LING83800: Git and GitHub

Kyle Gorman and Michael Mandel

1 Why Git

Git is a *distributed version control system* (DVCS). Version control systems track changes to sets of data such as code, documentation, audio, graphics, and the like. They automate and systematicize the practice of manually archiving "snapshots" of important work in case it is needed in the future. Git allows us not only to make snapshots, but also to link them to metadata, such as time and purpose of the changes made, and to automatically switch between earlier and newer versions of files. Git is also *distributed* in the sense that multiple machines—servers or personal computers—can mirror the same repository, so that you are protected from issues like hardware failure, but can also work offline (at least temporarily). Git has several advantages over other DVCSes:

- It is usually faster, and more widely available.
- It has superior support for *branching*, a technique that makes it easier for multiple people to collaborate on multiple "features" or "issues" simultaneously.
- It is supported by GitHub, a popular Git hosting service that adds many additional services.

2 Preliminaries

In what follows I assume you already have Git installed; if not, instructions are given in the Chacon & Straub reading. Below, a `$` in typewriter text indicates a command entered at the command-line and `#` indicates a comment to the reader. Note that all Git commands are of the form `git VERB [ARGS]` where `VERB` is some type of action and `[ARGS]` are the arguments.

3 Creating a local repository

There are two ways to create a "local" repository, that is, one housed on your personal computer: either `init` (initialize) the current directory as an empty repository in the current directory, or clone `clone` an existing repository on another machine.

Initializing an empty repository To create an empty repository, simply create a directory, navigate to it, and then issue `git init`:

```
$ mkdir NewRepo
$ cd NewRepo
$ git init
Initialized empty Git repository in /home/mim/NewRepo/.git/
# Now the NewRepo directory and its subdirectories are part of a repository.
```

Cloning an existing repository To clone an existing repository, you will need its URL. If the repository you are cloning is on GitHub, for instance, navigate to the repository and click the green "Clone or download" button on the right side of the screen, then copy the URL. Then pass this as an argument to `git clone`.

```
# URL obtained from:
# https://github.com/mim/ling83800demo
$ git clone \
      https://github.com/mim/ling83800demo.git
Cloning into 'CloneMe'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
# This creates a `CloneMe` directory; now enter it.
$ cd CloneMe
```

4 Git basics

The normal workflow in Git consists of creating or editing files, *staging* the changes made, and then *committing* those changes to the database, creating a permanent snapshot of your repository.

Adding files or changes When you create or move a file into a Git repository, it is initially in an *untracked* state. To add a file to the repository, use `add`. Files that that have been added are said to be *staged*.

```
# Creating a new file.
$ vi README.md
# This command gives the current status of the repository.
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

README.md
```

```
nothing added to commit but untracked files present (use "git add" to track)
$ git add README.md
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

new file:   README.md
```

The add verb is also used to stage modifications to files that have already been added to the repository.

```
# Editing an existing file.
$ vi README.md
$ git add README.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

modified:   README.md
```

Committing changes Now we wish to take a snapshot of our work, including all the staged changes, and write it to the repository database. To do this, use `commit`. Each commit is associated with a *commit message*, text describing the changes made. The command `git commit` by itself drops the user into a text editor and asked to write a commit message there. The command `git commit -m "My commit message here"` (as below) allows the user to avoid this step, instead passing a short commit message from the command line.

```
$ git commit -m "First commit"
[master 2cccbb0] First commit
 1 file changed, 1 insertion(+)
  create mode 100644 README.md
$ git status
On branch master
nothing to commit, working tree clean
```

5 Life cycle of a file

Let's be a bit more systematic. There are four states a file can be in, illustrated in figure 1:

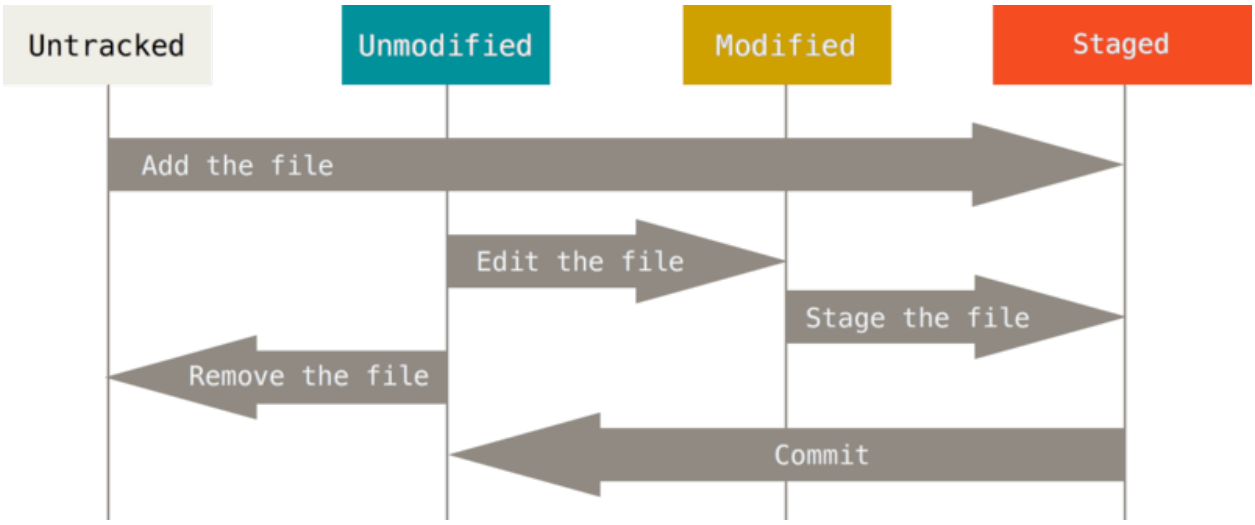

Figure 1: The life cycle of a file; from Chacon & Straub.

(a) Untracked: A file (inside of a repository directory) that has not yet been added

(b) Unmodified: A file which has not been changed since the last commit

(c) Modified: A file which has been changed since the last commit

(d) Staged: A modified file whose changes are staged for commit

Thus, `add` converts untracked (a) or modified (b) files to staged (d) files, and `commit` converts staged (d) files to unmodified files (b), writing the changes to the database. And of course, editing converts unmodified (b) files to modified (c) files. There are a few other operations that can be described using these four stages.

Reverting a modified file To revert (i.e., undo) modified (b) files to their unmodified (committed) form, use `checkout`:

```
# Editing a file.
$ vi README.md
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
# Now reverting the change.
$ git checkout -- README.md
```

```
$ git status
On branch master
nothing to commit, working tree clean
```

Unstaging a modified file To revert staged files (d) to their unmodified format (b), use `reset`:

```
# Editing a file.
$ vi README.md
# Staging it.
$ git add README.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

modified:   README.md
# Now unstaging the change.
$ git reset HEAD README.md
Unstaged changes after reset:
M README.md
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

Removing files To remove unmodified (b) or modified (c) files from a repository, use `rm`:

```
$ git rm README.md
rm 'README.md'
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

deleted:    README.md
```

NB: Git will also delete your copy of the file!

6 Working with a remote GitHub repositories

Cloning a GitHub repository, like above, creates a local copy but also stores the location of the so-called *remote* copy, which it calls `origin`. After creating an empty repository on GitHub, copy-and-pasteable instructions are shown for initializing the repository and the remote path.

A change made to either the local or the remote copy will cause the two repositories to go out of sync. The following commands synchronize the two repositories.

Changes from the remote To copy changes from `origin` to your local copy, use `pull`:

```
# We are one commit behind the remote.
$ git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/mim/ling83800demo
   c7c2147..084ce13  master     -> origin/master
Updating c7c2147..084ce13
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Changes to the remote To send local commits to the remote repository, use `push`:

```
# We are one commit ahead of the remote.
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
$ git push origin master
    Counting objects: 3, done.
Writing objects: 100% (3/3), 284 bytes | 284.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/mim/ling83800demo
   084ce13..e6a571e  master -> master
```

NB: one does not need to push after every commit, but the remote won't reflect your commits until they are pushed.

Figure 2: XKCD #1597

For later reading...

The following are a few of the advanced topics covered in the Chacon & Straub reading:

- Tagging commits with `tag`
- Managing separate *branches*
- Viewing the commit history with `log`
- Making *pull requests* to others' projects