NAME

git-shortlog - Summarize git log output

SYNOPSIS

git log --pretty=short | *git shortlog* [<options>]
git shortlog [<options>] [<revision range>] [[--] <path>...]

DESCRIPTION

Summarizes *git log* output in a format suitable for inclusion in release announcements. Each commit will be grouped by author and title.

Additionally, [PATCH] will be stripped from the commit description.

If no revisions are passed on the command line and either standard input is not a terminal or there is no current branch, *git shortlog* will output a summary of the log read from standard input, without reference to the current repository.

OPTIONS

-n, --numbered
: Sort output according to the number of commits per author instead of author alphabetic order.

-s, --summary
: Suppress commit description and provide a commit count summary only.

-e, --email
: Show the email address of each author.

--format[=<format>]
: Instead of the commit subject, use some other information to describe each commit. <*format*> can be any string accepted by the --format option of *git log*, such as ** [%h] %s*. (See the PRETTY FORMATS section of **git-log(1)**.)

  Each pretty-printed commit will be rewrapped before it is shown.

-w[<width>[,<indent1>[,<indent2>]]]
: Linewrap the output by wrapping each line at width. The first line of each entry is indented by indent1 spaces, and the second and subsequent lines are indented by indent2 spaces. width, indent1, and indent2 default to 76, 6 and 9 respectively.

  If width is 0 (zero) then indent the lines of the output without wrapping them.

<revision range>
: Show only commits in the specified revision range. When no <revision range> is specified, it defaults to HEAD (i.e. the whole history leading to the current commit). origin..HEAD specifies all the commits reachable from the current commit (i.e. HEAD), but not from origin. For a complete list of ways to spell <revision range>, see the Specifying Ranges section of **gitrevisions(7)**.

[--] <path>...
: Consider only commits that are enough to explain how the files that match the specified paths came to be.

  Paths may need to be prefixed with -- to separate them from options or the revision range, when confusion arises.

MAPPING AUTHORS

The .mailmap feature is used to coalesce together commits by the same person in the shortlog, where their name and/or email address was spelled differently.

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.

GIT

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