Package: git4r (via r-universe)

September 18, 2024

Type Package

Title Interactive Git for R

Version 0.1.1

Description An interactive git user interface from the R command line. Intuitive tools to make commits, branches, remotes, and diffs an integrated part of R coding. Built on git2r, a system installation of git is not required and has default on-premises remote option.

License MIT + file LICENSE

URL https://github.com/johnxhobbs/git4r

BugReports https://github.com/johnxhobbs/git4r/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports git2r, diffr, utils

Suggests gitcreds, rstudioapi, knitr, rmarkdown

RoxygenNote 7.1.1

VignetteBuilder knitr

Repository https://johnxhobbs.r-universe.dev

RemoteUrl https://github.com/johnxhobbs/git4r

RemoteRef HEAD

RemoteSha d2d297514f7419842030ffce58dede1191721e8d

Contents

`git` *Auto Git*

Description

Call the git pull-add-commit-push cycle interactively. This is the default helper function to call continuously to log changes and stay up-to-date.

Usage

```
git()
```

Value

Invisible NULL

See Also

git_remote, git_add, git_commit, git_push, git_branch, git_diff

---

`git_add` *Git Add*

Description

Prints all changed files since last commit and waits for you to hit ENTER to add everything, or type out the space-separated numbers of what you do want to add to the next commit, else ESCAPE to cancel.

Usage

```
git_add()
```

Details

Files can are selected by number by typing a list of space-separated values. The inverse can also be used, for example "-2 -4" would add all *except* the 2nd and 4th file.

Each file or directory is given a symbol for change type:

- `-` has been deleted
- `+` has been created
- `*` has been changed since `git_add()` was last called and should be re-added
- `@` has been renamed, but this will often show as a pair of `+` and `-`
- `?` contains conflict from latest merge; go in and edit by hand, searching for "«««< HEAD" up to "»»»> (some branch)"

For any file which has been flagged by `git_pull()` or `git_merge()` as containing conflicts, these are *not* included by hitting ENTER and must be added by number. This is to reduce the risk of accidentally committing a whole load of conflicts. Each file should be inspected manually before adding.

Value

Invisible NULL

`git_branch` *Git Branch*

Description

Swaps to a different branch if exists, otherwise makes one from most recent commit, and checks out this branch. If no branch name is specified, all branches will be printed.

Usage

```
git_branch(branchname = NULL)
```

Arguments

`branchname`	Branch name to move to or create, use empty string "" to list existing branches. This is asked interactively if left as NULL.

Details

Changing to a branch which is only listed under remote (for example origin/test) by `git_branch('test')` will automatically create a local copy of this branch.

If there are uncommitted changes, it will forbid changing to an existing branch which would result in irreversible loss of current working directory. If the branch does not exist yet, the working directory is not changed and the next commit will be on this new branch.

You are allowed to change branch to any which share the same latest commit so you can create a branch and instantly remove it, whilst keeping any uncommitted changes. The user is notified that the branch has been changed invisibly (namely with no change to the working directory). This may result in a warning that not all changes have been committed when returning much later to the first branch if some changes are staged,

Currently, this is the only situation you are allowed to delete a branch, because in other situations it would result in irreversibly loss of commits (although `git_undo()` will still work for a short time to find orphaned commits). For now, housekeeping and deleting branches should be done manually with system git or `git2r::branch_delete()` if you are really sure.

Value

Invisible NULL

See Also

git_merge

`git_checkout` *Git Checkout Commit*

Description

Rewind the entire working directory to a previous commit and start a new branch from there. Use this if you want to backtrack reversibly. If you want to wind back irreversibly to a previous commit, use `git_undo()`.

Usage

```
git_checkout(commit, onto_new_branch)
```

Arguments

`commit`	Commit object as returned by git_get() or git_history()
`onto_new_branch`	Name of new branch to start from this historic commit

Details

A typical work flow could be to retract three commits on the master branch. Use `git_checkout(commit=git_get(n=3), onto_new_branch='temp')`. Then move back to the master branch with `git_branch('master')` and merge the old version back in with `git_merge('temp')`.

The alternative (`git_undo()` and then select 3rd commit) will delete all record of the three intermediate commits in the long run.

As with changing branch, it is forbidden to change with changes yet-to-be-committed. This is to prevent irreversible loss if you ever wanted to change back again.

There is currently no option to checkout a specific file, however this can be done by calling `git_diff()` and finding the named 'git4r_abc123' in `tempdir()`

Value

Invisible NULL

See Also

git_get, git_history, git_diff, git_undo

`git_clone` *Git Clone from Default Remote*

Description

Clone another repository given as a URL of absolute filepath. Alternatively, if `GIT_DEFAULT_REMOTE` environmental variable has been set a list of repositories within this path is displayed and can be cloned by name.

Usage

```
git_clone(repo_name = "", to = "~/")
```

Arguments

`repo_name`	Name of repo to clone, leave blank to display a list of available repo names. Also accepted is the url or complete filepath to a repo to clone (identified if containing '/').
`to`	Local path to create local project folder in, defaults to home directory

Details

Running `repo_name=""` will print a list of valid repositories found at the `GIT_DEFAULT_REMOTE` path, and can then be re-run with one of these names to clone a local version. This is to make basic on-premises collaboration and backups more effortless. `GIT_DEFAULT_REMOTE` can be set in .Renviron file as per `?git_remote`

Other repositories can be cloned as expected by giving a URL. Credentials are handled automatically in the same way as `?git_push`.

Value

Invisible NULL

`git_commit` *Git Commit*

Description

Make a commit checkpoint of the entire working directory except any changes which have not been added, or files listed in `.gitignore`. Run `git_add()` immediately before to confirm what changes are going to be included / excluded.

Usage

```
git_commit(message = NULL)
```

Arguments

`message`	Commit message, usually one sentence about a specific change, character atomic. This is asked for interactively if left NULL.

Details

Commit messages are personal preference and many examples of good practice can be found online. These messages should be useful for:

- informing yourself and others of what development choices have been made so far
- searching for a particular change, for example like `git_history(message='bugfix 123')`
- explaining a particular change for somebody else to review and accept

The current user identity (username and email) is printed to confirm who will be tagged as making the commit. If this has not yet been configured, these details are prompted interactively.

Amending commits is not currently possible (see issue #213 for git2r) therefore a commit message cannot be left blank.

Value

Invisible NULL

`git_diff` *Git Diff*

Description

Display the difference between two commits of a file or a directory structure. Paths must be relative to the top-level git directory so it is recommended that `getwd()` is the same so that tab auto-complete gives the right path. The default filter is `n=1` and `NULL` meaning compare working tree (or a file contents) of latest commit with current working directory.

Usage

```
git_diff(path = ".", ...)
```

Arguments

`path`	Path or directory to compare - must be relative to the root directory of the repository, regardless of whether `getwd()` is a subdirectory.
`...`	Two arguments to be passed for which versions to select: can be a commit object from git_history() or get_git(), or a single filter argument which is passed to get_git().

Details

Two commits can be specified using `git_get()` or `git_history()[[i]]`, or a single named filter argument for each commit for simple requests. Use `NULL` to compare with the current working file (not even committed yet).

Hidden files (for example '.Rbuildignore') are omitted when listing current directory contents (a commit identifier of NULL), neither is .gitignore respected, so all ignored files will be flagged up.

Usually the more recent commit should be second so that additions are shown in green.

Previous versions of files can be opened for editing by finding them at the `tempdir()` path.

See ?git_history for the filter arguments that can be used

Value

A diffr htmlwidget object which is automatically opened in RStudio Viewer tab

See Also

git_history

Examples

```
## Not run:
# Compare the last committed change with current working version
git_diff() # this is exactly equivalent to next line
git_diff('.', n=1, NULL)
git_diff('README.md')
git_diff('README.md', n=2)
git_diff('README.md', n=2, n=1)

# Compare the directory structure now with a previous date
git_diff('.', before='2021-10-25')

# Latest between branches
git_diff('myfile', branch='master', branch='test')

# Compare file contents of a commit hash with most recent by author
git_diff('R/git_helper.R', hash='abc123', author='somebody')

# Can also get a git_commit object directly to pass over if multiple filtering
git_diff('R/', git_get(author='somebody', before='2021-10-25'), NULL)

# Or even taking a commit number from the branch history (but will be verbose)
git_diff('README.md', git_history()[[1]], n=1 )

## End(Not run)
```

`git_history` *Git Display History and Select Commit*

Description

Display the history of commits in a specific branch with the option to filter. The list of commits is returned invisibly and can be extracted using `[[i]]` alternatively use `git_get()` to return the newest commit after filtering.

Usage

```
git_history(path = ".", branch = NULL, top = 10, ...)

git_get(path = ".", branch = NULL, ...)
```

Arguments

`path`	Only display / search commits which affect this path (dir or file)
`branch`	Name of branch to display. If a match cannot be found with git_get() in this branch, all branches are searched. Remote branches can be viewed by using the format "origin/master" providing that fetch or pull has downloaded changes.

top	Maximum number of commits to display (post-filtering)
...	Filters such as before='2021-01-01' or author='somebody'

Details

By default the history of the current branch is returned. The commits of a remote branch can be found using branch='origin/master, for example.

The following filter arguments can be given:

- n - integer vector of commits back from latest (1 = most recent commit)
- hash - character to match the beginning of commit hash
- before and after - date object or string like '2021-01-31'
- message - regular expression string to find in commit message
- author and email - regular expression string of author or email of the commit

If nothing could be matched, get_git() will start looking on other branches, for example when looking for a particular hash. The user is notified if this happens.

Value

Return the git_commit object which best matches a wide variety of types. This is used for git_diff() and git_checkout().

Vector of commit objects invisibly

Single commit object

---

git_merge *Merge Branch into Current Branch*

Description

Merge a different branch into the current branch. The user is asked interactively to delete the merged (and now expendable) branch, and any conflicts arising are printed and the option given to open the files immediately for editing. Any files which resulted in conflicts will be flagged by git_add() to remind the user to manually confirm conflicts are resolved.

Usage

```
git_merge(branchname = NULL)
```

Arguments

branchname	Name of other branch to merge into the current one, will be asked interactively if left as NULL.

Details

A very helpful tool in RStudio is `Edit -> Find in Files...` which allows you to search your entire repository for where the conflicts are. These are identified by searching for the chevrons «<

Conflicts are the usual format for git and may also happen after `git_pull()`

```
<<<<<<< HEAD
code-from-the-branch-you-have-stayed-on
=======
code-from-the-branch-you-have-just-merged-and-deleted
>>>>>>> merged-and-deleted-branch-name
```

Value

Invisible NULL

`git_pull` *Git Pull*

Description

This will pull any updates from the 'origin' remote for every branch. This should be run before making a commit (as is done by wrapper `git()`) in order to avoid conflicts arising if somebody else has made changes on the same branch. If this happens, merge conflicts are raised in the usual way and `git_add()` will show you what needs resolving.

Usage

```
git_pull()
```

Details

To setup an 'origin' remote, use `git_remote()`. Credentials are handled in the same way as `git_push()`, see `?git_push` for details.

Value

Invisible NULL

See Also

git_push, git_remote

`git_push` *Git Push*

Description

Pushes just current branch to 'origin' by default, but allows selective pushing of branches to different remotes if this is declined, for example if you want to keep a test branch private.

Usage

```
git_push(do_default = NULL)
```

Arguments

`do_default`	Character or logical passed as the answer to the interactive question which is "push the current branch to origin". If 'Y' or TRUE, this function runs without user input (used by `git()`), the default value `NULL` will prompt the user to answer interactively.

Details

It is enforced that when pushing to 'origin' the branch is set to track this remote branch. If the branch does not exist yet in the remote, it is not pushed by default and must be manually added.

Credentials are sorted in the same way for `git_pull()` and `git_push()`. If `gitcreds` package is installed, this is used first to check whether the system git installation already has a username / password for this host. This can be changed or added using stand-alone git, or `gitcreds::gitcreds_set()`.

Alternatively, if `gitcreds` is not installed, a system-git is not available, or no existing git credentials are found, then the the environmental variables are searched for a suitable Personal Access Token. The variable name must begin `GIT_PAT` and any additional words are used to distinguish the PAT for the relevant host using a closest-string match for the remote URL, for example `GIT_PAT_AZURE=abc123def456` will be chosen to authenticate an Azure DevOps remote above `GIT_PAT_GITHUB` or `GIT_PAT_GITLAB`.

To set this up from scratch by creating a Personal Access Token and saving it to your `.Renviron` file with the name "GIT_PAT****" where asterisks can be replaced with part of the remote URL if you need to distinguish between several different PATs (and a good reminder of what it is).

Value

Invisible NULL

See Also

git_pull, git_remote

`git_remote` *Git Modify Remotes*

Description

View and interactively edit remotes for this repo. The most important of these is 'origin' which will be used by `git_pull()`, however you may also want a write-only mirror, for example where you push a production-ready commit. If `GIT_DEFAULT_REMOTE` environmental value has been set, this will be suggested for 'origin'.

Usage

```
git_remote(remote_name = NULL, remote_path = NULL)
```

Arguments

`remote_name`	Name of remote to add (such as 'origin') else leave NULL and answer interactively
`remote_path`	Path / URL of remote to add, empty string will use `GIT_DEFAULT_REMOTE` env variable, else leave NULL and answer interactively

Details

If using a filesystem path for the remote, the option is given to create a full working tree (copy of all files) or bare repository. A bare repository is recommended for 'origin' because no one should make any changes to this directly (instead, clone their own copy). For pushing to a 'backup' or 'production' remote, you will want to push a copy of the files themselves.

If you have set `GIT_DEFAULT_REMOTE` environmental value this will be suggested as the path for creating an 'origin' remote. This will make life easier because it will be the same as the rest of your projects / team and will be searched by `git_clone()`. Use `Sys.setenv(GIT_DEFAULT_REMOTE='P:/ath/to/teams/remote` or better still, put this value in your settings with `file.edit('~/.Renviron')`

It is possible to use an on-premises 'origin' remote which in turn synchronises with another server. This 'hybrid' is best achieved by using fully-fledged git installation set up to synchronise as a scheduled task or somehow triggered. The shared-drive remote repo can be overwritten with:

- `git clone --mirror <url>`

And then these commands run regularly will keep both in sync. If a branch gets divergent commits between cloud and shared-drive, the latter will take precedence. This has not been thoroughly tested and goes beyond the limitations of this package!

- `git push`
- `git remote update --prune`

Value

Invisible NULL

`git_undo` *Reset To Any Previous Commit*

Description

Reset working directory to any previous commit, or action. This will not change the active branch so do this *first* if you want to continue on a different branch. All previous commits are shown, even if the branch they were on has been deleted, and this undo 'reset' itself will appear as a 'reflog' action.

Usage

```
git_undo(top = 10)
```

Arguments

`top`	Number of undo commits to show

Value

Invisible NULL

Index