fabdeploit Documentation

Release 0.1.0

David Danier

Mar 18, 2018

Contents

Using git for deployments is a great solution for agile development processes and is implemented by different people with many different flavours. Any solution has one thing in common: You have to move the whole history of your project over to your server. This may be ok for some setups, but has - besides giving many internals to your clients - some disadvantages like:

- Rollbacks are more complex, as you need to know which commit is the last release (may be solved using tags)
- You have to copy the whole repository over to the server, so you cannot skip any files or add new ones only needed on the server (like aggregated CSS/JS)

fabdeploit tries to solve these issues by using a seprate release branch, not wired to the normal git branches and history. This release branch will only contain release history (one commit for every release/deployment) and allows you to use filters to change the contents of the commit tree. This way you are able to have a very slick deployment process only containing what is necessary, you may even add new files not contained in your normal history.

In addition fabdeploit contains helpers for professional deployment of some common CMS/Frameworks we use. This will help you implementing a clean process of common deployments, including enabling maintenance mode, running database migrations, clearing the caches, …

CHAPTER 1

About fabdeploit release history

As stated before fabdeploit allows you to use git for your releases while not pushing all your history to your or your customers server. It uses GitPython to manage release commits in a distinct branch which gets pushed to the server. This branch only contains release history.

Your history may then look like:

```
...
|
*    Commit after release (commit lives in "master")
|
| *  The actual release commit (living in "release/master")
| |
* |  Normal commit, it's tree is used for release commit above ("master")
| |
* |  Another normal commit ("master")
| |
| *  Another release commit ("release/master")
| |
* |  Another normal commit, tree used for release above ("master")
| |
| |  ...I think you should see the pattern now ;--)
```

As you may see release commits **never** have any parents except the previous release. This way your history will never be pushed directly to the server, as long as you only push the release branch. If you look at the release branch ("release/master") the history looks like:

```
*  The actual release commit (living in "release/master")
|
*  Another release commit ("release/master")
|
```

Each release commit may be run through a filter, which allows you to remove files from the release commit tree or add new files to the release commit. This may be used for many different use cases, like:

- Skip files not appropriate for the server (documentation, PSD files, . . .)

- Converting SASS to CSS and then add the new CSS files
- Create CSS/JS aggregates

CHAPTER 2

Reference:

2.1 fabdeploit reference

fabdeploit utilizes multiple helper classes to implement the desired behavior for a clean deployment process.

Each class provides some options (instance attributes) to setup where files live, which commands to call, etc. These options may be provided when constructing the class or be set as class attributes when using subclasses. Bold options are mandatory.

Each class will in addition provide some methods to do the real work (like running some command). You have to call these methods in the right way to get things working like intended.

Below you will find an overview of all classes, options, methods and an example workflow.

2.1.1 Git

Options

local_repository_path (mandatory) Path to the local repository.

remote_repository_path (mandatory) Remote repository path.

release_author You may set some author for the release commits, if None original author will be used.

release_branch (mandatory) The branch you want to release ("production", "staging"). fabdeploit will create an "release/…"-branch for the release history ("release/production")

release_commit_filter_class A filter class to be used when creating the release commit tree. The filter may remove files from or add files to the commit tree. A filter should be a subclass of GitFilter and implement the filter()-method. The filter may then use self.add() and self.remove() to change the tree. Use self.filtered_tree to access the tree itself.

Methods

pull_origin() Should be called before any additional actions, ensures the local repository is in sync with the origin.

pull() Calls self.pull_origin() if origin existy in local repository. Should be used instead of pull_origin() for a more generic appproach.

create_release_commit(message=None) Creates the new release commit by copying the last commit of release_branch into the release branch (applying the filter). An optional commit message may be provided.

tag_release(tag_name) Created a tag for the commit created by create_release_commit()

merge_release_back() Created a new commit in release_branch with the commit created by create_release_commit() and the last commit in this branch as parents. Can be used to push release commits back into the normal branches. SHOULD BE USED WITH CARE. SHOULD NEVER BE USED WHEN FILTERS ARE IN PLACE.

release(message=None, tag_name=None, merge_back=False) Calls create_release_commit(), tag_release() and merge_release_back() according to the parameters provided.

push_release() Pushes the commit created by create_release_commit() to the release git URL (see remote_repository_path). Created the repository if necessary.

push_origin() Pushes the release branch to origin.

push() Pushed the release branch to the release git URL and origin if origin exists in the local git repository.

switch_release(commit=None) Switched to a particular commit on the remote server. You may specify the commit yourself, if None the commit created by create_release_commit() will be used. This must be run to apply all changes on the server.

webserver_harden_remote_git() (Should be run after push)

Tries to disable access to the .git folder for common configuration. This is currently done by writing a .htaccess (for Apache) and setting permissions to rwx—— (700) so no other user is allowed to access to directory.

Note: *You should never push directly into the document root!* It is always better to have the document root being a sub-folder in your git repository for multiple good reasons (for example as "htdocs"). Anyways there are situations where you cannot ensure such a sane setup. This method helps you not shooting yourself in the foot. Please make sure it works for your setup, for example by browsing to www.your-domain.com/.git/config, you should get "access denied".

Example Workflow

```
git = Git(local_repository_path="...", remote_repository_path="...", release_branch=
↪"...")
git.pull()  # Make sure we have all remote changes
git.create_release_commit("New release")
git.push()  # Make sure the new release is copied to origin and server
# git.webserver_harden_remote_git() # No web access to .git
git.switch_release()  # Apply all file changes
```

Example GitFilter

```
class MyGitFilter(GitFilter):
    def filter(self):
        self.add('some/new/file.txt')
        self.remove('docs')  # remove whole docs directory
        self.remove('path/to/file.psd')

git = Git(release_commit_filter_class=MyGitFilter, "...")
# ...
```

2.1.2 Virtualenv

Options

python_commands List/tuple of possible python commands.

pip_commands List/tuple of possible pip commands.

virtualenv_commands List/tuple of possible virtualenv commands. If not found virtualenv will be installed from git by init().

virtualenv_path (mandatory) Path to virtualenv on the remote server.

virtualenv_download_branch: Branch to install virtualenv from when no virtualenv is available on the server. See init().

requirements_file Path to the requirements file used to install()/update() the virtualenv. Mandatory only if these methodes are called.

Note: Use Virtualenv2 or Virtualenv3 to setup python_commands/pip_commands/virtualenv_commands according to the proper python versions.

Methods

init() Set the virtualenv up by calling virtualenv bin for virtualenv_path. If no virtualenv bin is available on the server it will clone the virtualenv github repository to setup all manually. The git clone will use to branch specified in virtualenv_download_branch, if you need some particular version.

install()/update() Updates the virtualenv according to requirements_file.

git (Property) Allows you to put the entiry virtualenv under git control. May be used to rollback the virtualenv easily. Returns VirtualenvGit instance, see code for more details.

python_bin() Returns path to virtualenv python, may be used for running own commands.

Example Workflow

```
virtualenv = Virtualenv(virtualenv_path="...", requirements_file="...")
virtualenv.init()  # Make sure the virtualenv exists
virtualenv.update()  # Apply requirements_file

run("{python_bin} --version".format({
    "python_bin": virtualenv.python_bin()
}))
```

2.1.3 Django

Options

manage_path (mandatory) Path to manage.py

virtualenv (Constructor only, mandatory) Instance of Virtualenv

Methods

run() Run any manage.py command.

collectstatic()/syncdb()/migrate() Runs the appropriate manaage.py command.

Example Workflow

```
virtualenv = Virtualenv(virtualenv_path="...", requirements_file="...")
django = Django(virtualenv=virtualenv, manage_path="...")

django.collectstatic()
django.run("some_command", "---noinput", "--someparam")
```

2.1.4 Drupal

Note: This class uses drush, drush will be installed from git by init().

Options

php_commands List of possible PHP commands.

php_ini_path Optional specify your own php.ini (see php bin option "-c")

drupal_path (mandatory) Path to drupal installation.

drush_path (mandatory) Path where drush will be installed.

drush_download_branch Git branch to be downloaded for installation. Defauls to "6.x", this drush version is compatible with Drupal 6.x and 7.x.

Methods

init() Installs drush into drush_path() by downloading from github.

run() Run any drush command.

cache_clear()/updatedb()/pm_enable()/pm_disable()/variable_set() Shorthands for some drush commands.

maintenance_enable()/maintenance_disable() Enable/disable maintenance mode.

drush_bin() Returns path or drush binary, may be used to run own scripts.

Example Workflow

```
drush = Django(drupal_path="...", drush_path="...")
drush.maintenance_enable()
drush.cache_clear()
drush.updatedb()
drush.run("somecommand", "param1", "param2")
drush.maintenance_disable()
```

2.1.5 Magento

Options

php_commands List of possible PHP commands.

php_ini_path Optional specify your own php.ini (see php bin option "-c")

magento_path Path to magento installtion.

Methods

run() Run any shell command (see magento shell/ path).

compiler_compile()/indexer_index()/indexer_reindexall()/log_clean() Shorthands for some shell commands.

maintenance_enable()/maintenance_disable() Enable/disable maintenance mode.

shell_command_bin() Return base command for additional (/own) shell commands.

Example Workflow

```
magento = Magento(magento_path="...")
magento.maintenance_enable()
magento.log_clean()
magento.run("clear_cache.php")
magento.run("apply_migrations.php")
magento.maintenance_disable()
```

CHAPTER 3

Examples:

3.1 fabdeploit Examples

3.1.1 Basic example

```
import os
import git
import fabdeploit
from fabric.api import *


REPO_PATH = os.path.dirname(os.path.realpath(os.path.abspath(__file__)))
os.chdir(REPO_PATH)

env.hosts = ['some_hostname']
env.use_ssh_config = True  # allows you to specify some_hostname in .ssh/config


@task
def deploy():
    class GitFilter(fabdeploit.GitFilter):
        def filter(self):
            for obj in self.filtered_tree:
                if not obj.name in ('web', 'scripts'):
                    self.remove(obj.name)
            with lcd(self.repo.working_tree_dir):
                local('sass web/scss/style.scss web/css/style.css')
                self.add('web/css/style.css')


    class Git(fabdeploit.Git):
        local_repository_path = REPO_PATH
        remote_repository_path = 'path/to/htdocs'
        release_branch = 'production'
        release_author = 'Team23 GmbH & Co. KG <info@team23.de>'
        release_commit_filter_class = GitFilter

    git = Git()
    git.pull()
    git.release()
    git.push()
    # TODO: enable maintenance
    git.switch_release()
    # TODO: run deployment jobs (like clear cache, db migrations, ...)
    # TODO: disable maintenance
```

3.1.2 Django example

```
import os
import git
import fabdeploit
import posixpath
from fabric.api import *


REPO_PATH = os.path.dirname(os.path.realpath(os.path.abspath(__file__)))
os.chdir(REPO_PATH)

env.hosts = ['some_hostname']
env.use_ssh_config = True  # allows you to specify some_hostname in .ssh/config

@task
def deploy():
    REMOTE_PATH = 'path/to/htdocs'

    class GitFilter(fabdeploit.GitFilter):
        def filter(self):
            for obj in self.filtered_tree:
                if not obj.name in ('web', 'scripts'):
                    self.remove(obj.name)
            with lcd(self.repo.working_tree_dir):
                local('sass web/scss/style.scss web/css/style.css')
                self.add('web/css/style.css')


    class Git(fabdeploit.Git):
        local_repository_path = REPO_PATH
        remote_repository_path = REMOTE_PATH
        release_branch = 'production'
        release_author = 'Team23 GmbH & Co. KG <info@team23.de>'
        release_commit_filter_class = GitFilter

    class Virtualenv(fabdeploit.Virtualenv2):
        virtualenv_path = posixpath.join(REMOTE_PATH, '_env')
        requirements_file = posixpath.join(REMOTE_PATH, 'PYTHON_REQUIREMENTS')

    class Django(fabdeploit.Django):
        manage_path = posixpath.join(REMOTE_PATH, 'manage.py')

    git = Git()
    virtualenv = Virtualenv()
    django = Django(virtualenv=virtualenv)

    git.pull()
    commit = git.release()
    git.push()
    # TODO: enable maintenance
    git.switch_release()
    virtualenv.init()
    virtualenv.update()
    # make sure we can rollback virtualenv, too
    virtualenv.git.commit(tag='release/%s' % commit.hexsha)
    django.migrate()
    django.collectstatic()
    # TODO: run more deployment jobs?
    # TODO: disable maintenance
```