git-blame

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>] [--color-lines] [--color-by-age] [--progress] [--abbrev=<n>] [--contents <file>] [<rev> | --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 ------- include::blame-options.adoc[]

`-c`
Use the same output mode as linkgit: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 the 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.

`--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.

`--abbrev=<n>`
Instead of using the default *7* hexadecimal digits as the abbreviated object name, use at least *<n>* digits, but ensure the commit object names are unique. If commits marked with caret (boundary),

question mark (ignored) or asterisk (unblamable) are shown, extend unmarked object names to align them.

THE DEFAULT FORMAT ----------

When neither `--porcelain` nor `--incremental` option is specified, `git blame` will output annotation for each line with:

- abbreviated object name for the commit the line came from;
- author ident (by default the author name and date, unless `-s` or `-e` is specified); and
- line number

before the line contents.

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 are 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. Information which is specific to individual lines will not be grouped together, like revs to be marked `ignored` or `unblamable`. 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.

1. 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.).

1. 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).
   For people who do parsing: to make it more robust, just ignore any lines between the first and last one (*<40-byte-hex-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 ----------

See linkgit:gitmailmap[5].

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:

blame.blankBoundary
Show blank commit object name for boundary commits in linkgit:git-blame[1]. This option defaults to false.

blame.coloring
This determines the coloring scheme to be applied to blame output. It can be ’repeatedLines’, ’highlightRecent’, or ’none’ which is the default.

blame.date
Specifies the format used to output dates in linkgit:git-blame[1]. If unset the iso format is used. For supported values, see the discussion of the `--date` option at linkgit:git-log[1].

blame.showEmail
Show the author email instead of author name in linkgit:git-blame[1]. This option defaults to false.

blame.showRoot
Do not treat root commits as boundaries in linkgit:git-blame[1]. This option defaults to false.

blame.ignoreRevsFile
Ignore revisions listed in the file, one unabbreviated object name per line, in linkgit:git-blame[1]. Whitespace and comments beginning with `#` are ignored. If `.git-blame-ignore-revs` exists at the root of the working tree (or `HEAD:.git-blame-ignore-revs` in a bare repository), it is added as the initial element in the list of ignore-revs files. Other files listed in the configuration are also used, but an empty element makes all elements that appeared before in the list forgotten. This option will be handled before the command line option `--ignore-revs-file`.

blame.markUnblamableLines
Mark lines that were changed by an ignored revision that we could not attribute to another commit with a ’*’ in the output of linkgit:git-blame[1].

blame.markIgnoredLines
Mark lines that were changed by an ignored revision that we attributed to another commit with a ’?’ in the output of linkgit:git-blame[1].

SEE ALSO -------- linkgit:git-annotate[1]

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

□□□ #1
□ □□□ □□□ 2026-10-31 20:09:51 UTC
□ □□□ □□□ 2026-10-31 20:09:52 UTC