NAME

git-submodule - Initialize, update or inspect submodules

SYNOPSIS

git submodule [--quiet] add [-b <branch>] [-f|--force] [--name <name>]
[--reference <repository>] [--depth <depth>] [--] <repository> [<path>]
git submodule [--quiet] status [--cached] [--recursive] [--] [<path>...]
git submodule [--quiet] init [--] [<path>...]
git submodule [--quiet] deinit [-f|--force] [--] <path>...
git submodule [--quiet] update [--init] [--remote] [-N|--no-fetch]
[-f|--force] [--rebase|--merge] [--reference <repository>]
[--depth <depth>] [--recursive] [--] [<path>...]
git submodule [--quiet] summary [--cached|--files] [(-n|--summary-limit) <n>]
[commit] [--] [<path>...]
git submodule [--quiet] foreach [--recursive] <command>
git submodule [--quiet] sync [--recursive] [--] [<path>...]

DESCRIPTION

Submodules allow foreign repositories to be embedded within a dedicated subdirectory of the source tree, always pointed at a particular commit.

They are not to be confused with remotes, which are meant mainly for branches of the same project; submodules are meant for different projects you would like to make part of your source tree, while the history of the two projects still stays completely independent and you cannot modify the contents of the submodule from within the main project. If you want to merge the project histories and want to treat the aggregated whole as a single project from then on, you may want to add a remote for the other project and use the *subtree* merge strategy, instead of treating the other project as a submodule. Directories that come from both projects can be cloned and checked out as a whole if you choose to go that route.

Submodules are composed from a so-called gitlink tree entry in the main repository that refers to a particular commit object within the inner repository that is completely separate. A record in the .gitmodules (see **gitmodules(5)**) file at the root of the source tree assigns a logical name to the submodule and describes the default URL the submodule shall be cloned from. The logical name can be used for overriding this URL within your local repository configuration (see *submodule init*).

This command will manage the tree entries and contents of the gitmodules file for you, as well as inspect the status of your submodules and update them. When adding a new submodule to the tree, the *add* subcommand is to be used. However, when pulling a tree containing submodules, these will not be checked out by default; the *init* and *update* subcommands will maintain submodules checked out and at appropriate revision in your working tree. You can briefly inspect the up-to-date status of your submodules using the *status* subcommand and get a detailed overview of the difference between the index and checkouts using the *summary* subcommand.

COMMANDS

add

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.

This requires at least one argument: <repository>. The optional argument <path> is the relative location for the cloned submodule to exist in the superproject. If <path> is not given, the humanish part of the source repository is used (repo for /path/to/repo.git and foo for host.xz:foo/.git). The <path> is also used as the submodule's logical name in its configuration entries unless --name is used to specify a logical name.

<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 origin

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). If the superproject doesn't have an origin configured the superproject is its own authoritative upstream and the current working directory is used instead.

<path> is the relative location for the cloned submodule to exist in the superproject. If <path> does not exist, then the submodule is created by cloning from the named URL. If <path> does exist and is already a valid Git repository, then this is added to the changeset without cloning. This second form is provided to ease creating a new submodule from scratch, and presumes the user will later push the submodule to the given URL.

In either case, 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
: 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 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 --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
: Initialize the submodules recorded in the index (which were added and committed elsewhere) by copying submodule names and urls from .gitmodules to .git/config. Optional <path> arguments limit which submodules will be initialized. It will also copy the value of submodule.$name.update into .git/config. The key used in .git/config is submodule.$name.url. 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.

deinit
: 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 work tree anymore. If you really want to remove a submodule from the repository and commit that use **git-rm(1)** instead.

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

update
: Update the registered submodules, i.e. clone missing submodules and checkout the commit specified in the index of the containing repository. This will make the submodules HEAD be detached unless --rebase or --merge is specified or the key submodule.$name.update is set to rebase, merge or none. none can be overridden by specifying --checkout. Setting the key

submodule.$name.update to !command will cause command to be run. command can be any arbitrary shell command that takes a single argument, namely the sha1 to update to.

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.

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

summary

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

Evaluates an arbitrary shell command in each checked out submodule. The command has access to the variables $name, $path, $sha1 and $toplevel: $name is the name of the relevant submodule section in .gitmodules, $path is the name of the submodule directory relative to the superproject, $sha1 is the commit as recorded in the superproject, and $toplevel is the absolute path to the top-level of the superproject. 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, git submodule foreach echo $path ‘git rev-parse HEAD‘ will show the path and currently checked out commit for each submodule.

sync

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.

OPTIONS

-q, --quiet

Only print error messages.

-b, --branch

Branch of repository to add as submodule. The name of the branch is recorded as submodule.<name>.branch in .gitmodules for update --remote.

-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 work trees will be removed even if they contain local changes. When running update, 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 master, 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.$name.update when set to merge, rebase or none. If the key submodule.$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 and --shared options carefully.

--recursive
: This option is only valid for foreach, update and status 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)**

<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.

GIT

Part of the **git(1)** suite