pyhowto Documentation

Release 0.0.1

Paul Brian <paul@mikadosoftware.com>

May 16, 2014

> Don't write production code till you have broken the prototype
>
> —me

I am a software developer, turned entreprenuer. The goal is to create a forever company - a place where my career can be both software and business. I am writing a book on my journey from developer to consultant to SaaS entrepreneur - the technical half and the business half.

This is the Technical half.

CHAPTER 1

Building the platform

1.1 Supervisor

1.1.1 Keeping micro-web services alive, supervised and restarted

Waiting around to restart your web server is painful - Steve Huffman, one of the founders of Reddit talks about literally sleeping with his laptop next to his bed and constantly waking to restart a process at ungodly hours of the morning.

Micro-services are in my view a great way to arrange our business applications. By splitting the work done into smaller components, and having the work done by seperate webservers running just a small piece of code, we find several benefits

- our code is better structured, because we cannot rely on the big framework to pass bits around for us
- Testing the service is easier
- Things cen be upgrade, downgrade or side-ways-graded with more confidence. A fix to one tiny piece of independantly running code will hardly ever affect the others. In a monolithic application that is rarely true. Change the font size for Page X, ooops that just killed half the CSS on site.
- naturally robust / redundant.

And so...

Now imagine if he had built a micro-service architecture, 50 or more seperate web servers, and was manully trying to watch them all, Reddit could have failed and we would have lost millions of LoLCatz. The horror!

Eventually he discovered *supervisord* and got some sleep. This is a Good Thing.

We also want something that will monitor them and respawn them when they die.

This will we hope increase our chances of a service that is small, stays up and if it fails, is automatically brought back up.

We also want to keep things simple.

The simplest way possible to keep a service running

It would be lovely to write a web service, set it running and know that if it fell over, divided by zero, or otherwise went wrong it would immediately be picked up, dusted down and restarted.

If we can do that so it scales upwards really well when it is working.

Oh and it needs to make the tea too.

Ok, the first part is *process monitoring* - watching a running process, and if it falls over (or rather exits with non zero) restarting it.

There are a lot of options out there - *supervisord*, *god* are a couple that spring to mind. But these are inherently part of a language ecosystem (Pyhton and Ruby respecitively). This is not a *bad* thing, they do the job, but distributions and sysadmins have for a long time faced these problems and come up with effective solutions already. I think this is part of the packaging problem all language eco-systems seem to face - and often well solved by looking at the distribution tools first and then worrying.

The SysV / init camp of tools is old, and still very effective. Over the years new olutions have been proposed, three I shall cover: *rc.d*, *upstart* and *systemd*. Upstart was championed by Ubuntu/Canonical and is an attempt to keep the simple, file driven approach going. systemd is however winning hearts and minds, and has even driven upstart out of the debian world, and so ubuntu will soon follow suit. I shall however target 12.04 for the moment and that will remain upstart based.

rc.d comes from NetBSD, and so is a worthwhile investment in the FreeBSD world.

Python and WSGI

Web micro-services will consist mostly of well, web servers. And in the Python world that means WSGI.

Now, a WSGI app is a curious beast, essentially it cares nothing for web servers and threads and so on, it simply takes an *environment* (like CGI env) and a *start_response* callable.

A WSGI enabled server will wrap the core application (returns "hello world") in a chain of python functions, each one taking an env and a callable, each one modifying the response as we go up the chain.

I will have to write a small WSGI server to demonstrate.

Anyway, lets run a WSGI app on FreeBSD

FreeBSD

Overview

I want to have *the simplest possible way to run a micro web service*. For me this is to make use of the well thought out, well tested and well, simple, *init* services built into all Unix distributions.

I shall create two dummy web services, *hello.py* that is simply a Flask service returning "hello World", and this shall be started and stopped with an *rc.d* script.

A second more complex script shall follow, dealing with some config issues etc.

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

## trivial WSGI example


import datetime
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World! " + datetime.datetime.today().isoformat()
```

```
if __name__ == "__main__":
    app.run(port=5003)
```

This is about as simple as it gets (ripped, liberally, from the frontpage of flask.org.). However run in the terminal it will respond correctly to the various signals thrown at it.

```
$ python hello.py
 * Running on http://127.0.0.1:5003/
^C

We set the web server running, then can kill it with ctl-C
```

However we want to *prove* that if the app dies by itself, *rc* will restart it.

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import datetime
from flask import Flask
app = Flask(__name__)

## Just mark the log
open("/tmp/foobar","a").write("\nStarting: "
                              + datetime.datetime.today().isoformat()
                             )



@app.route("/")
def hello():
    return "Hello World! " + datetime.datetime.today().isoformat()

### I want to have the app die, and so prove rc restarts it
@app.route("/kill")
def killself():
    open("/tmp/foobar","a").write("\nkilled: "
                                  + datetime.datetime.today().isoformat()
                                  )
    #os._exit(1)
    return "foo"

if __name__ == "__main__":
    app.run(port=5003)

if we run this and then visit localhost:5003/kill,
the process will exit.

(errr... really?)
```

Lets make this "production-ready" [1]

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import sys, os
import datetime
from flask import Flask
```

[1] OK so its not actually production ready - monitoring, logging etc. But its a lot more production than the Flask default server.

```
from signal import SIGTERM
####
pid = os.getpid()


def hello():
    return "Hello World! My PID is : %s. Time is %s" % (pid,
                                                        datetime.datetime.today().strftime("%H%M%S"))


def killself():
    open("/tmp/foobar","w").write("killed")
    os.kill(pid, SIGTERM)

'''
this one will need to kill

supervisor-scripts(master)$ /usr/home/pbrian/venvs/test-rc/bin/waitress-serve --call hellofactory:app
serving on http://0.0.0.0:8080

#https://github.com/Pylons/waitress/blob/master/waitress/runner.py
.pth file in venv
add2virtualenv


'''


def appfactory():
    """
    an attempt at an app_factory
    """
    app = Flask(__name__)
    app.add_url_rule("/", view_func=hello)
    app.add_url_rule("/kill", view_func=killself)
    return app


if __name__=='__main__':
    appfactory().run(port=5003)
```

THe app factory issue...

Using the BSD rc.d init approach

The short approach is we put a simple .sh script in */usr/local/etc/rc.d*. This is called during system init, and it will run a command - that command will be a WSGI server (like uwsgi / gunicorn / waitress) which will bring up a wsgi app.

This app will serve HTTP happily, but if it falls over, rc.d will restart it.

notes (test-rc)supervisor-scripts(master)$ which gunicorn /usr/home/pbrian/venvs/test-rc/bin/gunicorn

Now deploying a simple WSGI app with Gunicorn but what about a djaongo app

Django deployed standalone WSGI with gunicorn

```
$ gunicorn --pythonpath readthedocs readthedocs.wsgi:application
$ /home/pbrian/venvs/docserver/bin/gunicorn  --pythonpath
/usr/home/pbrian/venvs/docserver/checkouts/readthedocs.org
readthedocs.wsgi:application
```

Have to create a wsgi.py in the readthedocs "app dir"

```
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "settings.sqlite")

# This application object is used by the development server
# as well as any WSGI server configured to use this file.
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

/home/pbrian/venvs/docserver/bin/gunicorn –pythonpath /usr/home/pbrian/venvs/docserver/checkouts/readthedocs.org/readthedocs wsgi:application

```
rc.d$ cat readthedocs
#!/bin/sh
###
# PROVIDE: readthedocs
# REQUIRE: LOGIN
# KEYWORD: shutdown

# We always require LOGIN unless we know what we are doing.
# shutdown is there to kill us off in case of system shutdown.

. /etc/rc.subr

name="readthedocs"
rcvar=readthedocs_enable
#pidfile=/tmp/${name}.pid

###########
venv_gunicorn=/home/pbrian/venvs/docserver/bin/gunicorn
django_app_path=/usr/home/pbrian/venvs/docserver/checkouts/readthedocs.org/readthedocs
django_app_wsgi=wsgi:application

command="$venv_gunicorn -D \
         --pythonpath $django_app_path \
         $django_app_wsgi"

#########

load_rc_config $name
run_rc_command "$1"
```

Todo

Also need to deal with command_interpreter.

Bibilio

- #pkgng@freenode

- https://www.freebsd.org/doc/en/articles/rc-scripting/article.html

Ubuntu and Debian

Well, I used to use *upstart*. That has been replaced with *systemd* now, and I ... well, keep using upstart.

[TBD - insert code from ubuntu setup in Bamboo]

Further reading

(get sphinx to link these up for me ...)

- Security for micro-web-services
- running your own CA

1.2 automated-builds

1.2.1 Building Servers and Workstations from Scratch

I have (re)written this capability many times, sometimes from scratch meant assembling your own hardware, sometimes "a bunch of shell scripts".

This time however, its true love. Well...

1.2.2 Workstation

I intend to use *salt-call –local state.highstate* on my newly installed laptop to bring it up to a state of pristine, hey aint that cool workability. All my stuff on it.

1.2.3 Cloud

I intend to use a *minion-master* arrangement to create different and new servers (rackspace, AWS, GCE) as needed and for Continuous Delivery.

Next steps

1.2.4 Building a FreeBSD laptop from scratch with Salt-stack

I am a big FreeBSD user, both in servers and as a personal workstation. Yes, Linux is *easier* as a workstation, but then so is Windows. And its nice to have the same environments.

I however often find myself putting off important upgrades because my laptop is my working life - losing a day here would cost real money. I need to automate the maintenance of my laptop. And Shell scripts don't seem "official" enough. (I may occassionally cheat however :)

So as well as installing servers using salt-stack here is how to build a set of DevOps quality scripts to keep your workstation rebuildable at a moments notice, and keep the day-to-day maintenance spruced up and ship shape!

The idea is to install the laptop with FreeBSD 10, point it at the new packaging system, run the miniuon bootstrap, download my own set of salt states, and ... presto.

Starting off

Firstly I need to wipe and install a FreeBSD minimum image onto a machine. Luckily this is easy - just download the latest *.img* file from FreeBSD.org, and copy it over to a USB.

Visit:

```
ftp://ftp.freebsd.org/pub/FreeBSD/snapshots/ISO-IMAGES/10.0/
```

and pull back the latest **memstick.img* file such as

```
FreeBSD-10.0-STABLE-amd64-20140421-r264704-memstick.img
```

Now we *dd* over the image from disk to USB.:

```
$ dd if=memstick.img of=/dev/da0 bs=1024 conv=sync
```

Reboot the laptop with USB key attached (ensuring of course the BIOS is set to boot from USB *before* HDD).

Now install FreeBSD as usual. It is a pretty straightforward install, and well documented in the FreeBSD Manual.

Download and run bootstrap installer

The salt stack bootstrap script essentially downloads salt onto your local machine (OK, a number of potential issues there), and will do various OS specific setups (in FreeBSD's case, prepare the *pkg* environment, point at the right URLs etc.)

Unfortunately fetch is a bit rubbish compared to wget, and barfs on SSL certificates like the dodgy ones at github. I should look into it.

> $ setenv SSL_NO_VERIFY_PEER 1 $ fetch http://bootstrap.saltstack.org
>
> This seems a very bad idea, as we are opening up to a MITM attack. I would prefer either: * md5 signing
> * download wget *first* https://github.com/saltstack/salt-bootstrap/issues/290

This will migrate me to using the *pkgng* (FreeBSD's own apt-get setup) convert the laptop to a minion, and then all I need to do is set the right values in /env/salt and call local highstate

---

Todo

More about pkgng [expand on pkgng]

---

1. Download the appropriate salt-states
2. apply to /usr/local/etc/salt
3. run *salt-call –local state.highstate*

This way I have now applied a set of states I want on the laptop to a local cache location (for these purposes, just Xorg). They include config files etc.

Changes We want to put the states in /usr/local/etc/salt/states We want to put our own execution modules in /usr/local/etc/salt/states *and* alter /usr/local/etc/salt/minion to change its *module_dirs []* list to include that.

How does salt-call work (brief)?

1. Process the base environment top.sls file. This is by default */srv/salt/top.sls* but can be changed (see file_roots)
2.

What do I actually want on my workstation?

An article per pkg, ala NTP plus the init.sls and assoc state files.

- sudo
- wget
- xfce4 hal dbus xorg randr and multi head
- urxvt / xterm for unicode http://www.cl.cam.ac.uk/~mgk25/unicode.html https://wiki.archlinux.org/index.php/fonts $ setfont Lat2-Terminus16
- aspell
- bash
- bsdstats
- security port scanner gnupg keychain ssh
- sysadmin jails
- dbases sqlite postgres
- webkit
- sound
- video
- curl
- wget
- emacs
- git / git gui
- sublime?
- fonts
- printing
- Firewall
- python eco-system
- ZFS (TBD)
- web browsers
-
- ImageMagick
- gimp
- rabbitMQ
- spreadsheets??

Other needs

- RPi Routers and NetFlow / packetburst for my local office network

Business Half

- Reporting and Dotted Co-ordination Framework

1.3 flask

```
from flask import Flask, request
app = Flask(__name__)

@app.route("/")
def hello():
    return str(request.environ)

if __name__ == "__main__":
    app.run()
```

1.4 Supervisor

1.4.1 Keeping micro-web services alive, supervised and restarted

Waiting around to restart your web server is painful - Steve Huffman, one of the founders of Reddit talks about literally sleeping with his laptop next to his bed and constantly waking to restart a process at ungodly hours of the morning.

Micro-services are in my view a great way to arrange our business applications. By splitting the work done into smaller components, and having the work done by seperate webservers running just a small piece of code, we find several benefits

- our code is better structured, because we cannot rely on the big framework to pass bits around for us
- Testing the service is easier
- Things cen be upgrade, downgrade or side-ways-graded with more confidence. A fix to one tiny piece of independantly running code will hardly ever affect the others. In a monolithic application that is rarely true. Change the font size for Page X, ooops that just killed half the CSS on site.
- naturally robust / redundant.

And so...

Now imagine if he had built a micro-service architecture, 50 or more seperate web servers, and was manully trying to watch them all, Reddit could have failed and we would have lost millions of LoLCatz. The horror!

Eventually he discovered *supervisord* and got some sleep. This is a Good Thing.

We also want something that will monitor them and respawn them when they die.

This will we hope increase our chances of a service that is small, stays up and if it fails, is automatically brought back up.

We also want to keep things simple.

The simplest way possible to keep a service running

It would be lovely to write a web service, set it running and know that if it fell over, divided by zero, or otherwise went wrong it would immediately be picked up, dusted down and restarted.

If we can do that so it scales upwards really well when it is working.

Oh and it needs to make the tea too.

Ok, the first part is *process monitoring* - watching a running process, and if it falls over (or rather exits with non zero) restarting it.

There are a lot of options out there - *supervisord*, *god* are a couple that spring to mind. But these are inherently part of a language ecosystem (Pyhton and Ruby respecitively). This is not a *bad* thing, they do the job, but distributions and sysadmins have for a long time faced these problems and come up with effective solutions already. I think this is part of the packaging problem all language eco-systems seem to face - and often well solved by looking at the distribution tools first and then worrying.

The SysV / init camp of tools is old, and still very effective. Over the years new olutions have been proposed, three I shall cover: *rc.d*, *upstart* and *systemd*. Upstart was championed by Ubuntu/Canonical and is an attempt to keep the simple, file driven approach going. systemd is however winning hearts and minds, and has even driven upstart out of the debian world, and so ubuntu will soon follow suit. I shall however target 12.04 for the moment and that will remain upstart based.

rc.d comes from NetBSD, and so is a worthwhile investment in the FreeBSD world.

Python and WSGI

Web micro-services will consist mostly of well, web servers. And in the Python world that means WSGI.

Now, a WSGI app is a curious beast, essentially it cares nothing for web servers and threads and so on, it simply takes an *environment* (like CGI env) and a *start_response* callable.

A WSGI enabled server will wrap the core application (returns "hello world") in a chain of python functions, each one taking an env and a callable, each one modifying the response as we go up the chain.

I will have to write a small WSGI server to demonstrate.

Anyway, lets run a WSGI app on FreeBSD

FreeBSD

Overview

I want to have *the simplest possible way to run a micro web service*. For me this is to make use of the well thought out, well tested and well, simple, *init* services built into all Unix distributions.

I shall create two dummy web services, *hello.py* that is simply a Flask service returning "hello World", and this shall be started and stopped with an *rc.d* script.

A second more complex script shall follow, dealing with some config issues etc.

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

## trivial WSGI example


import datetime
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
```

```
    return "Hello World! " + datetime.datetime.today().isoformat()

if __name__ == "__main__":
    app.run(port=5003)
```

This is about as simple as it gets (ripped, liberally, from the frontpage of flask.org.). However run in the terminal it will respond correctly to the various signals thrown at it.

```
$ python hello.py
 * Running on http://127.0.0.1:5003/
^C

We set the web server running, then can kill it with ctl-C
```

However we want to *prove* that if the app dies by itself, *rc* will restart it.

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import datetime
from flask import Flask
app = Flask(__name__)

## Just mark the log
open("/tmp/foobar","a").write("\nStarting: "
                              + datetime.datetime.today().isoformat()
                             )



@app.route("/")
def hello():
    return "Hello World! " + datetime.datetime.today().isoformat()

### I want to have the app die, and so prove rc restarts it
@app.route("/kill")
def killself():
    open("/tmp/foobar","a").write("\nkilled: "
                                  + datetime.datetime.today().isoformat()
                                  )
    #os._exit(1)
    return "foo"

if __name__ == "__main__":
    app.run(port=5003)

if we run this and then visit localhost:5003/kill,
the process will exit.

(errr... really?)
```

Lets make this "production-ready" [2]

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import sys, os
```

[2] OK so its not actually production ready - monitoring, logging etc. But its a lot more production than the Flask default server.

```
import datetime
from flask import Flask
from signal import SIGTERM
####
pid = os.getpid()


def hello():
    return "Hello World! My PID is : %s. Time is %s" % (pid,
                                                        datetime.datetime.today().strftime("%H%M%S"))


def killself():
    open("/tmp/foobar","w").write("killed")
    os.kill(pid, SIGTERM)

'''
this one will need to kill

supervisor-scripts(master)$ /usr/home/pbrian/venvs/test-rc/bin/waitress-serve --call hellofactory:app
serving on http://0.0.0.0:8080

#https://github.com/Pylons/waitress/blob/master/waitress/runner.py
.pth file in venv
add2virtualenv


'''


def appfactory():
    """
    an attempt at an app_factory
    """
    app = Flask(__name__)
    app.add_url_rule("/", view_func=hello)
    app.add_url_rule("/kill", view_func=killself)
    return app


if __name__=='__main__':
    appfactory().run(port=5003)
```

THe app factory issue...

Using the BSD rc.d init approach

The short approach is we put a simple .sh script in */usr/local/etc/rc.d*. This is called during system init, and it will run a command - that command will be a WSGI server (like uwsgi / gunicorn / waitress) which will bring up a wsgi app.

This app will serve HTTP happily, but if it falls over, rc.d will restart it.

notes (test-rc)supervisor-scripts(master)$ which gunicorn /usr/home/pbrian/venvs/test-rc/bin/gunicorn

Now deploying a simple WSGI app with Gunicorn but what about a djaongo app

Django deployed standalone WSGI with gunicorn

```
$ gunicorn --pythonpath readthedocs readthedocs.wsgi:application
$ /home/pbrian/venvs/docserver/bin/gunicorn  --pythonpath
/usr/home/pbrian/venvs/docserver/checkouts/readthedocs.org
readthedocs.wsgi:application
```

Have to create a wsgi.py in the readthedocs "app dir"

```
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "settings.sqlite")

# This application object is used by the development server
# as well as any WSGI server configured to use this file.
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

/home/pbrian/venvs/docserver/bin/gunicorn –pythonpath /usr/home/pbrian/venvs/docserver/checkouts/readthedocs.org/readthedocs wsgi:application

```
rc.d$ cat readthedocs
#!/bin/sh
###
# PROVIDE: readthedocs
# REQUIRE: LOGIN
# KEYWORD: shutdown

# We always require LOGIN unless we know what we are doing.
# shutdown is there to kill us off in case of system shutdown.

. /etc/rc.subr

name="readthedocs"
rcvar=readthedocs_enable
#pidfile=/tmp/${name}.pid

###########
venv_gunicorn=/home/pbrian/venvs/docserver/bin/gunicorn
django_app_path=/usr/home/pbrian/venvs/docserver/checkouts/readthedocs.org/readthedocs
django_app_wsgi=wsgi:application

command="$venv_gunicorn -D \
         --pythonpath $django_app_path \
         $django_app_wsgi"

#########

load_rc_config $name
run_rc_command "$1"
```

Todo

Also need to deal with command_interpreter.

Bibilio

- #pkgng@freenode

- https://www.freebsd.org/doc/en/articles/rc-scripting/article.html

Ubuntu and Debian

Well, I used to use *upstart*. That has been replaced with *systemd* now, and I ... well, keep using upstart.

[TBD - insert code from ubuntu setup in Bamboo]

Further reading

(get sphinx to link these up for me ...)

- Security for micro-web-services
- running your own CA

1.5 sysadmin

1.5.1 Managing a PC BSD workstation

I am a big fan of the clean lines of FreeBSD Unix. However, I am in a minority, where the winner (Linux) has massive network effect advantages.

Migrating my main workstation(s) to Linux is not my preferred solution.

I used to build my workstation from scracth - rebuilding the whole from source (See OSBuilder). However that started taking more and more precious time.

So I am looking for productivity solutions.

I am a software professional choosing to use BSD to develop on, and choosing to do so in a manner that does not lose me days of client-billable time if a driver upgrade fails.

PC-BSD presents a workstation on FreeBSD. They spend a lot of time and effort making things work. But we can cut down a lot on the outliers.

- Why is the world changing when the basic stuff still works?

Networking

Ethernet networking just *works*. So stick with it. For a desktop machine this is fine. For a laptop - well I *used* to stick to one dongle and one set of configs - till that changed below me. Now - Netgear universal WiFi.

Portjails

PC-BSD have prebuilt pbis. THats fine but on an upgrade they wipe out all your nice ports you built. And if you dont use ports, you dont get source - fopr example you need to build postgres-client apps (psql) from source so that Python psycopg libraries can be built.

So - the PC_BSD upgrade doesnot wipe *portjails*

1. initialising the portjail
2. updating a working portjail

```
portjail console
su (sudo not configured here)
portsnap fetch extract (or update)
```

3.

GitHub

Setting up ghi

sudo /usr/local/bin/gem18 install iconv – –with-opt-dir=/usr/local/

Xorg and Dual Monitors

I know VGA moniotr works - I used to have iut running pre PC-BSD and I can dual moniotr with anohter laptop xrandr not finding VGA

[pbrian@hadrian ~]$ sudo lspci | grep -i vga 00:01.0 VGA compatible controller: ATI Technologies Inc Device 9802

sudo update-pciids

---

http://unix.stackexchange.com/questions/44299/how-to-deal-with-freebsds-move-to-pkgconf migrating from pkg_config to pkgconf

Basically a main junction is being replaced / renamed. We want to renameit thoughtout beforr more problems

```
sudo portmaster -o devel/pkgconf devel/pkg-config
```

1.6 single-sign-on

1.6.1 Single Sign-on

To Be covered:

- Session based SSO
- Client Certificate SSO

CHAPTER 2

Coding Best Practise

2.1 continuous-integration

2.1.1 PLACEHOLDER

fixme

2.2 error-handling

2.2.1 PLACEHOLDER

fixme

2.3 codereview-bestpractise

They *do* make a difference.

2.4 git

2.4.1 Git Commit messages

> Any software project is a collaborative project. It has at least two developers, the original developer and the original developer a few weeks or months later
>
> —Peter Hutterer <http://www.blogger.com/profile/17204066043271384535>

> The difference between a tolerable programmer and a great programmer is [...] whether they can communicate their ideas. By persuading other people, they get leverage. By writing clear comments and technical specs, they let other programmers understand their code, which means other programmers can use and work with their code instead of rewriting it. Absent this, their code is worthless.
>
> —Joel Spolsky <http://www.joelonsoftware.com/articles/CollegeAdvice.html>

What does a good commit message look like?

```
Scope: Subject line in imperative tense and less than 50 chars

Body text with why we needed to do this,
how this works at a high level
and any side-effects we know of or predict.
Do not exceed 72 chars on these lines, because 4 chars indent, will
catch you a lot.

[tags for closing bugs etc.]
```

The rules for git commit messages

1. Put a subject line as the first line, followed by a blank line then the body.

   Git grew out of the Linux Kernel Mailing List

2.

The two rules not to break:

Never have two lines of text at the top. Always have one line then blank line and body, or one line only.

Never exceed 72 chars per line.

The one rule to rarely break:

Always have 50 chars in the subject line

For your own sanity, make your project history clear, your documentation good.

Business justification: good code practises save time and money.

Commits and User Stories

My view is that in an Agile environment, User Stories should be about the same size as a commit. That is if there is a User Story:

```
2014.32.16 (16th sprint backlog item in 32nd sprint in 2014)

As the Sales Manager, in order to motivate my team,
the weekly sales report must be ordered by total revenue
in USD.
```

Then I should expect to see a commit history like:

```
$ git log --pretty=format:"%s"
reporting: Order RW23 (weekly sales) by total rev USD
reporting: Add function to convert sales.revenue to USD
externalapis: Store daily GBP -> USD conversion rates
reporting: Clean up code to PEP8 standards
```

Here I can see that I had to clean up the code in area of reporting, then store a GBP to USD conversion rate, add in a function to do the actual conversion and then change report RW23 to be ordered differently.

I would further expect these commit messages to reference *2014.32.16* such as

```
fixes: 2014.32.16
rel: 2014.32.16
rel: 2014.32.16
rel: 2014.32.16
```

Preferred workflow

- Integration Manager (or pull-requests)

Breaking Changes into different commits

The ideal is to use *git add -p*. This should be done once or twice by everyone. Then stop being a masochist and use *git gui* which allows you to choose line by line which parts go into the next commit and which wait.

Very very useful. There are many other tools for doing this but git gui is fone.

(Just be aware to always pick the right - and + changes in one go. Unless you know what you are doing)

Git is not a backup

I am guilty of this almost constantly.

I am weening myself off this habit by backing up all my development dirs via tarsnap.

(Actually I may have to stop this to overcome EU data protection laws, and rsync to a Dutch backup location)

Biblio

http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html https://docs.google.com/a/mikadosoftware.com/document/d/1C
http://www.mediawiki.org/wiki/Gerrit/Commit_message_guidelines https://wiki.openstack.org/wiki/GitCommitMessages
https://www.kernel.org/doc/Documentation/SubmittingPatches

Other projects:

- time tracking based on git commits.
-

2.5 python-packages

2.5.1 Python Packaging Best Practises

Rule number 1: Run! Run for the hills !

2.6 python-tricks

2.6.1 PLACEHOLDER

fixme

2.6.2 closures

Lexical closures are a really useful trick.

Basically, we can create a function at run-time, and the complete call-stack that exists *at the time it was created* gets wrapped up and kept neatly until the next time it is called. So its a way of capturing a variable at a frozen point in time. (This is to me useful, but has no value in say a functional language).

```
def foo(avar):
    def bar():
        return "The variable avar was \
                this value when I was created %s" % avar
    return bar

results = []
for i in range(10):
    results.append(foo(i))

for x in results:
    print x()
```

resulting in

```
$ python closures.py
The variable avar was                this value when I was created 0
The variable avar was                this value when I was created 1
The variable avar was                this value when I was created 2
The variable avar was                this value when I was created 3
The variable avar was                this value when I was created 4
The variable avar was                this value when I was created 5
The variable avar was                this value when I was created 6
The variable avar was                this value when I was created 7
The variable avar was                this value when I was created 8
The variable avar was                this value when I was created 9
```

2.6.3 Fizzbuzz

The fizzbuzz test is a pretty good determinate of ability to code.

It has a low false negative rate - that is if you can code, its pretty hard to fail. It also has a low false positive rate - if you cannot code, you simply wont pass. As such it is apparently depressing why it is still a necessary evil - I can attest to sitting slackjawed as candidates could not understand a SELECT

> query let alone attempt a for loop.

I am not a big fan of programming puzzles like codility. I think they certainly have their place - I suspect as ongoing development for coders. Having an hour or two a week just to work on Project Euler problems would sharpen anyone's saw.

It must be admitted that I have only seen *early* candidates fail so amazingly. In general there is a pool of people who apply for every programming gig going, and will have CVs tuned to get through the doors. But there is a reason they don't get hired and so are available to interview the first day you post the ad.

This is why recruitment is an ongoing investment.

I however have flamed out over the more complex puzzles. I remember happily designing a scalable architecture for some pretend web app, discussing the benefits and costs of Rabbit MQ, and then being asked to write a routine to calculate prime numbers, whilst the director of Engineering harangued looking like the dog with the bone if he saw a

missed optimisation. I have never yet turned round in an interview and said shut the fuck up and let me write on the board. But it was touch and go.

As such, lets look at fizzbuzz, and Python and Tail Call Recursion.

Todo

show to AST for this and how elimating tail call could elimnate the stack.

```
def fb(i, n, result):
    if i == n:
        return result
    elif i % 15 == 0:
       result.append("FizzBuzz")
    elif i % 3 == 0:
       result.append("Fizz")
    elif i % 5 == 0:
       result.append("Buzz")
    else:
       result.append(i)
    return fb(i+1, n, result)

#print fb(1,999,[])

def fb2(i, n, result):
    if i == n:
        return result
    elif i % 15 == 0:
       result.append("FizzBuzz")
    elif i % 3 == 0:
       result.append("Fizz")
    elif i % 5 == 0:
       result.append("Buzz")
    else:
       result.append(i)
    return (lambda: fb2(i+1, n, result))

#this now returns a function that when called ( a() )
# returns the next recursion


# http://paulbutler.org/archives/tail-recursion-in-python/
# basicakky turning the recursion into a whle loop
# I have expanded out the params
def tail_rec(fun):
   def tail(fun):
      a = fun
      while callable(a):
         a = a()          # here a is a function that returns a function
                          # that returns the next recursive step
                          # essentially we convert the recursion into a while
                          # loop of each step in the recusrion.
                          # instead of the recursive function calling itself
                          # we make a while loop call it.
      return a
   return (lambda *x: tail(fun(*x)))

fbr = tail_rec(fb2)
print fbr(1,2000,[])
```

```
"""
if we use fb in tail_rec it still recurses.
SO what is going on?

the inner part of tail_rec takes any callable and calls it in a while loop
till it returns its halt condition.

This block is then returned as an anonymous function.
So we convert any recursive function into a lambda function that
returns a function that unwinds the recursive function

Now we make that recursive function at its tail, not call itself,
but return a lambda that returns its "call myself"

so...

fbr is now a lambda function that given <params> will
repeatedly set a = a() for the



"""
```

But if I exceed pythons recursion depth limit

1001

2.7 shell-redirect

This is a demo of an annoying problem - shell redirection. It gets everyone sometimes.

2.8 logging

2.8.1 Best practise for logging in Python

Replace print stmts with logging.debug

If all you ever do with logging is

```
import logging
...
logging.warn("help")
```

You are 100% better off than

```
print "help"
```

By using logging stmts throughout your code, with no more real thought than sprinkling with print stmts, we do the hard part (sprinkling print stmts in difficult places) whilst leaving the easy but valuable part (collating and reviewing logs) for later.

For example, later on we can create in the *run* start up of any module, something like:

```
logging.BasicConfig(filename="/var/log/myapp.log",
                    level=logging.DEBUG)
```

This will log all this app's calls at or above DEBUG to /var/log. Which is where it is supposed to. And something extra like logging to syslog, then using rsyslog to pull to a central logging server, is just a *salt* call away.

Do not use string formatting in the calling module

```
>>> logging.error("Help %s has gone wrong." % myvar)
```

If myvar does not exist here, then we shall raise an error

Do not use Python3 style formatting

Obviously Python 3 stuff is in flux, but the logging module expects %s style formating, and is unlikely to change for sometime as it will break backwards compatibility.

So stick to {'reason': MyVar} for the moment.

ReRaise errors by default

How we handle errors will vary a lot, depending on the app and the error. But, if we are logging something, its often because an exception has been thrown. In most cases in my personal view, we want to log the existence of the problem, and then rethrow

```
>>> import logging
... try:
...     open("Notmuchchanceofthisexisting.txt")
... except IOError, e:
...     logging.error("Could not open file")
...     raise
```

http://victorlin.me/posts/2012/08/26/good-logging-practice-in-python Logging ——-

http://docs.python.org/2/howto/logging-cookbook.html#logging-cookbook http://css.dzone.com/articles/best-practices-python-logging http://victorlin.me/2012/08/good-logging-practice-in-python/ http://www.robg3d.com/?p=906

Basics:

At app start up: import logging name a logger

in every other module import logging set a module level global with that same logging name

2.9 dbases

2.9.1 Installing Sqlite in a venv

Seems to be painful - best answer so far seems to be to link or mv.

So we build sqlite database as usual in local. Then build the venv and link externally, or copy in the .so into venv site-packages.

$ ln -s /usr/local/lib/python2.7/site-packages/_sqlite3.so .

>> import sqlite3

2.9.2 Using PostgreSQL on FreeBSD

Find enum types:

```
dbtest=> SELECT pg_type.typname AS enumtype,
dbtest->      pg_enum.enumlabel AS enumlabel
dbtest->  FROM pg_type
dbtest->  JOIN pg_enum
dbtest->      ON pg_enum.enumtypid = pg_type.oid;

   enumtype    | enumlabel
---------------+------------
 cnxrole_type | author
 cnxrole_type | maintainer
 cnxrole_type | copyright
(3 rows)
```

Setting up postgres on FreeBSD

```
$ make install clean
$ /usr/local/bin/postgres -D /usr/local/pgsql/data
```

for some reason everything gets put into /usr/local/pgsql/data - the config files and so on.

Configuration and users

- postgresql.conf

For local service:

```
listen_addresses = 'localhost'
```

and in pg_hba.conf:

```
host    all             all             127.0.0.1/32            md5
```

We change `trust` to `md5` so that it exxpects user/password

$ createuser -sdrP test1 Enter password for new role: Enter it again:

pgsql$ sudo su - pgsql $ /usr/local/bin/createdb dbtest -O test1 encoding=UNICODE $

$ psql -U test1 -d dbtest ... dbtest=# CREATE USER repo WITH PASSWORD 'repopass'; CREATE ROLE dbtest=# GRANT ALL PRIVILEGES ON DATABASE dbtest to repo; GRANT

pgsql$ psql -h 127.0.0.1 -U repo -d dbtest Password for user repo: psql (9.2.1) Type "help" for help.

dbtest=>

(NB localhost will try to use Unix domain sockets so use 127.0.0.1)

For network server:

```
listen_addresses = '*'
```

Logging

Normally we want to

```
log_destination = 'syslog'
logging_collector = off
```

For development purposes we usually want to watch the SQL fly past:

```
log_destination = 'stderr'
logging_collector = on
log_directory = 'pg_log'
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'

log_statement = 'all'
```

2.10 styleguide

2.10.1 Style guides

$ pip install flake8 autopep8

$ flake8 myfile.py ## will list a lot of things you are doing wrong. ## A lot will be whitespace

$ flake8 myfile.py –show-pep8 ## Will show the text of what you are doing wrong ## so you can actually fix it.

2.10.2 User Interface Design

> Start with the User Need

User Interface Design, at least at the levels of "I want my small Enterprise App to be useful", is pretty simple.

It needs four components

1. Visual Style
2. Consistent base HTML elements
3. Higher Level Design patterns incl. behaviour
4. That extra dash of style

We have three levels, style-core, style-pkg, style-xp This way we have manageable paths for different discussions to grow. eXPerimental styles, or interactions, the CSS and the JS for them go into these packages.

If after a tryout the XP stuff works, it

I have several different bits in different levels. My core is used everywhere, for each client. pkgs usually get pulled in, but are there to overwrite things like bootstrap date pickers.

Visual Style

CSS, color theory, logo etc

Consistent base HTML elements

Design Elements

Handling higher level elements.

Extra Dash of Something

Here is where I bow out. I can do an "alright" job with the leverage of things like colorwheels, bootstrap, and a bit of common sense. But I am not pixel perfect obessive with a mental memory of thousands of interactive sites, the deep understanding of design, trade-offs, a finger on the pulse of the *zeitgeist* and the ability to mix all these into one intuitive expression of your company's essence.

That's why you hire a professional. But you can do a lot before you get there, which is lucky, 'cos the good ones are expensive :-)

Living guides

Enhancement Proposals

Not changing stuff willy-nilly

Biblio

http://govuk-elements.herokuapp.com/ http://alistapart.com/article/creating-style-guides

2.10.3 Example ReST directives

A Subtitle

Bullet lists

- One
- Two
- Three

Footnotes here [1] and another here [2]

Sidebar

A sidebar

Body of a SideBar

[1] Footnote First
[2] Footnote Second

Topic

a Topic

Body of a Topic

Code

```
def my_function():
    "just a test"
    print 8/2
```

Math

Rubric

Its like a pull quote but not

Epigraph

> And You ma'am are ugly.
>
> But madam, in the morning I shall be sober – Winston

Highlight

> Highlight
>
> Just highlighting this

Pull-Quote

> Quote me
>
> Its a quote from the body but bigger.

The 'rm' command is very dangerous. If you are logged in as root and enter

```
cd /
rm -rf *
```

you will erase the entire contents of your file system.

This paragraph might be rendered in a custom way.

CHAPTER 3

The Front End

3.1 colours

3.1.1 Valid Colour Names for Web

What colours to pick form on the web? What when you have to cycle through s load to keep table stipes weorking? http://en.wikipedia.org/wiki/X11_color_names

I have extracted the colors, then divided up the pastels. As a non desginer, that looks fine to me

```
colour_list = [
'Alice Blue',
'Antique White',
'Aqua',
'Aquamarine',
'Azure',
'Beige',
'Bisque',
'Black',
'Blanched Almond',
'Blue',
'Blue Violet',
'Brown',
'Burlywood',
'Cadet Blue',
'Chartreuse',
'Chocolate',
'Coral',
'Cornflower',
'Cornsilk',
'Crimson',
'html_name=Aqua',
'Dark Blue',
'Dark Cyan',
'Dark Goldenrod',
'Dark Gray',
'Dark Green',
'Dark Khaki',
'Dark Magenta',
'Dark Olive Green',
'Dark Orange',
'Dark Orchid',
```

```
'Dark Red',
'Dark Salmon',
'Dark Sea Green',
'Dark Slate Blue',
'Dark Slate Gray',
'Dark Turquoise',
'Dark Violet',
'Deep Pink',
'Deep Sky Blue',
'Dim Gray',
'Dodger Blue',
'Firebrick',
'Floral White',
'Forest Green',
'Fuchsia',
'Gainsboro',
'Ghost White',
'Gold',
'Goldenrod',
'Gray',
'Green',
'Green Yellow',
'Honeydew',
'Hot Pink',
'Indian Red',
'Indigo',
'Ivory',
'Khaki',
'Lavender',
'Lavender Blush',
'Lawn Green',
'Lemon Chiffon',
'Light Blue',
'Light Coral',
'Light Cyan',
'Light Goldenrod',
'Light Gray',
'Light Green',
'Light Pink',
'Light Salmon',
'Light Sea Green',
'Light Sky Blue',
'Light Slate Gray',
'Light Steel Blue',
'Light Yellow',
'Lime',
'Lime Green',
'Linen',
'Fuchsia',
'Maroon',
'Medium Aquamarine',
'Medium Blue',
'Medium Orchid',
'Medium Purple',
'Medium Sea Green',
'Medium Slate Blue',
'Medium Spring Green',
'Medium Turquoise',
```

```
'Medium Violet Red',
'Midnight Blue',
'Mint Cream',
'Misty Rose',
'Moccasin',
'Navajo White',
'Navy',
'Old Lace',
'Olive',
'Olive Drab',
'Orange',
'Orange Red',
'Orchid',
'Pale Goldenrod',
'Pale Green',
'Pale Turquoise',
'Pale Violet Red',
'Papaya Whip',
'Peach Puff',
'Peru',
'Pink',
'Plum',
'Powder Blue',
'Purple',
'Red',
'Rosy Brown',
'Royal Blue',
'Saddle Brown',
'Salmon',
'Sandy Brown',
'Sea Green',
'Seashell',
'Sienna',
'Silver',
'Sky Blue',
'Slate Blue',
'Slate Gray',
'Snow',
'Spring Green',
'Steel Blue',
'Tan',
'Teal',
'Thistle',
'Tomato',
'Turquoise',
'Violet',
'Wheat',
'White',
'White Smoke',
'Yellow',
'Yellow Green',


]
```

3.2 bootstrap

3.2.1 Using Less and ReSt to make nice looking documents

Not so long ago the only *right* way to transform markdown style documents into something decent was via LaTeX and thence into pdf.

These days HTML and CSS are the way to go it seems. Its just good enough and the toolchain is far far more likely to be available on your machine right now.

3.2.2 First steps

Lets see what a simple ReSt document looks like. We have the demo ReSt for the docutils project.

```
.. This is a comment. Note how any initial comments are moved by
   transforms to after the document title, subtitle, and docinfo.

================================
 reStructuredText Demonstration
================================

.. Above is the document title, and below is the subtitle.
   They are transformed from section titles after parsing.


Literal Blocks
--------------

Literal blocks are indicated with a double-colon ("::") at the end of
the preceding paragraph (over there ``-->``).  They can be indented::

    if literal_block:
        text = 'is left as-is'
        spaces_and_linebreaks = 'are preserved'
        markup_processing = None
```

Now this looks like the page here - a bit plain and boring.

But we want to sprinkle magic Bootstrap dust over it. Now boring people might suggest the correct way to do this is to alter the HTML-formatter in docutils and so get that to output HTML divs that bootstrap expects. This is actually the right way to do it but I wont for several reasons.

1. This is an article about configuring bootstrap, not docutils
2. docutils is a really dense impenetrable forest of code and I do not understand it nearly well enough to make a quick change.
3. I am pretty sure I can do this with a few lines of less. If so, result !

3.2.3 Compiling and modifying bootstrap

Building a simple web page these days is fraught with hidden traps for the unwary. A long time ago, I sat wearing a padded jacket in a cold office, hand-typing out webpages for multi-national companies. (That got old quick so we pretty soon fixed the heating and the production workflow).

But the point was, one person used to be able to write an entire web site, suitable for the Fortune 500, in a text editor.

These days, not so much. Obviously we still use text editors, but they produce programs that produce programs that produce web sites. And they are not one-person jobs.

However one person can still pull together all the right pieces, and with Twitter Bootstrap, its even easier to get "good enough" design and look, whilst still keeping everything manageable.

I intend to create the following page template, for an open source project I am working on and promoting - it takes a DataWarehouse ETL tool (Oracle ODI) and allows it to work bi-directionally with modern source control tools (http://github.com/pmsoftware/odietamo)

Installation

We shall

- git clone the Bootstrap repo,
- install the (mostly javascript) dependancies and tool chains
- run a simple compile
- celebrate.

```
## Some variables for later
rootdir=/opt/bootstrap

cd $rootdir
git clone https://github.com/twbs/bootstrap.git
```

Right, now a bit of tricky javascript. Install *node* and *npm* with your distribution's preferred method. For example that would be ports on FreeBSD - you are using BSD no? If not try

```
$ apt-get install -y node npm
```

If you have node and npm installed then:

```
$ cd $rootdir/bootstrap/bootstrap
$ sudo npm install
```

This will download all the dependancies and

NB - FreeBSD is not well supported for PhantomJS - which only matters for the self-tests. I ignore this on my local machine and blip over to a VM when I need.

If that did not make sense, do not worry.

How to compile

Its simple, in the bootstrap dir, where we can see the *Gruntfile.js* run:

```
$ grunt dist
```

This will start a javascript task runner (sort of make for js) that simply builds a */dist/* directory containing the compiled js, css and font files. This */dist/* dir is what gets served to the browser in production. The rest of the files are supporting acts.

OK, we should have seen something like this:

```
Running "copy:fonts" (copy) task
Copied 4 files

Running "concat:bootstrap" (concat) task
File "dist/js/bootstrap.js" created.

Running "uglify:bootstrap" (uglify) task
File "dist/js/bootstrap.min.js" created.

Done, without errors.
```

Serving

As this is all very local, we want to see the web browser deal with it

```
$ cd $rootdir
$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```

Now a webbrowser can ask for *http://localhost:8000/example-page.html* and it should see

Moving things around

You can skip this bit and come back later - it will make more sense then.

We don't want to directly change the files provided for us by the bootstrap team. We do however need to change *something*

Our first Bootstrap File

We are now using Bootstrap 3. This is a new, updated version, that is *mobile first*. Which means its designed to be really sensible on most devices. Hooray - that means we can deploy applications to a mobile, without going through the various appstores. Well sort of.

HTML5

We need a solid HTML5 template file. The best one to use is HTML5 boilerplate.

Make sure this is in the template file `<head>`:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Responsive friendly images. We need to add `.img-responsive` class to each `img` tag so that they are treated correctly by Bootstrap. I need to use a post-process step in the CMS to shoe-horn these in - whatever gets you through the night.

Containers

Everything in BootStrap is in a container. A container is defined as follows.

```
<div class="container">
...
</div>
```

The hero units (now seemingly renamed Jumbotrons), the forms, the tables, they

Shims and things

Shims and polyfills are bits of javascript code that provide HTML5 functionality on older, but still widely deployed, web browsers that do not support HTML5 natively. The IE range upto IE 7 is a notable case, still widely deployed in corporate environments.

An example is the HTML5 element `canvas` which allows javascript to draw on a canvas. Where there is no HTML5 `canvas` element built into the browser, the polyfill will call up perhaps a Silverlight plugin and perform the draw action on that.

These shims and polyfills are amazing pieces of work, but (as in polyfilla) they simply cover over cracks. And they are not perfect. So I am ignoring them for now in my template code. In production this may change but for our purposes they add complexity to understanding.

https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills http://remysharp.com/2010/10/08/what-is-a-polyfill/

A quick customisation

Lets start with *variables.less*. The CSS we use for Bootstrap is not hand-written - it is compiled into CSS after the original *less* code is parsed. The original *less* code is designed to make writing lots of CSS easier, so it supports things like variables and functions (called *mixins* but think returning function for ease).

So a quick snippet of *variables.less*:

```
 @gray-darker:            lighten(#000, 13.5%); // #222
 @gray-dark:              lighten(#000, 20%);   // #333
 @gray:                   lighten(#000, 33.5%); // #555
 @gray-light:             lighten(#000, 60%);   // #999
 @gray-lighter:           lighten(#000, 93.5%); // #eee

 // Scaffolding
 // -------------------------

 @body-bg:               #fff;
 @text-color:            @gray-dark;
`
```

Now, lets have some fun. Firstly our *test.html* page. This is quite a bit of code, but it is about the simplest HTML5 + bootstrap page you can make, and it liberally ripped off from the bootstrap site.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Navbar Template for Bootstrap</title>
    <!-- Bootstrap core CSS -->
    <link href="bootstrap/dist/css/bootstrap.css" rel="stylesheet">
    <!-- Custom styles for this template -->
    <link href="navbar.css" rel="stylesheet">
  </head>
```

```
  <body>

    <div class="container">

      <!-- Main component for a primary marketing message or call to action -->
      <div class="jumbotron" id="showcasing" style="background: url('LHC.jpg') repeat-x top center;">
        <h1>It works !</h1>
        <p>Hooray</p>
        <p>
          <a class="btn btn-primary" href="#">Press me</a>
        </p>
      </div>

    </div> <!-- /container -->

    <!-- Bootstrap core JavaScript
    ================================================== -->
    <!-- Placed at the end of the document so the pages load faster -->
    <script src="bootstrap/assets/js/jquery.js"></script>
    <script src="bootstrap/dist/js/bootstrap.min.js"></script>
  </body>
</html>
```

Now the above html, is about the simplest one can get for bootstrap. And it looks like this:

Firstly let's adjust the simplest things we can - the LESS variables.

```
// added pbrian
@brand-back: #8a928e;
@brand-fore:  #8e393f;

@gray-darker:            lighten(#000, 13.5%); // #222
@gray-dark:              lighten(#000, 20%);   // #333
@gray:                   lighten(#000, 33.5%); // #555


@brand-primary:         @brand-fore;

                        ^^^^^^
                        This will roll out across the site.
```

And the result is :

Ok, nothing spectacular, but one line change gives us a new colour set across the board. What else can we do?

Fonts

TBD

Images

TBD

biblio

https://raw.github.com/mikadosoftware/screengrab/master/screenshots/colorschemedesigner.png
http://bootstrap.lesscss.ru/less.html http://copyhackers.com/2012/12/the-3-step-hack-for-startups-bootstrapping-their-design/ http://24ways.org/2012/how-to-make-your-site-look-half-decent/ http://www.sitediscount.ru/verso/index.html

Choose a color scheme: http://colorschemedesigner.com/

```
#####  Color Palette by Color Scheme Designer
#####  Palette URL: http://colorschemedesigner.com/#0Q51Rw0w0w0w0
#####  Color Space: RGB;



*** Primary Color:

   var. 1 = #FF9F00 = rgb(255,159,0)
   var. 2 = #BF8930 = rgb(191,137,48)
   var. 3 = #A66800 = rgb(166,104,0)
   var. 4 = #FFB740 = rgb(255,183,64)
   var. 5 = #FFCA73 = rgb(255,202,115)

*** Secondary Color A:

   var. 1 = #FFC700 = rgb(255,199,0)
   var. 2 = #BFA030 = rgb(191,160,48)
   var. 3 = #A68100 = rgb(166,129,0)
   var. 4 = #FFD540 = rgb(255,213,64)
   var. 5 = #FFE073 = rgb(255,224,115)

*** Secondary Color B:

   var. 1 = #FF6700 = rgb(255,103,0)
   var. 2 = #BF6A30 = rgb(191,106,48)
   var. 3 = #A64300 = rgb(166,67,0)
   var. 4 = #FF8D40 = rgb(255,141,64)
   var. 5 = #FFAB73 = rgb(255,171,115)


#####  Generated by Color Scheme Designer (c) Petr Stanicek 2002-2010


   @colorA1:  #FF9F00;
   @colorA2:  #BF8930;
   @colorA3:  #A66800;
   @colorA4:  #FFB740;
   @colorA5:  #FFCA73;

   @colorB1:  #FFC700;
   @colorB2:  #BFA030;
   @colorB3:  #A68100;
   @colorB4:  #FFD540;
   @colorB5:  #FFE073;

   @colorC1:  #FF6700;
   @colorC2:  #BF6A30;
   @colorC3:  #A64300;
   @colorC4:  #FF8D40;
   @colorC5:  #FFAB73;
```

We can also adjust the fonts site wide.

New fonts: http://www.google.com/fonts/

<link href='http://fonts.googleapis.com/css?family=Roboto' rel='stylesheet' type='text/css'>

font-family: 'Roboto', sans-serif; <link href='http://fonts.googleapis.com/css?family=Press+Start+2P' rel='stylesheet' type='text/css'> (ad nauseum example)

We can see the results (quite horrible) here:

http://coding.smashingmagazine.com/2013/03/12/customizing-bootstrap/

Building my masterpiece

OK, so I want to transform this

Into this:

```
Tada
```

So lets start with Design basics - cribbed liberally from the folks on InterWebs

- Step 1 - Make it look good in Black and White first
- Step 2 - Write the right things
- Step 3 - Do less. No less than that.

Color theory

Well, just colour wheels really.

Building my own bootstrap classes

Its pretty obvious what means. But that does not mean its a good idea. Yopu want a sidebar, not a span6.

So we define our own classes, and use those in the HTML instead.

This is accounted for in Bootstrap by http://getbootstrap.com/css/#grid-less

http://ruby.bvision.com/blog/please-stop-embedding-bootstrap-classes-in-your-html

Building our own mobile aware classes

Oh fuck it, really. This is the shit designers are supposed to obessess over. Here I stop. If I cannot make do with a main column and a sidebar I am going to have to go back to pen and paper.

3.2.4 How to Compile and Customise Twitter Bootstrap so you can stop worrying and get on with coding

Customising Bootstrap for Hackers

I have never produced a decent looking website - even the ones I got paid for have needed me to go and find a competent designer halfway through because my own eyeballs would refuse to look at the screen any longer, the ocular equivalent of hiding behind the sofa while the Daleks attacked.

Even the widely accepted fixes have not always helped - I have had a sort of love-hate relationship with CSS - the idea was fine in theory, but the practical reality felt more like balancing the last Ace of Spades on top of a rather worrisome house of cards, and at any moment the wind was going to pick up through that window. Probably a Dalek passing by.

So Twitter Bootstrap (or its new name "the bootstrap formerly associated with Twitter") seemed like a godsend - except ... I always used it less like a framework and more like a template. Just take the defaults and go.

You could instantly create something like this :

but while that correctly does responsive mobile blah de blah, it is - well it's the tutorial example.

So, my geek reflexes told me, I need to compile it, to customise and compile it so it can meet my needs. Bascially I opened a can of control-freakery and dove right in.

This is the result - a mini-series of using Bootstrap to build a small product site that does not look rubbish, is easy and simple HTML to maintain and code with, and frankly does not need a designer to help me.

- Part 1 - Getting Bootstrap, and compiling it from the command line !!!!
- Part 2 - Actually designing something, with colour wheels and stuff.
- Part 3 - The bit where I need a three part series but have run out of things to say.

Compiling and modifying bootstrap

Building a simple web page these days is fraught with hidden traps for the unwary. A long time ago, I sat wearing a padded jacket in a cold office, hand-typing out webpages for multi-national companies. (That got old quick so we pretty soon fixed the heating and the production workflow).

But the point was, one person used to be able to write an entire web site, suitable for the Fortune 500, in a text editor.

These days, not so much. Obviously we still use text editors, but they produce programs that produce programs that produce web sites. And they are not one-person jobs.

However one person can still pull together all the right pieces, and with Twitter Bootstrap, its even easier to get "good enough" design and look, whilst still keeping everything manageable.

I intend to create the following page template, for an open source project I am working on and promoting - it takes a DataWarehouse ETL tool (Oracle ODI) and allows it to work bi-directionally with modern source control tools (http://github.com/pmsoftware/odietamo)

Installation We shall

- git clone the Bootstrap repo,
- install the (mostly javascript) dependancies and tool chains
- run a simple compile
- celebrate.

```
## Some variables for later
rootdir=/opt/bootstrap

cd $rootdir
git clone https://github.com/twbs/bootstrap.git
```

Right, now a bit of tricky javascript. Install *node* and *npm* with your distribution's preferred method. For example that would be ports on FreeBSD - you are using BSD no? If not try

```
$ apt-get install -y node npm
```

If you have node and npm installed then:

```
$ cd $rootdir/bootstrap/bootstrap
$ sudo npm install
```

This will download all the dependancies and

NB - FreeBSD is not well supported for PhantomJS - which only matters for the self-tests. I ignore this on my local machine and blip over to a VM when I need.

If that did not make sense, do not worry.

How to compile Its simple, in the bootstrap dir, where we can see the *Gruntfile.js* run:

```
$ grunt dist
```

This will start a javascript task runner (sort of make for js) that simply builds a */dist/* directory containing the compiled js, css and font files. This */dist/* dir is what gets served to the browser in production. The rest of the files are supporting acts.

OK, we should have seen something like this:

```
Running "copy:fonts" (copy) task
Copied 4 files

Running "concat:bootstrap" (concat) task
File "dist/js/bootstrap.js" created.

Running "uglify:bootstrap" (uglify) task
File "dist/js/bootstrap.min.js" created.

Done, without errors.
```

Serving As this is all very local, we want to see the web browser deal with it

```
$ cd $rootdir
$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```

Now a webbrowser can ask for *http://localhost:8000/example-page.html* and it should see

Moving things around You can skip this bit and come back later - it will make more sense then.

We don't want to directly change the files provided for us by the bootstrap team. We do however need to change *something*

Our first Bootstrap File We are now using Bootstrap 3. This is a new, updated version, that is *mobile first*. Which means its designed to be really sensible on most devices. Hooray - that means we can deploy applications to a mobile, without going through the various appstores. Well sort of.

HTML5 We need a solid HTML5 template file. The best one to use is HTML5 boilerplate.

Make sure this is in the template file `<head>`:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Responsive friendly images. We need to add `.img-responsive` class to each `img` tag so that they are treated correctly by Bootstrap. I need to use a post-process step in the CMS to shoe-horn these in - whatever gets you through the night.

Containers Everything in BootStrap is in a container. A container is defined as follows.

```
<div class="container">
...
</div>
```

The hero units (now seemingly renamed Jumbotrons), the forms, the tables, they

Shims and things Shims and polyfills are bits of javascript code that provide HTML5 functionality on older, but still widely deployed, web browsers that do not support HTML5 natively. The IE range upto IE 7 is a notable case, still widely deployed in corporate environments.

An example is the HTML5 element `canvas` which allows javascript to draw on a canvas. Where there is no HTML5 `canvas` element built into the browser, the polyfill will call up perhaps a Silverlight plugin and perform the draw action on that.

These shims and polyfills are amazing pieces of work, but (as in polyfilla) they simply cover over cracks. And they are not perfect. So I am ignoring them for now in my template code. In production this may change but for our purposes they add complexity to understanding.

https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills http://remysharp.com/2010/10/08/what-is-a-polyfill/

A quick customisation Lets start with *variables.less*. The CSS we use for Bootstrap is not hand-written - it is compiled into CSS after the original *less* code is parsed. The original *less* code is designed to make writing lots of CSS easier, so it supports things like variables and functions (called *mixins* but think returning function for ease).

So a quick snippet of *variables.less*:

```
 @gray-darker:            lighten(#000, 13.5%); // #222
 @gray-dark:              lighten(#000, 20%);   // #333
 @gray:                   lighten(#000, 33.5%); // #555
 @gray-light:             lighten(#000, 60%);   // #999
 @gray-lighter:           lighten(#000, 93.5%); // #eee

 // Scaffolding
 // -------------------------

 @body-bg:               #fff;
 @text-color:            @gray-dark;
`
```

Now, lets have some fun. Firstly our *test.html* page. This is quite a bit of code, but it is about the simplest HTML5 + bootstrap page you can make, and it liberally ripped off from the bootstrap site.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Navbar Template for Bootstrap</title>
    <!-- Bootstrap core CSS -->
    <link href="bootstrap/dist/css/bootstrap.css" rel="stylesheet">
    <!-- Custom styles for this template -->
    <link href="navbar.css" rel="stylesheet">
  </head>

  <body>

    <div class="container">

      <!-- Main component for a primary marketing message or call to action -->
      <div class="jumbotron" id="showcasing" style="background: url('LHC.jpg') repeat-x top center;">
        <h1>It works !</h1>
        <p>Hooray</p>
        <p>
          <a class="btn btn-primary" href="#">Press me</a>
        </p>
      </div>

    </div> <!-- /container -->

    <!-- Bootstrap core JavaScript
    ================================================== -->
    <!-- Placed at the end of the document so the pages load faster -->
    <script src="bootstrap/assets/js/jquery.js"></script>
    <script src="bootstrap/dist/js/bootstrap.min.js"></script>
  </body>
</html>
```

Now the above html, is about the simplest one can get for bootstrap. And it looks like this:

Firstly let's adjust the simplest things we can - the LESS variables.

```
// added pbrian
@brand-back: #8a928e;
@brand-fore:  #8e393f;

@gray-darker:            lighten(#000, 13.5%); // #222
@gray-dark:              lighten(#000, 20%);   // #333
@gray:                   lighten(#000, 33.5%); // #555


@brand-primary:         @brand-fore;

                        ^^^^^^
                        This will roll out across the site.
```

And the result is :

Ok, nothing spectacular, but one line change gives us a new colour set across the board. What else can we do?

Fonts TBD

Images TBD

biblio https://raw.github.com/mikadosoftware/screengrab/master/screenshots/colorschemedesigner.png http://bootstrap.lesscss.ru/less.html http://copyhackers.com/2012/12/the-3-step-hack-for-startups-bootstrapping-their-design/ http://24ways.org/2012/how-to-make-your-site-look-half-decent/ http://www.sitediscount.ru/verso/index.html

Choose a color scheme: http://colorschemedesigner.com/

```
#####  Color Palette by Color Scheme Designer
#####  Palette URL: http://colorschemedesigner.com/#0Q51Rw0w0w0w0
#####  Color Space: RGB;



*** Primary Color:

   var. 1 = #FF9F00 = rgb(255,159,0)
   var. 2 = #BF8930 = rgb(191,137,48)
   var. 3 = #A66800 = rgb(166,104,0)
   var. 4 = #FFB740 = rgb(255,183,64)
   var. 5 = #FFCA73 = rgb(255,202,115)

*** Secondary Color A:

   var. 1 = #FFC700 = rgb(255,199,0)
   var. 2 = #BFA030 = rgb(191,160,48)
   var. 3 = #A68100 = rgb(166,129,0)
   var. 4 = #FFD540 = rgb(255,213,64)
   var. 5 = #FFE073 = rgb(255,224,115)

*** Secondary Color B:

   var. 1 = #FF6700 = rgb(255,103,0)
   var. 2 = #BF6A30 = rgb(191,106,48)
   var. 3 = #A64300 = rgb(166,67,0)
   var. 4 = #FF8D40 = rgb(255,141,64)
   var. 5 = #FFAB73 = rgb(255,171,115)


#####  Generated by Color Scheme Designer (c) Petr Stanicek 2002-2010


   @colorA1:  #FF9F00;
   @colorA2:  #BF8930;
   @colorA3:  #A66800;
   @colorA4:  #FFB740;
   @colorA5:  #FFCA73;

   @colorB1:  #FFC700;
   @colorB2:  #BFA030;
   @colorB3:  #A68100;
   @colorB4:  #FFD540;
```

```
@colorB5:  #FFE073;

@colorC1:  #FF6700;
@colorC2:  #BF6A30;
@colorC3:  #A64300;
@colorC4:  #FF8D40;
@colorC5:  #FFAB73;
```

We can also adjust the fonts site wide.

New fonts: http://www.google.com/fonts/

<link href='http://fonts.googleapis.com/css?family=Roboto' rel='stylesheet' type='text/css'>

font-family: 'Roboto', sans-serif; <link href='http://fonts.googleapis.com/css?family=Press+Start+2P' rel='stylesheet' type='text/css'> (ad nauseum example)

We can see the results (quite horrible) here:

http://coding.smashingmagazine.com/2013/03/12/customizing-bootstrap/

Building my masterpiece OK, so I want to transform this

Into this:

```
Tada
```

So lets start with Design basics - cribbed liberally from the folks on InterWebs

- Step 1 - Make it look good in Black and White first
- Step 2 - Write the right things
- Step 3 - Do less. No less than that.

Color theory Well, just colour wheels really.

Building my own bootstrap classes Its pretty obvious what means. But that does not mean its a good idea. Yopu want a sidebar, not a span6.

So we define our own classes, and use those in the HTML instead.

This is accounted for in Bootstrap by http://getbootstrap.com/css/#grid-less

http://ruby.bvision.com/blog/please-stop-embedding-bootstrap-classes-in-your-html

Building our own mobile aware classes Oh fuck it, really. This is the shit designers are supposed to obessess over. Here I stop. If I cannot make do with a main column and a sidebar I am going to have to go back to pen and paper.

3.3 browser-automation

3.3.1 Browser Automation

```
#!/usr/bin/env python
#! -*- coding: utf-8 -*-

###
# Copyright (c) Rice University 2012-13
# This software is subject to
# the provisions of the GNU Affero General
# Public License version 3 (AGPLv3).
# See LICENCE.txt for details.
###


from splinter.browser import Browser
browser = Browser()
browser.visit('http://www.office.mikadosoftware.com')
browser.click_link_by_href('/login')
openidbox = browser.find_by_xpath('''id('column2')/form/p/input[1]''').first
openidbox.fill('https://www.google.com/accounts/o8/id')
browser.find_by_value('Sign in').first.click()
#id="Email"
#id="Passwd"
browser.find_by_id("Email").first.fill('paul@mikadosoftware.com')
browser.find_by_id("Passwd").first.fill('empathy1')
browser.find_by_value('Sign in').first.click()

browser.uncheck('remember_choices')
browser.find_by_value('Allow').first.click()
print browser.title
print

for z in browser.cookies.driver.get_cookies():
    if z['domain'] == u'www.office.mikadosoftware.com' and z['name'] == u'session':
        print z['value']
```

TBC

3.4 html5

3.4.1 What is HTML5 and why does it matter?

HTML5 is a new set of standards and an ecosystem, all designed to work in the browser. It is *not* just a markup defintion, but a set of (javascript) APIs, a new way to work with attributes and define them, improved CSS,

So its improved HTML *and* javascript APIs that allow a browser to access almost everything on a device, from battery status to peer-to-peer video calls.

What is out there?

Biblio

- http://en.wikipedia.org/wiki/HTML5
- http://davidwalsh.name/html5-apis

A few Major APIs

Contacts - The HTML5 specification mentions that the Contacts API allows to have a common contacts repository in the browser which can be access by any web application. Selection - The selection API supports selecting items in DOM (supports CSS3 type of selectors), to be used along with JQUERY. Offline apps - This API allows marking pages to be available in Offline mode. This is useful if a resource requires dynamic processing. Indexed database - This API is meant for a database of records holding simple values (including hierarchical objects). Every record has a key and a value. An indexed database is supposed to be implemented using b-trees. Web SQL DB is no longer being pursued as part of HTML5 specification. Web workers - This API is meant to be invoked by web application to spawn background workers to execute scripts which run in parallel to UI page. The concept of web works is similar to worker threads which get spawned for tasks which need to invoked separate from the UI thread. Web storage - This specification defines an API for persistent data storage of key-value pair data in Web clients. Web sockets - This API used for persisting data storage of data in a key-value pair format for Web clients. Server-Sent Events - This API is used for opening an HTTP connection to receive push notifications from a server. These events are received as DOM events. This API is supposed to be used with Push SMS. XMLHttpRequest2 - This API is used to provide scripted client functionality to transfer data between a server and a client. Geolocation - This API is used to provide web applications with scripted access to geographical location information of the hosting device. Canvas 2D Context - This API provides objects, methods and properties to draw and manipulate graphics on a canvas drawing surface. HTML Microdata - This API is used to annotate content with specific machine-readable labels, e.g. to allow generic scripts to provide services that are customized to a page. Microdata allows nested groups of name-value pairs to be added to documents. Media Capture - This API is used to facilitate user access to a device's media capture mechanism (camera, microphone, file upload control, etc.). This only coves a subset of media capture functionality of the web platform. Web Messaging - This API is used for communications between browsing contexts in HTML documents. Forms - The Forms API can be used with the new data types supported with HTML5. File API - The File APIs are used by the browser to provide secure access to the file system.

CHAPTER 4

Testing

4.1 burpsuite

4.1.1 Burp Suite

A java based, advanced proxy / intercept / wiretapping tool for watching your REST API talk and debug.

Its less user-friendly than charles, but it runs on FreeBSD...

It is also recommended by tpatcek

Options are stored in `.java/.userPrefs/burp/prefs.xml`

Installing a CA

We force Chrome to visit a site where a CA error will throw a User Warning. Basically this means almost any https site. The browser shows a warning- we click through to details of the certificate (it is the one sent by burp, and not the one recieved by burp from https://google

Then export the certificate, and now visit the "update cert" in the browser and import what we just saved.

4.2 testing

4.2.1 Testing

Nose is the standard for Python testing.

http://stackoverflow.com/questions/8054512/anyone-know-how-nosetests-m-i-and-e-work
https://groups.google.com/forum/?fromgroups=#!topic/nose-users/1IGF0xliWiA

Sphinx and DocTests

4.2.2 Network Testing

iperf is a useful tool for testing big bandwidth metrics

Fairly stroagihtforeward to put onto FreeBSD

https://github.com/esnet/iperf

Ubuntutu is more ocmplicated

sudo apt-get install uuid-dev git-core sudo apt-get install autotools-dev automake sudo ln -s /usr/bin/aclocal /usr/bin/aclocal-1.14

4.3 Jasmine unit testing for Javascript

4.3.1 install

We shall install the stand alone version first and go from there.

Also - jasmine-0jquery plugin

4.3.2 Biblio

- https://github.com/pivotal/jasmine/wiki
- https://github.com/velesin/jasmine-jquery

http://addyosmani.com/writing-modular-js/

4.3.3 AMD

Basically a single global fuinction -> define

    define(moduleId, dependaices, definitonFunc);

preferred is anonymous modules:

    define(dependancies, defintionfunciton)

    return the module is important

4.3.4 The Globals and script tags problem

JS has grown a lot since I started using it (95?) there is a lot more (seriously I used notepad)

Hybrid Vs return values

4.4 webtest

4.4.1 webtest

Writing a WSGI app Writing a WSGI *server* Testing a WSGI server - http://wsgi.readthedocs.org/en/latest/learn.html Writing WSGI Middleware

Testing the whole thing:

```
Locally
On HTTP
Load test
```

Let us imagine we have a nice wsgi app. Lets imagine it is a Flask app. We want to test that the "hello world" actually returns hello world.:

```
>>> from flask import Flask
>>> app = Flask("myapp")
>>> wapp = app.wsgi_app

def build_environ():
    """
    We are playing at a low level with WSGI - wanting to wrap repoze.
    http://www.python.org/dev/peps/pep-0333/

    To test manually we need to generate correct HTTP Headers
    """
    import StringIO
    request_fo = StringIO.StringIO()
    err_fo = StringIO.StringIO()

    ###wsgi reqd keys and default valus
    wsgi_specific_headers = {"wsgi.version": (1,0),
                             "wsgi.url_scheme": "http",
                             "wsgi.input": request_fo,
                             "wsgi.errors": err_fo,
                             "wsgi.multithread": False,
                             "wsgi.multiprocess": False,
                             "wsgi.run_once": False
                            }

    ### key = HEADER (RFCLOC, NOTNULL, validvalues)
    HTTP_HEADERS = {"REQUEST_METHOD": "GET",
                    "SCRIPT_NAME": "module",
                    "PATH_INFO": "/cnxmodule:1234/",
                    "QUERY_STRING": "",
                    "CONTENT_TYPE": "",
                    "CONTENT_LENGTH": "",
                    "SERVER_NAME": "1.2.3.4",
                    "SERVER_PORT": "80",
                    "SERVER_PROTOCOL": "HTTP/1.1"
                    }
    d = {}
    d.update(wsgi_specific_headers)
    d.update(HTTP_HEADERS)
    return d
```

4.5 webtest-cookie

4.5.1 Cookies, Testing and security

I wish to use a cookie to store a random UUID that represents a server-side session. This will then be looked-up server side and the server can trust the user info gathered (reasonably so)

This works as a useful cut-off point between the authentication system and everything else. We can test by sending in a fake session cookie.

Example

If we create a simple wsgi app that just says hello, and returns its environ.

```
def methodapp(environ, start_response):
    sessid = capture_session(environ)
    userd = lookup_session(sessid)
    if userd:
        userstr = "You are user %s" % userd['name']
    else:
        userstr = "Dont know user"

    s = userstr + "</br>".join(["%s::%s" % (k,v) for k, v in environ.items()])
    status = "200 OK"
    respstr =  s

    import random
    i = random.randint(0,9)
    response_headers = [('Content-type', 'text/html'),
                        ('Content-length', str(len(respstr))),
                        ('Set-Cookie', "cnxsessionid=00000000-0000-0000-0000-00000000000%s" % i)
```

Now we shall write a simple lookup (development only but it will do for now)

```
def capture_session(env):
    """
    >>> env = {"HTTP_COOKIE": "cnxsessionid=00000000",}
    >>> capture_session(env)
    '00000000'

    >>> env = {"HTTP_COOKIE": "name=value",}
    >>> capture_session(env) is None
    True

    >>> env = {"foobar": "wibble",}
    >>> capture_session(env) is None
    True

    """

    if "HTTP_COOKIE" in env:
        cookiestr = env.get("HTTP_COOKIE")
    else:
        return None

    cookieset = Cookie.BaseCookie(cookiestr)

    if CNXSESSIONID in cookieset.keys():
        sessid = cookieset[CNXSESSIONID].value
    else:
        return None

    return sessid

def lookup_session(sessid):
    """
    We would expect this to be redis-style cache in production
    """
    onehour = datetime.timedelta(hours=1)
```

```
    developercache = {"00000000-0000-0000-0000-000000000000":
                        {'name': 'Paul',
                         'useruri':'/users/00000000-0000-0000-0000-000000000000',
                         'starttimeUTC': datetime.datetime.utcnow(),
                         'starttimeUTC': datetime.datetime.utcnow()+onehour}
                       }
    try:
        return developercache[sessid]
    except:
        return None
        #in reality here we might want to try the redis chace or contact user server
```

Now lets test this.

```
from webtest import TestApp

def header_from_cookiejar(cj):
    """
    ignoring domains etc for now
    """
    cookies = []

    for ck in cj:
        cookies.append("%s=%s" % (ck.name, ck.value))

    ident_header = {'COOKIE':';'.join(cookies)}
    return ident_header



from simpleapp import methodapp
fakeuuid = "00000000-0000-0000-0000-000000000000"

app = TestApp(methodapp, use_unicode=True)
ident_header = {'COOKIE':'cnxsessionid=%s' % fakeuuid}
app.cookies.update(ident_header)
#CookieJar.add_cookie_header
r = app.get("/")
print r

### get a new cookie header
cj = app.cookiejar
r = app.get("/", headers=header_from_cookiejar(cj))
print r


### get a new cookie header
cj = app.cookiejar
r = app.get("/", headers=header_from_cookiejar(cj))
print r


### get a new cookie header
cj = app.cookiejar
r = app.get("/", headers=header_from_cookiejar(cj))
print r
```

```
### get a new cookie header
cj = app.cookiejar
r = app.get("/", headers=header_from_cookiejar(cj))
print r
```

Here we see we are passing a header of the format desired into the test case. And the output from the webtest shows that the environ held a cookie of the right form:

```
Response: 200 OK
Content-Type: text/html
You are user PaulHTTP_COOKIE::cnxsessionid=00000
```

We should use a 2x2 grid to determine responses etc (TBC)

Basics

We do not store sensitive information in a cookie. Thats not what they are for, it is totally out of our control and we can never know that the encryption loop has not been broken [1]

This leaves us with only one option for session cookies - a random number that is the key to looking up the session details on the server somehow. Even this number will need to be in a suiffiociently large space that guessing is impractical. And we will need to ensure https traffic end to end to prevent traffic-sniffing.

So, we shall use a tiered-lookup-cache. I would expect it to go like this

1. User GET request to restricted page /secret
2. User receives 401 auth request, and by some means authenticates
3. Once authentication is completed, we fire server-side "after-auth"
4. after-auth generates a session
   - user_uri, session_starttime, sesssion_endtime, randomUUID
   - store that in local session cache
   - set response header for cookie with randomUUID
5. next request includes sessioncookie
   - lookup randomUUID in local session cache
   - set REMOTE_USER in environ,
6. scaling - I would only recommend moving to a network local redis style lookup and dropping the local cache - dual cache invalidation for such likely small speed up is foolish
7. logout
   - delete the session details from cache
   - unset the
   - add previosu randomUUID to the users *already ised list* (this is where the bloom filter may help)

[1] Encryption loop is my own term. It is *very* unlikely that the OpenSSH implementation of AES is flawed or explotable. It is really quite likely that the methods of using AES and setting headers and managing requests and decrypting using a salt stored in a ini file and all the other stuff *is* flawed, mostly because I wrote it.

Onwards work

The use of sensible cookie handling for sessions is a good thing, however, there are many many other aspects to a secure system, ranging from acls on the server,

Philosophy

I want to digress just a little here. I am not a security researcher and so I have no special expertese in this area. However I am trying to minimise the errors made in a security system, maximise the likelihood that developers (as opposed to security researchers) will find bugs, and make plain the trade offs being made.

So, encryption is very very hard to get right even from basic building blocks like OpenSSH (basically keyczar and nacl are wrappers around such functionality to prepackage choices of how to put the building blocks together)

I would prefer to thus avoid relying on encryption for a simple reason - I can easily get it wrong, rely on it being right and never spot my flaws. In many other areas I am likely to spot a flaw (oh, look a SQL stmt using string formatting)

I am making a trade off here - that the cache lookups will not become impossible to manage

Bibliography

Never build unnecessary crypto into your design https://news.ycombinator.com/item?id=4680444

OWASP top 10 https://www.owasp.org/index.php/Top_10_2013-T10

AES Oracle attacks https://news.ycombinator.com/item?id=1687770

4.6 writing_documentation

4.6.1 Documentation

If it is not automated, it does not exist. If it is not documented, it may as well not exist.

Sphinx is a really useful document generator. It is linked to readthedocs.org

http://sphinx-doc.org/markup/ https://bitbucket.org/birkenfeld/sphinx-contrib

CHAPTER 5

Keeping Tabs on Business

5.1 ExecutiveDashboards

Building a Dashboard to see what is happening in your system / business.

5.2 marketing

5.2.1 Marketing Mikado

I have three business personas

- Paul of Mikado Software, an experience CTO / Python hacker and Agile Coach
- Paul of OSS4Gov.org, a campaigner
- Paul of ShowYouWhatIMean (sywim.com) - SaaS entrpreneur

5.2.2 Brand Persona

Mission

Vision

Brand

- Who are you
- what do youbeleive in
- what do you say and how do yousay it

Target Audience

Market Niche

http://i.imgur.com/TNcqG4k.png

Team Development Questionnaire

Why do we exist?

What difference do we make in our customers' lives?

What do we believe in?

What do we NOT believe in?

How do we want a customer to feel about us?

Who are our most ideal customers?

Who are we not right for?

How are we different?

5.3 Agile

5.3.1 A more nuanced view on Agile

Lets start with the "anti" brigade. No not the nut jobs, the saner lot. http://www.sdtimes.com/content/article.aspx?ArticleID=68991&page=1

This is Agile for the 68%. THe first std deviation of software engineering.

The three steps of software enginerring maturity

Three std deviations.

CHAPTER 6

The bits I cannot think of a section for

6.1 wsgi

6.1.1 An example of running doclit

We can see

```
>>> from doctest2.doclit import doclit
>>> starting_wsgi = doclit.import_docs("starting_wsgi.rst", "starting_wsgi")
>>> starting_wsgi.foo('HH')
'HH'
```

from gunicorn.app.pasterapp import paste_server from paste.deploy import loadapp

app = loadapp('config:foobar/foobar.ini', relative_to='.') paste_server(app, host='0.0.0.0', port=8002)

6.1.2 Writing a simple WSGI app

https://bitbucket.org/vithon/vithon-forum/src/abab8f2a7aef/viforum/forum.py

What should have happend?

start_response probably should have been left out, and ... well thats the hard part - you need to be able to send back up the chain a stream of bytes (body) *and* a environ / header dict So it does need two channels.

Support for non-blocking requests? A (thread) in an event-based server needs to have mechanisms for halting its own processing while an I/O event occurs, and then restarting

This leads to rewrites of pretty much everything.

Application

An "app" is *only* a piece of code that calls start_response and so begins the response process.

This is how middleware can operate - by deciding if or not to respond instead of the "officially configured" app.

I think this "officially configured" app is also the source of much confusion.:

```
>>> from doctest2.doclit import doclit
... def paul_app(environ, start_response):
...     """ """
...     #do something with environ
```

```
...     resp_headers = {'Content-Type': 'text/html'}
...     start_response("200 OK",
...     return "Hello World"
```

Serving the app

```
>>> from waitress import serve
... serve(paul_app)
```

Adjusting a environ on way in

TBC

Adjusting environ on way out

TBC

Setting and unsetting cookies

See RSSSO

Secure cookies

Some people think that secure cookies is about encrypting the text on the cookie so bad guys cannot read it. Frankly, this is nonsense. You will make a mistake, and they will read it. So we have an issue - which is faster an in-memory redis lookup or a decryption of a *really secure* piece of text.

This looks like an interesting test...

6.1.3 THis is starting wsgi

```
>>> def foo(s):
...     return s
```

6.1.4 The unusual behaviour of urljoin

os.path.join behaves in a pretty robust ways.

(examples)

urlparse.urljoin seems not to do so.

```
>>> urlparse.urljoin("http://localhost/module/", "cnx1234")
'http://localhost/module/cnx1234'
```

I expect the above behaviour.:

```
>>> urlparse.urljoin("http://localhost:8000/module", "cnx1234")
'http://localhost:8000/cnx1234'
```

Woooo. What happened to module?:

```
>>> urlparse.urljoin("http://localhost.com/module/", "cnx1234")
'http://localhost.com/module/cnx1234'
```

one last gotcha:

```
>>> urlparse.urljoin("http://localhost.com/module/", "/cnx1234")
'http://localhost.com/cnx1234'
```

http://stackoverflow.com/questions/10893374/python-confusions-with-urljoin

So for RESTful navigation we want something else...

6.1.5 webtest

Testing a WSGI server - http://wsgi.readthedocs.org/en/latest/learn.html

Testing the whole thing Locally On HTTP Load test

Let us imagine we have a nice wsgi app. Lets imagine it is a Flask app.

We want to test that the "hello world" actually returns hello world.

```
>>> from flask import Flask
>>> app = Flask("myapp")
>>> wapp = app.wsgi_app

def build_environ():
    """
    We are playing at a low level with WSGI - wanting to wrap repoze.
    http://www.python.org/dev/peps/pep-0333/

    To test manually we need to generate correct HTTP Headers
    """
    import StringIO
    request_fo = StringIO.StringIO()
    err_fo = StringIO.StringIO()

    ###wsgi reqd keys and default valus
    wsgi_specific_headers = {"wsgi.version": (1,0),
                             "wsgi.url_scheme": "http",
                             "wsgi.input": request_fo,
                             "wsgi.errors": err_fo,
                             "wsgi.multithread": False,
                             "wsgi.multiprocess": False,
                             "wsgi.run_once": False
                            }

    ### key = HEADER (RFCLOC, NOTNULL, validvalues)
    HTTP_HEADERS = {"REQUEST_METHOD": "GET",
                    "SCRIPT_NAME": "module",
                    "PATH_INFO": "/cnxmodule:1234/",
                    "QUERY_STRING": "",
                    "CONTENT_TYPE": "",
                    "CONTENT_LENGTH": "",
                    "SERVER_NAME": "1.2.3.4",
                    "SERVER_PORT": "80",
                    "SERVER_PROTOCOL": "HTTP/1.1"
```

```
                    }
    d = {}
    d.update(wsgi_specific_headers)
    d.update(HTTP_HEADERS)
    return d
```

6.2 wsgi-nonblocking

6.2.1 Non blocking

How can we simply improve the performance (concurrent requests) of a WSGI server?

The simplest issue is - threads. OS level threads are realtively speaking, cumbersome and high process. Our Python interpreter already sits in one happily churning away, and to create another is an "expensive" process.

Pseudo-threads, like greenlets, can provide an answer.

installing gevent (FreeBSD)

Firstly install system-wide libevent2 - I find simplest to install /devel/py-gevent and ensure all compile well outisde pip.

Then - pip does not look for the libevent system files so we tell it where to look.

$ export CFLAGS=-I/usr/local/include $ export LDFLAGS=-L/usr/local/lib (http://kdl.nobugware.com/post/2011/11/15/compile-gevent-osx-or-freebsd-pip)

Now pip install inside the venv and all should be well.

Start simple

Simplest app we can

```
1     """
2     Stolen from PEP333
3     NB - same applies here - we are returning an instance of a class that
4     is an iterable - we do not return the class)
5     """
6     def __init__(self, environ, start_response):
7         self.environ = environ
8         self.start_response = start_response
9
10    def __iter__(self):
11        status = "200 OK"
12        response_headers = [('Content-type', 'text/plain'),
13                            ('X-paul','pbrian')]
14        self.start_response(status, response_headers)
15        yield "Hello"
16        time.sleep(0.1)
17        yield "World"
```

We run it using wsgiref

```
#import greenapp
import simpleapp
from wsgiref.simple_server import make_server

#app = greenap.myapp

app = simpleapp.myapp

httpd = make_server('localhost', 5000, app)
httpd.serve_forever()
```

We can run the funkload ... (funkload needs a seperate article) fl-run-test testcase.py

Bench - concurrent users

fl-run-bench <unittestfile> <ClassinFile>.<testmethodinclass>

fl-run-bench testcase.py MyWSGITest.test_simple

Results

I can see my plain WSGI server work quite acceptably at 20+ concurrent users if total response time < 0.01 seconds However when response time hits 0.1 seconds, I get a 90% fail rate, as threads are sleeping (ie processing) whilst the new requests come in.

Ok, sprinkle on magic pixie dust

lets add in greenlets to the application...

```
from gevent import monkey; monkey.patch_all()
import time



class myapp(object):
    """
    Stolen from PEP333
    NB - same applies here - we are returning an instance of a class that
    is an iterable - we do not return the class)
    """
    def __init__(self, environ, start_response):
        self.environ = environ
        self.start_response = start_response

    def __iter__(self):
        status = "200 OK"
        response_headers = [('Content-type', 'text/plain'),
                            ('X-paul','pbrian')]
        self.start_response(status, response_headers)
        yield "Hello"
        time.sleep(0.1)
        yield "World"
```

And, nothing much changes in the bench tests...

This is because the thread that decides to sleep is able to be a greenlet, the server is still creating OS threads, becasue the server does not know about greenlets. Lets fix that/

```
### Now we need to run a server that co-operates in threads

import greenapp

#from wsgiref.simple_server import make_server
from gevent import pywsgi

app = greenapp.myapp


httpd = pywsgi.WSGIServer(('localhost', 5000), app)
httpd.serve_forever()
```

Now, we can go from 10 to 100 threads before seeing any problems.

An order of magnitude improvement on the same laptop.

Follow on

How many threads are spun up? How do i see those threads Will dtrace help

Biblio

http://bottlepy.org/docs/dev/async.html

6.3 Show your workings

I spend quite a lot of time working out how to do things in code. Which tends to make it look like I am not doing much *producing* of code

Which means I want a nice way of "showing my working"

1. I have a story to do
2. If it is not trivial, I need to
   (a) experiement with different solutions
   (b) measure, collate the results
   (c) publish the results.

There is no ground-breaking stuff in here. Just a nice way of saying how to do stuff.

Think ... each experiment should result in a blog post like a web framework shoot-out.

6.3.1 Examples

Link to a Python file

Here is a `link` to a python file stored next to this ReSt text file.

Include a python file

This is a snippet from a file

```
def hello_world(name):
    """
    A new way of saying hello
    """
    return "Hello World and %s" % name

def dontshowthisone():
    """
    shown in a full include, not in by-object include
    """
    pass
```

This is including just one object

```
def hello_world(name):
    """
    A new way of saying hello
    """
    return "Hello World and %s" % name
```

This is including just lines

```
def hello_world(name):
    """
    A new way of saying hello
    """
    return "Hello World and %s" % name
```

We can also use : include : to include a whole file but it is not really meeting our purpoases

6.4 misc

6.4.1 Misc Notes

Trouble building pysqlite on freebsd

http://unix.stackexchange.com/questions/53336/pysqlite-install-error-on-freebsd-in-virtualenv

Useful Python Reads:

http://stackoverflow.com/questions/231767/the-python-yield-keyword-explained/231855#231855

6.4.2 Domain name system

Get your own domain.

Seriously, stop mucking about and start getting some google juice.

Now, the simplest way to start is using a nice helpful domain registrar. Do not touch GoDaddy with a long stick (See **'HN <http://news.ycombinator.com'_**). I use easily.co.uk. A bit pricey but they work OK. There are better ones but for the price difference, its hardly worth my time moving.

Now, what is worth my time is a good API based way of changing the DNS names.

I *used* to run my own DNS servers in house. Which was fine, everyone was happy. But like all cloud things, barely worth the effort now - I like to use Amazon Route 53. I have some code for working with them, will OSS it soon I hope.

6.4.3 Running a Small Office / Home Office Network

Well, this article has changed in the 8 or so years since I last wrote it. Way back then, we ran everything in-house. Firewall, email, shared files, DNS and DHCP, and phones, oh dear phones !

These days... barely any of it. Most of us have GMail on the web browser and on our phones, share files via dropbox and skype or video chat. Pretty much all that is left is the firewall and our own personal workstations.

Building a FreeBSD laptop from scratch with Salt-stack

Firewalls, routers and analytics

What is important now, and possibly more than ever, is the *analytics*. This article is about the ways to monitor your SoHo office.

It's useful to look back on these articles - but this is by far the biggest change. I have notes on Samba, and cherrypy. All of which have almost no relevance now.

File Storage

Dropbox is *fine*. But does not run on BSD. So I can look at bittorrent Sync, or seafile.

I will get back to you :-)

6.4.4 Web Servers - why has apache lost?

Wow, this clean out of files is really finding some golden oldies.

I did have a write up on Apache, setup, configuration etc. But... these days I use nginx and **'Micro-web services <>'_** to great effect.

So please see *Nginx - load balancing* and lets leave poor old apache alone for a while.

6.4.5 How to run a simple backup strategy

> You will never be forgiven for losing data when people want it. Even if they never looked at it for a year previously.

1. git is not a backup service. It is a development history. 2.

3 kinds of backup - code - svn mostly 1GB - 1mb pday max delta - strutured data - database - 1gb+ 10mb pday - semi structured - emails - 200gb - 100mb pday - unstrucutred - files docs etc

I want the first two off site online backups. Use replicaton for database

Samba - daily backup to local, using yesterday, t-2 t-3 approach for 5 days. Issues: - verification. THe cheapest way is to mount the yesterday backup as a samba file and people can come and look at it - off-site - how do we keep yestedays backup off site. Or do we accept that as a potential loss and just keep the rolling backup off site

Rolling Archive - archive to disk a snapshot each 1st month and keep.

Never remove Archive all files > 1yr old, leaving placeholder txt files? These go to Ongoing archiveof ll files with modification date > 1yr

Email backup - ? I want exchange totally isolated - with all smtp traffic handled by postfix sendmail both in and out. Then i want all mail to be side tracked to mysql Then how do we backup pst files?

SPec

Our Backup strategy

We have 4 types of data to contend with and 4 types of loss to defend against

Data types

1. 'code' - developed Code, config files for servers etc
2. structured data - SQL basically
3. email data - I would prefer it if it was MailDir, but we have got pst files or rather a huge NTBackup file
4. file server documents - ie Samba

Data Loss

1. Disk failure
2. Accidental deletion
3. file corruption
4. site loss

The strategies for all 4 loss scenarios are very different

1. Disk failure - either - striping (ie RAID 5, or google-like map-reduce) - mirrored backups (ie a copy on big backup machine)
2. accidental deletion - mirrored and temporarily preserved backups (Daily snaphot, rotated weekly) - mirrored and permanently preserved backups (archive snapshot kept forever)
3. Random file corruption - the strategy is same as accidental deletion, however the approach to detection

   is to use something like mtree (hashes file signatures)
4. site loss - disaster recovery using replacement hardware, new site, and a full mirrored backup

Extra strategies

1. archiving unused documents
2. online backups only go to hot swap not to data dump By this I mean we only backup over the wire to a hot application, to a location where if we suffer site loss is very simple and easy to set up.

   ie replicate MYSQL to a MYSQL server in US/docklands mirror samba to another samba server that is sitting in a spare office

We do not want to be in a situation where we have dumped samba to a disk in docklands And then lose the site, decide to buy new servers and put them in alans house, have to wait hours/days to download that disk over alans DSL. We should have servers in Alans house and mirror the backup to that hot server. So if we need to we just move to alans house and hey presto the data is alsready there

Strategy

1. Backup all to large array of disks at far end of office
2. Install tape drive on large array and perform backup
3. online backup of svn, mysql

4. backup tapes are rotated on daily basis out of office and a snapshot kept every 14 days

Bibliography

Disk failure (quite common - see biblio. Basically google says its more common than you think. So given MTBF of 100,000 hours, thats bascially 1/2 of all disks will fail in 100,000hours in 10 years. We have close on 100 disks (pcs, latops, raids) so 50 will fail in next 10 years, expect 5 to fail each year). Most MTBF is 300,000 - 500,000. So 1 disk a year from MTBF, but google says more. We have lost 4 since I got here, so 1-2% failure rate is close.

Google results http://labs.google.com/papers/disk_failures.pdf

Blog on google results http://storagemojo.com/2007/02/19/googles-disk-failure-experience/

How to calculate RAID MTBF from HDD MTBF http://answers.google.com/answers/threadview?id=730165

General article http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9073158&source=rss_topic17

http://www.freebsd.org/doc/en/books/handbook/backup-strategies.html

6.4.6 Cabling and other hardware

Like a number of others this is for historical purposes, but yes, I really did run a real business with a "server room" that was cooled using a aircon unit we just hauled in from B&Q.

Power, and Cooling

At some point a small company will move from having a few laptops, to having a *server*.[#]_ And at some point after that they get another one, and an IT manager, and soon they have to think hardware.

A Pentium4 CPU can hit 90C, and stay there under heavy load. The fans and other noise are designed to throw that heat away from the CPU. One of the most common causes of computer failure is overheating in summer. [1]

So in essence we are running an office full of bar heaters.

How do I cool this all down ?

Its all down to Watts. 1 watt is a rate of power consumption equal to 1 joule per second

on average a single server will consume 60-120W (yes that is a big varience, but dual cores and load all make a difference. Measure it using a kill-a-watt meter for ease) .

so, 120 W = 120*3.41 = 409 BTU for a single server So, 15 servers, all running at once will pump out about 9000 BTU 9000 BTU is the bottom end rating for a DIY air conditioner. Do you see where we are going?

So if I have a 6000 BTU rated air conditioner I can in theory have 30 servers on at full blast and the temperature will stay the same.

[1] Admittedly that point gets further off as years pass - web

based services really do offer a small business a real benefit if they are willing to risk hosting. [Ha - can you see my bold prediction of the cloud-computing era. Well at least I saw it's benefits :-)]

less commonly reported. My assumption is that in the UK most of the year it is cold enough to get away with no (or insufficient) dedicated cooling. When summer hits, the servers pay the price. In a hot climate, that is an ever present threat and is factored in from the start.

6.4.7 Debugging .core files for when it all goes wrong

This is a short tutorial on debugging compiled c / c++ programs as a last resort for finding out what is going on.

It is a tough nut to crack but even with some simple tools and a bit of perseverance you can make decent headway into a problem that otherwise has no traction.

Debugging C/C++ programs

SOmetimes a program dies, and leaves a file of its last memory state. Using the gdb debugger we can see what happened

recently I installed firefox 3 from ports. Then it died when I tried printing.

```
$gdb core firefox-bin.core
```

```
# gdb core firefox-bin.core
GNU gdb 6.1.1 [FreeBSD]
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-marcel-freebsd"...core: No such file or directory.

Core was generated by `firefox-bin'.
Program terminated with signal 11, Segmentation fault.
#0  0x299c5c0b in ?? ()
```

The problem here is the last call made before everything went blooey was in memory address 0x299c5c0b (Hex for 698113035). unfortunately if this binary had been compiled with debugging symbols I could see the name of that function not ??

Normally bt (backtrace) would help but it is useless too...

so I recompiled firefox with debugging symbols, and tried to break it again. (I altered the options file in /var/db/ports/firefox3/ - WITHOUT_DEBUG=false. Yes double megatives do not help my mental state right now)

bibliography

http://www.freebsd.org/doc/en/books/developers-handbook/debugging.html http://www.freebsd.org/doc/en/books/developers-handbook/kerneldebug-gdb.html http://www.unix.com/unix-advanced-expert-users/19128-how-do-core-dump-analysis.html http://www.unknownroad.com/rtfm/gdbtut/gdbuse.html

6.4.8 pkng for FreeBSD

Well, this *was* a really good article on mirroring FreeBSD's CVSUP architecture, to keep bang upto date with any changes. Unfortunately release 10 has got rid of that entirely, so we shall instead cover pkgng.

Now I have a bit of fondness for pkgng because my first accepted patch to *saltstack* was a fix for the pkgng setup. And I am working on it as part of pyholodeck

More to come...

6.4.9 emacs

Aha - at last, a subject that if I write about it five or more years ago, the article is still relevant. Nothing to worry about here... :-)

You gotta love emacs, it never changes...

.emacs file

this is the "init file"

Setting variables in the init file -

setting modes in the init file. I like to use 80 char line breaks so,

```
      (add-hook 'text-mode-hook
        '(lambda () (auto-fill-mode 1)))

(add-hook 'text-mode-hook 'turn-on-auto-fill)
```

OK, each mode has a hook that is a variable holding a list of functions that are called when the hook activates. in this example when we start text-mode we shall execute the list of one function, a lambda that simply sets auto-fill-mode to 1

OK, what is the column after which I should break - well I set it at 75

```
(setq fill-column 75)
```

examining variables

C-h v <VARNAME>

6.4.10 Email

And here again is my trip down memory lane, interupted by small things like Google Mail. I no longer bother with running my own email servers, keeping the clam-av uptodate, handling spam, worst of all synching with my mobile. All done for me, 40 bucks a year.

Admittedly my email is much easier for the NSA to read and for Google to profile me and sell me ads, but ... well. Hmmmm.

Anyway.

Email is the serial-killer app of the Internet, it keeps on being the one thing everyone wants in each generation. It has become part of everyday life and it is vital to manainging correctly.

The correct way is no longer, yourself. So we shall cover the following later:

Todo

Mail encryption. Keeping safe what you say to criminals and neer-do-wells.

Todo

Mail rules - setting up and managing Google Apps for business email rules.

6.4.11 NFS File sharing

Summary

First there was NFS - it was written to share files over trusted networks (cos back then everyone trusted everyone else, and wore flowers in their hair.) The program that supports this is nfsd.

Then came SMB/CIFS which did not trust everyone, and was built to work with WIndows machines. THe program that supports this is SAMBA.

Then people started to really miss NFS, so they made it work with Kerberos.

This chapter starts with NFS, and then moves onto NFS with Kerberos. For file sharing within a windows domain see the chapter on SAMBA.

NFS Plain

We need a server and a client.

Server This needs to run nfsd (which handles incoming requests) and mountd (which deals with actually sending receiving the files). rpcbind is also needed so clients can see what port the nfsd is on.

/etc/rc.conf

```
rpcbind_enable="YES"
nfs_server_enable="YES"
mountd_flags="-r"
```

client /etc/rc.conf

```
nfs_client_enable="YES"
```

Sharing On the server we determine which directories we want to 'share' or 'export'. These are listed line by line in /etc/exports along with the hosts that can see them

```
/workspace/projects/projects/build_system/trunk/bsdbasebuild/new belmarsh belmarsh2
```

maproot The -maproot=root flag allows the root user on the remote system to write data on the exported file system as root. If the -maproot=root flag is not specified, then even if a user has root access on the remote system, he will not be able to modify files on the exported file system.

To connect. Easiest way is to reboot both machines then

```
mount server:/path /local/mnt/point
```

Trouble shooting

- symlinks.

This gets *everyone*. NFS cannot deal with (in fact for security is designed not to deal with) exporting from the server a path witha symlink in it. If you find that the mounting on the client is failing with *permission denied*, suspect this first. To check tail /var/log/messages. You will almost certainly find a message like

```
Apr 27 12:39:35 paullaptop mountd[643]: mount request denied from 10.137.1.42 for /usr/home/pbrian/wo
```

now if I look at my /etc/exports

```
/root/workspace/projects/projects/build_system/trunk/bsdbasebuild/new -maproot=root belmarsh belmarsh
```

aha!

lrwxr-xr-x 1 root wheel 22B Dec 11 11:11 workspace -> /home/pbrian/workspace

(yes, I am evil and am running as root)

so I solve it by altering the /etc/exports to

then I give NFS a kick (do this whenever adjusting /etc/exports)

```
kill -HUP `cat /var/run/mountd.pid`
```

bibliography

http://www.freebsd.org/doc/en/books/handbook/network-nfs.html http://www.freesoftwaremagazine.com/columns/securing_nfs http://www.the-labs.com/FreeBSD/JailTools/cookbook.html

6.4.12 Hardware

Historical Note

At first I thought this was going to be running your own servers in your own (air-conditioned) rooms. I mean how 20th Century But its local mini servers, RPis and the like. Very 21st.

Todo

links on aircon

6.4.13 What should I buy as a server?

Depends

http://www.soekris.com/ - small form, firewalls routers etc. Freebsd compatible

beagleboard - low power, single board computer. - hobbyist, aimed at wearable computers etc. http://arstechnica.com/open-source/news/2008/08/ti-launches-hackable-beagle-board-for-hobbyist-projects.ars

How to determine what hardware is running?

sysctl -a | grep -i hw.model hw.model: Intel(R) Pentium(R) 4 CPU 1.90GHz

sysctl -a | grep -i memory Virtual Memory: (Total: 2161196K, Active 56164K) Real Memory: (Total: 19760K Active 10812K) Shared Virtual Memory: (Total: 6184K Active: 4760K) Shared Real Memory: (Total: 4944K Active: 3668K) Free Memory Pages: 219288K hw.cbb.start_memory: 2281701376 p1003_1b.memory_protection: 0 p1003_1b.shared_memory_objects: 1

That is usually enough for one day.

#sysctl -ad Will show all measures and a description

6.4.14 biblio

http://www.freebsd.org/cgi/man.cgi?query=sysctl&sektion=8 http://www.onlamp.com/pub/a/bsd/2005/01/13/FreeBSD_Basics.html

6.4.15 read the source

ee /usr/src/sys/sys/sysctl.h

6.4.16 Java

Historical Note

I need to update the URLs etc, but the same annoying install process still exists.

Most awkward

Back in the mists of time (or 2006) Sun released its Java spource code (the compiler / VM etc) as open source. This Oracle does not license its version of the Java SDK / VM as open source, except for "personal" use. This means they can try and ensure they still get paid for enterprise Java under Installing Java is not that simple. I beleive that OpenJDK is pretty simple to install, but that is not the supported Oracle version of the JVM, and as I want to use Java for Oracle's ODI I guess I am going to do this the long winded way.

For various license-based reasons (and no I don't know and don't care right now) several parts of the java source need to be downloaded manually.

```
Due to licensing restrictions, certain files must be fetched manually.

Please download the Update 3 Source from
http://www.java.net/download/jdk6/6u3/promoted/b05/jdk-6u3-fcs-src-b05-jrl-24_sep_2007.jar
and the Source Binaries from
http://www.java.net/download/jdk6/6u3/promoted/b05/jdk-6u3-fcs-bin-b05-jrl-24_sep_2007.jar
and the Mozilla Headers from
http://www.java.net/download/jdk6/6u3/promoted/b05/jdk-6u3-fcs-mozilla_headers-b05-unix-24_sep_2007.j
.

Please open http://www.oracle.com/technetwork/java/javase/downloads/index.html
in a web browser and follow the "Download" link for
"JDK DST Timezone Update Tool - 1_3_45" to obtain the
time zone update file, tzupdater-1_3_45-2011n.zip.

Please download the patchset, bsd-jdk16-patches-4.tar.bz2, from
http://www.eyesbeyond.com/freebsddom/java/jdk16.html.
```

```
Please place the downloaded file(s) in /usr/ports/distfiles
and restart the build.

(NB halfway through the compile you will be asked to download diablo-caffe
manually too.)
```

So lets do that. Download the above as needed. Please note you get the above instructions from running $make install clean. Run this yourself on your source as the page may be out of date.

You will be asked to agree to a (non-free, personal use only) license.

You may want to use OpenJDK instead. I am not doing so as I am aiming at a Oracle application build and am guessing it will have less problems. I shall compare and contrast the OpenJDK and the Oracle JDK in a few months time.

6.4.17 Keeping src up to date

(Re)build a custom Kernel

1. Make sure we have latest sources

```
$ cat src-supfile

*default tag=RELENG_9
*default host=cvsup3.uk.freebsd.org
*default prefix=/usr
*default base=/var/db
*default release=cvs delete use-rel-suffix compress

src-all

$ sudo cvsup src-supfile
```

RELENG_9 is the STABLE release of FreeBSD. We shall stick with that.

We want to (we should never not!) build the kernel and the "world" (userland) from the same src update.

2. Read /usr/src/UPDATING

```
less /usr/src/UPDATING

Honestly, it is worth it. The times I did not bother, of course the
thing I cared about was right there in UPDATING.
```

3. The run through (from my OSBuilder code)

```
echo `$dd` start makeworld >> $log
echo NO_PROFILE=true >> /etc/make.conf
cd /usr/src
make -j6 buildworld

echo `$dd` start kernelbuild >> $log
cd /usr/src
make buildkernel KERNCONF=GENERIC
make installkernel KERNCONF=GENERIC


echo `$dd` start installworld >> $log
cd /usr/src
echo make installworld
```

```
make installworld
echo now run mergemaster
 #run so that it automatically installs new config files, and puts any diffs into /var/tmp/tempr
mergemaster -ia
```

4. do the above, manually ...

http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/makeworld.html

6.4.18 misc stuff I will find another place for

Using sudo and not root

I am a very bad boy and have a tendancy to just use root to get things done quick. This very nicely phrased page on why you should never login as root tells me off.

in short

1. sudoers is there for audit purposes and audit is half of security.
2. use sudo sh -c "echo foo >> /etc/bar"

this gets around the problem that sudo echo foo >> /etc/bar is two commands

passwords

/etc/passwd is a bit long in the tooth. /etc/master.passwd is where the action is now kept. if you look at that and need to, for example, change the login shell you need to run pwd_mkdb /etc/master.passwd afterwords

Setting Bash prompt

You need to set the environment variable PS1. The simplest approach is the best here I feel.:

```
PS1="\[\033[0;36m\][\$(date +%H:%M)][\u@\h:\w]$\[\033[0m\] ";export PS1
```

We set the PS1 variable. If we had used

```
PS1="$ "
```

Then our prompts would be a boring but functional

```
$ ls -l
```

But instead we are being hopefully a little cleaner. OK

```
\[\033[0;36m\]
^^^^^^^     ^^
always needed to *wrap* any thing we want to pass to terminal, but not print.  In this case, the comm
```

biblio http://www.linuxdoc.org/HOWTO/Bash-Prompt-HOWTO/x329.html

Colours under Bash

Oh yes, please. Just type ls -G (or like above, alias it.) This is great, right up to the point that the cool green on black screen you have tries to show you all your directories in a dark blue. I know I am getting old but taht is unreadable.

So ...

in ~/.bash_profile:

```
CLICOLOR=1; export CLICOLOR
LSCOLORS='gxfxcxdxbxegedabagacad'; export LSCOLORS
```

CLICOLOR is the equivalent of ls -G but always set. LSCOLORS is a string of 11 pairs, so gx is foreground-magenta, background-default, and the first pair is how to display a directory in ls (hence LSCOLORS). See the man page for details, but suffice to say gx gives me readble directory names now.

Note that in Linux, the environment variable is LS_COLORS, and its 'ls –colors'. However this is BSD world (mac included!).

Escape sequences etc etc

Here is a PS1 setting:

```
export PS1="\[\033[0;36m\][\u@\h:\W]$\[\033[0m\] "
```

ESC is ANSI 27 033 is the octal representation of ESC 0x1B is HEX e is whatever

We want this *not* to be interpreted by shell so

```
echo "\033[H\033[2J"
```

will not work. But:

```
printf "\033[H\033[2J"
```

will (its clear screen btw).:

```
tput cl
```

```
tput cl | less
```

tput is the command to execute escape sequences. DOnt worry.

So we can look up escape sequences in tables (wikiepdia) and build our own

```
"\[\033[0;36m\][\u@\h:\W]$\[\033[0m\] "
```

```
leftsquarebracket, Escape code (^[), blue colour, username@host:pwd$ reset to defaults/
```

```
gives:
```

```
[pbrian@hadrian:pbrian]$
```

biblio http://www.bigsoft.co.uk/blog/index.php/2008/04/11/configuring-ls_colors
http://unix.stackexchange.com/questions/2897/clicolor-and-ls-colors-in-bash

Temperature

$ sysctl hw.acpi.thermal.tz0.temperature hw.acpi.thermal.tz0.temperature: 54.0C

scp

Very useful in all respects but wildcards are a bugger

$ scp *.txt myremotesvr:/tmp/misc

will copy all txt file in local dir to remote server. Thats because the shell *expands* the commands you give it *before* it passes the commands to the program (scp). Normally thats good (in other words the shell took *.txt and actually sent the scp program

```
$ scp a.txt b.txt c.txt myremotesvr:/tmp/misc
```

however

$ scp myremotesvr:/tmp/misc/*.txt ./

fails. It is trying to send

```
$ scp myremotesvr:/tmp/misc/a.txt  b.txt c.txt ./
```

and it is invalid.

So to solve the problem, escape the wildcard, the shell will not expand it but will 'unescape' and pass it to the program as we want

$ scp myremotesvr:/tmp/misc/*.txt ./

http://books.google.co.uk/books?id=3XzIFG3w8-YC&pg=PA316&lpg=PA316&dq=scp+wildcard+match&source=bl&ots=oEjO3_aptl
YFh37YI4hLdFX8hWasA9N0NEOs&hl=en&ei=E5bnSe74CpG1-Qb02LjoBQ&sa=X&oi=book_result&ct=result&resnum=3http://boo
YC&pg=PA316&lpg=PA316&dq=scp+wildcard+match&source=bl&ots=oEjO3_aptE&sig=-
YFh37YI4hLdFX8hWasA9N0NEOs&hl=en&ei=E5bnSe74CpG1-Qb02LjoBQ&sa=X&oi=book_result&ct=result&resnum=3

mirroring with wget

wget -m -k -K -E http://url/of/web/site

emacs and sudo

Annoyingly, if I run

```
sudo emacs /etc/rc.conf
```

sudo runs emacs, but the permissions granted by sudo do not carry over to my emacs session, making it not possible to edit the locked down file.

tramp opens a subshell and then usually allows editing over ssh. it is useful for sudo too. ## C-c C-f /sudo::/my/path/file (require 'tramp)

emacs and annoying cut paste issues

http://www.jwz.org/doc/x-cut-and-paste.html

sort and sed and uniq

An interesting one from Dru Lavigne's little nuggets of wisdom

```
pbrian@delli7 Desktop$ pkg_info | sort | sed -e 's/-[0-9].*$//' | uniq -c | egrep -v '\ *1'
  2 autoconf
  3 automake
  2 font-adobe
  2 font-adobe-utopia
  2 font-bh
  2 font-bh-lucidatypewriter
```

```
  2 font-bitstream
  2 glib
  2 gtk
  3 xorg-fonts
pbrian@delli7 Desktop$ pkg_info | sort | sed -e 's/-[0-9].*$//' | uniq -c | grep -v '^[[:space:]]*1'
  2 autoconf
  3 automake
  2 font-adobe
  2 font-adobe-utopia
  2 font-bh
  2 font-bh-lucidatypewriter
  2 font-bitstream
  2 glib
  2 gtk
  3 xorg-fonts
```

Quick discussion - we get pkg_info sending us things like

```
zip-3.0             Create/update ZIP files compatible with pkzip
```

- We use sed to remove everything from the first -Numeral
- sed -e "s for substitute / this / for that /"
- Then uniq -c gives each word and a count of the occurances
- then grep -v to weed out anything that only occurs once.

Nice example showing a lot of the standard goodies on the command line.

Also interesting is the grep [[:space:]] - but I find '' more readable

Restart networking in FreeBSD

```
$ /etc/rc.d/netif restart && /etc/rc.d/routing restart
```

biblio href="http://www.cyberciti.biz/tips/freebsd-how-to-start-restart-stop-network-service.html"

- More emacs

```
  find /usr/local/share/emacs/ -iname '*.el'

Where the 'builtin' files are kept - but cannot see rst.el
Thats because rst is now 'builtin' to the emacs, so its supplied compiled, and need search for .elc
```

- Rest mode in emacs

  It has a really horrible highlight the text of each header. With a dark background its distracting and unreadable. SO we get rid of it.

  in .emacs file

  ```
  '(rst-level-face-base-light 0)
  ```

Red output on StdErr on terminal

http://superuser.com/questions/28869/immediately-tell-which-output-was-sent-to-stderr put this in .basdh_profile:

```
# Red STDERR
# rse <command string>
function rse()
{
   # We need to wrap each phrase of the command in quotes to preserve arguments that contain whitespa
   # Execute the command, swap STDOUT and STDERR, colour STDOUT, swap back
   ((eval $(for phrase in "$@"; do echo -n "'$phrase' "; done)) 3>&1 1>&2 2>&3 | sed -e "s/^\(.*\)$/$
}
```

then you can run

```
rse mycommand.sh
```

or

```
command.sh | rse
```

and stderr will print in red.

6.4.19 mysql

I am replacing this in favour of the postgresql cheatsheet Sad really, another thing lost to time.

6.4.20 Network Monitoring

historical note

This area has moved on considerably as well. netflow is still the heart of it and RRTD graphs exist everywhere still but graphite and the new PacketBeat are showing a new wave.

Introduction

Monitoring what is flowing over your network tubes is more important than ever. For reasons of capacity planning, as well as security and "getting a feel for the network" good tools matter.

Net-flow and cousins

http://forums.freebsd.org/showthread.php?t=248

net-mgmt/softflowd $ softflowd -i rl0 -n 192.168.1.20:8888

$ softflowctl statistics softflowd[11241]: Accumulated statistics: Number of active flows: 182 Packets processed: 2529 Fragments: 0 Ignored packets: 5 (5 non-IP, 0 too short) Flows expired: 0 (0 forced) Flows exported: 0 in 0 packets (0 failures) Packets received by libpcap: 2579 Packets dropped by libpcap: 0 Packets dropped by interface: 63

net-mgmt/flow-tools mkdir /var/log/netflow

echo /usr/local/bin/flow-capture -p /var/run/flow-capture.pid -n 287 -N 0 -w /var/log/netflow/ -S 5 0/0/8888

http://onlamp.com/bsd/2005/08/18/Big_Scary_Daemons.html

6.4.21 Nginx - load balancing

Almost out of nowehre in 2007/8 nginx came on the scene and started to beat out the competition of the Open Apache and the closed Zeus.

It is now pretty much the defacto web server going, easy to install, simple(r) to configure and fast, fast, fast.

Heres a look at how nginx does basic load balancing

```
upstream  yoursite  {
   server   yoursite1.yoursite.com;
   server   yoursite2.yoursite.com;
}

server {
   server_name www.yoursite.com;
   location / {
      proxy_pass  http://yoursite;
   }
}
```

This configuration will send 50% of the requests for www.yoursite.com to yoursite1.yoursite.com and the other 50% to yoursite2.yoursite.com. ip_hash

You can specify the ip_hash directive that guarantees the client request will always be transferred to the same server. If this server is considered inoperative, then the request of this client will be transferred to another server.:

```
upstream  yoursite  {
   ip_hash;
   server   yoursite1.yoursite.com;
   server   yoursite2.yoursite.com;
}
```

Down

If one of the servers must be removed for some time, you must mark that server as down.:

```
upstream  yoursite  {
   ip_hash;
   server   yoursite1.yoursite.com down;
   server   yoursite2.yoursite.com;
}
```

weight

If you add a weight tag onto the end of the server definition you can modify the percentages of the requests send to the servers. When there?s no weight set, the weight is equal to one.:

```
upstream  yoursite  {
   server   yoursite1.yoursite.com weight=4;
   server   yoursite2.yoursite.com;
}
```

This configuration will send 80% of the requests to yoursite1.yoursite.com and the other 20% to yoursite2.yoursite.com.

note: It?s not possible to combine ip_hash and weight directives. max_fails and fail_timeout

max_fails is a directive defining the number of unsuccessful attempts in the time period defined by fail_timeout before the server is considered inoperative. If not set, the number of attempts is one. A value of 0 turns off this check. If fail_timeout is not set the time is 10 seconds.

```
upstream  yoursite  {
   server   yoursite1.yoursite.com;
   server   yoursite2.yoursite.com max_fails=3  fail_timeout=30s;
}
```

In this configuration nginx will consider yoursite2.yoursite.com as inoperative if a request fails 3 times with a 30s timeout. backup

If the non-backup servers are all down or busy, the server(s) with the backup directive will be used.

> **upstream yoursite {** server yoursite1.yoursite.com max_fails=3; server yoursite2.yoursite.com max_fails=3; server yoursite3.yoursite.com backup;
>
> }

This configuration will send 50% of the requests for www.yoursite.com to yoursite1.yoursite.com and the other 50% to yoursite2.yoursite.com. If yoursite1.yoursite.com and yoursite2.yoursite.com both fails 3 times the requests will be send to yoursite3.yoursite.com.

6.4.22 pxe boot

PXE booting is the process of an ethernet chip searching on the local lan for a suitable DHCP provider from which it can then download and boot a tiny kernel, which is then used to install the normal kernel / OS.#

Its a nice idea for the new generation of small single board computers like the RaspberryPi, soekris or beagleboards.

http://etherboot.org/wiki/pxechaining http://etherboot.org/wiki/httpboot

6.4.23 FreeBSD - RAMDISK

```
#!/bin/sh

case "$1" in
start)
        /sbin/mdmfs -s 256M md10 /mnt/ramdisk
        echo "256MB ramdisk created on /dev/md10 and mounted on /mnt/ramdisk"
        exit 0
        ;;
stop)
        /sbin/umount /mnt/ramdisk
        /sbin/mdconfig -d -u 10
        echo "ramdisk unmounted from /mnt/ramdisk and deleted from /dev/md10"
        ;;
*)
        echo "Usage: `basename $0` {start|stop}" >&2
        exit 64
        ;;
esac
```

biblio:

http://www.freebsdwiki.net/index.php/RAMdisks,_creating_under_FreeBSD_5.x

6.4.24 Basics of routing on your own LAN

historical note

Well this might be a little out of date :-)

I wanted to add my laptop to marshallfamily SSID

```
ifconfig_ath0="ssid marshallfamily wepmode on weptxkey 1 wepkey 1:xxxxxxx"


ifconfig ath0 ssid marshallfamily wepmode on weptxkey 1 wepkey 1:xxxxxxx
```

but cannot seem to connect it and bind to ip address

so force it

```
ifconfig ath0 192.168.1.3 netmask 255.255.255.0
```

then force a default route to the router that this connects too

```
net add default 192.168.1.1
```

6.4.25 Samba File and Print Server

historical note

This is an interesting sliver of history - but I am amazed there are still people doing this. I thought DropBox ruled the world!

http://us1.samba.org/samba/docs/using_samba/toc.html http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/network-samba.html http://www.samba.org/samba/docs/man/Samba-HOWTO-Collection/

Example:

```
[global]
    workgroup = PIRC
    server string = NAGIOS SMB Server
    security = share
    log file = /var/log/samba/log.%m
    max log size = 50
    dns proxy = no
    guest account = guest
    unix extensions = off

[install]
    comment  = Unattended
    writable = yes
    locking  = no
    path     = /usr/local/PIRC/w32install/install
    guest ok = yes
```

Troubleshooting

1. The above file has

guest account = guest

But that account (guest) did not exist on my Box so I got

tail /var/log/samba/log.smbd

[2008/01/13 22:56:38, 0] smbd/server.c:main(1059) ERROR: failed to setup guest info.

Interestingly I want to view the source itself so /usr/ports/distfiles holds all the tarballs, expand and have a look

SWAT

uncomment inetd.conf line to run swat

#!/bin/sh

#smbclient -U paul.brian -L \ADC

mounting samba mount_smbfs //nobody@samba/FILES /mnt/samba/FILES mount_smbfs //nobody@samba/PDOXDATA0 /mnt/samba/PDOXDATA0 mount_smbfs //nobody@samba/PDOXDATA1 /mnt/samba/PDOXDATA1 mount_smbfs //nobody@samba/IT /mnt/samba/IT

mount_smbfs //paul.brian@ADC/Managers /mnt/ADC/Managers

6.4.26 Useful command line utilities

learn the command line

:: # echo day | sed s/day/night/ night # echo daylight | sed s/day/night/ nightlight

Oh yeah baby. Power.

Let me show you how Linus [2] did a simple check of his bash history

::

history

grep git
sed 's/[0-9]*(git[ ]*[a-z-]*).*/1/'
sort
uniq -c
sort -n

so lets go through that and see what is worth learning

```
#!/bin/ksh

mystring="01.Driver Report|userid@company.com"
echo "$mystring"|while IFS="|" read f1 f2
do
name="$f1"
email="$f2"
done
echo "$name"
echo "$email"
```

[2] http://www.linux-mag.com/cache/7314/1.html

bibliography

http://www.grymoire.com/Unix/Sed.html#uh-0

6.4.27 Time

Time waits for no man

Or it flies like an arrow. Something like that. But still keeping accurate time is as important now as it was when Harrison built his first clock. In a networked world, having one machine with accurate time is not the worst problem - all the machines we use need to be synchronised to the same time, else trying to compare logs on different machines is quite simply a nightmare.

The solution to both these problems is called Network Time Protocol or NTP. NTP is a way for any server to get the current time from an atomic clock, even if it is thousands of miles away. Atomic clocks keep, what is simply the most accurate time humankind can keep. These clocks then are used to keep *stratum 1* servers up to date. These stratum 1 servers are the starting point for NTP, and are primarily run by academic and governmental institutions. They are the source point for, rather boringly, *stratum 2* servers. These stratum 2 servers are run, often by volunteers to provide you and I with a machine that knows precisely what the time is, and more importantly, are willing to tell us.

Whilst the servers themselves are run in wildly different locations, and by different organisations, a sensible DNS round-robin is in place.

```
Name:   pool.ntp.org
Address: 64.202.112.75 (ntp.your.org.)

Name:   pool.ntp.org
Address: 38.99.80.156 (clock.fihn.net.)
```

Here two servers run by totally different organisations are pulled up when I ask for pool.ntp.org. So all I need to remember is one dns name and my ntp requests will alwasy find a suitable match. This is good news for me, but also very good news for Poul Henning Kemp [3]

Anyway, NTP works in a clever and comlicated way, but the Simple IT manager can get by with a good enough explanation, and a reference for more info. The good enough explanation is that the NTP client asks for a time check and waits for the answer to come back several times. With enough times and a clever algorithm, the latency in the network can be estimated and that latency can be added to the time in the received message to give an accurate time *right now*.

```
NTPv4 can usually maintain time to within 10 milliseconds (1/100 s) over the
public Internet, and can achieve accuracies of 200 microseconds (1/5000 s) or
better in local area networks under ideal conditions.

http://en.wikipedia.org/wiki/Network_Time_Protocol
```

How to get the time right on your server

1. Set your current timezone
2. set the time correctly from an atomic clock
3. keep on adjusting it

[3] Poul-Henning Kemp is a respected FreeBSD developer who ran a Stratum 1 Server. His server was referenced by a commercial domestic router sold in its millions, and by IP address. He was getting 3 million packets a day extra. Full story is at http://www.lightbluetouchpaper.org/2006/04/07/when-firmware-attacks-ddos-by-d-link/

1. Set your current timezone

In /usr/share/zoneinfo are a series of binary files that represent to the server how to calculate the timezone - so /usr/share/zoneinfo/Europe/London tells us how to adjust for the GMT zone. We want to copy the one that is right for our location in to /etc/localtime

diff /etc/localtime /usr/share/zoneinfo/Europe/London

(there may not be a /etc/localtime file. In fact if this is the first install, there wont be)

So let us have some fun:

```
manhattan# cp /usr/share/zoneinfo/Asia/Tokyo  /etc/localtime
manhattan# adjkerntz -a
manhattan# date
Sat Aug 16 20:16:02 JST 2008
manhattan#
manhattan#
manhattan# cp /usr/share/zoneinfo/Europe/London /etc/localtime
manhattan# adjkerntz -a
manhattan# date
Sat Aug 16 12:16:23 BST 2008
```

2. clock synchronisation

Ok, we can control what timezone the machine thinks it is in, but how do we get the time correct? The configuration file for ntp daemon is /etc/ntpd.conf

put this minimal file into /etc/ntpd.conf:

```
driftfile /var/db/ntp.drift
server pool.ntp.org
restrict default ignore  #stops this machine being used by anyone else for ntp

##also
touch /var/db/ntp.drift
```

(NB there is a diff between the location in freebsd handook and the ntp ssite for conf file ntpd.conf vs ntp.conf. ntp.conf is right - follow the handbook)

This tells the server to use a drift file, which is how ntpd calculates the average latency for its requests, and which servers it should contact - if you wanted a really minimalist conf file, the last line will suffice.

Enable the server at boot time:

```
ntpd_enable="YES" to /etc/rc.conf
```

TO force the server to change to a new time, especially if the time is a long way out, then we can use

ntpdate pool.ntp.org

ntpdate forces a time update no matter how far out the server is - this is useful, but you should rely on ntpd to correct the time - if it finds itself correcting large leaps it will scream to the logs. This is good - there are only two reasons for regular large corrections, either your motherboards clock is dying or you have invented time travel.

Sample rc.conf file with a full time set on boot:

```
ntpdate_enable="YES"
ntpdate_flags="pool.ntp.org"
```

```
#enable ntpd after ntpdate - ntpdate does nothing if  ntpd is running.
ntpd_enable="YES"
```

Trouble shooting

I often get tripped up by this when running ntpdate manually:

```
"step-systime: Operation not permitted"
```

It is a permissions problem, but crops up most often in Jails and related virtualised servers. The access to the hardware layer is mediated and so even as root on a Jail, you cannot set time - time is fixed on the host OS.

Notes

Bibliography

http://www.freebsd.org/doc/en/books/handbook/network-ntp.html http://support.ntp.org/bin/view/Servers/NTPPoolServers

6.4.28 Adding a new disk

Adding a new disk is relatively simple to a BSD system, However the most common way of adding a disk is through the USB subsystem. So I am trying to cover the two approaches here. Superficially the USB disk and the 'normal' disk just need to be 'recognised' - that is have a device file put in /dev/ with an appropraite driver.

Lets get to the point where a disk is recognised by dmesg.

Getting to dmesg

We can add a disk inside the PC as 'normal' - that is unscrew the covers and connect up a PATA (IDE) or SATA disk. In pretty much every case here, you will see that disk show up in dmesg

```
ad4: 57231MB <Hitachi HTS541660J9SA00 SBBOC70P> at ata2-master SATA150
```

So as expected I have a 60GB HDD on sata

However if I then plug in my USB enclosed external hard drive I see on Dmesg

```
umass0: <Maxtor OneTouch, class 0/0, rev 2.00/1.22, addr 2> on uhub2
da0 at umass-sim0 bus 0 target 0 lun 0
da0: <Maxtor OneTouch 0122> Fixed Direct Access SCSI-4 device
da0: 40.000MB/s transfers
da0: 114473MB (234441648 512 byte sectors: 255H 63S/T 14593C)
```

now this shows two things. FIrstly the umass driver. this is the USB-mass storage driver and it has found something on the USB Bus Then the driver is able to recognise the actual model, and so I have now got a device driver lined up and a 'file' sitting in /dev/da0

I can now use this as normal and proceed to the next step.

If umass discovers the disk but there is no device listed, frankly its tough. The drivers are not there and unless its vital, go buy another supported disk. See umass(8) for supported disks. There are loads, and given you can buy a 1TB for less than a hundred quid, its not worth worrying.

However if you really need that disk, try a firewire enclosure. This has less driver issues than USB.

Preparing and mounting the disk

0. Zero the disk
1. fdisk - Being nice to the PC BIOS
2. bsdlabel - prepping for BSD
3. newfs - writing the filesystems
4. mount

bibliography: http://www.freebsd.org/doc/en/books/handbook/disks-adding.html

zero the disk

This is straightforward dd from /dev/zero to your disk. I find that as a rough rule of thumb you can zero an entire 40GB disk in 20 minutes. [4]

```
#this overwrites the first 1024 bytes, effectviely killing the boot partition.
#leave off count to do the whole disk, make count 1000 to do first 1MB etc etc

dd if=/dev/zero of=/dev/da0 bs=1024 count=1
```

fdisk

You do not have to do this, except for the boot disk iSo for a USB disk that is solely for backup, fine, used 'dedicated' mode

PC BIOS usually 'controls' a HDD. THat is it mediates the access from the OS to the BIOS to the HDD. THats how the PC was designed, and frankly BSD ignores it and can control the disk directly. However if BSD is controlling the disk directly, it cannot be used by some other OS that expects the BIOS to do the work. So BSD has 'dedicated' mode that means you use that disk only fafter BSD has booted - DOS wont like it, and BIOS wont boot from it.

a dedicated disk will look like /dev/da0e

We shall ignore that for now - the handbook recommends playing nice - using 'slices' A slice is a BSD term for one of the four BIOS partitions. so a BSD disk with 4 partitions will have device files called

```
/dev/ad4s1 /dev/ad4s2 /dev/ad4s3  /dev/ad4s4
```

Look at what is there

```
# fdisk -s /dev/ad4
/dev/ad4: 116280 cyl 16 hd 63 sec
Part        Start        Size Type Flags
   1:           63   117210177 0xa5 0x80


# fdisk /dev/ad4
******* Working on device /dev/ad4 *******
parameters extracted from in-core disklabel are:
cylinders=116280 heads=16 sectors/track=63 (1008 blks/cyl)
```

[4] Will this be unreadable? Well yes, with caveats. Overwriting the disk with zeros once makes it unrecoverable except by scanning the surface of the disk with specialised equipment which can detect the tiny magnetic differences between a zero that was overwritten with zero and a one that was overwritten with zero. Unless you are a suspected terrorist or have stolen a billion dollars, overwriting with zeros once is fine. If you are a terrorist, firstly do not put your secret plans on a computer in the first place, secondly, use a masonary drill to destroy the platters, a few holes sufficient heat and presto, no recovery. Frankly this is all a bit academic. If you know the cops will get you in sufficient time for you to zero a disk or otherwise destroy it, the cops need to get better funding.

```
Figures below won't work with BIOS for partitions not in cyl 1
parameters to be used for BIOS calculations are:
cylinders=116280 heads=16 sectors/track=63 (1008 blks/cyl)

Media sector size is 512
Warning: BIOS sector numbering starts with sector 1
Information from DOS bootblock is:
The data for partition 1 is:
sysid 165 (0xa5),(FreeBSD/NetBSD/386BSD)
    start 63, size 117210177 (57231 Meg), flag 80 (active)
        beg: cyl 0/ head 1/ sector 1;
        end: cyl 1023/ head 254/ sector 63
The data for partition 2 is:
<UNUSED>
The data for partition 3 is:
<UNUSED>
The data for partition 4 is:
<UNUSED>
```

this tells me that there are 4 slices or partitions, just as the BIOS expects, but that only one is initialised in a form readable to BIOS (ie the first partition, which the BIOS needs to read to boot the disk). The rest is handled by BSD itself, using infomration found in bsdlabel

So finally how to initialise my external USB hard disk to have slices the nice way

```
fdisk -BI /dev/da0
#this will
# -I Initialize sector 0 slice table for one FreeBSD slice covering the entire disk.
# -B Reinitialize the boot code contained in sector 0 of the disk

GEOM not found is considered a benign warning by the man file.
```

[5]

You will know have one slice on the disk, /dev/da0s1 This is generally what happens when you use sysinstall and just choose 'use whole disk'

bsdlabel

Here is where we create the traditional BSD partitions (not restrictive 4 BIOS paritions which are called slices now). The bsdlabel stores partition size and location info, as well as other stuff. it is read by BSD and used to control the HDD, bypassing the BIOS.

How do BSD partitions work? Basically there are eight partitions (a-h). (see http://www.freebsd.org/doc/en/books/handbook/install-steps.html)

The a partition is solely for system disks (ie the disk you boot from). It is only used for the root (/) partition. The b partition is used for swap space. Any number of disks can have swap space, and the rule of thumb is to have twice RAM as swap. The c partition is a sort of whole disk placeholder

the e partition is usually the point to put /var in system disks and everything for other disks the f partition is usually the everything point (/usr) for system disks

[5] In order for the BIOS to boot the kernel, certain conventions must be adhered to. Sector 0 of the disk must contain boot code, a slice table, and a magic number. BIOS slices can be used to break the disk up into several pieces. The BIOS brings in sector 0 and verifies the magic num- ber. The sector 0 boot code then searches the slice table to determine which slice is marked ''active''. This boot code then brings in the bootstrap from the active slice and, if marked bootable, runs it. Under DOS, you can have one or more slices with one active. The DOS fdisk utility can be used to divide space on the disk into slices and set one active.

-B will turn a slice into a bootable slice - by copying the boot code from /boot/boot into the bootsector of the slice being worked on. It useful for system disks but this is not a system disk

```
#bsdlabel -w /dev/da0s1
```

OK this has built a standard label onto that slice. But it does not have our BSD partitions in it. we want swap, we want /usr etc. We get this by using

```
bsdlabel -e /dev/da0s1
```

this will use vi (EDITOR) to edit the label. Now a couple of points. firstly, use -n

```
bsdlabel -e -n /dev/da0s1
```

This will not write to the label, but do all calculations and output some hints for you. Second, the bsdlabel is good for calculating human-sized amounts

so when I freshly create the bsdlabel and use -e I get to see a label like this

```
# /dev/da0s1:
8 partitions:
#        size   offset    fstype   [fsize bsize bps/cpg]
  a: 234436466       16    unused        0     0
  c: 234436482        0    unused        0     0         # "raw" part, don't edit
```

The whole disk is taken up with the a partition which is OKish but not to convention.

I simply edit it using vi as follows

```
# /dev/da0s1:
8 partitions:
#        size   offset    fstype   [fsize bsize bps/cpg]
  c: 234436482        0    unused        0     0         # "raw" part, don't edit
  e: *                *    4.2BSD        0     0
```

I have told it to create a e: partition of 'the whole remainder of the disk' and it does the hard part for me.

Alternatives include

```
# /dev/da0s1:
8 partitions:
#        size   offset    fstype   [fsize bsize bps/cpg]
  a:       30G      16    4.2BSD        0     0
  b:        6G       *    swap
  c: 251658225       0    unused        0     0         # "raw" part, don't edit
  e:         *       *    4.2BSD        0     0
(cf http://keramida.wordpress.com/2008/09/14/moving-a-freebsd-installation/)
```

Building the file system

Now in the first example of editing the bsdlabel above, I have only created one partition so I only need one file system to be written on the partition. I just need to run

```
newfs -L BACKUP /dev/da0s1e
```

however with the second where it is looking a bit like a system disk

```
newfs -L ROOT /dev/da0s1a
#swap needs no filesystem
newfs -L USR /dev/da0s1e
```

Mount

```
# mkdir /mnt/disk1
# mount /dev/da0s1e /mnt/disk1
```

Summary

```
# dd if=/dev/zero of=/dev/da1 bs=1k count=1
# fdisk -BI da1 #Initialize your new disk
# bsdlabel -B -w da1s1 auto #Label it.
# bsdlabel -e da1s1 # Edit the bsdlabel just created and add any partitions.
# mkdir -p /1
# newfs /dev/da1s1e # Repeat this for every partition you created.
# mount /dev/da1s1e /1 # Mount the partition(s)
# vi /etc/fstab # Add the appropriate entry/entries to your /etc/fstab.
```

Actual results

```
[root@paullaptop ~]# dd if=/dev/zero of=/dev/da0 bs=1024 count=10
10+0 records in
10+0 records out
10240 bytes transferred in 0.208999 secs (48995 bytes/sec)

[root@paullaptop ~]# fdisk -BI /dev/da0
******* Working on device /dev/da0 *******
fdisk: invalid fdisk partition table found
fdisk: Geom not found: "da0"
```

Just format a USB Disk for use between BSD and Windows

- http://lists.freebsd.org/pipermail/freebsd-questions/2010-May/215819.html
- http://en.wikipedia.org/wiki/BSD_disklabel

```
In BSD-derived computer operating systems (including NetBSD,
OpenBSD, FreeBSD and DragonFly BSD) and in related operating
systems such as SunOS, a disklabel is a record stored on a data
storage device such as a hard disk that contains information about
the location of the partitions on the disk. Disklabels were
introduced in the 4.3BSD-Tahoe release.[1] Disklabels are usually
edited using the disklabel utility. In later versions of FreeBSD
this was renamed as bsdlabel.
```

6.4.29 Useful Tools

Things I use everyday, or need to

http://www.pythonregex.com/ wow - really ideal. I never get regexes right the first (ten) times.

http://www.balsamiq.com/blog/2009/10/30/tools/ an interesting list similar to this one, but longer.

6.4.30 VirtualBox on FreeBSD

Historical note

This never made it into the FreeBSD manual because I was a lazy a** and never did the re-formating they wanted, but I am still quite proud of it.

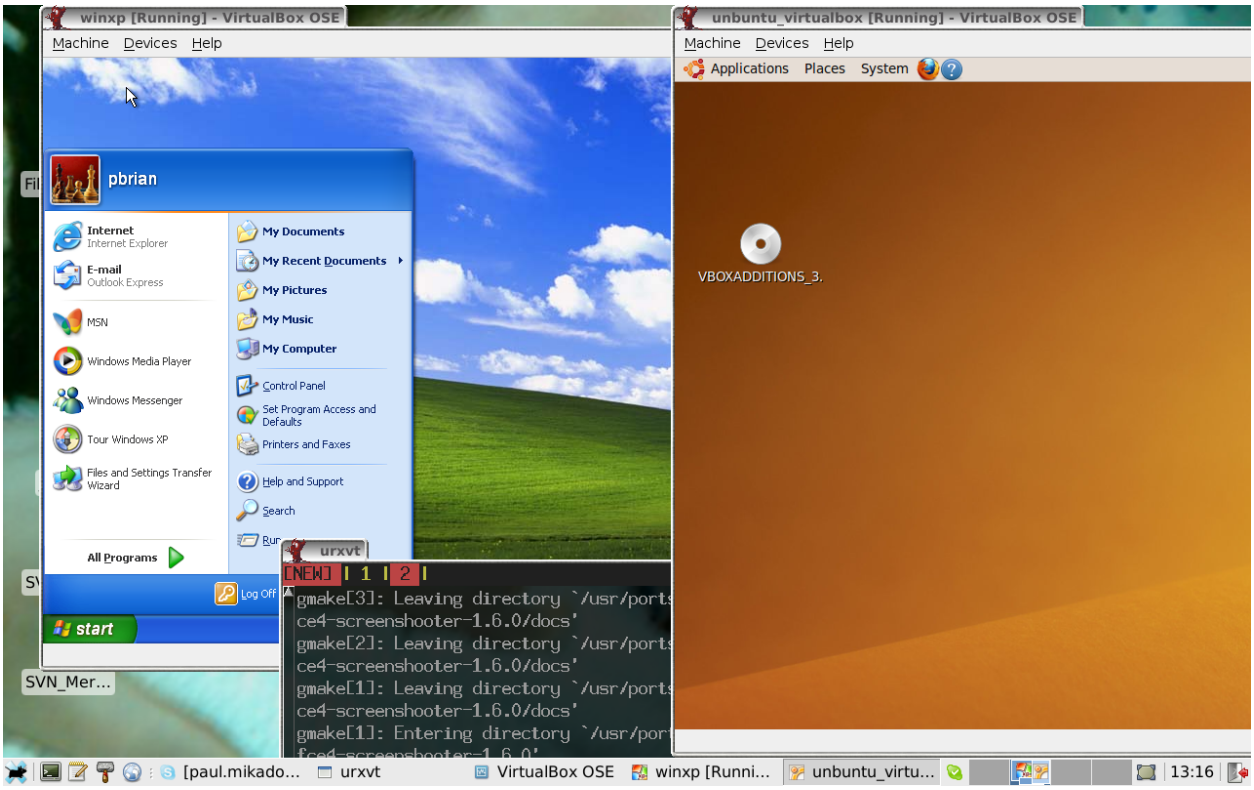


Figure 6.1: Windows and Ubunutu running inside a FreeBSD Host.

Introduction

Virtualisation, running more than one instance of (potentially) more than one Operating System on a single physical computer, is growing in popularity and power.

FreeBSD offers two different types of virtualisation, which can be characterised as single-kernel virtualisation and multiple-kernel virtualisation.

The first, implemented natively to FreeBSD, is Jails, allowing one FreeBSD kernel to run many instances of a compatible FreeBSD system. This approach is simple, and fast and is probably the best choice if you wish to run several non-GUI instances (ie a mail machine, a web server and a database). The author has successfully used this approach for some years commercially and at home. However if the author wished to run Windows or any other operating system on the Jail Host it would not work - every instance in a Jail uses the same kernel, so not even different versions of FreeBSD will run on a jail. In order to run different operating systems one turns to multi-kernel virtualisation.

This second approach, often called "true" virtualisation, (but usually only by the vendors of this class of software), is where a Virtual Machine Manager (VMM) is created that acts as a "interpretation layer" between the Host OS (in our case FreeBSD) and the Guest OS (for example a Windows XP instance).

VirtualBox, currently owned by Oracle, is a dual-licensed VMM, with a commercial version and and OSE (Open Source Edition). This Open Source Edition has been ported natively to FreeBSD, and this chapter shall demonstrate the steps necessary to successful implementation.

It is worth noting that VirtualBox OSE has been intentionally crippled - it will not pass through RDP requests (you cannot "remote desktop" into the box) and it will not allow USB devices plugged into the Host OS to be accessible to the Guest OS. The commerical version does allow these functions.

Hardware support

Until recently virtualised machines were noticeably slow - because the "interpretation layer" was implemented solely in software. However as virtualisation has grown in popularity, the major chip vendors have worked to develop hardware extensions to improve speed.

There have been two major innovations in hardware-based virtualisation. VMX extensions and nested paging. The first has been around for a couple of years, and in essence performs the marking of each process in the host as belonging to the correct Guest OS.

The second is most recent, found in the Nehalem-range on Intel CPUs and their AMD competitor (?)- without this, a VM must pretend it holds a paging cache - so any cache misses end up being processed twice, once in the "real" cache, once in the VM version.

Anecdotally, a CPU without VMX flags and Nehalem, can be around 3-5 times slower in virtualisation, and for a desktop machine this is very noticeable.

You can detect if your CPU supports the VMX extensions by looking at dmesg for CPU flags, notable VMX and SSE flags

```
CPU: Intel(R) Core(TM) i7 CPU       M 620  @ 2.67GHz (2660.02-MHz 686-class CPU) Origin = "GenuineInt

Features2=0x298e3ff<SSE3,PCLMULQDQ,DTES64,MON,DS_CPL,VMX,SMX,EST,TM2,SSSE3,
                                                      ^^^
                     CX16,xTPR,PDCM,SSE4.1,SSE4.2,POPCNT,AESNI>
                                   ^^^^^^^^^^^^^
```

VMX (almost certainly a requirement for decent Desktop vitualisation) and SSE4 extensions are shown above.

Prerequisities

You will need a properly configured X-Windows installation (see Handbook) and sufficient RAM and Hard Disk space. For each desktop-style Guest OS you would expect to need between 256-384MB of RAM and 6+GB of space, and the more especially of RAM the better the experience.

Installing VirtualBox

NB. These instructions deal with port at or after OSE 3.1.2/

To build VirtualBox use the port found at

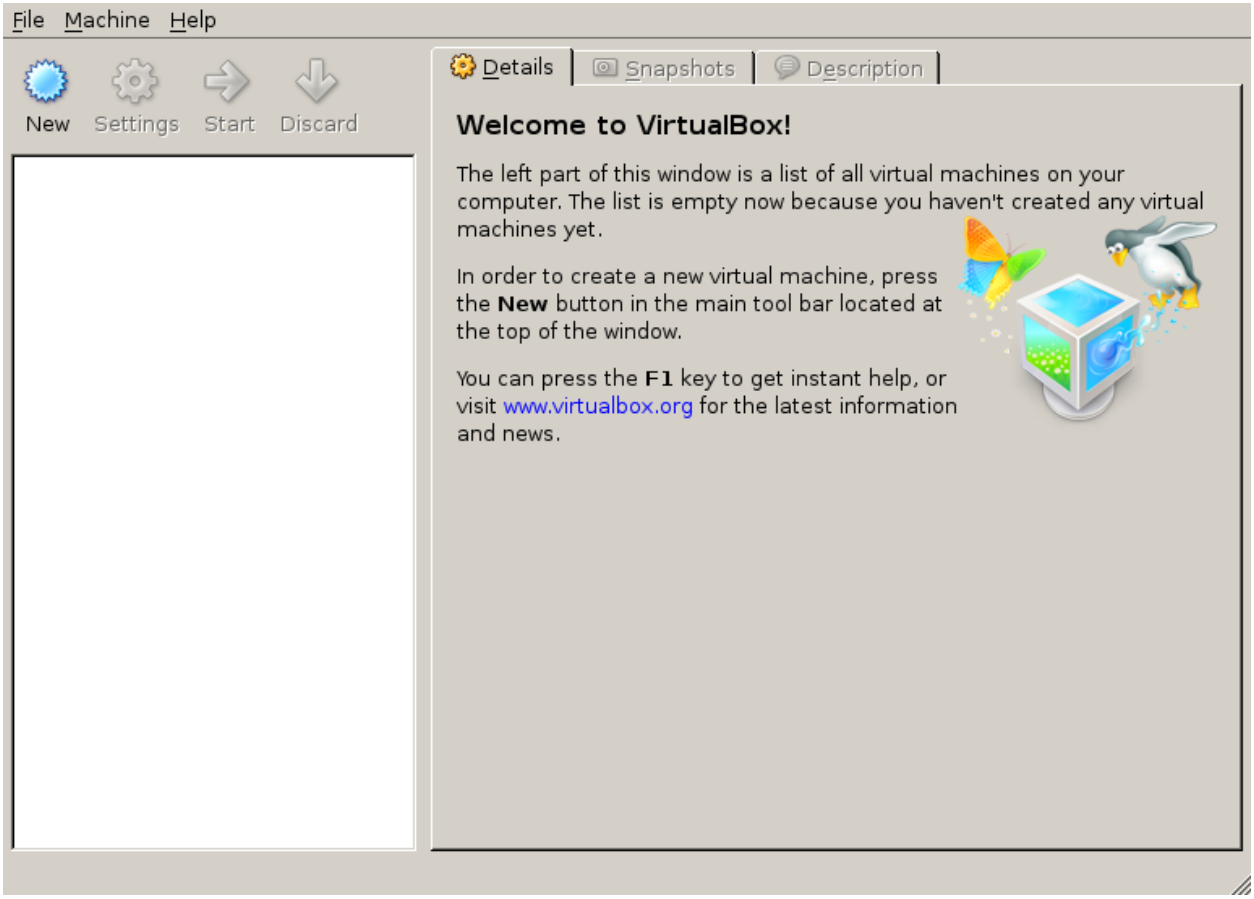
```
cd /usr/ports/emulators/virtualbox; make install clean
```

You are *strongly* recommended to select the GuestAdditions option when configuring the port. This will install the port virtualbox-ose-additions, an .iso of dirvers and other useful features. If you find Windows is unable to exceed 800x600 then Guest additions is not installed.

This port has been rewritten recently to make VirtualBox install simpler and easier. Once you have compiled it, you will need to adjust /boot/loader.conf as below

```
echo vboxdrv_load="YES" >> /boot/loader.conf
```

If you enter “VirtualBox” from a terminal you should see a start up screen like this:



Preparing for first Guest OS

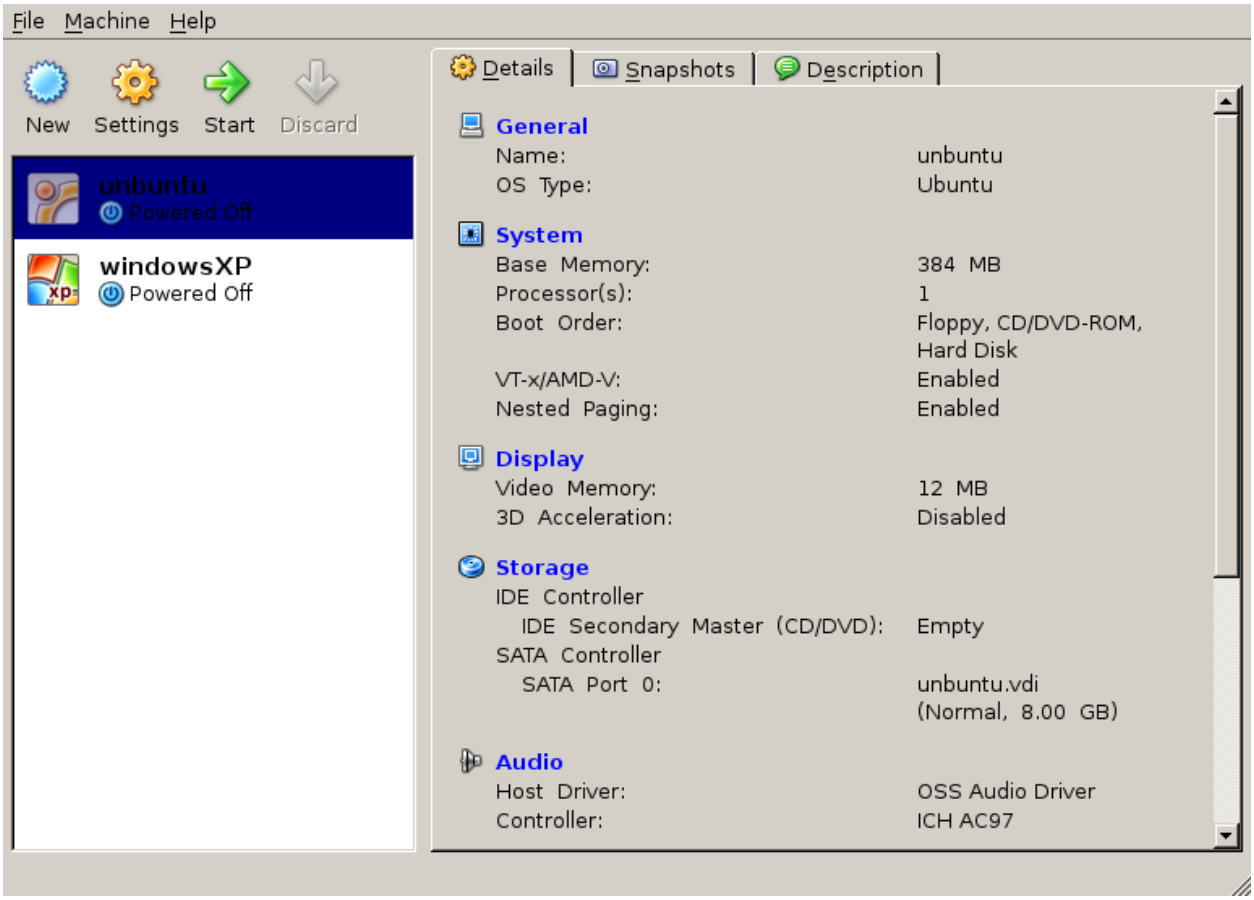
The principle behind the setup is simple. Allocate a certain amount of your harddisk to become a single file. That single file will pretend to be a real Hard Disk to the VM, which will also have access to all your peripherals, network card and a pre-determined slice of RAM.

The VMM has a simple to use wizard and it is recommended to use this for the first VM. You will need to experiment a little on the optimum Hard disk and RAM allocation for your machine, but 384MB and 8GB has been found to be a workable minimum for Windows XP and ubuntu installs.

After the wizard has run you should now have a “virtual machine” ready to start - 8GB of Harddrive (actually a single file on your real harddrive) and 384MB of RAM ready to go. At this point we can boot up the instance, and you should see a complaint that there is no bootable image. We shall now install a OS onto the vitual machine.

Setting up first Guest OS

We want the VM to boot up, and then look for a bootable image (ie the VM boots and is given a install CD). This can be done in two ways - by allowing the VM access to our own CD/DVD drive, or by mounting an ISO for it. THe second is a lot more convenient - especially for ubuntu.


File Machine Help
New
Settings
Start
Discard
unbuntu
Powered Off
windowsXP
Powered Off
Details
Snapshots
Description
General
Name: unbuntu
OS Type: Ubuntu
System
Base Memory: 384 MB
Processor(s): 1
Boot Order: Floppy, CD/DVD-ROM, Hard Disk
VT-x/AMD-V: Enabled
Nested Paging: Enabled
Display
Video Memory: 12 MB
3D Acceleration: Disabled
Storage
IDE Controller
IDE Secondary Master (CD/DVD): Empty
SATA Controller
SATA Port 0: unbuntu.vdi (Normal, 8.00 GB)
Audio
Host Driver: OSS Audio Driver
Controller: ICH AC97

Obtain ISO You can just download the latest Ubuntu ISO, or you can extract your *licensed* copy of XP by placing the CD in your CD drive and

```
# dd if=/dev/acd0 of=/home/pbrian/downloads/xp.iso bs=2048
```

(NB the block size setting is *very* important - without it you will not copy anything from a CD drive)

Now visit the CD tab in the VMM GUI. tick the 'Mount CD Drive' and then tick 'Mount from ISO'. Simply find the iso image on your HDD, and now the Virtual Machine you selected will be able to "see" the CD as if it was in a normal CD drive.

Install From ISO Start up the VM instance, now you will be able to install the chosen OS as you would expect, perhaps a little faster than you are used to.

After installing your chosen OS, you will be able to "start" the VM from the VMM control panel - do so now, and you will see the usual boot up screens and then a working instance of another Operating System in your FreeBSD machine.

Networking

VirtualBox offers a NAT-based networking address for your VM out-of-the-box. This enables you to initiate connections from the VM, but to have connections initiated *to* the VM, you will need to set up "Ethernet Bridging".

Bridged Networking Netgraph is one such popular way of setting up Bridged Networking, creating a driver that is able to bypass the usual networking stack.

With this the VM can take a packet off the real NIC, read it and then put its own reply back onto the real NIC at the same layer - it looks to the Host OS as if right next door on the Ethernet network is another NIC reading the packets it reads and putting more on the network.

The Virtual machine on the other hand just sets up its stack as normal, thinking it has access to a genuine real NIC.

Its a bit like this:

```
Host  VM
 \    /
  \  /
   --  Netgraph
   |
   NIC
```

You will need to re-compile your kernel (see Handbook for instructions). The following can be used as a kernel configuration file:

```
include GENERIC
ident NGRAPH

options NETGRAPH
```

You will then need to switch the current virtual NIC over to Bridged mode in the VMM GUI and select the appropriate driver - PCNet III works well the Netgraph driver.

bibliography

- https://help.ubuntu.com/community/Installation/QemuEmulator
- https://help.ubuntu.com/community/WindowsXPUnderQemuHowTo
- http://wiki.freebsd.org/qemu
- http://dryice.name/blog/freebsd/using-freebsd-as-a-network-bridge-and-use-dummynet-to-shape-the-tra
- http://www.freebsd.org/doc/en/books/handbook/network-bridging.html

Bibliography

http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/virtualization-host.html
http://wiki.freebsd.org/VirtualBox

6.4.31 Virtualisation

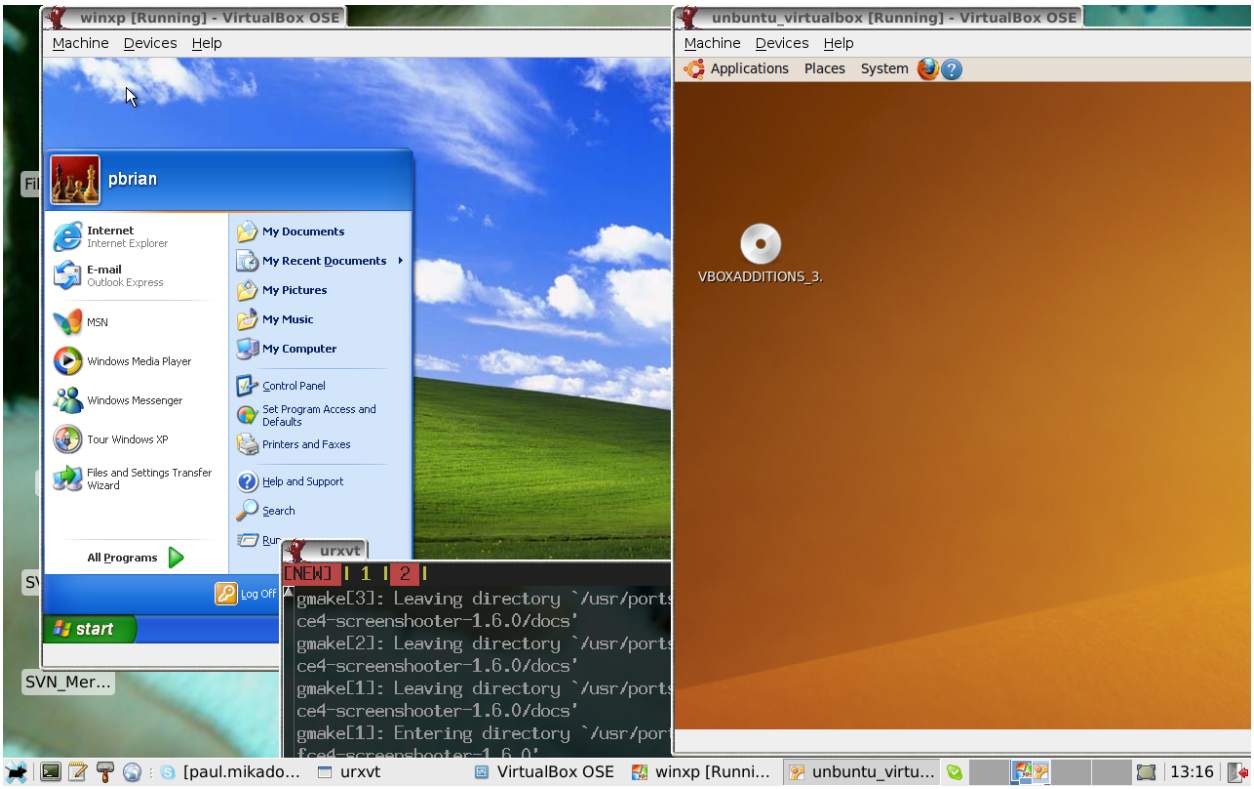


Figure 6.2: Windows and Ubunutu running inside a FreeBSD Host. Cool Huh?

There are two major types of virtualisation possible these days Jails about which I have written ad-nauseam and think make excellant option for a FreeBSD system. Jails however can only virtualise the host OS - that is I can have a dozen FreeBSD servers running on one host. But I cannot put Windows on.

If I want to run Windows, then it is over to “true” virtualisation. Its only called true by vendors of this class of software, but frankly the name has stuck a little.

So, how does one do this, and more importantly, why?

Why virtualise?

Well, the obvious reasons do count - improved power usage, space usage, reduction in sheer complexity. Its usually cheaper and easier to buy one big server and put a dozen servers on it, than to actually buy a dozen servers. [6]

I use Jails all the time - in fact it is a driving reason and enabler for giving each developer a complete logical clone of the production system to develop *on*. If every developer has a set of machines they can put the complete system onto, several things happen

- deployments are scripted. If the developer has to deploy to a server each time they want to test, magically deploy scripts are written *and kept up-to-date*.
- integration problems become more visible early. Deploying your local working copy to your local server set is fine, but if someone checks in bad code, it gets noticed, quick.

But that is fine for Jails. Why this mucking around with Windows?

Quite simple - for testing.

Want to do some web-based UI testing and verification. Then Selenium from ThoughtWorks is your friend. I can script my browser to open websites, enter in data, capture results, under Python control and so able to do all that tedious testing you previously hated.

So here we go...

VirtualBox or KVM?

VirtualBox is a commercial company that launched a virtualisation machine manager (VMM) and then realised it might improve sales if it open-sourced the code. The OSS version is slightly crippled (it cannot pass through USB hard disks or something mostly irrelevant) but it is a good, solid platform and I highly recommend it. In fact, the above screenshot is from VirtualBox.

refs:

```
http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/virtualization-host.html
http://wiki.freebsd.org/VirtualBox
```

Install

```
cd emulators/virtualbox; make install clean
echo vboxdrv_load="YES" >> /boot/loader.conf
```

The principle behind setup is simple. Allocate a certain amount of your harddisk to become a single file. That single file will pretend to be a real Hard Disk to the VM, which will also have access to all your peripherals, network card and a pre-determined slice of RAM.

For once this is simple enough to do with a GUI - and I recommend keeping this simple. For me clever virtualisation using VMM is for the birds. Jails do more than enough if all you want is servers.

proc filesystem

VirtualBox makes use of the /proc file system (not same as linprocfs which we enabled for qemu - qemu was run as linux compatibility. Virtualbox runs native on FreeBSD.)

either

[6] Well, attitudes to this might be changing. ARM-based blade servers can actually deliver more CPU cycles per Watt, actually being more green, but other issues start to dominate, ranging from deciding if your work is IO bound or CPU bound, or handling the logistics required.

```
mount -t procfs proc /proc

(proc is a virtual device so it has no leading / - the mountpoint does however)
```

or put this into /etc/fstab. Probably best to do this...

```
proc            /proc           procfs  rw              0       0
```

so you see:

```
pbrian_laptop# cat /etc/fstab
# Device    Mountpoint FStype Options Dump    Pass#
/dev/ad4s1b none      swap    sw      0       0
...
proc              /proc       procfs  rw      0       0
```

Thats it. (Well, nearly). To start run

```
$ VirtualBox
```

You will get a GUI wizard (!).

It seems straightforward - firstly create the virtual hard disk (VirtualBox uses a format no other virtualisation software does, so this is the only way), assign the amount of RAM you want to let the VM use. Since my only intention with this form of virtualisation is to emulate simple desktop clients, 192MB is more than enough all round. (well, OK, this depends, emulating XP running a many flash clients *eats* RAM. God, this can be a painful setup.)

Oh, yes, 6GB of Hard disk space seems to be a good amount too. Down to 2GB and both ubunutu and windows might complain. I had a kernel panic - but I increased RAM from 64 to 192 (and gave it 2 cpus). worked ok.

Install the VM

We want the VM to boot up, and then look for a bootable image (ie the VM boots and is given a install CD). This can be done in two ways - by allowing the VM access to our own CD/DVD drive, or by mounting an ISO for it. THe second is a lot more convenient - especially for ubuntu.

So download the latest Ubuntu .iso (weirdly I was experimenting with this on the day of a new Ubunutu release, so my download took forever.) And then goto the CD tab in the VMM GUI. tick the 'Mount CD Drive' and then tick 'Mount from ISO'. Simply find the iso image on your HDD, and away we go.

Speeding things up

There are two innovations in hardware-based virtualisation. VMX extensions and nested paging. The first has been around for a couple of years, and, in essence, performs in hardware the process-marking function that we see in Jails (ie marking each process as belonging to a given VM).

The second is most recent, found in the Nehalem-range on Intel CPUs (Nehalem is surprisingly awkward to pronounce - I have it on good assurance that it is pronounced Neham-a-llama-la-la-ding-dong). Without this, a VM must pretend it holds a paging cache - so any cache misses end up being processed twice, once in the "real" cache, once in the VM version.

Anyway, a CPU without VMX flags and Neham-a-llama-la-la-ding-dong, can be around 3-5 times slower in virtualisation. So its worth it.

screensize

Refs

http://forums.opensuse.org/new-user-how-faq-read-only/unreviewed-how-faq/385392-virtualbox-screen-si

My god - what a palaver. The basic screen size of 800x640 pixels is what we get out of the (Virtual)Box. But try making the thing show the right size. You need the extra drivers produced by VirtualBox.

VirtualBox comes with a set of extra goodies, called GuestAdditions. This is an iso of drivers and so forth that amoungst other things allow you to set the screen resolution > 800x600.

Now, the instructions say click Devices, then Download Guest Additions. Unfortunately the download fails. The iso is not kept where the VMM thinks it is

The reason - the iso is normally included in the binary that one downloads, for FreeBSD its not.

so, to solve this we do the following ...

Open up my Ubunutu guest VM. In the VM, we install virtualbox-ose-guestaddition using synaptic package manager. So I download it from the Internet using the VM. But I need it in the host.

Then

```
$ updatedb
$ locate virtualbox | grep guest
```

There is a .deb package of 24MB, scp that to my actual host (yes, from the VM to the host) and use *deb2targz* to extract it, and eventually pull out the iso.

Now put the iso where devices> tries to look for it (/usr/local/share/virtualbox/VBoxGuestAdditions.iso)

Now it mounts a CDROM, inside the guest.

You need to run the appropriate install pkg on the cd rom for the guest OS (Linux/win32/solaris are what I can see)

Now with that in place we need to do some extra work

On the *host* machine set a VirtualBox parameter

```
$ VBoxManage setextradata global GUI/MaxGuestResolution any
```

Now restart your Guest OS. It is unlikely it is correct, and if that is a Unix box it is because the xorg.conf file is incorrect. I opened up /etc/xorg.conf in the guest machine and it had two lines from VirtualBox.

I then added

```
Section "Screen"
        Identifier "Screen0"
        Device     "Card0"
        Monitor    "Monitor0"
        DefaultDepth 24

        SubSection "Display"
                Viewport   0 0
                Depth     24
                Modes "1280x1024 1028x768"
        EndSubSection
EndSection
```

Restarted the Guest OS and suddenly I had a choice of resolutions.

Windows

Curiously windows resized, out of the box, with no fuss. But that is, I guess, I had already install GuestAdditions onto the VMM as above.

Loading from CD/DVD Drive

Booting from a downloaded iso is nice. However I have Windows Disk and wanted to load from CD.

This seems ok, but hald slowed my host machine down to a crawl - so I used a .iso of a windows disk This .iso approach is a *lot* simpler.

```
from the wiki
# atapicam kernel module needs to be loaded
# HAL has to run at the moment
# Permissions to access /dev/xpt0, /dev/cdN and /dev/passN

$ kldload atapicam
$ echo atapicam_load="YES" >> /boot/loader.conf
$ echo dbus_enable="YES" >> /etc/rc.conf
$ echo hald_enable="YES" >> /etc/rc.conf

http://www.freebsd.org/gnome/docs/halfaq.html
http://wiki.freebsd.org/VirtualBox
```

Networking

Getting a NAT based IP address for the VM machine is an out of the box issue - so one can initiate connections from the VM to anywhere. However, if you want to run SeleniumRC, you want to connect *to* the VM (the SeleniumRC runs a server that python client connects to and says "open browser to www.google.com".)

However, we want to do Ethernet bridging, - the simple way is to bring up the VMM. There are 4 virtual Ethernet adaptors - one can be the NAT, one can be a bridged adaptor to the real host NIC. So just set the NIC adapator number 2 to brided adaptor, and you can connect to that IP from any machine on the subnet the host is on.

KQemu

Qemu is a CPU emulator written by Fabrice Bellard. It is able to translate system calls from one OS, written for one CPU to the binary equivalents for another OS or another CPU. In short one can run Windows OS on a FreeBSD host.

With KQemu, this has been adapted to run on VMX flagged CPUs (specifically on Linux)

Install

```
cd emulators/kqemu-kmod; make install clean
cd emulators/qemu; make install clean

(setup Linux-compatibility on FreeBSD first too)
```

Somewhere to put it: we need a 'virtual disk' - a file that is nothing but empty space for the img to run. its very own hard disk, as it were.

```
qemu-img create /usr/local/DATA/ubuntu.disk 4G

or

dd of=/usr/local/DATA/ubuntu.disk bs=1024 seek=4194304 count=0
```

Now download the .iso file

```
fetch http://Whereever/the/hell/ubunutu/is/kept
```

Install it the normal way

```
qemu -hda /usr/local/DATA/ubuntu.disk -cdrom \
 /home/pbrian/downloads/ubuntu-9.10-desktop-i386.iso -m 192 -boot d
```

At this point the file image is just like a HDD that has been installed with Ubuntu But to start the image normally:

```
qemu -hda /usr/local/DATA/ubuntu.disk -m 192
```

Ripping XP CD

Its much easier with a .iso file than pass-through

> # dd if=/dev/acd0t01 of=/home/pbrian/downloads/xp.iso bs=2048

Really it was awful to get this far. Had to read the man page to the bottom - adding bs=2048 solved a lot.

dd if=/dev/acd0t01 of=/home/pbrian/downloads/xp.iso dd: /dev/acd0t01: Invalid argument 0+0 records in 0+0 records out 0 bytes transferred in 0.000084 secs (0 bytes/sec)

plus why the acd0t01 worked when acd0 did not?

Actually it seems to be the bs=2048 - CD's transfer at that rate and dd fails presumably after the first 512Bytes come and 513th is not buffered and a CRC fails somewhere.

Anyway,

> dd if=/dev/acd0 of=/home/pbrian/downloads/xp.iso bs=2048

works. Now I have the ISO I install it the usual way, and presto, Windows on Virtual Machine. And with Bridging.

VirtualBox

Installing

First guest

The first one to create is Unbuntu. Its simple and free. Download the latest version (www.ubuntu.com), and keep somewhere safe. Create a harddisk - there is a simple wizard on VirtualBox, I recommend 512MB RAM and min 8GB disk space, 5 years ago that was a damn good spec.!

Then open File > Virtual Media Manager, which will allow us to make the ISO visible. Just "add" a CD/DVD image, and it is available to the guest image. Now go to Settings for the image. Choose Storage and click on the CD below the IDE COntroller You should see on RHS "Attributes" of the CD device, and can select any one of the "mounted" CD drives above to "put into" the virtual CD tray.

Start the image as usual, and select boot menu (probably f12), boot from the CD. You should see a normal install from ubuntu. Just install.

Bridged Networking

Compile Kernel with Netgraph included Basically, bridge networking uses a driver in the Host OS, to bypass the usual host network stack (ie grab packets before they have gone through all the usual layers). Netgraph is one such popular way of doing this. Thus the virtual machine can take apacket off the real NIC, read it and then put its own reply back onto the real NIC at the same layer - it looks to the Host as if right next door on the Ethernet network is another NIC reading the packets it reads and putting more on the network.

The Virtual machine on the other hand just sets up its stack as normal, thinking it has access to a genuine real NIC.

Its a bit like this:

```
Host  VM
 \    /
  \  /
   --  Netgraph
   |
   NIC
```

But not really.

http://www.virtualbox.org/manual/ch06.html

bibliography

```
- https://help.ubuntu.com/community/Installation/QemuEmulator
- https://help.ubuntu.com/community/WindowsXPUnderQemuHowTo
- http://wiki.freebsd.org/qemu
- http://dryice.name/blog/freebsd/using-freebsd-as-a-network-bridge-and-use-dummynet-to-shape-the-tra
- http://www.freebsd.org/doc/en/books/handbook/network-bridging.html
```

6.4.32 wifi

Connecting to a wireless network is *de rigeur* these days. It's simple to do - apparently. For Mac users perhaps, but for us doughty FreeBSD folks, life is never so simple. Well, setting up a BSD wifi connection is not that bad really, it's just that compared to Mac, Windows or even Ubuntu, well it *is* awkward.

However, once we have, we find we can do other bigger things, like creating our own access point, running wireless sniffers and more, all just a little more confidently and knowledgably than those Mac and Windows users. And that frankly is the point - a little more pain in return for confidence, knowledge and clearer path into the 'advanced' functions.

Summary

With a normal Ethernet over wire, we establish a live *physical* circuit, send packets over that circuit and route accordingly.

With wifi we need to create a *radio ethernet*, essentially we sling a virtual wire between our adaptor and the Access Point (AP).

This virtual wire requires three things

- find the wifi adaptor on your machine (laptop)
- ensure the drivers are loaded into kernel and the adaptor is found
- configure adaptor with *ifconfig*

- authenticate with AP and so "associate"

Find the adaptor

To be honest this is clearly the most awkward point. I am using a work-supplied laptop, and finding the chipset that is built in is a RRPITA. To be perfectly honest, I am still unsure I have managed to.

Firstly look at dmesg for anything resembling an Ethernet driver or setup.

```
$ dmesg | grep -i ethernet
mskc0: <Marvell Yukon 88E8055 Gigabit Ethernet> mem 0xf5000000-0xf5003fff irq 19 at device 0.0 on pci
msk0: Ethernet address: 00:13:77:6f:93:ed
```

Well, thats my wired Ethernet. Now, if we want we can look for similar information in *sysctl -a*

```
mskc0: <Marvell Yukon 88E8055 Gigabit Ethernet> mem 0xf5000000-0xf5003fff irq 19 at device 0.0 on pci
msk0: <Marvell Technology Group Ltd. Yukon EC Ultra Id 0xb4 Rev 0x03> on mskc0
msk0: Ethernet address: 00:13:77:6f:93:ed
miibus0: <MII bus> on msk0
```

The thing to note here is that we can get useful stats from sysctl

```
dev.msk.0.stats.rx.ucast_frames: 0
dev.msk.0.stats.rx.bcast_frames: 0
```

OK, well, msk is the driver for my fixed ethernet. So, err, what about the wifi? Well the best I can find is what appears to be a dual function Broadcom chip. And all the Google-butt [7] says that Broadcom hates Unix and this chipset stands about as much chance working with FreeBSD as Michael Jackson does working with the Abused Children's Dance Therapy group.

```
ugen1: <Broadcom Corp BCM92045NMD, class 224/1, rev 2.00/3.54, addr 2> on uhub6
```

So I went and bought a USB stick

```
rum0: <Belkin Belkin 54g USB Network Adapter, class 0/0, rev 2.00/0.01, addr 2> on uhub7
rum0: MAC/BBP RT2573 (rev 0x2573a), RF RT2528
rum0: WARNING: using obsoleted IFF_NEEDSGIANT flag
rum0: Ethernet address: 00:22:75:fd:d7:8a
```

I plug it in, ugen (the standard USB driver) finds it, then the rum driver is identified and hooray.

I think I want to be more ... hackerish on this. But I have things to do, and that USB dongle has now served 3 laptops quite happily.

What drivers work with what chipsets?

[EXTEND - how does kernel locate drivers - device to driver mapping - find the right .h file.]

This is a slight diversion, but it is important - how does the Kernel know which driver to assign to which device. Firstly it identifies the device.

- device on pci bus
- device on usb bus

[7] Like scuttlebutt, but comes from reading Google found entries on whatever subject you are looking for. I just made it up cos it sounded good, but there are clearly some other interpretations. I just cannot stop hearing David Schwimmer singing "I like big butts ...". Sad or what.

What drivers for what chipset?

Basically, read the man pages. Each ethernet driver (I think thet are rum, ath, zyd, upgt ...) has a man page and in there it is listed which chipsets that driver supports

However we can run

```
pciconf -lv
```

which will show

```
none1@pci0:2:0:0:       class=0xff0000 card=0x3577103c chip=0x520910ec rev=0x01 hdr=0x00
  vendor     = 'Realtek Semiconductor Co., Ltd.'
re0@pci0:6:0:0: class=0x020000 card=0x3577103c chip=0x813610ec rev=0x05 hdr=0x00
  vendor     = 'Realtek Semiconductor Co., Ltd.'
  device     = 'RTL8101E/RTL8102E PCI Express Fast Ethernet controller'
  class      = network
  subclass   = ethernet
none2@pci0:7:0:0:       class=0x028000 card=0x1636103c chip=0x53901814 rev=0x00 hdr=0x00
  vendor     = 'Ralink corp.'
  class      = network
```

ifconfig settings

We are assuming that the wifi adaptor has been found and that a driver is assigned, We now want to connect to the Access Point (AP).

I do recommend reading the ifconfig man pages. It is really useful.

Scan for AP's

The first thing to do is see if the adaptor can find the AP

```
$ ifconfig wlan0 up
$ ifconfig wlan0 list scan
```

In a NY hotel room I get

```
pbrian_laptop# ifconfig rum0 list scan
SSID            BSSID              CHAN RATE   S:N     INT CAPS
Buckingham_...  00:c0:02:0f:ac:67    8   54M -89:-95  100 ES

           ^^^
```

Err, why the ... try -v option to view SSID's that are longer than usual.:

```
pbrian_laptop# ifconfig -v rum0 list scan
SSID                            BSSID              CHAN RATE   S:N     INT CAPS
Buckingham_Hotel                00:c0:02:0f:ac:67    8   54M -89:-95  100 ES
```

unencrypted

So, this is an unencrypted link, with an SSID and a channel.

/etc/rc.conf

```
ifconfig_rum0="ssid Buckingham_Hotel channel 8 authmode open DHCP"
```

(authmode - pretty much everyone uses open. When we are in an infrastructure mode (ie using Access Points that create an infrastucture the clients hang off, we need to say we are in infrastructure mode and then choose an Authentication mode. It is usually open (with WEP) or wpa). The alternative to infrastructure mode is ad-hoc - its like the OLPC idea of grid connections. Anyway, use open for now)

And, hey presto I get a connection to the AP in the hotel. And a web page wanting 12.95 a day for access... But it worked.

So next, encrypted links

WEP

WEP needs a shared key - one that is known to the client and the AP. Unfortunately most people never change the key and it can take only a few hundren megabytes of traffic to break the encryption, so WEP is not actually secure, just secure enough to prevent your neighbours from seeing what you surf [8]

```
ifconfig_rum0="ssid HOME channel 6 authmode open wepmode on weptxkey 1 wepkey 0xyyyyyyyyyyyyyyyyyyyyyy
```

WEP key is a 26 digit hex (0x prefix tells us that. If you get "String too long" errors, put 0x in to explain this is a hex string). Now "weptxkey 1" - there are several "slots" to use in the shared key - basically the AP has several shared keys it could use, you need to tell it which slot you are thinking of. Weptkey is the means. Ifconfig man suggests that using

```
wepkey 1:0xyyyyyyyyyyyy
```

but this does not work for me, Stick to being explicit.

So, this connects happily to my HOME AP.

Note on DHCP and ifconfig usually I can easily transliterate the following:

```
 ifconfig_rum0="ssid HOME channel 6 authmode open wepmode on weptxkey 1 wepkey 0xyyyyyyyyyyyyyyyyyyyyyy

 in /etc/rc.conf can also be set up during run time

 ifconfig rum0 ssid HOME channel 6 authmode open wepmode on weptxkey 1 wepkey 0xyyyyyyyyyyyyyyyyyyyyyy
netmask 255.255.255.0
```

however, the following raises an error

```
ifconfig rum0   ssid HOME channel 6 authmode open wepmode on weptxkey 1 wepkey 0xyyyyyyyyyyyyyyyyyyyyyy
```

while this, in /etc/rc.conf is fine

```
ifconfig_rum0="ssid HOME channel 6 authmode open wepmode on weptxkey 1 wepkey 0xyyyyyyyyyyyyyyyyyyyyyy
```

testing /etc settings Change settings in /etc/rc.conf, then run

```
pbrian_laptop# /etc/rc.d/netif start
```

[8] THis is not the same as SSL being broken. If your neighbour breaks your WEP key its as if he has plugged into your wired switch/hub. He can read all your packets. He can see which porn sites you look at, but if you use SSL to enter your credit card details, he cannot get the important information. (You do use SSL pages when buying your porn don't you :-). And when did you stop beating your wife ?

All Change

The above was very useful. Back in the day. However things have moved on and now the approach is to use wpa security.

in /etc/rc.conf:

```
wlan_load="YES"

#making wep /wpa work
wlan_wep_load="YES"
wlan_ccmp_load="YES"
wlan_tkip_load="YES"

wlans_rum0="wlan0"
ifconfig_wlan0="WPA DHCP"
```

this essentially replaces the ifconfig calls earlier in rc.conf. All of FreeBSD wifi is handled by a super driver - wlan. We tell this that we want to use the rum0 driver, and refer to it as wlan0.

using WPA security

As noted above, WEP security was essentially broken, and advice from respected security advisors such as Bruce Schneier was to run open wifi.[*]_

discussion of settings

SSID Here we state the SSID of the base station we want to connect to. (SSID *can* refer to more than one base station (usually if a company wants to give

> a seamless experiece in a wide area - in this case we can use BSSID which is a sort of mac address for base stations)

channel We can set the channel to use. Not always necessary but can be found from the data in "ifconfig rum0 list scan"

authmode open / shared. In general WEP is considered not secure. Shared authentication less so. Use Open for trivial home surfing and watch your SSL padlock

wepmode on / off Kind of obvious

weptxkey wep keys need a "slot" - the key itself is known to the client and the AP, but each needs to know which slot they are in. this is seen in ifconfig as

```
ifconfig
      status: associated
      ssid HOME channel 6 (2437 Mhz 11g) bssid 00:14:6c:d4:e4:e4
      authmode OPEN privacy ON *deftxkey 1* wepkey 1:104-bit txpower 50
      bmiss 7 scanvalid 60 bgscan bgscanintvl 300 bgscanidle 250
```

defttxkey - weptkey is deprecated in favour of this term

wepkey the key itself, use 0x12345 if 12345 is the HEX version of key

DHCP obvious

NB

- if usng ifconfig on comamnd line put DHCP in early not at end of string.

Well, thats about it for now folks. This will be an oft-revisited entry I think.

6.4.33 Advice to a Mate going Freelance for the first time

> 2 points by lifeisstillgood 2 days ago | link | parent | on: Ask HN: First Contract Job, what to do?
>
> Things I wish I had done:
>
> 1. Thought it would last longer than one month. I assumed (still do) that they would realise I was faking it and take it all away. So I paid extra for temporary access to things (rented a car not bought). A year later I still was. Stupid. Assume you are now a contractor for five years. You probably will be these don't sound like people to stick with.
> 2. Think of yourself as a business. Not a contractor - a business who needs to market and build a "sales pipeline". Who are you going to work for in six months time? doing what? How will you find them in the next three months and persuade them to wait?
> 3. taxes. Oh shit taxes. Get a accountant and do not leave their office on the first day until you have written a spreadsheet and calculated your own basic tax return. know this stuff inside out. You are now running 4 tabs on a spreadsheet and carrying over from one to the other:
>
> Business Income > Business Outgoings > Home Income > Home outgoings. Make a spreadsheet like that today. Fill it in.
>
> 4. Blog - it's probably the best middle path for marketing yourself. But choose your story, your unique take on things. Are you techie through and through? A Brogrammer? An outdoorsy hiker, a Beethoven lover? What is your (oh god) tribe? Talk to that tribe - somewhere there is a hiring manager who is also a member of your tribe. Probably quite a few - talk to her through your blog.
> 5. Never ever ever ever lie for anyone ever. Politics is a hard game especially in companies that are changing a lot, like startups. As a contractor you can have a professional armour - you have an obligation to give unbiased advice and opinion.
>
> Not the companies problems are not your problems. Do not take on board their views of "if only we can hit this deadline we can make the sales and hit 100M..." - you should give your real views on the state of the codebase, the likelihood of hitting the deadlines and the tech debt.

Go read the clean coder (with an r) by uncle bob martin.

5. be your own project manager - that is track in your own system the tasks needed to achieve the requirements and the likely ship dates. Don't keep this a secret but don't rely on someone else tracking this for you. I recommend fogbugz these days (it's for pay). Don't touch Redmine with a barge pole. Basically know that the two week estimate the sprint just made is bullshit. Be clear on what your estimates are - no need to be shrill, just be clear, polite and consistent.
6. Be calmer than everyone else and realise this all takes time.
7. Don't take career advice off random people in the Internet :-)

tptacek 2 days ago | link

This is great, but I don't think he's trying to start a consulting company; he's trying to navigate a temp-to-perm offer. :)

Sorry I got a little carried away (then noprocrast stopped me reediting.)

> Anyway, if I was to give our hero here any advice, it would be to see himself not as an employee but as a consulting company - if POV at Parc Xerox was worth 40 IQ points, in our business I would say POV is worth 40k pa.

Hello there old boy ! How are you ? Now, when I heard you were dropping the so-called safe corporate world for the stomry waters of freelancing, well, I turned my car around and we met in our favourite pub-by-the-Thames and had a chat.

You asked if I could point out some of the lessons I had learnt over the past few years, and what started as a chat, became a long email and now is an article or two. Sorry about that. Its likely this is more my therapy than your actionable advice, but I hope it proves useful.

Firstly, what this is not. This is not a list of things to go do now and you will be successful. Its far more a description of the landscape in front of you. If you are reasonably prepared for the terrain and weather, you will not get caught by surprise, not spend a lot of wasted time going in the wrong direction and hopefully this will mean your natural advantages will end up with you making a few quid.

Which leads onto the first subject. Money.

You are here to make it. So *don't ever drop your rates*

Money The future shape of the world

- new education

Money again (Tax) Money yet again (Domestic spending) Small companies, remote working Evolution of mgmt (the new freelancing company) - acqui-hire and loosely coupled investment. (international tax as a political force, specific exnterprise zones in metro areas, and ability to become a tax-associate of one ie an acquihire)

  -> we are becoming the new middle managers, and our job is three fold -> innovation, mentoring,

- The fundamental shift -> virtual and physical has been linked, and is growing closer.
- GeoPolitics -> much harder, but expect metroarea level focus, with the likely collapse of nation-state economic agreements coming in (basiclly no one expects we can keep paying western lifestyles. SOmething has to give and expecting tech to step in and make it all cheaper is denial)

Expected partial-failure of nation-states. There will not be an apocalypse.

In that situation, the sensible answer is to have at least two homes. Ready.

- New tech -> The internet is connecting everyone to everyone else (forget censorship, its as dead as privacy. You want in on the new world, you drop the firewall). But building on that will be green energy, internet of things and biotech. These will make it possible for a mega-city / metroarea to actually work.
- We are looking like angels with no capital. We are going to be the new apprentice masters for the new apprentices
- The means of recharging -> developing countries for a year.
- Collectivisit income co-operatives.
-

The end game.

Stuff fairness and equality. Become part of the rent-based upper class.

http://www.forbes.com/sites/venkateshrao/2012/05/18/nextbigthingology-the-world-after-facebook/2/

http://www.forbes.com/sites/venkateshrao/2012/09/03/entrepreneurs-are-the-new-labor-part-i/

6.5 postgres-cheatsheet

6.5.1 PostgreSQL Cheat Sheet

I am sure I stole this from somewhere but I cannot find the source. Please take this with a degree of plagarism.

Start off:

```
CREATE DATABASE dbName;

CREATE TABLE (with auto numbering integer id)

CREATE TABLE tableName (
 id serial PRIMARY KEY,
 name varchar(50) UNIQUE NOT NULL,
 dateCreated timestamp DEFAULT current_timestamp
);
```

Add a primary key:

```
ALTER TABLE tableName ADD PRIMARY KEY (id);
```

Create an INDEX:

```
CREATE UNIQUE INDEX indexName ON tableName (columnNames);
```

Backup a database (command line):

```
pg_dump dbName > dbName.sql
```

Backup all databases (command line):

```
pg_dumpall > pgbackup.sql
```

Run a SQL script (command line):

```
psql -f script.sql databaseName
```

Search using a regular expression:

```
SELECT column FROM table WHERE column ~ 'foo.*';
```

The first N records:

```
SELECT columns FROM table LIMIT 10;
```

Pagination:

```
SELECT cols FROM table LIMIT 10 OFFSET 30;
```

Prepared Statements:

```
PREPARE preparedInsert (int, varchar) AS
  INSERT INTO tableName (intColumn, charColumn) VALUES ($1, $2);
EXECUTE preparedInsert (1,'a');
EXECUTE preparedInsert (2,'b');
DEALLOCATE preparedInsert;
```

Create a Function:

```
CREATE OR REPLACE FUNCTION month (timestamp) RETURNS integer
 AS 'SELECT date_part(''month'', $1)::integer;'
LANGUAGE 'sql';
```

Table Maintenance:

```
VACUUM ANALYZE table;
```

Reindex a database, table or index:

```
REINDEX DATABASE dbName;
```

Show query plan:

```
EXPLAIN SELECT * FROM table;
```

Import from a file:

```
COPY destTable FROM '/tmp/somefile';
```

Show all runtime parameters:

```
SHOW ALL;
```

Grant all permissions to a user:

```
GRANT ALL PRIVILEGES ON table TO username;
```

Perform a transaction:

```
BEGIN TRANSACTION
 UPDATE accounts SET balance += 50 WHERE id = 1;
COMMIT;
```

Basic SQL

Get all columns and rows from a table:

```
SELECT * FROM table;
```

Add a new row:

```
INSERT INTO table (column1,column2)
VALUES (1, 'one');
```

Update a row:

```
UPDATE table SET foo = 'bar' WHERE id = 1;
```

Delete a row

```
DELETE FROM table WHERE id = 1;
```

6.6 Building a workstation laptop from Scratch

6.6.1 Sensible Mail Handling

1. Fetchmail delivers mail to you local drive

2. mutt / procmail / stuff sorts mail in your maildir
3. exim4 will accept from localhost and deliver to anyone or sendgrid

http://www.debian.org/releases/wheezy/i386/ch08s05.html.en

just use Google

6.6.2 aspell

This is an enormously painful setup and generally a painful program. But its a limited choice world it seems.

Install aspell is pretty simple. */usr/ports/textproc/aspell*

We then want to get dictionaries - which are word lists in specific formats. The only place to get these and remain sane is http://ftp.gnu.org/gnu/aspell/dict/en

After untarring, we run:

```
$ ./configure
Finding Dictionary file location ... /usr/local/share/aspell
Finding Data file location ... /usr/local/lib/aspell-0.60
$ sudo make
$ sudo make install
```

Now if we look in /usr/local/share/aspell we have a huge list of english words in aspell formats.

Using it with emacs:

```
M-$
or
M-x flyspell-mode
```

6.7 closure_like_functions

6.7.1 Closures in Python

XXX

```
"""
Its hard to say if this is closures. It kind of is I guess.
"""

callbacks = []

def generate(a):
    def acallback():
        return "the value of a when I existed was %s" % a
    callbacks.append(acallback)

generate(1)
generate(2)
generate(3)

for i in callbacks:
    print i()
```

6.8 Using Sphinx and GitHub (gh-pages)

6.8.1 Setting up python repos to use Sphinx and Github Pages

We want to use Sphinx to generate our documentation but then use github to host the documentation (not withstanding the excellent readthedocs.)

Github pages are a relatively simple idea - create a branch with a special name (*gh-pages*) in your repo. Whatever (html) is in the root of that branch will be served from http://reponame.github.com as if it was the document root.

So our basic process will be to create an orphan branch in our repo called "gh-pages", then generate the docs in the normal fashion, and copy over the docs to the new branch and commit.

Obviously I would like to automate the complete process, of building the docs, but that will wait till we have greater control and comfort with creating venvs automatically and running docs within them (Its doable - just ...)

First stage

We want to create a new orphan branch called gh-pages (orphans carry no history baggage so its neater)

:: # cd into repo on disk $ git checkout –orphan gh-pages Switched to a new branch 'gh-pages' $ git rm -rf .

> rhaptos2.user$ echo "Initial commit" > index.html rhaptos2.user$ git add index.html rhaptos2.user$ git commit -m "First cut for gh-pages"

Now that we have on github a gh-pages branch, we want to put something in it. Create a file like the one below, in the root of say master.

```
#
SRCDOCS=/home/pbrian/src/rhaptos2.user/docs/_build/html

cd $SRCDOCS
MSG="Adding gh-pages docs for `git log -1 --pretty=short --abbrev-commit`"

TMPREPO=/tmp/docs/user
rm -rf $TMPREPO
mkdir -p -m 0755 $TMPREPO

git clone git@github.com:Connexions/rhaptos2.user.git $TMPREPO
cd $TMPREPO
git checkout gh-pages  ###gh-pages has previously one off been set to be nothing but html
cp -r $SRCDOCS/ $TMPREPO
git add -A
git commit -m "$MSG" && git push origin gh-pages
```

I am assuming that we have already built the documentation in branch <master> as in:

```
$ cd docs
$ workon vuser
$ (vuser) make clean && make html
```

This can all clearly be adjusted to suit each repo, and easily generic-isized. But it does the job - of creating a temporary clone of gh-pages, and bringing the pre-built html into the right location in gh-pages.

/usr/pbi/rxvt-unicode-amd64/lib/urxvt/perl/ thats were tabbed brwstring is in urxvt

6.9 Todo

A collation of to do notes in the docs.

Todo

More about pkgng [expand on pkgng]

(The *original entry* is located in /var/build/user_builds/executableopinions/checkouts/latest/docs/labs/automated-builds/workstation.rst, line 74.)

Todo

Mail encryption. Keeping safe what you say to criminals and neer-do-wells.

(The *original entry* is located in /var/build/user_builds/executableopinions/checkouts/latest/docs/labs/misc/email.rst, line 23.)

Todo

Mail rules - setting up and managing Google Apps for business email rules.

(The *original entry* is located in /var/build/user_builds/executableopinions/checkouts/latest/docs/labs/misc/email.rst, line 27.)

Todo

links on aircon

(The *original entry* is located in /var/build/user_builds/executableopinions/checkouts/latest/docs/labs/misc/hardware.rst, line 12.)

Todo

show to AST for this and how elimating tail call could elimnate the stack.

(The *original entry* is located in /var/build/user_builds/executableopinions/checkouts/latest/docs/labs/python-tricks/fizzbuzz.rst, line 37.)

Todo

Also need to deal with command_interpreter.

(The *original entry* is located in /var/build/user_builds/executableopinions/checkouts/latest/docs/labs/supervisor-scripts/well-behaved-services.rst, line 237.)