that displays results in textual form.

It prints out the names of tests as they are run, errors as they occur, and a summary of the results at the end of the test run.

run (*test*)
Run the given test case or test suite.

class `autotest.client.shared.test_utils.unittest.TestLoader`
Bases: `object`

This class is responsible for loading tests according to various criteria and returning them wrapped in a `TestSuite`

classSuiteClass
alias of `ClassTestSuite`

getTestCaseNames (*testCaseClass*)
Return a sorted sequence of method names found within `testCaseClass`

loadTestsFromModule (*module*)
Return a suite of all tests cases contained in the given module

loadTestsFromName (*name, module=None*)
Return a suite of all tests cases given a string specifier.

The name may resolve either to a module, a test case class, a test method within a test case class, or a callable object which returns a `TestCase` or `TestSuite` instance.

The method optionally resolves the names relative to a given module.

loadTestsFromNames (*names, module=None*)
Return a suite of all tests cases found using the given sequence of string specifiers. See `'loadTestsFromName()'`.

loadTestsFromTestCase (*testCaseClass*)
Return a suite of all tests cases contained in `testCaseClass`

sortTestMethodsUsing ()
cmp(x, y) -> integer
Return negative if $x < y$, zero if $x == y$, positive if $x > y$.

suiteClass
alias of `TestSuite`

testMethodPrefix = `'test'`

class `autotest.client.shared.test_utils.unittest.FunctionTestCase` (*testFunc, setUp=None, tearDown=None, description=None*)

Bases: `autotest.client.shared.test_utils.unittest.TestCase`

A test case that wraps a test function.

This is useful for slipping pre-existing test functions into the unittest framework. Optionally, set-up and tidy-up functions can be supplied. As with TestCase, the tidy-up ('tearDown') function will always be called if the set-up ('setUp') function ran successfully.

```

id()
runTest()
setUp()
shortDescription()
tearDown()

```

```

autotest.client.shared.test_utils.unittest.main
    alias of TestProgram

```

exception autotest.client.shared.test_utils.unittest.**SkipTest**
 Bases: `exceptions.Exception`

Raise this exception in a test to skip it.

Usually you can use `TestResult.skip()` or one of the skipping decorators instead of raising this directly.

```

autotest.client.shared.test_utils.unittest.skip(reason)
    Unconditionally skip a test.

```

```

autotest.client.shared.test_utils.unittest.skipIf(condition, reason)
    Skip a test if the condition is true.

```

```

autotest.client.shared.test_utils.unittest.skipUnless(condition, reason)
    Skip a test unless the condition is true.

```

```

autotest.client.shared.test_utils.unittest.expectedFailure(func)

```

```

autotest.client.shared.test_utils.unittest.getTestCaseNames(testCaseClass,
                                                             prefix,
                                                             sortUsing=<built-in
                                                             function cmp>)

```

```

autotest.client.shared.test_utils.unittest.makeSuite(testCaseClass, prefix='test',
                                                       sortUsing=<built-in function
                                                       cmp>, suiteClass=<class 'au-
                                                       totest.client.shared.test_utils.unittest.TestSuite'>)

```

```

autotest.client.shared.test_utils.unittest.findTestCases(module, prefix='test',
                                                           sortUsing=<built-
                                                           in function cmp>,
                                                           suiteClass=<class 'au-
                                                           totest.client.shared.test_utils.unittest.TestSuite'>)

```

tools Package

JUnit_api Module

```

class autotest.client.tools.JUnit_api.errorType(message=None, type=None, val-
                                                ueOf=None)
    Bases: autotest.client.tools.JUnit_api.GeneratedSuper

```

The error message. e.g., if a java exception is thrown, the return value of `getMessage()`The type of error that occurred. e.g., if a java exception is thrown the full class name of the exception.

```

build(node)

```

```

buildAttributes (node, attrs, already_processed)
buildChildren (child_, node, nodeName_, fromsubclass_=False)
export (outfile, level, namespace_='', name_='errorType', namespacedef_='')
exportAttributes (outfile, level, already_processed, namespace_='', name_='errorType')
exportChildren (outfile, level, namespace_='', name_='errorType', fromsubclass_=False)
exportLiteral (outfile, level, name_='errorType')
exportLiteralAttributes (outfile, level, already_processed, name_)
exportLiteralChildren (outfile, level, name_)
static factory (*args_, **kwargs_)
get_message ()
get_type ()
get_valueOf_ ()
hasContent_ ()
set_message (message)
set_type (type_)
set_valueOf_ (valueOf_)
subclass = None
superclass = None
class autotest.client.tools.JUnit_api.failureType (message=None, type_=None, valueOf_=None)
    Bases: autotest.client.tools.JUnit_api.GeneratedSuper
    The message specified in the assertThe type of the assert.
build (node)
buildAttributes (node, attrs, already_processed)
buildChildren (child_, node, nodeName_, fromsubclass_=False)
export (outfile, level, namespace_='', name_='failureType', namespacedef_='')
exportAttributes (outfile, level, already_processed, namespace_='', name_='failureType')
exportChildren (outfile, level, namespace_='', name_='failureType', fromsubclass_=False)
exportLiteral (outfile, level, name_='failureType')
exportLiteralAttributes (outfile, level, already_processed, name_)
exportLiteralChildren (outfile, level, name_)
static factory (*args_, **kwargs_)
get_message ()
get_type ()
get_valueOf_ ()
hasContent_ ()
set_message (message)

```

```
    set_type (type_)
    set_valueOf_ (valueOf_)
    subclass = None
    superclass = None
class autotest.client.tools.JUnit_api.propertiesType (property=None)
    Bases: autotest.client.tools.JUnit_api.GeneratedsSuper
    add_property (value)
    build (node)
    buildAttributes (node, attrs, already_processed)
    buildChildren (child_, node, nodeName_, fromsubclass_=False)
    export (outfile, level, namespace_='', name_='propertiesType', namespacedef_='')
    exportAttributes (outfile, level, already_processed, namespace_='', name_='propertiesType')
    exportChildren (outfile, level, namespace_='', name_='propertiesType', fromsubclass_=False)
    exportLiteral (outfile, level, name_='propertiesType')
    exportLiteralAttributes (outfile, level, already_processed, name_)
    exportLiteralChildren (outfile, level, name_)
    static factory (*args_, **kwargs_)
    get_property ()
    hasContent_ ()
    insert_property (index, value)
    set_property (property)
    subclass = None
    superclass = None
class autotest.client.tools.JUnit_api.propertyType (name=None, value=None)
    Bases: autotest.client.tools.JUnit_api.GeneratedsSuper
    build (node)
    buildAttributes (node, attrs, already_processed)
    buildChildren (child_, node, nodeName_, fromsubclass_=False)
    export (outfile, level, namespace_='', name_='propertyType', namespacedef_='')
    exportAttributes (outfile, level, already_processed, namespace_='', name_='propertyType')
    exportChildren (outfile, level, namespace_='', name_='propertyType', fromsubclass_=False)
    exportLiteral (outfile, level, name_='propertyType')
    exportLiteralAttributes (outfile, level, already_processed, name_)
    exportLiteralChildren (outfile, level, name_)
    static factory (*args_, **kwargs_)
    get_name ()
    get_value ()
```

```

hasContent_ ()
set_name (name)
set_value (value)
subclass = None
superclass = None

```

```

class autotest.client.tools.JUnit_api.system_err
  Bases: autotest.client.tools.JUnit_api.GeneratedsSuper
  Data that was written to standard error while the test was executed

  build (node)
  buildAttributes (node, attrs, already_processed)
  buildChildren (child_, node, nodeName_, fromsubclass_=False)
  export (outfile, level, namespace_=' ', name_='system-err', namespacedef_=' ')
  exportAttributes (outfile, level, already_processed, namespace_=' ', name_='system-err')
  exportChildren (outfile, level, namespace_=' ', name_='system-err', fromsubclass_=False)
  exportLiteral (outfile, level, name_='system-err')
  exportLiteralAttributes (outfile, level, already_processed, name_)
  exportLiteralChildren (outfile, level, name_)
  static factory (*args_, **kwargs_)
  hasContent_ ()
  subclass = None
  superclass = None

```

```

class autotest.client.tools.JUnit_api.system_out
  Bases: autotest.client.tools.JUnit_api.GeneratedsSuper
  Data that was written to standard out while the test was executed

  build (node)
  buildAttributes (node, attrs, already_processed)
  buildChildren (child_, node, nodeName_, fromsubclass_=False)
  export (outfile, level, namespace_=' ', name_='system-out', namespacedef_=' ')
  exportAttributes (outfile, level, already_processed, namespace_=' ', name_='system-out')
  exportChildren (outfile, level, namespace_=' ', name_='system-out', fromsubclass_=False)
  exportLiteral (outfile, level, name_='system-out')
  exportLiteralAttributes (outfile, level, already_processed, name_)
  exportLiteralChildren (outfile, level, name_)
  static factory (*args_, **kwargs_)
  hasContent_ ()
  subclass = None
  superclass = None

```

```
class autotest.client.tools.JUnit_api.testcaseType (classname=None, name=None,
                                                    time=None, error=None, failure=None)
                                                    ure=None)
```

Bases: autotest.client.tools.JUnit_api.GeneratedsSuper

Name of the test methodFull class name for the class the test method is in.Time taken (in seconds) to execute the test

build (node)

buildAttributes (node, attrs, already_processed)

buildChildren (child_, node, nodeName_, fromsubclass_=False)

export (outfile, level, namespace_=' ', name_='testcaseType', namespacedef_='')

exportAttributes (outfile, level, already_processed, namespace_=' ', name_='testcaseType')

exportChildren (outfile, level, namespace_=' ', name_='testcaseType', fromsubclass_=False)

exportLiteral (outfile, level, name_='testcaseType')

exportLiteralAttributes (outfile, level, already_processed, name_)

exportLiteralChildren (outfile, level, name_)

static factory (*args_, **kwargs_)

get_classname ()

get_error ()

get_failure ()

get_name ()

get_time ()

hasContent_ ()

set_classname (classname)

set_error (error)

set_failure (failure)

set_name (name)

set_time (time)

subclass = None

superclass = None

```
class autotest.client.tools.JUnit_api.testsuite (tests=None, errors=None, name=None,
                                                  timestamp=None, hostname=None,
                                                  time=None, failures=None, properties=None,
                                                  testcase=None, system_out=None, system_err=None,
                                                  extensiontype_=None)
```

Bases: autotest.client.tools.JUnit_api.GeneratedsSuper

Contains the results of exexuting a testsuiteFull class name of the test for non-aggregated testsuite documents. Class name without the package for aggregated testsuites documentswhen the test was executed. Timezone may not be specified.Host on which the tests were executed. 'localhost' should be used if the hostname cannot be determined.The total number of tests in the suiteThe total number of tests in the suite that failed. A failure is a test which the code has explicitly failed by using the mechanisms for that purpose. e.g., via an assertEqualsThe

total number of tests in the suite that errored. An errored test is one that had an unanticipated problem. e.g., an unchecked throwable; or a problem with the implementation of the test. Time taken (in seconds) to execute the tests in the suite

```

add_testcase (value)
build (node)
buildAttributes (node, attrs, already_processed)
buildChildren (child_, node, nodeName_, fromsubclass_=False)
export (outfile, level, namespace_='', name_='testsuite', namespacedef_='')
exportAttributes (outfile, level, already_processed, namespace_='', name_='testsuite')
exportChildren (outfile, level, namespace_='', name_='testsuite', fromsubclass_=False)
exportLiteral (outfile, level, name_='testsuite')
exportLiteralAttributes (outfile, level, already_processed, name_)
exportLiteralChildren (outfile, level, name_)
static factory (*args_, **kwargs_)
get_errors ()
get_extensiontype_ ()
get_failures ()
get_hostname ()
get_name ()
get_properties ()
get_system_err ()
get_system_out ()
get_testcase ()
get_tests ()
get_time ()
get_timestamp ()
hasContent_ ()
insert_testcase (index, value)
set_errors (errors)
set_extensiontype_ (extensiontype_)
set_failures (failures)
set_hostname (hostname)
set_name (name)
set_properties (properties)
set_system_err (system_err)
set_system_out (system_out)
set_testcase (testcase)

```

```
set_tests (tests)
set_time (time)
set_timestamp (timestamp)
subclass = None
superclass = None
validate_ISO8601_DATETIME_PATTERN (value)
class autotest.client.tools.JUnit_api.testsuiteType (tests=None, errors=None,
name=None, timestamp=None,
hostname=None, time=None,
failures=None, properties=None,
testcase=None, system_out=None,
system_err=None, id=None, pack-
age=None)
Bases: autotest.client.tools.JUnit_api.testsuite
Derived from testsuite/@name in the non-aggregated documentsStarts at '0' for the first testsuite and is incre-
mented by 1 for each following testsuite
build (node)
buildAttributes (node, attrs, already_processed)
buildChildren (child_, node, nodeName_, fromsubclass_=False)
export (outfile, level, namespace_=' ', name_='testsuiteType', namespacedef_='')
exportAttributes (outfile, level, already_processed, namespace_=' ', name_='testsuiteType')
exportChildren (outfile, level, namespace_=' ', name_='testsuiteType', fromsubclass_=False)
exportLiteral (outfile, level, name_='testsuiteType')
exportLiteralAttributes (outfile, level, already_processed, name_)
exportLiteralChildren (outfile, level, name_)
static factory (*args_, **kwargs_)
get_id ()
get_package ()
hasContent_ ()
set_id (id)
set_package (package)
subclass = None
superclass
alias of testsuite
class autotest.client.tools.JUnit_api.testsuites (testsuite=None)
Bases: autotest.client.tools.JUnit_api.GeneratedsSuper
Contains an aggregation of testsuite results
add_testsuite (value)
build (node)
buildAttributes (node, attrs, already_processed)
```

```

buildChildren (child_, node, nodeName_, fromsubclass_=False)
export (outfile, level, namespace_='', name_='testsuites', namespacedef_='')
exportAttributes (outfile, level, already_processed, namespace_='', name_='testsuites')
exportChildren (outfile, level, namespace_='', name_='testsuites', fromsubclass_=False)
exportLiteral (outfile, level, name_='testsuites')
exportLiteralAttributes (outfile, level, already_processed, name_)
exportLiteralChildren (outfile, level, name_)
static factory (*args_, **kwargs_)
get_testsuite ()
hasContent_ ()
insert_testsuite (index, value)
set_testsuite (testsuite)
subclass = None
superclass = None

```

boottool Module

A boottool clone, but written in python and relying mostly on grubby[1].

[1] - <http://git.fedorahosted.org/git/?p=grubby.git>

class `autotest.client.tools.boottool.Grubby` (*path=None*, *opts=None*)

Bases: `object`

Grubby wrapper

This class calls the grubby binary for most commands, but also adds some functionality that is not really suited to be included in int, such as boot-once.

SUPPORTED_BOOTLOADERS = ('lilo', 'grub2', 'grub', 'extlinux', 'yaboot', 'elilo')

add_args (*kernel*, *args*)

Add cmdline arguments for the specified kernel.

Parameters

- **kernel** – can be a position number (index) or title
- **args** – argument to be added to the current list of args

add_kernel (*path*, *title='autoserv'*, *root=None*, *args=None*, *initrd=None*, *default=False*, *position='end'*)

Add a kernel entry to the bootloader (or replace if one exists already with the same title).

Parameters

- **path** – string path to the kernel image file
- **title** – title of this entry in the bootloader config
- **root** – string of the root device
- **args** – string with cmdline args
- **initrd** – string path to the initrd file

- **default** – set to True to make this entry the default one (default False)
- **position** – where to insert the new entry in the bootloader config file (default ‘end’, other valid input ‘start’, or # of the title)
- **xen_hypervisor** – xen hypervisor image file (valid only when xen mode is enabled)

arch_probe ()

Get the system architecture

This is much simpler version then the original boottool version, that does not attempt to filter the result of the command / system call that returns the architecture.

Returns string with system architecture, such as x86_64, ppc64, etc

boot_once (title=None)

Configures the bootloader to boot an entry only once

This is not implemented by grubby, but directly implemented here, via the ‘boot_once_<bootloader>’ method.

boot_once_elilo (entry_index)

Implements boot once for machines with kernel >= 2.6

This manipulates EFI variables via the interface available at /sys/firmware/efi/vars

boot_once_grub (entry_index)

Implements the boot once feature for the grub bootloader

boot_once_grub2 (entry_index)

Implements the boot once feature for the grub2 bootloader

Caveat: this assumes the default set is of type “saved”, and not a numeric value.

boot_once_yaboot (entry_title)

Implements the boot once feature for the yaboot bootloader

bootloader_probe ()

Get the bootloader name that is detected on this machine

This module performs the same action as client side boottool.py get_type() method, but with a better name IMHO.

Returns name of detected bootloader

default ()

Get the default entry index.

This module performs the same action as client side boottool.py get_default() method, but with a better name IMHO.

Returns an integer with the the default entry.

get_architecture ()

Get the system architecture

This is much simpler version then the original boottool version, that does not attempt to filter the result of the command / system call that returns the architecture.

Returns string with system architecture, such as x86_64, ppc64, etc

get_bootloader ()

Get the bootloader name that is detected on this machine

This module performs the same action as client side boottool.py get_type() method, but with a better name IMHO.

Returns name of detected bootloader

get_default ()

Get the default entry index.

This module performs the same action as client side boottool.py get_default() method, but with a better name IMHO.

Returns an integer with the the default entry.

get_default_index ()

Get the default entry index.

This module performs the same action as client side boottool.py get_default() method, but with a better name IMHO.

Returns an integer with the the default entry.

get_default_title ()

Get the default entry title.

Conforms to the client side boottool.py API, but rely directly on grubby functionality.

Returns a string of the default entry title.

get_entries ()

Get all entries information.

Returns a dictionary of index -> entry where entry is a dictionary of entry information as described for get_entry().

get_entry (*search_info*)

Get a single bootloader entry information.

NOTE: if entry is “fallback” and bootloader is grub use index instead of kernel title (“fallback”) as fallback is a special option in grub

Parameters *search_info* – can be ‘default’, position number or title

Returns a dictionary of key->value where key is the type of entry information (ex. ‘title’, ‘args’, ‘kernel’, etc) and value is the value for that piece of information.

get_grubby_version ()

Get the version of grubby that is installed on this machine

Returns tuple with (major, minor) grubby version

get_grubby_version_raw ()

Get the version of grubby that is installed on this machine as is

Returns string with raw output from grubby –version

get_info (*entry*=‘ALL’)

Returns information on a given entry, or all of them if not specified

The information is returned as a set of lines, that match the output of ‘grubby –info=<entry>’

Parameters *entry* (*string*) – entry description, usually an index starting from 0

Returns set of lines

get_info_lines (*entry*=‘ALL’)

Returns information on a given entry, or all of them if not specified

The information is returned as a set of lines, that match the output of ‘grubby –info=<entry>’

Parameters **entry** (*string*) – entry description, usually an index starting from 0

Returns set of lines

get_title_for_kernel (*path*)

Returns a title for a particular kernel.

Parameters **path** – path of the kernel image configured in the boot config

Returns if the given kernel path is found it will return a string with the title for the found entry, otherwise returns None

get_titles ()

Get the title of all boot entries.

Returns list with titles of boot entries

get_type ()

Get the bootloader name that is detected on this machine

This module performs the same action as client side boottool.py get_type() method, but with a better name IMHO.

Returns name of detected bootloader

grubby_build (*topdir, tarball*)

Attempts to build grubby from the source tarball

grubby_install (*path=None*)

Attempts to install a recent enough version of grubby

So far tested on:

- Fedora 16 x86_64
- Debian 6 x86_64
- SuSE 12.1 x86_64
- RHEL 4 on ia64 (with updated python 2.4)
- RHEL 5 on ia64
- RHEL 6 on ppc64

grubby_install_backup (*path*)

Backs up the current grubby binary to make room the one we’ll build

Parameters **path** (*string*) – path to the binary that should be backed up

grubby_install_fetch_tarball (*topdir*)

Fetches and verifies the grubby source tarball

grubby_install_patch_makefile ()

Patch makefile, making CFLAGS more forgivable to older toolchains

remove_args (*kernel, args*)

Removes specified cmdline arguments.

Parameters

- **kernel** – can be a position number (index) or title
- **args** – argument to be removed of the current list of args

remove_kernel (*kernel*)

Removes a specific entry from the bootloader configuration.

Parameters kernel – entry position or entry title.

FIXME: param kernel should also take ‘start’ or ‘end’.

set_default (*index*)

Sets the given entry number to be the default on every next boot

To set a default only for the next boot, use `boot_once()` instead.

This module performs the same action as client side `boottool.py set_default()` method, but with a better name IMHO.

Note: both `–set-default=<kernel>` and `–set-default-index=<index>` on `grubby` returns no error when it doesn’t find the kernel or index. So this method will, until `grubby` gets fixed, always return success.

Parameters index – entry index number to set as the default.

set_default_by_index (*index*)

Sets the given entry number to be the default on every next boot

To set a default only for the next boot, use `boot_once()` instead.

This module performs the same action as client side `boottool.py set_default()` method, but with a better name IMHO.

Note: both `–set-default=<kernel>` and `–set-default-index=<index>` on `grubby` returns no error when it doesn’t find the kernel or index. So this method will, until `grubby` gets fixed, always return success.

Parameters index – entry index number to set as the default.

class `autotest.client.tools.boottool.OptionParser` (***kwargs*)

Bases: `optparse.OptionParser`

Command line option parser

Aims to maintain compatibility at the command line level with `boottool`

check_values (*opts, args*)

Validate the option the user has supplied

option_parser_usage = ‘%prog [options]’

opts_get_action (*opts*)

Gets the selected action from the parsed opts

opts_has_action (*opts*)

Checks if (parsed) opts has a first class action

class `autotest.client.tools.boottool.EfiVar` (*name, data, guid=None, attributes=None*)

Bases: `object`

Helper class to manipulate EFI firmware variables

This class has no notion of the EFI firmware variables interface, that is, where it should read from or write to in order to create or delete EFI variables.

On systems with kernel \geq 2.6, that interface is a directory structure under `/sys/firmware/efi/vars`.

On systems with kernel \leq 2.4, that interface is going to be a directory structure under `/proc/efi/vars`. But be advised: this has not been tested yet on kernels \leq 2.4.

ATTR_BOOTSERVICE_ACCESS = 2

ATTR_NON_VOLATILE = 1

ATTR_RUNTIME_ACCESS = 4

DEFAULT_ATTRIBUTES = 7

FMT = '512H16B1L512H1L1I'

GUID_CONTENT = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

GUID_FMT = '16B'

get_data()

Returns the variable data in a list ready for struct.pack()

get_name()

Returns the variable name in a list ready for struct.pack()

get_packed()

Returns the EFI variable raw data packed by struct.pack()

This data should be written to the appropriate interface to create an EFI variable

class autotest.client.tools.boottool.**EfiToolSys**

Bases: `object`

Interfaces with `/sys/firmware/efi/vars` provided by the kernel

This interface is present on kernels `>= 2.6` with `CONFIG_EFI` and `CONFIG_EFI_VARS` options set.

BASE_PATH = '/sys/firmware/efi/vars'

DEL_VAR = '/sys/firmware/efi/vars/del_var'

NEW_VAR = '/sys/firmware/efi/vars/new_var'

check_basic_structure()

Checks the basic directory structure for the `/sys/.../vars` interface

create_variable (*name, data, guid=None, attributes=None*)

Creates a new EFI variable

Parameters

- **name** (*string*) – the name of the variable that will be created
- **data** (*string*) – user data that will populate the variable
- **guid** (*tuple*) – content for the guid value that composes the full variable name
- **attributes** – integer
- **attributes** – bitwise AND of the EFI attributes this variable will have set

delete_variable (*name, data, guid=None, attributes=None*)

Deletes an existing EFI variable

Parameters

- **name** (*string*) – the name of the variable that will be deleted
- **data** (*string*) – user data that will populate the variable
- **guid** (*tuple*) – content for the guid value that composes the full variable name
- **attributes** – integer
- **attributes** – bitwise AND of the EFI attributes this variable will have set

class `autotest.client.tools.boottool.EliloConf` (*path*='etc/elilo.conf')

Bases: `object`

A simple parser for elilo configuration file

Has simple features to add and remove global options only, as this is all we need. grubby takes care of manipulating the boot entries themselves.

add_global_option (*key*, *val*=None)

Adds a global option to the updated elilo configuration file

Parameters

- **key** (*string*) – option name
- **key** – option value or None for options with no values

Returns None

get_updated_content ()

Returns the config file content with options to add and remove applied

keyval_to_line (*keyval*)

Transforms a tuple into a text line suitable for the config file

Parameters **keyval** (*tuple*) – a tuple containing key and value

Returns a text line suitable for the config file

line_to_keyval (*line*)

Transforms a text line from the configuration file into a tuple

Parameters **line** (*string*) – line of text from the configuration file

Returns a tuple with key and value

matches_global_option_to_add (*line*)

Utility method to check if option is to be added

Parameters **line** (*string*) – line of text from the configuration file

Returns True or False

matches_global_option_to_remove (*line*)

Utility method to check if option is to be removed

Parameters **line** (*string*) – line of text from the configuration file

Returns True or False

remove_global_option (*key*, *val*=None)

Removes a global option to the updated elilo configuration file

Parameters

- **key** (*string*) – option name
- **key** – option value or None for options with no values

Returns None

update ()

Writes the updated content to the configuration file

`autotest.client.tools.boottool.find_executable` (*executable*, *favorite_path*=None)

Returns whether the system has a given executable

Parameters **executable** (*string*) – the name of a file that can be read and executed

`autotest.client.tools.boottool.parse_entry (entry_str, separator='')`
Parse entry as returned by boottool.

Parameters `entry_str` – one entry information as returned by boottool

Returns dictionary of key -> value where key is the string before the first ":" in an entry line and value is the string after it

common Module

crash_handler Module

Simple crash handling application for autotest

copyright Red Hat Inc 2009

author Lucas Meneghel Rodrigues <lmr@redhat.com>

`autotest.client.tools.crash_handler.gdb_report (path)`
Use GDB to produce a report with information about a given core.

Parameters `path` – Path to core file.

`autotest.client.tools.crash_handler.generate_random_string (length)`
Return a random string using alphanumeric characters.

@length: length of the string that will be generated.

`autotest.client.tools.crash_handler.get_info_from_core (path)`
Reads a core file and extracts a dictionary with useful core information.

Right now, the only information extracted is the full executable name.

Parameters `path` – Path to core file.

`autotest.client.tools.crash_handler.get_parent_pid (pid)`
Returns the parent PID for a given PID, converted to an integer.

Parameters `pid` – Process ID.

`autotest.client.tools.crash_handler.get_results_dir_list (pid, core_dir_basename)`
Get all valid output directories for the core file and the report. It works by inspecting files created by each test on /tmp and verifying if the PID of the process that crashed is a child or grandchild of the autotest test process. If it can't find any relationship (maybe a daemon that died during a test execution), it will write the core file to the debug dirs of all tests currently being executed. If there are no active autotest tests at a particular moment, it will return a list with ['/tmp'].

Parameters

- `pid` – PID for the process that generated the core
- `core_dir_basename` – Basename for the directory that will hold both the core dump and the crash report.

`autotest.client.tools.crash_handler.write_cores (core_data, dir_list)`
Write core files to all directories, optionally providing reports.

Parameters

- `core_data` – Contents of the core file.
- `dir_list` – List of directories the cores have to be written.
- `report` – Whether reports are to be generated for those core files.

`autotest.client.tools.crash_handler.write_to_file(filename, data, report=False)`
Write contents to a given file path specified. If not specified, the file will be created.

Parameters

- **file_path** – Path to a given file.
- **data** – File contents.
- **report** – Whether we'll use GDB to get a backtrace report of the file.

process_metrics Module

Program that parses autotest metrics results and prints them to stdout, so that the jenkins measurement-plots plugin can parse them.

Authors: Steve Conklin <sconklin@canonical.com> Brad Figg <brad.figg@canonical.com>

Copyright (C) 2012 Canonical Ltd.

This script is distributed under the terms and conditions of the GNU General Public License, Version 2 or later. See <http://www.gnu.org/copyleft/gpl.html> for details.

`autotest.client.tools.process_metrics.main(path)`

`autotest.client.tools.process_metrics.usage()`

regression Module

Program that parses standard format results, compute and check regression bug.

copyright Red Hat 2011-2012

author Amos Kong <akong@redhat.com>

class `autotest.client.tools.regression.Sample` (*type, arg*)

Bases: `object`

Collect test results in same environment to a sample

getAvg (*avg_update=None*)

getAvgPercent (*avgs_dict*)

getSD ()

getSDRate (*sds_dict*)

getTtestPvalue (*fs_dict1, fs_dict2, paired=None*)

scipy lib is used to compute p-value of Ttest scipy: <http://www.scipy.org/> t-test: http://en.wikipedia.org/wiki/Student's_t-test

`autotest.client.tools.regression.analyze(test, type, arg1, arg2, configfile)`

Compute averages/p-values of two samples, print results nicely

`autotest.client.tools.regression.display(lists, rates, allpvalues, f, ignore_col, sum='Augment Rate', prefix0=None, prefix1=None, prefix2=None, prefix3=None)`

Display lists data to standard format

param lists: row data lists param rates: augment rates lists param f: result output file param ignore_col: do not display some columns param sum: compare result summary param prefix0: output prefix in head lines param

prefix1: output prefix in Avg/SD lines param prefix2: output prefix in Diff Avg/P-value lines param prefix3:
output prefix in total Sign line

`autotest.client.tools.regression.exec_sql (cmd, conf='./global_config.ini')`

`autotest.client.tools.regression.get_test_keyval (jobid, keyname, default='')`

`autotest.client.tools.regression.is_int (n)`

`autotest.client.tools.regression.tee (content, file)`

Write content to standard output and file

results2junit Module

Program that parses the autotest results and generates JUnit test results in XML format.

`autotest.client.tools.results2junit.dbg (ostr)`

`autotest.client.tools.results2junit.dump (obj)`

`autotest.client.tools.results2junit.file_load (file_name)`

Load the indicated file into a string and return the string.

`autotest.client.tools.results2junit.main (basedir, resfiles)`

`autotest.client.tools.results2junit.parse_results (text)`

Parse text containing Autotest results.

Returns A list of result 4-tuples.

`autotest.client.tools.results2junit.text_clean (text)`

This always seems like such a hack, however, there are some characters that we can't deal with properly so this function just removes them from the text passed in.

scan_results Module

Program that parses the autotest results and return a nicely printed final test result.

copyright Red Hat 2008-2009

`autotest.client.tools.scan_results.main (resfiles)`

`autotest.client.tools.scan_results.parse_results (text)`

Parse text containing Autotest results.

Returns A list of result 4-tuples.

`autotest.client.tools.scan_results.print_result (result, name_width)`

Nicely print a single Autotest result.

Parameters

- **result** – a 4-tuple
- **name_width** – test name maximum width

CHAPTER 3

frontend Package

Subpackages

afe Package

`rpc_interface` Module

CHAPTER 4

Indices and tables

- `genindex`
- `modindex`
- `search`

a

autotest.client.autotest_local, 193
autotest.client.base_sysinfo, 193
autotest.client.base_utils, 194
autotest.client.bkr_proxy, 198
autotest.client.bkr_xml, 200
autotest.client.client_logging_config, 201
autotest.client.cmdparser, 201
autotest.client.common, 202
autotest.client.config, 202
autotest.client.cpuset, 202
autotest.client.fsdev_disks, 204
autotest.client.fsdev_mgr, 206
autotest.client.fsinfo, 206
autotest.client.harness, 207
autotest.client.harness_autoserv, 208
autotest.client.harness_beaker, 208
autotest.client.harness_simple, 209
autotest.client.harness_standalone, 210
autotest.client.job, 210
autotest.client.kernel, 213
autotest.client.kernel_config, 216
autotest.client.kernel_versions, 216
autotest.client.kernelexpand, 217
autotest.client.kvm_control, 217
autotest.client.local_host, 218
autotest.client.lv_utils, 218
autotest.client.net.basic_machine, 231
autotest.client.net.common, 231
autotest.client.net.net_tc, 231
autotest.client.net.net_utils, 234
autotest.client.net.net_utils_mock, 238
autotest.client.optparser, 219
autotest.client.os_dep, 219
autotest.client.parallel, 223
autotest.client.partition, 223
autotest.client.profiler, 228
autotest.client.profilers, 239
autotest.client.profilers.blktrace.blktrace, 239
autotest.client.profilers.catprofile.catprofile, 240
autotest.client.profilers.cmdprofile.cmdprofile, 240
autotest.client.profilers.cpiostat.cpiostat, 241
autotest.client.profilers.ftrace.ftrace, 241
autotest.client.profilers.inotify.inotify, 242
autotest.client.profilers.iostat.iostat, 242
autotest.client.profilers.kvm_stat.kvm_stat, 243
autotest.client.profilers.lockmeter.lockmeter, 243
autotest.client.profilers.lttng.lttng, 244
autotest.client.profilers.mpstat.mpstat, 244
autotest.client.profilers.oprofile.oprofile, 245
autotest.client.profilers.perf.perf, 245
autotest.client.profilers.powertop.powertop, 245
autotest.client.profilers.readprofile.readprofile, 246
autotest.client.profilers.sar.sar, 246
autotest.client.profilers.systemtap.systemtap, 247
autotest.client.profilers.vmstat.vmstat, 247
autotest.client.setup, 228
autotest.client.setup_job, 228
autotest.client.setup_modules, 229
autotest.client.shared.autotemp, 248
autotest.client.shared.backports, 329
autotest.client.shared.backports.collections,

[329](#)
[autotest.client.shared.backports.collect](#)
[330](#)
[autotest.client.shared.backports.collections.namedtuple,](#)
[331](#)
[autotest.client.shared.backports.collect](#)
[329](#)
[autotest.client.shared.barrier,](#) [248](#)
[autotest.client.shared.base_barrier,](#) [248](#)
[autotest.client.shared.base_check_version,](#)
[250](#)
[autotest.client.shared.base_job,](#) [250](#)
[autotest.client.shared.base_packages,](#)
[258](#)
[autotest.client.shared.base_syncdata,](#)
[263](#)
[autotest.client.shared.boottool,](#) [264](#)
[autotest.client.shared.check_version,](#)
[264](#)
[autotest.client.shared.common,](#) [264](#)
[autotest.client.shared.control_data,](#) [264](#)
[autotest.client.shared.distro,](#) [33](#)
[autotest.client.shared.distro_def,](#) [266](#)
[autotest.client.shared.enum,](#) [267](#)
[autotest.client.shared.error,](#) [268](#)
[autotest.client.shared.git,](#) [272](#)
[autotest.client.shared.host_protections,](#)
[274](#)
[autotest.client.shared.host_queue_entry_](#)
[274](#)
[autotest.client.shared.hosts,](#) [332](#)
[autotest.client.shared.hosts.base_classes,](#)
[332](#)
[autotest.client.shared.hosts.common,](#) [336](#)
[autotest.client.shared.iscsi,](#) [274](#)
[autotest.client.shared.iso9660,](#) [275](#)
[autotest.client.shared.jsontemplate,](#) [276](#)
[autotest.client.shared.kernel_versions,](#)
[278](#)
[autotest.client.shared.log,](#) [279](#)
[autotest.client.shared.logging_config,](#)
[279](#)
[autotest.client.shared.logging_manager,](#)
[280](#)
[autotest.client.shared.magic,](#) [282](#)
[autotest.client.shared.mail,](#) [282](#)
[autotest.client.shared.mock,](#) [283](#)
[autotest.client.shared.openvswitch,](#) [287](#)
[autotest.client.shared.packages,](#) [290](#)
[autotest.client.shared.pidfile,](#) [290](#)
[autotest.client.shared.profiler_manager,](#)
[291](#)
[autotest.client.shared.progressbar,](#) [291](#)
[autotest.client.shared.report,](#) [292](#)
[autotest.client.shared.service,](#) [293](#)
[autotest.client.shared.settings,](#) [294](#)
[autotest.client.shared.software_manager,](#)
[295](#)
[autotest.client.shared.syncdata,](#) [299](#)
[autotest.client.shared.test,](#) [299](#)
[autotest.client.shared.test_utils.config_change_val,](#)
[336](#)
[autotest.client.shared.test_utils.functools_24,](#)
[336](#)
[autotest.client.shared.test_utils.mock,](#)
[337](#)
[autotest.client.shared.test_utils.unittest,](#)
[339](#)
[autotest.client.shared.utils,](#) [301](#)
[autotest.client.shared.utils_cgroup,](#) [319](#)
[autotest.client.shared.utils_koji,](#) [322](#)
[autotest.client.shared.utils_memory,](#) [327](#)
[autotest.client.shared.version,](#) [328](#)
[autotest.client.sysinfo,](#) [229](#)
[autotest.client.test,](#) [229](#)
[autotest.client.test_config,](#) [230](#)
[autotest.client.tools.boottool,](#) [353](#)
[autotest.client.tools.common,](#) [360](#)
[autotest.client.tools.crash_handler,](#) [360](#)
[autotest.client.tools.JUnit_api,](#) [346](#)
[autotest.client.tools.process_metrics,](#)
[361](#)
[autotest.client.tools.regression,](#) [361](#)
[autotest.client.tools.results2junit,](#) [362](#)
[autotest.client.tools.scan_results,](#) [362](#)
[autotest.client.utils,](#) [230](#)
[autotest.client.xen,](#) [231](#)
[autotest.frontend.afe.model_logic,](#) [95](#)
[autotest.frontend.afe.models,](#) [95](#)
[autotest.frontend.tko.models,](#) [96](#)

A

- AB_MODE** (autotest.client.net.net_utils.bonding attribute), 234
abbrev_list() (in module autotest.client.cpuset), 202
active() (autotest.client.shared.profiler_manager.profiler_manager method), 291
AD_MODE (autotest.client.net.net_utils.bonding attribute), 234
add() (autotest.client.shared.profiler_manager.profiler_manager method), 291
add() (autotest.client.shared.utils.run_randomly method), 316
add_args() (autotest.client.tools.boottool.Grubby method), 353
add_br() (autotest.client.shared.openvswitch.OpenVSwitchControl method), 288
add_br() (autotest.client.shared.openvswitch.OpenVSwitchControlCli method), 288
add_child() (autotest.client.net.net_tc.tcclass method), 233
add_class() (autotest.client.net.net_tc.classful_qdisc method), 232
add_console_handlers() (autotest.client.shared.logging_config.LoggingConfig method), 279
add_debug_file_handlers() (autotest.client.client_logging_config.ClientLoggingConfig method), 201
add_debug_file_handlers() (autotest.client.shared.logging_config.LoggingConfig method), 279
add_fake_br() (autotest.client.shared.openvswitch.OpenVSwitchControlCli method), 288
add_file_handler() (autotest.client.shared.logging_config.LoggingConfig method), 279
add_file_handler() (autotest.client.shared.logging_config.TestingConfig method), 279
add_filter() (autotest.client.net.net_tc.classful_qdisc method), 232
add_global_option() (autotest.client.tools.boottool.EliloConf method), 359
add_kernel() (autotest.client.tools.boottool.Grubby method), 353
add_maddr() (autotest.client.net.net_utils.network_interface method), 236
add_param() (autotest.client.net.net_tc.netem method), 232
add_port() (autotest.client.shared.openvswitch.OpenVSwitchControl method), 288
add_port() (autotest.client.shared.openvswitch.OpenVSwitchControlCli method), 288
add_port_tag() (autotest.client.shared.openvswitch.OpenVSwitchControl method), 288
add_port_tag() (autotest.client.shared.openvswitch.OpenVSwitchControlCli method), 288
add_port_trunk() (autotest.client.shared.openvswitch.OpenVSwitchControl method), 288
add_port_trunk() (autotest.client.shared.openvswitch.OpenVSwitchControl method), 288
add_property() (autotest.client.tools.JUnit_api.propertiesType method), 348
add_repo() (autotest.client.shared.software_manager.AptBackend method), 295
add_repo() (autotest.client.shared.software_manager.YumBackend method), 297
add_repo() (autotest.client.shared.software_manager.ZypperBackend method), 298
add_repository() (autotest.client.job.base_client_job method), 210
add_repository() (autotest.client.shared.base_packages.BasePackageManagement method), 258
add_rule() (autotest.client.net.net_tc.u32filter method), 234
add_stream_handler() (autotest.client.shared.logging_config.LoggingConfig method), 279
add_stream_handler() (autotest.client.shared.logging_config.TestingConfig method), 279

- method), 280
- add_sysinfo_command() (autotest.client.job.base_client_job method), 210
- add_sysinfo_logfile() (autotest.client.job.base_client_job method), 210
- add_testcase() (autotest.client.tools.JUnit_api.testsuite method), 351
- add_testsuite() (autotest.client.tools.JUnit_api.testsuites method), 352
- add_to_bootloader() (autotest.client.kernel.BootableKernel method), 213
- add_to_bootloader() (autotest.client.kernel.rpm_kernel_suse method), 215
- add_to_bootloader() (autotest.client.xen.xen method), 231
- addError() (autotest.client.shared.test_utils.unittest.TestResult method), 340
- addExpectedFailure() (autotest.client.shared.test_utils.unittest.TestResult method), 340
- addFailure() (autotest.client.shared.test_utils.unittest.TestResult method), 340
- addSkip() (autotest.client.shared.test_utils.unittest.TestResult method), 340
- addSuccess() (autotest.client.shared.test_utils.unittest.TestResult method), 340
- addTest() (autotest.client.shared.test_utils.unittest.TestSuite method), 344
- addTests() (autotest.client.shared.test_utils.unittest.TestSuite method), 344
- addTypeEqualityFunc() (autotest.client.shared.test_utils.unittest.TestCase method), 341
- addUnexpectedSuccess() (autotest.client.shared.test_utils.unittest.TestResult method), 340
- after_run_once() (autotest.client.shared.test.base_test method), 300
- all() (in module autotest.client.shared.backports), 329
- all_cgroup_delete() (in module autotest.client.shared.utils_cgroup), 321
- all_drive_names() (in module autotest.client.cpuset), 202
- AllowBelowSeverity (class in autotest.client.shared.logging_config), 279
- analyze() (in module autotest.client.tools.regression), 361
- analyze_perf_constraints() (autotest.client.shared.test.base_test method), 300
- and_raises() (autotest.client.shared.test_utils.mock.function_mapping method), 341
- and_return() (autotest.client.shared.test_utils.mock.function_mapping method), 341
- method), 337
- any() (in module autotest.client.shared.backports), 329
- anything_comparator (class in autotest.client.shared.test_utils.mock), 337
- append_path() (in module autotest.client.base_utils), 194
- apply_overrides() (in module autotest.client.kernel_config), 216
- apply_patches() (autotest.client.kernel.kernel method), 214
- apply_patches() (autotest.client.kernel.srpm_kernel method), 215
- AptBackend (class in autotest.client.shared.software_manager), 295
- arch_probe() (autotest.client.tools.boottool.Grubby method), 354
- archive_as_tarball() (in module autotest.client.shared.utils), 306
- args_to_dict() (in module autotest.client.shared.utils), 306
- argument_comparator (class in autotest.client.shared.test_utils.mock), 337
- ask() (in module autotest.client.shared.utils), 307
- assert_() (autotest.client.shared.test.base_test method), 300
- assert_() (autotest.client.shared.test_utils.unittest.TestCase method), 343
- assert_any_call() (autotest.client.shared.mock.NonCallableMock method), 286
- assert_called_once_with() (autotest.client.shared.mock.NonCallableMock method), 286
- assert_called_with() (autotest.client.shared.mock.NonCallableMock method), 286
- assert_config_change() (in module autotest.client.shared.test_utils.config_change_validation), 336
- assert_config_change_dict() (in module autotest.client.shared.test_utils.config_change_validation), 336
- assert_has_calls() (autotest.client.shared.mock.NonCallableMock method), 286
- assertAlmostEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 341
- assertAlmostEquals() (autotest.client.shared.test_utils.unittest.TestCase method), 341
- assertDictContainsSubset() (autotest.client.shared.test_utils.unittest.TestCase method), 341
- assertDictEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 341
- assertEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 341

assertEquals() (autotest.client.shared.test_utils.unittest.TestCase method), 341

assertEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 343

assertFalse() (autotest.client.shared.test_utils.unittest.TestCase method), 341

assertTrue() (autotest.client.shared.test_utils.unittest.TestCase method), 343

assertGreater() (autotest.client.shared.test_utils.unittest.TestCase method), 341

assertTupleEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 343

assertGreaterEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 341

AsyncJob (class in autotest.client.shared.utils), 301

aton() (in module autotest.client.shared.utils), 307

assertIn() (autotest.client.shared.test_utils.unittest.TestCase method), 341

attach_mock() (autotest.client.shared.mock.NonCallableMock method), 286

assertIs() (autotest.client.shared.test_utils.unittest.TestCase method), 341

ATTR_BOOTSERVICE_ACCESS (autotest.client.tools.boottool.EfiVar attribute), 357

assertIsNone() (autotest.client.shared.test_utils.unittest.TestCase method), 341

ATTR_NON_VOLATILE (autotest.client.tools.boottool.EfiVar attribute), 357

assertIsNot() (autotest.client.shared.test_utils.unittest.TestCase method), 341

ATTR_RUNTIME_ACCESS (autotest.client.tools.boottool.EfiVar attribute), 357

assertIsNotNone() (autotest.client.shared.test_utils.unittest.TestCase method), 341

auto_kernel() (in module autotest.client.kernel), 213

assertLess() (autotest.client.shared.test_utils.unittest.TestCase method), 342

auto_obj (autotest.client.kernel.kernel attribute), 214

assertLessEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 342

autodir (autotest.client.shared.base_job.base_job attribute), 252

assertListEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 342

automatic_test_tag (autotest.client.shared.base_job.base_job attribute), 252

assertMultiLineEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 342

AutoservDiskFullHostError, 270

assertNotAlmostEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 342

AutoservError, 270

AutoservFetcher (class in autotest.client.harness_autoserv), 208

assertNotAlmostEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 342

AutoservHardwareHostError, 270

AutoservHardwareRepairRequestedError, 272

AutoservHardwareRepairRequiredError, 268

assertNotEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 342

AutoservHostError, 270

AutoservHostIsShuttingDownError, 268

assertNotEquals() (autotest.client.shared.test_utils.unittest.TestCase method), 342

AutoservInstallError, 271

AutoservNotMountedHostError, 272

assertNotIn() (autotest.client.shared.test_utils.unittest.TestCase method), 342

AutoservRebootError, 269

AutoservRunError, 271

assertRaises() (autotest.client.shared.test_utils.unittest.TestCase method), 342

AutoservShutdownError, 268

AutoservSshPermissionDeniedError, 272

assertRaisesRegexp() (autotest.client.shared.test_utils.unittest.TestCase method), 342

AutoservSshPingHostError, 272

AutoservSSTimeout, 270

assertRegexpMatches() (autotest.client.shared.test_utils.unittest.TestCase method), 342

AutoservSubcommandError, 269

AutoservUnsupportedError, 269

AutoservVirtError, 271

assertSameElements() (autotest.client.shared.test_utils.unittest.TestCase method), 343

autotest.client.autotest_local (module), 193

autotest.client.base_sysinfo (module), 193

autotest.client.base_utils (module), 194

autotest.client.bkr_proxy (module), 198

assertSequenceEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 343

autotest.client.bkr_xml (module), 200

autotest.client.client_logging_config (module), 201

autotest.client.cmdparser (module), 201

- autotest.client.common (module), 202
- autotest.client.config (module), 202
- autotest.client.cpuset (module), 202
- autotest.client.fsdev_disks (module), 204
- autotest.client.fsdev_mgr (module), 206
- autotest.client.fsinfo (module), 206
- autotest.client.harness (module), 207
- autotest.client.harness_autoserv (module), 208
- autotest.client.harness_beaker (module), 208
- autotest.client.harness_simple (module), 209
- autotest.client.harness_standalone (module), 210
- autotest.client.job (module), 210
- autotest.client.kernel (module), 213
- autotest.client.kernel_config (module), 216
- autotest.client.kernel_versions (module), 216
- autotest.client.kernelexpand (module), 217
- autotest.client.kvm_control (module), 217
- autotest.client.local_host (module), 218
- autotest.client.lv_utils (module), 218
- autotest.client.net.basic_machine (module), 231
- autotest.client.net.common (module), 231
- autotest.client.net.net_tc (module), 231
- autotest.client.net.net_utils (module), 234
- autotest.client.net.net_utils_mock (module), 238
- autotest.client.optparser (module), 219
- autotest.client.os_dep (module), 219
- autotest.client.parallel (module), 223
- autotest.client.partition (module), 223
- autotest.client.profiler (module), 228
- autotest.clientprofilers (module), 239
- autotest.clientprofilers.blktrace.blktrace (module), 239
- autotest.clientprofilers.catprofile.catprofile (module), 240
- autotest.clientprofilers.cmdprofile.cmdprofile (module), 240
- autotest.clientprofilers.cpiostat.cpiostat (module), 241
- autotest.clientprofilers.ftrace.ftrace (module), 241
- autotest.clientprofilers.inotify.inotify (module), 242
- autotest.clientprofilers.iostat.iostat (module), 242
- autotest.clientprofilers.kvm_stat.kvm_stat (module), 243
- autotest.clientprofilers.lockmeter.lockmeter (module), 243
- autotest.clientprofilers.lttng.lttng (module), 244
- autotest.clientprofilers.mpstat.mpstat (module), 244
- autotest.clientprofilers.oprofile.oprofile (module), 245
- autotest.clientprofilers.perf.perf (module), 245
- autotest.clientprofilers.powertop.powertop (module), 245
- autotest.clientprofilers.readprofile.readprofile (module), 246
- autotest.clientprofilers.sar.sar (module), 246
- autotest.clientprofilers.systemtap.systemtap (module), 247
- autotest.clientprofilers.vmstat.vmstat (module), 247
- autotest.client.setup (module), 228
- autotest.client.setup_job (module), 228
- autotest.client.setup_modules (module), 229
- autotest.client.shared.autotemp (module), 248
- autotest.client.shared.backports (module), 329
- autotest.client.shared.backports.collections (module), 329
- autotest.client.shared.backports.collections.defaultdict (module), 330
- autotest.client.shared.backports.collections.namedtuple (module), 331
- autotest.client.shared.backports.collections.OrderedDict (module), 329
- autotest.client.shared.barrier (module), 248
- autotest.client.shared.base_barrier (module), 248
- autotest.client.shared.base_check_version (module), 250
- autotest.client.shared.base_job (module), 250
- autotest.client.shared.base_packages (module), 258
- autotest.client.shared.base_syncdata (module), 263
- autotest.client.shared.boottool (module), 264
- autotest.client.shared.check_version (module), 264
- autotest.client.shared.common (module), 264
- autotest.client.shared.control_data (module), 264
- autotest.client.shared.distro (module), 33, 265
- autotest.client.shared.distro_def (module), 266
- autotest.client.shared.enum (module), 267
- autotest.client.shared.error (module), 268
- autotest.client.shared.git (module), 272
- autotest.client.shared.host_protections (module), 274
- autotest.client.shared.host_queue_entry_states (module), 274
- autotest.client.shared.hosts (module), 332
- autotest.client.shared.hosts.base_classes (module), 332
- autotest.client.shared.hosts.common (module), 336
- autotest.client.shared.iscsi (module), 274
- autotest.client.shared.iso9660 (module), 275
- autotest.client.shared.jsontemplate (module), 276
- autotest.client.shared.kernel_versions (module), 278
- autotest.client.shared.log (module), 279
- autotest.client.shared.logging_config (module), 279
- autotest.client.shared.logging_manager (module), 280
- autotest.client.shared.magic (module), 282
- autotest.client.shared.mail (module), 282
- autotest.client.shared.mock (module), 283
- autotest.client.shared.openvswitch (module), 287
- autotest.client.shared.packages (module), 290
- autotest.client.shared.pidfile (module), 290
- autotest.client.shared.profiler_manager (module), 291
- autotest.client.shared.progressbar (module), 291
- autotest.client.shared.report (module), 292
- autotest.client.shared.service (module), 293
- autotest.client.shared.settings (module), 294
- autotest.client.shared.software_manager (module), 295
- autotest.client.shared.syncdata (module), 299
- autotest.client.shared.test (module), 299

- autotest.client.shared.test_utils.config_change_validation (module), 336
- autotest.client.shared.test_utils.functools_24 (module), 336
- autotest.client.shared.test_utils.mock (module), 337
- autotest.client.shared.test_utils.unittest (module), 339
- autotest.client.shared.utils (module), 301
- autotest.client.shared.utils_cgroup (module), 319
- autotest.client.shared.utils_koji (module), 322
- autotest.client.shared.utils_memory (module), 327
- autotest.client.shared.version (module), 328
- autotest.client.sysinfo (module), 229
- autotest.client.test (module), 229
- autotest.client.test_config (module), 230
- autotest.client.tools.boottool (module), 353
- autotest.client.tools.common (module), 360
- autotest.client.tools.crash_handler (module), 360
- autotest.client.tools.JUnit_api (module), 346
- autotest.client.tools.process_metrics (module), 361
- autotest.client.tools.regression (module), 361
- autotest.client.tools.results2junit (module), 362
- autotest.client.tools.scan_results (module), 362
- autotest.client.utils (module), 230
- autotest.client.xen (module), 231
- autotest.frontend.afe.model_logic (module), 95
- autotest.frontend.afe.models (module), 95
- autotest.frontend.tko.models (module), 96
- AutotestError, 269
- AutotestHostRunError, 269
- AutotestLocalApp (class in autotest.client.autotest_local), 193
- AutotestLocalOptionParser (class in autotest.client.optparser), 219
- AutotestRunError, 271
- AutotestTimeoutError, 272
- avail_mbytes() (in module autotest.client.cpuset), 202
- available_exclusive_mem_nodes() (in module autotest.client.cpuset), 203
- avgttime_print() (in module autotest.client.base_utils), 194
- ## B
- BAD_CHAR_REGEX (autotest.client.shared.base_job.status_log_entry attribute), 256
- BadFormatter, 276
- BadPredicate, 276
- barrier (class in autotest.client.shared.base_barrier), 248
- barrier() (autotest.client.job.base_client_job method), 210
- BarrierAbortError, 248, 269
- BarrierError, 271
- base_check_python_version (class in autotest.client.shared.base_check_version), 250
- base_client_job (class in autotest.client.job), 210
- base_job (class in autotest.client.shared.base_job), 251
- base_mapping (class in autotest.client.shared.test_utils.mock), 337
- BASE_PATH (autotest.client.tools.boottool.EfiToolSys attribute), 358
- base_sysinfo (class in autotest.client.base_sysinfo), 193
- base_test (class in autotest.client.shared.test), 300
- BaseBackend (class in autotest.client.shared.software_manager), 296
- BaseFsdevManager (class in autotest.client.fsdev_mgr), 206
- BasePackageManager (class in autotest.client.shared.base_packages), 258
- BeakerXMLParser (class in autotest.client.bkr_xml), 200
- before_run_once() (autotest.client.shared.test.base_test method), 300
- before_start() (autotest.client.shared.profiler_manager.profiler_manager method), 291
- BgJob (class in autotest.client.shared.utils), 302
- bin() (in module autotest.client.shared.backports), 329
- bind() (autotest.client.net.net_utils_mock.socket_stub method), 239
- bindir (autotest.client.shared.base_job.base_job attribute), 252
- binrpm_pattern (autotest.client.kernel.srpm_kernel attribute), 215
- bitlist_to_string() (in module autotest.client.shared.utils), 307
- BkrProxy (class in autotest.client.bkr_proxy), 198
- BkrProxyException, 198
- blktrace (class in autotest.clientprofilers.blktrace.blktrace), 239
- bond() (in module autotest.client.net.net_utils), 234
- bonding (class in autotest.client.net.net_utils), 234
- boot() (autotest.client.kernel.kernel method), 214
- boot() (autotest.client.kernel.rpm_kernel method), 215
- boot() (autotest.client.kernel.srpm_kernel method), 215
- boot_once() (autotest.client.tools.boottool.Grubby method), 354
- boot_once_elilo() (autotest.client.tools.boottool.Grubby method), 354
- boot_once_grub() (autotest.client.tools.boottool.Grubby method), 354
- boot_once_grub2() (autotest.client.tools.boottool.Grubby method), 354
- boot_once_yaboot() (autotest.client.tools.boottool.Grubby method), 354
- BootableKernel (class in autotest.client.kernel), 213
- bootloader_probe() (autotest.client.tools.boottool.Grubby method), 354
- bootstrap() (autotest.client.cmdparser.CommandParser method), 202

- bootstrap() (autotest.client.harness_beaker.harness_beaker method), 208
 - boottool (class in autotest.client.shared.boottool), 264
 - br_exist() (autotest.client.shared.openvswitch.OpenVSwitchController method), 288
 - br_exist() (autotest.client.shared.openvswitch.OpenVSwitchController method), 288
 - build() (autotest.client.kernel.kernel method), 214
 - build() (autotest.client.kernel.rpm_kernel method), 215
 - build() (autotest.client.kernel.srpm_kernel method), 215
 - build() (autotest.client.tools.JUnit_api.errorType method), 346
 - build() (autotest.client.tools.JUnit_api.failureType method), 347
 - build() (autotest.client.tools.JUnit_api.propertiesType method), 348
 - build() (autotest.client.tools.JUnit_api.propertyType method), 348
 - build() (autotest.client.tools.JUnit_api.system_err method), 349
 - build() (autotest.client.tools.JUnit_api.system_out method), 349
 - build() (autotest.client.tools.JUnit_api.testcaseType method), 350
 - build() (autotest.client.tools.JUnit_api.testsuite method), 351
 - build() (autotest.client.tools.JUnit_api.testsuites method), 352
 - build() (autotest.client.tools.JUnit_api.testsuiteType method), 352
 - build() (autotest.client.xen.xen method), 231
 - build_timed() (autotest.client.kernel.kernel method), 214
 - build_timed() (autotest.client.xen.xen method), 231
 - buildAttributes() (autotest.client.tools.JUnit_api.errorType method), 346
 - buildAttributes() (autotest.client.tools.JUnit_api.failureType method), 347
 - buildAttributes() (autotest.client.tools.JUnit_api.propertiesType method), 348
 - buildAttributes() (autotest.client.tools.JUnit_api.propertyType method), 348
 - buildAttributes() (autotest.client.tools.JUnit_api.system_err method), 349
 - buildAttributes() (autotest.client.tools.JUnit_api.system_out method), 349
 - buildAttributes() (autotest.client.tools.JUnit_api.testcaseType method), 350
 - buildAttributes() (autotest.client.tools.JUnit_api.testsuite method), 351
 - buildAttributes() (autotest.client.tools.JUnit_api.testsuites method), 352
 - buildAttributes() (autotest.client.tools.JUnit_api.testsuiteType method), 352
 - buildChildren() (autotest.client.tools.JUnit_api.errorType method), 347
 - buildChildren() (autotest.client.tools.JUnit_api.failureType method), 347
 - buildChildren() (autotest.client.tools.JUnit_api.propertiesType method), 348
 - buildChildren() (autotest.client.tools.JUnit_api.propertyType method), 348
 - buildChildren() (autotest.client.tools.JUnit_api.system_err method), 349
 - buildChildren() (autotest.client.tools.JUnit_api.system_out method), 349
 - buildChildren() (autotest.client.tools.JUnit_api.testcaseType method), 350
 - buildChildren() (autotest.client.tools.JUnit_api.testsuite method), 351
 - buildChildren() (autotest.client.tools.JUnit_api.testsuites method), 353
 - buildChildren() (autotest.client.tools.JUnit_api.testsuiteType method), 352
- C**
- call (in module autotest.client.shared.mock), 285
 - call_args (autotest.client.shared.mock.NonCallableMock attribute), 286
 - call_args_list (autotest.client.shared.mock.NonCallableMock attribute), 286
 - call_count (autotest.client.shared.mock.NonCallableMock attribute), 286
 - called (autotest.client.shared.mock.NonCallableMock attribute), 286
 - cat_file_to_cmd() (in module autotest.client.base_utils), 194
 - catprofile (class in autotest.clientprofilers.catprofile.catprofile), 240
 - cgclassify_cgroup() (autotest.client.shared.utils_cgroup.Cgroup method), 319
 - cgconfig_condrestart() (in module autotest.client.shared.utils_cgroup), 321
 - cgconfig_exists() (in module autotest.client.shared.utils_cgroup), 321
 - cgconfig_is_running() (in module autotest.client.shared.utils_cgroup), 321
 - cgconfig_restart() (in module autotest.client.shared.utils_cgroup), 321
 - cgconfig_start() (in module autotest.client.shared.utils_cgroup), 321
 - cgconfig_stop() (in module autotest.client.shared.utils_cgroup), 321
 - cgdelete_all_cgroups() (autotest.client.shared.utils_cgroup.Cgroup method), 319

- cgdelete_cgroup() (autotest.client.shared.utils_cgroup.Cgroup method), 319
- cgexec() (autotest.client.shared.utils_cgroup.Cgroup method), 319
- Cgroup (class in autotest.client.shared.utils_cgroup), 319
- CgroupModules (class in autotest.client.shared.utils_cgroup), 321
- cgset_property() (autotest.client.shared.utils_cgroup.Cgroup method), 319
- check() (autotest.client.shared.openvswitch.OpenVSwitchSystem method), 289
- check() (autotest.client.test_config.config_loader method), 230
- check_basic_structure() (autotest.client.tools.boottool.EfiToolSys method), 358
- check_db_daemon() (autotest.client.shared.openvswitch.OpenVSwitchSystem method), 289
- check_db_file() (autotest.client.shared.openvswitch.OpenVSwitchSystem method), 289
- check_db_socket() (autotest.client.shared.openvswitch.OpenVSwitchSystem method), 289
- check_diskspace() (autotest.client.shared.hosts.base_classes.Host method), 332
- check_diskspace() (in module autotest.client.shared.base_packages), 262
- CHECK_FILE (autotest.client.shared.distro.Probe attribute), 35, 265
- CHECK_FILE_CONTAINS (autotest.client.shared.distro.Probe attribute), 35, 265
- CHECK_FILE_DISTRO_NAME (autotest.client.shared.distro.Probe attribute), 35, 265
- check_for_kernel_feature() (in module autotest.client.base_utils), 194
- check_glibc_ver() (in module autotest.client.base_utils), 194
- check_installed() (autotest.client.shared.software_manager.DpkgBackend method), 296
- check_installed() (autotest.client.shared.software_manager.RpmsBackend method), 296
- check_kernel_ver() (in module autotest.client.base_utils), 194
- check_mount_point() (autotest.client.fsdev_mgr.BaseFsdevManager method), 206
- check_name_for_file() (autotest.client.shared.distro.Probe method), 35, 265
- check_name_for_file_contains() (autotest.client.shared.distro.Probe method), 35, 265
- check_parameter() (autotest.client.test_config.config_loader method), 230
- check_partitions() (autotest.client.shared.hosts.base_classes.Host method), 332
- check_playback() (autotest.client.shared.test_utils.mock.mock_god method), 338
- check_port_in_br() (autotest.client.shared.openvswitch.OpenVSwitchControl method), 288
- check_python_version (class in autotest.client.shared.check_version), 264
- check_release() (autotest.client.shared.distro.Probe method), 35, 265
- check_repair_versions() (autotest.client.shared.utils.VersionableClass class method), 306
- check_stand_alone_client_run() (autotest.client.shared.settings.Settings method), 294
- check_switch_daemon() (autotest.client.shared.openvswitch.OpenVSwitchSystem method), 289
- check_values() (autotest.client.tools.boottool.OptionParser method), 357
- check_version() (autotest.client.shared.distro.Probe method), 35, 266
- CHECK_VERSION_REGEX (autotest.client.shared.distro.Probe attribute), 34, 35, 265
- check_write() (in module autotest.client.shared.base_packages), 262
- checkout() (autotest.client.shared.git.GitRepoHelper method), 273
- CheckPlaybackError, 337
- CHECKSUM_LEN (autotest.client.net.net_utils.ethernet attribute), 235
- choices() (autotest.client.shared.enum.Enum method), 268
- ClassBackend (autotest.client.net.net_tc.classful_qdisc attribute), 232
- RpmsBackend (autotest.client.net.net_tc.classless_qdisc attribute), 232
- classful_qdisc (class in autotest.client.net.net_tc), 232
- classless_qdisc (class in autotest.client.net.net_tc), 232
- classSuiteClass (autotest.client.shared.test_utils.unittest.TestLoader attribute), 345
- ClassTestSuite (class in autotest.client.shared.test_utils.unittest), 344
- clean() (autotest.client.kernel.kernel method), 214
- clean() (autotest.client.shared.autotemp.tempdir method), 248
- clean() (autotest.client.shared.autotemp.tempfile method), 248

- method), 248
- clean() (autotest.client.shared.base_syncdata.TempDir method), 263
- clean() (autotest.client.shared.openvswitch.OpenVSwitch method), 287
- clean() (autotest.client.shared.openvswitch.OpenVSwitchSystemd method), 289
- clean() (autotest.client.shared.test.Subtest method), 299
- cleanup() (autotest.client.shared.hosts.base_classes.Host method), 333
- cleanup() (autotest.client.shared.iscsi.Iscsi method), 274
- cleanup() (autotest.client.shared.test.base_test method), 300
- cleanup() (autotest.client.shared.utils.AsyncJob method), 301
- cleanup() (autotest.client.shared.utils.BgJob method), 302
- cleanup_kernels() (autotest.client.shared.hosts.base_classes.Host method), 333
- clear() (autotest.client.shared.backports.collections.OrderedDict method), 329
- clientdir (autotest.client.shared.base_job.base_job attribute), 252
- ClientLoggingConfig (class in autotest.client.client_logging_config), 201
- close() (autotest.client.net.net_utils.raw_socket method), 237
- close() (autotest.client.net.net_utils_mock.socket_stub method), 239
- close() (autotest.client.shared.base_barrier.listen_server method), 250
- close() (autotest.client.shared.base_syncdata.SessionData method), 263
- close() (autotest.client.shared.base_syncdata.SyncData method), 263
- close() (autotest.client.shared.base_syncdata.SyncListenServer method), 263
- close() (autotest.client.shared.hosts.base_classes.Host method), 333
- close() (autotest.client.shared.iso9660.Iso9660IsoRead method), 275
- close() (autotest.client.shared.iso9660.Iso9660Mount method), 275
- close() (autotest.client.shared.test_utils.mock.SaveDataAfterCloseStringIO method), 337
- close_file() (autotest.client.shared.pidfile.PidFileManager method), 290
- close_log_file() (in module autotest.client.shared.utils), 307
- CMD_LOOKUP_ORDER (autotest.client.shared.utils_koji.KojiClient attribute), 322
- CmdError, 269
- CmdParserLoggingConfig (class in autotest.client.cmdparser), 201
- cmdprofile (class in autotest.clientprofilers.cmdprofile.cmdprofile), 240
- CmdResult (class in autotest.client.shared.utils), 302
- Command (class in autotest.client.base_sysinfo), 193
- command() (in module autotest.client.os_dep), 220
- COMMAND_LIST (autotest.client.cmdparser.CommandParser attribute), 201
- CommandParser (class in autotest.client.cmdparser), 201
- commands() (in module autotest.client.os_dep), 220
- compare() (autotest.client.shared.magic.MagicTest method), 282
- compare_checksum() (autotest.client.shared.base_packages.BasePackageManager method), 258
- compare_features() (in module autotest.client.fsinfo), 206
- compare_versions() (in module autotest.client.shared.utils), 307
- CompilationError, 276
- CompileTemplate() (in module autotest.client.shared.jsontemplate), 277
- complete() (autotest.client.job.base_client_job method), 210
- compose() (in module autotest.client.shared.test_utils.functools_24), 336
- compute_checksum() (autotest.client.shared.base_packages.BasePackageManager method), 258
- conf_command (autotest.client.net.net_tc.tcfilter attribute), 233
- conf_device (autotest.client.net.net_tc.tcfilter attribute), 233
- conf_flowid (autotest.client.net.net_tc.tcfilter attribute), 233
- conf_name (autotest.client.net.net_tc.tcfilter attribute), 233
- conf_params (autotest.client.net.net_tc.tcfilter attribute), 233
- conf_parent (autotest.client.net.net_tc.tcfilter attribute), 233
- conf_stringid (autotest.client.net.net_tc.tcfilter attribute), 233
- conf_protocol (autotest.client.net.net_tc.tcfilter attribute), 233
- conf_qdiscid (autotest.client.net.net_tc.tcfilter attribute), 233
- conf_rules (autotest.client.net.net_tc.tcfilter attribute), 233
- conf_type (autotest.client.net.net_tc.tcfilter attribute), 233
- config (autotest.client.shared.settings.Settings attribute), 294

- [config \(class in autotest.client.config\)](#), 202
[config\(\) \(autotest.client.kernel.kernel method\)](#), 214
[config\(\) \(autotest.client.kernel.srpm_kernel method\)](#), 215
[config\(\) \(autotest.client.xen.xen method\)](#), 231
[config_by_name\(\) \(in module autotest.client.kernel_config\)](#), 216
[config_file \(autotest.client.shared.settings.Settings attribute\)](#), 294
[config_get\(\) \(autotest.client.job.base_client_job method\)](#), 210
[config_loader \(class in autotest.client.test_config\)](#), 230
[CONFIG_MAP \(autotest.client.shared.utils_koji.KojiClient attribute\)](#), 322
[config_record\(\) \(autotest.client.kernel_config.kernel_config method\)](#), 216
[config_sched_tunables\(\) \(autotest.client.fsdev_disks.fsdev_disks method\)](#), 204
[config_set\(\) \(autotest.client.job.base_client_job method\)](#), 210
[configdir \(autotest.client.shared.base_job.base_job attribute\)](#), 252
[ConfigurationError](#), 276
[configure\(\) \(in module autotest.client.shared.utils\)](#), 307
[configure_crash_handler\(\) \(autotest.client.shared.test.base_test method\)](#), 300
[configure_crash_handler\(\) \(autotest.client.test.test method\)](#), 229
[configure_logging\(\) \(autotest.client.client_logging_config.ClientLoggingConfig method\)](#), 201
[configure_logging\(\) \(autotest.client.cmdparser.CmdParserLoggingConfig method\)](#), 201
[configure_logging\(\) \(autotest.client.shared.logging_config.LoggingConfig method\)](#), 279
[configure_logging\(\) \(autotest.client.shared.logging_config.TestingConfig method\)](#), 280
[configure_logging\(\) \(autotest.client.shared.magic.MagicLoggingConfig method\)](#), 282
[configure_logging\(\) \(autotest.client.shared.report.ReportLoggingConfig method\)](#), 292
[configure_logging\(\) \(autotest.client.shared.software_manager.SoftwareManagerLoggingConfig method\)](#), 297
[configure_logging\(\) \(in module autotest.client.shared.logging_manager\)](#), 281
[configure_mock\(\) \(autotest.client.shared.mock.NonCallableMock method\)](#), 286
[conmuxdir \(autotest.client.shared.base_job.base_job attribute\)](#), 252
[console_formatter \(autotest.client.shared.logging_config.LoggingConfig attribute\)](#), 279
[consume_one_config\(\) \(autotest.client.kernel.srpm_kernel method\)](#), 215
[container_bytes\(\) \(in module autotest.client.cpuset\)](#), 203
[container_exists\(\) \(in module autotest.client.cpuset\)](#), 203
[container_mbytes\(\) \(in module autotest.client.cpuset\)](#), 203
[context\(\) \(in module autotest.client.shared.error\)](#), 268
[context_aware\(\) \(in module autotest.client.shared.error\)](#), 268
[control_get\(\) \(autotest.client.job.base_client_job method\)](#), 210
[control_set\(\) \(autotest.client.job.base_client_job method\)](#), 211
[ControlData \(class in autotest.client.shared.control_data\)](#), 264
[ControlVariableException](#), 265
[convert_conf_opt\(\) \(in module autotest.client.fsinfo\)](#), 206
[convert_data_size\(\) \(in module autotest.client.shared.utils\)](#), 307
[convert_ipv4_to_ipv6\(\) \(in module autotest.client.shared.utils\)](#), 308
[convert_systemd_target_to_runlevel\(\) \(in module autotest.client.shared.service\)](#), 293
[convert_sysv_runlevel\(\) \(in module autotest.client.shared.service\)](#), 293
[convert_task_to_control\(\) \(autotest.client.harness_beaker.harness_beaker method\)](#), 208
[convert_version_to_int\(\) \(autotest.client.shared.openvswitch.OpenVSwitchControl static method\)](#), 288
[copy\(\) \(autotest.client.shared.backports.collections.defaultdict.defaultdict method\)](#), 330
[copy\(\) \(autotest.client.shared.backports.collections.OrderedDict.OrderedDict method\)](#), 330
[copy\(\) \(autotest.client.shared.iso9660.Iso9660IsoRead method\)](#), 275
[copy\(\) \(autotest.client.shared.iso9660.Iso9660Mount method\)](#), 275
[copy_data\(\) \(in module autotest.client.bkr_proxy\)](#), 198
[copy_local\(\) \(in module autotest.client.bkr_proxy\)](#), 199
[copy_remote\(\) \(in module autotest.client.bkr_proxy\)](#), 199
[count_cpus\(\) \(in module autotest.client.base_utils\)](#), 194
[count_cpus\(\) \(in module autotest.client.base_utils\)](#), 194
[countTestCases\(\) \(autotest.client.shared.test_utils.unittest.TestCase method\)](#), 343
[MockTestCases\(\) \(autotest.client.shared.test_utils.unittest.TestSuite method\)](#), 344

- cpistat (class in autotest.clientprofilers.cpistat.cpistat), 241
 - cpu_affinity_by_task() (in module autotest.client.shared.utils), 308
 - cpu_count() (autotest.client.job.base_client_job method), 211
 - cpu_has_flags() (in module autotest.client.base_utils), 194
 - cpu_online_map() (in module autotest.client.base_utils), 194
 - cpus_path() (in module autotest.client.cpuset), 203
 - cpuset_attr() (in module autotest.client.cpuset), 203
 - crash_handler_report() (autotest.client.shared.test.base_test method), 300
 - crash_handler_report() (autotest.client.test.test method), 230
 - create_autospec() (in module autotest.client.shared.mock), 285
 - create_container_directly() (in module autotest.client.cpuset), 203
 - create_container_via_memcg() (in module autotest.client.cpuset), 203
 - create_container_with_mbytes_and_specific_cpus() (in module autotest.client.cpuset), 203
 - create_container_with_specific_mems_cpus() (in module autotest.client.cpuset), 203
 - create_directory() (in module autotest.client.shared.base_packages), 262
 - create_mock_class() (autotest.client.shared.test_utils.mock.mock_god method), 338
 - create_mock_class_obj() (autotest.client.shared.test_utils.mock.mock_god method), 338
 - create_mock_function() (autotest.client.shared.test_utils.mock.mock_god method), 338
 - create_subnet_mask() (in module autotest.client.shared.utils), 308
 - create_variable() (autotest.client.tools.boottool.EfiToolSys method), 358
 - create_x509_dir() (in module autotest.client.shared.utils), 308
 - current_profilers() (autotest.client.shared.profiler_manager.profiler_manager method), 291
 - customtestdir (autotest.client.shared.base_job.base_job attribute), 252
- ## D
- DataSyncError, 270
 - dbg() (in module autotest.client.tools.results2junit), 362
 - debug() (autotest.client.shared.test_utils.unittest.TestCase method), 343
 - debug() (autotest.client.shared.test_utils.unittest.TestSuite method), 344
 - decompose_kernel() (in module autotest.client.kernelexpand), 217
 - decompose_kernel_2x_once() (in module autotest.client.kernelexpand), 217
 - decompose_kernel_post_2x_once() (in module autotest.client.kernelexpand), 217
 - decored() (autotest.client.shared.test.Subtest method), 299
 - decrement() (autotest.client.job.status_indenter method), 213
 - decrement() (autotest.client.shared.base_job.status_indenter method), 256
 - default() (autotest.client.tools.boottool.Grubby method), 354
 - DEFAULT_ATTRIBUTES (autotest.client.tools.boottool.EfiVar attribute), 358
 - default_profile_only (autotest.client.shared.base_job.base_job attribute), 252
 - DEFAULT_REBOOT_TIMEOUT (autotest.client.shared.hosts.base_classes.Host attribute), 332
 - DEFAULT_WIDTH (autotest.client.shared.progressbar.ProgressBar attribute), 291
 - defaultdict (class in autotest.client.shared.backports.collections.defaultdict), 330
 - defaultTestResult() (autotest.client.shared.test_utils.unittest.TestCase method), 343
 - del_br() (autotest.client.shared.openvswitch.OpenVSwitchControl method), 288
 - del_br() (autotest.client.shared.openvswitch.OpenVSwitchControlCli_140 method), 288
 - del_maddr() (autotest.client.net.net_utils.network_interface method), 236
 - del_port() (autotest.client.shared.openvswitch.OpenVSwitchControl method), 288
 - del_port() (autotest.client.shared.openvswitch.OpenVSwitchControlCli_140 method), 288
 - delete_files_copies() (in module autotest.client.shared.test_utils.config_change_validation), 336
 - DEL_VAR (autotest.client.tools.boottool.EfiToolSys attribute), 358
 - delete() (autotest.client.shared.profiler_manager.profiler_manager method), 291
 - delete_leftover_test_containers() (in module autotest.client.cpuset), 203
 - delete_pid_file_if_exists() (in module au-

- totest.client.shared.utils), 308
 - delete_target() (autotest.client.shared.iscsi.Iscsi method), 274
 - delete_variable() (autotest.client.tools.boottool.EfiToolSys method), 358
 - deprecated() (in module autotest.client.shared.utils), 308
 - describe() (autotest.client.shared.utils_koji.KojiPkgSpec method), 325
 - describe_invalid() (autotest.client.shared.utils_koji.KojiPkgSpec method), 325
 - deserialize() (autotest.client.base_sysinfo.base_sysinfo method), 193
 - destroy() (autotest.client.partition.virtual_partition method), 227
 - detect() (in module autotest.client.shared.distro), 36, 266
 - diff_configs() (in module autotest.client.kernel_config), 216
 - difflist() (in module autotest.client.base_utils), 194
 - DISABLE (autotest.client.net.net_utils.network_interface attribute), 236
 - disable() (autotest.client.net.net_utils.bonding method), 234
 - disable_external_logging() (autotest.client.job.base_client_job method), 211
 - disable_ip_local_loopback() (autotest.client.net.net_utils.network_utils method), 237
 - disable_ipfilters() (autotest.client.shared.hosts.base_classes.Host method), 333
 - disable_loopback() (autotest.client.net.net_utils.network_interface method), 236
 - disable_promisc() (autotest.client.net.net_utils.network_interface method), 236
 - disable_warnings() (autotest.client.job.base_client_job method), 211
 - discard() (autotest.client.shared.base_job.job_state method), 254
 - discard_namespace() (autotest.client.shared.base_job.job_state method), 255
 - discover_container_style() (in module autotest.client.cpuset), 203
 - disk_block_size() (in module autotest.client.base_utils), 194
 - disk_usage_monitor (class in autotest.client.job), 213
 - display() (in module autotest.client.tools.regression), 361
 - display_data_size() (in module autotest.client.shared.utils), 308
 - DISTRO_PKG_INFO_LOADERS (in module autotest.client.shared.distro_def), 267
 - DistroDef (class in autotest.client.shared.distro_def), 267
 - do_not_report_as_logging_caller() (in module autotest.client.shared.logging_manager), 281
 - down() (autotest.client.net.net_utils.network_interface method), 236
 - DpkgBackend (class in autotest.client.shared.software_manager), 296
 - drop_caches() (in module autotest.client.shared.utils_memory), 327
 - drop_caches_between_iterations() (autotest.client.shared.test.base_test method), 300
 - dump() (autotest.client.shared.utils.SystemLoad method), 303
 - dump() (in module autotest.client.tools.results2junit), 362
 - dump_object() (in module autotest.client.base_utils), 195
- ## E
- EfiToolSys (class in autotest.client.tools.boottool), 358
 - EfiVar (class in autotest.client.tools.boottool), 357
 - EliloConf (class in autotest.client.tools.boottool), 358
 - EmailNotificationManager (class in autotest.client.shared.mail), 282
 - ENABLE (autotest.client.net.net_utils.network_interface attribute), 236
 - enable() (autotest.client.net.net_utils.bonding method), 234
 - enable_external_logging() (autotest.client.job.base_client_job method), 211
 - enable_ip_local_loopback() (autotest.client.net.net_utils.network_utils method), 237
 - enable_ipfilters() (autotest.client.shared.hosts.base_classes.Host method), 333
 - enable_loopback() (autotest.client.net.net_utils.network_interface method), 236
 - enable_promisc() (autotest.client.net.net_utils.network_interface method), 236
 - enable_warnings() (autotest.client.job.base_client_job method), 211
 - end_reboot() (autotest.client.job.base_client_job method), 211
 - end_reboot_and_verify() (autotest.client.job.base_client_job method), 211
 - enqueue_admin() (autotest.client.shared.mail.EmailNotificationManager method), 282
 - enqueue_exception_admin() (autotest.client.shared.mail.EmailNotificationManager method), 282
 - Enum (class in autotest.client.shared.enum), 267
 - environ() (in module autotest.client.base_utils), 195
 - equality_comparator (class in autotest.client.shared.test_utils.mock), 337

- erase_dir_contents() (autotest.client.shared.hosts.base_classes.Host method), 333
- Error, 276
- errorType (class in autotest.client.tools.JUnit_api), 346
- ETH_LLDP_DST_MAC (autotest.client.net.net_utils.ethernet attribute), 235
- ETH_P_ALL (autotest.client.net.net_utils.raw_socket attribute), 237
- ETH_PACKET_MAX_SIZE (autotest.client.net.net_utils.ethernet attribute), 235
- ETH_PACKET_MIN_SIZE (autotest.client.net.net_utils.ethernet attribute), 235
- ETH_TYPE_8021Q (autotest.client.net.net_utils.ethernet attribute), 235
- ETH_TYPE_ARP (autotest.client.net.net_utils.ethernet attribute), 235
- ETH_TYPE_CDP (autotest.client.net.net_utils.ethernet attribute), 235
- ETH_TYPE_IP (autotest.client.net.net_utils.ethernet attribute), 235
- ETH_TYPE_IP6 (autotest.client.net.net_utils.ethernet attribute), 235
- ETH_TYPE_LLDP (autotest.client.net.net_utils.ethernet attribute), 235
- ETH_TYPE_LOOPBACK (autotest.client.net.net_utils.ethernet attribute), 235
- ethernet (class in autotest.client.net.net_utils), 234
- ethernet_packet() (in module autotest.client.net.net_utils), 236
- etraceback() (in module autotest.client.shared.utils), 308
- EvaluationError, 276
- exception_context() (in module autotest.client.shared.error), 268
- exception_when_false_wrapper() (in module autotest.client.os_dep), 220
- exec_sql() (in module autotest.client.tools.regression), 362
- execute() (autotest.client.shared.git.GitRepoHelper method), 273
- execute() (autotest.client.shared.test.base_test method), 300
- exists() (autotest.client.net.net_utils.network_interface method), 236
- exit_status (autotest.client.shared.error.TestBaseException attribute), 270
- exit_status (autotest.client.shared.error.TestError attribute), 271
- exit_status (autotest.client.shared.error.TestFail attribute), 271
- exit_status (autotest.client.shared.error.TestNAError attribute), 270
- exit_status (autotest.client.shared.error.TestWarn attribute), 269
- expand() (autotest.client.shared.jsontemplate.Template method), 278
- expand() (in module autotest.client.shared.jsontemplate), 278
- expand_classic() (in module autotest.client.kernelexpand), 217
- expect_any_call() (autotest.client.shared.test_utils.mock.mock_function method), 338
- expect_call() (autotest.client.shared.test_utils.mock.mock_function method), 338
- expectedFailure() (in module autotest.client.shared.test_utils.unittest), 346
- export() (autotest.client.tools.JUnit_api.errorType method), 347
- export() (autotest.client.tools.JUnit_api.failureType method), 347
- export() (autotest.client.tools.JUnit_api.propertiesType method), 348
- export() (autotest.client.tools.JUnit_api.propertyType method), 348
- export() (autotest.client.tools.JUnit_api.system_err method), 349
- export() (autotest.client.tools.JUnit_api.system_out method), 349
- export() (autotest.client.tools.JUnit_api.testcaseType method), 350
- export() (autotest.client.tools.JUnit_api.testsuite method), 351
- export() (autotest.client.tools.JUnit_api.testsuites method), 353
- export() (autotest.client.tools.JUnit_api.testsuiteType method), 352
- export_target() (autotest.client.shared.iscsi.Iscsi method), 274
- exportAttributes() (autotest.client.tools.JUnit_api.errorType method), 347
- exportAttributes() (autotest.client.tools.JUnit_api.failureType method), 347
- exportAttributes() (autotest.client.tools.JUnit_api.propertiesType method), 348
- exportAttributes() (autotest.client.tools.JUnit_api.propertyType method), 348
- exportAttributes() (autotest.client.tools.JUnit_api.system_err method), 349
- exportAttributes() (autotest.client.tools.JUnit_api.system_out method), 349
- exportAttributes() (autotest.client.tools.JUnit_api.testcaseType method), 350
- exportAttributes() (autotest.client.tools.JUnit_api.testsuite method), 351

exportAttributes() (autotest.client.tools.JUnit_api.testsuites method), 353

exportAttributes() (autotest.client.tools.JUnit_api.testsuiteType method), 352

exportChildren() (autotest.client.tools.JUnit_api.errorType method), 347

exportChildren() (autotest.client.tools.JUnit_api.failureType method), 347

exportChildren() (autotest.client.tools.JUnit_api.propertiesType method), 348

exportChildren() (autotest.client.tools.JUnit_api.propertyType method), 348

exportChildren() (autotest.client.tools.JUnit_api.system_err method), 349

exportChildren() (autotest.client.tools.JUnit_api.system_out method), 349

exportChildren() (autotest.client.tools.JUnit_api.testcaseType method), 350

exportChildren() (autotest.client.tools.JUnit_api.testsuite method), 351

exportChildren() (autotest.client.tools.JUnit_api.testsuites method), 353

exportChildren() (autotest.client.tools.JUnit_api.testsuiteType method), 351

exportChildren() (autotest.client.tools.JUnit_api.testsuiteType method), 352

exportChildren() (autotest.client.tools.JUnit_api.testsuites exportLiteralChildren() (autotest.client.tools.JUnit_api.errorType method), 347

exportChildren() (autotest.client.tools.JUnit_api.testsuiteType exportLiteralChildren() (autotest.client.tools.JUnit_api.failureType method), 347

exportLiteral() (autotest.client.tools.JUnit_api.errorType method), 347

exportLiteral() (autotest.client.tools.JUnit_api.failureType exportLiteralChildren() (autotest.client.tools.JUnit_api.propertiesType method), 348

exportLiteral() (autotest.client.tools.JUnit_api.propertiesType exportLiteralChildren() (autotest.client.tools.JUnit_api.propertyType method), 348

exportLiteral() (autotest.client.tools.JUnit_api.system_err exportLiteralChildren() (autotest.client.tools.JUnit_api.system_err method), 349

exportLiteral() (autotest.client.tools.JUnit_api.system_out exportLiteralChildren() (autotest.client.tools.JUnit_api.system_out method), 349

exportLiteral() (autotest.client.tools.JUnit_api.testcaseType exportLiteralChildren() (autotest.client.tools.JUnit_api.system_out method), 349

exportLiteral() (autotest.client.tools.JUnit_api.testsuite exportLiteralChildren() (autotest.client.tools.JUnit_api.testcaseType method), 350

exportLiteral() (autotest.client.tools.JUnit_api.testsuites exportLiteralChildren() (autotest.client.tools.JUnit_api.testsuite method), 351

exportLiteralAttributes() (autotest.client.tools.JUnit_api.errorType method), 347

exportLiteralAttributes() (autotest.client.tools.JUnit_api.failureType method), 347

exportLiteralAttributes() (autotest.client.tools.JUnit_api.propertiesType method), 348

exportLiteralAttributes() (autotest.client.tools.JUnit_api.propertyType method), 348

exportLiteralAttributes() (autotest.client.tools.JUnit_api.system_err exportLiteralChildren() (autotest.client.tools.JUnit_api.system_err method), 349

exportLiteralAttributes() (autotest.client.tools.JUnit_api.system_out exportLiteralChildren() (autotest.client.tools.JUnit_api.system_out method), 349

exportLiteralAttributes() (autotest.client.tools.JUnit_api.testcaseType exportLiteralChildren() (autotest.client.tools.JUnit_api.system_out method), 349

exportLiteralAttributes() (autotest.client.tools.JUnit_api.testsuite exportLiteralChildren() (autotest.client.tools.JUnit_api.testcaseType method), 350

exportLiteralAttributes() (autotest.client.tools.JUnit_api.testsuites exportLiteralChildren() (autotest.client.tools.JUnit_api.testsuite method), 351

exportLiteralAttributes() (autotest.client.tools.JUnit_api.testsuiteType exportLiteralChildren() (autotest.client.tools.JUnit_api.testsuite method), 352

exportLiteralAttributes() (autotest.client.tools.JUnit_api.propertiesType ext_mkfs_options() (in module autotest.client.fsinfo), 206

exportLiteralAttributes() (autotest.client.tools.JUnit_api.propertiesType ext_tunables() (in module autotest.client.fsinfo), 206

exportLiteralAttributes() (autotest.client.kernel.kernel method), 214

exportLiteralAttributes() (autotest.client.kernel.kernel method), 214

extract_all_time_results() (in module au-

- totest.client.base_utils), 195
- extract_config_changes() (in module autotest.client.shared.test_utils.config_change_validation), 336
- extract_tarball() (in module autotest.client.base_utils), 195
- extract_tarball_to_dir() (in module autotest.client.base_utils), 195
- extract_version() (autotest.client.shared.base_check_version.base_check_python_version method), 250
- extraversion() (autotest.client.kernel.kernel method), 214
- F**
- factory() (autotest.client.tools.JUnit_api.errorType static method), 347
- factory() (autotest.client.tools.JUnit_api.failureType static method), 347
- factory() (autotest.client.tools.JUnit_api.propertiesType static method), 348
- factory() (autotest.client.tools.JUnit_api.propertyType static method), 348
- factory() (autotest.client.tools.JUnit_api.system_err static method), 349
- factory() (autotest.client.tools.JUnit_api.system_out static method), 349
- factory() (autotest.client.tools.JUnit_api.testcaseType static method), 350
- factory() (autotest.client.tools.JUnit_api.testsuite static method), 351
- factory() (autotest.client.tools.JUnit_api.testsuites static method), 353
- factory() (autotest.client.tools.JUnit_api.testsuiteType static method), 352
- fail() (autotest.client.shared.test_utils.unittest.TestCase method), 343
- failed (autotest.client.shared.test.Subtest attribute), 299
- failIf() (autotest.client.shared.test_utils.unittest.TestCase method), 343
- failIfAlmostEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 343
- failIfEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 343
- failUnless() (autotest.client.shared.test_utils.unittest.TestCase method), 343
- failUnlessAlmostEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 343
- failUnlessEqual() (autotest.client.shared.test_utils.unittest.TestCase method), 344
- failUnlessRaises() (autotest.client.shared.test_utils.unittest.TestCase method), 344
- failureException (autotest.client.shared.test_utils.unittest.TestCase attribute), 344
- failureType (class in autotest.client.tools.JUnit_api), 347
- fastcut() (in module autotest.client.shared.test_utils.functools_24), 336
- FdRedirectionLoggingManager (class in autotest.client.shared.logging_manager), 280
- feature_enabled() (in module autotest.client.kernel_config), 216
- file_check_python_version() (autotest.client.cmdparser.CommandParser method), 202
- fetch() (autotest.client.shared.git.GitRepoHelper method), 273
- fetch_package() (autotest.client.harness_autoserv.harness_autoserv method), 208
- fetch_pkg() (autotest.client.shared.base_packages.BasePackageManager method), 258
- fetch_pkg_file() (autotest.client.harness_autoserv.AutoservFetcher method), 208
- fetch_pkg_file() (autotest.client.shared.base_packages.GitFetcher method), 260
- fetch_pkg_file() (autotest.client.shared.base_packages.HttpFetcher method), 261
- fetch_pkg_file() (autotest.client.shared.base_packages.LocalFilesystemFetcher method), 261
- fetch_pkg_file() (autotest.client.shared.base_packages.RepositoryFetcher method), 261
- file_contains_pattern() (in module autotest.client.base_utils), 195
- file_formatter (autotest.client.shared.logging_config.LoggingConfig attribute), 279
- file_load() (in module autotest.client.tools.results2junit), 362
- FileFieldMonitor (class in autotest.client.shared.utils), 302
- FileFieldMonitor.Monitor (class in autotest.client.shared.utils), 302
- filesystem() (autotest.client.job.base_client_job method), 211
- filesystems() (in module autotest.client.partition), 224
- filter() (autotest.client.shared.logging_config.AllowBelowSeverity method), 279
- filter_partition_list() (in module autotest.client.partition), 224
- filtertype (autotest.client.net.net_tc.u32filter attribute), 234
- final_data (autotest.client.shared.test_utils.mock.SaveDataAfterCloseString attribute), 337
- find_command() (in module autotest.client.shared.utils), 308
- find_desired_python() (autotest.client.shared.base_check_version.base_check_python_version method), 250
- find_executable() (in module autotest.client.tools.boottool), 359

- find_free_port() (in module autotest.client.shared.utils), 309
- find_free_ports() (in module autotest.client.shared.utils), 309
- find_recipe() (autotest.client.harness_beaker.harness_beaker method), 208
- find_substring() (in module autotest.client.shared.utils), 309
- findTestCases() (in module autotest.client.shared.test_utils.unittest), 346
- finish_fsdev() (in module autotest.client.fsdev_disks), 204
- finish_init() (autotest.client.kernel.srpm_kernel method), 215
- finish_init() (autotest.client.kernel.srpm_kernel_suse method), 215
- fix_up_xen_kernel_makefile() (autotest.client.xen.xen method), 231
- flush() (autotest.client.net.net_utils.network_interface method), 236
- flush() (autotest.client.shared.logging_manager.LoggingFile method), 280
- FMT (autotest.client.tools.boottool.EfiVar attribute), 358
- ForAll (class in autotest.client.shared.utils), 302
- ForAllIP (class in autotest.client.shared.utils), 302
- ForAllPSE (class in autotest.client.shared.utils), 303
- force_copy() (in module autotest.client.base_utils), 195
- force_link() (in module autotest.client.base_utils), 195
- fork_nuke_subprocess() (in module autotest.client.parallel), 223
- fork_start() (in module autotest.client.parallel), 223
- fork_waitfor() (in module autotest.client.parallel), 223
- fork_waitfor_timed() (in module autotest.client.parallel), 223
- format_error() (in module autotest.client.shared.error), 268
- format_ip_with_mask() (in module autotest.client.shared.utils), 309
- format_str_for_message() (in module autotest.client.shared.utils), 309
- FRAME_KEY_DST_MAC (autotest.client.net.net_utils.ethernet attribute), 235
- FRAME_KEY_PAYLOAD (autotest.client.net.net_utils.ethernet attribute), 235
- FRAME_KEY_PROTO (autotest.client.net.net_utils.ethernet attribute), 235
- FRAME_KEY_SRC_MAC (autotest.client.net.net_utils.ethernet attribute), 235
- freememtotal() (in module autotest.client.shared.utils_memory), 327
- freespace() (in module autotest.client.base_utils), 195
- FromFile() (in module autotest.client.shared.json_template), 277
- fromkeys() (autotest.client.shared.backports.collections.OrderedDict.OrderedDict class method), 330
- FromString() (in module autotest.client.shared.json_template), 277
- fs_tag (autotest.client.partition.FsOptions attribute), 223
- fsck() (autotest.client.partition.partition method), 225
- fsdev_disks (class in autotest.client.fsdev_disks), 204
- FsdevManager (class in autotest.client.fsdev_mgr), 206
- FsOptions (class in autotest.client.partition), 223
- fstype (autotest.client.partition.FsOptions attribute), 224
- ftrace (class in autotest.client.profilers.ftrace.ftrace), 241
- full_path() (in module autotest.client.cpuset), 203
- function_any_args_mapping (class in autotest.client.shared.test_utils.mock), 337
- function_mapping (class in autotest.client.shared.test_utils.mock), 337
- FunctionTestCase (class in autotest.client.shared.test_utils.unittest), 345
- ## G
- gdb_report() (in module autotest.client.tools.crash_handler), 360
- generate_bin_search_paths() (in module autotest.client.os_dep), 220
- generate_html_report() (in module autotest.client.shared.report), 292
- generate_include_search_paths() (in module autotest.client.os_dep), 221
- generate_json_file() (in module autotest.client.shared.report), 292
- generate_library_search_paths() (in module autotest.client.os_dep), 221
- generate_random_id() (in module autotest.client.shared.utils), 309
- generate_random_string() (in module autotest.client.shared.utils), 309
- generate_random_string() (in module autotest.client.tools.crash_handler), 360
- generate_tmp_file_name() (in module autotest.client.shared.utils), 309
- get() (autotest.client.config.config method), 202
- get() (autotest.client.shared.base_job.job_state method), 255
- get() (autotest.client.test_config.config_loader method), 230
- get_active_interfaces() (autotest.client.net.net_utils.bonding method), 234
- get_advertised_link_modes() (autotest.client.net.net_utils.network_interface method), 236

- get_all_controllers() (in module autotest.client.shared.utils_cgroup), 321
- get_arch() (autotest.client.shared.hosts.base_classes.Host method), 333
- get_arch() (autotest.client.shared.utils_koji.RPMFileNameInfo method), 327
- get_arch() (in module autotest.client.shared.utils), 309
- get_architecture() (autotest.client.tools.boottool.Grubby method), 354
- get_archive_tarball_name() (in module autotest.client.shared.utils), 309
- get_attr_name() (autotest.client.shared.enum.Enum static method), 268
- get_autodir() (autotest.client.shared.hosts.base_classes.Host method), 333
- get_autotest_root() (autotest.client.shared.logging_config.LoggingConfig class method), 279
- get_average() (autotest.client.shared.utils.Statistic method), 303
- get_beaker_code() (in module autotest.client.harness_beaker), 208
- get_boot_id() (autotest.client.shared.hosts.base_classes.Host method), 333
- get_boot_numa() (in module autotest.client.cpuset), 203
- get_bootloader() (autotest.client.tools.boottool.Grubby method), 354
- get_buddy_info() (in module autotest.client.shared.utils_memory), 327
- get_carrier() (autotest.client.net.net_utils.network_interface method), 236
- get_cc() (in module autotest.client.base_utils), 195
- get_cgroup_index() (autotest.client.shared.utils_cgroup.Cgroup method), 320
- get_cgroup_mountpoint() (in module autotest.client.shared.utils_cgroup), 321
- get_cgroup_name() (autotest.client.shared.utils_cgroup.Cgroup method), 320
- get_children_pids() (in module autotest.client.shared.utils), 310
- get_class() (autotest.client.net.net_tc.prio method), 232
- get_classname() (autotest.client.tools.JUnit_api.testcaseType method), 350
- get_cmdline() (autotest.client.shared.hosts.base_classes.Host method), 333
- get_context() (in module autotest.client.shared.error), 268
- get_cpu_arch() (in module autotest.client.base_utils), 195
- get_cpu_family() (in module autotest.client.base_utils), 195
- get_cpu_info() (in module autotest.client.base_utils), 195
- get_cpu_percentage() (in module autotest.client.shared.utils), 310
- get_cpu_stat() (in module autotest.client.base_utils), 195
- get_cpu_status_string() (autotest.client.shared.utils.SystemLoad method), 304
- get_cpu_vendor() (in module autotest.client.base_utils), 195
- get_cpu_vendor_name() (in module autotest.client.base_utils), 195
- get_cpus() (in module autotest.client.cpuset), 203
- get_current_kernel_arch() (in module autotest.client.base_utils), 195
- get_data() (autotest.client.tools.boottool.EfiVar method), 358
- get_data_files() (in module autotest.client.setup), 228
- get_default() (autotest.client.tools.boottool.Grubby method), 355
- get_default_command() (autotest.client.shared.utils_koji.KojiClient method), 322
- get_default_index() (autotest.client.tools.boottool.Grubby method), 355
- get_default_koji_tag() (in module autotest.client.shared.utils_koji), 327
- get_default_title() (autotest.client.tools.boottool.Grubby method), 355
- get_dest_qdisc() (autotest.client.net.net_tc.tcfilter method), 233
- get_device() (autotest.clientprofilers.blktrace.blktrace.blktrace method), 240
- get_device_name() (autotest.client.shared.iscsi.Iscsi method), 274
- get_disk_list() (in module autotest.client.fsdev_disks), 204
- get_disks() (in module autotest.client.base_utils), 195
- get_distro() (autotest.client.shared.distro.Probe method), 35, 266
- get_driver() (autotest.client.net.net_utils.network_interface method), 236
- get_driver() (autotest.client.net.net_utils_mock.network_interface_mock method), 239
- get_entries() (autotest.client.tools.boottool.Grubby method), 355
- get_entry() (autotest.client.tools.boottool.Grubby method), 355
- get_error() (autotest.client.tools.JUnit_api.testcaseType method), 350
- get_errors() (autotest.client.tools.JUnit_api.testsuite method), 351
- get_extentiontype_() (autotest.client.tools.JUnit_api.testsuite method), 351
- get_failure() (autotest.client.tools.JUnit_api.testcaseType method), 350

- get_failures() (autotest.client.tools.JUnit_api.testsuite method), 351
- get_fetcher() (autotest.client.shared.base_packages.BasePackageManager method), 258
- get_field() (in module autotest.client.shared.utils), 310
- get_file() (autotest.client.shared.hosts.base_classes.Host method), 333
- get_file() (in module autotest.client.shared.utils), 310
- get_file_arch() (in module autotest.client.base_utils), 195
- get_filelist() (in module autotest.client.setup), 228
- get_filename_without_arch() (autotest.client.shared.utils_koji.RPMFileNameInfo method), 327
- get_filename_without_suffix() (autotest.client.shared.utils_koji.RPMFileNameInfo method), 327
- get_fsck_exec() (autotest.client.partition.partition method), 225
- get_fsdev_mgr() (autotest.client.fsdev_disks.fsdev_disks method), 204
- get_full_pci_id() (in module autotest.client.shared.utils), 310
- get_full_text_result() (autotest.client.shared.test.Subtest class method), 299
- get_grubby_version() (autotest.client.tools.boottool.Grubby method), 355
- get_grubby_version_raw() (autotest.client.tools.boottool.Grubby method), 355
- get_handle() (autotest.client.net.net_tc.qdisc method), 232
- get_handle() (autotest.client.net.net_tc.tcfilter method), 233
- get_hash_from_file() (in module autotest.client.shared.utils), 310
- get_host_from_id() (in module autotest.client.shared.base_barrier), 249
- get_hostname() (autotest.client.tools.JUnit_api.testsuite method), 351
- get_huge_page_size() (in module autotest.client.shared.utils_memory), 327
- get_hwaddr() (autotest.client.net.net_utils.network_interface method), 236
- get_hwclock_seconds() (in module autotest.client.base_utils), 195
- get_id() (autotest.client.tools.JUnit_api.testsuiteType method), 352
- get_info() (autotest.client.tools.boottool.Grubby method), 355
- get_info_file() (in module autotest.client.shared.report), 292
- get_info_from_core() (in module autotest.client.tools.crash_handler), 360
- get_info_lines() (autotest.client.tools.boottool.Grubby method), 355
- get_io_scheduler() (autotest.client.partition.partition method), 225
- get_io_scheduler_list() (autotest.client.partition.partition method), 225
- get_iosched_path() (in module autotest.client.partition), 224
- get_ip_local() (autotest.client.net.net_utils.network_utils method), 237
- get_ip_local_port_range() (in module autotest.client.shared.utils), 310
- get_ipaddr() (autotest.client.net.net_utils.network_interface method), 236
- get_ipaddr() (autotest.client.net.net_utils_mock.network_interface_mock method), 239
- get_kernel_build_arch() (autotest.client.kernel.kernel method), 214
- get_kernel_build_ident() (autotest.client.kernel.kernel method), 214
- get_kernel_build_release() (autotest.client.kernel.kernel method), 214
- get_kernel_build_ver() (autotest.client.kernel.kernel method), 214
- get_kernel_tree() (autotest.client.kernel.kernel method), 214
- get_kernel_ver() (autotest.client.shared.hosts.base_classes.Host method), 333
- get_kvm_arch() (in module autotest.client.kvm_control), 217
- get_leaf_qdisc() (autotest.client.net.net_tc.tcclass method), 233
- get_load_per_cpu() (in module autotest.client.shared.utils_cgroup), 322
- get_loaded_modules() (in module autotest.client.base_utils), 196
- get_logging_manager() (in module autotest.client.shared.logging_manager), 281
- get_mappings_2x() (in module autotest.client.kernelexpand), 217
- get_mappings_post_2x() (in module autotest.client.kernelexpand), 217
- get_max() (autotest.client.shared.utils.Statistic method), 303
- get_mem_nodes() (in module autotest.client.cpuset), 203
- get_mem_status_string() (autotest.client.shared.utils.SystemLoad method), 304
- get_meminfo() (autotest.client.shared.hosts.base_classes.Host method), 333
- get_message() (autotest.client.tools.JUnit_api.errorType method), 347
- get_message() (autotest.client.tools.JUnit_api.failureType method), 347

- get_mii_status() (autotest.client.net.net_utils.bonding method), 234
- get_min() (autotest.client.shared.utils.Statistic method), 303
- get_minor() (autotest.client.net.net_tc.tcclass method), 233
- get_mirror_list() (autotest.client.shared.base_packages.BasePackageManager method), 258
- get_mode() (autotest.client.net.net_utils.bonding method), 234
- get_modules_dir() (in module autotest.client.base_utils), 196
- get_mount_info() (in module autotest.client.partition), 224
- get_mountpoint() (autotest.client.partition.partition method), 225
- get_name() (autotest.client.net.net_utils.network_interface method), 236
- get_name() (autotest.client.tools.boottool.EfiVar method), 358
- get_name() (autotest.client.tools.JUnit_api.propertyType method), 348
- get_name() (autotest.client.tools.JUnit_api.testcaseType method), 350
- get_name() (autotest.client.tools.JUnit_api.testsuite method), 351
- get_name_of_init() (in module autotest.client.shared.service), 293
- get_num_cpu() (autotest.client.shared.hosts.base_classes.Host method), 333
- get_num_huge_pages() (in module autotest.client.shared.utils_memory), 327
- get_num_logical_cpus_per_socket() (in module autotest.client.shared.utils), 310
- get_nvr_info() (autotest.client.shared.utils_koji.RPMFileNameInfo method), 327
- get_open_func() (autotest.client.shared.hosts.base_classes.Host method), 333
- get_os_vendor() (in module autotest.client.base_utils), 196
- get_package() (autotest.client.tools.JUnit_api.testsuiteType method), 352
- get_package_data() (in module autotest.client.setup), 228
- get_package_dir() (in module autotest.client.setup), 228
- get_package_management() (autotest.client.shared.software_manager.SystemInspector method), 297
- get_package_name() (autotest.client.shared.base_packages.BasePackageManager method), 258
- get_packages() (in module autotest.client.setup), 228
- get_packed() (autotest.client.tools.boottool.EfiVar method), 358
- get_param() (autotest.client.bkr_xml.Task method), 201
- get_parent_class() (autotest.client.net.net_tc.qdisc method), 233
- get_parent_class() (autotest.client.net.net_tc.tcclass method), 233
- get_parent_pid() (in module autotest.client.tools.crash_handler), 360
- get_package_manager() (autotest.client.net.net_tc.tcfilter method), 233
- get_partition_list() (in module autotest.client.partition), 224
- get_patches() (autotest.client.kernel.kernel method), 214
- get_path() (in module autotest.client.shared.utils), 310
- get_pid_cpu() (in module autotest.client.shared.utils), 310
- get_pid_from_file() (in module autotest.client.shared.utils), 311
- get_pid_path() (in module autotest.client.shared.utils), 311
- get_pids() (autotest.client.shared.utils_cgroup.Cgroup method), 320
- get_pkg_base_url() (autotest.client.shared.utils_koji.KojiClient method), 322
- get_pkg_info() (autotest.client.shared.utils_koji.KojiClient method), 322
- get_pkg_rpm_file_names() (autotest.client.shared.utils_koji.KojiClient method), 322
- get_pkg_rpm_info() (autotest.client.shared.utils_koji.KojiClient method), 323
- get_pkg_rpm_names() (autotest.client.shared.utils_koji.KojiClient method), 323
- get_pkg_urls() (autotest.client.shared.utils_koji.KojiClient method), 323
- get_pkgs() (autotest.client.shared.utils_koji.KojiClient method), 323
- get_priority() (autotest.client.net.net_tc.tcfilter method), 233
- get_process_name() (in module autotest.client.shared.utils), 311
- get_processed_tests() (autotest.client.harness_beaker.harness_beaker method), 208
- get_properties() (autotest.client.tools.JUnit_api.testsuite method), 351
- get_property() (autotest.client.shared.utils_cgroup.Cgroup method), 320
- get_property() (autotest.client.tools.JUnit_api.propertiesType method), 348
- get_protocol() (autotest.client.net.net_tc.tcfilter method), 233
- get_pwd() (autotest.client.shared.utils_cgroup.CgroupModules

- method), 321
- get_recipe() (autotest.client.bkr_proxy.BkrProxy method), 198
- get_recipe_from_LC() (autotest.client.harness_beaker.harness_beaker method), 208
- get_relative_path() (in module autotest.client.shared.utils), 311
- get_repo() (in module autotest.client.shared.git), 273
- get_result() (autotest.client.shared.test.Subtest class method), 299
- get_results_dir_list() (in module autotest.client.tools.crash_handler), 360
- get_scratch_base_url() (autotest.client.shared.utils_koji.KojiClient method), 323
- get_scratch_pkg_urls() (autotest.client.shared.utils_koji.KojiClient method), 323
- get_scratch_pkgs() (autotest.client.shared.utils_koji.KojiClient method), 323
- get_screen_text() (autotest.client.shared.progressbar.ProgressBar method), 291
- get_scripts() (in module autotest.client.setup), 228
- get_section_values() (autotest.client.shared.settings.Settings method), 294
- get_server_log_dir() (autotest.client.shared.logging_config.LoggingConfig class method), 279
- get_session_options() (autotest.client.shared.utils_koji.KojiClient method), 324
- get_slave_interfaces() (autotest.client.net.net_utils.bonding method), 234
- get_speed() (autotest.client.net.net_utils.network_interface method), 236
- get_state() (autotest.client.shared.base_job.base_job method), 252
- get_stats() (autotest.client.net.net_utils.network_interface method), 236
- get_stats_diff() (autotest.client.net.net_utils.network_interface method), 236
- get_status() (autotest.client.shared.utils.FileFieldMonitor method), 302
- get_stderr() (autotest.client.shared.utils.AsyncJob method), 301
- get_stderr_level() (in module autotest.client.shared.utils), 311
- get_stdout() (autotest.client.shared.utils.AsyncJob method), 302
- get_stream_tee_file() (in module autotest.client.shared.utils), 311
- get_string() (autotest.client.shared.enum.Enum method), 268
- get_submodules() (in module autotest.client.base_utils), 196
- get_supported_link_modes() (autotest.client.net.net_utils.network_interface method), 236
- get_system_err() (autotest.client.tools.JUnit_api.testsuite method), 351
- get_system_out() (autotest.client.tools.JUnit_api.testsuite method), 351
- get_systemmap() (in module autotest.client.base_utils), 196
- get_tarball_name() (autotest.client.shared.base_packages.BasePackageManager static method), 258
- get_target_id() (autotest.client.shared.iscsi.Iscsi method), 274
- get_tasks() (in module autotest.client.cpuset), 203
- get_temp_file_path() (in module autotest.client.shared.test_utils.config_change_validation), 336
- get_test_keyval() (in module autotest.client.tools.regression), 362
- get_test_name() (autotest.client.harness_beaker.harness_beaker method), 209
- get_testcase() (autotest.client.tools.JUnit_api.testsuite method), 351
- get_tests() (autotest.client.tools.JUnit_api.testsuite method), 351
- get_testsuite() (autotest.client.tools.JUnit_api.testsuites method), 353
- get_text_result() (autotest.client.shared.test.Subtest class method), 299
- get_thread_cpu() (in module autotest.client.shared.utils), 311
- get_time() (autotest.client.tools.JUnit_api.testcaseType method), 350
- get_time() (autotest.client.tools.JUnit_api.testsuite method), 351
- get_timestamp() (autotest.client.tools.JUnit_api.testsuite method), 351
- get_timestamped_log_name() (autotest.client.shared.logging_config.LoggingConfig class method), 279
- get_title_for_kernel() (autotest.client.tools.boottool.Grubby method), 356
- get_titles() (autotest.client.tools.boottool.Grubby method), 356
- get_tmp_dir() (autotest.client.shared.hosts.base_classes.Host method), 333
- get_top_commit() (autotest.client.shared.git.GitRepoHelper

- method), 273
 - get_top_tag() (autotest.client.shared.git.GitRepoHelper method), 273
 - get_type() (autotest.client.tools.boottool.Grubby method), 356
 - get_type() (autotest.client.tools.JUnit_api.errorType method), 347
 - get_type() (autotest.client.tools.JUnit_api.failureType method), 347
 - get_unique_name() (in module autotest.client.shared.utils), 311
 - get_unmounted_partition_list() (in module autotest.client.partition), 224
 - get_unused_port() (in module autotest.client.shared.utils), 311
 - get_updated_content() (autotest.client.tools.boottool.EliloConf method), 359
 - get_uptime() (in module autotest.client.base_utils), 196
 - get_value() (autotest.client.shared.enum.Enum method), 268
 - get_value() (autotest.client.shared.settings.Settings method), 295
 - get_value() (autotest.client.tools.JUnit_api.propertyType method), 348
 - get_valueOf_() (autotest.client.tools.JUnit_api.errorType method), 347
 - get_valueOf_() (autotest.client.tools.JUnit_api.failureType method), 347
 - get_vendor_from_pci_id() (in module autotest.client.shared.utils), 311
 - get_version() (autotest.client.shared.openvswitch.OpenVSwitchControl class method), 288
 - get_version() (autotest.client.shared.openvswitch.ServiceManagerInterface class method), 290
 - get_version() (autotest.client.shared.utils.VersionableClass class method), 306
 - get_version() (in module autotest.client.shared.version), 328
 - get_vmlinux() (in module autotest.client.base_utils), 196
 - get_wait_up_processes() (autotest.client.shared.hosts.base_classes.Host method), 333
 - get_wakeon() (autotest.client.net.net_utils.network_interface method), 236
 - get_xen_build_ver() (autotest.client.xen.xen method), 231
 - get_xen_kernel_build_ver() (autotest.client.xen.xen method), 231
 - getAvg() (autotest.client.tools.regression.Sample method), 361
 - getAvgPercent() (autotest.client.tools.regression.Sample method), 361
 - getSD() (autotest.client.tools.regression.Sample method), 361
 - getSDRate() (autotest.client.tools.regression.Sample method), 361
 - getTestCaseNames() (autotest.client.shared.test_utils.unittest.TestLoader method), 345
 - getTestCaseNames() (in module autotest.client.shared.test_utils.unittest), 346
 - getTtestPvalue() (autotest.client.tools.regression.Sample method), 361
 - git_archive_cmd_pattern (autotest.client.shared.base_packages.GitFetcher attribute), 260
 - git_cmd() (autotest.client.shared.git.GitRepoHelper method), 273
 - GitFetcher (class in autotest.client.shared.base_packages), 260
 - GitRepoHelper (class in autotest.client.shared.git), 272
 - global_level (autotest.client.shared.logging_config.LoggingConfig attribute), 279
 - grep() (in module autotest.client.base_utils), 196
 - Grubby (class in autotest.client.tools.boottool), 353
 - grubby_build() (autotest.client.tools.boottool.Grubby method), 356
 - grubby_install() (autotest.client.tools.boottool.Grubby method), 356
 - grubby_install_backup() (autotest.client.tools.boottool.Grubby method), 356
 - grubby_install_fetch_tarball() (autotest.client.tools.boottool.Grubby method), 356
 - grubby_install_patch_makefile() (autotest.client.tools.boottool.Grubby method), 356
 - guess_type() (in module autotest.client.shared.magic), 282
 - GUID_CONTENT (autotest.client.tools.boottool.EfiVar attribute), 358
 - GUID_FMT (autotest.client.tools.boottool.EfiVar attribute), 358
- ## H
- handle_persistent_option() (autotest.client.job.base_client_job method), 211
 - handle_recipe() (autotest.client.bkr_xml.BeakerXMLParser method), 201
 - handle_recipes() (autotest.client.bkr_xml.BeakerXMLParser method), 201
 - handle_starttag() (autotest.client.shared.utils_koji.KojiDirIndexParser method), 324
 - handle_task() (autotest.client.bkr_xml.BeakerXMLParser method), 201

[handle_task_param\(\)](#) (autotest.client.bkr_xml.BeakerXMLParser method), 201
[handle_task_params\(\)](#) (autotest.client.bkr_xml.BeakerXMLParser method), 201
[handle_tasks\(\)](#) (autotest.client.bkr_xml.BeakerXMLParser method), 201
[HARDWARE_REPAIR_REQUEST_THRESHOLD](#) (autotest.client.shared.hosts.base_classes.Host attribute), 332
[harness](#) (class in autotest.client.harness), 207
[harness_autoserv](#) (class in autotest.client.harness_autoserv), 208
[harness_beaker](#) (class in autotest.client.harness_beaker), 208
[harness_select\(\)](#) (autotest.client.job.base_client_job method), 211
[harness_simple](#) (class in autotest.client.harness_simple), 209
[harness_standalone](#) (class in autotest.client.harness_standalone), 210
[HarnessError](#), 272
[HarnessException](#), 208
[has\(\)](#) (autotest.client.shared.base_job.job_state method), 255
[has_failed\(\)](#) (autotest.client.shared.test.Subtest class method), 299
[has_pbzip2\(\)](#) (in module autotest.client.shared.base_packages), 262
[hasContent_\(\)](#) (autotest.client.tools.JUnit_api.errorType method), 347
[hasContent_\(\)](#) (autotest.client.tools.JUnit_api.failureType method), 347
[hasContent_\(\)](#) (autotest.client.tools.JUnit_api.propertiesType method), 348
[hasContent_\(\)](#) (autotest.client.tools.JUnit_api.propertyType method), 348
[hasContent_\(\)](#) (autotest.client.tools.JUnit_api.system_err method), 349
[hasContent_\(\)](#) (autotest.client.tools.JUnit_api.system_out method), 349
[hasContent_\(\)](#) (autotest.client.tools.JUnit_api.testcaseType method), 350
[hasContent_\(\)](#) (autotest.client.tools.JUnit_api.testsuite method), 351
[hasContent_\(\)](#) (autotest.client.tools.JUnit_api.testsuites method), 353
[hasContent_\(\)](#) (autotest.client.tools.JUnit_api.testsuiteType method), 352
[hash\(\)](#) (in module autotest.client.shared.utils), 311
[hash_file\(\)](#) (in module autotest.client.base_utils), 196
[HDR_LEN](#) (autotest.client.net.net_utils.ethernet attribute), 235
[header\(\)](#) (in module autotest.client.os_dep), 221
[headers\(\)](#) (in module autotest.client.os_dep), 221
[help\(\)](#) (autotest.client.cmdparser.CommandParser class method), 202
[Host](#) (class in autotest.client.shared.hosts.base_classes), 332
[HostInstallProfileError](#), 269
[HostInstallTimeoutError](#), 272
[HostRunErrorMixIn](#), 272
[HOURS_TO_WAIT_FOR_RECOVERY](#) (autotest.client.shared.hosts.base_classes.Host attribute), 332
[HttpFetcher](#) (class in autotest.client.shared.base_packages), 261
[human_format\(\)](#) (in module autotest.client.base_utils), 196
I
[id\(\)](#) (autotest.client.net.net_tc.qdisc method), 233
[id\(\)](#) (autotest.client.net.net_tc.tcclass method), 233
[id\(\)](#) (autotest.client.shared.test_utils.unittest.ClassTestSuite method), 344
[id\(\)](#) (autotest.client.shared.test_utils.unittest.FunctionTestCase method), 346
[id\(\)](#) (autotest.client.shared.test_utils.unittest.TestCase method), 344
[import_module\(\)](#) (in module autotest.client.setup_modules), 229
[import_site_class\(\)](#) (in module autotest.client.shared.utils), 312
[import_site_function\(\)](#) (in module autotest.client.shared.utils), 312
[import_site_module\(\)](#) (in module autotest.client.shared.utils), 312
[import_site_symbol\(\)](#) (in module autotest.client.shared.utils), 312
[include_partition\(\)](#) (autotest.client.fsdev_mgr.BaseFsdevManager method), 206
[increment\(\)](#) (autotest.client.job.status_indenter method), 213
[increment\(\)](#) (autotest.client.shared.base_job.status_indenter method), 256
[increment\(\)](#) (autotest.client.shared.progressbar.ProgressBar method), 292
[indent](#) (autotest.client.job.status_indenter attribute), 213
[indent](#) (autotest.client.shared.base_job.status_indenter attribute), 256
[init\(\)](#) (autotest.client.shared.git.GitRepoHelper method), 273
[init\(\)](#) (autotest.client.shared.utils_cgroup.CgroupModules method), 321
[init_db\(\)](#) (autotest.client.shared.openvswitch.OpenVSwitch method), 287

[init_new\(\)](#) (autotest.client.shared.openvswitch.OpenVSwitch.insert_property() (autotest.client.tools.JUnit_api.propertiesType method), 287
[init_recipe_from_beaker\(\)](#) (autotest.client.harness_beaker.harness_beaker method), 209
[init_system\(\)](#) (autotest.client.shared.openvswitch.OpenVSwitchSystemMethod method), 289
[init_task_params\(\)](#) (autotest.client.harness_beaker.harness_beaker method), 209
[init_test\(\)](#) (in module autotest.client.setup_job), 228
[initialize\(\)](#) (autotest.client.profiler.profiler method), 228
[initialize\(\)](#) (autotest.clientprofilers.blktrace.blktrace.blktrace method), 240
[initialize\(\)](#) (autotest.clientprofilers.catprofile.catprofile.catprofile method), 240
[initialize\(\)](#) (autotest.clientprofilers.cmdprofile.cmdprofile.cmdprofile method), 240
[initialize\(\)](#) (autotest.clientprofilers.cpiostat.cpiostat.cpiostat method), 241
[initialize\(\)](#) (autotest.clientprofilers.ftrace.ftrace.ftrace method), 241
[initialize\(\)](#) (autotest.clientprofilers.inotify.inotify.inotify method), 242
[initialize\(\)](#) (autotest.clientprofilers.iostat.iostat.iostat method), 242
[initialize\(\)](#) (autotest.clientprofilers.kvm_stat.kvm_stat.kvm_stat method), 243
[initialize\(\)](#) (autotest.clientprofilers.lockmeter.lockmeter.lockmeter method), 243
[initialize\(\)](#) (autotest.clientprofilers.lttng.lttng.lttng method), 244
[initialize\(\)](#) (autotest.clientprofilers.mpstat.mpstat.mpstat method), 244
[initialize\(\)](#) (autotest.clientprofilers.oprofile.oprofile.oprofile method), 245
[initialize\(\)](#) (autotest.clientprofilers.perf.perf.perf method), 245
[initialize\(\)](#) (autotest.clientprofilers.readprofile.readprofile.readprofile method), 246
[initialize\(\)](#) (autotest.clientprofilers.sar.sar.sar method), 246
[initialize\(\)](#) (autotest.clientprofilers.systemtap.systemtap.systemtap method), 247
[initialize\(\)](#) (autotest.clientprofilers.vmstat.vmstat.vmstat method), 247
[initialize\(\)](#) (autotest.client.shared.test.base_test method), 300
[initialize\(\)](#) (autotest.client.shared.utils_cgroup.Cgroup method), 320
[inner_containers_of\(\)](#) (in module autotest.client.cpuset), 203
[inotify](#) (class in autotest.client.profilers.inotify.inotify), 242
[insert_property\(\)](#) (autotest.client.tools.JUnit_api.propertiesType method), 348
[insert_testcase\(\)](#) (autotest.client.tools.JUnit_api.testsuite method), 351
[insert_testsuite\(\)](#) (autotest.client.tools.JUnit_api.testsuites method), 353
[install\(\)](#) (autotest.client.kernel.kernel method), 214
[install\(\)](#) (autotest.client.kernel.rpm_kernel method), 215
[install\(\)](#) (autotest.client.kernel.rpm_kernel_suse method), 215
[install\(\)](#) (autotest.client.kernel.srpm_kernel method), 215
[install\(\)](#) (autotest.client.shared.hosts.base_classes.Host method), 334
[install\(\)](#) (autotest.client.shared.software_manager.AptBackend method), 295
[install\(\)](#) (autotest.client.shared.software_manager.YumBackend method), 297
[install\(\)](#) (autotest.client.shared.software_manager.ZypperBackend method), 298
[install\(\)](#) (autotest.client.xen.xen method), 231
[install_distro_packages\(\)](#) (in module autotest.client.shared.software_manager), 298
[install_pkg\(\)](#) (autotest.client.job.base_client_job method), 211
[install_pkg\(\)](#) (autotest.client.shared.base_packages.BasePackageManager method), 259
[install_pkg_post\(\)](#) (autotest.client.shared.base_packages.GitFetcher method), 261
[install_pkg_post\(\)](#) (autotest.client.shared.base_packages.RepositoryFetcher method), 261
[install_pkg_setup\(\)](#) (autotest.client.shared.base_packages.RepositoryFetcher method), 261
[install_what_provides\(\)](#) (autotest.client.shared.software_manager.BaseBackend method), 296
[INSTALLED_OUTPUT](#) (autotest.client.shared.software_manager.DpkgBackend attribute), 296
[InstallError](#), 270
[interactive_download\(\)](#) (in module autotest.client.shared.utils), 312
[InterruptedThread](#) (class in autotest.client.shared.utils), 303
[InvalidAutotestResultDirError](#), 292
[InvalidOutputDirError](#), 292
[io_attr\(\)](#) (in module autotest.client.cpuset), 203
[iostat](#) (class in autotest.client.profilers.iostat.iostat), 242
[ip_to_long\(\)](#) (in module autotest.client.shared.utils), 313
[is_autoneg_advertised\(\)](#) (autotest.client.net.net_utils.network_interface method), 236
[is_autoneg_on\(\)](#) (autotest.client.net.net_utils.network_interface method), 236

- is_bondable() (autotest.client.net.net_utils.bonding method), 234
- is_cgroup() (autotest.client.shared.utils_cgroup.Cgroup method), 320
- is_command_valid() (autotest.client.shared.utils_koji.KojiClient method), 324
- is_config_valid() (autotest.client.shared.utils_koji.KojiClient method), 324
- is_down() (autotest.client.net.net_utils.network_interface method), 236
- is_down() (autotest.client.net.net_utils_mock.network_interface method), 239
- is_enabled() (autotest.client.net.net_utils.bonding method), 234
- is_end() (autotest.client.shared.base_job.status_log_entry method), 257
- is_failure() (in module autotest.client.shared.log), 279
- is_file_and_readable() (in module autotest.client.os_dep), 221
- is_file_and_rx() (in module autotest.client.os_dep), 221
- is_finished() (autotest.client.shared.base_syncdata.SessionData method), 263
- is_full_duplex() (autotest.client.net.net_utils.network_interface method), 236
- is_installed() (autotest.client.shared.openvswitch.OpenVSwitchSystem method), 289
- is_instance_comparator (class in autotest.client.shared.test_utils.mock), 338
- is_int() (in module autotest.client.tools.regression), 362
- is_linux_fs_type() (in module autotest.client.partition), 225
- is_loopback_enabled() (autotest.client.net.net_utils.network_interface method), 236
- is_loopback_enabled() (autotest.client.net.net_utils_mock.network_interface method), 239
- is_mounted() (in module autotest.client.shared.utils), 313
- is_pause_autoneg_on() (autotest.client.net.net_utils.network_interface method), 236
- is_pkg_spec_build_valid() (autotest.client.shared.utils_koji.KojiClient method), 324
- is_pkg_spec_tag_valid() (autotest.client.shared.utils_koji.KojiClient method), 324
- is_pkg_valid() (autotest.client.shared.utils_koji.KojiClient method), 324
- is_port_free() (in module autotest.client.shared.utils), 313
- is_release_candidate() (in module autotest.client.kernel_versions), 216
- is_release_candidate() (in module autotest.client.shared.kernel_versions), 278
- is_released_kernel() (in module autotest.client.kernel_versions), 216
- is_released_kernel() (in module autotest.client.shared.kernel_versions), 278
- is_right_version() (autotest.client.shared.openvswitch.OpenVSwitchControl class method), 288
- is_right_version() (autotest.client.shared.openvswitch.OpenVSwitchControl class method), 289
- is_right_version() (autotest.client.shared.openvswitch.ServiceManagerSystem class method), 290
- is_right_version() (autotest.client.shared.openvswitch.ServiceManagerSystem class method), 290
- is_right_version() (autotest.client.shared.utils.VersionableClass class method), 306
- is_root_cgroup() (autotest.client.shared.utils_cgroup.Cgroup method), 320
- is_rx_pause_on() (autotest.client.net.net_utils.network_interface method), 236
- is_rx_summing_on() (autotest.client.net.net_utils.network_interface method), 236
- is_satisfied_by() (autotest.client.shared.test_utils.mock.anything_comparator method), 337
- is_satisfied_by() (autotest.client.shared.test_utils.mock.argument_comparator method), 337
- is_satisfied_by() (autotest.client.shared.test_utils.mock.equality_comparator method), 337
- is_satisfied_by() (autotest.client.shared.test_utils.mock.is_instance_comparator method), 338
- is_satisfied_by() (autotest.client.shared.test_utils.mock.is_string_comparator method), 338
- is_satisfied_by() (autotest.client.shared.test_utils.mock.regex_comparator method), 339
- is_scatter_gather_on() (autotest.client.net.net_utils.network_interface method), 237
- is_shutting_down() (autotest.client.shared.hosts.base_classes.Host method), 334
- is_start() (autotest.client.shared.base_job.status_log_entry method), 257
- is_string_comparator (class in autotest.client.shared.test_utils.mock), 338
- is_tso_on() (autotest.client.net.net_utils.network_interface method), 237
- is_tx_pause_on() (autotest.client.net.net_utils.network_interface method), 237
- is_tx_summing_on() (autotest.client.net.net_utils.network_interface method), 237
- is_up() (autotest.client.shared.hosts.base_classes.Host method), 334
- is_url() (in module autotest.client.shared.utils), 313

is_valid() (autotest.client.shared.utils_koji.KojiPkgSpec method), 326

is_valid_disk() (in module autotest.client.partition), 225

is_valid_partition() (in module autotest.client.partition), 225

is_valid_status() (in module autotest.client.shared.log), 279

isatty() (autotest.client.shared.logging_manager.LoggingFile method), 280

Iscsi (class in autotest.client.shared.iscsi), 274

iscsi_discover() (in module autotest.client.shared.iscsi), 274

iscsi_get_nodes() (in module autotest.client.shared.iscsi), 274

iscsi_get_sessions() (in module autotest.client.shared.iscsi), 274

iscsi_login() (in module autotest.client.shared.iscsi), 274

iscsi_logout() (in module autotest.client.shared.iscsi), 275

iso9660() (in module autotest.client.shared.iso9660), 275

Iso9660IsoInfo (class in autotest.client.shared.iso9660), 275

Iso9660IsoRead (class in autotest.client.shared.iso9660), 275

Iso9660Mount (class in autotest.client.shared.iso9660), 275

items() (autotest.client.shared.backports.collections.OrderedDict method), 330

iteritems() (autotest.client.shared.backports.collections.OrderedDict method), 330

iterkeys() (autotest.client.shared.backports.collections.OrderedDict method), 330

itervalues() (autotest.client.shared.backports.collections.OrderedDict method), 330

J

job (autotest.client.shared.hosts.base_classes.Host attribute), 334

job (class in autotest.client.job), 213

job_directory (class in autotest.client.shared.base_job), 254

job_directory.JobDirectoryException, 254

job_directory.MissingDirectoryException, 254

job_directory.UncreatableDirectoryException, 254

job_directory.UnwritableDirectoryException, 254

job_state (class in autotest.client.shared.base_job), 254

job_statuses (autotest.client.shared.base_job.TAPReport attribute), 250

JobError, 271

join() (autotest.client.shared.utils.InterruptedThread method), 303

join_bg_jobs() (in module autotest.client.shared.utils), 313

join_command() (autotest.clientprofilers.ftrace.ftrace.ftrace static method), 241

K

kernel (class in autotest.client.kernel), 213

kernel() (autotest.client.job.base_client_job method), 211

kernel_config (class in autotest.client.kernel_config), 216

kernel_string (autotest.client.kernel.rpm_kernel attribute), 215

kernel_string (autotest.client.kernel.rpm_kernel_suse attribute), 215

kernelexpand() (autotest.client.kernel.kernel method), 214

keys() (autotest.client.shared.backports.collections.OrderedDict method), 330

keyval_to_line() (autotest.client.tools.boottool.EliloConf method), 359

kill_process_tree() (in module autotest.client.shared.utils), 313

kill_watchdog() (autotest.client.harness_beaker.harness_beaker method), 209

KojiClient (class in autotest.client.shared.utils_koji), 322

KojiDirIndexParser (class in autotest.client.shared.utils_koji), 324

KojiPkgSpec (class in autotest.client.shared.utils_koji), 324

KojiScratchPkgSpec (class in autotest.client.shared.utils_koji), 326

kvm_stat (class in autotest.client.profilers.kvm_stat.kvm_stat), 243

L

last_boot_tag (autotest.client.shared.base_job.base_job attribute), 252

LD_SO_CONF (autotest.client.os_dep.Ldconfig attribute), 219

Ldconfig (class in autotest.client.os_dep), 219

Ldconfig() (autotest.client.os_dep.Ldconfig method), 220

Ldconfig.DirEntry (class in autotest.client.os_dep), 219

libraries() (in module autotest.client.os_dep), 221

library() (in module autotest.client.os_dep), 222

line_to_keyval() (autotest.client.tools.boottool.EliloConf method), 359

LinuxDistro (class in autotest.client.shared.distro), 35, 265

list() (autotest.client.net.net_utils.network_utils method), 237

list_all() (autotest.client.shared.software_manager.DpkgBackend method), 296

list_all() (autotest.client.shared.software_manager.RpmBackend method), 297

list_br() (autotest.client.shared.openvswitch.OpenVSwitchControl method), 288

list_br() (autotest.client.shared.openvswitch.OpenVSwitchControlCli_140 method), 288

list_files() (autotest.client.shared.software_manager.DpkgBackend method), 296

list_files() (autotest.client.shared.software_manager.RpmBackend method), 297

list_files_glob() (autotest.client.local_host.LocalHost method), 218

list_files_glob() (autotest.client.shared.hosts.base_classes.Host method), 334

list_grep() (in module autotest.client.base_utils), 196

list_mount_devices() (in module autotest.client.partition), 225

list_mount_points() (in module autotest.client.partition), 225

list_ports() (autotest.client.shared.openvswitch.OpenVSwitch method), 288

list_tests() (autotest.client.cmdparser.CommandParser class method), 202

listen_server (class in autotest.client.shared.base_barrier), 249

load() (in module autotest.client.shared.distro_def), 266

load_all_client_tests() (in module autotest.client.setup_job), 228

load_from_tree() (in module autotest.client.shared.distro_def), 266

load_kvm() (in module autotest.client.kvm_control), 217

load_module() (in module autotest.client.base_utils), 196

load_profiler() (autotest.clientprofilers.profilers method), 239

load_profiler() (autotest.client.shared.profiler_manager.profiler_manager method), 291

load_sched_tunable_values() (autotest.client.fsdev_disks.fsdev_disks method), 204

loaded_module_info() (in module autotest.client.base_utils), 196

loadTestsFromModule() (autotest.client.shared.test_utils.unittest.TestLoader method), 345

loadTestsFromName() (autotest.client.shared.test_utils.unittest.TestLoader method), 345

loadTestsFromNames() (autotest.client.shared.test_utils.unittest.TestLoader method), 345

loadTestsFromTestCase() (autotest.client.shared.test_utils.unittest.TestLoader method), 345

LocalFilesystemFetcher (class in autotest.client.shared.base_packages), 261

LocalHost (class in autotest.client.local_host), 218

LOCALTIME_FIELD (autotest.client.shared.base_job.status_log_entry attribute), 256

locate() (in module autotest.client.base_utils), 197

lockfile() (in module autotest.client.shared.utils), 313

lockmeter (class in autotest.client.profilers.lockmeter.lockmeter), 243

log() (autotest.client.xen.xen method), 231

log_after_each_iteration() (autotest.client.base_sysinfo.base_sysinfo method), 193

log_after_each_test() (autotest.client.base_sysinfo.base_sysinfo method), 193

log_and_ignore_errors() (in module autotest.client.shared.log), 279

log_append() (autotest.client.shared.test.Subtest class method), 299

log_before_each_iteration() (autotest.client.base_sysinfo.base_sysinfo method), 193

log_before_each_test() (autotest.client.base_sysinfo.base_sysinfo method), 193

log_kernel() (autotest.client.shared.hosts.base_classes.Host method), 334

log_last_traceback() (in module autotest.client.shared.utils), 313

log_line() (in module autotest.client.shared.utils), 313

log_per_reboot_data() (autotest.client.base_sysinfo.base_sysinfo method), 193

log_reboot() (autotest.client.shared.hosts.base_classes.Host method), 334

log_test_keyvals() (autotest.client.base_sysinfo.base_sysinfo method), 193

logfile (class in autotest.client.base_sysinfo), 194

loggable (class in autotest.client.base_sysinfo), 194

logged_in() (autotest.client.shared.iscsi.Iscsi method), 274

logging_config_object (autotest.client.shared.logging_manager.LoggingManager attribute), 280

LoggingConfig (class in autotest.client.shared.logging_config), 279

LoggingFile (class in autotest.client.shared.logging_manager), 280

LoggingManager (class in autotest.client.shared.logging_manager), 280

login() (autotest.client.shared.iscsi.Iscsi method), 274

logout() (autotest.client.shared.iscsi.Iscsi method), 274

long_to_ip() (in module autotest.client.shared.utils), 313

longMessage (autotest.client.shared.test_utils.unittest.TestCase attribute), 344

lttng (class in autotest.client.profilers.lttng.Lttng), 244

lv_check() (in module autotest.client.lv_utils), 219

lv_list() (in module autotest.client.lv_utils), 219

lv_list_all() (in module autotest.client.lv_utils), 219

M

mac_binary_to_string() (autotest.client.net.net_utils.ethernet method), 235

mac_string_to_binary() (autotest.client.net.net_utils.ethernet method), 235

machine_install() (autotest.client.shared.hosts.base_classes.Base method), 334

MagicLoggingConfig (class in autotest.client.shared.magic), 282

MagicMock (class in autotest.client.shared.mock), 284

MagicTest (class in autotest.client.shared.magic), 282

main (in module autotest.client.shared.test_utils.unittest), 346

main() (autotest.client.autotest_local.AutotestLocalApp method), 193

main() (in module autotest.client.tools.process_metrics), 361

main() (in module autotest.client.tools.results2junit), 362

main() (in module autotest.client.tools.scan_results), 362

make() (in module autotest.client.shared.utils), 314

make_path_bkrache() (in module autotest.client.bkr_proxy), 199

make_path_cmdlog() (in module autotest.client.bkr_proxy), 199

make_path_log() (in module autotest.client.bkr_proxy), 199

make_path_recipe() (in module autotest.client.bkr_proxy), 200

make_path_result() (in module autotest.client.bkr_proxy), 200

make_path_searcher() (in module autotest.client.os_dep), 222

make_path_status() (in module autotest.client.bkr_proxy), 200

make_path_watchdog() (in module autotest.client.bkr_proxy), 200

make_temp_file_copies() (in module autotest.client.shared.test_utils.config_change_validation), 336

makeSuite() (in module autotest.client.shared.test_utils.unittest), 346

manage_stderr() (autotest.client.shared.logging_manager.LoggingManager method), 280

manage_stdout() (autotest.client.shared.logging_manager.LoggingManager method), 280

manage_stream() (autotest.client.shared.logging_manager.LoggingManager method), 280

map_drive_name() (autotest.client.fsdev_mgr.BaseFsdevManager method), 206

mask_function (class in autotest.client.shared.test_utils.mock), 338

match() (autotest.client.shared.test_utils.mock.base_mapping method), 337

match() (autotest.client.shared.test_utils.mock.function_any_args_mapping method), 337

match_ext_options() (in module autotest.client.fsinfo), 206

match_fs() (in module autotest.client.fsdev_disks), 205

match_mkfs_option() (in module autotest.client.fsinfo), 206

match_xfs_options() (in module autotest.client.fsinfo), 207

matches_global_option_to_add() (autotest.client.tools.boottool.EliloConf method), 359

matches_global_option_to_remove() (autotest.client.tools.boottool.EliloConf method), 359

matrix_to_string() (in module autotest.client.shared.utils), 314

MAX_RECURSION_DEPTH (autotest.client.os_dep.Ldconfig attribute), 219

mbytes_per_mem_node() (in module autotest.client.cpuset), 203

memory_path() (in module autotest.client.cpuset), 203

mems_path() (in module autotest.client.cpuset), 203

memtotal() (in module autotest.client.shared.utils_memory), 327

merge_configs() (autotest.client.shared.settings.Settings method), 295

merge_ext_features() (in module autotest.client.fsinfo), 207

merge_trees() (in module autotest.client.shared.utils), 314

mirror_kernel_components() (in module autotest.client.kernelexpand), 217

MissingFormatter, 276

mk_cgroup() (autotest.client.shared.utils_cgroup.Cgroup method), 320

mk_cgroup_cgcreate() (autotest.client.shared.utils_cgroup.Cgroup method), 320

mkfs() (autotest.client.partition.partition method), 225

mkfs_all_disks() (in module autotest.client.fsdev_disks), 205

mkfs_proc() (autotest.client.partition.partition method), 226

mkfs_flags (autotest.client.partition.FsOptions attribute), 224

mkinitrd() (autotest.client.kernel.kernel method), 214

Mock (class in autotest.client.shared.mock), 283

mock_add_spec() (autotest.client.shared.mock.MagicMock method), 284

mock_add_spec() (autotest.client.shared.mock.NonCallableMagicMock

- method), 287
- mock_add_spec() (autotest.client.shared.mock.NonCallableMock method), 286
- mock_calls (autotest.client.shared.mock.NonCallableMock attribute), 286
- mock_class (class in autotest.client.shared.test_utils.mock), 338
- mock_function (class in autotest.client.shared.test_utils.mock), 338
- mock_god (class in autotest.client.shared.test_utils.mock), 338
- mock_io() (autotest.client.shared.test_utils.mock.mock_godnetif method), 338
- mock_open() (in module autotest.client.shared.mock), 287
- mock_up() (autotest.client.shared.test_utils.mock.mock_godnetif method), 338
- module_is_loaded() (in module autotest.client.base_utils), 197
- modules_needed() (in module autotest.client.kernel_config), 216
- monitor_disk_usage() (autotest.client.job.base_client_job method), 211
- mount() (autotest.client.partition.partition method), 226
- mount() (in module autotest.client.shared.utils), 314
- mount_options (autotest.client.partition.FsOptions attribute), 224
- mountpoint (autotest.client.profilers.fttrace.fttrace.fttrace attribute), 241
- move_self_into_container() (in module autotest.client.cpuset), 204
- move_tasks_into_container() (in module autotest.client.cpuset), 204
- mpstat (class in autotest.client.profilers.mpstat.mpstat), 244
- my_available_exclusive_mem_nodes() (in module autotest.client.cpuset), 204
- my_container_name() (in module autotest.client.cpuset), 204
- my_lock() (in module autotest.client.cpuset), 204
- my_mem_nodes() (in module autotest.client.cpuset), 204
- my_unlock() (in module autotest.client.cpuset), 204
- ## N
- name (autotest.client.net.net_tc.netem attribute), 232
- name (autotest.client.net.net_tc.pfifio attribute), 232
- name (autotest.client.net.net_tc.prio attribute), 232
- name_for_file() (autotest.client.shared.distro.Probe method), 35, 266
- name_for_file_contains() (autotest.client.shared.distro.Probe method), 36, 266
- namedtuple() (in module autotest.client.shared.backports.collections.namedtuple), 331
- Mock_fake_numa() (in module autotest.client.cpuset), 204
- need_mem_containers() (in module autotest.client.cpuset), 204
- net_rcv_object() (in module autotest.client.shared.base_syncdata), 263
- net_send_object() (in module autotest.client.shared.base_syncdata), 264
- NetCommunicationError, 270
- netem (class in autotest.client.net.net_tc), 232
- netif() (in module autotest.client.net.net_utils), 236
- netif_stub (class in autotest.client.net.net_utils_mock), 238
- netutils_netif() (in module autotest.client.net.net_utils_mock), 239
- network() (in module autotest.client.net.net_utils), 236
- network_destabilizing (autotest.client.shared.test.base_test attribute), 300
- network_interface (class in autotest.client.net.net_utils), 236
- network_interface_mock (class in autotest.client.net.net_utils_mock), 239
- network_utils (class in autotest.client.net.net_utils), 237
- new_handle() (in module autotest.client.net.net_tc), 232
- NEW_VAR (autotest.client.tools.boottool.EfiToolSys attribute), 358
- next() (in module autotest.client.shared.backports), 329
- next_step() (autotest.client.job.base_client_job method), 211
- next_step_append() (autotest.client.job.base_client_job method), 211
- next_step_prepend() (autotest.client.job.base_client_job method), 211
- NO_DEFAULT (autotest.client.shared.base_job.job_state attribute), 254
- NO_MODE (autotest.client.net.net_utils.bonding attribute), 234
- node_avail_kbytes() (in module autotest.client.cpuset), 204
- node_size() (in module autotest.client.shared.utils_memory), 327
- nodes_avail_mbytes() (in module autotest.client.cpuset), 204
- NonCallableMagicMock (class in autotest.client.shared.mock), 287
- NonCallableMock (class in autotest.client.shared.mock), 285
- NONEXISTENT_ATTRIBUTE (autotest.client.shared.test_utils.mock.mock_god attribute), 338
- noop() (autotest.client.job.base_client_job method), 212
- normalize_hostname() (in module autotest.client.shared.utils), 314

- NotAvailableError, 210
- nuke_pid() (in module autotest.client.shared.utils), 314
- nuke_subprocess() (in module autotest.client.shared.utils), 314
- numa_nodes() (in module autotest.client.shared.utils_memory), 327
- ## O
- only() (autotest.client.shared.profiler_manager.profiler_manager method), 291
- open() (autotest.client.net.net_utils.raw_socket method), 237
- open() (autotest.client.net.net_utils_mock.os_stub method), 239
- open_file() (autotest.client.shared.pidfile.PidFileManager method), 290
- open_write_close() (in module autotest.client.shared.utils), 314
- OpenVSwitch (class in autotest.client.shared.openvswitch), 287
- OpenVSwitchControl (class in autotest.client.shared.openvswitch), 287
- OpenVSwitchControlCli (class in autotest.client.shared.openvswitch), 288
- OpenVSwitchControlCli_140 (class in autotest.client.shared.openvswitch), 288
- OpenVSwitchControlDB (class in autotest.client.shared.openvswitch), 289
- OpenVSwitchControlDB_140 (class in autotest.client.shared.openvswitch), 289
- OpenVSwitchSystem (class in autotest.client.shared.openvswitch), 289
- oprofile (class in autotest.clientprofilers.oprofile.oprofile), 245
- opt_string2dict() (in module autotest.client.fsinfo), 207
- option_parser_usage (autotest.client.tools.boottool.OptionParser attribute), 357
- OptionParser (class in autotest.client.tools.boottool), 357
- opts_get_action() (autotest.client.tools.boottool.OptionParser method), 357
- opts_has_action() (autotest.client.tools.boottool.OptionParser method), 357
- OrderedDict (class in autotest.client.shared.backports.collections.OrderedDict), 329
- os_open() (in module autotest.client.net.net_utils_mock), 239
- os_stub (class in autotest.client.net.net_utils_mock), 239
- output_prepare() (autotest.client.shared.utils.AsyncJob method), 302
- output_prepare() (autotest.client.shared.utils.BgJob method), 302
- override_value() (autotest.client.shared.settings.Settings method), 295
- ovs_vsctl() (autotest.client.shared.openvswitch.OpenVSwitchControlCli_140 method), 289
- ## P
- pack() (autotest.client.net.net_utils.ethernet static method), 235
- PACKAGE_TYPE (autotest.client.shared.software_manager.DpkgBackend attribute), 296
- PACKAGE_TYPE (autotest.client.shared.software_manager.RpmBackend attribute), 296
- PackageError, 269
- PackageFetchError, 271
- PackageInstallError, 269
- PackageManager (class in autotest.client.shared.packages), 290
- PackageRemoveError, 270
- PackageUploadError, 272
- PackagingError, 271
- parallel() (autotest.client.job.base_client_job method), 212
- parallel() (in module autotest.client.partition), 225
- parallel() (in module autotest.client.shared.utils), 314
- parse() (autotest.client.shared.base_job.status_log_entry class method), 257
- parse() (autotest.client.shared.utils_koji.KojiPkgSpec method), 326
- parse() (autotest.client.shared.utils_koji.KojiScratchPkgSpec method), 326
- parse_args() (autotest.client.cmdparser.CommandParser method), 202
- parse_args() (autotest.client.harness_beaker.harness_beaker method), 209
- parse_cmdline() (autotest.client.autotest_local.AutotestLocalApp method), 193
- parse_conf() (autotest.client.os_dep.Ldconfig method), 220
- parse_config_file() (autotest.client.shared.settings.Settings method), 295
- parse_control() (in module autotest.client.shared.control_data), 265
- parse_entry() (in module autotest.client.tools.boottool), 359
- parse_ethtool() (autotest.client.net.net_utils.network_interface method), 237
- parse_from_file() (autotest.client.bkr_xml.BeakerXMLParser method), 201
- parse_lsmod_for_module() (in module autotest.client.base_utils), 197

parse_mke2fs_conf() (in module autotest.client.fsinfo), 207
 parse_quickcmd() (autotest.client.harness_beaker.harness_beaker method), 209
 parse_results() (in module autotest.client.tools.results2junit), 362
 parse_results() (in module autotest.client.tools.scan_results), 362
 parse_results_dir() (in module autotest.client.shared.report), 292
 parse_ssh_path() (in module autotest.client.shared.base_packages), 262
 parse_tarball_name() (autotest.client.shared.base_packages.BasePackageManager static method), 259
 parse_unified_diff_output() (in module autotest.client.shared.test_utils.config_change_validation), 336
 parse_xml() (autotest.client.bkr_xml.BeakerXMLParser method), 201
 partition (class in autotest.client.partition), 225
 partition() (autotest.client.job.base_client_job method), 212
 partname_to_device() (in module autotest.client.partition), 227
 passed (autotest.client.shared.test.Subtest attribute), 299
 patch() (autotest.client.kernel.kernel method), 214
 patch() (in module autotest.client.shared.mock), 284
 path_exists() (autotest.client.shared.hosts.base_classes.Host method), 334
 path_joiner() (in module autotest.client.os_dep), 222
 perf (class in autotest.clientprofilers.perf.perf), 245
 pfifo (class in autotest.client.net.net_tc), 232
 pickle_dump() (autotest.client.kernel.kernel method), 214
 pickle_load() (in module autotest.client.base_utils), 197
 PICKLE_PROTOCOL (autotest.client.shared.base_job.job_state attribute), 254
 pid_exists() (in module autotest.client.shared.utils), 314
 pid_is_alive() (in module autotest.client.shared.utils), 314
 PidFileManager (class in autotest.client.shared.pidfile), 290
 ping_default_gateway() (in module autotest.client.base_utils), 197
 pkgdir (autotest.client.shared.base_job.base_job attribute), 253
 pop() (autotest.client.shared.backports.collections.OrderedDictOrderedDict method), 330
 pop_execution_context() (autotest.client.shared.base_job.base_job method), 253
 popitem() (autotest.client.shared.backports.collections.OrderedDictOrderedDict method), 330
 port_to_br() (autotest.client.shared.openvswitch.OpenVSwitchController method), 289
 portal_visible() (autotest.client.shared.iscsi.Iscsi method), 274
 postprocess() (autotest.client.shared.test.base_test method), 301
 postprocess_iteration() (autotest.client.shared.test.base_test method), 301
 powertop (class in autotest.client.profilers.powertop.powertop), 245
 prefix (autotest.client.kernel.srpm_kernel attribute), 215
 prefix (autotest.client.kernel.srpm_kernel_suse attribute), 215
 prep() (autotest.client.kernel.srpm_kernel method), 215
 prepare_disks() (in module autotest.client.fsdev_disks), 205
 prepare_fsdev() (in module autotest.client.fsdev_disks), 205
 prepend_path() (in module autotest.client.base_utils), 197
 preprocess_path() (in module autotest.client.kernel), 215
 present() (autotest.client.shared.profiler_manager.profiler_manager method), 291
 preserve_srcdir (autotest.client.profiler.profiler attribute), 228
 preserve_srcdir (autotest.client.profilers.powertop.powertop.powertop attribute), 246
 preserve_srcdir (autotest.client.shared.test.base_test attribute), 301
 print_change_diffs() (in module autotest.client.shared.test_utils.config_change_validation), 336
 print_result() (in module autotest.client.tools.scan_results), 362
 print_to_tty() (in module autotest.client.base_utils), 197
 prio (class in autotest.client.net.net_tc), 232
 Probe (class in autotest.client.shared.distro), 35, 265
 process_failed_constraints() (autotest.client.shared.test.base_test method), 301
 process_is_alive() (in module autotest.client.base_utils), 197
 process_mpstat() (autotest.client.net.net_utils.network_utils method), 237
 process_or_children_is_defunct() (in module autotest.client.shared.utils), 314
 process_output() (autotest.client.shared.utils.BgJob method), 302
 profdir (autotest.client.shared.base_job.base_job attribute), 253
 profiler (class in autotest.client.profiler), 228
 profiler_manager (class in autotest.client.profiler_manager), 291

totest.client.shared.profiler_manager), 291
 ProfilerNotPresentError, 291
 profilers (class in autotest.client.profilers), 239
 program_is_alive() (in module autotest.client.shared.utils), 315
 ProgressBar (class in autotest.client.shared.progressbar), 291
 propertiesType (class in autotest.client.tools.JUnit_api), 348
 property_factory() (autotest.client.shared.base_job.job_director.static method), 254
 property_factory() (autotest.client.shared.base_job.job_state.static method), 255
 PropertyMock (class in autotest.client.shared.mock), 287
 propertyType (class in autotest.client.tools.JUnit_api), 348
 provides() (autotest.client.shared.software_manager.AptBackend method), 295
 provides() (autotest.client.shared.software_manager.YumBackend method), 297
 provides() (autotest.client.shared.software_manager.ZypperBackend method), 298
 push_execution_context() (autotest.client.shared.base_job.base_job method), 253
 PYTHON_BIN_GLOB_STRINGS (autotest.client.shared.base_check_version.base_check_version attribute), 250

Q

qdisc (class in autotest.client.net.net_tc), 232
 quit() (autotest.client.job.base_client_job method), 212

R

rangelist_to_set() (in module autotest.client.cpuset), 204
 raw_socket (class in autotest.client.net.net_utils), 237
 read() (autotest.client.net.net_utils_mock.os_stub method), 239
 read() (autotest.client.shared.iso9660.Iso9660IsoInfo method), 275
 read() (autotest.client.shared.iso9660.Iso9660IsoRead method), 275
 read() (autotest.client.shared.iso9660.Iso9660Mount method), 275
 read_config() (autotest.client.shared.utils_koji.KojiClient method), 324
 read_file() (in module autotest.client.shared.utils), 315
 read_from_file() (autotest.client.shared.base_job.job_state method), 255
 read_from_meminfo() (in module autotest.client.shared.utils_memory), 327
 read_from_numa_maps() (in module autotest.client.shared.utils_memory), 327
 read_from_smaps() (in module autotest.client.shared.utils_memory), 328
 read_from_vmstat() (in module autotest.client.shared.utils_memory), 328
 read_keyval() (in module autotest.client.shared.utils), 315
 read_one_line() (in module autotest.client.shared.utils), 315
 readline() (autotest.client.base_sysinfo.loggable method), 194
 readprofile (class in autotest.client.profilers.readprofile.readprofile), 246
 readval (autotest.client.net.net_utils_mock.os_stub attribute), 239
 reboot() (autotest.client.job.base_client_job method), 212
 reboot() (autotest.client.shared.hosts.base_classes.Host method), 334
 reboot_followup() (autotest.client.shared.hosts.base_classes.Host method), 334
 reboot_setup() (autotest.client.job.base_client_job method), 212
 reboot_setup() (autotest.client.shared.hosts.base_classes.Host method), 334
 Recipe (class in autotest.client.bkr_xml), 201
 recipe_abort() (autotest.client.bkr_proxy.BkrProxy method), 198
 recipe_check_version (autotest.client.bkr_proxy.BkrProxy method), 198
 recipe_upload_file() (autotest.client.bkr_proxy.BkrProxy method), 198
 record() (autotest.client.shared.base_job.base_job method), 253
 record() (autotest.client.shared.base_job.TAPReport method), 250
 record() (autotest.client.shared.hosts.base_classes.Host method), 334
 record() (autotest.client.shared.utils.Statistic method), 303
 record() (in module autotest.client.shared.log), 279
 record_entry() (autotest.client.shared.base_job.base_job method), 253
 record_entry() (autotest.client.shared.base_job.status_logger method), 257
 record_keyval() (autotest.client.shared.base_job.TAPReport method), 250
 recv() (autotest.client.net.net_utils.network_interface method), 237
 recv() (autotest.client.net.net_utils.raw_socket method), 237
 recv() (autotest.client.net.net_utils_mock.socket_stub method), 239
 recv_from() (autotest.client.net.net_utils.raw_socket method), 238
 redirect() (autotest.client.shared.logging_manager.LoggingManager

- method), 281
- redirect_to_stream() (autotest.client.shared.logging_manager.LoggingManager method), 281
- refresh_cgroups() (autotest.client.shared.utils_cgroup.Cgroup method), 320
- regex_comparator (class in autotest.client.shared.test_utils.mock), 339
- register_after_iteration_hook() (autotest.client.shared.test.base_test method), 301
- register_before_iteration_hook() (autotest.client.shared.test.base_test method), 301
- register_probe() (in module autotest.client.shared.distro), 35, 36, 266
- relative_path() (autotest.client.job.base_client_job method), 212
- release() (autotest.client.shared.distro.Probe method), 36, 266
- release_container() (in module autotest.client.cpuset), 204
- remove() (autotest.client.shared.software_manager.AptBackend method), 296
- remove() (autotest.client.shared.software_manager.YumBackend method), 297
- remove() (autotest.client.shared.software_manager.ZypperBackend method), 298
- remove() (autotest.client.test_config.config_loader method), 230
- remove_args() (autotest.client.tools.boottool.Grubby method), 356
- remove_checksum() (autotest.client.shared.base_packages.BasePackageManager method), 259
- remove_empty_prio_classes() (in module autotest.client.cpuset), 204
- remove_global_option() (autotest.client.tools.boottool.EliloConf method), 359
- remove_kernel() (autotest.client.tools.boottool.Grubby method), 356
- remove_pkg() (autotest.client.shared.base_packages.BasePackageManager method), 259
- remove_pkg_file() (autotest.client.shared.base_packages.BasePackageManager method), 259
- remove_repo() (autotest.client.shared.software_manager.AptBackend method), 296
- remove_repo() (autotest.client.shared.software_manager.YumBackend method), 298
- remove_repo() (autotest.client.shared.software_manager.ZypperBackend method), 298
- render() (autotest.client.shared.base_job.status_log_entry method), 257
- render() (autotest.client.shared.jsontemplate.Template method), 278
- render_entry() (autotest.client.shared.base_job.status_logger method), 257
- RENDERED_NONE_VALUE (autotest.client.shared.base_job.status_log_entry attribute), 256
- rendezvous() (autotest.client.shared.base_barrier.barrier method), 249
- rendezvous_servers() (autotest.client.shared.base_barrier.barrier method), 249
- repair_filesystem_only() (autotest.client.shared.hosts.base_classes.Host method), 334
- repair_full() (autotest.client.shared.hosts.base_classes.Host method), 334
- repair_full_disk() (autotest.client.shared.hosts.base_classes.Host method), 334
- repair_software_only() (autotest.client.shared.hosts.base_classes.Host method), 334
- repair_with_protection() (autotest.client.shared.hosts.base_classes.Host method), 334
- repo_check() (autotest.client.shared.base_packages.BasePackageManager method), 259
- repo_run_command() (in module autotest.client.shared.base_packages), 262
- RepoDiskFullError, 269
- RepoError, 271
- report() (autotest.client.profiler.profiler method), 228
- report() (autotest.clientprofilers.blktrace.blktrace.blktrace method), 240
- report() (autotest.clientprofilers.catprofile.catprofile.catprofile method), 240
- report() (autotest.clientprofilers.inotify.inotify.inotify method), 242
- report() (autotest.clientprofilers.iostat.iostat.iostat method), 242
- report() (autotest.clientprofilers.kvm_stat.kvm_stat.kvm_stat method), 243
- report() (autotest.clientprofilers.lockmeter.lockmeter.lockmeter method), 243
- report() (autotest.clientprofilers.mpstat.mpstat.mpstat method), 244
- report() (autotest.clientprofilers.oprofile.oprofile.oprofile method), 245
- report() (autotest.clientprofilers.perf.perf.perf method), 245
- report() (autotest.clientprofilers.powertop.powertop.powertop method), 246
- report() (autotest.clientprofilers.readprofile.readprofile.readprofile method), 246

- report() (autotest.clientprofilers.sar.sar.sar method), 246
- report() (autotest.clientprofilers.systemtap.systemtap.systemtap method), 247
- report() (autotest.clientprofilers.vmstat.vmstat.vmstat method), 247
- report() (autotest.client.shared.profiler_manager.profiler_manager method), 291
- ReportLoggingConfig (class in autotest.client.shared.report), 292
- ReportOptionParser (class in autotest.client.shared.report), 292
- RepositoryFetcher (class in autotest.client.shared.base_packages), 261
- RepoUnknownError, 271
- RepoWriteError, 268
- request_hardware_repair() (autotest.client.shared.hosts.base_classes.Host method), 334
- require_gcc() (autotest.client.job.base_client_job method), 212
- reset() (autotest.client.net.net_utils.network_utils method), 237
- reset_mock() (autotest.client.shared.mock.NonCallableMock method), 286
- reset_values() (autotest.client.shared.settings.Settings method), 295
- resolve_task_cgroup_path() (in module autotest.client.shared.utils_cgroup), 322
- restart() (autotest.client.shared.base_check_version.base_check_version method), 250
- restart() (autotest.client.shared.openswitch.ServiceManagerInterface method), 290
- restart() (autotest.client.shared.openswitch.ServiceManagerSystemCmd method), 290
- restart() (autotest.client.shared.openswitch.ServiceManagerSystem method), 290
- restore() (autotest.client.net.net_tc.classful_qdisc method), 232
- restore() (autotest.client.net.net_tc.qdisc method), 233
- restore() (autotest.client.net.net_tc.tcclass method), 233
- restore() (autotest.client.net.net_tc.tcfilter method), 233
- restore() (autotest.client.net.net_tc.u32filter method), 234
- restore() (autotest.client.net.net_utils.network_interface method), 237
- restore() (autotest.client.shared.logging_manager.LoggingManager method), 281
- restore_disks() (in module autotest.client.fsdev_disks), 205
- result (autotest.client.shared.test.Subtest attribute), 299
- result_to_string() (autotest.client.shared.test.Subtest static method), 299
- result_to_string_debug() (autotest.client.shared.test.Subtest static method), 299
- result_upload_file() (autotest.client.bkr_proxy.BkrProxy method), 198
- resultdir (autotest.client.shared.base_job.base_job attribute), 253
- return_value (autotest.client.shared.mock.NonCallableMock attribute), 286
- rm_cgroup() (autotest.client.shared.utils_cgroup.Cgroup method), 320
- rounded_memtotal() (in module autotest.client.shared.utils_memory), 328
- rpm_kernel (class in autotest.client.kernel), 215
- rpm_kernel_suse (class in autotest.client.kernel), 215
- rpm_kernel_vendor() (in module autotest.client.kernel), 215
- RpmBackend (class in autotest.client.shared.software_manager), 296
- RPMFileNameInfo (class in autotest.client.shared.utils_koji), 327
- run() (autotest.client.base_sysinfo.command method), 194
- run() (autotest.client.base_sysinfo.logfile method), 194
- run() (autotest.client.cmdparser.CommandParser method), 202
- run() (autotest.client.local_host.LocalHost method), 218
- run() (autotest.client.shared.hosts.base_classes.Host method), 334
- run() (autotest.client.shared.test_utils.unittest.ClassTestSuite method), 344
- run() (autotest.client.shared.test_utils.unittest.TestCase method), 344
- run() (autotest.client.shared.test_utils.unittest.TestSuite method), 344
- run() (autotest.client.shared.test_utils.unittest.TextTestRunner method), 345
- run() (autotest.client.shared.utils.FileFieldMonitor.Monitor method), 302
- run() (autotest.client.shared.utils.InterruptedThread method), 303
- run() (autotest.client.shared.utils.run_randomly method), 316
- run() (in module autotest.client.setup), 228
- run() (in module autotest.client.shared.utils), 315
- run_abort() (autotest.client.harness.harness method), 207
- run_abort() (autotest.client.harness_beaker.harness_beaker method), 209
- run_bg() (in module autotest.client.shared.utils), 315
- run_complete() (autotest.client.harness.harness method), 207
- run_complete() (autotest.client.harness_beaker.harness_beaker method), 209
- run_group() (autotest.client.job.base_client_job method), 212
- run_once_profiling() (autotest.client.shared.test.base_test method), 301

- run_original_function() (autotest.client.shared.test_utils.mock.mask_function method), 338
- run_output() (autotest.client.shared.hosts.base_classes.Host method), 335
- run_parallel() (in module autotest.client.shared.utils), 315
- run_pause() (autotest.client.harness.harness method), 207
- run_pause() (autotest.client.harness_beaker.harness_beaker method), 209
- run_randomly (class in autotest.client.shared.utils), 316
- run_reboot() (autotest.client.harness.harness method), 207
- run_reboot() (autotest.client.harness_beaker.harness_beaker method), 209
- run_start() (autotest.client.harness.harness method), 207
- run_start() (autotest.client.harness_autoserv.harness_autoserv method), 208
- run_start() (autotest.client.harness_beaker.harness_beaker method), 209
- run_test() (autotest.client.job.base_client_job method), 212
- run_test() (autotest.client.partition.partition method), 226
- run_test_cleanup (autotest.client.shared.base_job.base_job attribute), 253
- run_test_complete() (autotest.client.harness.harness method), 207
- run_test_complete() (autotest.client.harness_autoserv.harness_autoserv method), 208
- run_test_complete() (autotest.client.harness_beaker.harness_beaker method), 209
- run_test_detail() (autotest.client.job.base_client_job method), 212
- run_test_on_partition() (autotest.client.partition.partition method), 226
- run_test_on_partitions() (in module autotest.client.partition), 227
- runjob() (in module autotest.client.job), 213
- running_config() (in module autotest.client.base_utils), 197
- running_os_full_version() (in module autotest.client.base_utils), 197
- running_os_ident() (in module autotest.client.base_utils), 197
- running_os_release() (in module autotest.client.base_utils), 197
- running_stand_alone_client (autotest.client.shared.settings.Settings attribute), 295
- runsubtest() (autotest.client.shared.test.Subtest method), 299
- runTest() (autotest.client.shared.test_utils.unittest.FunctionTestCase method), 346
- runtest() (in module autotest.client.shared.test), 301
- runtest() (in module autotest.client.test), 229
- ## S
- safe_kill() (in module autotest.client.shared.utils), 316
- safe_rmdir() (in module autotest.client.shared.utils), 316
- Sample (class in autotest.client.tools.regression), 361
- sar (class in autotest.clientprofilers.sar.sar), 246
- save() (autotest.client.test_config.config_loader method), 230
- save() (in module autotest.client.shared.distro_def), 266
- SaveDataAfterCloseStringIO (class in autotest.client.shared.test_utils.mock), 337
- select() (in module autotest.client.harness), 207
- select_kernel_components() (in module autotest.client.kernelexpand), 217
- selinux_enforcing() (in module autotest.client.shared.utils), 316
- send() (autotest.client.net.net_utils.network_interface method), 237
- send() (autotest.client.net.net_utils.raw_socket method), 238
- send() (autotest.client.net.net_utils_mock.socket_stub method), 239
- send() (autotest.client.shared.mail.EmailNotificationManager method), 282
- send() (in module autotest.client.shared.mail), 283
- send_admin() (autotest.client.shared.mail.EmailNotificationManager method), 283
- send_file() (autotest.client.shared.hosts.base_classes.Host method), 335
- send_queued_admin() (autotest.client.shared.mail.EmailNotificationManager method), 283
- send_to() (autotest.client.net.net_utils.raw_socket method), 238
- SEP (autotest.client.shared.utils_koji.KojiPkgSpec attribute), 325
- SEP (autotest.client.shared.utils_koji.KojiScratchPkgSpec attribute), 326
- serialize() (autotest.client.base_sysinfo.base_sysinfo method), 193
- serverdir (autotest.client.shared.base_job.base_job attribute), 253
- service_cgconfig_control() (in module autotest.client.shared.utils_cgroup), 322
- ServiceManager (class in autotest.client.shared.openswitch), 290
- ServiceManager() (in module autotest.client.shared.service), 293
- ServiceManagerInterface (class in autotest.client.shared.openswitch), 290
- ServiceManagerSystemD (class in autotest.client.shared.openswitch), 290

- ServiceManagerSysvinit (class in autotest.client.shared.openvswitch), 290
- SessionData (class in autotest.client.shared.base_syncdata), 263
- set() (autotest.client.config.config method), 202
- set() (autotest.client.shared.base_job.job_state method), 256
- set() (autotest.client.test_config.config_loader method), 230
- set_attr() (autotest.client.shared.control_data.ControlData method), 264
- set_author() (autotest.client.shared.control_data.ControlData method), 264
- set_autodir() (autotest.client.shared.hosts.base_classes.Host method), 335
- set_backing_file() (autotest.client.shared.base_job.job_state method), 256
- set_build_image() (autotest.client.kernel.kernel method), 214
- set_build_target() (autotest.client.kernel.kernel method), 214
- set_cgroup() (autotest.client.shared.utils_cgroup.Cgroup method), 320
- set_classname() (autotest.client.tools.JUnit_api.testcaseTypeset method), 350
- set_config_files() (autotest.client.shared.settings.Settings method), 295
- set_cross_cc() (autotest.client.kernel.kernel method), 214
- set_default() (autotest.client.tools.boottool.Grubby method), 357
- set_default_by_index() (autotest.client.tools.boottool.Grubby method), 357
- set_default_koji_tag() (in module autotest.client.shared.utils_koji), 327
- set_dependencies() (autotest.client.shared.control_data.ControlData method), 264
- set_dest_qdisc() (autotest.client.net.net_tc.tcfilter method), 233
- set_doc() (autotest.client.shared.control_data.ControlData method), 264
- set_error() (autotest.client.tools.JUnit_api.testcaseType method), 350
- set_errors() (autotest.client.tools.JUnit_api.testsuite method), 351
- set_experimental() (autotest.client.shared.control_data.ControlData method), 264
- set_extentiontype_() (autotest.client.tools.JUnit_api.testsuite method), 351
- set_fail_fast() (autotest.client.shared.test_utils.mock.mock method), 338
- set_failure() (autotest.client.tools.JUnit_api.testcaseType method), 350
- set_failures() (autotest.client.tools.JUnit_api.testsuite method), 351
- set_finish() (autotest.client.shared.base_syncdata.SessionData method), 263
- set_fs_options() (autotest.client.partition.partition method), 226
- set_handle() (autotest.client.net.net_tc.tcfilter method), 234
- set_hostname() (autotest.client.tools.JUnit_api.testsuite method), 351
- set_hwaddr() (autotest.client.net.net_utils.network_interface method), 237
- set_id() (autotest.client.tools.JUnit_api.testsuiteType method), 352
- set_io_controls() (in module autotest.client.cpuset), 204
- set_io_scheduler() (autotest.client.partition.partition method), 226
- set_ip_local_port_range() (in module autotest.client.shared.utils), 316
- set_ipaddr() (autotest.client.net.net_utils.network_interface method), 237
- set_leaf_qdisc() (autotest.client.net.net_tc.tcclass method), 233
- set_log_file_dir() (in module autotest.client.shared.utils), 316
- set_message() (autotest.client.tools.JUnit_api.errorType method), 347
- set_message() (autotest.client.tools.JUnit_api.failureType method), 347
- set_module() (autotest.client.shared.mail.EmailNotificationManager method), 283
- set_name() (autotest.client.shared.control_data.ControlData method), 264
- set_name() (autotest.client.tools.JUnit_api.propertyType method), 349
- set_name() (autotest.client.tools.JUnit_api.testcaseType method), 350
- set_name() (autotest.client.tools.JUnit_api.testsuite method), 351
- set_num_huge_pages() (in module autotest.client.shared.utils_memory), 328
- set_only() (autotest.client.shared.profiler_manager.profiler_manager method), 291
- set_package() (autotest.client.tools.JUnit_api.testsuiteType method), 352
- set_parent_class() (autotest.client.net.net_tc.qdisc method), 233
- set_parent_class() (autotest.client.net.net_tc.tcclass method), 233
- set_parent_qdisc() (autotest.client.net.net_tc.tcfilter method), 234
- set_power_state() (in module autotest.client.base_utils),

- 197
- set_priority() (autotest.client.net.net_tc.tcfilter method), 234
- set_priority_class() (autotest.client.shared.utils.VersionableClass class method), 306
- set_properties() (autotest.client.tools.JUnit_api.testsuite method), 351
- set_property() (autotest.client.shared.utils_cgroup.Cgroup method), 320
- set_property() (autotest.client.tools.JUnit_api.propertiesType method), 348
- set_property_h() (autotest.client.shared.utils_cgroup.Cgroup method), 321
- set_protocol() (autotest.client.net.net_tc.tcfilter method), 234
- set_root_cgroup() (autotest.client.shared.utils_cgroup.Cgroup method), 321
- set_run_verify() (autotest.client.shared.control_data.ControlData method), 265
- set_sched_tunables() (autotest.client.fsdev_disks.fsdev_disks method), 204
- set_socket_timeout() (autotest.client.net.net_utils.raw_socket method), 238
- set_state() (autotest.client.shared.base_job.base_job method), 253
- set_sync_count() (autotest.client.shared.control_data.ControlData method), 265
- set_system_err() (autotest.client.tools.JUnit_api.testsuite method), 351
- set_system_out() (autotest.client.tools.JUnit_api.testsuite method), 351
- set_test_category() (autotest.client.shared.control_data.ControlData method), 265
- set_test_class() (autotest.client.shared.control_data.ControlData method), 265
- set_test_parameters() (autotest.client.shared.control_data.ControlData method), 265
- set_test_type() (autotest.client.shared.control_data.ControlData method), 265
- set_testcase() (autotest.client.tools.JUnit_api.testsuite method), 351
- set_tests() (autotest.client.tools.JUnit_api.testsuite method), 351
- set_testsuite() (autotest.client.tools.JUnit_api.testsuites method), 353
- set_time() (autotest.client.shared.control_data.ControlData method), 265
- set_time() (autotest.client.tools.JUnit_api.testcaseType method), 350
- set_time() (autotest.client.tools.JUnit_api.testsuite method), 352
- set_timestamp() (autotest.client.tools.JUnit_api.testsuite method), 352
- set_tunable() (autotest.client.fsdev_disks.fsdev_disks method), 204
- set_type() (autotest.client.tools.JUnit_api.errorType method), 347
- set_type() (autotest.client.tools.JUnit_api.failureType method), 347
- set_value() (autotest.client.tools.JUnit_api.propertyType method), 349
- set_valueOf() (autotest.client.tools.JUnit_api.errorType method), 347
- set_valueOf() (autotest.client.tools.JUnit_api.failureType method), 348
- set_vlanmode() (autotest.client.shared.openvswitch.OpenVSwitchControl method), 288
- set_vlanmode() (autotest.client.shared.openvswitch.OpenVSwitchControl method), 289
- set_wake_alarm() (in module autotest.client.base_utils), 197
- setdefault() (autotest.client.shared.backports.collections.OrderedDict.OrderedDict method), 330
- settimeout() (autotest.client.net.net_utils_mock.socket_stub method), 239
- Settings (class in autotest.client.shared.settings), 294
- SettingsError, 295
- SettingsValueError, 295
- setup() (autotest.client.harness.harness method), 207
- setup() (autotest.client.net.net_tc.classful_qdisc method), 232
- setup() (autotest.client.net.net_tc.netem method), 232
- setup() (autotest.client.net.net_tc.pfifo method), 232
- setup() (autotest.client.net.net_tc.prio method), 232
- setup() (autotest.client.net.net_tc.qdisc method), 233
- setup() (autotest.client.net.net_tc.tcclass method), 233
- setup() (autotest.client.net.net_tc.tcfilter method), 234
- setup() (autotest.client.net.net_tc.u32filter method), 234
- setup() (autotest.client.profiler.profiler.profiler method), 228
- setup() (autotest.clientprofilers.blktrace.blktrace.blktrace method), 240
- setup() (autotest.clientprofilers.ftrace.ftrace.ftrace method), 241
- setup() (autotest.clientprofilers.lockmeter.lockmeter.lockmeter method), 243
- setup() (autotest.clientprofilers.ltng.ltng.ltng method), 244
- setup() (autotest.clientprofilers.oprofile.oprofile.oprofile method), 245
- setup() (autotest.clientprofilers.powertop.powertop.powertop method), 246
- setup() (autotest.clientprofilers.readprofile.readprofile.readprofile method), 246

- setup() (autotest.client.shared.hosts.base_classes.Host method), 335
- setup() (autotest.client.shared.test.base_test method), 301
- setUp() (autotest.client.shared.test_utils.unittest.TestCase method), 346
- setUp() (autotest.client.shared.test_utils.unittest.TestCase method), 344
- setup() (in module autotest.client.setup_modules), 229
- setup_before_test() (autotest.client.partition.partition method), 226
- setup_dep() (autotest.client.job.base_client_job method), 212
- setup_dirs() (autotest.client.job.base_client_job method), 212
- setup_done (autotest.clientprofilers.oprofile.oprofile.oprofile attribute), 245
- setup_job (class in autotest.client.setup_job), 228
- setup_source() (autotest.client.kernel.srpm_kernel method), 215
- setup_source() (autotest.client.kernel.srpm_kernel_suse method), 216
- setup_test() (in module autotest.client.setup_job), 228
- setup_tests() (in module autotest.client.setup_job), 229
- setUpInitSymlink() (autotest.client.harness_beaker.harness_beaker method), 209
- sh_escape() (in module autotest.client.shared.utils), 316
- shadow_file (autotest.client.shared.settings.Settings attribute), 295
- shortDescription() (autotest.client.shared.test_utils.unittest.TestCase method), 344
- shortDescription() (autotest.client.shared.test_utils.unittest.TestCase method), 346
- shortDescription() (autotest.client.shared.test_utils.unittest.TestCase method), 344
- side_effect (autotest.client.shared.mock.NonCallableMock attribute), 286
- signal_pid() (in module autotest.client.shared.utils), 316
- signal_program() (in module autotest.client.shared.utils), 316
- single_sync() (autotest.client.shared.base_syncdata.SyncData method), 263
- site_check_python_version (class in autotest.client.shared.check_version), 264
- site_job (in module autotest.client.job), 213
- site_testdir (autotest.client.shared.base_job.base_job attribute), 254
- SiteFsdevManager (in module autotest.client.fsdev_mgr), 206
- skip() (in module autotest.client.shared.test_utils.unittest), 346
- skipIf() (in module autotest.client.shared.test_utils.unittest), 346
- SkipTest, 346
- skipTest() (autotest.client.shared.test_utils.unittest.TestCase method), 344
- skipUnless() (in module autotest.client.shared.test_utils.unittest), 346
- smoke_test() (autotest.client.shared.utils_cgroup.Cgroup method), 321
- socket() (autotest.client.net.net_utils.raw_socket method), 238
- socket() (autotest.client.net.net_utils_mock.socket_stub method), 239
- socket_stub (class in autotest.client.net.net_utils_mock), 239
- SOCKET_TIMEOUT (autotest.client.net.net_utils.raw_socket attribute), 237
- socket_timeout() (autotest.client.net.net_utils.raw_socket method), 238
- SOFTWARE_COMPONENT_QRY (autotest.client.shared.software_manager.RpmBackend attribute), 296
- software_packages (autotest.client.shared.distro_def.DistroDef attribute), 267
- software_packages_type (autotest.client.shared.distro_def.DistroDef attribute), 267
- SoftwareManager (class in autotest.client.shared.software_manager), 297
- SoftwareManagerLoggingConfig (class in autotest.client.shared.software_manager), 297
- SoftwarePackage (class in autotest.client.shared.distro_def), 267
- SortingLoggingFile (class in autotest.client.shared.logging_manager), 281
- sortTestMethodsUsing() (autotest.client.shared.test_utils.unittest.TestLoader method), 345
- SpecificServiceManager() (in module autotest.client.shared.service), 293
- srpm_kernel (class in autotest.client.kernel), 215
- srpm_kernel_suse (class in autotest.client.kernel), 215
- srpm_kernel_vendor() (in module autotest.client.kernel), 216
- standby() (in module autotest.client.base_utils), 197
- start() (autotest.client.job.disk_usage_monitor method), 213
- start() (autotest.client.net.net_utils.network_utils method), 237
- start() (autotest.client.profiler.profiler method), 228
- start() (autotest.client.profilers.blktrace.blktrace.blktrace method), 240
- start() (autotest.client.profilers.catprofile.catprofile.catprofile method), 240
- start() (autotest.client.profilers.cmdprofile.cmdprofile.cmdprofile

- method), 240
- start() (autotest.clientprofilers.cpistat.cpistat.cpistat method), 241
- start() (autotest.clientprofilers.ftrace.ftrace.ftrace method), 242
- start() (autotest.clientprofilers.inotify.inotify.inotify method), 242
- start() (autotest.clientprofilers.iostat.iostat.iostat method), 242
- start() (autotest.clientprofilers.kvm_stat.kvm_stat.kvm_stat method), 243
- start() (autotest.clientprofilers.lockmeter.lockmeter.lockmeter method), 243
- start() (autotest.clientprofilers.ltng.ltng.ltng method), 244
- start() (autotest.clientprofilers.mpstat.mpstat.mpstat method), 244
- start() (autotest.clientprofilers.oprofile.oprofile.oprofile method), 245
- start() (autotest.clientprofilers.perf.perf.perf method), 245
- start() (autotest.clientprofilers.powertop.powertop.powertop.stdout_level (autotest.client.shared.logging_config.LoggingConfig attribute), 279
- start() (autotest.clientprofilers.readprofile.readprofile.readprofile method), 246
- start() (autotest.clientprofilers.sar.sar.sar method), 247
- start() (autotest.clientprofilers.systemtap.systemtap.systemtap method), 247
- start() (autotest.clientprofilers.vmstat.vmstat.vmstat method), 247
- start() (autotest.client.shared.openswitch.ServiceManagerInterface method), 290
- start() (autotest.client.shared.openswitch.ServiceManagerSystemD method), 290
- start() (autotest.client.shared.openswitch.ServiceManagerSysvinit method), 290
- start() (autotest.client.shared.profiler_manager.profiler_manager method), 291
- start() (autotest.client.shared.utils.FileFieldMonitor method), 302
- start() (autotest.client.shared.utils.SystemLoad method), 304
- start_loggers() (autotest.client.shared.hosts.base_classes.Host method), 335
- start_logging() (autotest.client.shared.logging_manager.FdRedirectionLoggingManager method), 280
- start_logging() (autotest.client.shared.logging_manager.LoggingManager method), 281
- start_ovs_vswitchd() (autotest.client.shared.openswitch.OpenVSwitch method), 287
- start_reboot() (autotest.client.job.base_client_job method), 212
- start_watchdog() (autotest.client.harness_beaker.harness_beaker method), 209
- startTest() (autotest.client.shared.test_utils.unittest.TestResult method), 340
- Statistic (class in autotest.client.shared.utils), 303
- status() (autotest.client.shared.openswitch.OpenVSwitchControl method), 288
- status() (autotest.client.shared.openswitch.OpenVSwitchControlCli_140 method), 289
- status() (autotest.client.shared.openswitch.ServiceManagerInterface method), 290
- status() (autotest.client.shared.openswitch.ServiceManagerSystemD method), 290
- status_indenter (class in autotest.client.job), 213
- status_indenter (class in autotest.client.shared.base_job), 256
- status_log_entry (class in autotest.client.shared.base_job), 256
- status_logger (class in autotest.client.shared.base_job), 257
- stderr_level (autotest.client.shared.logging_config.LoggingConfig attribute), 279
- stdout_level (autotest.client.shared.logging_config.LoggingConfig attribute), 279
- step_engine() (autotest.client.job.base_client_job method), 213
- StepError, 210
- stop() (autotest.client.job.disk_usage_monitor method), 213
- stop() (autotest.client.net.net_utils.network_utils method), 237
- stop() (autotest.client.profiler.profiler method), 228
- stop() (autotest.clientprofilers.blktrace.blktrace.blktrace method), 240
- stop() (autotest.clientprofilers.catprofile.catprofile.catprofile method), 240
- stop() (autotest.clientprofilers.cmdprofile.cmdprofile.cmdprofile method), 240
- stop() (autotest.clientprofilers.cpistat.cpistat.cpistat method), 241
- stop() (autotest.clientprofilers.ftrace.ftrace.ftrace method), 242
- stop() (autotest.clientprofilers.inotify.inotify.inotify method), 242
- stop() (autotest.clientprofilers.iostat.iostat.iostat method), 242
- stop() (autotest.clientprofilers.kvm_stat.kvm_stat.kvm_stat method), 243
- stop() (autotest.clientprofilers.lockmeter.lockmeter.lockmeter method), 243
- stop() (autotest.clientprofilers.ltng.ltng.ltng method), 244
- stop() (autotest.clientprofilers.mpstat.mpstat.mpstat method), 244
- stop() (autotest.clientprofilers.oprofile.oprofile.oprofile method), 245

- method), 245
- stop() (autotest.clientprofilers.perf.perf.perf method), 245
- stop() (autotest.clientprofilers.powertop.powertop.powertopsubclass method), 246
- stop() (autotest.clientprofilers.readprofile.readprofile.readprofilesubclass method), 246
- stop() (autotest.clientprofilers.sar.sar.sar method), 247
- stop() (autotest.clientprofilers.systemtap.systemtap.systemtap method), 247
- stop() (autotest.clientprofilers.vmstat.vmstat.vmstat method), 247
- stop() (autotest.client.shared.openswitch.ServiceManagerInterface method), 290
- stop() (autotest.client.shared.openswitch.ServiceManagerSystemD method), 290
- stop() (autotest.client.shared.openswitch.ServiceManagerSysvinit method), 290
- stop() (autotest.client.shared.profiler_manager.profiler_manager method), 291
- stop() (autotest.client.shared.test_utils.unittest.TestResult method), 340
- stop() (autotest.client.shared.utils.FileFieldMonitor method), 302
- stop() (autotest.client.shared.utils.SystemLoad method), 304
- stop_loggers() (autotest.client.shared.hosts.base_classes.Host method), 335
- stop_logging() (autotest.client.shared.logging_manager.LoggingManager method), 281
- stopTest() (autotest.client.shared.test_utils.unittest.TestResult method), 340
- STREAM_MANAGER_CLASS (autotest.client.shared.logging_manager.FdRedirectionLoggingManager attribute), 280
- STREAM_MANAGER_CLASS (autotest.client.shared.logging_manager.LoggingManager attribute), 280
- string_to_bitlist() (in module autotest.client.shared.utils), 316
- strip_console_codes() (in module autotest.client.shared.utils), 317
- strip_unicode() (in module autotest.client.shared.utils), 317
- stub_class() (autotest.client.shared.test_utils.mock.mock_god method), 339
- stub_class_method() (autotest.client.shared.test_utils.mock.mock_god method), 339
- stub_function() (autotest.client.shared.test_utils.mock.mock_god method), 339
- stub_function_to_return() (autotest.client.shared.test_utils.mock.mock_god method), 339
- stub_with() (autotest.client.shared.test_utils.mock.mock_god method), 339
- StubNotFoundError, 337
- (autotest.client.tools.JUnit_api.errorType attribute), 347
- (autotest.client.tools.JUnit_api.failureType attribute), 348
- (autotest.client.tools.JUnit_api.propertiesType attribute), 348
- (autotest.client.tools.JUnit_api.propertyType attribute), 349
- (autotest.client.tools.JUnit_api.system_err attribute), 349
- (autotest.client.tools.JUnit_api.system_out attribute), 349
- (autotest.client.tools.JUnit_api.testcaseType attribute), 350
- (autotest.client.tools.JUnit_api.testsuite attribute), 352
- (autotest.client.tools.JUnit_api.testsuites attribute), 353
- (autotest.client.tools.JUnit_api.testsuiteType attribute), 352
- Subtest (class in autotest.client.shared.test), 299
- subtest_fatal() (in module autotest.client.shared.test), 301
- subtest_nocleanup() (in module autotest.client.shared.test), 301
- suiteClass (autotest.client.shared.test_utils.unittest.TestLoader attribute), 345
- superclass (autotest.client.tools.JUnit_api.errorType attribute), 347
- superclass (autotest.client.tools.JUnit_api.failureType attribute), 348
- superclass (autotest.client.tools.JUnit_api.propertiesType attribute), 348
- superclass (autotest.client.tools.JUnit_api.propertyType attribute), 349
- superclass (autotest.client.tools.JUnit_api.system_err attribute), 349
- superclass (autotest.client.tools.JUnit_api.system_out attribute), 349
- superclass (autotest.client.tools.JUnit_api.testcaseType attribute), 350
- superclass (autotest.client.tools.JUnit_api.testsuite attribute), 352
- superclass (autotest.client.tools.JUnit_api.testsuites attribute), 353
- superclass (autotest.client.tools.JUnit_api.testsuiteType attribute), 352
- SUPPORTED_BOOTLOADERS (autotest.client.tools.boottool.Grubby attribute), 353
- supports_reboot (autotest.client.profiler.profiler attribute), 228

- supports_reboot (autotest.client.profilers.cmdprofile.cmdprofile attribute), 240
- suspend_to_disk() (in module autotest.client.base_utils), 197
- suspend_to_ram() (in module autotest.client.base_utils), 197
- symlink_closure() (autotest.client.local_host.LocalHost method), 218
- symlink_closure() (autotest.client.shared.hosts.base_classes.Host method), 335
- sync() (autotest.client.shared.base_syncdata.SyncData method), 263
- SyncData (class in autotest.client.shared.base_syncdata), 263
- SyncListenServer (class in autotest.client.shared.base_syncdata), 263
- sys_v_init_command_generator() (in module autotest.client.shared.service), 294
- sys_v_init_result_parser() (in module autotest.client.shared.service), 294
- sysctl() (in module autotest.client.base_utils), 197
- sysctl_kernel() (in module autotest.client.base_utils), 198
- sysrq_reboot() (autotest.client.shared.hosts.base_classes.Host method), 335
- system() (in module autotest.client.shared.utils), 317
- system_err (class in autotest.client.tools.JUnit_api), 349
- system_out (class in autotest.client.tools.JUnit_api), 349
- system_output() (in module autotest.client.shared.utils), 317
- system_output_parallel() (in module autotest.client.shared.utils), 317
- system_parallel() (in module autotest.client.shared.utils), 317
- systemd_command_generator() (in module autotest.client.shared.service), 294
- systemd_result_parser() (in module autotest.client.shared.service), 294
- SystemInspector (class in autotest.client.shared.software_manager), 297
- SystemLoad (class in autotest.client.shared.utils), 303
- systemtap (class in autotest.client.profilers.systemtap.systemtap), 247
- T**
- tag (autotest.client.shared.base_job.base_job attribute), 254
- tap_ok() (autotest.client.shared.base_job.TAPReport class method), 250
- TAPReport (class in autotest.client.shared.base_job), 250
- tar_package() (autotest.client.shared.base_packages.BasePackage method), 259
- Task (class in autotest.client.bkr_xml), 201
- task_cleanup() (autotest.client.bkr_proxy.BkrProxy method), 198
- task_result() (autotest.client.bkr_proxy.BkrProxy method), 198
- task_start() (autotest.client.bkr_proxy.BkrProxy method), 198
- task_stop() (autotest.client.bkr_proxy.BkrProxy method), 198
- task_upload_file() (autotest.client.bkr_proxy.BkrProxy method), 198
- tasks_path() (in module autotest.client.cpuset), 204
- tc_cmd() (autotest.client.net.net_tc.qdisc method), 233
- tc_cmd() (autotest.client.net.net_tc.tcfilter method), 234
- tcclass (class in autotest.client.net.net_tc), 233
- tcfilter (class in autotest.client.net.net_tc), 233
- tear_down() (autotest.client.harness_beaker.harness_beaker method), 209
- tearDown() (autotest.client.shared.test_utils.unittest.FunctionTestCase method), 346
- tearDown() (autotest.client.shared.test_utils.unittest.TestCase method), 344
- tee() (in module autotest.client.tools.regression), 362
- tee_output_logdir_mark() (in module autotest.client.kernel), 216
- tee_redirect() (autotest.client.shared.logging_manager.LoggingManager method), 281
- tee_redirect_debug_dir() (autotest.client.shared.logging_manager.LoggingManager method), 281
- tee_redirect_to_stream() (autotest.client.shared.logging_manager.LoggingManager method), 281
- tempdir (class in autotest.client.shared.autotemp), 248
- TempDir (class in autotest.client.shared.base_syncdata), 263
- tempfile (class in autotest.client.shared.autotemp), 248
- Template (class in autotest.client.shared.jsontemplate), 278
- TemplateSyntaxError, 276
- test (class in autotest.client.test), 229
- test() (autotest.client.shared.magic.MagicTest method), 282
- test() (autotest.client.shared.test.Subtest method), 300
- test() (autotest.client.shared.utils_cgroup.Cgroup method), 321
- test_status() (autotest.client.harness.harness method), 207
- test_status() (autotest.client.harness_autoserv.harness_autoserv method), 208
- test_status() (autotest.client.harness_beaker.harness_beaker method), 209
- test_status() (autotest.client.harness_simple.harness_simple method), 210
- test_status_detail() (autotest.client.harness.harness method), 207

- test_status_detail() (autotest.client.harness_beaker.harness_beaker method), 209
- TestBaseException, 270
- TestCase (class in autotest.client.shared.test_utils.unittest), 340
- testCaseType (class in autotest.client.tools.JUnit_api), 349
- testdir (autotest.client.shared.base_job.base_job attribute), 254
- TestError, 271
- TestFail, 271
- TestingConfig (class in autotest.client.shared.logging_config), 279
- TestLoader (class in autotest.client.shared.test_utils.unittest), 345
- testMethodPrefix (autotest.client.shared.test_utils.unittest.TestLoader attribute), 345
- TestNAError, 270
- TestResult (class in autotest.client.shared.test_utils.unittest), 340
- TestSuite (class in autotest.client.shared.test_utils.unittest), 344
- testsuite (class in autotest.client.tools.JUnit_api), 350
- testsuites (class in autotest.client.tools.JUnit_api), 352
- testsuiteType (class in autotest.client.tools.JUnit_api), 352
- TestWarn, 269
- text_clean() (in module autotest.client.tools.results2junit), 362
- TextTestRunner (class in autotest.client.shared.test_utils.unittest), 344
- thin_lv_create() (in module autotest.client.lv_utils), 219
- timeout() (autotest.client.shared.base_syncdata.SessionData method), 263
- timeout() (autotest.client.shared.base_syncdata.SyncData method), 263
- TIMESTAMP_FIELD (autotest.client.shared.base_job.status_log_entry attribute), 256
- tmpdir (autotest.client.shared.base_job.base_job attribute), 254
- to_seconds() (in module autotest.client.base_utils), 198
- to_text() (autotest.client.shared.utils_koji.KojiPkgSpec method), 326
- tokenstream() (autotest.client.shared.jsontemplate.Template method), 278
- toolsdir (autotest.client.shared.base_job.base_job attribute), 254
- tracing_dir (autotest.clientprofilers.ftrace.ftrace.ftrace attribute), 242
- trim_custom_directories() (in module autotest.client.shared.base_packages), 263
- u32filter (class in autotest.client.net.net_tc), 234
- umount() (in module autotest.client.shared.utils), 317
- UndefinedVariable, 276
- undo_redirect() (autotest.client.shared.logging_manager.FdRedirectionLogger method), 280
- undo_redirect() (autotest.client.shared.logging_manager.LoggingManager method), 281
- UnhandledJobError, 271
- UnhandledTestError, 270
- UnhandledTestFail, 269
- unique() (in module autotest.client.shared.utils), 318
- unique_not_false_list() (in module autotest.client.os_dep), 222
- UNKNOWN_DISTRO (in module autotest.client.shared.distro), 34
- unload_kvm() (in module autotest.client.kvm_control), 218
- unload_module() (in module autotest.client.base_utils), 198
- unlock_file() (in module autotest.client.shared.utils), 318
- unmap_url() (in module autotest.client.shared.utils), 318
- unmap_url_cache() (in module autotest.client.base_utils), 198
- unmock_io() (autotest.client.shared.test_utils.mock.mock_god method), 339
- unmount() (autotest.client.partition.partition method), 226
- unmount_force() (autotest.client.partition.partition method), 227
- unmount_partition() (in module autotest.client.partition), 227
- unpack() (autotest.client.net.net_utils.ethernet static method), 235
- unpath() (in module autotest.client.cpuset), 204
- unstub() (autotest.client.shared.test_utils.mock.mock_god method), 339
- unstub_all() (autotest.client.shared.test_utils.mock.mock_god method), 339
- untar_pkg() (autotest.client.shared.base_packages.BasePackageManager method), 260
- untar_required() (autotest.client.shared.base_packages.BasePackageManager method), 260
- up() (autotest.client.net.net_utils.network_interface method), 237
- update() (autotest.client.shared.backports.collections.OrderedDict.OrderedDict method), 330
- update() (autotest.client.shared.progressbar.ProgressBar method), 292
- update() (autotest.client.tools.boottool.EliloConf method), 359
- update_checksum() (autotest.client.shared.base_packages.BasePackageManager method), 260

update_config() (autotest.client.kernel_config.kernel_config method), 206
 update_config() (autotest.client.kernel_config.kernel_config method), 216
 update_screen() (autotest.client.shared.progressbar.ProgressBar method), 292
 update_spec() (autotest.client.kernel.srpm_kernel method), 215
 update_spec_line() (autotest.client.kernel.srpm_kernel method), 215
 update_spec_line() (autotest.client.kernel.srpm_kernel_suse method), 216
 update_version() (in module autotest.client.shared.utils), 318
 update_watchdog() (autotest.client.bkr_proxy.BkrProxy method), 198
 upgrade() (autotest.client.shared.software_manager.AptBackend method), 296
 upgrade() (autotest.client.shared.software_manager.YumBackend method), 298
 upgrade() (autotest.client.shared.software_manager.ZypperBackend method), 298
 upkeep() (autotest.client.shared.base_packages.BasePackageManager method), 260
 upload_pkg() (autotest.client.shared.base_packages.BasePackageManager method), 260
 upload_pkg_dir() (autotest.client.shared.base_packages.BasePackageManager method), 260
 upload_pkg_file() (autotest.client.shared.base_packages.BasePackageManager method), 260
 upload_pkg_parallel() (autotest.client.shared.base_packages.BasePackageManager method), 260
 upload_recipe_files() (autotest.client.harness_beaker.harness_beaker method), 209
 upload_result_files() (autotest.client.harness_beaker.harness_beaker method), 209
 upload_task_files() (autotest.client.harness_beaker.harness_beaker method), 209
 url (autotest.client.shared.base_packages.RepositoryFetcher attribute), 262
 url_accessible() (in module autotest.client.kernelexpand), 217
 urlopen() (in module autotest.client.shared.utils), 318
 urlretrieve() (in module autotest.client.shared.utils), 318
 usage() (autotest.client.autotest_local.AutotestLocalApp method), 193
 usage() (in module autotest.client.tools.process_metrics), 361
 use_fsdev_lib() (in module autotest.client.fsdev_disks), 205
 use_partition() (autotest.client.fsdev_mgr.BaseFsdevManager method), 206
 use_sequence_number (autotest.client.shared.base_job.base_job attribute), 254
 validate_ISO8601_DATETIME_PATTERN() (autotest.client.tools.JUnit_api.testsuite method), 352
 values() (autotest.client.shared.backports.collections.OrderedDict.OrderedDict method), 330
 verify() (autotest.client.shared.hosts.base_classes.Host method), 335
 verify_connectivity() (autotest.client.shared.hosts.base_classes.Host method), 335
 verify_hardware() (autotest.client.shared.hosts.base_classes.Host method), 335
 verify_running_as_root() (in module autotest.client.shared.utils), 318
 verify_software() (autotest.client.shared.hosts.base_classes.Host method), 335
 version (autotest.clientprofilers.blktrace.blktrace.blktrace attribute), 240
 version (autotest.clientprofilers.catprofile.catprofile.catprofile attribute), 240
 version (autotest.clientprofilers.cmdprofile.cmdprofile.cmdprofile attribute), 240
 version (autotest.clientprofilers.cpistat.cpistat.cpistat attribute), 241
 version (autotest.clientprofilers.ftrace.ftrace.ftrace attribute), 242
 version (autotest.clientprofilers.inotify.inotify.inotify attribute), 242
 version (autotest.clientprofilers.iostat.iostat.iostat attribute), 242
 version (autotest.clientprofilers.kvm_stat.kvm_stat.kvm_stat attribute), 243
 version (autotest.clientprofilers.lockmeter.lockmeter.lockmeter attribute), 243
 version (autotest.clientprofilers.ltng.ltng.ltng attribute), 244
 version (autotest.clientprofilers.mpstat.mpstat.mpstat attribute), 244
 version (autotest.clientprofilers.oprofile.oprofile.oprofile attribute), 245
 version (autotest.clientprofilers.perf.perf.perf attribute), 245
 version (autotest.clientprofilers.powertop.powertop.powertop attribute), 246
 version (autotest.clientprofilers.readprofile.readprofile.readprofile attribute), 246
 version (autotest.clientprofilers.sar.sar.sar attribute), 247

version (autotest.clientprofilers.systemtap.systemtap.systemtap attribute), 247

version (autotest.clientprofilers.vmstat.vmstat.vmstat attribute), 247

version() (autotest.client.shared.distro.Probe method), 36, 266

version_choose_config() (in module autotest.client.kernel_versions), 216

version_choose_config() (in module autotest.client.shared.kernel_versions), 278

version_encode() (in module autotest.client.kernel_versions), 216

version_encode() (in module autotest.client.shared.kernel_versions), 278

version_len() (in module autotest.client.kernel_versions), 216

version_len() (in module autotest.client.shared.kernel_versions), 278

version_limit() (in module autotest.client.kernel_versions), 216

version_limit() (in module autotest.client.shared.kernel_versions), 279

VersionableClass (class in autotest.client.shared.utils), 304

vg_check() (in module autotest.client.lv_utils), 219

vg_list() (in module autotest.client.lv_utils), 219

vg_ramdisk_cleanup() (in module autotest.client.lv_utils), 219

viewitems() (autotest.client.shared.backports.collections.OrderedDict method), 330

viewkeys() (autotest.client.shared.backports.collections.OrderedDict method), 330

viewvalues() (autotest.client.shared.backports.collections.OrderedDict method), 330

virtual_partition (class in autotest.client.partition), 227

vmstat (class in autotest.client.profilers.vmstat.vmstat), 247

W

wait_down() (autotest.client.shared.hosts.base_classes.Host method), 335

WAIT_DOWN_REBOOT_TIMEOUT (autotest.client.shared.hosts.base_classes.Host attribute), 332

WAIT_DOWN_REBOOT_WARNING (autotest.client.shared.hosts.base_classes.Host attribute), 332

wait_for() (autotest.client.shared.utils.AsyncJob method), 302

wait_for() (in module autotest.client.shared.utils), 318

wait_for_carrier() (autotest.client.net.net_utils.network_interface method), 237

wait_for_carrier() (autotest.client.net.net_utils_mock.netif_wait_for_carrier() (autotest.client.net.net_utils_mock.network_interface method), 238

wait_for_restart() (autotest.client.shared.hosts.base_classes.Host method), 335

wait_for_state_change() (autotest.client.net.net_utils.bonding method), 234

wait_up() (autotest.client.local_host.LocalHost method), 218

wait_up() (autotest.client.shared.hosts.base_classes.Host method), 335

warmup() (autotest.client.shared.test.base_test method), 301

wasSuccessful() (autotest.client.shared.test_utils.unittest.TestResult method), 340

watch() (autotest.client.job.disk_usage_monitor class method), 213

watchdog_loop() (autotest.client.harness_beaker.harness_beaker method), 209

wget_cmd_pattern (autotest.client.shared.base_packages.HttpFetcher attribute), 261

where_art_thy_filehandles() (in module autotest.client.base_utils), 198

which() (in module autotest.client.os_dep), 222

which_header() (in module autotest.client.os_dep), 223

which_library() (in module autotest.client.os_dep), 223

wipe() (autotest.client.partition.partition method), 227

wipe_disk() (in module autotest.client.fsdev_disks), 205

wipe_filesystem() (in module autotest.client.partition), 227

with_backing_file() (in module autotest.client.shared.base_job), 257

with_backing_lock() (in module autotest.client.shared.base_job), 257

write() (autotest.client.shared.base_job.TAPReport method), 251

write() (autotest.client.shared.logging_manager.LoggingFile method), 280

write_attr_keyval() (autotest.client.shared.test.base_test method), 301

write_cores() (in module autotest.client.tools.crash_handler), 360

write_html_report() (in module autotest.client.shared.report), 292

write_iteration_keyval() (autotest.client.shared.test.base_test method), 301

write_keyval() (in module autotest.client.shared.utils), 318

write_one_line() (in module autotest.client.shared.utils), 319

write_perf_keyval() (autotest.client.shared.test.base_test method), 301

`write_pid()` (in module `autotest.client.shared.utils`), 319
`write_processed_tests()` (`autotest.client.harness_beaker.harness_beaker` method), 209
`write_test_keyval()` (`autotest.client.shared.test.base_test` method), 301
`write_to_file()` (`autotest.client.shared.base_job.job_state` method), 256
`write_to_file()` (in module `autotest.client.tools.crash_handler`), 360
`writelines()` (`autotest.client.shared.logging_manager.LoggingFile` method), 280

X

`xen` (class in `autotest.client.xen`), 231
`xen()` (`autotest.client.job.base_client_job` method), 213
`xfs_mkfs_options()` (in module `autotest.client.fsinfo`), 207
`xfs_tunables()` (in module `autotest.client.fsinfo`), 207
`xml_attr()` (in module `autotest.client.bkr_xml`), 201
`xml_get_nodes()` (in module `autotest.client.bkr_xml`), 201

Y

`YumBackend` (class in `autotest.client.shared.software_manager`), 297

Z

`ZypperBackend` (class in `autotest.client.shared.software_manager`), 298