reproman Documentation

Release 0.0.1

ReproMan team

Feb 20, 2020

Contents

CHAPTER 1

Acknowledgments

ReproMan development is being performed as part of an NIH funded (1P41EB019936-01A1) “Center for Reproducible Neuroimaging Computation (CRNC)”. It’s initial development aims to provide a suite of tools for management of computational environments, which is the TR&D 3 sub-project of the CRNC, and is lead by Dr. Halchenko.

CHAPTER 2

Concepts and technologies

2.1 Background and motivation

2.1.1 Vision

TODO

2.1.2 Objective

TODO

2.2 Related efforts and solutions

TODO

2.2.1 Related technologies

- AWS (NITRC-CE)
- Docker/Packer
- Rocket (CoreOS)
- NIH Commons computing
- boutiques
- local clusters (CH to meet with UMMS IT)
- Neuroscience Gateway

2.3 Environments management use cases

2.4 Glossary

ReproMan uses terminology which is collated from various technologies. This glossary provides definitions for terms used in the ReproMan documentation and API, and provides additional references where to seek more information

cloud instance TODO

container TODO Docker and Singularity

environment TODO

package TODO

virtual machine TODO

2.5 High-level Package Handling (and ReproZip Architecture Discussion)

2.5.1 What ReproMan aims (not) to be

We want to leverage existing solutions (such as existing containers, cloud providers etc), which we will call 'backends', and provide a very high level, unified API, to interface them with purpose of running computations or interactive sessions.

We want to concentrate on (re)creation of such computation environments from a specification which is agnostic of a backend and concentrates on describing what constitutes the content of that environment relevant for the execution of computation. Backend-specific details of construction, execution and interfacing with the backend should be "templated" (or otherwise parametrized in sufficient detail) so an advanced user could still provide their tune ups). We will not aim at the specification to be OS agnostic, i.e. the package configuration will have terms that are specific to an architecture or distribution.

Construction of such environments would heavily depend on specification of "packages" which contain sufficient information to reconstruct and execute in the environment. Such specifications could be constructed manually, by ReproMan from loose human description, or via automated provenance collection of "shell" command. They also should provide sufficient expressive power to be able to tune them quickly for most common cases (e.g. upgrade from release X to release Y)

2.5.2 Packages, Package Managers, and Distributions

We would like to be able to identify, record, and install various **packages** of software and data. A package is a collection of files, potentially platform specific (in the case of binary packages) or requiring reconstruction (such as compiling applications from source). In addition, installing a package may have dependencies (additional packages required by the initial package to correctly operate).

Packages are installed, removed, and queried through the use of "package managers." There are different package managers for different components of an environment and have slightly different capabilities. For example, "yum" and "apt-get" are used to install binary and source files on a Linux operating system. "pip" provides download and compilation capability for the Python interpreted language, while "conda" is another Python package manager that can supports "virtual environments" (essentially subdirectories) that provide separate parallel Python environments. Different packages provide varying amount of meta-information to identify package a particular file belongs to, or

to gather meta-information identifying that package source so it could be reinstalled later on (e.g. "pip" from a git repository would not store a URL for that repository anywhere to be recovered).

A "distribution" is a set of packages (typically organized with their dependencies). Some distributions (such as Linux distros) are self-sufficient, in a sense that they could be deployed on a bare hardware or as an independent virtualized environment which would require nothing else. Many distributions though allow to mix a number of **origins**, where any package was or could be obtained from. E.g. it is multiple apt sources for Debian-based distributions and "channels" in conda.

Some distributions (such as the ones based on PIP, conda), do require some base environment on top of which they would work. But also might require some minimal set of tools being provided by the base environment. E.g. *conda* -based distribution would probably need nothing but basic shell (core OS dependent), and PIP-based would require Python to be installed. Therefore, there will be a dependency **between** package managers: Operating system packages (yum & apt-get) will need to be installed first, enabling other package managers (pip, conda, npm) to then run and build upon the base packages.

The fundamental challenge of ReproMan's "trace" ability is to identify and record the package managers, distributions, and packages from the files used in an experiment. Then to "create" an environment, ReproMan needs to reinstall the packages from the specification (ideally matching as many properties, such as version, architecture, size, and hash as possible).

Package Management and Environment Configuration

Here we discuss package managers and key distributions that ReproMan should cover (and list other potential package managers to consider)

OS Package Managers

- apt-get (dpkg) - Expected on Debian and Ubuntu Gnu/Linux distributions
- yum (rpm) - Expected on CentOS/RHEL and other Red Hat Gnu/Linux distributions
- snap - Linux packages (with sandboxed execution) - http://snapcraft.io/
  - Snaps may prove difficult for tracing because commands to download and build executibles can be embedded into snap packages

In addition, we should be aware of specific package repositories that will not stand on their own but depend upon specific OS distributions or configurations:

- NeuroDebian - a key source for NeuroImaging Debian/Ubuntu packages
- other PPAs/APT repositories, e.g. for cran

Finally, OS package managers (and related repositories and distributions) are typically used to install the language-specific package managers described in the next section. Therefore, ReproMan "create" will need to install OS packages first, followed by language-specific packages. We may need to allow the ReproMan environment specification to allow the user to order the package installation across multiple package managers to ensure resolution of dependencies.

Language-Related Package Managers

Python

- pip
  - PyPi Package Index: https://pypi.python.org/pypi
- conda

  - Anaconda Science Platform https://www.continuum.io/downloads
  - Conda-Forge https://conda-forge.github.io/

Others

- npm - node.js
- cpan - Perl
- CRAN - R
- brew, linuxbrew, gems - Ruby

Data Package Managers

- DataLad

Environment Configuration

Pretty much in every "computational environment", environment variables are of paramount importance since they instrument invocation and possibly pointers to where components would be located when executed. "Overlay" (Non-OS) packages rely on adjusting (at least) *PATH* env variable so that components they install, possibly overlaying OS-wide installation components, take precedence.

- virtualenv
  - Impacts the configuration of python environment (where execution is happening, custom python, ENV changes)
- modules
  - http://modules.sourceforge.net
  - Commonly used on HPC, which is the way to "extend" a POSIX distribution.
  - We might want to be aware of it (i.e., being able to detect etc), since it could provide at least versioning information which is conventionally specified for every installed "module". It might come handy during *trace* operation.

Provisioners

Provisioners allow you to automatically install software, alter configurations, and maintain files across multiple machines from a central server (or configuration specification). ReproMan may need to both recognize its use to create an environment and may have an opportunity to use any of the following provisioners to recreate an environment:

- ansible
- chef
- puppet
- salt
- fabric

Alternate Installation Approaches

While these are technically not package managers, we may wish to support other avenues for configuring software to be installed. These approaches may be impossible to detect automatically:

- VCS in general (git, git-annex) repositories – we can identify if particular files belong to which repo, where it is available from, what was the revision etc. We will not collect/record the entirety of the configuration (i.e. all the settings from .git/config), but only the information sufficient to reproduce the environment, not necessarily any other possible interaction with a given VCS
- Generic URL download
- File and directory copy, move, and rename
- Execution of specific commands - may be highly dependent upon the environment

NOTE: Packages that would generally be considered "Core OS" packages, could be installed using these alternate approaches

Backends (engine)

- native
- docker
- singularity (could be created from docker container)
- virtualbox
- vagrant
- aws
- chroot/schroot(somewhat Debian specific on my tries)
- more cloud providers? google CE, azure, etc… ?

Engines might need nesting, e.g.

> vagrant > docker aws > docker ssh > singularity

Image

(inspired by docker and singularity?) What represents a state of computation environment in a form which could be shared (natively or through some export mechanism), and/or could be used as a basis for instantiation of multiple instances or derived environments.

- native – none? or in some cases could be a tarball with all relevant pieces (think cde, reprozip)
- docker, singularity – image
- virtualbox – virtual appliance
- vagrant – box (virtualbox appliance with some bells iirc)
- aws – AMI
- **chroot/schroot – also natively doesn't have an 'image' stage unless we** easily enforce it – tarball (or possibly eventually fs/btrfs snapshots etc, would be neat) whatever chroot is bootstrapped!

Instance

- native – none, i.e. there is a singleton instance of the current env
- docker, singularity - container
- virtualbox – VM instance
- vagrant – ???
- aws – instance
- schroot – session (chroot itself doesn't track anything AFAIK)

2.5.3 Perspective "agents/classes"

Distribution

- bootstrap(spec, backend, instance=None) -> instance/image

  initialize (stage 1) which might include batch installation of a number (or all) of necessary packages; usually offloaded to some utility/backend. (e.g. debootstrap into a dir, docker build from basic Dockerfile, initiate aws ami from some image, etc). Should return an "instance" we could work with in "customization" stage

  customize (stage 2) more interactive (or provisioned) which would tune installation by interacting with the environment; so we should provide adapters on how such interaction would happen (e.g., we could establish common mechanism via ssh, so every env in stage1 would then get openssh deployed; but that would not work e.g. for schroot as easily)

  - at the end it should generate backend-appropriate "instance" which could be reused for derived containers?
  - overlay distributions would need an existing 'instance' to operate on

static methods (?) - get_package_url(package, version) -> urls

- find a URL providing the package of a given version. So, when necessary we could download/install those packages
- get_distribution_spec_from_package_list({package: version_spec}) -> spec
  - given a set of desired packages (with version specs), figure out distribution specification which would satisfy the specification. E.g. to determine which snapshot (which codename, date, components) in snapshots.d.o would carry specified packages

if instance would come out something completely agnostic of the distribution # since instance could actually "contain" multiple distributions. # Possibly tricky part is e.g. all APT "Distributions" would share invocation # – apt, although could (via temporarily augmenting pin priorities) tune it # to consider only its part of the distribution for installation... not sure # if needed - install(instance, package(s)) - uinstall(instance, package(s)) - upgrade(instance)

Probably not here but in instance...? and not now

- **activate() - for those which require changing of ENV. If we are to allow** specification of multiple commands where some aren't using the specific "distribution" we might want to spec which envs to be used and turn them on/off for specific commands
- deactivate()

Image

to be created by bootstrap or "exported" from instance (e.g. "docker commit" to create an image)

- shrink(spec=None) -> image
  - given a specification (or just some generic cleaning operations) we might want to produce a derived image which would be

??? not clear how image/instance would play out when deploying to e.g. HPC. E.g. having a docker/singularity image, and then running some task which would require instantiating that image for every job... condor has some builtin support already IIRC for deploying virtual machine images to run the tasks etc... familiarize more

Instance (bootstrapped, backend specific)

(many commands inspired by docker?)

- run(command) -> instantiate (possibly new container) environment and run a command
- exec(command) -> run a command in running env
- start(id)
- stop(id)

or it would be the resource (AWS, docker, remote HPC) which would be capable of deploying Instances

Backend

???

- should provide mapping from core Distributions specs to native base images (e.g. how to get base docker image for specific release of debian/ubuntu, ...; which AMIs to use as base, etc)
- we should provide default Core Distributions for case if we have a spec only with "overlay" distros (e.g. conda-based)
- bootstrap??

Resource

- instantiate (image, ...) -> instance(s)
  - obtain instance and make it available for execution on the resource
  - some are deployed since were bootstrapped on the resource, but we want to be able to deploy new docker image,
  - deployment might result in multiple instances being deployed (master + slaves for AWS orchestrated execution or is that at run stage... learn more)

(Possibly naive) questions/TODOs

- AMI – could be generated by taking a "snapshot" of existing/running or shutdown instance?

  if not – we might want to provide a mode where initial "investigation" is done locally on a running e.g. docker instance, then script generated for customization stage and only then full bootstrap (using one of the available tools for AMI provisioning) is used

- docker – could we export/import an image to get to the same state (possibly loosing overlays etc)
- singularity – the same

Next ones are more in realm of "exec" or "run" aspect which this discussion is not concentrating on ATM:

- anyone played with StarCluster/ElastiCluster?
- we should familiarize ourselves with built-in features of common PBS systems (condor, torque) to schedule jobs which run within containers...

Possibly useful modules/tools

distro-info python module for Debian/Ubuntu information about releases. uses data from *distro-info-data*

CHAPTER 3

Commands and API

3.1 Command line reference

3.1.1 Main command

reproman

Synopsis

```
reproman [-h] [-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}]
       [--version] [--dbg] [--idbg] [-C PATH] [-c CONFIG]
       {create,install,delete,start,stop,login,execute,run,ls,jobs,backend-parameters,
↪retrace,diff,test}
       ...
```

Description

ReproMan aims to ease construction and execution of computation environments based on collected provenance data.

Commands for manipulating computation environments

- create: Create a computation environment
- install: Install packages according to the provided specification(s)
- delete: Delete a computation environment
- start: Start a computation environment
- stop: Stop a computation environment
- login: Log into a computation environment
- execute: Execute a command in a computation environment

- run: Run a command on the specified resource

Miscellaneous commands

- ls: List known computation resources, images and environments
- jobs: View and manage *reproman run* jobs
- backend-parameters: Display available backend parameters
- retrace: Gather detailed package information from paths or a ReproZip trace file
- diff: Report if a specification satisfies the requirements in another
- test: Run internal ReproMan (unit)tests

General information

Detailed usage information for individual commands is available via command-specific –help, i.e.: reproman <command> –help

Options

{create,install,delete,start,stop,login,execute,run,ls,jobs,backend-parameters,retrace,diff,test}

-h, –help, –help-np

show this help message and exit. –help-np forcefully disables the use of a pager for displaying the help message

-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}, –log-level {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}

level of verbosity. Integers provide even more debugging information

–version

show the program's version and license information and exit

–dbg

enter Python debugger when uncaught exception happens

–idbg

enter IPython debugger when uncaught exception happens

-C PATH

run as if reproman were started in <path> instead of the current working directory. When multiple -C options are given, each subsequent non-absolute -C <path> is interpreted relative to the preceding -C <path>. This option affects the interpretations of the path names in that they are made relative to the working directory caused by the -C option

-c CONFIG, –config CONFIG

path to ReproMan configuration file. This option can be given multiple times, in which case values in the later files override previous ones.

“Reproducibly Manage Your Environments”

Authors

reproman is developed by The ReproMan Team and Contributors <team@reproman.org>.

3.1.2 Environments operations

reproman-ls

Synopsis

```
reproman-ls [--version] [-h]
          [-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}] [-v]
          [-r]
          [RESOURCE [RESOURCE ...]]
```

Description

List known computation resources, images and environments

Examples

$ reproman ls

Options

RESOURCE

Restrict the output to this resource name or ID. [Default: None]

–version

show the program’s version and license information and exit

-h, –help, –help-np

show this help message and exit. –help-np forcefully disables the use of a pager for displaying the help message

-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}, –log-level {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}

level of verbosity. Integers provide even more debugging information

-v, –verbose

provide more verbose listing. [Default: False]

-r, –refresh

Refresh the status of the resources listed. [Default: False]

Authors

reproman is developed by The ReproMan Team and Contributors <team@reproman.org>.

reproman-create

Synopsis

```
reproman-create [--version] [-h]
                [-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}]
                [-t RESOURCE_TYPE] [-b PARAM [PARAM ...]]
                NAME
```

Description

Create a computation environment

Options

NAME

Name of the resource to create. Constraints: value must be a string

–version

show the program's version and license information and exit

-h, –help, –help-np

show this help message and exit. –help-np forcefully disables the use of a pager for displaying the help message

-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}, –log-level {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}

level of verbosity. Integers provide even more debugging information

-t *RESOURCE_TYPE*, –resource-type *RESOURCE_TYPE*

Resource type to create. Constraints: value must be a string

-b PARAM [PARAM . . .], –backend-parameters PARAM [PARAM . . .]

One or more backend parameters in the form KEY=VALUE. Use the command *reproman backend-parameters* to see the list of available backend parameters.

Authors

reproman is developed by The ReproMan Team and Contributors <team@reproman.org>.

reproman-install

Synopsis

```
reproman-install [--version] [-h]
                 [-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}]
                 [--resref-type TYPE]
                 RESOURCE SPEC [SPEC ...]
```

Description

Install packages according to the provided specification(s)

Examples

$ reproman install docker recipe_for_failure.yml

Options

RESOURCE

Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string

SPEC

file with specifications (in supported formats) of packages used in executed environment. Constraints: value must be a string

–version

show the program's version and license information and exit

-h, –help, –help-np

show this help message and exit. –help-np forcefully disables the use of a pager for displaying the help message

-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}, –log-level {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}

level of verbosity. Integers provide even more debugging information

–resref-type TYPE

A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id') [Default: 'auto']

Authors

reproman is developed by The ReproMan Team and Contributors <team@reproman.org>.

reproman-delete

Synopsis

```
reproman-delete [--version] [-h]
                [-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}]
                [--resref-type TYPE] [-y] [-f]
                RESOURCE
```

Description

Delete a computation environment

Examples

$ reproman delete my-resource

Options

RESOURCE

Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string

–version

show the program's version and license information and exit

-h, –help, –help-np

show this help message and exit. –help-np forcefully disables the use of a pager for displaying the help message

-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}, –log-level {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}

level of verbosity. Integers provide even more debugging information

–resref-type TYPE

A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id') [Default: 'auto']

-y, –skip-confirmation

Delete resource without prompting user for confirmation. [Default: False]

-f, –force

Remove a resource from the local inventory regardless of connection errors. Use with caution!. [Default: False]

Authors

reproman is developed by The ReproMan Team and Contributors <team@reproman.org>.

reproman-start

Synopsis

```
reproman-start [--version] [-h]
               [-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}]
               [--resref-type TYPE]
               RESOURCE
```

Description

Start a computation environment

Examples

$ reproman start my-resource

Options

RESOURCE

Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string

–version

show the program's version and license information and exit

-h, –help, –help-np

show this help message and exit. –help-np forcefully disables the use of a pager for displaying the help message

-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}, –log-level {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}

level of verbosity. Integers provide even more debugging information

–resref-type TYPE

A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id') [Default: 'auto']

Authors

reproman is developed by The ReproMan Team and Contributors <team@reproman.org>.

reproman-stop

Synopsis

```
reproman-stop [--version] [-h]
              [-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}]
              [--resref-type TYPE]
              RESOURCE
```

Description

Stop a computation environment

Examples

$ reproman stop my-resource

Options

RESOURCE

Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string

–version

show the program's version and license information and exit

-h, –help, –help-np

show this help message and exit. –help-np forcefully disables the use of a pager for displaying the help message

-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}, –log-level {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}

level of verbosity. Integers provide even more debugging information

–resref-type TYPE

A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id') [Default: 'auto']

Authors

reproman is developed by The ReproMan Team and Contributors <team@reproman.org>.

reproman-login

Synopsis

```
reproman-login [--version] [-h]
               [-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}]
               [--resref-type TYPE]
               RESOURCE
```

Description

Log into a computation environment

Examples

$ reproman login my-resource

Options

RESOURCE

Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string

–version

show the program's version and license information and exit

-h, –help, –help-np

show this help message and exit. –help-np forcefully disables the use of a pager for displaying the help message

-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}, –log-level {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}

level of verbosity. Integers provide even more debugging information

–resref-type TYPE

A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id') [Default: 'auto']

Authors

reproman is developed by The ReproMan Team and Contributors <team@reproman.org>.

3.1.3 Miscellaneous commands

reproman-retrace

Synopsis

```
reproman-retrace [--version] [-h]
                 [-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}]
                 [--spec SPEC] [-o output_file] [-r RESOURCE]
                 [--resref-type TYPE]
                 [PATH [PATH ...]]
```

Description

Gather detailed package information from paths or a ReproZip trace file.

Examples

$ reproman retrace –spec reprozip_run.yml > reproman_config.yml

Options

PATH

path(s) to be traced. If spec is provided, would trace them after tracing the spec. Constraints: value must be a string [Default: None]

–version

show the program's version and license information and exit

-h, –help, –help-np

show this help message and exit. –help-np forcefully disables the use of a pager for displaying the help message

-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}, –log-level {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}

level of verbosity. Integers provide even more debugging information

–spec SPEC

ReproZip YML file to be analyzed. Constraints: value must be a string [Default: None]

-o output_file, –output-file output_file

Output file. If not specified - printed to stdout. Constraints: value must be a string [Default: None]

-r RESOURCE, –resource RESOURCE

Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string [Default: None]

–resref-type TYPE

A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id') [Default: 'auto']

Authors

reproman is developed by The ReproMan Team and Contributors <team@reproman.org>.

reproman-test

Synopsis

```
reproman-test [--version] [-h]
              [-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}]
```

Description

Run internal ReproMan (unit)tests.

This can be used to verify correct operation on the system

Options

–version

show the program's version and license information and exit

-h, –help, –help-np

show this help message and exit. –help-np forcefully disables the use of a pager for displaying the help message

-l {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}, –log-level {critical,error,warning,info,debug,1,2,3,4,5,6,7,8,9}

level of verbosity. Integers provide even more debugging information

Authors

reproman is developed by The ReproMan Team and Contributors <team@reproman.org>.

3.2 Python module reference

This module reference extends the manual with a comprehensive overview of the available functionality built into reproman. Each module in the package is documented by a general summary of its purpose and the list of classes and functions it provides.

3.2.1 High-level user interface

api	Python ReproMan API exposing user-oriented commands (also available via CLI)

api

Python ReproMan API exposing user-oriented commands (also available via CLI)

backend_parameters

reproman.api.**backend_parameters**(*backends=None*)
Display available backend parameters.

create

reproman.api.**create**(*name*, *resource_type*, *backend_parameters*)
Create a computation environment

Parameters

- **name** (*str*) – Name of the resource to create. Constraints: value must be a string.
- **resource_type** (*str*) – Resource type to create. Constraints: value must be a string.
- **backend_parameters** – One or more backend parameters in the form KEY=VALUE. Use the command *reproman backend-parameters* to see the list of available backend parameters.

delete

reproman.api.**delete**(*resref*, *resref_type='auto'*, *skip_confirmation=False*, *force=False*)
Delete a computation environment

Examples

$ reproman delete my-resource

Parameters

- **resref** (*str or None*) – Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string, or value must be *None*.

- **resref_type** (*{auto, name, id}, optional*) – A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id'). [Default: 'auto']
- **skip_confirmation** (*bool, optional*) – Delete resource without prompting user for confirmation. [Default: False]
- **force** (*bool, optional*) – Remove a resource from the local inventory regardless of connection errors. Use with caution!. [Default: False]

diff

reproman.api.**diff**(*prov1*, *prov2*, *satisfies*)

Report if a specification satisfies the requirements in another specification

Examples

$ reproman diff environment1.yml environment2.yml

Parameters

- **prov1** (*str*) – ReproMan provenance file. Constraints: value must be a string.
- **prov2** (*str*) – ReproMan provenance file. Constraints: value must be a string.
- **satisfies** (*bool*) – Make sure the first environment satisfies the needs of the second environment.

execute

reproman.api.**execute**(*command*, *args*, *resref=None*, *resref_type='auto'*, *internal=False*, *trace=False*)

Execute a command in a computation environment

Examples

$ reproman execute mkdir /home/blah/data

Parameters

- **command** (*str*) – name of the command to run. Constraints: value must be a string.
- **args** (*str*) – list of positional and keyword args to pass to the command. Constraints: list expected, each value must be a string.
- **resref** (*str or None, optional*) – Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string, or value must be *None*. [Default: None]
- **resref_type** (*{auto, name, id}, optional*) – A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id'). [Default: 'auto']

- **internal** (*bool, optional*) – Instead of running a generic/any command, execute the internal ReproMan command available within sessions. Known are: mkdir, isdir, put, get, chown, chmod. [Default: False]
- **trace** (*bool, optional*) – if set, trace execution within the environment. [Default: False]

install

reproman.api.**install**(*resref*, *spec*, *resref_type='auto'*)

Install packages according to the provided specification(s)

Examples

$ reproman install docker recipe_for_failure.yml

Parameters

- **resref** (*str or None*) – Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string, or value must be *None*.
- **spec** (*str*) – file with specifications (in supported formats) of packages used in executed environment. Constraints: list expected, each value must be a string.
- **resref_type** (*{auto, name, id}, optional*) – A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id'). [Default: 'auto']

jobs

reproman.api.**jobs**(*queries*, *action='auto'*, *all_=False*, *status=False*)

View and manage *reproman run* jobs.

The possible actions are

- list: Display a oneline list of all registered jobs
- show: Display more information for each job over multiple lines
- delete: Unregister a job locally
- fetch: Fetch a completed job
- auto: If jobs are specified (via JOB or –all), behave like 'fetch'. Otherwise, behave like 'list'.

Parameters

- **queries** – A full job ID or a unique substring.
- **action** (*{auto, list, show, delete, fetch}, optional*) – Operation to perform on the job(s). Constraints: value must be one of ('auto', 'list', 'show', 'delete', 'fetch'). [Default: 'auto']
- **all** (*bool, optional*) – Operate on all jobs. [Default: False]
- **status** (*bool, optional*) – Query the resource for status information when listing or showing jobs. [Default: False]

login

reproman.api.**login**(*resref*, *resref_type='auto'*)
Log into a computation environment

Examples

$ reproman login my-resource

Parameters

- **resref** (*str or None*) – Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string, or value must be *None*.
- **resref_type** (*{auto, name, id}, optional*) – A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id'). [Default: 'auto']

ls

reproman.api.**ls**(*resrefs=None*, *verbose=False*, *refresh=False*)
List known computation resources, images and environments

Examples

$ reproman ls

Parameters

- **resrefs** – Restrict the output to this resource name or ID. [Default: None]
- **verbose** (*bool, optional*) – provide more verbose listing. [Default: False]
- **refresh** (*bool, optional*) – Refresh the status of the resources listed. [Default: False]

retrace

reproman.api.**retrace**(*path=None*, *spec=None*, *output_file=None*, *resref=None*, *resref_type='auto'*)
Gather detailed package information from paths or a ReproZip trace file.

Examples

$ reproman retrace –spec reprozip_run.yml > reproman_config.yml

Parameters

- **path** (*str or None, optional*) – path(s) to be traced. If spec is provided, would trace them after tracing the spec. Constraints: list expected, each value must be a string, or value must be *None*. [Default: None]
- **spec** (*str or None, optional*) – ReproZip YML file to be analyzed. Constraints: value must be a string, or value must be *None*. [Default: None]

- **output_file** (*str or None, optional*) – Output file. If not specified - printed to stdout. Constraints: value must be a string, or value must be *None*. [Default: None]
- **resref** (*str or None, optional*) – Name or ID of the resource to operate on. To see available resources, run 'reproman ls'.Note: As a special case, a session instance can be passed as the value for *resref*. . Constraints: value must be a string, or value must be *None*. [Default: None]
- **resref_type** (*{auto, name, id}, optional*) – A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id'). [Default: 'auto']

run

reproman.api.**run**(*command=None*, *message=None*, *resref=None*, *resref_type='auto'*, *list_=None*, *submitter=None*, *orchestrator=None*, *batch_spec=None*, *batch_parameters=None*, *job_specs=None*, *job_parameters=None*, *inputs=None*, *outputs=None*, *follow=False*)

Run a command on the specified resource.

Two main options control how the job is executed: the orchestator and the submitter. The orchestrator that is selected controls details like how the data is made available on the resource and how the results are fetched. The submitter controls how the job is submitted on the resource (e.g., as a condor job). Use –list to see information on the available orchestrators and submitters.

Unless –follow is specified, the job is started and detached. Use *reproman jobs* to list and fetch detached jobs.

Parameters

- **command** – command for execution. [Default: None]
- **message** – Message to use when saving the run. The details depend on the orchestator, but in general this message will be used in the commit message. [Default: None]
- **resref** (*str or None, optional*) – Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string, or value must be *None*. [Default: None]
- **resref_type** (*{auto, name, id}, optional*) – A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id'). [Default: 'auto']
- **list** – Show available submitters, orchestrators, or job parameters. If an empty string is given, show all. [Default: None]
- **submitter** – Name of submitter. The submitter controls how the command should be submitted on the resource (e.g., with *condor_submit*). [Default: None]
- **orchestrator** – Name of orchestrator. The orchestrator performs pre- and post- command steps like setting up the directory for command execution and storing the results. [Default: None]
- **batch_spec** – YAML file that defines a series of records with parameters for commands. A command will be constructed for each record, with record values available in the command as well as the inputs and outputs as *{p[KEY]}*. See *batch_parameters* for an alternative method for simple combinations. [Default: None]

- **batch_parameters** – Define batch parameters with 'KEY=val1,val2,...'. Different keys can be specified by giving multiple values, in which case the product of the values are taken. For example, 'subj=mei,satsuki' and 'day=1,2' would expand to four records, pairing each subj with each day. Values can be a glob pattern to match against the current working directory. See *batch_spec* for specifying more complex records. [Default: None]
- **job_specs** – YAML files that define job parameters. Multiple paths can be given. If a parameter is defined in multiple specs, the value from the last path that defines it is used. [Default: None]
- **job_parameters** – A job parameter in the form KEY=VALUE. If the same parameter is defined via a job spec, the value given here takes precedence. The values are available as fields in the templates used to generate both the run script and submission script. [Default: None]
- **inputs** – An input file to the command. How input files are used depends on the orchestrator, but, at the very least, the orchestrator should try to make these paths available on the resource. [Default: None]
- **outputs** – An output file to the command. How output files are handled depends on the orchestrator. [Default: None]
- **follow** (*bool, optional*) – Continue to follow the submitted command instead of submitting it and detaching. [Default: False]

start

reproman.api.**start**(*resref*, *resref_type='auto'*)

Start a computation environment

Examples

$ reproman start my-resource

Parameters

- **resref** (*str or None*) – Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string, or value must be *None*.
- **resref_type** (*{auto, name, id}, optional*) – A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id'). [Default: 'auto']

stop

reproman.api.**stop**(*resref*, *resref_type='auto'*)

Stop a computation environment

Examples

$ reproman stop my-resource

Parameters

- **resref** (*str or None*) – Name or ID of the resource to operate on. To see available resources, run 'reproman ls'. Constraints: value must be a string, or value must be *None*.
- **resref_type** (*{auto, name, id}, optional*) – A resource can be referenced by its name or ID. In the unlikely case that a name collides with an ID, explicitly specify 'name' or 'id' to disambiguate. Constraints: value must be one of ('auto', 'name', 'id'). [Default: 'auto']

test

reproman.api.**test**()
Run internal ReproMan (unit)tests.

This can be used to verify correct operation on the system

3.2.2 Plumbing

cmd	Wrapper for command and function calls, allowing for dry runs and output handling
consts	reproman constants
log	
utils	
version	Defines version to be imported in the module and obtained from setup.py
support.configparserinc	

reproman.cmd

Wrapper for command and function calls, allowing for dry runs and output handling

class reproman.cmd.**GitRunner**(*cwd=None*, *env=None*, *protocol=None*)
Bases: *reproman.cmd.Runner*

Runner to be used to run git and git annex commands

Overloads the runner class to check & update GIT_DIR and GIT_WORK_TREE environment variables set to the absolute path if is defined and is relative path

static get_git_environ_adjusted(*env=None*)
Replaces GIT_DIR and GIT_WORK_TREE with absolute paths if relative path and defined

run(*cmd*, *env=None*, **args*, ***kwargs*)
Runs the command *cmd* using shell.

In case of dry-mode *cmd* is just added to *commands* and it is actually executed otherwise. Allows for separately logging stdout and stderr or streaming it to system's stdout or stderr respectively.

Note: Using a string as *cmd* and shell=True allows for piping, multiple commands, etc., but that implies shlex.split() is not used. This is considered to be a security hazard. So be careful with input.

Parameters

- **cmd** (*str, list*) – String (or list) defining the command call. No shell is used if cmd is specified as a list

- **log_stdout** (*bool, optional*) – If True, stdout is logged. Goes to sys.stdout otherwise.
- **log_stderr** (*bool, optional*) – If True, stderr is logged. Goes to sys.stderr otherwise.
- **log_online** (*bool, optional*) – Either to log as output comes in. Setting to True is preferable for running user-invoked actions to provide timely output
- **expect_stderr** (*bool, optional*) – Normally, having stderr output is a signal of a problem and thus it gets logged at ERROR level. But some utilities, e.g. wget, use stderr for their progress output. Whenever such output is expected, set it to True and output will be logged at DEBUG level unless exit status is non-0 (in non-online mode only, in online – would log at DEBUG)
- **expect_fail** (*bool, optional*) – Normally, if command exits with non-0 status, it is considered an ERROR and logged accordingly. But if the call intended for checking routine, such alarming message should not be logged as ERROR, thus it will be logged at DEBUG level.
- **cwd** (*string, optional*) – Directory under which run the command (passed to Popen)
- **env** (*string, optional*) – Custom environment to pass
- **shell** (*bool, optional*) – Run command in a shell. If not specified, then it runs in a shell only if command is specified as a string (not a list)

Returns

Return type (stdout, stderr)

Raises `CommandError` – if command's exitcode wasn't 0 or None. exitcode is passed to CommandError's *code*-field. Command's stdout and stderr are stored in CommandError's *stdout* and *stderr* fields respectively.

class reproman.cmd.**Runner**(*cwd=None*, *env=None*, *protocol=None*)

Bases: object

Provides a wrapper for calling functions and commands.

An object of this class provides a methods that calls shell commands or python functions, allowing for protocolling the calls and output handling.

Outputs (stdout and stderr) can be either logged or streamed to system's stdout/stderr during execution. This can be enabled or disabled for both of them independently. Additionally, a protocol object can be a used with the Runner. Such a protocol has to implement reproman.support.protocol.ProtocolInterface, is able to record calls and allows for dry runs.

call(*f*, **args*, ***kwargs*)

Helper to unify collection of logging all "dry" actions.

Calls *f* if *Runner*-object is not in dry-mode. Adds *f* along with its arguments to *commands* otherwise.

f : callable

****args*, **kwargs*:** Callable arguments

commands

cwd

dry

env

log(*msg*, *level=10*)
log helper

Logs at DEBUG-level by default and adds "Protocol:"-prefix in order to log the used protocol.

protocol

run(*cmd*, *log_stdout=True*, *log_stderr=True*, *log_online=False*, *expect_stderr=False*, *expect_fail=False*, *cwd=None*, *env=None*, *shell=None*)
Runs the command *cmd* using shell.

In case of dry-mode *cmd* is just added to *commands* and it is actually executed otherwise. Allows for separately logging stdout and stderr or streaming it to system's stdout or stderr respectively.

Note: Using a string as *cmd* and shell=True allows for piping, multiple commands, etc., but that implies shlex.split() is not used. This is considered to be a security hazard. So be careful with input.

Parameters

- **cmd** (*str, list*) – String (or list) defining the command call. No shell is used if cmd is specified as a list
- **log_stdout** (*bool, optional*) – If True, stdout is logged. Goes to sys.stdout otherwise.
- **log_stderr** (*bool, optional*) – If True, stderr is logged. Goes to sys.stderr otherwise.
- **log_online** (*bool, optional*) – Either to log as output comes in. Setting to True is preferable for running user-invoked actions to provide timely output
- **expect_stderr** (*bool, optional*) – Normally, having stderr output is a signal of a problem and thus it gets logged at ERROR level. But some utilities, e.g. wget, use stderr for their progress output. Whenever such output is expected, set it to True and output will be logged at DEBUG level unless exit status is non-0 (in non-online mode only, in online – would log at DEBUG)
- **expect_fail** (*bool, optional*) – Normally, if command exits with non-0 status, it is considered an ERROR and logged accordingly. But if the call intended for checking routine, such alarming message should not be logged as ERROR, thus it will be logged at DEBUG level.
- **cwd** (*string, optional*) – Directory under which run the command (passed to Popen)
- **env** (*string, optional*) – Custom environment to pass
- **shell** (*bool, optional*) – Run command in a shell. If not specified, then it runs in a shell only if command is specified as a string (not a list)

Returns

Return type (stdout, stderr)

Raises CommandError – if command's exitcode wasn't 0 or None. exitcode is passed to CommandError's *code*-field. Command's stdout and stderr are stored in CommandError's *stdout* and *stderr* fields respectively.

reproman.cmd.**get_runner**(**args*, ***kwargs*)

reproman.cmd.**link_file_load**(*src*, *dst*, *dry_run=False*)
Just a little helper to hardlink files's load

reproman.consts

reproman constants

reproman.log

class reproman.log.**ColorFormatter**(*use_color=None*, *log_name=False*, *log_pid=False*)
Bases: logging.Formatter

BLACK = 0

BLUE = 4

BOLD_SEQ = '\x1b[1m'

COLORS = {'CRITICAL': 3, 'DEBUG': 4, 'ERROR': 1, 'INFO': 7, 'WARNING': 3}

COLOR_SEQ = '\x1b[1;%dm'

CYAN = 6

GREEN = 2

MAGENTA = 5

RED = 1

RESET_SEQ = '\x1b[0m'

WHITE = 7

YELLOW = 3

format(*record*)
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

formatter_msg(*fmt*, *use_color=False*)

reproman.utils

class reproman.utils.**HashableDict**
Bases: dict

Dict that can be used as keys

class reproman.utils.**PathRoot**(*predicate*)
Bases: object

Find the root of paths based on a predicate function.

The path -> root mapping is cached across calls.

Parameters **predicate** (*callable*) – A callable that will be passed a path and should return true if that path should be considered a root.

class reproman.utils.**SemanticVersion**(*major*, *minor*, *patch*, *tag*)
Bases: tuple

major
Alias for field number 0

minor
Alias for field number 1

patch
Alias for field number 2

tag
Alias for field number 3

reproman.utils.**any_re_search**(*regexes*, *value*)
Return if any of regexes (list or str) searches succesfully for value

reproman.utils.**assure_bytes**(*s*, *encoding='utf-8'*)
Convert/encode unicode to bytes if of 'str'

Parameters **encoding** (*str, optional*) – Encoding to use. "utf-8" is the default

reproman.utils.**assure_dict_from_str**(*s*, ***kwargs*)
Given a multiline string with key=value items convert it to a dictionary

Parameters

- **s** (*str or dict*) –
- **None if input s is empty** (*Returns*) –

reproman.utils.**assure_dir**(**args*)
Make sure directory exists.

Joins the list of arguments to an os-specific path to the desired directory and creates it, if it not exists yet.

reproman.utils.**assure_list**(*s*)
Given not a list, would place it into a list. If None - empty list is returned

Parameters **s** (*list or anything*) –

reproman.utils.**assure_list_from_str**(*s*, *sep='\n'*)
Given a multiline string convert it to a list of return None if empty

Parameters **s** (*str or list*) –

reproman.utils.**assure_tuple_or_list**(*obj*)
Given an object, wrap into a tuple if not list or tuple

reproman.utils.**assure_unicode**(*s*, *encoding=None*, *confidence=None*)
Convert/decode to str if of 'bytes'

Parameters

- **encoding** (*str, optional*) – Encoding to use. If None, "utf-8" is tried, and then if not a valid UTF-8, encoding will be guessed
- **confidence** (*float, optional*) – A value between 0 and 1, so if guessing of encoding is of lower than specified confidence, ValueError is raised

reproman.utils.**attrib**(**args*, ***kwargs*)
Extend the attr.ib to include our metadata elements.

ATM we support additional keyword args which are then stored within *metadata*: - *doc* for documentation to describe the attribute (e.g. in –help)

Also, when the *default* argument of attr.ib is unspecified, set it to None.

reproman.utils.**auto_repr**(*cls*)
Decorator for a class to assign it an automagic quick and dirty __repr__

It uses public class attributes to prepare repr of a class

Original idea: http://stackoverflow.com/a/27799004/1265472

reproman.utils.**cached_property**(*prop*)
Cache a property's return value.

This avoids using *lru_cache*, which is more complicated than needed for simple properties and isn't available in Python 2's stdlib.

Use this only if the property's return value is constant over the life of the object. This isn't appropriate for a property with a setter or a property whose getter value may change based some outside state.

This should be positioned below the @property declaration.

class reproman.utils.**chpwd**(*path*, *mkdir=False*, *logsuffix=''*)
Bases: object

Wrapper around os.chdir which also adjusts environ['PWD']

The reason is that otherwise PWD is simply inherited from the shell and we have no ability to assess directory path without dereferencing symlinks.

If used as a context manager it allows to temporarily change directory to the given path

reproman.utils.**cmd_err_filter**(*err_string*)
Creates a filter for CommandErrors that match a specific error string

Parameters **err_string** (*basestring*) – The error string we want to match

Returns

Return type func object -> boolean

reproman.utils.**command_as_string**(*command*)
Convert *command* to the string representation.

Parameters **command** (*list or str*) – If it is a list, convert it to a string, quoting each element as needed. If it is a string, it is returned as is.

reproman.utils.**encode_filename**(*filename*)
Encode unicode filename

reproman.utils.**escape_filename**(*filename*)
Surround filename in "" and escape " in the filename

reproman.utils.**execute_command_batch**(*session*, *command*, *args*, *exception_filter=None*)
Generator that executes session.execute_command, with batches of args

We want to call commands like "apt-cache policy" on a large number of packages, but risk creating command-lines that are too long. This function is a generator that will call execute_command but with batches of arguments (to stay within the command-line length limit) and yield the results.

Parameters

- **session** – Session object that implements the execute_command() member
- **command** (*sequence*) – The command that we wish to execute
- **args** (*sequence*) – The long list of additional arguments we wish to pass to the command
- **exception_filter** (*func x -> bool*) – A filter of exception types that the calling code will gracefully handle

Returns stdout of the command, stderr of the command, and an exception that is in the list of expected exceptions

Return type (out, err, exception)

reproman.utils.**expandpath**(*path*, *force_absolute=True*)
Expand all variables and user handles in a path.

By default return an absolute path

reproman.utils.**file_basename**(*name*, *return_ext=False*)
Strips up to 2 extensions of length up to 4 characters and starting with alpha not a digit, so we could get rid of .tar.gz etc

reproman.utils.**find_files**(*regex*, *topdir='.'*, *exclude=None*, *exclude_vcs=True*, *exclude_reproman=False*, *dirs=False*)
Generator to find files matching regex

Parameters

- **regex** (*basestring*) –
- **exclude** (*basestring, optional*) – Matches to exclude
- **exclude_vcs** – If True, excludes commonly known VCS subdirectories. If string, used as regex to exclude those files (regex: *'/(?:git|gitattributes|svn|bzr|hg)(?:/|$)'*)
- **exclude_reproman** – If True, excludes files known to be reproman meta-data files (e.g. under .reproman/ subdirectory) (regex: *'/(?:reproman)(?:/|$)'*)
- **topdir** (*basestring, optional*) – Directory where to search
- **dirs** (*bool, optional*) – Either to match directories as well as files

reproman.utils.**generate_unique_name**(*pattern*, *nameset*)
Create a unique numbered name from a pattern and a set

Parameters

- **pattern** (*basestring*) – The pattern for the name (to be used with %) that includes one %d location
- **nameset** (*collection*) – Collection (set or list) of existing names. If the generated name is used, then add the name to the nameset.

Returns The generated unique name

Return type str

reproman.utils.**get_cmd_batch_len**(*arg_list*, *cmd_len*)
Estimate the maximum batch length for a given argument list

To make sure we don't call shell commands with too many arguments this function looks at an argument list and the command length without any arguments, and estimates the number of arguments we want to batch together at one time.

Parameters

- **arg_list** (*list*) – The list to process in the command
- **cmd_len** (*number*) – The length of the command without arguments

Returns The maximum number in a single batch

Return type number

reproman.utils.**get_func_kwargs_doc**(*func*)
Provides args for a function

Parameters **func** (*str*) – name of the function from which args are being requested

Returns of the args that a function takes in

Return type list

reproman.utils.**get_tempfile_kwargs**(*tkwargs={}*, *prefix=''*, *wrapped=None*)
Updates kwargs to be passed to tempfile. calls depending on env vars

reproman.utils.**getargspec**(*func*)
Backward-compatibility wrapper for inspect.getargspec.

reproman.utils.**getpwd**()
Try to return a CWD without dereferencing possible symlinks

If no PWD found in the env, output of getcwd() is returned

reproman.utils.**instantiate_attr_object**(*item_type*, *items*)
Instantiate item_type given items (for a list or dict)

Provides a more informative exception message in case if some arguments are incorrect

reproman.utils.**is_binarystring**(*s*)
Return true if an object is a binary string (not unicode)

reproman.utils.**is_explicit_path**(*path*)
Return whether a path explicitly points to a location

Any absolute path, or relative path starting with either '../' or './' is assumed to indicate a location on the filesystem. Any other path format is not considered explicit.

reproman.utils.**is_interactive**()
Return True if all in/outs are tty

reproman.utils.**is_subpath**(*path*, *directory*)
Test whether *path* is below (or is itself) *directory*.

Symbolic links are not resolved before the check.

reproman.utils.**is_unicode**(*s*)
Return true if an object is unicode

reproman.utils.**items_to_dict**(*l*, *attrs='name'*, *ordered=False*)
Given a list of attr instances, return a dict using specified attrs as keys

Parameters

- **attrs** (*str or list of str*) – Which attributes of the items to use to group
- **ordered** (*bool, optional*) – Either to return an ordered dictionary following the original order of items in the list

Raises ValueError – If there is a conflict - multiple items with the same attrs used for key

Returns

Return type dict or collections.OrderedDict

reproman.utils.**join_sequence_of_dicts**(*seq*)
Joins a sequence of dicts into a single dict

Parameters **seq** (*sequence*) – Sequence of dicts to join

Returns

Return type dict

Raises RuntimeError if a duplicate key is encountered.

reproman.utils.**knows_annex**(*path*)
Returns whether at a given path there is information about an annex

It is just a thin wrapper around GitRepo.is_with_annex() classmethod which also checks for *path* to exist first.

This includes actually present annexes, but also uninitialized ones, or even the presence of a remote annex branch.

reproman.utils.**line_profile**(*func*)
Q&D helper to line profile the function and spit out stats

reproman.utils.**lmtime**(*filepath*, *mtime*)
Set mtime for files, while not de-referencing symlinks.

To overcome absence of os.lutime

Works only on linux and OSX ATM

reproman.utils.**make_tempfile**(*content=None*, *wrapped=None*, ***tkwargs*)
Helper class to provide a temporary file name and remove it at the end (context manager)

Parameters

- **mkdir** (*bool, optional (default: False)*) – If True, temporary directory created using tempfile.mkdtemp()
- **content** (*str or bytes, optional*) – Content to be stored in the file created
- **wrapped** (*function, optional*) – If set, function name used to prefix temporary file name
- ****tkwargs** – All other arguments are passed into the call to tempfile.mk{,d}temp(), and resultant temporary filename is passed as the first argument into the function t. If no 'prefix' argument is provided, it will be constructed using module and function names ('.' replaced with '_').
- **change the used directory without providing keyword argument 'dir' set** (*To*) –
- **REPROMAN_TESTS_TEMPDIR.** –

Examples

```
>>> from os.path import exists
>>> from reproman.utils import make_tempfile
>>> with make_tempfile() as fname:
...     k = open(fname, 'w').write('silly test')
>>> assert not exists(fname)  # was removed
```

```
>>> with make_tempfile(content="blah") as fname:
...     assert open(fname).read() == "blah"
```

reproman.utils.**md5sum**(*filename*)

reproman.utils.**merge_dicts**(*ds*)
Convert an iterable of dictionaries.

In the case of key collisions, the last value wins.

Parameters **ds** (*iterable of dicts*) –

Returns

Return type dict

reproman.utils.**not_supported_on_windows**(*msg=None*)
A little helper to be invoked to consistently fail whenever functionality is not supported (yet) on Windows

reproman.utils.**only_with_values**(*d*)
Given a dictionary, return the one only with entries which had non-null values

reproman.utils.**optional_args**(*decorator*)
allows a decorator to take optional positional and keyword arguments. Assumes that taking a single, callable, positional argument means that it is decorating a function, i.e. something like this:

```
@my_decorator
def function(): pass
```

Calls decorator with decorator(f, **args*, ***kwargs*)

reproman.utils.**parse_kv_list**(*params*)
Create a dict from a "key=value" list.

Parameters **params** (*sequence of str or mapping*) – For a sequence, each item should have the form "<key>=<value". If *params* is a mapping, it will be returned as is.

Returns

Return type A mapping from backend key to value.

Raises ValueError if item in *params* does not match expected "key=value" format.

reproman.utils.**parse_semantic_version**(*version*)
Split version into major, minor, patch, and tag components.

Parameters **version** (*str*) – A version string X.Y.Z. X, Y, and Z must be digits. Any remaining text is treated as a tag (e.g., "-rc1").

Returns

Return type A namedtuple with the form (major, minor, patch, tag)

reproman.utils.**partition**(*items*, *predicate=<class 'bool'>*)
Partition *items* by *predicate*.

Parameters

- **items** (*iterable*) –
- **predicate** (*callable*) – A function that will be mapped over each element in *items*. The elements will partitioned based on whether the return value is false or true.

Returns

- *A tuple with two generators, the first for 'false' items and the second for*
- *'true' ones.*

Notes

Taken from Peter Otten's snippet posted at https://nedbatchelder.com/blog/201306/filter_a_list_into_two_parts.html

reproman.utils.**pycache_source**(*path*)
Map a pycache path to the original path.

Parameters **path** (*str*) – A Python cache file.

Returns

- Path of cached Python file (str) or None if *path* doesn't look like a
- *cache file.*

reproman.utils.**rmtemp**(*f*, **args*, ***kwargs*)
Wrapper to centralize removing of temp files so we could keep them around

It will not remove the temporary file/directory if REPROMAN_TESTS_KEEPTEMP environment variable is defined

reproman.utils.**rmtree**(*path*, *chmod_files='auto'*, **args*, ***kwargs*)
To remove git-annex .git it is needed to make all files and directories writable again first

Parameters

- **chmod_files** (*string or bool, optional*) – Either to make files writable also before removal. Usually it is just a matter of directories to have write permissions. If 'auto' it would chmod files on windows by default
- ***args** –
- ****kwargs** – Passed into shutil.rmtree call

reproman.utils.**rotree**(*path*, *ro=True*, *chmod_files=True*)
To make tree read-only or writable

Parameters

- **path** (*string*) – Path to the tree/directory to chmod
- **ro** (*bool, optional*) – Either to make it R/O (default) or RW
- **chmod_files** (*bool, optional*) – Either to operate also on files (not just directories)

reproman.utils.**safe_write**(*ostream*, *s*, *encoding='utf-8'*)
Safely write different string types to an output stream

reproman.utils.**setup_exceptionhook**(*ipython=False*)
Overloads default sys.excepthook with our exceptionhook handler.

If interactive, our exceptionhook handler will invoke pdb.post_mortem; if not interactive, then invokes default handler.

reproman.utils.**shortened_repr**(*value*, *l=30*)

reproman.utils.**sorted_files**(*dout*)
Return a (sorted) list of files under dout

reproman.utils.**swallow_logs**(*new_level=None*)
Context manager to consume all logs.

reproman.utils.**swallow_outputs**()
Context manager to help consuming both stdout and stderr, and print()

stdout is available as cm.out and stderr as cm.err whenever cm is the yielded context manager. Internally uses temporary files to guarantee absent side-effects of swallowing into StringIO which lacks .fileno.

print mocking is necessary for some uses where sys.stdout was already bound to original sys.stdout, thus mocking it later had no effect. Overriding print function had desired effect

reproman.utils.**to_binarystring**(*s*, *encoding='utf-8'*)
Converts any type string to binarystring

reproman.utils.**to_unicode**(*s*, *encoding='utf-8'*)
Converts any type string to unicode

reproman.utils.**unique**(*seq*, *key=None*)
Given a sequence return a list only with unique elements while maintaining order

This is the fastest solution. See https://www.peterbe.com/plog/uniqifiers-benchmark and http://stackoverflow.com/a/480227/1265472 for more information. Enhancement – added ability to compare for uniqueness using a key function

Parameters

- **seq** – Sequence to analyze
- **key** (*callable, optional*) – Function to call on each element so we could decide not on a full element, but on its member etc

reproman.utils.**updated**(*d*, *update*)
Return a copy of the input with the 'update'

Primarily for updating dictionaries

reproman.utils.**write_update**(*fname*, *content*, *encoding=None*)
Write *content* to *fname* unless it already has matching content.

This is the same as simply writing the content, except no writing occurs if the content of the existing file matches, the write or update is logged, and the leading directories of *fname* are created if needed.

Parameters

- **fname** (*str*) – Path to update.
- **content** (*str*) – Content to dump to path.
- **encoding** (*str or None, optional*) – Passed to *open*.

reproman.version

Defines version to be imported in the module and obtained from setup.py

reproman.support.configparserinc

class reproman.support.configparserinc.**SafeConfigParserWithIncludes**(**args*, ***kwargs*)
Bases: configparser.ConfigParser

Class adds functionality to SafeConfigParser to handle included other configuration files (or may be urls, whatever in the future)

File should have section [includes] and only 2 options implemented are 'files_before' and 'files_after' where files are listed 1 per line.

Example:

```
[INCLUDES]
before = 1.conf
         3.conf
```

(continues on next page)

(continued from previous page)

```
after = 1.conf
```

It is a simple implementation, so just basic care is taken about recursion. Includes preserve right order, ie new files are inserted to the list of read configs before original, and their includes correspondingly so the list should follow the leaves of the tree.

I wasn't sure what would be the right way to implement generic (aka c++ template) so we could base at any **configparser* class... so I will leave it for the future

SECTION_NAME = 'INCLUDES'

static getIncludes(*resource*, *seen=[]*)
Given 1 config resource returns list of included files (recursively) with the original one as well Simple loops are taken care about

read(*filenames*)
Read and parse a filename or an iterable of filenames.

Files that cannot be opened are silently ignored; this is designed so that you can specify an iterable of potential configuration file locations (e.g. current directory, user's home directory, systemwide directory), and all existing configuration files in the iterable will be read. A single filename may also be given.

Return list of successfully read files.

3.2.3 Configuration management

config	Registry-like monster for now simply borrowed from bigmess/pymvpa

reproman.config

Registry-like monster for now simply borrowed from bigmess/pymvpa

TODO: integration with cmdline etc

class reproman.config.**ConfigManager**(*filenames=None*, *load_default=True*)
Bases: *reproman.support.configparserinc.SafeConfigParserWithIncludes*, object

Central configuration registry for reproman.

The purpose of this class is to collect all configurable settings used by various parts of reproman. It is fairly simple and does only little more than the standard Python ConfigParser. Like ConfigParser it is blind to the data that it stores, i.e. no type checking is performed.

Configuration files (INI syntax) in multiple location are parsed when a class instance is created or whenever *Config.reload()* is called later on. Files are read and parsed in the order described by *LOCATIONS_DOC*.

Moreover, the constructor takes an optional argument with a list of additional file names to parse afterwards.

In addition to configuration files, this class also looks for special environment variables to read settings from. Names of such variables have to start with *REPROMAN_* following by the an optional section name and the variable name itself ('_' as delimiter). If no section name is provided, the variables will be associated with section *general*. Some examples:

```
REPROMAN_VERBOSE=1
```

will become:

```
[general]
verbose = 1
```

However, *REPROMAN_VERBOSE_OUTPUT=stdout* becomes:

```
[verbose]
output = stdout
```

Any length of variable name as allowed, e.g. REPROMAN_SEC1_LONG_NAME=1 becomes:

```
[sec1]
long name = 1
```

Settings from custom configuration files (specified by the constructor argument) have the highest priority and override settings found in any of the config files read from default locations (which are themselves read in the order stated above – overwriting earlier configuration settings accordingly). Finally, the content of any *REPROMAN_** environment variables overrides any settings read from any file.

dirs = <appdirs.AppDirs object>

get(*section*, *option*, *default=None*, ***kwargs*)
Wrapper around SafeConfigParser.get() with a custom default value.

This method simply wraps the base class method, but adds a *default* keyword argument. The value of *default* is returned whenever the config parser does not have the requested option and/or section.

get_as_dtype(*section*, *option*, *dtype*, *default=None*)
Convenience method to query options with a custom default and type

This method simply wraps the base class method, but adds a *default* keyword argument. The value of *default* is returned whenever the config parser does not have the requested option and/or section.

In addition, the returned value is converted into the specified *dtype*.

getboolean(*section*, *option*, *default=None*)
Wrapper around SafeConfigParser.getboolean() with a custom default.

This method simply wraps the base class method, but adds a *default* keyword argument. The value of *default* is returned whenever the config parser does not have the requested option and/or section.

getpath(**args*, ***kwargs*)
Wrapper around get to do additional path treatments such as expanduser

See documentation for *get*

reload(*filenames=None*)
Re-read settings from all configured locations.

3.2.4 Test infrastructure

tests.utils	Miscellaneous utilities to assist with testing

reproman.tests.utils

Miscellaneous utilities to assist with testing

class reproman.tests.utils.**SilentHTTPHandler**(**args*, ***kwargs*)
Bases: http.server.SimpleHTTPRequestHandler

A little adapter to silence the handler

log_message(*format*, **args*)
Log an arbitrary message.

This is used by all other logging functions. Override it if you have specific logging wishes.

The first argument, FORMAT, is a format string for the message to be logged. If the format string contains any % escapes requiring parameters, they should be specified as subsequent arguments (it's just like printf!).

The client ip and current date/time are prefixed to every message.

reproman.tests.utils.**assert_cwd_unchanged**(*func*, *ok_to_chdir=False*)
Decorator to test whether the current working directory remains unchanged

Parameters **ok_to_chdir** (*bool, optional*) – If True, allow to chdir, so this decorator would not then raise exception if chdir'ed but only return to original directory

reproman.tests.utils.**assert_equal**(*a*, *b*, *msg=None*)

reproman.tests.utils.**assert_false**(*x*, *msg=None*)

reproman.tests.utils.**assert_greater**(*a*, *b*, *msg=None*)

reproman.tests.utils.**assert_greater_equal**(*a*, *b*, *msg=None*)

reproman.tests.utils.**assert_in**(*x*, *collection*, *msg=None*)

reproman.tests.utils.**assert_in_in**(*substr*, *lst*)
Verify that a substring is in an element of a list

reproman.tests.utils.**assert_is**(*a*, *b*, *msg=None*)

reproman.tests.utils.**assert_is_instance**(*a*, *b*, *msg=None*)

reproman.tests.utils.**assert_is_subset_recur**(*a*, *b*, *subset_types=[]*)
Asserts that 'a' is a subset of 'b' (recursive on dicts and lists)

Parameters

- **a** (*dict or list*) – The desired subset collection (items that must be in b)
- **b** (*dict or list*) – The superset collection
- **subset_types** (*list*) – List of classes (from list, dict) that allow subsets. Otherwise we use strict matching.

reproman.tests.utils.**assert_not_equal**(*a*, *b*, *msg=None*)

reproman.tests.utils.**assert_not_in**(*x*, *collection*, *msg=None*)

reproman.tests.utils.**assert_re_in**(*regex*, *c*, *flags=0*)
Assert that container (list, str, etc) contains entry matching the regex

reproman.tests.utils.**assert_true**(*x*, *msg=None*)

reproman.tests.utils.**create_pymodule**(*directory*)
Create a skeleton Python module in *directory*.

Parameters **directory** (*str*) – Path to a non-existing directory.

reproman.tests.utils.**create_tree**(*path*, *tree*, *archives_leading_dir=True*)
Given a list of tuples (name, load) create such a tree

if load is a tuple itself – that would create either a subtree or an archive with that content and place it into the tree if name ends with .tar.gz

reproman.tests.utils.**eq_**(*a*, *b*, *msg=None*)

reproman.tests.utils.**get_most_obscure_supported_name**(*tdir*)
Return the most obscure filename that the filesystem would support under TEMPDIR

TODO: we might want to use it as a function where we would provide tdir

reproman.tests.utils.**in_**(*x*, *collection*, *msg=None*)

reproman.tests.utils.**neq_**(*a*, *b*, *msg=None*)

reproman.tests.utils.**nok_**(*x*, *msg=None*)

reproman.tests.utils.**nok_startswith**(*s*, *prefix*)

reproman.tests.utils.**ok_**(*x*, *msg=None*)

reproman.tests.utils.**ok_broken_symlink**(*path*)

reproman.tests.utils.**ok_endswith**(*s*, *suffix*)

reproman.tests.utils.**ok_file_has_content**(*path*, *content*)
Verify that file exists and has expected content

reproman.tests.utils.**ok_generator**(*gen*)

reproman.tests.utils.**ok_good_symlink**(*path*)

reproman.tests.utils.**ok_startswith**(*s*, *prefix*)

reproman.tests.utils.**ok_symlink**(*path*)
Checks whether path is either a working or broken symlink

reproman.tests.utils.**run_under_dir**(*func*, *newdir='.'*)
Decorator to run tests under another directory

It is somewhat ugly since we can't really chdir back to a directory which had a symlink in its path. So using this decorator has potential to move entire testing run under the dereferenced directory name – sideeffect.

The only way would be to instruct testing framework (i.e. nose in our case ATM) to run a test by creating a new process with a new cwd

reproman.tests.utils.**serve_path_via_http**(*tfunc*, **targs*)
Decorator which serves content of a directory via http url

reproman.tests.utils.**with_tempfile**(*t*, ***tkwargs*)
Decorator function to provide a temporary file name and remove it at the end

Parameters

- **change the used directory without providing keyword argument 'dir' set** (*To*) –
- **REPROMAN_TESTS_TEMPDIR.** –
- **mkdir** (*bool, optional (default: False)*) – If True, temporary directory created using tempfile.mkdtemp()
- **content** (*str or bytes, optional*) – Content to be stored in the file created
- **wrapped** (*function, optional*) – If set, function name used to prefix temporary file name
- ****tkwargs** – All other arguments are passed into the call to tempfile.mk{,d}temp(), and resultant temporary filename is passed as the first argument into the function t. If no 'prefix' argument is provided, it will be constructed using module and function names ('.' replaced with '_').

Examples

```
@with_tempfile
def test_write(tfile):
    open(tfile, 'w').write('silly test')
```

reproman.tests.utils.**with_testsui**(*t*, *responses=None*)
Switch main UI to be 'tests' UI and possibly provide answers to be used

reproman.tests.utils.**with_tree**(*t*, *tree=None*, *archives_leading_dir=True*, *delete=True*, ***tkwargs*)

reproman.tests.utils.**without_http_proxy**(*tfunc*)
Decorator to remove http*_proxy env variables for the duration of the test

3.2.5 Command line interface infrastructure

cmdline.main
cmdline.helpers
cmdline.common_args

reproman.cmdline.main

reproman.cmdline.main.**main**(*args=None*)

reproman.cmdline.main.**setup_parser**(*formatter_class=<class 'argparse.RawDescriptionHelpFormatter'>*, *return_subparsers=False*)

reproman.cmdline.helpers

class reproman.cmdline.helpers.**HelpAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
Bases: argparse.Action

class reproman.cmdline.helpers.**LogLevelAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
Bases: argparse.Action

class reproman.cmdline.helpers.**PBSAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
Bases: argparse.Action

Action to schedule actual command execution via PBS (e.g. Condor)

class reproman.cmdline.helpers.**RegexpType**
Bases: object

Factory for creating regular expression types for argparse

DEPRECATED AFAIK – now things are in the config file, but we might provide a mode where we operate solely from cmdline

reproman.cmdline.helpers.**get_repo_instance**(*path='.'*, *class_=None*)
Returns an instance of appropriate reproman repository for path. Check whether a certain path is inside a known type of repository and returns an instance representing it. May also check for a certain type instead of detecting the type of repository.

Parameters

- **path** (*str*) – path to check; default: current working directory
- **class** (*class*) – if given, check whether path is inside a repository, that can be represented as an instance of the passed class.

Raises RuntimeError, in case cwd is not inside a known repository.

reproman.cmdline.helpers.**parser_add_common_args**(*parser*, *pos=None*, *opt=None*, ***kwargs*)

reproman.cmdline.helpers.**parser_add_common_opt**(*parser*, *opt*, *names=None*, ***kwargs*)

reproman.cmdline.helpers.**run_via_pbs**(*args*, *pbs*)

reproman.cmdline.helpers.**strip_arg_from_argv**(*args*, *value*, *opt_names*)
Strip an originally listed option (with its value) from the list cmdline args

reproman.cmdline.common_args

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

P

R

S

T

U

V

W

Y