# An Overview of Using `Git`

Westminster College Computer Science Department

June 30, 2018

## Overview

The Git revision control system allows one to maintain different versions of files, create separate branches for experimentation, and restore to previous versions. It is an excellent tool for managing software projects, both individually as well as collaborating with a group. It can be combined with GitHub which provides a remote repository where several users can collaborate on a shared code base and synchronize and coordinate their activities.

This document provides an overview of a few simple git commands and concepts; online documentation provides more thorough coverage of this rich toolset.

## How to Create a Repository

Make a directory where your repository will be stored.

```
mkdir git-repo
```

and now `cd` to that directory.

Next, initialize the `git` repository:

```
git init
```

To determine the status of your repository, enter:

```
git status
```

Now, you must populate the repository with some files. Create the following 3 files in the `git-repo` directory:

```
file1.txt

file2.txt

file3.txt
```

## How to Commit to the Repository

`git` distinguishes between **staged** and **unstaged** files. Only staged files can be committed to the repository. To stage a file, you must add it to the repository:

```
git add file1.txt file2.txt file3.txt
```

This could have been simplified with the command

```
git add -A
```

which adds all files to the staging.

Next, you must commit the contents of the staging area to the repository:

```
git commit -m "Initial Commit"
```

The string following the `-m` option allows you to enter a string message describing the changes you are committing. In this instance, since it is only the initial commit, we will provide the appropriate message.

We may not want all files sent to the staging area and committed to the repository. In this instance, we can use a `.gitignore` file which specifies which files we **do not** want committed to the repository.

The following command can be used to determine the status of the repository since the most recent commit:

```
git status
```

## Branching and Merging

A git repository has a single **master** branch. From this master branch, you may create separate branches to try different features. The basic approach is that your master branch contains your working code, any separate **developer** branches contain features you are experimenting with. When code you are experimenting with has passed tests, you can then integrate – or **merge** – the development branch.

To create a branch named `dev` , enter the command

```
git branch dev
```

which creates the branch named `dev` . Next you must switch to that using using the `checkout` command:

```
git checkout dev
```

This switches your branch to the `dev` branch. Now, any work you perform in the `dev` branch is independent from the master branch. The `checkout` command allows you to switch between any branches you wish.

You can also merge your `dev` branch with the master by entering

```
git checkout master
git merge dev
```

This first switches you back to the master branch, and then merges any changes in the `dev` branch with the master branch.

It is possible the master and `dev` branches have inconsistent changes, and if git is unable to merge the two branches, the files containing conflicts will have messages that must be resolved. Once the conflicts have been addressed, the changes must be committed.

## Restoring Previous Commits

Each `git` commit associates an MD5 checksum with the transaction. This checksum allows you to revert to previous instances from the repository.

The following command outputs the log history of all commits, including the checksum associated with each commit:

```
git log
```

If you wish to restore the state to the most recent previous commit (let's assume you had made several changes to files and wish to revert to the previous commit), enter:

```
git reset —-hard [commit]
```

This restores the branch to the specified commit. For example, if the checksum for a commit you wish to revert to is `e9e4794b1218c312d44e7b3b16367ec510ae150a`, you would enter

If you wish to restore to earlier commits, you would enter

```
git reset —-hard e9e4794b1218c312d44e7b3b16367ec510ae150a
```

The `reset` command can also be used by specifying which version of `HEAD` to reset to. The value of `HEAD` is the most recent commit, `HEAD^` the second-most, etc. The following examples illustrate:

`git reset —-hard HEAD^` // one commit back

`git reset —-hard HEAD~3` // two commits back

and so forth.

**Working with a Remote Repository**

A `git` repository can be deployed as a remote repository. The steps outlined in the section describe using github as a remote repository. This requires having a github account and the repository has been constructed. Follow the instructions for creating a repository on github. This repository will initially be empty. You will then push (i.e. move) a local repository to github.

Let's assume the name of that repository on github is `git-repo` and is associated with the user `charliebrown` . The URL for this github repository appears as

```
https://github.com/charliebrown/git-repo.git
```

Initially, you will create a local repository and commit changes to it outlined in the steps for creating a repository and committing to it. Once you have committed changes to your repository, you can then push the repository to github according to the following steps:

```
git remote add origin https://github.com/charliebrown/git-repo.git
```

This specifies the URL name of the repository as the remote origin.

Now, you must **push** your local repo to github:

```
git push origin master
```

This sends your commits in your local repository to github.

Another collaborator can then **clone** the repository by entering

```
git clone https://github.com/charliebrown/git-repo.git
```

If one makes changes to the contents of their local repository, they must add the changed file to the staging area, followed by committing the change. Changes are then pushed to the remote repository.

Once someone has cloned a remote repository, they can then get the latest changes by **pulling** from the repository:

```
git pull origin master
```

After cloning a repository, a good practice is to always first pull from that remote repository before making changes. This ensures you are always working with the latest updates from the remote repository.