NAME

git-commit-tree - Create a new commit object

SYNOPSIS

git commit-tree <tree> [(-p <parent>)...] < changelog
git commit-tree [(-p <parent>)...] [-S[<keyid>]] [(-m <message>)...]
[(-F <file>)...] <tree>

DESCRIPTION

This is usually not what an end user wants to run directly. See **git-commit(1)** instead.

Creates a new commit object based on the provided tree object and emits the new commit object id on stdout. The log message is read from the standard input, unless -m or -F options are given.

A commit object may have any number of parents. With exactly one parent, it is an ordinary commit. Having more than one parent makes the commit a merge between several lines of history. Initial (root) commits have no parents.

While a tree represents a particular directory state of a working directory, a commit represents that state in time, and explains how to get there.

Normally a commit would identify a new HEAD state, and while Git doesn't care where you save the note about that state, in practice we tend to just write the result to the file that is pointed at by .git/HEAD, so that we can always see what the last committed state was.

OPTIONS

<tree>
: An existing tree object

-p <parent>
: Each *-p* indicates the id of a parent commit object.

-m <message>
: A paragraph in the commit log message. This can be given more than once and each <message> becomes its own paragraph.

-F <file>
: Read the commit log message from the given file. Use - to read from the standard input.

-S[<keyid>], --gpg-sign[=<keyid>]
: GPG-sign commit.

--no-gpg-sign
: Countermand commit.gpgsign configuration variable that is set to force each and every commit to be signed.

COMMIT INFORMATION

A commit encapsulates:

- all parent object ids
- author name, email and date
- committer name and email and the commit time.

While parent object ids are provided on the command line, author and committer information is taken from the following environment variables, if set:

GIT_AUTHOR_NAME
GIT_AUTHOR_EMAIL
GIT_AUTHOR_DATE
GIT_COMMITTER_NAME
GIT_COMMITTER_EMAIL
GIT_COMMITTER_DATE

(nb <, > and ns are stripped)

In case (some of) these environment variables are not set, the information is taken from the configuration items user.name and user.email, or, if not present, the environment variable EMAIL, or, if that is not set, system user name and the hostname used for outgoing mail (taken from /etc/mailname and falling back to the fully qualified hostname when that file does not exist).

A commit comment is read from stdin. If a changelog entry is not provided via < redirection, *git commit-tree* will just wait for one to be entered and terminated with ^D.

DATE FORMATS

The GIT_AUTHOR_DATE, GIT_COMMITTER_DATE environment variables support the following date formats:

Git internal format
: It is <unix timestamp> <time zone offset>, where <unix timestamp> is the number of seconds since the UNIX epoch. <time zone offset> is a positive or negative offset from UTC. For example CET (which is 2 hours ahead UTC) is +0200.

RFC 2822
: The standard email format as described by RFC 2822, for example Thu, 07 Apr 2005 22:13:13 +0200.

ISO 8601
: Time and date specified by the ISO 8601 standard, for example 2005-04-07T22:13:13. The parser accepts a space instead of the T character as well.

Note

In addition, the date part is accepted in the following formats: YYYY.MM.DD, MM/DD/YYYY and DD.MM.YYYY.

DISCUSSION

At the core level, Git is character encoding agnostic.

- The pathnames recorded in the index and in the tree objects are treated as uninterpreted sequences of non-NUL bytes. What readdir(2) returns are what are recorded and compared with the data Git keeps track of, which in turn are expected to be what lstat(2) and creat(2) accepts. There is no such thing as pathname encoding translation.
- The contents of the blob objects are uninterpreted sequences of bytes. There is no encoding translation at the core level.
- The commit log messages are uninterpreted sequences of non-NUL bytes.

Although we encourage that the commit log messages are encoded in UTF-8, both the core and Git Porcelain are designed not to force UTF-8 on projects. If all participants of a particular project find it more convenient to use legacy encodings, Git does not forbid it. However, there are a few things to keep in mind.

1. *git commit* and *git commit-tree* issues a warning if the commit log message given to it does not look like a valid UTF-8 string, unless you explicitly say your project uses a legacy encoding. The way to say this is to have i18n.commitencoding in .git/config file, like this:

   [i18n]
   commitencoding = ISO-8859-1

   Commit objects created with the above setting record the value of i18n.commitencoding in its encoding header. This is to help other people who look at them later. Lack of this header implies that the commit log message is encoded in UTF-8.
2. *git log*, *git show*, *git blame* and friends look at the encoding header of a commit object, and try to re-code the log message into UTF-8 unless otherwise specified. You can specify the desired output encoding with i18n.logoutputencoding in .git/config file, like this:

   [i18n]
   logoutputencoding = ISO-8859-1

If you do not have this configuration variable, the value of i18n.commitencoding is used instead.

Note that we deliberately chose not to re-code the commit log message when a commit is made to force UTF-8 at the commit object level, because re-coding to UTF-8 is not necessarily a reversible operation.

FILES

/etc/mailname

SEE ALSO

git-write-tree(1)

GIT

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