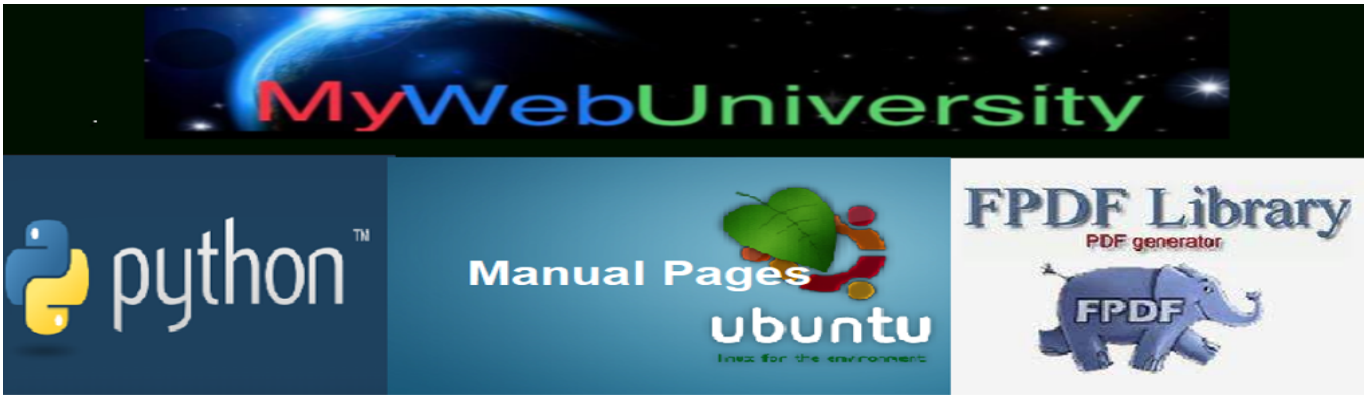

Rocky Enterprise Linux 9.2 Manual Pages on command 'git-diff.1'

$ man git-diff.1

GIT-DIFF(1) Git Manual 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>

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 git-add(1).

git diff [<options>] --no-index [--] <path> <path>

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.

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. A convenient way to produce the desired set of revisions is to use the ^@ suffix. For instance, if master names a merge commit, git diff master master^@ gives the same combined diff as git show master.

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>.

For a more complete list of ways to spell <commit>, see "SPECIFYING REVISIONS" section in 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 gitrevisions(7).

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

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

OPTIONS

-p, -u, --patch

Generate patch (see section on generating patches). This is the default.

-s, --no-patch

Suppress diff output. Useful for commands like git show that show the patch by default, or to cancel the effect of --patch.

-U<n>, --unified=<n>

Generate diffs with <n> lines of context instead of the usual three. Implies --patch.

--output=<file>

Output to a specific file instead of stdout.

--output-indicator-new=<char>, --output-indicator-old=<char>, --output-indicator-context=<char>

Specify the character used to indicate new, old or context lines in the generated patch. Normally they are +, - and ' ' respectively.

--raw

Generate the diff in raw format.

--patch-with-raw

Synonym for -p --raw.

--indent-heuristic

Enable the heuristic that shifts diff hunk boundaries to make patches easier to read. This is the default.

--no-indent-heuristic

Disable the indent heuristic.

--minimal

Spend extra time to make sure the smallest possible diff is produced.

--patience

Generate a diff using the "patience diff" algorithm.

--histogram

Generate a diff using the "histogram diff" algorithm.

--anchored=<text>

Generate a diff using the "anchored diff" algorithm.

This option may be specified more than once.

If a line exists in both the source and destination, exists only once, and starts with this text, this algorithm attempts to prevent it from appearing as a deletion or addition in the output. It uses the "patience diff" algorithm internally.

--diff-algorithm={patience|minimal|histogram|myers}

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".

For instance, if you configured the diff.algorithm variable to a non-default value and want to use the default one, then you have to use --diff-algorithm=default option.

--stat[=<width>[,<name-width>[,<count>]]]

Generate a diffstat. By default, as much space as necessary will be used for the filename part, and the rest for the graph part. Maximum width defaults to terminal width, or 80 columns if not connected to a terminal, and can be overridden by <width>.

The width of the filename part can be limited by giving another width <name-width> after a comma. The width of the graph part can be limited by using --stat-graph-width=<width> (affects all commands generating a stat graph) or by setting diff.statGraphWidth=<width> (does not affect git format-patch). By giving a

third parameter <count>, you can limit the output to the first <count> lines, followed by ... if there are more.

These parameters can also be set individually with --stat-width=<width>, --stat-name-width=<name-width> and --stat-count=<count>.

--compact-summary

Output a condensed summary of extended header information such as file creations or deletions ("new" or "gone", optionally "+l" if it?s a symlink) and mode changes ("+x" or "-x" for adding or removing executable bit respectively) in diffstat. The information is put between the filename part and the graph part. Implies --stat.

--numstat

Similar to --stat, but shows number of added and deleted lines in decimal notation and pathname without abbreviation, to make it more machine friendly. For binary files, outputs two - instead of saying 0 0.

--shortstat

Output only the last line of the --stat format containing total number of modified files, as well as number of added and deleted lines.

-X[<param1,param2,...>], --dirstat[=<param1,param2,...>]

Output the distribution of relative amount of changes for each sub-directory. The behavior of --dirstat can be customized by passing it a comma separated list of parameters. The defaults are controlled by the diff.dirstat configuration variable (see git-config(1)). 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: --dirstat=files,10,cumulative.

--cumulative

Synonym for --dirstat=cumulative

--dirstat-by-file[=<param1,param2>...]

Synonym for --dirstat=files,param1,param2...

--summary

Output a condensed summary of extended header information such as creations, renames and mode changes.

--patch-with-stat

Synonym for -p --stat.

-z

When --raw, --numstat, --name-only or --name-status has been given, do not munge pathnames and use NULs as output field terminators.

Without this option, pathnames with "unusual" characters are quoted as explained for the configuration variable core.quotePath (see git-config(1)).

--name-only

Show only names of changed files. The file names are often encoded in UTF-8. For more information see the discussion about encoding in the git-log(1) manual page.

--name-status

Show only names and status of changed files. See the description of the --diff-filter option on what the status letters mean. Just like --name-only the file names are often encoded in UTF-8.

--submodule[=<format>]

Specify how differences in submodules are shown. When specifying --submodule=short the short format is used. This format just shows the names of the commits at the beginning and end of the range. When --submodule or --submodule=log is specified, the log format is used. This format lists the commits in the range like git-submodule(1) summary does. When --submodule=diff is specified, the diff format is used. This format shows an inline diff of the changes in the submodule contents between the commit range. Defaults to diff.submodule or the short format if the config option is unset.

--color[=<when>]

Show colored diff. --color (i.e. without =<when>) is the same as --color=always. <when> can be one of always, never, or auto. It can be changed by the color.ui and color.diff configuration settings.

--no-color

Turn off colored diff. This can be used to override configuration settings. It is the same as --color=never.

--color-moved[=<mode>]

Moved lines of code are colored differently. It can be changed by the diff.colorMoved configuration setting. The <mode> defaults to no if the option is not given and to zebra if the option with no mode is given. The mode must be one of:

no

Moved lines are not highlighted.

default

Is a synonym for zebra. This may change to a more sensible mode in the future.

plain

Any line that is added in one location and was removed in another location will be colored with color.diff.newMoved. Similarly color.diff.oldMoved will be used for removed lines that are added somewhere else in the diff. This mode picks up any moved line, but it is not very useful in a review to determine if a block of code was moved without permutation.

blocks

Blocks of moved text of at least 20 alphanumeric characters are detected greedily. The detected blocks are painted using either the color.diff.{old,new}Moved color. Adjacent blocks cannot be told apart.

zebra

Blocks of moved text are detected as in blocks mode. The blocks are painted using either the color.diff.{old,new}Moved color or color.diff.{old,new}MovedAlternative. The change between the two colors indicates that a new block was detected.

dimmed-zebra

Similar to zebra, but additional dimming of uninteresting parts of moved code is performed. The bordering lines of two adjacent blocks are considered interesting, the rest is uninteresting. dimmed_zebra is a deprecated synonym.

--no-color-moved

Turn off move detection. This can be used to override configuration settings. It is the same as --color-moved=no.

--color-moved-ws=<modes>

This configures how whitespace is ignored when performing the move detection for --color-moved. It can be set by the diff.colorMovedWS configuration setting. These modes can be given as a comma separated list:

no

Do not ignore whitespace when performing move detection.

ignore-space-at-eol

Ignore changes in whitespace at EOL.

ignore-space-change

Ignore changes in amount of whitespace. This ignores whitespace at line end, and considers all other sequences of one or more whitespace characters to be equivalent.

ignore-all-space

Ignore whitespace when comparing lines. This ignores differences even if one line has whitespace where the other line has none.

allow-indentation-change

Initially ignore any whitespace in the move detection, then group the moved code

blocks only into a block if the change in whitespace is the same per line. This is incompatible with the other modes.

--no-color-moved-ws

Do not ignore whitespace when performing move detection. This can be used to override configuration settings. It is the same as --color-moved-ws=no.

--word-diff[=<mode>]

Show a word diff, using the <mode> to delimit changed words. By default, words are delimited by whitespace; see --word-diff-regex below. The <mode> defaults to plain, and must be one of:

color

Highlight changed words using only colors. Implies --color.

plain

Show words as [-removed-] and {+added+}. Makes no attempts to escape the delimiters if they appear in the input, so the output may be ambiguous.

porcelain

Use a special line-based format intended for script consumption. Added/removed/unchanged runs are printed in the usual unified diff format, starting with a +/-/` ` character at the beginning of the line and extending to the end of the line. Newlines in the input are represented by a tilde ~ on a line of its own.

none

Disable word diff again.

Note that despite the name of the first mode, color is used to highlight the changed parts in all modes if enabled.

--word-diff-regex=<regex>

Use <regex> to decide what a word is, instead of considering runs of non-whitespace to be a word. Also implies --word-diff unless it was already enabled.

Every non-overlapping match of the <regex> is considered a word. Anything between these matches is considered whitespace and ignored(!) for the purposes of finding differences. You may want to append |[^[:space:]] to your regular expression to make sure that it matches all non-whitespace characters. A match that contains a newline is silently truncated(!) at the newline.

For example, --word-diff-regex=. will treat each character as a word and,

correspondingly, show differences character by character.

The regex can also be set via a diff driver or configuration option, see gitattributes(5) or git-config(1). Giving it explicitly overrides any diff driver or configuration setting. Diff drivers override configuration settings.

--color-words[=<regex>]

Equivalent to --word-diff=color plus (if a regex was specified) --word-diff-regex=<regex>.

--no-renames

Turn off rename detection, even when the configuration file gives the default to do so.

--[no-]rename-empty

Whether to use empty blobs as rename source.

--check

Warn if changes introduce conflict markers or whitespace errors. What are considered whitespace errors is controlled by core.whitespace configuration. By default, trailing whitespaces (including lines that consist solely of whitespaces) and a space character that is immediately followed by a tab character inside the initial indent of the line are considered whitespace errors. Exits with non-zero status if problems are found. Not compatible with --exit-code.

--ws-error-highlight=<kind>

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. When this option is not given, and the configuration variable diff.wsErrorHighlight is not set, only whitespace errors in new lines are highlighted. The whitespace errors are colored with color.diff.whitespace.

--full-index

Instead of the first handful of characters, show the full pre- and post-image blob object names on the "index" line when generating patch format output.

--binary

In addition to --full-index, output a binary diff that can be applied with git-apply. Implies --patch.

--abbrev[=<n>]

Instead of showing the full 40-byte hexadecimal object name in diff-raw format output

and diff-tree header lines, show the shortest prefix that is at least <n> hexdigits long that uniquely refers the object. In diff-patch output format, --full-index takes higher precedence, i.e. if --full-index is specified, full blob names will be shown regardless of --abbrev. Non default number of digits can be specified with --abbrev=<n>.

-B[<n>][/<m>], --break-rewrites[=[<n>][/<m>]]

Break complete rewrite changes into pairs of delete and create. This serves two purposes:

It affects the way a change that amounts to a total rewrite of a file not as a series of deletion and insertion mixed together with a very few lines that happen to match textually as the context, but as a single deletion of everything old followed by a single insertion of everything new, and the number m controls this aspect of the -B option (defaults to 60%). -B/70% specifies that less than 30% of the original should remain in the result for Git to consider it a total rewrite (i.e. otherwise the resulting patch will be a series of deletion and insertion mixed together with context lines).

When used with -M, a totally-rewritten file is also considered as the source of a rename (usually -M only considers a file that disappeared as the source of a rename), and the number n controls this aspect of the -B option (defaults to 50%). -B20% specifies that a change with addition and deletion compared to 20% or more of the file?s size are eligible for being picked up as a possible source of a rename to another file.

-M[<n>], --find-renames[=<n>]

Detect renames. If n is specified, it is a threshold on the similarity index (i.e. amount of addition/deletions compared to the file?s size). For example, -M90% means Git should consider a delete/add pair to be a rename if more than 90% of the file hasn?t changed. Without a % sign, the number is to be read as a fraction, with a decimal point before it. I.e., -M5 becomes 0.5, and is thus the same as -M50%. Similarly, -M05 is the same as -M5%. To limit detection to exact renames, use -M100%. The default similarity index is 50%.

-C[<n>], --find-copies[=<n>]

Detect copies as well as renames. See also --find-copies-harder. If n is specified, it has the same meaning as for -M<n>.

--find-copies-harder

For performance reasons, by default, -C option finds copies only if the original file of the copy was modified in the same changeset. This flag makes the command inspect unmodified files as candidates for the source of copy. This is a very expensive operation for large projects, so use it with caution. Giving more than one -C option has the same effect.

-D, --irreversible-delete

Omit the preimage for deletes, i.e. print only the header but not the diff between the preimage and /dev/null. The resulting patch is not meant to be applied with patch or git apply; this is solely for people who want to just concentrate on reviewing the text after the change. In addition, the output obviously lacks enough information to apply such a patch in reverse, even manually, hence the name of the option. When used together with -B, omit also the preimage in the deletion part of a delete/create pair.

-l<num>

The -M and -C options involve some preliminary steps that can detect subsets of renames/copies cheaply, followed by an exhaustive fallback portion that compares all remaining unpaired destinations to all relevant sources. (For renames, only remaining unpaired sources are relevant; for copies, all original sources are relevant.) For N sources and destinations, this exhaustive check is O(N^2). This option prevents the exhaustive portion of rename/copy detection from running if the number of source/destination files involved exceeds the specified number. Defaults to diff.renameLimit. Note that a value of 0 is treated as unlimited.

--diff-filter=[(A|C|D|M|R|T|U|X|B)...[*]]

Select only files that are Added (A), Copied (C), Deleted (D), Modified (M), Renamed (R), have their type (i.e. regular file, symlink, submodule, ...) changed (T), are Unmerged (U), are Unknown (X), or have had their pairing Broken (B). Any combination of the filter characters (including none) can be used. When * (All-or-none) is added to the combination, all paths are selected if there is any file that matches other criteria in the comparison; if there is no file that matches other criteria, nothing is selected.

Also, these upper-case letters can be downcased to exclude. E.g. --diff-filter=ad excludes added and deleted paths.

Note that not all diffs can feature all types. For instance, diffs from the index to the working tree can never have Added entries (because the set of paths included in the diff is limited by what is in the index). Similarly, copied and renamed entries cannot appear if detection for those types is disabled.

-S<string>

Look for differences that change the number of occurrences of the specified string (i.e. addition/deletion) in a file. Intended for the scripter?s use.

It is useful when you?re looking for an exact block of code (like a struct), and want to know the history of that block since it first came into being: use the feature iteratively to feed the interesting block in the preimage back into -S, and keep going until you get the very first version of the block.

Binary files are searched as well.

-G<regex>

Look for differences whose patch text contains added/removed lines that match <regex>.

To illustrate the difference between -S<regex> --pickaxe-regex and -G<regex>, consider a commit with the following diff in the same file:

```
+    return frotz(nitfol, two->ptr, 1, 0);
...
-    hit = frotz(nitfol, mf2.ptr, 1, 0);
```

While git log -G"frotz\(nitfol" will show this commit, git log -S"frotz\(nitfol" --pickaxe-regex will not (because the number of occurrences of that string did not change).

Unless --text is supplied patches of binary files without a textconv filter will be ignored.

See the pickaxe entry in gitdiffcore(7) for more information.

--find-object=<object-id>

Look for differences that change the number of occurrences of the specified object. Similar to -S, just the argument is different in that it doesn?t search for a specific string but for a specific object id.

The object can be a blob or a submodule commit. It implies the -t option in git-log to also find trees.

--pickaxe-all

When -S or -G finds a change, show all the changes in that changeset, not just the

files that contain the change in <string>.

--pickaxe-regex

Treat the <string> given to -S as an extended POSIX regular expression to match.

-O<orderfile>

Control the order in which files appear in the output. This overrides the diff.orderFile configuration variable (see git-config(1)). To cancel diff.orderFile, use -O/dev/null.

The output order is determined by the order of glob patterns in <orderfile>. All files with pathnames that match the first pattern are output first, all files with pathnames that match the second pattern (but not the first) are output next, and so on. All files with pathnames that do not match any pattern are output last, as if there was an implicit match-all pattern at the end of the file. If multiple pathnames have the same rank (they match the same pattern but no earlier patterns), their output order relative to each other is the normal order.

<orderfile> is parsed as follows:

? Blank lines are ignored, so they can be used as separators for readability.

? Lines starting with a hash ("#") are ignored, so they can be used for comments. Add a backslash ("\") to the beginning of the pattern if it starts with a hash.

? Each other line contains a single pattern.

Patterns have the same syntax and semantics as patterns used for fnmatch(3) without the FNM_PATHNAME flag, except a pathname also matches a pattern if removing any number of the final pathname components matches the pattern. For example, the pattern "foo*bar" matches "fooasdfbar" and "foo/bar/baz/asdf" but not "foobarx".

--skip-to=<file>, --rotate-to=<file>

Discard the files before the named <file> from the output (i.e. skip to), or move them to the end of the output (i.e. rotate to). These were invented primarily for use of the git difftool command, and may not be very useful otherwise.

-R

Swap two inputs; that is, show differences from index or on-disk file to tree contents.

--relative[=<path>], --no-relative

When run from a subdirectory of the project, it can be told to exclude changes outside the directory and show pathnames relative to it with this option. When you are not in

a subdirectory (e.g. in a bare repository), you can name which subdirectory to make the output relative to by giving a <path> as an argument. --no-relative can be used to countermand both diff.relative config option and previous --relative.

-a, --text

Treat all files as text.

--ignore-cr-at-eol

Ignore carriage-return at the end of line when doing a comparison.

--ignore-space-at-eol

Ignore changes in whitespace at EOL.

-b, --ignore-space-change

Ignore changes in amount of whitespace. This ignores whitespace at line end, and considers all other sequences of one or more whitespace characters to be equivalent.

-w, --ignore-all-space

Ignore whitespace when comparing lines. This ignores differences even if one line has whitespace where the other line has none.

--ignore-blank-lines

Ignore changes whose lines are all blank.

-I<regex>, --ignore-matching-lines=<regex>

Ignore changes whose all lines match <regex>. This option may be specified more than once.

--inter-hunk-context=<lines>

Show the context between diff hunks, up to the specified number of lines, thereby fusing hunks that are close to each other. Defaults to diff.interHunkContext or 0 if the config option is unset.

-W, --function-context

Show whole function as context lines for each change. The function names are determined in the same way as git diff works out patch hunk headers (see Defining a custom hunk-header in gitattributes(5)).

--exit-code

Make the program exit with codes similar to diff(1). That is, it exits with 1 if there were differences and 0 means no differences.

--quiet

Disable all output of the program. Implies --exit-code.

--ext-diff

Allow an external diff helper to be executed. If you set an external diff driver with gitattributes(5), you need to use this option with git-log(1) and friends.

--no-ext-diff

Disallow external diff drivers.

--textconv, --no-textconv

Allow (or disallow) external text conversion filters to be run when comparing binary files. See gitattributes(5) for details. Because textconv filters are typically a one-way conversion, the resulting diff is suitable for human consumption, but cannot be applied. For this reason, textconv filters are enabled by default only for git-diff(1) and git-log(1), but not for git-format-patch(1) or diff plumbing commands.

--ignore-submodules[=<when>]

Ignore changes to submodules in the diff generation. <when> can be either "none", "untracked", "dirty" or "all", which is the default. Using "none" will consider the submodule modified when it either contains untracked or modified files or its HEAD differs from the commit recorded in the superproject and can be used to override any settings of the ignore option in git-config(1) or gitmodules(5). When "untracked" is used submodules are not considered dirty when they only contain untracked content (but they are still scanned for modified content). Using "dirty" ignores all changes to the work tree of submodules, only changes to the commits stored in the superproject are shown (this was the behavior until 1.7.0). Using "all" hides all changes to submodules.

--src-prefix=<prefix>

Show the given source prefix instead of "a/".

--dst-prefix=<prefix>

Show the given destination prefix instead of "b/".

--no-prefix

Do not show any source or destination prefix.

--line-prefix=<prefix>

Prepend an additional prefix to every line of output.

--ita-invisible-in-index

By default entries added by "git add -N" appear as an existing empty file in "git diff" and a new file in "git diff --cached". This option makes the entry appear as a

new file in "git diff" and non-existent in "git diff --cached". This option could be reverted with --ita-visible-in-index. Both options are experimental and could be removed in future.

For more detailed explanation on these common options, see also gitdiffcore(7).

-1 --base, -2 --ours, -3 --theirs

Compare the working tree with the "base" version (stage #1), "our branch" (stage #2) or "their branch" (stage #3). The index contains these stages only for unmerged entries i.e. while resolving conflicts. See 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 <paths> 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.

The "git-diff-tree" command begins its output by printing the hash of what is being compared. After that, all the commands print one output line per changed file.

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 unmerged.
3. a space.
4. mode for "dst"; 000000 if deletion or unmerged.
5. a space.
6. sha1 for "src"; 0{40} if creation or unmerged.
7. a space.
8. sha1 for "dst"; 0{40} if creation, unmerged or "look at work tree".
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.

<sha1> is shown as all 0?s if a file is new on the filesystem and it 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 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 git-diff(1), git-log(1), git-show(1), git-diff-index(1), git-diff-tree(1), or 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 git(1)), and the diff attribute (see gitattributes(5)).

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

1. It is preceded with 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 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 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 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 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 gitattributes(5) for details of how to tailor to 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 git-diff(1) or git-show(1). Note also that you can give suitable --diff-merges option to any of these commands to force generation of diffs in 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 with 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 contents movement (renames and copying detection) are designed to work with diff of two <tree-ish> and are not used by combined diff format.

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

   ```
   --- a/file
   +++ b/file
   ```

   Similar to two-line header for 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 a 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)
```

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

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.

SEE ALSO

diff(1), git-difftool(1), git-log(1), gitdiffcore(7), git-format-patch(1), git-apply(1), git-show(1)

GIT

Part of the git(1) suite