

NAME

`git-commit-tree` – Create a new commit object

SYNOPSIS

```
git commit-tree <tree> [(-p <parent>)...]
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.

The **-m** and **-F** options can be given any number of times, in any order. The commit log message will be composed in the order in which the 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. This can be given more than once and the content of each file becomes its own paragraph.

-S[<keyid>], **--gpg-sign**[=<keyid>]

GPG-sign commits. The **keyid** argument is optional and defaults to the committer identity; if specified, it must be stuck to the option without a space.

--no-gpg-sign

Do not GPG-sign commit, to countermand a **--gpg-sign** option given earlier on the command line.

COMMIT INFORMATION

A commit encapsulates:

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

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 1 hour ahead of UTC) is `+0100`.

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

Git is to some extent character encoding agnostic.

- The contents of the blob objects are uninterpreted sequences of bytes. There is no encoding translation at the core level.
- Path names are encoded in UTF-8 normalization form C. This applies to tree objects, the index file, ref names, as well as path names in command line arguments, environment variables and config files (`.git/config` (see `git-config(1)`), `gitignore(5)`, `gitattributes(5)` and `gitmodules(5)`).

Note that Git at the core level treats path names simply as sequences of non-NUL bytes, there are no path name encoding conversions (except on Mac and Windows). Therefore, using non-ASCII path names will mostly work even on platforms and file systems that use legacy extended ASCII encodings. However, repositories created on such systems will not work properly on UTF-8-based systems (e.g. Linux, Mac, Windows) and vice versa. Additionally, many Git-based tools simply assume path names to be UTF-8 and will fail to display other encodings correctly.

- Commit log messages are typically encoded in UTF-8, but other extended ASCII encodings are also supported. This includes ISO-8859-x, CP125x and many others, but *not* UTF-16/32, EBCDIC and CJK multi-byte encodings (GBK, Shift-JIS, Big5, EUC-x, CP9xx etc.).

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-commit(1)**

GIT

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