A Brief Git Primer for CS 350

~~Tyler Szepesi~~
(shamelessly stolen by Ben Cassell)

University of Waterloo
becassel@uwaterloo.ca

September 8, 2017

Overview

Introduction

Welcome to the CS 350 Git Primer tutorial!

What is version control?

- Tracks the history of files.
- Can safely commit new versions and restore old versions.

Why use version control?

- Easy file and change tracking.
- Peace of mind when disaster inevitably strikes.
- Helps with replication and off-site backup.
- Facilitates collaboration (not needed for CS 350).

Why version control?

(a) Using version control.

(b) Not using version control.

A scientific and unbiased comparison.

Git

What is Git?

- A specific version control system.
- (As opposed to Subversion, Mercurial, etc...)

Why should I use Git and not some other version control system?

- Not necessarily better or worse, but different.
- It is one of the most commonly used version control systems.
- CS 350 assumes you will use Git.

Pfft, **MY** favourite version control is far superior to Git.

- Cool story, bro! You should use that one!
- However, if something goes wrong, you'll need to fix it.

One-Time Setup

Let's go over the basics (this will help with Assignment 0):

- The purpose of Assignment 0 is to get your environment set up and ensure that you can edit, compile and run your kernel.
- Even if you plan on working outside of the student environment, you should start by setting there, because:
  - We test your code on the student environment.
  - You can only submit from the student environment.
  - Git is already set up on the student environment.

Getting Started

Instructions for OS/161 on the student environment (`https://www.student.cs.uwaterloo.ca/~cs350/common/Install161.html`):

- Step 0: SSH to the student environment.
- Step 1: Edit your `PATH` environment variable.
- Step 2: Get the OS/161 code.

Add a .gitignore

Many code bases, including OS/161, generate a large number of files during compilation. We do not want to track these.

A file named `.gitignore` lets us tell Git which files to omit.

Create the file `$HOME/cs350-os161/os161-1.99/.gitignore`:

```
*/*/*/.depend
*/*/*/build
build
cscope.out
tags
kern/compile/ASST*
# Automatically generated
user/testbin/randcall/calls.c
# Generated by configure
defs.mk
```

Commit the Initial Version

Now we are ready to set up the Git repository. First, we initialize a new repository:

```
$ cd $HOME/cs350-os161/os161-1.99
$ git init
```

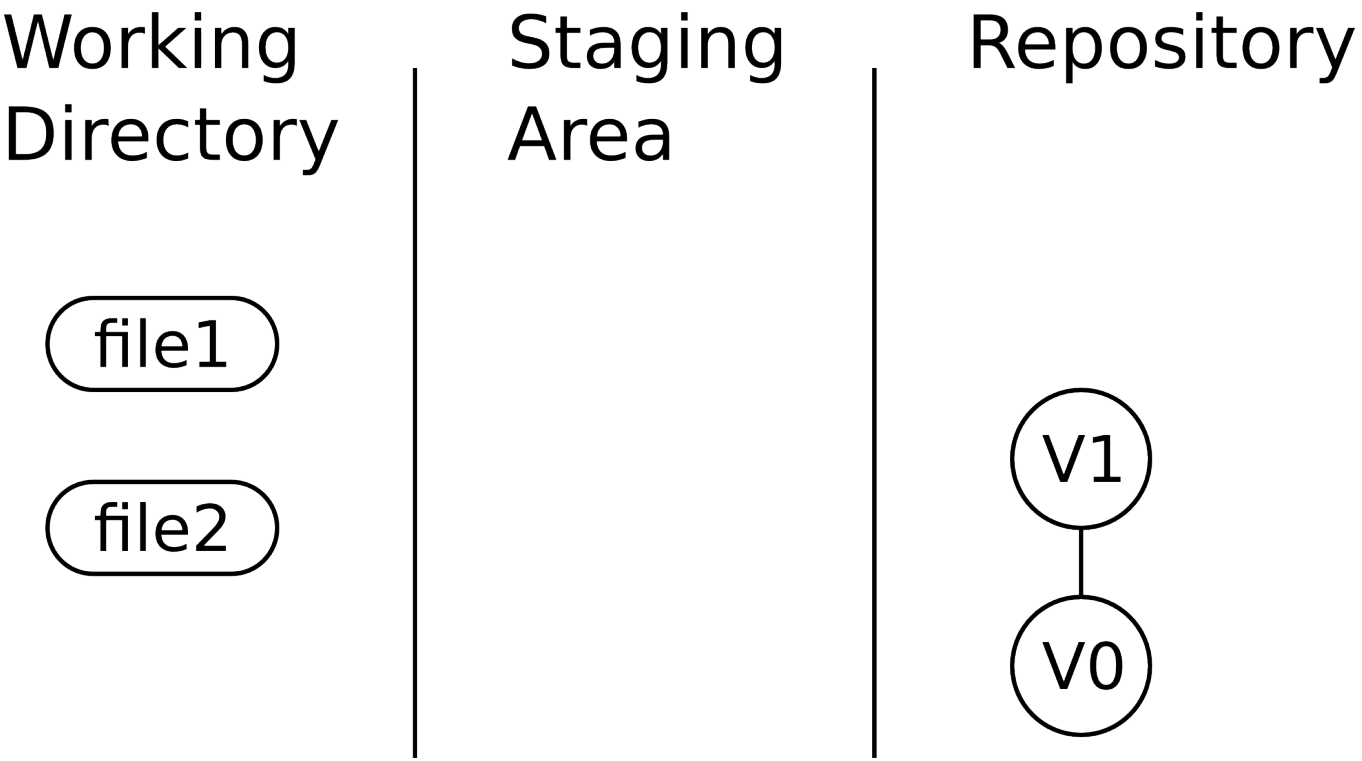
Now that we have a repository, we can add the initial version of the files:

```
$ git add .
$ git commit -m "Initial Commit"
```

And that's it! Go back to the OS/161 set up instructions to learn how to compile and run your kernel.

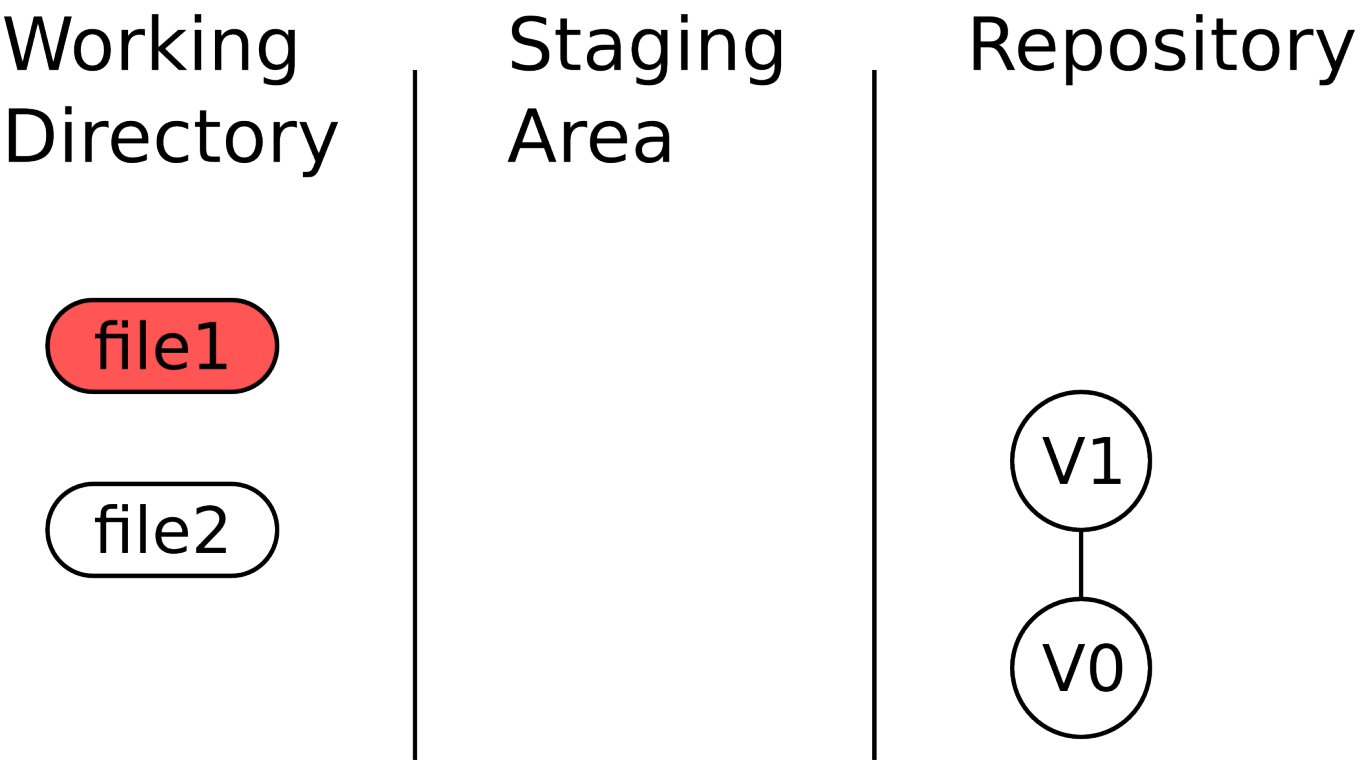
Overview of Git

Git has three main "areas" to know about:



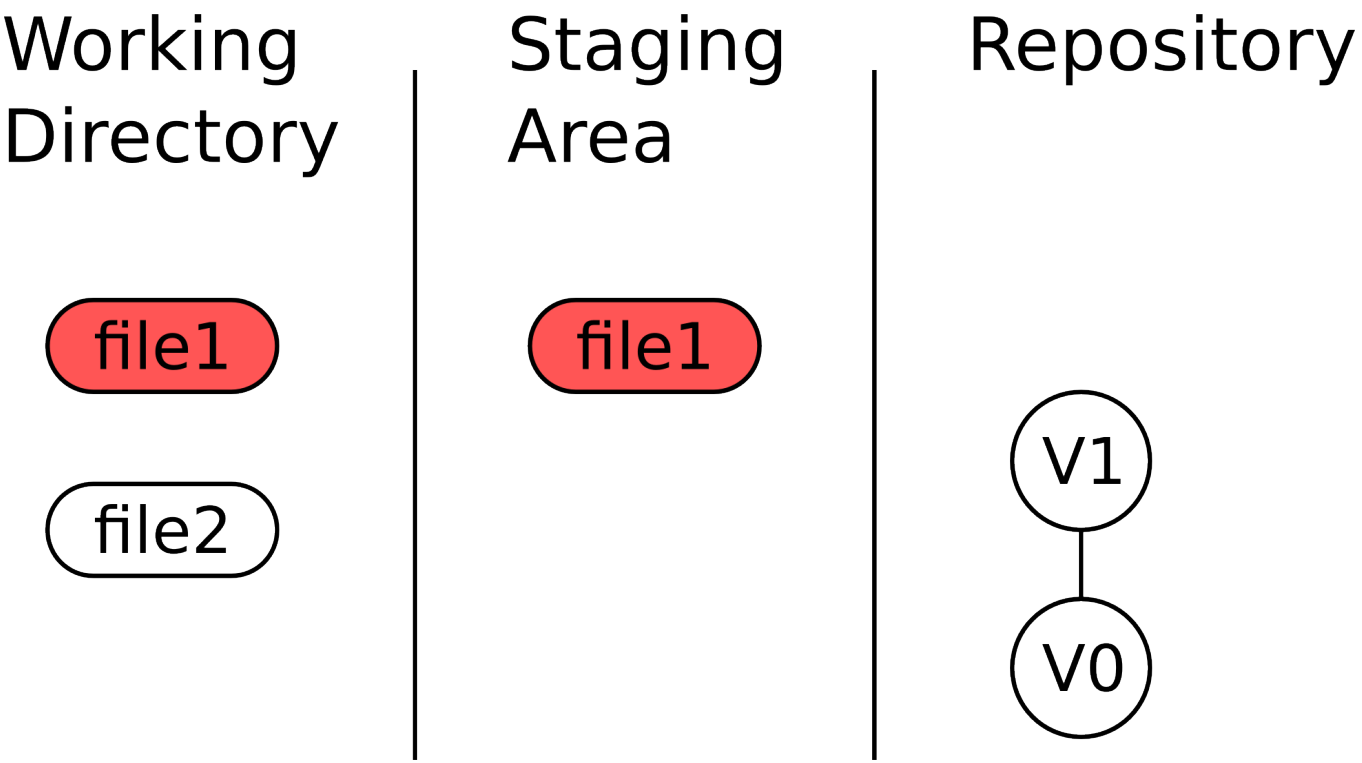
Overview of Git

Any changes you make to files while coding are being made in the working directory.



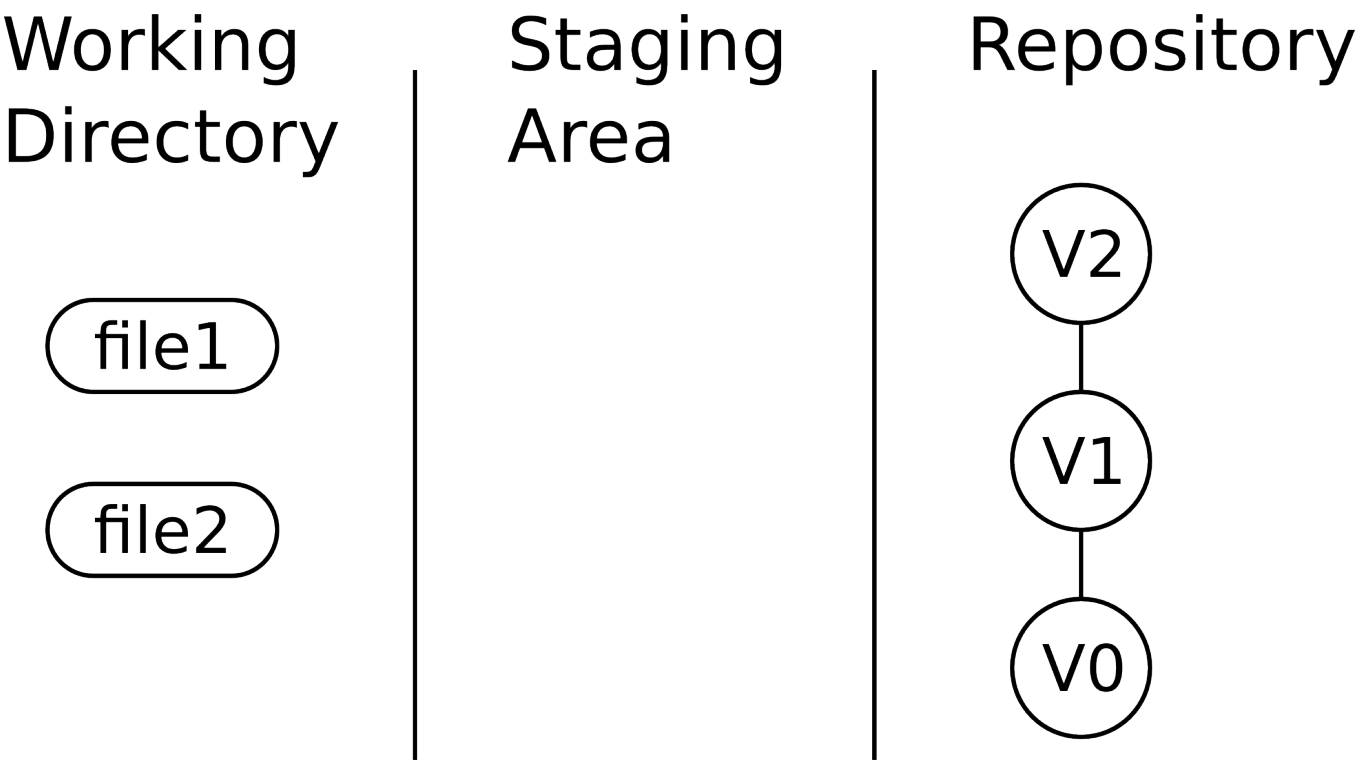
Overview of Git

Once you are satisfied with your changes, you add them to the staging area.



Overview of Git

Finally, you commit any changes in the staging area to the repository.



Staging

Staging changes to be committed is done with the add command. You can stage all of the changes in your working tree with the following command:

```
$ git add .
```

Or you can stage individual files to be committed:

```
$ git add file1 file2
```

Committing

Once you are happy with the changes you have staged, they can be committed to Git using the `commit` command:

```
$ git commit
```

This will open up your default text editor and have you write a log message. Alternatively, you can use the `-m` option and specify the log message directly:

```
$ git commit -m "some log message"
```

Summary

Your everyday usage of Git will be almost exclusively the following:

```
$ git add .
$ git commit -m "some log message"
```

It is so common that there is a built-in shorthand:

```
$ git commit -a -m "some log message"
```

When Mistakes Happen

It is not a matter of "if" you make a mistake...

It is not even a matter of "when" you make a mistake...

It is a matter of when you **realize** you made a mistake. In other words, how you fix a mistake depends on when you catch it.

Clean up the Working Directory

Let's pretend you have made a change you want to get rid of, by restoring a file to its last committed version.

With Git, this requires a single command:

```
$ git checkout file1
```

After running this command, non-committed changes to `file1` are removed.

This allows you to experiment with new code, without the risk of ruining your current, functional copy.

Unstaging Files

A less common error occurs when you stage a file to be committed, and then realize you didn't want to do so.

Unstaging the file is another one-liner:

```
$ git reset file1
```

Warning!

There are options to this particular command that make irreversible changes to your repository. I strongly encourage you to not use those options.

Reverting Commits

The worst mistakes are the ones you don't catch until a few days or weeks later. By this time, you have already committed the changes to the repository.

With Git, the `revert` command allows you to undo one or more commits. The simplest version of this is to undo the latest commit:

```
$ git revert HEAD
```

The `revert` command is considered safe because it does not delete the old version from your repository's history, but rather creates a new version that does not contain the specified commit's changes.

Reverting Multiple Commits

Sometimes, the mistake goes back further than your last commit.

To revert multiple commits, you can specify a range:

```
$ git revert HEAD~n..HEAD
```

`n` in the above example is an integer and specifies the `n`'th last commit. For example, the following will revert the last two commits:

```
$ git revert HEAD~2..HEAD
```

Working from Home

If you plan to work on a different environment, you can use Git to keep your code changes coordinated across the different computers.

The University of Waterloo provides a service for hosting Git repositories: `https://git.uwaterloo.ca`

You can sign in using your WatIAM credentials.

Warning!

Do **NOT** use GitHub, Bitbucket, or any other external Git service.

Create a Project

To create a new project on GitLab, click the plus symbol at the top right corner of the page.

For this course, name your project os161-1.99.

Click “Create project”.

Push Your Existing Repository

On the student environment:

```
$ cd $HOME/cs350-os161/os161-1.99
$ git remote add origin https://git.uwaterloo.ca/becassel/os161
    -1.99.git
$ git push -u origin master
```

This will initialize the newly created repository on GitLab with the contents of the repository you created on the student environment. Remember to replace "becassel" with your user name.

Cloning from GitLab

To get a copy of your repository on a different machine:

```
$ git clone https://git.uwaterloo.ca/becassel/os161-1.99.git
```

Now you can work on the code at home, the exact same way you would on the student environment. Again, remember to replace "becassel" with your user name.

Keeping Repositories Synchronized

After committing changes to a local repository, you will want to push those changes to the remote repository hosted on GitLab:

```
$ git push
```

When you start working on a different environment, you acquire the latest changes by pulling any updates from the GitLab repository:

```
$ git pull
```

Summary

Your everyday usage of Git, when you use multiple environments:

When you first sit down to start working:

```
$ git pull
```

When you commit a change:

```
$ git commit -a -m "some log message"
$ git push
```

General Tips

Configuring Git:

```
$ git config --global core.editor vim
$ git config --global user.name "Ben"
$ git config --global user.email "becassel@uwaterloo.ca"
```

Useful Resources

- www.atlassian.com/git/tutorials
- www-cs-students.stanford.edu/~blynn/gitmagic
- gitref.org
- www.kernel.org/pub/software/scm/git/docs/user-manual.html
- www.vogella.com/tutorials/Git/article.html
- git-scm.com/documentation

The End

Congratulations! You're on your way to becoming an expert!

Questions?