A Brief Git Primer for CS 350

Tyler Szepesi

University of Waterloo
stszepes@uwaterloo.ca

September 17, 2015

Overview

1. Introduction
2. One Time Setup
3. Using Git
   - Git on a Good Day
   - Git on a Bad Day
4. Working from Home
5. Conclusion

Introduction

Welcome to the CS 350 Git Primer tutorial!

What is version control?

- A way to track the history of a file.
- Allows you to create new versions and restore old versions.

Why use version control?

- Keep track of your files the right way.
- Allows you to restore old versions when you make the inevitable mistake.
- Added Bonus 1: keeping backups is easy.
- Added Bonus 2: collaboration is easy (BUT NOT DONE IN THIS COURSE!).

Why version control?

(a) Using version control.

(b) Not using version control.

A scientific and unbiased comparison.

Git

What is this Git you speak of?

- A specific implementation of a version control system.

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

- This tutorial is specifically for CS 350 and Git.
- It is one of the most commonly used version control systems.
- Learn it now, use it for everything!

But I really like **X** for version control!

- Cool, use that ...
- But we make no promises about our ability to support **X**.

One Time Setup

This is the part where I do 80% of 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 up on the student environment.

- We test your code on the student environment.
- You can only submit from the student environment.
- I am going to show you how to use Git to make your life easy.

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.
- Step 2: Get the OS/161 code.

Add a .gitignore

Generally, it is best to avoid version control on generated files.

A file named .gitignore lists all files that Git should ignore.

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
```

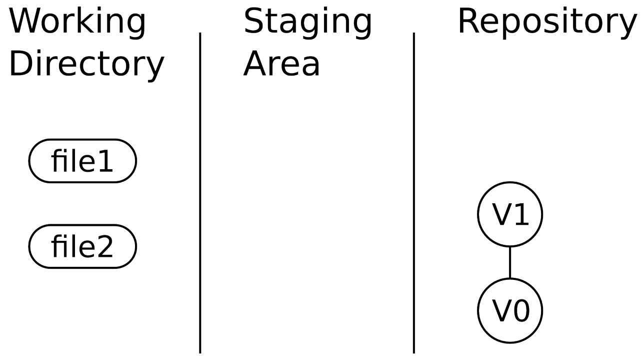
Now that we have a repository, we can add the initial version of the files:

```
$ 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.

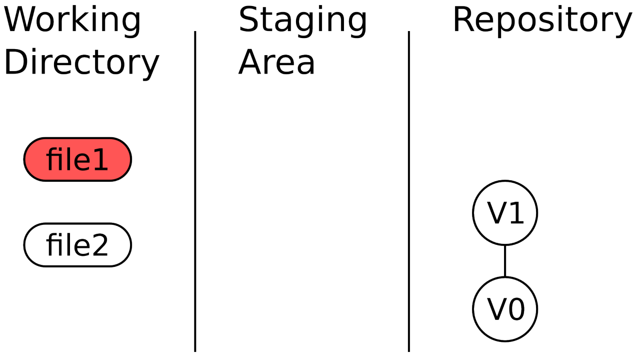
Overview of Git

When working with Git there are three areas you need to be aware of:



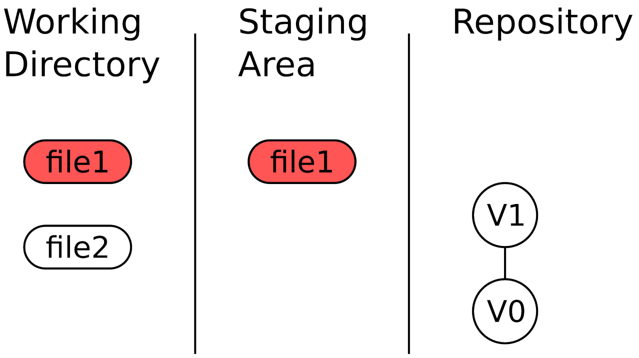
Overview of Git

As you are coding away, you are making changes to the files in the working directory.



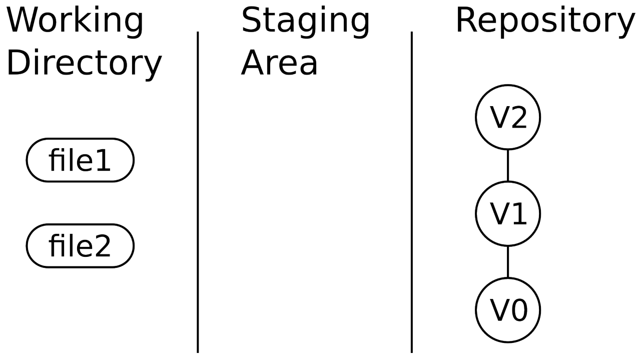
Overview of Git

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



Overview of Git

Finally, you commit the staging area to the repository.



Staging

Staging changes to be commited 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 commited:

```
$ git add file1 file2
```

Commiting

Once you are happy with the changes you have staged, they can be commited 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 ...”

How you deal with fixing a mistake depends on when you catch it.

Clean up the Working Directory

Sometimes you are programming away in the working directory, and realize that your “great” idea was not so great.

Now you want to go back to the latest working version of your code.

With Git, this is as simple as a single command:

```
$ git checkout file1
```

Now you are back to the version of `file1` that was last commited.

Notice the peace of mind this gives you! You can just try things without worrying about ruining hours of hard work.

Unstaging Files

This is less common, but let's say that you stage a file to be commited and then realize that you don't actually want to commit that file.

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 commited 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` here 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`

Login with your WatIAM credentials.

Warning!

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

Create a Project

Create a new project on GitLab by clicking the plus symbol in the top right corner.

Name it 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/stszepes/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.

Cloning from GitLab

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

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

Now you can work on the code at home, the exact same way you would on the student environment.

Keeping Repositories Synchronized

After commiting 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 "Tyler"
$ git config --global user.email "stszepesi@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?