git-branch

git-branch(1) ==========

NAME ---- git-branch - List, create, or delete branches

SYNOPSIS --------

git branch [--color[=<when>] | --no-color] [--show-current] [-v [--abbrev=<n> | --no-abbrev]] [--column[=<options>] | --no-column] [--sort=<key>] [--merged [<commit>]] [--no-merged [<commit>]] [--contains [<commit>]] [--no-contains [<commit>]] [(--forked <branch>)…] [--points-at <object>] [--format=<format>] [(-r|--remotes) | (-a|--all)] [--list] [<pattern>…] git branch [--track[=(direct|inherit)] | --no-track] [-f] [--recurse-submodules] <branch-name> [<start-point>] git branch (--set-upstream-to=<upstream>|-u <upstream>) [<branch-name>] git branch --unset-upstream [<branch-name>] git branch (-m|-M) [<old-branch>] <new-branch> git branch (-c|-C) [<old-branch>] <new-branch> git branch (-d|-D) [-r] <branch-name>… git branch --edit-description [<branch-name>] git branch [--dry-run] (--delete-merged <pattern>)… [<branch-pattern>…]

DESCRIPTION -----------

If `--list` is given, or if there are no non-option arguments, existing branches are listed; the current branch will be highlighted in green and marked with an asterisk. Any branches checked out in linked worktrees will be highlighted in cyan and marked with a plus sign. 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 may be 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), `--no-contains` inverts it. 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). With `--forked`, only branches whose configured upstream matches the given branch or pattern will be listed.

The command’s second form creates a new branch head named *<branch-name>* which points to the current `HEAD`, or *<start-point>* if given. As a special case, for *<start-point>*, you may use `<rev-A>…<rev-B>` as a shortcut for the merge base of *<rev-A>* and *<rev-B>* if there is exactly one

merge base. You can leave out at most one of <*rev-A*> and <*rev-B*>, in which case it defaults to `HEAD`.

Note that this will create the new branch, but it will not switch the working tree to it; use `git switch <new-branch>` 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, <*old-branch*> will be renamed to <*new-branch*>. If <*old-branch*> had a corresponding reflog, it is renamed to match <*new-branch*>, and a reflog entry is created to remember the branch renaming. If <*new-branch*> exists, `-M` must be used to force the rename to happen.

The `-c` and `-C` options have the exact same semantics as `-m` and `-M`, except instead of the branch being renamed, it will be copied to a new name, along with its config and reflog.

With a `-d` or `-D` option, <*branch-name*> 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 linkgit: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-to`.

`-D`
Shortcut for `--delete` `--force`.

`--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 `<branch-name>@{yesterday}`. Note that in non-bare repositories, reflogs are usually enabled by default by the `core.logAllRefUpdates` config option. The negated form `--no-create-reflog` only overrides an earlier `--create-reflog`, but currently does not negate the setting of `core.logAllRefUpdates`.

`-f`

`--force`
Reset <*branch-name*> to <*start-point*>, even if <*branch-name*> exists already. Without `-f`, `git branch` refuses to change an existing branch. In combination with `-d` (or `--delete`), allow deleting the branch irrespective of its merged status, or whether it even points to a valid commit. In

combination with `-m` (or `--move`), allow renaming the branch even if the new branch name already exists, the same applies for `-c` (or `--copy`).

Note that `git branch -f <branch-name> [<start-point>]`, even with `-f`, refuses to change an existing branch *<branch-name>* that is checked out in another worktree linked to the same repository.

`-m`

`--move`
Move/rename a branch, together with its config and reflog.

`-M`
Shortcut for `--move --force`.

`-c`

`--copy`
Copy a branch, together with its config and reflog.

`-C`
Shortcut for `--copy --force`.

`--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`.

`-i`

`--ignore-case`
Sorting and filtering branches are case insensitive.

`--omit-empty`
Do not print a newline after formatted refs where the format expands to the empty string.

`--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.

`--sort=<key>`
Sort based on *<key>*. Prefix `-` to sort in descending order of the value. You may use the `--sort=<key>` option multiple times, in which case the last key becomes the primary key. The keys supported are the same as those in linkgit:git-for-each-ref[1]. Sort order defaults to the value configured for the `branch.sort` variable if it exists, or to sorting based on the full refname (including `refs/…` prefix). This lists detached `HEAD` (if present) first, then local branches and finally remote-tracking branches. See linkgit:git-config[1].

`-r`

`--remotes`
List or delete (if used with `-d`) the remote-tracking branches. Combine with `--list` to match the optional pattern(s).

`-a`

`--all`
List both remote-tracking branches and local branches. Combine with `--list` to match optional pattern(s).

`-l`

`--list`
List branches. With optional `<pattern>…`, e.g. `git branch --list 'maint-*'`, list only the branches that match the pattern(s).

`--show-current`
Print the name of the current branch. In detached `HEAD` state, nothing is printed.

`--delete-merged` `<pattern>`
Delete local branches whose configured upstream matches *<pattern>*, but only when their tip is reachable from that upstream. In other words, the work on the branch has already landed on the upstream it tracks, so the local copy is no longer needed. *<pattern>* may name a ref, a remote (using the branch its `HEAD` points at), or a shell-style glob. The option can be repeated to widen the upstream match. Optional *<branch-pattern>* arguments limit which local branches are considered, e.g. `git branch --delete-merged 'origin/' 'topic-'`.

A branch is not deleted when:

- its configured upstream ref no longer exists,
- it is checked out in any worktree,
- pushing it to the remote configured by `branch.<name>.remote` would update its upstream, so it cannot be distinguished from a branch that just looks fully merged right after a pull; this is determined by the remote's configured push and fetch refspecs,
- it is the local upstream of a branch that is not being deleted, or
- `branch.<name>.deleteMerged` is set to `false`.

When such a local upstream branch has its own upstream deleted by the same operation, its upstream configuration is cleared.

A branch whose work has not yet been merged into its upstream is silently skipped. Delete it with `git branch -D` if you want to remove it anyway.

`--dry-run`
With `--delete-merged`, print which branches would be deleted and exit without touching any ref. Useful for sanity-checking a wide pattern like `'origin/*'` before committing to the deletion.

`-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 path of the linked worktree (if any) and the name of the upstream branch, as well (see also `git remote show <remote>`). Note that the current worktree's `HEAD` will not have its path printed (it will always be your current directory).

`-q`

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

`--abbrev=<n>`
In the verbose listing that show the commit object name, show the shortest prefix that is at least *<n>* hexdigits long that uniquely refers the object. 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[=(direct|inherit)]`
When creating a new branch, set up `branch.<name>.remote` and `branch.<name>.merge` configuration entries to set "upstream" tracking configuration for 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.

The exact upstream branch is chosen depending on the optional argument: `-t`, `--track`, or `--track=direct` means to use the start-point branch itself as the upstream; `--track=inherit` means to copy the upstream configuration of the start-point branch.

The `branch.autoSetupMerge` configuration variable specifies how `git switch`, `git checkout` and `git branch` should behave when neither `--track` nor `--no-track` are specified:

The default option, `true`, behaves as though `--track=direct` were given whenever the start-point is a remote-tracking branch. `false` behaves as if `--no-track` were given. `always` behaves as though `--track=direct` were given. `inherit` behaves as though `--track=inherit` were given. `simple` behaves as though `--track=direct` were given only when the <*start-point*> is a remote-tracking branch and the new branch has the same name as the remote branch.

See linkgit:git-pull[1] and linkgit:git-config[1] for additional discussion on how the `branch.<name>.remote` and `branch.<name>.merge` options are used.

`--no-track`
Do not set up "upstream" configuration, even if the `branch.autoSetupMerge` configuration variable is set.

`--recurse-submodules`
THIS OPTION IS EXPERIMENTAL! Cause the current command to recurse into submodules if `submodule.propagateBranches` is enabled. See `submodule.propagateBranches` in linkgit:git-config[1]. Currently, only branch creation is supported.

When used in branch creation, a new branch <*branch-name*> will be created in the superproject and all of the submodules in the superproject's <*start-point*>. In submodules, the branch will point to the submodule commit in the superproject's <*start-point*> but the branch's tracking information will be set up based on the submodule's branches and remotes e.g. `git branch --recurse-submodules topic origin/main` will create the submodule branch "topic" that points to the submodule commit in the superproject's "origin/main", but tracks the submodule's "origin/main".

`--set-upstream`
As this option had confusing syntax, it is no longer supported. Please use `--track` or `--set-upstream-to` instead.

`-u <upstream>`

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

`--unset-upstream`
Remove the upstream information for <*branch-name*>. 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. `format-patch`, `request-pull`, and `merge` (if enabled)). Multi-line explanations may be used.

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

`--no-contains` `[<commit>]`
Only list branches which don’t contain *<commit>* (`HEAD` if not specified). Implies `--list`.

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

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

`--forked` `<branch>`
Only list branches whose configured upstream matches *<branch>*. The argument can be a ref (e.g. `origin/main`, `master`), a remote name like `origin` for the branch its `origin/HEAD` points at, or a shell-style glob (e.g. `’origin/*’`). The option can be repeated to widen the filter. Implies `--list`.

`--points-at` `<object>`
Only list branches of *<object>*.

`--format` `<format>`
A string that interpolates `%(fieldname)` from a branch ref being shown and the object it points at. *<format>* is the same as that of linkgit:git-for-each-ref[1].

<branch-name>
The name of the branch to create or delete. The new branch name must pass all checks defined by linkgit: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.

<old-branch>
The name of an existing branch. If this option is omitted, the name of the current branch will be used instead.

<new-branch>
The new name for an existing branch. The same restrictions as for *<branch-name>* apply.

CONFIGURATION ---------- `pager.branch` is only respected when listing branches, i.e., when `--list` is used or implied. The default is to use a pager. See linkgit:git-config[1].

Everything above this line in this section isn’t included from the linkgit:git-config[1] documentation. The content that follows is the same as what’s found there:

`branch.autoSetupMerge`
Tells `git branch`, `git switch` and `git checkout` to set up new branches so that linkgit:git-pull[1] will appropriately merge from the starting point branch. Note that even if this option is not set, this behavior can be chosen per-branch using the `--track` and `--no-track` options. This option defaults to `true`. The valid settings are: `false`;; no automatic setup is done `true`;; automatic setup is done when the starting point is a remote-tracking branch `always`;; automatic setup is done when the starting point is either a local branch or remote-tracking branch `inherit`;; if the starting point has a tracking configuration, it is copied to the new branch `simple`;; automatic setup is done only when the starting point is a remote-tracking branch and the new branch has the same name as the remote branch.

`branch.autoSetupRebase`
When a new branch is created with `git branch`, `git switch` or `git checkout` that tracks another branch, this variable tells Git to set up pull to rebase instead of merge (see `branch.<name>.rebase`). The valid settings are: `never`;; rebase is never automatically set to true. `local`;; rebase is set to true for tracked branches of other local branches. `remote`;; rebase is set to true for tracked branches of remote-tracking branches. `always`;; rebase will be set to true for all tracking branches.

See `branch.autoSetupMerge` for details on how to set up a branch to track another branch. This option defaults to `never`.

`branch.sort`
This variable controls the sort ordering of branches when displayed by linkgit:git-branch[1]. Without the `--sort=<value>` option provided, the value of this variable will be used as the default. See linkgit:git-for-each-ref[1] field names for valid values.

`branch.<name>.remote`
When on branch <*name*>, it tells `git fetch` and `git push` which remote to fetch from or push to. The remote to push to may be overridden with `remote.pushDefault` (for all branches). The remote to push to, for the current branch, may be further overridden by `branch.<name>.pushRemote`. If no remote is configured, or if you are not on any branch and there is more than one remote defined in the repository, it defaults to `origin` for fetching and `remote.pushDefault` for pushing. Additionally, `.` (a period) is the current local repository (a dot-repository), see `branch.<name>.merge`'s final note below.

`branch.<name>.pushRemote`
When on branch <*name*>, it overrides `branch.<name>.remote` for pushing. It also overrides `remote.pushDefault` for pushing from branch <*name*>. When you pull from one place (e.g. your upstream) and push to another place (e.g. your own publishing repository), you would want to set `remote.pushDefault` to specify the remote to push to for all branches, and use this option to override it for a specific branch. The value may be the name of a configured remote or a repository URL.

`branch.<name>.merge`
Defines, together with `branch.<name>.remote`, the upstream branch for the given branch. It tells `git fetch`/`git pull`/`git rebase` which branch to merge and can also affect `git push` (see `push.default`). When in branch *<name>*, it tells `git fetch` the default refspec to be marked for merging in `FETCH_HEAD`. The value is handled like the remote part of a refspec, and must match a ref which is fetched from the remote given by `branch.<name>.remote`. The merge information is used by `git pull` (which first calls `git fetch`) to lookup the default branch for merging. Without this option, `git pull` defaults to merge the first refspec fetched. Specify multiple values to get an octopus merge. If you wish to setup `git pull` so that it merges into *<name>* from another branch in the local repository, you can point `branch.<name>.merge` to the desired branch, and use the relative path setting `.` (a period) for `branch.<name>.remote`.

`branch.<name>.mergeOptions`
Sets default options for merging into branch *<name>*. The syntax and supported options are the same as those of linkgit:git-merge[1], but option values containing whitespace characters are currently not supported.

`branch.<name>.rebase`
When true, rebase the branch *<name>* on top of the fetched branch, instead of merging the default branch from the default remote when `git pull` is run. See `pull.rebase` for doing this in a non branch-specific manner.

When `merges` (or just `m`), pass the `--rebase-merges` option to `git rebase` so that the local merge commits are included in the rebase (see linkgit:git-rebase[1] for details).

When the value is `interactive` (or just `i`), the rebase is run in interactive mode.

NOTE: this is a possibly dangerous operation; do **not** use it unless you understand the implications (see linkgit:git-rebase[1] for details).

`branch.<name>.description`
Branch description, can be edited with `git branch --edit-description`. Branch description is automatically added to the `format-patch` cover letter or `request-pull` summary.

`branch.<name>.deleteMerged`
If set to `false`, branch *<name>* is exempt from `git branch --delete-merged`. Useful for a topic branch you intend to develop further after an initial round has been merged upstream. Defaults to true. Explicit deletion via `git branch -d` is unaffected.

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 switch 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 `git` `fetch` or `git` `pull` will create them again unless you configure them not to. See linkgit: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.

Listing branches from a specific remote
$ git branch -r -l '/' ① $ git for-each-ref 'refs/remotes//' ②

1. Using `-a` would conflate *<remote>* with any local branches you happen to have been prefixed with the same *<remote>* pattern.
2. `for-each-ref` can take a wide range of options. See linkgit:git-for-each-ref[1]

Patterns will normally need quoting.

NOTES -----

If you are creating a branch that you want to switch to immediately, it is easier to use the `git` `switch` command with its `-c` option to do the same thing with a single command.

The options `--contains`, `--no-contains`, `--merged` and `--no-merged` serve four 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>*.

- `--no-contains` `<commit>` is the inverse of that, i.e. branches that don’t 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`.

When combining multiple `--contains` and `--no-contains` filters, only references that contain at least one of the `--contains` commits and contain none of the `--no-contains` commits are shown.

When combining multiple `--merged` and `--no-merged` filters, only references that are reachable from at least one of the `--merged` commits and from none of the `--no-merged` commits are shown.

SEE ALSO -------- linkgit:git-check-ref-format[1], linkgit:git-fetch[1], linkgit:git-remote[1], Understanding history: What is a branch? in the Git User’s Manual.

GIT --- Part of the linkgit:git[1] suite

□□□ #1
□ □□□ □□□ 2026-10-31 20:09:54 UTC
□ □□□ □□□ 2026-10-31 20:09:54 UTC