fabtools Documentation

Release 0.11.0

Ronan Amicel

February 15, 2013

CONTENTS

CHAPTER

ONE

ABOUT

`fabtools` includes useful functions to help you write your Fabric files.

`fabtools` makes it easier to manage system users, packages, databases, etc.

`fabtools` includes a number of low-level actions, as well as a higher level interface called `fabtools.require`.

Using `fabtools.require` allows you to use a more declarative style, similar to Chef or Puppet.

CHAPTER

TWO

INSTALLING

To install the latest release from PyPI

```
$ pip install fabtools
```

To install the latest development version from GitHub

```
$ pip install git+git://github.com/ronnix/fabtools.git
```

CHAPTER

THREE

EXAMPLE

Here is an example `fabfile.py` using `fabtools`

```
from fabric.api import *
from fabtools import require
import fabtools

@task
def setup():

    # Require some Debian/Ubuntu packages
    require.deb.packages([
        'imagemagick',
        'libxml2-dev',
    ])

    # Require a Python package
    with fabtools.python.virtualenv('/home/myuser/env'):
        require.python.package('pyramid')

    # Require an email server
    require.postfix.server('example.com')

    # Require a PostgreSQL server
    require.postgres.server()
    require.postgres.user('myuser', 's3cr3tp4ssw0rd')
    require.postgres.database('myappsdb', 'myuser')

    # Require a supervisor process for our app
    require.supervisor.process('myapp',
        command='/home/myuser/env/bin/gunicorn_paster /home/myuser/env/myapp/production.ini',
        directory='/home/myuser/env/myapp',
        user='myuser'
        )

    # Require an nginx server proxying to our app
    require.nginx.proxied_site('example.com',
        docroot='/home/myuser/env/myapp/myapp/public',
        proxy_url='http://127.0.0.1:8888'
        )

    # Setup a daily cron task
    fabtools.cron.add_daily('maintenance', 'myuser', 'my_script.py')
```

CHAPTER

FOUR

SUPPORTED TARGETS

`fabtools` currently supports the following target operating systems:

- Debian 6.0 (squeeze)
- Ubuntu 10.04 (lucid)
- Ubuntu 12.04 (precise)
- RHEL 5/6
- CentOS 5/6
- Scientific Linux 5/6
- SmartOS (Joyent)

Contributions to help support other Unix/Linux distributions are welcome!

CHAPTER

FIVE

API DOCUMENTATION

5.1 fabtools

5.1.1 fabtools.cron

Cron tasks

This module provides tools to manage periodic tasks using cron.

fabtools.cron.**add_task**(*name*, *timespec*, *user*, *command*)

Add a cron task.

The *command* will be run as *user* periodically.

You can use any valid crontab(5) *timespec*, including the `@hourly`, `@daily`, `@weekly`, `@monthly` and `@yearly` shortcuts.

Examples:

```
from fabtools.cron import add_task

# Run every month
add_task('cleanup', '@monthly', 'alice', '/home/alice/bin/cleanup.sh')

# Run every tuesday and friday at 5:30am
add_task('reindex', '30 5 * * 2,4', 'bob', '/home/bob/bin/reindex.sh')
```

fabtools.cron.**add_daily**(*name*, *user*, *command*)

Shortcut to add a daily cron task.

Example:

```
import fabtools

# Run every day
fabtools.cron.add_daily('backup', 'root', '/usr/local/bin/backup.sh')
```

5.1.2 fabtools.deb

Debian packages

This module provides tools to manage Debian/Ubuntu packages and repositories.

See Also:

fabtools.require.deb

fabtools.deb.**update_index**(*quiet=True*)
Update APT package definitions.

fabtools.deb.**upgrade**(*safe=True*)
Upgrade all packages.

fabtools.deb.**is_installed**(*pkg_name*)
Check if a package is installed.

fabtools.deb.**install**(*packages*, *update=False*, *options=None*)
Install one or more packages.

If *update* is `True`, the package definitions will be updated first, using `update_index()`.

Extra *options* may be passed to `apt-get` if necessary.

Example:

```
import fabtools

# Update index, then install a single package
fabtools.deb.install('build-essential', update=True)

# Install multiple packages
fabtools.deb.install([
    'python-dev',
    'libxml2-dev',
])
```

fabtools.deb.**uninstall**(*packages*, *purge=False*, *options=None*)
Remove one or more packages.

If *purge* is `True`, the package configuration files will be removed from the system.

Extra *options* may be passed to `apt-get` if necessary.

fabtools.deb.**preseed_package**(*pkg_name*, *preseed*)
Enable unattended package installation by preseeding `debconf` parameters.

Example:

```
import fabtools

# Unattended install of Postfix mail server
fabtools.deb.preseed_package('postfix', {
    'postfix/main_mailer_type': ('select', 'Internet Site'),
    'postfix/mailname': ('string', 'example.com'),
    'postfix/destinations': ('string', 'example.com, localhost.localdomain, localhost'),
})
fabtools.deb.install('postfix')
```

fabtools.deb.**get_selections**()
Get the state of `dkpg` selections.

Returns a dict with state => [packages].

fabtools.deb.**distrib_codename**()
Get the codename of the distrib.

Example:

```
from fabtools.deb import distrib_codename

if distrib_codename() == 'precise':
    print('Ubuntu 12.04 LTS')
```

fabtools.deb.**add_apt_key**(*filename*, *update=True*)
Trust packages signed with this public key.

Example:

```
import fabtools

# Download 3rd party APT public key
if not fabtools.is_file('rabbitmq-signing-key-public.asc'):
    run('wget http://www.rabbitmq.com/rabbitmq-signing-key-public.asc')

# Tell APT to trust that key
fabtools.deb.add_apt_key('rabbitmq-signing-key-public.asc')
```

5.1.3 fabtools.files

Files and directories

See Also:

fabtools.require.files

fabtools.files.**is_file**(*path*, *use_sudo=False*)
Check if a path exists, and is a file.

fabtools.files.**is_dir**(*path*, *use_sudo=False*)
Check if a path exists, and is a directory.

fabtools.files.**is_link**(*path*, *use_sudo=False*)
Check if a path exists, and is a symbolic link.

fabtools.files.**owner**(*path*, *use_sudo=False*)
Get the owner name of a file or directory.

fabtools.files.**group**(*path*, *use_sudo=False*)
Get the group name of a file or directory.

fabtools.files.**mode**(*path*, *use_sudo=False*)
Get the mode (permissions) of a file or directory.

Returns a string such as `'0755'`, representing permissions as an octal number.

fabtools.files.**upload_template**(*filename*, *template*, *context=None*, *use_sudo=False*, *user='root'*, *mkdir=False*, *chown=False*)
Upload a template file.

fabtools.files.**md5sum**(*filename*, *use_sudo=False*)
Compute the MD5 sum of a file.

class fabtools.files.**watch**(*filenames*, *callback=None*, *use_sudo=False*)
Context manager to watch for changes to the contents of some files.

The *filenames* argument can be either a string (single filename) or a list (multiple filenames).

You can read the *changed* attribute at the end of the block to check if the contents of any of the watched files has changed.

You can also provide a *callback* that will be called at the end of the block if the contents of any of the watched files has changed.

Example using an explicit check:

```
from fabric.contrib.files import comment, uncomment

from fabtools.files import watch
from fabtools.services import restart

# Edit configuration file
with watch('/etc/daemon.conf') as config:
    uncomment('/etc/daemon.conf', 'someoption')
    comment('/etc/daemon.conf', 'otheroption')

# Restart daemon if needed
if config.changed:
    restart('daemon')
```

Same example using a callback:

```
from functools import partial

from fabric.contrib.files import comment, uncomment

from fabtools.files import watch
from fabtools.services import restart

with watch('/etc/daemon.conf', callback=partial(restart, 'daemon')):
    uncomment('/etc/daemon.conf', 'someoption')
    comment('/etc/daemon.conf', 'otheroption')
```

5.1.4 fabtools.group

Groups

See Also:

fabtools.require.groups

fabtools.group.**exists**(*name*)
Check if a group exists.

fabtools.group.**create**(*name*, *gid=None*)
Create a new group.

Example:

```
import fabtools

if not fabtools.group.exists('admin'):
    fabtools.group.create('admin')
```

5.1.5 fabtools.mysql

MySQL users and databases

This module provides tools for creating MySQL users and databases.

See Also:

fabtools.require.mysql

Manage users

fabtools.mysql.**user_exists**(*name*, ***kwargs*)
Check if a MySQL user exists.

fabtools.mysql.**create_user**(*name*, *password*, *host='localhost'*, ***kwargs*)
Create a MySQL user.

Example:

```
import fabtools

# Create DB user if it does not exist
if not fabtools.mysql.user_exists('dbuser'):
    fabtools.mysql.create_user('dbuser', password='somerandomstring')
```

Manage databases

fabtools.mysql.**database_exists**(*name*, ***kwargs*)
Check if a MySQL database exists.

fabtools.mysql.**create_database**(*name*, *owner=None*, *owner_host='localhost'*, *charset='utf8'*, *collate='utf8_general_ci'*, ***kwargs*)
Create a MySQL database.

Example:

```
import fabtools

# Create DB if it does not exist
if not fabtools.mysql.database_exists('myapp'):
    fabtools.mysql.create_database('myapp', owner='dbuser')
```

5.1.6 fabtools.network

Network

fabtools.network.**interfaces**()
Get the list of network interfaces. Will return all datalinks on SmartOS.

fabtools.network.**address**(*interface*)
Get the IPv4 address assigned to an interface.

Example:

```
import fabtools

# Print all configured IP addresses
for interface in fabtools.network.interfaces():
    print(fabtools.network.address(interface))
```

fabtools.network.**nameservers**()
Get the list of nameserver addresses.

Example:

```
import fabtools

# Check that all name servers are reachable
for ip in fabtools.network.nameservers():
    run('ping -c1 %s' % ip)
```

5.1.7 fabtools.nodejs

Node.js

This module provides tools for installing Node.js and managing packages using npm.

See Also:

fabtools.require.nodejs

fabtools.nodejs.**install_from_source**(*version='0.8.19'*)
Install Node JS from source.

```
import fabtools

# Install Node.js
fabtools.nodejs.install_nodejs()
```

Note: This function may not work for old versions of Node.js.

---

fabtools.nodejs.**version**()
Get the version of Node.js currently installed.

Returns `None` if it is not installed.

fabtools.nodejs.**install_package**(*package*, *version=None*, *local=False*)
Install a Node.js package.

If *local* is `True`, the package will be installed locally.

```
import fabtools

# Install package globally
fabtools.nodejs.install_package('express')

# Install package locally
fabtools.nodejs.install_package('underscore', local=False)
```

fabtools.nodejs.**install_dependencies**()
Install Node.js package dependencies.

This function calls `npm install`, which will locally install all packages specified as dependencies in the `package.json` file found in the current directory.

```
from fabric.api import cd
from fabtools import nodejs

with cd('/path/to/nodejsapp/'):
    nodejs.install_dependencies()
```

fabtools.nodejs.**package_version**(*package*, *local=False*)
Get the installed version of a Node.js package.

Returns `None``is the package is not installed. If *local* is ``True`, returns the version of the locally installed package.

fabtools.nodejs.**update_package**(*package*, *local=False*)
Update a Node.js package.

If *local* is `True`, the package will be updated locally.

fabtools.nodejs.**uninstall_package**(*package*, *version=None*, *local=False*)
Uninstall a Node.js package.

If *local* is `True`, the package will be uninstalled locally.

```
import fabtools

# Uninstall package globally
fabtools.nodejs.uninstall_package('express')

# Uninstall package locally
fabtools.nodejs.uninstall_package('underscore', local=False)
```

5.1.8 fabtools.openvz

OpenVZ containers

This module provides high-level tools for managing OpenVZ templates and containers.

Warning: The remote host needs a patched kernel with OpenVZ support.

See Also:

fabtools.require.openvz

Manage templates

fabtools.openvz.**download_template**(*name=None*, *url=None*)
Download an OpenVZ template.

Example:

```
from fabtools.openvz import download_template

# Use custom OS template
download_template(url='http://example.com/templates/mybox.tar.gz')
```

If no *url* is provided, the OS template will be downloaded from the download.openvz.org repository:

```
from fabtools.openvz import download_template

# Use OS template from http://download.openvz.org/template/precreated/
download_template('debian-6.0-x86_64')
```

Manage containers

fabtools.openvz.**exists**(*ctid_or_name*)
Check if the container exists.

fabtools.openvz.**create**(*ctid*, *ostemplate=None*, *config=None*, *private=None*, *root=None*, *ipadd=None*, *hostname=None*, ***kwargs*)
Create an OpenVZ container.

fabtools.openvz.**set**(*ctid_or_name*, *save=True*, ***kwargs*)
Set container parameters.

fabtools.openvz.**status**(*ctid_or_name*)
Get the status of the container.

fabtools.openvz.**start**(*ctid_or_name*, *wait=False*, *force=False*, ***kwargs*)
Start the container.

If *wait* is `True`, wait until the container is up and running.

> **Warning:** `wait=True` is broken with vzctl 3.0.24 on Debian 6.0 (*squeeze*)

fabtools.openvz.**stop**(*ctid_or_name*, *fast=False*, ***kwargs*)
Stop the container.

fabtools.openvz.**restart**(*ctid_or_name*, *wait=True*, *force=False*, *fast=False*, ***kwargs*)
Restart the container.

fabtools.openvz.**destroy**(*ctid_or_name*)
Destroy the container.

Run commands inside a container

fabtools.openvz.**exec2**(*ctid_or_name*, *command*)
Run a command inside the container.

```
import fabtools

res = fabtools.openvz.exec2('foo', 'hostname')
```

> **Warning:** the command will be run as **root**.

fabtools.openvz.**guest**(**args*, ***kwds*)
Context manager to run commands inside a guest container.

Supported basic operations are: run, sudo and put.

> **Warning:** commands executed with `run()` will be run as **root** inside the container. Use `sudo(command, user='foo')` to run them as an unpriviledged user.

Example:

```
from fabtools.openvz import guest

with guest('foo'):
    run('hostname')
    sudo('whoami', user='alice')
    put('files/hello.txt')
```

Container class

class fabtools.openvz.container.**Container**(*ctid*)

Object-oriented interface to OpenVZ containers.

create(***kwargs*)

Create the container.

Extra args are passed to `fabtools.openvz.create()`.

destroy()

Destroy the container.

set(***kwargs*)

Set container parameters.

Extra args are passed to `fabtools.openvz.set()`.

start(***kwargs*)

Start the container.

Extra args are passed to `fabtools.openvz.start()`.

stop(***kwargs*)

Stop the container.

Extra args are passed to `fabtools.openvz.stop()`.

restart(***kwargs*)

Restart the container.

Extra args are passed to `fabtools.openvz.restart()`.

status()

Get the container's status.

running()

Check if the container is running.

exists()

Check if the container exists.

exec2(*command*)

Run a command inside the container.

```
from fabtools.require.openvz import container

with container('foo') as ct:
    res = ct.exec2('hostname')
```

> **Warning:** the command will be run as **root**.

5.1.9 fabtools.pkg

SmartOS packages

This module provides tools to manage SmartOS packages.

See Also:

fabtools.require.pkg

fabtools.pkg.**update_index**(*force=False*)
Update pkgin package definitions.

fabtools.pkg.**upgrade**(*full=False*)
Upgrade all packages.

fabtools.pkg.**is_installed**(*pkg_name*)
Check if a package is installed.

fabtools.pkg.**install**(*packages*, *update=False*, *yes=None*, *options=None*)
Install one or more packages.

If *update* is `True`, the package definitions will be updated first, using `update_index()`.

Extra *yes* may be passed to `pkgin` to validate license if necessary.

Extra *options* may be passed to `pkgin` if necessary.

Example:

```
import fabtools

# Update index, then verbosely install a single package
fabtools.pkg.install('redis', update=True, options='-V',)

# Install multiple packages
fabtools.pkg.install([
    'unzip',
    'top'
])
```

fabtools.pkg.**uninstall**(*packages*, *orphan=False*, *options=None*)
Remove one or more packages.

If *orphan* is `True`, orphan dependencies will be removed from the system.

Extra *options* may be passed to `pkgin` if necessary.

fabtools.pkg.**smartos_build**()
Get the build of SmartOS. Useful to determine provider for example.

Example:

```
from fabtools.pkg import smartos_build

if smartos_build().startswith('joyent'):
    print('SmartOS Joyent')
```

fabtools.pkg.**smartos_image**()
Get the SmartOS image. Useful to determine the image/dataset for example. Returns None if it can't be determined.

Example:

```
from fabtools.pkg import smartos_image

if smartos_image().startswith('percona'):
    sudo("mysql -uroot -psecretpassword -e 'show databases;'")
```

5.1.10 fabtools.postgres

PostgreSQL users and databases

This module provides tools for creating PostgreSQL users and databases.

See Also:

fabtools.require.postgres

Manage users

fabtools.postgres.**user_exists**(*name*)
Check if a PostgreSQL user exists.

fabtools.postgres.**create_user**(*name*, *password*, *superuser=False*, *createdb=False*, *createrole=False*, *inherit=True*, *login=True*, *connection_limit=None*, *encrypted_password=False*)
Create a PostgreSQL user.

Example:

```
import fabtools

# Create DB user if it does not exist
if not fabtools.postgres.user_exists('dbuser'):
    fabtools.postgres.create_user('dbuser', password='somerandomstring')

# Create DB user with custom options
fabtools.postgres.create_user('dbuser2', password='s3cr3t',
    createdb=True, createrole=True, connection_limit=20)
```

Manage databases

fabtools.postgres.**database_exists**(*name*)
Check if a PostgreSQL database exists.

fabtools.postgres.**create_database**(*name*, *owner*, *template='template0'*, *encoding='UTF8'*, *locale='en_US.UTF-8'*)
Create a PostgreSQL database.

Example:

```
import fabtools

# Create DB if it does not exist
if not fabtools.postgres.database_exists('myapp'):
    fabtools.postgres.create_database('myapp', owner='dbuser')
```

5.1.11 fabtools.python

Python environments and packages

This module includes tools for using virtual environments and installing packages using pip.

See Also:

fabtools.python_distribute and *fabtools.require.python*

Virtual environments

fabtools.python.**virtualenv**(**args*, ***kwds*)
Context manager to activate an existing Python virtual environment.

```
from fabric.api import run
from fabtools.python import virtualenv

with virtualenv('/path/to/virtualenv'):
    run('python -V')
```

Installing pip

fabtools.python.**is_pip_installed**(*version=None*)
Check if pip is installed.

fabtools.python.**install_pip**()
Install the latest version of pip.

```
import fabtools

if not fabtools.python.is_pip_installed():
    fabtools.python.install_pip()
```

Installing packages

fabtools.python.**is_installed**(*package*)
Check if a Python package is installed.

fabtools.python.**install**(*packages*, *upgrade=False*, *use_mirrors=True*, *use_sudo=False*, *user=None*, *download_cache=None*)
Install Python package(s) using pip.

Examples:

```
import fabtools

# Install a single package
fabtools.python.install('package', use_sudo=True)

# Install a list of packages
fabtools.python.install(['pkg1', 'pkg2'], use_sudo=True)
```

fabtools.python.**install_requirements**(*filename*, *upgrade=False*, *use_mirrors=True*, *use_sudo=False*, *user=None*, *download_cache=None*)
Install Python packages from a pip requirements file.

```
import fabtools

fabtools.python.install_requirements('project/requirements.txt')
```

5.1.12 fabtools.python_distribute

Python packages

This module provides tools for installing Python packages using the `easy_install` command provided by distribute.

See Also:

fabtools.python and *fabtools.require.python*

`fabtools.python_distribute.`**`is_distribute_installed`**`()`
Check if distribute is installed.

`fabtools.python_distribute.`**`install_distribute`**`()`
Install the latest version of distribute.

```
from fabtools.python_distribute import *

if not is_distribute_installed():
    install_distribute()
```

`fabtools.python_distribute.`**`install`**(*packages*, *upgrade=False*, *use_sudo=False*)
Install Python packages with `easy_install`.

Examples:

```
import fabtools

# Install a single package
fabtools.python_distribute.install('package', use_sudo=True)

# Install a list of packages
fabtools.python_distribute.install(['pkg1', 'pkg2'], use_sudo=True)
```

5.1.13 fabtools.rpm

RPM packages

This module provides tools to manage CentOS/RHEL/SL/Fedora packages and repositories.

See Also:

fabtools.require.rpm

`fabtools.rpm.`**`update`**(*kernel=False*)
Upgrade all packages, skip obsoletes if `obsoletes=0` in `yum.conf`.

Exclude *kernel* upgrades by default.

`fabtools.rpm.`**`upgrade`**(*kernel=False*)
Upgrade all packages, including obsoletes.

Exclude *kernel* upgrades by default.

fabtools.rpm.**groupupdate**(*group*, *options=None*)
Update an existing software group, skip obsoletes if `obsoletes=1` in `yum.conf`.

Extra *options* may be passed to `yum` if necessary.

fabtools.rpm.**is_installed**(*pkg_name*)
Check if a *package* is installed.

fabtools.rpm.**install**(*packages*, *repos=None*, *yes=None*, *options=None*)
Install one or more *packages*.

Extra *repos* may be passed to `yum` to enable extra repositories at install time.

Extra *yes* may be passed to `yum` to validate license if necessary.

Extra *options* may be passed to `yum` if necessary like: ['–nogpgcheck', '–exclude=package']

Example:

```
import fabtools

# Install a single package, in an alternative install root
fabtools.rpm.install('emacs', options='--installroot=/my/new/location')

# Install multiple packages silently
fabtools.rpm.install([
    'unzip',
    'nano'
], '--quiet')
```

fabtools.rpm.**groupinstall**(*group*, *options=None*)
Install a *group* of packages. Use `yum grouplist` to get the list of groups.

Extra *options* may be passed to `yum` if necessary like: ['–nogpgcheck', '–exclude=package']

Example:

```
import fabtools

# Install development packages
fabtools.rpm.groupinstall('Development tools')
```

fabtools.rpm.**uninstall**(*packages*, *options=None*)
Remove one or more *packages*.

Extra *options* may be passed to `yum` if necessary.

fabtools.rpm.**groupuninstall**(*group*, *options=None*)
Remove an existing software group.

Extra *options* may be passed to `yum` if necessary.

fabtools.rpm.**distrib_id**()
Get the ID of the distrib.

Example:

```
from fabtools.rpm import distrib_id

if distrib_id() == 'CentOS':
    print('%s is not running RHEL.' % (env.host))
```

fabtools.rpm.**distrib_codename**()
Get the codename of the distrib.

Example:

```
from fabtools.rpm import distrib_codename

if distrib_codename() == 'Final':
    print('%s is running final version of %s %s.'
      % (env.host, distrib_id(), distrib_release))
```

fabtools.rpm.**distrib_desc**()
Get the description of the distrib.

fabtools.rpm.**distrib_release**()
Get the release number of the distrib.

Example:

```
from fabtools.rpm import distrib_release

if distrib_release() == '6.1' and distrib_id == 'CentOS':
    print('CentOS 6.2 has been released. Please update.')
```

fabtools.rpm.**repolist**(*status=''*, *media=None*)
Get the list of `yum` repositories. Returns enabled repositories by default.

Extra *status* may be passed to list disabled repositories if necessary.

Media and debug repositories are kept disabled, except if you pass *media*.

Example:

```
import fabtools

# Install a package that may be included in disabled repositories
fabtools.rpm.install('vim', fabtools.rpm.repolist('disabled'))
```

5.1.14 fabtools.service

System services

This module provides low-level tools for managing system services, using the `service` command. It supports both upstart services and traditional SysV-style `/etc/init.d/` scripts.

See Also:

fabtools.require.service

fabtools.service.**is_running**(*service*)
Check if a service is running.

```
import fabtools

if fabtools.service.is_running('foo'):
    print "Service foo is running!"
```

fabtools.service.**start**(*service*)
Start a service.

```
import fabtools

# Start service if it is not running
```

```
    if not fabtools.service.is_running('foo'):
        fabtools.service.start('foo')
```

fabtools.service.**stop**(*service*)
Stop a service.

```
    import fabtools

    # Stop service if it is running
    if fabtools.service.is_running('foo'):
        fabtools.service.stop('foo')
```

fabtools.service.**restart**(*service*)
Restart a service.

```
    import fabtools

    # Start service, or restart it if it is already running
    if fabtools.service.is_running('foo'):
        fabtools.service.restart('foo')
    else:
        fabtools.service.start('foo')
```

fabtools.service.**reload**(*service*)
Reload a service.

```
    import fabtools

    # Reload service
    fabtools.service.reload('foo')
```

> **Warning:** The service needs to support the `reload` operation.

fabtools.service.**force_reload**(*service*)
Force reload a service.

```
    import fabtools

    # Force reload service
    fabtools.service.force_reload('foo')
```

> **Warning:** The service needs to support the `force-reload` operation.

5.1.15 fabtools.shorewall

Shorewall firewall

See Also:

fabtools.require.shorewall

Firewall status

fabtools.shorewall.**status**()
Get the firewall status.

fabtools.shorewall.**is_started**()
Check if the firewall is started.

fabtools.shorewall.**is_stopped**()
Check if the firewall is stopped.

Firewall rules

fabtools.shorewall.**rule**(*port*, *action='ACCEPT'*, *source='net'*, *dest='$FW'*, *proto='tcp'*)
Helper to build a firewall rule.

Examples:

```
from fabtools.shorewall import rule

# Rule to accept connections from example.com on port 1234
r1 = rule(port=1234, source=hosts(['example.com']))

# Rule to reject outgoing SMTP connections
r2 = rule(port=25, action='REJECT', source='$FW', dest='net')
```

fabtools.shorewall.**hosts**(*hostnames*, *zone='net'*)
Builds a host list suitable for use in a firewall rule.

fabtools.shorewall.**Ping**(***kwargs*)
Helper to build a firewall rule for ICMP pings.

Extra args will be passed to `rule()`.

fabtools.shorewall.**SSH**(*port=22*, ***kwargs*)
Helper to build a firewall rule for SSH connections

Extra args will be passed to `rule()`.

fabtools.shorewall.**HTTP**(*port=80*, ***kwargs*)
Helper to build a firewall rule for HTTP connections

Extra args will be passed to `rule()`.

fabtools.shorewall.**HTTPS**(*port=443*, ***kwargs*)
Helper to build a firewall rule for HTTPS connections

Extra args will be passed to `rule()`.

fabtools.shorewall.**SMTP**(*port=25*, ***kwargs*)
Helper to build a firewall rule for SMTP connections

Extra args will be passed to `rule()`.

5.1.16 fabtools.supervisor

Supervisor processes

This module provides high-level tools for managing long-running processes using supervisord.

See Also:

fabtools.require.supervisor

Manage supervisord

fabtools.supervisor.**reload_config**()
Reload supervisor configuration.

fabtools.supervisor.**update_config**()
Reread and update supervisor job configurations.

Less heavy-handed than a full reload, as it doesn't restart the backend supervisor process and all managed processes.

Manage processes

fabtools.supervisor.**process_status**(*name*)
Get the status of a supervisor process.

fabtools.supervisor.**start_process**(*name*)
Start a supervisor process

fabtools.supervisor.**stop_process**(*name*)
Stop a supervisor process

fabtools.supervisor.**restart_process**(*name*)
Restart a supervisor process

5.1.17 fabtools.system

System settings

Hostname

fabtools.system.**get_hostname**()
Get the fully qualified hostname.

fabtools.system.**set_hostname**(*hostname*, *persist=True*)
Set the hostname.

Kernel parameters

fabtools.system.**get_sysctl**(*key*)
Get a kernel parameter.

Example:

```
from fabtools.system import get_sysctl

print "Max number of open files:", get_sysctl('fs.file-max')
```

fabtools.system.**set_sysctl**(*key*, *value*)
Set a kernel parameter.

Example:

```
import fabtools

# Protect from SYN flooding attack
fabtools.system.set_sysctl('net.ipv4.tcp_syncookies', 1)
```

Locales

fabtools.system.**supported_locales**()
Gets the list of supported locales.

Each locale is returned as a `(locale, charset)` tuple.

5.1.18 fabtools.user

Users

See Also:

fabtools.require.users

fabtools.user.**exists**(*name*)
Check if a user exists.

fabtools.user.**create**(*name*, *comment=None*, *home=None*, *create_home=True*, *skeleton_dir=None*, *group=None*, *create_group=True*, *extra_groups=None*, *password=None*, *system=False*, *shell=None*, *uid=None*)
Create a new user and its home directory.

Example:

```
import fabtools

if not fabtools.user.exists('alice'):
    fabtools.user.create('alice')

with cd('/home/alice'):
    # ...
```

fabtools.user.**modify**(*name*, *comment=None*, *home=None*, *move_current_home=False*, *group=None*, *extra_groups=None*, *login_name=None*, *password=None*, *shell=None*, *uid=None*)
Modify an existing user.

Example:

```
import fabtools

if fabtools.user.exists('alice'):
    fabtools.user.modify('alice', shell='/bin/sh')
```

5.1.19 fabtools.utils

Utilities

fabtools.utils.**run_as_root**(*command*, **args*, ***kwargs*)
Run a remote command as the root user.

When connecting as root to the remote system, this will use Fabric's `run` function. In other cases, it will use `sudo`.

5.1.20 fabtools.vagrant

Vagrant helpers

fabtools.vagrant.**vagrant**()
Run the following tasks on a vagrant box.

First, you need to import this task in your `fabfile.py`:

```
from fabric.api import *
from fabtools.vagrant import vagrant

@task
def some_task():
    run('echo hello')
```

Then you can easily run tasks on your current Vagrant box:

```
$ fab vagrant some_task
```

fabtools.vagrant.**ssh_config**(*name=''*)
Get the SSH parameters for connecting to a vagrant VM.

fabtools.vagrant.**vagrant**
Run the following tasks on a vagrant box.

First, you need to import this task in your `fabfile.py`:

```
from fabric.api import *
from fabtools.vagrant import vagrant

@task
def some_task():
    run('echo hello')
```

Then you can easily run tasks on your current Vagrant box:

```
$ fab vagrant some_task
```

fabtools.vagrant.**vagrant_settings**(*name=''*, **args*, ***kwargs*)
Context manager that sets a vagrant VM as the remote host.

Use this context manager inside a task to run commands on your current Vagrant box:

```
from fabtools.vagrant import vagrant_settings

with vagrant_settings():
    run('hostname')
```

5.2 fabtools.require

5.2.1 fabtools.require.deb

Debian packages

This module provides high-level tools for managing Debian/Ubuntu packages and repositories.

See Also:

fabtools.deb

Repositories

fabtools.require.deb.**source**(*name*, *uri*, *distribution*, **components*)

Require a package source.

```
from fabtools import require

# Official MongoDB packages
require.deb.source('mongodb', 'http://downloads-distro.mongodb.org/repo/ubuntu-upstart', 'dist',
```

fabtools.require.deb.**ppa**(*name*)

Require a PPA package source.

Example:

```
from fabtools import require

# Node.js packages by Chris Lea
require.deb.ppa('ppa:chris-lea/node.js')
```

Packages

fabtools.require.deb.**package**(*pkg_name*, *update=False*)

Require a deb package to be installed.

Example:

```
from fabtools import require

require.deb.package('foo')
```

fabtools.require.deb.**packages**(*pkg_list*, *update=False*)

Require several deb packages to be installed.

Example:

```
from fabtools import require

require.deb.packages([
    'foo',
    'bar',
    'baz',
])
```

fabtools.require.deb.**nopackage**(*pkg_name*)
Require a deb package to be uninstalled.

Example:

```
from fabtools import require

require.deb.nopackage('apache2')
```

fabtools.require.deb.**nopackages**(*pkg_list*)
Require several deb packages to be uninstalled.

Example:

```
from fabtools import require

require.deb.nopackages([
    'perl',
    'php5',
    'ruby',
])
```

5.2.2 fabtools.require.files

Files and directories

This module provides high-level tools for managing files and directories.

See Also:

fabtools.files

fabtools.require.files.**directory**(*path*, *use_sudo=False*, *owner=''*, *group=''*, *mode=''*)
Require a directory to exist.

```
from fabtools import require

require.directory('/tmp/mydir', owner='alice', use_sudo=True)
```

Note: This function can be accessed directly from the `fabtools.require` module for convenience.

---

fabtools.require.files.**file**(*path=None*, *contents=None*, *source=None*, *url=None*, *md5=None*, *use_sudo=False*, *owner=None*, *group=''*, *mode=None*, *verify_remote=True*)
Require a file to exist and have specific contents and properties.

You can provide either:

- *contents*: the required contents of the file:

```
from fabtools import require

require.file('/tmp/hello.txt', contents='Hello, world')
```

- *source*: the local path of a file to upload:

```
from fabtools import require

require.file('/tmp/hello.txt', source='files/hello.txt')
```

•*url*: the URL of a file to download (*path* is then optional):

```
from fabric.api import cd
from fabtools import require

with cd('tmp'):
    require.file(url='http://example.com/files/hello.txt')
```

If *verify_remote* is `True` (the default), then an MD5 comparison will be used to check whether the remote file is the same as the source. If this is `False`, the file will be assumed to be the same if it is present. This is useful for very large files, where generating an MD5 sum may take a while.

Note: This function can be accessed directly from the `fabtools.require` module for convenience.

`fabtools.require.files.`**`template_file`**(*path=None*, *template_contents=None*, *template_source=None*, *context=None*, ***kwargs*)

Require a file whose contents is defined by a template.

5.2.3 fabtools.require.groups

System groups

See Also:

fabtools.group

`fabtools.require.groups.`**`group`**(*name*, *gid=None*)

Require a group.

```
from fabtools import require

require.group('mygroup')
```

Note: This function can be accessed directly from the `fabtools.require` module for convenience.

5.2.4 fabtools.require.mysql

MySQL

This module provides high-level tools for installing a MySQL server and creating MySQL users and databases.

See Also:

fabtools.mysql

`fabtools.require.mysql.`**`server`**(*version=None*, *password=None*)

Require a MySQL server to be installed and running.

Example:

```
from fabtools import require

require.mysql.server(password='s3cr3t')
```

fabtools.require.mysql.**user**(*name*, *password*, ***kwargs*)
Require a MySQL user.

Extra arguments will be passed to fabtools.mysql.create_user().

Example:

```
from fabtools import require

require.mysql.user('dbuser', 'somerandomstring')
```

fabtools.require.mysql.**database**(*name*, ***kwargs*)
Require a MySQL database.

Extra arguments will be passed to fabtools.mysql.create_database().

Example:

```
from fabtools import require

require.mysql.database('myapp', owner='dbuser')
```

5.2.5 fabtools.require.nginx

Nginx

This module provides high-level tools for installing the nginx web server and managing the configuration of web sites.

fabtools.require.nginx.**server**()
Require nginx server to be installed and running.

```
from fabtools import require

require.nginx.server()
```

fabtools.require.nginx.**enabled**(*config*)
Ensure link to /etc/nginx/sites-available/config exists and reload nginx configuration if needed.

fabtools.require.nginx.**disabled**(*config*)
Ensure link to /etc/nginx/sites-available/config doesn't exist and reload nginx configuration if needed.

fabtools.require.nginx.**site**(*server_name*, *template_contents=None*, *template_source=None*, *enabled=True*, *check_config=True*, ***kwargs*)
Require an nginx site.

You must provide a template for the site configuration, either as a string (*template_contents*) or as the path to a local template file (*template_source*).

```
from fabtools import require

CONFIG_TPL = '''
server {
    listen      %(port)d;
    server_name %(server_name)s %(server_alias)s;
    root        %(docroot)s;
    access_log  /var/log/nginx/%(server_name)s.log;
}'''

require.nginx.site('example.com', template_contents=CONFIG_TPL,
```

```
    port=80,
    server_alias='www.example.com',
    docroot='/var/www/mysite',
)
```

See Also:

`fabtools.require.files.template_file()`

`fabtools.require.nginx.`**`proxied_site`**(*server_name*, *enabled=True*, ***kwargs*)

Require an nginx site for a proxied app.

This uses a predefined configuration template suitable for proxying requests to a backend application server.

Required keyword arguments are:

- *port*: the port nginx should listen on
- *proxy_url*: URL of backend application server
- *docroot*: path to static files

```
from fabtools import require

require.nginx.proxied_site('example.com',
    port=80,
    proxy_url='http://127.0.0.1:8080/',
    docroot='/path/to/myapp/static',
)
```

5.2.6 fabtools.require.nodejs

Node.js

This module provides tools for installing Node.js and managing packages using npm.

See Also:

fabtools.nodejs

`fabtools.require.nodejs.`**`installed_from_source`**(*version='0.8.19'*)

Require Node.js to be installed from source.

```
from fabtools import require

require.nodejs.installed_from_source()
```

`fabtools.require.nodejs.`**`package`**(*pkg_name*, *version=None*, *local=False*)

Require a Node.js package.

If the package is not installed, and no *version* is specified, the latest available version will be installed.

If a *version* is specified, and a different version of the package is already installed, it will be updated to the specified version.

If *local* is `True`, the package will be installed locally.

```
from fabtools import require

# Install package system-wide
require.nodejs.package('foo')
```

```
# Install package locally
require.nodejs.package('bar', local=True)
```

5.2.7 fabtools.require.openvz

OpenVZ containers

This module provides high-level tools for managing OpenVZ templates and containers.

Warning: The remote host needs a patched kernel with OpenVZ support.

See Also:

fabtools.openvz

fabtools.require.openvz.**template**(*name=None*, *url=None*)

Require an OpenVZ OS template.

If the OS template is not installed yet, it will be downloaded from *url* using `download_template()`:

```
from fabtools import require

# Use custom OS template
require.openvz.template(url='http://example.com/templates/mybox.tar.gz')
```

If no *url* is provided, `download_template()` will attempt to download the OS template from the download.openvz.org repository:

```
from fabtools import require

# Use OS template from http://download.openvz.org/template/precreated/
require.openvz.template('debian-6.0-x86_64')
```

fabtools.require.openvz.**container**(*name*, *ostemplate*, ***kwargs*)

Require an OpenVZ container.

If it does not exist, the container will be created using the specified OS template (see `fabtools.require.openvz.template()`).

Extra args will be passed to `fabtools.openvz.create()`:

```
from fabtools import require

require.openvz.container('foo', 'debian', ipadd='1.2.3.4')
```

This function returns a `fabtools.openvz.Container` object, that can be used to perform further operations:

```
from fabtools.require.openvz import container

ct = container('foo', 'debian')
ct.set('ipadd', '1.2.3.4')
ct.start()
ct.exec2('hostname')
```

This function can also be used as a context manager:

```
from fabtools.require.openvz import container

with container('foo', 'debian') as ct:
    ct.set('ipadd', '1.2.3.4')
    ct.start()
    ct.exec2('hostname')
```

5.2.8 fabtools.require.pkg

SmartOS packages

This module provides high-level tools for managing SmartOS packages.

See Also:

fabtools.pkg

fabtools.require.pkg.**package**(*pkg_name*, *update=False*, *yes=None*)
Require a SmartOS package to be installed.

```
from fabtools import require

require.pkg.package('foo')
```

fabtools.require.pkg.**packages**(*pkg_list*, *update=False*)
Require several SmartOS packages to be installed.

```
from fabtools import require

require.pkg.packages([
    'top',
    'unzip',
    'zip',
])
```

fabtools.require.pkg.**nopackage**(*pkg_name*, *orphan=True*)
Require a SmartOS package to be uninstalled.

```
from fabtools import require

require.pkg.nopackage('top')
```

fabtools.require.pkg.**nopackages**(*pkg_list*, *orphan=True*)
Require several SmartOS packages to be uninstalled.

```
from fabtools import require

require.pkg.nopackages([
    'top',
    'zip',
    'unzip',
])
```

5.2.9 fabtools.require.postfix

Postfix

This module provides high-level tools for managing the Postfix email server.

fabtools.require.postfix.**server**(*mailname*)
Require a Postfix email server.

This makes sure that Postfix is installed and started.

```
from fabtools import require

# Handle incoming email for our domain
require.postfix.server('example.com')
```

5.2.10 fabtools.require.postgres

PostgreSQL users and databases

See Also:

fabtools.postgres

fabtools.require.postgres.**server**(*version=None*)
Require a PostgreSQL server to be installed and running.

```
from fabtools import require

require.postgres.server()
```

fabtools.require.postgres.**user**(*name*, *password*, *superuser=False*, *createdb=False*, *create-role=False*, *inherit=True*, *login=True*, *connection_limit=None*, *encrypted_password=False*)
Require the existence of a PostgreSQL user. The password and options provided will only be applied when creating a new user (existing users will *not* be modified).

```
from fabtools import require

require.postgres.user('dbuser', password='somerandomstring')

require.postgres.user('dbuser2', password='s3cr3t',
    createdb=True, create_role=True, connection_limit=20)
```

fabtools.require.postgres.**database**(*name*, *owner*, *template='template0'*, *encoding='UTF8'*, *locale='en_US.UTF-8'*)
Require a PostgreSQL database.

```
from fabtools import require

require.postgres.database('myapp', owner='dbuser')
```

5.2.11 fabtools.require.python

Python environments and packages

This module includes tools for using virtual environments and installing packages using pip.

See Also:

fabtools.python

Virtual environments

fabtools.require.python.**virtualenv**(*directory*, *system_site_packages=False*, *python=None*, *use_sudo=False*, *user=None*, *clear=False*)

Require a Python virtual environment.

```
from fabtools import require

require.python.virtualenv('/path/to/venv')
```

Installing packages

fabtools.require.python.**package**(*pkg_name*, *url=None*, ***kwargs*)

Require a Python package.

If the package is not installed, it will be installed using the pip installer.

```
from fabtools.python import virtualenv
from fabtools import require

# Install package system-wide
require.python.package('foo', use_sudo=True)

# Install package in an existing virtual environment
with virtualenv('/path/to/venv'):
    require.python.package('bar')
```

fabtools.require.python.**packages**(*pkg_list*, ***kwargs*)

Require several Python packages.

fabtools.require.python.**requirements**(*filename*, ***kwargs*)

Require Python packages from a pip requirements file.

fabtools.require.python.**pip**(*version=None*)

Require pip to be installed.

fabtools.require.python.**distribute**()

Require distribute to be installed.

5.2.12 fabtools.require.redis

Redis

This module provides high-level tools for managing Redis instances.

fabtools.require.redis.**installed_from_source**(*version='2.6.10'*)

Require Redis to be installed from source.

The compiled binaries will be installed in `/opt/redis-{version}/`.

fabtools.require.redis.**instance**(*name*, *version='2.6.10'*, ***kwargs*)

Require a Redis instance to be running.

The instance will be managed using supervisord, as a process named `redis_{name}`, running as the `redis` user.

```
from fabtools import require
from fabtools.supervisor import process_status

require.redis.installed_from_source()

require.redis.instance('db1', port='6379')
require.redis.instance('db2', port='6380')

print process_status('redis_db1')
print process_status('redis_db2')
```

See Also:

fabtools.supervisor and *fabtools.require.supervisor*

5.2.13 fabtools.require.rpm

RPM packages

This module provides high-level tools for managing CentOS/RHEL/SL packages and repositories.

See Also:

fabtools.rpm

fabtools.require.rpm.**package**(*pkg_name*, *repos=None*, *yes=None*, *options=None*)
Require a rpm package to be installed.

Example:

```
from fabtools import require

require.rpm.package('emacs')
```

fabtools.require.rpm.**packages**(*pkg_list*, *repos=None*, *yes=None*, *options=None*)
Require several rpm packages to be installed.

Example:

```
from fabtools import require

require.rpm.packages([
    'nano',
    'unzip',
    'vim',
])
```

fabtools.require.rpm.**nopackage**(*pkg_name*, *options=None*)
Require a rpm package to be uninstalled.

Example:

```
from fabtools import require

require.rpm.nopackage('emacs')
```

fabtools.require.rpm.**nopackages**(*pkg_list*, *options=None*)
Require several rpm packages to be uninstalled.

Example:

```
from fabtools import require

require.rpm.nopackages([
    'unzip',
    'vim',
    'emacs',
])
```

fabtools.require.rpm.**repository**(*name*)
Require a repository. Aimed for 3rd party repositories.

Name currently only supports EPEL and RPMforge.

Example:

```
from fabtools import require

# RPMforge packages for CentOS 6
require.rpm.repository('rpmforge')
```

5.2.14 fabtools.require.service

System services

This module provides high-level tools for managing system services. The underlying operations use the `service` command, allowing to support both upstart services and traditional SysV-style `/etc/init.d/` scripts.

See Also:

fabtools.service

fabtools.require.service.**started**(*service*)
Require a service to be started.

```
from fabtools import require

require.service.started('foo')
```

fabtools.require.service.**stopped**(*service*)
Require a service to be stopped.

```
from fabtools import require

require.service.stopped('foo')
```

fabtools.require.service.**restarted**(*service*)
Require a service to be restarted.

```
from fabtools import require

require.service.restarted('foo')
```

5.2.15 fabtools.require.shorewall

Shorewall firewall

See Also:

fabtools.shorewall

fabtools.require.shorewall.**firewall**(*zones=None*, *interfaces=None*, *policy=None*, *rules=None*, *routestopped=None*, *masq=None*)

Ensure that a firewall is configured.

Example:

```
from fabtools.shorewall import *
from fabtools import require

# We need a firewall with some custom rules
require.shorewall.firewall(
    rules=[
        Ping(),
        SSH(),
        HTTP(),
        HTTPS(),
        SMTP(),
        rule(port=1234, source=hosts(['example.com'])),
    ]
)
```

fabtools.require.shorewall.**started**()

Ensure that the firewall is started.

fabtools.require.shorewall.**stopped**()

Ensure that the firewall is stopped.

5.2.16 fabtools.require.supervisor

Supervisor processes

This module provides high-level tools for managing long-running processes using supervisor.

See Also:

fabtools.supervisor

fabtools.require.supervisor.**process**(*name*, ***kwargs*)

Require a supervisor process to be running.

Keyword arguments will be used to build the program configuration file. Some useful arguments are:

- `command`: complete command including arguments (**required**)
- `directory`: absolute path to the working directory
- `user`: run the process as this user
- `stdout_logfile`: absolute path to the log file

You should refer to the supervisor documentation for the complete list of allowed arguments.

Note: the default values for the following arguments differs from the `supervisor` defaults:

- `autorestart`: defaults to `true`
- `redirect_stderr`: defaults to `true`

Example:

```
from fabtools import require

require.supervisor.process('myapp',
    command='/path/to/venv/bin/myapp --config production.ini --someflag',
    directory='/path/to/working/dir',
    user='alice',
    stdout_logfile='/path/to/logs/myapp.log',
    )
```

5.2.17 fabtools.require.system

System settings

See Also:

fabtools.system

`fabtools.require.system.`**`hostname`**(*name*)
Require the hostname to have a specific value.

`fabtools.require.system.`**`sysctl`**(*key*, *value*, *persist=True*)
Require a kernel parameter to have a specific value.

Locales

`fabtools.require.system.`**`default_locale`**(*name*)
Require the locale to be the default.

`fabtools.require.system.`**`locale`**(*name*)
Require the locale to be available.

`fabtools.require.system.`**`locales`**(*names*)
Require the list of locales to be available.

5.2.18 fabtools.require.users

System users

See Also:

fabtools.user

`fabtools.require.users.`**`user`**(*name*, *comment=None*, *home=None*, *create_home=True*, *skeleton_dir=None*, *group=None*, *create_group=True*, *extra_groups=None*, *password=None*, *system=False*, *shell=None*, *uid=None*)
Require a user and its home directory.

```
from fabtools import require

# This will also create a home directory for alice
require.user('alice')

# Sometimes we don't need a home directory
require.user('mydaemon', create_home=False)
```

Note: This function can be accessed directly from the `fabtools.require` module for convenience.

`fabtools.require.users.`**`sudoer`**(*username*, *hosts='ALL'*, *operators='ALL'*, *passwd=False*, *commands='ALL'*)

Require sudo permissions for a given user.

Note: This function can be accessed directly from the `fabtools.require` module for convenience.

CHAPTER

SIX

HISTORY

6.1 Changelog

6.1.1 Version 0.11.0 (2013-02-15)

- Fix requiring an existing user (thanks to JonPeel)
- Upgrade default Redis version to 2.6.10
- Upgrade default Node.js version to 0.8.19
- Better support for remote hosts where sudo is not installed

6.1.2 Version 0.10.0 (2013-02-12)

- Enable/disable nginx sites (thanks to Sébastien Béal)
- Add support for SmartOS (thanks to Anthony Scalisi)
- Add support for RHEL/CentOS/SL (thanks to Anthony Scalisi)

6.1.3 Version 0.9.4 (2013-01-10)

- Add files missing in 0.9.3 (thanks to Stéfane Fermigier)

6.1.4 Version 0.9.3 (2013-01-08)

- Fix bugs in user creation (thanks pahaz and Stéphane Klein)
- Add support for group creation

6.1.5 Version 0.9.2 (2013-01-05)

- Add syntax highlighting in README (thanks to Artur Dryomov)

6.1.6 Version 0.9.1 (2013-01-04)

- Fix documentation formatting issues

6.1.7 Version 0.9.0 (2013-01-04)

- Improve user creation and modification
- Add support for BSD / OS X to `files.owner`, `files.group`, `files.mode` and `files.md5sum` (thanks to Troy J. Farrell)
- Improve PostgreSQL user creation (thanks to Troy J. Farrell and Axel Haustant)
- Add `reload` and `force_reload` operations to the `service` module (thanks to Axel Haustant)
- Fix missing import in `require.redis` (thanks to svevang and Sébastien Béal)
- Add `clear` option to Python virtualenv (thanks to pahaz)
- Upgrade default Redis version to 2.6.7
- Upgrade default Node.js version to 0.8.16
- Decrease verbosity of some operations
- Speed up functional tests

6.1.8 Version 0.8.1 (2012-10-26)

- Really fix pip version parsing issue
- Upgrade default pip version to 1.2.1

6.1.9 Version 0.8.0 (2012-10-26)

- Improve user module (thanks to Gaël Pasgrimaud)
- Fix locale support on Debian (thanks to Olivier Kautz)
- Fix version number in documentation (thanks to Guillaume Ayoub)
- Fix potential issue with pip version parsing

6.1.10 Version 0.7.0 (2012-10-13)

- Fix changed directory owner requirement (thanks to Troy J. Farrell)
- Add functions to get a file's owner, group and mode

6.1.11 Version 0.6.0 (2012-10-13)

- Add support for Node.js (thanks to Frank Rousseau)
- Fix dependency on Fabric >= 1.4.0 (thanks to Laurent Bachelier)

6.1.12 Version 0.5.1 (2012-09-21)

- Documentation and packaging fixes

6.1.13 Version 0.5 (2012-09-21)

- The `watch` context manager now allows you to either provide a callback or do an explicit check afterwards (**warning**: this change is not backwards compatible, please update your fabfiles)
- **Add support for some network-related operations:**
  - get the IPV4 address assigned to an interface
  - get the list of name server IP addresses
- The `services` module now supports both upstart and traditional SysV-style `/etc/init.d` scripts (thanks to Selwin Ong)
- The `virtualenv` context manager can now also be used with `local()` (thanks to khorn)
- The `supervisor` module now uses `update` instead of `reload` to avoid unnecessary restarts (thanks to Dan Fairs)
- Add support for OpenVZ containers (requires a kernel with OpenVZ patches)
- `pip` can now use a download cache
- Upgrade Redis version to 2.4.17
- Misc bug fixes and improvements
- Support for Ubuntu 12.04 LTS and Debian 6.0
- Documentation improvements

6.1.14 Version 0.4 (2012-05-30)

- Added support for requiring an arbitrary APT source
- Added support for adding APT signing keys
- Added support for requiring a user with a home directory
- Added vagrant helpers
- Fixed Python virtualenv context manager

6.1.15 Version 0.3.2 (2012-03-19)

- Fixed README formatting

6.1.16 Version 0.3.1 (2012-03-19)

- Fixed bug in functional tests runner

6.1.17 Version 0.3 (2012-03-19)

- Added support for Shorewall (Shoreline Firewall)
- Fixed Python 2.5 compatibility
- Refactored tests

6.1.18 Version 0.2.1 (2012-03-09)

- Packaging fixes

6.1.19 Version 0.2 (2012-03-09)

- Added support for hostname and sysctl (kernel parameters)
- Added support for Redis
- Simplified API for supervisor processes

6.1.20 Version 0.1.1 (2012-02-19)

- Packaging fixes

6.1.21 Version 0.1 (2012-02-19)

- Initial release

CHAPTER

SEVEN

DEVELOPMENT

7.1 Tests

7.1.1 Running tests

If you're using Python 2.7, you can launch the tests using the built-in unittest runner:

```
$ python -m unittest discover
```

If you're using Python 2.5 or 2.6, you'll need to install unittest2, and use the provided runner:

```
$ pip install unittest2
$ unit2 discover
```

Or you can run the tests on all supported Python versions using tox, which will take care of everything:

```
$ pip install tox
$ tox
```

7.1.2 Unit tests

Running unit tests requires the mock library.

7.1.3 Functional tests

Running functional tests requires Vagrant to launch virtual machines, against which all the tests will be run.

If Vagrant is not installed, the functional tests will be skipped automatically.

If Vagrant is installed, the default is to run the tests on all available base boxes. You can specify which base boxes should be used by setting the FABTOOLS_TEST_BOXES environment variable:

```
$ FABTOOLS_TEST_BOXES='ubuntu_10_04 ubuntu_12_04' tox -e py27
```

You can also use this to manually disable functional tests:

```
$ FABTOOLS_TEST_BOXES='' tox
```

CHAPTER

EIGHT

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

f