# Dan's Cheat Sheets Documentation

***Release 1***

**Dan Poirier**

November 05, 2017

Contents

Contents:

CHAPTER 1

# Ansible

This is growing into a minimal Ansible reference of sorts, since Ansible's own docs have nothing like a reference.

- Ansible.
- list of keys that common playbook objects can take.
- Release tarballs
- Ansible documentation for older releases

## Running Ansible tasks in the background

Example:

```
- name: start collectstatic in the background
  command: "{{ install_root }}/env/bin/python manage.py collectstatic --noinput -v 0"
  args:
    chdir: "{{ install_root }}/webapp"
  async: 1000
  poll: 0
  register: collectstatic_bg

################################################################################
#
# PUT TASKS HERE THAT DON'T NEED TO BE RUN BEFORE COLLECTSTATIC CAN START,
# AND THAT WON'T AFFECT THE BACKGROUND COLLECTSTATIC.
#
################################################################################
- name: clean up local tarball
  delegate_to: 127.0.0.1   # Run on localhost
  run_once: yes            # only once
  become: no               # Don't need sudo
  file:
    state: absent
    path: "{{ tarball }}"

- name: migrate
  command: "{{ install_root }}/env/bin/python manage.py migrate --noinput"
  args:
    chdir: "{{ install_root }}/webapp"

- name: install tasks
```

```
  command: "{{ install_root }}/env/bin/python manage.py installtasks --traceback"
  args:
    chdir: "{{ install_root }}/webapp"

################################################################################
#
# Check every 'delay' seconds, up to 'retries' times, until collectstatic is done
#
################################################################################
- name: wait for collectstatic to finish
  async_status: jid={{ collectstatic_bg.ansible_job_id }}
  register: job_result
  until: job_result.finished
  retries: 80
  delay: 15

################################################################################
#
# PUT TASKS AFTER THIS THAT CAN'T RUN UNTIL COLLECTSTATIC IS DONE
#
################################################################################
```

## Blocks

- Blocks doc
- A blog post about blocks
- Blog post with examples
- Complete list of possible keywords

Blocks can be used anywhere a task can (mostly?). They allow applying task keys to a group of tasks without having to repeat them over and over. They also provide a form of error handling.

Syntax:

```
block:
  - <task1>
  - <task2>
when: <condition>
become: true
become_user: <username>
....
[rescue:
  - debug: msg="This task runs if there's an error in the block"
  - <task2>
  ...
always:
  - debug: msg="I always run"
  ... more tasks ..
]
```

## Conditionals

doc

### Conditional tasks

See *Task*.

## Configuration

### Configuration file

**Syntax** .ini file

See The Ansible Configuration File doc.

**Ansible uses the first config file it finds on this list:**

- ANSIBLE_CONFIG (an environment variable)
- ansible.cfg (in the current directory)
- .ansible.cfg (in the home directory)
- /etc/ansible/ansible.cfg

Some useful vars in the `[defaults]` section:

#### hostfile

This is the default location of the inventory file, script, or directory that Ansible will use to determine what hosts it has available to talk to:

hostfile = /etc/ansible/hosts

#### roles_path

The roles path indicate additional directories beyond the 'roles/' subdirectory of a playbook project to search to find Ansible roles. For instance, if there was a source control repository of common roles and a different repository of playbooks, you might choose to establish a convention to checkout roles in /opt/mysite/roles like so:

roles_path = /opt/mysite/roles

Additional paths can be provided separated by colon characters, in the same way as other pathstrings:

roles_path = /opt/mysite/roles:/opt/othersite/roles

Roles will be first searched for in the playbook directory. Should a role not be found, it will indicate all the possible paths that were searched.

## Inventory

### Inventory directory

Whatever directory the *Inventory file* is in.

### Inventory file

**Default** `/etc/ansible/hosts`

**Change** set `ANSIBLE_HOSTS` in environment

```
ansible-playbook -i <inventoryfile> ...
```

set *hostfile* in configuration

**Syntax** .ini file, except initial lines don't need to be in a section

The inventory file is basically a list of hostnames or IP addresses, one per line. Can include port with `hostname:port` or `address:port`.

Ranges: Including `[m:n]` in a line will repeat the line for every value from `m` through `n`. `m` and `n` can be numbers or letters:

```
[mygroup]
host[1:25]
```

Host *Variables*: Can specify per-host options after hostname on the same line. E.g.:

```
jumper ansible_ssh_port=5555 ansible_ssh_host=192.168.1.50
```

See also *Variables files*.

Group *Variables*: add `[groupname:vars]` section and put var definitions in it, one per line. Example:

```
[all:vars]
project_name="PeterPan"
environment_name=staging
```

See also *Variables files*.

Groups of groups: add `[newgroupname:children]` and put other group names in it, one per line:

```
[group3]
host13
host14

[group3:children]
group1
group2

[group3:vars]
group3_var1=27
group3_var2="Hello, World"
```

## Invoking

### Ad-hoc

To run an ad-hoc command, use *ansible*. (But you almost always will want to run a playbook; see below.)

Examples of ad-hoc commands:

```
$ ansible all -m ping
# as bruce
$ ansible all -m ping -u bruce
```

```
# as bruce, sudoing to root
$ ansible all -m ping -u bruce --sudo
# as bruce, sudoing to batman
$ ansible all -m ping -u bruce --sdo --sudo-user batman
$ ansible all -a "/bin/echo hello"
```

Help:

```
Usage: ansible <host-pattern> [options]

Options:
  -a MODULE_ARGS, --args=MODULE_ARGS
                        module arguments
  -k, --ask-pass        ask for SSH password
  --ask-su-pass         ask for su password
  -K, --ask-sudo-pass   ask for sudo password
  --ask-vault-pass      ask for vault password
  -B SECONDS, --background=SECONDS
                        run asynchronously, failing after X seconds
                        (default=N/A)
  -C, --check           don't make any changes; instead, try to predict some
                        of the changes that may occur
  -c CONNECTION, --connection=CONNECTION
                        connection type to use (default=smart)
  -f FORKS, --forks=FORKS
                        specify number of parallel processes to use
                        (default=5)
  -h, --help            show this help message and exit
  -i INVENTORY, --inventory-file=INVENTORY
                        specify inventory host file
                        (default=/etc/ansible/hosts)
  -l SUBSET, --limit=SUBSET
                        further limit selected hosts to an additional pattern
  --list-hosts          outputs a list of matching hosts; does not execute
                        anything else
  -m MODULE_NAME, --module-name=MODULE_NAME
                        module name to execute (default=command)
  -M MODULE_PATH, --module-path=MODULE_PATH
                        specify path(s) to module library
                        (default=/usr/share/ansible/)
  -o, --one-line        condense output
  -P POLL_INTERVAL, --poll=POLL_INTERVAL
                        set the poll interval if using -B (default=15)
  --private-key=PRIVATE_KEY_FILE
                        use this file to authenticate the connection
  -S, --su              run operations with su
  -R SU_USER, --su-user=SU_USER
                        run operations with su as this user (default=root)
  -s, --sudo            run operations with sudo (nopasswd)
  -U SUDO_USER, --sudo-user=SUDO_USER
                        desired sudo user (default=root)
  -T TIMEOUT, --timeout=TIMEOUT
                        override the SSH timeout in seconds (default=10)
  -t TREE, --tree=TREE  log output to this directory
  -u REMOTE_USER, --user=REMOTE_USER
                        connect as this user (default=poirier)
  --vault-password-file=VAULT_PASSWORD_FILE
                        vault password file
```

```
  -v, --verbose         verbose mode (-vvv for more, -vvvv to enable
                        connection debugging)
  --version             show program's version number and exit
```

## Playbooks

To run a playbook, use ansible-playbook. Here's the help from 2.0.1.0:

```
Usage: ansible-playbook playbook.yml

Options:
  --ask-become-pass     ask for privilege escalation password
  -k, --ask-pass        ask for connection password
  --ask-su-pass         ask for su password (deprecated, use become)
  -K, --ask-sudo-pass   ask for sudo password (deprecated, use become)
  --ask-vault-pass      ask for vault password
  -b, --become          run operations with become (nopasswd implied)
  --become-method=BECOME_METHOD
                        privilege escalation method to use (default=sudo),
                        valid choices: [ sudo | su | pbrun | pfexec | runas |
                        doas ]
  --become-user=BECOME_USER
                        run operations as this user (default=root)
  -C, --check           don't make any changes; instead, try to predict some
                        of the changes that may occur
  -c CONNECTION, --connection=CONNECTION
                        connection type to use (default=smart)
  -D, --diff            when changing (small) files and templates, show the
                        differences in those files; works great with --check
  -e EXTRA_VARS, --extra-vars=EXTRA_VARS
                        set additional variables as key=value or YAML/JSON
  --flush-cache         clear the fact cache
  --force-handlers      run handlers even if a task fails
  -f FORKS, --forks=FORKS
                        specify number of parallel processes to use
                        (default=5)
  -h, --help            show this help message and exit
  -i INVENTORY, --inventory-file=INVENTORY
                        specify inventory host path
                        (default=/etc/ansible/hosts) or comma separated host
                        list.
  -l SUBSET, --limit=SUBSET
                        further limit selected hosts to an additional pattern
  --list-hosts          outputs a list of matching hosts; does not execute
                        anything else
  --list-tags           list all available tags
  --list-tasks          list all tasks that would be executed
  -M MODULE_PATH, --module-path=MODULE_PATH
                        specify path(s) to module library (default=None)
  --new-vault-password-file=NEW_VAULT_PASSWORD_FILE
                        new vault password file for rekey
  --output=OUTPUT_FILE  output file name for encrypt or decrypt; use - for
                        stdout
  --private-key=PRIVATE_KEY_FILE, --key-file=PRIVATE_KEY_FILE
                        use this file to authenticate the connection
  --scp-extra-args=SCP_EXTRA_ARGS
                        specify extra arguments to pass to scp only (e.g. -l)
```

```
  --sftp-extra-args=SFTP_EXTRA_ARGS
                        specify extra arguments to pass to sftp only (e.g. -f,
                        -l)
  --skip-tags=SKIP_TAGS
                        only run plays and tasks whose tags do not match these
                        values
  --ssh-common-args=SSH_COMMON_ARGS
                        specify common arguments to pass to sftp/scp/ssh (e.g.
                        ProxyCommand)
  --ssh-extra-args=SSH_EXTRA_ARGS
                        specify extra arguments to pass to ssh only (e.g. -R)
  --start-at-task=START_AT_TASK
                        start the playbook at the task matching this name
  --step                one-step-at-a-time: confirm each task before running
  -S, --su              run operations with su (deprecated, use become)
  -R SU_USER, --su-user=SU_USER
                        run operations with su as this user (default=root)
                        (deprecated, use become)
  -s, --sudo            run operations with sudo (nopasswd) (deprecated, use
                        become)
  -U SUDO_USER, --sudo-user=SUDO_USER
                        desired sudo user (default=root) (deprecated, use
                        become)
  --syntax-check        perform a syntax check on the playbook, but do not
                        execute it
  -t TAGS, --tags=TAGS  only run plays and tasks tagged with these values
  -T TIMEOUT, --timeout=TIMEOUT
                        override the connection timeout in seconds
                        (default=10)
  -u REMOTE_USER, --user=REMOTE_USER
                        connect as this user (default=None)
  --vault-password-file=VAULT_PASSWORD_FILE
                        vault password file
  -v, --verbose         verbose mode (-vvv for more, -vvvv to enable
                        connection debugging)
  --version             show program's version number and exit
```

## Hosts pulling config

Ansible-pull (doc) is a small script that will checkout a repo of configuration instructions from git, and then run ansible-playbook against that content.

Assuming you load balance your checkout location, ansible-pull scales essentially infinitely.

Help from ansible-pull 2.0.1.0:

```
Usage: ansible-pull -U <repository> [options]

Options:
  --accept-host-key     adds the hostkey for the repo url if not already added
  --ask-become-pass     ask for privilege escalation password
  -k, --ask-pass        ask for connection password
  --ask-su-pass         ask for su password (deprecated, use become)
  -K, --ask-sudo-pass   ask for sudo password (deprecated, use become)
  --ask-vault-pass      ask for vault password
  -C CHECKOUT, --checkout=CHECKOUT
                        branch/tag/commit to checkout.  Defaults to behavior
```

```
                      of repository module.
-c CONNECTION, --connection=CONNECTION
                      connection type to use (default=smart)
-d DEST, --directory=DEST
                      directory to checkout repository to
-e EXTRA_VARS, --extra-vars=EXTRA_VARS
                      set additional variables as key=value or YAML/JSON
-f, --force           run the playbook even if the repository could not be
                      updated
--full                Do a full clone, instead of a shallow one.
-h, --help            show this help message and exit
-i INVENTORY, --inventory-file=INVENTORY
                      specify inventory host path
                      (default=/etc/ansible/hosts) or comma separated host
                      list.
-l SUBSET, --limit=SUBSET
                      further limit selected hosts to an additional pattern
--list-hosts          outputs a list of matching hosts; does not execute
                      anything else
-m MODULE_NAME, --module-name=MODULE_NAME
                      Repository module name, which ansible will use to
                      check out the repo. Default is git.
-M MODULE_PATH, --module-path=MODULE_PATH
                      specify path(s) to module library (default=None)
--new-vault-password-file=NEW_VAULT_PASSWORD_FILE
                      new vault password file for rekey
-o, --only-if-changed
                      only run the playbook if the repository has been
                      updated
--output=OUTPUT_FILE  output file name for encrypt or decrypt; use - for
                      stdout
--private-key=PRIVATE_KEY_FILE, --key-file=PRIVATE_KEY_FILE
                      use this file to authenticate the connection
--purge               purge checkout after playbook run
--scp-extra-args=SCP_EXTRA_ARGS
                      specify extra arguments to pass to scp only (e.g. -l)
--sftp-extra-args=SFTP_EXTRA_ARGS
                      specify extra arguments to pass to sftp only (e.g. -f,
                      -l)
--skip-tags=SKIP_TAGS
                      only run plays and tasks whose tags do not match these
                      values
-s SLEEP, --sleep=SLEEP
                      sleep for random interval (between 0 and n number of
                      seconds) before starting. This is a useful way to
                      disperse git requests
--ssh-common-args=SSH_COMMON_ARGS
                      specify common arguments to pass to sftp/scp/ssh (e.g.
                      ProxyCommand)
--ssh-extra-args=SSH_EXTRA_ARGS
                      specify extra arguments to pass to ssh only (e.g. -R)
-t TAGS, --tags=TAGS  only run plays and tasks tagged with these values
-T TIMEOUT, --timeout=TIMEOUT
                      override the connection timeout in seconds
                      (default=10)
-U URL, --url=URL     URL of the playbook repository
-u REMOTE_USER, --user=REMOTE_USER
                      connect as this user (default=None)
```

```
  --vault-password-file=VAULT_PASSWORD_FILE
                        vault password file
  -v, --verbose         verbose mode (-vvv for more, -vvvv to enable
                        connection debugging)
  --verify-commit       verify GPG signature of checked out commit, if it
                        fails abort running the playbook. This needs the
                        corresponding VCS module to support such an operation
  --version             show program's version number and exit
```

# Loops

See http://docs.ansible.com/playbooks_loops.html

## Iterating with nested loops

Write a task:

```
- module: args
  with_subelements:
    - thelist
    - fieldname
```

Then Ansible will essentially do this:

```
for thing in thelist:
   item.0 = thing
   for fieldvalue in get(thing, fieldname):
      item.1 = fieldvalue
      EXECUTE (module, args)
```

In other words, it'll iterate over the first value as a list, call it item.0, then get the list from that value's field named 'fieldname', and iterate over that as well, calling it item.1.

Presumably you could nest this deeper.

Example from the docs. With variables:

```
---
users:
  - name: alice
    authorized:
      - /tmp/alice/onekey.pub
      - /tmp/alice/twokey.pub
  - name: bob
    authorized:
      - /tmp/bob/id_rsa.pub
```

You can write tasks:

```
- user: name={{ item.name }} state=present generate_ssh_key=yes
  with_items: "{{users}}"

- authorized_key: "user={{ item.0.name }} key='{{ lookup('file', item.1) }}'"
  with_subelements:
     - users
     - authorized
```

# Playbook

## Playbook directory

**Default** current dir

## Playbook

**Syntax** A YAML file defining a list of *Play* s and *Playbook include* s:

```
    - <play>
    - <play>
    - include: <path to playbook>
    - include: <path to playbook>
```

**Templating** A playbook is rendered as a Jinja2 template (doc) before processing it, but playbooks should not use loops and conditionals.

## Playbook include

A playbook can include other playbooks:

```
- include: <path to playbook>
```

Note that, unlike *Task include* s, playbook includes cannot set variables.

## Play

Complete list of possible keys

A dictionary:

```
hosts:  hosta:pattern1:pattern2   # required
vars:
  var1: value1
  var2: value2
roles:
  - <rolename1>
  - {role: <rolename2>, var1: value1, tags: ['tag1', 'tag2']}
tags:
  - <tag1>
  - <tag2>
remote_user: username
sudo: yes|no
sudo_user: username
tasks:
  - <task>
  - include: <taskfile>
  - include: <taskfile2>
    tags: [tag1, tag2]
  - <task>
handlers:
  - <task>
  - include: <taskfile>
```

```
  - <task>
notify:
  - <handler name>
  - <handler name>
vars_files:
  - <path to external variables file>
  - [<path1>, <path2>, ...]   (ansible loads the first one found)
  - <path to external variables file>
strategy: linear|free
serial: <number>|"<number>%"
```

Required keys:

**hosts** A string, containing one or more *Host pattern* s separated by colons

Optional keys:

**handlers** list of *Handler* s and *Task include* s.

**pre_tasks** list of *Task* s and *Task include* s. These are executed before roles.

**roles** list of names of *Role* s to include in the play. You can add parameters, tags, and conditionals:

```
    roles:
      - common
      - { role: foo_app_instance, dir: '/opt/a', tags: ["bar", "baz"] }
      - { role: foo_app_instance, dir: '/opt/b', when: "ansible_os_family == 'RedHat'" }
```

**serial** Set how many hosts at a time to run at a time. The default is to run tasks on all of a play's machines at once. See also *strategy*.

**strategy** How plays are run on multiple hosts. The default is "linear", where each task is run on up to *serial* hosts in parallel, and then Ansible waits for them all to complete before starting the next task on all the hosts.

"free" lets each host run independently, starting its next task as soon as it finishes the previous one, regardless of how far other hosts have gotten.

**tags** see *Tags*.

**tasks** list of *Task* s and *Task include* s. These are executed after the *roles*.

**post_tasks** list of *Task* s and *Task include* s. These are executed after the *tasks*.

**notify** list of names of *Handler* s to trigger when done, but only if something changed

**vars** A dictionary defining additional *Variables*

**remote_user** user to login as remotely

**sudo** yes|no

**sudo_user** user to sudo to remotely

## Running a playbook

ansible-playbook <filepath of playbook> [options]

**ansible-playbook playbook.yml –start-at="install packages"** The above will start executing your playbook at a task named "install packages".

**ansible-playbook playbook.yml –step** This will cause ansible to stop on each task, and ask if it should execute that task.

# Roles

## Role

A role is a directory with specified contents. The role directory must be in one of the directories on the *roles_path* and its name is used to refer to the role elsewhere.

Complete list of possible keywords

Inside the role's top-level directory, you might see a tree like this (most of this is optional).

**defaults/main.yml** variables within will be defined at the lowest priority (can be overridden by variables declared anywhere else, even inventory variables)

**files/** Any copy tasks can reference files in roles/x/files/ without having to path them relatively or absolutely

Any script tasks can reference scripts in roles/x/files/ without having to path them relatively or absolutely

**handlers/main.yml** handlers listed therein will be added to the play

**library/** modules here (directories) will be available in the role, and any roles called after this role

**meta/main.yml** *Role dependencies file*

**tasks/main.yml** *Tasks file*

Any include tasks can reference files in roles/x/tasks/ without having to path them relatively or absolutely

**templates/** Any template tasks can reference files in roles/x/templates/ without having to path them relatively or absolutely

**vars/main.yml** variables listed therein will be added to the play. These override almost any other variables except those on the command line, so this is really better for the role's "constants" than variables :-)

## Role dependencies file

**Syntax** YAML file

**Templating** Jinja2

**Contents** A dictionary

The role dependencies file defines what other roles this role depends on.

Keys:

**dependencies** A list of *Dependency dictionary* s

**allow-duplicates** yes|no

Defaults to no, preventing the same role from being listed as a dependency more than once. Set to yes if you want to list the same role with different variables.

Example:

```
---
dependencies:
  - role: role1
  - role: role2
    varname: value
```

### Dependency dictionary

Required keys:

**role**

name of role, or quoted path to role file, or quoted repo URL:

role: postgres

role: '/path/to/common/roles/foo'

role: 'git+http://git.example.com/repos/role-foo,v1.1,foo'

role: '/path/to/tar/file.tgz,,friendly-name'

Optional keys: any parameters for the role - these define *Variables*

### Embedding modules in roles

## Secrets

Ansible handles secrets using a feature called Vault.

Vault lets you encrypt any of your `.yml` files, but typically you'd apply it to files containing variable definitions, then use the variables' values as needed elsewhere.

Vault provides subcommands that let you encrypt a file in place, decrypt a file in place, edit a file that's encrypted in one step, etc.

When ansible is running your playbook or whatever, any time it comes across a .yml file that appears to be encrypted, it will decrypt it (in memory) and use the decrypted contents, fairly transparently. You can have as many of your files encrypted as you want.

However, all the encrypted files have to use the same password.

### Providing the password to Ansible

1. Have Ansible prompt for it by passing `--ask-vault-pass`. Most secure, but inconvenient.
2. Put it plaintext in a well-protected file, and pass `--vault-password-file <filename>`. Most insecure, but more convenient than the prompt.
3. Write a script or program that outputs the password on stdout, mark it executable, and pass that: `--vault-password-file <path-to-program>`. This makes it possible to use a local system keychain or something, which might be more secure than the other options. Or worse...

### Ways to use it

One approach I've used is to have a single encrypted `secrets.yml` file in my base directory containing all my secret variables, and another file with very restrictive permissions (and outside of source control) containing my password, then add these arguments when running ansible:

```
--extra-vars @secrets.yml --vault-password-file path/to/passfile
```

The advantage of that is that if I don't need the secrets, I can leave all that off and Ansible will run fine. (As opposed to having the encrypted file automatically read by Ansible every time.)

I'm not sure if that will scale, though.

### Limitations

- This is symmetric encryption. In other words, anyone with the password can encrypt and decrypt a file.
- All the encrypted files must be encrypted using the same password.
- That means you have to protect the decrypting password (the only password) very carefully, and makes providing it to Ansible awkward.

Links:

- Vault
- Not logging secrets
- How to upload encrypted file using ansible vault?
- Managing Secrets with Ansible Vault – The Missing Guide (Part 1 of 2)
- Managing Secrets with Ansible Vault – The Missing Guide (Part 2 of 2)

## synchronize

The Ansible synchronize module gets its own page because it is a bitch.

(Update: apparently some of these bad behaviors were bugs in Ansible 2.0.0.x, but I'm keeping this page around for history.)

Let me count the ways:

- By default, it tries to become *locally* the user you've specified using the `become_user` variable that you have said you want to become *remotely*. [Apparently that was a bug in 2.0.0.x and works correctly in 1.9.x and 2.0.1+.]
- Then it does *not* try to *remotely* become the user you've specified; you have to hack it by setting `rsync_path: "sudo rsync"`. [I have not tried this again with 2.0.1+.]
- Unlike every other Ansible module, the `owner` and `group` options are *booleans*, not the names or numbers of users and groups. If true, it'll try to copy the owner of the local files, but if you want to specify the ownership of the target files yourself, you'll have to fix it afterward.

Here's a working example:

```
- name: sync source from local directory
  synchronize:
    dest: "{{ source_dir }}"
    src: "{{ local_project_dir }}/"
    delete: yes
    rsync_path: "sudo rsync"  # Use sudo on the remote system
    recursive: true
    rsync_opts:
      - "--exclude=.git"
      - "--exclude=*.pyc"
  become: no  # stops synchronize trying to sudo locally
```

NOTE: Ansible 2.0.1 fixed numerous bugs in synchronize:

- Fixes a major compatibility break in the synchronize module shipped with 2.0.0.x. That version of synchronize ran sudo on the controller prior to running rsync. In 1.9.x and previous, sudo was run on the host that rsync connected to. 2.0.1 restores the 1.9.x behaviour.
- Additionally, several other problems with where synchronize chose to run when combined with delegate_to were fixed. In particular, if a playbook targetted localhost and then delegated_to a remote host the prior behavior (in 1.9.x and 2.0.0.x) was to copy files between the src and destination directories on the delegated host. This has now been fixed to copy between localhost and the delegated host.
- Fix a regression where synchronize was unable to deal with unicode paths.
- Fix a regression where synchronize deals with inventory hosts that use localhost but with an alternate port.

# Tags

When you apply tags to things, you can then control whether they're executed by adding command line options.

## How to tag things

Plays and tasks have optional `tags` attributes where you can specify a list of tags. Here are some tagged *Task* s:

```
tasks:
    - module: parm1=a parm2=b
      tags:
         - packages

    - module2: parm1=x parm2=y
      tags:
         - configuration
```

And here's a playbook with some tagged *Play* s:

```
- hosts: all
  tags:
    - foo
    - bar
  roles:
    - role1
    - role2
```

You can also apply tags when invoking a role from a playbook:

```
roles:
  - { role: webserver, port: 5000, tags: [ 'web', 'foo' ] }
```

and when including tasks:

```
- include: foo.yml
  tags: [web,foo]
```

## What tags do

Adding a tag to a play or task says that *if* ansible is invoked with `--tags=x,y,z`, that the tagged play or task will only be executed if at least one of its tags is included in the list of tags from the command line.

Specifying `--tags=all` is equivalent to the default behavior, where all playbooks and tasks are run regardless of their tags.

Specifying `--tags=tagged` runs only things that have *some* tag, while `-tags=untagged` runs only things that have *no* tag.

You could alternatively invoke ansible with `--skip-tags=a,b,c` and it will execute all plays and tasks that are *not* tagged with a, b, or c.

Presumably `--skip-tags=tagged` does the opposite of `--tags=tagged`, and `--skip-tags=untagged` does the opposite of `--tags=untagged`.

If a play or task is tagged `always`, then it will be executed *unless* ansible is invoked with `skip-tags=always`.

# Task

## Tasks file

**Syntax** YAML FILE

**Templating** Jinja2

**Content** A list of task definitions, task includes, and *Blocks*.

## Task include

Anywhere there can be a task definition, you can also use a task include:

```
- include: <path to tasks file> [options]
```

The path is relative to the *Playbook directory*, or the file is also searched for in the tasks directory of a role.

`[options]` is an optional list of additional variable settings, e.g.:

```
- include: tasks/footasks.yml vara=1 varb=2 varc=3
```

You can use an expanded syntax with a vars setting to set more complicated values:

```
- include: wordpress.yml
  vars:
      wp_user: timmy
      some_list_variable:
        - alpha
        - beta
        - gamma
```

Or use this more compact but apparently equivalent syntax:

```
- { include: wordpress.yml, wp_user: timmy, ssh_keys: [ 'keys/one.txt', 'keys/two.txt' ] }
```

## Task

doc, complete list of possible keywords

A dictionary:

```
name: string      # optional but highly recommended
module: args      # required; the "action"
environment: dictionary
remote_user: username
sudo: yes|no
sudo_user: username
otheroption: othervalue   # depending on module
tags:
 - <tag1>
 - <tag2>
```

Required keys:

**name** text

***modulename*** options

Optional keys that can be used on any task:

**environment** dictionary (in YAML, or variable containing dictionary)

**ignore_errors** if true, continue even if task fails

**register <varname>** store result of task in <varname>. See also `when` for some ways to use.

**remote_user** user to login as remotely

**sudo** yes|no

**sudo_user** user to sudo to remotely

**tags** list of tags to associate with the task

**when** expression controls whether task is executed doc:

```
    when: <varname>
    when: not <varname>
```

Special filters for checking result of a prior task:

```
    when: <varname>|failed
    when: <varname>|skipped
    when: <varname>|success
```

Additional keys might be required and optional depending on the module being used.

### Handler

Same syntax as a *Task*, it just gets triggered under different circumstances.

## Variables

### Variables

Some variables alter the behavior of ansible (see http://docs.ansible.com/intro_inventory.html#list-of-behavioral-inventory-parameters for a list). You can set some of these using environment variables (doc).

CORRECTION: Use `ansible_ssh_user`, not `ansible_user`.

Any of them can be used anywhere Jinja2 templating is in effect.

Places to define variables:

- inventory
- playbook
- included files and roles
- local facts
- ansible command line (`--extra-vars "foo=1 bar=2"` or `--extra-vars @filepath.json` or `--extra-vars @filepath.yml`)

And here's the precedence order:

- extra vars (-e in the command line) always win
- **then comes connection variables defined in inventory (ansible_ssh_user, etc)**
  - Do NOT put things like ansible_sudo=[yes|no] here because it'll override the values set in plays, tasks, etc. which need to be able to control it themselves
- then comes "most everything else" (command line switches, vars in play, included vars, role vars, etc)
- then comes the rest of the variables defined in inventory
- then comes facts discovered about a system
- then "role defaults", which are the most "defaulty" and lose in priority to everything.

There are also three scopes (doc) but I don't know how these relate to precedence:

- Global: this is set by config, environment variables and the command line
- Play: each play and contained structures, vars entries, include_vars, role defaults and vars.
- Host: variables directly associated to a host, like inventory, facts or registered task outputs

## Variables file

A variables file (doc) is a file that defines values of *Variables*.

**Syntax** YAML defining a single dictionary

**Templating** The file does not appear to undergo template expansion, but the values of variables do??

## Variables files

Ansible will look in *Inventory directory* and *Playbook directory* for directories named `host_vars` or `group_vars`. Inside those directories, you can put a single *Variables file* with the same name as a host or group (respectively) and Ansible will use those *Variables* definitions.

Or a file named `all` that will be used for all hosts or groups.

Or you can create a directory with the same name as a host or group and Ansible will use all the files in that directory as *Variables file* s.

You can also include vars files from a *Play* (doc).

### Facts

Ansible automatically defines a whole bunch of variables with information about the system that it's running on (the system the plays and tasks are running on, not the system you're controlling ansible from).

You can add to the facts with config files called local facts (doc) though I don't know how that's any better than putting variables in all the other places you can set them...

To see a list of all of the facts that are available about a machine, you can run the "setup" module as an ad-hoc action:

```
ansible -m setup hostname
```

This will print out a dictionary of all of the facts that are available for that particular host. And here's an example.

## Ansible Galaxy

### Links

- doc
- popular roles and recent activity
- search

### Role specification

Format when installing roles from galaxy:

- `username.rolename[,version]`
- `scm+repo_url[,version]`
- `tarball_url`

Versions represent tags in the role's source repository.

E.g.:

```
user2.role2
user1.role1,v1.0.0
user1.role2,master
git+http://bitbucket.org/willthames/git-ansible-galaxy
https://some.webserver.example.com/files/master.tar.gz
```

### Ways of installing

#### Command-line

List roles on the command line:

```
ansible-galaxy install user2.role2 user1.role1,v1.0.9
```

### Simple file

List roles in a file, one per line. Example file:

```
# file: roles.txt
user2.role2
user1.role1,v1.0.0
```

And install with `-r`:

```
ansible-galaxy install -r roles.txt
```

### YAML file

Use a YAML file to provide more control. The YAML file should contain a list of dictionaries. Each dictionary specifies a role to install. Keys can include:

**src** (*required*) a role specification as above. (Since there's a separate dictionary key for `version`, I don't know whether you can include version here, or if you're required to list it separately as `version`.)

**path** Where to install (directory, can be relative)

**version** version to install. e.g. `master` or `v1.4`.

**name** install as a different name

**scm** default `git` but could say `hg` and then in `src` provide a URL to a mercurial repository.

Example:

```
# install_roles.yml

# from galaxy
- src: yatesr.timezone

# from github
- src: https://github.com/bennojoy/nginx

# from github installing to a relative path
- src: https://github.com/bennojoy/nginx
  path: vagrant/roles/

# from github, overriding the name and specifying a specific tag
- src: https://github.com/bennojoy/nginx
  version: master
  name: nginx_role

# from a webserver, where the role is packaged in a tar.gz
- src: https://some.webserver.example.com/files/master.tar.gz
  name: http-role

# from bitbucket, if bitbucket happens to be operational right now :)
- src: git+http://bitbucket.org/willthames/git-ansible-galaxy
  version: v1.4

# from bitbucket, alternative syntax and caveats
- src: http://bitbucket.org/willthames/hg-ansible-galaxy
  scm: hg
```

And again install with `-r`:

```
ansible-galaxy install -r install_roles.yml
```

Misc. stuff I need to file somewhere:

## Ad-hoc command

ansible *Host pattern* -m <module> [options]

e.g.

$ ansible all -m ping –ask-pass

Shortcut to run a command:

$ ansible all -a "/bin/echo hello"

options: see output of "ansible –help" for now

See doc for ad-hoc commands.

## Host pattern

See doc for host patterns.

<hosts>:

"all" = all hosts in inventory file

CHAPTER 2

# AWS

Contents:

## S3

### Access control

- How S3 evaluates access control
- Guidelines for Using the Available Access Policy Options

“The only recommended use case for the bucket ACL is to grant write permission to the Amazon S3 Log Delivery group”...

“In general, you can use either a user policy or a bucket policy to manage permissions.”

Here’s a bucket policy to grant some IAM user complete access to a bucket:

```
{
    "Statement": [
        {
          "Sid":"PublicReadForGetBucketObjects",
          "Effect":"Allow",
          "Principal": {
                "AWS": "*"
             },
          "Action":["s3:GetObject"],
          "Resource":["arn:aws:s3:::BUCKET-NAME/*"
          ]
        },
        {
            "Action": "s3:*",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::BUCKET-NAME",
                "arn:aws:s3:::BUCKET-NAME/*"
            ],
            "Principal": {
                "AWS": [
                    "USER-ARN"
                ]
            }
        }
```

```
    ]
}
```

What about read-only access? Let's see...

seems like s3auth.com used this example:

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject", "s3:GetBucketWebsite"],
      "Resource": [
        "arn:aws:s3:::bucket-1.example.com/*",
        "arn:aws:s3:::bucket-2.example.com/*"
      ]
    }
  ]
}
```

## Updating metadata to improve response headers for caching

Install s3cmd, then do it like this:

```
s3cmd --recursive modify \
    --add-header="Expires: Thu, 31 Dec 2099 20:00:00 GMT" \
    --add-header="Cache-Control: max-age=94608000" \
    s3://caktus-website-production-2015/media/community_logos
```

You can use `s3cmd ls` to get a list of the buckets you can access.

CHAPTER 3

# Bootstrap

**Warning:** THIS IS NOT DONE AND PROBABLY WRONG.

The grid.

SO what does a class *col-SIZE-N* mean?

Each SIZE has a BREAKPOINT:

xs: -1 sm: 750px md: 970px lg: 1170px

Call the window width WIDTH.

For a single class col-SIZE-N:

```
if WIDTH >= BREAKPOINT(SIZE), then
    ELEMENT-WIDTH=WIDTH*N/12
    display INLINE (same line as previous element if possible)
else
    ELEMENT-WIDTH=100%
    display BLOCK (element gets its own line)
```

What if we have col-SIZE1-N1 and col-SIZE2-N2, with BREAKPOINT(SIZE1) < BREAKPOINT(SIZE2)?:

```
IF WIDTH >= BREAKPOINT(SIZE2), then
   ELEMENT_WIDTH = WIDTH * N2 / 12
   INLINE
elif WIDTH >= BREAKPOINT(SIZE1), then
   ELEMENT_WIDTH = WIDTH * N1 / 12
   INLINE
else:
   BLOCK display
```

and so forth - just look at the class with the largest size

NOTE: Since all widths are >= the breakpoint of XS, then if XS is present, the element will ALWAYS be laid out inline. Though col-xs-12 is pretty much equivalent to not having an XS class, right???????????/

CHAPTER 4

# Debian

## Services

`update-rc.d`:

- Make a service run in its default runlevels:

```
update-rc.d <service> defaults
```

  or:

```
update-rc.d <service> enable
```

- Make a service not run in any runlevel:

```
update-rc.d <service> disable
```

Making a new init script:

- Read `/etc/init.d/README`, which will point to other docs
- Copy `/etc/init.d/skeleton` and edit it.

## Packages

- List packages that match a pattern: `dpkg -l <pattern>`
- List contents of a package: `dpkg -L packagename`
- Show packages that installed files matching pattern: `dpkg -S pattern`
- Show info about an installed package: `dpkg-query -s packagename`
- show info about a package that is known: `apt-cache showpkg packagename`
- Reconfigure a package: `dpkg-reconfigure packagename`
- Change alternatives: `update-alternatives ...`

## Alternatives

Change 'alternatives' default browser or editor:

```
sudo update-alternatives --set x-www-browser /usr/bin/chromium-browser
sudo update-alternatives --set editor /usr/bin/emacs24
```

Be prompted for which alternative you prefer for a link group:

```
sudo update-alternatives --config x-www-browser
```

Find out what the top-level link groups are:

```
sudo update-alternatives --get-selections
```

Set xdg program to open/browse a directory (DOES NOT WORK) (do NOT use sudo):

```
xdg-mime default /usr/share/applications/Thunar.desktop x-directory/normal
```

Change 'xdg' default browser (for user):

```
xdg-settings get default-web-browser
xdg-settings set default-web-browser google-chrome.desktop
xdg-settings set default-web-browser firefox.desktop
```

Install without any prompts (http://askubuntu.com/questions/146921/how-do-i-apt-get-y-dist-upgrade-without-a-grub-config-prompt):

```
sudo DEBIAN_FRONTEND=noninteractive apt-get -y \
-o Dpkg::Options::="--force-confdef" -o Dpkg::Options::="--force-confold" \
<COMMAND>
```

## Desktop applications

Put your own .desktop files in ~/.local/share/applications.

Archlinux on desktop entries

Desktop file spec

To let the system know about new or changed desktop files:

```
update-desktop-database [directory]
```

Launch the application from command line that has a <name>.desktop file somewhere:

```
gtk-launch <name>
```

# Diet

## Foods with low glycemic index

- Breads
- Dense wholegrain breads
- White corn tortillas
- Grain and seed breads
- Fruit Loaf such as Raisin
- Multigrain breads (look for breads where you can see lots of grains)
- Authentic Sourdough bread
- Breakfast Cereals
- Traditional porridge oats
- Muesli*
- Bircher Muesli
- Wholegrain high fibre cereals
- Vegetables
- Sweetcorn
- Silverbeet
- Carrots
- Zucchini
- Peas, frozen or fresh
- Snowpeas
- CarismaTM Potatoes*
- Green Beans
- Broccoli
- Eggplant
- Cauliflower

- Squash
- Capsicum
- Salad Vegetables
- Celery
- Leeks
- Tomatoes
- Mushrooms – very low carb or no GI rating
- Butternut Pumpkin (lower GI)
- Avocadoes
- Drinks
- Milo®
- Skim Latte
- Sustagen®
- Soy Drinks
- Fruit Smoothies
- Fruit Juice
- Snacks
- Grain & Fruit bars
- Wholegrain crackers
- Nut & Seed bars
- Dried fruit and nuts
- Legumes
- Split Peas; Green or red Lentils
- Baked Beans
- Canned & Dried beans – kidney, cannellini, butter, borlotti, chickpeas
- Spreads
- Fruit Spreads
- Hummus
- Nut butters
- Main Meal Carbs
- Doongara Low GI White rice
- Fresh Noodles – Hokkein, Udon, Rice
- Low GI Brown rice*
- Soba Noodles
- Basmati rice (lower GI)
- Buckwheat

- Pasta, cooked al dente*
- Vermicelli
- Pearl Couscous*
- Bulgur
- Quinoa*
- Semolina
- Pearl Barley
- Cracked Wheat
- Fruit
- Apples*
- Pears*
- Bananas
- Kiwi Fruit
- Grapes*
- Mango
- Strawberries
- Oranges
- Peaches
- Grapefruits
- Apricots
- Berries, fresh or frozen
- Plums
- Dried fruits such as prunes, raisins, sultanas, apricots
- Canned Fruit in natural juice
- Dairy Foods
- Reduced fat milk
- Reduced fat custard
- Reduced fat yoghurt, plain or fruit flavoured
- Low fat ice-cream*

## For lowering triglycerides

- Decrease or eliminate:
  - Sweets
  - Alcohol
  - Refined carbohydrates:

        - White rice
        - bread and pasta made from white flour or semolina
        - Saturated fats and fried foods:
        - high fat meats
        - skin on poultry
        - sauces and spreads
        - Trans fatty acids and hidden fats:
        - hydrogenated vegetable oil
        - regular fat meats
        - lunchmeats
        - hot dogs
        - fatty snack foods
- Eat more:
    - omega 3 fatty acids:
        - fatty fish
        - salmon
        - mackerel
        - sardines
        - tuna
        - trout
        - ground flax seed
        - flaxseed oil
        - soy products
        - legumes
        - walnuts
        - dark leafy green vegetables
    - high fiber foods:
    - beans
    - whole grains
    - ground flaxseed
    - pumpkin seeds
    - rice bran
    - oat bran
    - fruits and vegetables
    - Eat more plant foods: Vegetable proteins such as
    - dried beans,

- peas, and
- soy products;
- White poultry, prepared without the skin, is also a good source of protein without a lot of fat content.

CHAPTER 6

# Django

These are just things I always find myself looking up, so I try to make some notes of the most important parts that I can refer back to.

Contents:

## Admin

### URLs

List {{ app_label }}_{{ model_name }}_changelist Change {{ app_label }}_{{ model_name }}_change object_id

https://docs.djangoproject.com/en/stable/ref/contrib/admin/#reversing-admin-urls

### Customize top-right corner of admin pages

Create your own *templates/admin/base_site.html* that comes ahead of the admin's default one in the templates path.

At least in Django 1.8+, this gives you a "View site" link for free:

> % extends "admin/base.html" %}
>
> {% block title %}{{ title }} | {{ site_title|default:_('Django site admin') }}{% endblock %}
>
> {% block branding %} <h1 id="site-name"><a href="{% url 'admin:index' %}">{{ site_header|default:_('Django administration') }}</a></h1> {% endblock %}
>
> **{% block userlinks %}** <a href="{% url "clear-cache" %}">Clear cache</a> / {{ block.super }}
>
> {% endblock userlinks %}

Prior to Django 1.8:

> {% extends "admin/base.html" %}
>
> {% block title %}{{ title }} | Caktus Admin{% endblock %}
>
> {% block branding %}<h1 id="site-name">Caktus Admin</h1>{% endblock %}
>
> **{% block nav-global %}**
>
> > ** ** <a href='{% url "home" %}'>Return to Caktus Home</a> | <a href='{% url "clear-cache" %}'>Clear cache</a>
> >
> > 

{% endblock %}

## Applications

https://docs.djangoproject.com/en/stable/ref/applications/#django.apps.AppConfig

In `__init__.py`:

```
# programs/__init__.py

default_app_config = 'programs.apps.ProgramsConfig'
```

In `apps.py`:

```
# programs/apps.py

from django.apps import AppConfig

class ProgramsConfig(AppConfig):
    name = 'programs'  # required: must be the Full dotted path to the app
    label = 'programs'  # optional: app label, must be unique in Django project
    verbose_name = "Rock 'n' roll"  # optional

    def ready():
        """
        This runs after all models have been loaded, but you may not
        modify the database in here.

        Here's a trick to run something after each migration, which is often
        good enough.
        """
        from django.db.models.signals import post_migrate

        post_migrate.connect(`callable`)
```

## Celery

(Yes, I know Celery isn't Django-specific.)

http://docs.celeryproject.org/en/latest/

### Useful settings

http://docs.celeryproject.org/en/latest/configuration.html

CELERY_ALWAYS_EAGER: If this is True, all tasks will be executed locally by blocking until the task returns. apply_async() and Task.delay() will return an EagerResult instance, which emulates the API and behavior of AsyncResult, except the result is already evaluated.

That is, tasks will be executed locally instead of being sent to the queue.

CELERY_EAGER_PROPAGATES_EXCEPTIONS: If this is True, eagerly executed tasks (applied by task.apply(), or when the CELERY_ALWAYS_EAGER setting is enabled), will propagate exceptions.

It's the same as always running apply() with throw=True.

CELERY_IGNORE_RESULT: Whether to store the task return values or not (tombstones). If you still want to store errors, just not successful return values, you can set CELERY_STORE_ERRORS_EVEN_IF_IGNORED.

*CELERYD_HIJACK_ROOT_LOGGER:* By default any previously configured handlers on the root logger will be removed. If you want to customize your own logging handlers, then you can disable this behavior by setting CELERYD_HIJACK_ROOT_LOGGER = False.

CELERYBEAT_SCHEDULE: In each task, you can add an 'options' dictionary and set 'expires' to a number of seconds. If the task doesn't run within that time, it'll be discarded rather than run when it finally gets to a worker. This can help a lot with periodic tasks when workers or the queue gets hung up for a while and then unjammed - without this, the workers will have to work through a huge backlog of the same periodic tasks over and over, for no reason.

Example:

```
CELERYBEAT_SCHEDULE = {
    'process_new_scans': {
        'task': 'tasks.process_new_scans',
        'schedule': timedelta(minutes=15),
        'options': {
            'expires': 10*60,  # 10 minutes
        }
    },
}
```

CELERY_DEFAULT_QUEUE: In the absence of more complicated configuration, celery will use this queue name for everything. Handy when multiple instances of a site are sharing a queue manager:

```
CELERY_DEFAULT_QUEUE = 'queue_%s' % INSTANCE
```

# django-compressor

django-compressor docs

Warning: much of the documentation is casual about saying things that are only true in some scenarios, without making clear that that's the case.

## ACTUALLY USING

Here are some practical scenarios for using django-compressor.

For what to put in your templates, you can go by the django-compressor documentation, and be sure to use {% static %} and not STATIC_URL.

For what to put in your settings... it's a lot more complicated. Set the compressor filters and precompilers however you want. For the rest, keep reading.

### Scenario: Development using runserver, DEBUG, not offline

If DEBUG is True, then compressor won't even do anything and so everything should just work.

### Scenario: Running using local files, not offline

This is the typical small server situation. You unpack your project on the server, run collectstatic, point nginx or some other server at STATIC_ROOT and go.

Example settings:

```
# Django settings
DEBUG = False
STATIC_ROOT = '/var/www/static/'
STATIC_URL = '/static/'
# set compressor filters and precompilers as desired.
# leave other compressor settings at defaults.
```

```
# nginx settings
location /static {
   alias /var/www/static;
}
```

### Scenario: running using local files, with offline

Like the previous scenario, but you want compressor to do all its work at deploy time so the results are cached and ready to go immediately when you start your server.

```
# Django settings like before, plus:
COMPRESS_OFFLINE = True
```

Now at deploy time you have more steps:

```
$ python manage.py collectstatic
$ python manage.py compress
```

Run `compress` *after* collectstatic so that compressor can find its input files. It'll write its output files under `{STATIC_ROOT}/CACHE`, and get them from there at runtime.

### Scenario: running with storage on the network, with offline

In this scenario, you're putting your static files somewhere off of the server where you're running Django. For example, S3. Or just your own static file server somewhere. Whatever.

Let's start with how this would be setup without django-compressor, then we can modify it to add django-compressor.

```
# settings/no_compressor.py
STATIC_ROOT = None  # Unused
STATIC_URL = None  # Unused
STATIC_FILE_STORAGE = 'package.of.FileStorageClass'
```

At deploy time you can just run collectstatic, and all your static files will be pushed to the network:

```
$ python manage.py collectstatic
```

And at runtime, `{% static %}` will ask your file storage class to come up with a URL for each file, which will turn out to be on your other server, or S3, or whatever.

Now, suppose we want to add compressor with offline processing (not using offline makes no sense with network storage). Here are the settings you can use at runtime for that, assuming things have been prepared correctly:

```
# settings/deployed.py
# Django settings we'll use in production
STATIC_ROOT = None  # Unused
STATIC_URL = None  # Unused
STATIC_FILE_STORAGE = 'path.to.network.filestorage'
```

```
COMPRESS_ENABLED = True
COMPRESS_OFFLINE = True
```

The preparation is the tricky part. It turns out that for compress to work, a copy of the static files must be gathered in a local directory first. Most of the tools we might use to compile, compress, etc. are going to read local files and write local output.

To gather the static files into a local directory, we might, for example, use a different settings file that uses the default file storage class, and run collectstatic. E.g.:

```
# settings/gather.py
# Django settings when first running collectstatic
from .deployed import *

# Override a few settings to make storage local
STATIC_ROOT = '/path/to/tmp/dir'
STATIC_URL = None  # Unused
STATIC_FILE_STORAGE = 'django.core.files.storage.FileSystemStorage'
```

```
$ python manage.py collectstatic --settings=settings.gather
```

After running `collectstatic` with these settings, all your source static files will be gathered under '/path/to/tmp/dir'.

Now you could run `compress`, and the resulting files would be added under */path/to/tmp/dir*. There's an important *gotcha* that will cause problems, though - for compressor to match up the output it makes now with what it'll be looking for later, the contents of each `{% compress %}` tag must be identical now to what it'll be then, which means the URLs must point at the production file server. We can accomplish this by setting STATIC_URL before running the compress:

```
# settings/compress.py
# Django settings when running compress
from .deployed import *

# Override a few settings to make storage local, but URLs look remote
STATIC_ROOT = '/path/to/tmp/dir'
STATIC_URL = 'https://something.s3.somewhere/static/'  # URL prefix for runtime
STATIC_FILE_STORAGE = 'django.core.files.storage.FileSystemStorage'
```

```
$ python manage.py compress --settings=settings.compress
```

The problem now is to get all these files onto the remote server. You could just use `rsync` or `s3cmd` or something, which will work fine. But for maximum flexibility, let's figure out a way to do it using Django. Our approach will be to tell Django that our SOURCE static files are in '/path/to/tmp/dir', and we want them collected using our production file storage class, which will put them where we want them.

```
# Django settings when running collectstatic again after compress,
# to copy the resulting files to the network
# settings/copy_upstream.py
from .deployed import *  # Set up for network file storage
# Tell collectstatic to use the files we collected and compressed
STATICFILES_FINDERS = ['django.contrib.staticfiles.finders.FileSystemFinder']
STATICFILES_DIRS = ['/path/to/tmp/dir']
```

```
$ python manage.py collectstatic --settings=settings.copy_upstream
```

That should copy things to the network. Then if you run using the 'deployed' settings, things should work!

TODO: TEST THAT!!!!!!!!!!!!!!!!!!!

### Other approaches

The compressor docs suggest a different approach – hack the storage class you're using so when you run collectstatic, it saves a copy of each file into a local directory in addition to pushing it upstream. Then you can use the same storage class for collectstatic, compress, and runtime.

## More detailed notes

### Cache

For some things, compressor uses the cache named by `COMPRESS_CACHE_BACKEND`, which defaults to `None`, which gives us the default Django cache.

### Principles of compression

Whether compressor is processing templates offline ahead of time or at runtime, there are some common principles.

First, if `COMPRESS_ENABLED` is False, the `{% compress %}` tag will simply render as its contents; compressor won't change anything.

Otherwise, compressor will

1. parse the contents of the tag and figure out which css and javascript files would be included
2. fetch those files (See "accessing the files to be compressed")
3. run those files through any configured preprocessors
4. concatenate the result and save it using COMPRESS_STORAGE
5. at rendering, the tag and contents will be replaced with one or two HTML elements that will load the compressed file instead of the original ones.

### Offline

If `COMPRESS_OFFLINE` is True, compressor expects all uses of `{% compress ... %}` in templates to have been pre-processed by running `manage.py compress` ahead of time, which puts the results in compressor's *offline* cache. If anything it needs at run-time is not found there, things break/throw errors/render wrong etc.

---

**Note:** If COMPRESS_OFFLINE is True and files have not been pre-compressed, compressor will *not* compress them at runtime. Things will break.

---

The offline cache manifest is a json file, stored using COMPRESS_STORAGE, in the subdirectory `COMPRESS_OUTPUT_DIR` (default: `CACHE`), using the filename `COMPRESS_OFFLINE_MANIFEST` (default: `manifest.json`).

The keys in the offline cache manifest are generated from *the template content inside each compress tag*, *not* the contents of the compressed files. So, you must arrange to re-run the offline compression anytime your content files might have changed, or it'll be serving up compressed files generated from the old file contents.

---

**Note:** It sounds like you must *also* be *sure* the contents of the compress tags don't change between precompressing and runtime, for example by changing the URL prefix!

---

The values in the offline cache manifest are paths of the compressed files in COMPRESS_STORAGE.

**Note:** RECOMMENDATION FROM DOCS: make `COMPRESS_OFFLINE_MANIFEST` change depending on the current code revision, so that during deploys, servers running different versions of the code will each use the manifest appropriate for the version of the code they're running. Otherwise, servers might use the wrong manifest and strange things could happen.

### Not offline

If `COMPRESS_OFFLINE` is False, compressor will look in COMPRESS_STORAGE for previously processed results, but if not found, will create them on the fly and save them to use again.

### Storage

Compressor uses a Django storage class for some of its operations, controlled by the setting `COMPRESS_STORAGE`.

The default storage class is `compressor.storage.CompressorFileStorage`, which is a subclass of the standard filesystem storage class. It uses `COMPRESS_ROOT` as the base directory in the local filesystem to store files in, and builds URLs by prefixing file paths within the storage with `COMPRESS_URL`.

If you change `COMPRESS_STORAGE`, then *ignore* anything in the docs about `COMPRESS_ROOT` and `COMPRESS_URL` as they won't apply anymore (except in a few cases... see exceptions noted as they come up, below).

### Accessing the files to be compressed

For each file to be compressed, compressor starts with the URL from the rendered original content inside the compress tag. For example, if part of the content is `<script src="http://example.com/foo.js"></script>`, then it extracts `"http://example.com/foo.js"` as the URL.

It checks that the URL starts with COMPRESS_STORAGE's `base_url`, or if accessing that fails (quite possible since `base_url` is not a standard part of the file storage class API), uses `COMPRESS_URL`.

**Note:** This is a place where compressor can use COMPRESS_URL even if it's not using its default storage.

If the URL doesn't start with that string, compressor throws a possibly misleading error, "'%s' isn't accessible via COMPRESS_URL ('%s') and can't be compressed".

Otherwise, compressor tries to come up with a local filepath to access the file, as follows:

- Try to get a local filepath from COMPRESS_STORAGE using `.path()`.
- If that's not implemented (for example, for remote storages), it tries again using `compressor.storage.CompressorFileStorage` (regardless of what COMPRESS_STORAGE is set to), so basically it's going to look for it under COMPRESS_ROOT.
- If it still can't get a local filepath, throws an error: "'%s' could not be found in the COMPRESS_ROOT '%s'%s" which is very misleading if you're not using a storage class that looks at COMPRESS_ROOT.

# Data fixtures

Export/dump data to use as a fixture:

```
python manage.py dumpdata --format=yaml --natural app.model >data.yaml
```

Load it again:

```
python manage.py loaddata data.yaml
```

## Natural keys

https://docs.djangoproject.com/en/stable/topics/serialization/#natural-keys

```
from django.db import models

class PersonManager(models.Manager):
    def get_by_natural_key(self, first_name, last_name):
        return self.get(first_name=first_name, last_name=last_name)

class Person(models.Model):
    objects = PersonManager()
    ...

    def natural_key(self):
        return (self.first_name, self.last_name)

    class Meta:
        unique_together = (('first_name', 'last_name'),)
```

### Dependencies

If part of the natural key is a reference to another model, then that model needs to be deserialized first:

```
class Book(models.Model):
    name = models.CharField(max_length=100)
    author = models.ForeignKey(Person)

    def natural_key(self):
        return (self.name,) + self.author.natural_key()
    natural_key.dependencies = ['example_app.person']
```

# Databases

## Performance

From Django 1.6 on, always add *CONN_MAX_AGE* to database settings to enable persistent connections. *300* is a good starting value (5 minutes). *None* will keep them open indefinitely.

BUT - keep in mind that every open connection to Postgres consumes database server resources. So you might want *instead* to run pgbouncer locally.

# Django Debug Toolbar

Install/config

Install:

```
pip install django-debug-toolbar
```

settings.py:

```
DEBUG = True
INTERNAL_IPS = ['127.0.0.1']
INSTALLED_APPS += [
   'debug_toolbar',
]
# The order of MIDDLEWARE and MIDDLEWARE_CLASSES is important. You should include
# the Debug Toolbar middleware as early as possible in the list. However, it must
# come after any other middleware that encodes the response's content, such as
# GZipMiddleware.
MIDDLEWARE = [
    'debug_toolbar.middleware.DebugToolbarMiddleware',
] + MIDDLEWARE
```

urls.py:

```
from django.conf import settings
from django.conf.urls import include, url

if settings.DEBUG:
    import debug_toolbar
    urlpatterns += [
        url(r'^__debug__/', include(debug_toolbar.urls)),
    ]
```

# Dokku

Readying a Django project for deploying to Dokku.

This lists the things to add or change to easily deploy a Django application to Dokku.

It started out not trying to cover all of setting up a site on Dokku, only the parts relevant to a Django project – but it has grown. Still, you should read the Dokku getting started docs, then use this as a cheatsheet to quickly enable existing Django projects to deploy on Dokku.

Start with the pages in this list, in order, then come back to this page and continue reading:

## Dokku server administration

This page has information for those who have to set up and maintain a Dokku server. If you're just using one, you can ignore this.

### Initial install

The Dokku docs recommend setting up a new OS install of a supported operating system, then running the Dokku install script.

Experience suggests that that approach is more likely to work than trying to install Dokku on a system that has already had some configuration done for other things.

### Simple hostnames

The simple way to set up hostnames is:

- Pick a hostname you can control, e.g. `dokku.me`.
- During initial setup of Dokku, configure that as the server's name.
- Create a DNS A record pointing `dokku.me` at the server's IP address.
- Add a wildcard entry for `*.dokku.me` at the same address.
- For each app you put on that server, give the app the same name you want to use for its subdomain. For example, an app named `foo` would be accessible on the internet at `foo.dokku.me`, without having to make any more changes to your DNS settings.

### Managing users

In other words, who can mess with the apps on a dokku server?

The way this currently works is that everybody ends up sshing to the server as the `dokku` user to do things. To let them do that, we want to add a public key for them to the dokku config, by doing this (from any system):

```
$ cat /path/to/ssh_keyfile.pub | ssh dokku ssh-keys:add <KEYNAME>
```

The <KEYNAME> is just to identify the different keys. I suggest using the person's typical username. Just remember there will not be a user of that name on the dokku server.

When it's time to revoke someone's access:

```
$ ssh dokku ssh_keys:remove <KEYNAME>
```

and now you see why the <KEYNAME> is useful.

For now, there's not a simple way to limit particular users to particular apps or commands.

## Files

Setting up files in a Django project for deploying it to Dokku

### requirements.txt

There needs to be a `requirements.txt` file at the top level. If you prefer to keep your requirements somewhere else, the top-level one can just look like:

```
-r path/to/real/requirements.txt
```

Wherever your requirements are, add the latest versions of:

```
dj-database-url
gunicorn
whitenoise
```

### settings

Add a `.../deploy.py` settings file, e.g. `<appname>/settings/deploy.py`.

It can start out looking like this (edit the top line if your main settings file isn't base.py):

```
# Settings when deployed to Dokku
from .base import *  # noqa
import dj_database_url

# Disable Django's own staticfiles handling in favour of WhiteNoise, for
# greater consistency between gunicorn and `./manage.py runserver`. See:
# http://whitenoise.evans.io/en/stable/django.html#using-whitenoise-in-development
INSTALLED_APPS.remove('django.contrib.staticfiles')
INSTALLED_APPS.extend([
    'whitenoise.runserver_nostatic',
    'django.contrib.staticfiles',
])

MIDDLEWARE.remove('django.middleware.security.SecurityMiddleware')
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'whitenoise.middleware.WhiteNoiseMiddleware',
] + MIDDLEWARE

# Update database configuration with $DATABASE_URL.
db_from_env = dj_database_url.config(conn_max_age=500)
DATABASES['default'].update(db_from_env)

# Honor the 'X-Forwarded-Proto' header for request.is_secure()
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')

# Allow all host headers (feel free to make this more specific)
ALLOWED_HOSTS = ['*']

# Simplified static file serving.
# https://warehouse.python.org/project/whitenoise/
STATICFILES_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage'
```

### wsgi.py

Find your `wsgi.py` file.

1. Edit to change the default settings module to `<appname>.settings.deploy` (the path to the new settings file you created above).
2. Add to the end:

```
from whitenoise.django import DjangoWhiteNoise
application = DjangoWhiteNoise(application)
```

### Procfile

Create Procfile (more) in the top directory. For our simple case, it can just contain one line, starting with "web: " and containing the command to start gunicorn for our site:

```
web: gunicorn {{ project_name }}.wsgi
```

See also the section on running Celery and other processes.

### runtime.txt

Create `runtime.txt` in the top directory. It only needs one line, e.g.:

```
python-3.6.1
```

This *has* to be specific. E.g. `python-3.5.2` or `python-3.6.1` might work if the dokku server supports it, but `python-3.5` or `python-3.6` probably won't.

### app.json

Create app.json in the top-level project directory. You might see examples on the Interwebs with lots of things in app.json (because Heroku uses app.json for lots of things), but as of this writing, dokku ignores everything but `scripts.dokku.predeploy` and `scripts.dokku.postdeploy`. Example:

```
{
  "scripts": {
    "dokku": {
      "predeploy": "python manage.py migrate --noinput"
    }
  }
}
```

**Note:** Dokku automatically runs `collectstatic` for you, so you don't need to do that from `app.json`.

### buildpacks

If your app is not pure Python - e.g. if it uses node - you'll need to override the automatic buildpack detection, because it only works for a single application type.

Do this by adding a top-level `.buildpacks` file, containing links to the buildpacks to use:

```
https://github.com/heroku/heroku-buildpack-nodejs.git
https://github.com/heroku/heroku-buildpack-python.git
https://github.com/heroku/heroku-buildpack-apt
```

Heroku maintains a list of buildpacks.

## Postgres with Dokku

There's nothing Django-specific about this, but I'm including it just because we probably want to do it on every single Django deploy.

To install the postgresql plugin, inside your server run (because plugins must be installed as root):

```
$ sudo dokku plugin:install https://github.com/dokku/dokku-postgres.git
```

Now you need to create a database, and link the database to the app. You can do this from your own system:

```
$ ssh dokku postgres:create example-database
$ ssh dokku postgres:link example-database django-tutorial
```

Now when dokku runs your app, it'll set an env var to tell it where its DB is, e.g.:

```
DATABASE_URL=postgres://user:pass@host:port/db
```

For Django, install the tiny `dj_database_url` package, then in settings.py:

```
import dj_database_url
db_from_env = dj_database_url.config(conn_max_age=500)
DATABASES['default'].update(db_from_env)
```

There are built-in commands making it easy to backup and restore databases:

```
$ ssh dokku postgres:export [db_name] > [db_name].dump
$ ssh dokku postgres:import [db_name] < [db_name].dump
```

## SSL for Dokku (Letsencrypt etc.)

### Letsencrypt

*Note:* Get the site up and running, and accessible from the Internet, first. Let's Encrypt will not be able to get you a certificate until then.

There's nothing Django-specific about this part, but I'm including it just because we probably want to do it on every single Django deploy.

To add SSL with the Let's Encrypt plugin (more), first install the plugin by running on the dokku server (plugins must be installed as root):

```
$ sudo dokku plugin:install https://github.com/dokku/dokku-letsencrypt.git
```

Then on your system, to configure your app and tell letsencrypt to manage its certs and renew them periodically:

```
$ ssh dokku config:set --no-restart <appname> DOKKU_LETSENCRYPT_EMAIL=your@email.tld
$ ssh dokku letsencrypt myapp
$ ssh dokku letsencrypt:cron-job --add <appname>
```

### Forcing SSL from Django

If we don't want to figure out how to override the default nginx config to redirect non-SSL requests to SSL, we can have Django do it with a few settings.

First, set SECURE_SSL_REDIRECT to True. This will tell Django to do the redirect.

```
SECURE_SSL_REDIRECT = True
```

Commit, deploy, and make sure the site still works.

Second, set SECURE_HSTS_SECONDS to a relatively small number of seconds.

```
SECURE_HSTS_SECONDS = 1800
```

This adds a header to all responses, telling any browser that receives it that this site should only be accessed via SSL, for that many seconds.

Commit, deploy, and make sure the site still works.

If everything still seems good, bite the bullet, increase SECURE_HSTS_SECONDS to a large number (e.g. 31536000 seconds, or 1 year), commit, deploy, and test again.

Additional info (move to their own files as they're ready)...

## Environment variables

I can't find docs on what environment variables Dokku sets globally when running apps. A little poking around in one of my deployed apps showed that there doesn't seem to be a whole lot beyond what the plugins are providing, but here are a few that look useful:

DOKKU_DEPLOY_BRANCH

The name of the branch that is running, e.g. `master` or `develop`.

and... that's about it?

Actually, *not even that* seems to be reliably provided.

Still, just about any Django app is going to be linked to a database, so if you need to detect whether the app is running under Dokku or Heroku, looking for DATABASE_URL should be good enough.

## Static files

We use whitenoise to serve static files from Django. If the site gets incredible amounts of traffic, throw a CDN in front, but honestly, very few sites actually need that. (If you have a philosophical objection to serving static files from Django, you can customize the nginx config through Dokku and probably manage to get nginx to do the static file serving, but I haven't bothered figuring it out myself.)

Or put your static files on S3.

## Django media

If your site requires uploaded files to be persisted, remember that the container running your code is ephemeral and any changes made to files inside it will vanish at the next deploy.

First, you can use S3 for your media files.

Or you can use persistent storage on Dokku, which is a way of mounting directories from the dokku server inside the running container where your site code can store and read files that will continue to exist past the lifetime of that particular container.

## Simple hostnames

If the server is set up properly, assuming the server's domain name is `dokkuserver.domain`, creating an app named `foo` will automatically `foo.dokkuserver.domain` resolve to the server's address, and requests with host `foo.dokkuserver.domain` to be routed to that app.

If you want to get fancier, http://dokku.viewdocs.io/dokku/configuration/domains/.

Also, note that any requests simply addressed to `dokku.me` will get routed to the alphabetically first app on the server, but you can change that: http://dokku.viewdocs.io/dokku/configuration/domains/ or just set up a "00default" app.

## Zero downtime deploys

WRITEME - see http://dokku.viewdocs.io/dokku/deployment/zero-downtime-deploys/

## Behind a load balancer

If requests are being terminated at a load balancer and then proxied to our dokku server, some nginx config customization will be needed so your app can see the actual origin of the requests: http://dokku.viewdocs.io/dokku/configuration/ssl/#running-behind-a-load-balancer

## Run a command

Suppose you want to run something like `python manage.py createsuperuser` in the app environment?

```
$ ssh dokku run <appname> python manage.py createsuperuser
```

will do it.

## Running other daemons (like Celery)

Suppose you need to run another instance of your app in another way, for example to run `celery beat` and `celery worker`.

Use the `Procfile` to tell Dokku what processes to run. E.g. if your Procfile is:

```
web: gunicorn path/to/file.wsgi
```

try editing it to:

```
web: gunicorn path/to/file.wsgi
beat: celery beat -A appname -linfo
worker: celery worker -A appname -linfo
```

With just that, the extra processes won't run automatically. You can run them by telling Dokku to scale them up, e.g.:

```
$ ssh dokku ps:scale <appname> beat=1 worker=4
```

You can check the current scaling settings:

```
$ ssh dokku ps:scale <appname>
-----> Scaling for <appname>
-----> proctype           qty
-----> --------           ---
-----> web                1
-----> beat               1
-----> worker             4
```

and see what's actually running (example from another app that only has one process):

```
$ ssh dokku ps:report <appname>
=====> <appname>
       Processes:           1
       Deployed:            true
       Running:             true
       Restore:             true
       Restart policy:      on-failure:10
       Status web.1         true       (CID: 03ea8977f37e)
```

Since we probably don't want to have to remember to manually scale these things up and check that they're running, we can add a DOKKU_SCALE file to our repo:

```
web=1
beat=1
worker=4
```

which is equivalent to running `ps:scale web=1 beat=1 worker=4`

## Secrets

First, if possible, use Dokku plugin integrations for things like databases, Redis, cache, etc. They automatically set environment variables in each app that your settings can read, so you don't have to manage different settings for each environment. See each plugin's doc, of course, for more on that.

The way to handle other secrets for each environment is to set them as config on the app, which will add them as environment variables, again so your settings can read them at runtime.

The downside is that the secrets aren't being managed/versioned etc for us... but I think we can handle the few we'll need by manually keeping them in Lastpass or something.

Here are the docs for setting config.

Before setting any config, note that by default, making any change to the config triggers a new deploy. If you're not ready for that, include `--no-restart` in the command, as these examples will do.

To set config variables:

```
$ ssh dokku config:set <appname> --no-restart VAR1=VAL1 VAR2=VAL2 ...
```

To remove a variable:

```
$ ssh dokku config>unset <appname> --no-restart VAR1
```

Check the value of some variable:

```
$ ssh dokku config:get <appname> VAR1
VAL1
```

Get all the settings in a single line, handy for use in shell commands or to set on another app:

```
$ ssh dokku config <appname> --shell
VAR1=VAL1 VAR2=VAL2
$ export $(ssh dokku config <appname> --shell)
$ ssh dokku config:set <appname1> $(ssh dokku config <appname2> --shell)
```

Get all the settings in a format handy to save in a file for later sourcing in a local shell:

```
$ ssh dokku config <appname> --export
export VAR1=VAL1
export VAR2=VAL2
$ ssh dokku config <appname> --export >appname.env
$ . appname.env
```

Note: you can also set config vals globally - just change `<appname>` to `--global` in any of these commands.

## Logs

Dokku collects stdout from your processes and you can view it with the `dokku logs` command.

nginx logs are similarly stored on the server and can be accessed using `dokku nginx:access-log <appname>` or `dokku nginx:error-log <appname>`.

Dokku event logging can be enabled with dokku events:on and then viewed with `dokku events`. This shows things like deploy steps.

## Deploying from private git repos

Note: this doesn't apply to your main project repo. That can be private and Dokku doesn't care, because you're pushing it directly from your development system to the Dokku server.

But if your requirements include references to private git repos, then you'll need to arrange for Dokku to get access to those repos when it's `pip installing` your requirements.

Docs, such as they are...

I think the upshot is:

- Create a new ssh key for deploying
- Add it to the repo on Github (or whatever) as an authorized deploy key (TODO: Link to github docs on that)
- Drop a copy of the public key file into `/home/dokku/.ssh/` on the Dokku system (with appropriate permissions)

## Deploying non-master branch

The docs

By default, dokku deploys when the app's master branch on the dokku server is updated. There are (at least) two ways to deploy a branch other than master.

1. Push your non-master local branch to the master branch on the server:

```
$ git push dokku <local branch>:master
```

but that requires you to always remember that if you have apps that are always supposed to use a different branch than master.

2. Configure your app so the default branch is different, by changing the config value `DOKKU_DEPLOY_BRANCH`:

```
$ ssh dokku config:set --no-restart <app-name> DOKKU_DEPLOY_BRANCH=<some-branch>
```

This seems like a more useful approach. Just "git push dokku some-branch" every time you want to deploy your app.

## Developing with multiple remote apps

Suppose you have local development and sometimes you want to push to staging and sometimes to production. Maybe you have other apps too.

The key is to set up a different git remote for each remote app. E.g.:

```
$ ssh dokku app:create staging
$ ssh dokku config:set --no-restart staging DOKKU_DEPLOY_BRANCH=develop

$ git remote add staging dokku@my-dokku-server.com:staging

$ ssh dokku app:create production
$ ssh dokku config:set --no-restart production DOKKU_DEPLOY_BRANCH=master
```

```
$ git remote add production dokku@my-dokku-server.com:production
```

Then to deploy, push the approprate branch to the appropriate branch:

```
$ git push staging develop
$ git push production master
```

### Customizing nginx config

If you need to completely override the nginx config, you'll need to provide an nginx config file template.

Luckily, much customization can be done just by providing snippets of configuration for nginx to include after it's base config file.

To do this, arrange for the snippets to get copied to `/home/dokku/<appname>/nginx.conf.d/` during deployment, probably in a pre- or post-deploy script.

### Logging to papertrail

Use the logspout plugin.

### Adding Sentry service

https://github.com/darklow/dokku-sentry

## Elastic Beanstalk with Django

SSH into a random instance. This assumes that you have copied the SSH private key into your `$HOME/.ssh` directory:

```
(bringfido)$ eb ssh staging
```

Open the AWS Elasticbeanstalk web console:

```
(bringfido)$ eb console staging
```

Scale the application to N web instances:

```
(bringfido)$ eb scale <N> staging
```

Check the overall status of the environment, or detailed info about each instance:

```
(bringfido)$ eb status -v staging
(bringfido)$ eb health staging
```

If you need to work with Django on a server, after ssh'ing in:

```
$ . /opt/python/current/env
$ cd /opt/python/current/app
$ python manage.py ....
```

# Email

https://docs.djangoproject.com/en/stable/topics/email/

## API

Send one email:

```
send_mail(subject, message, from_email, recipient_list, fail_silently=False, auth_user=None, auth_pas
```

## Attachments

msg = EmailMessage(...) msg.attach(

filename="any string", content=b"the contents", mimetype="application/sunshine"

)

or

msg.attach(instance of MIMEBase)

## Email backends/handlers

https://docs.djangoproject.com/en/stable/topics/email/#email-backends

For development:

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

For real:

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
```

In-memory backend - The 'locmem' backend stores messages in a special attribute of the django.core.mail module. The outbox attribute is created when the first message is sent. It's a list with an EmailMessage instance for each message that would be sent:

```
EMAIL_BACKEND = 'django.core.mail.backends.locmem.EmailBackend'
```

This backend is not intended for use in production – it is provided as a convenience that can be used during development and testing.

## Settings for email addresses

ADMINS A tuple that lists people who get code error notifications:

```
(('John', 'john@example.com'), ('Mary', 'mary@example.com'))
```

MANAGERS Not needed

DEFAULT_FROM_EMAIL Default email address to use for various automated correspondence from the site manager(s). This doesn't include error messages sent to ADMINS and MANAGERS; for that, see SERVER_EMAIL.

SERVER_EMAIL The email address that error messages come from, such as those sent to ADMINS and MANAGERS.

## Fabric

Sample tasks:

```
@task
def db_backup():
    """
    Backup the database to S3 just like the nightly cron job
    """
    require('environment')
    require('instance', provided_by='instance')
    manage_run("dbbackup --encrypt")


def db_exists(dbname):
    """
    Return True if a db named DBNAME exists on the remote host.
    """
    require('environment', provided_by=SERVER_ENVIRONMENTS)
    output = sudo('psql -l --pset=format=unaligned', user='postgres')
    dbnames = [line.split('|')[0] for line in output.splitlines()]
    return dbname in dbnames


@task
def db_dump(file):
    """
    Dump an instance's database to a remote file.

    Example:

      `fab staging instance:iraq db_dump:/tmp/staging_iraq.dump`

    dumps to staging_iraq.dump
    """
    require('environment', provided_by=SERVER_ENVIRONMENTS)
    require('instance', provided_by='instance')
    remote_file = file

    if files.exists(file):
        if not confirm("Remote file {file} exists and will be overwritten.  Okay?"
                .format(file=remote_file)):
            abort("ERROR: aborting")

    # Don't need remote DB user and password because we're going to run pg_dump as user postgres
    sudo('pg_dump --format=custom --file={outputfile} {dbname}'
         .format(dbname=env.db_name, outputfile=remote_file),
         user='postgres')
    print("Database from {environment} {instance} has been dumped to remote file {file}"
          .format(environment=env.environment, instance=env.instance, file=remote_file))


@task
def local_restore(file):
    """
    Restore a local dump file to the local instance's database.
    :param file:
    :return:
```

```
    """
    # Find out the local DB settings
    import sys
    sys.path[0:0] = ['.']
    from cts.settings.local import DATABASES
    DB = DATABASES['default']
    assert DB['ENGINE'] == 'django.contrib.gis.db.backends.postgis'
    dbname = DB['NAME']
    owner = DB['USER'] or os.getenv('USER')
    local('dropdb {dbname} || true'.format(dbname=dbname), shell="/bin/sh")
    local('createdb --encoding UTF8 --lc-collate=en_US.UTF-8 --lc-ctype=en_US.UTF-8 --template=templa
    local('sudo -u postgres pg_restore -Ox -j4 --dbname={dbname} {file}'.format(dbname=dbname, file=f


@task
def db_restore(file):
    """
    Restore a remote DB dump file to a remote instance's database.

    This will rename the existing database to {previous_name}_bak
    and create a completely new database with what's in the dump.

    If there's already a backup database, the restore will fail.

    Example:

      `fab staging instance:iraq db_restore:/tmp/staging_iraq.dump`

    :param file: The remote file to restore.
    """
    require('environment', provided_by=SERVER_ENVIRONMENTS)
    require('instance', provided_by='instance')

    renamed = False
    restored = False

    if not files.exists(file):
        abort("Remote file {file} does not exist".format(file=file))

    try:
        if db_exists(env.db_name):
            # Rename existing DB to backup
            db_backup = '{dbname}_bak'.format(dbname=env.db_name)
            if db_exists(db_backup):
                if confirm("There's already a database named {db_backup}. Replace with new backup?"
                           .format(db_backup=db_backup)):
                    sudo('dropdb {db_backup}'.format(db_backup=db_backup),
                         user='postgres')
                else:
                    abort("ERROR: There's already a database named {db_backup}. "
                          "Restoring would clobber it."
                          .format(db_backup=db_backup))
            sudo('psql -c "ALTER DATABASE {dbname} RENAME TO {db_backup}"'
                 .format(dbname=env.db_name, db_backup=db_backup),
                 user='postgres')
            renamed = True
            print("Renamed {dbname} to {db_backup}".format(dbname=env.db_name, db_backup=db_backup))
```

```
        remote_file = file

        # Create new, very empty database.
        # * We can't use --create on the pg_restore because that will always restore to whatever
        #   db name was saved in the dump file, and we don't want to be restricted that way.
        # * Any extensions the backed-up database had will be included in the restore, so we
        #   don't need to enable them now.

        # If these parameters change, also change the parameters in conf/salt/project/db/init.sls
        # (TODO: we could use the output of psql -l to copy most of these settings from the
        # existing database.)
        sudo('createdb --encoding UTF8 --lc-collate=en_US.UTF-8 '
             '--lc-ctype=en_US.UTF-8 --template=template0 --owner {owner} {dbname}'
             .format(dbname=env.db_name, owner=env.db_owner),
             user='postgres')

        # Don't need remote DB user and password because we're going to
        # run pg_restore as user postgres
        sudo('pg_restore -1 --dbname={dbname} {filename}'
             .format(dbname=env.db_name, filename=remote_file),
             user='postgres')
        restored = True

        # Run ANALYZE on the db to help Postgres optimize how it accesses it
        sudo('psql {dbname} -c ANALYZE'.format(dbname=env.db_name),
             user='postgres')

        print("Database for {environment} {instance} has been restored from remote file {file}"
              .format(environment=env.environment, instance=env.instance, file=remote_file))
    finally:
        if renamed and not restored:
            print("Error occurred after renaming current database, trying to rename it back")
            if db_exists(env.db_name):
                # We already created the new db, but restore failed; delete it
                sudo('dropdb {dbname}'.format(dbname=env.dbname), user='postgres')
            sudo('psql -c "ALTER DATABASE {db_backup} RENAME TO {dbname}"'
                 .format(dbname=env.db_name, db_backup=db_backup),
                 user='postgres')
            print("Successfully put back the original database.")
```

# Forms

- https://docs.djangoproject.com/en/stable/topics/forms/
- https://docs.djangoproject.com/en/stable/ref/forms/api/

A basic form:

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
```

Processing a form in a view function:

```
def contact(request):
    if request.method == 'POST': # If the form has been submitted...
```

```
        form = ContactForm(request.POST) # A form bound to the POST data
        if form.is_valid(): # All validation rules pass
            # Process the data in form.cleaned_data
            # ...
            return HttpResponseRedirect('/thanks/') # Redirect after POST
    else:
        form = ContactForm() # An unbound form

    return render_to_response('contact.html', {
        'form': form,
    })
```

https://docs.djangoproject.com/en/stable/topics/forms/modelforms/

A model form:

```
from django.forms import ModelForm, ValidationError

# Create the form class.
class ArticleForm(ModelForm):
    class Meta:
        model = Article
        fields = ('name', 'title')
        # or
        exclude = ('birth_date')

    def clean_fieldname(self):
        if 'fieldname' in self.cleaned_data:
            data = self.cleaned_data['fieldname']
            if not /valid/:
                raise ValidationError("msg")
            return data

    def clean(self):
        data = self.cleaned_data
        if not /valid/:
            raise ValidationError("msg")
        return data

# Creating a form to add an article.
form = ArticleForm()
...
new_article = form.save()

# Creating a form to change an existing article.
article = Article.objects.get(pk=1)
form = ArticleForm(instance=article)
...
form.save()

# Create a form to edit an existing Article, but use POST data to populate the form.
a = Article.objects.get(pk=1)
f = ArticleForm(request.POST, instance=a)
f.save()
```

Render form in template:

```
<!-- Using table - avoid that part - but this does show how to render the fields individually -->
      <form {% if form.is_multipart %}enctype="multipart/form-data"{% endif %} action="" method="post
```

```
        <table>
          <fieldset>
            {% if form.non_field_errors %}
              <tr><td colspan="2">{{ form.non_field_errors }}</td></tr>
            {% endif %}
            {% for field in form %}
              <tr{% if field.field.required %} class="required"{% endif %}>
                <th style="text-align: left"><label for="{{ field.id_for_label }}">{{ field.label }}
                <td>{% if field.errors %}
                      {{ field.errors }}<br/>
                    {% endif %}

                    {{ field }}

                    or even

                    <input id="{{ field.id_for_label }}"
                     name="{{ field.html_name }}"
                     value="{{ field.value }}"
                    {% if field.field.max_length != None %}
                        maxlength="{{ field.field.max_length }}"
                    {% endif %}
                    {% if field.field.min_length != None %}
                        minlength="{{ field.field.min_length }}"
                    {% endif %}
                     >
                    {% if field.help_text %}
                     <br/><span class="helptext">{{ field.help_text }}</span>
                     {% endif %}
                </td>
              </tr>
            {% endfor %}
          </fieldset>
        </table>
        <div class="ctrlHolder buttonHolder">
          <button type="submit" class="primaryAction" name="submit_changes">Submit changes</button>
        </div>
      </form>

<!-- Using a list, which is preferred -->

    <form {% if form.is_multipart %}enctype="multipart/form-data"{% endif %} action="" method="post"
        <fieldset>
            <ul>
                {{ form.as_ul }}
                <li>
                    <div class="ctrlHolder buttonHolder">
                        <button type="submit" class="primaryAction" name="submit_changes">Submit char
                    </div>
                </li>
            </ul>
        </fieldset>
    </form>
```

## Read-only form

Call this on the form:

```
def make_form_readonly(form):
    """
    Set some attributes on a form's fields that, IN COMBINATION WITH TEMPLATE CHANGES,
    allow us to display it as read-only.
    """

    # Note that a new BoundField is constructed on the fly when you access
    # form[name], so any data we want to persist long enough for the template
    # to access needs to be on the "real" field.  We just use the BoundField
    # to get at the field value.

    for name in form.fields:
        field = form.fields[name]
        bound_field = form[name]
        if hasattr(field.widget, 'choices'):
            try:
                display_value = dict(field.widget.choices)[bound_field.value()]
            except KeyError:
                display_value = ''
        else:
            display_value = bound_field.value()

        field.readonly = True
        field.display_value = display_value
```

Do things like this in the templates:

```
{# Date field #}
{% if field.field.readonly %}
    <span class="form-control">{{ field.value|date:'c' }}</span>
{% else %}
    <input type="date" class="form-control" id="{{ field.id_for_label }}" name="{{ field.html_name }}
{% endif %}

{# input fields #}
{% if field.field.readonly %}
    <span class="form-control">{{ field.value }}</span>
{% else %}
    <input type="{% block input_field_type %}text{% endblock %}" class="form-control" id="{{ field.id
{% endif %}

{# select fields #}
{% if field.field.readonly %}
    <span class="form-control">{{ field.field.display_value }}</span>
{% else %}
    <select class="form-control" id="{{ field.id_for_label }}"  name="{{ field.html_name }}" placehol
      {% for val, label in field.field.widget.choices %}
        <option value="{{ val }}"{% if field.value|stringformat:'s' == val|stringformat:'s' %} select
      {% endfor %}
    </select>
{% endif %}
```

## Quick And Dirty Home Page

In `urls.py`:

```
from django.views.generic import TemplateView

urlpatterns = [
    ...
    url(r'^$', TemplateView.as_view(template_name='home.html'), name='home'),
]
```

## Logging

Some best practices for Django logging.

https://docs.djangoproject.com/en/stable/topics/logging/ https://docs.djangoproject.com/en/stable/topics/logging/#configuring-logging

https://docs.python.org/2/howto/logging.html

https://docs.python.org/2/library/logging.handlers.html#rotatingfilehandler https://docs.python.org/2/library/logging.handlers.html#time

INFO level logging for celery is very verbose

If you have DEBUG on, Django logs all SQL queries

### Default

Here's what Django uses (around 1.7, anyway) if you don't configure logging:

```
# Default logging for Django. This sends an email to the site admins on every
# HTTP 500 error. Depending on DEBUG, all other log records are either sent to
# the console (DEBUG=True) or discarded by mean of the NullHandler (DEBUG=False).
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse',
        },
        'require_debug_true': {
            '()': 'django.utils.log.RequireDebugTrue',
        },
    },
    'handlers': {
        'console': {
            'level': 'INFO',
            'filters': ['require_debug_true'],
            'class': 'logging.StreamHandler',
        },
        'null': {
            'class': 'logging.NullHandler',
        },
        'mail_admins': {
            'level': 'ERROR',
            'filters': ['require_debug_false'],
```

```
            'class': 'django.utils.log.AdminEmailHandler'
        }
    },
    'loggers': {
        'django': {
            'handlers': ['console'],
        },
        'django.request': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            'propagate': False,
        },
        'django.security': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            'propagate': False,
        },
        'py.warnings': {
            'handlers': ['console'],
        },
    }
}
```

## Console

Log errors to console in local.py:

```
LOGGING.setdefault('formatters', {})
LOGGING['formatters']['verbose'] = {
    'format': '[%(name)s] Message "%(message)s" from %(pathname)s:%(lineno)d in %(funcName)s'
}
LOGGING['handlers']['console'] = {
    'class': 'logging.StreamHandler',
    'formatter': 'verbose',
    'level': 'ERROR',
}
LOGGING['loggers']['django'] = {
            'handlers': ['console'],
            'level': 'ERROR',
            'propagate': True,
}
```

## Development

For local development, we want lots of output saved to a log file in case we need to look back at a problem, but no emailing of exceptions and such.

In settings:

```
LOG_DIR = os.path.join(PROJECT_ROOT, '..', 'log')

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {  # Log to stdout
```

```
            'level': 'INFO',
            'class': 'logging.StreamHandler',
        },
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': os.path.join(LOG_DIR, 'django_debug.log',
        }
    },
    'root': {  # For dev, show errors + some info in the console
        'handlers': ['console'],
        'level': 'INFO',
    },
    'loggers': {
        'django.request': {  # debug logging of things that break requests
            'handlers': ['file'],
            'level': 'DEBUG',
            'propagate': True,
        },
    },
}
```

Or how about:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'simple': {
            'format': '%(name)-20s %(levelname)-8s %(message)s',
        },
    },
    'handlers': {
        'console': {  # Log to stdout
            'level': 'INFO',
            'class': 'logging.StreamHandler',
            'formatter': 'simple',
        },
    },
    'root': {  # For dev, show errors + some info in the console
        'handlers': ['console'],
        'level': 'INFO',
    },
}
```

## Staging

FIXME: Add celery exceptions

@tobiasmcnulty also mentioned: "re: celery error emails, this is a good setting to have enabled: http://celery.readthedocs.org/en/latest/configuration.html#celery-send-task-error-emails"

On staging, we still want lots of info logged semi-permanently (to files), but we also want to be emailed about exceptions to make sure we find out about problems before we deploy them to production.

Emails should go to the devs, not the client or production site admins.

Like so:

```
ADMINS = (
    ('XXX DevTeam', 'xxx-dev-team@example.com'),
)

LOG_DIR = os.path.join(PROJECT_ROOT, '..', 'log')

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'file': {  # Rotate log file daily, only keep 1 backup
            'level': 'DEBUG',
            'class': 'logging.handlers.TimedRotatingFileHandler',
            'filename': os.path.join(LOG_DIR, 'django_debug.log',
            'when': 'd',
            'interval': 1,
            'backupCount': 1,
        },
        'mail_admins': {
            'level': 'ERROR',
            'class': 'django.utils.log.AdminEmailHandler'
        },
    },
    # EMAIL all errors (might not want this, but let's try it)
    'root': {
        'handlers': ['mail_admins'],
        'level': 'ERROR',
    },
    'loggers': {
        'django.request': {
            'handlers': ['file'],
            'level': 'INFO',
            'propagate': True,
        },
    },
}
```

## Production

Mark says: for production I like to log to syslog which can then be shipped elsewhere without changing the application

(https://docs.python.org/2/library/logging.handlers.html#logging.handlers.SysLogHandler ?)

@Scottm and I have been talking about making that more common: log to syslog, ship to Logstash, monitor via Kibana http://www.elasticsearch.org/overview/kibana/

getting Nginx to log to syslog is kind of a pain you basically have to get syslog to monitor the file and ship it Logstash + Kibana looks much easier to manage/configure than Graylog2

the plan was to add it to Ona but that isn't done yet (as of Aug 28, 2014) CCSR was/is using Graylog2 Minidam does syslog –> Loggly libya is using logstash -> graylog (in addition to sentry)

## Example

Here's what we've got set up for Django logging on one project. This sends everything level INFO and higher to a local log file and a Graylog instance. Anything ERROR and higher is emailed to admins and sent to a Sentry instance, which can send more notifications.

In environment:

```
SENTRY_DSN: http://long_hex_string:long_hex_string@hostname:9000/3
```

Requirements:

```
raven==3.6.1
```

Settings:

```
INSTALLED_APPS = (
    ...
    'raven.contrib.django.raven_compat',  # Sentry logging client
    ...
)

CELERY_SEND_TASK_ERROR_EMAILS = True

# Send ERRORS to email and sentry.
# Send a fair bit of info to graylog and a local log file
# (but not debug level messages, ordinarily).
LOGGING = {
    'version': 1,
    'disable_existing_loggers': True,
    'filters': {
        # This filter strips out request information from the message record
        # so it can be sent to Graylog (the request object is not picklable).
        'django_exc': {
            '()': 'our_filters.RequestFilter',
        },
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse'
        },
        # This filter adds some identifying information to each message, to make
        # it easier to filter them further, e.g. in Graylog.
        'static_fields': {
            '()': 'our_filters.StaticFieldFilter',
            'fields': {
                'deployment': 'project_name',
                'environment': 'staging'   # can be overridden, e.g. 'staging' or 'production'
            },
        },
    },
    'formatters': {
        'basic': {
            'format': '%(asctime)s %(name)-20s %(levelname)-8s %(message)s',
        },
    },
    'handlers': {
        'file': {
            'level': 'DEBUG',  # Nothing here logs DEBUG level messages ordinarily
            'class': 'logging.handlers.RotatingFileHandler',
            'formatter': 'basic',
            'filename': os.path.join(LOG_ROOT, 'django.log'),
            'maxBytes': 10 * 1024 * 1024,  # 10 MB
            'backupCount': 10,
        },
        'graylog': {
            'level': 'INFO',
            'class': 'graypy.GELFHandler',
```

```
            'host': env_or_default('GRAYLOG_HOST', 'monitor.caktusgroup.com'),
            'port': 12201,
            'filters': ['static_fields', 'django_exc'],
        },
        'mail_admins': {
            'level': 'ERROR',
            'class': 'django.utils.log.AdminEmailHandler',
            'include_html': False,
            'filters': ['require_debug_false'],
        },
        'sentry': {
            'level': 'ERROR',
            'class': 'raven.contrib.django.raven_compat.handlers.SentryHandler',
        },
    },
    'root': {
        # graylog (or any handler using the 'django_exc' filter ) should be last
        # because it will alter the LogRecord by removing the `request` field
        'handlers': ['file', 'mail_admins', 'sentry', 'graylog'],
        'level': 'WARNING',
    },
    'loggers': {
        # These 2 loggers must be specified, otherwise they get disabled
        # because they are specified by django's DEFAULT_LOGGING and then
        # disabled by our 'disable_existing_loggers' setting above.
        # BEGIN required loggers #
        'django': {
            'handlers': [],
            'propagate': True,
        },
        'py.warnings': {
            'handlers': [],
            'propagate': True,
        },
        # END required loggers #
        # The root logger will log anything WARNING and higher, so there's
        # no reason to add loggers here except to add logging of lower-level information.
        'libya_elections': {
            'handlers': ['file', 'graylog'],
            'level': 'INFO',
        },
        'nlid': {
            'handlers': ['file', 'graylog'],
            'level': 'INFO',
        },
        'register': {
            'handlers': ['file', 'graylog'],
            'level': 'INFO',
        },
        'bulk_sms': {
            'handlers': ['file', 'graylog'],
            'level': 'INFO',
        },
    }
}

#
# our_filters.py
```

```
#
import logging


class QuotelessStr(str):
    """
    Return the repr() of this string *without* quotes.  This is a
    temporary fix until https://github.com/severb/graypy/pull/34 is resolved.
    """
    def __repr__(self):
        return self


class StaticFieldFilter(logging.Filter):
    """
    Python logging filter that adds the given static contextual information
    in the ``fields`` dictionary to all logging records.
    """
    def __init__(self, fields):
        self.static_fields = fields

    def filter(self, record):
        for k, v in self.static_fields.items():
            setattr(record, k, QuotelessStr(v))
        return True


class RequestFilter(logging.Filter):
    """
    Python logging filter that removes the (non-pickable) Django ``request``
    object from the logging record.
    """
    def filter(self, record):
        if hasattr(record, 'request'):
            del record.request
        return True
```

## Including info like the emailed errors do

```
from django.views.debug import TECHNICAL_500_TEXT_TEMPLATE, get_safe_settings, \
    get_exception_reporter_filter
from django.views.decorators.debug import sensitive_post_parameters

t = Template(TECHNICAL_500_TEXT_TEMPLATE)
filter = get_exception_reporter_filter(request)
r = t.render(Context({
    'request': request,
    'is_email': True,
    'filtered_POST': filter.get_post_parameters(request),
    'settings': get_safe_settings(),
    'server_time': timezone.now(),
    'django_version_info': get_version(),
}, autoescape=False))
logger.error(
    "Got CSRF failure, reason=%s. %s", reason, r,
)
```

# Django login and logout

Django doc

- In settings.py, add:

```
LOGIN_URL = '/login/'
```

- In urls.py, add:

```
urlpatterns += patterns('',
                        (r'^login/$', django.contrib.auth.views.login),
                        (r'^logout/$', django.contrib.auth.views.logout),
                        )
```

- Create a template "registration/login.html", copying from the sample in the doc
- Add a logout link to your base template:

```
<a href="/logout/">Logout</a>
```

- On each view function where users should be logged in before using, add the decorator:

```
@login_required
def myview(...)
```

# Migrations

Data migration:

```
def no_op(apps, schema_editor):
    pass

def create_types(apps, schema_editor):
    ServiceType = apps.get_model('services', 'ServiceType')
    db_alias = schema_editor.connection.alias
    # do stuff
    ServiceType.objects.using(db_alias)....


class Migration(migrations.Migration):
    ...
    operations = [
        migrations.RunPython(create_types, no_op),
    ]
```

## Getting past bad migrations

For example, earlier migrations refer to models in apps that no longer exist.

The simplest thing is to start from an existing database so you don't have to migrate.

If you need to start from scratch, you should be able to:

```
syncdb --all
migrate --fake
```

# NGINX

Some tips on Nginx for use with Django

## Files

Just add a new file to `/etc/nginx/sites-eanbled` for each site, making sure `server_name` is set correctly in each.

## Redirecting to SSL

We usually want to force SSL:

```
server {
  listen *:80;
  listen [::]:80;
  server_name DOMAIN;
  access_log PATHo_access.log;
  error_log PATH_error.log;
  return 301 https://DOMAIN$request_uri;
}
```

## Proxying to gunicorn

Serve static and media files with nginx, and proxy everything else to Django:

```
upstream django {
  server unix:/tmp/PATH fail_timeout=0;
}

server {
  listen *:443 ssl;   # add spdy here too if you want
  listen [::]:443 ssl;
  server_name DOMAIN;
  ssl_certificate PATH.crt;
  ssl_certificate_key PATH.key;

  access_log PATH_access.log;
  error_log PATH_error.log;
  root PATH;
  location /media {
    alias PATH;
  }
  location /static {
    alias PATH;
  }
  location / {
    client_max_body_size 500M;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header Host $host;
    proxy_redirect off;
    proxy_buffering on;
```

```
    proxy_intercept_errors on;
    proxy_pass http://django;
  }

  # See https://www.trevorparker.com/hardening-ssl-in-nginx/
  ssl_protocols             TLSv1 TLSv1.1 TLSv1.2;
  ssl_prefer_server_ciphers on;
  ssl_ciphers               DHE-RSA-AES128-GCM-SHA256:DHE-DSS-AES128-GCM-SHA256:ECDHE-RSA-AES1\
28-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM\
-SHA384:kEDH+AESGCM:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES\
256-SHA256:DHE-DSS-AES256-SHA:DHE-RSA-AES256-SHA:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SH\
A256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SH\
A384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SH\
A256:AES256-SHA256:AES128-SHA:AES256-SHA:AES:CAMELLIA:DES-CBC3-SHA:!aNULL:!eNULL:!EXPORT:!DES:\
!RC4:!MD5:!PSK:!aECDH:!EDH-DSS-DES-CBC3-SHA:!EDH-RSA-DES-CBC3-SHA:!KRB5-DES-CBC3-SHA;
  ssl_session_timeout       5m;
  ssl_session_cache         shared:SSL:10m;

  add_header Strict-Transport-Security max-age=31536000;
}
```

## Permissions

(Note: this page is about authorization, not authentication.)

Link: https://docs.djangoproject.com/en/stable/topics/auth/default/#topic-authorization

User objects have two many-to-many fields: groups and user_permissions. User objects can access their related objects in the same way as any other Django model:

```
myuser.groups = [group_list]
myuser.groups.add(group, group, ...)
myuser.groups.remove(group, group, ...)
myuser.groups.clear()
myuser.user_permissions = [permission_list]
myuser.user_permissions.add(permission, permission, ...)
myuser.user_permissions.remove(permission, permission, ...)
myuser.user_permissions.clear()
```

Assuming you have an application with an app_label foo and a model named Bar, to test for basic permissions you should use:

- add: user.has_perm('foo.add_bar')
- change: user.has_perm('foo.change_bar')
- delete: user.has_perm('foo.delete_bar')

**Note:** There is no default permission for read access.

The Permission model is rarely accessed directly, but here it is:

```
permission = Permission.objects.create(codename='can_publish',
                                       name='Can Publish Posts',  # verbose human-readable name
                                       content_type=content_type)
```

Permissions are more commonly referred to by a string, of the form "app_label.codename". E.g., `if user.has_perm('myapp.codename'): do something`.

Confusingly, the `Permission` model has no `app_label` field. It uses `content_type__app_label` for that.

> **Warning:** This means all permissions need a content type, whether it makes sense for that permission to be applied to a particular model or not.

To create a new permission programmatically:

```
content_type = ContentType.objects.get_for_model(BlogPost)
permission = Permission.objects.create(codename='can_publish',
                                       name='Can Publish Posts',
                                       content_type=content_type)
```

## Default permissions

https://docs.djangoproject.com/en/stable/topics/auth/default/#default-permissions

For every model in an installed app, Django automatically creates three permissions: *applabel.add_modelname*, *applabel.change_modelname*, and *applabel.delete_modelname*, where the modelname is lowercased.

## Adding model permissions

https://docs.djangoproject.com/en/stable/ref/models/options/#permissions

You can ask Django to create more permissions for a model:

```
class Meta:
    permissions = [
        ('codename', 'verbose name'),
    ]
```

When the table is created during `syncdb`, Django will create the additional Permission objects too.

> **Warning:** Prior to Django 1.7, such permissions were *only* created when Django creates the model's table during `syncdb`. Adding permissions to the model definition later does nothing (unless you drop the table and force Django to create it again).

You can programmatically force Django to create additional Permissions with code like:

```
from django.db.models import get_models, get_app
from django.contrib.auth.management import create_permissions

apps = set([get_app(model._meta.app_label) for model in get_models()])
for app in apps:
    create_permissions(app, None, 2)
```

## Best practices

- Plan not to give users specific permissions, except when you have to make an exception to your usual policies.
- Design groups with useful sets of permissions.
- Plan to add users to the appropriate groups depending on their roles.

- Provide a way to ensure the groups continue to have the permissions you want.

Fixtures aren't a bad way to provide initial data, but setting them up for automatic loading is deprecated with Django 1.7 and will go away with Django 2.0. Instead, load them from a data migration. This is better in some ways anyway, because the migration will use the same version of the models that the fixtures were written for at the time. (Though, this doesn't matter so much for Permissions and Groups, which we don't really expect to change their schemas...)

Add utility methods like this, maybe in *accounts/utils.py* or equivalent:

```
def permission_names_to_objects(names):
    """
    Given an iterable of permission names (e.g. 'app_label.add_model'),
    return an iterable of Permission objects for them.  The permission
    must already exist, because a permission name is not enough information
    to create a new permission.
    """
    result = []
    for name in names:
        app_label, codename = name.split(".", 1)
        # Is that enough to be unique? Hope so
        try:
            result.append(Permission.objects.get(content_type__app_label=app_label,
                                                 codename=codename))
        except Permission.DoesNotExist:
            logger.exception("NO SUCH PERMISSION: %s, %s" % (app_label, codename))
            raise
    return result


def get_all_perm_names_for_group(group):
    # Return the set of permission names that the group should contain


def create_or_update_groups():
    for group_name, perm_names in GROUP_PERMISSIONS.iteritems():
        group, created = Group.objects.get_or_create(name=group_name)
        perms_to_add = permission_names_to_objects(get_all_perm_names_for_group(group))
        group.permissions.add(*perms_to_add)
        if not created:
            # Group already existed - make sure it doesn't have any perms we didn't want
            to_remove = set(group.permissions.all()) - set(perms_to_add)
            if to_remove:
                group.permissions.remove(*to_remove)
```

## Checking permissions in templates

https://docs.djangoproject.com/en/stable/topics/auth/default/#authentication-data-in-templates

> {% if user.is_authenticated %} {% if perms.applabel %} {# user has any permissions in app *applabel* #} {% if 'applabel' in perms %} {# same as above %} {% if perms.applabel.change_thing %} {# user has 'change_thing' permission in app *applabel* #} {% if 'applabel.change_thing' in perms %} {# same as above #}

# Queries and Querysets

## Field lookups

exact, iexact, contains, icontains, startswith, istartswith, endswith, iendswith, in, gt, gte, lt, lte, range, year, month, day, week_day, hour, minute, second, isnull, search, regex

## Optimizing Django queries on big data

Suppose you have a query that needs to run against a table or tables with many millions of rows. Maybe you need to operate on a couple million of them. What are the do's and don't's of a Django query that will not pessimize performance (time and memory use)?

- Don't bother with .iterator(), it downloads the whole result and then iterates over it. It does not do what many of us think/thought it did (use a database cursor to pull down the results only as you work through them)
- Do limit the query ([start:end]) and run it repeatedly in reasonable sized batches, to avoid downloading too big a chunk
- Do use .only() and its kin to minimize how much of each record is downloaded to what you need
- Do not order_by() unless you must - it forces the DB to collect the results of the whole query first so it can sort them, even if you then limit the results you retrieve
- Same for .distinct().

## The model that a queryset is over

queryset.model

## Combining querysets

Given two querysets over the same model, you can do things like this:

```
queryset = queryset1 & queryset2
queryset = queryset1 | queryset2
queryset = queryset1 & ~queryset2
```

(similar to `Q` objects)

## Custom QuerySets

Calling custom QuerySet methods from the manager

Creating a manager with QuerySet methods¶:

```
class Person(models.Model):
    ...
    people = PersonQuerySet.as_manager()


class BaseManager(....):
    ....
```

```
class MyModel(models.Model):
    objects = BaseManager.from_queryset(CustomQuerySet)()
```

### Custom Lookups

Adding to the kwargs you can pass to *filter* and *exclude* etc.

Custom Lookups

## Security

### Protect internal services

Vse a VPN, or check out oauth2_proxy or similar services.

### Django

(django-secure appears to be abandoned. Last change was in 2014, and it doesn't load under Django 1.11/Python 3.6.)

-Best practice: install `django-secure` and run `manage.py checksecure` to make sure all the right settings are enabled.-

See also OWASP.

### Admin

Don't leave it externally accessible, even with a password.

### SSH

Two important settings in `/etc/sshd_config`:

- Disable root login:

```
    PermitRootLogin no
```

- Disable password auth:

```
    PasswordAuthentication no
```

Also consider changing to some port other than 22.

### SSL

SEE ALSO *NGINX* and Django docs on SSL and https.

Basically, make sure nginx is setting X-Forwarded-Proto, then add to settings:

```
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')
```

Django security

djangocon 2011 Paul McMillan

http://subversivecode.com/talks/djangocon-us-2011

HSTS

django needs better password hash (SHA1 not broken but very fast)

OpenID much more secure against password cracking (because cracker won't have passwords)

password reset strings can be eventually worked out with a timing attack (if you have a long time and a fast connection)

same for which userids exist on the site

you should do rate limiting:

mod_evasive (apache) HttpLimitReqModule (nginx)

do NOT use random.Random() for security functions, not cryptographically secure; use random.SystemRandom() instead e.g.:

```
from random import SystemRandom as random
xxxx random.choice(yyyy)...
```

Be very careful with pickle, it'll execute anything in the pickled data when you unpickle it

BOOK: The web application hacker's handbook (new version coming out soon (as of 9/8/2011))

SITE: lost.org? (not sure I heard that right)(no I didn't)

# Settings

## Using the right one

Baumgartner suggests symlinking the desired one (e.g. *dev.py* or *deploy.py*) to *local.py* and hard-coding that in *manage.py*.

Greenfelds suggest... (FILL THIS IN)

12-factor says there should only be one settings file, and any values that vary by deploy should be pulled from the environment. See *Env vars*.

## Secret key

Generate a secret key:

```
from django.utils.crypto import get_random_string
chars = 'abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*(-_=+)'
SECRET_KEY = get_random_string(50, chars)
```

(https://github.com/django/django/blob/master/django/core/management/commands/startproject.py#L26)

## Env vars

Suppose you have env vars in a .env file:

```
SECRET_KEY=jdfsdfsdf
PASSWORD=jsdkfjsdlkfjdsf
```

You can load them into Django using dotenv. Pop open *manage.py*. Add:

```
import dotenv
dotenv.read_dotenv()
```

Or in a settings file:

```
SECRET_KEY = os.environ.get("SECRET_KEY")
```

And if they're not all strings, use ast:

```
import ast, os

DEBUG = ast.literal_eval(os.environ.get("DEBUG", "True"))
TEMPLATE_DIRS = ast.literal_eval(os.environ.get("TEMPLATE_DIRS", "/path1,/path2"))
```

You can load them into a shell this way:

```
export $(cat .env | grep -v ^# | xargs)
```

# Templates

## Template tag to set variables

Example usage:

```
{% set foo="bar" a=1 %}
...
Hello everyone, foo is {{ foo }} and a is {{ a }}.
```

Here's the code:

```
from django import template

register = template.Library()

@register.simple_tag(takes_context=True)
def set(context, **kwargs):
    context.update(kwargs)
    return ''
```

## Working block tag with arguments

Here's an example of a working block tag. Usage is `{% detail_link arg1=1 arg2=2 %}...{% end_detail_link %}` and what ends up in the output is `<a arg1="1" arg2="2" target="_blank">...</a>`.

The code:

```
from django import template
from django.template.base import token_kwargs, TemplateSyntaxError
from django.utils.html import format_html, format_html_join
from django.utils.safestring import mark_safe
```

```
register = template.Library()


def do_detail_link(parser, token):
    """
    Block tag to help render links to detail pages consistently
    with an option to open in a new tab or window.

    {% detail_link href="xxxx" arg1="yyy" ... %} what to display {% end_detail_link %}

    is rendered as

    <a href="xxxx" arg1="yyy" ... target="_blank" %} what to display </a>

    This adds `target="_blank"` to open the link in a new tab or window.
    That's the main purpose of this block tag (and so we can disable that in
    one place, here, if we ever want to).  But you can override it by specifying
    another value for `target` if you want.
    """
    # This is called each time a detail_link tag is encountered while parsing
    # a template.

    # Split the contents of the tag itself
    args = token.split_contents()  # e.g. ["detail_link", "arg1='foo'", "arg2=bar"]
    tag_name = args.pop(0)

    # Parse out the arg1=foo arg2=bar ... arguments from the arg list into a dictionary.
    # kwargs will have "arg1" etc as keys, while the values will be
    # template thingies that can later be rendered using different contexts
    # to get their value at different times.
    kwargs = token_kwargs(args, parser)

    # If there are any args left, we have a problem; this tag only
    # accepts kwargs.
    if args:
        raise TemplateSyntaxError("%r only accepts named kwargs" % tag_name)

    # Open in new tab unless otherwise told (by setting target to something else).
    if 'target' not in kwargs:
        kwargs['target'] = parser.compile_filter('"_blank"')

    # Parse inside of block *until* we're looking at {% end_detail_link %},
    # then we don't care about end_detail_link, so discard it.
    # When we return, the parsing will then continue after our end tag.
    nodelist = parser.parse(('end_detail_link',))
    parser.delete_first_token()

    # Now return a node for the parsed template
    return DetailLinkNode(nodelist, tag_name, kwargs)


register.tag('detail_link', do_detail_link)


class DetailLinkNode(template.Node):
    """
    Stores info about one occurrence of detail_link in a template.
```

```
    See also `do_detail_link`.
    """
    def __init__(self, nodelist, tag_name, kwargs):
        self.nodelist = nodelist
        self.tag_name = tag_name
        self.kwargs = kwargs

    def render(self, context):
        """Turn this node into text using the given context."""

        # Start with the part inside the block
        innerds = self.nodelist.render(context)

        # Now work out the <a> wrapper.
        args = format_html_join(
            ' ',
            '{}="{}"',
            ((name, value.resolve(context)) for name, value in self.kwargs.items())
        )
        result = format_html(
            mark_safe("<a {}>{}</a>"),
            args,
            mark_safe(innerds)
        )
        return result
```

### Debugging template syntax errors during tests

The normal error message when a view fails rendering a template during testing gives no clue where the error is.

You can get a better idea by temporarily editing your local Django installation. Find the file `django/template/base.py`. Around line 194 (in Django 1.8.x), in the `__init__` method of the `Template` class, look for this code:

```
self.nodelist = engine.compile_string(template_string, origin)
```

and change it to:

```
try:
    self.nodelist = engine.compile_string(template_string, origin)
except TemplateSyntaxError:
    print("ERROR COMPILING %r" % origin.name)
    raise
```

TODO: would be nice to get a line number too (this just gives a filename, which is often enough in combination with the error message).

## Testing

```
self.assertEqual(a, b, msg=None)


rsp = self.client.get(url, [follow=True])
rsp = self.client.post(url, data, [follow=True])
```

```
rsp.content is a byte string
rsp['HeaderName']
rsp.context['template_var']

assert self.client.login(**login_parms)
login(email='foo@example.com', password='cleartextpassword')
```

# http://docs.python.org/library/unittest.html # https://docs.djangoproject.com/en/stable/topics/testing/

```
from django.test import TestCase
from django.contrib.auth.models import User


class XxxTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user('tester', 'test@example.com', 'testpass')

    def test_something(self):
        response = self.client.get(show_timemap)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.context['lat'], '123.123')
        self.assertEqual(response.context['lon'], '456.456')
        self.assertContains(response, "some text")
        self.assertNotContains(response, "other text")
        self.assertIsNone(val)
        self.assertIsNotNone(val)
        self.assertIn(thing, iterable)  # Python >= 2.7
        self.assertNotIn(thing, iterable)  # Python >= 2.7
```

# Test uploading a file

(from https://docs.djangoproject.com/en/stable/topics/testing/tools/#django.test.Client.post)

Submitting files is a special case. To POST a file, you need only provide the file field name as a key, and a file handle to the file you wish to upload as a value. For example:

```
>>> c = Client()
>>> with open('wishlist.doc') as fp:
...     c.post('/customers/wishes/', {'name': 'fred',
...                                   'attachment': fp})
```

(The name attachment here is not relevant; use whatever name your file-processing code expects.)

Note that if you wish to use the same file handle for multiple post() calls then you will need to manually reset the file pointer between posts. The easiest way to do this is to manually close the file after it has been provided to post(), as demonstrated above.

You should also ensure that the file is opened in a way that allows the data to be read. If your file contains binary data such as an image, this means you will need to open the file in rb (read binary) mode.

## Writing a test for a separately-distributed Django app

# setup.py:

```
...
setup(
    ...
    test_suite="runtests.runtests",
```

```
    ...
)
```

# runtests.py:

```
#!/usr/bin/env python
import os
import sys

from django.conf import settings

if not settings.configured:
    settings.configure(
        DATABASES={
            'default': {
                'ENGINE': 'django.db.backends.sqlite3',
                'NAME': ':memory:',
            }
        },
        INSTALLED_APPS=(
            'selectable',
        ),
        SITE_ID=1,
        SECRET_KEY='super-secret',
        ROOT_URLCONF='selectable.tests.urls',
    )


from django.test.utils import get_runner


def runtests():
    TestRunner = get_runner(settings)
    test_runner = TestRunner(verbosity=1, interactive=True, failfast=False)
    args = sys.argv[1:] or ['selectable', ]
    failures = test_runner.run_tests(args)
    sys.exit(failures)


if __name__ == '__main__':
    runtests()
```

## Translation

Switch context to a new language:

```
from django.utils import translation
translation.activate('en-us')
```

(link to Using translations outside views and templates)

# URLs

## reverse

```
from django.core.urlresolvers import reverse

reverse(viewname='myview',    # remaining args optional
        urlconf=None,
        args=(1, 2),
        kwargs={'pk': foo.pk},
        current_app=None)
```

## redirect

```
from django.shortcuts import redirect
```

**redirect** (*to*[, *permanent=False*], **args*, ***kwargs*)

Returns an `HttpResponseRedirect` to the appropriate URL for the arguments passed.

The arguments could be:

- A model: the model's *get_absolute_url()* function will be called.
- A view name, possibly with arguments: *urlresolvers.reverse()* will be used to reverse-resolve the name.
- A URL, which will be used as-is for the redirect location.

By default issues a temporary redirect; pass `permanent=True` to issue a permanent redirect

### Examples

You can use the `redirect()` function in a number of ways.

1. By passing some object; that object's `get_absolute_url()` method will be called to figure out the redirect URL:

```
    def my_view(request):
        ...
        object = MyModel.objects.get(...)
        return redirect(object)
```

2. By passing the name of a view and optionally some positional or keyword arguments; the URL will be reverse resolved using the `reverse()` method:

```
    def my_view(request):
        ...
        return redirect('some-view-name', foo='bar')
```

3. By passing a hardcoded URL to redirect to:

```
    def my_view(request):
        ...
        return redirect('/some/url/')
```

This also works with full URLs:

```
    def my_view(request):
        ...
        return redirect('http://example.com/')
```

By default, `redirect()` returns a temporary redirect. All of the above forms accept a `permanent` argument; if set to `True` a permanent redirect will be returned:

```
def my_view(request):
    ...
    object = MyModel.objects.get(...)
    return redirect(object, permanent=True)
```

https://docs.djangoproject.com/en/stable/topics/http/urls/

```
from django.conf.urls.defaults import patterns, include, url
from django.contrib import admin admin.autodiscover()

urlpatterns = patterns('',
    (r'^polls/', include('polls.urls')),
    url(r'^admin/', include(admin.site.urls)),
)
urlpatterns = patterns('polls.views',
    (r'^', 'index'),
    (r'^(?P<poll_id>\d+)/', 'detail'),
    (r'^(?P<poll_id>\d+)/results/', 'results'),
    (r'^(?P<poll_id>\d+)/vote/', 'vote'),
    url(r'^(?P<poll_id>\d+)/foo/', 'fooview', name='app-viewname'),
)
```

Misc to file:

Avoid circular imports:

```
from django.db.models import get_model
MyModel = get_model('applabel', 'mymodelname'.lower())
```

CHAPTER 7

# Elasticsearch

## Documentation

- Elasticsearch Reference
- Elasticsearch Definitive Guide

Contents:

### Indices

#### Create an index

Create Index API

Example args:

```
{
    "settings" : {
        "number_of_shards" : 3,
        "number_of_replicas" : 2
    }
}

{
    "settings" : {
        "number_of_shards" : 1
    },
    "mappings" : {
        "type1" : {
            "_source" : { "enabled" : false },
            "properties" : {
                "field1" : { "type" : "string", "index" : "not_analyzed" }
            }
        }
    }
}
```

#### Index settings

TBD

**dynamic** Dynamic mapping Values `true`, `false`, `"strict"`.

## Mappings

Mappings API

Mappings reference

Intro to Mappings in the Guide

Types and Mappings in the Guide

Example mapping for a `tweet` doctype:

```
{
    "tweet" : {
        "properties" : {
            "message" : {
                "type" : "string",
                "store" : true,
                "index" : "analyzed",
                "null_value" : "na"
            },
            "user" : {
                "type" : "string",
                "index" : "not_analyzed",
                "norms" : {
                    "enabled" : false
                }
            },
            "postDate" : {"type" : "date"},
            "priority" : {"type" : "integer"},
            "rank" : {"type" : "float"}
        }
    }
}
```

### The string type

The string type is analyzed as full text by default.

String type reference includes all the possible attributes.

### The number type

Number type reference

### The date type

Date type reference

### The boolean type

Boolean type reference and it's `boolean` not `Boolean`.

### Multi fields

Multi fields reference

You can store the same source field in several index fields, analyzed differently.

### Object type

Object Type Ref

You can define a *field* to contain other fields.

### Root object type

Root object type ref

Root object in guide - better

The top-level doc type in a mapping is also an object type, but has some special characteristics. For example, you can set the default analyzers that fields without an explicit analyzer will use.

You can also turn off dynamic mapping for a doctype, though the Root object type ref doesn't mention this. See Dynamic mapping. You can even turn it back on for one field. Example:

```
{
    "my_type": {
        "dynamic":      "strict",
        "properties": {
            ...
            "stash": {
                "type": "object",
                "dynamic": True
        }
    }
}
```

## Analysis

Analysis reference

Analysis and Analyzers in the Guide

An analyzer consists of:

- Zero or more CharFilters
- One Tokenizer
- Zero or more TokenFilters

Analysis can be configured when creating an index with the top-level `analysis` key in the API argument. Example:

```
analysis :
    analyzer :
        standard :
            type : standard
            stopwords : [stop1, stop2]
        myAnalyzer1 :
            type : standard
```

```
            stopwords : [stop1, stop2, stop3]
            max_token_length : 500
        # configure a custom analyzer which is
        # exactly like the default standard analyzer
        myAnalyzer2 :
            tokenizer : standard
            filter : [standard, lowercase, stop]
    tokenizer :
        myTokenizer1 :
            type : standard
            max_token_length : 900
        myTokenizer2 :
            type : keyword
            buffer_size : 512
    filter :
        myTokenFilter1 :
            type : stop
            stopwords : [stop1, stop2, stop3, stop4]
        myTokenFilter2 :
            type : length
            min : 0
            max : 2000
```

### Built-in analyzers

Built-in analyzers in the Guide

### Custom analyzers

You can define custom analyzers.

Custom Analyzers in the Guide

Example:

```
analysis :
    analyzer :
        myAnalyzer2 :
            type : custom
            tokenizer : myTokenizer1
            filter : [myTokenFilter1, myTokenFilter2]
            char_filter : [my_html]
            position_offset_gap: 256
    tokenizer :
        myTokenizer1 :
            type : standard
            max_token_length : 900
    filter :
        myTokenFilter1 :
            type : stop
            stopwords : [stop1, stop2, stop3, stop4]
        myTokenFilter2 :
            type : length
            min : 0
            max : 2000
    char_filter :
          my_html :
```

```
            type : html_strip
            escaped_tags : [xxx, yyy]
            read_ahead : 1024
```

CHAPTER 8

# Elixir

## Basic types

Examples:

```
iex> 1          # integer
iex> 0x1F       # integer (31)
iex> 0x1010     # integer (10)
iex> 0o777      # integer (511)
iex> 1.0        # float
iex> 1.0e-10    # float
iex> true       # boolean
iex> false      # boolean
iex> :atom      # atom / symbol
iex> "elixir"   # string
iex> [1, 2, 3]  # list
iex> {1, 2, 3}  # tuple
iex> is_atom(false)     # true
iex> is_atom(:false)     # true
iex> is_boolean(true)   # true
iex> is_boolean(:true)   # true
iex> is_integer(1.0)    # false
iex> is_float("foo")
iex> is_number(1.0)     # true
```

## Arithmetic

Infix: `40 + 2` is 42. `/` on integers returns a float, e.g. `10 / 2` is 5.0:

```
10 / 2          5.0
div(10, 2)      5
div 10, 2       5
rem 10, 3       1
round(3.58)     4
trunc(3.58)     3
```

## Boolean expressions

`true == false` is `false`.

`true != false` is `true`.

You can also use `or`, `and`, and `not`. `or` and `and` are short-circuiting operators. All three of these require Boolean arguments.

Elixir also has `||`, `&&`, and `!`, which accept values of any type, and where any value except `false` and `nil` are truthy.

For comparison: `==`, `!=`, `===`, `!==`, `<=`, `>=`, `<`, and `>`. The only difference between `==` and `===` is that `1 == 1.0` is true but `1 === 1.0` is not true.

Elixir allows applying the comparison operators to values of any type or combination of types, for simplicity when doing things like sorting. There is an ordering among types, e.g. `<number> < <atom>`.

## Strings

Double quoted literals are strings (single quoted literals are char lists, not strings).

Use `<>` to concatenate, e.g. `"hello" <> " world"`

You can interpolate:

```
iex> "hellö #{:world}"
"hellö world"
```

The typical `\x` char codes work, e.g. `"hello\nworld"`

Internally strings are binary, sequences of bytes (UTF-8):

```
iex> String.length("hellö")
5
iex> is_binary("hellö")
true
iex> byte_size("hellö")
6
```

The String module has lots of useful methods like `upcase`.

You can print a string using `IO.puts/1`:

```
iex> IO.puts "hello\nworld"
hello
world
:ok
```

Note that the return value of `IO.puts` is `:ok`.

## Functions

Note: Functions in Elixir are identified by name and by number of arguments (i.e. arity). Therefore, is_boolean/1 identifies a function named is_boolean that takes 1 argument. is_boolean/2 identifies a different (nonexistent) function with the same name but different arity.:

```
iex> is_boolean(true)
true
iex> is_boolean(1)
false
```

You can get help on a function with `h` and its name/arity:

```
iex> h is_boolean
iex> h is_boolean/1
iex> h ==/2
```

*BUT* you can't call a function using its full name and arity, you have to leave off the arity:

```
iex> is_boolean/1(1)
** (SyntaxError) iex:2: syntax error before: '('
```

## Anonymous functions

Define anonymous functions with `fn`, `->`, and `end`:

```
iex> add = fn a, b -> a + b end
...
iex> is_function(add)
true
iex> is_function(add, 2)
true
iex> is_function(add, 1)
false
```

Anonymous functions require a dot `.` to invoke:

```
iex> add.(1, 2)
3
```

Anonymous functions are closures and can access variables that were in scope when they were defined.

Variables assigned inside a function do *not* affect the surrounding environment, though:

```
iex> x = 42
42
iex> (fn -> x = 0 end).()
0
iex> x
42
```

## Lists

Literal lists are written with square brackets. Values can be a mix of any types:

```
iex> length [1, 2, true, 3]
4
```

Lists are concatenated using `++/2` and can be "subtracted" using `--/2`:

```
iex> [1, 2, 3] ++ [4, 5, 6]
[1, 2, 3, 4, 5, 6]
iex> [1, true, 2, false, 3, true] -- [true, false]
[1, 2, 3, true]
```

The "head" of a list is like Lisp's `car` but is accessed using the `hd/1` function. Similarly, the "tail" is the `cdr` and you get it with `tl/1`:

```
iex> list = [1,2,3]
iex> hd(list)
1
iex> tl(list)
[2, 3]
```

You can add a new head to a list with `|`:

```
iex> [1 | [2, 3]]
[1, 2, 3]
```

Getting the head or the tail of an empty list is an error:

```
iex> hd []
** (ArgumentError) argument error
```

A list of small integers is printed by Elixir as a single-quoted "string" - but it's not a string, it's a list of chars:

```
iex> [11, 12, 13]
'\v\f\r'
iex> [104, 101, 108, 108, 111]
'hello'
```

## Introspection

Use `i/1` to get information about a value:

```
iex(2)> i 'hello'
Term
  'hello'
Data type
  List
Description
  This is a list of integers that is printed as a sequence of characters
  delimited by single quotes because all the integers in it represent valid
  ASCII characters. Conventionally, such lists of integers are referred to as
  "char lists" (more precisely, a char list is a list of Unicode codepoints,
  and ASCII is a subset of Unicode).
Raw representation
  [104, 101, 108, 108, 111]
Reference modules
  List
iex(3)> i "hello"
Term
  "hello"
Data type
  BitString
Byte size
  5
```

```
Description
  This is a string: a UTF-8 encoded binary. It's printed surrounded by
  "double quotes" because all UTF-8 encoded codepoints in it are printable.
Raw representation
  <<104, 101, 108, 108, 111>>
Reference modules
  String, :binary
```

## Tuples

Literal tuples are written with curly brackets `{1, :ok, true}`. Access any element with `elem/2` using 0-indexing, get the length with `tuple_size/1`, and return a new tuple with an element changed using `put_elem/3`:

```
iex> elem({:ok, "hello"})
"hello"
iex> tuple_size({:ok, "hello"})
2
iex> put_elem({:ok, "hello"}, 1, "world"})
{:ok, "world"}
```

CHAPTER 9

# Git

## Fetching

Update all local remote tracking branches from all remotes:

```
git fetch --all
```

Update all local remote tracking branches from origin:

```
git fetch origin
```

Update/create local branch origin/master from remote origin's branch master with default configuration:

```
git fetch origin master
```

Update/create local branch 'tmp' from remote origin's branch master (but only updates if fast-forward is possible):

```
get fetch origin master:tmp
```

Peek at an artitrary remote's branch by pulling it into a (temporary) local branch, then check its log. The temporary local branch will eventually be garbage collected:

```
git fetch git://git.kernel.org/pub/scm/git/git.git maint
git log FETCH_HEAD
```

## Branches and checkouts

Check out an existing branch:

```
git checkout <branch>
```

Create new branch:

```
git branch <branchname> [<start point>]
```

Create new branch and check it out in one command:

```
git checkout -b <newbranch> [<start point>]
```

## Import one repo into another with history

http://stackoverflow.com/questions/1683531/how-to-import-existing-git-repository-into-another

## Cleaning

Delete untracked files (be careful!):

```
git clean -fdx
```

Prune branches that have been merged and are no longer upstream:

```
http://devblog.springest.com/a-script-to-remove-old-git-branches
```

Prune branches that track remote branches that no longer exist (http://kparal.wordpress.com/2011/04/15/git-tip-of-the-day-pruning-stale-remote-tracking-branches/):

```
$ git remote prune origin --dry-run
$ git remote prune origin
```

## Pulls

Easier access to pull requests on Github. Add to config:

```
# This will make pull requests visible in your local repo
# with branch names like 'origin/pr/NNN'
# WARNING: This also breaks adding a new remote called "origin" manually
# because git thinks there already is one.  Comment this out temporarily
# in that case, unless you can think of a better solution.
[remote "pulls"]
    fetch = +refs/pull/*/head:refs/remotes/origin/pr/*
```

## Aliases

Handy aliases for config:

```
[alias]
lg = log --oneline --graph --date-order
lgd = log --oneline --graph --date-order --format=format:\"%ai %d %s\"

cb = checkout -b
cd = checkout develop
co = checkout

gd = !git fetch origin && git checkout develop && git pull origin develop
gm = !git fetch origin && git checkout master && git pull origin master

# push -u the current branch
pu = "!CURRENT=$(git symbolic-ref --short HEAD) && git push -u origin $CURRENT"

# push -f
pf = push -f
```

```
# Find the common ancestor of HEAD and develop and show a diff
# from that to HEAD
dd = "!git diff $(git merge-base develop HEAD)"
# Find the common ancestor of HEAD and master and show a diff
# from that to HEAD
dm = "!git diff $(git merge-base master HEAD)"

# These need 'hub' installed.
# Create pull request against develop.  Must pass issue number.
#pr = pull-request -b develop -i
# Create pull request against develop, not passing issue number:
pr = pull-request -b develop

# Checkout pull request
# Assume origin/pr/NN is pull request NN
# Need a bash function because we need to concatenate something to $1
#cpr = "!f() {set -x;git checkout origin/pr/$1; };f"
cpr = "!gitcpr"

# Undo any uncommited changes
abort = checkout -- .
```

## Submodules

This will typically fix things:

```
git submodule update --init --recursive
```

(and yes, you need –init every time)

Add a new submodule [http://git-scm.com/book/en/Git-Tools-Submodules]

```
$ git submodule add git@github.com:mozilla/basket-client basket-client
```

## Combining feature branches

Suppose you have branch A and branch B, which branched off of master at various times, and you want to create a branch C that contains the changes from both A & B.

According to Calvin: checkout the first branch, then git checkout -b BRANDNEWBRANCH. then rebase it on the second.

(SEE DIAGRAMS BELOW)

Example:

```
# Start from master
git checkout master
git pull [--rebase]

# Create the new branch from tip
git checkout -b C

# rebase A on master
git checkout A
```

```
git rebase -i master
# merge A into C
git checkout C
git merge A

# rebase B
git checkout B
git rebase -i master
# merge B into C
git checkout C
git merge B

# I think???
# Review before using, and verify the result
```

Combining git branches diagrams

Start:

```
o - o - o - o <--- master
 \   \
  \   o - o - o  <--- A
   o - o - o <--- B
```

Rebase A on master:

```
                 master
                 /
o - o - o - o - o - o - o <--- A
 \
  o - o - o <--- B
```

Create new branch N from master:

```
                master
                 /
o - o - o - o - o - o - o <--- A
 \               \
  \               N
   \
    o - o - o <--- B
```

Switch to N and merge A:

```
                master
                 /
o - o - o - o - o - o - o <--- A
 \               \
  \               o - o - o  <--- N  (includes A)
   \
    o - o - o <--- B
```

Rebase B on master:

```
                master
                 /
o - o - o - o - o - o - o - o <--- A
                |\
                |  o - o - o <--- N (includes A)
                \
                  o - o - o  <--- B
```

On N, merge B:

```
                master
                /
o - o - o - o - o - o - o - o <--- A
                |\
                | o - o - o -  o - o - o <--- N (includes A and B)
                \
                 o - o - o  <--- B
```

Delete A and B if desired.

## Undoing things

If you've committed some changes, then for some reason decide you didn't want to commit them yet - but still want the changes present in your local working directory - there are several options.

To get rid of the actual `commit` but keep all those changes staged:

```
$ git reset --soft HEAD~
```

To get rid of the actual commit and keep the changes, but not staged:

```
$ git reset HEAD~
```

And if you didn't want those changes at all - *WARNING* this will lose changes - gone:

```
$ git reset --hard HEAD~
```

CHAPTER 10

# Google APIs

Google APIs and Docs

Google Apps

Google Apps APIs

Google Drive API

Google Apps Script

## Google libraries

These are all for Python.

## Older libraries (Google Data)

## Newer libraries

OAuth 2.0

Google APIs client library for Python (beta)

CHAPTER 11

# Haskell

By analogy to Python...

## Imports

| Python | Haskell | Notes |
|---|---|---|
| from module import * | import module | Import everything from the module directly into the namespace |
| from module import a,b | import module (a,b) | Import selected items from the module directly into the namespace |
| import module | import qualified module | Make all `module.NAME` available in the name space |
| import module M = module del module | import qualified module as M | Give module a shorter alias |
| • | import module hiding (c) | Import everything from the module directly into the namespec, except selected items |

## Math

| Python | Haskell | Notes |
|---|---|---|
| a+b, a*b, a-b | a+b, a*b, a-b | |
| | a/b | Integer division giving a float |
| a and b, a or b, not a | a && b, a \|\| b, not a | |
| a == b, a != b | a==b, a /= b | |
| min(a,b), max(a,b) | min a b, max a b | |
| min([a,b,c]), max([a,b,c]), sum([a,b,c]) | minimum [a,b,c], maximum [a,b,c], sum [a,b,c] | |
| | | |
| | | |

## Expressions

| Python | Haskell | Notes |
|---|---|---|
| b if a else c | if a then b else c | Both are expressions, not statements |
| | | |
| | | |
| | | |

## Lists

| Python | Haskell | Notes |
|---|---|---|
| [1,2,3,4] | [1,2,3,4] | |
| [1, 'a', "foo"] | • | Haskell lists must be homogeneous |
| [1,2] + [3,4] | [1,2] ++ [3,4] | |
| [1] + [2,3] | 1:[2,3] | Haskell has more syntax for list expressions |
| | | |
| ['a','b','c'][1] | ['a', 'b', 'c'] !! 1 | Extract a list element |
| ['a','b','c'][:3] | take 3 ['a','b','c'] | First N elements |
| ['a','b','c'][3:] | drop 3 ['a','b','c'] | Drop N, take rest |
| ['a','b','c'][1:2] | take 2 . drop 1 ['a','b','c'] | Slices are uglier in Haskell |
| ['a','b','c'][0] | head ['a','b','c'] | First elt |
| ['a','b','c'][1:] | tail ['a','b','c'] | All but first elt |
| ['a','b','c'][-1] | last ['a','b','c'] | |
| ['a','b','c'][:-1] | init ['a','b','c'] | All but last elt |
| len(['a','b','c']) | length ['a','b','c'] | |
| not len(['a','b','c']) | null ['a','b','c'] | is list empty (but would probably just do bool(list) in Python) |
| reverse(['a','b','c']) | reverse ['a','b','c'] | |
| 'a' in ['a','b','c'] | 'a' *elem* ['a','b','c'] | |
| range(1,21) | [1..20] | Python counts to the last value before the 2nd parm, Haskell includes it |
| set(x) - set(y) | x \ y | Set difference for lists x and y - except Haskell isn't really, it just removes one value 'z' from x for each occurrence of 'z' in y. See Data.Set for real sets. |
| set(x) + set(y) | x *union* y | Haskell adds one occurrence of each element of y to x if x doesn't already have one |
| | | |
| | | |

## Functions

| Python | Haskell | Notes |
|---|---|---|
| **def doubleMe(x):** return x + x | doubleMe x = x + x | |
| **def doubleUs(x, y):** return x*2 + y*2 | doubleUs x y = x*2 + y+2 | |
| | | |
| | | |

CHAPTER 12

# i3

My i3 bindings

"Ŵ" is the "windows" key

Change what we see:

```
Ŵ-<NUMBER>: switch to workspace NUMBER on whatever monitor it's
            attached to.
Ŵ-Control-1: Only use built-in laptop display
Ŵ-Control-2: Use built-in laptop display, and external display
             positioned to its right
Ŵ-Control-3: Use built-in laptop display, and external display
             positioned above it
Ŵ-<n>:    Switch to workspace <n> (need not already exist)
          (if workspace <n> is on another screen, it'll switch
          that screen to workspace <n>, not your current screen)
Ŵ-<n> where <n> is the current workspace: Switch back to previous
          workspace (So you can just do Ŵ-1 (look at screen) Ŵ-1
          and be back where you started)
```

Focus:

```
Ŵ-j, Ŵ-<left>   left
Ŵ-k, Ŵ-<down>   down
Ŵ-l, Ŵ-<up>     up
Ŵ-;, Ŵ-<right>  right
```

Move things:

```
Ŵ-Control-<ARROW>: Move current workspace to another monitor.
Ŵ-Shift-Number: Move current window to another workspace
       (need not already exist)
Shift-<FOCUS COMMAND>: Move current window within workspace
```

Layouts:

```
Ŵ-e             default (splith/splitv), repeat to toggle
                splith/splitv
Ŵ-s             stacked
Ŵ-w             tabbed
Ŵ-f             fullscreen (toggle)
Ŵ-S-<spc>       float  (toggle)
Ŵ-mouse1-drag   move floating
```

```
Ŵ-h             Make the current window/container a horizontal
                split container. New windows opened when this
                container is focused will be created
                by splitting this container horizontally
                (side-by-side)
Ŵ-v             Like Ŵ-h, but vertical (one above another)

Ŵ-e             toggle between defaulting to horizontal and
                defaulting to vertical
```

Start/end things:

```
Ŵ-return: open new terminal
Ŵ-D: open dmenu at top to enter a command (output invisible,
     use to start new graphical programs)
Ŵ-S-q       kill window
```

Control I3:

```
Ŵ-S-c       reload I3 config
Ŵ-S-r       restart I3
Ŵ-S-e       kill I3 (logout)
```

Resizing:

```
Ŵ-mouse2-drag  stretch or shrink window
```

Screen capture:

```
<Printscreen> - capture whole screen
Shift-<Printscreen> - select a rectangle or window (?)
Control-<Printscreen> - capture currently focused window
```

CHAPTER 13

# IPv6

## Localhost

Localhost is *::1*

## Any addr

Any addr (equivalent of 0.0.0.0) is *::*

## With port

Adding ":<portnumber>" to an IPv6 address would be ambiguous. The solution is to put "[]" around the address part, e.g. *[::1]:8000*.

## URLs with IPv6 addresses

You need the *[]* here too, at least if you're using a hexadecimal IPv6 address. (Even without a port number.):

```
http://[fe80::b746:6473:e65f:5dd4]/foo/bar
http://[fe80::b746:6473:e65f:5dd4]:8000/foo/bar
```

## Ping

Use ping6:

```
ping6 ::1
```

## Django

To run a local dev server listening on any IPv6 address:

```
python manage.py runserver --ipv6 "[::]:8000"
```

It does NOT appear possible to have the dev server listen on both IPv4 and IPv6, at least not in Django 1.8. (But I'm sure you could put nginx in front to do that for you.)

## Private IPv6 network addresses

Try http://simpledns.com/private-ipv6.aspx to get a random private address range.

Example output:

```
Here is a unique private IPv6 address range generated just for you (refresh page to get another one)

Prefix/L:     fd
Global ID:    442da008f4
Subnet ID:    cf4f
Combined/CID:         fd44:2da0:08f4:cf4f::/64
IPv6 addresses:       fd44:2da0:08f4:cf4f:xxxx:xxxx:xxxx:xxxx

If you have multiple locations/sites/networks, you should assign each one a different "Subnet ID", bu
```

To add an address to your loopback interface:

```
sudo ifconfig lo add fd44:2da0:08f4:cf4f::1/64
```

## Find out your ipv6 address

In theory, run "ifconfig" and look for "inet6 addr". E.g.:

```
enp0s25   Link encap:Ethernet  HWaddr 54:ee:75:8b:03:99
          inet addr:172.20.1.110  Bcast:172.20.3.255  Mask:255.255.252.0
          inet6 addr: fe80::b746:6473:e65f:5dd4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

Try to find an "inet6 addr" that isn't "Scope:Link" or "Scope:Host"; you want "Scope:Global".

Better way - if you think you have real IPv6 connectivity to the internet, go to google.com and search for "what's my IP". If you're connecting to Google over IPv6, Google will tell you your IPv6 address (which might be a NATting router, I suppose - I don't know if IPv6 still does that?) But that won't help if you're just running IPv6 locally for testing.

CHAPTER 14

# Javascript

Contents:

## Javascript Builtins

### Arrays

Traditional literals, append, extend:

```
>> a = [1, 2, 3]                          # literal
>> console.log(a)
[1, 2, 3]
>> b = [4, 5]
>> Array.prototype.push.apply(a, b);     # 'extend'
>> console.log(a)
[1, 2, 3, 4, 5]
>> a.push(6)                              # 'append'
>> console.log(a)
[1, 2, 3, 4, 5, 6]
```

ES2015 literals, append, extend (Spread operator ):

```
>> a = [1, 2, 3]
>> b = [0, ...a]                          # literal incorporating another array
>> console.log(b)
[0, 1, 2, 3]
>> a.push(4)                              # append
>> console.log(a)
[1, 2, 3, 4]
>> console.log(b)
[0, 1, 2, 3]
>> a.push(...b)                          # extend
>> console.log(a)
[1, 2, 3, 4, 0, 1, 2, 3]
```

### Iterate over an array

Run code for each element (ES5+) (forEach):

```
arr.forEach(function(value, index, arr){}[, thisArg])
```

See if none, some, or all elements pass a test (ES5+) (every, some):

```
all_passed = arr.every(function(value, index, arr){ return bool;}[, thisArg])
some_passed = arr.some(function(value, index, arr){ return bool;}[, thisArg])
none_passed = !arr.some(...)
```

Create a new array by applying a function to each element of an existing array (ES5+) (map):

```
arr2 = arr1.map(function(value, index, arr){}[, thisArg])
```

Create a new array only including elements that pass a test (ES5+ (MDN):

```
arr2 = arr1.filter(function(value, index, arr){}[, thisArg])
```

## Iterate over a "dictionary" (JS object)

This will include enumerable properties of prototype classes (traditional) (for...in):

```
for (var key in object) {
   value = object[key]
   # ... do stuff ...
}
```

**Note:** For objects that you're just using as dictionaries, *for...in* is perfectly good. Just be careful not to use it on instances of anything more complicated.

For more complicated objects, here's how you get just properties that are directly on the object (not inherited from a prototype). Traditional:

```
for (var key in object) {
    if (object.hasOwnProperty(key)) {
        value = object[key]
        # ... do stuff ...
    }
}
```

Just properties directly on the object, with ES2015+ (keys):

```
var keys = object.keys();
for (var i = 0; i < keys.length; i++) {
    value = object[keys[i]];
    # ... do stuff ...
}
```

## Is a key in a dictionary?

dict.hasOwnProperty(key)

## Does an object have a key (possibly inherited)?

key in object

### Remove key from dictionary

delete dict["key"] delete dict.key

### String operations

String, endsWith, includes, indexOf, join, match, replace, search, split, startsWith, substr, substring

```
arr = str.split(sep=(str|regex)[, limit])
str = arr.join(sep)
index = str.indexOf(substr[, startIndex])  # returns -1 if not found
sub = str.substr(firstIndex[, length])  # firstIndex can be negative to count back from end
sub = str.substring(firstIndex[, lastIndex+1])
str2 = str.replace(regexp|substr, newSubStr|function)
bool = str.startsWith(str2)
bool = str.includes(str2)
bool = str.endsWith(str2)
[matchstr, groups...] = str.match(regexp)  # returns null if doesn't match
[matchstr, groups...] = str.search(regexp)  # returns null if doesn't match entire string
```

Contains: haystack.indexOf(needle) != -1

### Timer

setTimeout:

```
window.setTimeout(func, delay, param1, param2, ...);
```

All but func is optional. *delay* defaults to 0.

timerId = window.setTimeout(func, [delay, param1, param2, ...]); window.clearTimeout(timerId);

## Javascript Events

### Event object

- currentTarget: the element that has the handler attached that is currently running
- target: the element that received the event initially

## Promises

MDN docs

quoting from there:

A Promise is in one of these states:

- pending: initial state, not fulfilled or rejected.
- fulfilled: meaning that the operation completed successfully.
- rejected: meaning that the operation failed.

A promise that is not pending is "settled".

You can create a Promise with new, passing it an executor function:

```
let p = new Promise((resolve, reject) => {
     if (things go well) {
       resolve(resulting_value)
     } else {
        reject(error)
     }
})
```

At any time after the promise is created, you can call `then` on it and pass in a success handler, a failure handler, or both:

p.then(on_success) p.then(undefined, on_failure) p.then(on_success, on_failure)

If the promise is still pending, the appropriate handler will be called once it is settled. If it's settled already, the handler will be calleed immediately (or ASAP, anyway).

The `on_success` handler is called with the resolved value from the promise, while the `on_failure` handler is called with the error value from the promise.

`then()` returns a *new* promise, so you can chain calls:

```
p.then(on_success).then(another_on_success).then(a_third_on_success)
```

If the on_success handler returns a new promise, *that* promise will be the return value from `then()` (or an equivalent promise, anyway).

The handlers will be called in order. If one fails, then the promise returned by that then() call will be rejected. "Fail" can mean raises an exception.

Many async functions nowadays return promises.

## Pausing

Here's a way of doing something after a delay:

```
let p = new Promise((resolve) => {
    setTimeout(resolve, 100)
}).then(()=>{
    do stuff
})
```

## Some notes I made a while ago

Here are some examples:

```
// Return a new promise.
return new Promise(function(resolve, reject) {
   // do stuff.  if it works:
   resolve(result);
   // but if it fails:
   reject(Error(error text));
});

// USE a promise
promise.then(function(result) {
```

```
   // use result here
}, function(error) {
   // do something with error here
});

// CHAINING
// Note that 'then' returns a new promise

promise.then(function(val1) {
   return val1 + 2;
}).then(function(val2) {
   val2 is here val1 + 2!!!
});

// MORE CHAINING
promise.then(function(result){
  // do something, then return a new promise
  return promise2;
}).then(function(result2) {
  // this isn't invoked until promise2 resolves, and it gets promise2's result.
  ...
}
```

# Javascript Syntax

## Spread Operator

Spread operator

Given function and args:

```
function myFunction(x, y, z) { }
var args = [0, 1, 2];
```

Traditional:

```
myFunction.apply(null, args);
```

ES2015:

```
myFunction(...args);
myFunction(1, ...[2, 3]);
```

**Caution:** The ES2016 ... operator is NOT the same as * in a Python function call. ... basically splices the array it's applied to into the list at the place where it's used. It can be used repeatedly, and in any combination with other unnamed arguments. Python's * can only be used to extend the list of unnamed arguments at the end.

# DOM operations with JavaScript

## Finding elements by selector

https://developer.mozilla.org/en-US/docs/Web/API/Element/querySelectorAll

```
items = document.querySelectorAll('a.class-name')
```

Returns a NodeList.

## Iterating over elements

https://developer.mozilla.org/en-US/docs/Web/API/NodeList

**Note:** Although NodeList is not an Array, it is possible to iterate on it using forEach(). Several older browsers have not implemented this method yet.

```
// Newer browsers
items.forEach(function(item){ do something with item })

// Old (how old?) browsers
Array.prototype.forEach.call(items, function(item) { do something with item })
```

Here's a polyfill for ES5+ browsers from that NodeList.forEach page linked above:

```
if (window.NodeList && !NodeList.prototype.forEach) {
    NodeList.prototype.forEach = function (callback, thisArg) {
        thisArg = thisArg || window;
        for (var i = 0; i < this.length; i++) {
            callback.call(thisArg, this[i], i, this);
        }
    };
}
```

CHAPTER 15

# LXDE

Using LXDE desktop with i3 window manager.

There's a brief note here but this gives a little more depth.

- Install lxde:

```
sudo apt install lxde lxsession-logout
```

- Logout of the desktop
- Login again, this time choosing the LXDE desktop
- Create an executable shell script somewhere on your path, naming it "i3wm". It should run "i3". (I don't know why it doesn't work to just set the window manager to i3, but it doesn't. Maybe someday I'll take the time to debug that.)
- Edit ~/.config/lxsession/LXDE/desktop.conf. In the `[Session]` section, change windows_manager/command: `windows_manager/command=i3wm`
- In ~/.config/lxsession/LXDE/autostart, remove "@pcmanfm –desktop –profile LXDE", it interferers with i3.
- Logout and login again.
- If you like, bind a key in i3 to "lxsession-logout" and use that to logout. Exiting i3 will *not* log you out with this configuration. Or just use the menus in lxpanel to log out.

CHAPTER 16

# Logitech Harmony

Programming it using Linux

I have the Harmony One model (no longer produced).

Some of this information comes from http://ubuntuforums.org/showthread.php?t=781059, but I'm not using the GUI tool (congruity), just command line.

Help **:ref:'devices.rst'_** for finding devices.

## Install Concordance

Install the concordance tool.

I'm using v1.0 on Ubuntu 15.10 64-bit.

Briefly:

```
sudo apt-get install libusb-dev libzzip-dev
wget http://downloads.sourceforge.net/project/concordance/concordance/1.0/concordance-1.0.tar.bz2
tar xjf concordance-1.0.tar.bz2
cd concordance-1.0/libconcord
./configure
make
sudo make install
sudo ldconfig
cd ../concordance
./configure
make
sudo make install
```

## Arrange to run without needing sudo

- Run 'lsusb' to see what devices are already attached to your computer.
- Plug in your remote
- Run 'lsusb', looking for the device that wasn't there before. E.g. my Harmony One produced this:

```
    Bus 001 Device 021: ID 046d:c121 Logitech, Inc.
```

- Unplug the remote

- Now (using sudo as needed) create a new file, /etc/udev/rules.d/custom-concordance.rules, substituting in the values for your own remote:

```
ATTRS{idVendor}=="046d", ATTRS{idProduct}=="c121", MODE="666"
```

- Test
  - Plug the remote in
  - Run "concordance -i". It should print out information about the attached remote

## Programming the remote

- Make a directory to hold firmware and config backups, e.g. ~/Documents/LogitechHarmonyOne.
- *cd ~/Documents/LogitechHarmonyOne*
- Backup firmware if you haven't previously:

```
concordance --dump-firmware <filename> (default: firmware.EZUp)
```

- Backup config if you haven't previously:

```
concordance --dump-config <filename>  (default: config.EZHex)
```

- Go to http://members.harmonyremote.com/EasyZapper/ and log in (ignore the message about upgrading your software). If you don't already have an account, create one.
- Update your remote's configuration using the web site.
- When ready to update your remote:
  - Click "Update your remote". It'll prompt to connect your remote.
  - Connect the remote to the computer via USB.
  - Click "Next" on the web page.
  - Your browser will download a file named "Connectivity.EZHex". Save it to your LogitechHarmonyOne directory.
  - Go to your shell.
  - Run:

```
concordance Connectivity.EZHex
```

  - Go back to your browser.
  - Click "Next".
  - Wait... could be several minutes.
  - Eventually your browser will download a file named "Update.EZHex". Save it to your LogitechHarmonyOne directory.
  - Go to your shell.
  - Run this command. This will take quite a while (5 minutes?), but it will print progress status as it goes:

```
concordance Update.EZHex
```

  - When that's done, disconnect your remote and try it out.

# MPD

## Fixing on Ubuntu

On Ubuntu, mpd is installed broken. You'll have to go read this page and do some work to fix things so mpd will actually work on Ubuntu. Sigh.

**Important:** To play *anything*, you must first get it into the current playlist. Even if you just want to play one thing.

## Playlist commands

Commands that change the playlist:

**clear** Empty the playlist (stops playing if anything was playing)

**crop** Delete all playlist entries *except* the one currently playing

**del <number>** Delete one item from the playlist; 0 means the currently playing item, otherwise items are numbered starting at 1. (To see playlist with numbers, try `mpc playlist | cat -n`. There's probably a better way.)

**add <file>** Add an item from the database to the current playlist, at the end. <file> should be a path to the item, as shown by "mpc ls".

**insert <file>** Like `add`, only it puts the item immediately after the currently playing item, so it'll play next. If random mode is enabled, it'll still be played next.

**mv|move <from> <to>** Move item at position <from> to be at position <to>.

**save <name>** Save current playlist as database playlist with name <name>.

**load <name>** *Add* contents of database playlist named <name> to the current playlist.

**rm <name>** Delete database playlist named <name> from database.

## Playing things

Status:

**playlist [-f <format>]** List songs in current playlist. See "man mpc" for format string syntax.

**current** Show what's playing right now

Control:

**play [<position>]** Start playing the item at <position> in the playlist. Deafult is number 1.

**pause** Pause playing

**stop** Stop playing

**next** Stop playing the current entry from the current playlist and start playing the next one.

**prev** Reverse of `next`.

## Playing a playlist from the music database with mpc

Suppose you have a playlist in the database already - e.g. a file "/var/lib/mpd/playlists/WCPE.m3u" that you've created earlier.

Now you want to play that playlist.

```
$ mpc clear
$ mpc lsplaylists
WUNC
WCPE
Favorites
$ mpc load WCPE
loading: WCPE
$ mpc play
volume: 96%   repeat: on    random: off   single: off   consume: off
loading: WCPE
http://audio-ogg.ibiblio.org:8000/wcpe.ogg
[playing] #1/1   0:00/0:00 (0%)
volume: 96%   repeat: on    random: off   single: off   consume: off
```

CHAPTER 18

# MySQL with Django

Ubuntu: need to install (to use MySQL with Django):

```
sudo apt-get install mysql-client mysql-server libmysqlclient-dev
```

Django:

```
pip install mysqlclient

DATABASES['default'] = {
   'ENGINE': 'django.db.backends.mysql',
   'NAME': 'dbname',
   'USER': 'username',
}
```

## Using the client

Starting the client:

```
$ mysql --user=username [database]      # if user has no password
$ mysql --user=username --password [database]   # to be prompted for password
```

## Users and permissions

In the client:

```
mysql> SELECT user, host from mysql.user;                          # List existing users
mysql> CREATE USER 'username' IDENTIFIED BY 'plaintextpassword';       # Create user with password
mysql> CREATE USER 'username'@'localhost';   # no password, can only connect locally
mysql> SHOW DATABASES;
mysql> CREATE DATABASE databasename;
mysql> GRANT ALL ON databasename.* TO "username"@"hostname" IDENTIFIED BY "password";
mysql> FLUSH PRIVILEGES;
mysql> DROP DATABASE databasename;
mysql> DROP USER username;
mysql> EXIT
Bye
```

## Change user password

Note: default host is '%' which will not let you connect via unix socket, must set password for host 'localhost' to allow that:

```
mysql> update mysql.user set password=password('foo'),host='localhost' where user='poirier_wordpres';
mysql> flush privileges;
```

## Recover lost password

http://dev.mysql.com/doc/refman/5.5/en/resetting-permissions.html

C.5.4.1.3. Resetting the Root Password: Generic Instructions On any platform, you can set the new password using the mysql client:

```
Stop mysqld
Restart it with the --skip-grant-tables option. This enables anyone to connect without a password and

$ mysql
mysql> UPDATE mysql.user SET Password=PASSWORD('MyNewPass') WHERE User='root';
mysql> FLUSH PRIVILEGES;
mysql> EXIT

Stop the server
Restart it normally (without the --skip-grant-tables and --skip-networking options).
```

## Dumps

Make a dump:

```
mysqldump --single-transaction _dbname_ > dumpfile.sql
mysqldump --result-file=dumpfile.sql --single-transaction _dbname_
```

(Use `--single-transaction` to avoid locking the DB during the dump.)

Restore a dump:

```
mysql dbname < dumpfile.sql
```

## Create a new MySQL database

Step by step:

```
$ mysql -u root -p
<ENTER MYSQL ROOT PASSWORD>
mysql> create user 'ctsv2_TR'@'localhost';
mysql> create database ctsv2_TR;
mysql> grant all on ctsv2_TR.* to 'cstv2_TR'@'localhost';
mysql> flush privileges;
mysql> exit
Bye
```

CHAPTER 19

# NPM

Make *npm install* less noisy:

```
npm config set loglevel warn
```

or add this to ~/.npmrc:

```
loglevel=warn
```

source.

CHAPTER 20

# OpenSSL

Some of this from http://www.coresecuritypatterns.com/blogs/?p=763 and http://www.bogpeople.com/networking/openssl.shtml.

## End-user Functions

### Create key

Create a 2048-bit key pair:

```
openssl genrsa 2048 > myRSA-key.pem
openssl genrsa -out blah.key.pem
openssl genrsa -out blah.key.pem 2048
```

Create a password-protected 2048-bit key pair:

```
openssl genrsa 2048 -aes256 -out myRSA-key.pem
```

OpenSSL will prompt for the password to use. Algorithms: AES (aes128, aes192 aes256), DES/3DES (des, des3).

Remove passphrase from a key:

```
openssl rsa -in server.key -out server-without-passphrase.key
```

Extract public key:

```
openssl rsa -in blah.key.pem -out public.key -pubout
```

### Getting Certificates

Create Certificate Request and Unsigned Key:

```
openssl req -nodes -new -keyout blah.key.pem -out blah.csr.pem
```

More thorough example:

```
openssl req -new rsa:1024 -node -out myCSR.pem \
    -keyout myPrivCertkey.pem \
    -subj "/C=US/ST=MA/L=Burlington/CN=myHost.domain.com/emailAddress=user@example.com"
```

Create a self-signed certificate:

```
openssl req -nodes -x509 -newkey rsa:1024 -days 365 \
    -out mySelfSignedCert.pem  -set_serial 01 \
    -keyout myPrivServerKey.pem \
    -subj "/C=US/ST=MA/L=Burlington/CN=myHost.domain.com/emailAddress=user@example.com"
```

`-x509` identifies it as a self-signed certificate and `-set_serial` sets the serial number for the server certificate.

Create a single file that contains both private key and the self-signed certificate:

```
openssl req -x509 -nodes -days 365 -newkey rsa:1024 \
    -keyout myServerCert.pem -out myServerCert.pem \
    -subj "/C=US/ST=MA/L=Burlington/CN=myHost.domain.com/emailAddress=user@example.com"
```

Fingerprint for Unsigned Certificate:

```
openssl x509 -subject -dates -fingerprint -in blah.key.pem
```

Display Certificate Information:

```
openssl x509 -in blah.crt.pem -noout -text
```

Creating a PEM File for Servers:

```
cat blah.key.pem blah.crt.pem blah.dhp.pem > blah.pem
```

Download some server's certificate:

```
openssl s_client -connect www.example.com:443
```

(then hit ^C out of the interactive shell)

## Viewing Certificate Contents

X.509 certificates are usually stored in one of two formats. Most applications understand one or the other, some understand both:

- DER which is raw binary data.
- PEM which is a text-encoded format based on the Privacy-Enhanced Mail standard (see RFC1421). PEM-format certificates look something like this:

```
    -----BEGIN CERTIFICATE-----
    MIIBrjCCAWwCAQswCQYFKw4DAhsFADBTMQswCQYDVQQGEwJBVTETMBEGA1UECBMK
    U29tZS1TdGF0ZTEhMB8GA1UEChMYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMQww
    :
    :
    MQAwLgIVAJ4wtQsANPxHo7Q4IQZYsL12SKdbAhUAjJ9n38zxT+iai2164xS+LIfa
    C1Q=
    -----END CERTIFICATE-----

    OpenSSL uses the PEM format by default, but you can tell it to process DER format certificates..
```

The command to view an X.509 certificate is:

```
openssl x509 -in filename.cer -inform der -text
```

You can specifiy -inform pem if you want to look at a PEM-format certificate

## Convert Between Formats

If you have a PEM-format certificate which you want to convert into DER-format, you can use the command:

```
openssl x509 -in filename.pem -inform pem -out filename.cer -outform der
```

## PKCS12 files

PKCS12 files are a standard way of storing multiple keys and certificates in a single file. Think of it like a zip file for keys & certificates, which includes options to password protect etc.

Don't worry about this unless you need it because some application requires a PKCS12 file or you're given one that you need to get stuff out of.

Viewing PKCS12 Keystore Contents:

```
openssl pkcs12 -in filename.p12 -info
```

If you have two separate files containing your certificate and private key, both in PEM format, you can combine these into a single PKCS12 file using the command:

```
openssl pkcs12 -in cert.pem -inkey key.pem -export -out filename.p12
```

## Encrypting and signing things

Signing E-mails:

```
openssl smine -sign -in msg.txt -text -out msg.encrypted -signer blah.crt.pem -inkey blah.key.pem
```

Sign some text:

```
openssl dgst -sign private.key -out signature.asc
```

Verify signature:

```
if openssl dgst -verify public.key -signature signature.asc ; then echo GOOD; else echo BAD; fi
```

Encrypt and decrypt a single file:

```
openssl aes-128-cbc -salt -in file -out file.aes
openssl aes-128-cbc -d -salt -in file.aes -out file
```

Simple file encryption:

```
openssl enc -bf -A -in file_to_encrypt.txt
```

(password will be prompted)

Simple file decryption:

```
openssl enc -bf -d -A -in file_to_encrypt.txt
```

tar and encrypt a whole directory:

```
tar -cf - directory | openssl aes-128-cbc -salt -out directory.tar.aes
openssl aes-128-cbc -d -salt -in directory.tar.aes | tar -x
```

tar zip and encrypt a whole directory:

```
tar -zcf - directory | openssl aes-128-cbc -salt -out directory.tgz.aes
openssl aes-128-cbc -d -salt -in directory.tgz.aes | tar -xz
```

## Certificate Authority Functions

When setting up a new CA on a system, make sure index.txt and serial exist (empty and set to 01, respectively), and create directories private and newcert.

Edit openssl.cnf - change default_days, certificate and private_key, possibly key size (1024, 1280, 1536, 2048) to whatever is desired.

Create CA Certificate:

```
openssl req -new -x509 -keyout private/something-CA.key.pem \
-out ./something-CA.crt.pem -days 3650
```

Export CA Certificate in DER Format:

```
openssl x509 -in something-CA.crt.pem -outform der \
-out something-CA.crt
```

Revoke Certificate:

```
openssl ca -revoke blah.crt.pem
```

Generate Certificate Revokation List:

```
openssl ca -gencrl -out crl/example.com-CA.crl
```

Sign Certificate Request:

```
openssl ca -out blah.crt.pem -in blah.req.pem
```

Create Diffie-Hoffman Parameters for Current CA:

```
openssl dhparam -out example.com-CA.dhp.pem 1536
```

Creating Self-Signed Certificate from Generated Key:

```
openssl req -new -x509 -key blah.key.pem -out blah.crt.pem
```

Use only when you've no CA and will only be generating one key/certificate (useless for anything that requires signed certificates on both ends)

CHAPTER 21

# Org mode (Emacs)

http://orgmode.org/org.html See also http://orgmode.org/orgcard.pdf

| Binding | Operation |
|---|---|
| *Tasks* | |
| M-S-Ret | Add another item at same level |
| C-c C-t | Change TODO state |
| C-c / t | Show only uncompleted todos |
| *Agenda* | |
| C-c C-a n | View schedule and unscheduled tasks |
| b | Move backward (previous day) |
| f | Move forward (next day) |
| *Scheduling* | |
| C-c C-s | Schedule a task (set a date and optional time to do it) |
| C-u C-c C-s | Unschedule a task (remove schedule date/time) |

Keys outside org-mode:

```
Key     What
C-c g   my gtd file
C-c c t   Create task
C-c a X     Agenda view X

C-c l      org-store-link
C-c c      org-capture
C-c b      org-iswitchb (?)
```

Keys in org-mode file:

```
C-c C-x p   org-set-property
M-Return    org-meta-return - start new line with new heading at same level
M-S-right arrow  move current heading one deeper

C-c C-s     schedule task
C-c C-d     set task deadline
C-c C-q     org-set-tags-command add task tag - USE FOR CONTEXT
C-c / d     org-check-deadlines - spared tree with deadlines that are past-due or soon to be

<TAB>       org-cycle
S-<TAB>     org-global-cycle
```

Keys in agenda views:

```
TBD
```

CHAPTER 22

# Postfix

Retry sending all queued mail:

```
postfix flush
```

Delete all queued mail:

```
postsuper -d ALL deferred
```

CHAPTER 23

# Postgres

## Snippets

- In psql:

```
#  \d *pattern*
```

Display definition of table, view, or other relations with name matching the pattern

See http://jacobian.org/writing/pg-encoding-ubuntu/ to fix postgres to default to UTF-8 on ancient Ubuntus. (Destroys data.)

To set up your user on ubuntu to auth to local postgres automatically https://help.ubuntu.com/community/PostgreSQL

Check replication:

```
ON master, select * from pg_stat_replication;
(from http://www.dansketcher.com/2013/01/27/monitoring-postgresql-streaming-replication/)
```

## Environment Variables

The following environment variables can be used to select default connection parameter values, which will be used if no value is directly specified. These are useful to avoid hard-coding database connection information into simple client applications, for example.

PGHOST behaves the same as host connection parameter.

PGHOSTADDR behaves the same as hostaddr connection parameter. This can be set instead of or in addition to PGHOST to avoid DNS lookup overhead.

PGPORT behaves the same as port connection parameter.

PGDATABASE behaves the same as dbname connection parameter.

PGUSER behaves the same as user connection parameter.

PGPASSWORD behaves the same as password connection parameter. Use of this environment variable is not recommended for security reasons (some operating systems allow non-root users to see process environment variables via ps); instead consider using the ~/.pgpass file (see Section 30.14).

PGPASSFILE specifies the name of the password file to use for lookups. If not set, it defaults to ~/.pgpass (see Section 30.14).

## Dump Postgres table to a .CSV file

Started with this: http://stackoverflow.com/questions/1120109/export-postgres-table-to-csv-file-with-headings

Using COPY requires superuser but the error message helpfully tells you that you can use copy instead :-)

Using caktus' django template, something like:

```
$ fab -udpoirier production manage_run:dbshell
[huellavaliente.com:2222] out: venezuelasms_production=> \copy messagelog_message to '/tmp/messages.c
[huellavaliente.com:2222] out: venezuelasms_production=> \q

$ sftp -o Port=2222 dpoirier@huellavaliente.com
Connected to huellavaliente.com.
sftp> cd /tmp
sftp> ls
messages.csv
sftp> get messages.csv
Fetching /tmp/messages.csv to messages.csv
/tmp/messages.csv
sftp> ^D
```

## Postgres with non-privileged users

How do we do things on Postgres without giving superuser to the user that actually uses the database every day? The following assumes a Postgres superuser named 'master'. (Or the RDS 'master' user, who has most superuser privileges.)

In the examples below, for readability I'm omitting most of the common arguments to specify where the postgres server is, what the database name is, etc. You can set some environment variables to use as defaults for things:

```
export PGDATABASE=dbname
export PGHOST=xxxxxxxxx
export PGUSER=master
export PGPASSWORD=xxxxxxxxxx
```

## Create user

This is pretty standard. To create user `$username` with plain text password `$password`:

```
export PGUSER=master
export PGDATABASE=postgres
createuser -DERS $username
psql -c "ALTER USER $username WITH PASSWORD '$password';"
```

Yes, none of the options in `-DERS` are strictly required, but if you don't mention them explicitly, createuser asks you about them one at a time.

If not on RDS, for the user to actually do something useful like connect to postgres, you might also have to edit pg_hba.conf and add a line like:

```
local   <dbname>   <rolename>                                md5
```

to let it connect using host=" (unix domain socket) and provide a password to access <dbname>. You could also put "all" there to let it access any password it otherwise has auth for. E.g. to allow local connections via both unix socket and tcp connections to localhost:

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

## Create database

If you need a database owned by `$project_user`, you can:

- Create it as `$project_user` if that user has CREATEDB:

```
    export PGUSER=$project_user
    createdb --template=template0 $dbname
```

- Create it as a superuser and specify that the owner should be `$project_user`:

```
    export PGUSER=postgres
    createdb --template=template0 --owner=$project_user $dbname
```

- Create it as any other user, so long as the other user is a member, direct or indirect, of the `$project_user` role. That suggests that we could add `master` to that role... need to research that. I think we could do:

```
    export PGUSER=master
    psql -c "grant $project_user to master;" postgres
    createdb --template=template0 --owner=$project_user $dbname
```

  The question would be: Does master have enough privileges to grant itself membership in another role?

- Finally, you could create it as `master` when master is not a member of the project_user role. To do that, you'll need to create it as `master` and then modify the ownership and permissions:

```
    export PGUSER=master
    createdb --template=template0 $dbname
    psql -c "revoke all on database $dbname from public;"
    psql -c "grant all on database $dbname to master;"
    psql -c "grant all on database $dbname to $project_user;"
```

If you need to enable extensions etc, do that now (see below). When done, then:

```
psql -c "alter database $dbname owner to $project_user;"
```

A superuser could create the database already owned by a specific user, but RDS's master user cannot.

## PostGIS

To enable PostGIS, as the master user:

```
export PGUSER=master
psql -c "create extension postgis;"
psql -c "alter table spatial_ref_sys OWNER TO $project_user;"
```

where `$project_user` is the postgres user who will be using the database.

(Outside of RDS, only a superuser can use `create extension`; RDS has special handling for a whitelist of extensions.)

## Hstore

Hstore is simpler, but you still have to use the master user:

```
export PGUSER=master
psql -c "create extension hstore;"
```

## Grant read-only access to a database

Only let *readonly_user* do reads:

```
$ psql -c "GRANT CONNECT ON DATABASE $dbname TO $readonly_user;"
$ psql -c "GRANT SELECT ON ALL TABLES IN SCHEMA PUBLIC TO $readonly_user;" $dbname
```

## Restore a dump to a new database

Create the database as above, including changing ownership to the project user, and enabling any needed extensions. Then as the project user:

```
export PGUSER=$project_user
pg_restore --no-owner --no-acl --dbname=$dbname file.dump
```

Note that you might get some errors during the restore if it tries to create extensions that already exist and that kind of thing, but those are harmless. It does mean you can't use `--one-transaction` or `--exit-on-error` for the restore though, because they abort on the first error.

## Dump the database

This is pretty standard and can be done by the project user:

```
export PGUSER=$project_user
pg_dump --file=output.dump --format=custom $dbname
```

## Drop database

When it comes time to drop a database, only master has the permission, but master can only drop databases it owns, so it takes two steps. Also, you can't drop the database you're connected to, so you need to connect to a different database for the `dropdb`. The `postgres` database is as good as any:

```
export PGUSER=master PGDATABASE=postgres
psql -c "alter database $dbname owner to master;"
psql -c "drop database if exists $dbname;"
```

(Outside of RDS, a superuser can drop any database. A superuser still has to be connected to some other database when doing it, though.)

## Drop user

This is standard too. Just beware that you cannot drop a user if anything they own still exists, including things like permissions on databases.:

```
$ export PGUSER=master
$ dropuser $user
```

## Postgres on RDS

- Add `django-extensions` to the requirements and *django_extensions* to the *INSTALLED_APPS* so we can use the [sqldsn](http://django-extensions.readthedocs.org/en/latest/sqldsn.html) management command to get the exact Postgres settings we need to access the database from outside of Django. Here's how it works:

```
    manage.py [--settings=xxxx] sqldsn
```

CHAPTER 24

# Python

Contents:

## Asyncio

### What is it

`asyncio` is a library included in Python 3.5 that supports a programming model where sometimes, operations that would normally block the thread until some other event happened (like getting a response from a network connection) instead allow other code to run on that thread while waiting.

asyncio takes a very, very explicit approach to asynchronous programming: only code written in methods flagged as `async` can call any code in an asynchronous way.

Which creates a chicken/egg problem: your *async* methods can only be called by other *async* methods, so how do you call the first one?

The answer: you don't. What you have to do instead is turn over control of the thread to an event loop, after arranging for the loop to (sooner or later) invoke your async code.

Then once you start the loop running, it can invoke the async code.

### What good is it

Note first that you can use threads to accomplish the same things as asyncio in most cases, with better performance. So what good is asyncio?

For one thing, it leads to more straightforward code than managing multiple threads, protecting data structures from concurrent access, etc. There's only one thread and no preemptive multitasking.

If you want to play with async programming in Python, asyncio looks easier to work with and understand than Twisted, but that's not a very practical reason.

More significantly, threads won't scale as well if you need to wait for many, many things at the same time - asyncio might be somewhat slower, but might be the only way that some tasks can be run at all. Each thread can take 50K of memory, while a coroutine might take only 3K.

## Event loops

Async code can only run inside an *event loop*. The event loop is the driver code that manages the cooperative multitasking.

(I think) a typical pattern would be to get or create an event loop, set up some things to be run by it, then start the event loop running and have it run until the program is finished.

If it's useful for some reason, you can create multiple threads and run different event loops in each of them. For example, Django uses the main thread to wait for incoming requests, so we can't run an asyncio event loop there, but we can start a separate worker thread for our event loop.

## Coroutines

coroutines

- Python distinguishes between a *coroutine function* and a *coroutine object*
- Write a coroutine function by putting `async` in front of the `def`.
- Only a coroutine function can use `await`, non-coroutine functions cannot.
- Calling a *coroutine function* does *not* execute it, but rather returns a *coroutine object*. (This is analogous to generator functions - calling them doesn't execute the function, it returns a generator object, which we then use later.)
- To execute a *coroutine object*, either:
  - use it in an expression with `await` in front of it, or
  - schedule it with ensure_future() or create_task().

Example with await:

```
async def coro_function():
    return 2 + 2

coro = coro_function()
# not executed yet; coro is a coroutine, not 4

print(await coro)
# prints "4"
```

Example of scheduling it:

```
async def coro_function(hostname):
    conn = await .... connect async to hostname somehow...

coro = coro_function("example.com")
asyncio.ensure_future(coro)
```

Of course, usually you wouldn't split it onto two lines with a temp variable:

```
asyncio.ensure_future(coro_function("example.com"))
```

or:

```
asyncio.get_event_loop().create_task(coro_function("example.com"))
```

## Futures

A future is an object that represents something uncompleted. It makes it easy for code in one place to indicate when the work is done, and optionally what the result was, and for code elsewhere that was interested in it to find out about it.

In other words, you can use future objects to manage synchronization more explicitly.

Create one on the fly by calling loop.create_future():

```
future = loop.create_future()
```

Arrange for something to be called when the future becomes done:

```
future.add_done_callback(fn)
```

You can add lots of callbacks. They'll all be called (one at a time).

The callback receives the future object as an argument. Use functools.partial as usual if you want to pass other arguments.

When the future is done, mark it done and set its result:

```
future.set_result(value)
```

The callbacks can call *future.result()* to find out what the result was if they care.

## Tasks

A Task is a way to arrange for a coroutine to be executed by an event loop, while also providing the caller a way to find out what the result was.

A task is automatically scheduled for execution when it is created.

There are two ways to do this, which seem equivalent as far as I can tell:

```
future = loop.create_task(coroutine)
future = asyncio.ensure_future(coroutine[, loop=loop])
```

Now you can add callbacks if you want:

```
future.add_done_callback(fn1)
```

Also, if the loop isn't already running and you just want to run the loop for this one thing, you can now:

```
loop.run_until_complete(future)
```

## Awaitables

Coroutine *objects* and future *objects* are called *awaitables* - either can be used with `await`.

Note: You can only invoke an awaitable *once*; after that, it's completed, done, it runs no more.

## Event loops

### Creating/getting one

- To get the current thread's default event loop object, call asyncio.get_event_loop()

- *get_event_loop* will *not* create an event loop object unless you're on the main thread, and otherwise will raise an exception if the current thread doesn't have a default loop set.
- To create a new event loop: new_event_loop()
- To make a loop the default loop for the current thread: set_event_loop(loop)

So, to use an event loop in the main thread, you can just do:

```
loop = asyncio.get_event_loop()
# use loop....
```

But to run an event loop in another thread, you would do something like:

```
loop = asyncio.new_event_loop()
asyncio.set_event_loop(loop)
# use loop...
```

You don't have to set your loop as the thread's default, though, if you're willing to pass your loop object to all the APIs that otherwise use the default loop. But that's a pain.

### Running a loop

If you want a long-running loop that keeps responding to events until it's told to stop, use loop.run_forever().

If you want to compute some finite work using coroutines and then stop, use loop.run_until_complete(<future or coroutine>).

### Stopping a loop

Use loop.stop().

### Getting a loop to call a synchronous callable

By *a synchronous callable*, I mean a callable that is *not* an *awaitable* as described above.

This is more like Javascript's callback-style async programming than in the spirit of Python's coroutines, but sometimes you need it.

To call the callable as soon as possible, use loop.call_soon(callback). If you want to pass args to the callable, use functools.partial:

```
loop.call_soon(functools.partial(callable, arg1, arg2))
```

To delay for *N* seconds before calling it, use loop.call_later(delay, callable).

To schedule a callback from a different thread, the AbstractEventLoop.call_soon_threadsafe() method should be used. Example:

```
loop.call_soon_threadsafe(callback, *args)
```

### Getting a loop to call an awaitable

Use asyncio.ensure_future(awaitable, *, loop=None).

Or *loop.run_until_complete*, but as noted above, that just runs the loop as long as it takes to complete the awaitable.

If you're doing this from another thread, then you need to use a different method, asyncio.run_coroutine_threadsafe(coro, loop):

```
future = asyncio.run_coroutine_threadsafe(coroutine, loop)
```

### Running blocking code in another thread

If you need to call some blocking code from a coroutine, and don't want to block the whole thread, you can make it run in another thread using coroutine AbstractEventLoop.run_in_executor(executor, func, *args):

```
fn = functools.partial(method, *args)
result = await loop.run_in_executor(None, fn)
```

### Sleep

Calling asyncio.sleep(seconds) does not sleep; it returns a *coroutine object*. When you *execute* it by invoking it with `await` etc, it will complete after <*seconds*> seconds. So, mostly you'd do:

```
await asyncio.sleep(10)  # pause 10 seconds
```

## Mock

- http://www.voidspace.org.uk/python/mock/mock.html
- https://docs.python.org/3/library/unittest.mock.html

### Mock a function being called in another module

Function being called from another module (medley/photos/tasks/galleryimport.py):

```
from urllib2 import urlopen
```

our test.py:

```
with mock.patch('medley.photos.tasks.galleryimport.urlopen') as urlopen:
    urlopen.side_effect = ValueError("Deliberate exception for testing")
```

### Mock a method on a class

Use `mock.patch('python.packagepath.ClassName.method_name`:

```
# medley/photos/models.py
class MedleyPhotoManager(...):
    def build_from_url(...):
```

Test code:

```
with mock.patch('medley.photos.models.MedleyPhotoManager.build_from_url') as build_from_url:
    build_from_url.return_value = None
```

### Replace something with an existing object or literal

Use `mock.patch` and pass `new=`:

```
with mock.patch("medley.photos.tasks.galleryimport.cache", new=get_cache('locmem://')):
    ...
with mock.patch(target='medley.photos.tasks.galleryimport.MAX_IMPORTS', new=2):
    ...
```

### Mock an attribute on an object we have a reference to

Use `mock.patch.object(obj, 'attrname')`:

**with mock.patch.object(obj, 'attrname') as foo:** ...

### Data on the mock

Attributes on mock objects:

```
obj.call_count  # number of times it was called
obj.called == obj.call_count > 0
obj.call_args_list  # a list of (args,kwargs), one for each call
obj.call_args  # obj.call_args_list[-1] (args,kwargs from last call)
obj.return_value  # set to what it should return
obj.side_effect  # set to an exception class or instance that should be raised when its called
obj.assert_called() # doesn't work with autospec=True? just assert obj.called
obj.assert_called_with(*args, **kwargs)  # last call was with (*args, **kwargs)
```

## Python Packaging

The authoritative docs.

Example setup.py:

```
# Always prefer setuptools over distutils
from setuptools import setup, find_packages
# To use a consistent encoding
from codecs import open
from os import path

here = path.abspath(path.dirname(__file__))

setup(
    name='ursonos',
    version='0.0.1',
    packages=find_packages(),
    url='',
    license='',
    author='poirier',
    author_email='dan@poirier.us',
    description='Urwid application to control Sonos',
    install_requires=[
        'soco',
        'urwid'
```

```
    ]
)
```

To include non-Python files in the packaging, create a MANIFEST.in file. Example:

```
include path/to/*.conf
```

adds the files matching path/to/*.conf. Another:

```
recursive-include subdir/path *.txt *.rst
```

adds all files matching *.txt or *.rst that are anywhere under subdir/path. Finally:

```
prune examples/sample?/build
```

should be obvious.

### Wheels

To build a Universal Wheel:

```
python setup.py bdist_wheel --universal
```

You can also permanently set the `--universal` flag in "setup.cfg" (e.g., see sampleproject/setup.cfg)

```
[bdist_wheel]
universal=1
```

Only use the `--universal` setting, if:

1. Your project runs on Python 2 and 3 with no changes (i.e. it does not require 2to3).
2. Your project does not have any C extensions.

### Upload

The docs recommend `twine` but I haven't had much luck getting it working, so...

```
python setup.py sdist bdist_wheel upload
```

## Pip

Based on A Better Pip Workflow, from a hackernews comment by blaze33:

To keep your file structure with commands and nicely separate your dependencies from your dependencies' dependencies:

```
pip freeze -r requirements-to-freeze.txt > requirements.txt
```

instead of just:

```
pip freeze > requirements.txt
```

**Danger:** beware of git urls being replaced by egg names in the process.

# Timezones in Python

## Key points

- Install the pytz package to provide actual time zones. Python doesn't come with them.
- There are two kinds of datetime objects in Python, and you need to always know which you're working with:
- naive - has no timezone info. (datetime.tzinfo is None)
- aware - has timezone info (datetime.tzinfo is not None)
- There will always be some things you want to do with datetimes that are just inherently ambiguous. Get used to it.

## Some use cases

Start by importing useful modules:

```
import datetime, time, pytz
```

### Given a formatted time string with timezone, end up with a datetime object

Suppose we have RFC 2822 format:

```
s = "Tue, 3 July 2012 14:11:03 -0400"
```

It would be nice if strptime could just parse this and give you an aware datetime object, but no such luck:

```
>>> fmt = "%a, %d %B %Y %H:%M:%S %Z"
>>> datetime.datetime.strptime(s, fmt)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/_strptime.py", line 32
    (data_string, format))
ValueError: time data 'Tue, 3 July 2012 14:11:03 -0400' does not match format '%a, %d %B %Y %H:%M:%S
>>> fmt = "%a, %d %B %Y %H:%M:%S %z"
>>> datetime.datetime.strptime(s, fmt)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "/System/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/_strptime.py", line 31
    (bad_directive, format))
ValueError: 'z' is a bad directive in format '%a, %d %B %Y %H:%M:%S %z'
```

So, we have to parse it without the time zone:

```
>>> fmt = "%a, %d %B %Y %H:%M:%S"
>>> dt = datetime.datetime.strptime(s[:-6], fmt)
>>> dt
datetime.datetime(2012, 7, 3, 14, 11, 3)
```

That is assuming we know exactly how long the timezone string was, but we might not. Try again:

```
>>> last_space = s.rindex(' ')
>>> last_space
25
>>> datetime.datetime.strptime(s[:last_space], fmt)
```

Now, we need to figure out what that timezone string means. Pick it out:

```
>>> tzs = s[last_space+1:]
>>> tzs
'-0400'
```

We could have a timezone name or offset, but let's assume the offset for now. RFC 2282 says this is in the format [+-]HHMM:

```
>>> sign = 1
>>> if tzs.startswith("-"):
...     sign = -1
...     tzs = tzs[1:]
... elif tzs.startswith("+"):
...     tzs = tzs[1:]
...
>>> tzs
'0400'
>>> sign
-1
```

Now compute the offset:

```
>>> minutes = int(tzs[0:2])*60 + int(tzs[2:4])
>>> minutes *= sign
>>> minutes
-240
```

Unfortunately, we can't just plug that offset into our datetime. To create an aware object, Python wants a tzinfo object that has more information about the timezone than just the offset from UTC at this particular moment.

So here's one of the problems - we don't KNOW what timezone this date/time is from, we only know the current offset from UTC.

So, the best we can do is to figure out the corresponding time in UTC, then create an aware object in UTC. We know this time is 240 minutes less than the corresponding UTC time, so:

```
>>> import time
>>> time_seconds = time.mktime(dt.timetuple())
>>> time_seconds -= 60*minutes
>>> utc_time = datetime.datetime.fromtimestamp(time_seconds, pytz.utc)
>>> utc_time
datetime.datetime(2012, 7, 3, 22, 11, 3, tzinfo=<UTC>)
```

And there we have it, an aware datetime object for that moment in time.

## Virtualenv

Using Python virtual environments

Assumes virtualenv and virtualenvwrapper is installed

pip install into site.USER_BASE (kind of a special virtualenv):

```
PIP_REQUIRE_VIRTUALENV= pip install --user ...
```

http://www.doughellmann.com/docs/virtualenvwrapper:

```
sudo /usr/bin/easy_install pip
sudo /usr/local/bin/pip install virtualenvwrapper
```

Then add to .bashrc:

```
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
```

creates and activates a new env, //envname//:

```
mkvirtualenv //envname//
```

switch to //envname2//:

```
workon //envname2//
```

no longer working with a virtual env:

```
deactivate
```

List all of the environments:

```
lsvirtualenv
```

Show the details for a single virtualenv:

```
showvirtualenv
```

delete a virtual env (must deactivate first):

```
rmvirtualenv
```

# XML in Python

## Formatting an etree Element

Like this:

```
def format_element(element):
    """
    Given an etree Element object, return a string with the contents
    formatted as an XML file.
    """
    tree = ElementTree(element)
    bytes = StringIO()
    tree.write(bytes, encoding='UTF-8')
    return bytes.getvalue().decode('utf-8')
```

# Most minimal logging

Python doesn't provide any logging handlers by default, resulting in not seeing anything but an error from the logging package itself... Add a handler to the root logger so we can see the actual errors.:

```
import logging
logging.getLogger('').addHandler(logging.StreamHandler())
```

## Binary data to file-like object (readable)

f = io.BytesIO(binary_data) # Python 2.7 and 3

## Join list items that are not None

Special case use of *filter*:

```
join(', ', filter(None, the_list))
```

## Declare file encoding

Top of .py file:

```
# -*- coding: UTF-8 -*-
# vim:fileencoding=UTF-8
```

## Quick 'n' easy static web server

- Change to the top level directory of the static files
- Run `python -m http.server`

## Python snippet copy logging to stdout

add to top:

```
import logging

handler = logging.StreamHandler()
root_logger = logging.getLogger('')
root_logger.setLevel(logging.DEBUG)
root_logger.addHandler(handler)
```

## Warnings

Hiding python warnings. e.g. Deprecations:

```
python -Wignore::DeprecationWarning ...
```

## Pylint

Running it:

```
python /usr/bin/pylint --rcfile=.pylintrc -f colorized aremind.apps.patients | less -r
```

Disabling a warning in code:

```
# pylint: disable-msg=E1101
```

CHAPTER 25

# Raspberry Pi

## USB drive with Raspberry PI

Here are detailed steps to get a Raspberry PI running entirely off a USB drive (after booting from its SD card).

Some sources:

- **Move root to USB drive:** http://magnatecha.com/using-a-usb-drive-as-os-root-on-a-raspberry-pi/ http://elinux.org/Transfer_system_disk_from_SD_card_to_hard_disk
- **UUIDs:** http://liquidat.wordpress.com/2013/03/13/uuids-and-linux-everything-you-ever-need-to-know/

Detailed log:

- Start with 4 GB SD card
- Copy 2013-07-26-wheezy-raspbian.img onto it:

  sudo dd if=path/2013-07-26-wheezy-raspbian.img of=/dev/mmcblk0 bs=1M
- Partition USB drive:
  - Partition 1: system root: 16 GB
  - Partition 2: Swap: 1 GB
  - Partition 3: /usr/local: rest of drive
- **Format the partitions too (ext4)**
  - sudo mkfs.ext3 -q -m 0 -L ROOT /dev/sdb1
  - sudo mkswap -q -L SWAP120 /dev/sdb2
  - sudo mkfs.ext3 -q -m 0 -L LOCAL /dev/sdb3
- Boot off 4 GB SD card
- ssh to its IP address (get from router or whatever):

```
ssh pi@[IP ADDRESS]
password: raspberry
```

- Copy current root partition to USB drive (see blog post mentioned above to make sure you're using the right partitions):

```
sudo dd if=/dev/mmcblk0p2 of=/dev/sda1 bs=4M
```

- Resize:

```
sudo e2fsck -f /dev/sda1
sudo resize2fs /dev/sda1
```

- See what UUID it got:

```
$ sudo blkid /dev/sda1
/dev/sda1: UUID="9c7e2035-df9b-490b-977b-d60f2170889d" TYPE="ext4"
```

- Mount:

```
sudo mount /dev/sda1 /mnt
```

- Edit config files and change current root partition to the new root UUID in fstab, and /dev/sda1 in cmdline.txt (cmdline.txt doesn't support UUID, darn)
  - *vi /mnt/etc/fstab*:

```
UUID=9c7e2035-df9b-490b-977b-d60f2170889d    /    ext4    defaults,noatime,async  0       1
```

  - *vi /boot/cmdline.txt* (http://elinux.org/RPi_cmdline.txt)
- Umount /mnt:

```
sudo umount /mnt
```

- Reboot and check things out

Swap:

- Format swap partition:

```
    $ sudo mkswap -L swappart /dev/sda2
    Setting up swapspace version 1, size = 1048572 KiB
LABEL=swappart, UUID=a471af01-938b-4ad0-8653-dafe211cdfba
```

- Make sure it'll work:

```
sudo swapon -U a471af01-938b-4ad0-8653-dafe211cdfba
free -h
```

- Edit /etc/fstab, add at end:

```
UUID=a471af01-938b-4ad0-8653-dafe211cdfba swap swap defaults 0 0
```

- Remove the default 100M swap file:

```
sudo apt-get purge dphys-swapfile
```

- reboot and check swap space again, should be 1 G (not 1.1 G)

Now move /usr/local to the USB drive:

- Format partition:

```
sudo mkfs.ext4 -L usr.local /dev/sda3
```

- Find out its UUID:

```
$ blkid /dev/sda3
/dev/sda3: LABEL="usr.local" UUID="3c6e0024-d0e4-412e-a4ab-35d7c9027070" TYPE="ext4"
```

- Mount it temporarily on /mnt:

```
sudo mount UUID="3c6e0024-d0e4-412e-a4ab-35d7c9027070" /mnt
```

- Copy the current /usr/local over there:

```
(cd /usr/local;sudo tar cf - .) | ( cd /mnt;sudo tar xf -)
```

- Umount:

```
sudo umount /mnt
```

- Remove files from /usr/local:

```
sudo rm -rf /usr/local/*
```

- Edit /etc/fstab to mount /dev/sda3 on /usr/local at boot:

```
UUID=3c6e0024-d0e4-412e-a4ab-35d7c9027070       /usr/local      ext4    defaults,noatime
```

- See if that works:

```
sudo mount -a
df -h
```

- reboot and make sure it works again

## Crashplan on Raspberry PI

**Good blog post: http://www.bionoren.com/blog/2013/02/raspberry-pi-crashplan/** READ THE COMMENTS!

I made a number of changes, e.g.:

Use Oracle Java: http://www.raspberrypi.org/archives/4920

> sudo apt-get update && sudo apt-get install oracle-java7-jdk

See first my SimpleNote: "USB drive with Raspberry PI", then continue here

- Install oracle Java:

  > sudo apt-get update && sudo apt-get install oracle-java7-jdk

- Download [crashplan](http://www.crashplan.com/consumer/download.html?os=Linux) and extract it
- Run the crashplan installer (Crashplan-install/install.sh). I'll assume that you installed to the default /usr/local/crashplan/. If you get an error "The current version of Java (1.8) is incompatible with CrashPlan", edit install.sh and change OKJAVA="1.5 1.6 1.7" to OKJAVA="1.5 1.6 1.7 1.8"
- (It'll say it started, but it'll crash immediately so don't worry about it yet.)
- download some fixed/extra files:

  > wget http://www.jonrogers.co.uk/wp-content/uploads/2012/05/libjtux.so wget http://www.jonrogers.co.uk/wp-content/uploads/2012/05/libmd5.so

- copy:

  > sudo cp libjtux.so libmd5.so /usr/local/crashplan

- Install libjna-java:

  > sudo apt-get install libjna-java

- Hack:

edit /usr/local/crashplan/bin/CrashPlanEngine and find the line that *begins* with FULL_CP= (its around the start case) add /usr/share/java/jna.jar: to the begining of the string. This fixes the cryptic "backup disabled — not available" error. I ended up with:

> FULL_CP="/usr/share/java/jna.jar:$TARGETDIR/lib/com.backup42.desktop.jar:$TARGETDIR/lang"

- This will backup by default to /usr/local/var/crashplan. We'll fix that next. But for now, reboot and make sure crashplan is now running:

  > sudo reboot ... wait, log back in ... /usr/local/crashplan/bin/CrashPlanEngine status

Nope, it's stopped. Did I miss a step?

From a crashplan log (/usr/local/crashplan/log/engine_output.log):

> Exiting!!! java.lang.UnsatisfiedLinkError: /usr/local/crashplan/libjtux.so: /usr/local/crashplan/libjtux.so: cannot open shared object file: No such file or directory (Possible cause: can't load IA 32-bit .so on a ARM-bit platform).

But that could well be from the initial attempt to start, before we replaced the .so file.

Deleted log file, tried "sudo service crashplan start" again, checked status again... it's running. So maybe it's just not being started at boot? The installer implied it would be...

There's a rc2.d/S99crashplan which is promising. Try booting again, and wait a sec?

Nope, after several minute, still stopped.

Try this:

> sudo update-rc.d crashplan defaults reboot

Yes! It's running.

now continue with my "Backing up to ReadyNAS using Crashplan on RaspberryPI" simplenote.

## Backing up to ReadyNAS using Crashplan on RaspberryPI

This assumes the setup from "USB drive with RaspberryPI" and "Crashplan on Raspberry PI" have already been done.

We have Crashplan running on our Raspberry Pi. We want it to just backup to a share on our ReadyNAS with no complicated configuration within Crashplan. It looks like a good way to do that might be to just make /usr/local/var/crashplan be a mount from the NAS. Let's try that.

Get NFS client working on Raspberry Pi (https://help.ubuntu.com/community/SettingUpNFSHowTo#NFS_Client)

- *sudo apt-get install rpcbind nfs-common*
- *sudo update-rc.d rpcbind enable*
- *sudo update-rc.d nfs-common enable*
- *sudo reboot* (to make sure it's all started)

Enable a share for this on the ReadyNAS:

- Login to ReadyNAS admin web
- Go to Services/Standard File Protocols
- Enable NFS and click Apply if it wasn't enabled already
- go to Shares/Add shares
- enter name=pibackup and a suitable description, turn off Public Access

- Click Apply
- Go to Shares/Share listing
- Next to pibackup, click the NFS icon
- Under Write-enabled hosts, add the Pi's IP address

Mount it on the Pi for testing:

- *sudo showmount -e 10.28.4.2*
- *sudo mkdir /mnt/pibackup*
- *sudo mount 10.28.4.2:/c/pibackup /mnt/pibackup*
- *sudo umount /mnt/pibackup*

Now arrange to mount it where crashplan is going to do backups:

- *sudo service crashplan stop* (for safety)
- *sudo -e /etc/fstab*

    10.28.4.2:/c/pibackup /usr/local/var/crashplan nfs soft,intr,rsize=8192,wsize=8192

- *sudo service crashplan start*

Configure Crashplan and try it out. First, we'll do a small backup from here to Crashplan servers, just for sanity.

- Arrange to run the desktop client connected to the Raspberry Pi server (see Crashplan on Headless Boxes)
- Login with your Crashplan account if needed (first time).
- In Settings, give this computer a unique name (poirpi?)
- In Backup/files, select some small set of files to backup - maybe */etc* (after showing hidden files)
- In Backup/Destinations, next to CrashPlan Central, click Start Backup. (Enter Archive encryption key if prompted)
- Wait until the backup is complete
- Exit the desktop

Now, let's get another computer to backup to here (which is what we really want).

- Run CrashPlanDesktop locally on the source computer
- Go to Settings/Backup tab
- Find a small backup set and select it.
- under Destinations, click Change...
- Find the new Raspberry Pi and click its checkbox
- Click OK
- Click Save
- Go to Backup
- Check the status of the backup of that backup set on the new computer.

CHAPTER 26

# reStructuredText

REST reStructuredText notes (see also Sphinx notes)

- rst tutorial
- rst primer
- rst markup
- autodoc

Example:

```
file1.rst
---------

Contents:

.. toctree::
   :maxdepth: 2

   keys
   api


Go read :ref:`morestuff` for more.  Read about python package :py:mod:`package.name`
and python function :py:func:`package.name.function`.  The python class
:py:class:`package.name.ClassName` might also be useful, and its method
:py:meth:`package.name.ClassName.method`.  Not to mention the attribute
:py:attr:`package.name.ClassName.attrname`.  The code might throw the exception
:py:exc:`package.name.MyException`.

file2.rst
---------

.. _morestuff:

More stuff
==========

To show code::

    Indent this 4 spaces.
       You can indent more or less.
    And keep going.
```

```
    Even include blank lines

It ends after a blank line and returning to the original indentation.

You can also markup ``short inline code`` like this.
```

Automatically include the doc for a class like this:

```
.. _api:

.. autoclass:: healthvaultlib.healthvault.HealthVaultConn
    :members:
```

And document them:

```
class MyClassName(object):
   """
   Description.  Can refer to :py:meth:`.method1` here or anywhere in the file.

   :param string parm1: His name
   :param long parm2: His number
   """

   def __init__(self, parm1, parm2):
      pass

   def method1(self, arg1, arg2):
      """
      Description

      :param unicode arg1: something
      :param object arg2: something else
      """
```

http://techblog.ironfroggy.com/2012/06/how-to-use-sphinx-autodoc-on.html

Various code blocks:

```
.. code-block:: bash|python|text
   :linenos:
   :emphasize-lines: 1,3-5

   # Hi
   # there
   # all
   # you
   # coders
   # rejoice
```

```
1 # Hi
2 # there
3 # all
4 # you
5 # coders
6 # rejoice
```

You can include an `HTML link`_ like this and the definition can go nearby or at the bottom of the page:

```
.. _HTML link: http://foo.bar.com/
```

Or you can just write `HTML link <http://foo.bar.com>`_ all in one place.

http://sphinx-doc.org/markup/inline.html#ref-role

Link to a filename in this set of docs using :doc:`Any text you want </path/to/page>` or just :doc:`path`.

Don't include the ".rst" on the end of the filename. Relative filenames work too. But it's better to use :ref:, see next.

You can define an anchor point, which Sphinx calls a label. Put this above a section header:

```
.. _my-label:

My Section
----------
```

Now from somewhere else, you can write :ref:`my-label` and it'll be rendered as "My Section" and will link to the section. If you want some other text in the link, you can write :ref:`any text <my-label>` instead.

# Salt Stack

https://docs.saltstack.com/en/latest/topics/tutorials/pillar.html

## Fetching pillar data in templates

In the Jinja2 context, `pillar` is just a dictionary, so you can use the usual Python dictionary methods, e.g.:

```
{% for user, uid in pillar.get('users', {}).items() %}
{{user}}:
  user.present:
    - uid: {{uid}}
{% endfor %}
```

If you have nested data, it can be easier to use `salt['pillar.get']`, which accepts a single key parameter like `key1:key2`. E.g. if the pillar data looks like:

```
apache:
    user: fred
```

you could access the username as:

```
{{ salt['pillar.get']['apache:user'] }}
```

instead of:

```
{{ pillar['apache']['user'] }}
```

though that would work too.

CHAPTER 28

# Tmux

My tmux char: C-g (control-g)

session: a whole set of processes. Your tmux client is attached to one session. window:

Bindings:

```
Open prompt:    :

Client:
  detach current:  d
  detach any:      D
  suspend:         C-z

Sessions:
  rename   $
  select   s
  last     L

Windows:
  Info:       i
  Create:     c
  Kill:       &
  Switch to:  '
  Choose:     w
  Rename:     ,
  Switch to 0-9: 0-9
  Previously selected:   l
  Next:                  n
  Previous:              p
  Next window with bell   M-n

Pane:
  Kill:        x
  Previous:    ;
  swap with previous   {
  swap with next       }

Panes:
  Split current pane into two:
    Top and bottom: -
    Left and right: |
```

CHAPTER 29

# Travis CI

Magic syntax to run tox tests with multiple Python versions (as of Nov 1, 2017 anyway):

```
# Use travis container-based build system for speed
sudo: false

# Ubuntu trusty (14.04) - latest that Travis offers
dist: trusty

# Make sure all the python versions we need are pre-installed
# (apt-get is not available in the container-based build system)
addons:
  apt:
    sources:
    - deadsnakes
    packages:
    - python2.7
    - python3.4
    - python3.5
    - python3.6

language: python

# The version of Python that'll be used to invoke tox. Has no effect
# on what version of Python tox uses to run each set of tests.
python:
  - "3.5"

# Test a sampling of combinations
env:
  - TOXENV=py27-1.7
  - TOXENV=py27-1.8
  - TOXENV=py27-1.10
  - TOXENV=py27-1.11
  - TOXENV=py34-1.7
  - TOXENV=py34-1.8
  - TOXENV=py34-1.11
  - TOXENV=py35-1.9
  - TOXENV=py35-1.10
  - TOXENV=py35-1.11
  - TOXENV=py36-1.8
  - TOXENV=py36-1.10
  - TOXENV=py36-1.11
```

```
install:
  - pip install tox

script:
  - tox
#
#matrix:
#  allow_failures:
#    - env: TOXENV=py27-trunk,py33-trunk
```

CHAPTER 30

# Video

Video tools to use on Linux

## avidemux

GUI tool that's good for chopping up longer videos into shorter pieces without re-encoding when you need to watch the video and scan back and forth to find the places you want to split.

## makemkv

CLI good for ripping DVDs and Blu-Rays to mkv.

(Has GUI too.)

## mkvmerge

CLI only.

Join mkv streams end-to-end without re-encoding. (Inverse of avidemux.)

Split mkv streams at timestamps or chapters without re-encoding.

CHAPTER 31

# Virtualbox

List running VMs:

```
VBoxManage list runningvms
```

Stop a running VM:

```
VBoxManage controlvm <UUID|VMNAME> pause|resume|reset|poweroff etc.
```

Delete a VM:

```
VboxManage unregistervm <UUID|VMNAME> [--delete]
```

CHAPTER 32

# YAML

## YAML syntax

A value:

```
value
```

A value named “foo”:

```
foo: value
```

A list:

```
- 1
- 2
- 'a string'
```

A list named “bar”:

```
bar:
  - 1
  - 2
  - 'a string'
```

Alternate list syntax (“Flow style”):

```
bar: [1, 2, 'a string']
```

A dictionary:

```
key1: val1
key2: val2
key3: val3
```

A dictionary named “joe”:

```
joe:
  key1: val1
  key2: val2
  key3: val3
```

Dictionary in flow style:

```
joe: {key1: val1, key2: val2, key3: val3}
```

A list of dictionaries:

```
children:
    -   name: Jimmy Smith
        age: 15
    -   name: Jenny Smith
        age 12
```

CHAPTER 33

# Indices and tables

- genindex
- modindex
- search

## R