Git - Command Line Guide

Anthony McGlone

May 17, 2022

Table of Contents

Chapter 1

Introduction

1.1 What is Git?

Git is a repository which is used to store files. Git is used by Software Engineers to store and version code for software releases. It's a powerful tool for collaborating on large projects.

Git versions the files within its repository, so every time a commit operation is made, the files in the repository are saved - a unique ID is also created and associated with the commit. This means that you can see the history of the changes in the repository. You can also delete the changes in the repository by going back to a specific commit.

In this guide you will see how the command line tool `git` is used to manage this repository.

1.2 Installing Git

1.2.1 MacOS

1. Open a terminal. Then install Homebrew (steps located: here).
2. Install Git by running `brew install git`
3. Open a terminal and run `git --version` (if Git is installed, the version number should be printed to console)

1.2.2 Windows

1. Install Git by downloading the Windows binary from git-scm (here)
2. Run the installer
3. Open a Git Bash terminal and run `git --version` (if Git is installed, the version number should be printed to console)

1.2.3 Linux (Ubuntu)

1. Install Git by opening a terminal and running `sudo apt-get install git`
2. Run `git --version` (if Git is installed, the version number should be printed to console)

1.2.4 Linux (Red Hat)

1. Install git by opening a terminal and running `sudo yum install git`
2. Run `git --version` (if Git is installed, the version number should be printed to console)

Chapter 2

Basic Git command line operations

2.1 Setting up a local repository

First create a folder called `store`. Then open a terminal and navigate into that folder. Run this command:

```
git init
```

This command creates the repository and also creates a `.git` subdirectory. This subdirectory stores information about commits, and the location of your remote repository. This remote repository is stored on a server (hosted on the internet or on a company's internal network). To share your commits with others, you have to push your local commits to the remote repository. We'll set up the remote repository later.

Let's add your username and email to your local repository now. This will be required so your commits can be associated with you. Start with the username. Run the following command (before doing so, add your full name between the quotes in the command below):

```
git config --global user.name "INSERT NAME HERE"
```

Now run the command to set up your email (again, insert your email address between the quotes):

```
git config --global user.email "INSERT EMAIL HERE"
```

Now run the command `git config --global user.name`. If everything is correct, you should see your username. Run `git config --global user.email` to see if your email prints out to console.

Run the following command to verify that your local repository was set up:

```
git status
```

You should see the following text in your terminal:

```
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

That's it! Your local repository is now successfully set up. The branch `master` is the main branch in your local repository. By default, it's the place where your files are stored. You can create other branches (basically copies of `master`) and work on edits there before saving them back into the `master` branch. For now, we'll just work with the `master` branch. We'll cover branching strategies and remote branches (stored in a remote repository) later.

2.2 Committing files into the `store` repository

First, create a `code.txt` file in the `store` folder. Then run `git status` in your terminal. You should see the following output:

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

        code.txt
```

The file is untracked, meaning Git doesn't see it yet. If you tried to do a commit operation now, Git wouldn't save the file or version it. Run the add operation below in your terminal to get Git to track the file.

```
git add code.txt
```

Now run `git status` again to see the tracking:

```
On branch master

No commits yet

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

        new file:  code.txt
```

Now commit these changes by running `git commit -m "This is my first commit!"` You can run `git log` after to see the commit ID. It should look something similar to this:

```
commit b973b70df3ffcfd8dbe0284c76e0d3bd7c30f3b6 (HEAD -> master)
Author:  anthonymcglone2022 <anthonymcglone2022@gmail.com>
Date:  Wed May 11 20:14:36 2022 +0100

    This is my first commit!
```

2.3 Creating a remote repository

GitHub is a website that offers free hosting of remote Git repositories. In order to create a repository there, there are a few steps that have to be completed.

- Signup to GitHub: here
- Generate an SSH key: Procedure here (select one of Mac, Linux, Windows)
- Adding that SSH key into GitHub: Procedure
- Create a repository (name it `store` and skip the README initilization step. Stop after Create repository step): Procedure

Let's push the commit we made to the remote repository. First, we will point our local repo to the remote repo. Take your GitHub user name and replace `username` in the following command - then run it.

```
git remote add origin git@github.com:username/store.git
```

To see if the repository was updated, run `git remote -v` in your terminal. Now push your code to the remote repository (i.e. your remote master branch). Run the following:

```
git push origin master
```

Open up your `store` repo in GitHub. If your command was successful then you should see the `code.txt` file inside it.

2.4 Understanding Branches

At this point, you have shared your work with others. Your local master branch is synched with the remote master branch. Your colleagues will clone your repository to their local machines using the `git clone` command. They will make changes, commit them to their local master branches and push them up to the repository that you just created. Git's strength is that it will track these changes. It also offers the tools to make sure that you don't overwrite anyone's work when you push up your own changes. We'll see how to do this now. Then we'll talk a little about branches.

Let's mimic a colleague by cloning your remote repository to a different folder on your machine. Make sure this folder is outside of your existing `store` directory. Create a folder called `colleaguesrepository` and then navigate inside that folder using the terminal. Then run the following command (replacing `username` with your GitHub user name):

```
git clone git@github.com:username/store.git
```

When the command finishes, you should find another copy of the `store` directory downloaded. Create a file called `colleaguesfile.txt` in this new `store` directory. Then

navigate inside the directory using the terminal. We will add, commit and push this file to our remote repo. Run the following:

```
- git add colleaguesfile.txt
- git commit -m "I am a new colleague committing some new code"
- git push origin master
```

Ok, so our colleague has committed their changes. We have our own work to finish, and we need to push it to remote. So let's do that now. Use your terminal to navigate back to your **initial** `store` repository. Create a file in there called `mysecondfile.txt`. Then run the add, commit, push sequence of commands:

```
- git add mysecondfile.txt
- git commit -m "I am pushing up my work now"
- git push origin master
```

Now your push should be rejected. You will see a message like:

```
To github.com:username/store.git
!  [rejected] master -> master (fetch first)
error:  failed to push some refs to 'git@github.com:username/store.git'
hint:  Updates were rejected because the remote contains work you dont
hint:  have locally.  This is usually caused by another repository pushing
hint:  to the same ref.  You may want to first integrate the remote changes
hint:  (e.g., 'git pull ...')  before pushing again.
hint:  See the 'Note about fast-forwards' in 'git push --help' for details.
```

This occurred because your remote repository has been updated with your colleague's work. This is how Git is useful. It will stop you from overwriting another person's work. Another mistake we made here was to commit our changes directly to local master. This is where branching strategy becomes important. You should always create a local branch from your local master. Then if your colleagues push to remote while you are working on your changes, you can pull their changes down to your local master. That way it always stays in synch with the remote master branch. When you're ready to push your own changes up, you can "merge" your changes into your updated local master and push it to remote. Let's do that now. Firstly let's undo the commit we just made. Run:

```
git reset --hard HEAD~1
```

If you run `git log` now, you will see that your most recent commit has been wiped from the Git version history. The file `mysecondfile.txt` is also gone. This is what a hard reset does. A soft reset `git reset --soft HEAD~1` would undo the latest commit but keep the files associated with that commit (it will take you back in time to the point just before you made your last commit). We'll explore these poweruser commands in more detail later.

Let's create a new branch (we'll call it `first`) to make our changes on:

```
git checkout -b first
```

If you run `git status` now you will see the output `On branch first`. If you run `git log` to see your commit history, you will see that you only have one commit. Your branch `first` is an exact copy of local master. You can switch between branches by running `git checkout branchname` replacing `branchname` with the name of the branch you want to switch to. As an excerise, switch now to `master` and examine your commit history, then switch back to the `first` branch.

Now, let's commit some changes on the `first` branch. Afterwards, we will try to merge our changes to local master. Start by creating a file in the `store` repo called `mysecondfile.txt`. Then run the add and commit commands:

```
- git add mysecondfile.txt
- git commit -m "My first commit on a branch"
```

Run `git log` to check the commit was successful. Now let's push our changes to remote. The following is an overview of that flow:

```
- Switch back to local master
- Get our colleagues changes from remote master
- Merge our changes into updated local master (2 commands)
- Push our changes to remote master
```

Here is the command flow:

```
- git checkout master
- git pull origin master
- git merge --squash first
- git commit -m "Commiting branch changes"
- git push origin master
```

If you check your GitHub repository now, you should see the file that you added in the remote repo. Now let's unpack what happened with the above commands.