fabtools Documentation

Release 0.16.0

Ronan Amicel

November 13, 2013

Contents

CHAPTER 1

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 2

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 3

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 4

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)
- Archlinux

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

4.1 Automatic integration test result

CHAPTER 5

API Documentation

5.1 fabtools

5.1.1 fabtools.apache

fabtools.apache.**enable**(*config*)
Create link from /etc/apache2/sites-available/ in /etc/apache2/sites-enabled/

(does not reload apache config)

```
from fabtools import require

require.apache.enable('default')
```

See Also:

`fabtools.require.apache.enabled()`

fabtools.apache.**disable**(*config*)
Delete link in /etc/apache/sites-enabled/

(does not reload apache config)

```
from fabtools import require

require.apache.disable('default')
```

See Also:

`fabtools.require.apache.disabled()`

5.1.2 fabtools.cron

Cron tasks

This module provides tools to manage periodic tasks using cron.

fabtools.cron.**add_task**(*name*, *timespec*, *user*, *command*, *environment=None*)
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.

You can also provide an optional dictionary of environment variables that should be set when running the periodic command.

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')
```

5.1.3 `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*, *version=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',
    ])

    # Install a specific version
    fabtools.deb.install('emacs', version='23.3+1-1ubuntu9')
```

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.**apt_key_exists**(*keyid*)
Check if the given key id exists in apt keyring.

fabtools.deb.**add_apt_key**(*filename=None*, *url=None*, *keyid=None*, *keyserver='subkeys.pgp.net'*, *update=False*)
Trust packages signed with this public key.

Example:

```
    import fabtools

    # Varnish signing key from URL and verify fingerprint)
    fabtools.deb.add_apt_key(keyid='C4DEFFEB', url='http://repo.varnish-cache.org/debian/GPG-key.txt

    # Nginx signing key from default key server (subkeys.pgp.net)
    fabtools.deb.add_apt_key(keyid='7BD9BF62')

    # From custom key server
    fabtools.deb.add_apt_key(keyid='7BD9BF62', keyserver='keyserver.ubuntu.com')

    # From a file
    fabtools.deb.add_apt_key(keyid='7BD9BF62', filename='nginx.asc'
```

5.1.4 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')
```

fabtools.files.**uncommented_lines**(*filename*, *use_sudo=False*)
Get the lines of a remote file, ignoring empty or commented ones

5.1.5 fabtools.git

Git

This module provides low-level tools for managing Git repositories. You should normally not use them directly but rather use the high-level wrapper `fabtools.require.git.working_copy()` instead.

See Also:

fabtools.require.git

fabtools.git.**clone**(*remote_url*, *path=None*, *use_sudo=False*, *user=None*)
Clone a remote Git repository into a new directory.

Parameters

- **remote_url** (*str*) – URL of the remote repository to clone.
- **path** (*str*) – Path of the working copy directory. Must not exist yet.
- **use_sudo** (*bool*) – If `True` execute `git` with `fabric.operations.sudo()`, else with `fabric.operations.run()`.
- **user** (*str*) – If `use_sudo is True`, run `fabric.operations.sudo()` with the given user. If `use_sudo is False` this parameter has no effect.

fabtools.git.**fetch**(*path*, *use_sudo=False*, *user=None*)
Fetch changes from the default remote repository.

This will fetch new changesets, but will not update the contents of the working tree unless yo do a merge or rebase.

Parameters

- **path** (*str*) – Path of the working copy directory. This directory must exist and be a Git working copy with a default remote to fetch from.
- **use_sudo** (*bool*) – If `True` execute `git` with `fabric.operations.sudo()`, else with `fabric.operations.run()`.
- **user** (*str*) – If `use_sudo is True`, run `fabric.operations.sudo()` with the given user. If `use_sudo is False` this parameter has no effect.

fabtools.git.**pull**(*path*, *use_sudo=False*, *user=None*, *force=False*)
Fetch changes from the default remote repository and merge them.

Parameters

- **path** (*str*) – Path of the working copy directory. This directory must exist and be a Git working copy with a default remote to pull from.
- **use_sudo** (*bool*) – If `True` execute `git` with `fabric.operations.sudo()`, else with `fabric.operations.run()`.
- **user** (*str*) – If `use_sudo is True`, run `fabric.operations.sudo()` with the given user. If `use_sudo is False` this parameter has no effect.
- **force** (*bool*) – If `True`, append the `--force` option to the command.

`fabtools.git.`**`checkout`**(*path*, *branch='master'*, *use_sudo=False*, *user=None*, *force=False*)

Checkout a branch to the working directory.

Parameters

- **path** (*str*) – Path of the working copy directory. This directory must exist and be a Git working copy.
- **branch** (*str*) – Name of the branch to checkout.
- **use_sudo** (*bool*) – If `True` execute `git` with `fabric.operations.sudo()`, else with `fabric.operations.run()`.
- **user** (*str*) – If `use_sudo is True`, run `fabric.operations.sudo()` with the given user. If `use_sudo is False` this parameter has no effect.
- **force** (*bool*) – If `True`, append the `--force` option to the command.

5.1.6 `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')
```

5.1.7 `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*, *host='localhost'*, ***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')
```

5.1.8 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)
```

5.1.9 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='0.10.13'*)
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)
```

5.1.10 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**.

5.1.11 fabtools.oracle_jdk

Oracle JDK

This module provides tools for installing Oracle JDK

See Also:

fabtools.require.oracle_jdk

fabtools.oracle_jdk.**install_from_oracle_site**(*version='7u25-b15'*)
Download tarball from Oracle site and install JDK.

```
import fabtools

# Install Oracle JDK
fabtools.oracle_jdk.install_from_oracle_site()
```

fabtools.oracle_jdk.**version**()
Get the version of currently installed JDK.

Returns `None` if it is not installed.

5.1.12 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;'")
```

5.1.13 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')
```

5.1.14 fabtools.python

Python environments and packages

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

See Also:

fabtools.python_setuptools 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*, *pip_cmd='pip'*)

Check if pip is installed.

fabtools.python.**install_pip**(*python_cmd='python'*, *use_sudo=True*)

Install the latest version of pip, using the given Python interpreter.

```
import fabtools

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

Note: pip is automatically installed inside a virtualenv, so there is no need to install it yourself in this case.

Installing packages

fabtools.python.**is_installed**(*package*, *pip_cmd='pip'*)

Check if a Python package is installed (using pip).

Package names are case insensitive.

Example:

```
from fabtools.python import virtualenv
import fabtools

with virtualenv('/path/to/venv'):
    fabtools.python.install('Flask')
    assert fabtools.python.is_installed('flask')
```

fabtools.python.**install**(*packages*, *upgrade=False*, *use_mirrors=False*, *use_sudo=False*, *user=None*, *download_cache=None*, *quiet=False*, *pip_cmd='pip'*)

Install Python package(s) using pip.

Package names are case insensitive.

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=False*, *use_sudo=False*, *user=None*, *download_cache=None*, *quiet=False*, *pip_cmd='pip'*)

Install Python packages from a pip requirements file.

```
import fabtools

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

5.1.15 fabtools.python_setuptools

Python packages

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

See Also:

fabtools.python and *fabtools.require.python*

fabtools.python_setuptools.**package_version**(*name*, *python_cmd='python'*)

Get the installed version of a package

Returns `None` if it can't be found.

fabtools.python_setuptools.**is_setuptools_installed**(*python_cmd='python'*)

Check if setuptools is installed.

fabtools.python_setuptools.**install_setuptools**(*python_cmd='python'*, *use_sudo=True*)

Install the latest version of setuptools.

```
import fabtools

fabtools.python_setuptools.install_setuptools()
```

fabtools.python_setuptools.**install**(*packages*, *upgrade=False*, *use_sudo=False*, *python_cmd='python'*)

Install Python packages with `easy_install`.

Examples:

```
import fabtools

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

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

Note: most of the time, you'll want to use `fabtools.python.install()` instead, which uses `pip` to install packages.

5.1.16 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.**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'))
```

5.1.17 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

See Also:

fabtools.systemd

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.

5.1.18 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()`.

5.1.19 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

5.1.20 fabtools.system

System settings

OS detection

fabtools.system.**distrib_id**()
Get the OS distribution ID.

Returns one of `"Debian"`, `"Ubuntu"`, `"RHEL"`, `"CentOS"`, `"Fedora"`, `"Archlinux"`, `"SunOS"`...

Example:

```
from fabtools.system import distrib_id

if distrib_id() != 'Debian':
    abort(u"Distribution is not supported")
```

fabtools.system.**distrib_family**()
Get the distribution family.

Returns one of `debian`, `redhat`, `sun`, `other`.

fabtools.system.**distrib_release**()
Get the release number of the distribution.

Example:

```
from fabtools.system import distrib_id, distrib_release

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

fabtools.system.**distrib_codename**()
Get the codename of the Linux distribution.

Example:

```
from fabtools.deb import distrib_codename

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

fabtools.system.**distrib_desc**()
Get the description of the Linux distribution.

For example: `Debian GNU/Linux 6.0.7 (squeeze).`

Hardware detection

fabtools.system.**get_arch**()
Get the CPU architecture.

Example:

```
from fabtools.system import get_arch

if get_arch() == 'x86_64':
    print(u"Running on a 64-bit Intel/AMD system")
```

fabtools.system.**cpus**()
Get the number of CPU cores.

Example:

```
from fabtools.system import cpus

nb_workers = 2 * cpus() + 1
```

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.

5.1.21 fabtools.systemd

Systemd services

This module provides low-level tools for managing systemd services.

See Also:

fabtools.service

fabtools.systemd.**enable**(*service*)
Enable a service.

```
fabtools.enable('httpd')
```

Note: This function is idempotent.

---

fabtools.systemd.**disable**(*service*)
Disable a service.

```
fabtools.systemd.disable('httpd')
```

Note: This function is idempotent.

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

```
if fabtools.systemd.is_running('httpd'):
    print("Service httpd is running!")
```

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

```
if not fabtools.systemd.is_running('httpd'):
    fabtools.systemd.start('httpd')
```

Note: This function is idempotent.

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

```
if fabtools.systemd.is_running('foo'):
    fabtools.systemd.stop('foo')
```

Note: This function is idempotent.

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

```
if fabtools.systemd.is_running('httpd'):
    fabtools.systemd.restart('httpd')
else:
    fabtools.systemd.start('httpd')
```

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

```
fabtools.systemd.reload('foo')
```

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

fabtools.systemd.**start_and_enable**(*service*)
Start and enable a service (convenience function).

Note: This function is idempotent.

fabtools.systemd.**stop_and_disable**(*service*)
Stop and disable a service (convenience function).

Note: This function is idempotent.

5.1.22 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=None*, *skeleton_dir=None*, *group=None*, *create_group=True*, *extra_groups=None*, *password=None*, *system=False*, *shell=None*, *uid=None*, *ssh_public_keys=None*, *non_unique=False*)
Create a new user and its home directory.

If *create_home* is `None` (the default), a home directory will be created for normal users, but not for system users. You can override the default behaviour by setting *create_home* to `True` or `False`.

If *system* is `True`, the user will be a system account. Its UID will be chosen in a specific range, and it will not have a home directory, unless you explicitely set *create_home* to `True`.

If *shell* is `None`, the user's login shell will be the system's default login shell (usually `/bin/sh`).

ssh_public_keys can be a (local) filename or a list of (local) filenames of public keys that should be added to the user's SSH authorized keys (see `fabtools.user.add_ssh_public_keys()`).

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*, *ssh_public_keys=None*, *non_unique=False*)
Modify an existing user.

ssh_public_keys can be a (local) filename or a list of (local) filenames of public keys that should be added to the user's SSH authorized keys (see `fabtools.user.add_ssh_public_keys()`).

Example:

```
import fabtools

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

fabtools.user.**home_directory**(*name*)
Get the absolute path to the user's home directory

Example:

```
import fabtools

home = fabtools.user.home_directory('alice')
```

fabtools.user.**local_home_directory**(*name=''*)
Get the absolute path to the local user's home directory

Example:

```
import fabtools

local_home = fabtools.user.local_home_directory()
```

fabtools.user.**authorized_keys**(*name*)
Get the list of authorized SSH public keys for the user

fabtools.user.**add_ssh_public_key**(*name*, *filename*)
Add a public key to the user's authorized SSH keys.

filename must be the local filename of a public key that should be added to the user's SSH authorized keys.

Example:

```
import fabtools

fabtools.user.add_ssh_public_key('alice', '~/.ssh/id_rsa.pub')
```

fabtools.user.**add_ssh_public_keys**(*name*, *filenames*)
Add multiple public keys to the user's authorized SSH keys.

filenames must be a list of local filenames of public keys that should be added to the user's SSH authorized keys.

Example:

```
import fabtools

fabtools.user.add_ssh_public_keys('alice', [
    '~/.ssh/id1_rsa.pub',
    '~/.ssh/id2_rsa.pub',
])
```

fabtools.user.**add_host_keys**(*name*, *hostname*)
Add all public keys of a host to the user's SSH known hosts file

5.1.23 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`.

5.1.24 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.apache

Apache

This module provides high-level tools for installing the apache2 web server and managing the configuration of web sites.

fabtools.require.apache.**server**()
Require apache2 server to be installed and running.

```
from fabtools import require

require.apache.server()
```

fabtools.require.apache.**enabled**(*config*)
Ensure link to /etc/apache2/sites-available/config exists and reload apache2 configuration if needed.

fabtools.require.apache.**disabled**(*config*)
Ensure link to /etc/apache2/sites-available/config doesn't exist and reload apache2 configuration if needed.

fabtools.require.apache.**site**(*config_name*, *template_contents=None*, *template_source=None*, *enabled=True*, *check_config=True*, ***kwargs*)
Require an apache2 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 = '''
<VirtualHost *:%(port)s>
    ServerName %(hostname})s

    DocumentRoot %(document_root)s

    <Directory %(document_root)s>
        Options Indexes FollowSymLinks MultiViews

        AllowOverride All

        Order allow,deny
        allow from all
    </Directory>
</VirtualHost>
'''

require.apache.site(
    'example.com',
    template_contents=CONFIG_TPL,
    port=80,
    hostname='www.example.com',
    document_root='/var/www/mysite',
)
```

See Also:

fabtools.require.files.template_file()

5.2.2 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*, *version=None*)

Require a deb package to be installed.

Example:

```
from fabtools import require

# Require a package
require.deb.package('foo')

# Require a specific version
require.deb.package('firefox', version='11.0+build1-0ubuntu4')
```

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',
])
```

5.2.3 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.**directories**(*path_list*, *use_sudo=False*, *owner=''*, *group=''*, *mode=''*)

Require a list of directories to exist.

```
from fabtools import require
dirs=[
    '/tmp/mydir',
    '/tmp/mydear',
    '/tmp/my/dir'
]
require.directories(dirs, owner='alice', mode='750')
```

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.

5.2.4 fabtools.require.git

Git

This module provides high-level tools for managing Git repositories.

See Also:

fabtools.git

`fabtools.require.git.`**`command`**()

Require the git command-line tool.

Example:

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

require.git.command()
run('git --help')
```

`fabtools.require.git.`**`working_copy`**(*remote_url*, *path=None*, *branch='master'*, *update=True*, *use_sudo=False*, *user=None*)

Require a working copy of the repository from the `remote_url`.

The `path` is optional, and defaults to the last segment of the remote repository URL, without its `.git` suffix.

If the `path` does not exist, this will clone the remote repository and check out the specified branch.

If the `path` exists and `update` is `True`, it will fetch changes from the remote repository, check out the specified branch, then merge the remote changes into the working copy.

If the `path` exists and `update` is `False`, it will only check out the specified branch, without fetching remote changesets.

Parameters

- **remote_url** (*str*) – URL of the remote repository (e.g. https://github.com/ronnix/fabtools.git). The given URL will be the `origin` remote of the working copy.
- **path** (*str*) – Absolute or relative path of the working copy on the filesystem. If this directory doesn't exist yet, a new working copy is created through `git clone`. If the directory does exist *and* `update == True`, a `git fetch` is issued. If `path is None` the `git clone` is issued in the current working directory and the directory name of the working copy is created by `git`.
- **branch** (*str*) – Branch to check out.
- **update** (*bool*) – Whether or not to fetch and merge remote changesets.
- **use_sudo** (*bool*) – If `True` execute `git` with `fabric.operations.sudo()`, else with `fabric.operations.run()`.
- **user** (*str*) – If `use_sudo is True`, run `fabric.operations.sudo()` with the given user. If `use_sudo is False` this parameter has no effect.

5.2.5 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.

5.2.6 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 fabric.api import settings
from fabtools import require

with settings(mysql_user='root', mysql_password='s3cr3t'):
    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 fabric.api import settings
from fabtools import require

with settings(mysql_user='root', mysql_password='s3cr3t'):
    require.mysql.database('myapp', owner='dbuser')
```

5.2.7 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',
)
```

5.2.8 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='0.10.13'*)

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)
```

5.2.9 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')
```

5.2.10 fabtools.require.oracle_jdk

Oracle JDK

This module provides tools for installing Oracle JDK

See Also:

fabtools.oracle_jdk

fabtools.require.oracle_jdk.**installed**(*version='7u25-b15'*)

Require Oracle JDK to be installed.

```
from fabtools import require

require.oracle_jdk.installed()
```

5.2.11 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',
])
```

5.2.12 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')
```

5.2.13 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*, *createrole=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')
```

5.2.14 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*, *venv_python=None*, *use_sudo=False*, *user=None*, *clear=False*, *prompt=None*, *virtualenv_cmd='virtualenv'*, *pip_cmd='pip'*, *python_cmd='python'*)

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*, *pip_cmd='pip'*, *python_cmd='python'*, ***kwargs*)

Require a Python package.

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

Package names are case insensitive.

```
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*, *pip_cmd='pip'*, *python_cmd='python'*, ***kwargs*)
Require several Python packages.

Package names are case insensitive.

fabtools.require.python.**requirements**(*filename*, *pip_cmd='pip'*, *python_cmd='python'*, ***kwargs*)
Require Python packages from a pip requirements file.

fabtools.require.python.**pip**(*version='1.3.1'*, *pip_cmd='pip'*, *python_cmd='python'*)
Require pip to be installed.

If pip is not installed, or if a version older than *version* is installed, the latest version will be installed.

fabtools.require.python.**setuptools**(*version='0.7'*, *python_cmd='python'*)
Require setuptools to be installed.

If setuptools is not installed, or if a version older than *version* is installed, the latest version will be installed.

5.2.15 fabtools.require.redis

Redis

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

fabtools.require.redis.**installed_from_source**(*version='2.6.16'*)
Require Redis to be installed from source.

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

fabtools.require.redis.**instance**(*name*, *version='2.6.16'*, *bind='127.0.0.1'*, *port=6379*, ***kwargs*)
Require a Redis instance to be running.

The required Redis version will be automatically installed using `fabtools.require.redis.installed_from_source` if needed.

You can specify the IP address and port on which to listen to using the *bind* and *port* parameters.

> **Warning:** Redis is designed to be accessed by trusted clients inside trusted environments. It is usually not a good idea to expose the Redis instance directly to the internet. Therefore, with the default settings, the Redis instance will only listen to local clients.

If you want to make your Redis instance accessible to other servers over an untrusted network, you should probably add some firewall rules to restrict access. For example:

```
from fabtools import require
from fabtools.shorewall import Ping, SSH, hosts, rule

# The computers that will need to talk to the Redis server
REDIS_CLIENTS = [
    'web1.example.com',
    'web2.example.com',
]

# The Redis server port
```

```
REDIS_PORT = 6379

# Setup a basic firewall
require.shorewall.firewall(
    rules=[
        Ping(),
        SSH(),
        rule(port=REDIS_PORT, source=hosts(REDIS_CLIENTS)),
    ]
)

# Make the Redis instance listen on all interfaces
require.redis.instance('mydb', bind='0.0.0.0', port=REDIS_PORT)
```

See Also:

Redis Security

You can also use any valid Redis configuration directives as extra keyword arguments. For directives that can be repeated on multiple lines (such as `save`), you can supply a list of values.

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.instance('mydb')

print process_status('redis_mydb')
```

See Also:

fabtools.supervisor and *fabtools.require.supervisor*

The default settings enable persistence using periodic RDB snapshots saved in the */var/db/redis* directory.

You may want to use AOF persistence instead:

```
require.redis.instance('mydb', appendonly='yes', save=[])
```

In certain situations, you may want to disable persistence completely:

```
require.redis.instance('cache', port=6380, save=[])
```

See Also:

Redis Persistence

5.2.16 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')
```

5.2.17 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')
```

5.2.18 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.

5.2.19 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',
    )
```

5.2.20 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.

5.2.21 fabtools.require.users

System users

See Also:

fabtools.user

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

See `fabtools.user.create()` for a detailed description of arguments.

```
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)

# Require a user without shell access
require.user('nologin', shell='/bin/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 6

History

6.1 Changelog

6.1.1 Version 0.16.0 (2013-10-26)

- **Redis improvements**
  - Make bind and port arguments explicit
  - Improve documentation
  - Upgrade default version to 2.6.16
- **Python improvements**
  - Improve support for using specific Python interpreters (**warning**: API changes)
  - Expose low-level virtualenv operations
  - Improve pip installation
  - Switch from distribute to setuptools 0.7+ after project merge (**warning**: API changes)
  - Do not install *curl* and *python-dev* packages when setuptools is already installed (ponty)
  - Make package names case-insensitive in python.is_installed (ponty)
  - Fix pip version parsing when using `pythonbrew switch`
- Fix `require.system.locales` when a prefix is set
- Fix require.system.locale() on fresh Ubuntu systems
- Add optional environment variables in crontab
- Fix crontab permissions
- Allow special characters in MySQL password (Régis Behmo)
- Fix bug with some services not starting correctly (Chris Marinos)
- Add `getdevice_by_uuid` to the disk module (Bruno Adele)
- Fix implicit directory name in `git.working_copy` (iiie)
- Make `require.sysctl` robust to procps start failure

6.1.2 Version 0.15.0 (2013-07-25)

- Fix missing import in `user.local_home_directory()` (Sebastien Beal)
- **Improved Arch Linux support:**
  - Fix locale support in Arch Linux (Bruno Adele)
  - Add support for yaourt package manager in Arch Linux (Bruno Adele)
- **Improvements to the `redis` module:**
  - Fix Redis startup after reboot (Victor Perron)
  - Upgrade default Redis version to 2.6.14
- **Improvements to the `git` module:**
  - Add optional force parameter to git pull and checkout (Sebastien Beal)
- **Improvements to the `python` module:**
  - Add parameter to use a specific Python interpreter (Bruno Adele)
  - Stop using PyPI mirrors now that it has a CDN (Dominique Lederer)
- **Debian/Ubuntu improvements:**
  - Add optional version parameter to deb.install() (Anthony Scalisi)
  - Improved support for installing APT public keys (Santiago Mola)
- **SmartOS improvements (Andreas Kaiser):**
  - Fix md5sum on recent SmartOS
  - Fix bug in pkg.is_installed with certain package names
  - Add support for SmartOS in remote system identification
  - Add support for SmartOS in require.git.command()
- **RedHat improvements:**
  - Fix broken rpm.install() (Sho Shimauchi)
- **Oracle JDK improvements:**
  - Upgrade default version to 7u25-b15 (Sebastien Beal)
  - Fix Oracle JDK version parsing when OpenJDK is installed
  - Fix Oracle JDK installation on Debian squeeze (Stéphane Klein)
- Better tests documentation (thanks to Stéphane Klein)
- Add require.directories() (Edouard de Labareyre)
- Add support for Apache web server (Stéphane Klein)
- Upgrade default Node.js version to 0.10.13

6.1.3 Version 0.14.0 (2013-05-22)

Note: Fabtools now requires Fabric >= 1.6.0

- Upgrade default pip version to 1.3.1

- **Improved vagrant support:**
  - Add support for Vagrant 1.1 providers in functional tests
  - Also set `env.user` and `env.hosts` in `vagrant` context manager
- Add `fabtools.system.cpus` to get the host's CPU count
- Less verbose output
- Move OS detection functions to `fabtools.system`
- Better support for Red Hat based systems
- **Improvements to the `user` module:**
  - Fix home dir default behaviour in `require.user`
  - Add support for SSH authorized keys (Kamil Chmielewski)
  - Add support for SSH known hosts public keys
  - Add `non_unique` argument to user functions (Zhang Erning)
  - Get absolute path to the local user's home dir (Sebastien Beal)
- Use `SHOW DATABASES` to test existence of MySQL (Zhang Erning)
- **Improvements to the `git` module**
  - Expose lower level `fetch` operation (Andreas Kaiser)
  - Fix missing import in `require` module (Muraoka Yusuke)
  - Require `git` command line tool
- Use `ifconfig` as root in `network` module
- Update OpenVZ guest context manager for Fabric 1.6.0
- **Improvements to the `python` module:**
  - Improved detection of distribute
  - Add support for virtualenv `--prompt` option (Artem Nezvigin)
  - Allow relative path in `virtualenv` context manager
- **Improvements to the `oracle_jdk` module:**
  - Upgrade default Oracle JDK version to 7u21-b11 (Kamil Chmielewski)
  - Add support for Oracle JDK version 6 (Sebastien Beal)
- Fix broken `fabtools.deb.upgrade`
- Add support for Arch Linux packages (Bruno Adele)
- Add support for Linux disk partitions (Bruno Adele)
- Add OpenSSH server hardening (Adam Patterson)
- Add `systemd` module (Jakub Stasiak)
- **Improvements to the `redis` module:**
  - Fix broken Redis configuration (Victor Perron)
  - Upgrade default Redis version to 2.6.13
- Abort on nginx configuration errors

- Upgrade default Node.js version to 0.10.7

6.1.4 Version 0.13.0 (2013-03-15)

- Add support for managing remote git repositories (Andreas Kaiser)
- Add intersphinx to docs (Andreas Kaiser)
- Add HTTP proxy support to speed up functional tests
- Upgrade default Node.js version to 0.10.0
- Upgrade default Redis version to 2.6.11
- Upgrade default Oracle JDK version to 7u17-b02
- Fix vagrant support (thanks to Dominique Lederer and anentropic)

6.1.5 Version 0.12.0 (2013-03-04)

- Do not create home directory for system users
- Fix `pkg.is_installed` on SmartOS (thanks to Anthony Scalisi)
- Fix `system.get_arch` (thanks to Kamil Chmielewski)
- Add support for installing Oracle JDK (thanks to Kamil Chmielewski)
- Add support for creating Postgres schemas (thanks to Michael Bommarito)
- Fix `mysql.user_exists` (thanks to Serge Travin)

6.1.6 Version 0.11.0 (2013-02-15)

- Fix requiring an existing user (thanks to Jonathan Peel)
- 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

6.1.7 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)

6.1.8 Version 0.9.4 (2013-01-10)

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

6.1.9 Version 0.9.3 (2013-01-08)

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

6.1.10 Version 0.9.2 (2013-01-05)

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

6.1.11 Version 0.9.1 (2013-01-04)

- Fix documentation formatting issues

6.1.12 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

6.1.13 Version 0.8.1 (2012-10-26)

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

6.1.14 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

6.1.15 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

6.1.16 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)

6.1.17 Version 0.5.1 (2012-09-21)

- Documentation and packaging fixes

6.1.18 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

6.1.19 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

6.1.20 Version 0.3.2 (2012-03-19)

- Fixed README formatting

6.1.21 Version 0.3.1 (2012-03-19)

- Fixed bug in functional tests runner

6.1.22 Version 0.3 (2012-03-19)

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

6.1.23 Version 0.2.1 (2012-03-09)

- Packaging fixes

6.1.24 Version 0.2 (2012-03-09)

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

6.1.25 Version 0.1.1 (2012-02-19)

- Packaging fixes

6.1.26 Version 0.1 (2012-02-19)

- Initial release

CHAPTER 7

Development

7.1 Tests

7.1.1 Running tests

Using tox

The preferred way to run tests is to use tox. It will take care of everything and run the tests on all supported Python versions, each in its own virtual environment:

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

You can ask tox to run tests only against specific Python versions like this:

```
$ tox -e py25
$ tox -e py26,py27
```

Note: If tox ever gives you trouble, you can ask it to recreate its virtualenvs by using the `-r` (or `--recreate`) option. Alternatively, you can start over completely by removing the `.tox` directory.

Using unittest

Alternatively, 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 first, then use the provided `unit2` command:

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

7.1.2 Unit tests

The goal of the unit tests is to test the internal logic of fabtools functions, without actually running shell commands on a target system.

Running unit tests requires the mock library.

7.1.3 Functional tests

The goal of the functional tests is to test that fabtools functions have the expected effect when run against a real target system.

Functional tests are ordinary fabfiles, contained in the `fabtools/tests/fabfiles/` folder.

Requirements

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.

Selecting base boxes

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, and run only the unit tests:

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

Selecting which tests to run

If you only want to execute specific fabfiles during a test run, you can select them using the `FABTOOLS_TEST_INCLUDE` environment variable:

```
$ FABTOOLS_TEST_INCLUDE='oracle.py redis.py' tox -e py27
```

If you want to exclude some fabfiles from a test run using the `FABTOOLS_TEST_EXCLUDE` environment variable:

```
$ FABTOOLS_TEST_EXCLUDE='nginx.py git.py' tox -e py27
```

Debugging functional tests

When you're working on a test fabfile, sometimes you'll want to manually inspect the state of the Vagrant VM. To do that, you can prevent it from being destroyed at the end of the test run by using the `FABTOOLS_TEST_NODESTROY` environment variable:

```
$ FABTOOLS_TEST_NODESTROY=1 tox -e py27
$ cd fabtools/tests
$ vagrant ssh
```

CHAPTER 8

Indices and tables

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

Python Module Index

f