# 2.6 Git Basics - Tagging

## Tagging

Like most VCSs, Git has the ability to tag specific points in a repository's history as being important. Typically, people use this functionality to mark release points (`v1.0`, `v2.0` and so on). In this section, you'll learn how to list existing tags, how to create and delete tags, and what the different types of tags are.

### Listing Your Tags

Listing the existing tags in Git is straightforward. Just type `git tag` (with optional `-l` or `--list`):

```
$ git tag
v1.0
v2.0
```

This command lists the tags in alphabetical order; the order in which they are displayed has no real importance.

You can also search for tags that match a particular pattern. The Git source repo, for instance, contains more than 500 tags. If you're interested only in looking at the 1.8.5 series, you can run this:

```
$ git tag -l "v1.8.5*"
v1.8.5
v1.8.5-rc0
v1.8.5-rc1
v1.8.5-rc2
v1.8.5-rc3
v1.8.5.1
v1.8.5.2
v1.8.5.3
v1.8.5.4
v1.8.5.5
```

Listing tag wildcards requires `-l` or `--list` option

Note

If you want just the entire list of tags, running the command `git tag` implicitly assumes you want a listing and provides one; the use of `-l` or `--list` in this case is optional.

If, however, you're supplying a wildcard pattern to match tag names, the use of `-l` or `--list` is mandatory.

### Creating Tags

Git supports two types of tags: *lightweight* and *annotated*.

A lightweight tag is very much like a branch that doesn't change — it's just a pointer to a specific commit.

Annotated tags, however, are stored as full objects in the Git database. They're checksummed; contain the tagger name, email, and date; have a tagging message; and can be signed and verified with GNU Privacy Guard (GPG). It's generally recommended that you create annotated tags so you can have all this information; but if you want a temporary tag or for some reason don't want to keep the other information, lightweight tags are available too.

### Annotated Tags

Creating an annotated tag in Git is simple. The easiest way is to specify `-a` when you run the `tag` command:

```
$ git tag -a v1.4 -m "my version 1.4"
$ git tag
v0.1
v1.3
v1.4
```

The `-m` specifies a tagging message, which is stored with the tag. If you don't specify a message for an annotated tag, Git launches your editor so you can type it in.

You can see the tag data along with the commit that was tagged by using the `git show` command:

```
$ git show v1.4
tag v1.4
Tagger: Ben Straub <ben@straub.cc>
Date:   Sat May 3 20:19:12 2014 -0700

my version 1.4

commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number
```

That shows the tagger information, the date the commit was tagged, and the annotation message before showing the commit information.

## Lightweight Tags

Another way to tag commits is with a lightweight tag. This is basically the commit checksum stored in a file — no other information is kept. To create a lightweight tag, don't supply any of the `-a`, `-s`, or `-m` options, just provide a tag name:

```
$ git tag v1.4-lw
$ git tag
v0.1
v1.3
v1.4
v1.4-lw
v1.5
```

This time, if you run `git show` on the tag, you don't see the extra tag information. The command just shows the commit:

```
$ git show v1.4-lw
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number
```

## Tagging Later

You can also tag commits after you've moved past them. Suppose your commit history looks like this:

```
$ git log --pretty=oneline
15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch 'experiment'
a6b4c97498bd301d84096da251c98a07c7723e65 Create write support
0d52aaab4479697da7686c15f77a3d64d9165190 One more thing
6d52a271eda8725415634dd79daabbc4d9b6008e Merge branch 'experiment'
0b7434d86859cc7b8c3d5e1dddfed66ff742fcbc Add commit function
4682c3261057305bdd616e23b64b0857d832627b Add todo file
166ae0c4d3f420721acbb115cc33848dfcc2121a Create write support
9fceb02d0ae598e95dc970b74767f19372d61af8 Update rakefile
```

```
964f16d36dfccde844893cac5b347e7b3d44abbc Commit the todo
8a5cbc430f1a9c3d00faaeffd07798508422908a Update readme
```

Now, suppose you forgot to tag the project at v1.2, which was at the “Update rakefile” commit. You can add it after the fact. To tag that commit, you specify the commit checksum (or part of it) at the end of the command:

```
$ git tag -a v1.2 9fceb02
```

You can see that you’ve tagged the commit:

```
$ git tag
v0.1
v1.2
v1.3
v1.4
v1.4-lw
v1.5

$ git show v1.2
tag v1.2
Tagger: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Feb 9 15:32:16 2009 -0800

version 1.2
commit 9fceb02d0ae598e95dc970b74767f19372d61af8
Author: Magnus Chacon <mchacon@gee-mail.com>
Date:   Sun Apr 27 20:43:35 2008 -0700

    Update rakefile
...
```

## Sharing Tags

By default, the `git push` command doesn’t transfer tags to remote servers. You will have to explicitly push tags to a shared server after you have created them. This process is just like sharing remote branches — you can run `git push origin <tagname>`.

```
$ git push origin v1.5
Counting objects: 14, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (12/12), done.
Writing objects: 100% (14/14), 2.05 KiB | 0 bytes/s, done.
Total 14 (delta 3), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new tag]         v1.5 -> v1.5
```

If you have a lot of tags that you want to push up at once, you can also use the `--tags` option to the `git push`command. This will transfer all of your tags to the remote server that are not already there.

```
$ git push origin --tags
Counting objects: 1, done.
Writing objects: 100% (1/1), 160 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new tag]         v1.4 -> v1.4
 * [new tag]         v1.4-lw -> v1.4-lw
```

Now, when someone else clones or pulls from your repository, they will get all your tags as well.

Note `git push` pushes both types of tags

> `git push <remote> --tags` will push both lightweight and annotated tags. There is currently no option to push only lightweight tags, but if you use `git push <remote> --follow-tags` only annotated tags will be pushed to the remote.

## Deleting Tags

To delete a tag on your local repository, you can use `git tag -d <tagname>`. For example, we could remove our lightweight tag above as follows:

```
$ git tag -d v1.4-lw
Deleted tag 'v1.4-lw' (was e7d5add)
```

Note that this does not remove the tag from any remote servers. There are two common variations for deleting a tag from a remote server.

The first variation is `git push <remote> :refs/tags/<tagname>`:

```
$ git push origin :refs/tags/v1.4-lw
To /git@github.com:schacon/simplegit.git
 - [deleted]         v1.4-lw
```

The way to interpret the above is to read it as the null value before the colon is being pushed to the remote tag name, effectively deleting it.

The second (and more intuitive) way to delete a remote tag is with:

```
$ git push origin --delete <tagname>
```

## Checking out Tags

If you want to view the versions of files a tag is pointing to, you can do a `git checkout` of that tag, although this puts your repository in “detached HEAD” state, which has some ill side effects:

```
$ git checkout v2.0.0
Note: switching to 'v2.0.0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 99ada87... Merge pull request #89 from schacon/appendix-final

$ git checkout v2.0-beta-0.1
Previous HEAD position was 99ada87... Merge pull request #89 from schacon/appendix-final
HEAD is now at df3f601... Add atlas.json and cover image
```

In “detached HEAD” state, if you make changes and then create a commit, the tag will stay the same, but your new commit won’t belong to any branch and will be unreachable, except by the exact commit hash. Thus, if you need to make changes — say you’re fixing a bug on an older version, for instance — you will generally want to create a branch:

```
$ git checkout -b version2 v2.0.0
Switched to a new branch 'version2'
```

If you do this and make a commit, your `version2` branch will be slightly different than your `v2.0.0` tag since it will move forward with your new changes, so do be careful.

prev | next