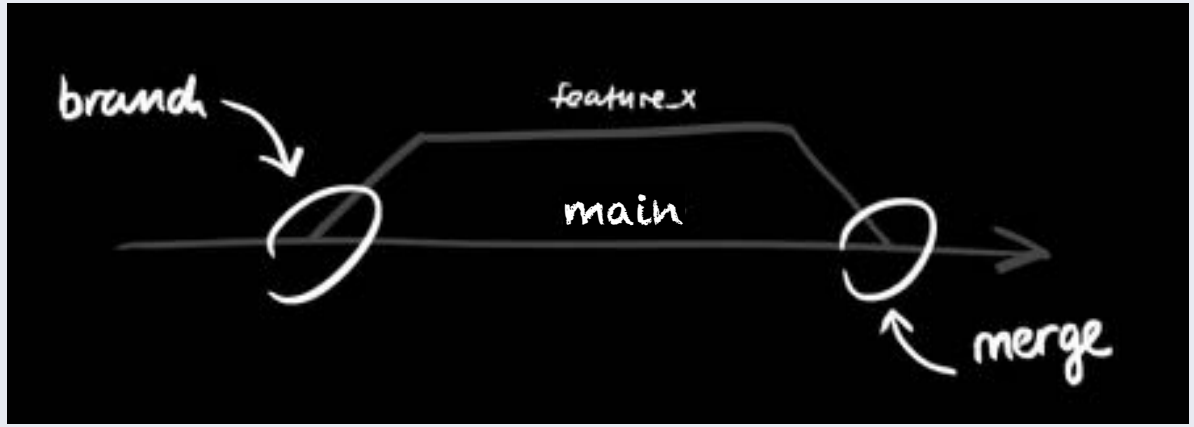
CPSC 250
Programming for Data Manipulation (CS2)

Course Tool Chain and Workflow

Christopher Newport University
Department of Physics, Computer Science & Engineering

Why bother?

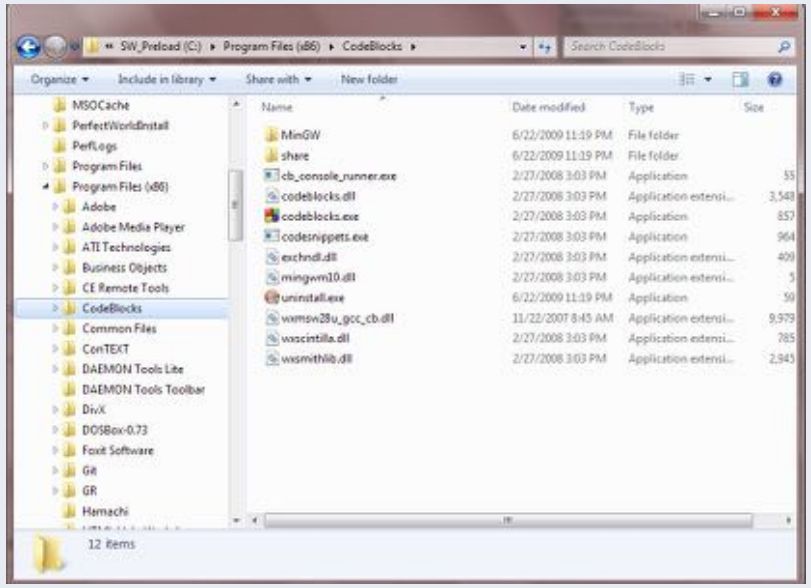
- Save hours of frustration
  - Easy backup and restore
  - Easy collaboration with others
  - Easy to experiment in a separate “branch”
  - Minimal loss of data if your computer crashes
  - Recover lost files or undo mistakes
  - Traceability (when did I introduce that bug?)



- 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)

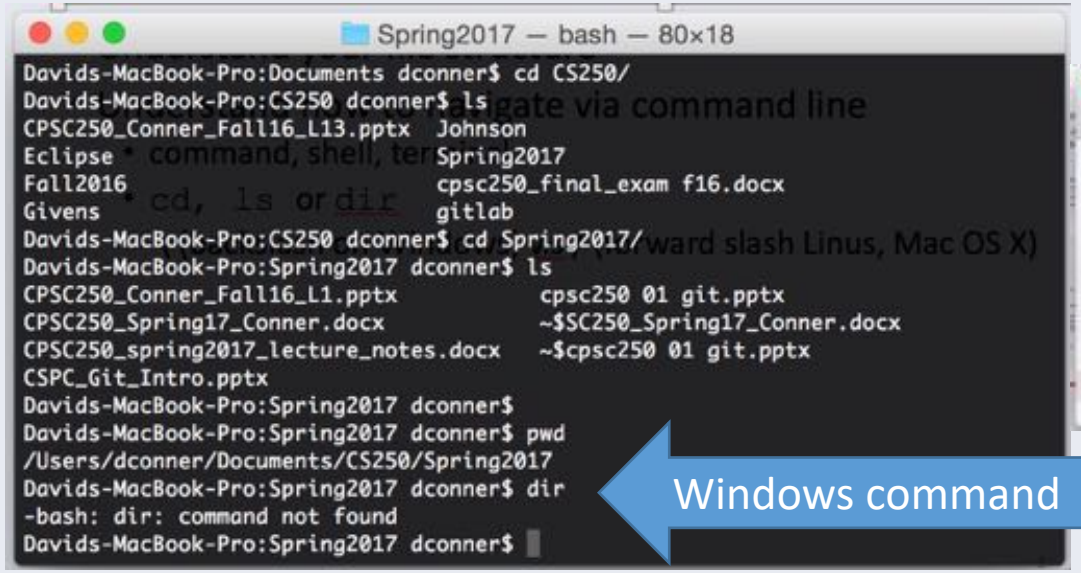
Required for this course!

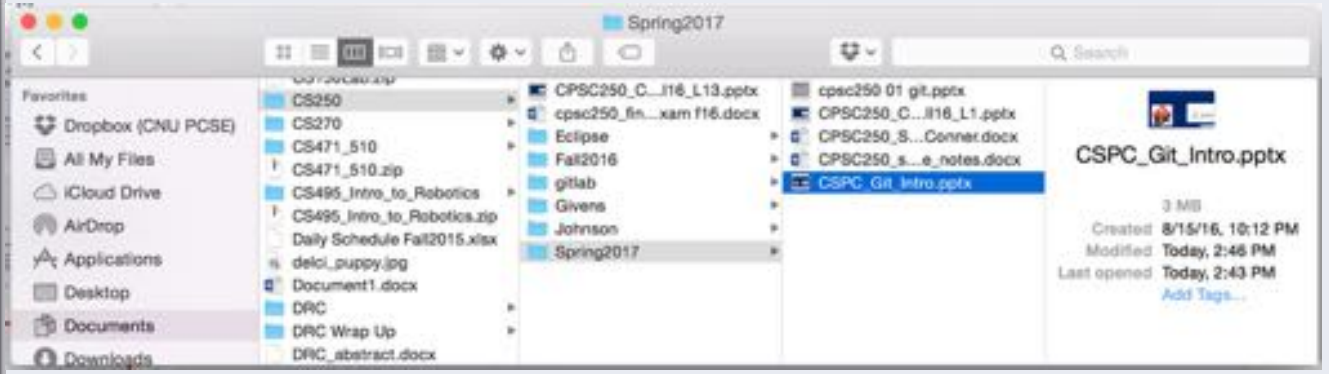
Prerequisites



- 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



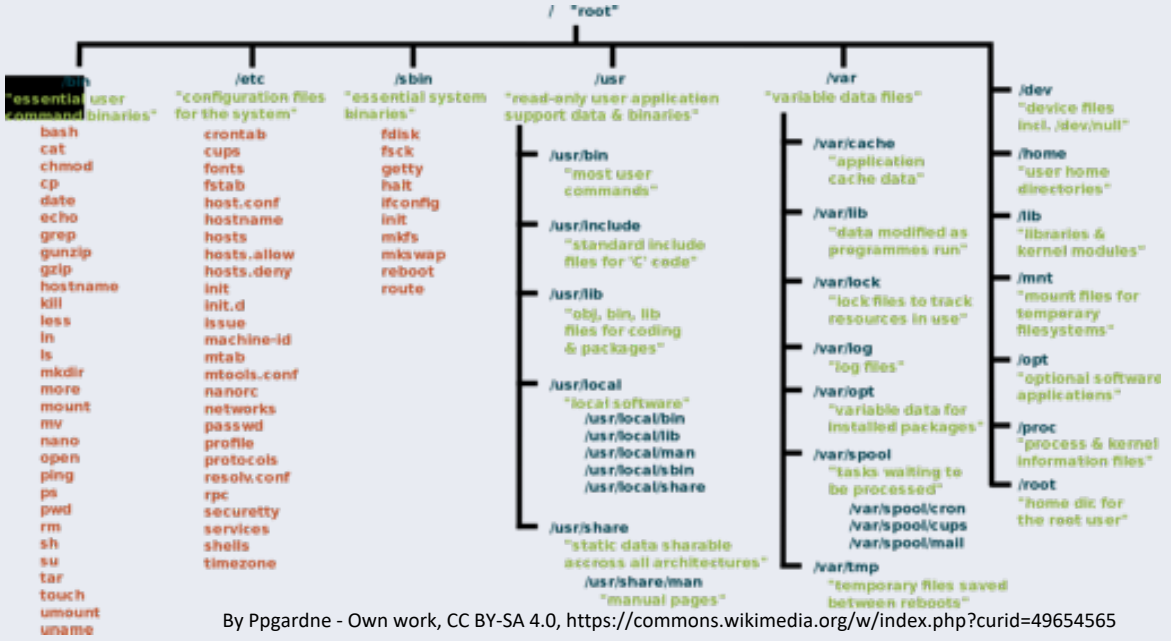


File/Directory/Folder Structure

This is focus of Lab #1
Learn it!

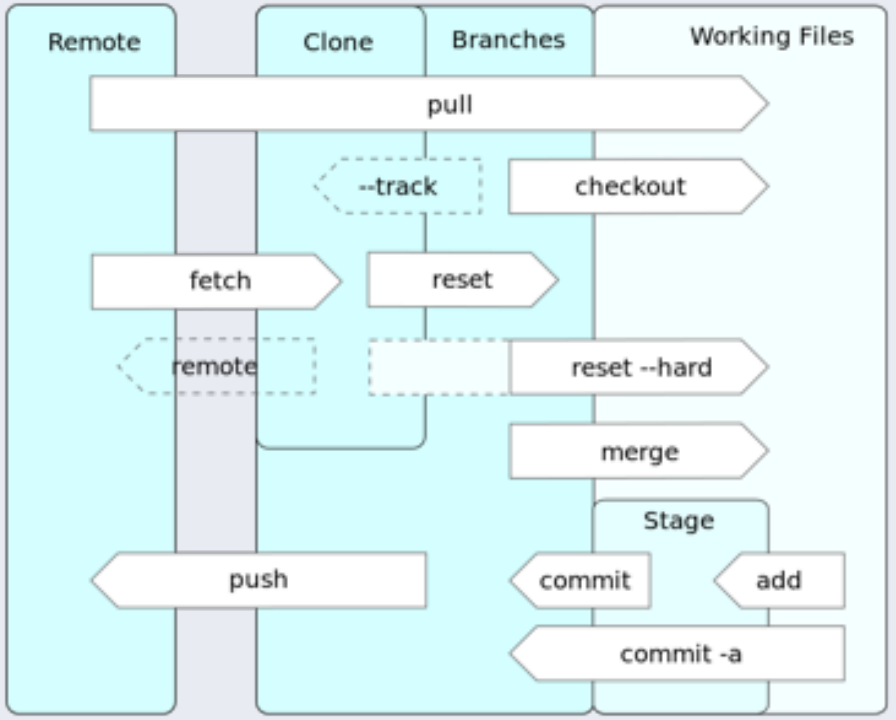
- http://www.cis.rit.edu/class/simg211/unixintro/Basics_of_File.html
- https://en.wikipedia.org/wiki/Directory_structure
- https://en.wikipedia.org/wiki/Unix_filesystem#Conventional_directory_layout

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



By Ppgardne - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=49654565

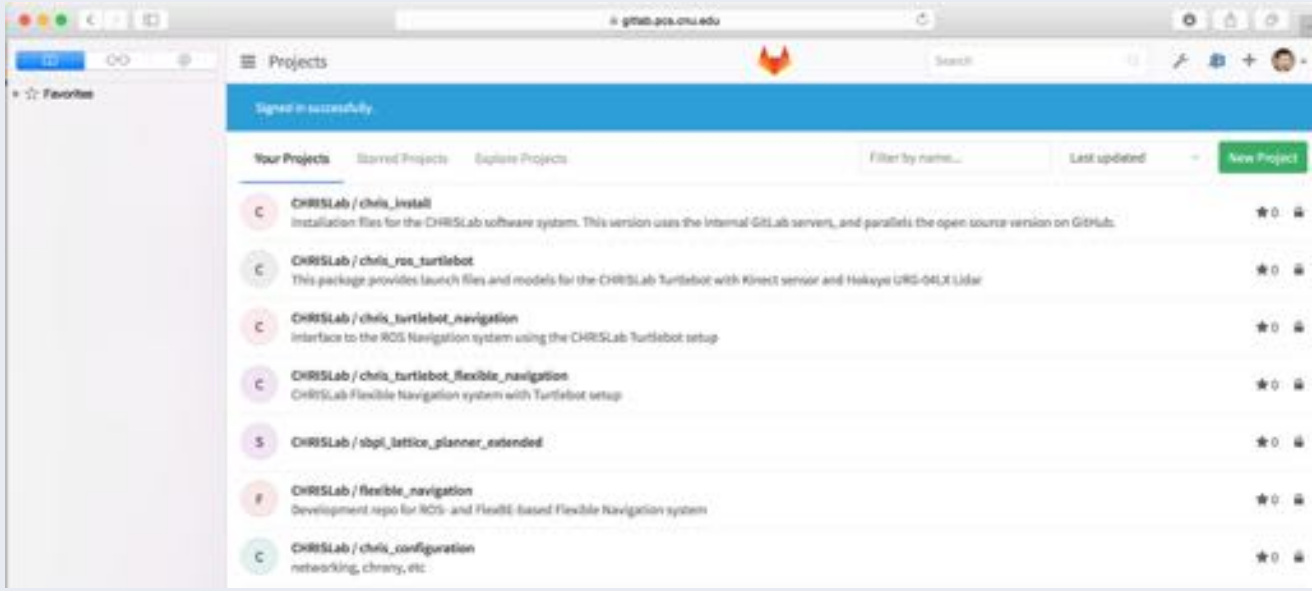
Git Basics



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

- 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

GitLab Remote Server

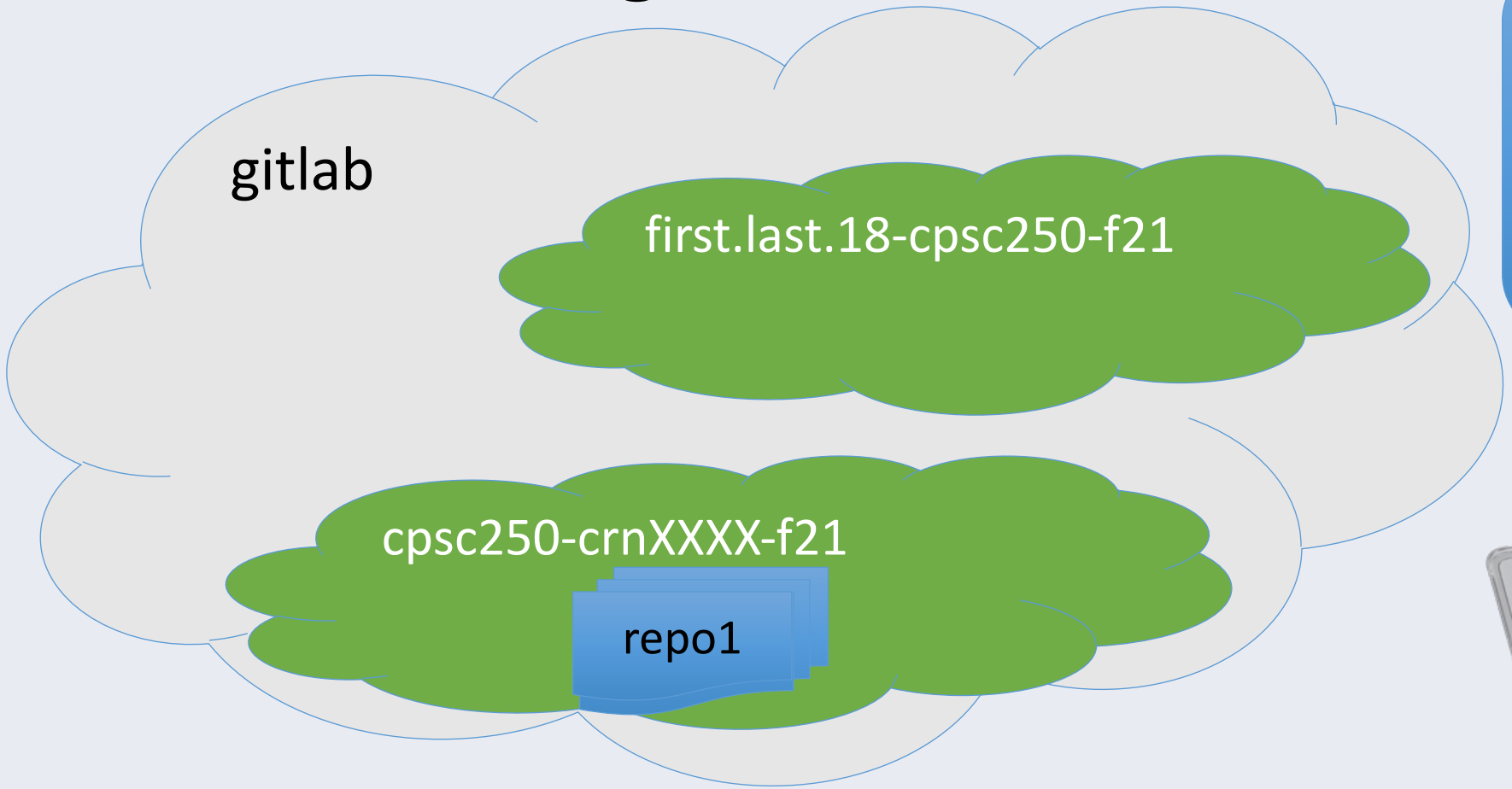


Change from s21 !

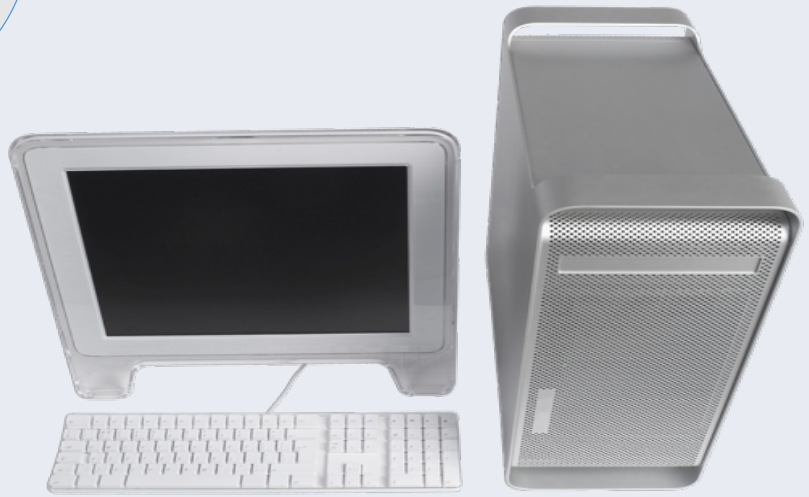
- Our common remote server
  - https://student-gitlab.pcs.cnu.edu
- Accounts already created using your email prefix (first.last.YY)
- You are owner of your personal `first.last.YY-cpsc250-f21` group
- You are also a member(reporter) of your instructor's group:
  - cpsc250-crnXXXX-f21
- Keep all your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

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

What does gitlab look like?

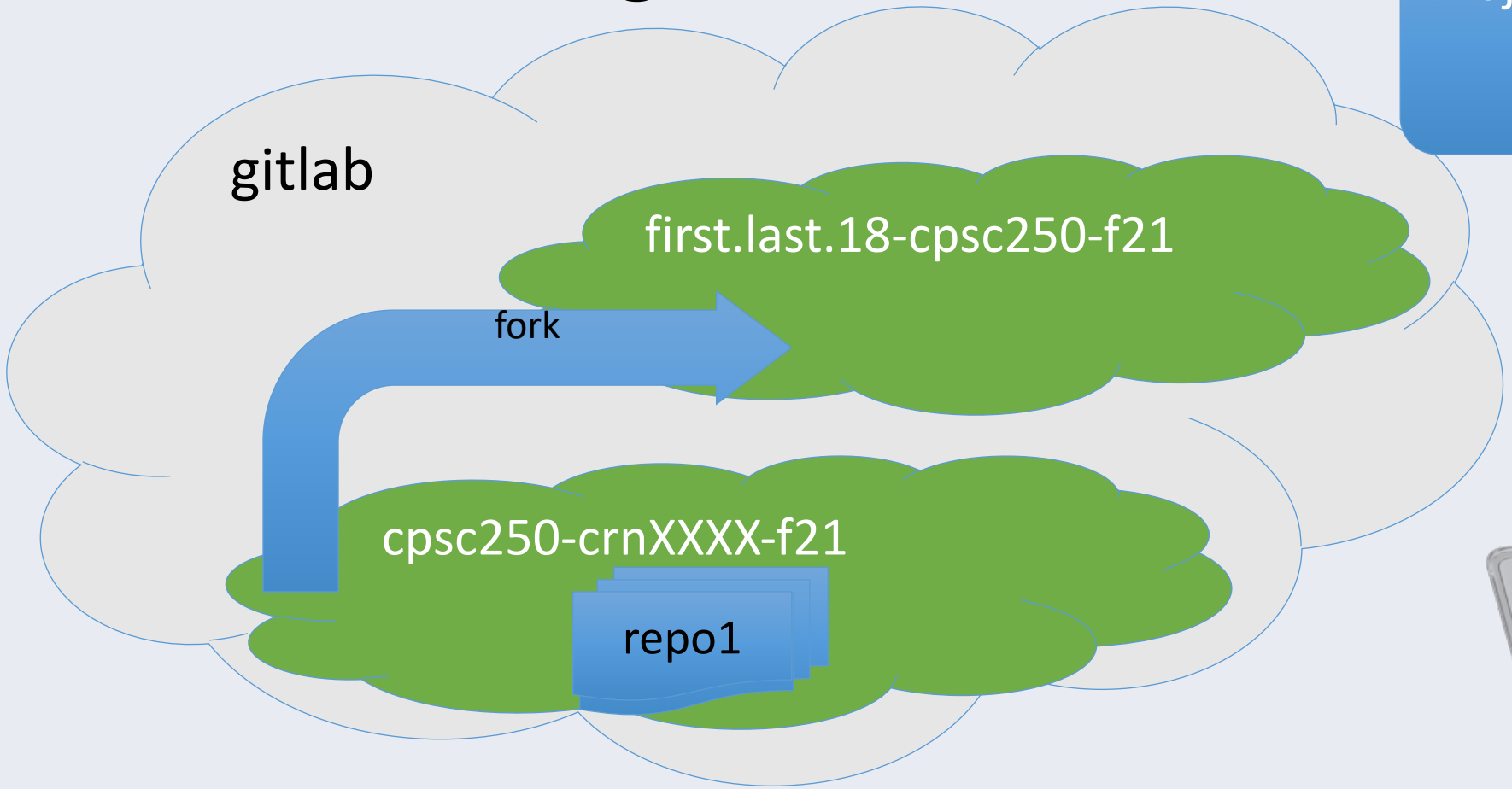


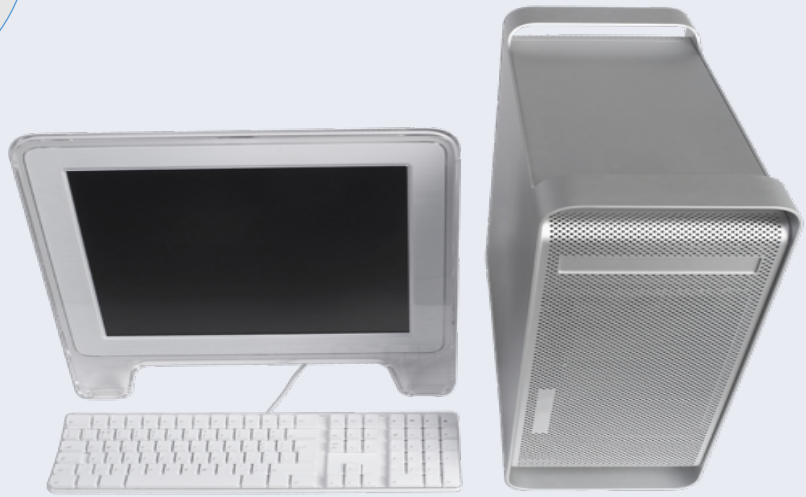
There is a group for you individually, and a group for the class. Groups can have projects in them.



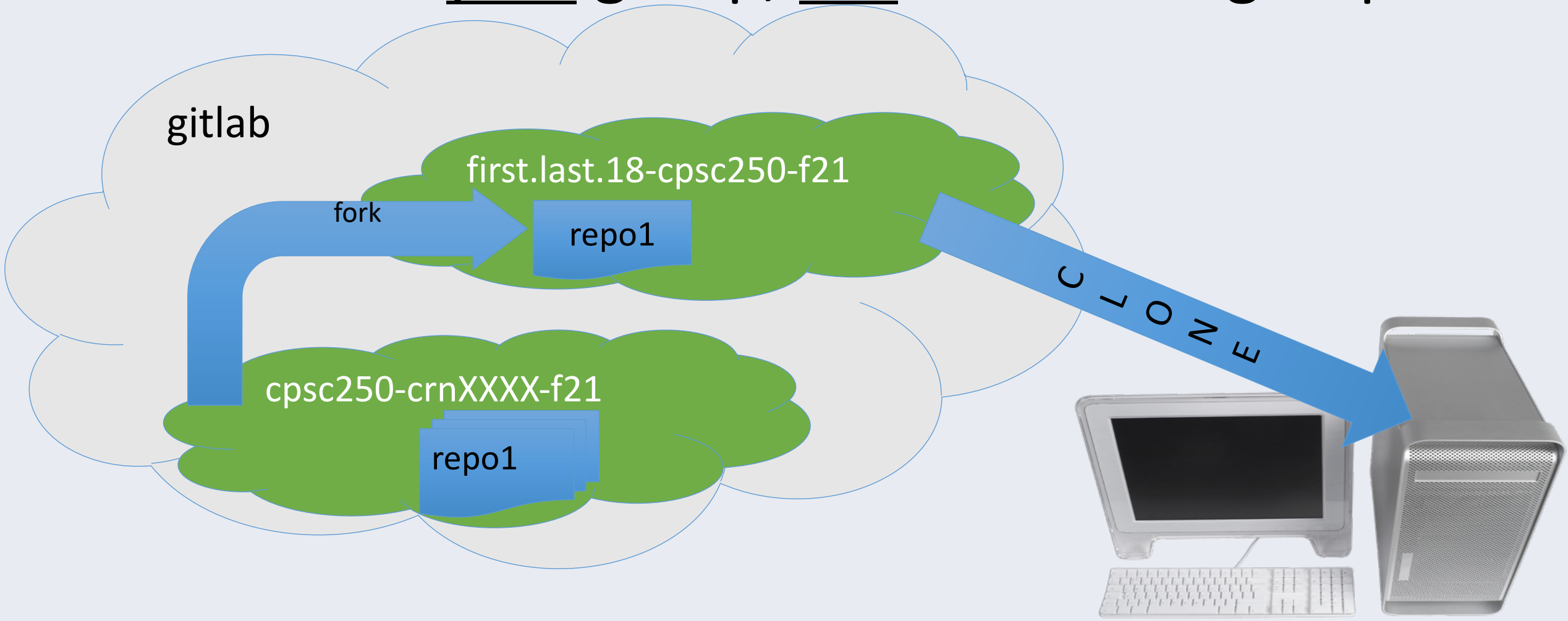
What does gitlab look like?

Projects are forked.



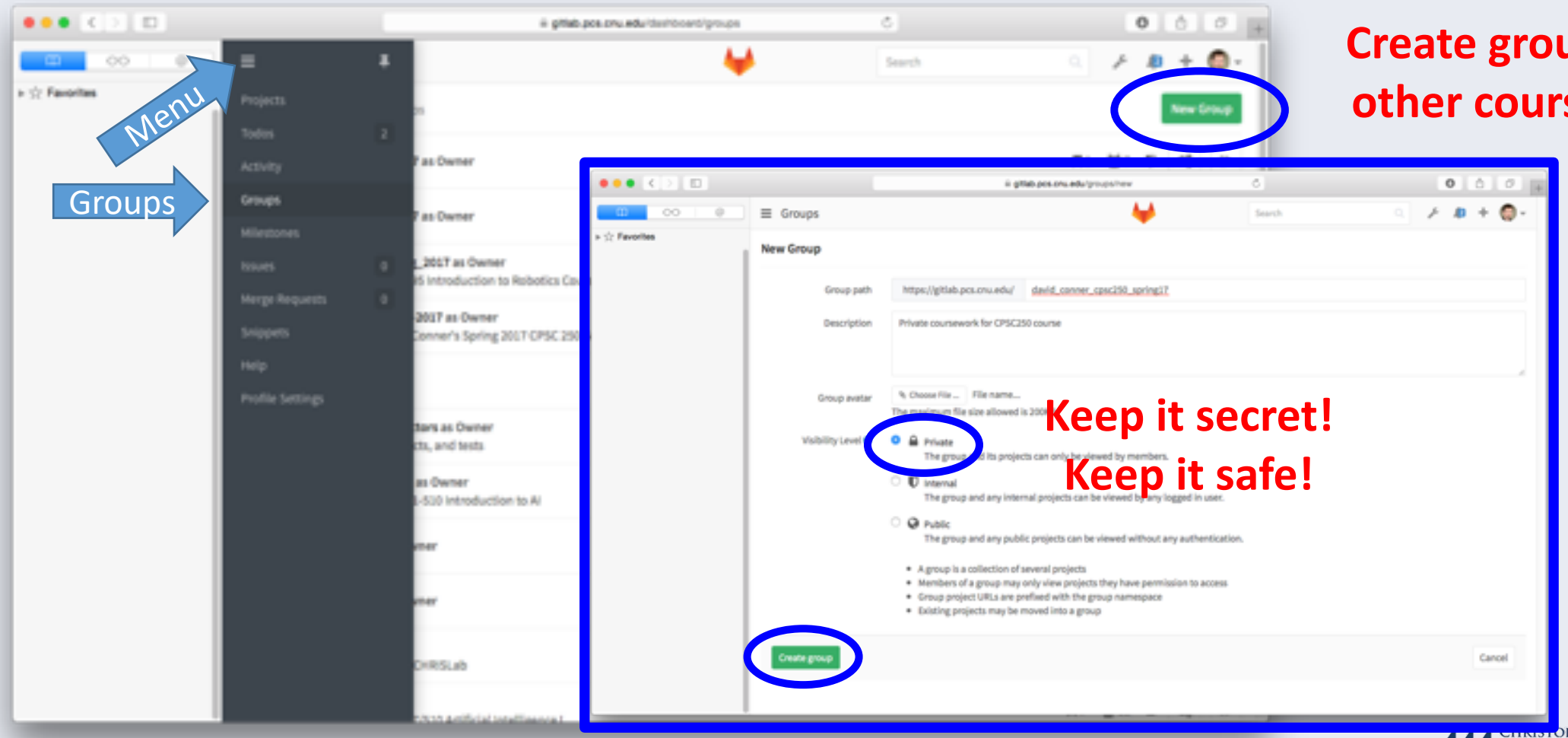


After you have forked, clone from your group, not the class's group

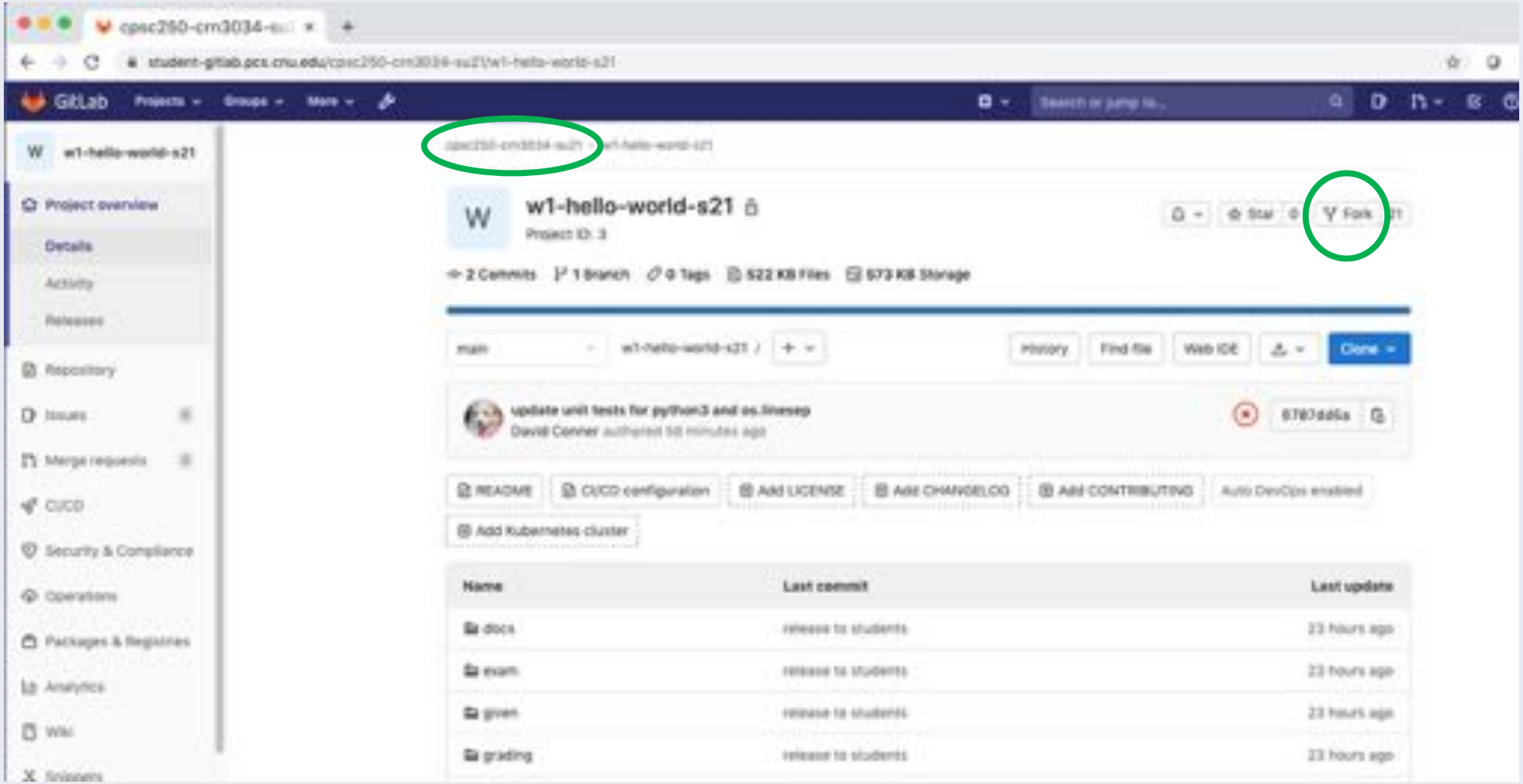


Use a Private Group to organize class projects

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

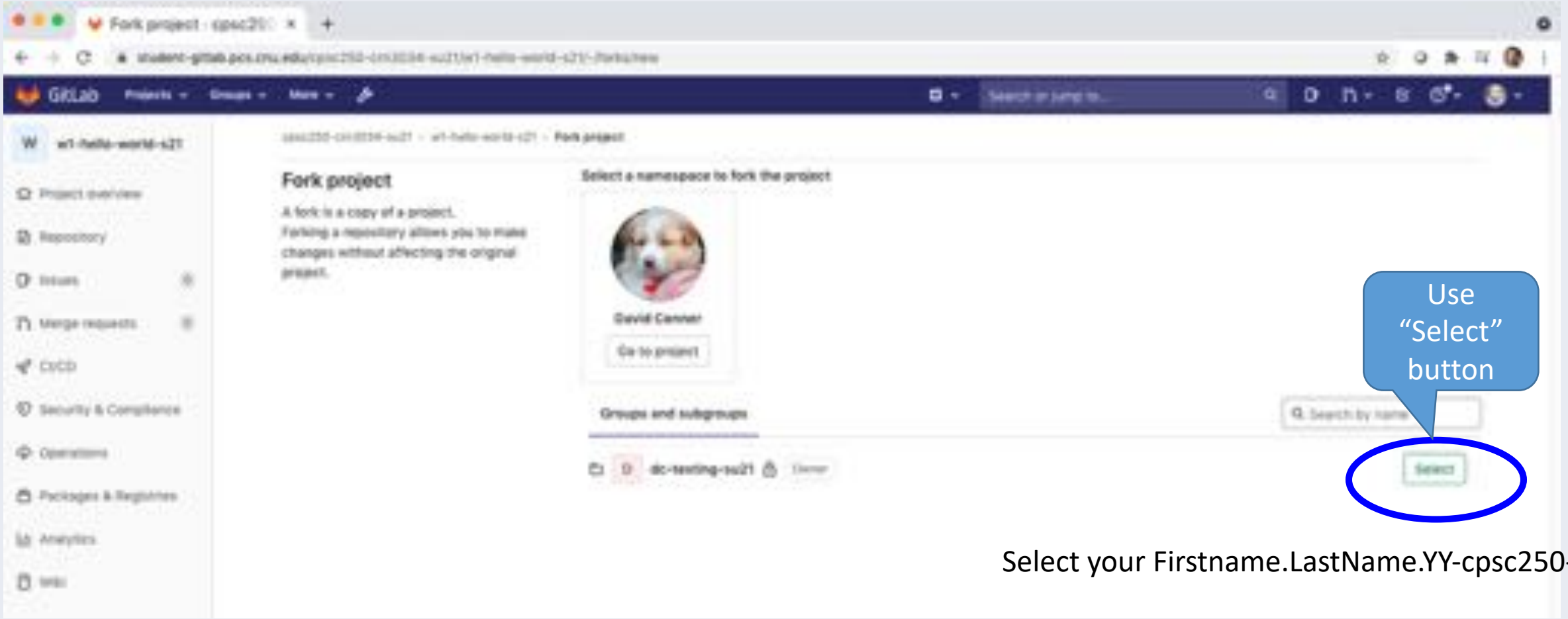


You want to Fork the Project from the GitLab course group to your Gitlab group via Browser



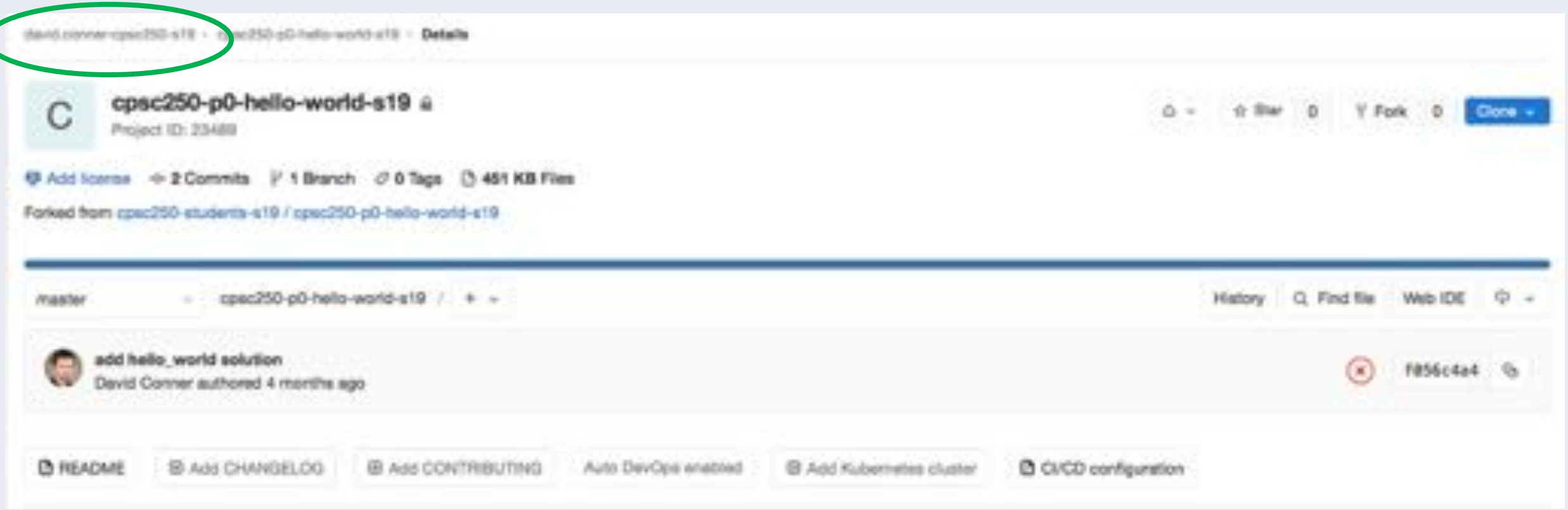
Select your personal first.last.yy-cpsc250-f21 group as target !

Select Project Fork to your **Course** Group

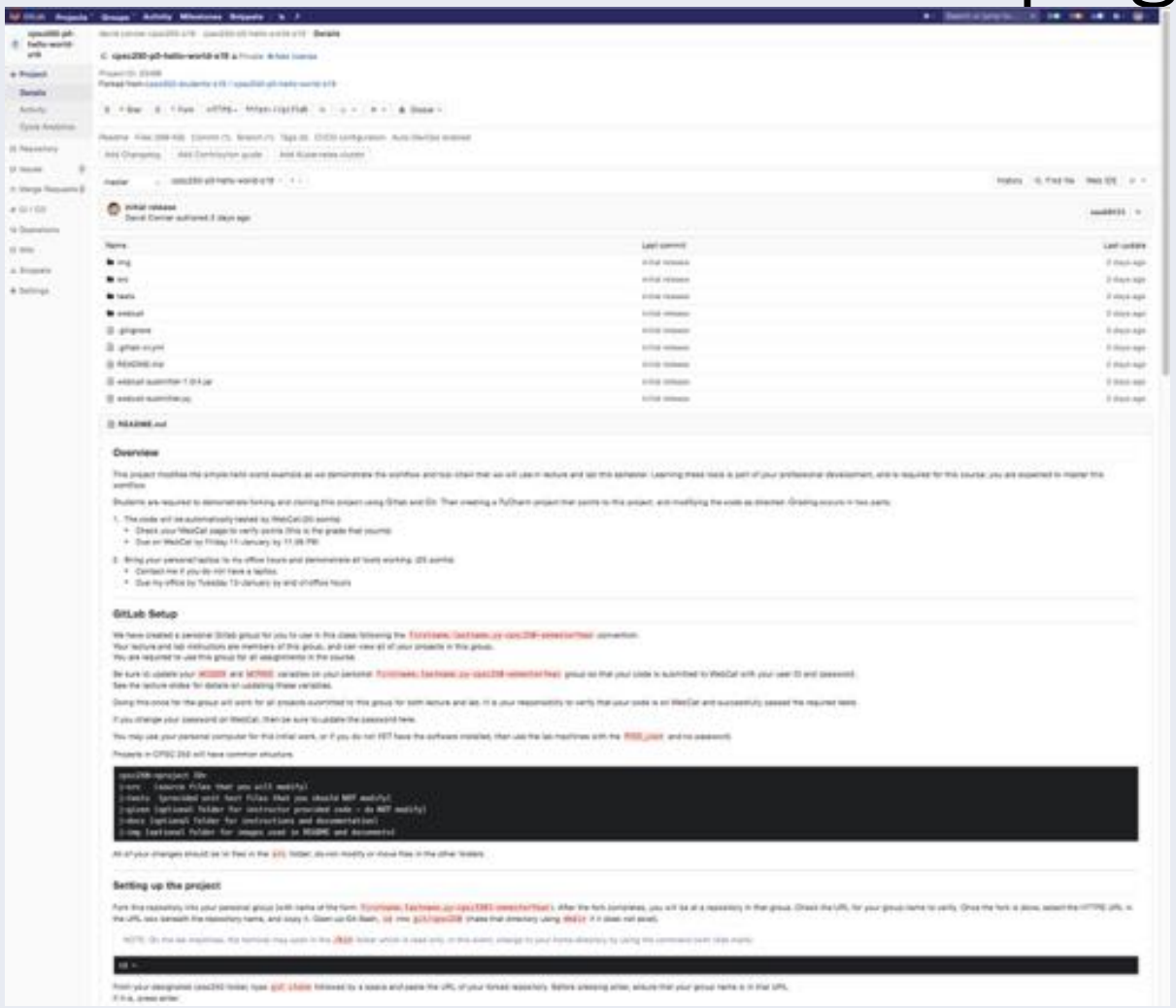


Select your Firstname.LastName.YY-cpsc250-f21

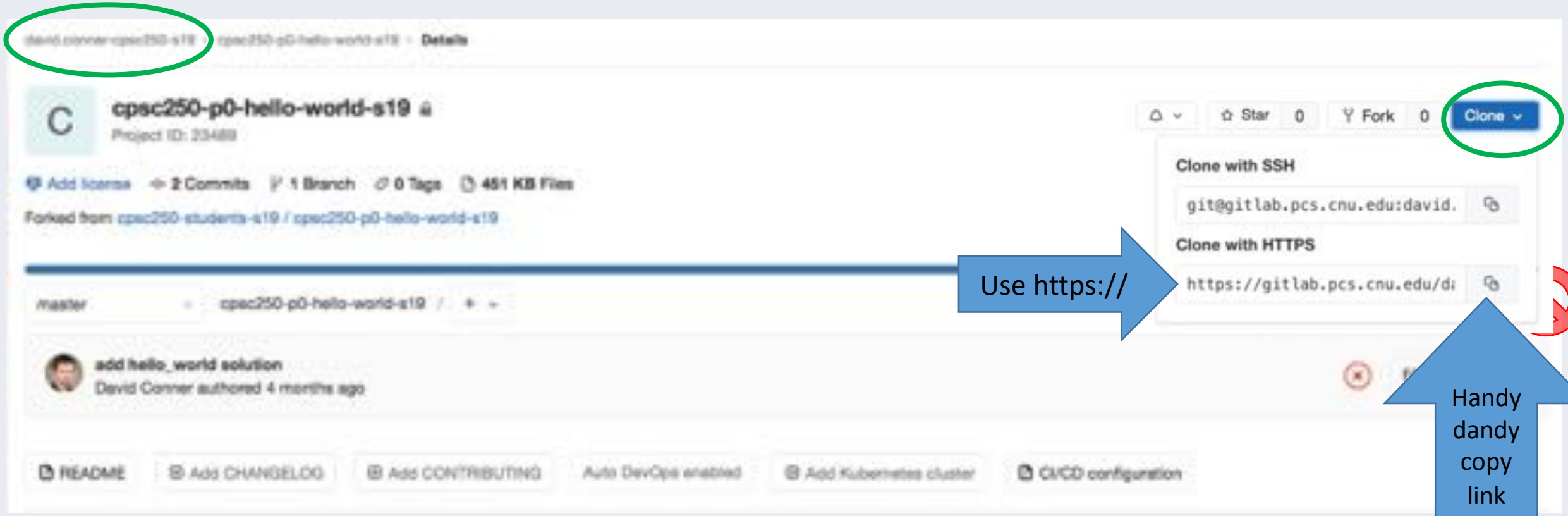
Work with your Private Fork!



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



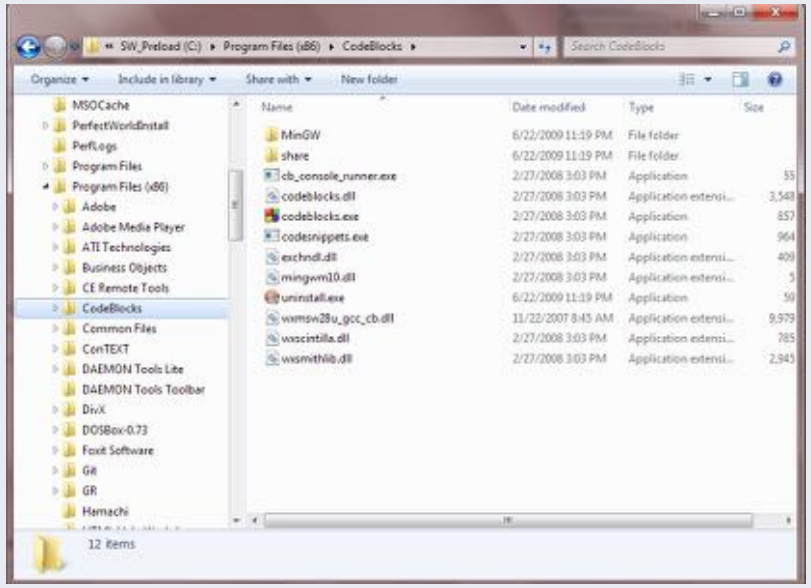
To clone, first copy link from **your private fork**



Use Command line and clone !

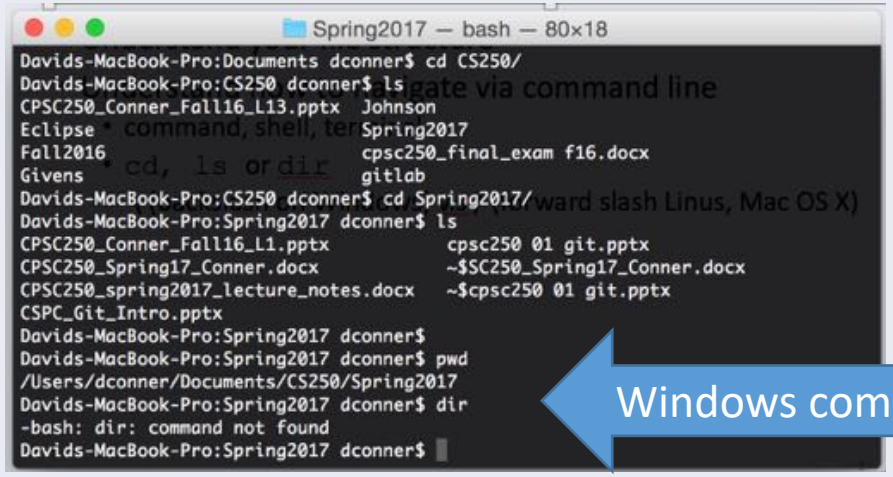
Never download or upload via web page

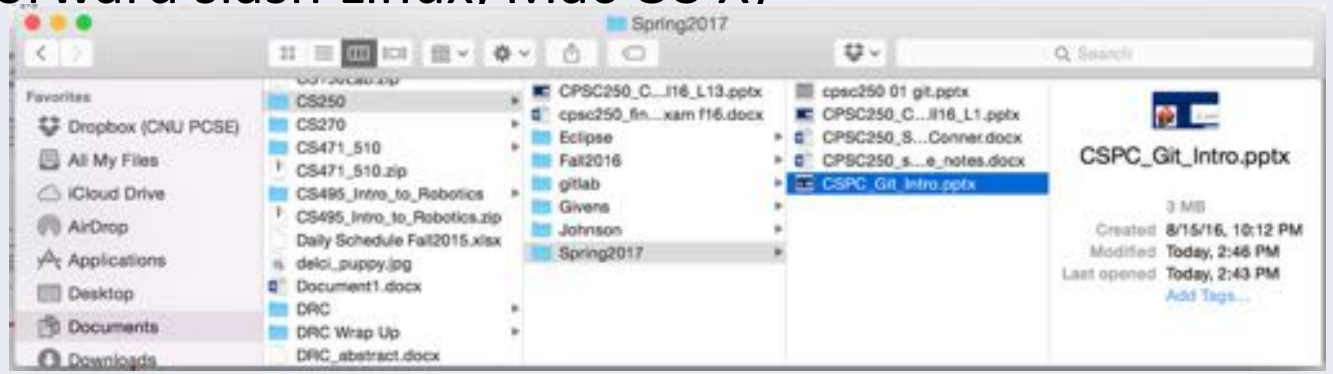
Now, go back to local machine Prerequisites



- 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)

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!

`git clone` command

Command Line Prompt

Copied URL (Typically Ctrl-v or Cmd-V to paste)

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

Note the quotation marks around name and helper

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

Two dashes (minus signs)

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

For Ubuntu (Linux)

- **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!

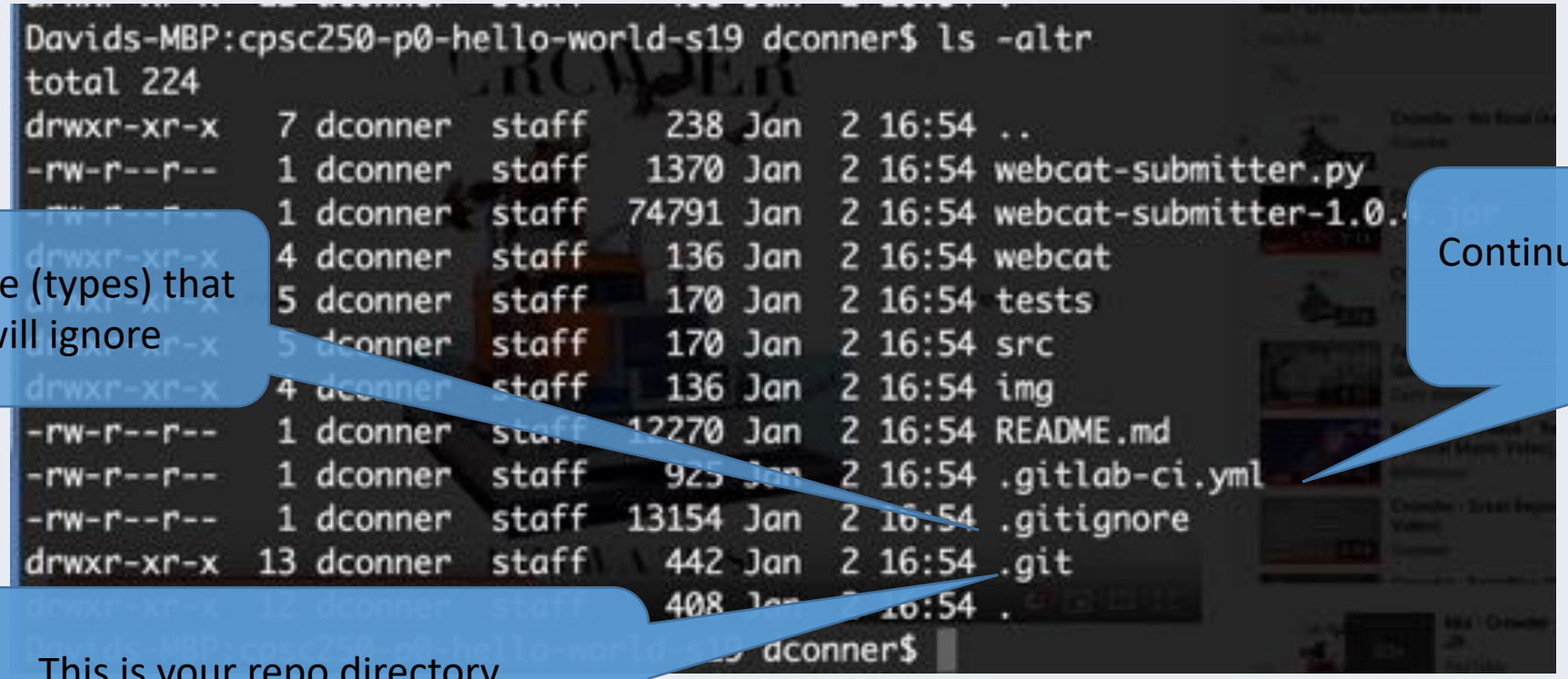
```
(base) davids-mbp:cs250-su21 dconner$ cd w1-hello-world-s21/
(base) davids-mbp:w1-hello-world-s21 dconner$ pwd
/Users/dconner/Documents/CS250/Su21/git/cs250-su21/w1-hello-world-s21
(base) davids-mbp:w1-hello-world-s21 dconner$ ls
README.md                       given                           src                             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)

a – all files including hidden files (start with .)
l - (lower case ell) list in long form
t – sort by last modified time
r – reverse the sort
use `man ls` to see all options

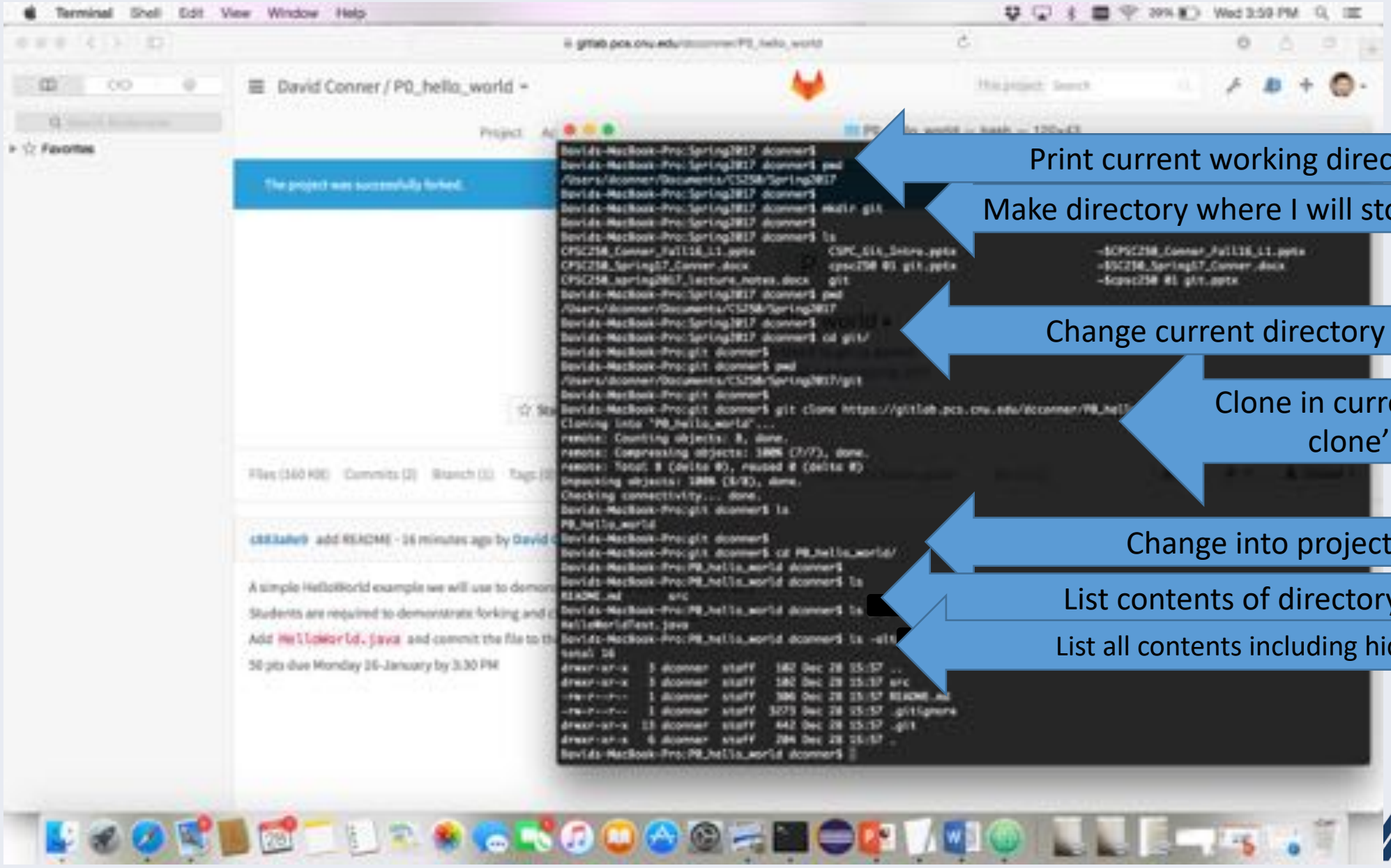


List of file (types) that git will ignore

Continuous Integration (CI) rules

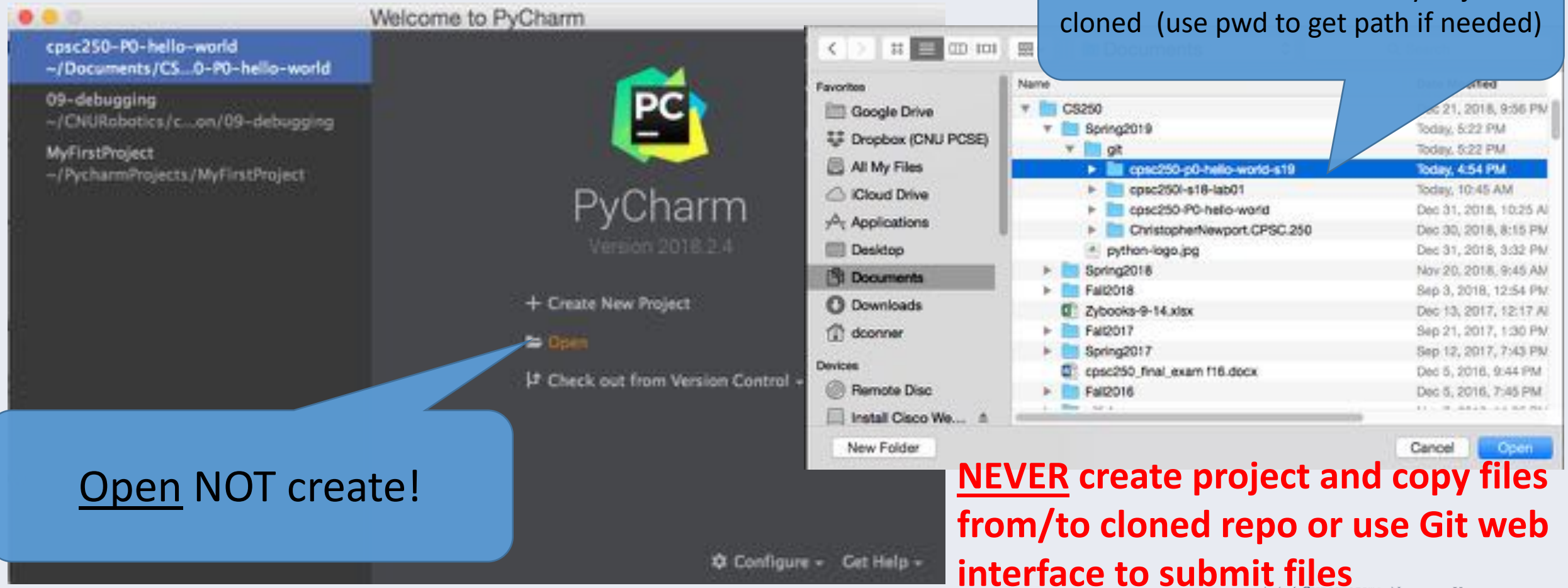
This is your repo directory – don't mess with it!

More (git)Bash commands



Ready to Begin Editing in PyCharm

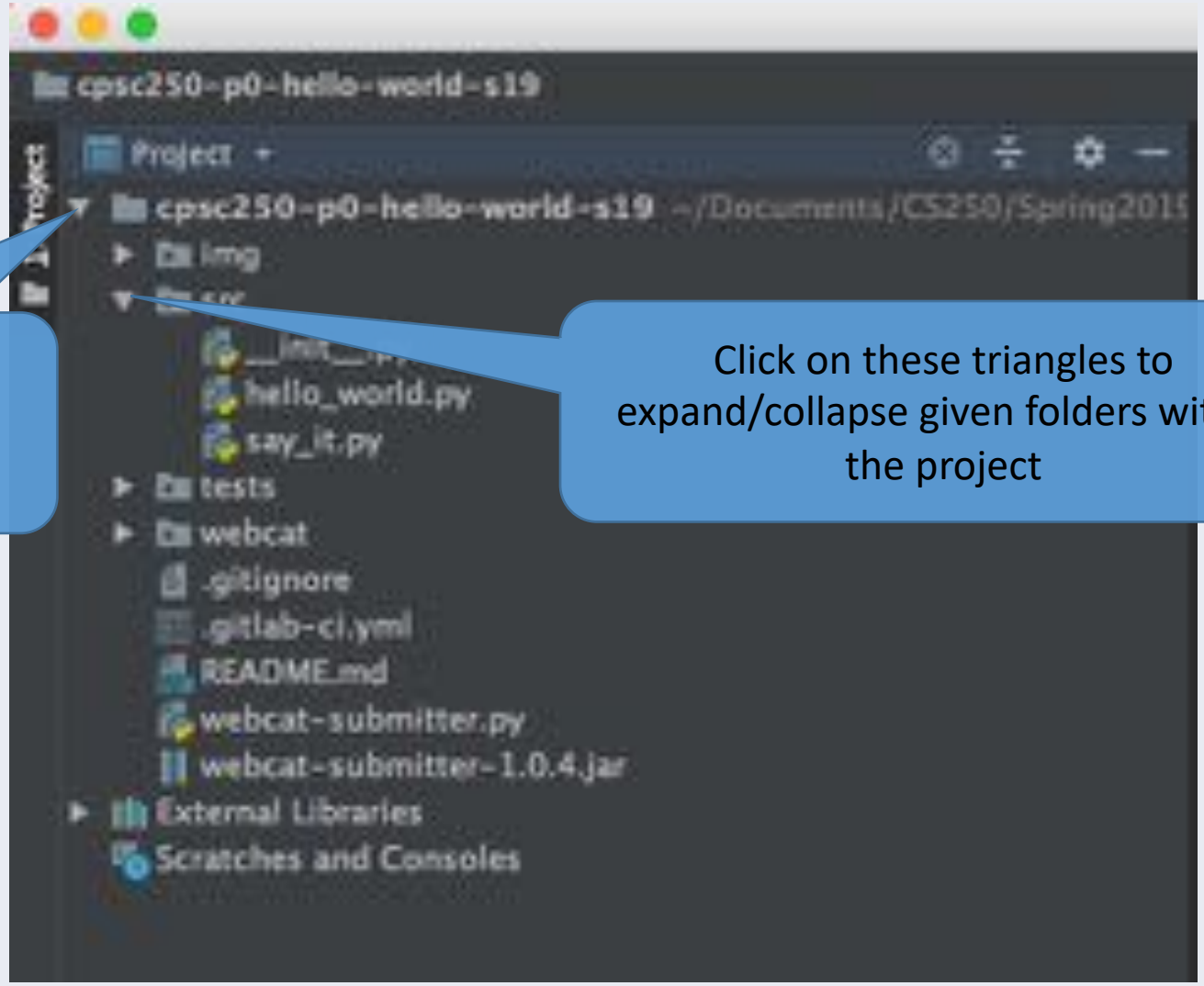
- After git directory is set up, and you have cloned your repo
- Now start PyCharm and **Open** existing project



Open NOT create!

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

PyCharm IDE



Click on this triangle to expand/collapse the project explorer view

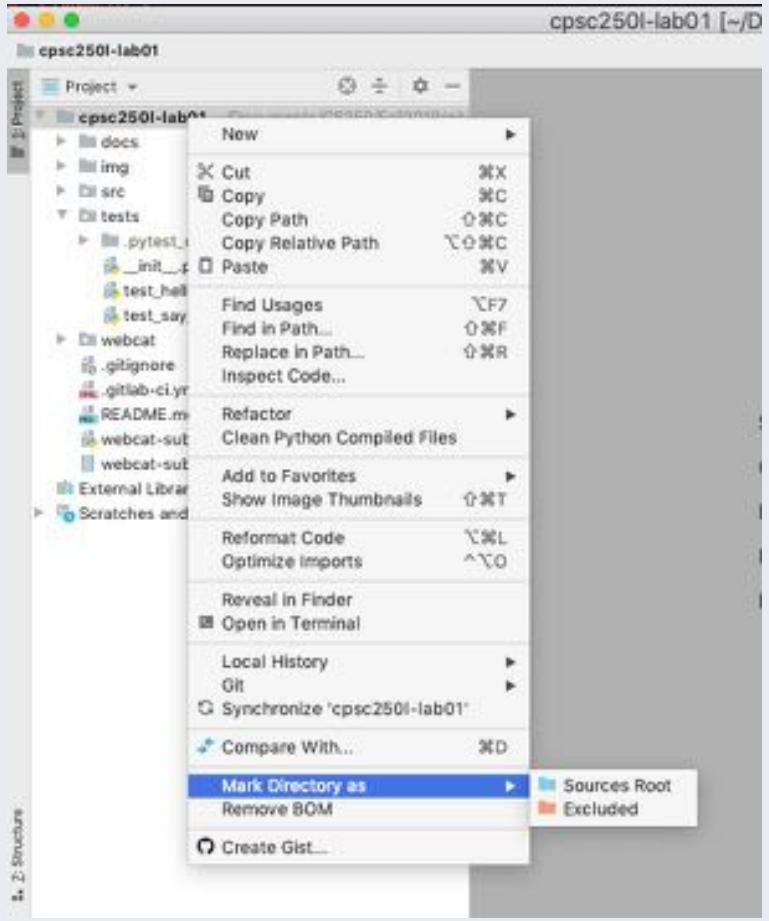
Click on these triangles to expand/collapse given folders within the project

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

        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!"
-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

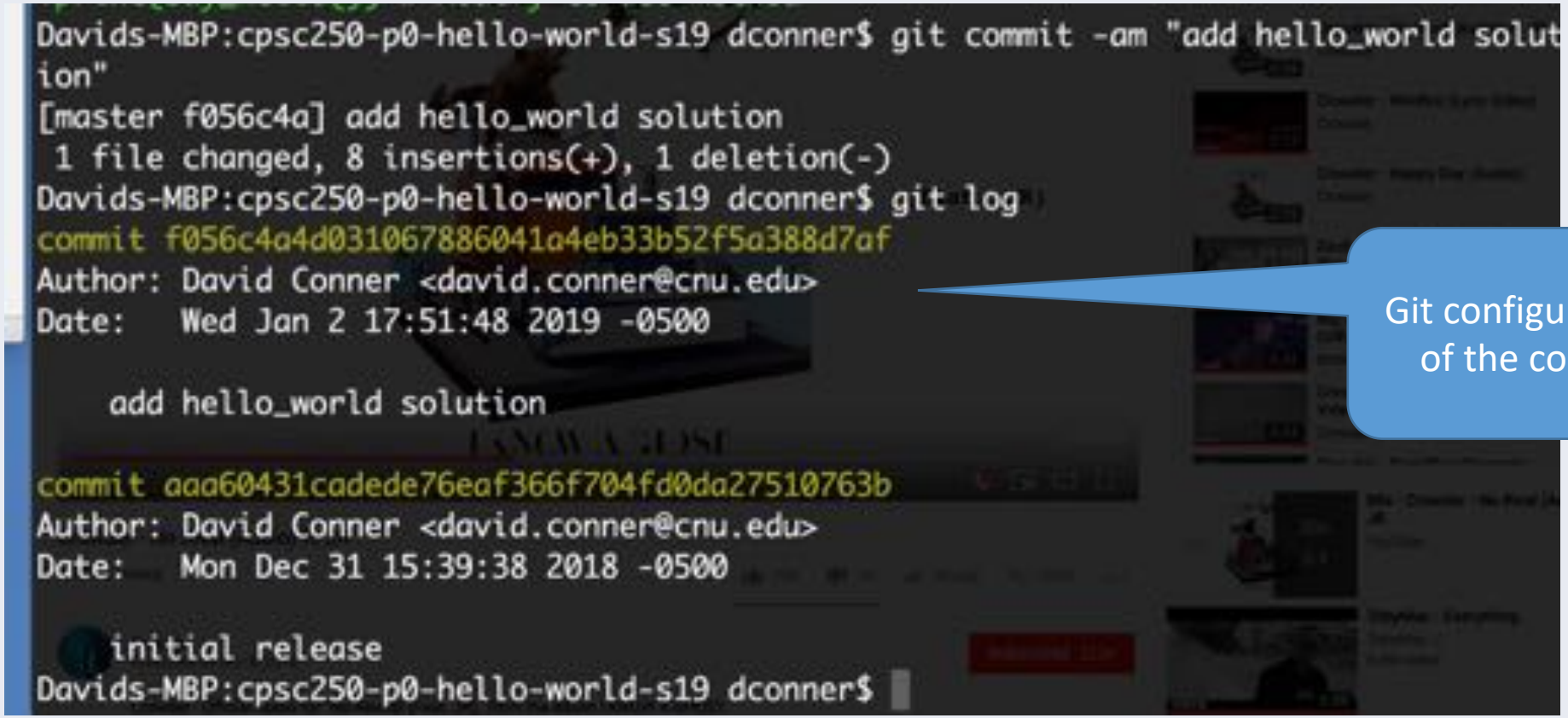
- Two options :
  1. Separate add and commit commands
     - `git add .` (the . says add all untracked files in directory)
     - `git commit –m “a relevant message”`

  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



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

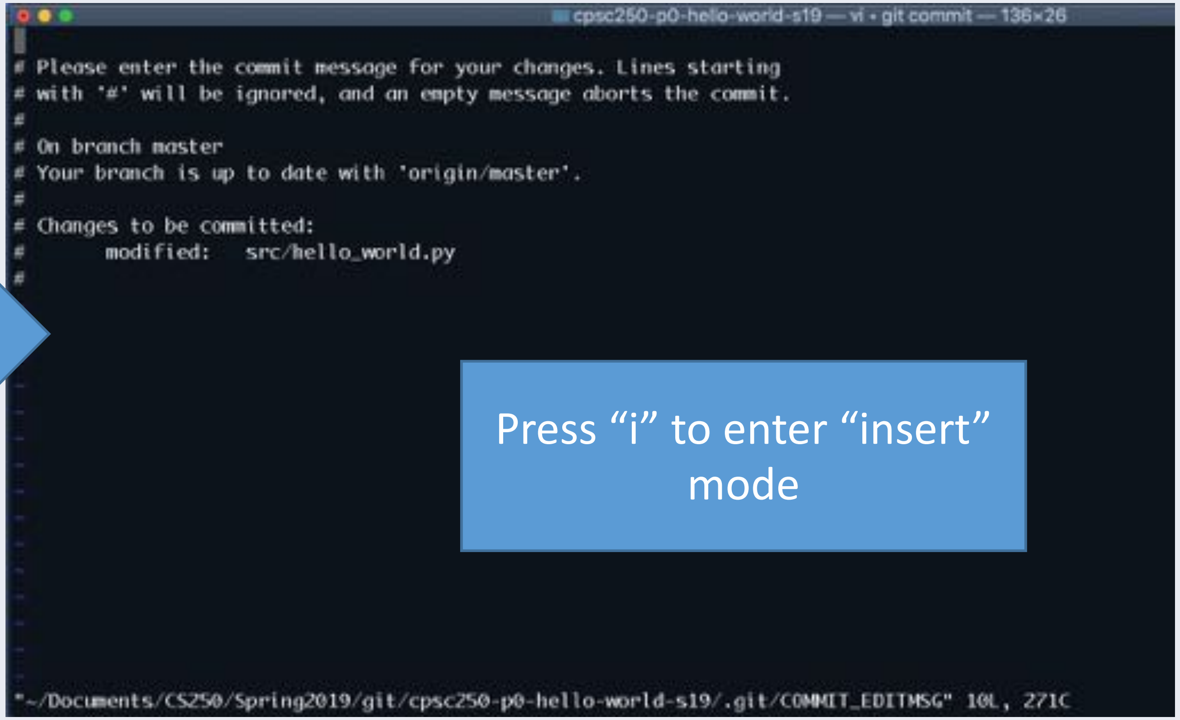
`git status` let's us know the status; now we are ready to commit

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

Forgetting the –m

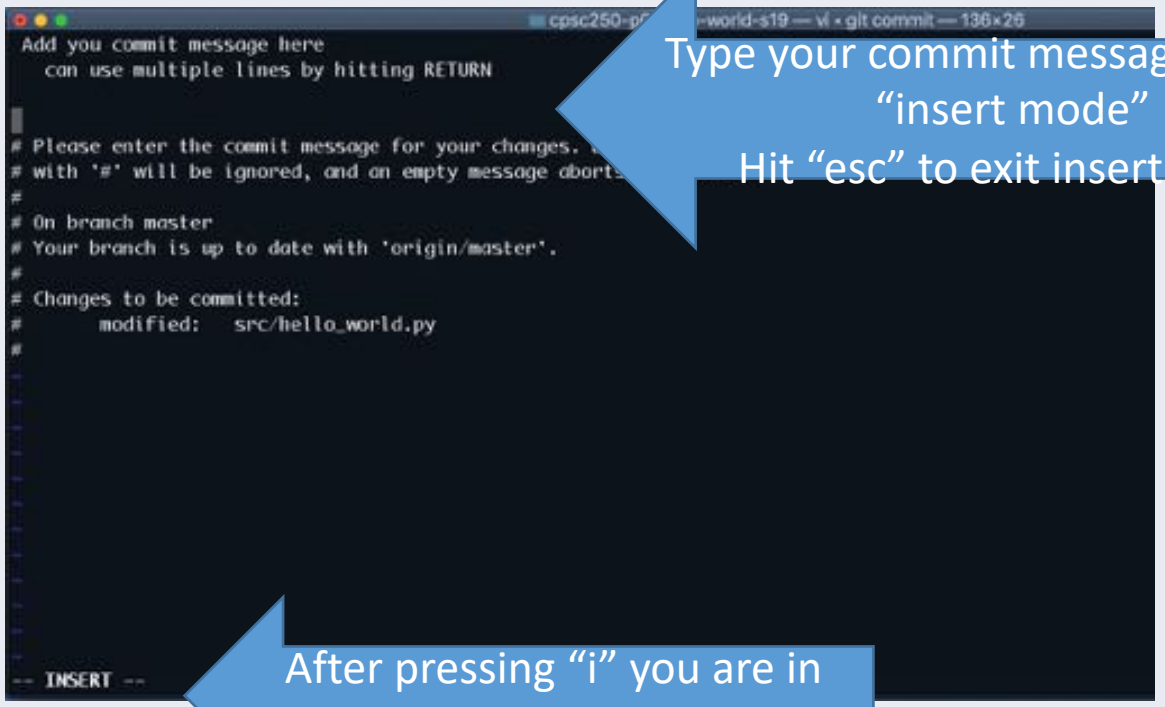
May open this “vim”-like editor



Press “i” to enter “insert” mode

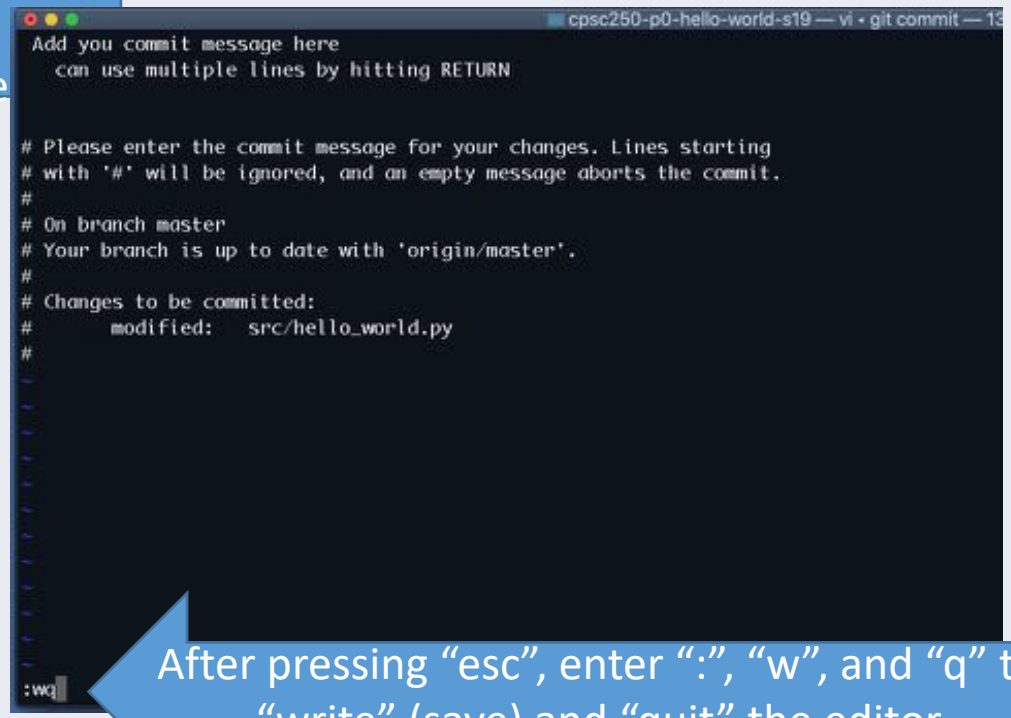
WARNING: Avoid tricky editors

- 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



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

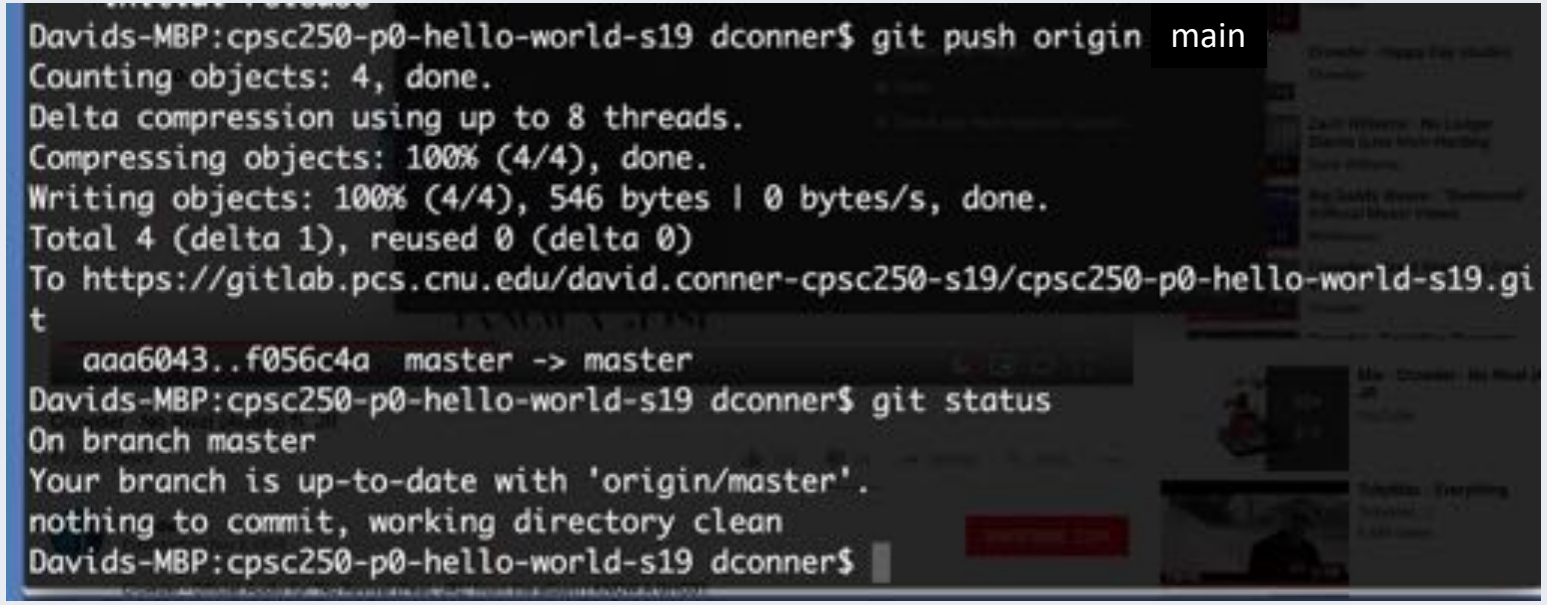
After pressing “i” you are in “insert” mode



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)

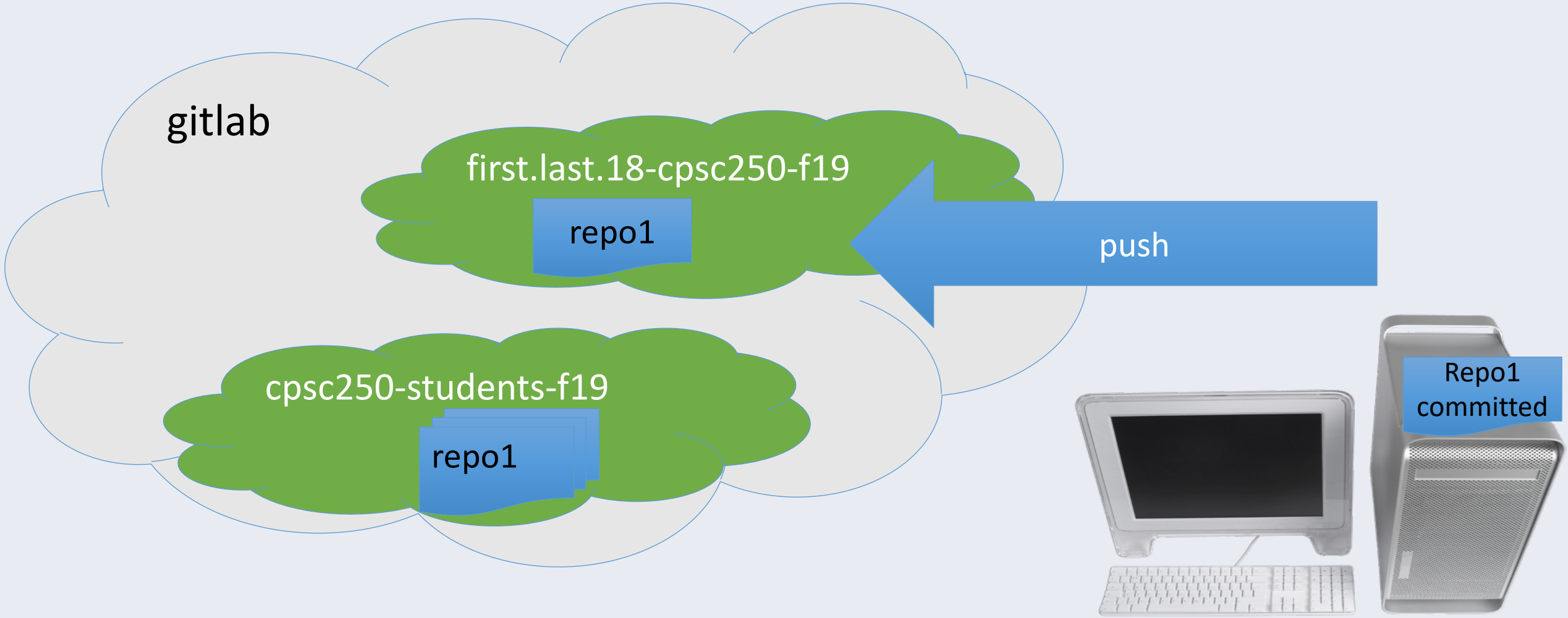


"git push" by itself will push all local changes in all branches, which can be dangerous;
get used to specifying target branches. Or …

```
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

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!

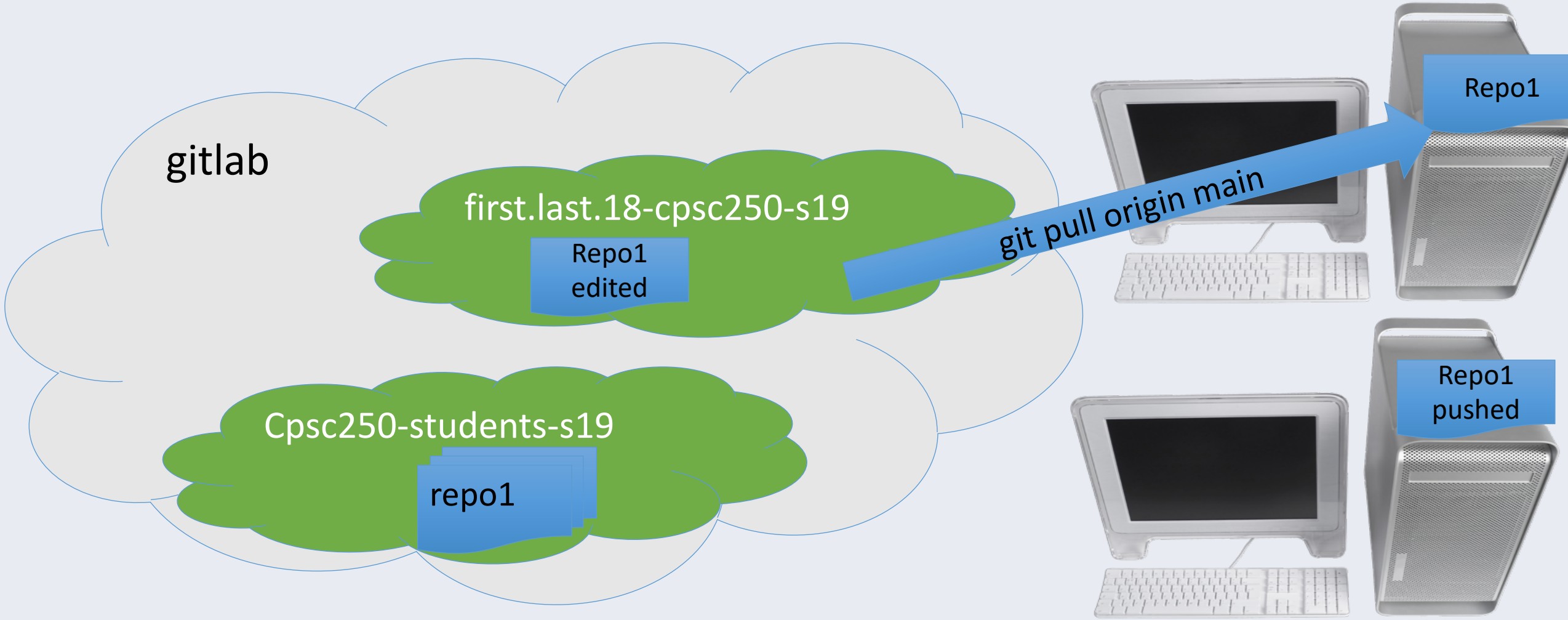
git status shows we are ahead of remote

git push local committed changes to remote

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

Christopher Newport University
Department of Physics, Computer Science & Engineering

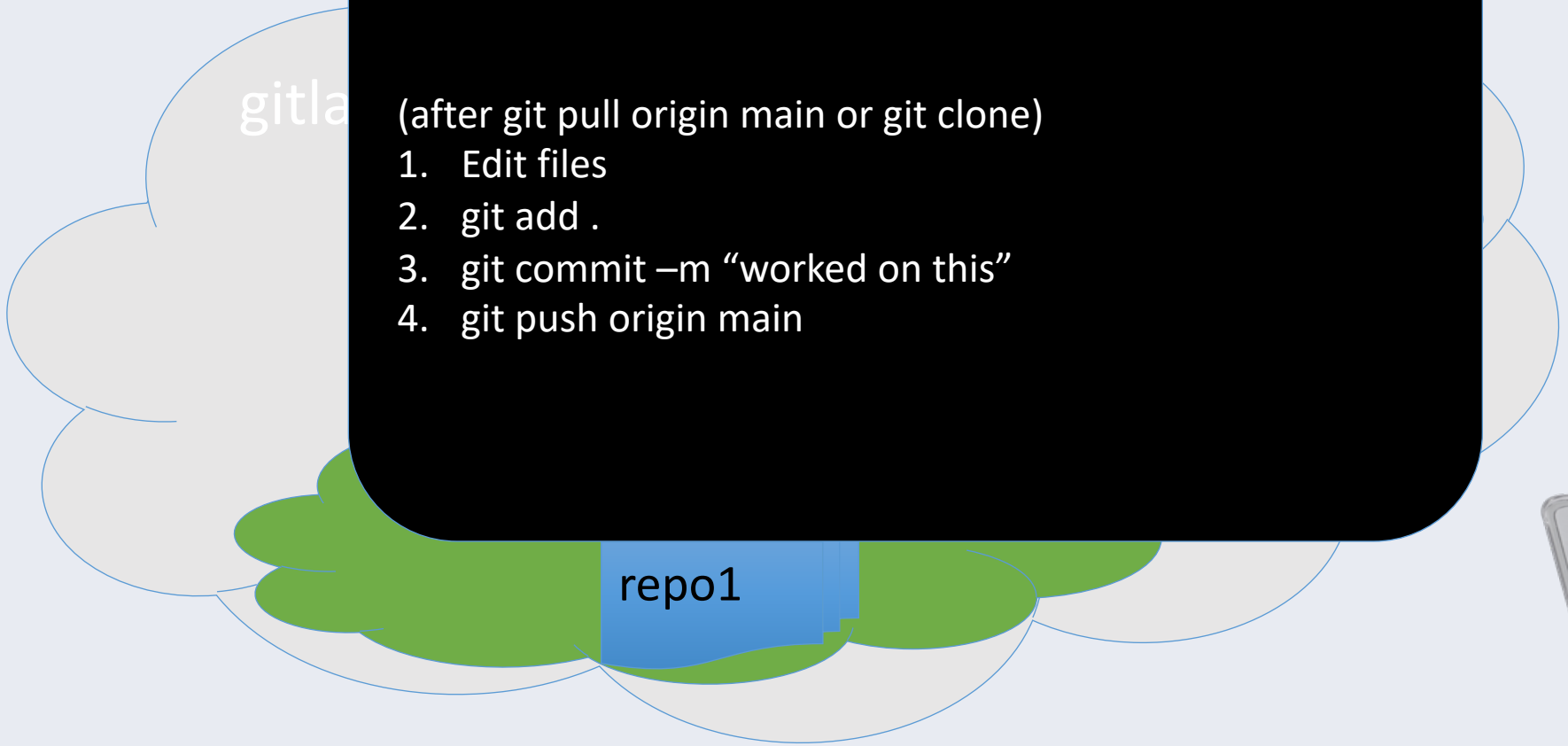
If working with partners, or on different machines after a clone, do a git pull before each add, commit, push session

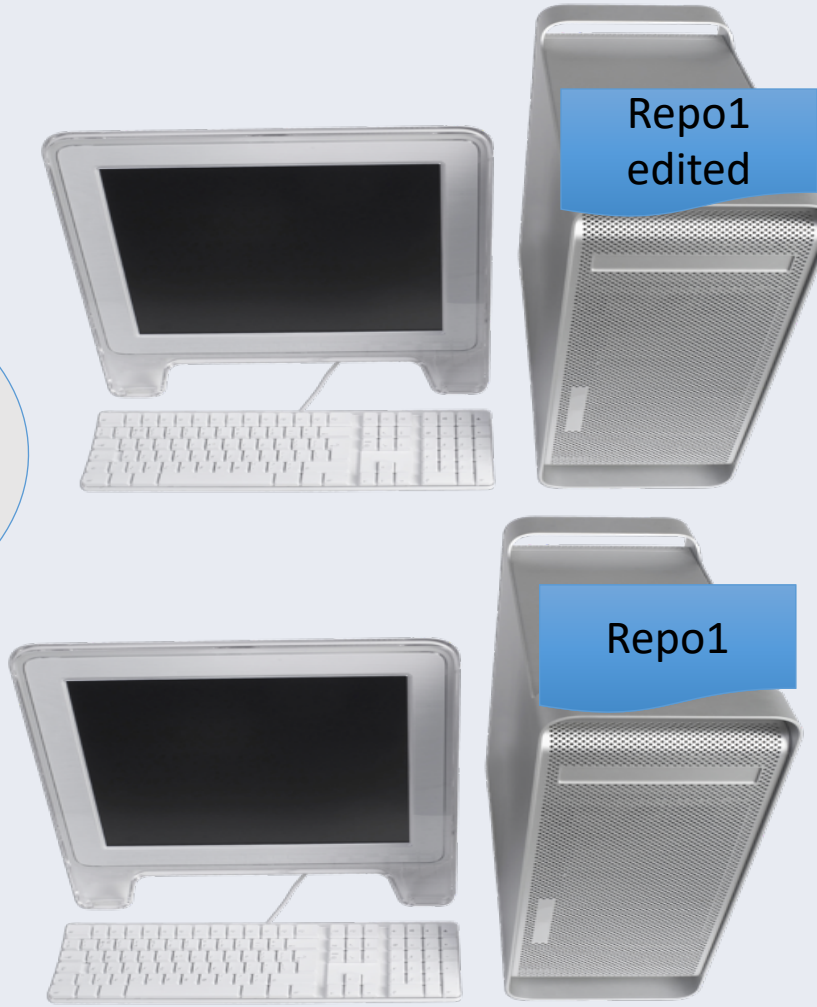


After cloning, before each add, commit, push sessi

(after git pull origin main or git clone)

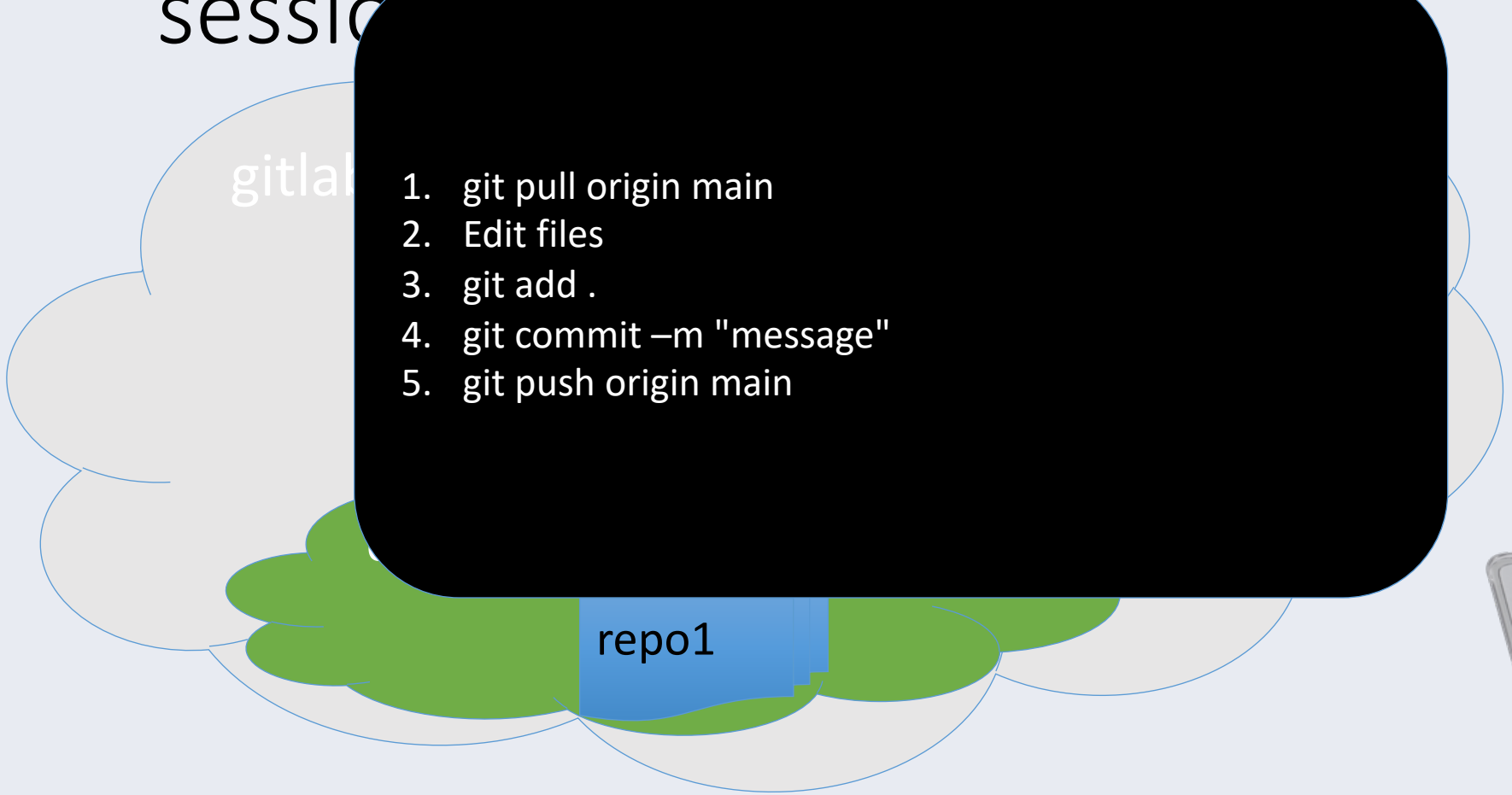
1. Edit files
2. git add .
3. git commit –m “worked on this”
4. git push origin main

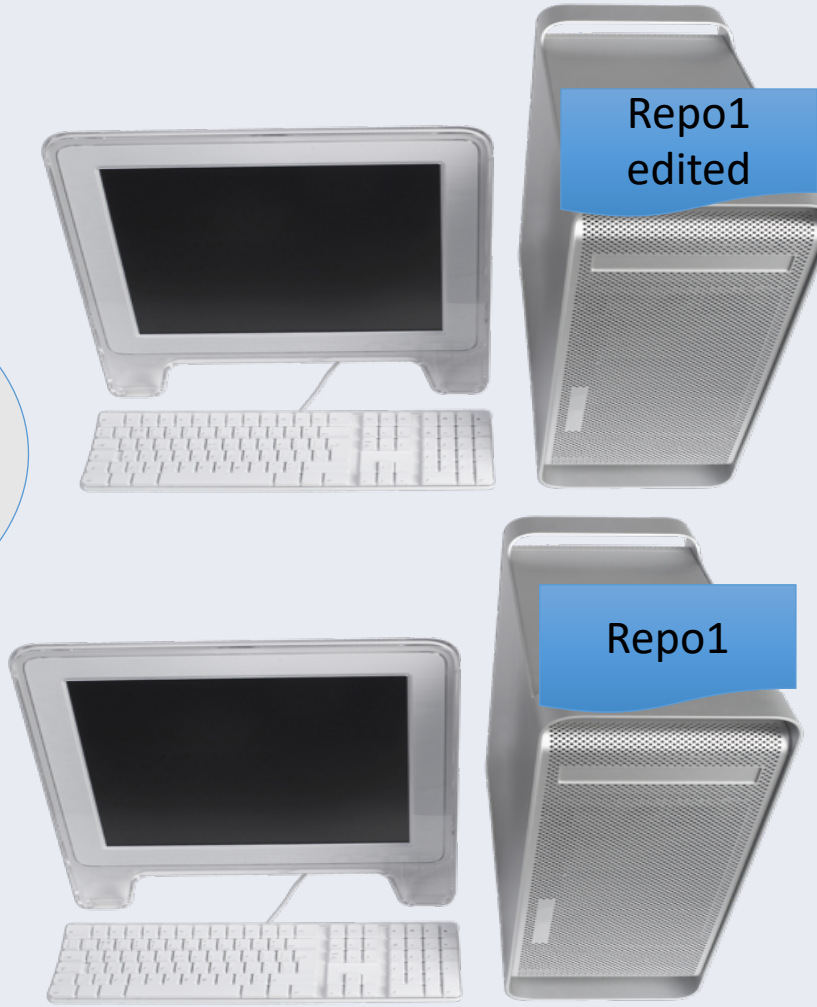




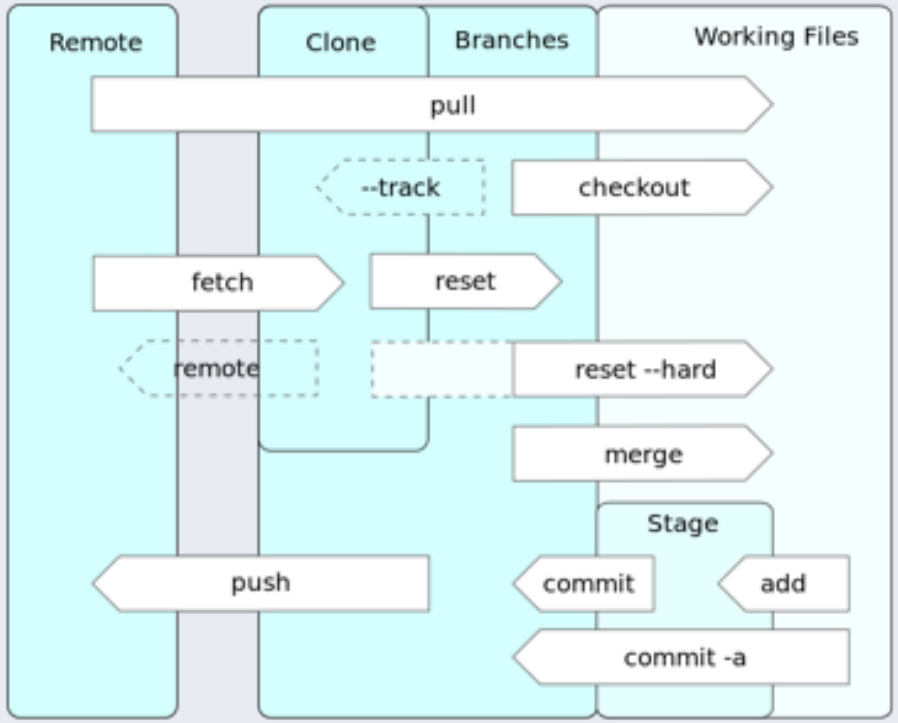
After cloning, before each add, commit, push sessio

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`

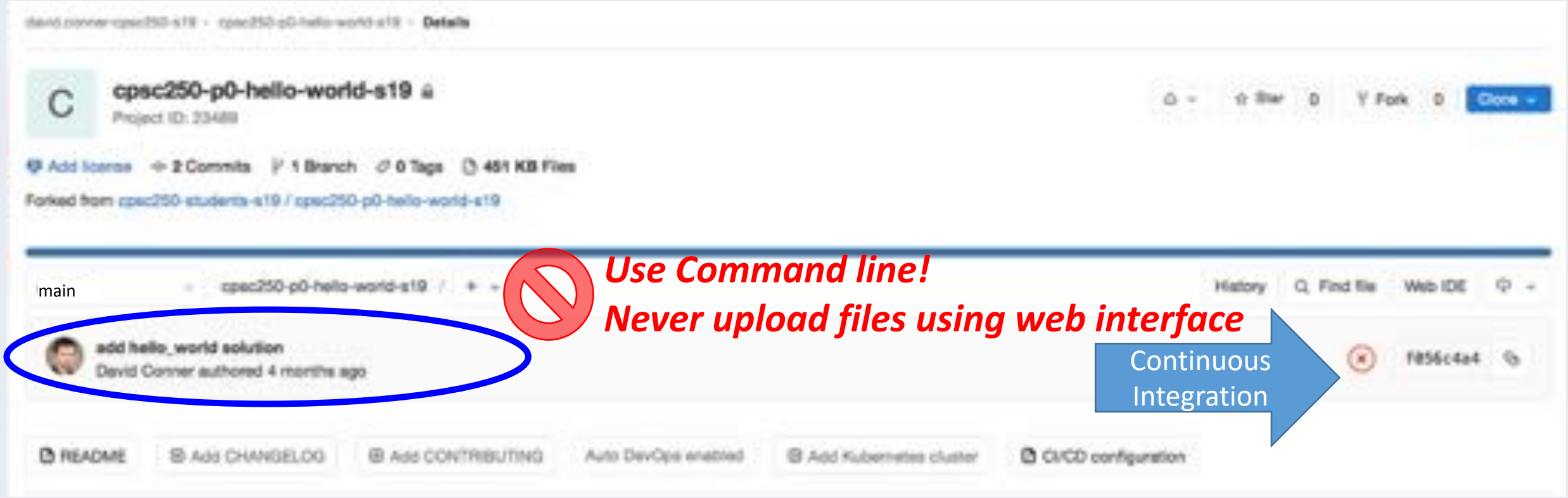
- Stopped here Tuesday 24-Aug-21

Organization Suggestions

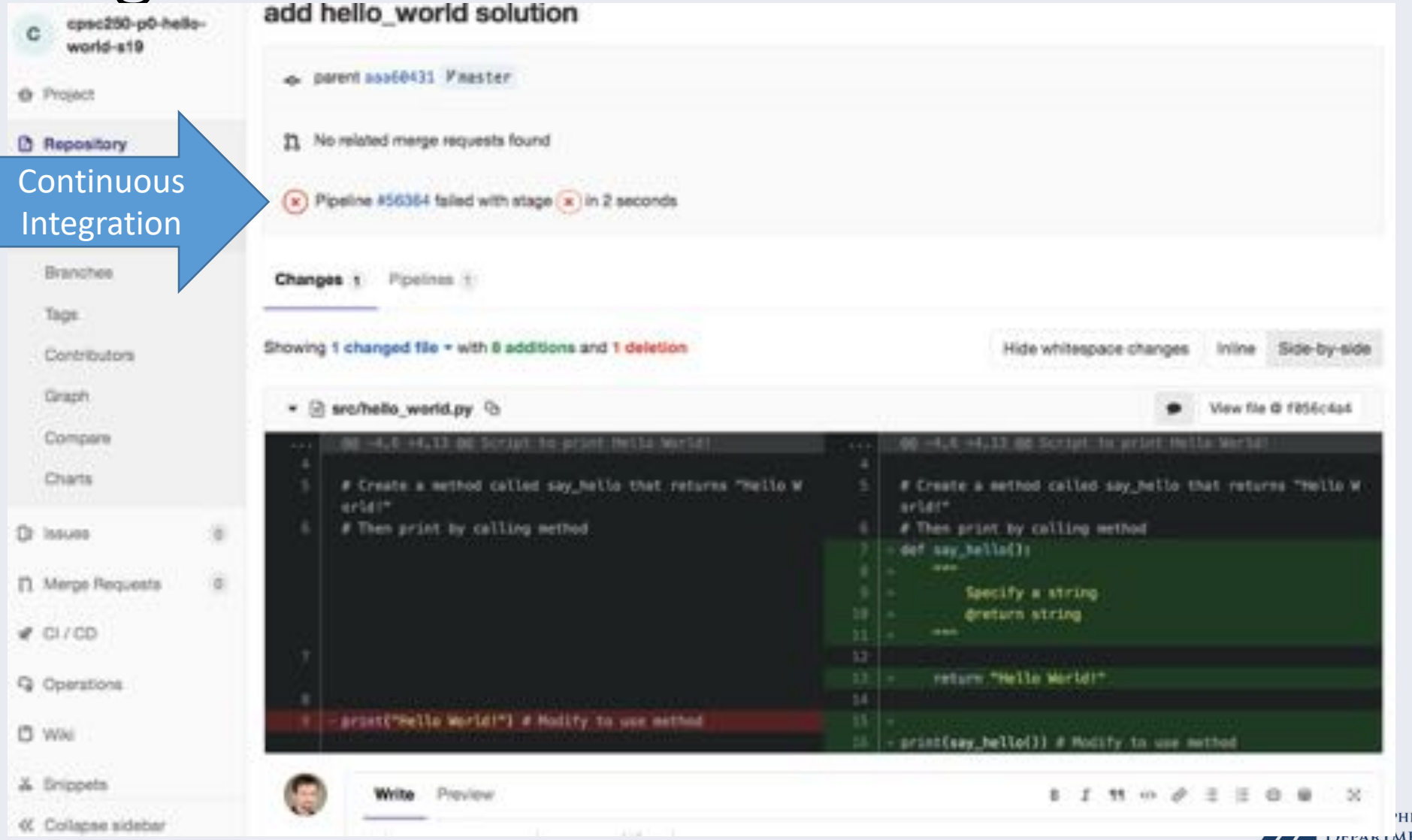
- Create a single directory (or “folder”) on your laptop hard drive to hold all of your work for this class
  - This is your “course” directory
- Clone each git repository (project) into the course directory (folder)
  - Do NOT clone a project into another git repo
    - It is possible, and useful under 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



Clicking on commit name shows differences



Continuous Integration

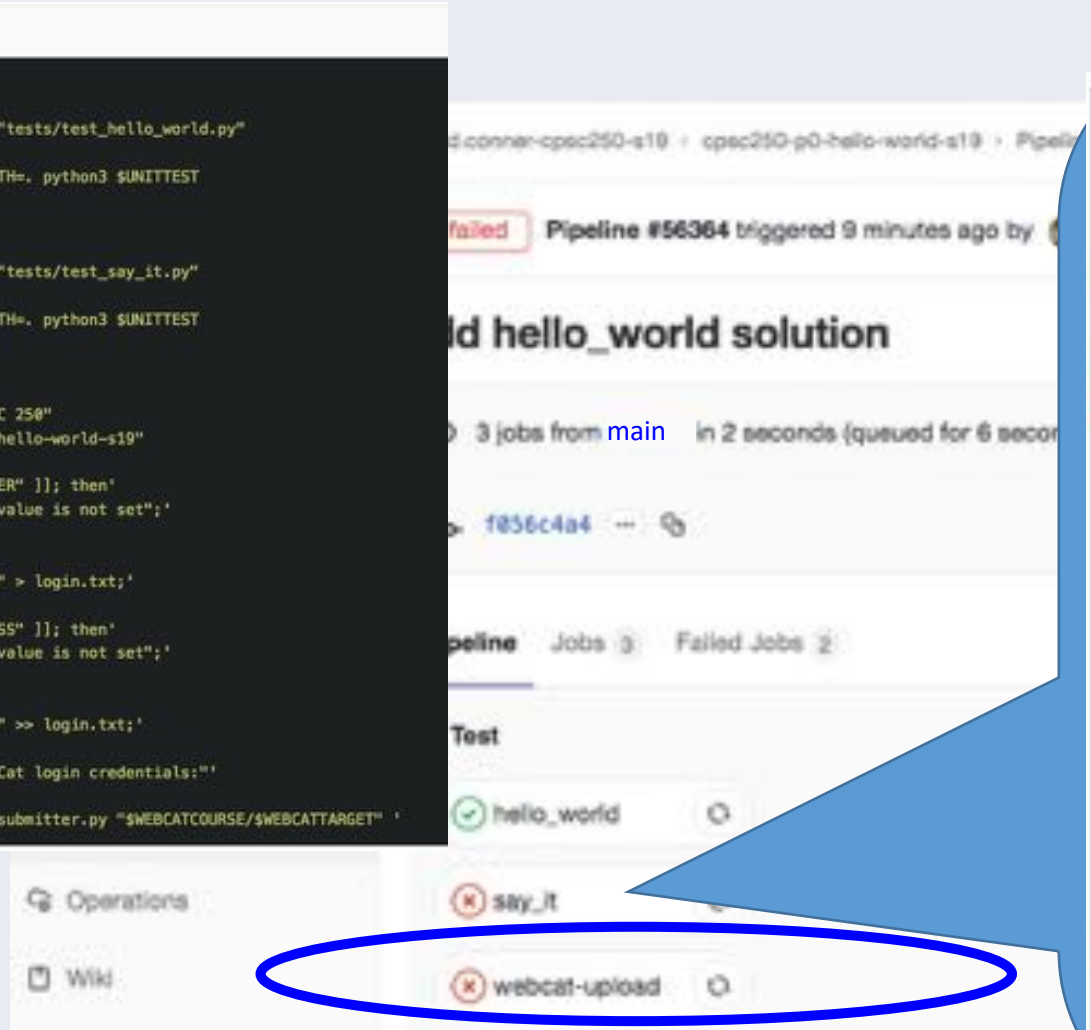
Gitlab Continuous Integration (Testing)

.gitlab-ci.yml 925 Bytes

```
hello_world:
        variables:
                UNITTEST: "tests/test_hello_world.py"
        script:
                - PYTHONPATH=. python3 $UNITTEST

say_it:
        variables:
                UNITTEST: "tests/test_say_it.py"
        script:
                - PYTHONPATH=. python3 $UNITTEST

webcat-upload:
    variables:
        WEBCATCOURSE: "CPSC 250"
        WEBCATTARGET: "p0-hello-world-s19"
    script:
        - 'if [[ -z "$WCUSER" ]]; then'
        - '  echo "WCUSER value is not set";'
        - '  exit 1;'
        - 'else'
        - '  echo "$WCUSER" > login.txt;'
        - 'fi'
        - 'if [[ -z "$WCPASS" ]]; then'
        - '  echo "WCPASS value is not set";'
        - '  exit 1;'
        - 'else'
        - '  echo "$WCPASS" >> login.txt;'
        - 'fi'
        - 'echo "using WebCat login credentials:"'
        - 'cat login.txt'
        - 'python3 webcat-submitter.py "$WEBCATCOURSE/$WEBCATTARGET" '
```

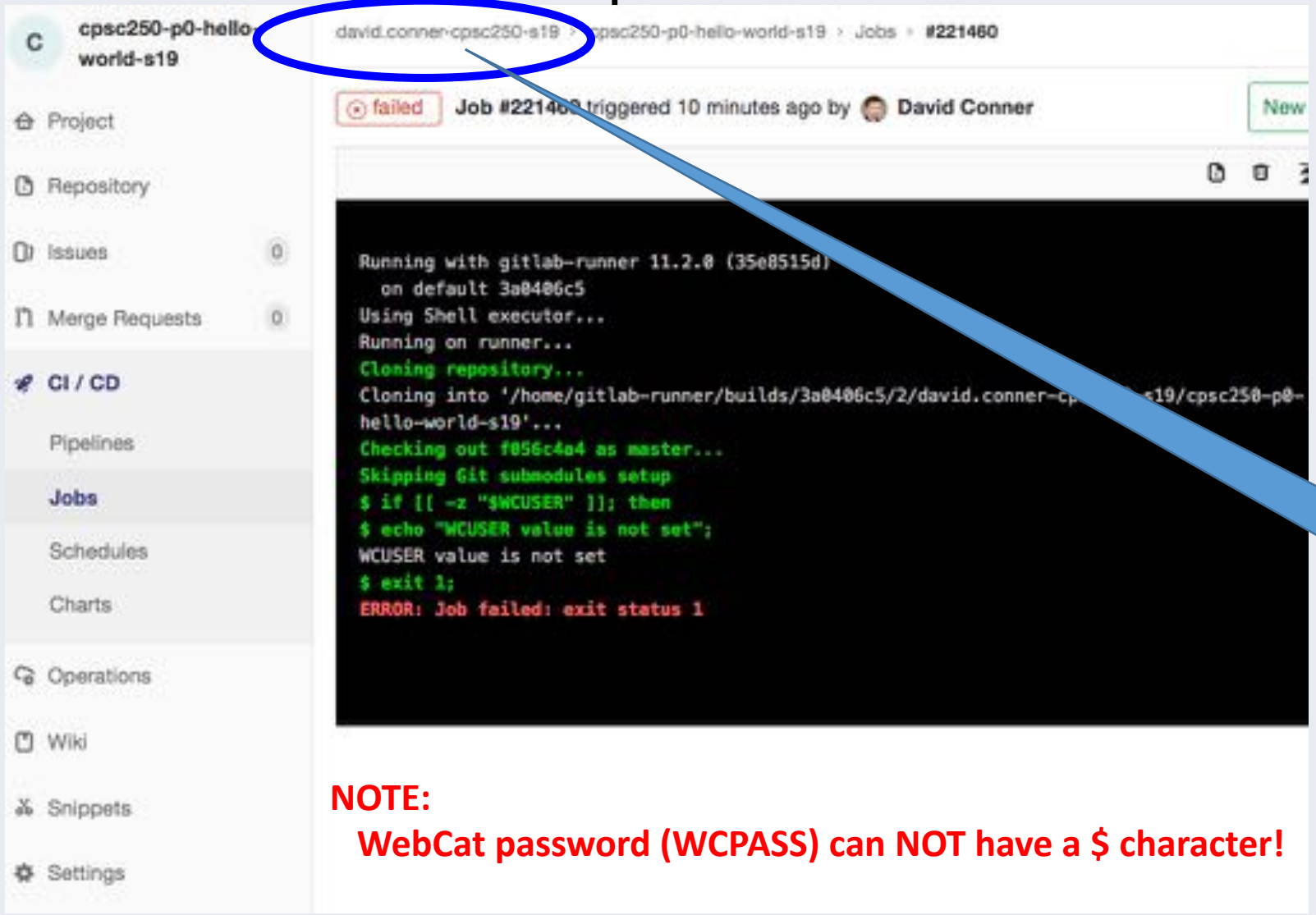



```
Running with gitlab-runner 11.2.0 (35e8515d)
  on default 3a0406c5
Using Shell executor...
Running on runner...
Cloning repository...
Cloning into '/home/gitlab-runner/builds/3a0406c5/1/david.conner-cpsc250-s19/cpsc250-p0-
hello-world-s19'...
Checking out f056c4a4 as master...
Skipping Git submodules 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({})
Actual ({})".format(expected, output_str))
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



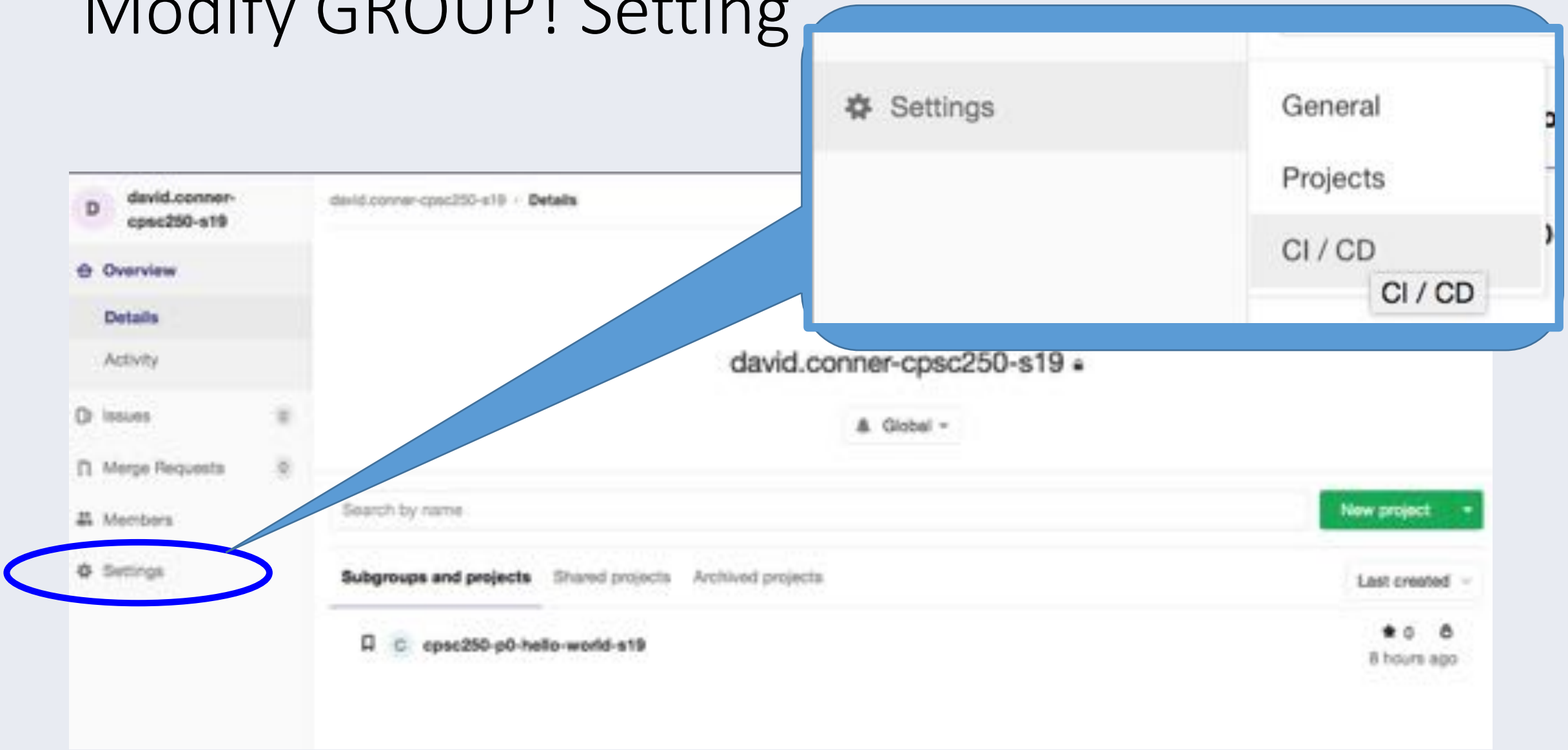
NOTE:
WebCat password (WCPASS) can NOT have a $ character!

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

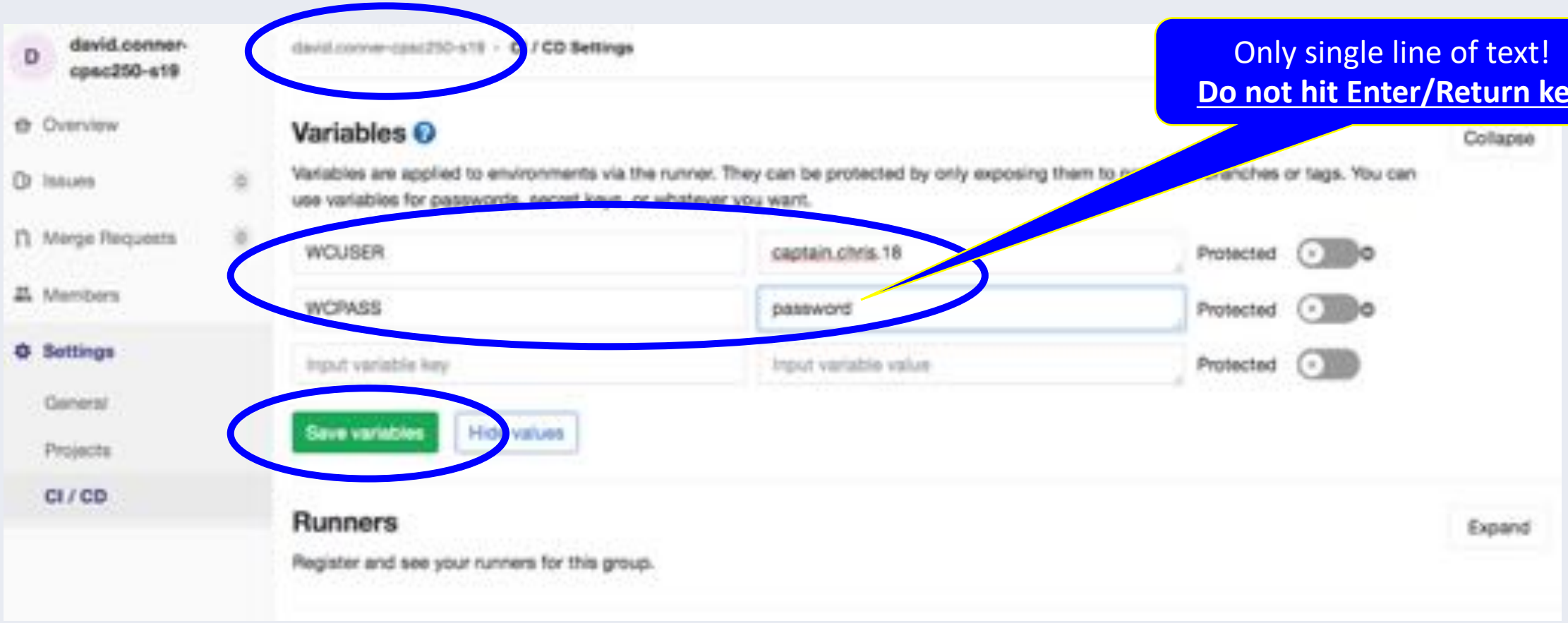
We need to do this <u>ONCE</u> for group (not individual project); then all future projects will work

Click on GROUP name in upper left

Modify GROUP! Setting

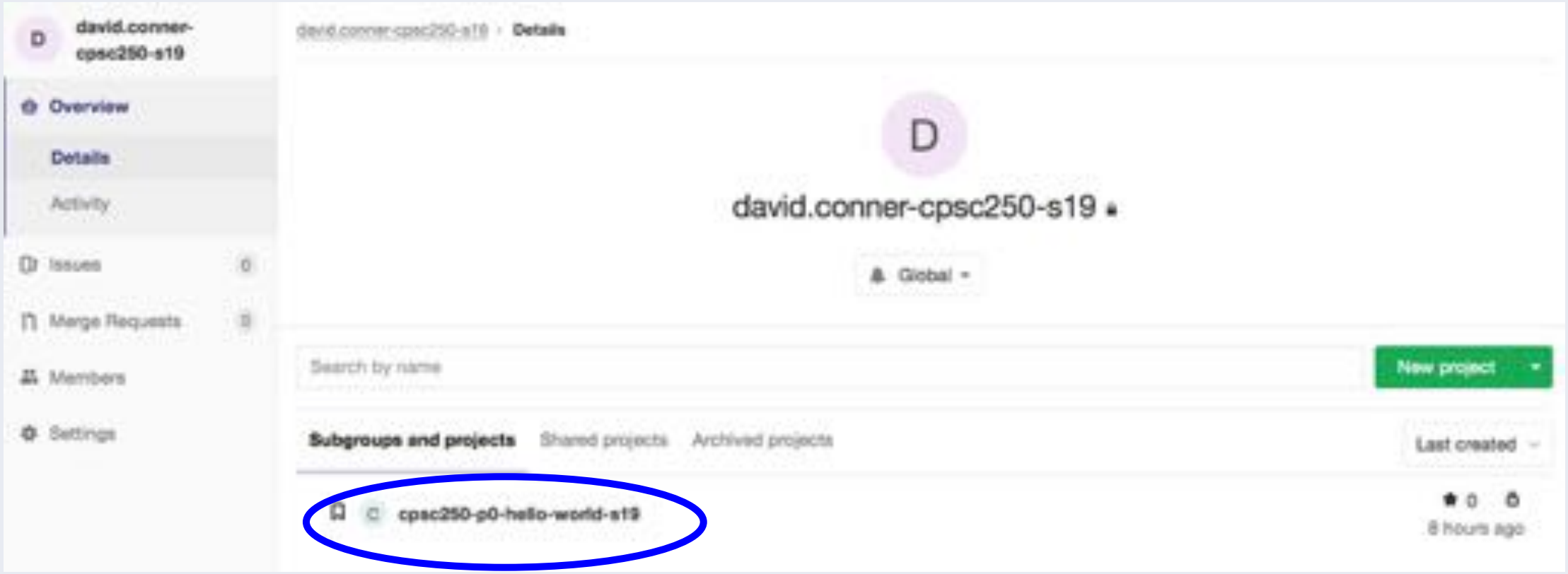


Set variables, save, click group name

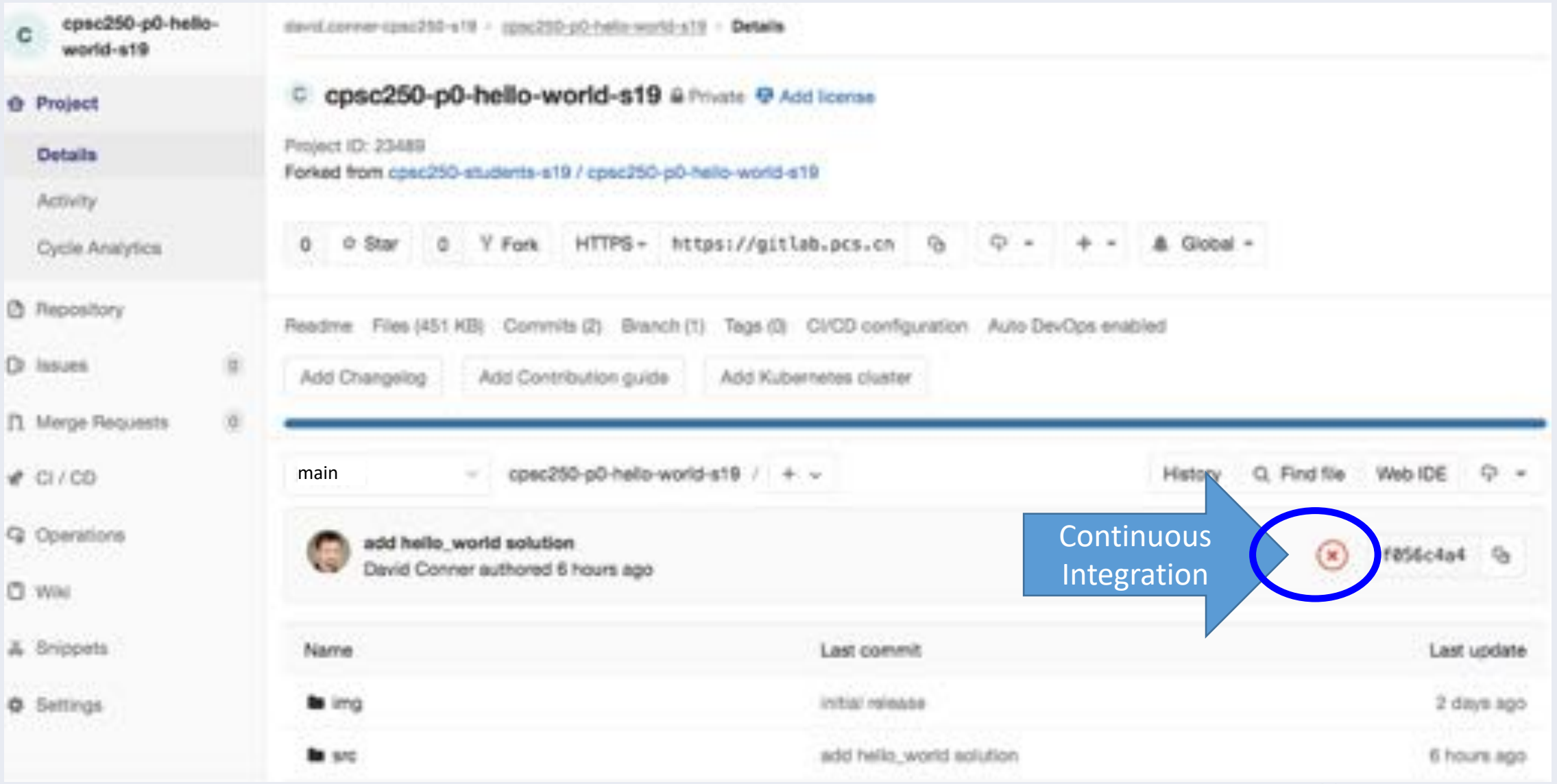


NOTE:
WebCat password (WCPASS) can NOT have a $ character!

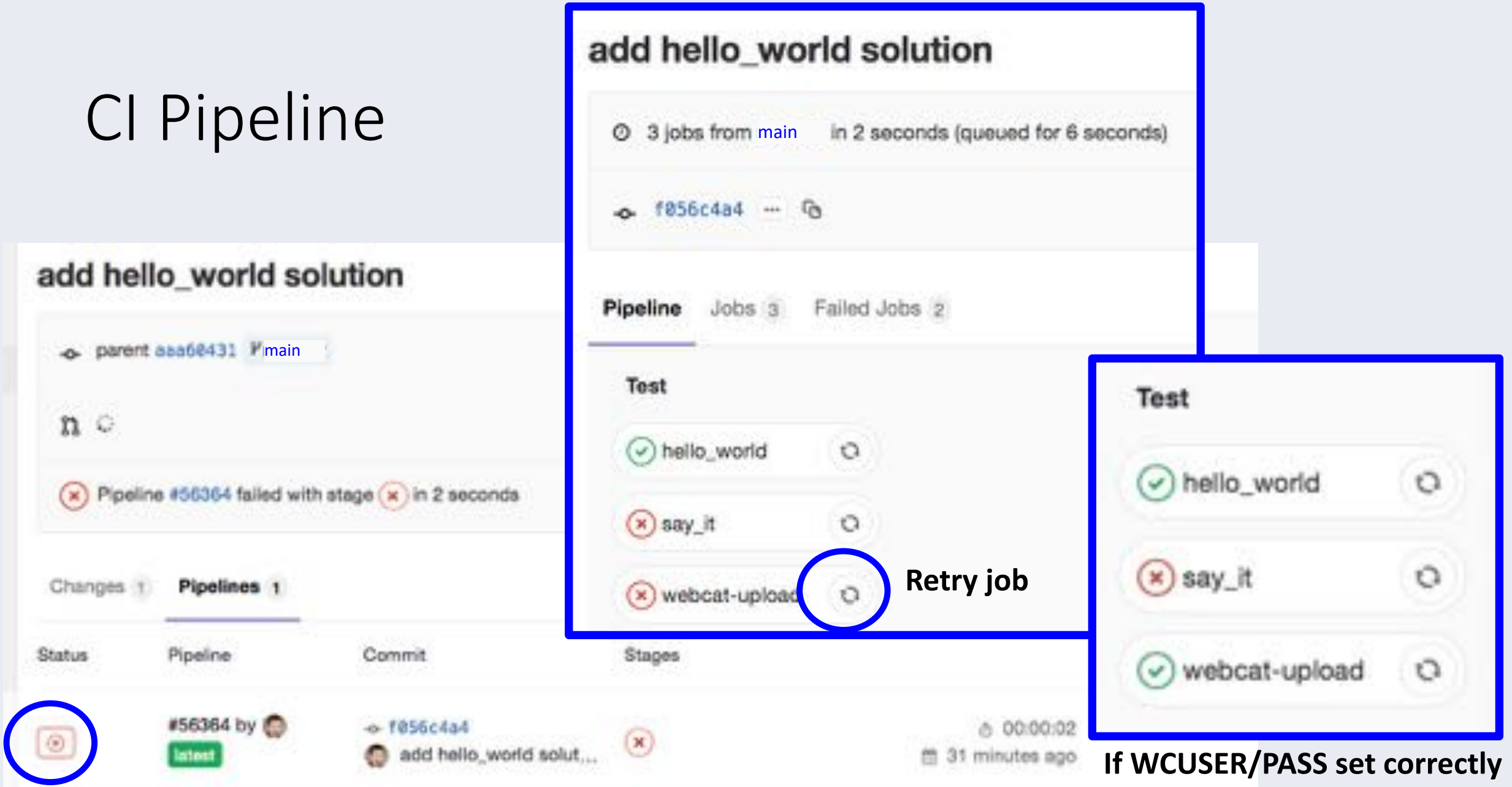
Select project name



Gitlab: Verify Changes on Remote

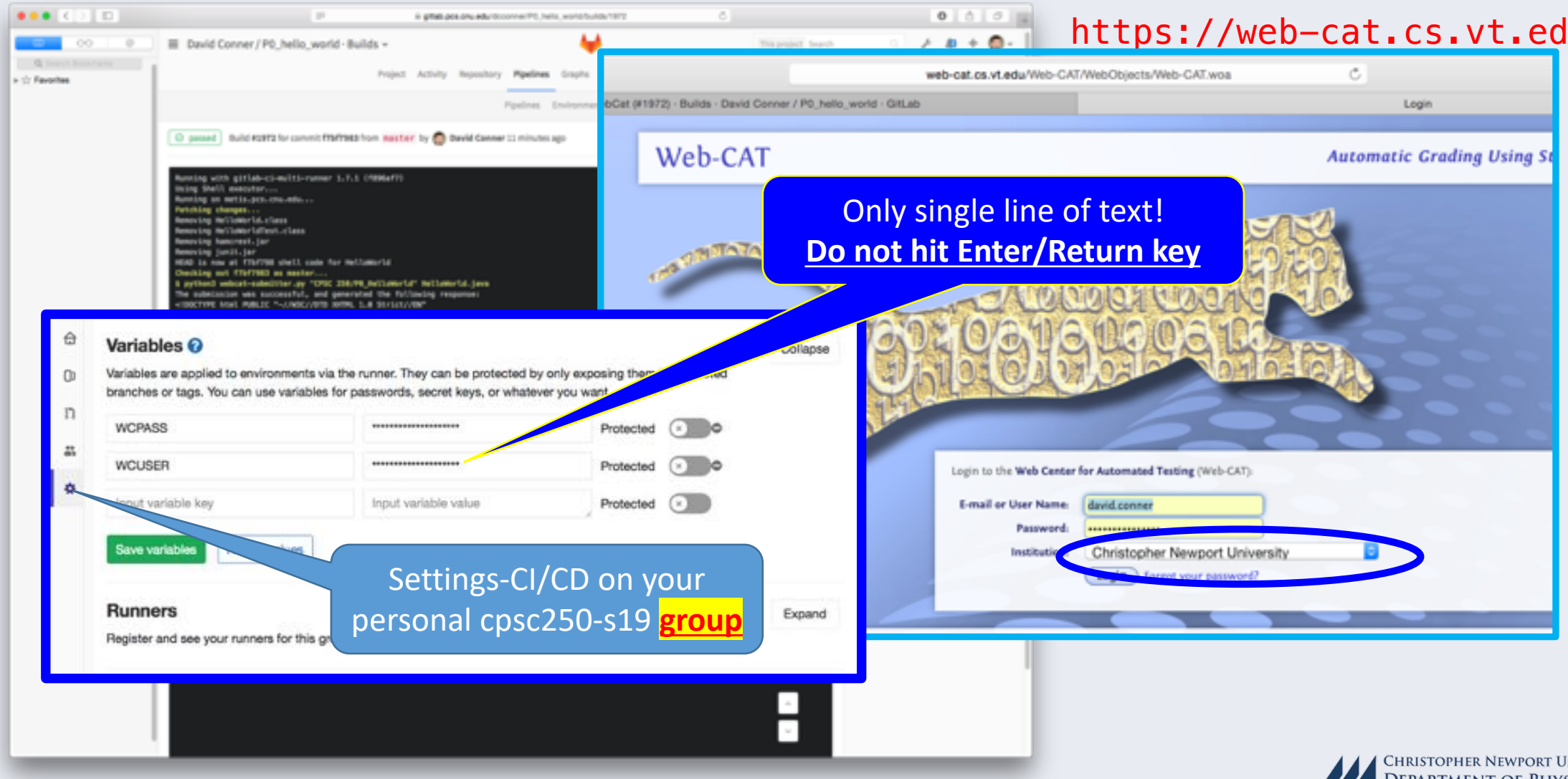


CI Pipeline

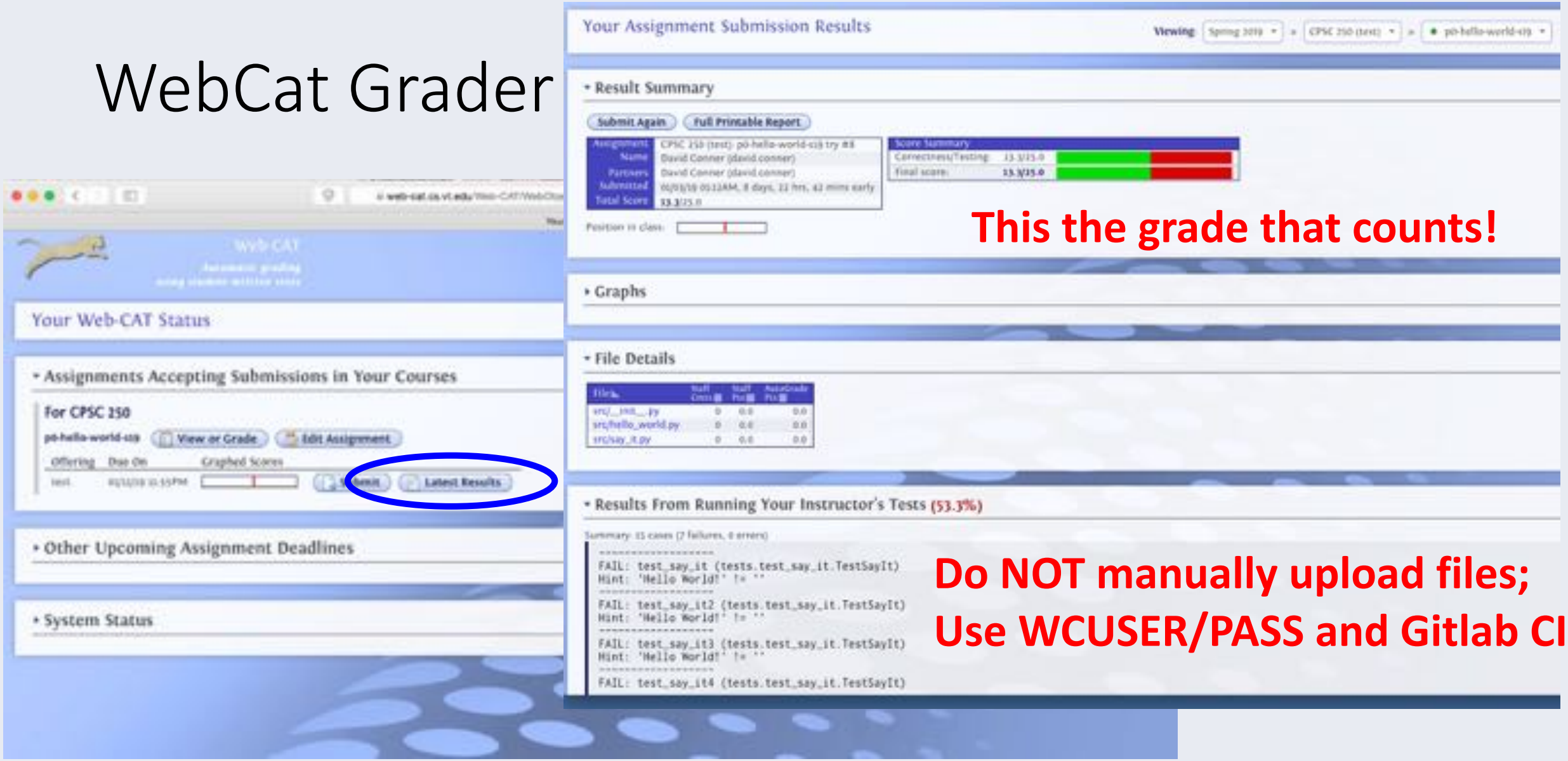

add hello_world solution
3 jobs from main
in 2 seconds (queued for 6 seconds)
f056c4a4
Pipeline
Jobs 3
Failed Jobs 2
Test
hello_world
say_it
webcat-upload
Retry job
add hello_world solution
parent aaa60431
main
Pipeline #56364 failed with stage in 2 seconds
Changes 1
Pipelines 1
Status
Pipeline
Commit
Stages
#56364 by
latest
f056c4a4
add hello_world solut...
00:00:02
31 minutes ago
Test
hello_world
say_it
webcat-upload
If WCUSER/PASS set correctly

Automatic WebCat Submission

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



WebCat Grader



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!