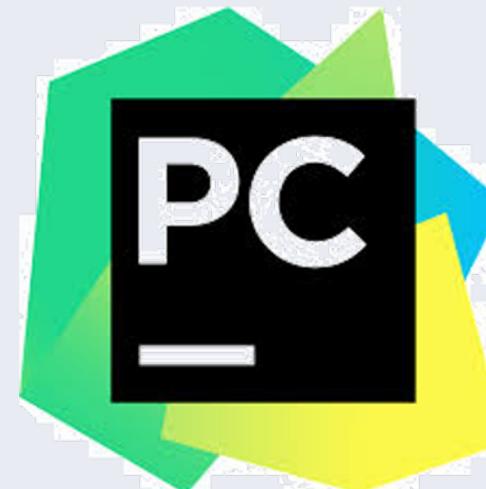


CPSC 250

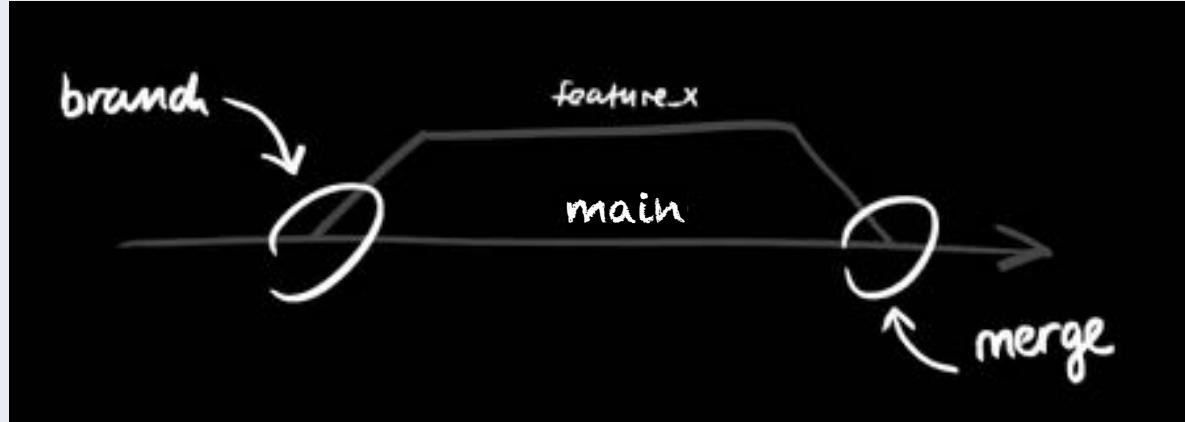
Programming for Data Manipulation (CS2)

Course Tool Chain and Workflow



Why bother?

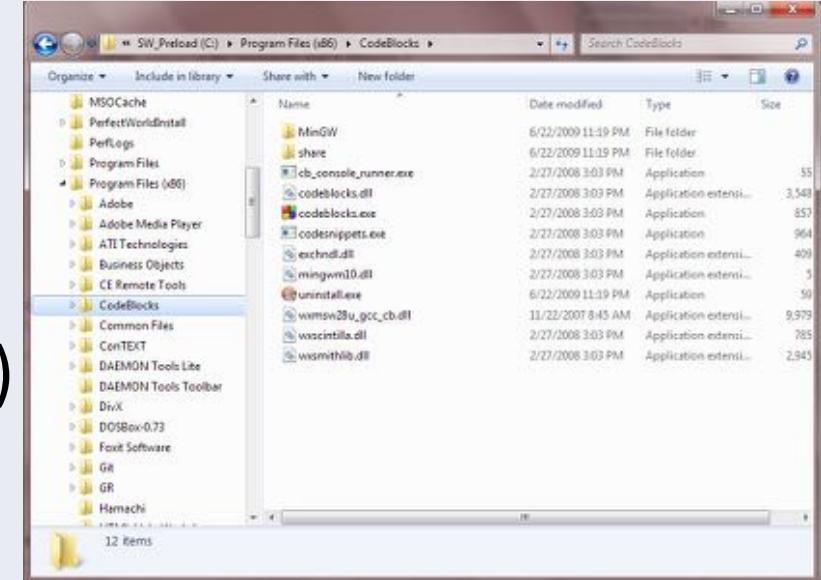
- Save hours of frustration
 - Easy backup and restore
 - Easy collaboration with others
 - Easy to experiment in a separate “branch”
 - Minimal loss of data if your computer crashes
 - Recover lost files or undo mistakes
 - Traceability (when did I introduce that bug?)
- Good professional practice
 - Git is a widely used “tool” in industry and research
 - Source Code Management is what, Git is how!
 - Atlassian professional tools, Online (GitHub, Bitbucket, GitLab)
 - Google recommends learning this your first semester
 - Put this on your resume
- Git Community
 - Open source code for variety of interests (github, bitbucket, gitlab)



Required for
this course!

Prerequisites

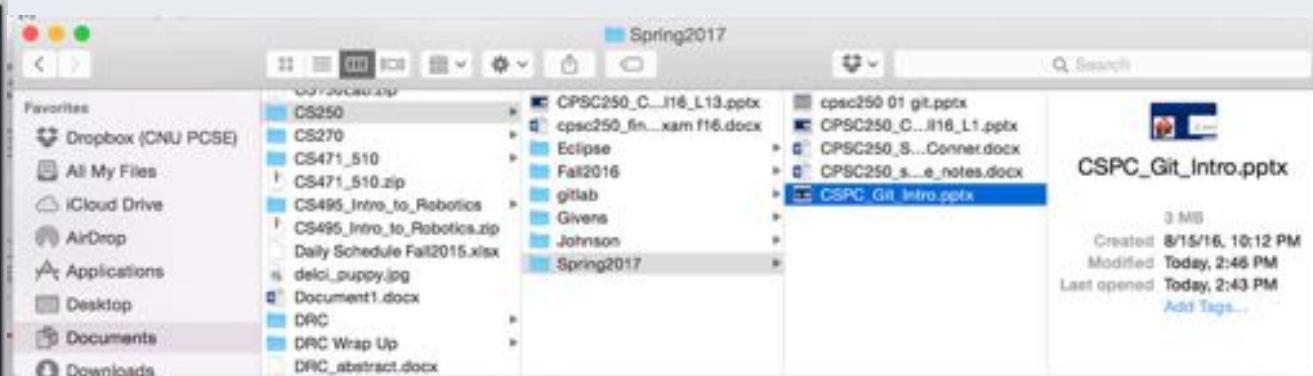
- Understand your file structure!
- ***Use the command line*** (won't always have GUI)
- Understand how to navigate via command line
 - command, shell, terminal
 - cd, pwd, ls or dir
 - \ (backslash on Windows) vs. / (forward slash Linux, Mac OS X)



<https://gitlab.pcs.cnu.edu/cpsc-labs/bash-and-git-wiki/wikis/home>

```
Spring2017 – bash – 80x18
Davids-MacBook-Pro:Documents dconner$ cd CS250/
Davids-MacBook-Pro:CS250 dconner$ ls
CPS250_Conner_Fall16_L13.pptx Johnson
Eclipse * command, shell, ter Spring2017
Fall2016 * cd, ls or dir gitlab
Givens * cpsc250_final_exam f16.docx
Davids-MacBook-Pro:CS250 dconner$ cd Spring2017/ ward slash Linus, Mac OS X)
Davids-MacBook-Pro:Spring2017 dconner$ ls
CPSC250_Conner_Fall16_L1.pptx cpsc250_01_git.pptx
CPSC250_Spring17_Conner.docx ~$SC250_Spring17_Conner.docx
CPSC250_spring2017_lecture_notes.docx ~$cpsc250_01_git.pptx
CSPC_Git_Intro.pptx
Davids-MacBook-Pro:Spring2017 dconner$
Davids-MacBook-Pro:Spring2017 dconner$ pwd
/Users/dconner/Documents/CS250/Spring2017
Davids-MacBook-Pro:Spring2017 dconner$ dir
-bash: dir: command not found
Davids-MacBook-Pro:Spring2017 dconner$
```

Windows command

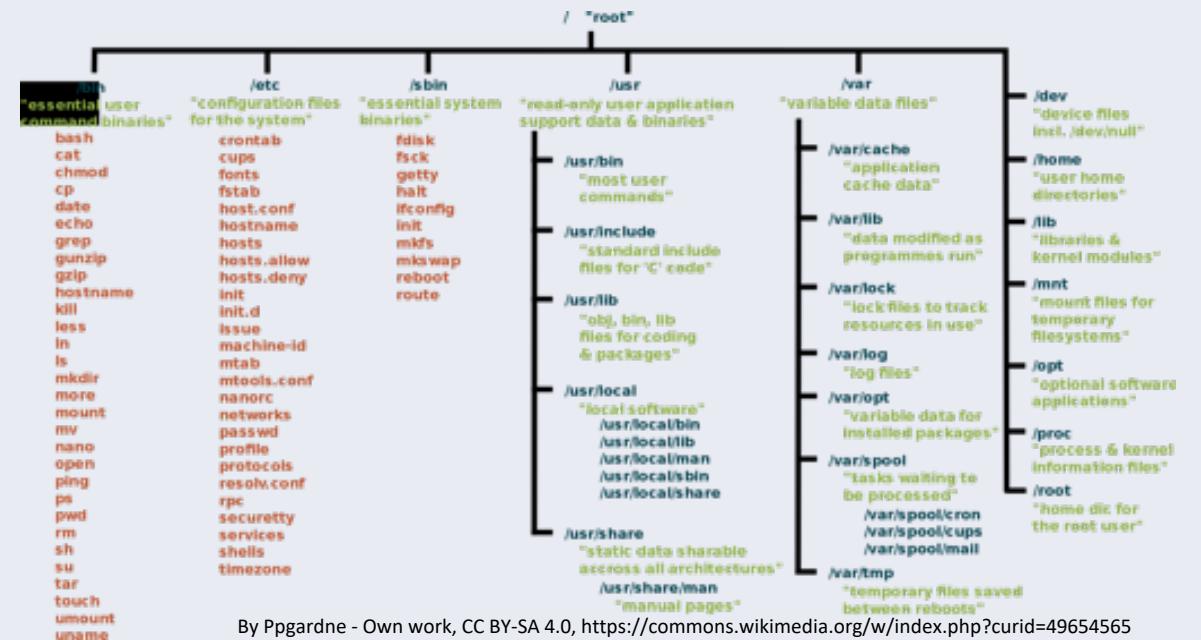


This is focus of Lab #1
Learn it!

File/Directory/Folder Structure

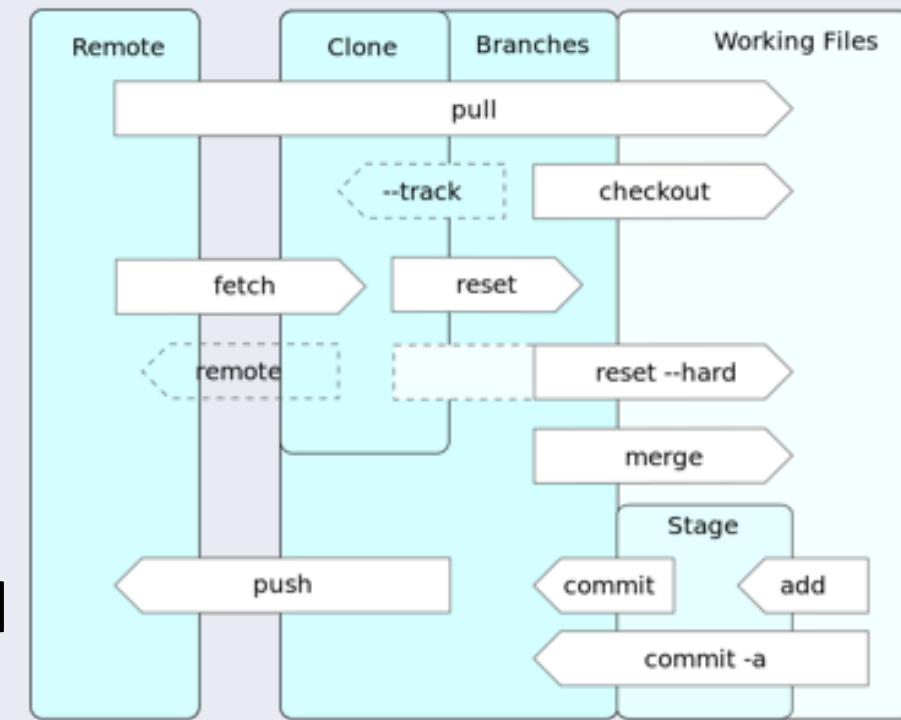
- http://www.cis.rit.edu/class/simg211/unixintro/Basics_of_File.html
- https://en.wikipedia.org/wiki/Directory_structure
- https://en.wikipedia.org/wiki/Unix_filesystem#Conventional_directory_layout

- “Folder” == “Directory”
- Windows vs. Linux separators
- “Drive” C:\ vs. “root” /
- “Home” directory “~” tilde
 - Bash: “cd ~”
- “Bash commands”
 - cd, pwd, ls



Git Basics

- Git is a distributed source code management tool
 - Every user has a **clone** of the entire repository
 - Users **commit** their local changes to their local repository
 - Users can **push** local changes to the remote server
 - Users can **pull** or **fetch** updates from the remote server
 - Users must **merge** changes from remote server on their local before pushing changes
 - For now, our work flow will try to avoid merge requirements

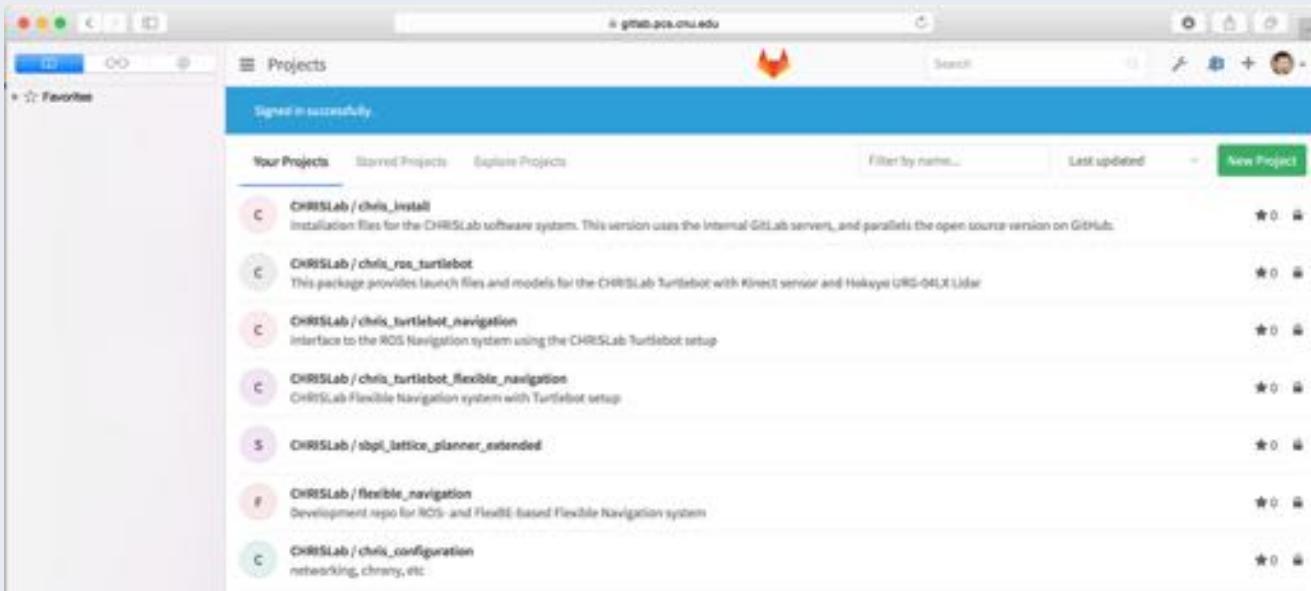


https://en.wikipedia.org/wiki/Git#/media/File:Git_operations.svg

GitLab Remote Server

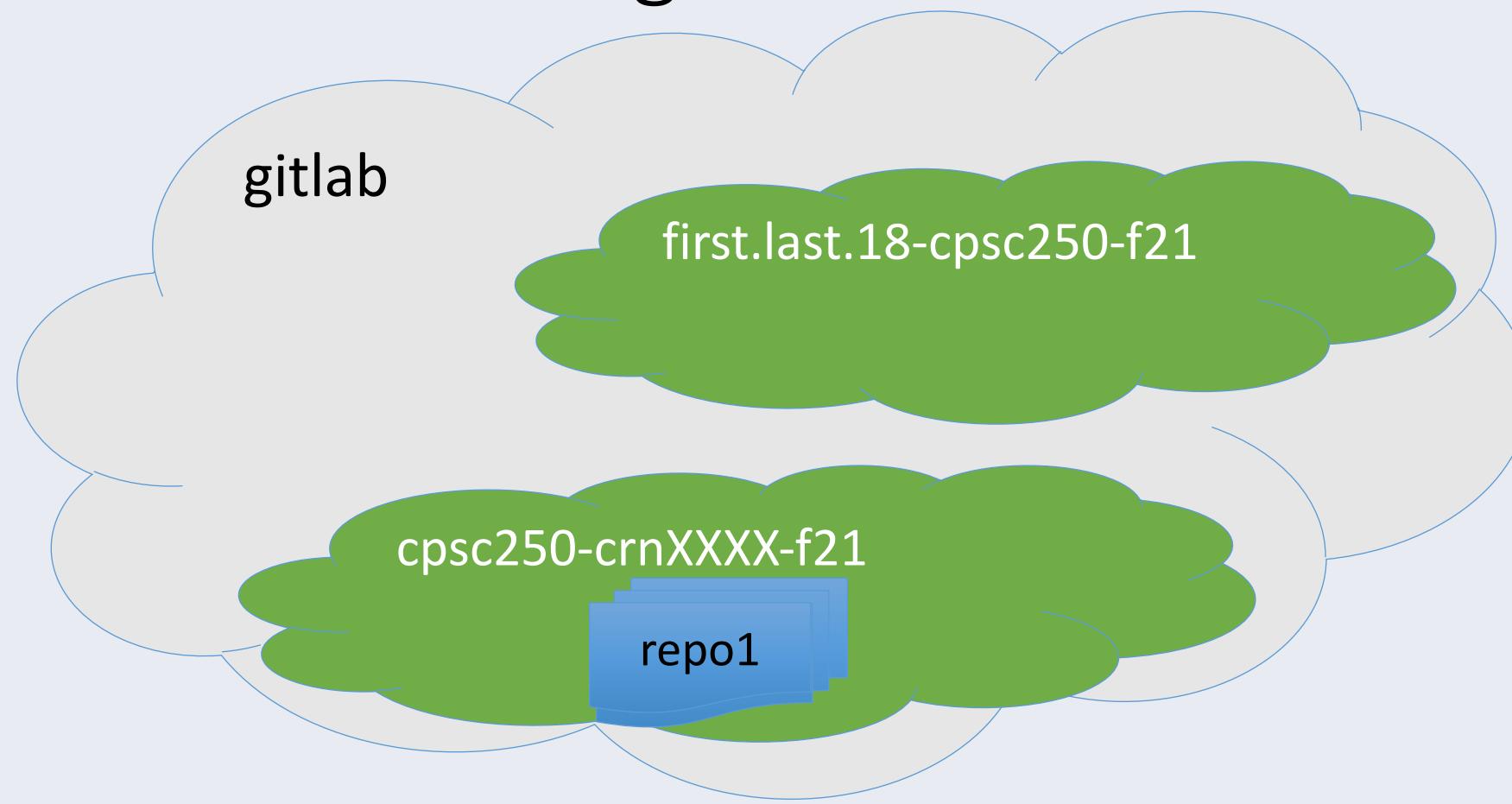
Change from s21 !

- Our common remote server
 - <https://student-gitlab.pcs.cnu.edu>
- Accounts already created using your email prefix (first.last.YY)
- You are owner of your personal first.last.YY-cpsc250-f21 group
- You are also a member(reporter) of your instructor's group:
 - cpsc250-crnXXXX-f21
- Keep all your repositories **private!**



Do NOT use github.com or gitlab.com for this course!

What does gitlab look like?

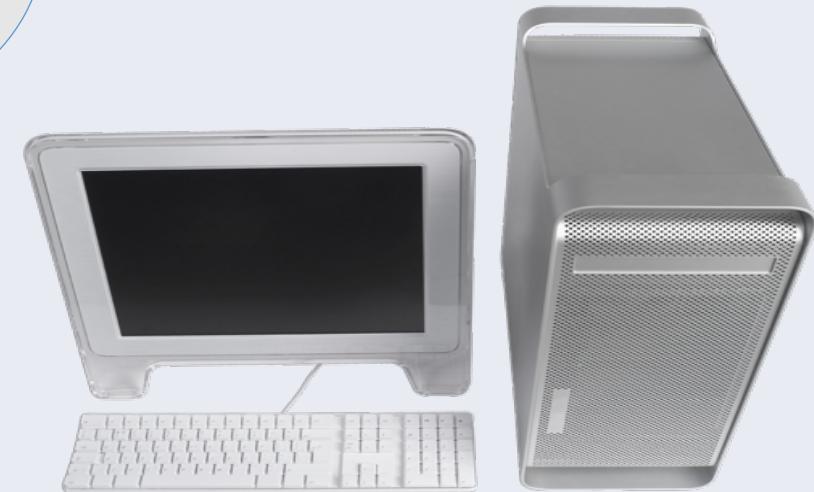
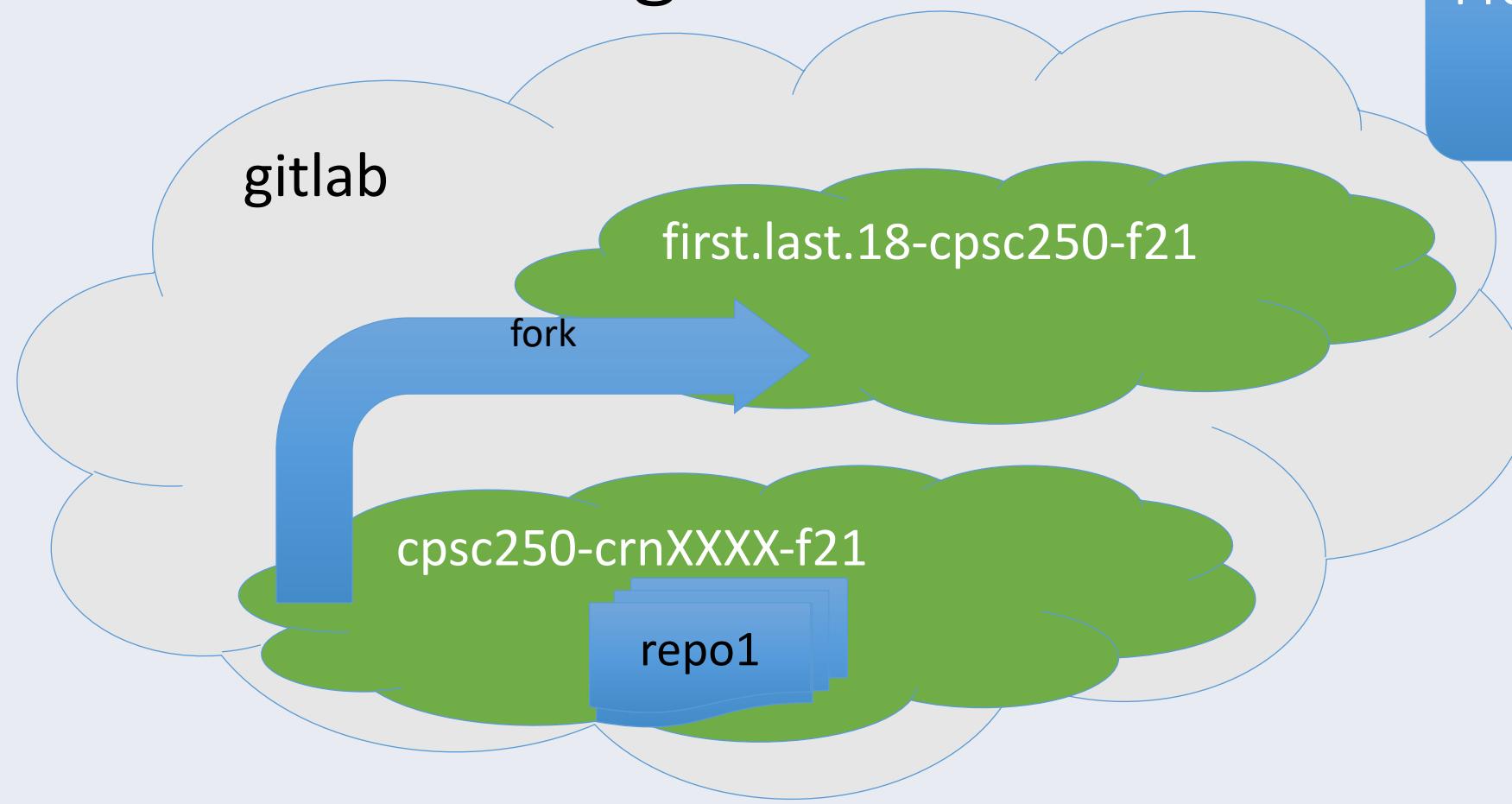


There is a group for you individually, and a group for the class. Groups can have projects in them.

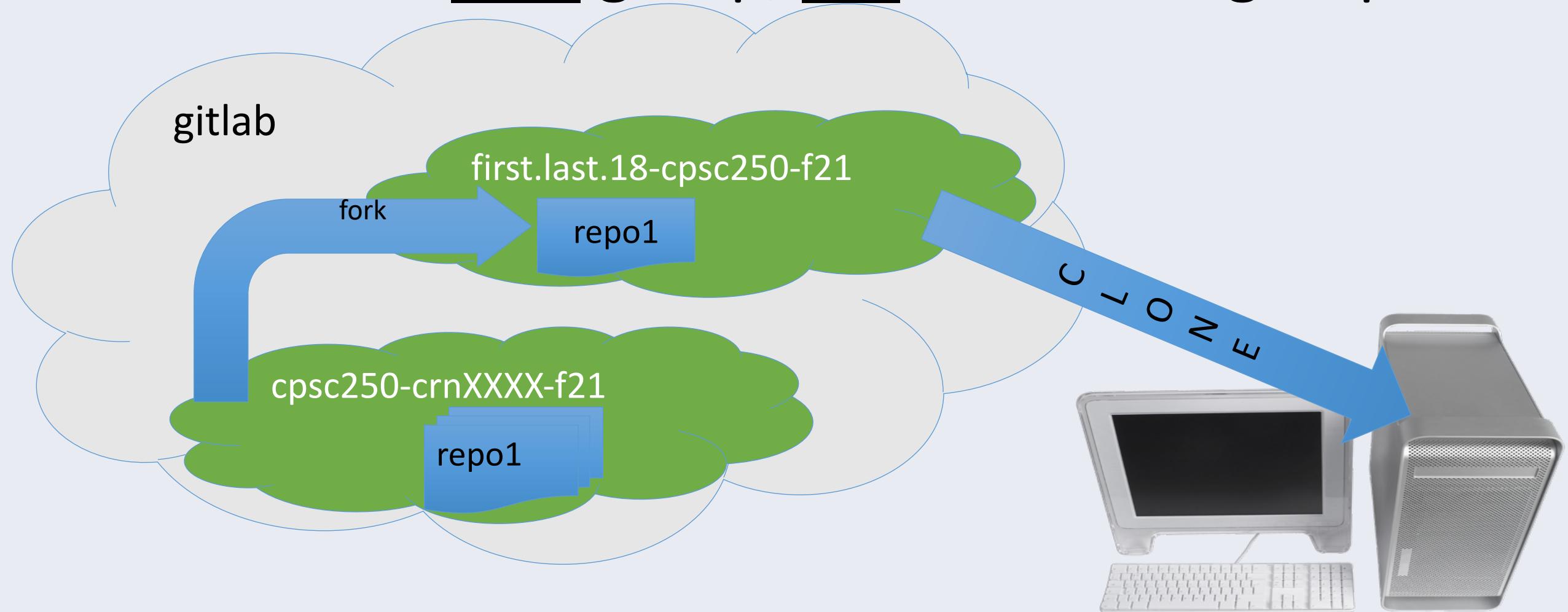


What does gitlab look like?

Projects are forked.

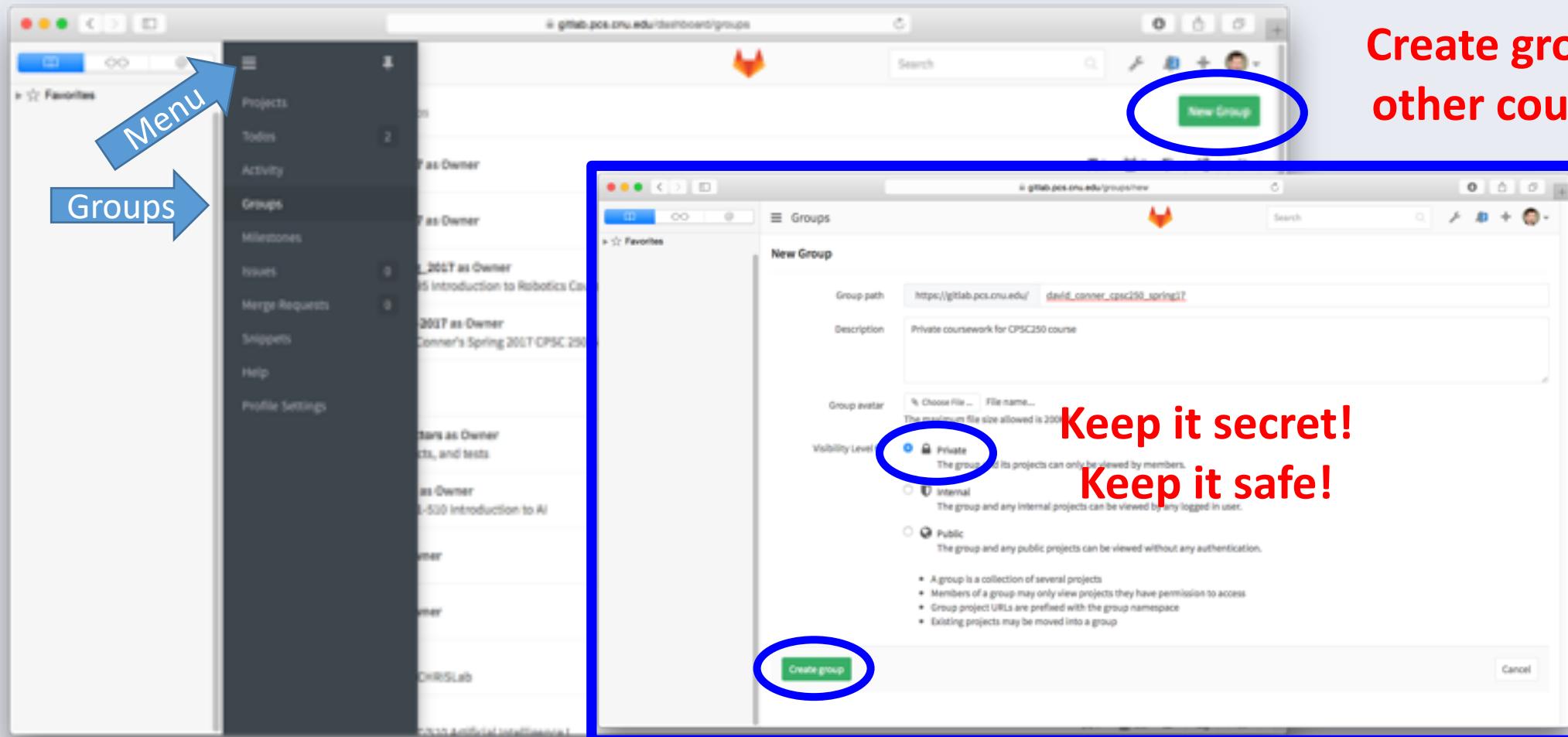


After you have forked,
clone from your group, not the class's group



Use a Private Group to organize class projects

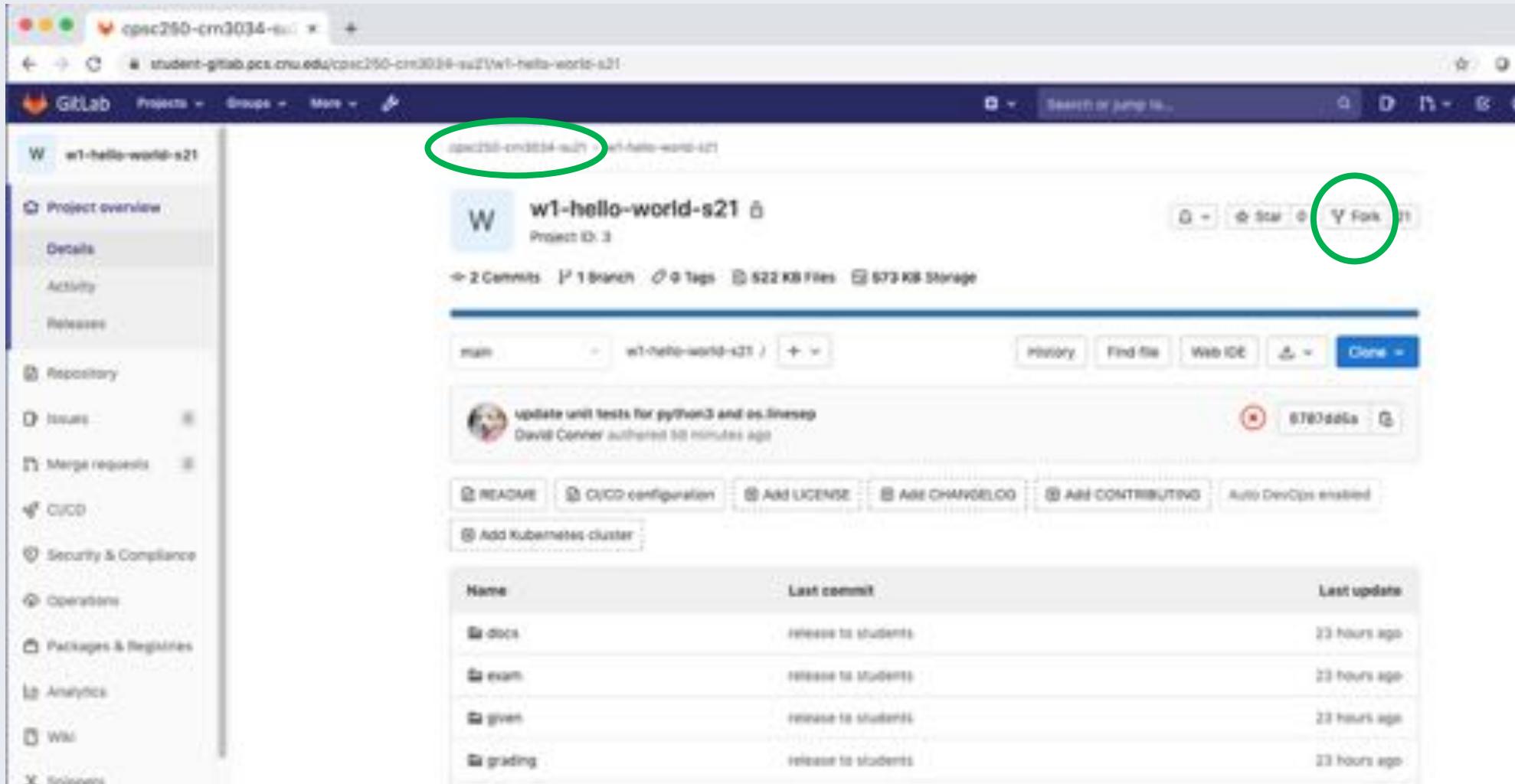
- Firstname.LastName.YY-cpsc250-f21 **We have already done this for you!**



Create groups for your other courses as well!

Keep it secret!
Keep it safe!

You want to Fork the Project from the GitLab course group to your Gitlab group via Browser



Select your personal first.last.yy-cpsc250-f21 group as target !

Select Project Fork to your Course Group

Fork project · cpsc250 · +

student-gmail.pcsc.cnu.edu/cpsc250-195336-hu21/wel1-hello-world-f21/forkbutton

GITHUB Projects Groups More Search or jump to...

wel1-hello-world-f21

Project overview Repository Issues Merge requests CI/CD Security & Compliance Operations Packages & Registries Analytics Wiki

Fork project Select a namespace to fork the project

A fork is a copy of a project. Forking a repository allows you to make changes without affecting the original project.

David Canner Go to project

Groups and subgroups

dc-testing-f21

Search by name...

Use "Select" button

Select your Firstname.Lastname.YY-cpsc250-f21

Work with your Private Fork!



The screenshot shows a GitHub fork page for a repository named `cpec250-p0-hello-world-s19`. A green oval highlights the URL in the top navigation bar: `stardavidconner/cpec250-s19/cpec250-p0-hello-world-s19`.

C cpec250-p0-hello-world-s19 Project ID: 23488

Add license 2 Commits 1 Branch 0 Tags 451 KB Files

Forked from `cpec250-students-s19/cpec250-p0-hello-world-s19`

master · cpec250-p0-hello-world-s19 · + · History · Find file · Web IDE · ⚙

 add hello_world solution · f856c4a4 · 6 · 4 months ago · David Conner authored

README · Add CHANGELOG · Add CONTRIBUTING · Auto DevOps enabled · Add Kubernetes cluster · CI/CD configuration

Project directions are in README.md (best viewed as formatted webpage in GitLab)

The screenshot shows a GitLab project interface with the README.md file open. The page has a dark header bar with the GitLab logo and navigation links. Below the header is a sidebar containing project details like 'Project ID: 23368' and 'Project URL: https://gitlab.cs.vt.edu/courses/phys2300-fall2019-101/spec2300-ph2300-world-101'. The main content area displays the README.md file's text.

README.md

This project involves the simple field-oriented example as we demonstrate the various exercise checklist that we will use in lecture and test this behavior. Learning these tools is part of your professional development, and is required for this course. You are expected to master this content.

Students are required to demonstrate forking and cloning this project using GitHub.com. Then viewing a Python project parts to the project, and modifying the code as directed. Grading occurs in two parts:

- The code will be automatically graded by WebCat (20 points).
 - Check your WebCat page to verify points. This is the grade that you will receive.
 - Due on WebCat by Friday, 11 January by 11:59pm.
- Bring your personal laptop to the office hours and demonstrate all four working (20 points).
 - Contact me if you do not have a laptop.
 - Give my office by Tuesday 15 January by end-of-office hours.

GitLab Setup

We have created a private GitLab group for you to use in this class, following the [GitLab/GitHub/Java/C/C++-specific workflow](#) convention. Your lecturer and lab instructors are members of this group, and can view all of your projects in this group. You are required to use this group for all assignments in this course.

Be sure to change your `username` and `password` variation to your lecturer ([Christopher.Neptunus.cs.vt.edu@2300-repository](#)) prior to that your code is submitted to WebCat with your user ID and password. See the lecture slides for details on updating these variables.

Using this course for all projects assigned by this group for both lecture and lab. It is your responsibility to verify that your code is on WebCat and successfully passed the required tests. If you change your password on WebCat, then be sure to update the password here.

You may use your personal computer for this initial work, or if you do not have the software installed, then use the lab machines with the `username` and no password.

Projects in C/C++ 2019 will have common structures:

```
spec2300-project_001
├── .gitignore
└── source [files that you will modify]
    └── main.c [source file that you should NOT modify]
        └── [empty folder for uploaded compiled code - do NOT modify]
            └── [empty folder for instructions and documentation]
                └── [empty folder for upload user's README and documentation]
```

All of your changes should be in files in the `.git` folder, do not modify or delete files in the other folders.

Setting up the project

From this repository, into your personal gitlab (or GitHub) account, after the fork completes, you will be a collaborator in that group. Check the URL for your group name to verify. Once the fork is done, select the HTTPS URL in the URL section below the repository name, and copy it. Open up Git Bash, `cd` into `git/repoURL` (make that directory using `mkdir` if it does not exist).

`git clone` (or use `git pull`, if you have already cloned in the `git` folder which is used only in this action) change to your home directory by using the command `cd ~` (optional).

From your designated spec2300 folder type `git add .` followed by a space and paste the URL of your linked repository. Before pressing enter, make sure that your group name is in that URL. If it is, press enter.

To clone, first copy link from your private fork

The screenshot shows a GitLab repository page for a forked project. A green oval highlights the URL in the browser's address bar: `gitlab.pcs.cnu.edu/david/conner/cpac250-s19/cpac250-p0-hello-world-s19`. A large blue arrow points from this URL to the right, labeled "Use https://". On the right side of the page, there is a "Clone" button with a dropdown menu. This menu is also circled in green. The "Clone with HTTPS" option is selected, showing the URL: `https://gitlab.pcs.cnu.edu/david/conner/cpac250-s19/cpac250-p0-hello-world-s19`. A red arrow points from this URL to a blue callout box containing the text "Handy dandy copy link".

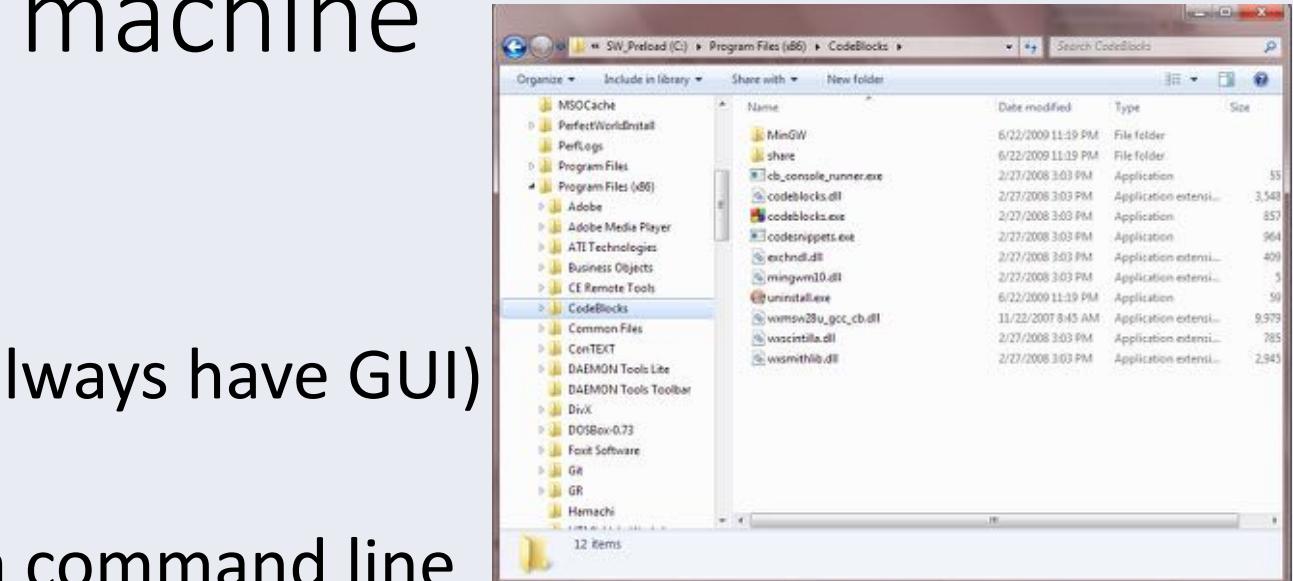
Use Command line and clone !
Never download or upload via web page

Now, go back to local machine Prerequisites

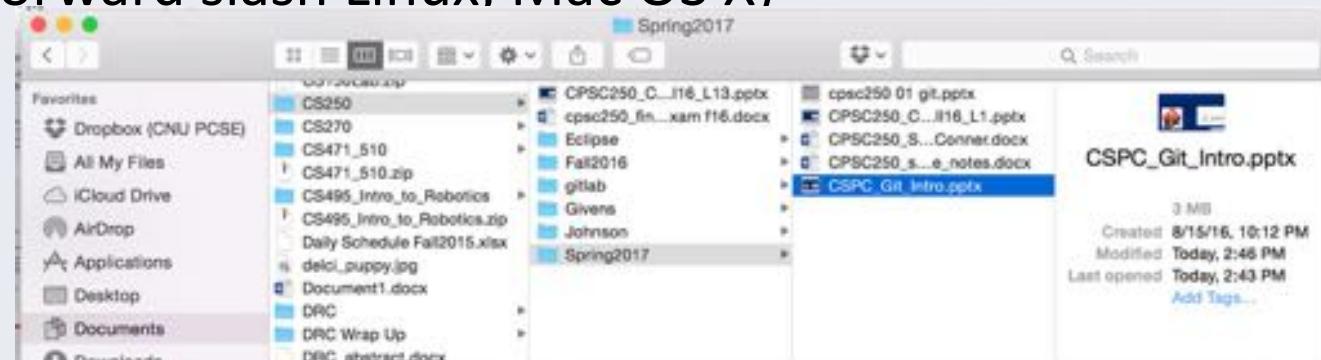
- Understand your file structure!
- ***Use the command line*** (won't always have GUI)
 - Prefer **gitbash** on Windows
- Understand how to navigate via command line
 - command, shell, terminal
 - cd, pwd, ls or dir
 - \ (backslash on Windows) vs. / (forward slash Linux. Mac OS X)

```
Spring2017 – bash – 80x18
Davids-MacBook-Pro:Documents dconner$ cd CS250/
Davids-MacBook-Pro:CS250 dconner$ ls
CPSC250_Conner_Fall16_L13.pptx Johnson
Eclipse - Command, shell, ter Spring2017
Fall2016 cd, ls or dir cpsc250_final_exam f16.docx
Givens gitlab
Davids-MacBook-Pro:CS250 dconner$ cd Spring2017/ward slash Linus, Mac OS X)
Davids-MacBook-Pro:Spring2017 dconner$ ls
CPSC250_Conner_Fall16_L1.pptx cpsc250 01 git.pptx
CPSC250_Spring17_Conner.docx ~$SC250_Spring17_Conner.docx
CPSC250_Spring2017_lecture_notes.docx ~$cpsc250 01 git.pptx
CSPC_Git_Intro.pptx
Davids-MacBook-Pro:Spring2017 dconner$
Davids-MacBook-Pro:Spring2017 dconner$ pwd
/Users/dconner/Documents/CS250/Spring2017
Davids-MacBook-Pro:Spring2017 dconner$ dir
-bash: dir: command not found
Davids-MacBook-Pro:Spring2017 dconner$
```

Windows command



<https://gitlab.pcs.cnu.edu/cpsc-labs/bash-and-git-wiki/wikis/home>



Define folder for projects

- On lab machines, make sure you are in writeable folder
 - cd ~ (from gitbash)
 - cd \users\pcse_user (Windows)
- You might choose to put under Desktop or Documents
 - Remember where you put the files!
- Create course folder
 - mkdir cpsc250 (bash)
 - md cpsc250 (on Windows)
- Move to course folder
 - cd cpsc250

Clone using the terminal!



Clone results

```
(base) davids-mbp:cs250-su21 dconner$ git clone https://student-gitlab.pcs.cnu.edu/cpsc250-crn3034-su21/w1-hello-world-s21.git
Cloning into 'w1-hello-world-s21'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 35 (delta 0), reused 0 (delta 0), pack-reused 30
Unpacking objects: 100% (35/35), done.
(base) davids-mbp:cs250-su21 dconner$
```

It will likely request your login credentials

If you get an error ,

it may require clearing your credentials (Windows Credential Manager), or
you might need to reboot the lab machine (“remote not available”)

Set Your Git Configuration Information

- `git config --global user.name "First Last"`
- `git config --global user.email first.last.YY@cnu.edu`

Two dashes
(minus signs)

Note the quotation marks around name and helper

For Ubuntu (Linux)

**Do this once on personal machine,
but every time you log into a lab machine!**

- `git config --global credential.helper='cache --timeout=1800'`
- For Windows/Mac, just use the built-in credential manager

To check config, use

- `git config --global -l`
- `git config --global --unset credential.helper`
if you already set credential manager on Windows

Change into cloned folder and list files

cd w1<tab> to use tab completion to save time!

```
(base) davids-mbp:cs250-su21 dconner$ cd w1-hello-world-s21/
(base) davids-mbp:w1-hello-world-s21 dconner$ pwd
/Users/dconner/Documents/CS250/Su21/git/cs250-su21/w1-hello-world-s21
(base) davids-mbp:w1-hello-world-s21 dconner$ ls
README.md          given           src           webcat-submitter.py
docs               grading         tests
exam              img             webcat-submitter-1.0.4.jar
(base) davids-mbp:w1-hello-world-s21 dconner$ █
```

ls to list files in bash terminal (unix/mac or gitbash)

To see all the files

`ls -altr` to list all files and details (unix/mac or gitbash)

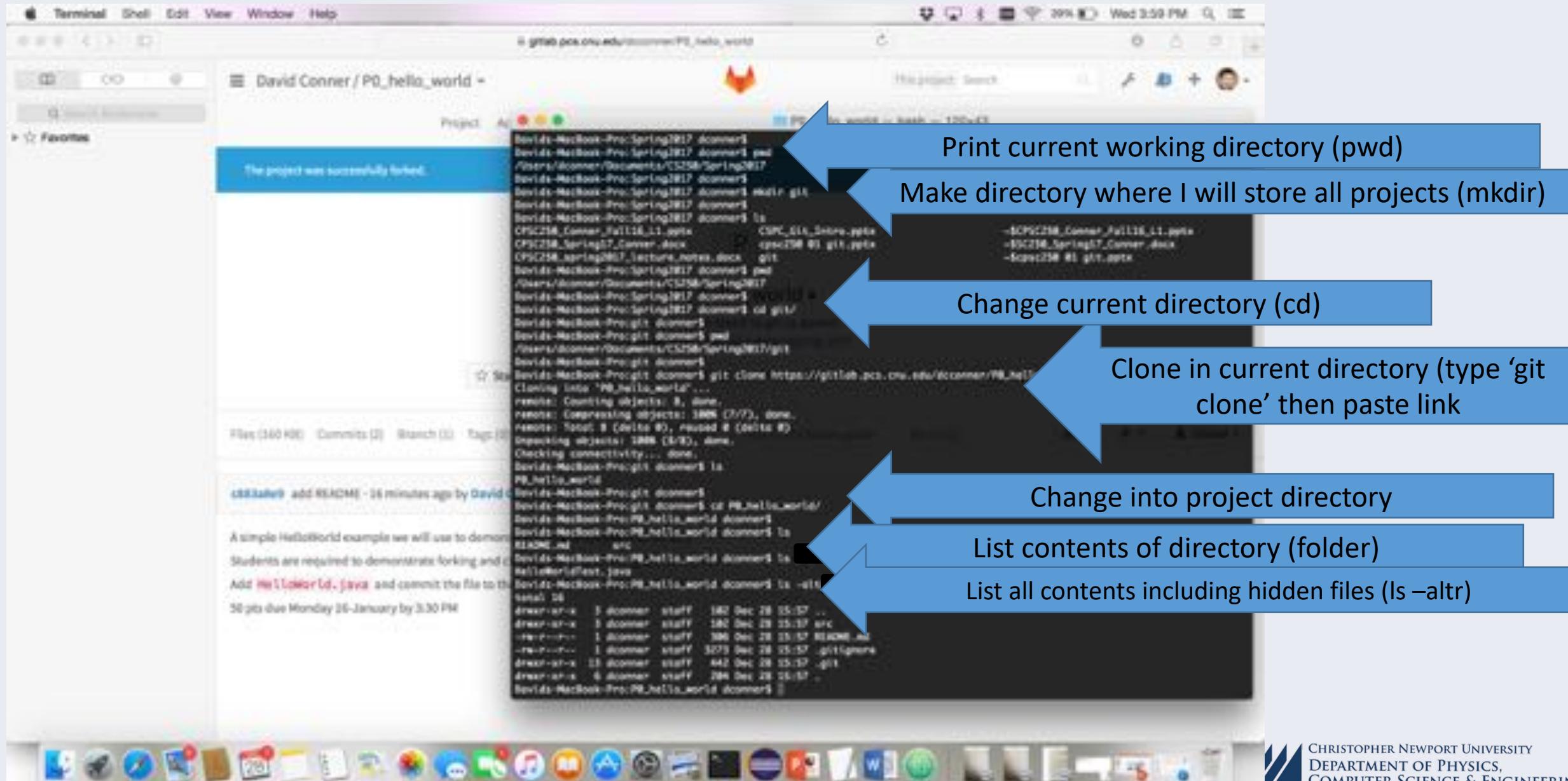
```
Davids-MBP:cpsc250-p0-hello-world-s19 dconner$ ls -altr
total 224
drwxr-xr-x  7 dconner  staff   238 Jan  2 16:54 ..
-rw-r--r--  1 dconner  staff  1370 Jan  2 16:54 webcat-submitter.py
-rw-r--r--  1 dconner  staff 74791 Jan  2 16:54 webcat-submitter-1.0.0
drwxr-xr-x  4 dconner  staff   136 Jan  2 16:54 webcat
drwxr-xr-x  5 dconner  staff   170 Jan  2 16:54 tests
drwxr-xr-x  5 dconner  staff   170 Jan  2 16:54 src
drwxr-xr-x  4 dconner  staff   136 Jan  2 16:54 img
-rw-r--r--  1 dconner  staff 12270 Jan  2 16:54 README.md
-rw-r--r--  1 dconner  staff   925 Jan  2 16:54 .gitlab-ci.yml
-rw-r--r--  1 dconner  staff 13154 Jan  2 16:54 .gitignore
drwxr-xr-x 13 dconner  staff   442 Jan  2 16:54 .git
408 Jan  2 16:54 .
```

List of file (types) that
git will ignore

Continuous Integration (CI)
rules

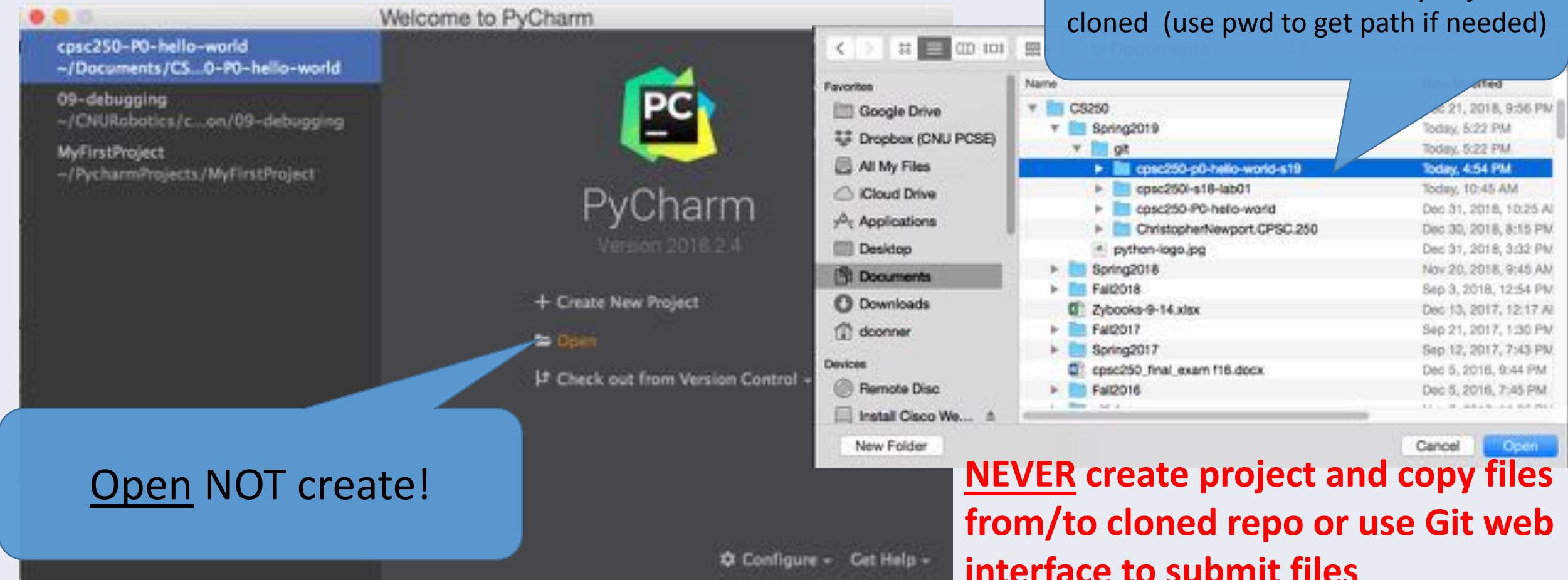
This is your repo directory
– don't mess with it!

More (git)Bash commands



Ready to Begin Editing in PyCharm

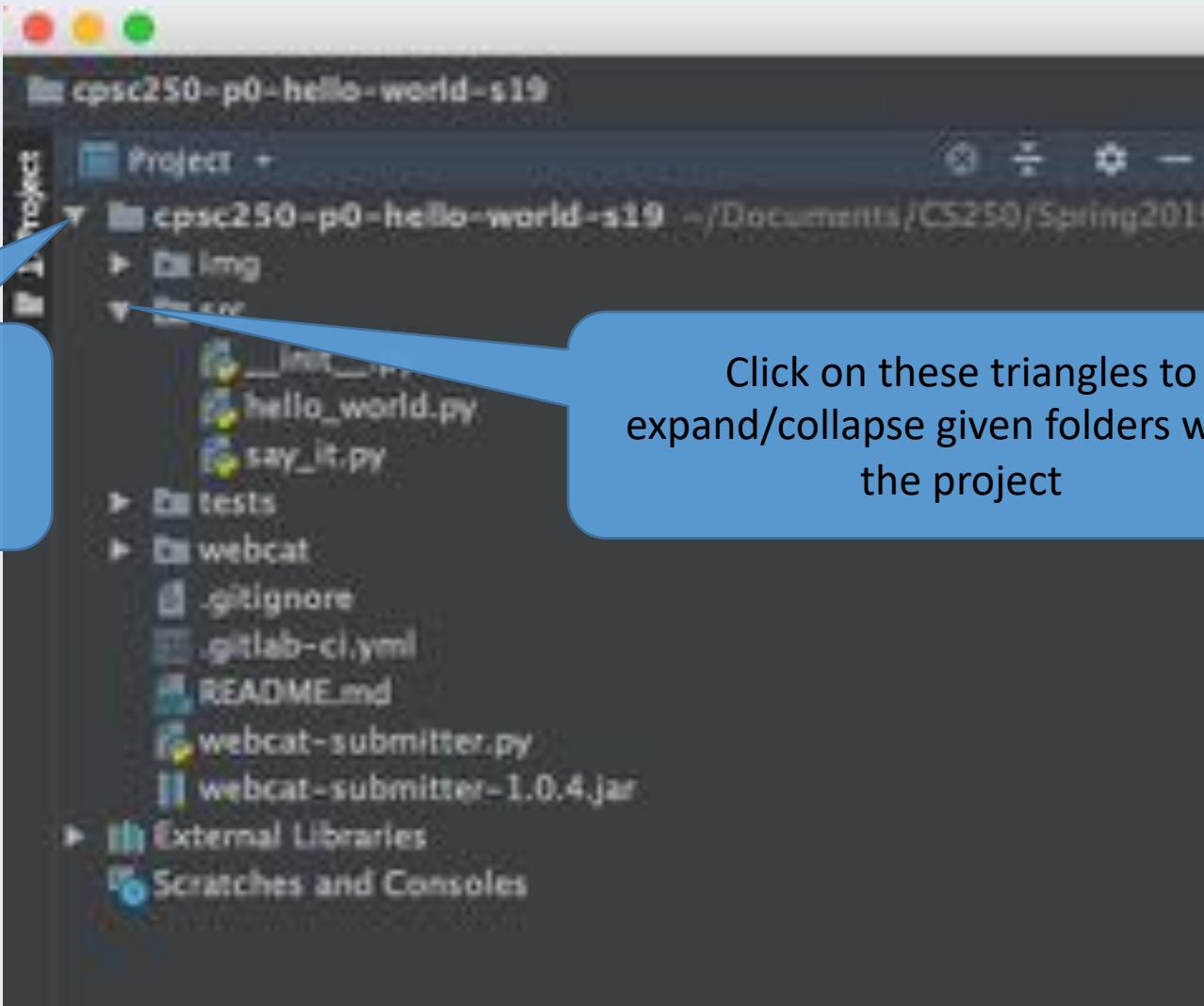
- After git directory is set up, and you have cloned your repo
- Now start PyCharm and **Open** existing project



Open NOT create!

NEVER create project and copy files from/to cloned repo or use Git web interface to submit files

PyCharm IDE



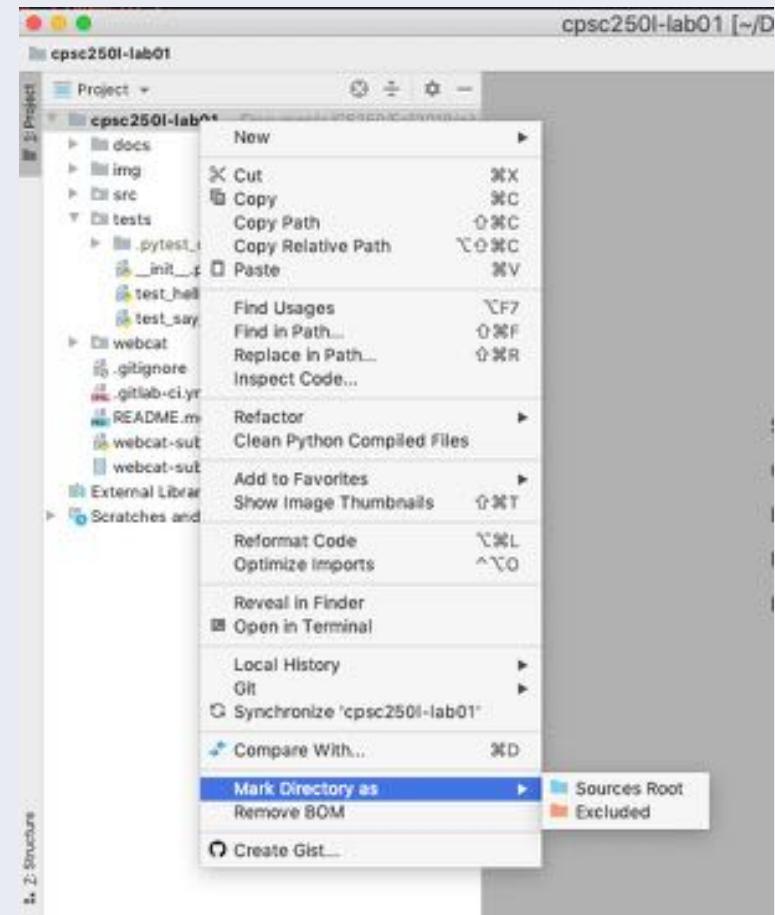
Set the “Sources Root” folder

Right click on top level folder (project name)

Mark Directory As
Sources Root

This cause scripts to run as if they were
executed in the main project folder;
this is required for tests to run properly

Do this for all projects in CPSC 250



Now Edit your code in PyCharm

- Follow directions for PyCharm setup for w1
- Do Exercise 1 and then return to command line (terminal)

Git Status and Diff to Verify Changes

```
Davids-MBP:cpsc250-p0-hello-world-s19 dconner$ git status
On branch master
Your branch is up-to-date with 'origin/master'. (feat.. JR)
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:   src/hello_world.py

no changes added to commit (use "git add" and/or "git commit -a")
Davids-MBP:cpsc250-p0-hello-world-s19 dconner$
```

Oh good, we changed the correct file in correct location!

```
Davids-MBP:cpsc250-p0-hello-world-s19 dconner$ git diff
diff --git a/src/hello_world.py b/src/hello_world.py
index 4dbbbb0..2bef5f8 100644
--- a/src/hello_world.py
+++ b/src/hello_world.py
@@ -4,6 +4,13 @@ Script to print Hello World!
# Create a method called say_hello that returns "Hello World!"
# Then print by calling method
+def say_hello():
+    """
+        Specify a string
+        @return string
+    """
+    return "Hello World!"

-Crowder - No Rival (Audio).ft..JR
+print("Hello World!") # Modify to use method
+
+print(say_hello()) # Modify to use method
Davids-MBP:cpsc250-p0-hello-world-s19 dconner$
```

Yep, that's what I changed!

Git Add and Commit

- Two options :
 1. Separate add and commit commands
 - `git add .` (the `.` says add all untracked files in directory)
 - `git commit -m "a relevant message"`
 2. Combined add and commit for previous tracked
 - `git commit -am "a relevant message"`
 - “a” for *all* modified previously tracked files, and
 - “m” for message
- Use `-m` to avoid tricky editors (e.g. default Vim)

Git Log to verify commits

```
Davids-MBP:cpsc250-p0-hello-world-s19 dconner$ git commit -am "add hello_world solution"
[master f056c4a] add hello_world solution
 1 file changed, 8 insertions(+), 1 deletion(-)
Davids-MBP:cpsc250-p0-hello-world-s19 dconner$ git log
commit f056c4a4d031067886041a4eb33b52f5a388d7af
Author: David Conner <david.conner@cnu.edu>
Date:   Wed Jan 2 17:51:48 2019 -0500

    add hello_world solution

commit aaa60431cadede76eaf366f704fd0da27510763b
Author: David Conner <david.conner@cnu.edu>
Date:   Mon Dec 31 15:39:38 2018 -0500

    initial release
Davids-MBP:cpsc250-p0-hello-world-s19 dconner$
```

Git configuration info
of the committer



WARNING: Use -a option or “git add” first

```
davids-mbp:cpsc250-p0-hello-world-s19 dconner$ git status
On branch master
Your branch is up to date with 'origin/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:   src/hello_world.py

no changes added to commit (use "git add" and/or "git commit -a")
davids-mbp:cpsc250-p0-hello-world-s19 dconner$ git commit
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  modified:   src/hello_world.py

no changes added to commit
davids-mbp:cpsc250-p0-hello-world-s19 dconner$ git add .
davids-mbp:cpsc250-p0-hello-world-s19 dconner$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   src/hello_world.py
```

- If you forget the -a option you must “add” the files first!

Commit doesn't do anything because nothing added to be “staged for commit”

git add . Says to add everything in current folder (.) can also add specific files

git status let's us know the status; now we are ready to commit

WARNING: Avoid tricky editors

- If you forget the -m “commit message”, it will open a “tricky editor”
 - The default is “vim-like”, which works but takes some getting used to

```
davids-mbp:cpsc250-p0-hello-world-s19 dconner$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   src/hello_world.py

davids-mbp:cpsc250-p0-hello-world-s19 dconner$ git commit
[master dbf434F] Add your commit message here.  Can use multi-line
 1 file changed, 1 insertion(+), 1 deletion(-)
davids-mbp:cpsc250-p0-hello-world-s19 dconner$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
davids-mbp:cpsc250-p0-hello-world-s19 dconner$
```

Forgetting the -m

May open this “vim”-like editor

```
cpsc250-p0-hello-world-s19 — vi + git commit — 136x26

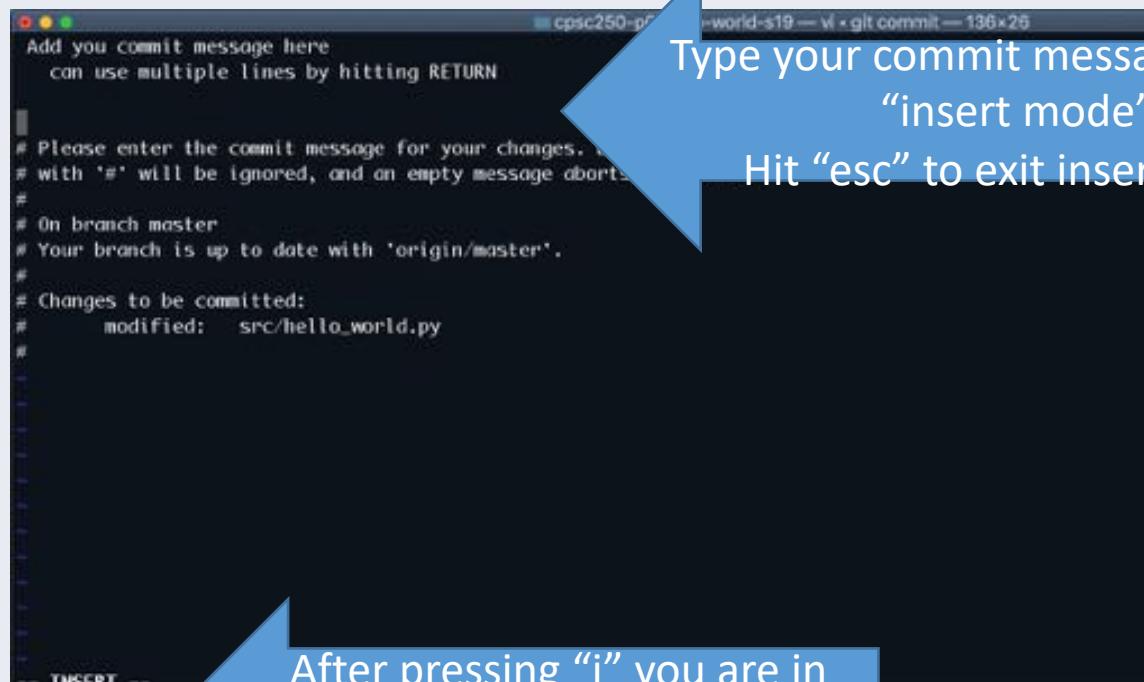
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Your branch is up to date with 'origin/master'.
#
# Changes to be committed:
#       modified:   src/hello_world.py

~/Documents/CS250/Spring2019/git/cpsc250-p0-hello-world-s19/.git/COMMIT_EDITMSG" 10L, 271C
```

Press “i” to enter “insert” mode

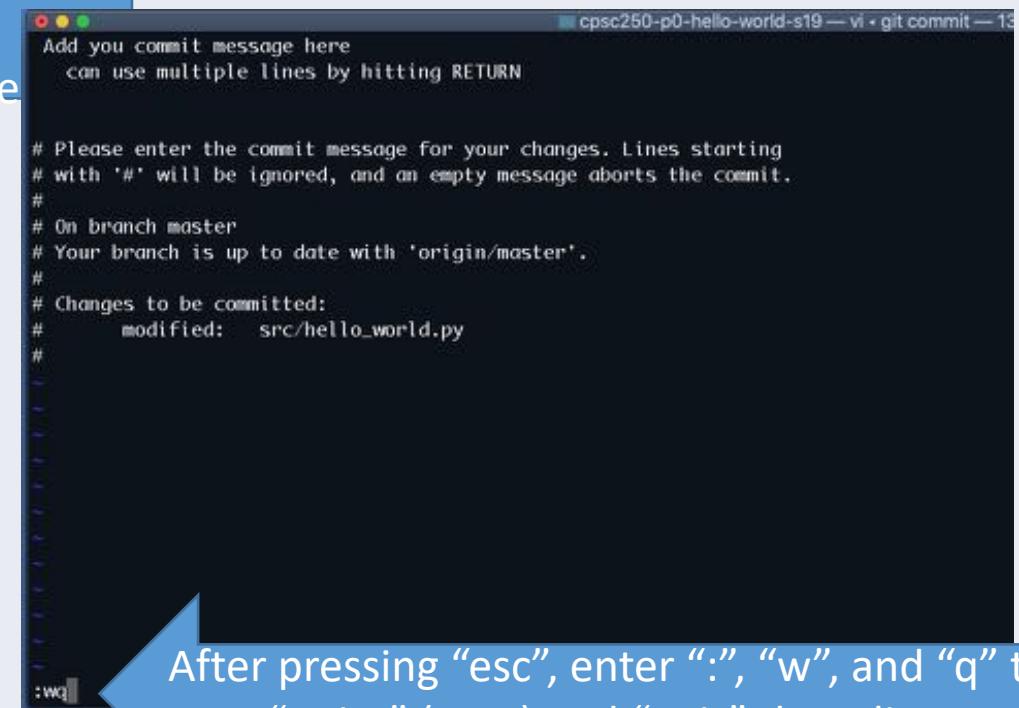
WARNING: Avoid tricky editors

- If you forget the -m “commit message”, it will open a “tricky editor”
 - The default is “vim-like”, which works but takes some getting used to



```
Add you commit message here  
can use multiple lines by hitting RETURN  
  
# Please enter the commit message for your changes.  
# Lines starting # will be ignored, and an empty message aborts  
#  
# On branch master  
# Your branch is up to date with 'origin/master'.  
#  
# Changes to be committed:  
#   modified: src/hello_world.py  
#
```

Type your commit message while in
“insert mode”
Hit “esc” to exit insert mode



```
Add you commit message here  
can use multiple lines by hitting RETURN  
  
# Please enter the commit message for your changes. Lines starting  
# with '#' will be ignored, and an empty message aborts the commit.  
#  
# On branch master  
# Your branch is up to date with 'origin/master'.  
#  
# Changes to be committed:  
#   modified: src/hello_world.py  
#
```

After pressing “esc”, enter “:”, “w”, and “q” to
“write” (save) and “quit” the editor

Git push to back up changes to remote

Origin is the default name of the remote
Main is the default branch name
(more about branching later)

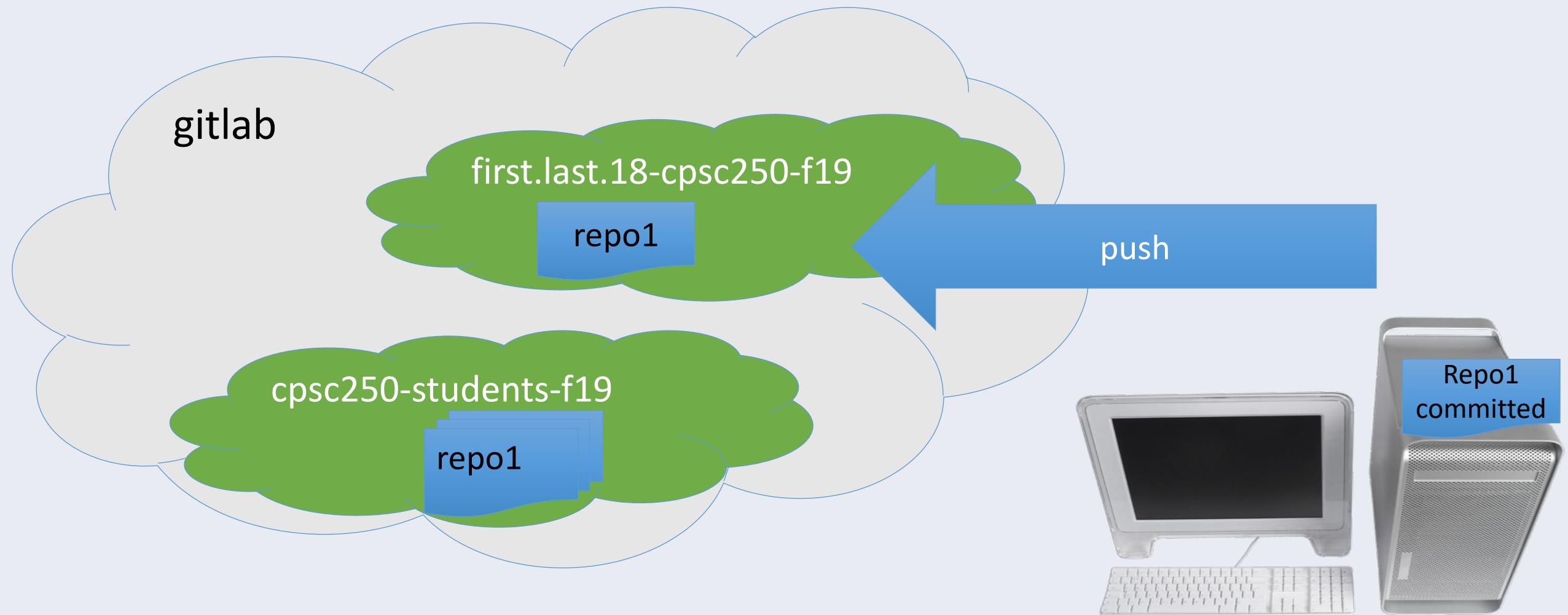
```
Davids-MBP:cpsc250-p0-hello-world-s19 dconner$ git push origin main
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 546 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To https://gitlab.pcs.cnu.edu/david.conner-cpsc250-s19/cpsc250-p0-hello-world-s19.git
   aaa6043..f056c4a  master -> master
Davids-MBP:cpsc250-p0-hello-world-s19 dconner$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
Davids-MBP:cpsc250-p0-hello-world-s19 dconner$
```

“git push” by itself will push all local changes in all branches, which can be dangerous;
get used to specifying target branches. Or ...

git config push.default current

- upstream - push the current branch to its upstream branch...
- simple - like upstream, but refuses to push if the upstream branch's name is different from the local one...
- current - push the current branch to a branch of the same name.

Git Push to the cloud



Gitbash : Add, commit, push

```
PO_hello_world - bash - 120x43
drwxr-xr-x  6 dconner  staff  204 Dec 28 15:57
Davids-MacBook-Pro:PO_hello_world dconner$ ls
HelloWorld.java      HelloWorldTest.java
Davids-MacBook-Pro:PO_hello_world dconner$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    src/HelloWorld.java

nothing added to commit but untracked files present (use "git add" to track)
Davids-MacBook-Pro:PO_hello_world dconner$ git add src/HelloWorld.java
Davids-MacBook-Pro:PO_hello_world dconner$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   src/HelloWorld.java

Davids-MacBook-Pro:PO_hello_world dconner$ git commit -am "added HelloWorld that passed JUnit!"
-bash: !": event not found
Davids-MacBook-Pro:PO_hello_world dconner$ git commit -am "added HelloWorld that passed JUnit."
[master cf20035] added HelloWorld that passed JUnit.
 1 file changed, 9 insertions(+)
 create mode 100644 src/HelloWorld.java
Davids-MacBook-Pro:PO_hello_world dconner$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
Davids-MacBook-Pro:PO_hello_world dconner$ git push origin main
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 486 bytes | 0 bytes/s, done.
Total 4 (delta 0)
To https://gitlab.pcs.cmu.edu/dconner/PO_hello_world.git
 c883a8e..cf20035  main --> main
Davids-MacBook-Pro:PO_hello_world dconner$
```

List file in current directory

git status shows untracked file

git add to start “tracking” (use git add .)

git status shows files “staged” to commit

git commit to record the changes in local repo (git commit –m “message”)

Do this frequently!

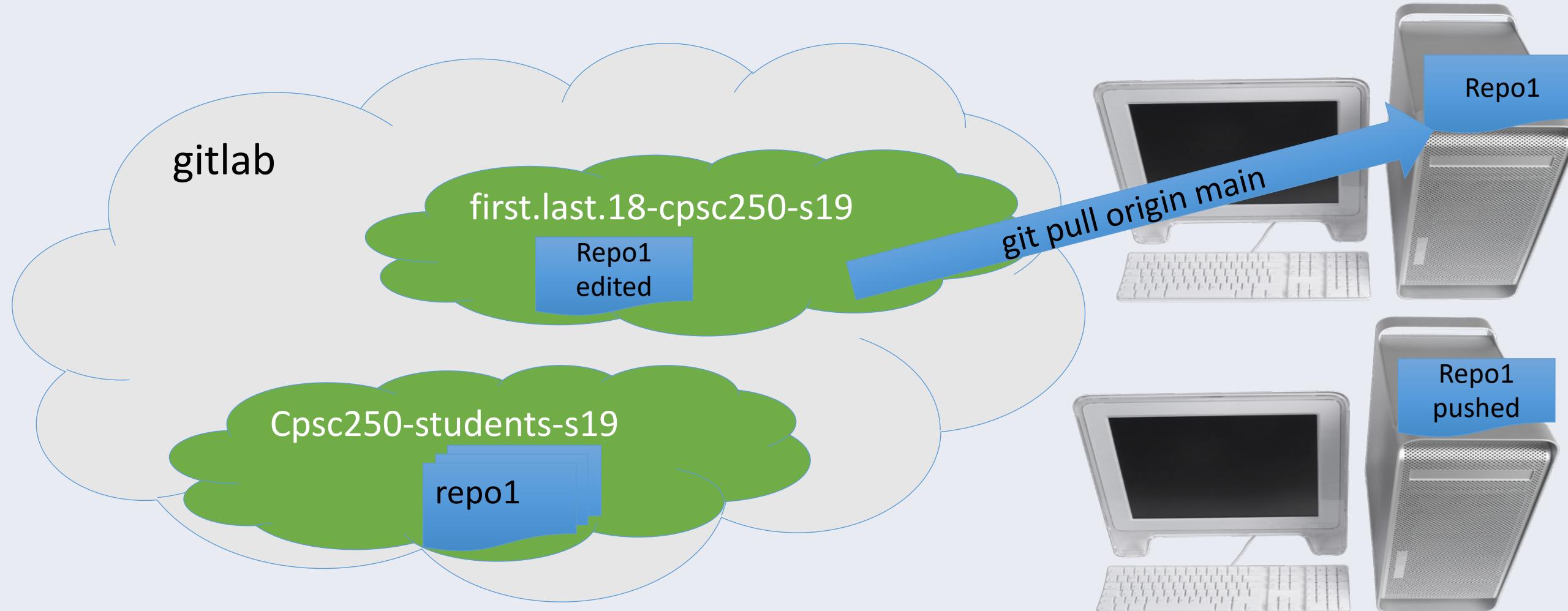
git status shows we are ahead of remote

git push local committed changes to remote

Do this at least at the end of every coding session!



If working with partners, or on different machines after a clone, do a git pull before each add, commit, push session



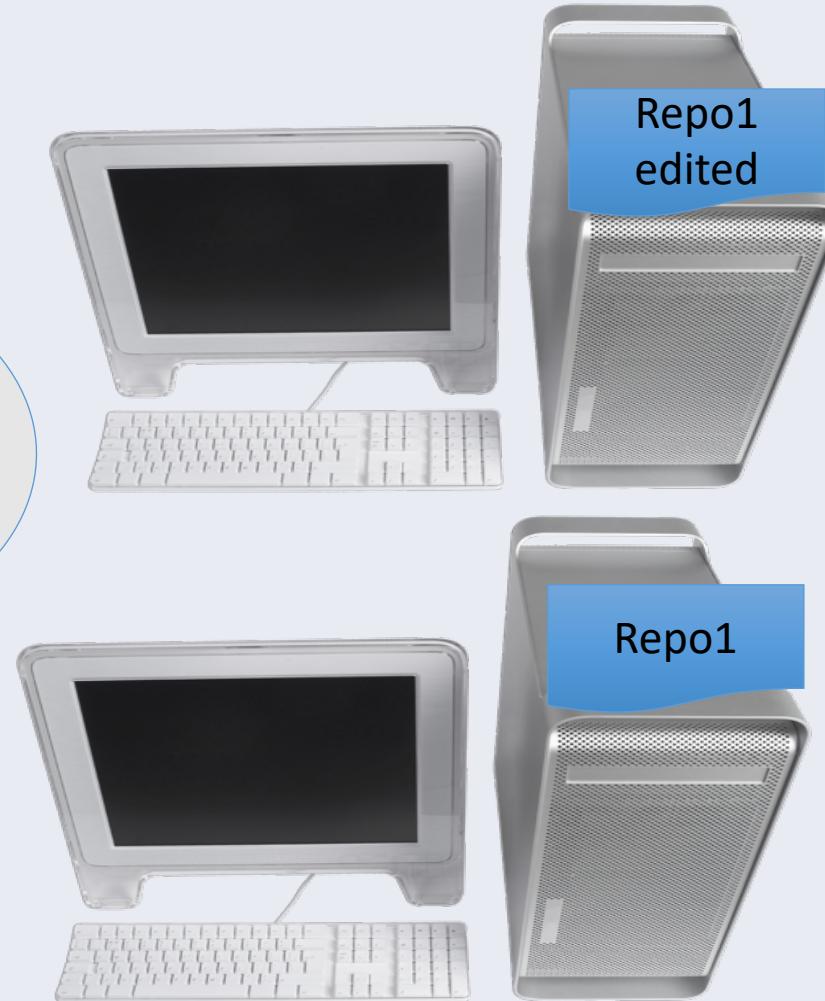
After cloning, before each add, commit, push session

gitlab

(after git pull origin main or git clone)

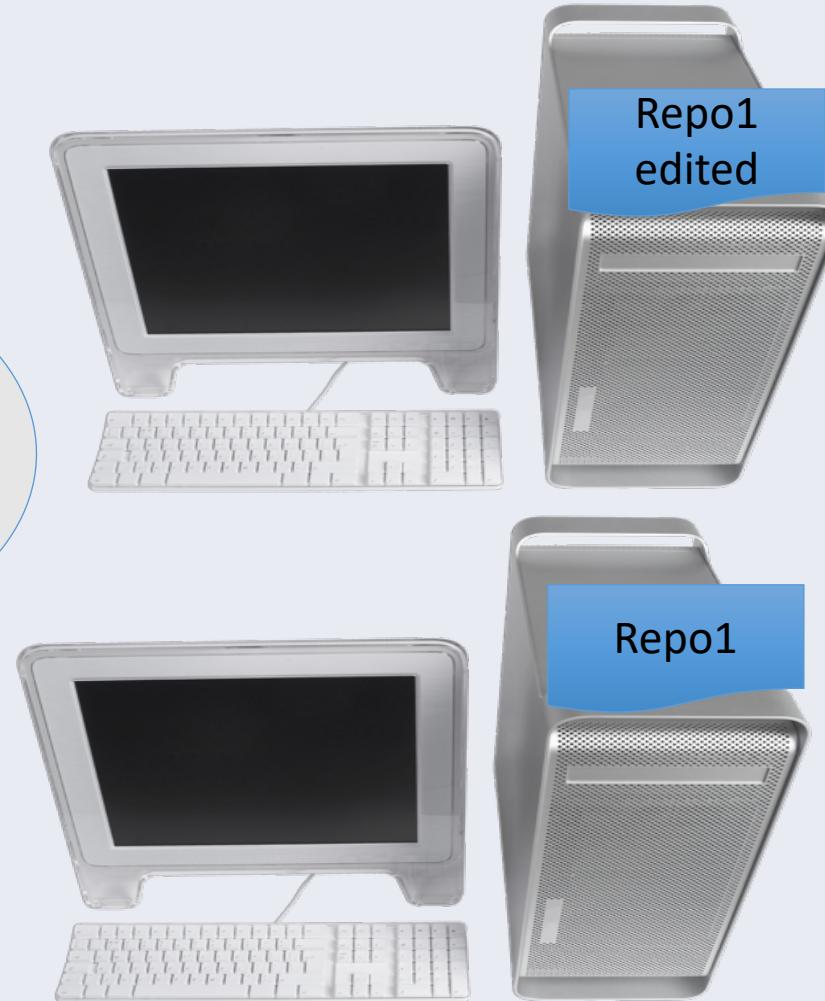
1. Edit files
2. git add .
3. git commit -m "worked on this"
4. git push origin main

repo1



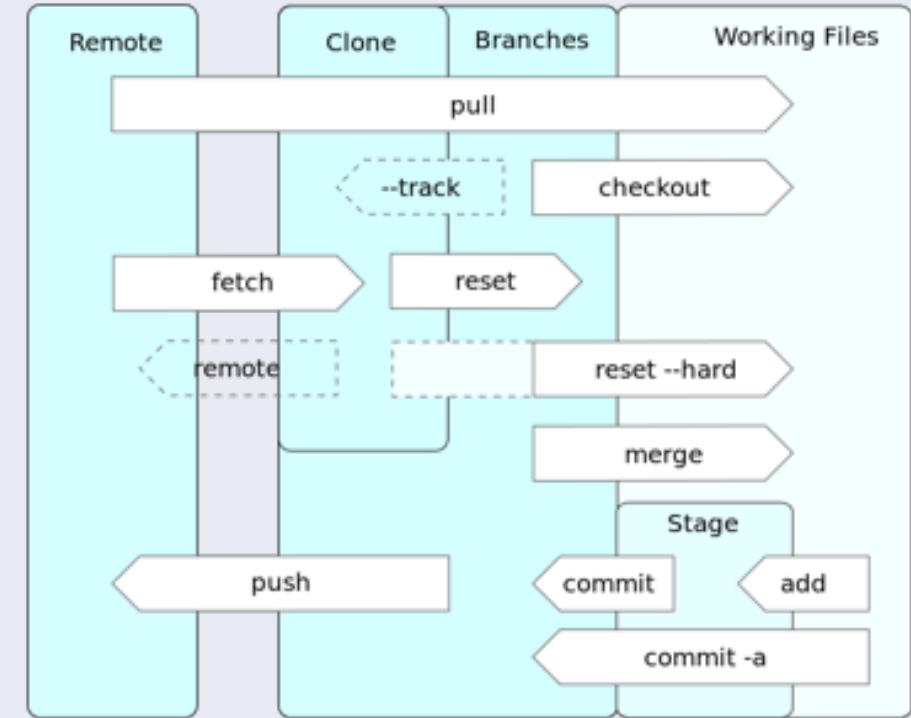
After cloning, before each add, commit, push session

- 
1. git pull origin main
 2. Edit files
 3. git add .
 4. git commit -m "message"
 5. git push origin main



Git Workflow

- **Fork**
 - Only do this **once** per project on Gitlab
 - Creates a personal (separate) copy of repo on server
- **Clone**
 - Only do this **once** per machine
 - Creates a copy of repo on your local machine
- While project is active
 - **Pull** latest changes as needed (e.g. partner changes or edited on different machine)
 - While actively coding during day
 - Edit, compile, and test
 - **Add** new or specific files to staging
 - **Commit**
 - Only do this via command line or local GUI
 - **Never** commit via Gitlab web interface!
 - Use descriptive commit messages (e.g. “added the hello world print command”)
 - **Push** local commits to remote server



If two people change same files between pulls, then you will need to merge

Try to avoid this by coordinating and starting with git pull

- Stopped here Tuesday 24-Aug-21

Organization Suggestions

- Create a single directory (or “folder”) on your laptop hard drive to hold all of your work for this class
 - This is your “course” directory
- Clone each git repository (project) into the course directory (folder)
 - Do NOT clone a project into another git repo
 - It is possible, and useful under some circumstances, ***but don't do it here!***
 - **NEVER** clone a project into another git repo
 - Always start from “course” directory when doing the clone command
 - Make sure you do **NOT** clone a new project into an existing git repo
- Open a separate PyCharm project that points to each repo
 - Do **NOT** put working files in one workspace and then manually copy files into repo
- PyCharm has git built in
 - **Learn to use the command line anyway**
 - I will not teach PyCharm Git interface
 - I will test you on command line interface
- Commit and push frequently
 - Anything done on PCSE computers is lost after you log out
 - Assume the power will go out or your computer may crash at any moment. Back up your work often!
 - At a minimum, commit and push at the end of **every** practice session
- Log out of PCSE computers here and in Hunter Creech after using
 - Do not expose your work for others!
 - You are responsible if someone accesses your files and submits them as their work



If I don't see a commit, then I presume you were not working

Gitlab: Verify Changes on Remote

The screenshot shows a GitLab project page for 'cpec250-p0-hello-world-s19'. The main content area displays a commit history with one commit highlighted by a blue oval. The commit message is 'add hello_world solution' and it was authored by David Conner 4 months ago. To the right of the commit message, a large red circle with a diagonal slash (prohibited symbol) is overlaid on the text 'Use Command line! Never upload files using web interface'. A blue arrow points from this text towards a 'Continuous Integration' button at the bottom right of the commit details. The top of the page shows standard GitLab navigation and project statistics.

Use Command line!
Never upload files using web interface

Continuous Integration

Clicking on commit name shows differences

The screenshot shows a Git commit history interface. On the left, a sidebar lists repository navigation options like 'Repository', 'Branches', 'Tags', etc. A large blue arrow points from this sidebar to the commit names in the main list. The main area displays a commit titled 'add hello_world solution' with a green checkmark icon. It includes details like 'parent aea60431 Master', 'No related merge requests found', and a failed pipeline message. Below this, a 'Changes' section shows a diff between two commits. The diff highlights changes in 'src/hello_world.py'. The left side of the diff shows the original code, and the right side shows the modified code with changes highlighted in green. The modified code includes comments explaining the creation of a 'say_hello' method and its use.

```
diff --git a/src/hello_world.py b/src/hello_world.py
--- a/src/hello_world.py
+++ b/src/hello_world.py
@@ -1,5 +1,11 @@
 #!/usr/bin/env python3
 #!/usr/bin/python3.6
+
+# Create a method called say_hello that returns "Hello World!"
+
+# Then print by calling method
+
+def say_hello():
+    """
+    Specify a string
+    return string
+    """
+
+    return "Hello World!"
```

Continuous Integration

Gitlab Continuous Integration (Testing)

The screenshot shows a GitLab pipeline interface. On the left, a code editor displays a .gitlab-ci.yml file with three stages: hello_world, say_it, and webcat-upload. The hello_world stage has passed, while the say_it stage has failed. The webcat-upload stage has also failed. A blue oval highlights the failed webcat-upload stage. In the center, a large card for the 'hello_world' solution shows a failed pipeline with two jobs. One job, 'f856c4a4', is shown in progress. Below the pipeline summary, there are tabs for Pipeline, Jobs (3), Failed Jobs (2), and Test. Under the Test tab, the 'hello_world' job is marked as passed (green checkmark), while the 'say_it' job is marked as failed (red X). At the bottom, there's a sidebar with links for Operations, Wiki, Snippets, and Settings.

```
1 hello_world:
2     variables:
3         UNITTEST: "tests/test_hello_world.py"
4         script:
5             - PYTHONPATH=. python3 $UNITTEST
6
7 say_it:
8     variables:
9         UNITTEST: "tests/test_say_it.py"
10        script:
11            - PYTHONPATH=. python3 $UNITTEST
12
13 webcat-upload:
14    variables:
15        WEBCATCOURSE: "CPSC 250"
16        WEBCATTARGET: "p0-hello-world-s19"
17        script:
18            - 'if [[ -z "$WCUSER" ]]; then'
19            - '    echo "WCUSER value is not set"'
20            - '    exit 1'
21            - 'else'
22            - '    echo "$WCUSER" > login.txt'
23            - 'fi'
24            - 'if [[ -z "$WCPASS" ]]; then'
25            - '    echo "WCPASS value is not set"'
26            - '    exit 1'
27            - 'else'
28            - '    echo "$WCPASS" >> login.txt'
29            - 'fi'
30            - 'echo "using WebCat login credentials"'
31            - 'cat login.txt'
32            - 'python3 webcat-submitter.py "$WEBCATCOURSE/$WEBCATTARGET"'
```

The screenshot shows the terminal output for the failed webcat-upload stage. It starts with the runner configuration and repository cloning. It then checks out the master branch and runs the test_say_it script. The test fails with an assertion error. The error message indicates that the output of say_it.py was expected to be an empty string but was actually 'Hello World!'. The test suite ran 2 tests in 0.033 seconds and failed with 1 error. The final message shows an exit status of 1.

```
Running with gitlab-runner 11.2.0 (35e8515d)
on default 3a8486c5
Using Shell executor...
Running on runner...
Cloning repository...
Cloning into '/home/gitlab-runner/builds/3a8486c5/1/david.conner-cpsc250-s19/cpsc250-p0-hello-world-s19'...
Checking out f856c4a4 as master...
Skipping Git submodules setup
$ PYTHONPATH=. python3 $UNITTEST
.F
FAIL: test_say_it (__main__.TestSayIt)

Traceback (most recent call last):
  File "tests/test_say_it.py", line 33, in test_say_it
    self.assertEqual(expected, output_str,msg="say_it.py output is incorrect: Expected()")
AssertionError: 'Hello World!' != ''
- Hello World!
+
: say_it.py output is incorrect: Expected('Hello World!') Actual ()
```

```
Ran 2 tests in 0.033s

FAILED (failures=1)
ERROR: Job failed: exit status 1
```

Automatic Upload to WebCat for Grading

The screenshot shows a GitLab interface. On the left, a sidebar lists various project management sections: Project, Repository, Issues, Merge Requests, CI / CD, Pipelines, Jobs (which is selected), Schedules, Charts, Operations, Wiki, Snippets, and Settings. In the main area, a group named "cpsc250-p0-hello-world-s19" is selected, highlighted by a blue oval. A specific job within this group is shown, with its name "david.conner-cpsc250-s19" also circled in blue. The job status is "failed". The log output for the job shows the following text:

```
Running with gitlab-runner 11.2.8 (35e8515d)
on default 3a0406c5
Using Shell executor...
Running on runner...
Cloning repository...
Cloning into '/home/gitlab-runner/builds/3a0406c5/2/david.conner-cpsc250-p0-hello-world-s19'...
Checking out f056c4a4 as master...
Skipping Git submodules setup
$ if [[ -z "$WCUSER" ]]; then
$ echo "WCUSER value is not set";
WCUSER value is not set
$ exit 1;
ERROR: Job failed: exit status 1
```

NOTE:
WebCat password (WCPASS) can NOT have a \$ character!

Must set our WebCat Username (WCUSER) and password (WCPASS) variables

We need to do this ONCE for group (not individual project); then all future projects will work

Click on GROUP name in upper left

Modify GROUP! Setting

The screenshot shows a user interface for managing a project. On the left, there's a sidebar with links: Overview, Details (which is selected), Activity, Issues, Merge Requests, Members, and Settings. The 'Settings' link is circled with a blue oval. A large blue arrow points from this circled link to a modal window titled 'Settings'. This modal contains tabs for General, Projects, CI / CD, and another 'CI / CD' tab which is currently active. Below the tabs, the project name 'david.conner-cpsc250-s19' is displayed, along with a 'Global' dropdown and a 'New project' button. At the bottom of the modal, there are buttons for 'Last created' and a timestamp '8 hours ago'.

david.conner-cpsc250-s19

Overview

Details

Activity

Issues

Merge Requests

Members

Settings

Subgroups and projects Shared projects Archived projects

david.conner-cpsc250-s19

Global

New project

Last created

8 hours ago

Set variables, save, click group name

The screenshot shows a user interface for managing CI/CD variables. On the left, a sidebar lists project navigation options: Overview, Issues, Merge Requests, Members, Settings (which is selected), General, Projects, and CI / CD. The main content area is titled "Variables" and contains a brief description of what variables are used for. Below this, there are two variable entries:

Variable Key	Value	Protected
WCUUSER	captain.chris.18	Protected (switch off)
WCPASS	password	Protected (switch off)

Below the variables, there are input fields for "Input variable key" and "Input variable value". At the bottom of the variable section, there are two buttons: "Save variables" (highlighted with a blue oval) and "Hide values". A yellow arrow points from the "password" input field towards a callout bubble.

**Only single line of text!
Do not hit Enter/Return key**

NOTE:

WebCat password (WCPASS) can NOT have a \$ character!

Select project name

The screenshot shows a GitLab project page for 'david.conner-cpsc250-s19'. The left sidebar contains navigation links: Overview (selected), Details, Activity, Issues (0), Merge Requests (0), Members, and Settings. The main area displays the project name 'david.conner-cpsc250-s19' with a 'Global' dropdown. Below the project name is a search bar labeled 'Search by name' and a 'New project' button. A horizontal menu at the bottom includes 'Subgroups and projects' (selected), 'Shared projects', and 'Archived projects'. A 'Last created' dropdown shows a recent project: 'cpsc250-p0-hello-world-s19' (8 hours ago). A blue oval highlights this specific project entry.

Gitlab: Verify Changes on Remote

The screenshot shows a GitLab project page for 'cpsc250-p0-hello-world-s19'. The left sidebar lists various project sections: Project, Details, Activity, Cycle Analytics, Repository, Issues, Merge Requests, CI / CD, Operations, Wiki, Snippets, and Settings. The 'Merge Requests' section is currently selected, indicated by a blue underline. A large blue arrow points from the text 'Continuous Integration' towards the CI / CD section. In the main content area, there is a merge request titled 'add hello_world solution' by 'David Conner' (authored 6 hours ago). Below the merge request, there is a table with three rows:

Name	Last commit	Last update
img	initial release	2 days ago
src	add hello_world solution	6 hours ago

At the top right of the page, there is a red circle with a white 'X' icon, which is circled in blue. The URL 'https://gitlab.pcs.cn' is visible at the top of the page.

CI Pipeline

add hello_world solution

parent [aaa68431](#) ↗ main

✖ Pipeline #56364 failed with stage ✖ in 2 seconds

Changes 1 Pipelines 1

Status	Pipeline	Commit	Stages	Time	Created
✖	#56364 by	→ f056c4a4	✖ add hello_world solut...	00:00:02	31 minutes ago

add hello_world solution

3 jobs from main in 2 seconds (queued for 6 seconds)

f056c4a4 ... ⚡

Pipeline Jobs 3 Failed Jobs 2

Test

	Stage	Status
	hello_world	Success
✖	say_it	Failure
✖	webcat-upload	Failure

Retry job

Test

	Stage	Status
	hello_world	Success
✖	say_it	Failure
	webcat-upload	Success

If WCUSER/PASS set correctly



Automatic WebCat Submission

The screenshot shows a web browser with two tabs open. The left tab is a GitLab pipeline build log for a project named "P0_hello_world". It displays the command-line output of the build process, which includes running Java code and generating a response file. The right tab is the "Web-CAT" login page, featuring a cat silhouette made of binary code. The URL for the Web-CAT page is <https://web-cat.cs.vt.edu>. A blue callout box on the Web-CAT page contains the text: "Only single line of text! Do not hit Enter/Return key". Another blue callout box on the GitLab pipeline settings page contains the text: "Settings-CI/CD on your personal cpsc250-s19 group". A large blue arrow points from the "Variables" section of the pipeline settings to the "Institution" dropdown in the Web-CAT login form.

<https://web-cat.cs.vt.edu>

Only single line of text!
Do not hit Enter/Return key

Settings-CI/CD on your
personal cpsc250-s19 **group**

WebCat Grader

The screenshot shows the 'Your Web-CAT Status' page. In the 'Assignments Accepting Submissions in Your Courses' section, there is a table for 'CPSC 250' with one row for 'ps-hello-world-119'. The table includes columns for 'File', 'Staff', 'Staff', 'AutoGrade', and 'Pass'. Three files are listed: 'src/_init_.py', 'src/hello_world.py', and 'src/say_it.py', all with a staff grade of 0.0 and an auto-grade of 0.0. A blue oval highlights the 'Latest Results' button at the bottom right of this section.

The screenshot shows the 'Your Assignment Submission Results' page for 'CPSC 250 (inst) ps-hello-world-119 try #1'. The 'Result Summary' section shows a 'Total Score' of 33.3/15.0. The 'Score Summary' bar is green for correctness (13.3/15.0) and red for testing (20.0/15.0). The 'Position in class:' bar shows the user is 1st. The 'Graphs' and 'File Details' sections are also visible. Below these, the 'Results From Running Your Instructor's Tests' section shows a summary of 15 cases (7 failures, 0 errors) with a 53.3% success rate. The test results list four failing cases:

- FAIL: test_say_it (tests.test_say_it.TestSayIt)
Hint: 'Hello World!' != ''
- FAIL: test_say_it2 (tests.test_say_it.TestSayIt)
Hint: 'Hello World!' != ''
- FAIL: test_say_it3 (tests.test_say_it.TestSayIt)
Hint: 'Hello World!' != ''
- FAIL: test_say_it4 (tests.test_say_it.TestSayIt)

This is the grade that counts!

**Do NOT manually upload files;
Use WCUSER/PASS and Gitlab CI!**

This is the class workflow

- We will use this every lecture
- Exams will be graded this way
- Its use is an expected part of your professional development
- Put Git on your resume!

Advanced Git : Branching

- When you work on your project in class, you are actually working on the **main** branch of your project.
 - `git branch`
 - List branches on your local machine
 - `git branch -a`
 - List all branches including those on remote(s)
 - `git checkout branch`
 - Switch over to a branch (commit first)
 - `git checkout -b ...`
 - Check out an old copy and create a branch to work on it
 - Easy branching is one of the most powerful features of git.
 - `git merge ...`
 - Merge changes from some other branch into this one



More information

- <http://rogerdudler.github.io/git-guide/>
 - This is the no “deep stuff” quick guide
- <https://www.git-tower.com/learn/git/ebook/en/command-line/basics/basic-workflow>
 - Basic beginner guide
- <http://mywiki.wooledge.org/BashGuide>
 - Command line info
- <https://www.atlassian.com/git/tutorials>
 - Atlassian toolchain is widely used in industry

Now back to coding

- See README for directions and finishing remaining W1 exercises
- Edit code in PyCharm
- Test and debug in PyCharm
- Git add, commit, push in terminal
- Verify on Gitlab AND WebCat
- Repeat

Learn it, Use it, Love it!

Unsolicited Excellent Advice

- Commit often
 - If you aren't committing, then you are not working
 - Commit for any significant change
 - Added new method or functionality
 - Fixed a particular bug
 - Tried something, even if a failure
 - I see your commit history
 - I know when you start a project (**START EARLY!**)
 - I get VERY suspicious of single large commits just before due date

- Push frequently
 - At least once every coding session
 - After significant commits
 - If it would be annoying to re-do, then push to remote (BE paranoid!)
 - “My computer crashed” is not a valid excuse.
 - At most, it should take you 30 minutes to recover from computer crash
 - This includes the 10 minutes to walk to Hunter Creech!

More problematic
for at home
remote work;
have a backup
plan if possible!