

NAME

`git-describe` – Give an object a human readable name based on an available ref

SYNOPSIS

```
git describe [--all] [--tags] [--contains] [--abbrev=<n>] [<commit-ish>...]
git describe [--all] [--tags] [--contains] [--abbrev=<n>] --dirty[=<mark>]
git describe <blob>
```

DESCRIPTION

The command finds the most recent tag that is reachable from a commit. If the tag points to the commit, then only the tag is shown. Otherwise, it suffixes the tag name with the number of additional commits on top of the tagged object and the abbreviated object name of the most recent commit. The result is a "human-readable" object name which can also be used to identify the commit to other git commands.

By default (without `--all` or `--tags`) **git describe** only shows annotated tags. For more information about creating annotated tags see the `-a` and `-s` options to **git-tag**(1).

If the given object refers to a blob, it will be described as **<commit-ish>:<path>**, such that the blob can be found at **<path>** in the **<commit-ish>**, which itself describes the first commit in which this blob occurs in a reverse revision walk from HEAD.

OPTIONS

<commit-ish>...

Commit-ish object names to describe. Defaults to HEAD if omitted.

--dirty[=<mark>], --broken[=<mark>]

Describe the state of the working tree. When the working tree matches HEAD, the output is the same as "git describe HEAD". If the working tree has local modification "-dirty" is appended to it. If a repository is corrupt and Git cannot determine if there is local modification, Git will error out, unless '`--broken`' is given, which appends the suffix "-broken" instead.

--all

Instead of using only the annotated tags, use any ref found in **refs/** namespace. This option enables matching any known branch, remote-tracking branch, or lightweight tag.

--tags

Instead of using only the annotated tags, use any tag found in **refs/tags** namespace. This option enables matching a lightweight (non-annotated) tag.

--contains

Instead of finding the tag that predates the commit, find the tag that comes after the commit, and thus contains it. Automatically implies `--tags`.

--abbrev=<n>

Instead of using the default 7 hexadecimal digits as the abbreviated object name, use `<n>` digits, or as many digits as needed to form a unique object name. An `<n>` of 0 will suppress long format, only showing the closest tag.

--candidates=<n>

Instead of considering only the 10 most recent tags as candidates to describe the input commit-ish consider up to `<n>` candidates. Increasing `<n>` above 10 will take slightly longer but may produce a more accurate result. An `<n>` of 0 will cause only exact matches to be output.

--exact-match

Only output exact matches (a tag directly references the supplied commit). This is a synonym for `--candidates=0`.

--debug

Verbosely display information about the searching strategy being employed to standard error. The tag name will still be printed to standard out.

--long

Always output the long format (the tag, the number of commits and the abbreviated commit name) even when it matches a tag. This is useful when you want to see parts of the commit object name in "describe" output, even when the commit in question happens to be a tagged version. Instead of just emitting the tag name, it will describe such a commit as v1.2-0-gdeadbee (0th commit since tag v1.2 that points at object deadbee....).

--match <pattern>

Only consider tags matching the given **glob(7)** pattern, excluding the "refs/tags/" prefix. If used with **--all**, it also considers local branches and remote-tracking references matching the pattern, excluding respectively "refs/heads/" and "refs/remotes/" prefix; references of other types are never considered. If given multiple times, a list of patterns will be accumulated, and tags matching any of the patterns will be considered. Use **--no-match** to clear and reset the list of patterns.

--exclude <pattern>

Do not consider tags matching the given **glob(7)** pattern, excluding the "refs/tags/" prefix. If used with **--all**, it also does not consider local branches and remote-tracking references matching the pattern, excluding respectively "refs/heads/" and "refs/remotes/" prefix; references of other types are never considered. If given multiple times, a list of patterns will be accumulated and tags matching any of the patterns will be excluded. When combined with **--match** a tag will be considered when it matches at least one **--match** pattern and does not match any of the **--exclude** patterns. Use **--no-exclude** to clear and reset the list of patterns.

--always

Show uniquely abbreviated commit object as fallback.

--first-parent

Follow only the first parent commit upon seeing a merge commit. This is useful when you wish to not match tags on branches merged in the history of the target commit.

EXAMPLES

With something like git.git current tree, I get:

```
[torvalds@g5 git]$ git describe parent
v1.0.4-14-g2414721
```

i.e. the current head of my "parent" branch is based on v1.0.4, but since it has a few commits on top of that, describe has added the number of additional commits ("14") and an abbreviated object name for the commit itself ("2414721") at the end.

The number of additional commits is the number of commits which would be displayed by "git log v1.0.4..parent". The hash suffix is "-g" + 7-char abbreviation for the tip commit of parent (which was **2414721b194453f058079d897d13c4e377f92dc6**). The "g" prefix stands for "git" and is used to allow describing the version of a software depending on the SCM the software is managed with. This is useful in an environment where people may use different SCMs.

Doing a *git describe* on a tag-name will just show the tag name:

```
[torvalds@g5 git]$ git describe v1.0.4
v1.0.4
```

With **--all**, the command can use branch heads as references, so the output shows the reference path as well:

```
[torvalds@g5 git]$ git describe --all --abbrev=4 v1.0.5^2
tags/v1.0.0-21-g975b
```

```
[torvalds@g5 git]$ git describe --all --abbrev=4 HEAD^
heads/lt/describe-7-g975b
```

With `--abbrev` set to 0, the command can be used to find the closest tagname without any suffix:

```
[torvalds@g5 git]$ git describe --abbrev=0 v1.0.5^2
tags/v1.0.0
```

Note that the suffix you get if you type these commands today may be longer than what Linus saw above when he ran these commands, as your Git repository may have new commits whose object names begin with 975b that did not exist back then, and "-g975b" suffix alone may not be sufficient to disambiguate these commits.

SEARCH STRATEGY

For each commit-ish supplied, *git describe* will first look for a tag which tags exactly that commit. Annotated tags will always be preferred over lightweight tags, and tags with newer dates will always be preferred over tags with older dates. If an exact match is found, its name will be output and searching will stop.

If an exact match was not found, *git describe* will walk back through the commit history to locate an ancestor commit which has been tagged. The ancestor's tag will be output along with an abbreviation of the input commit-ish's SHA-1. If **--first-parent** was specified then the walk will only consider the first parent of each commit.

If multiple tags were found during the walk then the tag which has the fewest commits different from the input commit-ish will be selected and output. Here fewest commits different is defined as the number of commits which would be shown by **git log tag..input** will be the smallest number of commits possible.

BUGS

Tree objects as well as tag objects not pointing at commits, cannot be described. When describing blobs, the lightweight tags pointing at blobs are ignored, but the blob is still described as <committ-ish>:<path> despite the lightweight tag being favorable.

GIT

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