Open in new tab

Git & Github

Lesson Duration: 60 minutes

Learning Objectives

- Understand the purpose of source code control
- Be able to use Git for source code control
  - Stage and commit changes
  - Inspect history and view previous versions of code
- Be able to integrate local repositories with remotes for off-machine backups

What Is Git

Let's imagine we live in a sad world with no sunshine, no kittens, no sweets and certainly no source control.

If you were a dev in this sad world you might have a few things you'd like to do:

1. Back up our code
2. Keep versions of our code as "milestones" to rollback to
3. View a history of what we have done
4. Share our code with other developers (beware working on the same file!)

How would we achieve this without git?

Git allows us to solve these problems.

Git is a version control system. This means it lets us control changes to documents, or in our case, source code. "Source code" being the files that make up your computer programs.

In a nutshell, tools like Git simplify and facilitate sharing files, and changes to files, between multiple developers.

Git Alternatives

Before we go into detail with Git, it's worth noting that Git isn't the first, or only source code control choice.

Historically, source code control was centralised. A central server would store the files for the repository. In such a system, each developer has a snapshot of the most recent version of the codebase, and would "check out" the files they wanted to edit. While they were working on them, no-one else could make any changes.

The administrators of the server have complete control over the code. No-one can access it unless they can access the server. This has the side-effect of locking developers to the office, where there is connectivity to their servers.

Distributed Source Code Control

Git's major difference is that every developer has a complete copy of the code base, and all its history. Although there may still be a "central" repository for all the developers to get and share updates between each other, there is no reason for the developers to be connected to any servers to do their work.

Git simplifies the merging of changes between these distributed repositories, and encourages frequent small changes to files to get rapid development.

Some Jargon

We call the collection of code stored in git the "repository".

When we flag a file's changes to be saved to git, we say that the changes are "staged".

When we save files to git, we say they have been "committed".

When we roll-back changes we say they have been "reverted".

When we have to resolve differences between a file that has been worked on at the same time by multiple people we call this a "merge".

Git Lifecycle

1. Git can be initialised on any directory - this defines our "repository"
2. Add a file and "commit" it to Git
3. Editing the file for Git to consider it modified
4. Stage the file to let Git know you want to record the changes in history
5. Commit the staged changes, and Git logs the changes in history and considers the file unmodified again

Similarly we might edit a file, git will see it has been modified but we can then tell git we are not happy with the changes and to roll it back.

Typical Workflow

Create a wee folder

```
mkdir git_demo
```

Move into this folder

```
cd git_demo
```

Initialise an empty repository

```
git init
```

You will notice that it says “master”. This is what is known as a “branch” - do not worry about this too much for now we will revisit it later. All you need to remember is that code committed to this “master” branch should reflect a current stable version of the code.

Add a couple of files to the directory

```
touch pikachu.txt charmander.txt
```

Now let’s see what the current situation is. We can ask git to show us the current status of our repository.

```
git status
```

Oh look, git is telling us that there are new files and that they are not staged. So let’s now do that.

Prepare the file to be committed by adding it to the staging area

```
git add pikachu.txt
git status
```

We’ve added one file, but the other is still not staged. If necessary, we can add **all** files at once.

```
git add --all
```

See the status of the files

```
git status
```

Git now tells us that the files are staged and ready to be committed. Let’s do that now.

Commit the changes

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

```
git log
```

Add thunderbolt to pikachu, flamethrower to charmander

```
atom .
```

It’s always best to call git status before git add. We don’t want to accidentally add any files which we didn’t mean to change.

```
git status
```

Git now shows that there are files that were modified.

Stage the changes.

```
git add --all
```

NOTE: We are staging CHANGES to the files, not the files themselves.

Commit the changes

```
git commit -m "adds basic moves"
```

We should always try to make regular, small commits with descriptive messages.

Distributing Our Code

It’s important to understand that with Git, we have a local copy of code (like we just made) but there is also remote copies of code that we can “pull” down or “push up” to.

There are many ways to host remote Git repositories; but one of the most common ways is to use GitHub.

This is a place to store repositories remotely, acting as a “central” copy of the source code where all participating developers can “push” their code.

This means that we an pull down a copy of open source code and contribute to it.

> Have a look at some of the cool projects on Github. Git itself is on there, and the Ruby language!

The great thing about this is that if our laptop blows up, we can pull down a clean working version of our code at any time.

Adding our Repository to GitHub

If we want to add a remote for our repository, our first step is to log into GitHub, and create a new, empty repository.

Name your repository, and (optionally) give it a description. *DON’T* initialise the repository with a README. If you accidentally do - no need to panic; just delete the new repository on GitHub and create one again (skipping the README this time!)

> Emphasise that students should choose a meaningful name for their repos that accurately describes their work. (Rather than “Codeclan Homework”, for example.) Stress that it is likely that future employers will want to see their Github accounts, so well-named repos are really helpful.

Copy the SSH path to your clipboard, and then use it to add a new remote to your Git repository:

> NOTE: Make sure it’s the SSH link (not HTTPS) otherwise you will always be prompted for your Github password

```
git remote add origin git@github.com:USERNAME/REPOSITORYNAME.git
```

Note that “origin” is just a name - we could call it anything. Origin is just a convention.

When you want to send your local code to GitHub, push it with `git push origin master`. This will push all the committed code in the local master branch (we’ll cover branches later) to the remote called ‘origin’

Updating Remote Repositories

When we make changes to files, these alterations are not automatically saved. We need to “push” our changes to out remote branch when we are ready.

Let’s add another move to Pikachu.

```
atom .
```

```
git push origin master
```

There may be time we commit locally, but not to the remote straight away. This means we can roll-back to a point in our local history if we are not happy with our changes. We will talk about this more in depth in another lesson.

Also we are quite lazy, we don’t want to always have to say git push origin master do we? We are not paid per character so let’s make our job easier.

To do this we can set the upstream so we don’t have to say where we want to push to every time.

```
git push -u origin master
```

Now we can just type

```
git push
```

And our lives are that much easier.

Note: You will have to do this for each new repository you create.

Similarly we can pull down other people’s changes with the “pull command”

```
git pull
```

Creating a local copy of an existing GitHub repository

If you want to make a local copy of a repository that is already on GitHub, you will use the “clone” command.

Open the repository you want to clone in GitHub, and copy the SSH path to your clipboard, and then use it to clone the remote to a new directory on your computer (ensure to not be in a directory already tracked by Git):

```
git clone git@github.com:USERNAME/REPOSITORYNAME.git
```

Summary

We’ve learned how to initialise a Git repository:

```
git init
```

We’ve learned how to add files to our staging area:

```
git add .
```

We’ve learned how to commit our staged files:

```
git commit -m "Your commit message"
```

We’ve learned how to push our code to Github (assuming that the repo is set up there):

```
git push origin master
```

We’ve learned how to get important information about our repository:

```
git status
git log
```

Further Investigation

We’ll leave you to investigate other Git resources, cheat-sheets, and tutorials. Please find out what the “.gitignore” file does.

GitHub does give you a GUI interface you can install, but we’re going to work with the command-line interface for the most part, because we may not always have a GUI available to us.

- Git Docs
- Interactive Tutorial
- Git Guide

Published with GitHub Pages