An Introductory Tutorial to Git

Oliver Wilkins
wilkins@oxy.edu
Occidental College

1 Introduction

This tutorial will provide a basic introduction to git, with the intended audience being someone who is new to version control and wishes to use it for a long-term project. It assumes you will be using a command line interface on mac, with GitHub as the location for the remote repository. If you do not have a GitHub account, create one prior to following the tutorial.

Git is a version control system which helps developers keep their work organized by keeping track of what has been changed, when, and by whom. Learning how to use git is a worthwhile investment for a number of reasons. Firstly, it is used by nearly everyone who works in the computer science area, and is therefore important to know for the purpose of being able to find jobs and do them well. Beyond this, git is useful to know even for someone who does not intend to pursue work in the tech industry. Especially when working on large projects, it is important to be able to track changes and ensure your work does not get lost. Tracking changes is important because it allows you to go back to a previous version of your work if you accidentally break something, or to try out an experimental feature while leaving the primary version of your code unchanged.

Git works by storing files in four locations known as repositories, which can be though of as a storage location for a set of files. One is the remote repository (on GitHub), and three are local: the working directory, the staging area, and the local repository. The remote repository exists to store your work in a place while it will not be lost and can be easily shared with others. The three remote repositories exist as a place to make changes to your code and to organize them for the purpose of being sent to the remote repository. In this tutorial you will learn how to move files between these locations, as well as how to manage different versions of your work using stashing, branching, and merging. These terms will be defined as they are introduced.

2 Initialize local Repository

In this tutorial we will be initializing the repository locally using git init and connecting it to remote later, but it is also possible to initialize the repository on GitHub and use git clone to make a local repository.

In terminal, create a new directory for your project and navigate to it. Add a file file1.txt and write the letters a, b, and c, each on a separate line.

```
mkdir git-tutorial
cd git-tutorial
touch file1.txt
```

In the directory, initialize a local repository.

```
git init
```

The git-tutorial directory has now been transformed into a git repository. This means we can move its files to the staging area and local repository, before ultimately pushing them to remote.

2.1 First Commit

Let's move our text file to the staging area. Any file we want to put in our local repository must go through the staging area first.

```
git add file1.txt
```

Now that our text file is in the staging area we can add it to our local repository via a commit. Committing a file means that we want it to be included the next time we add files to our remote repository.

```
git commit
```

You will be prompted to add a comment to your commit. Write something about what you are changing, like "added file1, with a,b,c."

2.2 Connecting and pushing to Remote

Now that we have a local repository with a commit, it is time to connect to remote. This assigns a remote repository to our local one to which we can add our files. In GitHub, create an empty repository named git-tutorial.

Back in terminal, type the following command:

```
git remote add origin https://github.com/yourUsername/git-tutorial.git
```

This command adds the GitHub repository we just created as a remote associated with the local repository. 'origin' is a name we are assigning to the url so that we do not have to type it out again. We will now 'push' our local repository to remote. Pushing means to move what whatever files have been committed since our last push into our remote repository on GitHub.

```
git push –set-upstream origin main
```

In future pushes you will only need to type 'git push', the rest of the command is doing setup work that only needs to be done once.

Go back to GitHub and refresh the page. You should now see your file.txt in the repository.

2.3 Pulling from Remote

If you are working with other people, they may make changes to the remote repository which you want to see reflected in your local repository. To mimic this, we will add our own file in GitHub and 'pull' it to local. Pulling means to make your local repository match the remote one. In your GitHub repository, click on 'add file'. Name it new-file.txt and write whatever you want in it. Click 'commit changes'.

Back in terminal, pull from remote.

```
git pull
```

New-file.txt is now in your local repository.

2.4 Stashing

Stashing is a technique which sets aside something you are working on so that you can have a clean working directory but still come back later to the thing you were working on.

Create a file stash-file.txt and add it to the staging area.

```
touch stash-file.txt
git add stash-file.txt
```

Now let's stash this file away to come back to later.

```
git stash
```

If you run the ls command, you will see that stash-file.txt no longer shows up in the working directory. To see the stashes we have made, use the stash show command.

```
git stash show
```

You will see the stash we just made. To add the stashed file back to our working directory, use stash pop.

```
git stash pop
```

The stash-file.txt file is now back in our working directory (and staging area). Since we do not want this file to be committed, let's move it out of the staging area. We will use git status to track the contents of the staging area before and after to ensure it works.

```
git status
git reset
git status
```

The stash file is no longer in the staging area, but is still in the local directory.

2.5 Branching

Branching can be used to make changes in a way that does not impact the main version of your work. It creates a new working area which can later be added back into the main one. We will make a new branch called new-branch.

```
git branch new-branch
git branch
```

The git branch command (with no additional arguments) will show a list of all branches, and indicate which one you are currently on. It should show you as currently being on the main branch. Let's switch to the new one.

```
git switch new-branch
```

In the new branch, edit the file1.txt file by adding an e on the line below c. Add and commit the file, then switch back to the main branch.

```
git add file1.txt
git commit
git switch main
```

Back in the main branch, edit file1.txt again, this time adding a d on the line below c. Add and commit.

```
git add file1.txt
git commit
```

We now have two branches with two different versions of file1.txt.

2.6 Merging

Let's merge our new branch into our main branch so that the changes we have made can be consolidated into one place. Merging means to combine two different versions of code into one. Make sure you are still in the main branch.

```
git merge new-branch
```

This command should generate an error. This is because both branches made changes to the same file in a way that git does not know how to handle. To fix this, open file1.txt. You will see both d and e in the file, with messages indicating which branch each came from. Edit the file so the letters are in alphabetical order. Add and commit the edited file, then push to remote.

```
git add file1.txt
git commit
git push
```

The GitHub repository will now reflect the combined work of both branches.