. Documentation

Release

Author

November 10, 2014

Contents

CHAPTER 1

What is Fabliip?

Fabliip is a set of functions aimed at helping developers deploy their websites. It is meant to be used as part of Fabric scripts to ease for example the backup of a database, the upgrade of a remote git repository, etc.

CHAPTER 2

Installation

Install it with pip:

```
pip install fabliip
```

CHAPTER 3

Documentation

3.1 fabliip package

3.1.1 Subpackages

fabliip.database package

Submodules

fabliip.database.mysql module

fabliip.database.mysql.**dump**(*backup_path*, *database_name*, *user='root'*, *host=None*, *password=None*)
Backup MySQL database as a MySQL archive. If host is set to None, 127.0.0.1 will be used. If password is set to None, a prompt will ask a password.

fabliip.database.mysql.**get_password_param**(*user*, *password*)
Ask a password in the prompt

fabliip.database.mysql.**restore**(*backup_path*, *database_name*, *user='root'*, *host=None*, *password=None*)
Restore MySQL database. If host is set to None, 127.0.0.1 will be used. If password is set to None, a prompt will ask a password.

fabliip.database.pgsql module

fabliip.database.pgsql.**dump**(*backup_path*, *database_name*, *user='postgres'*, *host=None*, *password=None*)
Backs up the given database to the given file as a PostgreSQL archive. If host is set to None, a local connection will be used, so you'll need to be able to sudo to the given user. Otherwise, a standard password connection will be used and the user will be asked for a password.

Module contents

fabliip.drupal package

Module contents

Functions for Drupal sites.

The *drupal_root* environment variable is used in some functions of this module to run the commands in the correct directory. Make sure it is set to the root directory of your Drupal project.

This module requires *drush* to be installed on the remote server.

fabliip.drupal.**clear_cache**()
Clears the Drupal cache.

fabliip.drupal.**drush**(*command*)
Runs a drush command on the server.

Requires the *drupal_root* environment variable to be set.

fabliip.drupal.**enable_disable_modules**(*site=None*)
Enables and disables modules on the Drupal install to reflect the status of the modules.enabled/disabled files.

The optional *site* parameter allows you to have a multisite project with a global modules.enabled/disabled file and a site-specific modules.site.enabled/disabled file.

fabliip.drupal.**set_maintenance_mode**(*enabled*)
Enables or disables the maintenance mode.

fabliip.vcs package

Submodules

fabliip.vcs.git module

fabliip.vcs.git.**get_commit_messages**(*first_commit*, *last_commit*)
Returns all commit messages between first_commit and last_commit in an abbreviated form.

fabliip.vcs.git.**get_latest_commit**(*run_locally=True*)
Return the commit identified by the current HEAD.

Arguments: run_locally – Whether to get the latest local or remote HEAD (default True)

fabliip.vcs.git.**get_latest_tag**(*commit='HEAD'*, *run_locally=True*)
Return the latest reachable tag from the given commit.

Arguments: commit – The name of the commit to use (tag, hash, etc) (default HEAD) run_locally – Whether to get the latest local or remote tag (default True)

fabliip.vcs.git.**push_tag**(*tag*, *remote='origin'*)
Pushes the given tag to the given remote.

fabliip.vcs.git.**update_remote_repository_root**(*tag*)
Fetches the latest git objects on the remote, checks out the given tag and updates the submodules if necessary.

Requires the *repository_root* environment variable to be set.

Module contents

3.1.2 Submodules

3.1.3 fabliip.decorators module

fabliip.decorators.**multisite**(*func*)

Mark a task as being multisite.

Multisite tasks allow you to define a global configuration for multiple sites and specify the site to use when you run your command. All the dictionary keys of the given site will then be made available in the *env* variable. The structure is *env.sites[site_name][env_name][configuration_key] = configuration_value*, where *env_name* is the name of the decorated function. Also the decorated function must take a parameter *site*.

Here's an example with sites A and B, both having prod and staging environments:

```
env.sites = {
    'site_a': {
        'prod': {
            'project_root': '/var/www/site_a/prod/'
        },
        'staging': {
            'project_root': '/var/www/site_a/staging/'
        }
    },
    'site_b': {
        'prod': {
            'project_root': '/var/www/site_b/prod/'
        },
        'staging': {
            'project_root': '/var/www/site_b/staging/'
        }
    }
}

@task
@multisite
def prod(site):
    # You could also define hosts in the `env.sites` dictionary if
    # they're different from one site to another
    # If the site is set to `site_a`, the decorator will take the
    # contents from env.sites['site_a']['prod'] and make it available
    # in the `env` variable
    env.hosts = ['www.my_host.com']

@task
@multisite
def staging(site):
    # You could also define hosts in the `env.sites` dictionary if
    # they're different from one site to another
    env.hosts = ['staging.my_host.com']
```

3.1.4 fabliip.file module

fabliip.file.**file_exists**(*path*)

Checks if the given path exists on the host and returns True if that's the case, False otherwise.

fabliip.file.**ls**(*path*)
Return the list of the files in the given directory, omitting . and ...

Arguments: path – The path of the directory to get the files from

3.1.5 fabliip.releases module

This module provides a deployment structure similar to what Capistrano does. By default, the project layout looks like that:

```
current -> releases/20140830180015_1.2.3  -- Symlink to the current release
releases/
    20140830151210_1.2.2/
    20140831180015_1.2.3/
repository.git/                           -- Git repository containing the project
shared/                                   -- Shared files not specific to a release
    config.yml
    media/
VERSION                                   -- The version of the current release
```

When you create a new release, the code from the *repository.git* directory is archived in a new directory named after the current date and the version you're deploying. After that, the symlink *current* is switched to the new release.

The following variables must be defined in Fabric's env for this module to work:

releases_root Path to the releases/ directory

repository_root Path to the repository.git/ directory

shared_root Path to the shared/ directory

shared_files Dictionary of shared files {target: link_name}

The following variable should be defined if you want to use the various methods of this module without having to pass it around :

release_name Name of the release

fabliip.releases.**activate_release**(**args*, ***kwargs*)
Activate the given release by making the `current` symlink point to it.

If `release_name is` not given, try to get it from `fabric.api.env.release_name`.

Arguments: release_name – The name of the release (usually a date like YmdHMS)

fabliip.releases.**clean_old_releases**(**args*, ***kwargs*)
Remove the old release directories from the releases directory, keeping x releases defined by the `keep` parameter.

Arguments: keep – The number of releases to keep

fabliip.releases.**create_release**(**args*, ***kwargs*)
Create the directory for a new release and extract the contents from the git repository at the given tag and put them in this directory.

Arguments: release_name – The name of the release (usually a date like YmdHMS) tag – The tag to install in this release

fabliip.releases.**determine_release_name**(*release_name*)
Try to get `release_name` from `fabric.api.env.release_name`.

fabliip.releases.**get_currently_installed_version**()
Return the currently installed version (tag) by reading the contents of the VERSION file, or None if the VERSION file could not be read.

fabliip.releases.**get_release_path**(*release_name=None*)
Return the absolute path to the directory of the given release.

If `release_name is` not given, try to get it from `fabric.api.env.release_name`.

Arguments: release_name – The name of the release (usually a date like YmdHMS)

fabliip.releases.**get_releases**()
Return the list of releases on the server, sorted by oldest to newest.

fabliip.releases.**invalidate_last_release**()
Invalidate the last made release so that it won't be a target for a rollback.

fabliip.releases.**link_shared_files**(**args*, ***kwargs*)
Create or update links to shared files defined in the `shared_files` env variable.

If `release_name is` not given, try to get it from `fabric.api.env.release_name`.

Arguments: release_name – The name of the release (usually a date like YmdHMS)

fabliip.releases.**update_version_file**(**args*, ***kwargs*)
Update the VERSION file with the given version.

3.1.6 fabliip.signals module

This module allows you to send/receive signals during your deployment process.

You can use it to create modules separated from your main fabfile and make them respond to certain signals. For example consider the following example, in your `fabfile.py`:

```
# This will trigger a pre_deploy and post_deploy signal
@signals.register
def deploy():
    deploy_my_app()
```

In your separate module, use the `on` decorator to hook on this signal:

```
# This goes in your separate module
@signals.on('fabfile.pre_deploy')
def my_hook():
    print("This is called at the beginning of the deploy task")
```

You can also emit signals independently:

```
def deploy():
    signals.emit('pre_deploy_my_app')
    deploy_my_app()
    signals.emit('hurray_my_app_is_deployed')
```

The `task()` decorator wraps the default `fabric.api.task()` decorator with the `register()` signal, allowing you to intercept pre/post signals without the need of adding the `register()` decorator or firing the signals yourself.

fabliip.signals.**emit**(*signal*)
Make all the receivers of this signal aware that it's been sent.

fabliip.signals.**on**(*signal*)
Decorator that will call the given function when the given signal is emitted.

fabliip.signals.**register**(*function*)
Decorator that will emit pre and post signals before and after the function is executed. The signals are named after the function so if your function is named `seek_the_holy_grail()`, the signal `pre_seek_the_holy_grail` will be fired before your function gets executed and the `post_seek_the_holy_grail` signal will be fired after your function has been executed.

fabliip.signals.**task**(*function*)
Convenience decorator that wraps the default Fabric task decorator with the `register()` decorator.

3.1.7 fabliip.utils module

fabliip.utils.**local_run_wrapper**(**args*, ***kwargs*)
Wrapper around fabric's *local* command with the capture parameter always enabled.

3.1.8 fabliip.version module

fabliip.version.**get_version_files**(*from_version*, *to_version*, *directory*)
Get a list of the files named after a version number (eg. 0.1.py, 0.1.2.py, etc) from the given directory and return a sorted list of tuples (version, file) of files which version number match the from_version (non inclusive) and to_version (inclusive) limits.

Arguments: from_version – A string representing the low version number (eg. 1.0) to_version – A string representing the high version number (eg. 1.2.6) directory – The path to the directory that holds the version files

3.1.9 Module contents

CHAPTER 4

Indices and tables

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

f