Introduction

Tutorial 3

Owen Huyn

January 23, 2017

What is version/source control?

From Wikipedia:

"The management of changes to documents, computer programs, large web sites, and other collections of information."

- Tracks and provides control over changes to source code
- Used to keep a history of code (versions) over a period of time
- Provides useful documentation of different aspects of the code submitted over time
- Allows developers to work simultaneously (you won't see this in the course)

Why use version/source control?

- How would you see the progression of code as one develops on it?
- Or if there was a bug and you did not know where it occurred?
- How do you work with others concurrently on the same code base without having conflicts?

With version control, this can all be solved

- Code on version control would have a history associated to it allowing one to go back to an older version
    - Kind of like system restoring your code
- An analogy would be like saving your video game every time you reached a major milestone

Some popular version control tools

- Git
- SVN
- Mercurial
- TFS

Why is Git worth learning?

- It is a popular version control tool
- Nearly every software development company uses a version control tool
    - Once learned, switching between version control tools is a cinch
- Widely used by the open-source community (GitHub)
- Many large scale development projects use Git (in research and the industry)

How do I learn Git?

- This tutorial is only meant to teach can only teach you so much about Git
- There are great videos on YouTube and tutorials by simply Googling Git
- Many questions on Git are answered on StackOverflow, so look there
- **The best way to learn is to practice, practice, practice!**
- It is almost a necessity to learn at least one version control tool for CS/SE grads (**extremely** useful skill!)

Where do I start? – Ubuntu

On Ubuntu:

Run this command in your terminal.

```
apt-get install git
```

Let this run until it is finished.

Where do I start? - macOS/OS X
Two main options

Using the installer GUI:

Download from here and run through the instructions:

http://sourceforge.net/projects/git-osx-installer/

If you have Homebrew installed:

Then run: `brew install git`

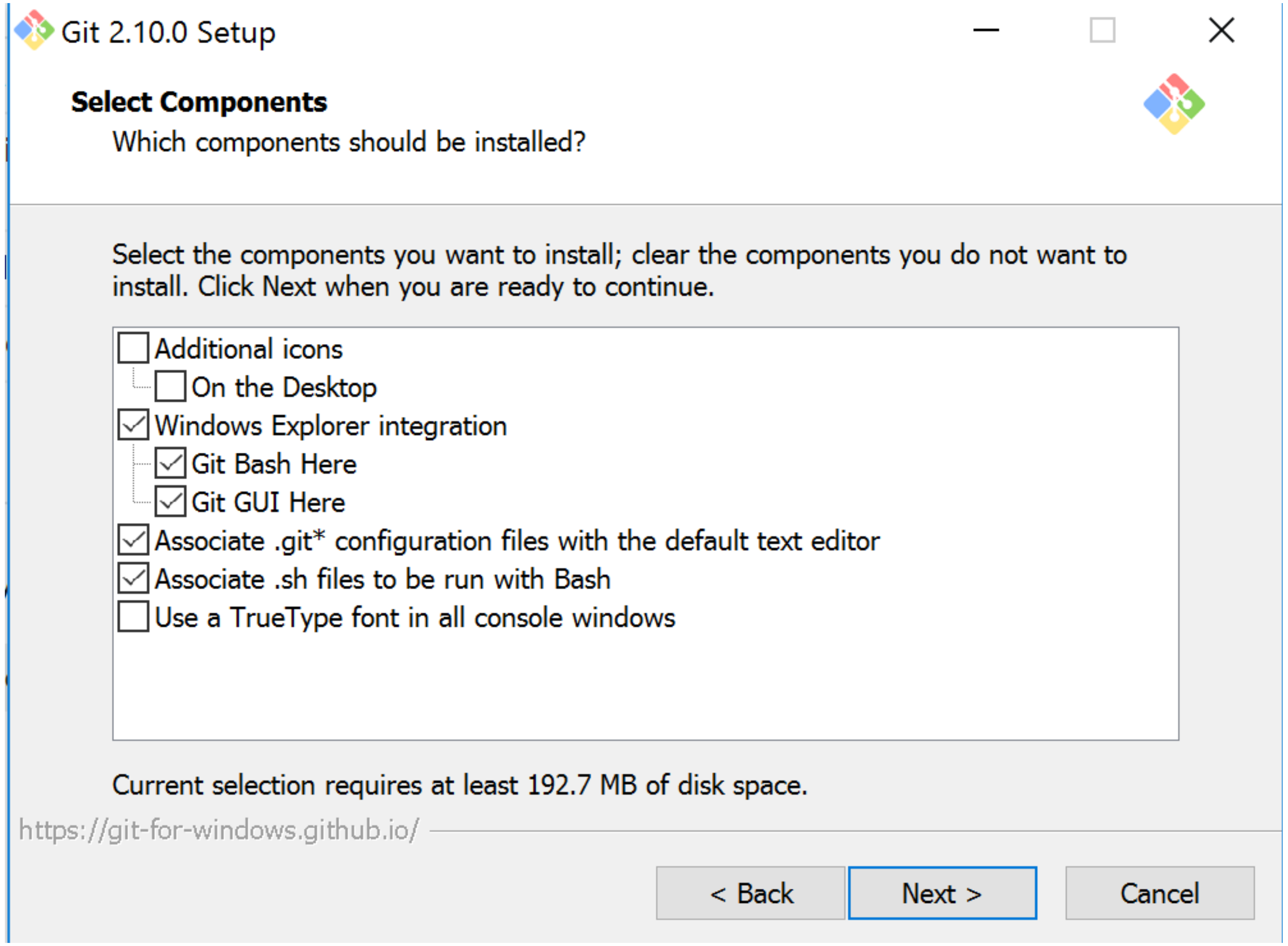
You may need to sudo to install, eg) `sudo brew install git`

Where do I start? - Windows

Download from here:

https://git-scm.com/download/win

Run the file when the download is complete.



Git 2.10.0 Setup

Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

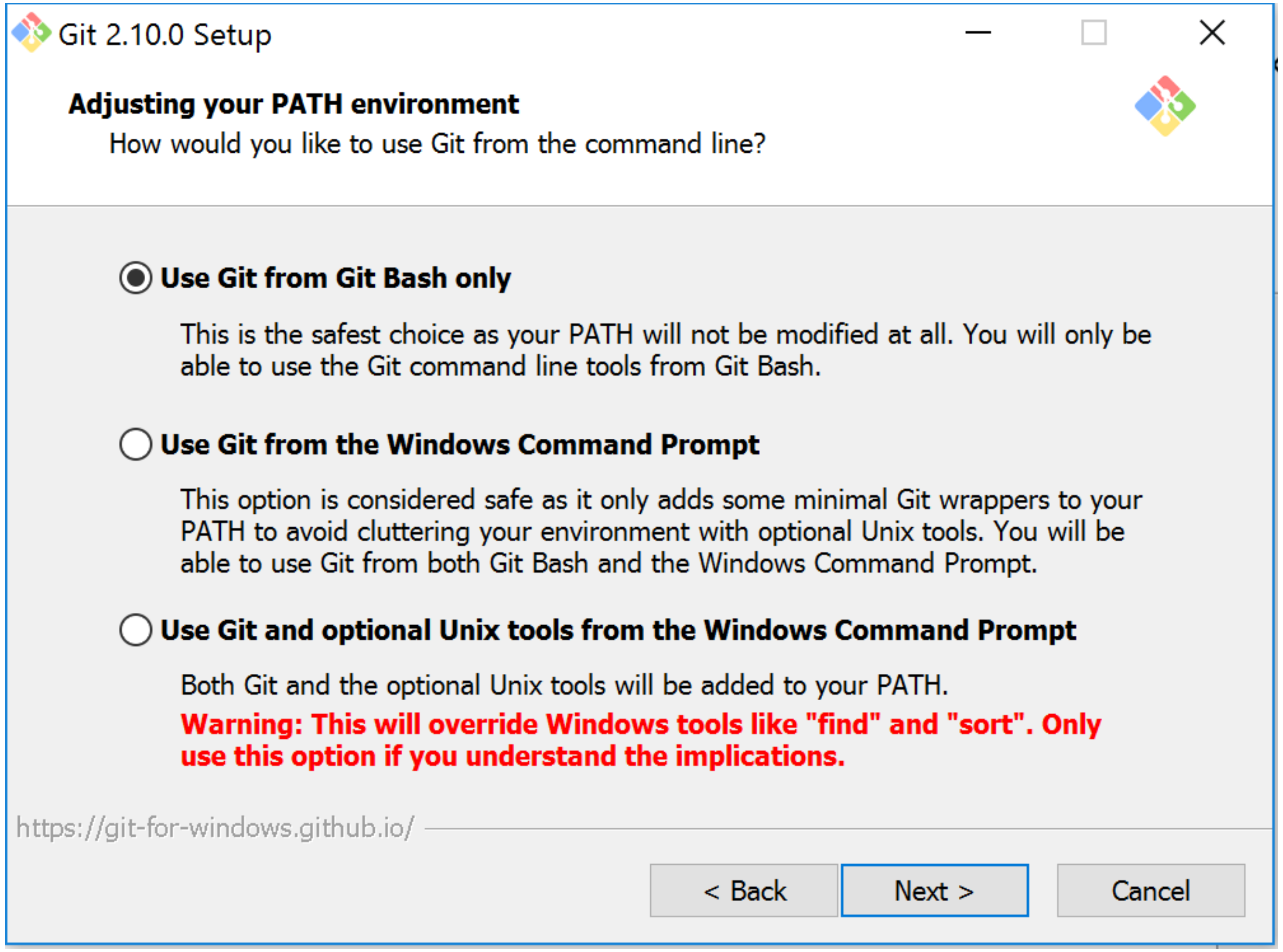
- [ ] Additional icons
  - [ ] On the Desktop
- [x] Windows Explorer integration
  - [x] Git Bash Here
  - [x] Git GUI Here
- [x] Associate .git* configuration files with the default text editor
- [x] Associate .sh files to be run with Bash
- [ ] Use a TrueType font in all console windows

Current selection requires at least 192.7 MB of disk space.

https://git-for-windows.github.io/

Where do I start? - Windows

- Make sure these settings are on
- Associate .sh files to be run with Bash is completely up to you to decide if you want Git to handle Bash files
- Press next when complete



Git 2.10.0 Setup

Adjusting your PATH environment

How would you like to use Git from the command line?

[✓ **Use Git from Git Bash only**]

This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

Use Git from the Windows Command Prompt

This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

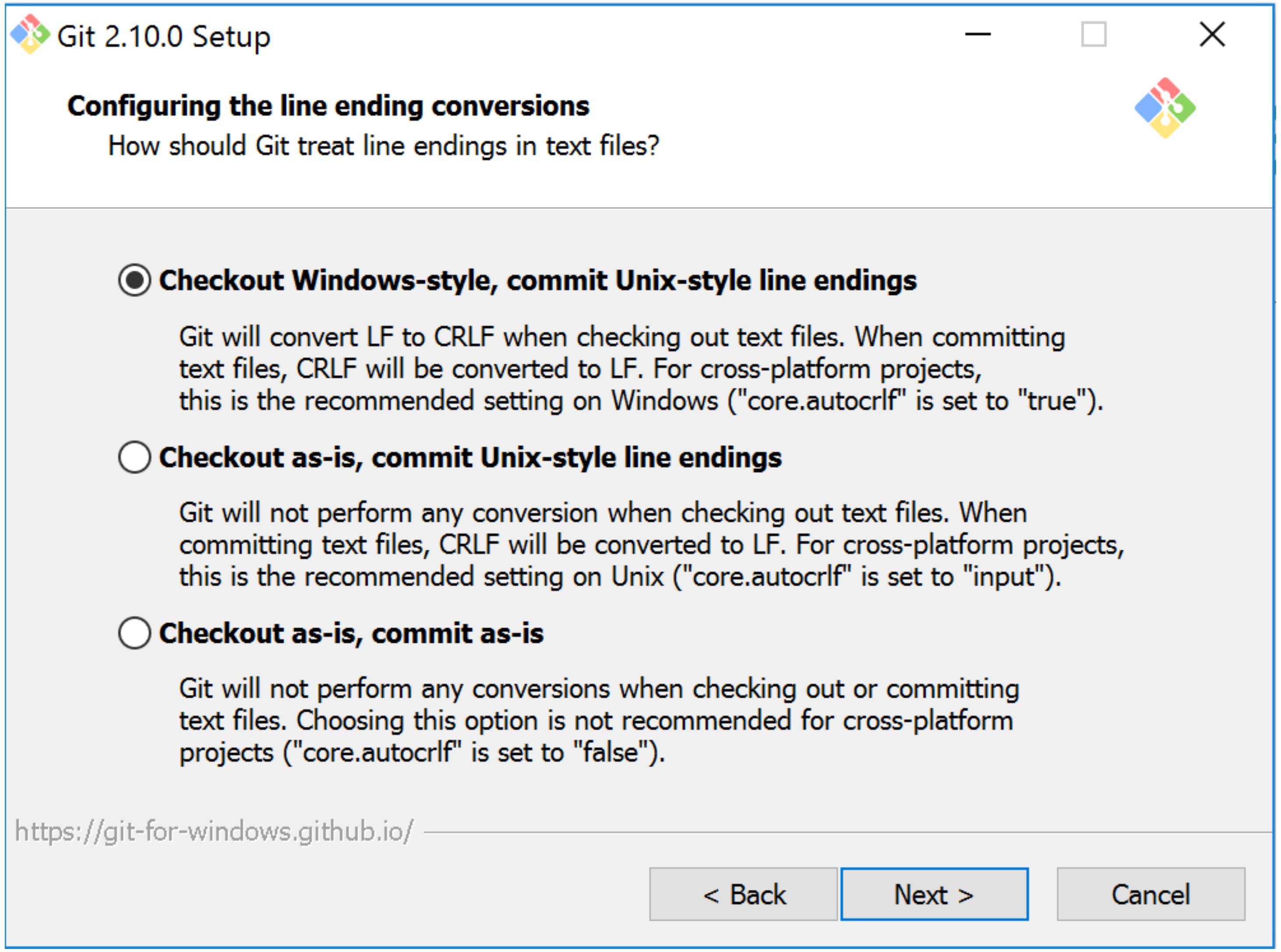
Use Git and optional Unix tools from the Windows Command Prompt

Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

https://git-for-windows.github.io/

Where do I start? - Windows

- Git Bash is a command line interface that knows Bash commands
- Git Bash can also SSH into Moore/Mills
- I use this as my daily driver for most of my Bash related work
- If you want to integrate Git into the Windows Command Prompt, you can select the second option
- **I recommend the first option**



Configuring the line ending conversions

How should Git treat line endings in text files?

[✓ **Checkout Windows-style, commit Unix-style line endings**]

Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

Checkout as-is, commit Unix-style line endings

Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

Checkout as-is, commit as-is

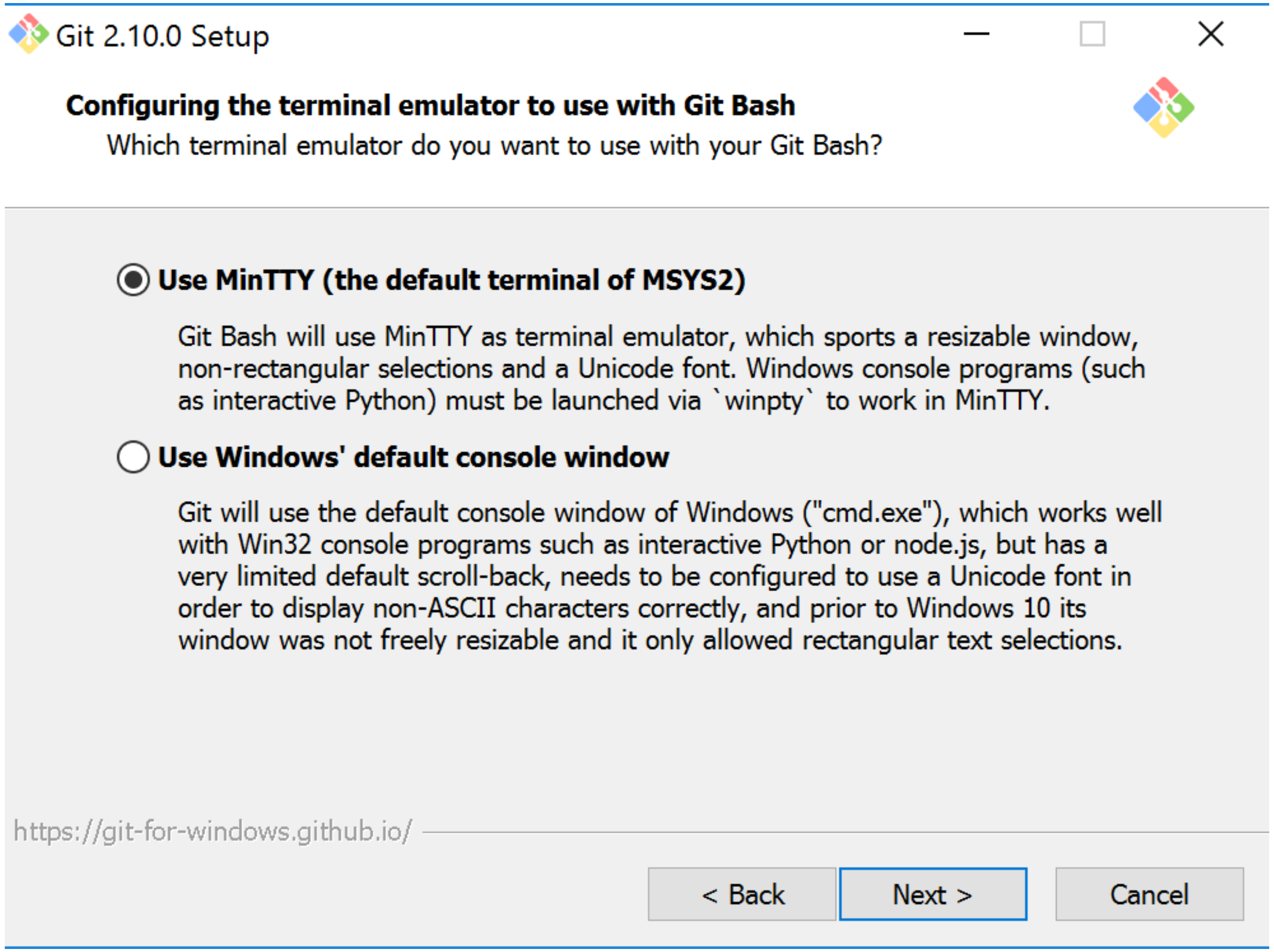
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

Where do I start? - Windows

- A problem with Windows is that Unix line ending and Windows line endings are not the same character
- This results in change conflicts even though the code may look exactly the same
- **The first option is preferred**

Where do I start? - Windows

- Use the first option



Git 2.10.0 Setup

Configuring the terminal emulator to use with Git Bash

Which terminal emulator do you want to use with your Git Bash?

[✓ **Use MinTTY (the default terminal of MSYS2)**]

Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

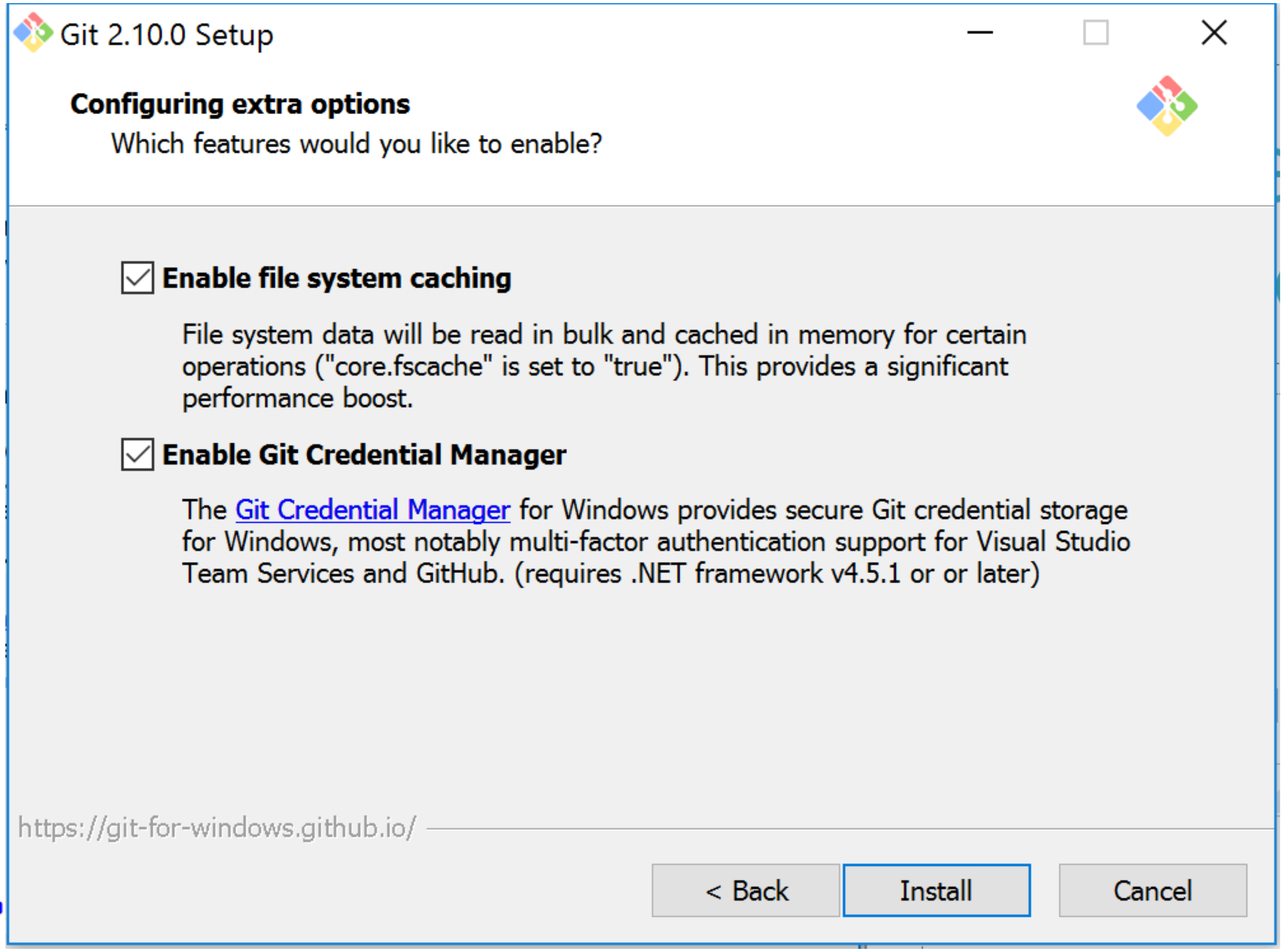
Use Windows' default console window

Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

https://git-for-windows.github.io/

Where do I start? - Windows

- Check those two options and click Install



Git 2.10.0 Setup

Configuring extra options

Which features would you like to enable?

[✓] **Enable file system caching**

File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.

[✓] **Enable Git Credential Manager**

The Git Credential Manager for Windows provides secure Git credential storage for Windows, most notably multi-factor authentication support for Visual Studio Team Services and GitHub. (requires .NET framework v4.5.1 or or later)

https://git-for-windows.github.io/

Great you’ve installed Git! Now open it

Ubuntu/macOS

- Open your Terminal and type in Git, it should be installed

Windows

- Type in “Git Bash” in your application search bar and open the application

A bit into version control

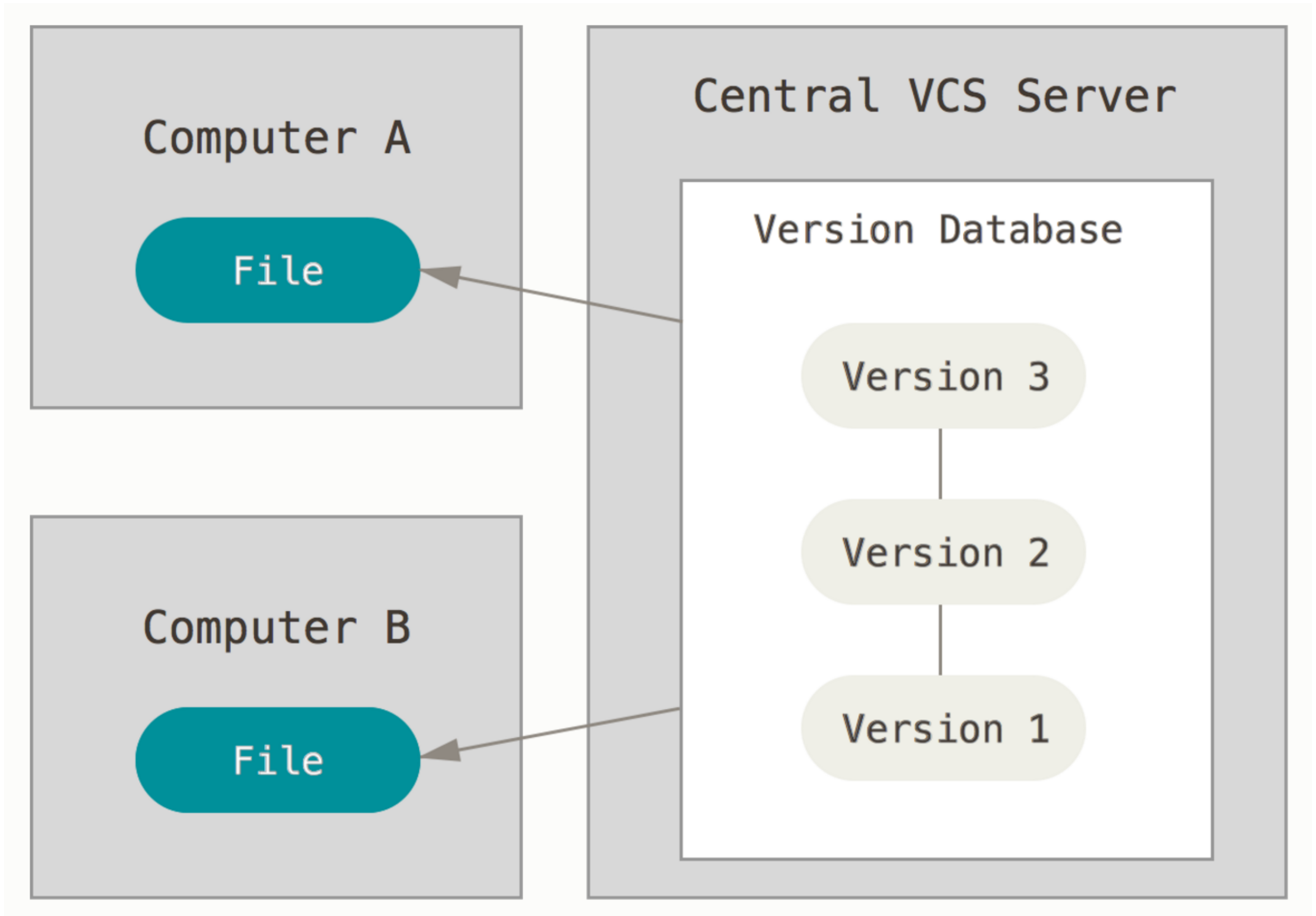
There are two main types of version control out there:

1. Centralized version control
2. Distributed version control

Centralized version control
(not Git)

- People needed to collaborate with developers on other systems
- Centralized version control was created as a result
- A single server contains all the versioned files and all clients would ‘checkout’ files from this central server
- Clients would have a ‘**snapshot**’ of the current state of the server repository

Centralized version control
(not Git)



Centralized version control

(not Git)

This was the standard for many years

- Many still use this within the community, however this is slowly changing

Advantages:

- Everyone knows what everyone else is doing to a certain degree
- Administrators have fine tuned access to the server
- Easy to setup for each client
- Learning curve is low

Centralized version control
(not Git)

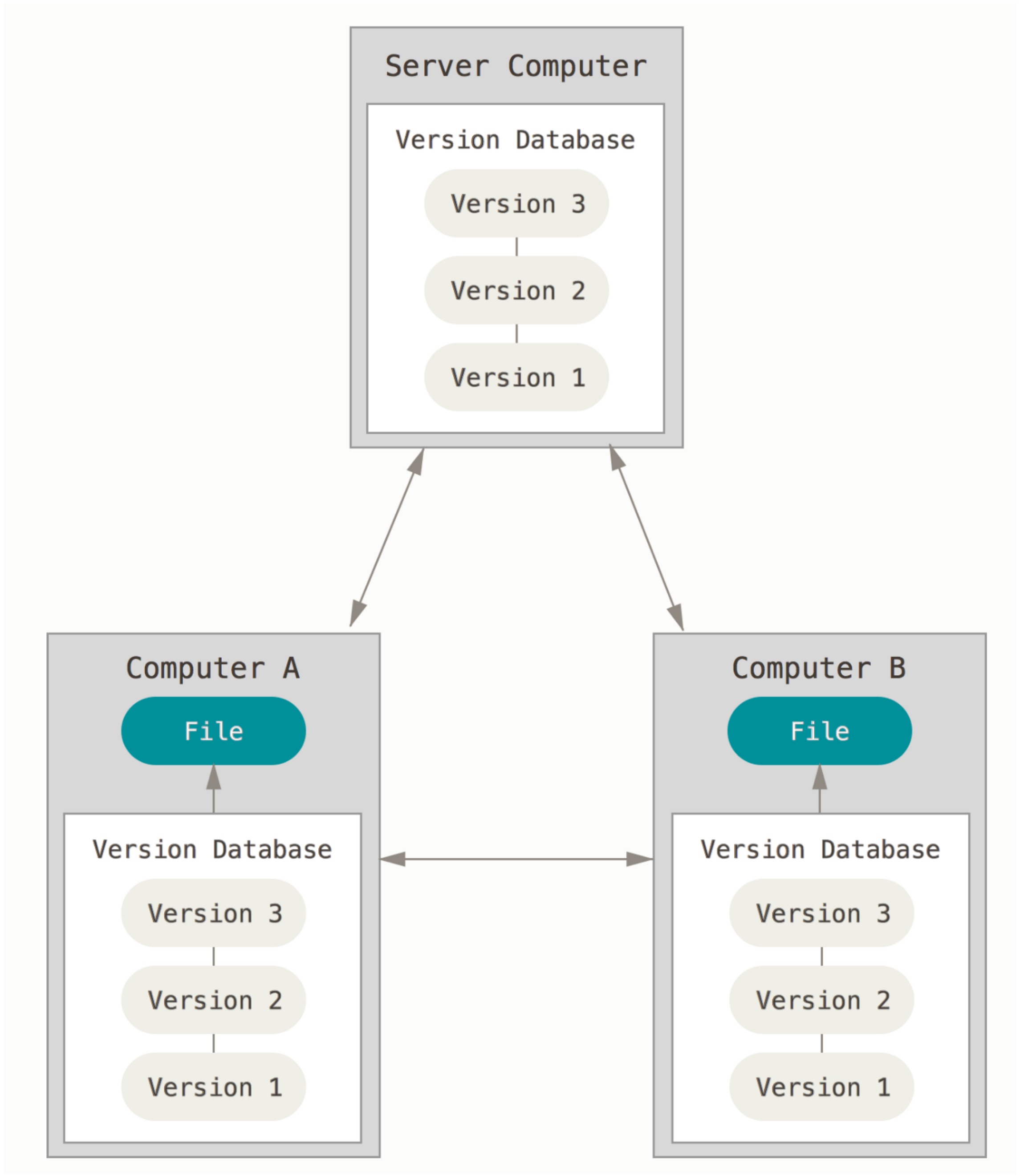
Disadvantages:

- If server goes down, no one can collaborate their work or push changes
    - If the server goes down, everyone pretty much can't work!
- If the server gets corrupted and there are no effective backups, you lose everything (code, history, all versioning)!
    - This also means you have to backup every so often; a nuisance
- Limited to a specific workflow

Distributed version control (Git)

- Instead of getting a ‘**snapshot**’ of the current state of the repository from the server, clients would **clone** the whole repository
- Every clone is practically a backup of the whole repository
- This allows a client to work locally rather than working off the server

Distributed version control (Git)



Distributed version control (Git)

Advantages:

- **Flexible**; no single workflow
- No reliance on the server
- Strong capabilities on developers concurrently working together
- Ability to share code with other developers before submitting them onto the main repository

Disadvantages:

- Steeper learning curve (it's worth learning!)
- Poor handling over binary files

So how does Git exactly work?

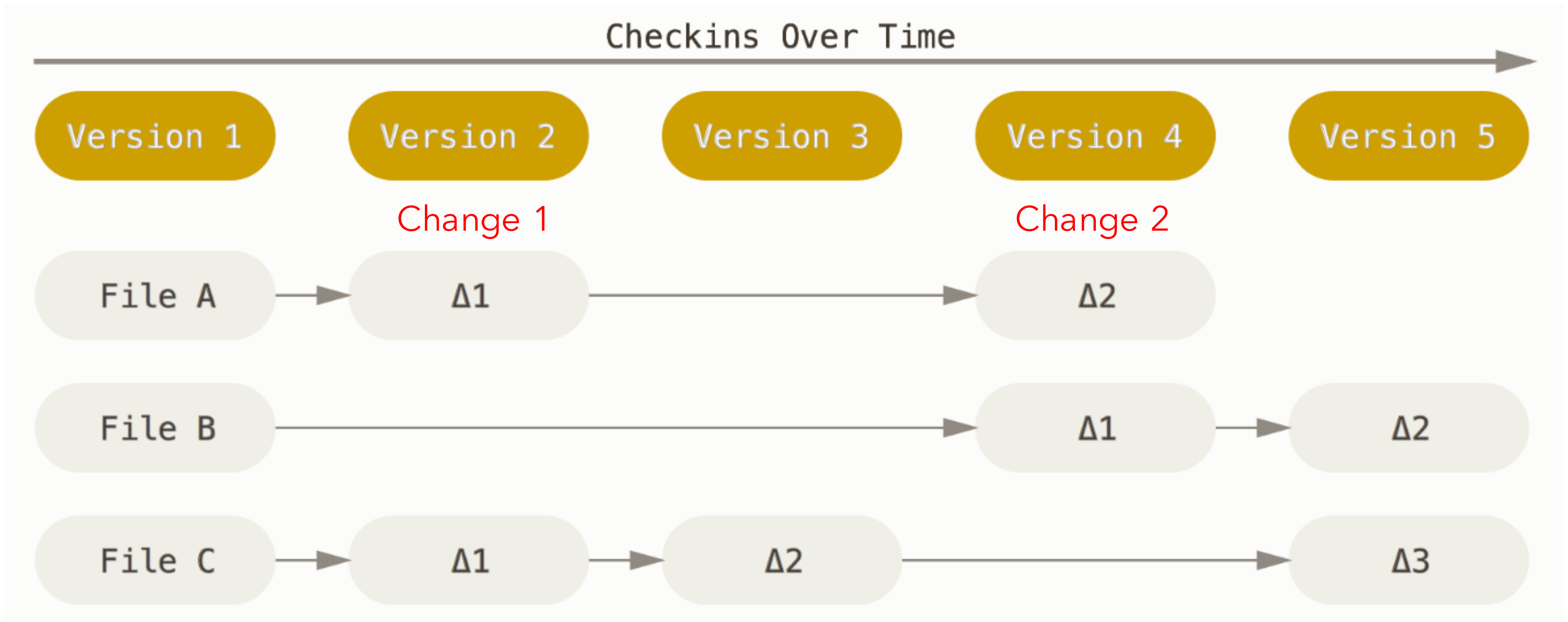
This part is important, so pay attention.

Git uses the concept of **‘snapshots’ and not differences** between files

The next two pictures will help exemplify this idea.

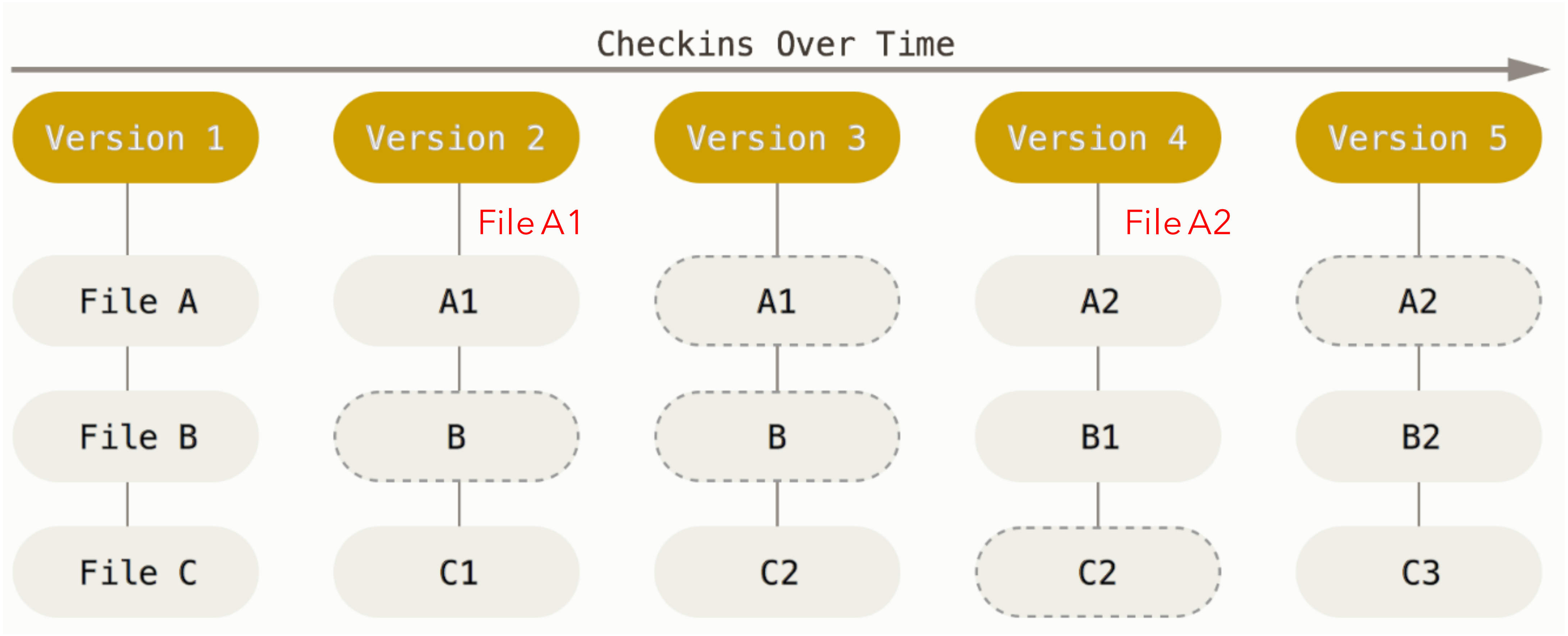
So how does Git exactly work?

This is an example on how a typical centralized version control system (not Git) would view data/files:



So how does Git exactly work?

And this is how Git views it:



So how does Git exactly work?

- Git **does not** view files as persistent changes from a base file
- Rather, it views different states of files as more check ins appear
- This is the concept of ‘**snapshotting**’
- It takes a **snapshot of your current state** of your program
- If a file has not changed, it will not store the file, rather it will link to a previous identical file that has been stored

Some other concepts about Git

Nearly every operation in Git is local

- This provides incredible speed versus server side operations
  - Remember, the working repository is cloned and not on the server
- Once enough changes are done, a developer can 'push' their changes onto the server; a command done once in a while
  - For example, you can do work on the bus or airplane and once you have Internet connection, you can apply your changes onto the server

Git will track your changes

- It is impossible to change files without Git recognizing it
- It uses a hash key to determine this (you can read more about this on the Git documentation)

Three states of Git (workflow)

Three main states that your files can reside in:

Committed

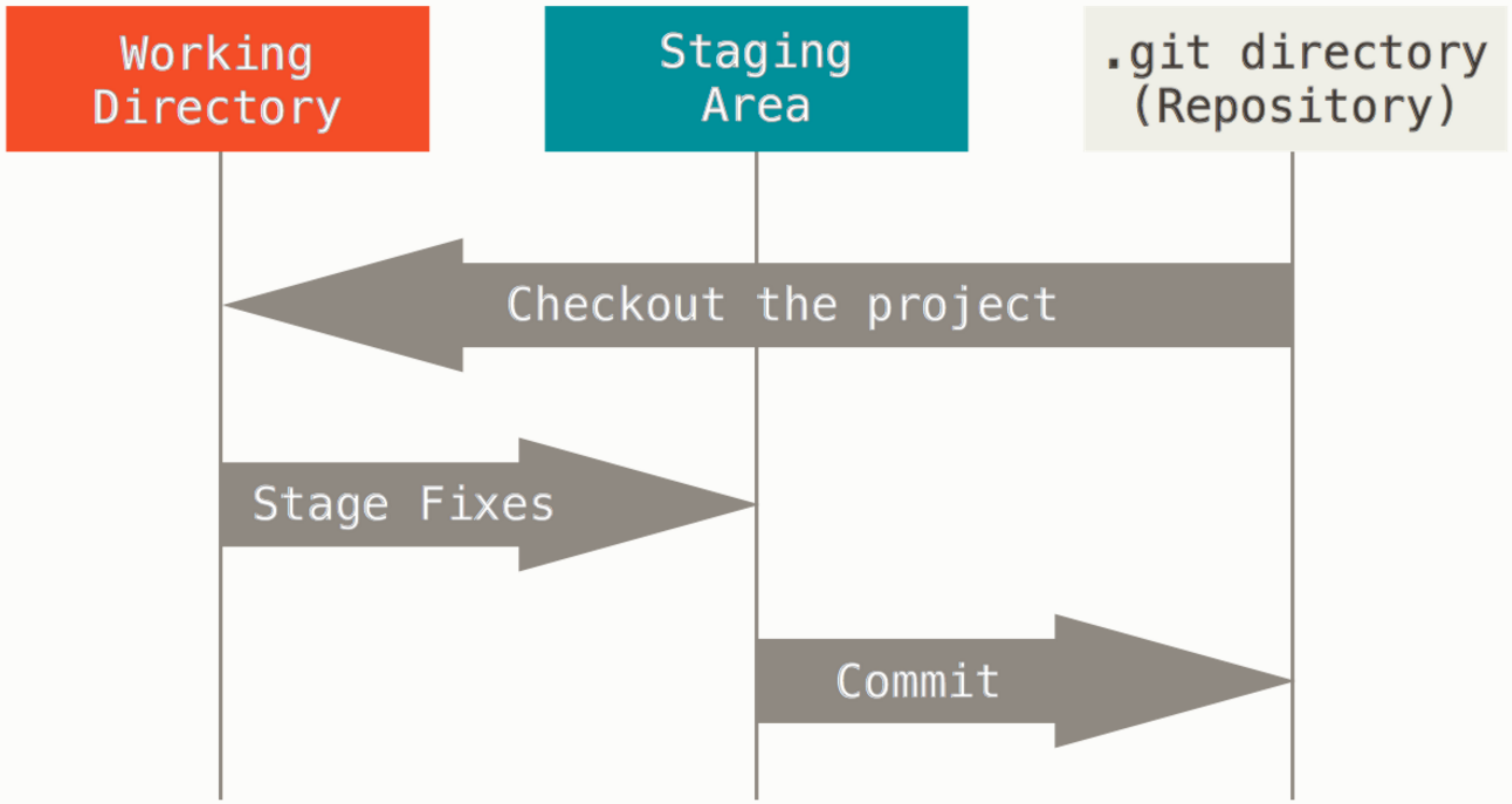
- Data is safely stored in your local database

Modified

- You have changed the file but have not committed it to your database yet

Staged

- You have marked a modified file in its current version to go into your next commit snapshot


Working Directory
Staging Area
.git directory (Repository)
Checkout the project
Stage Fixes
Commit

End of Git concepts
Now time for a Demo

I encourage you to follow along if you have your computers and Git installed.

If you don’t have Git installed on your computer yet, get some of your peers to help you out after class.

We will go through basic Git commands.

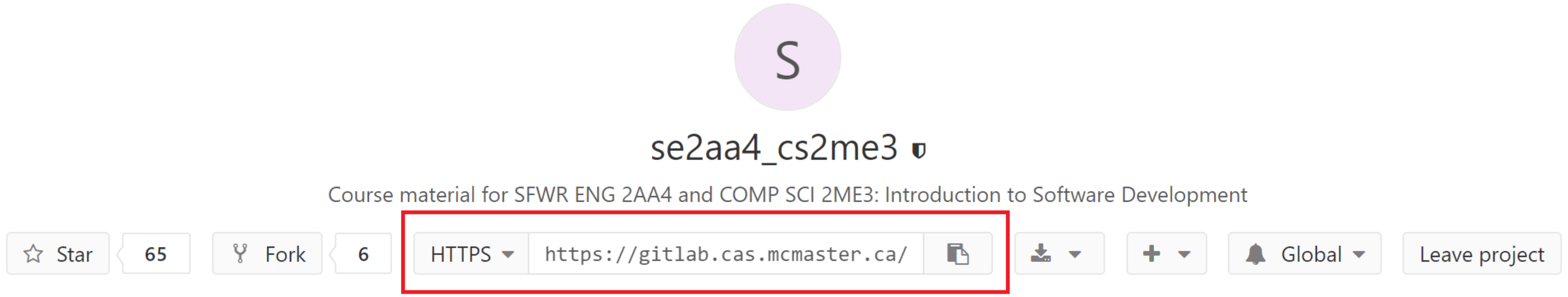
Initializing/Cloning a Git repository

Initializing a Git repository (just for demo purposes):

```
git init <your name for your repository goes here>
```

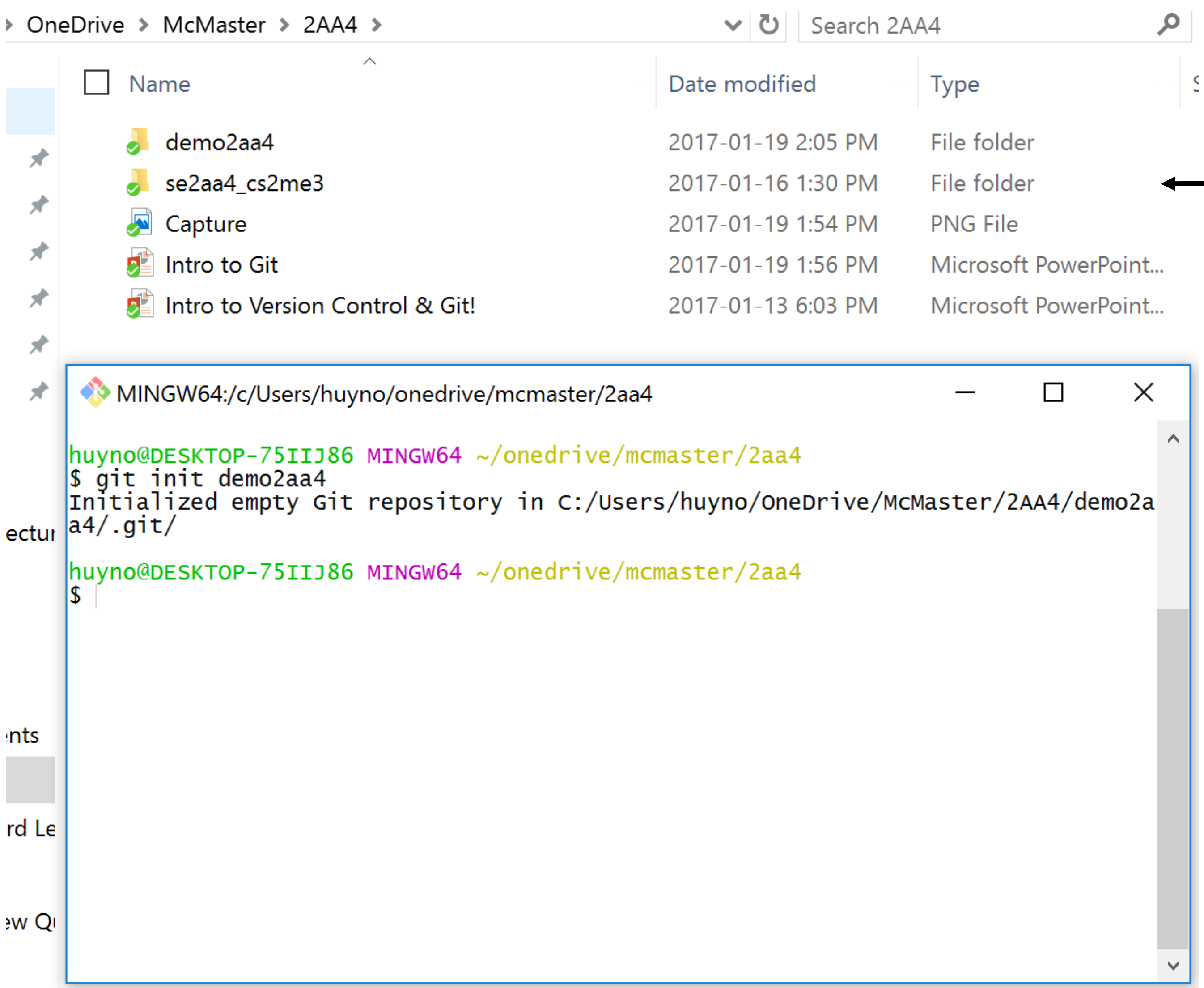
Cloning a Git repository (this is what you'll need for your repository):

```
git clone <your Git URL goes here>
```


S

se2aa4_cs2me3

Course material for SFWR ENG 2AA4 and COMP SCI 2ME3: Introduction to Software Development



Name	Date modified	Type
demo2aa4	2017-01-19 2:05 PM	File folder
se2aa4_cs2me3	2017-01-16 1:30 PM	File folder
Capture	2017-01-19 1:54 PM	PNG File
Intro to Git	2017-01-19 1:56 PM	Microsoft PowerPoint...
Intro to Version Control & Git!	2017-01-13 6:03 PM	Microsoft PowerPoint...

MINGW64:/c/Users/huyno/onedrive/mcmaster/2aa4

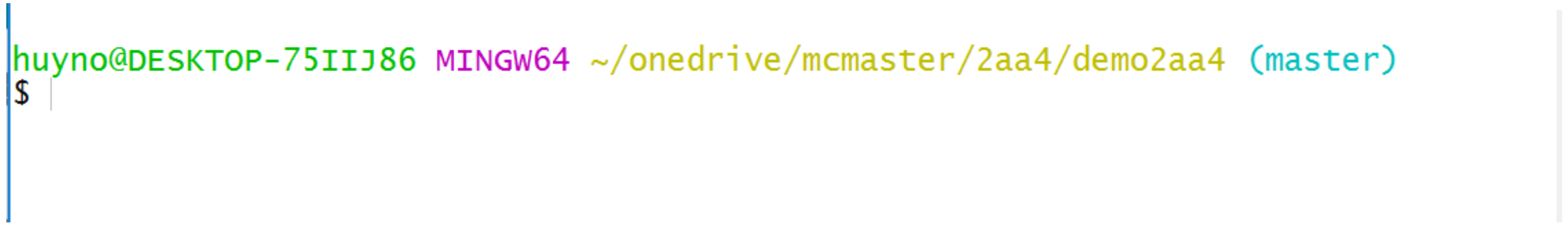
```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4
$ git init demo2aa4
Initialized empty Git repository in C:/Users/huyno/OneDrive/McMaster/2AA4/demo2a
a4/.git/

huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4
$
```

$\longleftarrow$ The repo has been created

First time in the repository

Now after you initialized your Git repository, let's try demoing how changes appear to Git.

cd into your Git repository that you've created.



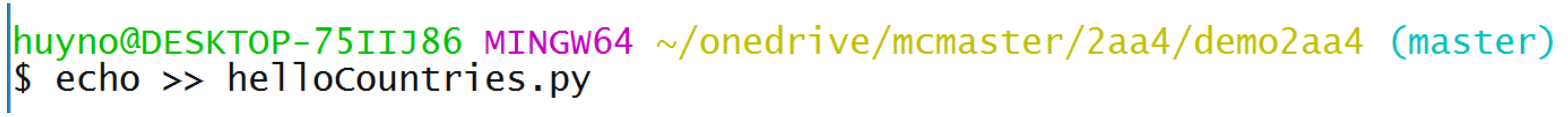
```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$
```

Notice how '**master**' appears, this is the main branch of the repository, knowledge of branches are not needed for the scope of this course.

Now after you initialized your Git repository, let's try demoing how changes appear to Git.

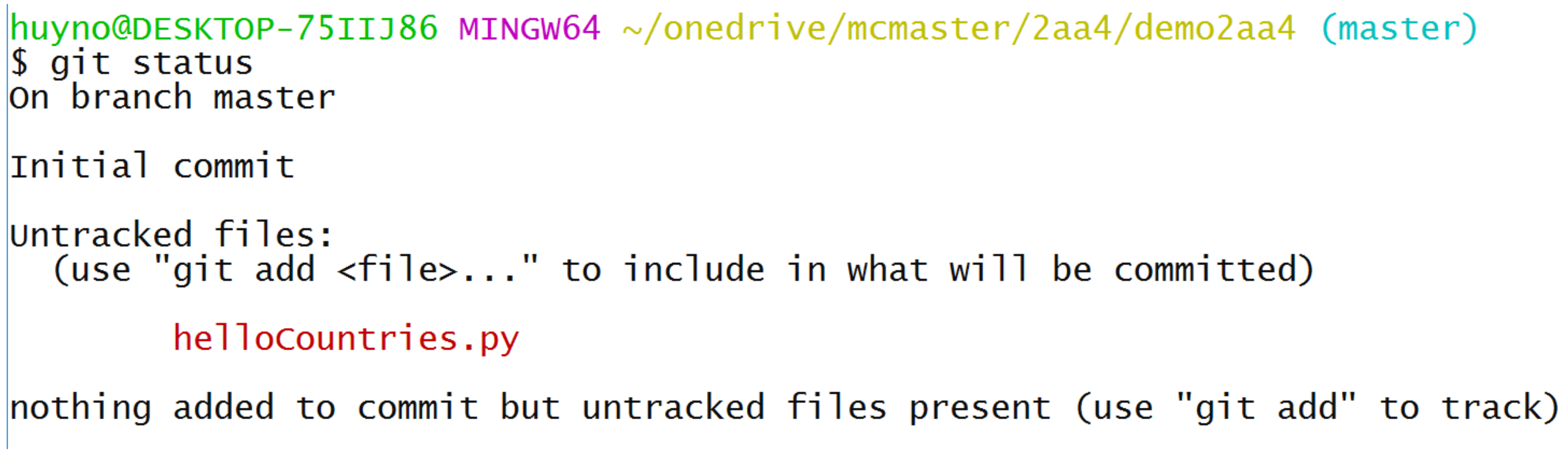
Tracking a file

Let's create a new file called "helloCountries.py"



```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$ echo >> helloCountries.py
```

Now let's do a 'git status' command



```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$ git status
On branch master

Initial commit

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

        helloCountries.py

nothing added to commit but untracked files present (use "git add" to track)
```

`git status`

- **`git status`** displays paths that have differences between the current status of your repository and the current HEAD commit
- HEAD commit refers to the latest commit in the current branch right now
- In this case, the branch is '**master**' and the commit is on its very first initial commit right now (no files)
- **Use this command often, it will help you determine what state your files are in**

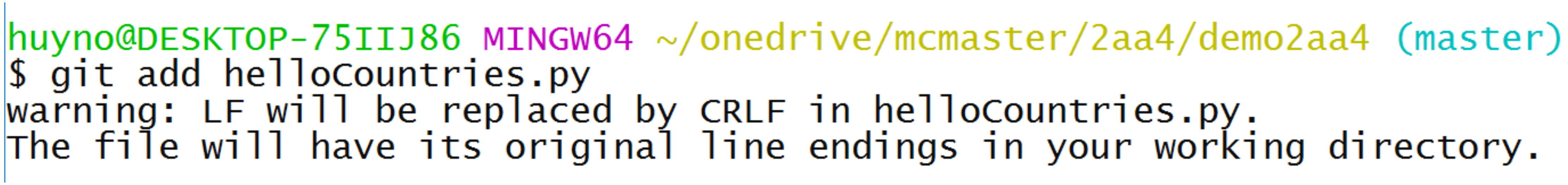
Making your very first commit!

Going back to the previous diagram, we first need to **stage** our files before we can commit them.

It captures a snapshot of the current state of the file. We will exemplify this later in an example later in this tutorial. **Just remember this.**

To stage your file, you will need to do

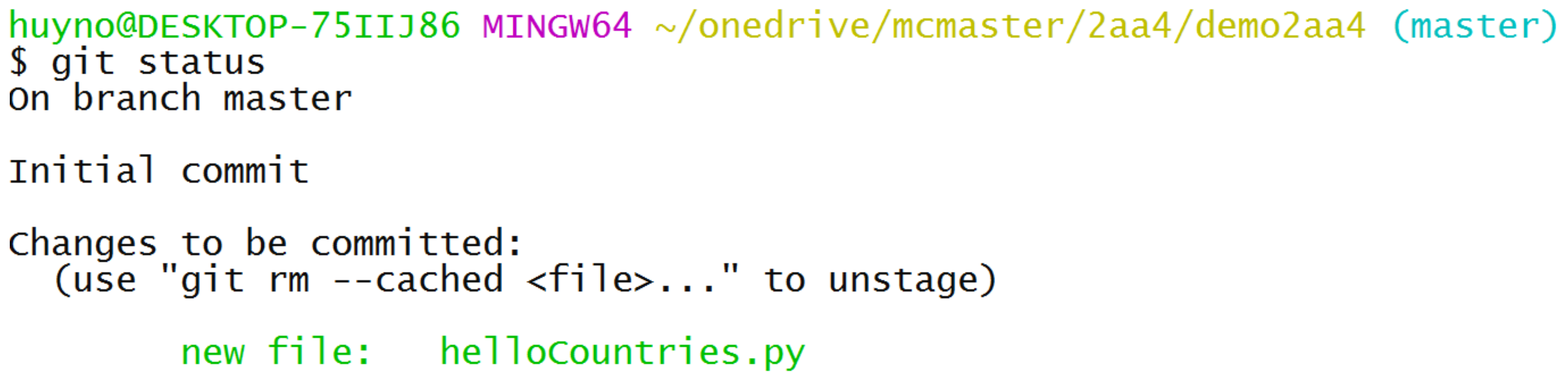
```
git add <your relative path to your file>
```



```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$ git add helloCountries.py
warning: LF will be replaced by CRLF in helloCountries.py.
The file will have its original line endings in your working directory.
```

You have staged your file!

Congratulations, you have staged your file! Now do a git status to check on the current state of your repository.



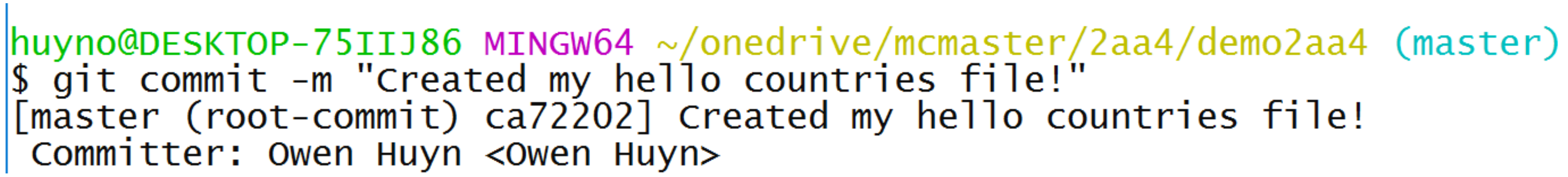
```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$ git status
On branch master

Initial commit

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

        new file:   helloCountries.py
```

It is now ready to be committed.

Committing your file

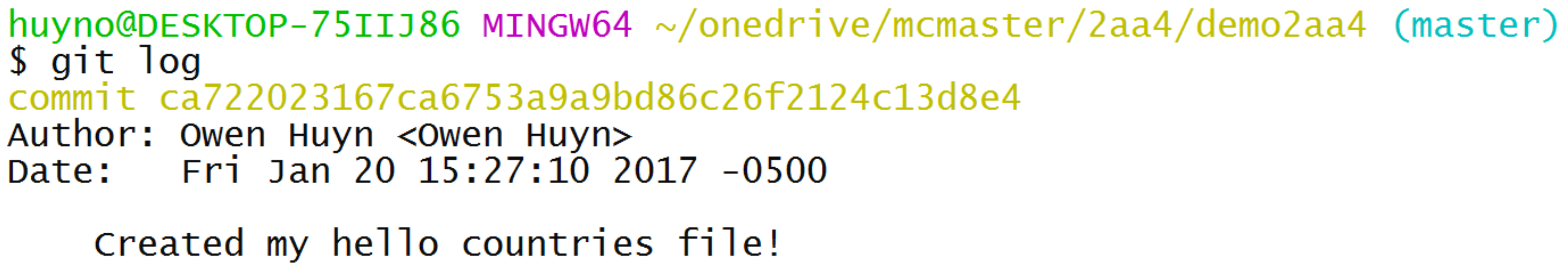
To commit your file type in: `git commit –m <“your message”>`

Make sure it is an appropriate message explaining your changes so other developers along with yourself can see what changes you’ve made.



```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$ git commit -m "Created my hello countries file!"
[master (root-commit) ca72202] Created my hello countries file!
 Committer: Owen Huyn <Owen Huyn>
```

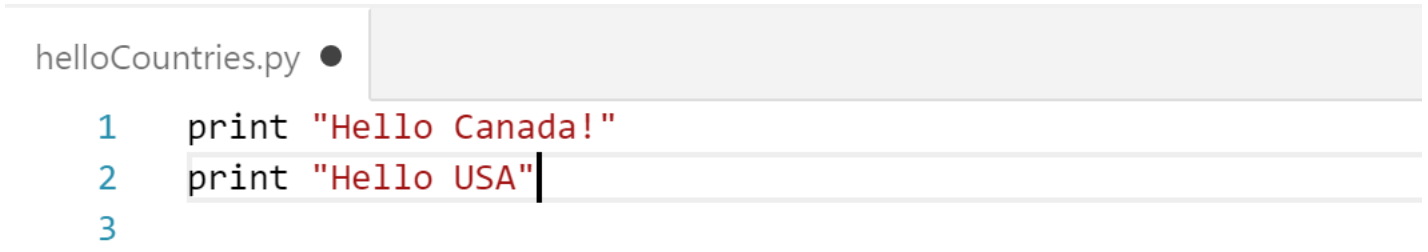
Now that you’ve committed your changes, you can check your log history.



```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$ git log
commit ca722023167ca6753a9a9bd86c26f2124c13d8e4
Author: Owen Huyn <Owen Huyn>
Date:   Fri Jan 20 15:27:10 2017 -0500

    Created my hello countries file!
```

Now, let's say we made changes

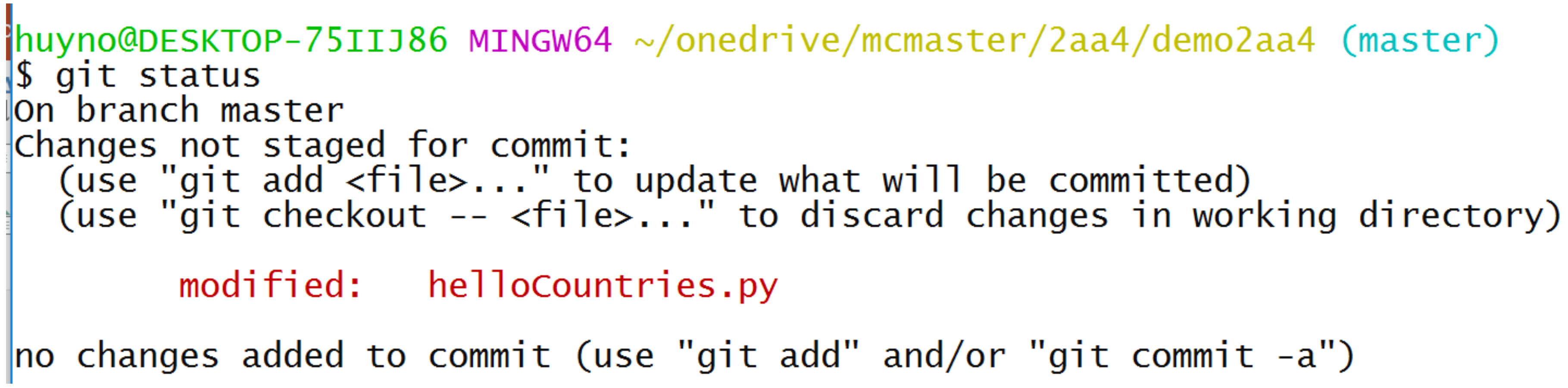
In the helloCountries.py, let's add some code and save it (you can do this in any text editor of your choice).



helloCountries.py ●

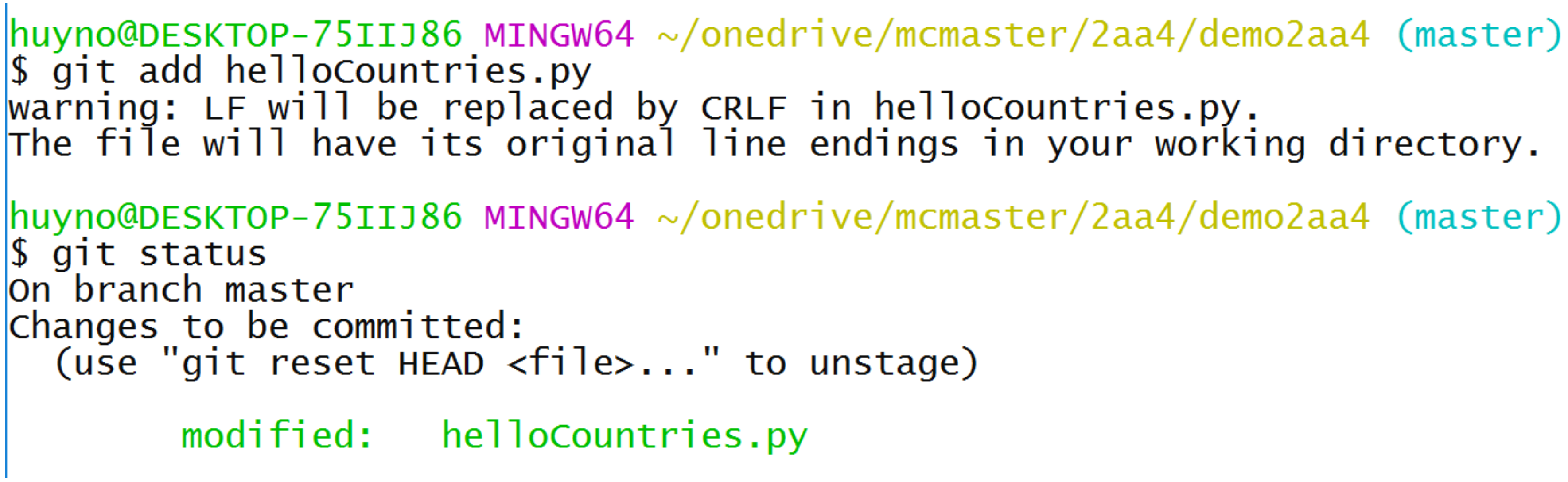
```
print "Hello Canada!"
print "Hello USA"
```

Now, let's do a git status.



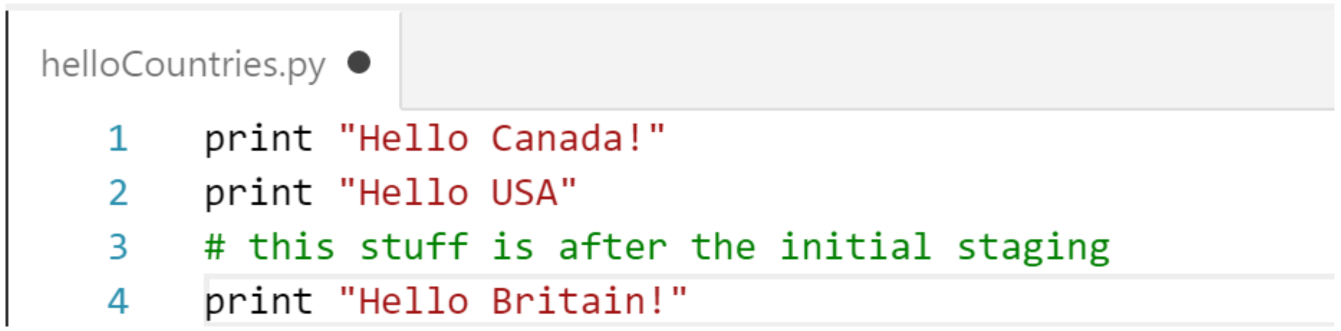
```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   helloCountries.py

no changes added to commit (use "git add" and/or "git commit -a")
```

Notice how Git recognizes that the file is modified? Let's do the same thing as before and stage our changes.



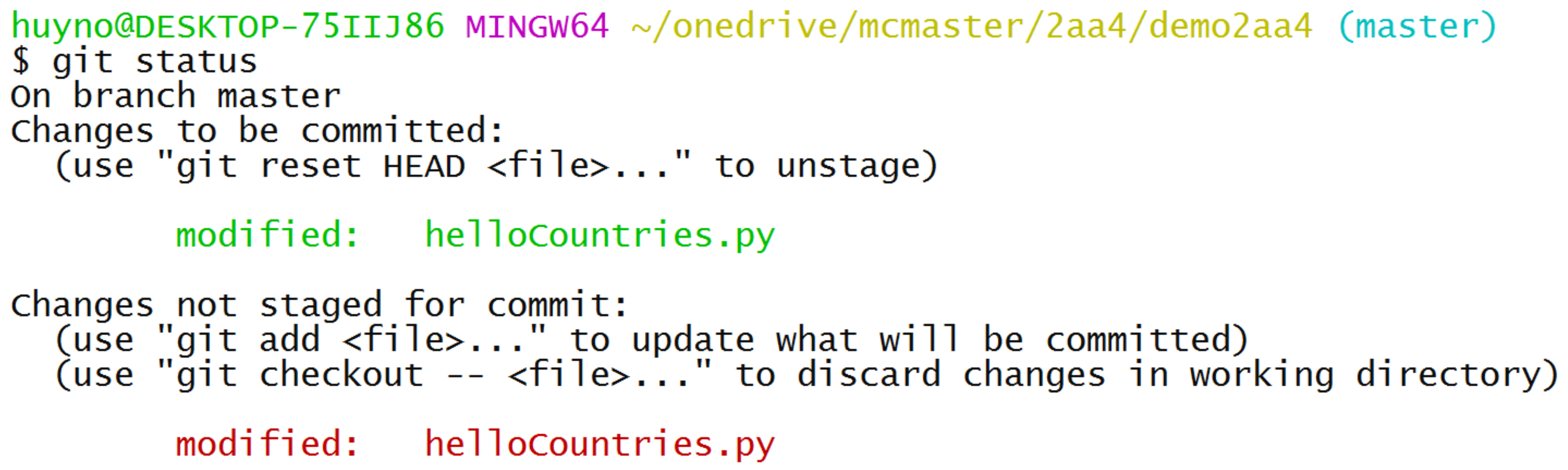
```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$ git add helloCountries.py
warning: LF will be replaced by CRLF in helloCountries.py.
The file will have its original line endings in your working directory.

huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   helloCountries.py
```

What if we want to modify a file that has been staged?



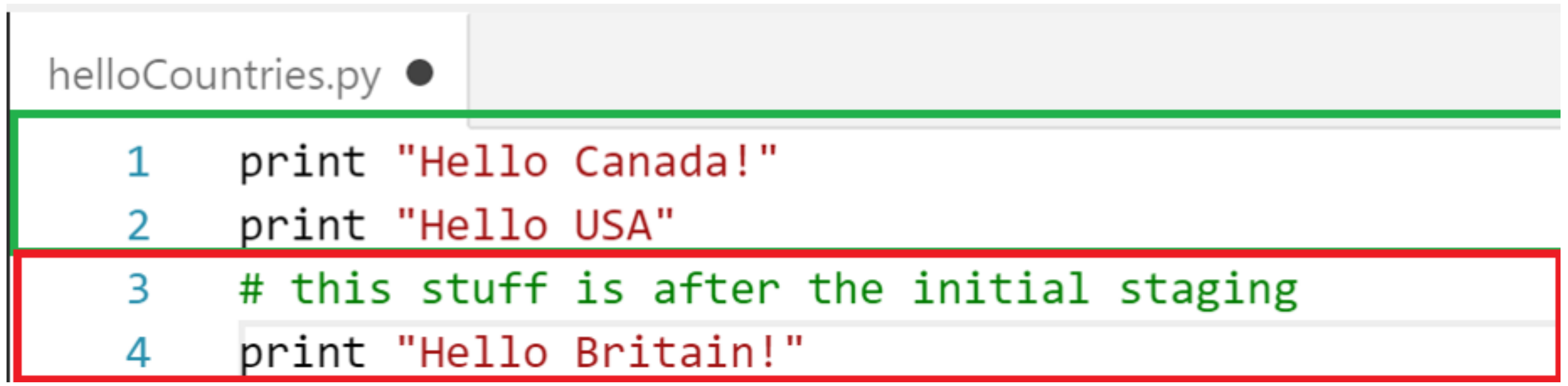
helloCountries.py ●

```
print "Hello Canada!"
print "Hello USA"
# this stuff is after the initial staging
print "Hello Britain!"
```



```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   helloCountries.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   helloCountries.py
```

Notice how it was staged and not staged? This goes back to what I said when Git snapshots the file. Here is a picture to exemplify this idea:

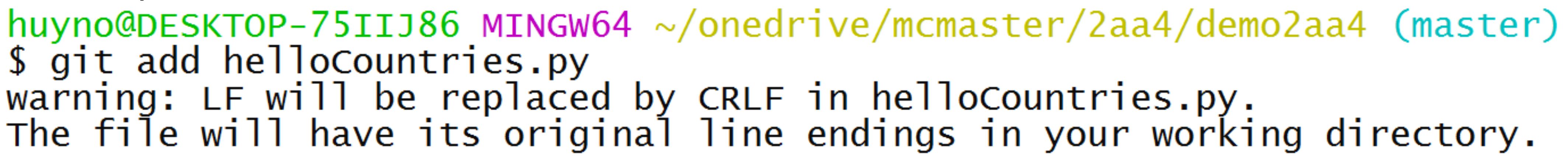


helloCountries.py ●

```
print "Hello Canada!"
print "Hello USA"
# this stuff is after the initial staging
print "Hello Britain!"
```

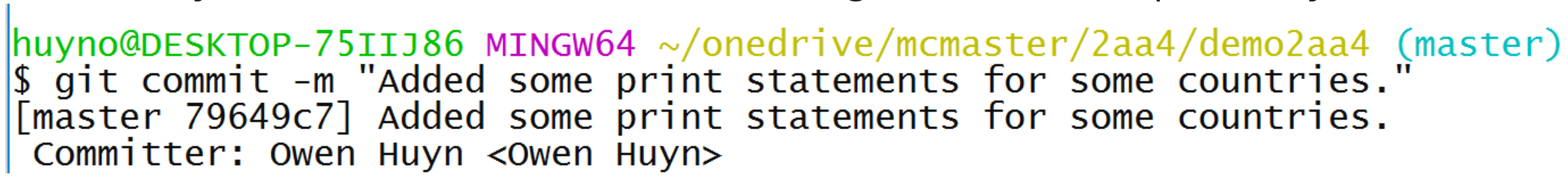
If you commit this now, you will only get the part that is staged (green)

OK, let's say we decided we want both parts. We just need to restage the new part as well.



```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$ git add helloCountries.py
warning: LF will be replaced by CRLF in helloCountries.py.
The file will have its original line endings in your working directory.
```

Similarly, let's commit these new changes into the repository



```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/demo2aa4 (master)
$ git commit -m "Added some print statements for some countries."
[master 79649c7] Added some print statements for some countries.
 Committer: Owen Huyn <Owen Huyn>
```

When you done enough commits for your assignments on your cloned repository, you can apply the command: `git push`

This will push your commits/files onto the server where any users tied to your repository will be able to see it.

(Note that this will not work for this demo repository since no online server is currently tied to this repository. It will work for cloned online repos.)

Removing files

To remove files from the repository, you can't just delete the file in your directory.

What you have to do is delete the file, stage the change and commit it.

OR

1. `git rm <your relative file path>`
2. Commit your change

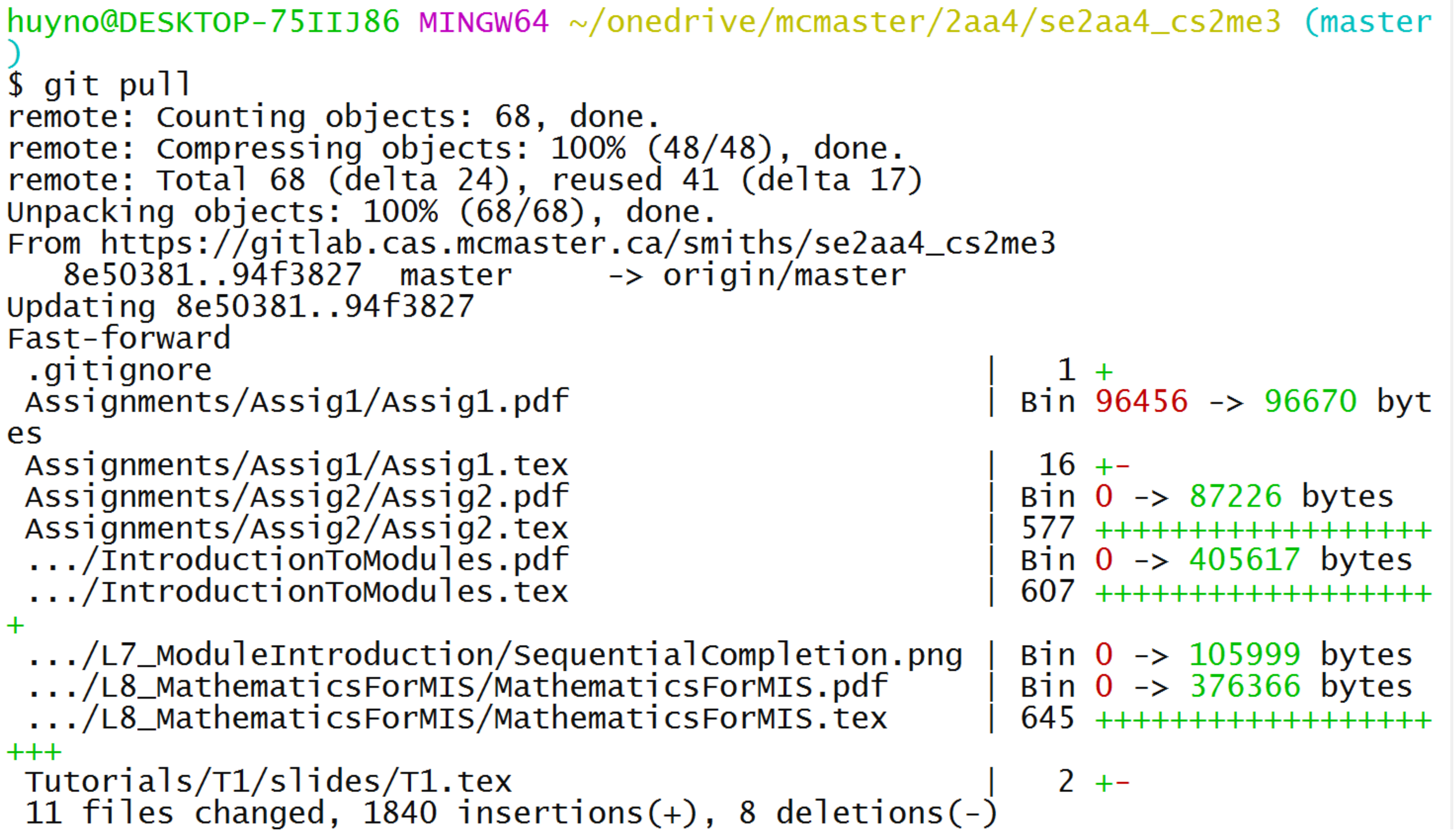
End of basics to Git commits
Now onto syncing

The course material is on GitLab, you should clone the course repository to access the files.

To sync with the repository, enter the following command:

```
git pull
```

This will sync the current branch that you are currently on with the one on the server.



```
huyno@DESKTOP-75IIJ86 MINGW64 ~/onedrive/mcmaster/2aa4/se2aa4_cs2me3 (master
)
$ git pull
remote: Counting objects: 68, done.
remote: Compressing objects: 100% (48/48), done.
remote: Total 68 (delta 24), reused 41 (delta 17)
Unpacking objects: 100% (68/68), done.
From https://gitlab.cas.mcmaster.ca/smiths/se2aa4_cs2me3
   8e50381..94f3827  master     -> origin/master
Updating 8e50381..94f3827
Fast-forward
 .gitignore                                          |    1 +
 Assignments/Assig1/Assig1.pdf                       | Bin 96456 -> 96670 byt
es
 Assignments/Assig1/Assig1.tex                       |   16 +-
 Assignments/Assig2/Assig2.pdf                       | Bin 0 -> 87226 bytes
 Assignments/Assig2/Assig2.tex                       | 577 ++++++++++++++++++
 .../IntroductionToModules.pdf                       | Bin 0 -> 405617 bytes
 .../IntroductionToModules.tex                       | 607 ++++++++++++++++++
+
 .../L7_ModuleIntroduction/SequentialCompletion.png | Bin 0 -> 105999 bytes
 .../L8_MathematicsForMIS/MathematicsForMIS.pdf     | Bin 0 -> 376366 bytes
 .../L8_MathematicsForMIS/MathematicsForMIS.tex     | 645 ++++++++++++++++++
+++
 Tutorials/T1/slides/T1.tex                          |    2 +-
 11 files changed, 1840 insertions(+), 8 deletions(-)
```

Dr. Smith made some changes to the course repo

Can’t I just use a GUI for all this?

You could (and honestly, it comes down to personal preference) but I’ll give you some points against it:

- The GUI may not necessarily have all the commands you need
- There is more flexibility with the command line
- There are many different GUIs but there is only one command line
- You gain a deeper understanding and appreciation of Git

On occasion, I do use a GUI (out of scope):

- Resolving merge conflicts
- Visualizing branches
- Going through a large commit history

Issue tracking

- Issues are used to keep track of tasks, enhancements and bugs on your codebase
- They provide a forum where your team can contribute discussion
- Issues can be assigned to a specific person and it can also be assigned to specific labels

Example in GitLab:
https://gitlab.cas.mcmaster.ca/smiths/se3xa3/tree/master/Labs/L03
Example of a popular issue tracking system on an open source project:
https://github.com/facebook/react/issues
Example of a bug issue in the same repository with discussion:
https://github.com/facebook/react/issues/6895

Great, you learned the basics to Git!

From here, you can do everything in the course related to Git.

Unfortunately, this is only a basic tutorial on the tool.

Now you will probably have a lot of struggles and questions about Git (trust me I've been there), but here are some resources you should use:

1. **Your peers (an invaluable resource)**
2. StackOverflow/Google
   - 99% of questions I had on Git was answered on the Internet
3. Your TAs and Avenue discussions

Great tutorials

Learn Git in 20 minutes:

https://www.youtube.com/watch?v=Y9XZQO1n_7c

The best tutorial website for Git:

https://www.atlassian.com/git

Official Git reference manual:

https://git-scm.com/doc

Advanced Git (the neat stuff)

If you would like to stay, we can go into the main features of Git that make it a special version control tool (not necessary for this course):

1. Branches and forking
2. Merging
3. Pull requests and code reviews

Uncommon features of Git: rebasing