git-submodule(1) Manual Page

NAME

git-submodule - Initialize, update or inspect submodules

SYNOPSIS

git submodule [--quiet] [--cached]
git submodule [--quiet] add [<options>] [--] <repository> [<path>]
git submodule [--quiet] status [--cached] [--recursive] [--] [<path>...]
git submodule [--quiet] init [--] [<path>...]
git submodule [--quiet] deinit [-f|--force] (--all|[--] <path>...)
git submodule [--quiet] update [<options>] [--] [<path>...]
git submodule [--quiet] set-branch [<options>] [--] <path>
git submodule [--quiet] set-url [--] <path> <newurl>
git submodule [--quiet] summary [<options>] [--] [<path>...]
git submodule [--quiet] foreach [--recursive] <command>
git submodule [--quiet] sync [--recursive] [--] [<path>...]
git submodule [--quiet] absorbgitdirs [--] [<path>...]

DESCRIPTION

Inspects, updates and manages submodules.

For more information about submodules, see gitsubmodules(7).

COMMANDS

With no arguments, shows the status of existing submodules. Several subcommands are available to perform operations on the submodules.

add [-b <branch>] [-f|--force] [--name <name>] [--reference <repository>] [--depth <depth>] [--] <repository> [<path>]

Add the given repository as a submodule at the given path to the changeset to be committed next to the current project: the current project is termed the "superproject".

<repository> is the URL of the new submodule's origin repository. This may be either an absolute URL, or (if it begins with ./ or ../), the location relative to the superproject's default remote repository (Please note that to specify a repository *foo.git* which is located right next to a superproject *bar.git*, you'll have to use `../foo.git` instead of `./foo.git` - as one might expect when following the rules for relative URLs - because the evaluation of relative URLs in Git is identical to that of relative directories).

The default remote is the remote of the remote-tracking branch of the current branch. If no such remote-tracking branch exists or the HEAD is detached, "origin" is assumed to be the default remote. If the superproject doesn't have a default remote configured the superproject is its own authoritative upstream and the current working directory is used instead.

The optional argument <path> is the relative location for the cloned submodule to exist in the superproject. If <path> is not given, the canonical part of the source repository is used ("repo" for "/path/to/repo.git" and "foo" for "host.xz:foo/.git"). If <path> exists and is already a valid Git repository, then it is staged for commit without cloning. The <path> is also used as the submodule's logical name in its configuration entries unless `--name` is used to specify a logical name.

The given URL is recorded into `.gitmodules` for use by subsequent users cloning the superproject. If the URL is given relative to the superproject's repository, the presumption is the superproject and submodule repositories will be kept together in the same relative location, and only the superproject's URL needs to be provided. git-submodule will correctly locate the submodule using the relative URL in `.gitmodules`.

status [--cached] [--recursive] [--] [<path>...]

Show the status of the submodules. This will print the SHA-1 of the currently checked out commit for each submodule, along with the submodule path and the output of *git describe* for the SHA-1. Each SHA-1 will possibly be prefixed with `-` if the submodule is not initialized, `+` if the currently checked out submodule commit does not match the SHA-1 found in the index of the containing repository and `U` if the submodule has merge conflicts.

If `--cached` is specified, this command will instead print the SHA-1 recorded in the superproject for each submodule.

If `--recursive` is specified, this command will recurse into nested submodules, and show their status as well.

If you are only interested in changes of the currently initialized submodules with respect to the commit recorded in the index or the HEAD, git-status(1) and git-diff(1) will provide that information too (and can also report changes to a submodule's work tree).

init [--] [<path>...]

Initialize the submodules recorded in the index (which were added and committed elsewhere) by setting `submodule.$name.url` in .git/config. It uses the same setting from `.gitmodules` as a template. If the URL is relative, it will be resolved using the default remote. If there is no default remote, the current repository will be assumed to be upstream.

Optional <path> arguments limit which submodules will be initialized. If no path is specified and submodule.active has been configured, submodules configured to be active will be initialized, otherwise all submodules are initialized.

When present, it will also copy the value of `submodule.$name.update`. This command does not alter existing information in .git/config. You can then customize the submodule clone URLs in .git/config for your local setup and proceed to `git submodule update`; you can also just use `git submodule update --init` without the explicit *init* step if you do not intend to customize any submodule locations.

See the add subcommand for the definition of default remote.

deinit [-f|--force] (--all|[--] <path>...)

Unregister the given submodules, i.e. remove the whole `submodule.$name` section from .git/config together with their work tree. Further calls to `git submodule update`, `git submodule foreach` and `git submodule sync` will skip any unregistered submodules until they are initialized again, so use this command if you don't want to have a local checkout of the submodule in your working tree anymore.

When the command is run without pathspec, it errors out, instead of deinit-ing everything, to prevent mistakes.

If `--force` is specified, the submodule's working tree will be removed even if it contains local modifications.

If you really want to remove a submodule from the repository and commit that use git-rm(1) instead. See gitsubmodules(7) for removal options.

update [--init] [--remote] [-N|--no-fetch] [--[no-]recommend-shallow] [-f|--force] [--checkout|--rebase|--merge] [--reference <repository>] [--depth <depth>] [--recursive] [--jobs <n>] [--[no-]single-branch] [--] [<path>...]

Update the registered submodules to match what the superproject expects by cloning missing submodules, fetching missing commits in submodules and updating the working tree of the submodules. The "updating" can be done in several ways depending on command line options and the value of `submodule.<name>.update` configuration variable. The command line option takes precedence over the configuration variable. If neither is given, a *checkout* is performed. The *update* procedures supported both from the command line as well as through the `submodule.<name>.update` configuration are:

checkout

the commit recorded in the superproject will be checked out in the submodule on a detached HEAD.

If `--force` is specified, the submodule will be checked out (using `git checkout --force`), even if the commit specified in the index of the containing repository already matches the commit checked out in the submodule.

rebase

the current branch of the submodule will be rebased onto the commit recorded in the superproject.

merge

the commit recorded in the superproject will be merged into the current branch in the submodule.

The following *update* procedures are only available via the `submodule.<name>.update` configuration variable:

custom command

arbitrary shell command that takes a single argument (the sha1 of the commit recorded in the superproject) is executed. When `submodule.<name>.update` is set to *!command*, the remainder after the exclamation mark is the custom command.

none

the submodule is not updated.

If the submodule is not yet initialized, and you just want to use the setting as stored in `.gitmodules`, you can automatically initialize the submodule with the `--init` option.

If `--recursive` is specified, this command will recurse into the registered submodules, and update any nested submodules within.

set-branch (-b|--branch) <branch> [--] <path>

set-branch (-d|--default) [--] <path>

Sets the default remote tracking branch for the submodule. The `--branch` option allows the remote branch to be specified. The `--default` option removes the submodule.<name>.branch configuration key, which causes the tracking branch to default to the remote *HEAD*.

set-url [--] <path> <newurl>

Sets the URL of the specified submodule to <newurl>. Then, it will automatically synchronize the submodule's new remote URL configuration.

summary [--cached|--files] [(-n|--summary-limit) <n>] [commit] [--] [<path>...]

Show commit summary between the given commit (defaults to HEAD) and working tree/index. For a submodule in question, a series of commits in the submodule between the given super project commit and the index or working tree (switched by `--cached`) are shown. If the option `--files` is given, show the series of commits in the submodule between the index of the super project and the working tree of the submodule (this option doesn't allow to use the `--cached` option or to provide an explicit commit).

Using the `--submodule=log` option with git-diff(1) will provide that information too.

foreach [--recursive] <command>

Evaluates an arbitrary shell command in each checked out submodule. The command has access to the variables $name, $sm_path, $displaypath, $sha1 and $toplevel: $name is the name of the relevant submodule section in `.gitmodules`, $sm_path is the path of the submodule as recorded in the immediate superproject, $displaypath contains the relative path from the current working directory to the submodules root directory, $sha1 is the commit as recorded in the immediate superproject, and $toplevel is the absolute path to the top-level of the immediate superproject. Note that to avoid conflicts with *$PATH* on Windows, the *$path* variable is now a deprecated synonym of *$sm_path* variable. Any submodules defined in the superproject but not checked out are ignored by this command. Unless given `--quiet`, foreach prints the name of each submodule before evaluating the command. If `--recursive` is given, submodules are traversed recursively (i.e. the given shell command is evaluated in nested submodules as well). A non-zero return from the command in any submodule causes the processing to terminate. This can be overridden by adding `|| :` to the end of the command.

As an example, the command below will show the path and currently checked out commit for each submodule:

```
git submodule foreach 'echo $sm_path `git rev-parse HEAD`'
```

sync [--recursive] [--] [<path>...]

Synchronizes submodules' remote URL configuration setting to the value specified in `.gitmodules`. It will only affect those submodules which already have a URL entry in .git/config (that is the case when they are initialized or freshly added). This is useful when submodule URLs change upstream and you need to update your local repositories accordingly.

`git submodule sync` synchronizes all submodules while `git submodule sync -- A` synchronizes submodule "A" only.

If `--recursive` is specified, this command will recurse into the registered submodules, and sync any nested submodules within.

absorbgitdirs

If a git directory of a submodule is inside the submodule, move the git directory of the submodule into its superproject's `$GIT_DIR/modules` path and then connect the git directory and its working directory by setting the `core.worktree` and adding a .git file pointing to the git directory embedded in the superprojects git directory.

A repository that was cloned independently and later added as a submodule or old setups have the submodules git directory inside the submodule instead of embedded into the superprojects git directory.

This command is recursive by default.

OPTIONS

-q

--quiet

Only print error messages.

--progress

This option is only valid for add and update commands. Progress status is reported on the standard error stream by default when it is attached to a terminal, unless -q is specified. This flag forces progress status even if the standard error stream is not directed to a terminal.

--all

This option is only valid for the deinit command. Unregister all submodules in the working tree.

-b <branch>

--branch <branch>

Branch of repository to add as submodule. The name of the branch is recorded as `submodule.<name>.branch` in `.gitmodules` for `update --remote`. A special value of `.` is used to indicate that the name of the branch in the submodule should be the same name as the current branch in the current repository. If the option is not specified, it defaults to the remote *HEAD*.

-f

--force

This option is only valid for add, deinit and update commands. When running add, allow adding an otherwise ignored submodule path. When running deinit the submodule working trees will be removed even if they contain local changes. When running update (only effective with the checkout procedure), throw away local changes in submodules when switching to a different commit; and always run a checkout operation in the submodule, even if the commit listed in the index of the containing repository matches the commit checked out in the submodule.

--cached

This option is only valid for status and summary commands. These commands typically use the commit found in the submodule HEAD, but with this option, the commit stored in the index is used instead.

--files

This option is only valid for the summary command. This command compares the commit in the index with that in the submodule HEAD when this option is used.

-n

--summary-limit

This option is only valid for the summary command. Limit the summary size (number of commits shown in total). Giving 0 will disable the summary; a negative number means unlimited (the default). This limit only applies to modified submodules. The size is always limited to 1 for added/deleted/typechanged submodules.

--remote

This option is only valid for the update command. Instead of using the superproject's recorded SHA-1 to update the submodule, use the status of the submodule's remote-tracking branch. The remote used is branch's remote (`branch.<name>.remote`), defaulting to `origin`. The remote branch used defaults to the remote `HEAD`, but the branch name may be overridden by setting the `submodule.<name>.branch` option in either `.gitmodules` or `.git/config` (with `.git/config` taking precedence).

This works for any of the supported update procedures (`--checkout`, `--rebase`, etc.). The only change is the source of the target SHA-1. For example, `submodule update --remote --merge` will merge upstream submodule changes into the submodules, while `submodule update --merge` will merge superproject gitlink changes into the submodules.

In order to ensure a current tracking branch state, `update --remote` fetches the submodule's remote repository before calculating the SHA-1. If you don't want to fetch, you should use `submodule update --remote --no-fetch`.

Use this option to integrate changes from the upstream subproject with your submodule's current HEAD. Alternatively, you can run `git pull` from the submodule, which is equivalent except for the remote branch name: `update --remote` uses the default upstream repository and `submodule.<name>.branch`, while `git pull` uses the submodule's `branch.<name>.merge`. Prefer `submodule.<name>.branch` if you want to distribute the default upstream branch with the superproject and `branch.<name>.merge` if you want a more native feel while working in the submodule itself.

-N

--no-fetch

This option is only valid for the update command. Don't fetch new objects from the remote site.

--checkout

This option is only valid for the update command. Checkout the commit recorded in the superproject on a detached HEAD in the submodule. This is the default behavior, the main use of this option is to override `submodule.$name.update` when set to a value other than `checkout`. If the key `submodule.$name.update` is either not explicitly set or set to `checkout`, this option is implicit.

--merge

This option is only valid for the update command. Merge the commit recorded in the superproject into the current branch of the submodule. If this option is given, the submodule's HEAD will not be detached. If a merge failure prevents this process, you will have to resolve the resulting conflicts within the submodule with the usual conflict resolution tools. If the key `submodule.$name.update` is set to `merge`, this option is implicit.

--rebase

This option is only valid for the update command. Rebase the current branch onto the commit recorded in the superproject. If this option is given, the submodule's HEAD will not be detached. If a merge failure prevents this process, you will have to resolve these failures with git-rebase(1). If the key `submodule.$name.update` is set to `rebase`, this option is implicit.

--init

This option is only valid for the update command. Initialize all submodules for which "git submodule init" has not been called so far before updating.

--name

This option is only valid for the add command. It sets the submodule's name to the given string instead of defaulting to its path. The name must be valid as a directory name and may not end with a /.

--reference <repository>

This option is only valid for add and update commands. These commands sometimes need to clone a remote repository. In this case, this option will be passed to the git-clone(1) command.

NOTE: Do **not** use this option unless you have read the note for git-clone(1)'s `--reference`, `--shared`, and `--dissociate` options carefully.

--dissociate

This option is only valid for add and update commands. These commands sometimes need to clone a remote repository. In this case, this option will be passed to the git-clone(1) command.

NOTE: see the NOTE for the `--reference` option.

--recursive

This option is only valid for foreach, update, status and sync commands. Traverse submodules recursively. The operation is performed not only in the submodules of the current repo, but also in any nested submodules inside those submodules (and so on).

--depth

This option is valid for add and update commands. Create a *shallow* clone with a history truncated to the specified number of revisions. See git-clone(1)

--[no-]recommend-shallow

This option is only valid for the update command. The initial clone of a submodule will use the recommended `submodule.<name>.shallow` as provided by the `.gitmodules` file by default. To ignore the suggestions use `--no-recommend-shallow`.

-j <n>

--jobs <n>

This option is only valid for the update command. Clone new submodules in parallel with as many jobs. Defaults to the `submodule.fetchJobs` option.

--[no-]single-branch

This option is only valid for the update command. Clone only one branch during update: HEAD or one specified by --branch.

<path>...

Paths to submodule(s). When specified this will restrict the command to only operate on the submodules found at the specified paths. (This argument is required with add).

FILES

When initializing submodules, a `.gitmodules` file in the top-level directory of the containing repository is used to find the url of each submodule. This file should be formatted in the same way as `$GIT_DIR/config`. The key to each submodule url is "submodule.$name.url". See gitmodules(5) for details.

SEE ALSO

gitsubmodules(7), gitmodules(5).

GIT

Part of the git(1) suite

Last updated 2022-04-15 02:08:45 UTC