viki-fabric-helpers Documentation

Release 0.0.1

Viki Inc.

June 03, 2014

Contents

A collection of Fabric helper functions used in some of Viki's projects.

CHAPTER 1

Installation

1.1 Installation

1.1.1 pip (requirements.txt)

Append this line to *requirements.txt*:

```
viki-fabric-helpers==0.0.1
```

Followed by running:

```
pip install -r requirements.txt
```

1.1.2 pip (command line installation)

From the command line:

```
pip install viki-fabric-helpers
```

1.1.3 Getting the code (via Github)

Cloning the repository:

```
git clone https://github.com/viki-org/viki-fabric-helpers.git
```

Downloading a tarball:

```
wget -O viki-fabric-helpers.tar.gz https://github.com/viki-org/viki-fabric-helpers/tarball/master
```

Downloading a zipball:

```
wget -O viki-fabric-helpers.zip https://github.com/viki-org/viki-fabric-helpers/zipball/master
```

CHAPTER 2

Usage

2.1 *viki.fabric.git* - A short guide

This page covers the use of the *viki.fabric.git* module. More detailed documentation for individual functions in this module can be found in the *API Documentation*.

Our focus here will be on running the *setup_server_for_git_clone* function. This function is used to setup a server for Git remote operations (such as cloning) involving secret repositories and assumes the following:

- Git remote operations are carried out using SSH
- An SSH private key is used for authentication to gain access to the secret repository

2.1.1 Configuration

For any code which imports the *viki.fabric.git* module, you will need to create a YAML file with some keys:

ssh_private_key

Basename of the SSH private key to copy to the server.

ssh_public_key

Basename of the SSH public key to copy to the server.

ssh_keys_local_copy_dir

Folder storing the *ssh_private_key* and *ssh_public_key* files on **your local machine**.

The path to this folder can be relative to where the Python script that imports the *viki.fabric.git* module is run, or an absolute path.

ssh_keys_dir

Folder on the remote server to copy the *ssh_private_key* and *ssh_public_key* files to.

NOTE: This folder is **relative to the $HOME directory** of the **remote user** during Fabric execution. This should normally be set to the string *'.ssh'*.

git_ssh_script_name

Basename of a template git wrapper script on **your local machine**. What this script should contain is outlined later in *this subsection*.

This file will be copied to the *$HOME* directory of the user on the remote server (for such a server and user involved in a Fabric task). You can set this to any valid filename. My personal preference for this value is the string *'gitwrap.sh'*.

git_ssh_script_local_folder

Folder **on your local machine** containing the *git_ssh_script_name* file.

The path to this folder can be relative to where the Python script that imports the *viki.fabric.git* module is run, or an absolute path.

2.1.2 Example YAML file (and what it implies)

```
ssh_private_key: "id_github_ssh_key"
ssh_public_key: "id_github_ssh_key.pub"
ssh_keys_local_copy_dir: "github-ssh-keys"
ssh_keys_dir: ".ssh"
git_ssh_script_name: "gitwrap.sh"
git_ssh_script_local_folder: "templates"
```

Suppose that Fred, a user of our library, has a Python Fabric File located at */home/fred/freds-repo/fabfile.py*, which he runs from the */home/fred/freds-repo* folder. Based on the values in the YAML file:

- There should be a */home/fred/freds-repo/github-ssh-keys* folder containing the *id_github_ssh_key* and *id_github_ssh_key.pub* SSH keypair.
- This SSH keypair will be copied to the *$HOME/.ssh* folder on the server during execution of the *setup_server_for_git_clone* Fabric task
- There is a */home/fred/freds-repo/templates* folder containing the *gitwrap.sh* file. We shall take a look at what this file should contain in the next section.

2.1.3 Git SSH Wrapper file

This is the file specified by the value of the *git_ssh_script_name* YAML key, and should contain the following code:

```
#!/bin/bash

ssh -i {{ ssh_private_key_path }} $@
```

The *{{ ssh_private_key_path }}* part of the code will be replaced by the *setup_server_for_git_clone* Fabric task before the script is copied to the server (A temporary file or similar is used, so your file will not be accidentally modified by this task).

2.1.4 Running the *setup_server_for_git_clone* Fabric task

Assume that our imaginary user Fred

- has everything setup as we mentioned above
- has his YAML file located at */home/fred/freds-repo/config/viki_fabric_git.yaml*
- runs the */home/fred/freds-repo/fabfile.py* file (contents right below) from the */home/fred/freds-repo* folder, using this command:

```
fab -H hostOne,hostTwo freds_fabric_task
```

Contents of */home/fred/freds-repo/fabfile.py* Fabric script:

```
from fabric.api import env, task

import os.path
import viki.fabric.git as fabric_git

# Fred uses SSH config
env.use_ssh_config = True

# NOTE: The 'initialize' function for the 'viki.fabric.git' module must
#       be called once in the entire program, before the
#       'setup_server_for_git_clone' task is run. The argument to this
#       function is the path to the YAML file we described above.
fabric_git.initialize(os.path.join("config", "viki_fabric_git.yaml"))

@task
def freds_fabric_task():
  # Fred wishes to setup the current server for handling secret repos
  fabric_git.setup_server_for_git_clone()
  # Fred's other code below
```

Suppose Fred's SSH config file looks like this (see the *env.use_ssh_config* line in the code above to understand why we put this here):

```
Host hostOne
  Hostname 1.2.3.4
  User ubuntu

Host hostTwo
  Hostname 1.2.3.5
  User ubuntu
```

The effect of successfully executing the *setup_server_for_git_clone* Fabric task (it's part of the *freds_fabric_task*):

- For the *ubuntu* user on *hostOne* and *hostTwo*, the *$HOME/.ssh* folder should contain the *id_github_ssh_key* and *id_github_ssh_key.pub* SSH keypair
- A templated *$HOME/gitwrap.sh* should be present for the *ubuntu* user on those 2 servers

Now, the *ubuntu* user on Fred's *hostOne* and *hostTwo* servers are ready for handling some secret git repositories. We shall go into that next.

2.1.5 Working with secret repos after running *setup_server_for_git_clone*

Suppose Fred SSHes into *hostOne* using the *ubuntu* user, and wishes to clone a secret repository whose clone url is *git@github.com:fred/top-secret-repo.git*, he should use this bash command to clone the git repository:

```
GIT_SSH=$HOME/gitwrap.sh git clone git@github.com:fred/top-secret-repo.git
```

In fact, this can be generalized to other Git remote operations for secret repos, such as *git fetch*. The pattern for the command to use is:

```
GIT_SSH=$HOME/gitwrap.sh <git command and args>
```

Which makes me wonder why we named the task *setup_server_for_git_clone*; perhaps this was our original use case.

CHAPTER 3

API Documentation

3.1 API Documentation

3.1.1 viki.fabric.helpers

viki.fabric.helpers.**run_and_get_stdout**(*cmdString*, *hostString=None*, *useSudo=False*)

Runs a command and grabs its output from standard output, without all the Fabric associated stuff and other crap (hopefully).

Args: cmdString(str): Command to run

hostString(str, optional): This should be passed the value of *env.host_string*

useSudo(bool, optional): If *True*, *sudo* will be used instead of *run* to execute the command

Returns: list of str: List of strings from running the command

```
>>> run_and_get_stdout("ls")
["LICENSE", "README.md", "setup.py"]
```

viki.fabric.helpers.**get_home_dir**()

Returns the home directory for the current user of a given server.

Returns:

str: the path to the home directory of the current host, or the string "$HOME"

```
>>> get_home_dir()
"/home/ubuntu"
```

viki.fabric.helpers.**download_remote_file_to_tempfile**(*remoteFileName*)

Downloads a file from a server to a tempfile.NamedTemporaryFile.

NOTE: This function calls the *close* method on the NamedTemporaryFile.

NOTE: The caller is reponsible for deleting the NamedTemporaryFile.

Args: remoteFileName(str): name of the file on the server

Returns:

str: name of the temporary file whose contents is the same as the file on the server

```
>>> downloadedFileName = download_remote_file_to_tempfile(
      "/home/ubuntu/a/search.rb"
    )
```

```
>>> with open(downloadedFileName, "r") as f:
      # do some processing here...
>>> os.unlink(downloadedFileName) # delete the file
```

viki.fabric.helpers.**copy_file_to_server_if_not_exists**(*localFileName*, *serverFileName*)

Copies a file to the server if it does not exist there.

Args: localFileName(str): local path of the file to copy to the server

serverFileName(str): path on the server to copy to

```
>>> copy_file_to_server_if_not_exists("helpers.py",
      os.path.join("my-repo", "helpers.py")
    )
```

viki.fabric.helpers.**is_dir**(*path*)

Checks if a given path on the server is a directory.

Args: path(str): path we wish to check

Returns: bool: True if the given path on the server is a directory, False otherwise

```
>>> is_dir("/home/ubuntu")
True
```

viki.fabric.helpers.**update_package_manager_package_lists**()

Updates the package list of the package manager (currently assumed to be apt-get)

```
>>> update_package_manage_package_lists()
```

viki.fabric.helpers.**install_software_using_package_manager**(*softwareList*)

Installs a list of software using the system's package manager if they have not been installed. Currently this assumes *apt-get* to be the package manager.

Args: softwareList(list of str): list of software to install

```
>>> install_software_using_package_manager(
      ["vim", "openjdk-6-jdk", "unzip"]
    )
```

viki.fabric.helpers.**is_installed_using_package_manager**(*software*)

Determines if a given software is installed on the system by its package manager (currently assumed to be apt-get).

Args: software(str): The name of the software

Return:

bool: Returns True if the software is installed on the system using the package manager, False otherwise

```
>>> is_installed_using_package_manager("python")
True
```

viki.fabric.helpers.**setup_vundle**(*homeDir=None*)

Clones the Vundle vim plugin (https://github.com/gmarik/Vundle.vim) to the server (if it hasn't been cloned), pulls updates, checkout v0.10.2, and installs vim plugins managed by Vundle.

Args:

homeDir(str, optional): home directory for the server. If not supplied or if *None* is supplied, the return value of the *get_home_dir* function is used

```
>>> setup-vundle()
```

3.1.2 viki.fabric.git

viki.fabric.git.**initialize**(*yamlConfigFile*)

Initializes globals in this module. This function should be called before using any public functions defined in this module.

Args:

yamlConfigFile(str): path to the YAML configuration file to read the configuration from.

```
>>> initialize("fabric_git.yaml")
```

viki.fabric.git.**is_dir_under_git_control**(*dirName*)

Determines if a directory on a server is under Git control.

Args:

dirName(str): directory name on the server for which we wish to determine whether it's under Git control

Returns:

bool: True if the directory on the server is under Git control, False otherwise.

```
>>> is_dir_under_git_control("/home/ubuntu/viki-fabric-helpers")
True
```

viki.fabric.git.**setup_server_for_git_clone**()

Fabric task that sets up the ssh keys and a wrapper script for GIT_SSH to allow cloning of private Github repositories.

Args:

homeDir(str, optional): home directory for the server. If not supplied or if *None* is supplied, the return value of the *fabric_helpers.get_home_dir* function is used

For a Python Fabric script that imports the *viki.fabric.git* module using:

```
import viki.fabric.git
```

we can use this Fabric task from the command line, like so:

```
fab -H host1,host2,host3 viki.fabric.git.setup_server_for_git_clone
```

Alternatively, for a Python Fabric script that imports the *viki.fabric.git* module using:

```
import viki.fabric.git as fabric_git
```

we can use this Fabric task from the command like, like so:

```
fab -H host1,host2,host3 fabric_git.setup_server_for_git_clone
```

This function can also be called as a normal function (hopefully from within another Fabric task).

viki.fabric.git.**is_fabtask_setup_server_for_git_clone_run**(*homeDir=None*, *printWarnings=True*)

Determines if the *setup_server_for_git_clone* Fabric task has been run.

This task checks for the existence of some files on the server to determine whether the *setup_server_for_git_clone* task has been run.

Args:

homeDir(str, optional): home directory for the server. If not supplied or if *None* is supplied, the return value of the *fabric_helpers.get_home_dir* function is used

printWarnings(boolean): true if the *setup_server_for_git_clone* task has been run, false otherwise.

Returns:

bool: True if the *setup_server_for_git_clone* Fabric task has been run for the current server, False otherwise.

```
>>> is_fabtask_setup_server_for_git_clone_run()
False # along with some other output before this return value
```

viki.fabric.git.**get_git_ssh_script_path**(*homeDir=None*)

Returns the path to the git ssh script

Args:

homeDir(str, optional): home directory for the server. If not supplied or if *None* is supplied, the return value of the *fabric_helpers.get_home_dir* function is used

Returns: str: the path to the git ssh script

```
>>> get_git_ssh_script_path()
"/home/ubuntu/git_ssh_wrap.sh"
```

CHAPTER 4

Indices and tables

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

V