NAME

git-branch - List, create, or delete branches

SYNOPSIS

git branch [--color[=<when>] | --no-color] [-r | -a]
[--list] [-v [--abbrev=<length> | --no-abbrev]]
[--column[=<options>] | --no-column]
[(--merged | --no-merged | --contains) [<commit>]] [<pattern>...]
git branch [--set-upstream | --track | --no-track] [-l] [-f] <branchname> [<start-point>]
git branch (--set-upstream-to=<upstream> | -u <upstream>) [<branchname>]
git branch --unset-upstream [<branchname>]
git branch (-m | -M) [<oldbranch>] <newbranch>
git branch (-d | -D) [-r] <branchname>...
git branch --edit-description [<branchname>]

DESCRIPTION

If --list is given, or if there are no non-option arguments, existing branches are listed; the current branch will be highlighted with an asterisk. Option -r causes the remote-tracking branches to be listed, and option -a shows both local and remote branches. If a <pattern> is given, it is used as a shell wildcard to restrict the output to matching branches. If multiple patterns are given, a branch is shown if it matches any of the patterns. Note that when providing a <pattern>, you must use --list; otherwise the command is interpreted as branch creation.

With --contains, shows only the branches that contain the named commit (in other words, the branches whose tip commits are descendants of the named commit). With --merged, only branches merged into the named commit (i.e. the branches whose tip commits are reachable from the named commit) will be listed. With --no-merged only branches not merged into the named commit will be listed. If the <commit> argument is missing it defaults to *HEAD* (i.e. the tip of the current branch).

The command's second form creates a new branch head named <branchname> which points to the current *HEAD*, or <start-point> if given.

Note that this will create the new branch, but it will not switch the working tree to it; use git checkout <newbranch> to switch to the new branch.

When a local branch is started off a remote-tracking branch, Git sets up the branch (specifically the branch.<name>.remote and branch.<name>.merge configuration entries) so that *git pull* will appropriately merge from the remote-tracking branch. This behavior may be changed via the global branch.autosetupmerge configuration flag. That setting can be overridden by using the --track and --no-track options, and changed later using git branch --set-upstream-to.

With a -m or -M option, <oldbranch> will be renamed to <newbranch>. If <oldbranch> had a corresponding reflog, it is renamed to match <newbranch>, and a reflog entry is created to remember the branch renaming. If <newbranch> exists, -M must be used to force the rename to happen.

With a -d or -D option, <branchname> will be deleted. You may specify more than one branch for deletion. If the branch currently has a reflog then the reflog will also be deleted.

Use -r together with -d to delete remote-tracking branches. Note, that it only makes sense to delete remote-tracking branches if they no longer exist in the remote repository or if *git fetch* was configured not to fetch them again. See also the *prune* subcommand of **git-remote(1)** for a way to clean up all obsolete remote-tracking branches.

OPTIONS

-d, --delete

Delete a branch. The branch must be fully merged in its upstream branch, or in HEAD if no upstream was set with --track or --set-upstream.

-D
: Delete a branch irrespective of its merged status.

-l, --create-reflog
: Create the branch's reflog. This activates recording of all changes made to the branch ref, enabling use of date based sha1 expressions such as <branchname>@{yesterday}. Note that in non-bare repositories, reflogs are usually enabled by default by the core.logallrefupdates config option.

-f, --force
: Reset <branchname> to <startpoint> if <branchname> exists already. Without -f*git branch* refuses to change an existing branch.

-m, --move
: Move/rename a branch and the corresponding reflog.

-M
: Move/rename a branch even if the new branch name already exists.

--color[=<when>]
: Color branches to highlight current, local, and remote-tracking branches. The value must be always (the default), never, or auto.

--no-color
: Turn off branch colors, even when the configuration file gives the default to color output. Same as --color=never.

--column[=<options>], --no-column
: Display branch listing in columns. See configuration variable column.branch for option syntax.--column and --no-column without options are equivalent to *always* and *never* respectively.

  This option is only applicable in non-verbose mode.

-r, --remotes
: List or delete (if used with -d) the remote-tracking branches.

-a, --all
: List both remote-tracking branches and local branches.

--list
: Activate the list mode. git branch <pattern> would try to create a branch, use git branch --list <pattern> to list matching branches.

-v, -vv, --verbose
: When in list mode, show sha1 and commit subject line for each head, along with relationship to upstream branch (if any). If given twice, print the name of the upstream branch, as well (see also git remote show <remote>).

-q, --quiet
: Be more quiet when creating or deleting a branch, suppressing non-error messages.

--abbrev=<length>
: Alter the sha1's minimum display length in the output listing. The default value is 7 and can be overridden by the core.abbrev config option.

--no-abbrev
: Display the full sha1s in the output listing rather than abbreviating them.

-t, --track
: When creating a new branch, set up branch.<name>.remote and branch.<name>.merge configuration entries to mark the start-point branch as upstream from the new branch. This configuration will tell git to show the relationship between the two branches in git status and git branch -v. Furthermore, it directs git pull without arguments to pull from the upstream

when the new branch is checked out.

This behavior is the default when the start point is a remote-tracking branch. Set the branch.autosetupmerge configuration variable to false if you want git checkout and git branch to always behave as if *--no-track* were given. Set it to always if you want this behavior when the start-point is either a local or remote-tracking branch.

--no-track
Do not set up upstream configuration, even if the branch.autosetupmerge configuration variable is true.

--set-upstream
If specified branch does not exist yet or if --force has been given, acts exactly like --track. Otherwise sets up configuration like --track would when creating the branch, except that where branch points to is not changed.

-u <upstream>, --set-upstream-to=<upstream>
Set up <branchname>s tracking information so <upstream> is considered <branchname>s upstream branch. If no <branchname> is specified, then it defaults to the current branch.

--unset-upstream
Remove the upstream information for <branchname>. If no branch is specified it defaults to the current branch.

--edit-description
Open an editor and edit the text to explain what the branch is for, to be used by various other commands (e.g. request-pull).

--contains [<commit>]
Only list branches which contain the specified commit (HEAD if not specified). Implies --list.

--merged [<commit>]
Only list branches whose tips are reachable from the specified commit (HEAD if not specified). Implies --list.

--no-merged [<commit>]
Only list branches whose tips are not reachable from the specified commit (HEAD if not specified). Implies --list.

<branchname>
The name of the branch to create or delete. The new branch name must pass all checks defined by **git-check-ref-format(1)**. Some of these checks may restrict the characters allowed in a branch name.

<start-point>
The new branch head will point to this commit. It may be given as a branch name, a commit-id, or a tag. If this option is omitted, the current HEAD will be used instead.

<oldbranch>
The name of an existing branch to rename.

<newbranch>
The new name for an existing branch. The same restrictions as for <branchname> apply.

EXAMPLES

Start development from a known tag

```
$ git clone git://git.kernel.org/pub/scm/.../linux-2.6 my2.6
$ cd my2.6
$ git branch my2.6.14 v2.6.14 (1)
$ git checkout my2.6.14
```

1. This step and the next one could be combined into a single step with checkout -b my2.6.14 v2.6.14.

Delete an unneeded branch

```
$ git clone git://git.kernel.org/.../git.git my.git
$ cd my.git
$ git branch -d -r origin/todo origin/html origin/man (1)
$ git branch -D test (2)
```

1. Delete the remote-tracking branches todo, html and man. The next *fetch* or *pull* will create them again unless you configure them not to. See **git-fetch(1)**.
2. Delete the test branch even if the master branch (or whichever branch is currently checked out) does not have all commits from the test branch.

NOTES

If you are creating a branch that you want to checkout immediately, it is easier to use the git checkout command with its -b option to create a branch and check it out with a single command.

The options --contains, --merged and --no-merged serve three related but different purposes:

- --contains <commit> is used to find all branches which will need special attention if <commit> were to be rebased or amended, since those branches contain the specified <commit>.
- --merged is used to find all branches which can be safely deleted, since those branches are fully contained by HEAD.
- --no-merged is used to find branches which are candidates for merging into HEAD, since those branches are not fully contained by HEAD.

SEE ALSO

git-check-ref-format(1), **git-fetch(1)**, **git-remote(1)**, **"Understanding history: What is a branch?"**[1] in the Git User's Manual.

GIT

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

NOTES

1. "Understanding history: What is a branch?" file:///usr/share/doc/git/html/user-manual.html#what-is-a-branch