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

C:\>man git-blame.1

GIT-BLAME(1) Git Manual GIT-BLAME(1)

NAME

git-blame - Show what revision and author last modified each line of a file

SYNOPSIS

git blame [-c] [-b] [-l] [--root] [-t] [-f] [-n] [-s] [-e] [-p] [-w] [--incremental]
[-L <range>] [-S <revs-file>] [-M] [-C] [-C] [-C] [--since=<date>]
[--ignore-rev <rev>] [--ignore-revs-file <file>]
[--progress] [--abbrev=<n>] [<rev> | --contents <file> | --reverse <rev>..<rev>]
[--] <file>

DESCRIPTION

Annotates each line in the given file with information from the revision which last modified the line. Optionally, start annotating from the given revision.

When specified one or more times, -L restricts annotation to the requested lines.

The origin of lines is automatically followed across whole-file renames (currently there is no option to turn the rename-following off). To follow lines moved from one file to another, or to follow lines that were copied and pasted from another file, etc., see the -C and -M options.

The report does not tell you anything about lines which have been deleted or replaced; you need to use a tool such as git diff or the "pickaxe" interface briefly mentioned in the following paragraph.

Apart from supporting file annotation, Git also supports searching the development history for when a code snippet occurred in a change. This makes it possible to

track when a code snippet was added to a file, moved or copied between files, and eventually deleted or replaced. It works by searching for a text string in the diff. A small example of the pickaxe interface that searches for blame_usage:

$ git log --pretty=oneline -S'blame_usage'

5040f17eba15504bad66b14a645bddd9b015ebb7 blame -S <ancestry-file>

ea4c7f9bf69e781dd0cd88d2bccb2bf5cc15c9a7 git-blame: Make the output

OPTIONS

-b

Show blank SHA-1 for boundary commits. This can also be controlled via the blame.blankboundary config option.

--root

Do not treat root commits as boundaries. This can also be controlled via the blame.showRoot config option.

--show-stats

Include additional statistics at the end of blame output.

-L <start>,<end>, -L :<funcname>

Annotate only the given line range. May be specified multiple times. Overlapping ranges are allowed.

<start> and <end> are optional. ?-L <start>? or ?-L <start>,? spans from <start> to end of file. ?-L ,<end>? spans from start of file to <end>.

<start> and <end> can take one of these forms:

? number

If <start> or <end> is a number, it specifies an absolute line number (lines count from 1).

? /regex/

This form will use the first line matching the given POSIX regex. If <start> is a regex, it will search from the end of the previous -L range, if any, otherwise from the start of file. If <start> is ?^/regex/?, it will search from the start of file. If <end> is a regex, it will search starting at the line given by <start>.

? +offset or -offset

This is only valid for <end> and will specify a number of lines before or after the line given by <start>.

If ?:<funcname>? is given in place of <start> and <end>, it is a regular expression that denotes the range from the first funcname line that matches <funcname>, up to the next funcname line. ?:<funcname>? searches from the end of the previous -L range, if any, otherwise from the start of file. ?^:<funcname>? searches from the start of file.

-l

Show long rev (Default: off).

-t

Show raw timestamp (Default: off).

-S <revs-file>

Use revisions from revs-file instead of calling git-rev-list(1).

--reverse <rev>..<rev>

Walk history forward instead of backward. Instead of showing the revision in which a line appeared, this shows the last revision in which a line has existed. This requires a range of revision like START..END where the path to blame exists in START. git blame --reverse START is taken as git blame --reverse START..HEAD for convenience.

-p, --porcelain

Show in a format designed for machine consumption.

--line-porcelain

Show the porcelain format, but output commit information for each line, not just the first time a commit is referenced. Implies --porcelain.

--incremental

Show the result incrementally in a format designed for machine consumption.

--encoding=<encoding>

Specifies the encoding used to output author names and commit summaries. Setting it to none makes blame output unconverted data. For more information see the discussion about encoding in the git-log(1) manual page.

--contents <file>

When <rev> is not specified, the command annotates the changes starting backwards from the working tree copy. This flag makes the command pretend as if the working tree copy has the contents of the named file (specify - to make the command read from the standard input).

--date <format>

Specifies the format used to output dates. If --date is not provided, the value of the blame.date config variable is used. If the blame.date config variable is also not set, the iso format is used. For supported values, see the discussion of the --date option at git-log(1).

--[no-]progress

Progress status is reported on the standard error stream by default when it is attached to a terminal. This flag enables progress reporting even if not attached to a terminal. Can?t use --progress together with --porcelain or --incremental.

-M[<num>]

Detect moved or copied lines within a file. When a commit moves or copies a block of lines (e.g. the original file has A and then B, and the commit changes it to B and then A), the traditional blame algorithm notices only half of the movement and typically blames the lines that were moved up (i.e. B) to the parent and assigns blame to the lines that were moved down (i.e. A) to the child commit. With this option, both groups of lines are blamed on the parent by running extra passes of inspection.

<num> is optional but it is the lower bound on the number of alphanumeric characters that Git must detect as moving/copying within a file for it to associate those lines with the parent commit. The default value is 20.

-C[<num>]

In addition to -M, detect lines moved or copied from other files that were modified in the same commit. This is useful when you reorganize your program and move code around across files. When this option is given twice, the command additionally looks for copies from other files in the commit that creates the file. When this option is given three times, the command additionally looks for copies from other files in any commit.

<num> is optional but it is the lower bound on the number of alphanumeric characters that Git must detect as moving/copying between files for it to associate those lines with the parent commit. And the default value is 40. If there are more than one -C options given, the <num> argument of the last -C will take effect.

--ignore-rev <rev>

Ignore changes made by the revision when assigning blame, as if the change never happened. Lines that were changed or added by an ignored commit will be blamed on the previous commit that changed that line or nearby lines. This option may be specified multiple times to ignore more than one revision. If the blame.markIgnoredLines config option is set, then lines that were changed by an ignored commit and attributed to another commit will be marked with a ? in the blame output. If the blame.markUnblamableLines config option is set, then those lines touched by an ignored commit that we could not attribute to another revision are marked with a *.

--ignore-revs-file <file>

Ignore revisions listed in file, which must be in the same format as an fsck.skipList. This option may be repeated, and these files will be processed after any files specified with the blame.ignoreRevsFile config option. An empty file name, "", will clear the list of revs from previously processed files.

-h

Show help message.

-c

Use the same output mode as git-annotate(1) (Default: off).

--score-debug

Include debugging information related to the movement of lines between files (see -C) and lines moved within a file (see -M). The first number listed is the score. This is the number of alphanumeric characters detected as having been moved between or within files. This must be above a certain threshold for git blame to consider those lines of code to have been moved.

-f, --show-name

Show the filename in the original commit. By default the filename is shown if there is any line that came from a file with a different name, due to rename detection.

-n, --show-number

Show the line number in the original commit (Default: off).

-s

Suppress the author name and timestamp from the output.

-e, --show-email

Show the author email instead of author name (Default: off). This can also be controlled via the blame.showEmail config option.

-w

Ignore whitespace when comparing the parent?s version and the child?s to find where the lines came from.

--abbrev=<n>

Instead of using the default 7+1 hexadecimal digits as the abbreviated object name, use <n>+1 digits. Note that 1 column is used for a caret to mark the boundary commit.

THE PORCELAIN FORMAT

In this format, each line is output after a header; the header at the minimum has the first line which has:

? 40-byte SHA-1 of the commit the line is attributed to;

? the line number of the line in the original file;

? the line number of the line in the final file;

? on a line that starts a group of lines from a different commit than the previous one, the number of lines in this group. On subsequent lines this field is absent.

This header line is followed by the following information at least once for each commit:

? the author name ("author"), email ("author-mail"), time ("author-time"), and time zone ("author-tz"); similarly for committer.

? the filename in the commit that the line is attributed to.

? the first line of the commit log message ("summary").

The contents of the actual line is output after the above header, prefixed by a TAB. This is to allow adding more header elements later.

The porcelain format generally suppresses commit information that has already been seen. For example, two lines that are blamed to the same commit will both be shown, but the details for that commit will be shown only once. This is more efficient, but may require more state be kept by the reader. The --line-porcelain option can be used to output full commit information for each line, allowing simpler (but less efficient) usage like:

```
# count the number of lines attributed to each author
git blame --line-porcelain file |
sed -n 's/^author //p' |
sort | uniq -c | sort -rn
```

SPECIFYING RANGES

Unlike git blame and git annotate in older versions of git, the extent of the annotation can be limited to both line ranges and revision ranges. The -L option, which limits annotation to a range of lines, may be specified multiple times.

When you are interested in finding the origin for lines 40-60 for file foo, you can use the -L option like so (they mean the same thing ? both ask for 21 lines starting at line 40):

```
git blame -L 40,60 foo
git blame -L 40,+21 foo
```

Also you can use a regular expression to specify the line range:

```
git blame -L '/^sub hello {/,/^}$/' foo
```

which limits the annotation to the body of the hello subroutine.

When you are not interested in changes older than version v2.6.18, or changes older than 3 weeks, you can use revision range specifiers similar to git rev-list:

```
git blame v2.6.18.. -- foo
git blame --since=3.weeks -- foo
```

When revision range specifiers are used to limit the annotation, lines that have not changed since the range boundary (either the commit v2.6.18 or the most recent commit that is more than 3 weeks old in the above example) are blamed for that range boundary commit.

A particularly useful way is to see if an added file has lines created by copy-and-paste from existing files. Sometimes this indicates that the developer was being sloppy and did not refactor the code properly. You can first find the commit that introduced the file with:

```
git log --diff-filter=A --pretty=short -- foo
```

and then annotate the change between the commit and its parents, using commit^! notation:

```
git blame -C -C -f $commit^! -- foo
```

INCREMENTAL OUTPUT

When called with --incremental option, the command outputs the result as it is built. The output generally will talk about lines touched by more recent commits first (i.e. the lines will be annotated out of order) and is meant to be used by interactive viewers.

The output format is similar to the Porcelain format, but it does not contain the actual lines from the file that is being annotated.

1. Each blame entry always starts with a line of:

   <40-byte hex sha1> <sourceline> <resultline> <num_lines>

   Line numbers count from 1.

2. The first time that a commit shows up in the stream, it has various other information about it printed out with a one-word tag at the beginning of each line describing the extra commit information (author, email, committer, dates, summary, etc.).

3. Unlike the Porcelain format, the filename information is always given and terminates the entry:

   "filename" <whitespace-quoted-filename-goes-here>

   and thus it is really quite easy to parse for some line- and word-oriented parser (which should be quite natural for most scripting languages).

   Note

   For people who do parsing: to make it more robust, just ignore any lines between the first and last one ("<sha1>" and "filename" lines) where you do not recognize the tag words (or care about that particular one) at the beginning of the "extended information" lines. That way, if there is ever added information (like the commit encoding or extended commit commentary), a blame viewer will not care.

MAPPING AUTHORS

If the file .mailmap exists at the toplevel of the repository, or at the location pointed to by the mailmap.file or mailmap.blob configuration options, it is used to map author and committer names and email addresses to canonical real names and email addresses.

In the simple form, each line in the file consists of the canonical real name of an author, whitespace, and an email address used in the commit (enclosed by < and >) to map to the name. For example:

Proper Name <commit@email.xx>

The more complex forms are:

<proper@email.xx> <commit@email.xx>

which allows mailmap to replace only the email part of a commit, and:

Proper Name <proper@email.xx> <commit@email.xx>

which allows mailmap to replace both the name and the email of a commit matching the specified commit email address, and:

Proper Name <proper@email.xx> Commit Name <commit@email.xx>

which allows mailmap to replace both the name and the email of a commit matching both the specified commit name and email address.

Example 1: Your history contains commits by two authors, Jane and Joe, whose names appear in the repository under several forms:

Joe Developer <joe@example.com>

Joe R. Developer <joe@example.com>

Jane Doe <jane@example.com>

Jane Doe <jane@laptop.(none)>

Jane D. <jane@desktop.(none)>

Now suppose that Joe wants his middle name initial used, and Jane prefers her family name fully spelled out. A proper .mailmap file would look like:

Jane Doe <jane@desktop.(none)>

Joe R. Developer <joe@example.com>

Note how there is no need for an entry for <jane@laptop.(none)>, because the real name of that author is already correct.

Example 2: Your repository contains commits from the following authors:

nick1 <bugs@company.xx>

nick2 <bugs@company.xx>

nick2 <nick2@company.xx>

santa <me@company.xx>

claus <me@company.xx>

CTO <cto@coompany.xx>

Then you might want a .mailmap file that looks like:

<cto@company.xx> <cto@coompany.xx>

Some Dude <some@dude.xx> nick1 <bugs@company.xx>

Other Author <other@author.xx> nick2 <bugs@company.xx>

Other Author <other@author.xx> <nick2@company.xx>

Santa Claus <santa.claus@northpole.xx> <me@company.xx>

Use hash # for comments that are either on their own line, or after the email address.

SEE ALSO

git-annotate(1)

GIT

Part of the git(1) suite