Durham University

Version Control: Getting Started

COMP51915 – *Collaborative Software Development*
Michaelmas Term 2024

Christopher Marcotte[1]

[1] For errata and questions please contact christopher.marcotte@durham.ac.uk

Outline

- Navigating the GitHub website
- Command line `git` usage
- Confusing `git` terminology
- Troubleshooting techniques

Learning Outcomes

- Ability to create and use a local repository with `git`
- Ability to interact with a repo on the Github website
- Ability to connect a local repo with the remote one
- Ability to understand when something has gone wrong with `git`

Navigating the GitHub website

GitHub Website features

- User Page
  - The only subpage that's relevant for us is *Repositories*
- Repository Page
  - Issues – List of problems with the content in a repo
  - Pull requests – Proposed solutions to problems with the content in a repo
  - ~~Discussions – A built-in forum for a repository~~
  - Actions – Configuration of automated procedures for the repo
  - ~~Projects – Organizational tooling for Issues and Pull requests~~
  - ~~Security – A policy for reporting security issues responsibly~~
  - ~~Insights – Statistical perspectives on a repo~~
  - Settings – repository settings; importantly: access rules

Command line `git` usage

The website is fine, but we can't spend all our time there.

Open up a terminal[2] and enter the command:

```
git --version
```

[2] On recent Windows: Windows Terminal; on macOS: Terminal; on Linux: *you already know*.

Command line `git` usage

The website is fine, but we can't spend all our time there.

Open up a terminal[3] and enter the command:

```
git --version
```

If `git` is installed, you should see something like `git version 2.43.0`.

If `git` is not installed, or it is very old (`v2.46.1` is current), get it from here.

[3] On recent Windows: Windows Terminal; on macOS: Terminal; on Linux: *you already know*.

Essential `git` commands

In this lecture I will discuss some essential `git` commands:

Essential `git` commands

In this lecture I will discuss some essential `git` commands:

- `git init` & `git clone`
- `git add`
- `git commit`
- `git push` & `git pull`

Essential `git` commands

In this lecture I will discuss some essential `git` commands:

- `git init` & `git clone`
- `git add`
- `git commit`
- `git push` & `git pull`

These are the basic commands for using `git` productively by **oneself**.

We will look at *maintenance* and *collaboration* in the next two lectures.

Initializing a repository

In your terminal, move to your development directory[4]:

```
mkdir -p ./COMP51915
cd COMP51915/
git init .
```

[4] Do you have a development directory? For me, it is `~/Development/`

[5] Historically, the most fundamental branch was called *master*; in recent efforts to decolonize the field, people have moved to calling this branch *main*.

[6] The `.*` means 'all the hidden stuff in the current directory'.

Initializing a repository

In your terminal, move to your development directory[7]:

```
mkdir -p ./COMP51915
cd COMP51915/
git init .
```

You may get a *hint* here about the branch name used[8], which you can change.

[7] Do you have a development directory? For me, it is `~/Development/`

[8] Historically, the most fundamental branch was called *master*; in recent efforts to decolonize the field, people have moved to calling this branch *main*.

Initializing a repository

In your terminal, move to your development directory[10]:

```
mkdir -p ./COMP51915
cd COMP51915/
git init .
```

You may get a *hint* here about the branch name used[11], which you can change.

Calling `ls .*` yields[12]:

```
.git:
  HEAD        description info        refs
  config      hooks       objects
```

[10] Do you have a development directory? For me, it is `~/Development/`

[11] Historically, the most fundamental branch was called *master*; in recent efforts to decolonize the field, people have moved to calling this branch *main*.

[12] The `.*` means 'all the hidden stuff in the current directory'.

Cloning a repository

Instead of initializing an empty repository, you can clone an existing one.

```
git clone <url-ending-in.git>
```

This copies a GitHub repository from that url to the current directory, which may or may not be used with GitHub.

Cloning a repository

Instead of initializing an empty repository, you can clone an existing one.

```
git clone <url-ending-in.git>
```

This copies a GitHub repository from that url to the current directory, which may or may not be used with GitHub.

You'll often be using `git clone` with source-only software that you wish to *use*.

If you want to *contribute* to the software, you'll frequently use a GitHub *fork*.

We'll discuss this distinction in more detail when discussing collaboration.

Adding files to a repository – *staging*

We now have a git repository, in our current directory.

Create two files, e.g. with `touch file1.c file2.py`. Preferably these should have *something* in them.

How do we get `git` to track these files?

[13] Except those under `gitignore`; this is useful if you're programming something in a single language and want everything included every time.

Adding files to a repository – *staging*

We now have a git repository, in our current directory.

Create two files, e.g. with `touch file1.c file2.py`. Preferably these should have *something* in them.

How do we get `git` to track these files?

At the command line:

```
git add file1.c file2.py
```

Alternatively: `git add *` to add *all* files in the current directory to `git`.[14]

[14] Except those under `gitignore`; this is useful if you're programming something in a single language and want everything included every time.

`git add` limitations and caveats

The `git add` command updates the `git` index with the current content.

`git add` limitations and caveats

The `git add` command updates the `git` index with the current content.

You can check that a file has been added correctly with `git status`:

```
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
  new file:   file1.c
  new file:   file2.py
```

This is telling us that:

- we are on the *main* branch
- there are no commits
- we've staged the files to be committed

Committing to a repository

Our files have been staged with `git add`, but we have not yet *committed* them to the repository.[15] To do so:

```
git commit -m "adding file1.c file2.py to repo"
```

[15] On my terminal, before the commit the directory was adorned with a ×, afterward with a ✓.

[16] We will discuss resolution strategies later in the session.

Committing to a repository

Our files have been staged with `git add`, but we have not yet *committed* them to the repository.[17] To do so:

```
git commit -m "adding file1.c file2.py to repo"
```

First you must *stage* your changes with `git add` – listing the changed files you want to eventually update – and then you commit.

[17] On my terminal, before the commit the directory was adorned with a ×, afterward with a ✓.

[18] We will discuss resolution strategies later in the session.

Committing to a repository

Our files have been staged with `git add`, but we have not yet *committed* them to the repository.[19] To do so:

```
git commit -m "adding file1.c file2.py to repo"
```

First you must *stage* your changes with `git add` – listing the changed files you want to eventually update – and then you commit.

This is often the point where people run into issues in collaborative development – several people have made changes and want to commit them, but they conflict.[20]

---

[19] On my terminal, before the commit the directory was adorned with a ×, afterward with a ✓.

[20] We will discuss resolution strategies later in the session.

Interlude: Good commits

There is, in effect, a style guide for commits endorsed by `git`.

1. Commit small changes consistently, rather than large changes occassionally
2. Make your changes address one particular issue, rather than several
3. ensure your commit messages are descriptive, on-topic, and conform to the expectations of the project

If you run `git diff --check` before committing, then `git` will notify you of some common issues in your files before you commit them.

Interlude: Setting up `ssh` access

In a local-only repo this is the end of the story. Edit, `git add`, `git commit`…

Interlude: Setting up `ssh` access

In a local-only repo this is the end of the story. Edit, `git add`, `git commit`…

You will mostly use `git` in tandem with your GitHub account – i.e. with `ssh`.

Setting this up is not difficult but it does take a little time. Broadly two choices:

- `ssh-agent` forwarding (convenient, secure; limited, windows issues)
- manually creating and adding public-private-key pairs to your account

Interlude: Setting up `ssh` access

In a local-only repo this is the end of the story. Edit, `git add`, `git commit`…

You will mostly use `git` in tandem with your GitHub account – i.e. with `ssh`.

Setting this up is not difficult but it does take a little time. Broadly two choices:

- `ssh-agent` forwarding (convenient, secure; limited, windows issues)
- manually creating and adding public-private-key pairs to your account

I use the latter *on my machine(s)*, but this is a personal decision and I leave it to you to read about it and decide for yourself.

Pushing & Pulling (with `ssh`)

With `ssh` we can interact with remote repositories.

`git push` and `git pull` are commands for interacting with a remote repository from your local repository.

`git push` instructs the `git` program to *push* the commits in the current local repository to the remote counterpart. I.e., puts your local changes on GitHub.

`git pull` instructs the `git` program to *pull* the changes from the remote repository to your local copy. I.e., puts GitHub stuff in your local repository.

Pushing & Pulling (with `ssh`)

With `ssh` we can interact with remote repositories.

`git push` and `git pull` are commands for interacting with a remote repository from your local repository.

`git push` instructs the `git` program to *push* the commits in the current local repository to the remote counterpart. I.e., puts your local changes on GitHub.

`git pull` instructs the `git` program to *pull* the changes from the remote repository to your local copy. I.e., puts GitHub stuff in your local repository.

You need `ssh` access to a corresponding GitHub repo to use these commands.

Back to our example: `git push`

We run `git status` and see:

```
On branch main
nothing to commit, working tree clean
```

[21] Many people prefer to simply *create* the repo on GitHub from the beginning, rather than locally, and clone the empty repo to create the local workspace.

Back to our example: `git push`

We run `git status` and see:

```
On branch main
nothing to commit, working tree clean
```

The command `git remote add <name> <url>` connects our *local repo* to the *remote repo* at `<url>`.

Having done so[22], we can run `git push` to push our local files to the remote repo. Later on, someone else may wish to `git pull` our updates!

[22] Many people prefer to simply *create* the repo on GitHub from the beginning, rather than locally, and clone the empty repo to create the local workspace.

Confusing `git` terminology

We need to discuss some distinctions.[23]

- forking and `git clone`
  - forking a repository is a GitHub feature; it makes a copy of the repo under your name and is the first step to contributing to someone else's software
  - `git clone` is a `git` feature – this creates your own local copy of an extant repo, which you may or may not use with GitHub
- Pull requests are a GitHub feature – except there is a `git request-pull` command for accomplishing the same thing

[23] Julia Evans – amazing, watch her talks! – has spent considerable time using, thinking about, and cataloguing `git`'s usability shortcomings.

Troubleshooting techniques

Many things can become frustrating showstoppers for your repository, though the `git` data model should prevent data loss.

In particular, we'll run into issues with collaboration on a single code-base, like merge conflicts, and we'll discuss strategies for addressing these.

When you run into a `git` error:

1. read the hint output by the program, and then
2. search the web for advice.