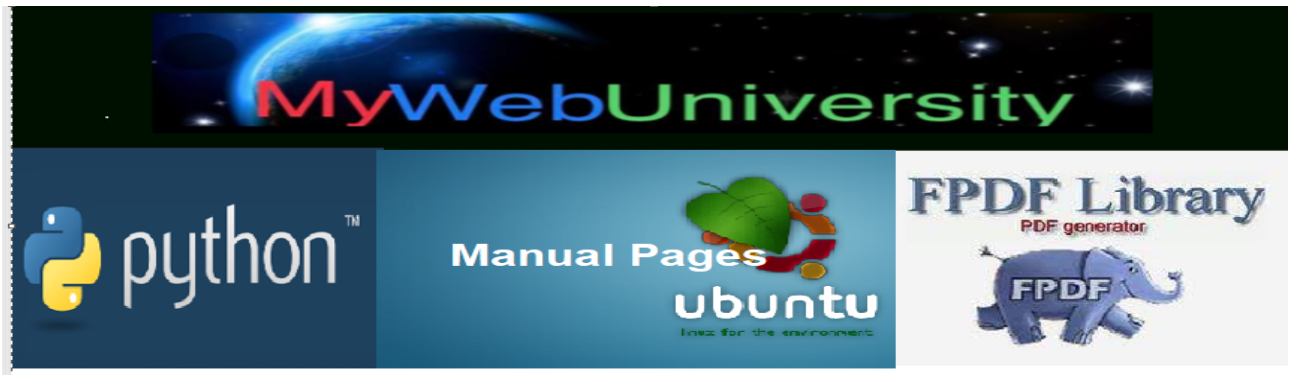

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'git-interpret-trailers.1'

$ man git-interpret-trailers.1

GIT-INTERPRET-TRAILERS(1) Git Manual GIT-INTERPRET-TRAILERS(1)

NAME

git-interpret-trailers - Add or parse structured information in commit messages

SYNOPSIS

git interpret-trailers [--in-place] [--trim-empty]

[(--trailer (<key>|<key-alias>)[(=|:)<value>])...]

[--parse] [<file>...]

DESCRIPTION

Add or parse trailer lines that look similar to RFC 822 e-mail headers, at the end of the otherwise free-form part of a commit message. For example, in the following commit message

subject

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Signed-off-by: Alice <alice@example.com>

Signed-off-by: Bob <bob@example.com>

the last two lines starting with "Signed-off-by" are trailers.

This command reads commit messages from either the <file> arguments or the standard input if no <file> is specified. If --parse is specified, the output consists of the parsed trailers coming from the input, without influencing them with any command line options or configuration variables.

Otherwise, this command applies trailer.* configuration variables (which could potentially add new trailers, as well as reposition them), as well as any command line arguments that can override configuration variables (such as --trailer=... which could also add new trailers), to each input file. The result is emitted on the standard output.

This command can also operate on the output of git-format-patch(1), which is more elaborate than a plain commit message. Namely, such output includes a commit message (as above), a "---" divider line, and a patch part. For these inputs, the divider and patch parts are not modified by this command and are emitted as is on the output, unless --no-divider is specified.

Some configuration variables control the way the --trailer arguments are applied to each input and the way any existing trailer in the input is changed. They also make it possible to automatically add some trailers.

By default, a <key>=<value> or <key>:<value> argument given using --trailer will be appended after the existing trailers only if the last trailer has a different (<key>, <value>) pair (or if there is no existing trailer). The <key> and <value> parts will be trimmed to remove starting and trailing whitespace, and the resulting trimmed <key> and <value> will appear in the output like this:

```
key: value
```

This means that the trimmed <key> and <value> will be separated by ': ' (one colon followed by one space).

For convenience, a <key-alias> can be configured to make using --trailer shorter to type on the command line. This can be configured using the trailer.<key-alias>.key configuration variable. The <keyAlias> must be a prefix of the full <key> string, although case sensitivity does not matter. For example, if you have

```
trailer.sign.key "Signed-off-by: "
```

in your configuration, you only need to specify --trailer="sign: foo" on the command line instead of --trailer="Signed-off-by: foo".

By default the new trailer will appear at the end of all the existing trailers. If there is no existing trailer, the new trailer will appear at the end of the input. A blank line will be added before the new trailer if there isn?t one already.

Existing trailers are extracted from the input by looking for a group of one or more lines that (i) is all trailers, or (ii) contains at least one Git-generated or user-configured trailer and consists of at least 25% trailers. The group must be preceded by one or more empty (or whitespace-only) lines. The group must either be at the end of the input or be the last non-whitespace lines before a line that starts with --- (followed by a space or the end of the line).

When reading trailers, there can be no whitespace before or inside the <key>, but any number

of regular space and tab characters are allowed between the <key> and the separator. There can be whitespaces before, inside or after the <value>. The <value> may be split over multiple lines with each subsequent line starting with at least one whitespace, like the "folding" in RFC 822. Example:

```
key: This is a very long value, with spaces and
  newlines in it.
```

Note that trailers do not follow (nor are they intended to follow) many of the rules for RFC 822 headers. For example they do not follow the encoding rule.

OPTIONS

--in-place

Edit the files in place.

--trim-empty

If the <value> part of any trailer contains only whitespace, the whole trailer will be removed from the output. This applies to existing trailers as well as new trailers.

--trailer <key>[(=|:)<value>]

Specify a (<key>, <value>) pair that should be applied as a trailer to the inputs. See the description of this command.

--where <placement>, --no-where

Specify where all new trailers will be added. A setting provided with --where overrides the trailer.where and any applicable trailer.<keyAlias>.where configuration variables and applies to all --trailer options until the next occurrence of --where or --no-where. Upon encountering --no-where, clear the effect of any previous use of --where, such that the relevant configuration variables are no longer overridden. Possible placements are after, before, end or start.

--if-exists <action>, --no-if-exists

Specify what action will be performed when there is already at least one trailer with the same <key> in the input. A setting provided with --if-exists overrides the trailer.ifExists and any applicable trailer.<keyAlias>.ifExists configuration variables and applies to all --trailer options until the next occurrence of --if-exists or --no-if-exists. Upon encountering --no-if-exists, clear the effect of any previous use of --if-exists, such that the relevant configuration variables are no longer overridden. Possible actions are addIfDifferent, addIfDifferentNeighbor, add, replace and doNothing.

--if-missing <action>, --no-if-missing

Specify what action will be performed when there is no other trailer with the same <key> in the input. A setting provided with --if-missing overrides the trailer.ifMissing and any applicable trailer.<keyAlias>.ifMissing configuration variables and applies to all --trailer options until the next occurrence of --if-missing or --no-if-missing. Upon encountering --no-if-missing, clear the effect of any previous use of --if-missing, such that the relevant configuration variables are no longer overridden. Possible actions are doNothing or add.

--only-trailers

Output only the trailers, not any other parts of the input.

--only-input

Output only trailers that exist in the input; do not add any from the command-line or by applying trailer.* configuration variables.

--unfold

If a trailer has a value that runs over multiple lines (aka "folded"), reformat the value into a single line.

--parse

A convenience alias for --only-trailers --only-input --unfold. This makes it easier to only see the trailers coming from the input without influencing them with any command line options or configuration variables, while also making the output machine-friendly with --unfold.

--no-divider

Do not treat --- as the end of the commit message. Use this when you know your input contains just the commit message itself (and not an email or the output of git format-patch).

CONFIGURATION VARIABLES

Everything below this line in this section is selectively included from the git-config(1) documentation. The content is the same as what?s found there:

trailer.separators

This option tells which characters are recognized as trailer separators. By default only : is recognized as a trailer separator, except that = is always accepted on the command line for compatibility with other git commands.

The first character given by this option will be the default character used when another separator is not specified in the config for this trailer.

For example, if the value for this option is "%=$", then only lines using the format <key><sep><value> with <sep> containing %, = or $ and then spaces will be considered trailers. And % will be the default separator used, so by default trailers will appear like: <key>% <value> (one percent sign and one space will appear between the key and the value).

trailer.where

This option tells where a new trailer will be added.

This can be end, which is the default, start, after or before.

If it is end, then each new trailer will appear at the end of the existing trailers.

If it is start, then each new trailer will appear at the start, instead of the end, of the existing trailers.

If it is after, then each new trailer will appear just after the last trailer with the same <key>.

If it is before, then each new trailer will appear just before the first trailer with the same <key>.

trailer.ifexists

This option makes it possible to choose what action will be performed when there is already at least one trailer with the same <key> in the input.

The valid values for this option are: addIfDifferentNeighbor (this is the default), addIfDifferent, add, replace or doNothing.

With addIfDifferentNeighbor, a new trailer will be added only if no trailer with the same (<key>, <value>) pair is above or below the line where the new trailer will be added.

With addIfDifferent, a new trailer will be added only if no trailer with the same (<key>, <value>) pair is already in the input.

With add, a new trailer will be added, even if some trailers with the same (<key>, <value>) pair are already in the input.

With replace, an existing trailer with the same <key> will be deleted and the new trailer will be added. The deleted trailer will be the closest one (with the same <key>) to the place where the new one will be added.

With doNothing, nothing will be done; that is no new trailer will be added if there is already one with the same <key> in the input.

trailer.ifmissing

This option makes it possible to choose what action will be performed when there is not yet any trailer with the same <key> in the input.

The valid values for this option are: add (this is the default) and doNothing.

With add, a new trailer will be added.

With doNothing, nothing will be done.

trailer.<keyAlias>.key

Defines a <keyAlias> for the <key>. The <keyAlias> must be a prefix (case does not matter) of the <key>. For example, in git config trailer.ack.key "Acked-by" the "Acked-by" is the <key> and the "ack" is the <keyAlias>. This configuration allows the shorter --trailer "ack:..." invocation on the command line using the "ack" <keyAlias> instead of the longer --trailer "Acked-by:...".

At the end of the <key>, a separator can appear and then some space characters. By default the only valid separator is :, but this can be changed using the trailer.separators config variable.

If there is a separator in the key, then it overrides the default separator when adding the trailer.

trailer.<keyAlias>.where

This option takes the same values as the trailer.where configuration variable and it overrides what is specified by that option for trailers with the specified <keyAlias>.

trailer.<keyAlias>.ifexists

This option takes the same values as the trailer.ifexists configuration variable and it overrides what is specified by that option for trailers with the specified <keyAlias>.

trailer.<keyAlias>.ifmissing

This option takes the same values as the trailer.ifmissing configuration variable and it overrides what is specified by that option for trailers with the specified <keyAlias>.

trailer.<keyAlias>.command

Deprecated in favor of trailer.<keyAlias>.cmd. This option behaves in the same way as trailer.<keyAlias>.cmd, except that it doesn?t pass anything as argument to the specified command. Instead the first occurrence of substring $ARG is replaced by the <value> that would be passed as argument.

Note that $ARG in the user?s command is only replaced once and that the original way of replacing $ARG is not safe.

When both trailer.<keyAlias>.cmd and trailer.<keyAlias>.command are given for the same

<keyAlias>, trailer.<keyAlias>.cmd is used and trailer.<keyAlias>.command is ignored.

trailer.<keyAlias>.cmd

This option can be used to specify a shell command that will be called once to automatically add a trailer with the specified <keyAlias>, and then called each time a --trailer <keyAlias>=<value> argument is specified to modify the <value> of the trailer that this option would produce.

When the specified command is first called to add a trailer with the specified <keyAlias>, the behavior is as if a special --trailer <keyAlias>=<value> argument was added at the beginning of the "git interpret-trailers" command, where <value> is taken to be the standard output of the command with any leading and trailing whitespace trimmed off.

If some --trailer <keyAlias>=<value> arguments are also passed on the command line, the command is called again once for each of these arguments with the same <keyAlias>. And the <value> part of these arguments, if any, will be passed to the command as its first argument. This way the command can produce a <value> computed from the <value> passed in the --trailer <keyAlias>=<value> argument.

EXAMPLES

? Configure a sign trailer with a Signed-off-by key, and then add two of these trailers to a commit message file:

```
$ git config trailer.sign.key "Signed-off-by"
$ cat msg.txt
subject
body text
$ git interpret-trailers --trailer 'sign: Alice <alice@example.com>' --trailer 'sign: Bob <bob@example.com>' <msg.txt
subject
body text
Signed-off-by: Alice <alice@example.com>
Signed-off-by: Bob <bob@example.com>
```

? Use the --in-place option to edit a commit message file in place:

```
$ cat msg.txt
subject
body text
Signed-off-by: Bob <bob@example.com>
```

```
$ git interpret-trailers --trailer 'Acked-by: Alice <alice@example.com>' --in-place msg.txt
$ cat msg.txt
subject
body text
Signed-off-by: Bob <bob@example.com>
Acked-by: Alice <alice@example.com>
```

? Extract the last commit as a patch, and add a Cc and a Reviewed-by trailer to it:

```
$ git format-patch -1
0001-foo.patch
$ git interpret-trailers --trailer 'Cc: Alice <alice@example.com>' --trailer 'Reviewed-by: Bob <bob@example.com>'
0001-foo.patch >0001-bar.patch
```

? Configure a sign trailer with a command to automatically add a 'Signed-off-by: ' with the author information only if there is no 'Signed-off-by: ' already, and show how it works:

```
$ cat msg1.txt
subject
body text
$ git config trailer.sign.key "Signed-off-by: "
$ git config trailer.sign.ifmissing add
$ git config trailer.sign.ifexists doNothing
$ git config trailer.sign.cmd 'echo "$(git config user.name) <$(git config user.email)>"'
$ git interpret-trailers --trailer sign <msg1.txt
subject
body text
Signed-off-by: Bob <bob@example.com>
$ cat msg2.txt
subject
body text
Signed-off-by: Alice <alice@example.com>
$ git interpret-trailers --trailer sign <msg2.txt
subject
body text
Signed-off-by: Alice <alice@example.com>
```

? Configure a fix trailer with a key that contains a # and no space after this character, and show how it works:

```
$ git config trailer.separators ":#"
$ git config trailer.fix.key "Fix #"
$ echo "subject" | git interpret-trailers --trailer fix=42
subject
Fix #42
```

? Configure a help trailer with a cmd use a script glog-find-author which search specified author identity from git log in git repository and show how it works:

```
$ cat ~/bin/glog-find-author
#!/bin/sh
test -n "$1" && git log --author="$1" --pretty="%an <%ae>" -1 || true
$ cat msg.txt
subject
body text
$ git config trailer.help.key "Helped-by: "
$ git config trailer.help.ifExists "addIfDifferentNeighbor"
$ git config trailer.help.cmd "~/bin/glog-find-author"
$ git interpret-trailers --trailer="help:Junio" --trailer="help:Couder" <msg.txt
subject
body text
Helped-by: Junio C Hamano <gitster@pobox.com>
Helped-by: Christian Couder <christian.couder@gmail.com>
```

? Configure a ref trailer with a cmd use a script glog-grep to grep last relevant commit from git log in the git repository and show how it works:

```
$ cat ~/bin/glog-grep
#!/bin/sh
test -n "$1" && git log --grep "$1" --pretty=reference -1 || true
$ cat msg.txt
subject
body text
$ git config trailer.ref.key "Reference-to: "
$ git config trailer.ref.ifExists "replace"
```

```
$ git config trailer.ref.cmd "~/bin/glog-grep"
$ git interpret-trailers --trailer="ref:Add copyright notices." <msg.txt
subject
body text
Reference-to: 8bc9a0c769 (Add copyright notices., 2005-04-07)
```

? Configure a see trailer with a command to show the subject of a commit that is related, and show how it works:

```
$ cat msg.txt
subject
body text
see: HEAD~2
$ cat ~/bin/glog-ref
#!/bin/sh
git log -1 --oneline --format="%h (%s)" --abbrev-commit --abbrev=14
$ git config trailer.see.key "See-also: "
$ git config trailer.see.ifExists "replace"
$ git config trailer.see.ifMissing "doNothing"
$ git config trailer.see.cmd "glog-ref"
$ git interpret-trailers --trailer=see <msg.txt
subject
body text
See-also: fe3187489d69c4 (subject of related commit)
```

? Configure a commit template with some trailers with empty values (using sed to show and keep the trailing spaces at the end of the trailers), then configure a commit-msg hook that uses git interpret-trailers to remove trailers with empty values and to add a git-version trailer:

```
$ cat temp.txt
***subject***
***message***
Fixes: Z
Cc: Z
Reviewed-by: Z
Signed-off-by: Z
```

```
$ sed -e 's/ Z$/ /' temp.txt > commit_template.txt
$ git config commit.template commit_template.txt
$ cat .git/hooks/commit-msg
#!/bin/sh
git interpret-trailers --trim-empty --trailer "git-version: \$(git describe)" "\$1" > "\$1.new"
mv "\$1.new" "\$1"
$ chmod +x .git/hooks/commit-msg
```

SEE ALSO

git-commit(1), git-format-patch(1), git-config(1)

GIT

Part of the git(1) suite