Getting Started with Git

Simon Wells

Aims

At the end of the practical portion of this topic you will:

- Be able to log into the University system
- Be able to run simple programs from the command line
- Remember how to program simple software
- Have retrieved some supplementary texts from an online publisher

1 Introduction

Git is basically a multi-tool for working with text-based files (which is basically all source code). If we want to keep a history of changes to a file, if we want support in integrating changes from lots of people, if we want assistance in tracking down who, when, and where bugs have been introduced to a code-base. Git helps in all these things and many more.

There is a lot more to Git than just what is covered in this document. You should be practising with Git, turning the add-commit-push cycle into a habit but also learning additional features as you go along. Using Git counts as a professional skill for most developers and also for the people working alongside them because it is a great collaboration tool once you know how to use it well.

NOTICE: Make sure to type in commands rather than copying & pasting from this document. Whilst typing can introduce typos, copying and pasting frequently introduces errors, particularly white-space related errors, that can cause things to go awry. More importantly, typing in commands is a part of your deliberate practise, because our minds can gloss over things whilst reading about them, but actively doing something can help to make it stick.

2 Getting The Software

Git on Windows is a bit different to Git on other platforms so this tutorial tries to make things similar between each platform. However there will always be slight differences. Once you have the software installed your experiences of Git should be comparable between platforms and you shold be able to directly translate your experiences from Window to Linux to OS X and back again (more or less).

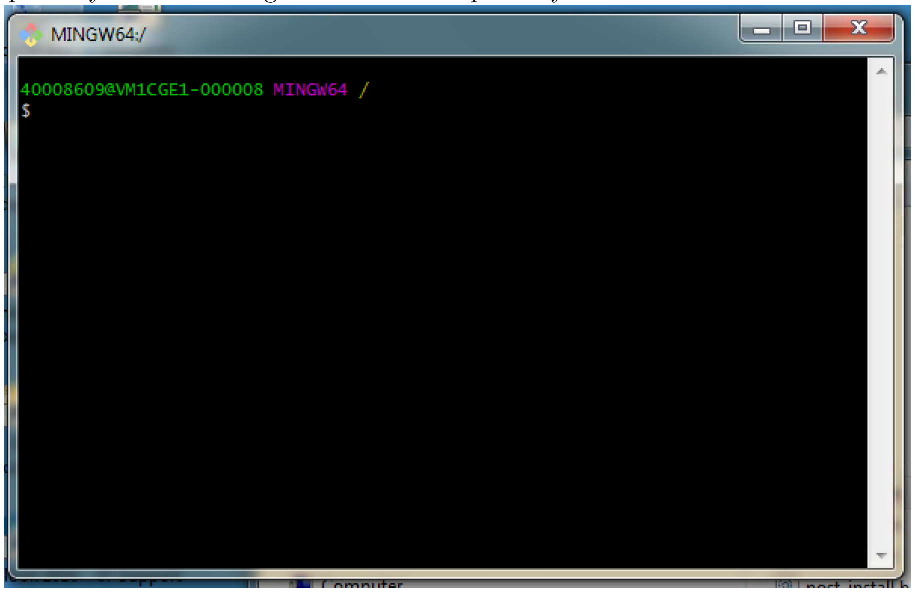
2.1 Windows

If you're on Windows go to the Git SCM website[1] - don't download the software that automatically starts downloading but instead choose the link labelled "PortableGit" (nowadays the 64-bit version is usually best [2]. Let it download, click the exe and extract/install it somewhere useful.

When running Git you will want to click the git-bash.exe rather than git-cmd.exe. They work in different ways and the commands differ between the two. Using git-bash.exe means that we can use the same commands regardless of whether we are on a Mac, Linux, or Windows machine.

[1] `https://git-scm.com/download/win`
[2] `https://github.com/git-for-windows/git/releases/latest`

When you run git-bash you will get a window that looks like a command terminal. It contains enough command line commands and Git software for you to navigate the file system, create and delete files and folders, move them around, then do all of the usual Git stuff in addition like cloning a repository or committing new files to a repository.



```
40008609@VM1CGE1-000008 MINGW64 /
$
```

2.2 MacOS

Install the Apple Command Line Developer tools (you should really have these anyway if your developing software) because one of the tools that's installed is Git:

```
$ xcode-select --install
```

Once the developer tools are installed you can open a terminal and type the commands that are used in the remainder of this tutorial. Note that some examples, like references to the 'H:' drive aren't applicable to a Mac or Linux machine.

2.3 Linux

Assuming a Debian based system:

```
$ apt-get install git
```

3 The Most Basic Use Case for Git

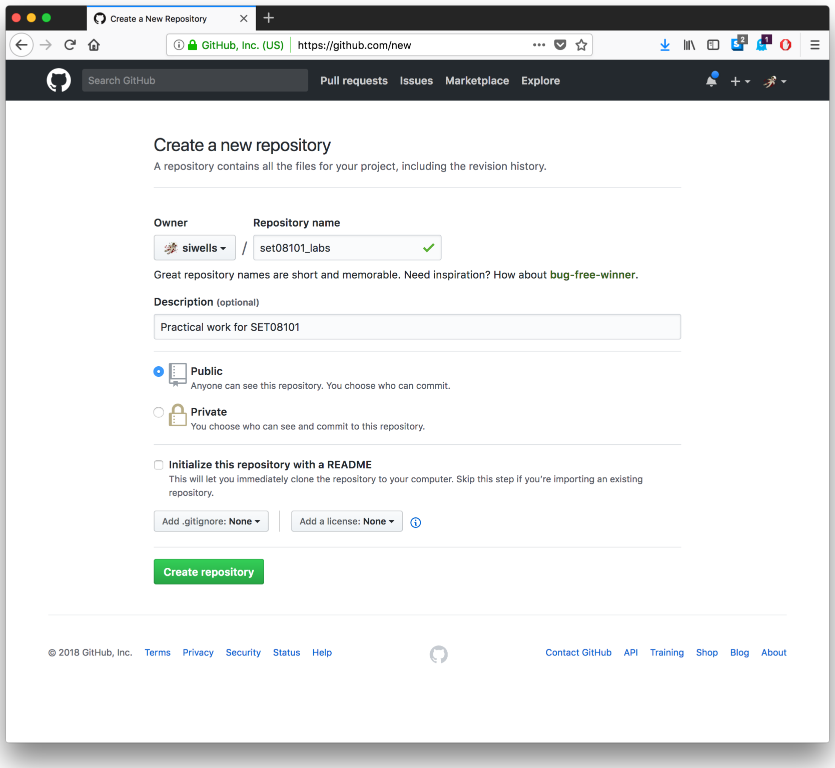
We'll start by assuming that you're using a GitHub account. If not you might need to adjust some of the command a little. Start git-bash then use the following commands so that Git can label your commits with your name (obviously replacing 'Your Name' for your real world name in double quotes):

```
$ git config --global user.name ``Your Name''
```

Now tell Git which email address to associate with your commits. You should use the email address that you signed up to GitHub with:

```
$ git config --global user.email ``example@youremail.com''
```

Now, log into GitHub and create a repository. Give it a useful project name and description.


Create a New Repository
GitHub, Inc. (US) https://github.com/new
Search GitHub
Pull requests Issues Marketplace Explore

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: siwells / Repository name: set08101_labs

Great repository names are short and memorable. Need inspiration? How about bug-free-winner.

Description (optional): Practical work for SET08101

Public — Anyone can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

Initialize this repository with a README — This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

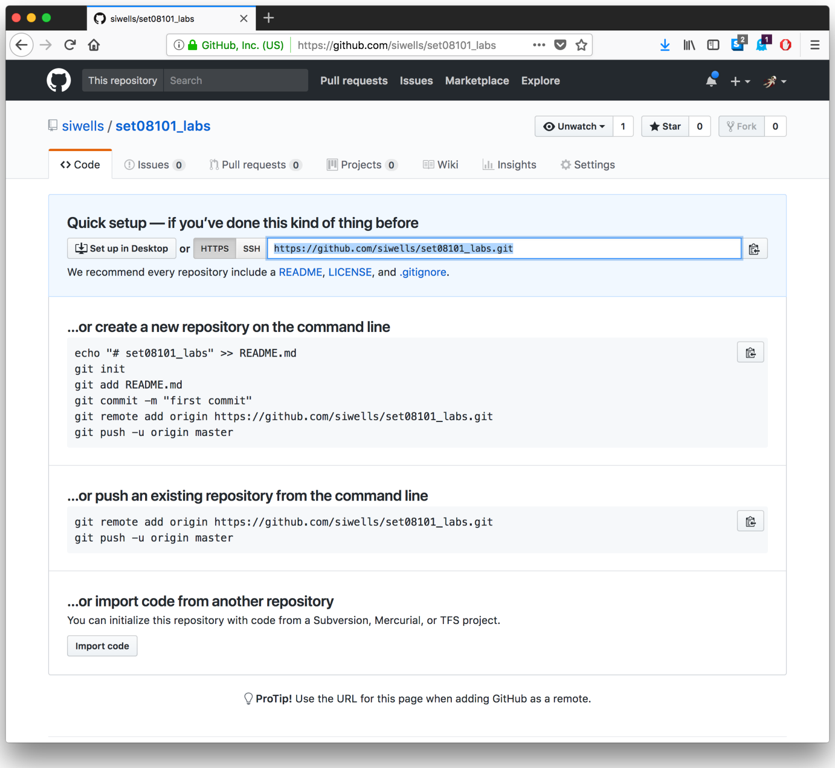
Add .gitignore: None | Add a license: None

Create repository

© 2018 GitHub, Inc. Terms Privacy Security Status Help
Contact GitHub API Training Shop Blog About

Once the repository is created you will be able to clone your empty repository. This means make a copy of the repository that is on GitHub on your local machine. The address for this is available from GitHub under the green ”Clone or download” button. You need to copy and paste the address, which looks something like this:

`https://github.com/siwells/daka.git`


siwells/set08101_labs
GitHub, Inc. (US) https://github.com/siwells/set08101_labs
This repository Search
Pull requests Issues Marketplace Explore
siwells / set08101_labs
Unwatch 1 Star 0 Fork 0
Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `https://github.com/siwells/set08101_labs.git`

We recommend every repository include a README, LICENSE, and .gitignore.

…or create a new repository on the command line

```
echo "# set08101_labs" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/siwells/set08101_labs.git
git push -u origin master
```

…or push an existing repository from the command line

```
git remote add origin https://github.com/siwells/set08101_labs.git
git push -u origin master
```

…or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

ProTip! Use the URL for this page when adding GitHub as a remote.

so that you can use it in a command in git-bash. Note that the address contains your GitHub user name and the repository name that you used when you created the repository. You can retrieve an address that starts with either "https" or "git@github.com". For the moment we should use the one starting with "https".

In your GitBash terminal you should navigate to a place on your machine where you are going to store your code, e.g. on lab machines you might want to use the H: directory. I'll assume H: from now on but your should organise your work how you see fit.

```
    $ cd h:/
    $ mkdir daka
    $ cd daka
```

These three commands changed the directory we were in (cd is short for change directory) then made a new directory in the new location (mkdir is short for make directory). Finally we changed directory again to move into the new daka directory.

Now we'll clone our repository from GitHub:

```
    $ git clone https://github.com/siwells/daka.git
```

NB. I've used the clone address for this module's repository but your should replace the address, everythin after "git clone", with what you copied from GitHub earlier.

There should be some messages on screen, and when they're done there should be a message along the following lines:

```
    You have successfully cloned to an empty repository.
```

If you now type ls you should get a list of folders and one of the folders will be named for the repository that you just cloned from GitHub. This means that there is an exact copy of your repository on GitHub and one on your local machine. You can make as many clones of a repository as you like on as many machines as you like.

Now you can change directory into your project folder and run Git commands on your fresh repository:

```
    $ cd daka
    $ git status
```

The output from the git status command will look quite cryptic at the moment but you'll get used to it and quickly discover that it is telling you really important stuff about your repository and code. You can also open this folder in Window Explorer (of finder on a Mac) to manipulate your files if you find that easier but you will always need to use the git-bash program to update your repository and access the Git commands.

If we add a file, e.g. hello.html to our daka folder then run git status again we should get a different message to before, one telling us that the repository has changed because, well, we just added a new file, and it is Git's job to notice changes. Note that Git commands only work properly within a Git repository, that means you have to cd into a folder that is within your project that Git is tracking.

Whenever we add a new file to our project, or make a change to an existing file, we will need to tell Git to take note of the changes since last time. We do that with the add command:

```
    $ git add .
```

The '.' just means add all files in the repository that are either new or changed. You can also specify individual files that you want to add if you want to alter the granularity of the changes that Git tracks. Once your changes are added then you will need to commit them, this means that Git enters the changes it noticed from the add command into a record, the history, of the repository:

```
  $ git commit -m "a commit message"
```

"a commit message" should be replaced with a short descriptive message about the changes that are being committed this helps remind you, and inform others, of why the changes that are being committed were made, e.g. "Fix file save bug" or "add new edit widget".

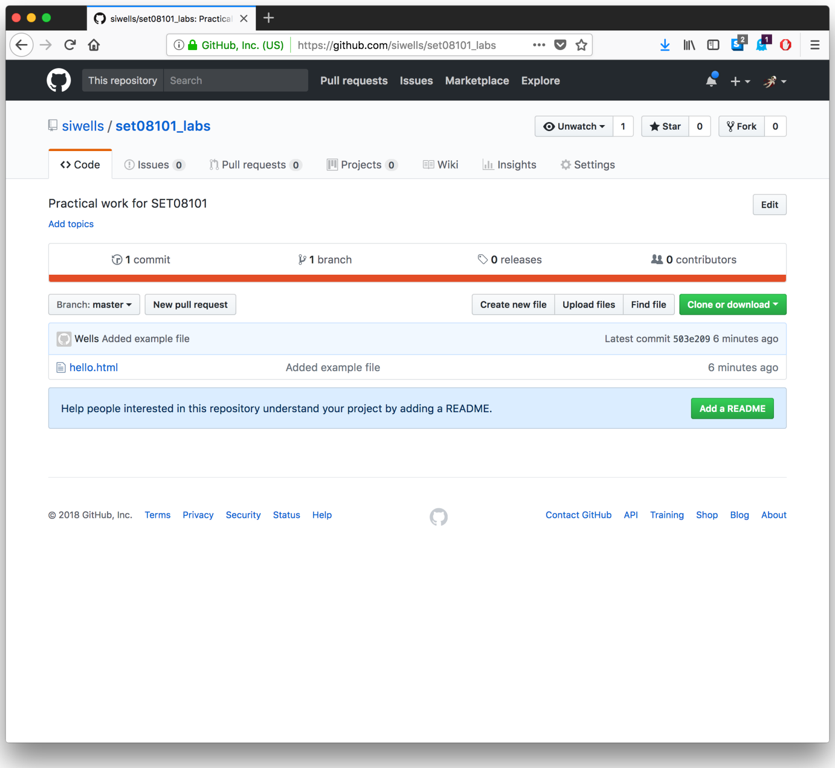
As we develop we will keep doing this cycle of add then commit so that each change we make to our code is tracked with a descriptive message of the change. The list of messages and changes is called the history and you can inspect this using:

```
  $ git log
```

However adding and committing is only tracking changes locally. If we want to make sure that there is a copy of our repository on GitHub then we need to "push" changes to GitHub. We do this with the push command:

```
  $ git push
```

You should now be able to see the changes on the repositories GitHub page (you might need to refresh it first). Note that if this hasn't worked then you can try adding the repository address to the end of the command, e.g. $ git push https://github.com/siwells/daka.git - although make sure to change this address for your own as you won't have permissions to update the daka repository.



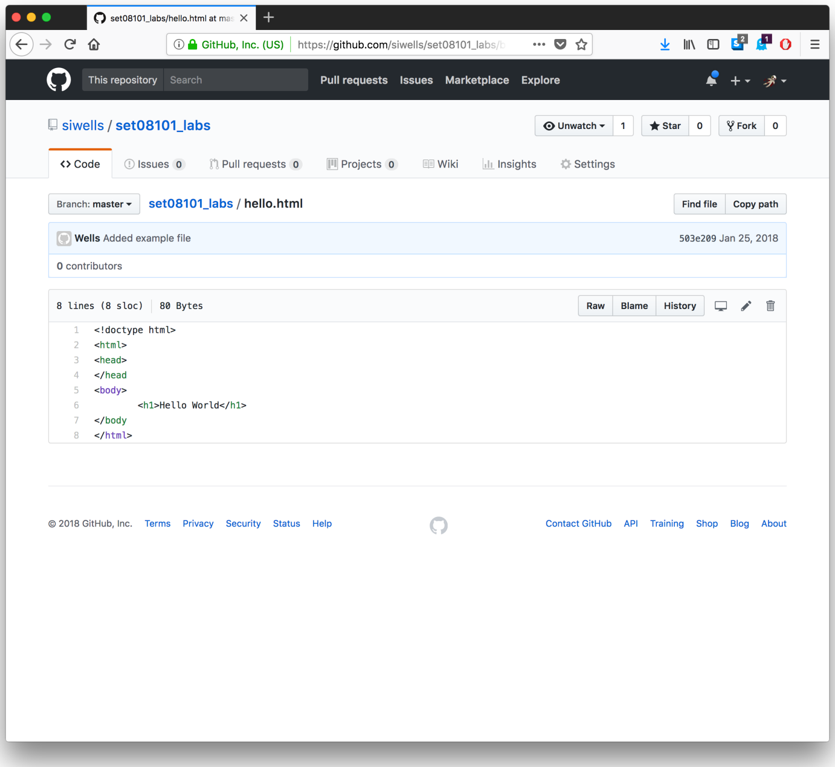
Practical work for SET08101

Wells Added example file   Latest commit 503e209 6 minutes ago

hello.html   Added example file   6 minutes ago

Help people interested in this repository understand your project by adding a README.

You can also use the Github interface to look at the files you've made changes to. This is useful sometimes to confirm that what you think is stored in GitHub actuall is, e.g.



Wells Added example file 503e209 Jan 25, 2018

0 contributors

8 lines (8 sloc) | 80 Bytes

```
<!doctype html>
<html>
<head>
</head
<body>
        <h1>Hello World</h1>
</body
</html>
```

We need to push our work periodically so that we keep our "remote" repository up to date with any changes we've made. This gives us a useful backup of our work and also means if we want to work on our code elsewhere then all we have to do is clone our repository to the machine we are on, make changes, add and commit them, then push the changes back to GitHub.

If you ever make changes to your repository from a different machine, e.g. at home, and push them to GitHub you will need to get those changes onto your lab machine. We do this with the opposite of push, we pull. Running the git pull command in your repository should fetch all new changes that are not already known locally and apply them to your code (so long as you haven't already made changes locally, in which case things get a little more complicated - but that is a story for another time [or a job for you self-directed study]).

4 Going Further

This is only scratching the surface of Git usage. There are also numerous interactive tutorials and resources to help you get started with Git. I'd suggest reading at least the Git parable (which explains, through example, why Git is useful) and the learn Git in 15 minutes tutorial. The important thing is to actually type in the commands and see how things work in practise rather than just reading:

1. Github's Git 'Hello World Exercise': `https://docs.github.com/en/get-started/quickstart/hello-world`
2. Learn Git Branching `https://learngitbranching.js.org/`
3. Git Immersion `http://gitimmersion.com/lab_01.html`

To set all of this in context and answer questions about why we needs all this stuff, it can be useful to read the Git parable. This uses a story, scenario, and some challenges, to demonstrate how the things that Git does satisfy a lot of things that we frequently need to do when working on real world projects.

1. The Git Parable: `http://tom.preston-werner.com/2009/05/19/the-git-parable.html`