Package: git2r (via r-universe)

October 21, 2024

Type Package

Title Provides Access to Git Repositories

Description Interface to the 'libgit2' library, which is a pure C implementation of the 'Git' core methods. Provides access to 'Git' repositories to extract data and running some basic 'Git' commands.

Version 0.35.0

License GPL-2

URL https://docs.ropensci.org/git2r/, https://github.com/ropensci/git2r

BugReports https://github.com/ropensci/git2r/issues

Imports graphics, utils

Depends R (>= 4.0)

Suggests getPass

NeedsCompilation yes

SystemRequirements libgit2 (>= 1.0): libgit2-devel (rpm) or libgit2-dev (deb)

Collate 'blame.R' 'blob.R' 'branch.R' 'bundle_r_package.R' 'checkout.R' 'commit.R' 'config.R' 'contributions.R' 'credential.R' 'diff.R' 'fetch.R' 'git2r.R' 'index.R' 'libgit2.R' 'merge.R' 'note.R' 'odb.R' 'plot.R' 'pull.R' 'punch_card.R' 'push.R' 'reference.R' 'reflog.R' 'refspec.R' 'remote.R' 'repository.R' 'reset.R' 'revparse.R' 'sha.R' 'signature.R' 'stash.R' 'status.R' 'tag.R' 'time.R' 'tree.R' 'when.R'

Encoding UTF-8

RoxygenNote 7.3.2

Author Stefan Widgren [aut, cre] (<https://orcid.org/0000-0001-5745-2284>), Gabor Csardi [ctb], Gregory Jefferis [ctb], Jennifer Bryan [ctb], Jeroen Ooms

[ctb], Jim Hester [ctb], John Blischak [ctb], Karthik Ram [ctb], Peter Carbonetto [ctb], Scott Chamberlain [ctb], Thomas Rosendal [ctb]

Maintainer Stefan Widgren <stefan.widgren@gmail.com>

Repository CRAN

Date/Publication 2024-10-20 23:00:02 UTC

Contents

add 4
ahead_behind 6
as.data.frame.git_repository 7
as.data.frame.git_tree 8
as.list.git_tree 9
blame 10
blob_create 12
branches 13
branch_create 14
branch_delete 15
branch_get_upstream 16
branch_remote_name 17
branch_remote_url 18
branch_rename 19
branch_set_upstream 20
branch_target 21
bundle_r_package 22
checkout 23
clone 25
commit 26
commits 28
config 30
content 31
contributions 32
cred_env 34
cred_ssh_key 35
cred_token 36
cred_user_pass 37
default_signature 38
descendant_of 39
diff.git_repository 40
discover_repository 43
fetch 44
fetch_heads 46
git2r 47
git_config_files 48
git_time 48
hash 50

hashfile . . . 50
head.git_repository . . . 51
index_remove_bypath . . . 52
init . . . 53
in_repository . . . 54
is_bare . . . 55
is_binary . . . 56
is_blob . . . 57
is_branch . . . 58
is_commit . . . 59
is_detached . . . 60
is_empty . . . 61
is_head . . . 62
is_local . . . 63
is_merge . . . 64
is_shallow . . . 65
is_tag . . . 66
is_tree . . . 67
last_commit . . . 68
length.git_blob . . . 69
length.git_diff . . . 70
length.git_tree . . . 70
libgit2_features . . . 71
libgit2_version . . . 71
lookup . . . 72
lookup_commit . . . 73
ls_tree . . . 74
merge.git_branch . . . 76
merge_base . . . 78
notes . . . 79
note_create . . . 80
note_default_ref . . . 82
note_remove . . . 83
odb_blobs . . . 84
odb_objects . . . 85
parents . . . 86
plot.git_repository . . . 87
print.git_reflog_entry . . . 88
pull . . . 89
punch_card . . . 91
push . . . 92
references . . . 93
reflog . . . 95
remotes . . . 96
remote_add . . . 97
remote_ls . . . 98
remote_remove . . . 99
remote_rename . . . 100

remote_set_url . . . 101
remote_url . . . 102
repository . . . 103
repository_head . . . 105
reset . . . 106
revparse_single . . . 107
rm_file . . . 108
sha . . . 109
ssh_path . . . 111
ssl_cert_locations . . . 112
stash . . . 112
stash_apply . . . 114
stash_drop . . . 115
stash_list . . . 116
stash_pop . . . 118
status . . . 119
summary.git_repository . . . 120
summary.git_stash . . . 122
summary.git_tree . . . 123
tag . . . 124
tags . . . 125
tag_delete . . . 126
tree . . . 127
when . . . 128
workdir . . . 129
[.git_tree . . . 130

Index **132**

`add` *Add file(s) to index*

Description

Add file(s) to index

Usage

```
add(repo = ".", path = NULL, force = FALSE)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`path`	Character vector with file names or shell glob patterns that will matched against files in the repository's working directory. Each file that matches will be added to the index (either updating an existing entry or adding a new entry).
`force`	Add ignored files. Default is FALSE.

Value

invisible(NULL)

Examples

```
## Not run:
## Initialize a repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file
writeLines("a", file.path(path, "a.txt"))

## Add file to repository and view status
add(repo, "a.txt")
status(repo)

## Add file with a leading './' when the repository working
## directory is the current working directory
setwd(path)
writeLines("b", file.path(path, "b.txt"))
add(repo, "./b.txt")
status(repo)

## Add a file in a sub-folder with sub-folder as the working
## directory. Create a file in the root of the repository
## working directory that will remain untracked.
dir.create(file.path(path, "sub_dir"))
setwd("./sub_dir")
writeLines("c", file.path(path, "c.txt"))
writeLines("c", file.path(path, "sub_dir/c.txt"))
add(repo, "c.txt")
status(repo)

## Add files with glob expansion when the current working
## directory is outside the repository's working directory.
setwd(tempdir())
dir.create(file.path(path, "glob_dir"))
writeLines("d", file.path(path, "glob_dir/d.txt"))
writeLines("e", file.path(path, "glob_dir/e.txt"))
writeLines("f", file.path(path, "glob_dir/f.txt"))
writeLines("g", file.path(path, "glob_dir/g.md"))
add(repo, "glob_dir/*txt")
status(repo)

## Add file with glob expansion with a relative path when
## the current working directory is inside the repository's
## working directory.
```

```
setwd(path)
add(repo, "./glob_dir/*md")
status(repo)

## End(Not run)
```

---

ahead_behind *Ahead Behind*

Description

Count the number of unique commits between two commit objects.

Usage

```
ahead_behind(local = NULL, upstream = NULL)
```

Arguments

local	a git_commit object. Can also be a tag or a branch, and in that case the commit will be the target of the tag or branch.
upstream	a git_commit object. Can also be a tag or a branch, and in that case the commit will be the target of the tag or branch.

Value

An integer vector of length 2 with number of commits that the upstream commit is ahead and behind the local commit

Examples

```
## Not run:
## Create a directory in tempdir
path <- tempfile(pattern="git2r-")
dir.create(path)

## Initialize a repository
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "test.txt"))
add(repo, "test.txt")
commit_1 <- commit(repo, "Commit message 1")
tag_1 <- tag(repo, "Tagname1", "Tag message 1")

# Change file and commit
lines <- c(
```

```
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path, "test.txt"))
add(repo, "test.txt")
commit_2 <- commit(repo, "Commit message 2")
tag_2 <- tag(repo, "Tagname2", "Tag message 2")

ahead_behind(commit_1, commit_2)
ahead_behind(tag_1, tag_2)

## End(Not run)
```

`as.data.frame.git_repository` *Coerce Git repository to a* `data.frame`

Description

The commits in the repository are coerced to a `data.frame`

Usage

```
## S3 method for class 'git_repository'
as.data.frame(x, ...)
```

Arguments

`x`	The repository `object`
`...`	Additional arguments. Not used.

Details

The `data.frame` have the following columns:

sha The 40 character hexadecimal string of the SHA-1

summary the short "summary" of the git commit message.

message the full message of a commit

author full name of the author

email email of the author

when time when the commit happened

Value

`data.frame`

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create three files and commit
writeLines("First file",  file.path(path, "example-1.txt"))
writeLines("Second file", file.path(path, "example-2.txt"))
writeLines("Third file",  file.path(path, "example-3.txt"))
add(repo, "example-1.txt")
commit(repo, "Commit first file")
add(repo, "example-2.txt")
commit(repo, "Commit second file")
add(repo, "example-3.txt")
commit(repo, "Commit third file")

## Coerce commits to a data.frame
df <- as.data.frame(repo)
df

## End(Not run)
```

`as.data.frame.git_tree`

Coerce entries in a git_tree to a `data.frame`

Description

The entries in a tree are coerced to a `data.frame`

Usage

```
## S3 method for class 'git_tree'
as.data.frame(x, ...)
```

Arguments

`x`	The tree `object`
`...`	Additional arguments. Not used.

Details

The `data.frame` have the following columns:

filemode The UNIX file attributes of a tree entry

type String representation of the tree entry type

sha The sha of a tree entry

name The filename of a tree entry

Value

`data.frame`

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
dir.create(file.path(path, "subfolder"))
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create three files and commit
writeLines("First file",  file.path(path, "example-1.txt"))
writeLines("Second file", file.path(path, "subfolder/example-2.txt"))
writeLines("Third file",  file.path(path, "example-3.txt"))
add(repo, c("example-1.txt", "subfolder/example-2.txt", "example-3.txt"))
commit(repo, "Commit message")

## Display tree
tree(last_commit(repo))

## Coerce tree to a data.frame
df <- as.data.frame(tree(last_commit(repo)))
df

## End(Not run)
```

`as.list.git_tree` *Coerce entries in a git_tree to a list of entry objects*

Description

Coerce entries in a git_tree to a list of entry objects

Usage

```
## S3 method for class 'git_tree'
as.list(x, ...)
```

Arguments

x	The tree object
...	Unused

Value

list of entry objects

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
dir.create(file.path(path, "subfolder"))
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create three files and commit
writeLines("First file",  file.path(path, "example-1.txt"))
writeLines("Second file", file.path(path, "subfolder/example-2.txt"))
writeLines("Third file",  file.path(path, "example-3.txt"))
add(repo, c("example-1.txt", "subfolder/example-2.txt", "example-3.txt"))
commit(repo, "Commit message")

## Inspect size of each blob in tree
invisible(lapply(as(tree(last_commit(repo)), "list"),
  function(obj) {
    if (is_blob(obj))
      summary(obj)
    NULL
  }))

## End(Not run)
```

---

blame *Get blame for file*

Description

Get blame for file

Usage

```
blame(repo = ".", path = NULL)
```

Arguments

repo	a path to a repository or a git_repository object. Default is '.'
path	Path to the file to consider

Value

git_blame object with the following entries:

path The path to the file of the blame

hunks List of blame hunks

repo The git_repository that contains the file

lines_in_hunk The number of lines in this hunk

final_commit_id The sha of the commit where this line was last changed

final_start_line_number The 1-based line number where this hunk begins, in the final version of the file

final_signature Final committer

orig_commit_id The sha of the commit where this hunk was found. This will usually be the same as 'final_commit_id'.

orig_start_line_number The 1-based line number where this hunk begins in the file named by 'orig_path' in the commit specified by 'orig_commit_id'.

orig_signature Origin committer

orig_path The path to the file where this hunk originated, as of the commit specified by 'orig_commit_id'

boundary TRUE iff the hunk has been tracked to a boundary commit.

repo The git_repository object that contains the blame hunk

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a first user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Create a second user and change the file
config(repo, user.name = "Bob", user.email = "bob@example.org")
writeLines(c("Hello world!", "HELLO WORLD!", "HOLA"),
```

```
           file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "Second commit message")

## Check blame
blame(repo, "example.txt")

## End(Not run)
```

`blob_create` *Create blob from file on disk*

Description

Read a file from the filesystem and write its content to the Object Database as a loose blob. The method is vectorized and accepts a vector of files to create blobs from.

Usage

```
blob_create(repo = ".", path = NULL, relative = TRUE)
```

Arguments

`repo`	The repository where the blob(s) will be written. Can be a bare repository. A `git_repository` object, or a path to a repository, or `NULL`. If the `repo` argument is `NULL`, the repository is searched for with `discover_repository` in the current working directory.
`path`	The file(s) from which the blob will be created.
`relative`	TRUE if the file(s) from which the blob will be created is relative to the repository's working dir. Default is TRUE.

Value

list of S3 class git_blob `objects`

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create blobs from files relative to workdir
writeLines("Hello, world!", file.path(path, "example-1.txt"))
writeLines("test content", file.path(path, "example-2.txt"))
blob_list_1 <- blob_create(repo, c("example-1.txt",
                                   "example-2.txt"))
```

```
## Create blobs from files not relative to workdir
temp_file_1 <- tempfile()
temp_file_2 <- tempfile()
writeLines("Hello, world!", temp_file_1)
writeLines("test content", temp_file_2)
blob_list_2 <- blob_create(repo, c(temp_file_1, temp_file_2),
                           relative = FALSE)

## End(Not run)
```

`branches` *Branches*

Description

List branches in repository

Usage

```
branches(repo = ".", flags = c("all", "local", "remote"))
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`flags`	Filtering flags for the branch listing. Valid values are 'all', 'local' or 'remote'

Value

list of branches in repository

Examples

```
## Not run:
## Initialize repositories
path_bare <- tempfile(pattern="git2r-")
path_repo <- tempfile(pattern="git2r-")
dir.create(path_bare)
dir.create(path_repo)
repo_bare <- init(path_bare, bare = TRUE)
repo <- clone(path_bare, path_repo)

## Config first user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path_repo, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")
```

```
## Push commits from repository to bare repository
## Adds an upstream tracking branch to branch 'master'
push(repo, "origin", "refs/heads/master")

## List branches
branches(repo)

## End(Not run)
```

---

branch_create *Create a branch*

Description

Create a branch

Usage

```
branch_create(commit = last_commit(), name = NULL, force = FALSE)
```

Arguments

commit	Commit to which the branch should point. The default is to use the last_commit() function to determine the commit to which the branch should point.
name	Name for the branch
force	Overwrite existing branch. Default = FALSE

Value

invisible git_branch object

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
lines <- "Hello world!"
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit_1 <- commit(repo, "First commit message")

## Create a branch
branch_1 <- branch_create(commit_1, name = "test-branch")
```

```
## Add one more commit
lines <- c("Hello world!", "HELLO WORLD!")
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit_2 <- commit(repo, "Another commit message")

## Create a branch with the same name should fail
try(branch_create(commit_2, name = "test-branch"), TRUE)

## Force it
branch_2 <- branch_create(commit_2, name = "test-branch", force = TRUE)

## End(Not run)
```

---

branch_delete *Delete a branch*

Description

Delete a branch

Usage

```
branch_delete(branch = NULL)
```

Arguments

branch	The branch

Value

invisible NULL

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit_1 <- commit(repo, "First commit message")

## Create a 'dev' branch
dev <- branch_create(commit_1, name = "dev")
branches(repo)
```

```
## Delete 'dev' branch
branch_delete(dev)
branches(repo)

## End(Not run)
```

`branch_get_upstream` *Get remote tracking branch*

Description

Get remote tracking branch, given a local branch.

Usage

```
branch_get_upstream(branch = NULL)
```

Arguments

`branch`	The branch

Value

`git_branch` object or NULL if no remote tracking branch.

Examples

```
## Not run:
## Initialize two temporary repositories
path_bare <- tempfile(pattern="git2r-")
path_repo <- tempfile(pattern="git2r-")
dir.create(path_bare)
dir.create(path_repo)
repo_bare <- init(path_bare, bare = TRUE)
repo <- clone(path_bare, path_repo)

## Config user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path_repo, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Push commits from repository to bare repository
## Adds an upstream tracking branch to branch 'master'
push(repo, "origin", "refs/heads/master")
```

```
## Get remote tracking branch
branch_get_upstream(repository_head(repo))

## End(Not run)
```

---

branch_remote_name *Remote name of a branch*

Description

The name of remote that the remote tracking branch belongs to

Usage

```
branch_remote_name(branch = NULL)
```

Arguments

branch	The branch

Value

character string with remote name

Examples

```
## Not run:
## Initialize two temporary repositories
path_bare <- tempfile(pattern="git2r-")
path_repo <- tempfile(pattern="git2r-")
dir.create(path_bare)
dir.create(path_repo)
repo_bare <- init(path_bare, bare = TRUE)
repo <- clone(path_bare, path_repo)

## Config user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path_repo, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Push commits from repository to bare repository
## Adds an upstream tracking branch to branch 'master'
push(repo, "origin", "refs/heads/master")

## Get remote name
branch_remote_name(branches(repo)[[2]])
```

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

---

branch_remote_url *Remote url of a branch*

Description

Remote url of a branch

Usage

```
branch_remote_url(branch = NULL)
```

Arguments

branch	The branch

Value

character string with remote url

Examples

```
## Not run:
## Initialize two temporary repositories
path_bare <- tempfile(pattern="git2r-")
path_repo <- tempfile(pattern="git2r-")
dir.create(path_bare)
dir.create(path_repo)
repo_bare <- init(path_bare, bare = TRUE)
repo <- clone(path_bare, path_repo)

## Config user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path_repo, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Push commits from repository to bare repository
## Adds an upstream tracking branch to branch 'master'
push(repo, "origin", "refs/heads/master")

## Get remote url of tracking branch to branch 'master'
branch_remote_url(branch_get_upstream(repository_head(repo)))

## End(Not run)
```

branch_rename *Rename a branch*

Description

Rename a branch

Usage

```
branch_rename(branch = NULL, name = NULL, force = FALSE)
```

Arguments

branch	Branch to rename
name	The new name for the branch
force	Overwrite existing branch. Default is FALSE

Value

invisible renamed git_branch object

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Config user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Rename 'master' branch to 'dev'
branches(repo)
branch_rename(repository_head(repo), "dev")
branches(repo)

## End(Not run)
```

`branch_set_upstream` *Set remote tracking branch*

Description

Set the upstream configuration for a given local branch

Usage

```
branch_set_upstream(branch = NULL, name)
```

Arguments

`branch`	The branch to configure
`name`	remote-tracking or local branch to set as upstream. Pass NULL to unset.

Value

invisible NULL

Examples

```
## Not run:
## Initialize two temporary repositories
path_bare <- tempfile(pattern="git2r-")
path_repo <- tempfile(pattern="git2r-")
dir.create(path_bare)
dir.create(path_repo)
repo_bare <- init(path_bare, bare = TRUE)
repo <- clone(path_bare, path_repo)

## Config user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path_repo, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Push commits from repository to bare repository
## Adds an upstream tracking branch to branch 'master'
push(repo, "origin", "refs/heads/master")

## Unset remote remote tracking branch
branch_get_upstream(repository_head(repo))
branch_set_upstream(repository_head(repo), NULL)
branch_get_upstream(repository_head(repo))
```

```
## Set remote tracking branch
branch_set_upstream(repository_head(repo), "origin/master")
branch_get_upstream(repository_head(repo))

## End(Not run)
```

---

branch_target *Get target (sha) pointed to by a branch*

Description

Get target (sha) pointed to by a branch

Usage

```
branch_target(branch = NULL)
```

Arguments

branch	The branch

Value

sha or NA if not a direct reference

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Config user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Get target (sha) pointed to by 'master' branch
branch_target(repository_head(repo))

## End(Not run)
```

bundle_r_package *Bundle bare repo of package*

Description

Clone the package git repository as a bare repository to pkg/inst/pkg.git

Usage

```
bundle_r_package(repo = ".")
```

Arguments

repo	a path to a repository or a git_repository object. Default is '.'

Value

Invisible bundled git_repository object

Examples

```
## Not run:
## Initialize repository
path <- tempfile()
dir.create(path)
path <- file.path(path, "git2r")
repo <- clone("https://github.com/ropensci/git2r.git", path)

## Bundle bare repository in package
bundle_r_package(repo)

## Build and install bundled package
wd <- setwd(dirname(path))
system(sprintf("R CMD build %s", path))
pkg <- list.files(".", pattern = "[.]tar[.]gz$")
system(sprintf("R CMD INSTALL %s", pkg))
setwd(wd)

## Reload package
detach("package:git2r", unload = TRUE)
library(git2r)

## Summarize last five commits of bundled repo
repo <- repository(system.file("git2r.git", package = "git2r"))
invisible(lapply(commits(repo, n = 5), summary))

## Plot content of bundled repo
plot(repo)

## End(Not run)
```

checkout *Checkout*

Description

Update files in the index and working tree to match the content of the tree pointed at by the treeish object (commit, tag or tree). The default checkout strategy (`force = FALSE`) will only make modifications that will not lose changes. Use `force = TRUE` to force working directory to look like index.

Usage

```
checkout(
  object = NULL,
  branch = NULL,
  create = FALSE,
  force = FALSE,
  path = NULL,
  ...
)
```

Arguments

`object`	A path to a repository, or a `git_repository` object, or a `git_commit` object, or a `git_tag` object, or a `git_tree` object.
`branch`	name of the branch to check out. Only used if object is a path to a repository or a `git_repository` object.
`create`	create branch if it doesn't exist. Only used if object is a path to a repository or a `git_repository` object.
`force`	If `TRUE`, then make working directory match target. This will throw away local changes. Default is `FALSE`.
`path`	Limit the checkout operation to only certain paths. This argument is only used if branch is NULL. Default is `NULL`.
`...`	Additional arguments. Not used.

Value

invisible NULL

Examples

```
## Not run:
## Create directories and initialize repositories
path_bare <- tempfile(pattern="git2r-")
path_repo_1 <- tempfile(pattern="git2r-")
path_repo_2 <- tempfile(pattern="git2r-")
```

```
dir.create(path_bare)
dir.create(path_repo_1)
dir.create(path_repo_2)
repo_bare <- init(path_bare, bare = TRUE)

## Clone to repo 1 and config user
repo_1 <- clone(path_bare, path_repo_1)
config(repo_1, user.name = "Alice", user.email = "alice@example.org")

## Add changes to repo 1 and push to bare
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path_repo_1, "test.txt"))
add(repo_1, "test.txt")
commit(repo_1, "First commit message")
push(repo_1, "origin", "refs/heads/master")

## Create and checkout 'dev' branch in repo 1
checkout(repo_1, "dev", create = TRUE)

## Add changes to 'dev' branch in repo 1 and push to bare
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path_repo_1, "test.txt"))
add(repo_1, "test.txt")
commit(repo_1, "Second commit message")
push(repo_1, "origin", "refs/heads/dev")

## Clone to repo 2
repo_2 <- clone(path_bare, path_repo_2)
config(repo_2, user.name = "Bob", user.email = "bob@example.org")

## Read content of 'test.txt'
readLines(file.path(path_repo_2, "test.txt"))

## Checkout dev branch
checkout(repo_2, "dev")

## Read content of 'test.txt'
readLines(file.path(path_repo_2, "test.txt"))

## Edit "test.txt" in repo_2
writeLines("Hello world!", con = file.path(path_repo_2, "test.txt"))

## Check status
status(repo_2)

## Checkout "test.txt"
checkout(repo_2, path = "test.txt")

## Check status
status(repo_2)
```

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

`clone` *Clone a remote repository*

Description

Clone a remote repository

Usage

```
clone(
  url = NULL,
  local_path = NULL,
  bare = FALSE,
  branch = NULL,
  checkout = TRUE,
  credentials = NULL,
  progress = TRUE
)
```

Arguments

`url`	The remote repository to clone
`local_path`	Local directory to clone to.
`bare`	Create a bare repository. Default is FALSE.
`branch`	The name of the branch to checkout. Default is NULL which means to use the remote's default branch.
`checkout`	Checkout HEAD after the clone is complete. Default is TRUE.
`credentials`	The credentials for remote repository access. Default is NULL. To use and query an ssh-agent for the ssh key credentials, let this parameter be NULL (the default).
`progress`	Show progress. Default is TRUE.

Value

A `git_repository` object.

See Also

repository, `cred_user_pass`, `cred_ssh_key`

Examples

```
## Not run:
## Initialize repository
path_repo_1 <- tempfile(pattern="git2r-")
path_repo_2 <- tempfile(pattern="git2r-")
dir.create(path_repo_1)
dir.create(path_repo_2)
repo_1 <- init(path_repo_1)

## Config user and commit a file
config(repo_1, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
writeLines(
    "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
    file.path(path_repo_1, "example.txt"))
add(repo_1, "example.txt")
commit(repo_1, "First commit message")

## Change file and commit
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path_repo_1, "example.txt"))
add(repo_1, "example.txt")
commit(repo_1, "Second commit message")

## Change file again and commit.
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad",
  "minim veniam, quis nostrud exercitation ullamco laboris nisi ut")
writeLines(lines, file.path(path_repo_1, "example.txt"))
add(repo_1, "example.txt")
commit(repo_1, "Third commit message")

## Clone to second repository
repo_2 <- clone(path_repo_1, path_repo_2)

## List commits in repositories
commits(repo_1)
commits(repo_2)

## End(Not run)
```

`commit` *Commit*

Description

Commit

Usage

```
commit(
  repo = ".",
  message = NULL,
  all = FALSE,
  session = FALSE,
  author = NULL,
  committer = NULL
)
```

Arguments

repo	a path to a repository or a git_repository object. Default is '.'
message	The commit message.
all	Stage modified and deleted files. Files not added to Git are not affected.
session	Add sessionInfo to commit message. Default is FALSE.
author	Signature with author and author time of commit.
committer	Signature with committer and commit time of commit.

Value

A list of class git_commit with entries:

sha The 40 character hexadecimal string of the SHA-1

author An author signature

committer The committer signature

summary The short "summary" of a git commit message, comprising the first paragraph of the message with whitespace trimmed and squashed.

message The message of a commit

repo The git_repository object that contains the commit

Examples

```
## Not run:
## Initialize a repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Config user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")
```

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

`commits` *Commits*

Description

Commits

Usage

```
commits(
  repo = ".",
  topological = TRUE,
  time = TRUE,
  reverse = FALSE,
  n = NULL,
  ref = NULL,
  path = NULL
)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`topological`	Sort the commits in topological order (parents before children); can be combined with time sorting. Default is TRUE.
`time`	Sort the commits by commit time; Can be combined with topological sorting. Default is TRUE.
`reverse`	Sort the commits in reverse order; can be combined with topological and/or time sorting. Default is FALSE.
`n`	The upper limit of the number of commits to output. The default is NULL for unlimited number of commits.
`ref`	The name of a reference to list commits from e.g. a tag or a branch. The default is NULL for the current branch.
`path`	The path to a file. If not NULL, only commits modifying this file will be returned. Note that modifying commits that occurred before the file was given its present name are not returned; that is, the output of `git log` with `--no-follow` is reproduced.

Value

list of commits in repository

Examples

```
## Not run:
## Initialize a repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Config user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Change file and commit
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "Second commit message")

## Create a tag
tag(repo, "Tagname", "Tag message")

## Change file again and commit
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad",
  "minim veniam, quis nostrud exercitation ullamco laboris nisi ut")
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "Third commit message")

## Create a new file containing R code, and commit.
writeLines(c("x <- seq(1,100)",
             "print(mean(x))"),
           file.path(path, "mean.R"))
add(repo, "mean.R")
commit(repo, "Fourth commit message")

## List the commits in the repository
commits(repo)

## List the commits starting from the tag
commits(repo, ref = "Tagname")

## List the commits modifying example.txt and mean.R.
commits(repo, path = "example.txt")
commits(repo, path = "mean.R")
```

```
## Create and checkout 'dev' branch in the repo
checkout(repo, "dev", create = TRUE)

## Add changes to the 'dev' branch
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "Commit message in dev branch")

## Checkout the 'master' branch again and list the commits
## starting from the 'dev' branch.
checkout(repo, "master")
commits(repo, ref = "dev")

## End(Not run)
```

`config` *Config*

Description

Config file management. To display the configuration variables, call method `config` without the `user.name`, `user.email` or `...` options.

Usage

```
config(repo = NULL, global = FALSE, user.name, user.email, ...)
```

Arguments

`repo`	The `repository`. Default is NULL.
`global`	Write option(s) to global configuration file. Default is FALSE.
`user.name`	The user name. Use NULL to delete the entry
`user.email`	The e-mail address. Use NULL to delete the entry
`...`	Additional options to write or delete from the configuration.

Details

There are two ways git2r can find the local repository when writing local options (1) Use the `repo` argument. (2) If the `repo` argument is `NULL` but the current working directory is inside the local repository, then `git2r` uses that repository.

Value

S3 class `git_config`. When writing options, the configuration is returned invisible.

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern = "git2r-")
dir.create(path)
repo <- init(path)

## Set user name and email.
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Display configuration
config(repo)

## Delete user email.
config(repo, user.email = NULL)

## Display configuration
config(repo)

## End(Not run)
```

`content` *Content of blob*

Description

Content of blob

Usage

```
content(blob = NULL, split = TRUE, raw = FALSE)
```

Arguments

`blob`	The blob object.
`split`	Split blob content to text lines. Default TRUE.
`raw`	When `TRUE`, get the content of the blob as a raw vector, else as a character vector. Default is `FALSE`.

Value

The content of the blob. NA_character_ if the blob is binary and `raw` is `FALSE`.

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Display content of blob.
content(tree(commits(repo)[[1]])["example.txt"])

## End(Not run)
```

`contributions` *Contributions*

Description

See contributions to a Git repo

Usage

```
contributions(
  repo = ".",
  breaks = c("month", "year", "quarter", "week", "day"),
  by = c("commits", "author")
)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`breaks`	Default is `month`. Change to year, quarter, week or day as necessary.
`by`	Contributions by "commits" or "author". Default is "commits".

Value

A `data.frame` with contributions.

Examples

```
## Not run:
## Create directories and initialize repositories
path_bare <- tempfile(pattern="git2r-")
path_repo_1 <- tempfile(pattern="git2r-")
path_repo_2 <- tempfile(pattern="git2r-")
dir.create(path_bare)
dir.create(path_repo_1)
dir.create(path_repo_2)
repo_bare <- init(path_bare, bare = TRUE)

## Clone to repo 1 and config user
repo_1 <- clone(path_bare, path_repo_1)
config(repo_1, user.name = "Alice", user.email = "alice@example.org")

## Add changes to repo 1 and push to bare
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path_repo_1, "test.txt"))
add(repo_1, "test.txt")
commit(repo_1, "First commit message")

## Add more changes to repo 1
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path_repo_1, "test.txt"))
add(repo_1, "test.txt")
commit(repo_1, "Second commit message")

## Push to bare
push(repo_1, "origin", "refs/heads/master")

## Clone to repo 2
repo_2 <- clone(path_bare, path_repo_2)
config(repo_2, user.name = "Bob", user.email = "bob@example.org")

## Add changes to repo 2
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad",
  "minim veniam, quis nostrud exercitation ullamco laboris nisi ut")
writeLines(lines, file.path(path_repo_2, "test.txt"))
add(repo_2, "test.txt")
commit(repo_2, "Third commit message")

## Push to bare
push(repo_2, "origin", "refs/heads/master")

## Pull changes to repo 1
pull(repo_1)

## View contributions by day
```

```
contributions(repo_1)

## View contributions by author and day
contributions(repo_1, by = "author")

## End(Not run)
```

---

cred_env *Create a new environmental credential object*

Description

Environmental variables can be written to the file `.Renviron`. This file is read by *R* during startup, see `Startup`.

Usage

```
cred_env(username = NULL, password = NULL)
```

Arguments

`username`	The name of the environmental variable that holds the username for the authentication.
`password`	The name of the environmental variable that holds the password for the authentication.

Value

A list of class `cred_env` with entries:

username The name of the environmental variable that holds the username for the authentication.

password The name of the environmental variable that holds the password for the authentication.

See Also

Other git credential functions: `cred_ssh_key()`, `cred_token()`, `cred_user_pass()`

Examples

```
## Not run:
## Create an environmental credential object for the username and
## password.
cred <- cred_env("NAME_OF_ENV_VARIABLE_WITH_USERNAME",
                 "NAME_OF_ENV_VARIABLE_WITH_PASSWORD")
repo <- repository("git2r")
push(repo, credentials = cred)

## End(Not run)
```

cred_ssh_key *Create a new passphrase-protected ssh key credential object*

Description

Create a new passphrase-protected ssh key credential object

Usage

```
cred_ssh_key(
  publickey = ssh_path("id_rsa.pub"),
  privatekey = ssh_path("id_rsa"),
  passphrase = character(0)
)
```

Arguments

publickey	The path to the public key of the credential. Default is `ssh_path("id_rsa.pub")`
privatekey	The path to the private key of the credential. Default is `ssh_path("id_rsa")`
passphrase	The passphrase of the credential. Default is `character(0)`. If getPass is installed and private key is passphrase protected `getPass::getPass()` will be called to allow for interactive and obfuscated interactive input of the passphrase.

Value

A list of class `cred_ssh_key` with entries:

publickey The path to the public key of the credential

privatekey The path to the private key of the credential

passphrase The passphrase of the credential

See Also

Other git credential functions: `cred_env()`, `cred_token()`, `cred_user_pass()`

Examples

```
## Not run:
## Create a ssh key credential object. It can optionally be
## passphrase-protected
cred <- cred_ssh_key(ssh_path("id_rsa.pub"), ssh_path("id_rsa"))
repo <- repository("git2r")
push(repo, credentials = cred)

## End(Not run)
```

cred_token *Create a new personal access token credential object*

Description

The personal access token is stored in an envrionmental variable. Environmental variables can be written to the file `.Renviron`. This file is read by *R* during startup, see `Startup`. On GitHub, personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, see the "Creating a personal access token" article on GitHub Docs.

Usage

```
cred_token(token = "GITHUB_PAT")
```

Arguments

`token`	The name of the environmental variable that holds the personal access token for the authentication. Default is `GITHUB_PAT`.

Value

A list of class `cred_token` with entry:

token The name of the environmental variable that holds the personal access token for the authentication.

See Also

Other git credential functions: `cred_env()`, `cred_ssh_key()`, `cred_user_pass()`

Examples

```
## Not run:
## Create a personal access token credential object.
## This example assumes that the token is stored in
## the 'GITHUB_PAT' environmental variable.
repo <- repository("git2r")
cred <- cred_token()
push(repo, credentials = cred)

## End(Not run)
```

`cred_user_pass` *Create a new plain-text username and password credential object*

Description

Create a new plain-text username and password credential object

Usage

```
cred_user_pass(username = NULL, password = NULL)
```

Arguments

`username`	The username of the credential
`password`	The password of the credential. If getPass is installed and the only input is username, `getPass::getPass()` will be called to allow for interactive and obfuscated interactive input of the password.

Value

A list of class `cred_user_pass` with entries:

username The username of the credential

password The password of the credential

See Also

Other git credential functions: `cred_env()`, `cred_ssh_key()`, `cred_token()`

Examples

```
## Not run:
## Create a plain-text username and password credential object
cred_user_pass("Random Developer", "SecretPassword")

## End(Not run)
```

default_signature *Get the signature*

Description

Get the signature according to the repository's configuration

Usage

```
default_signature(repo = ".")
```

Arguments

repo	a path to a repository or a git_repository object. Default is '.'

Value

A git_signature object with entries:

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Get the default signature
default_signature(repo)

## Change user
config(repo, user.name = "Bob", user.email = "bob@example.org")

## Get the default signature
default_signature(repo)

## End(Not run)
```

descendant_of *Descendant*

Description

Determine if a commit is the descendant of another commit

Usage

```
descendant_of(commit = NULL, ancestor = NULL)
```

Arguments

commit	a git_commit object. Can also be a tag or a branch, and in that case the commit will be the target of the tag or branch.
ancestor	a git_commit object to check if ancestor to commit. Can also be a tag or a branch, and in that case the commit will be the target of the tag or branch.

Value

TRUE if commit is descendant of ancestor, else FALSE

Examples

```
## Not run:
## Create a directory in tempdir
path <- tempfile(pattern="git2r-")
dir.create(path)

## Initialize a repository
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "test.txt"))
add(repo, "test.txt")
commit_1 <- commit(repo, "Commit message 1")
tag_1 <- tag(repo, "Tagname1", "Tag message 1")

# Change file and commit
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path, "test.txt"))
add(repo, "test.txt")
commit_2 <- commit(repo, "Commit message 2")
tag_2 <- tag(repo, "Tagname2", "Tag message 2")
```

```
descendant_of(commit_1, commit_2)
descendant_of(commit_2, commit_1)
descendant_of(tag_1, tag_2)
descendant_of(tag_2, tag_1)

## End(Not run)
```

`diff.git_repository` *Changes between commits, trees, working tree, etc.*

Description

Changes between commits, trees, working tree, etc.

Usage

```
## S3 method for class 'git_repository'
diff(
  x,
  index = FALSE,
  as_char = FALSE,
  filename = NULL,
  context_lines = 3,
  interhunk_lines = 0,
  old_prefix = "a",
  new_prefix = "b",
  id_abbrev = NULL,
  path = NULL,
  max_size = NULL,
  ...
)

## S3 method for class 'git_tree'
diff(
  x,
  new_tree = NULL,
  index = FALSE,
  as_char = FALSE,
  filename = NULL,
  context_lines = 3,
  interhunk_lines = 0,
  old_prefix = "a",
  new_prefix = "b",
  id_abbrev = NULL,
  path = NULL,
  max_size = NULL,
  ...
)
```

Arguments

x	A git_repository object or the old git_tree object to compare to.
index	***When object equals a git_repository*** Whether to compare the index to HEAD. If FALSE (the default), then the working tree is compared to the index. ***When object equals a git_tree*** Whether to use the working directory (by default), or the index (if set to TRUE) in the comparison to object.
as_char	logical: should the result be converted to a character string?. Default is FALSE.
filename	If as_char is TRUE, then the diff can be written to a file with name filename (the file is overwritten if it exists). Default is NULL.
context_lines	The number of unchanged lines that define the boundary of a hunk (and to display before and after). Defaults to 3.
interhunk_lines	The maximum number of unchanged lines between hunk boundaries before the hunks will be merged into one. Defaults to 0.
old_prefix	The virtual "directory" prefix for old file names in hunk headers. Default is "a".
new_prefix	The virtual "directory" prefix for new file names in hunk headers. Defaults to "b".
id_abbrev	The abbreviation length to use when formatting object ids. Defaults to the value of 'core.abbrev' from the config, or 7 if NULL.
path	A character vector of paths / fnmatch patterns to constrain diff. Default is NULL which include all paths.
max_size	A size (in bytes) above which a blob will be marked as binary automatically; pass a negative value to disable. Defaults to 512MB when max_size is NULL.
...	Not used.
new_tree	The new git_tree object to compare, or NULL. If NULL, then we use the working directory or the index (see the index argument).

Value

A git_diff object if as_char is FALSE. If as_char is TRUE and filename is NULL, a character string, else NULL.

Line endings

Different operating systems handle line endings differently. Windows uses both a carriage-return character and a linefeed character to represent a newline in a file. While Linux and macOS use only the linefeed character for a newline in a file. To avoid problems in your diffs, you can configure Git to properly handle line endings using the core.autocrlf setting in the Git config file, see the Git documentation (https://git-scm.com/).

Examples

```
## Not run:
## Initialize a repository
path <- tempfile(pattern="git2r-")
```

```
dir.create(path)
repo <- init(path)

## Config user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add, commit
writeLines("Hello world!", file.path(path, "test.txt"))
add(repo, "test.txt")
commit(repo, "Commit message")

## Change the file
writeLines(c("Hello again!", "Here is a second line", "And a third"),
           file.path(path, "test.txt"))

## diff between index and workdir
diff_1 <- diff(repo)
summary(diff_1)
cat(diff(repo, as_char=TRUE))

## Diff between index and HEAD is empty
diff_2 <- diff(repo, index=TRUE)
summary(diff_2)
cat(diff(repo, index=TRUE, as_char=TRUE))

## Diff between tree and working dir, same as diff_1
diff_3 <- diff(tree(commits(repo)[[1]]))
summary(diff_3)
cat(diff(tree(commits(repo)[[1]]), as_char=TRUE))

## Add changes, diff between index and HEAD is the same as diff_1
add(repo, "test.txt")
diff_4 <- diff(repo, index=TRUE)
summary(diff_4)
cat(diff(repo, index=TRUE, as_char=TRUE))

## Diff between tree and index
diff_5 <- diff(tree(commits(repo)[[1]]), index=TRUE)
summary(diff_5)
cat(diff(tree(commits(repo)[[1]]), index=TRUE, as_char=TRUE))

## Diff between two trees
commit(repo, "Second commit")
tree_1 <- tree(commits(repo)[[2]])
tree_2 <- tree(commits(repo)[[1]])
diff_6 <- diff(tree_1, tree_2)
summary(diff_6)
cat(diff(tree_1, tree_2, as_char=TRUE))

## Binary files
set.seed(42)
writeBin(as.raw((sample(0:255, 1000, replace=TRUE))),
         con=file.path(path, "test.bin"))
```

```
add(repo, "test.bin")
diff_7 <- diff(repo, index=TRUE)
summary(diff_7)
cat(diff(repo, index=TRUE, as_char=TRUE))

## End(Not run)
```

`discover_repository` *Find path to repository for any file*

Description

Find path to repository for any file

Usage

```
discover_repository(path = ".", ceiling = NULL)
```

Arguments

`path`	A character vector specifying the path to a file or folder
`ceiling`	The default is to not use the ceiling argument and start the lookup from path and walk across parent directories. When ceiling is 0, the lookup is only in path. When ceiling is 1, the lookup is in both the path and the parent to path.

Value

Character vector with path (terminated by a file separator) to repository or NULL if this cannot be established.

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "example-1.txt"))
add(repo, "example-1.txt")
commit(repo, "First commit message")

## Create a second file. The file is not added for version control
## in the repository.
dir.create(file.path(path, "example"))
file_2 <- file.path(path, "example/example-2.txt")
```

```
writeLines("Not under version control", file_2)

## Find the path to the repository using the path to the second file
discover_repository(file_2)

## Demonstrate the 'ceiling' argument
wd <- workdir(repo)
dir.create(file.path(wd, "temp"))

## Lookup repository in 'file.path(wd, "temp")'. Should return NULL
discover_repository(file.path(wd, "temp"), ceiling = 0)

## Lookup repository in parent to 'file.path(wd, "temp")'.
## Should not return NULL
discover_repository(file.path(wd, "temp"), ceiling = 1)

## End(Not run)
```

`fetch` *Fetch new data and update tips*

Description

Fetch new data and update tips

Usage

```
fetch(
  repo = ".",
  name = NULL,
  credentials = NULL,
  verbose = TRUE,
  refspec = NULL
)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`name`	the remote's name
`credentials`	The credentials for remote repository access. Default is NULL. To use and query an ssh-agent for the ssh key credentials, let this parameter be NULL (the default).
`verbose`	Print information each time a reference is updated locally. Default is `TRUE`.
`refspec`	The refs to fetch and which local refs to update, see examples. Pass NULL to use the `remote.<repository>.fetch` variable. Default is `NULL`.

Value

invisible list of class `git_transfer_progress` with statistics from the fetch operation:

total_objects Number of objects in the packfile being downloaded

indexed_objects Received objects that have been hashed

received_objects Objects which have been downloaded

total_deltas Total number of deltas in the pack

indexed_deltas Deltas which have been indexed

local_objects Locally-available objects that have been injected in order to fix a thin pack

received_bytes Size of the packfile received up to now

Examples

```
## Not run:
## Initialize three temporary repositories
path_bare <- tempfile(pattern="git2r-")
path_repo_1 <- tempfile(pattern="git2r-")
path_repo_2 <- tempfile(pattern="git2r-")

dir.create(path_bare)
dir.create(path_repo_1)
dir.create(path_repo_2)

bare_repo <- init(path_bare, bare = TRUE)
repo_1 <- clone(path_bare, path_repo_1)
repo_2 <- clone(path_bare, path_repo_2)

config(repo_1, user.name = "Alice", user.email = "alice@example.org")
config(repo_2, user.name = "Bob", user.email = "bob@example.org")

## Add changes to repo 1
writeLines("Lorem ipsum dolor sit amet",
           con = file.path(path_repo_1, "example.txt"))
add(repo_1, "example.txt")
commit(repo_1, "Commit message")

## Push changes from repo 1 to origin (bare_repo)
push(repo_1, "origin", "refs/heads/master")

## Fetch changes from origin (bare_repo) to repo 2
fetch(repo_2, "origin")

## List updated heads
fetch_heads(repo_2)

## Checking out GitHub pull requests locally
path <- tempfile(pattern="ghit-")
repo <- clone("https://github.com/leeper/ghit", path)
fetch(repo, "origin", refspec = "pull/13/head:refs/heads/BRANCHNAME")
checkout(repo, "BRANCHNAME")
```

```
summary(repo)

## End(Not run)
```

`fetch_heads` *Get updated heads during the last fetch.*

Description

Get updated heads during the last fetch.

Usage

```
fetch_heads(repo = ".")
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'

Value

list with `git_fetch_head` entries. NULL if there is no FETCH_HEAD file.

Examples

```
## Not run:
## Initialize three temporary repositories
path_bare <- tempfile(pattern="git2r-")
path_repo_1 <- tempfile(pattern="git2r-")
path_repo_2 <- tempfile(pattern="git2r-")

dir.create(path_bare)
dir.create(path_repo_1)
dir.create(path_repo_2)

bare_repo <- init(path_bare, bare = TRUE)
repo_1 <- clone(path_bare, path_repo_1)
repo_2 <- clone(path_bare, path_repo_2)

config(repo_1, user.name = "Alice", user.email = "alice@example.org")
config(repo_2, user.name = "Bob", user.email = "bob@example.org")

## Add changes to repo 1
writeLines("Lorem ipsum dolor sit amet",
           con = file.path(path_repo_1, "example.txt"))
add(repo_1, "example.txt")
commit(repo_1, "Commit message")

## Push changes from repo 1 to origin (bare_repo)
push(repo_1, "origin", "refs/heads/master")
```

```
## Fetch changes from origin (bare_repo) to repo 2
fetch(repo_2, "origin")

## List updated heads
fetch_heads(repo_2)

## End(Not run)
```

`git2r` *git2r: R bindings to the libgit2 library*

Description

git2r: R bindings to the libgit2 library.

Author(s)

Maintainer: Stefan Widgren <stefan.widgren@gmail.com> (ORCID)

Other contributors:

- Gabor Csardi [contributor]
- Gregory Jefferis [contributor]
- Jennifer Bryan [contributor]
- Jeroen Ooms [contributor]
- Jim Hester [contributor]
- John Blischak [contributor]
- Karthik Ram [contributor]
- Peter Carbonetto [contributor]
- Scott Chamberlain [contributor]
- Thomas Rosendal [contributor]

See Also

Useful links:

- https://docs.ropensci.org/git2r/
- https://github.com/ropensci/git2r
- Report bugs at https://github.com/ropensci/git2r/issues

git_config_files *Locate the path to configuration files*

Description

Potential configuration files:

system Locate the path to the system configuration file. If '/etc/gitconfig' doesn't exist, it will look for '%PROGRAMFILES%'.

xdg Locate the path to the global xdg compatible configuration file. The xdg compatible configuration file is usually located in '$HOME/.config/git/config'. This method will try to guess the full path to that file, if the file exists.

global The user or global configuration file is usually located in '$HOME/.gitconfig'. This method will try to guess the full path to that file, if the file exists.

local Locate the path to the repository specific configuration file, if the file exists.

Usage

```
git_config_files(repo = ".")
```

Arguments

repo	a path to a repository or a git_repository object. Default is '.'

Value

a data.frame with one row per potential configuration file where NA means not found.

git_time *Time*

Description

The class git_time stores the time a Git object was created.

Usage

```
## S3 method for class 'git_time'
as.character(x, tz = "GMT", origin = "1970-01-01", usetz = TRUE, ...)

## S3 method for class 'git_time'
format(x, tz = "GMT", origin = "1970-01-01", usetz = TRUE, ...)

## S3 method for class 'git_time'
as.POSIXct(x, tz = "GMT", origin = "1970-01-01", ...)

## S3 method for class 'git_time'
print(x, tz = "GMT", origin = "1970-01-01", usetz = TRUE, ...)
```

Arguments

x	R object to be converted.
tz	a character string. The time zone specification to be used for the conversion, *if one is required*. System-specific (see time zones), but "" is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.
origin	a date-time object, or something which can be coerced by as.POSIXct(tz = "GMT") to such an object. Optional since R 4.3.0, where the equivalent of "1970-01-01" is used.
usetz	logical. Should the time zone abbreviation be appended to the output? This is used in printing times, and more reliable than using "%Z".
...	further arguments to be passed to or from other methods.

Details

The default is to use tz = "GMT" and origin = "1970-01-01". To use your local timezone, set tz = Sys.timezone().

See Also

when

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a first user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Create tag
tag(repo, "Tagname", "Tag message")

as.POSIXct(commits(repo)[[1]]$author$when)
as.POSIXct(tags(repo)[[1]]$tagger$when)
as.POSIXct(tags(repo)[[1]]$tagger$when, tz = Sys.timezone())

## End(Not run)
```

hash *Determine the sha from a blob string*

Description

The blob is not written to the object database.

Usage

```
hash(data = NULL)
```

Arguments

data	The string vector to hash.

Value

A string vector with the sha for each string in data.

Examples

```
## Not run:
identical(hash(c("Hello, world!\n",
                 "test content\n")),
               c("af5626b4a114abcb82d63db7c8082c3c4756e51b",
                 "d670460b4b4aece5915caf5c68d12f560a9fe3e4"))

## End(Not run)
```

hashfile *Determine the sha from a blob in a file*

Description

The blob is not written to the object database.

Usage

```
hashfile(path = NULL)
```

Arguments

path	The path vector with files to hash.

Value

A vector with the sha for each file in path.

Examples

```
## Not run:
## Create a file. NOTE: The line endings from writeLines gives
## LF (line feed) on Unix/Linux and CRLF (carriage return, line feed)
## on Windows. The example use writeChar to have more control.
path <- tempfile()
f <- file(path, "wb")
writeChar("Hello, world!\n", f, eos = NULL)
close(f)

## Generate hash
hashfile(path)
identical(hashfile(path), hash("Hello, world!\n"))

## End(Not run)
```

`head.git_repository` *Get HEAD for a repository*

Description

Get HEAD for a repository

Usage

```
## S3 method for class 'git_repository'
head(x, ...)
```

Arguments

`x`	The repository `x` to check head
`...`	Additional arguments. Unused.

Value

NULL if unborn branch or not found. A git_branch if not a detached head. A git_commit if detached head

Examples

```
## Not run:
## Create and initialize a repository in a temporary directory
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
writeLines("Hello world!", file.path(path, "example.txt"))
```

```
add(repo, "example.txt")
commit(repo, "Commit message")

## Get HEAD of repository
repository_head(repo)

## End(Not run)
```

`index_remove_bypath` *Remove an index entry corresponding to a file on disk*

Description

Remove an index entry corresponding to a file on disk

Usage

```
index_remove_bypath(repo = ".", path = NULL)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`path`	character vector with filenames to remove. The path must be relative to the repository's working folder. It may exist. If this file currently is the result of a merge conflict, this file will no longer be marked as conflicting. The data about the conflict will be moved to the "resolve undo" (REUC) section.

Value

invisible(NULL)

Examples

```
## Not run:
## Initialize a repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file
writeLines("Hello world!", file.path(path, "file-to-remove.txt"))

## Add file to repository
add(repo, "file-to-remove.txt")

## View status of repository
```

```
status(repo)

## Remove file
index_remove_bypath(repo, "file-to-remove.txt")

## View status of repository
status(repo)

## End(Not run)
```

`init` *Init a repository*

Description

Init a repository

Usage

```
init(path = ".", bare = FALSE, branch = NULL)
```

Arguments

`path`	A path to where to init a git repository
`bare`	If TRUE, a Git repository without a working directory is created at the pointed path. If FALSE, provided path will be considered as the working directory into which the .git directory will be created.
`branch`	Use the specified name for the initial branch in the newly created repository. If `branch=NULL`, fall back to the default name.

Value

A `git_repository` object

See Also

repository

Examples

```
## Not run:
## Initialize a repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)
is_bare(repo)

## Initialize a bare repository
path_bare <- tempfile(pattern="git2r-")
```

```
dir.create(path_bare)
repo_bare <- init(path_bare, bare = TRUE)
is_bare(repo_bare)

## End(Not run)
```

`in_repository` *Determine if a directory is in a git repository*

Description

The lookup start from path and walk across parent directories if nothing has been found.

Usage

```
in_repository(path = ".")
```

Arguments

`path`	The path to the directory.

Value

TRUE if directory is in a git repository else FALSE

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Check if path is in a git repository
in_repository(path)

## Check if working directory is in a git repository
setwd(path)
in_repository()

## End(Not run)
```

`is_bare` *Check if repository is bare*

Description

Check if repository is bare

Usage

```
is_bare(repo = ".")
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'

Value

`TRUE` if bare repository, else `FALSE`

See Also

init

Examples

```
## Not run:
## Initialize a repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)
is_bare(repo)

## Initialize a bare repository
path_bare <- tempfile(pattern="git2r-")
dir.create(path_bare)
repo_bare <- init(path_bare, bare = TRUE)
is_bare(repo_bare)

## End(Not run)
```

`is_binary` *Is blob binary*

Description

Is blob binary

Usage

```
is_binary(blob = NULL)
```

Arguments

`blob`	The blob `object`.

Value

TRUE if binary data, FALSE if not.

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Commit a text file
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit_1 <- commit(repo, "First commit message")

## Check if binary
b_text <- tree(commit_1)["example.txt"]
is_binary(b_text)

## Commit plot file (binary)
x <- 1:100
y <- x^2
png(file.path(path, "plot.png"))
plot(y ~ x, type = "l")
dev.off()
add(repo, "plot.png")
commit_2 <- commit(repo, "Second commit message")

## Check if binary
b_png <- tree(commit_2)["plot.png"]
```

```
is_binary(b_png)

## End(Not run)
```

`is_blob` *Check if object is S3 class git_blob*

Description

Check if object is S3 class git_blob

Usage

```
is_blob(object)
```

Arguments

`object`	Check if object is S3 class git_blob

Value

TRUE if object is S3 class git_blob, else FALSE

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Commit a text file
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit_1 <- commit(repo, "First commit message")
blob_1 <- tree(commit_1)["example.txt"]

## Check if blob
is_blob(commit_1)
is_blob(blob_1)

## End(Not run)
```

`is_branch` *Check if object is* `git_branch`

Description

Check if object is `git_branch`

Usage

```
is_branch(object)
```

Arguments

`object`	Check if object is of class `git_branch`

Value

TRUE if object is class `git_branch`, else FALSE

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Commit a text file
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

branch <- branches(repo)[[1]]

## Check if branch
is_branch(branch)

## End(Not run)
```

is_commit *Check if object is a git_commit object*

Description

Check if object is a git_commit object

Usage

```
is_commit(object)
```

Arguments

object	Check if object is a git_commit object

Value

TRUE if object is a git_commit, else FALSE

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Commit a text file
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit_1 <- commit(repo, "First commit message")

## Check if commit
is_commit(commit_1)

## End(Not run)
```

`is_detached` *Check if HEAD of repository is detached*

Description

Check if HEAD of repository is detached

Usage

```
is_detached(repo = ".")
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'

Value

`TRUE` if repository HEAD is detached, else `FALSE`.

Examples

```
## Not run:
## Create and initialize a repository in a temporary directory
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit_1 <- commit(repo, "Commit message 1")

## Change file, add and commit
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "Commit message 2")

## HEAD of repository is not detached
is_detached(repo)

## Checkout first commit
checkout(commit_1)

## HEAD of repository is detached
is_detached(repo)
```

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

`is_empty` *Check if repository is empty*

Description

Check if repository is empty

Usage

```
is_empty(repo = ".")
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'

Value

`TRUE` if repository is empty else `FALSE`.

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Check if it's an empty repository
is_empty(repo)

## Commit a file
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Check if it's an empty repository
is_empty(repo)

## End(Not run)
```

is_head *Check if branch is head*

Description

Check if branch is head

Usage

```
is_head(branch = NULL)
```

Arguments

branch	The branch object to check if it's head.

Value

TRUE if branch is head, else FALSE.

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## List branches
branches(repo)

## Check that 'master' is_head
master <- branches(repo)[[1]]
is_head(master)

## Create and checkout 'dev' branch
checkout(repo, "dev", create = TRUE)

## List branches
branches(repo)

## Check that 'master' is no longer head
is_head(master)

## End(Not run)
```

is_local *Check if branch is local*

Description

Check if branch is local

Usage

```
is_local(branch)
```

Arguments

branch	The branch object to check if it's local

Value

TRUE if branch is local, else FALSE.

Examples

```
## Not run:
## Initialize repositories
path_bare <- tempfile(pattern="git2r-")
path_repo <- tempfile(pattern="git2r-")
dir.create(path_bare)
dir.create(path_repo)
repo_bare <- init(path_bare, bare = TRUE)
repo <- clone(path_bare, path_repo)

## Config first user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path_repo, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Push commits from repository to bare repository
## Adds an upstream tracking branch to branch 'master'
push(repo, "origin", "refs/heads/master")

## List branches
branches(repo)

## Check if first branch is_local
is_local(branches(repo)[[1]])

## Check if second branch is_local
```

```
is_local(branches(repo)[[2]])

## End(Not run)
```

`is_merge` *Is merge*

Description

Determine if a commit is a merge commit, i.e. has more than one parent.

Usage

```
is_merge(commit = NULL)
```

Arguments

`commit`	a git_commit object.

Value

TRUE if commit has more than one parent, else FALSE

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
writeLines(c("First line in file 1.", "Second line in file 1."),
           file.path(path, "example-1.txt"))
add(repo, "example-1.txt")
commit(repo, "First commit message")

## Create and add one more file
writeLines(c("First line in file 2.", "Second line in file 2."),
           file.path(path, "example-2.txt"))
add(repo, "example-2.txt")
commit(repo, "Second commit message")

## Create a new branch 'fix'
checkout(repo, "fix", create = TRUE)

## Update 'example-1.txt' (swap words in first line) and commit
writeLines(c("line First in file 1.", "Second line in file 1."),
           file.path(path, "example-1.txt"))
```

```
add(repo, "example-1.txt")
commit(repo, "Third commit message")

checkout(repo, "master")

## Update 'example-2.txt' (swap words in second line) and commit
writeLines(c("First line in file 2.", "line Second in file 2."),
           file.path(path, "example-2.txt"))
add(repo, "example-2.txt")
commit(repo, "Fourth commit message")

## Merge 'fix'
merge(repo, "fix")

## Display parents of last commit
parents(lookup(repo, branch_target(repository_head(repo))))

## Check that last commit is a merge
is_merge(lookup(repo, branch_target(repository_head(repo))))

## End(Not run)
```

`is_shallow` *Determine if the repository is a shallow clone*

Description

Determine if the repository is a shallow clone

Usage

```
is_shallow(repo = ".")
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'

Value

`TRUE` if shallow clone, else `FALSE`

Examples

```
## Not run:
## Initialize repository
path_repo_1 <- tempfile(pattern="git2r-")
path_repo_2 <- tempfile(pattern="git2r-")
dir.create(path_repo_1)
dir.create(path_repo_2)
repo_1 <- init(path_repo_1)
```

```
## Config user and commit a file
config(repo_1, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path_repo_1, "example.txt"))
add(repo_1, "example.txt")
commit(repo_1, "First commit message")

## Change file and commit
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path_repo_1, "example.txt"))
add(repo_1, "example.txt")
commit(repo_1, "Second commit message")

## Change file again and commit.
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad",
  "minim veniam, quis nostrud exercitation ullamco laboris nisi ut")
writeLines(lines, file.path(path_repo_1, "example.txt"))
add(repo_1, "example.txt")
commit(repo_1, "Third commit message")

## Clone to second repository
repo_2 <- clone(path_repo_1, path_repo_2)

## Check if it's a shallow clone
is_shallow(repo_2)

## End(Not run)
```

`is_tag` *Check if object is a git_tag object*

Description

Check if object is a git_tag object

Usage

```
is_tag(object)
```

Arguments

`object`	Check if object is a git_tag object

Value

TRUE if object is a git_tag, else FALSE

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Commit a text file
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Create tag
tag(repo, "Tagname", "Tag message")

is_tag(tags(repo)[[1]])
is_tag(last_commit(repo))

## End(Not run)
```

---

is_tree *Check if object is S3 class git_tree*

Description

Check if object is S3 class git_tree

Usage

```
is_tree(object)
```

Arguments

object	Check if object is S3 class git_tree

Value

TRUE if object is S3 class git_tree, else FALSE

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Commit a text file
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit_1 <- commit(repo, "First commit message")
tree_1 <- tree(commit_1)

## Check if tree
is_tree(commit_1)
is_tree(tree_1)

## End(Not run)
```

---

last_commit *Last commit*

Description

Get last commit in the current branch.

Usage

```
last_commit(repo = ".")
```

Arguments

repo	a path to a repository or a `git_repository` object. Default is '.'

Examples

```
## Not run:
## Initialize a repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Config user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
```

```
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Get last commit
last_commit(repo)
last_commit(path)

## Coerce the last commit to a data.frame
as.data.frame(last_commit(path), "data.frame")

## Summary of last commit in repository
summary(last_commit(repo))

## End(Not run)
```

`length.git_blob` *Size in bytes of the contents of a blob*

Description

Size in bytes of the contents of a blob

Usage

```
## S3 method for class 'git_blob'
length(x)
```

Arguments

`x`	The blob `object`

Value

a non-negative integer

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Commit a text file
writeLines("Hello world!", file.path(path, "example.txt"))
```

```
add(repo, "example.txt")
commit_1 <- commit(repo, "First commit message")
blob_1 <- tree(commit_1)["example.txt"]

## Get length in size of bytes of the content of the blob
length(blob_1)

## End(Not run)
```

`length.git_diff` *Number of files in git_diff object*

Description

Number of files in git_diff object

Usage

```
## S3 method for class 'git_diff'
length(x)
```

Arguments

`x`	The git_diff `object`

Value

a non-negative integer

`length.git_tree` *Number of entries in tree*

Description

Number of entries in tree

Usage

```
## S3 method for class 'git_tree'
length(x)
```

Arguments

`x`	The tree `object`

Value

a non-negative integer or double (which will be rounded down)

libgit2_features *Compile time options for libgit2.*

Description

Compile time options for libgit2.

Usage

```
libgit2_features()
```

Value

A list with threads, https and ssh set to TRUE/FALSE.

Examples

```
libgit2_features()
```

libgit2_version *Version of the libgit2 library*

Description

Version of the libgit2 library that the bundled source code is based on

Usage

```
libgit2_version()
```

Value

A list with major, minor and rev

Examples

```
libgit2_version()
```

lookup *Lookup*

Description

Lookup one object in a repository.

Usage

```
lookup(repo = ".", sha = NULL)
```

Arguments

repo	a path to a repository or a git_repository object. Default is '.'
sha	The identity of the object to lookup. Must be 4 to 40 characters long.

Value

a git_blob or git_commit or git_tag or git_tree object

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit_1 <- commit(repo, "First commit message")

## Create tag
tag(repo, "Tagname", "Tag message")

## First, get SHAs to lookup in the repository
sha_commit <- sha(commit_1)
sha_tree <- sha(tree(commit_1))
sha_blob <- sha(tree(commit_1)["example.txt"])
sha_tag <- sha(tags(repo)[[1]])

## SHAs
sha_commit
sha_tree
sha_blob
sha_tag
```

```
## Lookup objects
lookup(repo, sha_commit)
lookup(repo, sha_tree)
lookup(repo, sha_blob)
lookup(repo, sha_tag)

## Lookup objects, using only the first seven characters
lookup(repo, substr(sha_commit, 1, 7))
lookup(repo, substr(sha_tree, 1, 7))
lookup(repo, substr(sha_blob, 1, 7))
lookup(repo, substr(sha_tag, 1, 7))

## End(Not run)
```

`lookup_commit` *Lookup the commit related to a git object*

Description

Lookup the commit related to a git_reference, git_tag or git_branch object.

Usage

```
lookup_commit(object)

## S3 method for class 'git_branch'
lookup_commit(object)

## S3 method for class 'git_commit'
lookup_commit(object)

## S3 method for class 'git_tag'
lookup_commit(object)

## S3 method for class 'git_reference'
lookup_commit(object)
```

Arguments

`object`	a git object to get the related commit from.

Value

A git commit object.

Examples

```
## Not run:
## Create a directory in tempdir
path <- tempfile(pattern="git2r-")
dir.create(path)

## Initialize a repository
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, con = file.path(path, "test.txt"))
add(repo, "test.txt")
commit(repo, "Commit message 1")

## Get the commit pointed to by the 'master' branch
lookup_commit(repository_head(repo))

## Create a tag
a_tag <- tag(repo, "Tagname", "Tag message")

## Get the commit pointed to by 'a_tag'
lookup_commit(a_tag)

## End(Not run)
```

`ls_tree` *List the contents of a tree object*

Description

Traverse the entries in a tree and its subtrees. Akin to the 'git ls-tree' command.

Usage

```
ls_tree(tree = NULL, repo = ".", recursive = TRUE)
```

Arguments

`tree`	default (`NULL`) is the tree of the last commit in `repo`. Can also be a `git_tree` object or a character that identifies a tree in the repository (see 'Examples').
`repo`	never used if `tree` is a `git_tree` object. A `git_repository` object, or a path (default = '.') to a repository.
`recursive`	default is to recurse into sub-trees.

Value

A data.frame with the following columns:

mode UNIX file attribute of the tree entry

type type of object

sha sha of the object

path path relative to the root tree

name filename of the tree entry

len object size of blob (file) entries. NA for other objects.

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
dir.create(file.path(path, "subfolder"))
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create three files and commit
writeLines("First file",  file.path(path, "example-1.txt"))
writeLines("Second file", file.path(path, "subfolder/example-2.txt"))
writeLines("Third file",  file.path(path, "example-3.txt"))
add(repo, c("example-1.txt", "subfolder/example-2.txt", "example-3.txt"))
commit(repo, "Commit message")

## Traverse tree entries and its subtrees.
## Various approaches that give identical result.
ls_tree(tree = tree(last_commit(path)))
ls_tree(tree = tree(last_commit(repo)))
ls_tree(repo = path)
ls_tree(repo = repo)

## Skip content in subfolder
ls_tree(repo = repo, recursive = FALSE)

## Start in subfolder
ls_tree(tree = "HEAD:subfolder", repo = repo)

## End(Not run)
```

`merge.git_branch` *Merge a branch into HEAD*

Description

Merge a branch into HEAD

Usage

```
## S3 method for class 'git_branch'
merge(x, y = NULL, commit_on_success = TRUE, merger = NULL, fail = FALSE, ...)

## S3 method for class 'git_repository'
merge(x, y = NULL, commit_on_success = TRUE, merger = NULL, fail = FALSE, ...)

## S3 method for class 'character'
merge(
  x = ".",
  y = NULL,
  commit_on_success = TRUE,
  merger = NULL,
  fail = FALSE,
  ...
)
```

Arguments

`x`	A path (default '.') to a repository, or a `git_repository` object, or a `git_branch`.
`y`	If `x` is a `git_repository`, the name of the branch to merge into HEAD. Not used if `x` is a `git_branch`.
`commit_on_success`	If there are no conflicts written to the index, the merge commit will be committed. Default is TRUE.
`merger`	Who made the merge. The default (`NULL`) is to use `default_signature` for the repository.
`fail`	If a conflict occurs, exit immediately instead of attempting to continue resolving conflicts. Default is `FALSE`.
`...`	Additional arguments (unused).

Value

A list of class `git_merge_result` with entries:

up_to_date TRUE if the merge is already up-to-date, else FALSE.

fast_forward TRUE if a fast-forward merge, else FALSE.

conflicts TRUE if the index contain entries representing file conflicts, else FALSE.

sha If the merge created a merge commit, the sha of the merge commit. NA if no merge commit created.

Examples

```
## Not run:
## Create a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)
config(repo, user.name="Alice", user.email = "alice@example.org")

## Create a file, add and commit
writeLines("Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
           con = file.path(path, "test.txt"))
add(repo, "test.txt")
commit_1 <- commit(repo, "Commit message 1")

## Create first branch, checkout, add file and commit
checkout(repo, "branch1", create = TRUE)
writeLines("Branch 1", file.path(path, "branch-1.txt"))
add(repo, "branch-1.txt")
commit(repo, "Commit message branch 1")

## Create second branch, checkout, add file and commit
b_2 <- branch_create(commit_1, "branch2")
checkout(b_2)
writeLines("Branch 2", file.path(path, "branch-2.txt"))
add(repo, "branch-2.txt")
commit(repo, "Commit message branch 2")

## Make a change to 'test.txt'
writeLines(c("Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
             "eiusmod tempor incididunt ut labore et dolore magna aliqua."),
           con = file.path(path, "test.txt"))
add(repo, "test.txt")
commit(repo, "Second commit message branch 2")

## Checkout master
checkout(repo, "master", force = TRUE)

## Merge branch 1
merge(repo, "branch1")

## Merge branch 2
merge(repo, "branch2")

## Create third branch, checkout, change file and commit
checkout(repo, "branch3", create=TRUE)
writeLines(c("Lorem ipsum dolor amet sit, consectetur adipisicing elit, sed do",
             "eiusmod tempor incididunt ut labore et dolore magna aliqua."),
           con = file.path(path, "test.txt"))
add(repo, "test.txt")
```

```
commit(repo, "Commit message branch 3")

## Checkout master and create a change that creates a merge conflict
checkout(repo, "master", force=TRUE)
writeLines(c("Lorem ipsum dolor sit amet, adipisicing consectetur elit, sed do",
             "eiusmod tempor incididunt ut labore et dolore magna aliqua."),
           con = file.path(path, "test.txt"))
add(repo, "test.txt")
commit(repo, "Some commit message branch 1")

## Merge branch 3
merge(repo, "branch3")

## Check status; Expect to have one unstaged unmerged conflict.
status(repo)

## End(Not run)
```

`merge_base` *Find a merge base between two commits*

Description

Find a merge base between two commits

Usage

```
merge_base(one = NULL, two = NULL)
```

Arguments

one	One of the commits
two	The other commit

Value

git_commit

Examples

```
## Not run:
## Create a directory in tempdir
path <- tempfile(pattern="git2r-")
dir.create(path)

## Initialize a repository
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")
```

```
## Create a file, add and commit
writeLines("Master branch", file.path(path, "master_branch.txt"))
add(repo, "master_branch.txt")
commit_1 <- commit(repo, "Commit message 1")

## Create first branch, checkout, add file and commit
branch_1 <- branch_create(commit_1, "branch_1")
checkout(branch_1)
writeLines("Branch 1", file.path(path, "branch_1.txt"))
add(repo, "branch_1.txt")
commit_2 <- commit(repo, "Commit message branch_1")

## Create second branch, checkout, add file and commit
branch_2 <- branch_create(commit_1, "branch_2")
checkout(branch_2)
writeLines("Branch 2", file.path(path, "branch_2.txt"))
add(repo, "branch_2.txt")
commit_3 <- commit(repo, "Commit message branch_2")

## Check that merge base equals commit_1
stopifnot(identical(merge_base(commit_2, commit_3), commit_1))

## End(Not run)
```

`notes` *List notes*

Description

List all the notes within a specified namespace.

Usage

```
notes(repo = ".", ref = NULL)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`ref`	Reference to read from. Default (ref = NULL) is to call `note_default_ref`.

Value

list with git_note objects

Examples

```
## Not run:
## Create and initialize a repository in a temporary directory
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit_1 <- commit(repo, "Commit message 1")

## Create another commit
writeLines(c("Hello world!",
             "HELLO WORLD!"),
           file.path(path, "example.txt"))
add(repo, "example.txt")
commit_2 <- commit(repo, "Commit message 2")

## Create note in default namespace
note_create(commit_1, "Note-1")
note_create(commit_1, "Note-2", force = TRUE)

## Create note in named (review) namespace
note_create(commit_1, "Note-3", ref="refs/notes/review")
note_create(commit_2, "Note-4", ref="review")

## Create note on blob and tree
note_create(tree(commit_1), "Note-5")
note_create(tree(commit_1)["example.txt"], "Note-6")

## List notes in default namespace
notes(repo)

## List notes in 'review' namespace
notes(repo, "review")

## End(Not run)
```

`note_create` *Add note for a object*

Description

Add note for a object

Usage

```
note_create(
  object = NULL,
  message = NULL,
  ref = NULL,
  author = NULL,
  committer = NULL,
  force = FALSE
)
```

Arguments

object	The object to annotate (git_blob, git_commit or git_tree).
message	Content of the note to add
ref	Canonical name of the reference to use. Default is note_default_ref.
author	Signature of the notes note author
committer	Signature of the notes note committer
force	Overwrite existing note. Default is FALSE

Value

git_note

Examples

```
## Not run:
## Create and initialize a repository in a temporary directory
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit_1 <- commit(repo, "Commit message 1")

## Create another commit
writeLines(c("Hello world!",
             "HELLO WORLD!"),
           file.path(path, "example.txt"))
add(repo, "example.txt")
commit_2 <- commit(repo, "Commit message 2")

## Check that notes is an empty list
notes(repo)

## Create note in default namespace
note_create(commit_1, "Note-1")
```

```
## Create note in named (review) namespace
note_create(commit_1, "Note-2", ref="refs/notes/review")
note_create(commit_2, "Note-3", ref="review")

## Create note on blob and tree
note_create(tree(commit_1), "Note-4")
note_create(tree(commit_1)["example.txt"], "Note-5")

## End(Not run)
```

`note_default_ref` *Default notes reference*

Description

Get the default notes reference for a repository

Usage

```
note_default_ref(repo = ".")
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'

Value

Character vector of length one with name of default notes reference

Examples

```
## Not run:
## Create and initialize a repository in a temporary directory
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## View default notes reference
note_default_ref(repo)

## End(Not run)
```

note_remove *Remove the note for an object*

Description

Remove the note for an object

Usage

```
note_remove(note = NULL, author = NULL, committer = NULL)
```

Arguments

note	The note to remove
author	Signature of the notes commit author.
committer	Signature of the notes commit committer.

Value

invisible NULL

Examples

```
## Not run:
## Create and initialize a repository in a temporary directory
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit_1 <- commit(repo, "Commit message 1")


## Create note in default namespace
note_1 <- note_create(commit_1, "Note-1")

## Create note in named (review) namespace
note_2 <- note_create(commit_1, "Note-2", ref="refs/notes/review")

## List notes in default namespace
notes(repo)

## List notes in 'review' namespace
notes(repo, "review")

## Remove notes
```

```
note_remove(note_1)
note_remove(note_2)

## List notes in default namespace
notes(repo)

## List notes in 'review' namespace
notes(repo, "review")

## End(Not run)
```

---

odb_blobs *Blobs in the object database*

Description

List all blobs reachable from the commits in the object database. For each commit, list blob's in the commit tree and sub-trees.

Usage

```
odb_blobs(repo = ".")
```

Arguments

repo	a path to a repository or a `git_repository` object. Default is '.'

Value

A data.frame with the following columns:

sha The sha of the blob

path The path to the blob from the tree and sub-trees

name The name of the blob from the tree that contains the blob

len The length of the blob

commit The sha of the commit

author The author of the commit

when The timestamp of the author signature in the commit

Note

A blob sha can have several entries

Examples

```
## Not run:
## Create a directory in tempdir
path <- tempfile(pattern="git2r-")
dir.create(path)

## Initialize a repository
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "test.txt"))
add(repo, "test.txt")
commit(repo, "Commit message 1")

## Change file and commit
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path, "test.txt"))
add(repo, "test.txt")
commit(repo, "Commit message 2")

## Commit same content under different name in a sub-directory
dir.create(file.path(path, "sub-directory"))
file.copy(file.path(path, "test.txt"),
          file.path(path, "sub-directory", "copy.txt"))
add(repo, "sub-directory/copy.txt")
commit(repo, "Commit message 3")

## List blobs
odb_blobs(repo)

## End(Not run)
```

`odb_objects` *List all objects available in the database*

Description

List all objects available in the database

Usage

```
odb_objects(repo = ".")
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'

Value

A data.frame with the following columns:

sha The sha of the object

type The type of the object

len The length of the object

Examples

```
## Not run:
## Create a directory in tempdir
path <- tempfile(pattern="git2r-")
dir.create(path)

## Initialize a repository
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "test.txt"))
add(repo, "test.txt")
commit(repo, "Commit message 1")

## Create tag
tag(repo, "Tagname", "Tag message")

## List objects in repository
odb_objects(repo)

## End(Not run)
```

---

parents *Parents*

Description

Get parents of a commit.

Usage

```
parents(object = NULL)
```

Arguments

object	a git_commit object.

Value

list of git_commit objects

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
writeLines("First line.",
           file.path(path, "example.txt"))
add(repo, "example.txt")
commit_1 <- commit(repo, "First commit message")

## commit_1 has no parents
parents(commit_1)

## Update 'example.txt' and commit
writeLines(c("First line.", "Second line."),
           file.path(path, "example.txt"))
add(repo, "example.txt")
commit_2 <- commit(repo, "Second commit message")

## commit_2 has commit_1 as parent
parents(commit_2)

## End(Not run)
```

---

plot.git_repository *Plot commits over time*

Description

Plot commits over time

Usage

```
## S3 method for class 'git_repository'
plot(
  x,
  breaks = c("month", "year", "quarter", "week", "day"),
  main = NULL,
  ...
)
```

Arguments

x	The repository to plot
breaks	Default is month. Change to year, quarter, week or day as necessary.
main	Default title for the plot is "Commits on repo:" and repository workdir basename. Supply a new title if you desire one.
...	Additional arguments affecting the plot

Examples

```
## Not run:
## Initialize repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- clone("https://github.com/ropensci/git2r.git", path)

## Plot commits
plot(repo)

## End(Not run)
```

---

print.git_reflog_entry

Print a reflog entry

Description

Print a reflog entry

Usage

```
## S3 method for class 'git_reflog_entry'
print(x, ...)
```

Arguments

x	The reflog entry
...	Unused

Value

None (invisible 'NULL').

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## View repository HEAD reflog
reflog(repo)

## End(Not run)
```

`pull` *Pull*

Description

Pull

Usage

```
pull(repo = ".", credentials = NULL, merger = NULL)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`credentials`	The credentials for remote repository access. Default is NULL. To use and query an ssh-agent for the ssh key credentials, let this parameter be NULL (the default).
`merger`	Who made the merge, if the merge is non-fast forward merge that creates a merge commit. The `default_signature` for `repo` is used if this parameter is `NULL`.

Value

A list of class `git_merge_result` with entries:

up_to_date TRUE if the merge is already up-to-date, else FALSE.

fast_forward TRUE if a fast-forward merge, else FALSE.

conflicts TRUE if the index contain entries representing file conflicts, else FALSE.

sha If the merge created a merge commit, the sha of the merge commit. NA if no merge commit created.

Examples

```
## Not run:
## Initialize repositories
path_bare <- tempfile(pattern="git2r-")
path_repo_1 <- tempfile(pattern="git2r-")
path_repo_2 <- tempfile(pattern="git2r-")
dir.create(path_bare)
dir.create(path_repo_1)
dir.create(path_repo_2)
repo_bare <- init(path_bare, bare = TRUE)
repo_1 <- clone(path_bare, path_repo_1)

## Config first user and commit a file
config(repo_1, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path_repo_1, "example.txt"))
add(repo_1, "example.txt")
commit(repo_1, "First commit message")

## Push commits from first repository to bare repository
## Adds an upstream tracking branch to branch 'master'
push(repo_1, "origin", "refs/heads/master")

## Clone to second repository
repo_2 <- clone(path_bare, path_repo_2)
config(repo_2, user.name = "Bob", user.email = "bob@example.org")

## Change file and commit
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path_repo_1, "example.txt"))
add(repo_1, "example.txt")
commit(repo_1, "Second commit message")

## Push commits from first repository to bare repository
push(repo_1)

## Pull changes to repo_2
pull(repo_2)

## Change file again and commit. This time in repository 2
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad",
  "minim veniam, quis nostrud exercitation ullamco laboris nisi ut")
writeLines(lines, file.path(path_repo_2, "example.txt"))
add(repo_2, "example.txt")
commit(repo_2, "Third commit message")
```

```
## Push commits from second repository to bare repository
push(repo_2)

## Pull changes to repo_1
pull(repo_1)

## List commits in repositories
commits(repo_1)
commits(repo_2)
commits(repo_bare)

## End(Not run)
```

---

punch_card *Punch card*

Description

Punch card

Usage

```
punch_card(repo = ".", main = NULL, ...)
```

Arguments

repo	a path to a repository or a git_repository object. Default is '.'
main	Default title for the plot is "Punch card on repo:" and repository workdir basename. Supply a new title if you desire one.
...	Additional arguments affecting the plot

Value

invisible NULL

Examples

```
## Not run:
## Initialize repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- clone("https://github.com/ropensci/git2r.git", path)

## Plot
punch_card(repo)

## End(Not run)
```

push *Push*

Description

Push

Usage

```
push(
  object = ".",
  name = NULL,
  refspec = NULL,
  force = FALSE,
  credentials = NULL,
  set_upstream = FALSE
)
```

Arguments

object	path to repository, or a git_repository or git_branch.
name	The remote's name. Default is NULL.
refspec	The refspec to be pushed. Default is NULL.
force	Force your local revision to the remote repo. Use it with care. Default is FALSE.
credentials	The credentials for remote repository access. Default is NULL. To use and query an ssh-agent for the ssh key credentials, let this parameter be NULL (the default).
set_upstream	Set the current local branch to track the remote branch. Default is FALSE.

Value

invisible(NULL)

See Also

cred_user_pass, cred_ssh_key

Examples

```
## Not run:
## Initialize two temporary repositories
path_bare <- tempfile(pattern="git2r-")
path_repo <- tempfile(pattern="git2r-")
dir.create(path_bare)
dir.create(path_repo)
repo_bare <- init(path_bare, bare = TRUE)
```

```
## Clone the bare repository. This creates remote-tracking
## branches for each branch in the cloned repository.
repo <- clone(path_bare, path_repo)

## Config user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path_repo, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Push commits from repository to bare repository
push(repo, "origin", "refs/heads/master")

## Now, unset the remote-tracking branch to NULL to demonstrate
## the 'set_upstream' argument. Then push with 'set_upstream = TRUE'
## to add the upstream tracking branch to branch 'master' again.
branch_get_upstream(repository_head(repo))
branch_set_upstream(repository_head(repo), NULL)
branch_get_upstream(repository_head(repo))
push(repo, "origin", "refs/heads/master", set_upstream = TRUE)
branch_get_upstream(repository_head(repo))

## Change file and commit
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path_repo, "example.txt"))
add(repo, "example.txt")
commit(repo, "Second commit message")

## Push commits from repository to bare repository
push(repo)

## List commits in repository and bare repository
commits(repo)
commits(repo_bare)

## End(Not run)
```

`references` *Get all references that can be found in a repository.*

Description

Get all references that can be found in a repository.

Usage

```
references(repo = ".")
```

Arguments

repo	a path to a repository or a `git_repository` object. Default is '.'

Value

Character vector with references

Examples

```
## Not run:
## Initialize two temporary repositories
path_bare <- tempfile(pattern="git2r-")
path_repo <- tempfile(pattern="git2r-")
dir.create(path_bare)
dir.create(path_repo)
repo_bare <- init(path_bare, bare = TRUE)
repo <- clone(path_bare, path_repo)

## Config user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path_repo, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Push commits from repository to bare repository
## Adds an upstream tracking branch to branch 'master'
push(repo, "origin", "refs/heads/master")

## Add tag to HEAD
tag(repo, "v1.0", "First version")

## Create a note
note_create(commits(repo)[[1]], "My note")

## List all references in repository
references(repo)

## End(Not run)
```

reflog *List and view reflog information*

Description

List and view reflog information

Usage

```
reflog(repo = ".", refname = "HEAD")
```

Arguments

repo	a path to a repository or a git_repository object. Default is '.'
refname	The name of the reference to list. 'HEAD' by default.

Value

S3 class git_reflog with git_reflog_entry objects.

Examples

```
## Not run:
## Initialize a repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Config user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Write to a file and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Change file and commit
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "Second commit message")

## Change file again and commit
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad",
```

```
  "minim veniam, quis nostrud exercitation ullamco laboris nisi ut")
writeLines(lines, file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "Third commit message")

## View reflog
reflog(repo)

## End(Not run)
```

`remotes` *Get the configured remotes for a repo*

Description

Get the configured remotes for a repo

Usage

```
remotes(repo = ".")
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'

Value

Character vector with remotes

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name="Alice", user.email="alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Add a remote
remote_add(repo, "playground", "https://example.org/git2r/playground")
remotes(repo)
remote_url(repo, "playground")

## Rename a remote
remote_rename(repo, "playground", "foobar")
```

```
remotes(repo)
remote_url(repo, "foobar")

## Set remote url
remote_set_url(repo, "foobar", "https://example.org/git2r/foobar")
remotes(repo)
remote_url(repo, "foobar")

## Remove a remote
remote_remove(repo, "foobar")
remotes(repo)

## End(Not run)
```

`remote_add` *Add a remote to a repo*

Description

Add a remote to a repo

Usage

```
remote_add(repo = ".", name = NULL, url = NULL)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`name`	Short name of the remote repository
`url`	URL of the remote repository

Value

NULL, invisibly

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name="Alice", user.email="alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")
```

```
## Add a remote
remote_add(repo, "playground", "https://example.org/git2r/playground")
remotes(repo)
remote_url(repo, "playground")

## Rename a remote
remote_rename(repo, "playground", "foobar")
remotes(repo)
remote_url(repo, "foobar")

## Set remote url
remote_set_url(repo, "foobar", "https://example.org/git2r/foobar")
remotes(repo)
remote_url(repo, "foobar")

## Remove a remote
remote_remove(repo, "foobar")
remotes(repo)

## End(Not run)
```

`remote_ls` *List references in a remote repository*

Description

Displays references available in a remote repository along with the associated commit IDs. Akin to the 'git ls-remote' command.

Usage

```
remote_ls(name = NULL, repo = NULL, credentials = NULL)
```

Arguments

`name`	Character vector with the "remote" repository URL to query or the name of the remote if a `repo` argument is given.
`repo`	an optional repository object used if remotes are specified by name.
`credentials`	The credentials for remote repository access. Default is NULL. To use and query an ssh-agent for the ssh key credentials, let this parameter be NULL (the default).

Value

Character vector for each reference with the associated commit IDs.

Examples

```
## Not run:
remote_ls("https://github.com/ropensci/git2r")

## End(Not run)
```

`remote_remove` *Remove a remote*

Description

All remote-tracking branches and configuration settings for the remote will be removed.

Usage

```
remote_remove(repo = ".", name = NULL)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`name`	The name of the remote to remove

Value

NULL, invisibly

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name="Alice", user.email="alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Add a remote
remote_add(repo, "playground", "https://example.org/git2r/playground")
remotes(repo)
remote_url(repo, "playground")

## Rename a remote
remote_rename(repo, "playground", "foobar")
remotes(repo)
remote_url(repo, "foobar")
```

```
## Set remote url
remote_set_url(repo, "foobar", "https://example.org/git2r/foobar")
remotes(repo)
remote_url(repo, "foobar")

## Remove a remote
remote_remove(repo, "foobar")
remotes(repo)

## End(Not run)
```

`remote_rename` *Rename a remote*

Description

Rename a remote

Usage

```
remote_rename(repo = ".", oldname = NULL, newname = NULL)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`oldname`	Old name of the remote
`newname`	New name of the remote

Value

NULL, invisibly

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name="Alice", user.email="alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Add a remote
remote_add(repo, "playground", "https://example.org/git2r/playground")
```

```
remotes(repo)
remote_url(repo, "playground")

## Rename a remote
remote_rename(repo, "playground", "foobar")
remotes(repo)
remote_url(repo, "foobar")

## Set remote url
remote_set_url(repo, "foobar", "https://example.org/git2r/foobar")
remotes(repo)
remote_url(repo, "foobar")

## Remove a remote
remote_remove(repo, "foobar")
remotes(repo)

## End(Not run)
```

`remote_set_url` *Set the remote's url in the configuration*

Description

This assumes the common case of a single-url remote and will otherwise raise an error.

Usage

```
remote_set_url(repo = ".", name = NULL, url = NULL)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`name`	The name of the remote
`url`	The `url` to set

Value

NULL, invisibly

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
```

```
config(repo, user.name="Alice", user.email="alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Add a remote
remote_add(repo, "playground", "https://example.org/git2r/playground")
remotes(repo)
remote_url(repo, "playground")

## Rename a remote
remote_rename(repo, "playground", "foobar")
remotes(repo)
remote_url(repo, "foobar")

## Set remote url
remote_set_url(repo, "foobar", "https://example.org/git2r/foobar")
remotes(repo)
remote_url(repo, "foobar")

## Remove a remote
remote_remove(repo, "foobar")
remotes(repo)

## End(Not run)
```

`remote_url` *Get the remote url for remotes in a repo*

Description

Get the remote url for remotes in a repo

Usage

```
remote_url(repo = ".", remote = NULL)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`remote`	Character vector with the remotes to get the url from. Default is the remotes of the repository.

Value

Character vector with remote_url for each of the remote

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name="Alice", user.email="alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Add a remote
remote_add(repo, "playground", "https://example.org/git2r/playground")
remotes(repo)
remote_url(repo, "playground")

## Rename a remote
remote_rename(repo, "playground", "foobar")
remotes(repo)
remote_url(repo, "foobar")

## Set remote url
remote_set_url(repo, "foobar", "https://example.org/git2r/foobar")
remotes(repo)
remote_url(repo, "foobar")

## Remove a remote
remote_remove(repo, "foobar")
remotes(repo)

## End(Not run)
```

`repository` *Open a repository*

Description

Open a repository

Usage

```
repository(path = ".", discover = TRUE)
```

Arguments

`path`	A path to an existing local git repository.
`discover`	Discover repository from path. Default is TRUE.

Value

A `git_repository` object with entries:

path Path to a git repository

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

# Configure a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
writeLines("Hello world!", file.path(path, "test-1.txt"))
add(repo, 'test-1.txt')
commit_1 <- commit(repo, "Commit message")

## Make one more commit
writeLines(c("Hello world!", "HELLO WORLD!"),
           file.path(path, "test-1.txt"))
add(repo, 'test-1.txt')
commit(repo, "Next commit message")

## Create one more file
writeLines("Hello world!",
           file.path(path, "test-2.txt"))

## Brief summary of repository
repo

## Summary of repository
summary(repo)

## Workdir of repository
workdir(repo)

## Check if repository is bare
is_bare(repo)

## Check if repository is empty
is_empty(repo)

## Check if repository is a shallow clone
is_shallow(repo)

## List all references in repository
references(repo)
```

```
## List all branches in repository
branches(repo)

## Get HEAD of repository
repository_head(repo)

## Check if HEAD is head
is_head(repository_head(repo))

## Check if HEAD is local
is_local(repository_head(repo))

## List all tags in repository
tags(repo)

## End(Not run)
```

`repository_head` *Get HEAD for a repository*

Description

Get HEAD for a repository

Usage

```
repository_head(repo = ".")
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'

Value

NULL if unborn branch or not found. A git_branch if not a detached head. A git_commit if detached head

Examples

```
## Not run:
## Create and initialize a repository in a temporary directory
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "Commit message")
```

```
## Get HEAD of repository
repository_head(repo)

## End(Not run)
```

`reset` *Reset current HEAD to the specified state*

Description

Reset current HEAD to the specified state

Usage

```
reset(object, reset_type = c("soft", "mixed", "hard"), path = NULL)
```

Arguments

`object`	Either a `git_commit`, a `git_repository` or a character vector. If `object` is a `git_commit`, HEAD is moved to the `git_commit`. If `object` is a `git_repository`, resets the index entries in the `path` argument to their state at HEAD. If `object` is a character vector with paths, resets the index entries in `object` to their state at HEAD if the current working directory is in a repository.
`reset_type`	If object is a 'git_commit', the kind of reset operation to perform. 'soft' means the HEAD will be moved to the commit. 'mixed' reset will trigger a 'soft' reset, plus the index will be replaced with the content of the commit tree. 'hard' reset will trigger a 'mixed' reset and the working directory will be replaced with the content of the index.
`path`	If object is a 'git_repository', resets the index entries for all paths to their state at HEAD.

Value

invisible NULL

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

# Configure a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
```

```
writeLines("Hello world!", file.path(path, "test-1.txt"))
add(repo, "test-1.txt")
commit_1 <- commit(repo, "Commit message")

## Change and stage the file
writeLines(c("Hello world!", "HELLO WORLD!"), file.path(path, "test-1.txt"))
add(repo, "test-1.txt")
status(repo)

## Unstage file
reset(repo, path = "test-1.txt")
status(repo)

## Make one more commit
add(repo, "test-1.txt")
commit(repo, "Next commit message")

## Create one more file
writeLines("Hello world!", file.path(path, "test-2.txt"))

## 'soft' reset to first commit and check status
reset(commit_1)
status(repo)

## 'mixed' reset to first commit and check status
commit(repo, "Next commit message")
reset(commit_1, "mixed")
status(repo)

## 'hard' reset to first commit and check status
add(repo, "test-1.txt")
commit(repo, "Next commit message")
reset(commit_1, "hard")
status(repo)

## End(Not run)
```

`revparse_single` *Revparse*

Description

Find object specified by revision.

Usage

```
revparse_single(repo = ".", revision = NULL)
```

Arguments

repo	a path to a repository or a git_repository object. Default is '.'
revision	The revision string, see http://git-scm.com/docs/git-rev-parse.html#_specifying_revisions

Value

a git_commit or git_tag or git_tree object

Examples

```
## Not run:
## Create a directory in tempdir
path <- tempfile(pattern="git2r-")
dir.create(path)

## Initialize a repository
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "test.txt"))
add(repo, "test.txt")
commit(repo, "First commit message")

# Change file and commit
lines <- c(
  "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do",
  "eiusmod tempor incididunt ut labore et dolore magna aliqua.")
writeLines(lines, file.path(path, "test.txt"))
add(repo, "test.txt")
commit(repo, "Second commit message")

revparse_single(repo, "HEAD^")
revparse_single(repo, "HEAD:test.txt")

## End(Not run)
```

---

rm_file *Remove files from the working tree and from the index*

Description

Remove files from the working tree and from the index

Usage

```
rm_file(repo = ".", path = NULL)
```

Arguments

repo	a path to a repository or a git_repository object. Default is '.'
path	character vector with filenames to remove. Only files known to Git are removed.

Value

invisible(NULL)

Examples

```
## Not run:
## Initialize a repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file
writeLines("Hello world!", file.path(path, "file-to-remove.txt"))

## Add file to repository
add(repo, "file-to-remove.txt")
commit(repo, "First commit message")

## Remove file
rm_file(repo, "file-to-remove.txt")

## View status of repository
status(repo)

## End(Not run)
```

---

sha *Get the SHA-1 of a git object*

Description

Get the 40 character hexadecimal string of the SHA-1.

Usage

```
sha(object)

## S3 method for class 'git_blob'
sha(object)
```

```
## S3 method for class 'git_branch'
sha(object)

## S3 method for class 'git_commit'
sha(object)

## S3 method for class 'git_note'
sha(object)

## S3 method for class 'git_reference'
sha(object)

## S3 method for class 'git_reflog_entry'
sha(object)

## S3 method for class 'git_tag'
sha(object)

## S3 method for class 'git_tree'
sha(object)

## S3 method for class 'git_fetch_head'
sha(object)

## S3 method for class 'git_merge_result'
sha(object)
```

Arguments

object	a git object to get the SHA-1 from.

Value

The 40 character hexadecimal string of the SHA-1.

Examples

```
## Not run:
## Create a directory in tempdir
path <- tempfile(pattern="git2r-")
dir.create(path)

## Initialize a repository
repo <- init(path)
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file, add and commit
lines <- "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do"
writeLines(lines, file.path(path, "test.txt"))
add(repo, "test.txt")
```

```
commit(repo, "Commit message 1")

## Get the SHA-1 of the last commit
sha(last_commit(repo))

## End(Not run)
```

---

ssh_path *Compose usual path to ssh keys*

Description

This function provides a consistent means across OS-types to access the .ssh directory.

Usage

```
ssh_path(file = "")
```

Arguments

file	basename of file for which path is requested

Details

On Windows-based systems, path.expand("~") returns "C:/Users/username/Documents", whereas the usual path to the .ssh directory is "C:/Users/username".

On other operating systems, path.expand("~") returns the usual path to the .ssh directory.

Calling ssh_path() with no arguments will return the usual path to the .ssh directory.

Value

Full path to the file

Examples

```
ssh_path()
ssh_path("is_rsa.pub")
```

ssl_cert_locations *Set the SSL certificate-authority locations*

Description

Set the SSL certificate-authority locations

Usage

```
ssl_cert_locations(filename = NULL, path = NULL)
```

Arguments

filename	Location of a file containing several certificates concatenated together. Default NULL.
path	Location of a directory holding several certificates, one per file. Default NULL.

Value

invisible(NULL)

Note

Either parameter may be 'NULL', but not both.

stash *Stash*

Description

Stash

Usage

```
stash(
  repo = ".",
  message = as.character(Sys.time()),
  index = FALSE,
  untracked = FALSE,
  ignored = FALSE,
  stasher = NULL
)
```

Arguments

repo	a path to a repository or a git_repository object. Default is '.'
message	Optional description. Defaults to current time.
index	All changes already added to the index are left intact in the working directory. Default is FALSE
untracked	All untracked files are also stashed and then cleaned up from the working directory. Default is FALSE
ignored	All ignored files are also stashed and then cleaned up from the working directory. Default is FALSE
stasher	Signature with stasher and time of stash

Value

invisible git_stash object if anything to stash else NULL

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

# Configure a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

# Create a file, add and commit
writeLines("Hello world!", file.path(path, "test.txt"))
add(repo, 'test.txt')
commit(repo, "Commit message")

# Change file
writeLines(c("Hello world!", "HELLO WORLD!"), file.path(path, "test.txt"))

# Check status of repository
status(repo)

# Create stash in repository
stash(repo)

# Check status of repository
status(repo)

# View stash
stash_list(repo)

## End(Not run)
```

stash_apply *Apply stash*

Description

Apply a single stashed state from the stash list.

Usage

```
stash_apply(object = ".", index = 1)
```

Arguments

object	path to a repository, or a git_repository object, or the stash object to pop. Default is a path = '.' to a reposiory.
index	The index to the stash to apply. Only used when object is a path to a repository or a git_repository object. Default is index = 1.

Details

If local changes in the working directory conflict with changes in the stash then an error will be raised. In this case, the index will always remain unmodified and all files in the working directory will remain unmodified. However, if you are restoring untracked files or ignored files and there is a conflict when applying the modified files, then those files will remain in the working directory.

Value

invisible NULL

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

# Configure a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

# Create a file, add and commit
writeLines("Hello world!", file.path(path, "test.txt"))
add(repo, 'test.txt')
commit(repo, "Commit message")

# Change file
writeLines(c("Hello world!", "HELLO WORLD!"), file.path(path, "test.txt"))

# Create stash in repository
```

```
stash(repo)

# Change file
writeLines(c("Hello world!", "HeLlO wOrLd!"), file.path(path, "test.txt"))

# Create stash in repository
stash(repo)

# View stashes
stash_list(repo)

# Read file
readLines(file.path(path, "test.txt"))

# Apply latest git_stash object in repository
stash_apply(stash_list(repo)[[1]])

# Read file
readLines(file.path(path, "test.txt"))

# View stashes
stash_list(repo)

## End(Not run)
```

`stash_drop` *Drop stash*

Description

Drop stash

Usage

```
stash_drop(object = ".", index = 1)
```

Arguments

`object`	path to a repository, or a `git_repository` object, or the stash `object` to drop. Default is a `path = '.'` to a reposiory.
`index`	The index to the stash to drop. Only used when `object` is a path to a repository or a `git_repository` object. Default is `index = 1`.

Value

invisible NULL

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

# Configure a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

# Create a file, add and commit
writeLines("Hello world!", file.path(path, "test.txt"))
add(repo, 'test.txt')
commit(repo, "Commit message")

# Change file
writeLines(c("Hello world!", "HELLO WORLD!"), file.path(path, "test.txt"))

# Create stash in repository
stash(repo)

# Change file
writeLines(c("Hello world!", "HeLlO wOrLd!"), file.path(path, "test.txt"))

# Create stash in repository
stash(repo)

# View stashes
stash_list(repo)

# Drop git_stash object in repository
stash_drop(stash_list(repo)[[1]])

## Drop stash using an index to stash
stash_drop(repo, 1)

# View stashes
stash_list(repo)

## End(Not run)
```

`stash_list` *List stashes in repository*

Description

List stashes in repository

Usage

```
stash_list(repo = ".")
```

Arguments

repo	a path to a repository or a git_repository object. Default is '.'

Value

list of stashes in repository

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

# Configure a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

# Create a file, add and commit
writeLines("Hello world!", file.path(path, "test-1.txt"))
add(repo, 'test-1.txt')
commit(repo, "Commit message")

# Make one more commit
writeLines(c("Hello world!", "HELLO WORLD!"), file.path(path, "test-1.txt"))
add(repo, 'test-1.txt')
commit(repo, "Next commit message")

# Create one more file
writeLines("Hello world!", file.path(path, "test-2.txt"))

# Check that there are no stashes
stash_list(repo)

# Stash
stash(repo)

# Only untracked changes, therefore no stashes
stash_list(repo)

# Stash and include untracked changes
stash(repo, "Stash message", untracked=TRUE)

# View stash
stash_list(repo)

## End(Not run)
```

stash_pop *Pop stash*

Description

Apply a single stashed state from the stash list and remove it from the list if successful.

Usage

```
stash_pop(object = ".", index = 1)
```

Arguments

object	path to a repository, or a git_repository object, or the stash object to pop. Default is a path = '.' to a reposiory.
index	The index to the stash to pop. Only used when object is a path to a repository or a git_repository object. Default is index = 1.

Value

invisible NULL

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

# Configure a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

# Create a file, add and commit
writeLines("Hello world!", file.path(path, "test.txt"))
add(repo, 'test.txt')
commit(repo, "Commit message")

# Change file
writeLines(c("Hello world!", "HELLO WORLD!"), file.path(path, "test.txt"))

# Create stash in repository
stash(repo)

# Change file
writeLines(c("Hello world!", "HeLlO wOrLd!"), file.path(path, "test.txt"))

# Create stash in repository
stash(repo)
```

```
# View stashes
stash_list(repo)

# Read file
readLines(file.path(path, "test.txt"))

# Pop latest git_stash object in repository
stash_pop(stash_list(repo)[[1]])

# Read file
readLines(file.path(path, "test.txt"))

# View stashes
stash_list(repo)

## End(Not run)
```

`status` *Status*

Description

Display state of the repository working directory and the staging area.

Usage

```
status(
  repo = ".",
  staged = TRUE,
  unstaged = TRUE,
  untracked = TRUE,
  ignored = FALSE,
  all_untracked = FALSE
)
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'
`staged`	Include staged files. Default TRUE.
`unstaged`	Include unstaged files. Default TRUE.
`untracked`	Include untracked files and directories. Default TRUE.
`ignored`	Include ignored files. Default FALSE.
`all_untracked`	Shows individual files in untracked directories if `untracked` is `TRUE`.

Value

`git_status` with repository status

Examples

```
## Not run:
## Initialize a repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Config user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file
writeLines("Hello world!", file.path(path, "test.txt"))

## Check status; untracked file
status(repo)

## Add file
add(repo, "test.txt")

## Check status; staged file
status(repo)

## Commit
commit(repo, "First commit message")

## Check status; clean
status(repo)

## Change the file
writeLines(c("Hello again!", "Here is a second line", "And a third"),
           file.path(path, "test.txt"))

## Check status; unstaged file
status(repo)

## Add file and commit
add(repo, "test.txt")
commit(repo, "Second commit message")

## Check status; clean
status(repo)

## End(Not run)
```

`summary.git_repository` *Summary of repository*

Description

Summary of repository

Usage

```
## S3 method for class 'git_repository'
summary(object, ...)
```

Arguments

object	The repository object
...	Additional arguments affecting the summary produced.

Value

None (invisible 'NULL').

Examples

```
## Not run:
## Initialize a repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Config user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Create a file
writeLines("Hello world!", file.path(path, "test.txt"))
summary(repo)

## Add file
add(repo, "test.txt")
summary(repo)

## Commit
commit(repo, "First commit message")
summary(repo)

## Change the file
writeLines(c("Hello again!", "Here is a second line", "And a third"),
           file.path(path, "test.txt"))
summary(repo)

## Add file and commit
add(repo, "test.txt")
commit(repo, "Second commit message")
summary(repo)

## End(Not run)
```

summary.git_stash *Summary of a stash*

Description

Summary of a stash

Usage

```
## S3 method for class 'git_stash'
summary(object, ...)
```

Arguments

object	The stash object
...	Additional arguments affecting the summary produced.

Value

None (invisible 'NULL').

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

# Configure a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

# Create a file, add and commit
writeLines("Hello world!", file.path(path, "test.txt"))
add(repo, 'test.txt')
commit(repo, "Commit message")

# Change file
writeLines(c("Hello world!", "HELLO WORLD!"), file.path(path, "test.txt"))

# Create stash in repository
stash(repo, "Stash message")

# View summary of stash
summary(stash_list(repo)[[1]])

## End(Not run)
```

summary.git_tree *Summary of tree*

Description

Summary of tree

Usage

```
## S3 method for class 'git_tree'
summary(object, ...)
```

Arguments

object	The tree object
...	Additional arguments affecting the summary produced.

Value

None (invisible 'NULL').

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

summary(tree(last_commit(repo)))

## End(Not run)
```

tag *Create tag targeting HEAD commit in repository*

Description

Create tag targeting HEAD commit in repository

Usage

```
tag(
  object = ".",
  name = NULL,
  message = NULL,
  session = FALSE,
  tagger = NULL,
  force = FALSE
)
```

Arguments

object	The repository object.
name	Name for the tag.
message	The tag message. Specify a tag message to create an annotated tag. A lightweight tag is created if the message parameter is NULL.
session	Add sessionInfo to tag message. Default is FALSE.
tagger	The tagger (author) of the tag
force	Overwrite existing tag. Default = FALSE

Value

invisible(git_tag) object

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Commit a text file
filename <- file.path(path, "example.txt")
writeLines("Hello world!", filename)
add(repo, "example.txt")
```

```
commit(repo, "First commit message")

## Create an annotated tag
tag(repo, "v1.0", "Tag message")

## List tags
tags(repo)

## Make a change to the text file and commit.
writeLines(c("Hello world!", "HELLO WORLD!"), filename)
add(repo, "example.txt")
commit(repo, "Second commit message")

## Create a lightweight tag
tag(repo, "v2.0")

## List tags
tags(repo)

## End(Not run)
```

`tags` *Tags*

Description

Tags

Usage

```
tags(repo = ".")
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'

Value

list of tags in repository

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")
```

```
## Commit a text file
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Create tag
tag(repo, "Tagname", "Tag message")

## List tags
tags(repo)

## End(Not run)
```

`tag_delete` *Delete an existing tag reference*

Description

Delete an existing tag reference

Usage

```
tag_delete(object = ".", name = NULL)
```

Arguments

`object`	Can be either the path (default is ".") to a repository, or a `git_repository` object, or a `git_tag` object. or the tag name.
`name`	If the `object` argument is a path to a repository or a `git_repository`, the name of the tag to delete.

Value

```
invisible(NULL)
```

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

## Commit a text file
writeLines("Hello world!", file.path(path, "example.txt"))
```

```
add(repo, "example.txt")
commit(repo, "First commit message")

## Create two tags
tag(repo, "Tag1", "Tag message 1")
t2 <- tag(repo, "Tag2", "Tag message 2")

## List the two tags in the repository
tags(repo)

## Delete the two tags in the repository
tag_delete(repo, "Tag1")
tag_delete(t2)

## Show the empty list with tags in the repository
tags(repo)

## End(Not run)
```

`tree` *Tree*

Description

Get the tree pointed to by a commit or stash.

Usage

```
tree(object = NULL)
```

Arguments

`object`	the `commit` or `stash` object

Value

A S3 class git_tree object

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a first user and commit a file
config(repo, user.name = "Alice", user.email = "alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
```

```
commit(repo, "First commit message")

tree(last_commit(repo))

## End(Not run)
```

---

when *When*

Description

Help method to extract the time as a character string from a git_commit, git_signature, git_tag and git_time object.

Usage

```
when(object, tz = "GMT", origin = "1970-01-01", usetz = TRUE)
```

Arguments

object	the object to extract the time slot from.
tz	a character string. The time zone specification to be used for the conversion, *if one is required*. System-specific (see time zones), but "" is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.
origin	a date-time object, or something which can be coerced by as.POSIXct(tz = "GMT") to such an object. Optional since R 4.3.0, where the equivalent of "1970-01-01" is used.
usetz	logical. Should the time zone abbreviation be appended to the output? This is used in printing times, and more reliable than using "%Z".

Value

A character vector of length one.

See Also

git_time

Examples

```
## Not run:
## Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
repo <- init(path)

## Create a first user and commit a file
```

```
config(repo, user.name = "Alice", user.email = "alice@example.org")
writeLines("Hello world!", file.path(path, "example.txt"))
add(repo, "example.txt")
commit(repo, "First commit message")

## Create tag
tag(repo, "Tagname", "Tag message")

when(commits(repo)[[1]])
when(tags(repo)[[1]])
when(tags(repo)[[1]], tz = Sys.timezone())

## End(Not run)
```

`workdir` *Workdir of repository*

Description

Workdir of repository

Usage

```
workdir(repo = ".")
```

Arguments

`repo`	a path to a repository or a `git_repository` object. Default is '.'

Value

Character vector with the path of the workdir. If the repository is bare, `NULL` will be returned.

Examples

```
## Not run:
## Create a directory in tempdir
path <- tempfile(pattern="git2r-")
dir.create(path)

## Initialize a repository
repo <- init(path)

## Get the path of the workdir for repository
workdir(repo)

## End(Not run)
```

`[.git_tree` *Extract object from tree*

Description

Lookup a tree entry by its position in the tree

Usage

```
## S3 method for class 'git_tree'
x[i]
```

Arguments

`x`	The tree `object`
`i`	The index (integer or logical) of the tree object to extract. If negative values, all elements except those indicated are selected. A character vector to match against the names of objects to extract.

Value

Git object

Examples

```
## Not run:
##' Initialize a temporary repository
path <- tempfile(pattern="git2r-")
dir.create(path)
dir.create(file.path(path, "subfolder"))
repo <- init(path)

##' Create a user
config(repo, user.name = "Alice", user.email = "alice@example.org")

##' Create three files and commit
writeLines("First file",  file.path(path, "example-1.txt"))
writeLines("Second file", file.path(path, "subfolder/example-2.txt"))
writeLines("Third file",  file.path(path, "example-3.txt"))
add(repo, c("example-1.txt", "subfolder/example-2.txt", "example-3.txt"))
new_commit <- commit(repo, "Commit message")

##' Pick a tree in the repository
tree_object <- tree(new_commit)

##' Display tree
tree_object

##' Select item by name
```

```
tree_object["example-1.txt"]

##' Select first item in tree
tree_object[1]

##' Select first three items in tree
tree_object[1:3]

##' Select all blobs in tree
tree_object[vapply(as(tree_object, 'list'), is_blob, logical(1))]

## End(Not run)
```

Index

∗ **git credential functions**
  cred_env, 34
  cred_ssh_key, 35
  cred_token, 36
  cred_user_pass, 37
∗ **methods**
  is_tree, 67
  libgit2_features, 71
  libgit2_version, 71
  ssl_cert_locations, 112
[.git_tree, 130

add, 4
ahead_behind, 6
as.character.git_time *(*git_time*)*, 48
as.data.frame.git_repository, 7
as.data.frame.git_tree, 8
as.list.git_tree, 9
as.POSIXct.git_time *(*git_time*)*, 48

blame, 10
blob_create, 12
branch_create, 14
branch_delete, 15
branch_get_upstream, 16
branch_remote_name, 17
branch_remote_url, 18
branch_rename, 19
branch_set_upstream, 20
branch_target, 21
branches, 13
bundle_r_package, 22

checkout, 23
clone, 25
commit, 26
commits, 28
config, 30
content, 31
contributions, 32

cred_env, 34, *35–37*
cred_ssh_key, *25*, *34*, 35, *36*, *37*, *92*
cred_token, *34*, *35*, 36, *37*
cred_user_pass, *25*, *34–36*, 37, *92*

default_signature, 38
descendant_of, 39
diff.git_repository, 40
diff.git_tree *(*diff.git_repository*)*, 40
discover_repository, *12*, 43

fetch, 44
fetch_heads, 46
format.git_time *(*git_time*)*, 48

git2r, 47
git2r-package *(*git2r*)*, 47
git_config_files, 48
git_time, 48, *128*

hash, 50
hashfile, 50
head.git_repository, 51

in_repository, 54
index_remove_bypath, 52
init, 53, *55*
is_bare, 55
is_binary, 56
is_blob, 57
is_branch, 58
is_commit, 59
is_detached, 60
is_empty, 61
is_head, 62
is_local, 63
is_merge, 64
is_shallow, 65
is_tag, 66
is_tree, 67

last_commit, 68
length.git_blob, 69
length.git_diff, 70
length.git_tree, 70
libgit2_features, 71
libgit2_version, 71
lookup, 72
lookup_commit, 73
ls_tree, 74

merge.character *(*merge.git_branch*)*, 76
merge.git_branch, 76
merge.git_repository *(*merge.git_branch*)*, 76
merge_base, 78

note_create, 80
note_default_ref, 82
note_remove, 83
notes, 79

odb_blobs, 84
odb_objects, 85

parents, 86
plot.git_repository, 87
print.git_reflog_entry, 88
print.git_time *(*git_time*)*, 48
pull, 89
punch_card, 91
push, 92

references, 93
reflog, 95
remote_add, 97
remote_ls, 98
remote_remove, 99
remote_rename, 100
remote_set_url, 101
remote_url, 102
remotes, 96
repository, *25*, *53*, 103
repository_head, 105
reset, 106
revparse_single, 107
rm_file, 108

sha, 109
ssh_path, 111
ssl_cert_locations, 112
Startup, *34*, *36*
stash, 112
stash_apply, 114
stash_drop, 115
stash_list, 116
stash_pop, 118
status, 119
summary.git_repository, 120
summary.git_stash, 122
summary.git_tree, 123

tag, 124
tag_delete, 126
tags, 125
time zones, *49*, *128*
tree, 127

when, *49*, 128
workdir, 129