Using Git
CS 2ME3/SE 2AA4

Owen Huyn
Steven Palmer

Department of Computing and Software
McMaster University

January 8 - 12

Outline

1. What is Version Control?
2. Prerequisite Software
3. Using Git
   - Git Workflow
   - Cloning a Repository
   - A Small Example
   - Modifying Files
   - Pushing Changes
   - Syncing Local with Remote
   - `.gitignore`
   - Issue Tracking
4. Additional Resources

What is Version Control?

- Tracks and provides control over changes to files (source code, documents, etc.)
- Used to keep a history of code (versions) over a period of time
  - Similar to system restore under Windows
  - Analogy: saving your progress in a video game
  - If a bug is introduced, easy to narrow down to a specific version and roll back
- Allows developers to work simultaneously (you won't see this in the course)

Motivation for Learning to Use Git

- It is a popular version control tool
  - Most software development companies use a version control tool
  - Once you have learned one, switching between other version control tools is easy (same ideas with different syntax)
- Widely used in the open-source community (GitHub, GitLab)
- Many large scale development projects use git

Installing Git

Linux (Ubuntu):

Run the command:

```
sudo apt-get install git
```

OSX:

Two options:

1. If you have Homebrew installed, then run:

   ```
   brew install git
   ```

2. Else you can use the installer

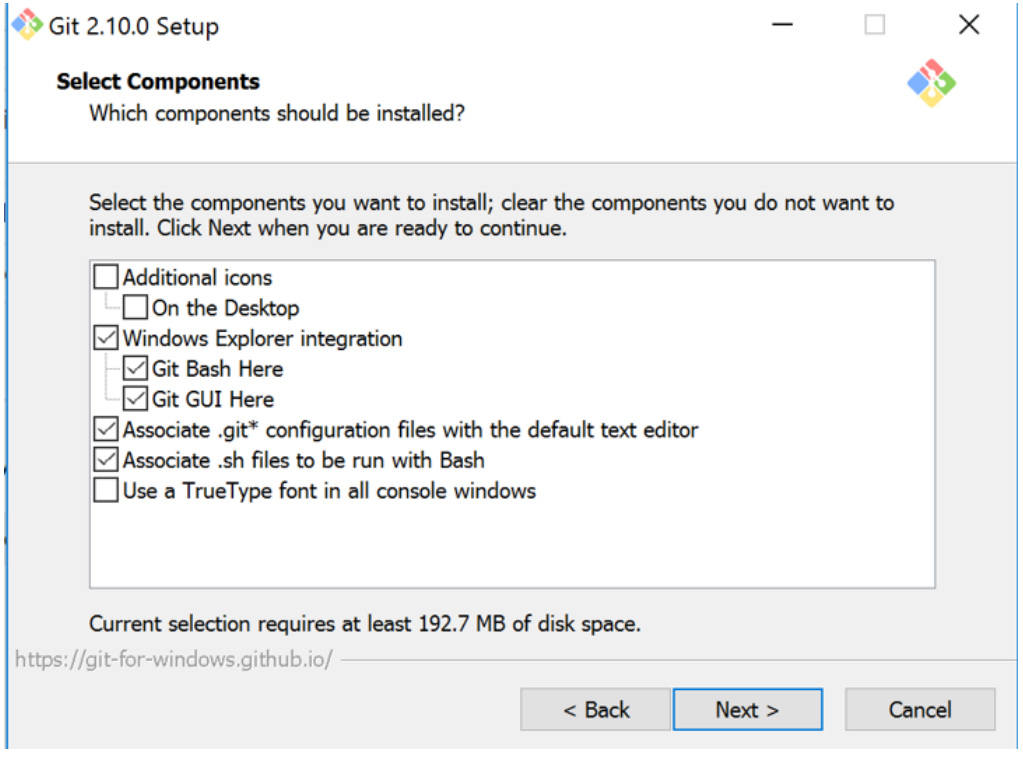
Installing Git

Windows:

1. Download the installer
2. Run the installer when download is complete
3. Follow steps on the next slides for selecting options

Installing Git

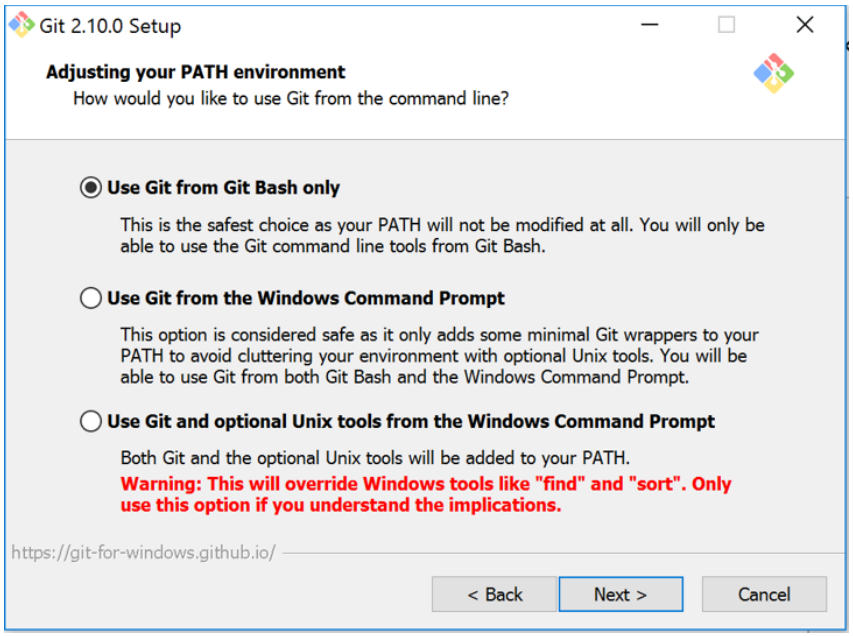
Windows:



- Make sure these settings are on
- Association of .sh files is optional

Installing Git

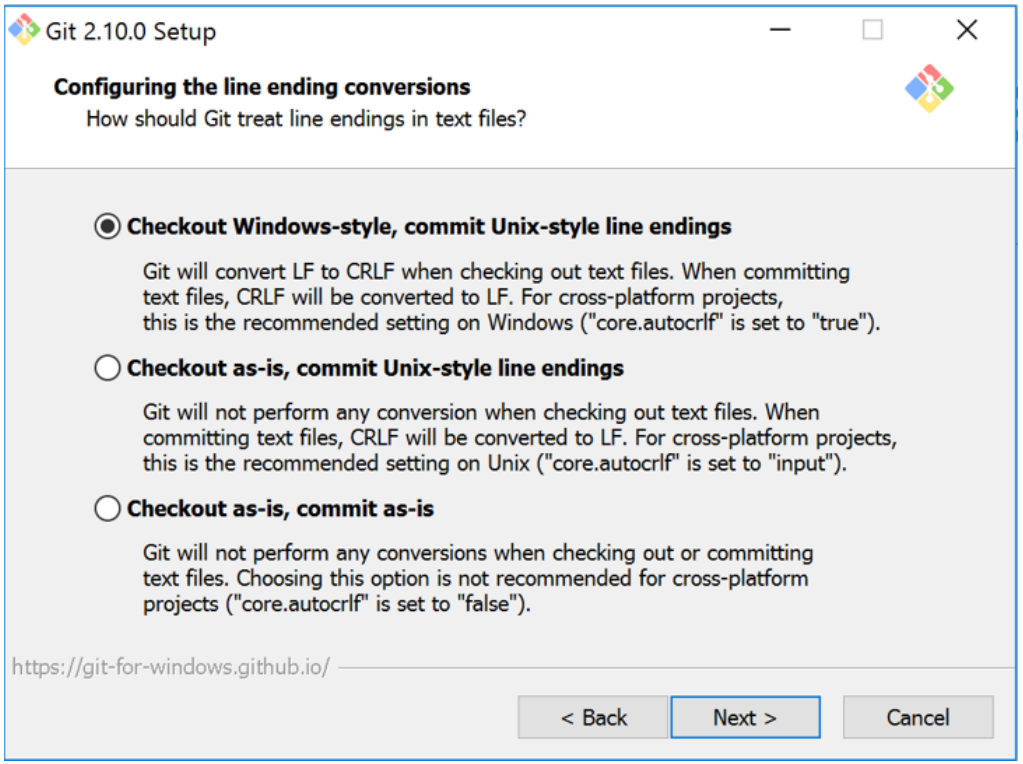
Windows:



- Git Bash is a command line interface in Windows that knows Bash commands
- If you want to use Git Bash exclusively for git, select the first option
- The second option will allow you to run git commands from the command prompt as well as Git Bash
- The third option is **not** recommended

Installing Git

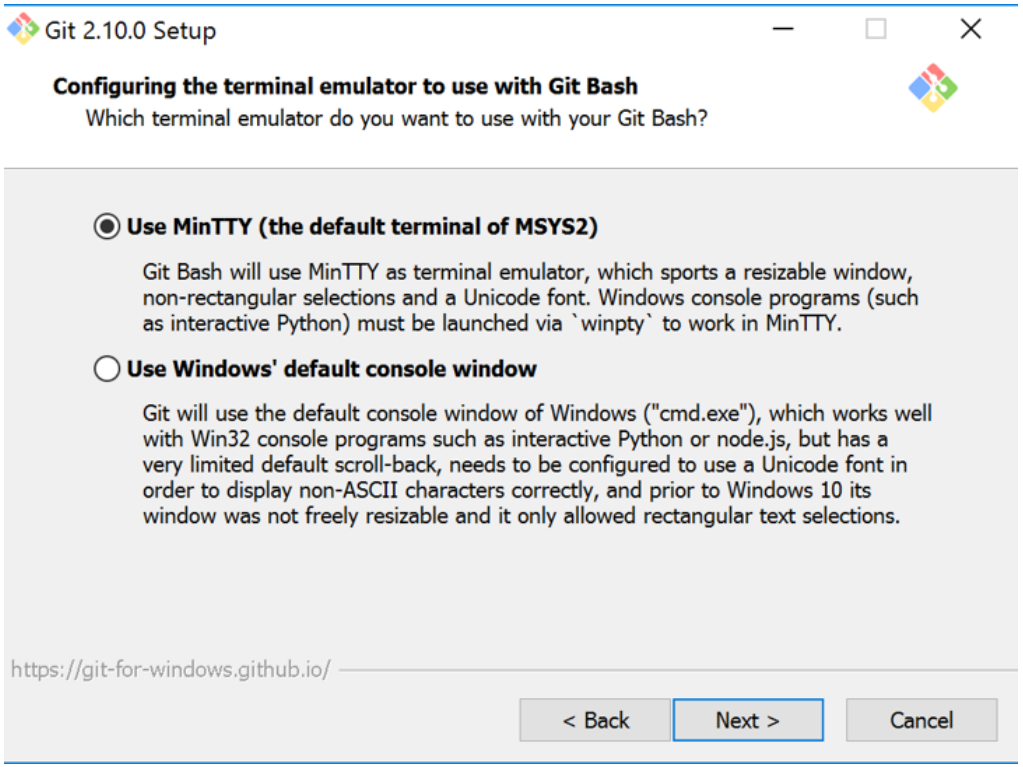
Windows:



- Select the first option here
- The other options will result in conflicts when other people are making changes to your repo using different operating systems

Installing Git

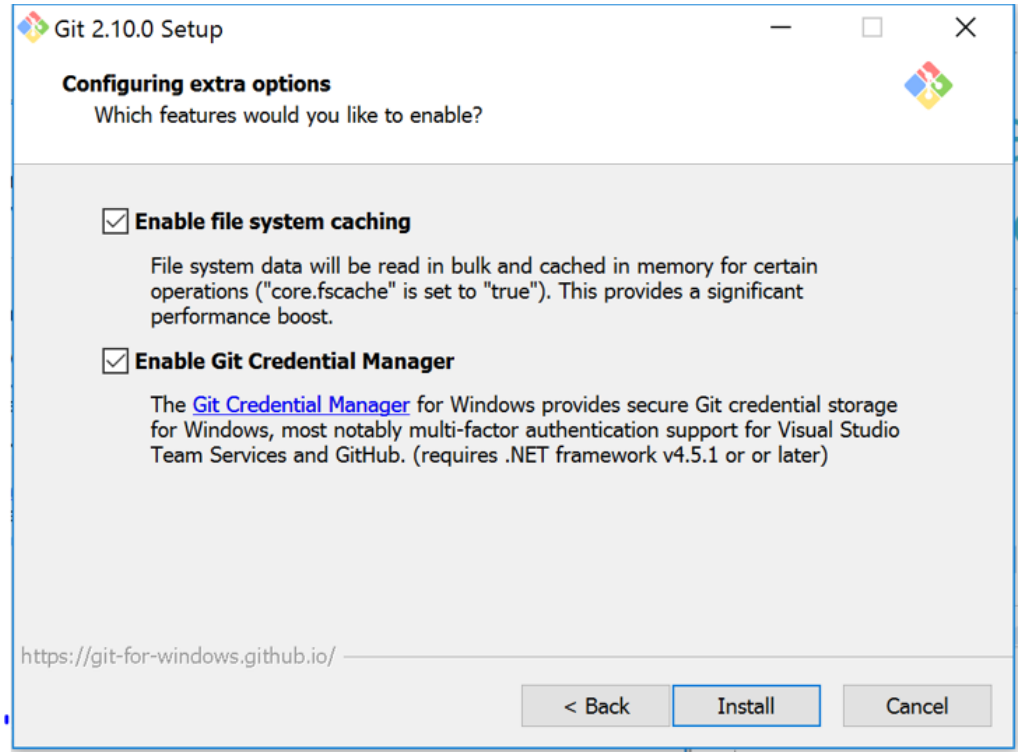
Windows:



- Select the first option here

Installing Git

Windows:



- Check both options

Simplified Overview of Git

- The repository for your project is stored on a remote server
  - In this course, we will use the CAS GitLab as the remote server
  - You should be able to create an account by signing in with your MACID and password
  - If your password isn't working, there is a link to reset your CAS password to your MACID password on the left
- You will use a local version of the repository to work in and make changes
- As changes are made, you can **stage** them and create a **commit** that contains the staged changes
- Commits are then **pushed** to the remote server

Creating a Local Version of a Git Repository

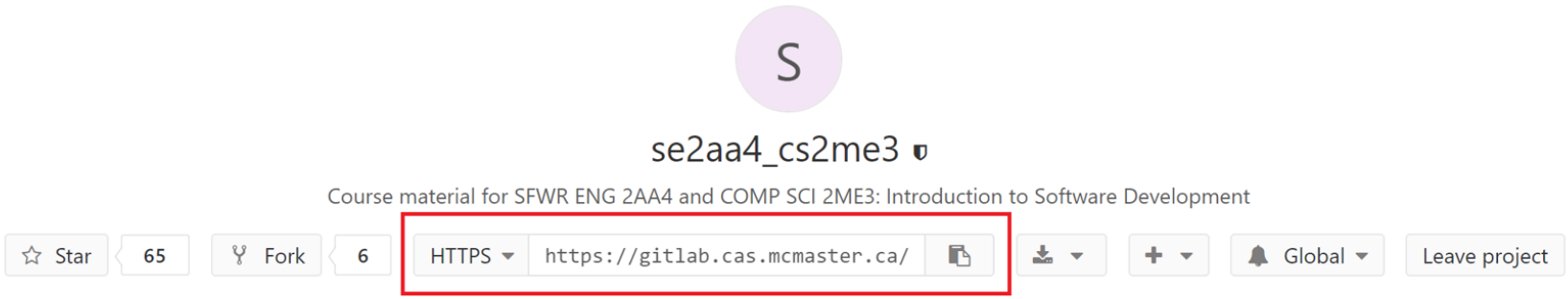
- You can create a local version of a remote repository to work in by **cloning** it
- This is done using the command:

```
git clone <link to repo>
```

- This will create a new folder in whatever directory you were in on your terminal/command prompt
- The new folder will have the same name as your project and contain copies of all of the files on the remote repository

Creating a Local Version of a Git Repository

- `git clone <link to repo>`
- To get the `<link to repo>` using GitLab, go to the main page of the GitLab project you want to clone and copy/paste the following:



Creating a Local Version of a Git Repository

Exercise 1: Clone the course repository

The course repository is located at:

`https://gitlab.cas.mcmaster.ca/smiths/se2aa4_cs2me3`

Find the git link to the repository and clone it.

A Small Example

Exercise 2: A small example

We will use a small example to learn some git commands. We will start by creating a new GitLab project:

1. Log in to GitLab in your web browser.
2. Create a new project using the green "New Project" button at the top right of the page.
3. Give your project a name and click "Create project" (you can leave all the other options as their defaults).
4. You are now on the main page for your new empty project.
5. Clone your empty repository using the repository link near the top of the page.
6. Follow the instructions on the slides that follow.

Tracking a File

- Create a new file in your empty repository called "helloCountries.py"
- Now run the `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 state of the repo and the last commit
- **Use this command often: it will help you determine what state your files are in.**

Making a Commit

- Commits are essentially different versions of your repository.
- Making a new commit means bringing your repository to a new version, with changes made on top of the last commit.
- To make a new commit, you first need to **stage** your changed files.
- To stage a file, use the following command:

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

- To stage all modified files, use:

```
git add *
```

Making a Commit

- You can stage your new file "helloCountries.py" using the command:

  `git add helloCountries.py`

- Running the `git status` will show you the new state of your repository after staging:

```
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
```

Making a Commit

- Now that we have staged the file, we can create a commit using the command:

  `git commit -m “commit message”`

- Make sure to use a meaningful message that explains your changes so you and other developers can easily tell 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>
```

Modifying Files

- Now let's make some changes to our file. Add the following lines to "helloCountries.py" and save it:

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

- Now run the `git status` command and notice that git is aware that the file has been modified:

```
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")
```

Modifying Files

- Now 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
```

Modifying Files

- What if we want to modify a file that has been staged?
- Add the following line to the end of "helloCountries.py" and save it:

  `print "Hello Britain!"`
- Now run the `git status` command:

```
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
```

Modifying Files

- Notice that "helloCountries.py" is both staged and unstaged
- This is not a problem:
  - Our original 2 lines are currently staged.
  - The additional line `print "Hello Britain!"` is not staged.
  - If we commit now, only the first 2 lines that we initially staged will be part of the commit.

Modifying Files

- Let's assume that we want both parts to be included in our next commit.
- All we have to do is stage the file again:

```
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.
```

- Now let's commit the changes:

```
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>
```

Pushing Changes to Remote

- When you've made a commit(s) locally, only you can see them
- For others to see your changes, you must **push** them to the remote repository on GitLab using the following command:

```
git push
```

- **If you don't push your changes, the TAs will not be able to see your changes. It is very important to remember to push your changes if you want to receive a grade for your work!**

Pushing Changes to Remote

Exercise 3: Push your changes

Try pushing the changes you've made to your new repository using `git push`. After you push, you will be able to see the updates on GitLab in your web browser.

Syncing Your Local Repository with the Remote

- When other people push changes to a remote repository on GitLab, these changes will not automatically show up in your local version
- In order to get these changes, you must sync your local version with the remote using the following command:

```
git pull
```

- You should frequently execute the `git pull` command on the course repo so that you have up-to-date lecture slides, assignment instructions, etc. (Dr. Smith makes frequent changes to the repo!)

Syncing Your Local Repository with the Remote

```
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(-)
```

Using .gitignore

- When we are keeping a code base under version control, it is often the case that we end up with generated files that should not be stored in the repository
- Compilation in particular generates many files, for example:
  - C/C++ .o files
  - Python .pyc files
  - LaTeX compilation files (.aux, .log, etc.)
  - executables
- By adding a file called `.gitignore` to your repository, you can tell `git` about file patterns that should be ignored
- Take a look at the `.gitignore` file in the course repository as an example (depending on your OS, you might need to unhide this file)

Issue Tracking

- GitLab provides an issue tracker for each individual project
  - Note that this is a feature of GitLab and not part of `git` itself
  - Access via web browser on GitLab project page
- Issues are used to keep track of tasks, enhancements and bugs
- They provide a forum where your team can contribute discussion
- Issues can be assigned to a specific person and categorized with specific labels
- You might find it useful to use the issue tracker to assign issues to yourself as you are working on assignments in this course

Issue Tracking

- Some examples of using the issue tracker:
  1. Example in GitLab
  2. Example of a popular issue tracking system on an open source project
  3. Example of a bug issue in the same repository as (2) with discussion

Additional Resources

- The basics covered in this presentation are sufficient for the needs of this course
- Should you run into problems or have any questions, here are some additional resources:
  1. **Your peers (an invaluable resource)**
  2. StackOverflow
  3. Your TAs and Avenue discussions
  4. YouTube tutorials, such as this one
  5. Atlassian git tutorials
  6. The git reference manual