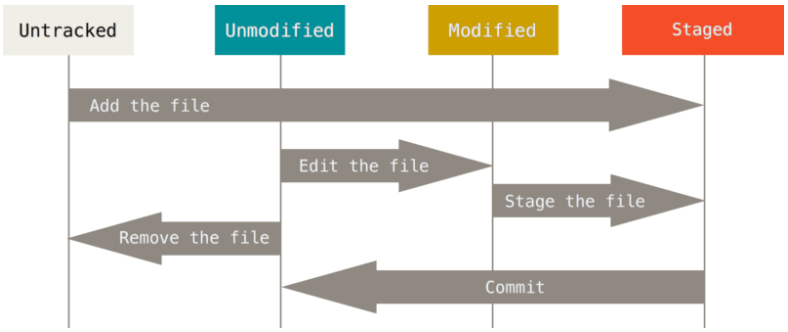
Git basics!

Table of contents

1. Edit

Lifecycle



As you **edit** files, Git sees them as **modified**, because you've changed them since your last commit. As you work, you selectively **stage** these modified files and then **commit** all those staged changes, and the **cycle repeats**.

tracked vs untracked

Each file in working directory can be in **one of two states**:

- tracked
- untracked.

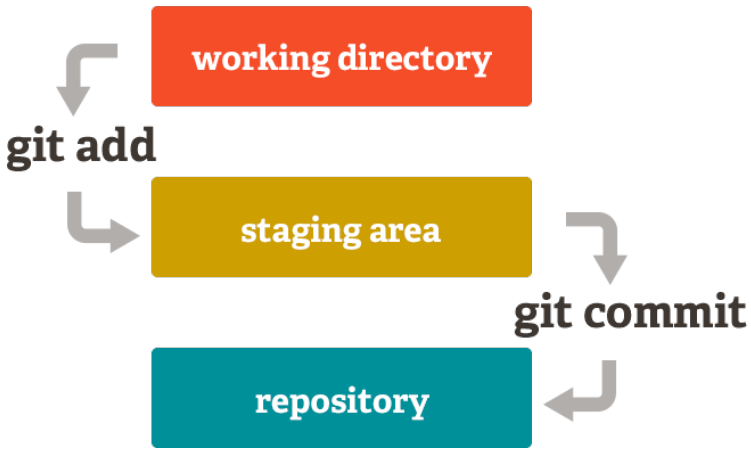
Tracked files are files that were in the last snapshot; they can be:

- unmodified
- modified
- staged

tracked files are files that Git knows about.

2. Staging

Staging Area



stage some of your files and commit them without committing all of the other modified files in working directory.

This allows you to stage only portions of a modified file.

Git has something called the **"staging area"** or **"index"**.

This is an **intermediate area** where commits can be formatted and reviewed before completing the commit.

3. Tagging

Staging Area

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)

3.1. Listing Your Tags

list in alphabetical order

```
git tag
```

Listing tag wildcards requires -l or --list option

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.

3.2. Creating Tags

Git supports two types of tags:

- lightweight
- annotated

annotated

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

```
The -m specifies a tagging message, which is stored with the tag
```

lightweight

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

```
To create a lightweight tag, don't supply any of the -a, -s, or -m options, just
provide a tag name
```

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;

3.3. 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

sharing tags

```
git push origin v1
```

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**

git push <remote> --follow-tags will **only annotated tags will be pushed** to the remote.

3.4. Deleting Tags

To delete a tag on your local repository, you can use git tag **-d <tagname>**

Remove our lightweight tag above as follows:

remove tag

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

Note that this does not remove the tag from any remote servers.

3.4.1. remove from server

remove the tag from any remote servers

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

3.5. 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
```

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
```

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**.

```
One of the more helpful options is -p or --patch, which shows the difference (the
patch output) introduced in each commit.
```