NAME

git-merge-tree - Perform merge without touching index or working tree

SYNOPSIS

git merge-tree [--write-tree] [<options>] <branch1> <branch2>
git merge-tree [--trivial-merge] <base-tree> <branch1> <branch2> (deprecated)

DESCRIPTION

This command has a modern **--write-tree** mode and a deprecated **--trivial-merge** mode. With the exception of the DEPRECATED DESCRIPTION section at the end, the rest of this documentation describes the modern **--write-tree** mode.

Performs a merge, but does not make any new commits and does not read from or write to either the working tree or index.

The performed merge will use the same features as the "real" **git-merge**(1), including:

- way content merges of individual files
- detection
- directory/file conflict handling
- ancestor consolidation (i.e. when there is more than one merge base, creating a virtual merge base by merging the merge bases)

After the merge completes, a new toplevel tree object is created. See **OUTPUT** below for details.

OPTIONS

-z

Do not quote filenames in the <Conflicted file info> section, and end each filename with a NUL character rather than newline. Also begin the messages section with a NUL character instead of a newline. See the section called "OUTPUT" below for more information.

--name-only
: In the Conflicted file info section, instead of writing a list of (mode, oid, stage, path) tuples to output for conflicted files, just provide a list of filenames with conflicts (and do not list filenames multiple times if they have multiple conflicting stages).

--[no-]messages
: Write any informational messages such as "Auto-merging <path>" or CONFLICT notices to the end of stdout. If unspecified, the default is to include these messages if there are merge conflicts, and to omit them otherwise.

--allow-unrelated-histories
: merge-tree will by default error out if the two branches specified share no common history. This flag can be given to override that check and make the merge proceed anyway.

--merge-base=<tree-ish>
: Instead of finding the merge-bases for <branch1> and <branch2>, specify a merge-base for the merge, and specifying multiple bases is currently not supported. This option is incompatible with **--stdin**.

  As the merge-base is provided directly, <branch1> and <branch2> do not need to specify commits; trees are enough.

OUTPUT

For a successful merge, the output from git-merge-tree is simply one line:

```
<OID of toplevel tree>
```

Whereas for a conflicted merge, the output is by default of the form:

```
<OID of toplevel tree>
<Conflicted file info>
<Informational messages>
```

These are discussed individually below.

However, there is an exception. If **--stdin** is passed, then there is an extra section at the beginning, a NUL character at the end, and then all the sections repeat for each line of input. Thus, if the first merge is conflicted and the second is clean, the output would be of the form:

```
<Merge status>
```

<OID of toplevel tree>
<Conflicted file info>
<Informational messages>
NUL
<Merge status>
<OID of toplevel tree>
NUL

Merge status

This is an integer status followed by a NUL character. The integer status is:

0: merge had conflicts
1: merge was clean
<0: something prevented the merge from running (e.g. access to repository objects denied by filesystem)

OID of toplevel tree

This is a tree object that represents what would be checked out in the working tree at the end of **git merge**. If there were conflicts, then files within this tree may have embedded conflict markers. This section is always followed by a newline (or NUL if **-z** is passed).

Conflicted file info

This is a sequence of lines with the format

<mode> <object> <stage> <filename>

The filename will be quoted as explained for the configuration variable **core.quotePath** (see **git-config**(1)). However, if the **--name-only** option is passed, the mode, object, and stage will be omitted. If **-z** is passed, the "lines" are terminated by a NUL character instead of a newline character.

Informational messages

This section provides informational messages, typically about conflicts. The format of the section varies significantly depending on whether **-z** is passed.

If **-z** is passed:

The output format is zero or more conflict informational records, each of the form:

<list-of-paths><conflict-type>NUL<conflict-message>NUL

where <list-of-paths> is of the form

<number-of-paths>NUL<path1>NUL<path2>NUL...<pathN>NUL

and includes paths (or branch names) affected by the conflict or informational message in <conflict-message>. Also, <conflict-type> is a stable string explaining the type of conflict, such as

•

•
(rename/delete)"

•
(submodule lacks merge base)"

•
(binary)"

and <conflict-message> is a more detailed message about the conflict which often (but not always) embeds the <stable-short-type-description> within it. These strings may change in future Git versions. Some examples:

•
<file>"

•
(rename/delete): <oldfile> renamed...but deleted in..."

•
to merge submodule <submodule> (no merge base)"

•
cannot merge binary files: <filename>"

If **-z** is NOT passed:

This section starts with a blank line to separate it from the previous sections, and then only contains the <conflict-message> information from the previous section (separated by newlines). These are non-stable strings that should not be parsed by scripts, and are just meant for human consumption. Also, note that while <conflict-message> strings usually do not contain embedded newlines, they

sometimes do. (However, the free-form messages will never have an embedded NUL character). So, the entire block of information is meant for human readers as an agglomeration of all conflict messages.

EXIT STATUS

For a successful, non-conflicted merge, the exit status is 0. When the merge has conflicts, the exit status is 1. If the merge is not able to complete (or start) due to some kind of error, the exit status is something other than 0 or 1 (and the output is unspecified). When --stdin is passed, the return status is 0 for both successful and conflicted merges, and something other than 0 or 1 if it cannot complete all the requested merges.

USAGE NOTES

This command is intended as low-level plumbing, similar to **git-hash-object**(1), **git-mktree**(1), **git-commit-tree**(1), **git-write-tree**(1), **git-update-ref**(1), and **git-mktag**(1). Thus, it can be used as a part of a series of steps such as:

```
NEWTREE=$(git merge-tree --write-tree $BRANCH1 $BRANCH2)
test $? -eq 0 || die "There were conflicts..."
NEWCOMMIT=$(git commit-tree $NEWTREE -p $BRANCH1 -p $BRANCH2)
git update-ref $BRANCH1 $NEWCOMMIT
```

Note that when the exit status is non-zero, **NEWTREE** in this sequence will contain a lot more output than just a tree.

For conflicts, the output includes the same information that you'd get with **git-merge**(1):

- would be written to the working tree (the OID of toplevel tree)
- higher order stages that would be written to the index (the Conflicted file info)
- messages that would have been printed to stdout (the Informational messages)

INPUT FORMAT

git merge-tree --stdin input format is fully text based. Each line has this format:

[<base-commit> --]<branch1> <branch2>

If one line is separated by --, the string before the separator is used for specifying a merge-base for the merge and the string after the separator describes the branches to be merged.

MISTAKES TO AVOID

Do NOT look through the resulting toplevel tree to try to find which files conflict; parse the Conflicted file info section instead. Not only would parsing an entire tree be horrendously slow in large repositories, there are numerous types of conflicts not representable by conflict markers (modify/delete, mode conflict, binary file changed on both sides, file/directory conflicts, various rename conflict permutations, etc.)

Do NOT interpret an empty Conflicted file info list as a clean merge; check the exit status. A merge can have conflicts without having individual files conflict (there are a few types of directory rename conflicts that fall into this category, and others might also be added in the future).

Do NOT attempt to guess or make the user guess the conflict types from the Conflicted file info list. The information there is insufficient to do so. For example: Rename/rename(1to2) conflicts (both sides renamed the same file differently) will result in three different files having higher order stages (but each only has one higher order stage), with no way (short of the Informational messages section) to determine which three files are related. File/directory conflicts also result in a file with exactly one higher order stage. Possibly-involved-in-directory-rename conflicts (when "merge.directoryRenames" is unset or set to "conflicts") also result in a file with exactly one higher order stage. In all cases, the Informational messages section has the necessary info, though it is not designed to be machine parseable.

Do NOT assume that each path from Conflicted file info, and the logical conflicts in the Informational messages have a one-to-one mapping, nor that there is a one-to-many mapping, nor a many-to-one mapping. Many-to-many mappings exist, meaning that each path can have many logical conflict types in a single merge, and each logical conflict type can affect many paths.

Do NOT assume all filenames listed in the Informational messages section had conflicts. Messages can be included for files that have no conflicts, such as "Auto-merging <file>".

AVOID taking the OIDS from the Conflicted file info and re-merging them to present the conflicts to the user. This will lose information. Instead, look up the version of the file found within the OID of toplevel tree and show that instead. In particular, the latter will have conflict markers annotated with the original branch/commit being merged and, if renames were involved, the original filename. While you could include the original branch/commit in the conflict marker annotations when re-merging, the original filename is not available from the Conflicted file info and thus you would be losing

information that might help the user resolve the conflict.

DEPRECATED DESCRIPTION

Per the DESCRIPTION and unlike the rest of this documentation, this section describes the deprecated **--trivial-merge** mode.

Other than the optional **--trivial-merge**, this mode accepts no options.

This mode reads three tree-ish, and outputs trivial merge results and conflicting stages to the standard output in a semi-diff format. Since this was designed for higher level scripts to consume and merge the results back into the index, it omits entries that match <branch1>. The result of this second form is similar to what three-way *git read-tree -m* does, but instead of storing the results in the index, the command outputs the entries to the standard output.

This form not only has limited applicability (a trivial merge cannot handle content merges of individual files, rename detection, proper directory/file conflict handling, etc.), the output format is also difficult to work with, and it will generally be less performant than the first form even on successful merges (especially if working in large repositories).

GIT

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