

**NAME**

git-merge - Join two or more development histories together

**SYNOPSIS**

```
git merge [-n] [--stat] [--no-commit] [--squash] [--no-edit]
[-s <strategy>] [-X <strategy-option>] [-S[<key-id>]]
[--no-rerere-autoupdate] [-m <msg>] [<commit>...]
git merge <msg> HEAD <commit>...
git merge --abort
```

**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.

```
A---B---C topic
/
D---E---F---G---H master
```

The second syntax (`<msg> HEAD <commit>...`) is supported for historical reasons. Do not use it from the command line or in new scripts. It is the same as `git merge -m <msg> <commit>...`

The third syntax (`git merge --abort`) can only be run after the merge has resulted in conflicts. *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 but pretend the merge failed and do not autocommit, to give the user a chance to inspect and further tweak the merge result before committing.

--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.

- ff**  
When the merge resolves as a fast-forward, only update the branch pointer, without creating a merge commit. This is the default behavior.
- no-ff**  
Create a merge commit even when the merge resolves as a fast-forward. This is the default behaviour when merging an annotated (and possibly signed) tag.
- ff-only**  
Refuse to merge and exit with a non-zero status unless the current HEAD is already up-to-date or the merge can be resolved as a fast-forward.
- 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.
- 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.
- 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**.
- 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 (*git merge-recursive* when merging a single head, *git merge-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 commits being merged have good and trusted GPG signatures and abort the merge in case they do not.
- 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.
- S[<keyid>], --gpg-sign[=<keyid>]**  
GPG-sign the resulting merge commit.
- 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.

`--[no-]rerere-autoupdate`

Allow the rerere mechanism to update the index with the result of auto-conflict resolution if possible.

`--abort`

Abort the current conflict resolution process, and try to reconstruct the pre-merge state.

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.

`<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.

## 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. (One exception is when the changed index entries are in the state that would result from the merge already.)

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 program; i.e. 3-way merge results with familiar conflict markers <<<===>>>.
5. 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.

```
<<<<<<< yours:sample.txt
Conflict resolution is hard;
lets 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 `diff3`.

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
Conflict resolution is hard;
lets go shopping.
|||||
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* to seal the deal.

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

- Use a mergetool. `git mergetool` to launch a graphical mergetool which will work you through the merge.
- Look at the diffs. `git diff` will show a three-way diff, highlighting changes from both the HEAD and MERGE\_HEAD versions.
- 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`.

`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 and is considered generally safe and fast.

`recursive`

This 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 can detect and handle merges involving renames. This is the default merge strategy when pulling or merging one branch.

The *recursive* 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 to 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*.

`patience`

With this option, *merge-recursive* spends a little extra time to avoid mismerges that sometimes occur due to unimportant matching lines (e.g., braces from distinct functions). Use this when the branches to be merged have diverged wildly. See also [git-diff\(1\)--patience](#).

`diff-algorithm=[patience|minimal|histogram|myers]`

Tells *merge-recursive* to use a different diff algorithm, 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](#).

`ignore-space-change`, `ignore-all-space`, `ignore-space-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`, and `--ignore-space-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 a file when resolving 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.

`rename-threshold=<n>`

Controls the similarity threshold used for rename detection. See also [git-diff\(1\)-M](#).

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.

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 *recursive* merge strategy.

subtree

This is a modified recursive 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, *recursive*), 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

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.

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.

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.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.renameLimit

The number of files to consider when performing rename detection during a merge; if not specified, defaults to the value of `diff.renameLimit`.

**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, 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**

Whether to print the diffstat between ORIG\_HEAD and the merge result at the end of the merge. True by default.

**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.

- araxis
- bc3
- codecompare
- deltawalker
- diffmerge
- diffuse
- ecmmerge
- emerge
- gvimdiff
- gvimdiff2
- gvimdiff3
- kdiff3
- meld
- opendiff
- p4merge
- tkdiff
- tortoisemerge
- vimdiff
- vimdiff2
- vimdiff3
- xxdiff

**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.

**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.

**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