



Linux Ubuntu 26.04 Manual Pages on command 'git-switch.1'

\$ man git-switch.1

GIT-SWITCH(1) Git Manual GIT-SWITCH(1)

NAME

git-switch - Switch branches

SYNOPSIS

```
git switch [<options>] [--no-guess] <branch>
git switch [<options>] --detach [<start-point>]
git switch [<options>] (-c|-C) <new-branch> [<start-point>]
git switch [<options>] --orphan <new-branch>
```

DESCRIPTION

Switch to a specified branch. The working tree and the index are updated to match the branch. All new commits will be added to the tip of this branch.

Optionally a new branch could be created with either -c, -C, automatically from a remote branch of same name (see --guess), or detach the working tree from any branch with --detach, along with switching.

Switching branches does not require a clean index and working tree (i.e. no differences compared to HEAD). The operation is aborted however if the operation leads to loss of local changes, unless told otherwise with --discard-changes or --merge.

OPTIONS

<branch>

Branch to switch to.

<new-branch>

Name for the new branch.

<start-point>

The starting point for the new branch. Specifying a <start-point> allows you to create a branch based on some other point in history than where HEAD currently points. (Or, in the case of --detach, allows you to inspect and detach from some other point.)

You can use the @{-<N>} syntax to refer to the <N>-th last branch/commit switched to using git switch or git checkout operation. You may also specify - which is synonymous to @{-1}. This is often used to switch quickly between two branches, or to undo a branch switch by mistake.

As a special case, 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.

-c <new-branch>, --create <new-branch>

Create a new branch named <new-branch> starting at <start-point> before switching to the branch. This is the transactional equivalent of

```
$ git branch <new-branch>
```

```
$ git switch <new-branch>
```

that is to say, the branch is not reset/created unless git switch is successful (e.g., when the branch is in use in another worktree, not just the current branch stays the same, but the branch is not reset to the start-point, either).

-C <new-branch>, --force-create <new-branch>

Similar to --create except that if <new-branch> already exists, it will be reset to <start-point>. This is a convenient shortcut for:

```
$ git branch -f _<new-branch>_
```

```
$ git switch _<new-branch>_
```

-d, --detach

Switch to a commit for inspection and discardable experiments. See the "DETACHED HEAD" section in git-checkout(1) for details.

--guess, --no-guess

If <branch> is not found but there does exist a tracking branch in exactly one remote (call it <remote>) with a matching name, treat as equivalent to

```
$ git switch -c <branch> --track <remote>/<branch>
```

If the branch exists in multiple remotes and one of them is named by the checkout.defaultRemote configuration variable, we'll use that one for the purposes of disambiguation, even if the <branch> isn't unique across all remotes. Set it to e.g.

checkout.defaultRemote=origin to always checkout remote branches from there if <branch> is ambiguous but exists on the origin remote. See also checkout.defaultRemote in git-config(1).

--guess is the default behavior. Use --no-guess to disable it.

The default behavior can be set via the checkout.guess configuration variable.

-f, --force

An alias for --discard-changes.

--discard-changes

Proceed even if the index or the working tree differs from HEAD. Both the index and working tree are restored to match the switching target. If --recurse-submodules is specified, submodule content is also restored to match the switching target. This is used to throw away local changes.

-m, --merge

If you have local modifications to one or more files that are different between the current branch and the branch to which you are switching, the command refuses to switch branches in order to preserve your modifications in context. However, with this option, a three-way merge between the current branch, your working tree contents, and the new branch is done, and you will be on the new branch.

When a merge conflict happens, the index entries for conflicting paths are left unmerged, and you need to resolve the conflicts and mark the resolved paths with git add (or git rm if the merge should result in deletion of the path).

--conflict=<style>

The same as --merge option above, but changes the way the conflicting hunks are presented, overriding the merge.conflictStyle configuration variable. Possible values are merge (default), diff3, and zdiff3.

-q, --quiet

Quiet, suppress feedback messages.

--progress, --no-progress

Progress status is reported on the standard error stream by default when it is attached to a terminal, unless --quiet is specified. This flag enables progress reporting even if not attached to a terminal, regardless of --quiet.

-t, --track[(direct|inherit)]

When creating a new branch, set up "upstream" configuration. -c is implied. See --track

in `git-branch(1)` for details.

If no `-c` option is given, the name of the new branch will be derived from the remote-tracking branch, by looking at the local part of the refspec configured for the corresponding remote, and then stripping the initial part up to the `"*"`. This would tell us to use `hack` as the local branch when branching off of `origin/hack` (or `remotes/origin/hack`, or even `refs/remotes/origin/hack`). If the given name has no slash, or the above guessing results in an empty name, the guessing is aborted. You can explicitly give a name with `-c` in such a case.

`--no-track`

Do not set up "upstream" configuration, even if the `branch.autoSetupMerge` configuration variable is true.

`--orphan <new-branch>`

Create a new unborn branch, named `<new-branch>`. All tracked files are removed.

`--ignore-other-worktrees`

`git switch` refuses when the wanted ref is already checked out by another worktree. This option makes it check the ref out anyway. In other words, the ref can be held by more than one worktree.

`--recurse-submodules`, `--no-recurse-submodules`

Using `--recurse-submodules` will update the content of all active submodules according to the commit recorded in the superproject. If nothing (or `--no-recurse-submodules`) is used, submodules working trees will not be updated. Just like `git-submodule(1)`, this will detach HEAD of the submodules.

EXAMPLES

The following command switches to the "master" branch:

```
$ git switch master
```

After working in the wrong branch, switching to the correct branch would be done using:

```
$ git switch mytopic
```

However, your "wrong" branch and correct "mytopic" branch may differ in files that you have modified locally, in which case the above switch would fail like this:

```
$ git switch mytopic
```

error: You have local changes to 'frotz'; not switching branches.

You can give the `-m` flag to the command, which would try a three-way merge:

```
$ git switch -m mytopic
```

Auto-merging frotz

After this three-way merge, the local modifications are not registered in your index file, so git diff would show you what changes you made since the tip of the new branch.

To switch back to the previous branch before we switched to mytopic (i.e. "master" branch):

```
$ git switch -
```

You can grow a new branch from any commit. For example, switch to "HEAD~3" and create branch "fixup":

```
$ git switch -c fixup HEAD~3
```

```
Switched to a new branch 'fixup'
```

If you want to start a new branch from a remote branch of the same name:

```
$ git switch new-topic
```

```
Branch `new-topic` set up to track remote branch `new-topic` from `origin`
```

```
Switched to a new branch `new-topic`
```

To check out commit HEAD~3 for temporary inspection or experiment without creating a new branch:

```
$ git switch --detach HEAD~3
```

```
HEAD is now at 9fc9555312 Merge branch 'cc/shared-index-permbits'
```

If it turns out whatever you have done is worth keeping, you can always create a new name for it (without switching away):

```
$ git switch -c good-surprises
```

CONFIGURATION

Everything below this line in this section is selectively included from the git-config(1) documentation. The content is the same as what's found there:

checkout.defaultRemote

When you run git checkout <something> or git switch <something> and only have one remote, it may implicitly fall back on checking out and tracking e.g. origin/<something>. This stops working as soon as you have more than one remote with a <something> reference. This setting allows for setting the name of a preferred remote that should always win when it comes to disambiguation. The typical use-case is to set this to origin.

Currently this is used by git-switch(1) and git-checkout(1) when git checkout <something> or git switch <something> will checkout the <something> branch on another remote, and by git-worktree(1) when git worktree add refers to a remote branch. This

setting might be used for other checkout-like commands or functionality in the future.

checkout.guess

Provides the default value for the `--guess` or `--no-guess` option in `git checkout` and `git switch`. See `git-switch(1)` and `git-checkout(1)`.

checkout.workers

The number of parallel workers to use when updating the working tree. The default is one, i.e. sequential execution. If set to a value less than one, Git will use as many workers as the number of logical cores available. This setting and `checkout.thresholdForParallelism` affect all commands that perform checkout. E.g. `checkout`, `clone`, `reset`, `sparse-checkout`, etc.

Note

Parallel checkout usually delivers better performance for repositories located on SSDs or over NFS. For repositories on spinning disks and/or machines with a small number of cores, the default sequential checkout often performs better. The size and compression level of a repository might also influence how well the parallel version performs.

checkout.thresholdForParallelism

When running parallel checkout with a small number of files, the cost of subprocess spawning and inter-process communication might outweigh the parallelization gains. This setting allows you to define the minimum number of files for which parallel checkout should be attempted. The default is 100.

SEE ALSO

`git-checkout(1)`, `git-branch(1)`

GIT

Part of the `git(1)` suite

Git 2.53.0

03/02/2026

GIT-SWITCH(1)