

Package ‘gitr’

July 22, 2025

Title A Lightweight API for 'Git'

Version 0.1.0

Description A light-weight, dependency-free, application programming interface (API) to access system-level 'Git' <<https://git-scm.com/downloads>> commands from within 'R'. Contains wrappers and defaults for common data science workflows as well as 'Zsh' <<https://github.com/ohmyzsh/ohmyzsh>> plugin aliases. A generalized API syntax is also available.

License MIT + file LICENSE

URL <https://stufield.github.io/gitr/>

BugReports <https://github.com/stufield/gitr/issues>

Depends R (>= 4.1.0)

Suggests knitr, rmarkdown, spelling, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Encoding UTF-8

LazyLoad true

Copyright Stu Field 2025

Config/testthat/edition 3

Config/Needs/website tidyverse/tidytemplate

RoxygenNote 7.3.1

Language en-US

Collate 'gitr-params.R' 'gitr-package.R' 'gitr.R' 'gitr-branch.R'
'gitr-checkout.R' 'gitr-commit.R' 'gitr-lint-commit.R'
'gitr-pr.R' 'gitr-sitrep.R' 'gitr-tag.R' 'gitr-trim-sha.R'
'gitr-utils.R' 'zsh.R'

NeedsCompilation no

Author Stu Field [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0002-1024-5859>>)

Maintainer Stu Field <stu.g.field@gmail.com>

Repository CRAN

Date/Publication 2025-04-21 07:00:02 UTC

Contents

branch	2
checkout	3
commit	3
git	5
lint	6
params	6
pr	7
sha	8
sitrep	9
tag	9
zsh	10
Index	14

branch	<i>Git Branch Utilities</i>
--------	-----------------------------

Description

Git Branch Utilities

Usage

gitr_default_br()

gitr_current_br()

gitr_local_br()

Value

character(1). The name of the respective branch if found, otherwise NULL.

Functions

- gitr_default_br(): gets the default "main" branch, typically either master, main, or trunk.
- gitr_current_br(): gets the *current* branch.
- gitr_local_br(): gets all the *local* branches.

checkout	<i>Git Checkout</i>
----------	---------------------

Description

Checks out as a branch if doesn't exist. Branch oriented workflow for switching between branches. If file is passed, checks out the file. A common shortcut to undoing local changes to a file(s). Can be a vector of multiple files.

Usage

```
gitr_checkout(branch = NULL, file = NULL)
```

Arguments

branch	character(1). The name of a branch, typically a feature branch.
file	character(1). The path to a file.

Value

NULL ... invisibly.

Examples

```
## Not run:  
gitr_checkout("feature-br")  
  
gitr_checkout(file = "DESCRIPTION")  
  
## End(Not run)
```

commit	<i>Git Commit Utilities</i>
--------	-----------------------------

Description

Git Commit Utilities

Usage

```
gitr_commit_msgs(sha = NULL, n = 1L)  
  
scrape_commits(n)  
  
gitr_unstage(file = NULL)
```

```

gitr_reset_soft(n = 1L)

gitr_uncommit()

gitr_reset_hard()

gitr_diff_commits(top = 1L, n = 2L)

```

Arguments

sha	character(n). The commit secure hash algorithm (SHA-1). If NULL, typically points to the most recent commit on the current branch.
n	integer(1). How far back to go from current HEAD. Same as the command line <code>git log -n</code> parameter. For <code>git stash</code> commands, zero-index into the stash list.
file	character(1). The path to a file.
top	integer(1). The commit to consider the "top" of the commit stack. Defaults to <code>HEAD~0</code> or <code>top = 1</code> .

Value

NULL ... invisibly.

A list containing commit message entries. The sha and author of each commit is added as attributes.

Functions

- `gitr_commit_msgs()`: gets the commit messages corresponding to the commit sha. sha can be character(n), but must be valid SHAs corresponding to commits in the repository.
- `scrape_commits()`: scrapes n commit messages for useful change log commits to be used to create a NEWS.md.
- `gitr_unstage()`: un-stages a file from the index to the working directory. Default un-stages *all* files.
- `gitr_reset_soft()`: un-commits the most recently committed file(s) and add them to the staging area.
- `gitr_uncommit()`: un-commits the most recently committed file(s) and add them to the staging area. Wrapper around [gitr_reset_soft\(\)](#)
- `gitr_reset_hard()`: `git reset --hard origin/<branch>`.
- `gitr_diff_commits()`: gets the diff of the corresponding 2 commits. Order matters!

Examples

```

## Not run:
gitr_commit_msgs()

gitr_commit_msgs(n = 3)

```

```
## End(Not run)
```

git

Git Utilities

Description

Provides functionality for system-level Git commands from within R.

Usage

```
git(..., echo_cmd = TRUE)
```

```
is_git()
```

```
git_version()
```

Arguments

...	Additional arguments passed to the system command-line <code>git <command> [<args>]</code> call.
echo_cmd	logical(1). Whether to print the command to run to the console. Can be over-ridden globally via <code>option(gitr_echo_cmd = FALSE)</code> .

Value

`git()`: The system call ... invisibly.

`is_git()`: logical(1).

`git_version()`: character(1). The system version of git.

Functions

- `git()`: executes a git command line call from within R.
- `is_git()`: is current working directory a git repository?
- `git_version()`: gets the version of git in use.

Examples

```
## Not run:  
git("status", "-s")  
  
git("reset", "--soft", "HEAD~1")  
  
git("tag", "-n")  
  
is_git()
```

```
git_version()

## End(Not run)
```

lint	<i>Common Lints for Commit Messages</i>
------	---

Description

Lint a commit message for typical commit style and best practices for git.

Usage

```
lint_commit_msg(x)
```

Arguments

x A single commit message from `gitr_commit_msgs()`.

Value

integer(1). Invisibly returns the number of detected lints in the message.

Examples

```
## Not run:
  lapply(gitr_commit_msgs(7L), lint_commit_msg)

## End(Not run)
```

params	<i>Common Parameters for gitr</i>
--------	-----------------------------------

Description

Common Parameters for gitr

Arguments

n	integer(1). How far back to go from current HEAD. Same as the command line <code>git log -n</code> parameter. For <code>git stash</code> commands, zero-index into the stash list.
file	character(1). The path to a file.
branch	character(1). The name of a branch, typically a feature branch.
sha	character(n). The commit secure hash algorithm (SHA-1). If NULL, typically points to the most recent commit on the current branch.

Description

Git PR Utilities

Usage

```
gitr_pr_msgs(branch = NULL)
```

```
gitr_pr_sha(branch = NULL)
```

Arguments

branch character(1). The name of a branch, typically a feature branch.

Value

[gitr_pr_msgs\(\)](#): see [gitr_commit_msgs\(\)](#).

[gitr_pr_sha\(\)](#): character vector of SHAs corresponding to the PR (relative to the default branch).

Functions

- [gitr_pr_msgs\(\)](#): gets the commit messages for the *current* branch relative to the origin/{main,master} branch in the remote. Typically these "new" commits that would be merged as part of a PR to origin/{main,master}.
- [gitr_pr_sha\(\)](#): gets the commit SHA-1 a branch (by default *current*) relative to the default branch in the remote, usually either origin/main or origin/master. See [gitr_default_br\(\)](#). If there are un-pushed commit on the current default branch, it returns them.

Examples

```
## Not run:  
# SHAs from feature branch differ from default br  
gitr_pr_sha()  
  
# commit messages from the SHAs above  
# for a PR `branch` -> `remotes/origin/{main,master}`  
gitr_pr_msgs()  
  
# for a feature branch -> default branch  
gitr_pr_msgs("feature")  
  
## End(Not run)
```

sha

SHA1 Utilities

Description

SHA1 Utilities

Usage

`gitr_trim_sha(sha)`

`is_sha(sha)`

`gitr_current_sha()`

Arguments

`sha` `character(n)`. The commit secure hash algorithm (SHA-1). If NULL, typically points to the most recent commit on the current branch.

Value

`gitr_trim_sha()`: `character(1)`. The trimmed sha. If sha is not a SHA1 hash, the identical string unchanged.

`is_sha()`: `logical(1)`. If sha matches the SHA1 expected pattern.

`gitr_current_sha()`: `character(1)`. The sha of the current commit.

Functions

- `gitr_trim_sha()`: trims the SHA-1 hash from the default full length to the human-readable short version.
- `is_sha()`: determines whether strings to be tested are a SHA1 hash via regular expression ("^[a-f0-9]{5,40}\$") match.
- `gitr_current_sha()`: gets the current (most recent commit) SHA.

See Also

`grepl()`

sitrep	<i>Git Situation Report</i>
--------	-----------------------------

Description

Get a situation report of the current git repository.

Usage

```
gitr_sitrep()
```

Value

NULL ... invisibly.

tag	<i>Git Tag Utilities</i>
-----	--------------------------

Description

Git Tag Utilities

Usage

```
gitr_recent_tag()
```

```
gitr_tag_info()
```

Value

`gitr_recent_tag()`: character(1). The most recent tag.

`gitr_tag_info()`: A data frame summarizing the repository tags.

Functions

- `gitr_recent_tag()`: gets the *most* recent git tag.
- `gitr_tag_info()`: gets a data frame summary of the current git repository tags.

Examples

```
## Not run:  
gitr_recent_tag()  
  
gitr_tag_info()  
  
## End(Not run)
```

zsh

Z-shell Aliases

Description

Provides functions to common Z-shell git plugin aliases.

Usage

```
glog(n = 10L)
gcc(...)
gcmg(msg = "wip")
gco(branch = NULL)
gcb(branch = NULL)
gpr()
gp(...)
gpu()
gpd()
gst()
gss()
gba()
gbd(branch = NULL, force = FALSE)
gbmm(branch = gitr_default_br())
gbnm(branch = gitr_default_br())
gbm(branch = NULL)
ga(...)
gaa()
gau()
```

```
gsta()
gstl()
gstaa(n = 0)
gstd(n = 0)
gstc()
gsts(text = FALSE)
gpop()
gstp()
gtn()
gfa()
gac()
gwip()
gclean(dry_run = TRUE)
gdf(file = NULL, staged = FALSE)
gpf()
gnukey()
gcf(global = FALSE)
gcm()
grm(...)
grbc()
grba()
grbs()
grbm()
grv()
```

Arguments

n	integer(1). How far back to go from current HEAD. Same as the command line <code>git log -n</code> parameter. For <code>git stash</code> commands, zero-index into the stash list.
...	Additional arguments passed to the system command-line <code>git <command> [<args>]</code> call.
msg	character(1). The message for the commit subject line.
branch	character(1). The name of a branch, typically a feature branch.
force	logical(1). Should the branch delete be forced with the <code>-D</code> flag?
text	logical(1). Show the text diffs from the stash.
dry_run	logical(1). Clean as dry-run?
file	A full file path within the repository to diff.
staged	logical(1). Compare a staged file to HEAD? Otherwise the working directory is compared to the index (staged or HEAD).
global	logical(1). Query global repository. Alternatively local configuration only.

Value

Most aliases invisibly return NULL ... with some exceptions.

Functions

- `glog()`: Get the `git log` in a pretty format for the `n` most recent commits.
- `gcc()`: `git commit ....` To avoid masking the `base::gc()` function, this alias has been re-mapped to `gcc()`.
- `gcmmsg()`: `git commit -m <msg>`.
- `gco()`: `git checkout`.
- `gcb()`: `git checkout -b <branch>`.
- `gpr()`: `git pull --rebase`.
- `gp()`: `git push`.
- `gpu()`: `git push -u origin`.
- `gpd()`: `git push --dry-run`.
- `gst()`: `git status`.
- `gss()`: `git status -s`.
- `gba()`: `git branch -a`.
- `gbd()`: `git branch -dD`.
- `gbmm()`: `git branch --merged <branch>`.
- `gbnm()`: `git branch --no-merged <branch>`.
- `gbm()`: `git branch -m`.
- `ga()`: `git add ....`

- gaa(): git add --all.
- gau(): git add -u.
- gsta(): git stash.
- gstl(): git stash list.
- gstaa(): git stash apply. **Note** zero-indexing!
- gstd(): git stash drop. **Note** zero-indexing!
- gstc(): git stash clear. Danger!
- gsts(): git stash show.
- gpop(): git stash pop --quiet --index.
- gstp(): See gpop().
- gtn(): git tag -n.
- gfa(): git fetch --all --prune.
- gac(): git commit --no-verify --amend --no-edit.
- gwip(): git commit --no-verify -m 'wip'.
- gclean(): git clean -f -d.
- gdf(): git diff <file>.
- gpf(): git push --force-with-lease.
- gnuke(): git reset --hard && git clean -df.
- gcf(): git config --local or git config --global.
- gcm(): Checkout the default branch.
- grm(): git rm
- grbc(): git rebase --continue.
- grba(): git rebase --abort.
- grbs(): git rebase --skip.
- grbm(): git rebase gitr_default_br().
- grv(): git remote -v.

Examples

```
## Not run:  
glog()
```

```
## End(Not run)
```

Index

base::gc(), 12
branch, 2

checkout, 3
commit, 3

ga (zsh), 10
gaa (zsh), 10
gac (zsh), 10
gau (zsh), 10
gba (zsh), 10
gbd (zsh), 10
gbm (zsh), 10
gbmm (zsh), 10
gbnm (zsh), 10
gcb (zsh), 10
gcc (zsh), 10
gcc(), 12
gcf (zsh), 10
gclean (zsh), 10
gcm (zsh), 10
gcmsg (zsh), 10
gco (zsh), 10
gdf (zsh), 10
gfa (zsh), 10
git, 5
git(), 5
git_version (git), 5
gitr_checkout (checkout), 3
gitr_commit_msgs (commit), 3
gitr_commit_msgs(), 6, 7
gitr_current_br (branch), 2
gitr_current_sha (sha), 8
gitr_current_sha(), 8
gitr_default_br (branch), 2
gitr_default_br(), 7
gitr_diff_commits (commit), 3
gitr_local_br (branch), 2
gitr_pr_msgs (pr), 7
gitr_pr_msgs(), 7

gitr_pr_sha (pr), 7
gitr_pr_sha(), 7
gitr_recent_tag (tag), 9
gitr_recent_tag(), 9
gitr_reset_hard (commit), 3
gitr_reset_soft (commit), 3
gitr_reset_soft(), 4
gitr_sitrep (sitrep), 9
gitr_tag_info (tag), 9
gitr_tag_info(), 9
gitr_trim_sha (sha), 8
gitr_trim_sha(), 8
gitr_uncommit (commit), 3
gitr_unstage (commit), 3
glog (zsh), 10
gnuke (zsh), 10
gp (zsh), 10
gpd (zsh), 10
gpf (zsh), 10
gpop (zsh), 10
gpr (zsh), 10
gpu (zsh), 10
grba (zsh), 10
grbc (zsh), 10
grbm (zsh), 10
grbs (zsh), 10
grepl(), 8
grm (zsh), 10
grv (zsh), 10
gss (zsh), 10
gst (zsh), 10
gsta (zsh), 10
gstaa (zsh), 10
gstc (zsh), 10
gstd (zsh), 10
gstl (zsh), 10
gstp (zsh), 10
gsts (zsh), 10
gtn (zsh), 10

gwip (zsh), [10](#)

is_git (git), [5](#)

is_sha (sha), [8](#)

is_sha(), [8](#)

lint, [6](#)

lint_commit_msg (lint), [6](#)

params, [6](#)

pr, [7](#)

scrape_commits (commit), [3](#)

sha, [8](#)

sitrep, [9](#)

tag, [9](#)

zsh, [10](#)