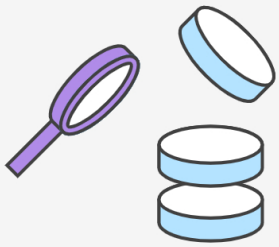


Git tag

git status / git tag / git blame

Tagging

This document will discuss the Git concept of tagging and the `git tag` command. Tags are ref's that point to specific points in Git history. Tagging is generally used to capture a point in history that is used for a marked version release (i.e. v1.0.1). A tag is like a branch that doesn't change. Unlike branches, tags, after being created, have no further history of commits. For more info on branches visit the `git branch` page. This document will cover the different kind of tags, how to create tags, listing all tags, deleting tags, sharing tags, and more.

Creating a tag

To create a new tag execute the following command:

```
git tag <tagname>
```

Replace `<tagname>` with a semantic identifier to the state of the repo at the time the tag is being created. A common pattern is to use version numbers like `git tag v1.4`. Git supports two different types of tags, annotated and lightweight tags. The previous example created a lightweight tag. Lightweight tags and Annotated tags differ in the amount of accompanying meta data they store. A best practice is to consider Annotated tags as public, and Lightweight tags as private. Annotated tags store extra meta data such as: the tagger name, email, and date. This is important data for a public release. Lightweight tags are essentially 'bookmarks' to a commit, they are just a name and a pointer to a commit, useful for creating quick links to relevant commits.

Annotated Tags

Annotated tags are stored as full objects in the Git database. To reiterate, They store extra meta data such as: the tagger name, email, and date. Similar to commits and commit messages Annotated tags have a tagging message. Additionally, for security, annotated tags can be signed and verified with GNU Privacy Guard (GPG). Suggested best practices for git tagging is to prefer annotated tags over lightweight so you can have all the associated meta-data.

Ready to learn Git? Try this interactive tutorial.

Get started now

```
git tag -a v1.4
```

Executing this command will create a new annotated tag identified with `v1.4`. The command will then open up the configured default text editor to prompt for further meta data input.

```
git tag -a v1.4 -m "my version 1.4"
```

Executing this command is similar to the previous invocation, however, this version of the command is passed the `-m` option and a message. This is a convenience method similar to `git commit -m` that will immediately create a new tag and forgo opening the local text editor in favor of saving the message

passed in with the `-m` option.

Lightweight Tags

```
git tag v1.4-lw
```

Executing this command creates a lightweight tag identified as `v1.4-lw`. Lightweight tags are created with the absence of the `-a`, `-s`, or `-m` options. Lightweight tags create a new tag checksum and store it in the `.git/` directory of the project's repo.

Listing Tags

To list stored tags in a repo execute the following:

```
git tag
```

This will output a list of tags:

```
v0.10.0
v0.10.0-rc1
v0.11.0
v0.11.0-rc1
v0.11.1
v0.11.2
v0.12.0
v0.12.0-rc1
v0.12.1
v0.12.2
v0.13.0
v0.13.0-rc1
v0.13.0-rc2
```


To refine the list of tags the `-l` option can be passed with a wild card expression:

```
$ git tag -l *-rc*
v0.10.0-rc1
v0.11.0-rc1
v0.12.0-rc1
v0.13.0-rc1
v0.13.0-rc2
v0.14.0-rc1
v0.9.0-rc1
v15.0.0-rc.1
v15.0.0-rc.2
v15.4.0-rc.3
```

This previous example uses the `-l` option and a wildcard expression of `-rc` which returns a list of all tags marked with a `-rc` prefix, traditionally used to identify *release candidates*.

Tagging Old Commits

The previous tagging examples have demonstrated operations on implicit commits. By default, `git tag` will create a tag on the commit that `HEAD` is referencing. Alternatively `git tag` can be passed as a ref to a specific commit. This will tag the passed commit instead of defaulting to `HEAD`. To gather a list of older commits execute the `git log` command.


```
$ git log --pretty=oneline
15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch
a6b4c97498bd301d84096da251c98a07c7723e65 add update m
0d52aaab4479697da7686c15f77a3d64d9165190 one more thi
6d52a271eda8725415634dd79daabbc4d9b6008e Merge branch
```

Executing `git log` will output a list of commits. In this example we will pick the top most commit `Merge branch 'feature'` for the new tag. We will need to reference to the commit SHA hash to pass to Git:

```
git tag -a v1.2 15027957951b64cf874c3557a0f3547bd83b3f
```

Executing the above `git tag` invocation will create a new annotated commit identified as `v1.2` for the commit we selected in the previous `git log` example.

ReTagging/Replacing Old Tags

If you try to create a tag with the same identifier as an existing tag, Git will throw an error like:

```
fatal: tag 'v0.4' already exists
```

Additionally if you try to tag an older commit with an existing tag identifier Git will throw the same error.

In the event that you must update an existing tag, the `-f FORCE` option must be used.

```
git tag -a -f v1.4 15027957951b64cf874c3557a0f3547bd83
```

Executing the above command will map the `15027957951b64cf874c3557a0f3547bd83b3ff6` commit to the `v1.4` tag identifier. It will override any existing content for the `v1.4` tag.

Sharing: Pushing Tags to Remote

Sharing tags is similar to pushing branches. By default, `git push` will not push tags. Tags have to be explicitly passed to `git push`.

```
$ git push origin v1.4
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, d
Total 14 (delta 3), reused 0 (delta 0)
To git@bitbucket.com:atlasbro/gittagdocs.git
* [new tag] v1.4 -> v1.4
```

To push multiple tags simultaneously pass the `--tags` option to `git push` command. When another user clones or pulls a repo they will receive the new tags.

Checking Out Tags

You can view the state of a repo at a tag by using the `git checkout` command.

```
git checkout v1.4
```

The above command will checkout the `v1.4` tag. This puts the repo in a detached `HEAD` state. This means any changes made will not update the tag. They will create a new detached commit. This new detached commit will not be part of any branch and will only be reachable directly by the commits SHA hash. Therefore it is a best practice to create a new branch anytime you're making changes in a detached `HEAD` state.

Deleting Tags

Deleting tags is a straightforward operation. Passing the `-d` option and a tag identifier to `git tag` will delete the identified tag.

```
$ git tag
v1
v2
v3
$ git tag -d v1
$ git tag
v2
v3
```

In this example `git tag` is executed to display a list of tags showing v1, v2, v3, Then `git tag -d v1` is executed which deletes the v1 tag.

Summary

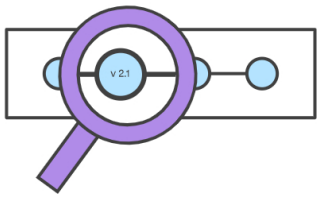
To recap, Tagging is an additional mechanism used to create a snap shot of a Git repo. Tagging is traditionally used to create semantic version number identifier tags that correspond to software release cycles. The `git tag` command is the primary driver of tag: creation, modification and deletion. There are two types of tags; annotated and lightweight. Annotated tags are generally the better practices as they store additional valuable meta data about the tag. Additional Git commands covered in this

document were `git push`, and `git checkout`. Visit their corresponding pages for discussion on their extended use.

Ready to learn Git?

Try this interactive tutorial.

Get started now



Next up:

Git blame

START NEXT TUTORIAL

Powered By

Bitbucket

Recommend

Want future articles?

Enter Your Email For Git News

Site hosted by

ATLASSIAN