git-diff

git-diff(1) ==========

NAME ---- git-diff - Show changes between commits, commit and working tree, etc

SYNOPSIS --------

git diff [<options>] [<commit>] [--] [<path>…] git diff [<options>] --cached [--merge-base] [<commit>] [--] [<path>…] git diff [<options>] [--merge-base] <commit> [<commit>…] <commit> [--] [<path>…] git diff [<options>] <commit>…<commit> [--] [<path>…] git diff [<options>] <blob> <blob> git diff [<options>] --no-index [--] <path> <path> [<pathspec>…]

DESCRIPTION ----------- Show changes between the working tree and the index or a tree, changes between the index and a tree, changes between two trees, changes resulting from a merge, changes between two blob objects, or changes between two files on disk.

`git` `diff` `[<options>]` `[--]` `[<path>…]`

```
This form is to view the changes you made relative to
the index (staging area for the next commit).  In other
words, the differences are what you _could_ tell Git to
further add to the index but you still haven't.  You can
stage these changes by using linkgit:git-add[1].
```

`git` `diff` `[<options>]` `--no-index` `[--]` `<path>` `<path>` `[<pathspec>…]`

```
This form is to compare the given two paths on the
filesystem.  You can omit the `--no-index` option when
running the command in a working tree controlled by Git and
at least one of the paths points outside the working tree,
or when running the command outside a working tree
controlled by Git. This form implies `--exit-code`. If both
paths point to directories, additional pathspecs may be
provided. These will limit the files included in the
difference. All such pathspecs must be relative as they
apply to both sides of the diff.
```

`git` `diff` `[<options>]` `--cached` `[--merge-base]` `[<commit>]` `[--]` `[<path>…]`

```
This form is to view the changes you staged for the next
commit relative to the named _<commit>_.  Typically you
would want comparison with the latest commit, so if you
do not give _<commit>_, it defaults to `HEAD`.
If `HEAD` does not exist (e.g. unborn branches) and
_<commit>_ is not given, it shows all staged changes.
`--staged` is a synonym of `--cached`.
```

If `--merge-base` is given, instead of using *<commit>*, use the merge base of *<commit>* and `HEAD`. `git diff --cached --merge-base A` is equivalent to `git diff --cached $(git merge-base A HEAD)`.

`git diff [<options>] [--merge-base] <commit> [--] [<path>…]`

```
This form is to view the changes you have in your
working tree relative to the named _<commit>_.  You can
use `HEAD` to compare it with the latest commit, or a
branch name to compare with the tip of a different
branch.
```

If `--merge-base` is given, instead of using *<commit>*, use the merge base of *<commit>* and `HEAD`. `git diff --merge-base A` is equivalent to `git diff $(git merge-base A HEAD)`.

`git diff [<options>] [--merge-base] <commit> <commit> [--] [<path>…]`

```
This is to view the changes between two arbitrary
_<commit>_.
```

If `--merge-base` is given, use the merge base of the two commits for the "before" side. `git diff --merge-base A B` is equivalent to `git diff $(git merge-base A B) B`.

`git diff [<options>] <commit> <commit>…<commit> [--] [<path>…]`

```
This form is to view the results of a merge commit.  The first
listed _<commit>_ must be the merge itself; the remaining two or
more commits should be its parents.  Convenient ways to produce
the desired set of revisions are to use the suffixes `@` and
`^!`.  If `A` is a merge commit, then `git diff A A^@`,
`git diff A^!` and `git show A` all give the same combined diff.
```

`git` `diff` `[<options>]` `<commit>..<commit>` `[--]` `[<path>…]`

```
This is synonymous to the earlier form (without the `..`) for
viewing the changes between two arbitrary _<commit>_.  If _<commit>_ on
one side is omitted, it will have the same effect as
using `HEAD` instead.
```

`git` `diff` `[<options>]` `<commit>…<commit>` `[--]` `[<path>…]`

```
This form is to view the changes on the branch containing
and up to the second _<commit>_, starting at a common ancestor
of both _<commit>_.  `git diff A...B` is equivalent to
`git diff $(git merge-base A B) B`.  You can omit any one
of _<commit>_, which has the same effect as using `HEAD` instead.
```

Just in case you are doing something exotic, it should be noted that all of the *<commit>* in the above description, except in the `--merge-base` case and in the last two forms that use `..` notations, can be any *<tree>*. A tree of interest is the one pointed to by the ref named `AUTO_MERGE`, which is written by the `ort` merge strategy upon hitting merge conflicts (see linkgit:git-merge[1]). Comparing the working tree with `AUTO_MERGE` shows changes you've made so far to resolve textual conflicts (see the examples below).

For a more complete list of ways to spell *<commit>*, see "SPECIFYING REVISIONS" section in linkgit:gitrevisions[7]. However, `diff` is about comparing two *endpoints*, not ranges, and the range notations (`<commit>..<commit>` and `<commit>…<commit>`) do not mean a range as defined in the "SPECIFYING RANGES" section in linkgit:gitrevisions[7].

`git` `diff` `[<options>]` `<blob>` `<blob>`

```
This form is to view the differences between the raw
contents of two blob objects.
```

OPTIONS ------- :git-diff: 1 include::diff-options.adoc[]

`-1`

`--base`

`-2`

`--ours`

`-3`

`--theirs`
Compare the working tree with

- the "base" version (stage #1) when using `-1` or `--base`,
- "our branch" (stage #2) when using `-2` or `--ours`, or
- "their branch" (stage #3) when using `-3` or `--theirs`.

The index contains these stages only for unmerged entries i.e. while resolving conflicts. See linkgit:git-read-tree[1] section "3-Way Merge" for detailed information.

`-0`
Omit diff output for unmerged entries and just show "Unmerged". Can be used only when comparing the working tree with the index.

`<path>…`
The *<path>* parameters, when given, are used to limit the diff to the named paths (you can give directory names and get diff for all files under them).

Raw output format ----------

The raw output format from `git-diff-index`, `git-diff-tree`, `git-diff-files` and `git diff --raw` are very similar.

These commands all compare two sets of things; what is compared differs:

`git-diff-index <tree-ish>`
compares the *<tree-ish>* and the files on the filesystem.

`git-diff-index --cached <tree-ish>`
compares the *<tree-ish>* and the index.

`git-diff-tree [-r] <tree-ish-1> <tree-ish-2> [<pattern>…]`
compares the trees named by the two arguments.

`git-diff-files [<pattern>…]`
compares the index and the files on the filesystem.

All the commands print one output line per changed file, except `git diff-files` in the case of an unmerged file, which prints both an "unmerged" and an "in-place edit" line.

An output line is formatted this way:

```
in-place edit  :100644 100644 bcd1234 0123456 M file0
copy-edit      :100644 100644 abcd123 1234567 C68 file1 file2
rename-edit    :100644 100644 abcd123 1234567 R86 file1 file3
```

```
create         :000000 100644 0000000 1234567 A file4
delete         :100644 000000 1234567 0000000 D file5
unmerged       :000000 000000 0000000 0000000 U file6
```

That is, from the left to the right:

1. a colon.
2. mode for "src"; 000000 if creation, or if "src" is from the index and is unmerged.
3. a space.
4. mode for "dst"; 000000 if deletion, or if "dst" is from the index and is unmerged.
5. a space.
6. sha1 for "src"; 0{40} if creation, or if "src" is from the index and is unmerged.
7. a space.
8. sha1 for "dst"; 0{40} if deletion, if "dst" is from the index and is unmerged, or if "dst" is from the work tree and is out of sync with the index.
9. a space.
10. status, followed by optional "score" number.
11. a tab or a NUL when `-z` option is used.
12. path for "src"
13. a tab or a NUL when `-z` option is used; only exists for C or R.
14. path for "dst"; only exists for C or R.
15. an LF or a NUL when `-z` option is used, to terminate the record.

Possible status letters are:

- `A`: addition of a file
- `C`: copy of a file into a new one
- `D`: deletion of a file
- `M`: modification of the contents or mode of a file
- `R`: renaming of a file
- `T`: change in the type of the file (regular file, symbolic link or submodule)
- `U`: file is unmerged (you must complete the merge before it can be committed)
- `X`: "unknown" change type (most probably a bug, please report it)

Status letters `C` and `R` are always followed by a score (denoting the percentage of similarity between the source and target of the move or copy). Status letter `M` may be followed by a score (denoting the percentage of dissimilarity) for file rewrites.

The sha1 for "dst" is shown as all 0’s if a file on the filesystem is out of sync with the index.

Example:

```
:100644 100644 5be4a4a 0000000 M file.c
```

Without the `-z` option, pathnames with "unusual" characters are quoted as explained for the configuration variable `core.quotePath` (see linkgit:git-config[1]). Using `-z` the filename is output verbatim and the line is terminated by a NUL byte.

diff format for merges ----------

`git-diff-tree`, `git-diff-files` and `git-diff` `--raw` can take `-c` or `--cc` option to generate diff output also for merge commits. The output differs from the format described above in the following way:

1. there is a colon for each parent
2. there are more "src" modes and "src" sha1
3. status is concatenated status characters for each parent
4. no optional "score" number
5. tab-separated pathname(s) of the file

For `-c` and `--cc`, only the destination or final path is shown even if the file was renamed on any side of history. With `--combined-all-paths`, the name of the path in each parent is shown followed by the name of the path in the merge commit.

Examples for `-c` and `--cc` without `--combined-all-paths`:

```
::100644 100644 100644 fabadb8 cc95eb0 4866510 MM␣desc.c
::100755 100755 100755 52b7a2d 6d1ac04 d2ac7d7 RM␣bar.sh
::100644 100644 100644 e07d6c5 9042e82 ee91881 RR␣phooey.c
```

Examples when `--combined-all-paths` added to either `-c` or `--cc`:

```
::100644 100644 100644 fabadb8 cc95eb0 4866510 MM␣desc.c␣desc.c␣desc.c
::100755 100755 100755 52b7a2d 6d1ac04 d2ac7d7 RM␣foo.sh␣bar.sh␣bar.sh
::100644 100644 100644 e07d6c5 9042e82 ee91881 RR␣fooey.c␣fuey.c␣phooey.c
```

Note that ’combined diff’ lists only files which were modified from all parents.

Generating patch text with -p ----------

Running linkgit:git-diff[1], linkgit:git-log[1], linkgit:git-show[1], linkgit:git-diff-index[1], linkgit:git-diff-tree[1], or linkgit:git-diff-files[1] with the `-p` option produces patch text. You can customize the creation of patch text via the `GIT_EXTERNAL_DIFF` and the `GIT_DIFF_OPTS` environment variables (see linkgit:git[1]), and the `diff` attribute (see linkgit:gitattributes[5]).

What the `-p` option produces is slightly different from the traditional diff format:

1. It is preceded by a "git diff" header that looks like this:

```
diff --git a/file1 b/file2
```

The `a/` and `b/` filenames are the same unless rename/copy is involved. Especially, even for a creation or a deletion, `/dev/null` is *not* used in place of the `a/` or `b/` filenames.
When a rename/copy is involved, `file1` and `file2` show the name of the source file of the rename/copy and the name of the file that the rename/copy produces, respectively.

2. It is followed by one or more extended header lines:
   old mode <mode> new mode <mode> deleted file mode <mode> new file mode <mode> copy from <path> copy to <path> rename from <path> rename to <path> similarity index <number> dissimilarity index <number> index <hash>..<hash> <mode>
   File modes <*mode*> are printed as 6-digit octal numbers including the file type and file permission bits.
   Path names in extended headers do not include the `a/` and `b/` prefixes.
   The similarity index is the percentage of unchanged lines, and the dissimilarity index is the percentage of changed lines. It is a rounded down integer, followed by a percent sign. The similarity index value of 100% is thus reserved for two equal files, while 100% dissimilarity means that no line from the old file made it into the new one.
   The index line includes the blob object names before and after the change. The <*mode*> is included if the file mode does not change; otherwise, separate lines indicate the old and the new mode.

3. Pathnames with "unusual" characters are quoted as explained for the configuration variable `core.quotePath` (see linkgit:git-config[1]).

4. All the `file1` files in the output refer to files before the commit, and all the `file2` files refer to files after the commit. It is incorrect to apply each change to each file sequentially. For example, this patch will swap a and b:

```
diff --git a/a b/b
rename from a
rename to b
diff --git a/b b/a
rename from b
rename to a
```

5. Hunk headers mention the name of the function to which the hunk applies. See "Defining a custom hunk-header" in linkgit:gitattributes[5] for details of how to tailor this to specific languages.

Combined diff format ----------

Any diff-generating command can take the `-c` or `--cc` option to produce a ’combined diff’ when showing a merge. This is the default format when showing merges with linkgit:git-diff[1] or linkgit:git-show[1]. Note also that you can give suitable `--diff-merges` option to any of these commands to force generation of diffs in a specific format.

A "combined diff" format looks like this:

```
diff --combined describe.c
index fabadb8,cc95eb0..4866510
--- a/describe.c
+++ b/describe.c
@@@ -98,20 -98,12 +98,20 @@@
 return (a_date > b_date) ? -1 : (a_date == b_date) ? 0 : 1;
  }

- static void describe(char *arg)
 -static void describe(struct commit *cmit, int last_one)
++static void describe(char *arg, int last_one)
  {
 +unsigned char sha1[20];
 +struct commit *cmit;
 struct commit_list *list;
 static int initialized = 0;
 struct commit_name *n;

 +if (get_sha1(arg, sha1) < 0)
 +usage(describe_usage);
 +cmit = lookup_commit_reference(sha1);
 +if (!cmit)
 +usage(describe_usage);
 +
 if (!initialized) {
 initialized = 1;
 for_each_ref(get_name);
```

1. It is preceded by a "git diff" header, that looks like this (when the `-c` option is used):

```
diff --combined file
```

or like this (when the `--cc` option is used):

```
diff --cc file
```

2. It is followed by one or more extended header lines (this example shows a merge with two parents):
   index <hash>,<hash>..<hash> mode <mode>,<mode>..<mode> new file mode <mode> deleted file mode <mode>,<mode>
   The `mode` `<mode>,<mode>..<mode>` line appears only if at least one of the <mode> is different from the rest. Extended headers with information about detected content movement (renames and copying detection) are designed to work with the diff of two *<tree-ish>* and are not used by combined diff format.

3. It is followed by a two-line from-file/to-file header:

- a/file +++ b/file
  Similar to the two-line header for the traditional 'unified' diff format, `/dev/null` is used to signal created or deleted files.
  However, if the --combined-all-paths option is provided, instead of a two-line from-file/to-file, you get an N+1 line from-file/to-file header, where N is the number of parents in the merge commit:

- a/file
- a/file
- a/file +++ b/file
  This extended format can be useful if rename or copy detection is active, to allow you to see the original name of the file in different parents.

4. Chunk header format is modified to prevent people from accidentally feeding it to `patch` `-p1`. Combined diff format was created for review of merge commit changes, and was not meant to be applied. The change is similar to the change in the extended 'index' header:

```
@@@ <from-file-range> <from-file-range> <to-file-range> @@@
```

There are (number of parents + 1) `@` characters in the chunk header for combined diff format.

Unlike the traditional 'unified' diff format, which shows two files A and B with a single column that has `-` (minus -- appears in A but removed in B), `+` (plus — missing in A but added to B), or `" "` (space — unchanged) prefix, this format compares two or more files file1, file2,… with one file X, and shows how X differs from each of fileN. One column for each of fileN is prepended to the output line to note how X's line is different from it.

A `-` character in the column N means that the line appears in fileN but it does not appear in the result. A `+` character in the column N means that the line appears in the result, and fileN does not have that line (in other words, the line was added, from the point of view of that parent).

In the above example output, the function signature was changed from both files (hence two `-` removals from both file1 and file2, plus `++` to mean one line that was added does not appear in either file1 or file2). Also, eight other lines are the same from file1 but do not appear in file2 (hence prefixed with `+`).

When shown by `git` `diff-tree` `-c`, it compares the parents of a merge commit with the merge result (i.e. file1..fileN are the parents). When shown by `git` `diff-files` `-c`, it compares the two unresolved merge parents with the working tree file (i.e. file1 is stage 2 aka "our version", file2 is stage 3 aka "their version").

other diff formats ----------

The `--summary` option describes newly added, deleted, renamed and copied files. The `--stat` option adds `diffstat`(1) graph to the output. These options can be combined with other options, such as `-p`, and are meant for human consumption.

When showing a change that involves a rename or a copy, `--stat` output formats the pathnames compactly by combining common prefix and suffix of the pathnames. For example, a change that moves `arch/i386/Makefile` to `arch/x86/Makefile` while modifying 4 lines will be shown like this:

```
arch/{i386 => x86}/Makefile    |   4 +--
```

The `--numstat` option gives the diffstat(1) information but is designed for easier machine consumption. An entry in `--numstat` output looks like this:

```
1␉2␉README
3␉1␉arch/{i386 => x86}/Makefile
```

That is, from left to right:

1. the number of added lines;
2. a tab;
3. the number of deleted lines;
4. a tab;
5. pathname (possibly with rename/copy information);
6. a newline.

When `-z` output option is in effect, the output is formatted this way:

```
1␀2␀README NUL
3␀1␀NUL arch/i386/Makefile NUL arch/x86/Makefile NUL
```

That is:

1. the number of added lines;
2. a tab;
3. the number of deleted lines;
4. a tab;
5. a NUL (only exists if renamed/copied);
6. pathname in preimage;
7. a NUL (only exists if renamed/copied);
8. pathname in postimage (only exists if renamed/copied);
9. a NUL.

The extra `NUL` before the preimage path in renamed case is to allow scripts that read the output to tell if the current record being read is a single-path record or a rename/copy record without reading ahead. After reading added and deleted lines, reading up to `NUL` would yield the pathname, but if that is `NUL`, the record will show two paths.

EXAMPLES --------

Various ways to check your working tree
$ git diff ① $ git diff --cached ② $ git diff HEAD ③ $ git diff AUTO_MERGE ④

1. Changes in the working tree not yet staged for the next commit.
2. Changes between the index and your last commit; what you would be committing if you run `git` `commit` without `-a` option.
3. Changes in the working tree since your last commit; what you would be committing if you run `git` `commit` `-a`
4. Changes in the working tree you’ve made to resolve textual conflicts so far.

Comparing with arbitrary commits
$ git diff test ① $ git diff HEAD -- ./test ② $ git diff HEAD^ HEAD ③

1. Instead of using the tip of the current branch, compare with the tip of "test" branch.
2. Instead of comparing with the tip of "test" branch, compare with the tip of the current branch, but limit the comparison to the file "test".
3. Compare the version before the last commit and the last commit.

Comparing branches
$ git diff topic master ① $ git diff topic..master ② $ git diff topic...master ③

1. Changes between the tips of the topic and the master branches.
2. Same as above.

3. Changes that occurred on the master branch since when the topic branch was started off it.

Limiting the diff output
$ git diff --diff-filter=MRC ① $ git diff --name-status ② $ git diff arch/i386 include/asm-i386 ③

1. Show only modification, rename, and copy, but not addition or deletion.
2. Show only names and the nature of change, but not actual diff output.
3. Limit diff output to named subtrees.

Munging the diff output
$ git diff --find-copies-harder -B -C ① $ git diff -R ②

1. Spend extra cycles to find renames, copies and complete rewrites (very expensive).
2. Output diff in reverse.

CONFIGURATION ----------

Everything below this line in this section is selectively included from the linkgit:git-config[1] documentation. The content is the same as what’s found there:

`diff.autoRefreshIndex`
When using `git` `diff` to compare with work tree files, do not consider stat-only changes as changed. Instead, silently run `git` `update-index` `--refresh` to update the cached stat information for paths whose contents in the work tree match the contents in the index. This option defaults to `true`. Note that this affects only `git` `diff` Porcelain, and not lower level `diff` commands such as `git` `diff-files`.

`diff.dirstat`
ifdef::git-diff[] A comma separated list of `--dirstat` parameters specifying the default behavior of the `--dirstat` option to `git` `diff` and friends. endif::git-diff[] ifndef::git-diff[] A comma separated list of `--dirstat` parameters specifying the default behavior of the `--dirstat` option to linkgit:git-diff[1] and friends. endif::git-diff[] The defaults can be overridden on the command line (using `--dirstat=<param>,…`). The fallback defaults (when not changed by `diff.dirstat`) are `changes,noncumulative,3`. The following parameters are available:

`changes`;; Compute the dirstat numbers by counting the lines that have been removed from the source, or added to the destination. This ignores the amount of pure code movements within a file. In other words, rearranging lines in a file is not counted as much as other changes. This is the default behavior when no parameter is given. `lines`;; Compute the dirstat numbers by doing the regular line-based diff analysis, and summing the removed/added line counts. (For binary files, count 64-byte chunks instead, since binary files have no natural concept of lines). This is a more expensive `--dirstat` behavior than the `changes` behavior, but it does count rearranged lines within a file as much as other changes. The resulting output is consistent with what you get from the other `--*stat` options. `files`;; Compute the dirstat numbers by counting the number of files

changed. Each changed file counts equally in the dirstat analysis. This is the computationally cheapest `--dirstat` behavior, since it does not have to look at the file contents at all. `cumulative`;; Count changes in a child directory for the parent directory as well. Note that when using `cumulative`, the sum of the percentages reported may exceed 100%. The default (non-cumulative) behavior can be specified with the `noncumulative` parameter. <*limit*>;; An integer parameter specifies a cut-off percent (3% by default). Directories contributing less than this percentage of the changes are not shown in the output.

Example: The following will count changed files, while ignoring directories with less than 10% of the total amount of changed files, and accumulating child directory counts in the parent directories: `files,10,cumulative`.

`diff.statNameWidth`
Limit the width of the filename part in `--stat` output. If set, applies to all commands generating `--stat` output except `format-patch`.

`diff.statGraphWidth`
Limit the width of the graph part in `--stat` output. If set, applies to all commands generating `--stat` output except `format-patch`.

`diff.context`
Generate diffs with <*n*> lines of context instead of the default of 3. This value is overridden by the `-U` option.

`diff.interHunkContext`
Show the context between diff hunks, up to the specified number of lines, thereby fusing the hunks that are close to each other. This value serves as the default for the `--inter-hunk-context` command line option.

`diff.external`
If this config variable is set, diff generation is not performed using the internal diff machinery, but using the given command. Can be overridden with the `GIT_EXTERNAL_DIFF` environment variable. The command is called with parameters as described under "git Diffs" in linkgit:git[1]. Note: if you want to use an external diff program only on a subset of your files, you might want to use linkgit:gitattributes[5] instead.

`diff.trustExitCode`
If this boolean value is set to `true` then the `diff.external` command is expected to return exit code 0 if it considers the input files to be equal or 1 if it considers them to be different, like `diff` (1). If it is set to `false`, which is the default, then the command is expected to return exit code `0` regardless of equality. Any other exit code causes Git to report a fatal error.

`diff.ignoreSubmodules`
Sets the default value of `--ignore-submodules`. Note that this affects only `git` `diff` Porcelain, and not lower level `diff` commands such as `git` `diff-files`. `git` `checkout` and `git` `switch` also honor this setting when reporting uncommitted changes. Setting it to `all` disables the submodule

summary normally shown by `git commit` and `git status` when `status.submoduleSummary` is set unless it is overridden by using the `--ignore-submodules` command-line option. The `git submodule` commands are not affected by this setting. By default this is set to untracked so that any untracked submodules are ignored.

`diff.mnemonicPrefix`
If set, `git diff` uses a prefix pair that is different from the standard `a/` and `b/` depending on what is being compared. When this configuration is in effect, reverse diff output also swaps the order of the prefixes: `git diff`;; compares the (i)ndex and the (w)ork tree; `git diff HEAD`;; compares a (c)ommit and the (w)ork tree; `git diff --cached`;; compares a (c)ommit and the (i)ndex; `git diff HEAD:<file1> <file2>`;; compares an (o)bject and a (w)ork tree entity; `git diff --no-index <a> <b>`;; compares two non-git things *<a>* and *<b>*.

`diff.noPrefix`
If set, `git diff` does not show any source or destination prefix.

`diff.srcPrefix`
If set, `git diff` uses this source prefix. Defaults to `a/`.

`diff.dstPrefix`
If set, `git diff` uses this destination prefix. Defaults to `b/`.

`diff.relative`
If set to `true`, `git diff` does not show changes outside of the directory and show pathnames relative to the current directory.

`diff.orderFile`
File indicating how to order files within a diff. ifdef::git-diff[] See the `-O` option for details. endif::git-diff[] ifndef::git-diff[] See the `-O` option to linkgit:git-diff[1] for details. endif::git-diff[] If `diff.orderFile` is a relative pathname, it is treated as relative to the top of the working tree.

`diff.renameLimit`
The number of files to consider in the exhaustive portion of copy/rename detection; equivalent to the `git diff` option `-l`. If not set, the default value is currently 1000. This setting has no effect if rename detection is turned off.

`diff.renames`
Whether and how Git detects renames. If set to `false`, rename detection is disabled. If set to `true`, basic rename detection is enabled. If set to `copies` or `copy`, Git will detect copies, as well. Defaults to `true`. Note that this affects only `git diff` Porcelain like linkgit:git-diff[1] and linkgit:git-log[1], and not lower level commands such as linkgit:git-diff-files[1].

`diff.suppressBlankEmpty`
A boolean to inhibit the standard behavior of printing a space before each empty output line. Defaults to `false`.

`diff.submodule`
Specify the format in which differences in submodules are shown. The `short` format just shows the names of the commits at the beginning and end of the range. The `log` format lists the commits in the range like linkgit:git-submodule[1] `summary` does. The `diff` format shows an inline diff of the changed contents of the submodule. Defaults to `short`.

`diff.wordRegex`
A POSIX Extended Regular Expression used to determine what is a "word" when performing word-by-word difference calculations. Character sequences that match the regular expression are "words", all other characters are **ignorable** whitespace.

`diff.<driver>.command`
The custom diff driver command. See linkgit:gitattributes[5] for details.

`diff.<driver>.trustExitCode`
If this boolean value is set to `true` then the `diff.<driver>.command` command is expected to return exit code 0 if it considers the input files to be equal or 1 if it considers them to be different, like `diff`(1). If it is set to `false`, which is the default, then the command is expected to return exit code 0 regardless of equality. Any other exit code causes Git to report a fatal error.

`diff.<driver>.xfuncname`
The regular expression that the diff driver should use to recognize the hunk header. A built-in pattern may also be used. See linkgit:gitattributes[5] for details.

`diff.<driver>.binary`
Set this option to `true` to make the diff driver treat files as binary. See linkgit:gitattributes[5] for details.

`diff.<driver>.textconv`
The command that the diff driver should call to generate the text-converted version of a file. The result of the conversion is used to generate a human-readable diff. See linkgit:gitattributes[5] for details.

`diff.<driver>.wordRegex`
The regular expression that the diff driver should use to split words in a line. See linkgit:gitattributes[5] for details.

`diff.<driver>.cachetextconv`
Set this option to `true` to make the diff driver cache the text conversion outputs. See linkgit:gitattributes[5] for details.

`diff.indentHeuristic`
Set this option to `false` to disable the default heuristics that shift diff hunk boundaries to make patches easier to read.

`diff.algorithm`
Choose a diff algorithm. The variants are as follows:

`default`;; `myers`;; The basic greedy diff algorithm. Currently, this is the default. `minimal`;; Spend extra time to make sure the smallest possible diff is produced. `patience`;; Use "patience diff" algorithm when generating patches. `histogram`;; This algorithm extends the patience algorithm to "support low-occurrence common elements".

`diff.wsErrorHighlight`
Highlight whitespace errors in the `context`, `old` or `new` lines of the diff. Multiple values are separated by comma, `none` resets previous values, `default` reset the list to `new` and `all` is a shorthand for `old,new,context`. The whitespace errors are colored with `color.diff.whitespace`. The command line option `--ws-error-highlight=<kind>` overrides this setting.

`diff.colorMoved`
If set to either a valid *<mode>* or a `true` value, moved lines in a diff are colored differently. ifdef::git-diff[] For details of valid modes see `--color-moved`. endif::git-diff[] ifndef::git-diff[] For details of valid modes see `--color-moved` in linkgit:git-diff[1]. endif::git-diff[] If simply set to `true` the default color mode will be used. When set to `false`, moved lines are not colored.

`diff.colorMovedWS`
When moved lines are colored using e.g. the `diff.colorMoved` setting, this option controls the mode how spaces are treated. For details of valid modes see `--color-moved-ws` in linkgit:git-diff[1].

SEE ALSO -------- `diff`(1), linkgit:git-difftool[1], linkgit:git-log[1], linkgit:gitdiffcore[7], linkgit:git-format-patch[1], linkgit:git-apply[1], linkgit:git-show[1]

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

□□□ #1
□ □□□ □□□ 2026-10-31 20:10:45 UTC
□ □□□ □□□ 2026-10-31 20:10:45 UTC