git-sh-sync Documentation

Release 0.0.0

Frieder Grießhammer

Sep 01, 2018

Contents

Python library to automatically synchronize git repositories via shell

Version 0.0

Release 0.0.0

Welcome to the documentation!

CHAPTER 1

git-sh-sync

GitHub https://github.com/spookey/git-sh-sync

Travis CI https://travis-ci.org/spookey/git-sh-sync

Read the Docs https://git-sh-sync.readthedocs.io

1.1 Contents

1.1.1 Proc Module

This module handles communication with the operating system. Namely launching commands.

`git_sh_sync.proc.`**`CODE_SUCCESS = 0`**
Returncode of a successful command

`git_sh_sync.proc.`**`CHAR_NEWLINE = '\n'`**
Newline character used for detailed log output

`class` `git_sh_sync.proc.`**`Command`**(*cmd*, *, *cwd=None*, *cin=None*)
This is a class-based command runner using `subprocess`

`__init__`(*cmd*, *, *cwd=None*, *cin=None*)
Initialize a new command

Parameters

- **`cmd`** – Commandline of command to launch
- **`cwd`** – Launch *cmd* inside some other current working directory
- **`cin`** – Send data via stdin into *cmd*

`cmd`

Returns `Splitted` output of original *cmd*

`cwd`

Returns Current working directory or `None`

cin

Returns Stdin data or `None`

exc

Returns If launching the command raised some exception it is available here, otherwise `None`

code

Returns The shell returncode after launching. Will be `None` on exception or before launch

stdout

Returns Unmodified output of command stdout or empty string

Return type str

stderr

Returns Unmodified output of command stderr or empty string

Return type str

command

Returns Joined and `quoted` output of internal *cmd*

launched

Returns `True` if command was launched, otherwise `False`

A command is considered launched if any of the *exception* or the *returncode* are not set to `None`

success

Returns `True` if command launch was successful, otherwise `False`

A command is considered successful if no *exception* was thrown and the *returncode* equals *CODE_SUCCESS*

out

Returns Splitted list output of *stdout*

Return type list

err

Returns Splitted list output of *stderr*

Return type list

fields

Returns Some information about the current command as dictionary

Return type dict

Before the command was *launched* only *cmd*, *cwd* and *cin* are included. After *launch* the result is extended by *stdout*, *stderr*, *exc* and *code*.

__repr__()
String representation of current command. Utilizes *fields()* and pprint.pformat() for that.

repr
Expand *__repr__()* with triple-quotes and *CHAR_NEWLINE*.

__call__()
Launches the command.

Returns Output of *success*

To avoid confusion a previously *launched* command will not run again, returning always False.

1.1.2 Repo Module

This module allows working with git repositories.

git_sh_sync.repo.**GIT_DIVIDER = '|-: ^_^ :-|'**
Format divider (e.g. used in log) - Should be different from any text inside a commit message

class git_sh_sync.repo.**GitStatus**

Parameters

- **clean** – Is True if there are pending changes, otherwise False
- **conflicting** – Files with conflicts \o/
- **deleted** – Removed files
- **modified** – Files with modifications
- **untracked** – Files not yet added end up here

class git_sh_sync.repo.**GitBranches**

Parameters

- **current** – Currently active branch
- **all** – All available branches (including *current*)

class git_sh_sync.repo.**GitLog**

Parameters

- **short** – Short commit hash
- **full** – Complete commit hash
- **message** – Commit message

class git_sh_sync.repo.**Repository**(*location*, ***, *master_branch='master'*, *remote_name='origin'*, *remote_url=None*)
Handles communications with git repositories, using native git inside *Command*

__init__(*location*, ***, *master_branch='master'*, *remote_name='origin'*, *remote_url=None*)
Initialize a new Repository

Parameters

- **location** – Local path of the repository
- **master_branch** – Name of the master branch
- **remote_name** – Default name of the remote
- **remote_url** – Remote URL of the repository

Calls then *initialize()* to set everything up

is_repo
Verifies if current *Repository* is indeed a git repository.

initialize(*remote_url=None*)
Is called from inside *__init__()* to prepare the repository. Checks *is_repo* first to bail out early. If no *remote_url* is given a new repository is initialized. Otherwise a clone from the *remote_url* is attempted.

status
Determines current status of the repository.

Returns Current status

Return type *GitStatus*

Generates lists of changed files according to matching state.

branches()
Collects all branches of the repository

Returns All branches

Return type *GitBranches*

Signals current branch and a list of all other branches.

remote_names
Emit names of the remotes. Do not confuse this property with the *remote_name*, which acts as a default value for cloning or pushing actions.

Returns Remote names

Return type list

remote_url(*remote=None*)
Retrieve URL of remote by name.

Parameters **remote** – Remote name

Returns The URL as String or `None` if nothing was found

log(*num=-1*)
Retrieve log of repository

Parameters **num** – Limit length of output. Use negative for unlimited output.

Returns Log entries containing short-, full-hash and commit message.

Return type list of *GitLog*

tags
Query existing tags.

Returns Name of tags, newest first

Return type list

tag(*name*)
Stick tags onto commits.

Parameters **name** – Tag name

Returns `True` if successful else `False`

checkout(*treeish=None*)
Checkout a commit, tag or branch.

Parameters **treeish** – Commit (short or full), tag or branch. If left blank, *master_branch* is assumed

Returns `True` if successful else `False`

If *treeish* is neither a known commit, tag or branch, a new branch is created.

mutate()
Collects all changes and tries to add/remove them.

Returns `True` if everything went well, else `False`

Will freak out if there are conflicts detected - thus returning `False` and writing issues into the log.

scrub(*branch_name=None*)
Uses *mutate()* to handle all changes and commits them into a temporary branch. Will merge the branches back into the original branch afterwards.

Parameters **branch_name** – Name of the temporary branch. Will use the *current hostname* if left blank.

Returns `True` if everything went well (or there is nothing to do), `False` otherwise

cleanup(*branch_name=None*, *remote_name=None*)
Uses *scrub()* to form a new commit and pulls afterwards.

Parameters

- **branch_name** – Name of the temporary branch (see *scrub()*)
- **remote_name** – Name of the remote to pull from. For best results this should be some part of *remote_names()*. If left blank, class wide *remote_name* is taken.

Returns `True` on success, `False` otherwise

__call__(*temp_branch_name=None*, *push_branch_name=None*, *remote_name=None*)
Does a *cleanup* and tries to push afterwards. Will not push if something goes wrong with the *cleanup*.

Parameters

- **temp_branch_name** – Name of the temporary branch (see *scrub()*)
- **push_branch_name** – Name of the branch to push into. Will be set to class wide *master_branch* if set to `None`
- **remote_name** – Name of the remote to pull from (see *cleanup()*)

Returns `True` if everything went well, `False` otherwise

1.1.3 Disk Util

This modules handles disk operations. Namely combining paths, checking them and creating folders..

`git_sh_sync.util.disk.`**`joined`**(**locs*)
Joins paths together.

Parameters **`locs`** – Single path elements to `join` together.

Returns A full path combined by the following rules:

- Leading `slashes` are stripped from all but the first element
- `Expanduser` is applied (~)
- `Expandvars` is applied (e.g. `$HOME` is then the same as ~)
- Finally `realpath` is applied to resolve symlinks and return a full path

`git_sh_sync.util.disk.`**`spare`**(**locs*, *folder=False*)
Checks if a path is not already occupied

Parameters

- **`locs`** – Input Parameter for *`joined()`*
- **`folder`** – Flag to signal if to check for a file or folder

Returns `True`, `False` or `None` by the following rules:

- If a folder is present and *folder* = `True`: `False`
- If a folder is present and *folder* = `False`: `None`
- If a file is present and *folder* = `False`: `False`
- If a file is present and *folder* = `True`: `None`
- If nothing is present: `True`

`git_sh_sync.util.disk.`**`ensured`**(**locs*, *folder=False*)
Checks if the path already exists and creates (parent-)folders if necessary

Parameters

- **`locs`** – Input Parameter for *`spare()`*
- **`folder`** – Treat *locs* Parameter as file or folder

Returns Output of *`joined()`*

1.1.4 Host Util

This module handles host operations. Namely getting the hostname of the local machine...

`git_sh_sync.util.host.`**`get_hostname`**(*short=True*)
Retrieves current hostname.

Parameters **short** – only emit the first part if `true`

Returns hostname (long or short form)

1.2 Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

Symbols

B

C

E

F

G

I

J

L

M

O

R

S

T