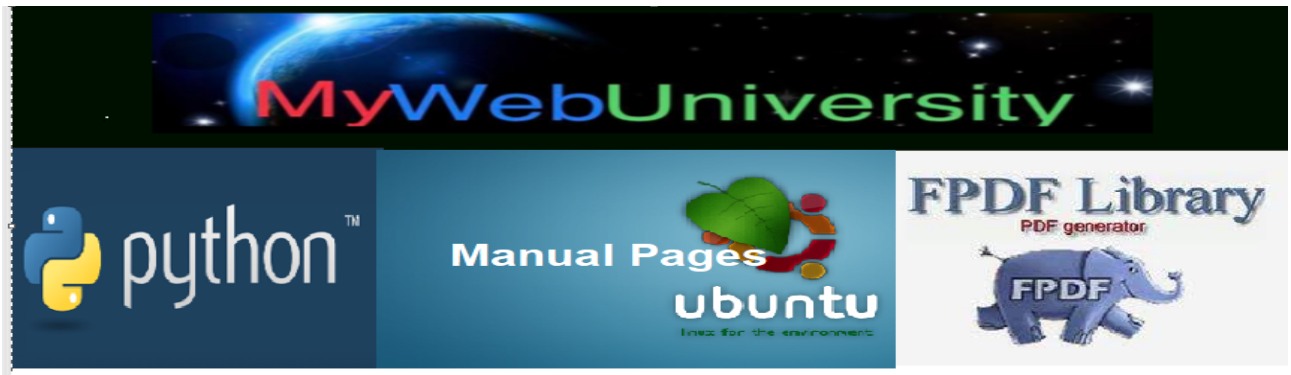

Full credit is given to the above companies including the OS that this PDF file was generated!

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

$ man git-tag.1

GIT-TAG(1) Git Manual GIT-TAG(1)

NAME

git-tag - Create, list, delete or verify tags

SYNOPSIS

git tag [-a | -s | -u <key-id>] [-f] [-m <msg> | -F <file>] [-e]

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

<tagname> [<commit> | <object>]

git tag -d <tagname>...

git tag [-n[<num>]] -l [--contains <commit>] [--no-contains <commit>]

[--points-at <object>] [--column[=<options>] | --no-column]

[--create-reflog] [--sort=<key>] [--format=<format>]

[--merged <commit>] [--no-merged <commit>] [<pattern>...]

git tag -v [--format=<format>] <tagname>...

DESCRIPTION

Add a tag reference in refs/tags/, unless -d/-l/-v is given to delete, list or verify tags.

Unless -f is given, the named tag must not yet exist.

If one of -a, -s, or -u <key-id> is passed, the command creates a tag object, and requires a tag message. Unless -m <msg> or -F <file> is given, an editor is started for the user to type in the tag message.

If -m <msg> or -F <file> or --trailer <token>[=<value>] is given and -a, -s, and -u <key-id> are absent, -a is implied.

Otherwise, a tag reference that points directly at the given object (i.e., a lightweight tag) is created.

A cryptographically signed tag object will be created when -s or -u <key-id> is used. The signing backend (GPG, X.509, SSH, etc.) is controlled by the gpg.format configuration variable, defaulting to OpenPGP. When -u <key-id> is not used, the committer identity for the current user is used to find the key for signing. The configuration variable gpg.program is used to specify a custom signing binary.

Tag objects (created with -a, -s, or -u) are called "annotated" tags; they contain a creation date, the tagger name and e-mail, a tagging message, and an optional cryptographic signature. Whereas a "lightweight" tag is simply a name for an object (usually a commit object).

Annotated tags are meant for release while lightweight tags are meant for private or temporary object labels. For this reason, some git commands for naming objects (like git describe) will ignore lightweight tags by default.

OPTIONS

-a, --annotate

Make an unsigned, annotated tag object

-s, --sign

Make a cryptographically signed tag, using the default signing key. The signing backend used depends on the gpg.format configuration variable. The default key is determined by the backend. For GPG, it?s based on the committer?s email address, while for SSH it may be a specific key file or agent identity. See git-config(1).

--no-sign

Override tag.gpgSign configuration variable that is set to force each and every tag to be signed.

-u <key-id>, --local-user=<key-id>

Make a cryptographically signed tag using the given key. The format of the <key-id> and the backend used depend on the gpg.format configuration variable. See git-config(1).

-f, --force

Replace an existing tag with the given name (instead of failing)

-d, --delete

Delete existing tags with the given names.

-v, --verify

Verify the cryptographic signature of the given tags.

-n<num>

<num> specifies how many lines from the annotation, if any, are printed when using -l.

Implies --list.

The default is not to print any annotation lines. If no number is given to -n, only the first line is printed. If the tag is not annotated, the commit message is displayed instead.

-l, --list

List tags. With optional <pattern>..., e.g. git tag --list 'v-*', list only the tags that match the pattern(s).

Running git tag without arguments also lists all tags. The pattern is a shell wildcard (i.e., matched using fnmatch(3)). Multiple patterns may be given; if any of them matches, the tag is shown.

This option is implicitly supplied if any other list-like option such as --contains is provided. See the documentation for each of those options for details.

--sort=<key>

Sort based on the key given. Prefix - to sort in descending order of the value. You may use the --sort=<key> option multiple times, in which case the last <key> becomes the primary key. Also supports "version:refname" or "v:refname" (tag names are treated as versions). The "version:refname" sort order can also be affected by the "versionsort.suffix" configuration variable. The keys supported are the same as those in git for-each-ref. Sort order defaults to the value configured for the tag.sort variable if it exists, or lexicographic order otherwise. See git-config(1).

--color[=<when>]

Respect any colors specified in the --format option. The <when> field must be one of always, never, or auto (if <when> is absent, behave as if always was given).

-i, --ignore-case

Sorting and filtering tags are case insensitive.

--omit-empty

Do not print a newline after formatted refs where the format expands to the empty string.

--column[=<options>], --no-column

Display tag listing in columns. See configuration variable column.tag for option syntax. --column and --no-column without options are equivalent to always and never respectively.

This option is only applicable when listing tags without annotation lines.

--contains [<commit>]

Only list tags which contain <commit> (HEAD if not specified). Implies --list.

--no-contains [<commit>]

Only list tags which don?t contain <commit> (HEAD if not specified). Implies --list.

--merged [<commit>]

Only list tags whose commits are reachable from <commit> (HEAD if not specified).

--no-merged [<commit>]

Only list tags whose commits are not reachable from <commit> (HEAD if not specified).

--points-at [<object>]

Only list tags of <object> (HEAD if not specified). Implies --list.

-m <msg>, --message=<msg>

Use <msg> (instead of prompting). If multiple -m options are given, their values are concatenated as separate paragraphs. Implies -a if none of -a, -s, or -u <key-id> is given.

-F <file>, --file=<file>

Take the tag message from <file>. Use - to read the message from the standard input. Implies -a if none of -a, -s, or -u <key-id> is given.

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

Specify a (<token>, <value>) pair that should be applied as a trailer. (e.g. git tag --trailer "Custom-Key: value" will add a "Custom-Key" trailer to the tag message.) The trailer.* configuration variables (git-interpret-trailers(1)) can be used to define if a duplicated trailer is omitted, where in the run of trailers each trailer would appear, and other details. The trailers can be extracted in git tag --list, using --format="%(trailers)" placeholder.

-e, --edit

Let further edit the message taken from file with -F and command line with -m.

--cleanup=<mode>

Set how the tag message is cleaned up. The <mode> can be one of verbatim, whitespace and strip. The strip mode is default. The verbatim mode does not change message at all, whitespace removes just leading/trailing whitespace lines and strip removes both whitespace and commentary.

--create-reflog

Create a reflog for the tag. To globally enable reflogs for tags, see core.logAllRefUpdates in git-config(1). The negated form --no-create-reflog only overrides an earlier --create-reflog, but currently does not negate the setting of core.logAllRefUpdates.

--format=<format>

A string that interpolates %(fieldname) from a tag ref being shown and the object it points at. The format is the same as that of git-for-each-ref(1). When unspecified, defaults to %(refname:strip=2).

<tagname>

The name of the tag to create, delete, or describe. The new tag name must pass all checks defined by git-check-ref-format(1). Some of these checks may restrict the characters allowed in a tag name.

<commit>, <object>

The object that the new tag will refer to, usually a commit. Defaults to HEAD.

CONFIGURATION

By default, git tag in sign-with-default mode (-s) will use your committer identity (of the form Your Name <your@email.address>) to find a key. If you want to use a different default key, you can specify it in the repository configuration as follows:

```
[user]
    signingKey = <key-id>
```

The signing backend can be chosen via the gpg.format configuration variable, which defaults to openpgp. See git-config(1) for a list of other supported formats.

The path to the program used for each signing backend can be specified with the gpg.<format>.program configuration variable. For the openpgp backend, gpg.program can be used as a synonym for gpg.openpgp.program. See git-config(1) for details.

pager.tag is only respected when listing tags, i.e., when -l is used or implied. The default is to use a pager.

See git-config(1) for more details and other configuration variables.

DISCUSSION

On Re-tagging

What should you do when you tag a wrong commit and you would want to re-tag?

If you never pushed anything out, just re-tag it. Use -f to replace the old one. And you?re done.

But if you have pushed things out (or others could just read your repository directly), then others will have already seen the old tag. In that case you can do one of two things:

1. The sane thing. Just admit you screwed up, and use a different name. Others have already seen one tag-name, and if you keep the same name, you may be in the situation that two people both have "version X", but they actually have different "X"'s. So just call it "X.1" and be done with it.
2. The insane thing. You really want to call the new version "X" too, even though others have already seen the old one. So just use git tag -f again, as if you hadn?t already published the old one.

However, Git does not (and it should not) change tags behind users back. So if somebody already got the old tag, doing a git pull on your tree shouldn?t just make them overwrite the old one.

If somebody got a release tag from you, you cannot just change the tag for them by updating your own one. This is a big security issue, in that people MUST be able to trust their tag-names. If you really want to do the insane thing, you need to just fess up to it, and tell people that you messed up. You can do that by making a very public announcement saying:

> Ok, I messed up, and I pushed out an earlier version tagged as X. I
> then fixed something, and retagged the *fixed* tree as X again.
> If you got the wrong tag, and want the new one, please delete
> the old one and fetch the new one by doing:
>
> ```
> git tag -d X
> git fetch origin tag X
> ```
>
> to get my updated tag.
> You can test which tag you have by doing
>
> ```
> git rev-parse X
> ```
>
> which should return 0123456789abcdef.. if you have the new version.
> Sorry for the inconvenience.

Does this seem a bit complicated? It should be. There is no way that it would be correct to just "fix" it automatically. People need to know that their tags might have been changed.

On Automatic following

If you are following somebody else?s tree, you are most likely using remote-tracking branches (eg. refs/remotes/origin/master). You usually want the tags from the other end.

On the other hand, if you are fetching because you would want a one-shot merge from somebody

else, you typically do not want to get tags from there. This happens more often for people near the toplevel but not limited to them. Mere mortals when pulling from each other do not necessarily want to automatically get private anchor point tags from the other person.

Often, "please pull" messages on the mailing list just provide two pieces of information: a repo URL and a branch name; this is designed to be easily cut&pasted at the end of a git fetch command line:

    Linus, please pull from

        git://git..../proj.git master

    to get the following updates...

becomes:

    $ git pull git://git..../proj.git master

In such a case, you do not want to automatically follow the other person?s tags.

One important aspect of Git is its distributed nature, which largely means there is no inherent "upstream" or "downstream" in the system. On the face of it, the above example might seem to indicate that the tag namespace is owned by the upper echelon of people and that tags only flow downwards, but that is not the case. It only shows that the usage pattern determines who are interested in whose tags.

A one-shot pull is a sign that a commit history is now crossing the boundary between one circle of people (e.g. "people who are primarily interested in the networking part of the kernel") who may have their own set of tags (e.g. "this is the third release candidate from the networking group to be proposed for general consumption with 2.6.21 release") to another circle of people (e.g. "people who integrate various subsystem improvements"). The latter are usually not interested in the detailed tags used internally in the former group (that is what "internal" means). That is why it is desirable not to follow tags automatically in this case.

It may well be that among networking people, they may want to exchange the tags internal to their group, but in that workflow they are most likely tracking each other?s progress by having remote-tracking branches. Again, the heuristic to automatically follow such tags is a good thing.

On Backdating Tags

If you have imported some changes from another VCS and would like to add tags for major releases of your work, it is useful to be able to specify the date to embed inside of the tag object; such data in the tag object affects, for example, the ordering of tags in the

gitweb interface.

To set the date used in future tag objects, set the environment variable GIT_COMMITTER_DATE (see the later discussion of possible values; the most common form is "YYYY-MM-DD HH:MM").

For example:

```
$ GIT_COMMITTER_DATE="2006-10-02 10:31" git tag -s v1.0.1
```

DATE FORMATS

The GIT_AUTHOR_DATE and 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 date 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. Fractional parts of a second will be ignored, for example 2005-04-07T22:13:13.019 will be treated as 2005-04-07T22:13:13.

Note

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

FILES

$GIT_DIR/TAG_EDITMSG

This file contains the message of an in-progress annotated tag. If git tag exits due to an error before creating an annotated tag then the tag message that has been provided by the user in an editor session will be available in this file, but may be overwritten by the next invocation of git tag.

CONFIGURATION

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:

tag.forceSignAnnotated

A boolean to specify whether annotated tags created should be GPG signed. If --annotate is specified on the command line, it takes precedence over this option.

tag.sort

This variable controls the sort ordering of tags when displayed by git-tag. Without the --sort=<value> option provided, the value of this variable will be used as the default.

tag.gpgSign

A boolean to specify whether all tags should be GPG signed. Use of this option when running in an automated script can result in a large number of tags being signed. It is therefore convenient to use an agent to avoid typing your GPG passphrase several times. Note that this option doesn?t affect tag signing behavior enabled by -u <keyid> or --local-user=<keyid> options.

NOTES

When combining multiple --contains and --no-contains filters, only references that contain at least one of the --contains commits and contain none of the --no-contains commits are shown.

When combining multiple --merged and --no-merged filters, only references that are reachable from at least one of the --merged commits and from none of the --no-merged commits are shown.

SEE ALSO

git-check-ref-format(1). git-config(1).

GIT

Part of the git(1) suite