



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

\$ man git-merge.1

GIT-MERGE(1)

Git Manual

GIT-MERGE(1)

NAME

git-merge - Join two or more development histories together

SYNOPSIS

git merge [-n] [--stat] [--compact-summary] [--no-commit] [--squash] [--[no-]edit]

[--no-verify] [-s <strategy>] [-X <strategy-option>] [-S[<keyid>]]

[--[no-]allow-unrelated-histories]

[--[no-]rerere-autoupdate] [-m <msg>] [-F <file>]

[--into-name <branch>] [<commit>...]

git merge (--continue | --abort | --quit)

DESCRIPTION

Incorporates changes from the named commits (since the time their histories diverged from the current branch) into the current branch. This command is used by git pull to incorporate changes from another repository and can be used by hand to merge changes from one branch into another.

Assume the following history exists and the current branch is master:

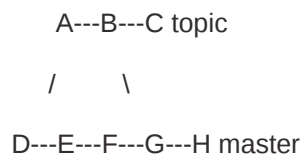
A---B---C topic

/

D---E---F---G master

Then git merge topic will replay the changes made on the topic branch since it diverged from master (i.e., E) until its current commit (C) on top of master, and record the result in a new commit along with the names of the two parent commits and a log message from the user describing the changes. Before the operation, ORIG_HEAD is set to the tip of the current

branch (G).



A merge stops if there's a conflict that cannot be resolved automatically or if `--no-commit` was provided when initiating the merge. At that point you can run `git merge --abort` or `git merge --continue`.

`git merge --abort` will abort the merge process and try to reconstruct the pre-merge state.

However, if there were uncommitted changes when the merge started (and especially if those changes were further modified after the merge was started), `git merge --abort` will in some cases be unable to reconstruct the original (pre-merge) changes. Therefore:

Warning

Running `git merge` with non-trivial uncommitted changes is discouraged: while possible, it may leave you in a state that is hard to back out of in the case of a conflict.

OPTIONS

`--commit`, `--no-commit`

Perform the merge and commit the result. This option can be used to override `--no-commit`.

With `--no-commit` perform the merge and stop just before creating a merge commit, to give the user a chance to inspect and further tweak the merge result before committing.

Note that fast-forward updates do not create a merge commit and therefore there is no way to stop those merges with `--no-commit`. Thus, if you want to ensure your branch is not changed or updated by the merge command, use `--no-ff` with `--no-commit`.

`--edit`, `-e`, `--no-edit`

Invoke an editor before committing successful mechanical merge to further edit the auto-generated merge message, so that the user can explain and justify the merge. The `--no-edit` option can be used to accept the auto-generated message (this is generally discouraged). The `--edit` (or `-e`) option is still useful if you are giving a draft message with the `-m` option from the command line and want to edit it in the editor.

Older scripts may depend on the historical behaviour of not allowing the user to edit the merge log message. They will see an editor opened when they run `git merge`. To make it easier to adjust such scripts to the updated behaviour, the environment variable

`GIT_MERGE_AUTOEDIT` can be set to `no` at the beginning of them.

`--cleanup=<mode>`

This option determines how the merge message will be cleaned up before committing. See `git-commit(1)` for more details. In addition, if the `<mode>` is given a value of `scissors`, scissors will be appended to `MERGE_MSG` before being passed on to the commit machinery in the case of a merge conflict.

`--ff`, `--no-ff`, `--ff-only`

Specifies how a merge is handled when the merged-in history is already a descendant of the current history. `--ff` is the default unless merging an annotated (and possibly signed) tag that is not stored in its natural place in the `refs/tags/` hierarchy, in which case `--no-ff` is assumed.

With `--ff`, when possible resolve the merge as a fast-forward (only update the branch pointer to match the merged branch; do not create a merge commit). When not possible (when the merged-in history is not a descendant of the current history), create a merge commit.

With `--no-ff`, create a merge commit in all cases, even when the merge could instead be resolved as a fast-forward.

With `--ff-only`, resolve the merge as a fast-forward when possible. When not possible, refuse to merge and exit with a non-zero status.

`-S[<key-id>]`, `--gpg-sign[=<key-id>]`, `--no-gpg-sign`

GPG-sign the resulting merge commit. The `<key-id>` argument is optional and defaults to the committer identity; if specified, it must be stuck to the option without a space.

`--no-gpg-sign` is useful to countermand both `commit.gpgSign` configuration variable, and earlier `--gpg-sign`.

`--log[=<n>]`, `--no-log`

In addition to branch names, populate the log message with one-line descriptions from at most `<n>` actual commits that are being merged. See also `git-fmt-merge-msg(1)`.

With `--no-log` do not list one-line descriptions from the actual commits being merged.

`--signoff`, `--no-signoff`

Add a Signed-off-by trailer by the committer at the end of the commit log message. The meaning of a signoff depends on the project to which you're committing. For example, it may certify that the committer has the rights to submit the work under the project's license or agrees to some contributor representation, such as a Developer Certificate of Origin. (See <https://developercertificate.org> for the one used by the Linux kernel and

Git projects.) Consult the documentation or leadership of the project to which you're contributing to understand how the signoffs are used in that project.

The `--no-signoff` option can be used to countermand an earlier `--signoff` option on the command line.

Git does not (and will not) have a configuration variable to enable the `--signoff` command line option by default; see the `commit.signoff` entry in `gitfaq(7)` for more details.

`--stat, -n, --no-stat`

Show a diffstat at the end of the merge. The diffstat is also controlled by the configuration option `merge.stat`.

With `-n` or `--no-stat` do not show a diffstat at the end of the merge.

`--compact-summary`

Show a compact-summary at the end of the merge.

`--squash, --no-squash`

Produce the working tree and index state as if a real merge happened (except for the merge information), but do not actually make a commit, move the HEAD, or record `$GIT_DIR/MERGE_HEAD` (to cause the next git commit command to create a merge commit).

This allows you to create a single commit on top of the current branch whose effect is the same as merging another branch (or more in case of an octopus).

With `--no-squash` perform the merge and commit the result. This option can be used to override `--squash`.

With `--squash`, `--commit` is not allowed, and will fail.

`--verify, --no-verify`

By default, the pre-merge and commit-msg hooks are run. When `--no-verify` is given, these are bypassed. See also `githooks(5)`.

`-s <strategy>, --strategy=<strategy>`

Use the given merge strategy; can be supplied more than once to specify them in the order they should be tried. If there is no `-s` option, a built-in list of strategies is used instead (ort when merging a single head, octopus otherwise).

`-X <option>, --strategy-option=<option>`

Pass merge strategy specific option through to the merge strategy.

`--verify-signatures, --no-verify-signatures`

Verify that the tip commit of the side branch being merged is signed with a valid key,

i.e. a key that has a valid uid: in the default trust model, this means the signing key has been signed by a trusted key. If the tip commit of the side branch is not signed with a valid key, the merge is aborted.

--summary, --no-summary

Synonyms to --stat and --no-stat; these are deprecated and will be removed in the future.

-q, --quiet

Operate quietly. Implies --no-progress.

-v, --verbose

Be verbose.

--progress, --no-progress

Turn progress on/off explicitly. If neither is specified, progress is shown if standard error is connected to a terminal. Note that not all merge strategies may support progress reporting.

--autostash, --no-autostash

Automatically create a temporary stash entry before the operation begins, record it in the ref MERGE_AUTOSTASH and apply it after the operation ends. This means that you can run the operation on a dirty worktree. However, use with care: the final stash application after a successful merge might result in non-trivial conflicts.

--allow-unrelated-histories

By default, git merge command refuses to merge histories that do not share a common ancestor. This option can be used to override this safety when merging histories of two projects that started their lives independently. As that is a very rare occasion, no configuration variable to enable this by default exists or will be added.

-m <msg>

Set the commit message to be used for the merge commit (in case one is created).

If --log is specified, a shortlog of the commits being merged will be appended to the specified message.

The git fmt-merge-msg command can be used to give a good default for automated git merge invocations. The automated message can include the branch description.

--into-name <branch>

Prepare the default merge message as if merging to the branch <branch>, instead of the name of the real branch to which the merge is made.

`-F <file>, --file=<file>`

Read the commit message to be used for the merge commit (in case one is created).

If `--log` is specified, a shortlog of the commits being merged will be appended to the specified message.

`--rerere-autoupdate, --no-rerere-autoupdate`

After the rerere mechanism reuses a recorded resolution on the current conflict to update the files in the working tree, allow it to also update the index with the result of resolution. `--no-rerere-autoupdate` is a good way to double-check what rerere did and catch potential mismerges, before committing the result to the index with a separate git add.

`--overwrite-ignore, --no-overwrite-ignore`

Silently overwrite ignored files from the merge result. This is the default behavior.

Use `--no-overwrite-ignore` to abort.

`--abort`

Abort the current conflict resolution process, and try to reconstruct the pre-merge state. If an autostash entry is present, apply it to the worktree.

If there were uncommitted worktree changes present when the merge started, git merge

`--abort` will in some cases be unable to reconstruct these changes. It is therefore

recommended to always commit or stash your changes before running git merge.

git merge `--abort` is equivalent to git reset `--merge` when `MERGE_HEAD` is present unless

`MERGE_AUTOSTASH` is also present in which case git merge `--abort` applies the stash entry

to the worktree whereas git reset `--merge` will save the stashed changes in the stash

list.

`--quit`

Forget about the current merge in progress. Leave the index and the working tree as-is.

If `MERGE_AUTOSTASH` is present, the stash entry will be saved to the stash list.

`--continue`

After a git merge stops due to conflicts you can conclude the merge by running git merge

`--continue` (see "HOW TO RESOLVE CONFLICTS" section below).

`<commit>...`

Commits, usually other branch heads, to merge into our branch. Specifying more than one commit will create a merge with more than two parents (affectionately called an Octopus merge).

If no commit is given from the command line, merge the remote-tracking branches that the current branch is configured to use as its upstream. See also the configuration section of this manual page.

When `FETCH_HEAD` (and no other commit) is specified, the branches recorded in the `.git/FETCH_HEAD` file by the previous invocation of `git fetch` for merging are merged to the current branch.

PRE-MERGE CHECKS

Before applying outside changes, you should get your own work in good shape and committed locally, so it will not be clobbered if there are conflicts. See also `git-stash(1)`. `git pull` and `git merge` will stop without doing anything when local uncommitted changes overlap with files that `git pull/git merge` may need to update.

To avoid recording unrelated changes in the merge commit, `git pull` and `git merge` will also abort if there are any changes registered in the index relative to the `HEAD` commit. (Special narrow exceptions to this rule may exist depending on which merge strategy is in use, but generally, the index must match `HEAD`.)

If all named commits are already ancestors of `HEAD`, `git merge` will exit early with the message "Already up to date."

FAST-FORWARD MERGE

Often the current branch head is an ancestor of the named commit. This is the most common case especially when invoked from `git pull`: you are tracking an upstream repository, you have committed no local changes, and now you want to update to a newer upstream revision. In this case, a new commit is not needed to store the combined history; instead, the `HEAD` (along with the index) is updated to point at the named commit, without creating an extra merge commit.

This behavior can be suppressed with the `--no-ff` option.

TRUE MERGE

Except in a fast-forward merge (see above), the branches to be merged must be tied together by a merge commit that has both of them as its parents.

A merged version reconciling the changes from all branches to be merged is committed, and your `HEAD`, index, and working tree are updated to it. It is possible to have modifications in the working tree as long as they do not overlap; the update will preserve them.

When it is not obvious how to reconcile the changes, the following happens:

1. The `HEAD` pointer stays the same.

2. The MERGE_HEAD ref is set to point to the other branch head.
3. Paths that merged cleanly are updated both in the index file and in your working tree.
4. For conflicting paths, the index file records up to three versions: stage 1 stores the version from the common ancestor, stage 2 from HEAD, and stage 3 from MERGE_HEAD (you can inspect the stages with `git ls-files -u`). The working tree files contain the result of the merge operation; i.e. 3-way merge results with familiar conflict markers `<<< ===>>>`.
5. A ref named AUTO_MERGE is written, pointing to a tree corresponding to the current content of the working tree (including conflict markers for textual conflicts). Note that this ref is only written when the ort merge strategy is used (the default).
6. No other changes are made. In particular, the local modifications you had before you started merge will stay the same and the index entries for them stay as they were, i.e. matching HEAD.

If you tried a merge which resulted in complex conflicts and want to start over, you can recover with `git merge --abort`.

MERGING TAG

When merging an annotated (and possibly signed) tag, Git always creates a merge commit even if a fast-forward merge is possible, and the commit message template is prepared with the tag message. Additionally, if the tag is signed, the signature check is reported as a comment in the message template. See also `git-tag(1)`.

When you want to just integrate with the work leading to the commit that happens to be tagged, e.g. synchronizing with an upstream release point, you may not want to make an unnecessary merge commit.

In such a case, you can "unwrap" the tag yourself before feeding it to `git merge`, or pass `--ff-only` when you do not have any work on your own. e.g.

```
git fetch origin
```

```
git merge v1.2.3^0
```

```
git merge --ff-only v1.2.3
```

HOW CONFLICTS ARE PRESENTED

During a merge, the working tree files are updated to reflect the result of the merge. Among the changes made to the common ancestor's version, non-overlapping ones (that is, you changed an area of the file while the other side left that area intact, or vice versa) are incorporated in the final result verbatim. When both sides made changes to the same area,

however, Git cannot randomly pick one side over the other, and asks you to resolve it by leaving what both sides did to that area.

By default, Git uses the same style as the one used by the "merge" program from the RCS suite to present such a conflicted hunk, like this:

```
Here are lines that are either unchanged from the common
ancestor, or cleanly resolved because only one side changed,
or cleanly resolved because both sides changed the same way.
```

```
<<<<<<< yours:sample.txt
```

```
Conflict resolution is hard;
```

```
let's go shopping.
```

```
=====
```

```
Git makes conflict resolution easy.
```

```
>>>>>>> theirs:sample.txt
```

```
And here is another line that is cleanly resolved or unmodified.
```

The area where a pair of conflicting changes happened is marked with markers <<<<<<<, =====, and >>>>>>>. The part before the ===== is typically your side, and the part afterwards is typically their side.

The default format does not show what the original said in the conflicting area. You cannot tell how many lines are deleted and replaced with Barbie?s remark on your side. The only thing you can tell is that your side wants to say it is hard and you?d prefer to go shopping, while the other side wants to claim it is easy.

An alternative style can be used by setting the merge.conflictStyle configuration variable to either diff3 or zdiff3. In diff3 style, the above conflict may look like this:

```
Here are lines that are either unchanged from the common
ancestor, or cleanly resolved because only one side changed,
```

```
<<<<<<< yours:sample.txt
```

```
or cleanly resolved because both sides changed the same way.
```

```
Conflict resolution is hard;
```

```
let's go shopping.
```

```
||||||| base:sample.txt
```

```
or cleanly resolved because both sides changed identically.
```

```
Conflict resolution is hard.
```

```
=====
```

or cleanly resolved because both sides changed the same way.

Git makes conflict resolution easy.

```
>>>>>> theirs:sample.txt
```

And here is another line that is cleanly resolved or unmodified.

while in zdiff3 style, it may look like this:

Here are lines that are either unchanged from the common ancestor, or cleanly resolved because only one side changed, or cleanly resolved because both sides changed the same way.

```
<<<<<< yours:sample.txt
```

Conflict resolution is hard;

let's go shopping.

```
||||||| base:sample.txt
```

or cleanly resolved because both sides changed identically.

Conflict resolution is hard.

```
=====
```

Git makes conflict resolution easy.

```
>>>>>> theirs:sample.txt
```

And here is another line that is cleanly resolved or unmodified.

In addition to the <<<<<<, =====, and >>>>>> markers, it uses another ||||| marker that is followed by the original text. You can tell that the original just stated a fact, and your side simply gave in to that statement and gave up, while the other side tried to have a more positive attitude. You can sometimes come up with a better resolution by viewing the original.

HOW TO RESOLVE CONFLICTS

After seeing a conflict, you can do two things:

- ? Decide not to merge. The only clean-ups you need are to reset the index file to the HEAD commit to reverse 2. and to clean up working tree changes made by 2. and 3.; git merge --abort can be used for this.
- ? Resolve the conflicts. Git will mark the conflicts in the working tree. Edit the files into shape and git add them to the index. Use git commit or git merge --continue to seal the deal. The latter command checks whether there is a (interrupted) merge in progress before calling git commit.

You can work through the conflict with a number of tools:

- ? Use a mergetool. `git mergetool` to launch a graphical mergetool which will work through the merge with you.
- ? Look at the diffs. `git diff` will show a three-way diff, highlighting changes from both the HEAD and MERGE_HEAD versions. `git diff AUTO_MERGE` will show what changes you've made so far to resolve textual conflicts.
- ? Look at the diffs from each branch. `git log --merge -p <path>` will show diffs first for the HEAD version and then the MERGE_HEAD version.
- ? Look at the originals. `git show :1:filename` shows the common ancestor, `git show :2:filename` shows the HEAD version, and `git show :3:filename` shows the MERGE_HEAD version.

EXAMPLES

- ? Merge branches fixes and enhancements on top of the current branch, making an octopus merge:

```
$ git merge fixes enhancements
```
- ? Merge branch obsolete into the current branch, using ours merge strategy:

```
$ git merge -s ours obsolete
```
- ? Merge branch maint into the current branch, but do not make a new commit automatically:

```
$ git merge --no-commit maint
```

This can be used when you want to include further changes to the merge, or want to write your own merge commit message.

You should refrain from abusing this option to sneak substantial changes into a merge commit. Small fixups like bumping release/version name would be acceptable.

MERGE STRATEGIES

The merge mechanism (`git merge` and `git pull` commands) allows the backend merge strategies to be chosen with `-s` option. Some strategies can also take their own options, which can be passed by giving `-X<option>` arguments to `git merge` and/or `git pull`.

`ort`

This is the default merge strategy when pulling or merging one branch. This strategy can only resolve two heads using a 3-way merge algorithm. When there is more than one common ancestor that can be used for 3-way merge, it creates a merged tree of the common ancestors and uses that as the reference tree for the 3-way merge. This has been reported to result in fewer merge conflicts without causing mismerges by tests done on actual merge commits taken from Linux 2.6 kernel development history. Additionally this

strategy can detect and handle merges involving renames. It does not make use of detected copies. The name for this algorithm is an acronym ("Ostensibly Recursive?s Twin") and came from the fact that it was written as a replacement for the previous default algorithm, recursive.

In the case where the path is a submodule, if the submodule commit used on one side of the merge is a descendant of the submodule commit used on the other side of the merge, Git attempts to fast-forward to the descendant. Otherwise, Git will treat this case as a conflict, suggesting as a resolution a submodule commit that is descendant of the conflicting ones, if one exists.

The ort strategy can take the following options:

ours

This option forces conflicting hunks to be auto-resolved cleanly by favoring our version. Changes from the other tree that do not conflict with our side are reflected in the merge result. For a binary file, the entire contents are taken from our side.

This should not be confused with the ours merge strategy, which does not even look at what the other tree contains at all. It discards everything the other tree did, declaring our history contains all that happened in it.

theirs

This is the opposite of ours; note that, unlike ours, there is no theirs merge strategy to confuse this merge option with.

ignore-space-change, ignore-all-space, ignore-space-at-eol, ignore-cr-at-eol

Treats lines with the indicated type of whitespace change as unchanged for the sake of a three-way merge. Whitespace changes mixed with other changes to a line are not ignored. See also `git-diff(1)` `-b`, `-w`, `--ignore-space-at-eol`, and `--ignore-cr-at-eol`.

? If their version only introduces whitespace changes to a line, our version is used;

? If our version introduces whitespace changes but their version includes a substantial change, their version is used;

? Otherwise, the merge proceeds in the usual way.

renormalize

This runs a virtual check-out and check-in of all three stages of any file which needs a three-way merge. This option is meant to be used when merging branches with

different clean filters or end-of-line normalization rules. See "Merging branches with differing checkin/checkout attributes" in gitattributes(5) for details.

no-renormalize

Disables the renormalize option. This overrides the merge.renormalize configuration variable.

find-renames[=<n>]

Turn on rename detection, optionally setting the similarity threshold. This is the default. This overrides the merge.renames configuration variable. See also git-diff(1) --find-renames.

rename-threshold=<n>

Deprecated synonym for find-renames=<n>.

no-renames

Turn off rename detection. This overrides the merge.renames configuration variable. See also git-diff(1) --no-renames.

histogram

Deprecated synonym for diff-algorithm=histogram.

patience

Deprecated synonym for diff-algorithm=patience.

diff-algorithm=(histogram|minimal|myers|patience)

Use a different diff algorithm while merging, which can help avoid mismerges that occur due to unimportant matching lines (such as braces from distinct functions).

See also git-diff(1) --diff-algorithm. Note that ort defaults to

diff-algorithm=histogram, while regular diffs currently default to the diff.algorithm config setting.

subtree[=<path>]

This option is a more advanced form of subtree strategy, where the strategy makes a guess on how two trees must be shifted to match with each other when merging.

Instead, the specified path is prefixed (or stripped from the beginning) to make the shape of two trees to match.

recursive

This is now a synonym for ort. It was an alternative implementation until v2.49.0, but was redirected to mean ort in v2.50.0. The previous recursive strategy was the default strategy for resolving two heads from Git v0.99.9k until v2.33.0.

resolve

This can only resolve two heads (i.e. the current branch and another branch you pulled from) using a 3-way merge algorithm. It tries to carefully detect criss-cross merge ambiguities. It does not handle renames.

octopus

This resolves cases with more than two heads, but refuses to do a complex merge that needs manual resolution. It is primarily meant to be used for bundling topic branch heads together. This is the default merge strategy when pulling or merging more than one branch.

ours

This resolves any number of heads, but the resulting tree of the merge is always that of the current branch head, effectively ignoring all changes from all other branches. It is meant to be used to supersede old development history of side branches. Note that this is different from the `-Xours` option to the `ort` merge strategy.

subtree

This is a modified `ort` strategy. When merging trees A and B, if B corresponds to a subtree of A, B is first adjusted to match the tree structure of A, instead of reading the trees at the same level. This adjustment is also done to the common ancestor tree.

With the strategies that use 3-way merge (including the default, `ort`), if a change is made on both branches, but later reverted on one of the branches, that change will be present in the merged result; some people find this behavior confusing. It occurs because only the heads and the merge base are considered when performing a merge, not the individual commits. The merge algorithm therefore considers the reverted change as no change at all, and substitutes the changed version instead.

CONFIGURATION

`branch.<name>.mergeOptions`

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

Everything above this line in this section isn't included from the `git-config(1)` documentation. The content that follows is the same as what's found there:

`merge.conflictStyle`

Specify the style in which conflicted hunks are written out to working tree files upon

merge. The default is "merge", which shows a <<<<<< conflict marker, changes made by one side, a ===== marker, changes made by the other side, and then a >>>>>> marker. An alternate style, "diff3", adds a ||||| marker and the original text before the ===== marker. The "merge" style tends to produce smaller conflict regions than diff3, both because of the exclusion of the original text, and because when a subset of lines match on the two sides, they are just pulled out of the conflict region. Another alternate style, "zdiff3", is similar to diff3 but removes matching lines on the two sides from the conflict region when those matching lines appear near either the beginning or end of a conflict region.

`merge.defaultToUpstream`

If merge is called without any commit argument, merge the upstream branches configured for the current branch by using their last observed values stored in their remote-tracking branches. The values of the branch.<current branch>.merge that name the branches at the remote named by branch.<current-branch>.remote are consulted, and then they are mapped via remote.<remote>.fetch to their corresponding remote-tracking branches, and the tips of these tracking branches are merged. Defaults to true.

`merge.ff`

By default, Git does not create an extra merge commit when merging a commit that is a descendant of the current commit. Instead, the tip of the current branch is fast-forwarded. When set to false, this variable tells Git to create an extra merge commit in such a case (equivalent to giving the --no-ff option from the command line). When set to only, only such fast-forward merges are allowed (equivalent to giving the --ff-only option from the command line).

`merge.verifySignatures`

If true, this is equivalent to the --verify-signatures command line option. See git-merge(1) for details.

`merge.branchdesc`

In addition to branch names, populate the log message with the branch description text associated with them. Defaults to false.

`merge.log`

In addition to branch names, populate the log message with at most the specified number of one-line descriptions from the actual commits that are being merged. Defaults to false, and true is a synonym for 20.

merge.suppressDest

By adding a glob that matches the names of integration branches to this multi-valued configuration variable, the default merge message computed for merges into these integration branches will omit "into <branch-name>" from its title.

An element with an empty value can be used to clear the list of globs accumulated from previous configuration entries. When there is no merge.suppressDest variable defined, the default value of master is used for backward compatibility.

merge.renameLimit

The number of files to consider in the exhaustive portion of rename detection during a merge. If not specified, defaults to the value of diff.renameLimit. If neither merge.renameLimit nor diff.renameLimit are specified, currently defaults to 7000. This setting has no effect if rename detection is turned off.

merge.renames

Whether Git detects renames. If set to false, rename detection is disabled. If set to true, basic rename detection is enabled. Defaults to the value of diff.renames.

merge.directoryRenames

Whether Git detects directory renames, affecting what happens at merge time to new files added to a directory on one side of history when that directory was renamed on the other side of history. Possible values are:

false

Directory rename detection is disabled, meaning that such new files will be left behind in the old directory.

true

Directory rename detection is enabled, meaning that such new files will be moved into the new directory.

conflict

A conflict will be reported for such paths.

If merge.renames is false, merge.directoryRenames is ignored and treated as false. Defaults to conflict.

merge.renormalize

Tell Git that canonical representation of files in the repository has changed over time (e.g. earlier commits record text files with CRLF line endings, but recent ones use LF line endings). In such a repository, for each file where a three-way content merge is

needed, Git can convert the data recorded in commits to a canonical form before performing a merge to reduce unnecessary conflicts. For more information, see section "Merging branches with differing checkin/checkout attributes" in `gitattributes(5)`.

`merge.stat`

What, if anything, to print between `ORIG_HEAD` and the merge result at the end of the merge. Possible values are:

`false`

Show nothing.

`true`

Show `git diff --diffstat --summary ORIG_HEAD`.

`compact`

Show `git diff --compact-summary ORIG_HEAD`.

but any unrecognised value (e.g., a value added by a future version of Git) is taken as `true` instead of triggering an error. Defaults to `true`.

`merge.autoStash`

When set to `true`, automatically create a temporary stash entry before the operation begins, and apply it after the operation ends. This means that you can run merge on a dirty worktree. However, use with care: the final stash application after a successful merge might result in non-trivial conflicts. This option can be overridden by the `--no-autostash` and `--autostash` options of `git-merge(1)`. Defaults to `false`.

`merge.tool`

Controls which merge tool is used by `git-mergetool(1)`. The list below shows the valid built-in values. Any other value is treated as a custom merge tool and requires that a corresponding `mergetool.<tool>.cmd` variable is defined.

`merge.guitool`

Controls which merge tool is used by `git-mergetool(1)` when the `-g/--gui` flag is specified. The list below shows the valid built-in values. Any other value is treated as a custom merge tool and requires that a corresponding `mergetool.<guitool>.cmd` variable is defined.

`araxis`

Use Araxis Merge (requires a graphical session)

`bc`

Use Beyond Compare (requires a graphical session)

bc3

Use Beyond Compare (requires a graphical session)

bc4

Use Beyond Compare (requires a graphical session)

codecompare

Use Code Compare (requires a graphical session)

deltawalker

Use DeltaWalker (requires a graphical session)

diffmerge

Use DiffMerge (requires a graphical session)

diffuse

Use Diffuse (requires a graphical session)

ecmerge

Use ECMerge (requires a graphical session)

emerge

Use Emacs' Emerge

examdiff

Use ExamDiff Pro (requires a graphical session)

guiffy

Use Guiffy's Diff Tool (requires a graphical session)

gvimdiff

Use gVim (requires a graphical session) with a custom layout (see git help mergetool's BACKEND SPECIFIC HINTS section)

gvimdiff1

Use gVim (requires a graphical session) with a 2 panes layout (LOCAL and REMOTE)

gvimdiff2

Use gVim (requires a graphical session) with a 3 panes layout (LOCAL, MERGED and REMOTE)

gvimdiff3

Use gVim (requires a graphical session) where only the MERGED file is shown

kdiff3

Use KDiff3 (requires a graphical session)

meld

Use Meld (requires a graphical session) with optional auto merge (see git help mergetool's CONFIGURATION section)

nvimdiff

Use Neovim with a custom layout (see git help mergetool's BACKEND SPECIFIC HINTS section)

nvimdiff1

Use Neovim with a 2 panes layout (LOCAL and REMOTE)

nvimdiff2

Use Neovim with a 3 panes layout (LOCAL, MERGED and REMOTE)

nvimdiff3

Use Neovim where only the MERGED file is shown

opendiff

Use FileMerge (requires a graphical session)

p4merge

Use HelixCore P4Merge (requires a graphical session)

smerge

Use Sublime Merge (requires a graphical session)

tkdiff

Use TkDiff (requires a graphical session)

tortoisemerge

Use TortoiseMerge (requires a graphical session)

vimdiff

Use Vim with a custom layout (see git help mergetool's BACKEND SPECIFIC HINTS section)

vimdiff1

Use Vim with a 2 panes layout (LOCAL and REMOTE)

vimdiff2

Use Vim with a 3 panes layout (LOCAL, MERGED and REMOTE)

vimdiff3

Use Vim where only the MERGED file is shown

vscode

Use Visual Studio Code (requires a graphical session)

winmerge

Use WinMerge (requires a graphical session)

xxdiff

Use xxdiff (requires a graphical session)

merge.verbosity

Controls the amount of output shown by the recursive merge strategy. Level 0 outputs nothing except a final error message if conflicts were detected. Level 1 outputs only conflicts, 2 outputs conflicts and file changes. Level 5 and above outputs debugging information. The default is level 2. Can be overridden by the GIT_MERGE_VERBOSITY environment variable.

merge.<driver>.name

Defines a human-readable name for a custom low-level merge driver. See gitattributes(5) for details.

merge.<driver>.driver

Defines the command that implements a custom low-level merge driver. See gitattributes(5) for details.

merge.<driver>.recursive

Names a low-level merge driver to be used when performing an internal merge between common ancestors. See gitattributes(5) for details.

SEE ALSO

git-fmt-merge-msg(1), git-pull(1), gitattributes(5), git-reset(1), git-diff(1), git-ls-files(1), git-add(1), git-rm(1), git-mergetool(1)

GIT

Part of the git(1) suite

Git 2.53.0

03/02/2026

GIT-MERGE(1)