**Git Bash Commands**

## NAME

git-status - Show the working tree status

## SYNOPSIS

git status [<options>…​] [--] [<pathspec>…​]

## DESCRIPTION

Displays paths that have differences between the index file and the current HEAD commit, paths that have differences between the working tree and the index file, and paths in the working tree that are not tracked by Git (and are not ignored by [gitignore[5]](https://git-scm.com/docs/gitignore)). The first are what you would commit by running git commit; the second and third are what you could commit by running git add before running git commit.

## OPTIONS

**-s**

**--short**

Give the output in the short-format.

**-b**

**--branch**

Show the branch and tracking info even in short-format.

**--show-stash**

Show the number of entries currently stashed away.

**--porcelain[=<version>]**

Give the output in an easy-to-parse format for scripts. This is similar to the short output, but will remain stable across Git versions and regardless of user configuration. See below for details.

The version parameter is used to specify the format version. This is optional and defaults to the original version v1 format.

**--long**

Give the output in the long-format. This is the default.

**-v**

**--verbose**

In addition to the names of files that have been changed, also show the textual changes that are staged to be committed (i.e., like the output of git diff --cached). If -v is specified twice, then also show the changes in the working tree that have not yet been staged (i.e., like the output of git diff).

**-u[<mode>]**

**--untracked-files[=<mode>]**

Show untracked files.

The mode parameter is used to specify the handling of untracked files. It is optional: it defaults to all, and if specified, it must be stuck to the option (e.g. -uno, but not -u no).

The possible options are:

* no - Show no untracked files.
* normal - Shows untracked files and directories.
* all - Also shows individual files in untracked directories.

When -u option is not used, untracked files and directories are shown (i.e. the same as specifying normal), to help you avoid forgetting to add newly created files. Because it takes extra work to find untracked files in the filesystem, this mode may take some time in a large working tree. Consider enabling untracked cache and split index if supported (see git update-index --untracked-cache and git update-index --split-index), Otherwise you can use no to have git status return more quickly without showing untracked files.

The default can be changed using the status.showUntrackedFiles configuration variable documented in [git-config[1]](https://git-scm.com/docs/git-config).

**--ignore-submodules[=<when>]**

Ignore changes to submodules when looking for changes. <when> can be either "none", "untracked", "dirty" or "all", which is the default. Using "none" will consider the submodule modified when it either contains untracked or modified files or its HEAD differs from the commit recorded in the superproject and can be used to override any settings of the ignore option in [git-config[1]](https://git-scm.com/docs/git-config) or [gitmodules[5]](https://git-scm.com/docs/gitmodules). When "untracked" is used submodules are not considered dirty when they only contain untracked content (but they are still scanned for modified content). Using "dirty" ignores all changes to the work tree of submodules, only changes to the commits stored in the superproject are shown (this was the behavior before 1.7.0). Using "all" hides all changes to submodules (and suppresses the output of submodule summaries when the config option status.submoduleSummary is set).

**--ignored[=<mode>]**

Show ignored files as well.

The mode parameter is used to specify the handling of ignored files. It is optional: it defaults to traditional.

The possible options are:

* traditional - Shows ignored files and directories, unless --untracked-files=all is specifed, in which case individual files in ignored directories are displayed.
* no - Show no ignored files.
* matching - Shows ignored files and directories matching an ignore pattern.

When matching mode is specified, paths that explicity match an ignored pattern are shown. If a directory matches an ignore pattern, then it is shown, but not paths contained in the ignored directory. If a directory does not match an ignore pattern, but all contents are ignored, then the directory is not shown, but all contents are shown.

**-z**

Terminate entries with NUL, instead of LF. This implies the --porcelain=v1 output format if no other format is given.

**--column[=<options>]**

**--no-column**

Display untracked files in columns. See configuration variable column.status for option syntax.--column and --no-column without options are equivalent to always and never respectively.

**<pathspec>…​**

See the pathspec entry in [gitglossary[7]](https://git-scm.com/docs/gitglossary).

## NAME

git-remote - Manage set of tracked repositories

## SYNOPSIS

git remote [-v | --verbose]

git remote add [-t <branch>] [-m <master>] [-f] [--[no-]tags] [--mirror=<fetch|push>] <name> <url>

git remote rename <old> <new>

git remote remove <name>

git remote set-head <name> (-a | --auto | -d | --delete | <branch>)

git remote set-branches [--add] <name> <branch>…​

git remote get-url [--push] [--all] <name>

git remote set-url [--push] <name> <newurl> [<oldurl>]

git remote set-url --add [--push] <name> <newurl>

git remote set-url --delete [--push] <name> <url>

git remote [-v | --verbose] show [-n] <name>…​

git remote prune [-n | --dry-run] <name>…​

git remote [-v | --verbose] update [-p | --prune] [(<group> | <remote>)…​]

## DESCRIPTION

Manage the set of repositories ("remotes") whose branches you track.

## OPTIONS

**-v**

**--verbose**

Be a little more verbose and show remote url after name. NOTE: This must be placed between remoteand subcommand.

## COMMANDS

With no arguments, shows a list of existing remotes. Several subcommands are available to perform operations on the remotes.

**add**

Adds a remote named <name> for the repository at <url>. The command git fetch <name> can then be used to create and update remote-tracking branches <name>/<branch>.

With -f option, git fetch <name> is run immediately after the remote information is set up.

With --tags option, git fetch <name> imports every tag from the remote repository.

With --no-tags option, git fetch <name> does not import tags from the remote repository.

By default, only tags on fetched branches are imported (see [git-fetch[1]](https://git-scm.com/docs/git-fetch)).

With -t <branch> option, instead of the default glob refspec for the remote to track all branches under the refs/remotes/<name>/ namespace, a refspec to track only <branch> is created. You can give more than one -t <branch> to track multiple branches without grabbing all branches.

With -m <master> option, a symbolic-ref refs/remotes/<name>/HEAD is set up to point at remote’s <master> branch. See also the set-head command.

When a fetch mirror is created with --mirror=fetch, the refs will not be stored in the refs/remotes/ namespace, but rather everything in refs/ on the remote will be directly mirrored into refs/ in the local repository. This option only makes sense in bare repositories, because a fetch would overwrite any local commits.

When a push mirror is created with --mirror=push, then git push will always behave as if --mirror was passed.

**rename**

Rename the remote named <old> to <new>. All remote-tracking branches and configuration settings for the remote are updated.

In case <old> and <new> are the same, and <old> is a file under $GIT\_DIR/remotes or $GIT\_DIR/branches, the remote is converted to the configuration file format.

**remove**

**rm**

Remove the remote named <name>. All remote-tracking branches and configuration settings for the remote are removed.

**set-head**

Sets or deletes the default branch (i.e. the target of the symbolic-ref refs/remotes/<name>/HEAD) for the named remote. Having a default branch for a remote is not required, but allows the name of the remote to be specified in lieu of a specific branch. For example, if the default branch for origin is set to master, then origin may be specified wherever you would normally specify origin/master.

With -d or --delete, the symbolic ref refs/remotes/<name>/HEAD is deleted.

With -a or --auto, the remote is queried to determine its HEAD, then the symbolic-ref refs/remotes/<name>/HEAD is set to the same branch. e.g., if the remote HEAD is pointed at next, "git remote set-head origin -a" will set the symbolic-ref refs/remotes/origin/HEAD to refs/remotes/origin/next. This will only work if refs/remotes/origin/next already exists; if not it must be fetched first.

Use <branch> to set the symbolic-ref refs/remotes/<name>/HEAD explicitly. e.g., "git remote set-head origin master" will set the symbolic-ref refs/remotes/origin/HEAD torefs/remotes/origin/master. This will only work if refs/remotes/origin/masteralready exists; if not it must be fetched first.

**set-branches**

Changes the list of branches tracked by the named remote. This can be used to track a subset of the available remote branches after the initial setup for a remote.

The named branches will be interpreted as if specified with the -t option on the git remote addcommand line.

With --add, instead of replacing the list of currently tracked branches, adds to that list.

**get-url**

Retrieves the URLs for a remote. Configurations for insteadOf and pushInsteadOf are expanded here. By default, only the first URL is listed.

With --push, push URLs are queried rather than fetch URLs.

With --all, all URLs for the remote will be listed.

**set-url**

Changes URLs for the remote. Sets first URL for remote <name> that matches regex <oldurl> (first URL if no <oldurl> is given) to <newurl>. If <oldurl> doesn’t match any URL, an error occurs and nothing is changed.

With --push, push URLs are manipulated instead of fetch URLs.

With --add, instead of changing existing URLs, new URL is added.

With --delete, instead of changing existing URLs, all URLs matching regex <url> are deleted for remote <name>. Trying to delete all non-push URLs is an error.

Note that the push URL and the fetch URL, even though they can be set differently, must still refer to the same place. What you pushed to the push URL should be what you would see if you immediately fetched from the fetch URL. If you are trying to fetch from one place (e.g. your upstream) and push to another (e.g. your publishing repository), use two separate remotes.

**show**

Gives some information about the remote <name>.

With -n option, the remote heads are not queried first with git ls-remote <name>; cached information is used instead.

**prune**

Deletes all stale remote-tracking branches under <name>. These stale branches have already been removed from the remote repository referenced by <name>, but are still locally available in "remotes/<name>".

With --dry-run option, report what branches will be pruned, but do not actually prune them.

**update**

Fetch updates for a named set of remotes in the repository as defined by remotes.<group>. If a named group is not specified on the command line, the configuration parameter remotes.default will be used; if remotes.default is not defined, all remotes which do not have the configuration parameter remote.<name>.skipDefaultUpdate set to true will be updated. (See [git-config[1]](https://git-scm.com/docs/git-config)).

With --prune option, prune all the remotes that are updated.

## DISCUSSION

The remote configuration is achieved using the remote.origin.url and remote.origin.fetchconfiguration variables. (See [git-config[1]](https://git-scm.com/docs/git-config)).

## Examples

* Add a new remote, fetch, and check out a branch from it
* $ git remote
* origin
* $ git branch -r
* origin/HEAD -> origin/master
* origin/master
* $ git remote add staging git://git.kernel.org/.../gregkh/staging.git
* $ git remote
* origin
* staging
* $ git fetch staging
* ...
* From git://git.kernel.org/pub/scm/linux/kernel/git/gregkh/staging
* \* [new branch] master -> staging/master
* \* [new branch] staging-linus -> staging/staging-linus
* \* [new branch] staging-next -> staging/staging-next
* $ git branch -r
* origin/HEAD -> origin/master
* origin/master
* staging/master
* staging/staging-linus
* staging/staging-next
* $ git checkout -b staging staging/master

...

* Imitate git clone but track only selected branches
* $ mkdir project.git
* $ cd project.git
* $ git init
* $ git remote add -f -t master -m master origin git://example.com/git.git/

$ git merge origin

## SEE ALSO

[git-fetch[1]](https://git-scm.com/docs/git-fetch) [git-branch[1]](https://git-scm.com/docs/git-branch) [git-config[1]](https://git-scm.com/docs/git-config)

## GIT

Part of the [git[1]](https://git-scm.com/docs/git) suite