

CptS 321 Homework Turn-in Setup Tutorial

By the TAs¹

This is a HW assignment and in-class exercise turn-in setup tutorial. For this course we will create 2 **EECS GitLab** repositories and we will use them throughout the semester. 1 repository will be used for all HW assignments and 1 repository will be used for all in-class exercises.

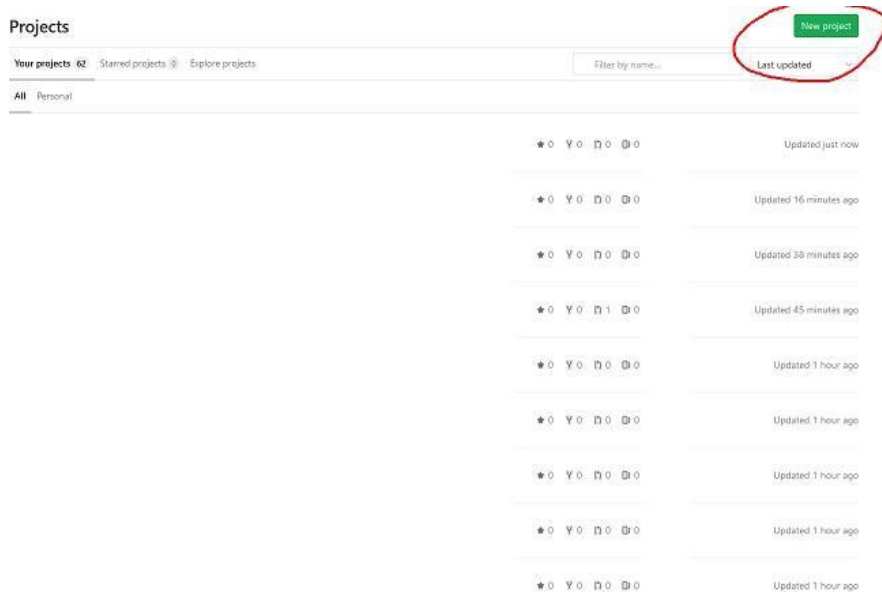
¹ Special credit to Daniel Yune for creating the initial version of this document and to all other TAs for their help to keep it up to date!

Contents

Initial Gitlab Setup:	3
Setting up Gitlab for TA and Professor access:	6
Cloning repository to local machine:	7
Setup Gitignore:	10
Creating a Branch for a Homework (via Gitlab Method):	15
Creating a Branch for Homework (via Bash Method):	18
After Branch Creation:	19
Tagging: your code is considered submitted if it is tagged on GitLab AND submitted via Canvas!	21
Ending:	23

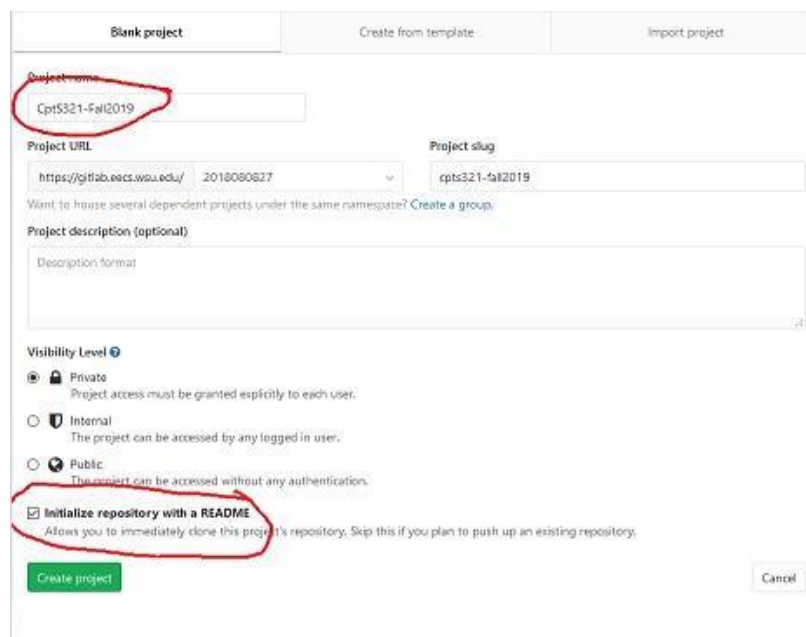
Initial Gitlab Setup:

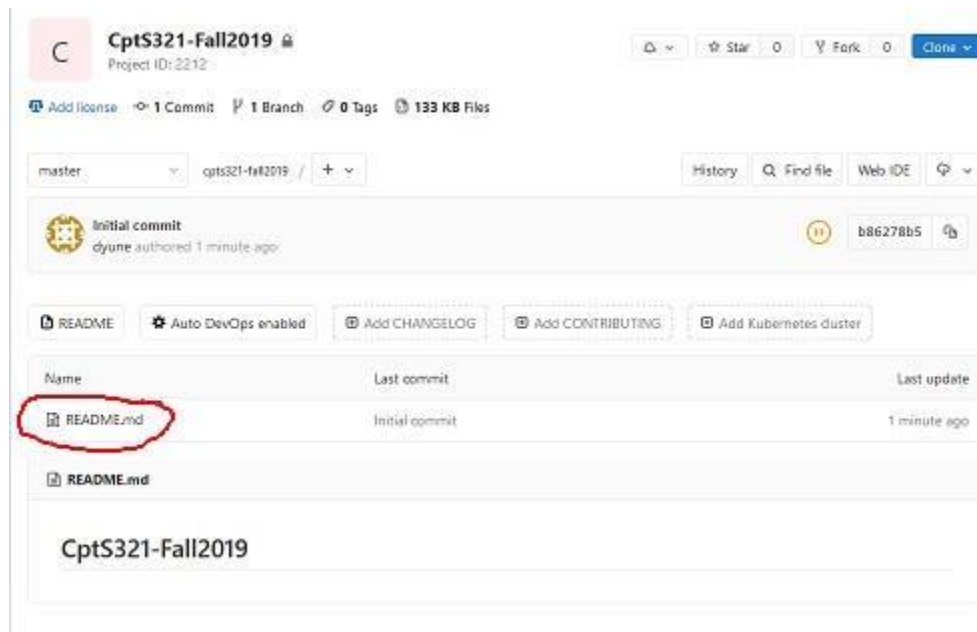
Go to <https://gitlab.eecs.wsu.edu/> and login with your credentials. If your password is expired or you don't now it check the VCEA Help Desk: <https://confluence.esg.wsu.edu/display/KB/EECS+-+Reset+your+EECS+Password>



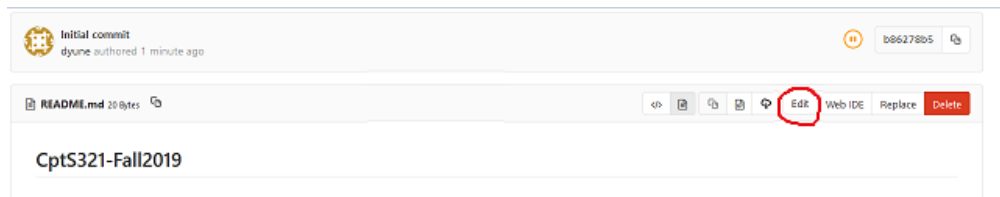
Click on “New Project” and type a Project Name, in this case I have put “CptS321-Fall2019” as an example but you can use “CptS321-HWs” for your HW assignments and “CptS321-in-class-exercises” for the in class coding exercises for example.

Check “Initialize repository with a README”. This will make it easy as now the repository will have a readme.md



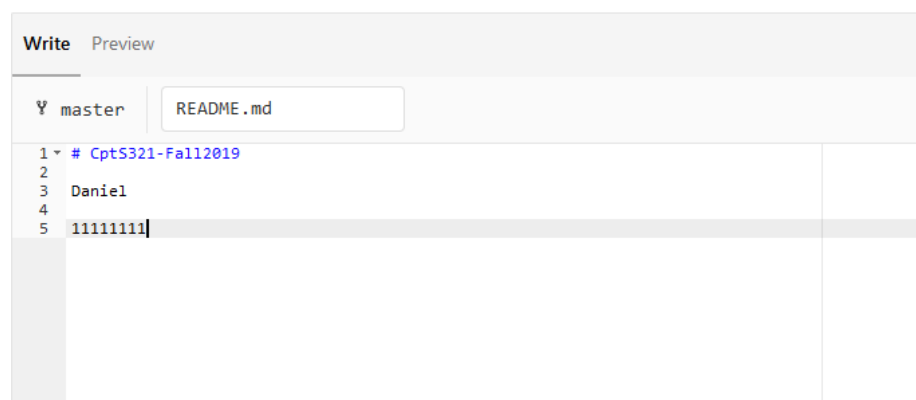


You can now see your repository. You can edit the readme.md right now directly in gitlab by clicking it



Click edit to open the built in text editor.

Edit file



Add your information (at a minimum it must contain your WSU ID and your name) then scroll down to find the Commit Changes button

```

1 - CptS321-Fall2019
2
3 Daniel
4
5 11111111

```

Commit message: Update README.md

Target Branch: master

Commit changes Cancel

After clicking that, you should see the changes back in the file browser view.

CptS321-Fall2019

Project

Repository

Files

Commits

Branches

Tags

Contributors

Graph

Compare

Charts

Issues: 0

dyune · CptS321-Fall2019 · Repository

master · cpts321-fall2019 / README.md

Update README.md
dyune authored 4 minutes ago

README.md 36 bytes

CptS321-Fall2019

Daniel

11111111

Click “Project” on the left toolbar and it should bring you to the Root home page for this project.

CptS321-Fall2019

Project ID: 2212

Add license · 1 Commit · 1 Branch · 0 Tags · 133 KB Files

master · cpts321-fall2019

Update README.md
dyune authored 5 minutes ago

README · Auto DevOps enabled · Add CHANGELOG · Add CONTRIBUTING · Add Kubernetes cluster

Name	Last commit	Last update
README.md	Update README.md	5 minutes ago

README.md

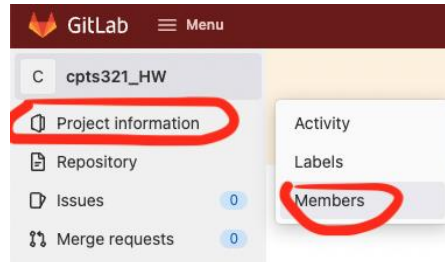
CptS321-Fall2019

Daniel

11111111

Next thing we need to do is add all the TAs and Dr. Arnaoudova to the gitlab project so we all can access everything we need to for grading.

Setting up Gitlab for TA and Professor access:



On the left, hover over “Project information” and click “Members”

Project members

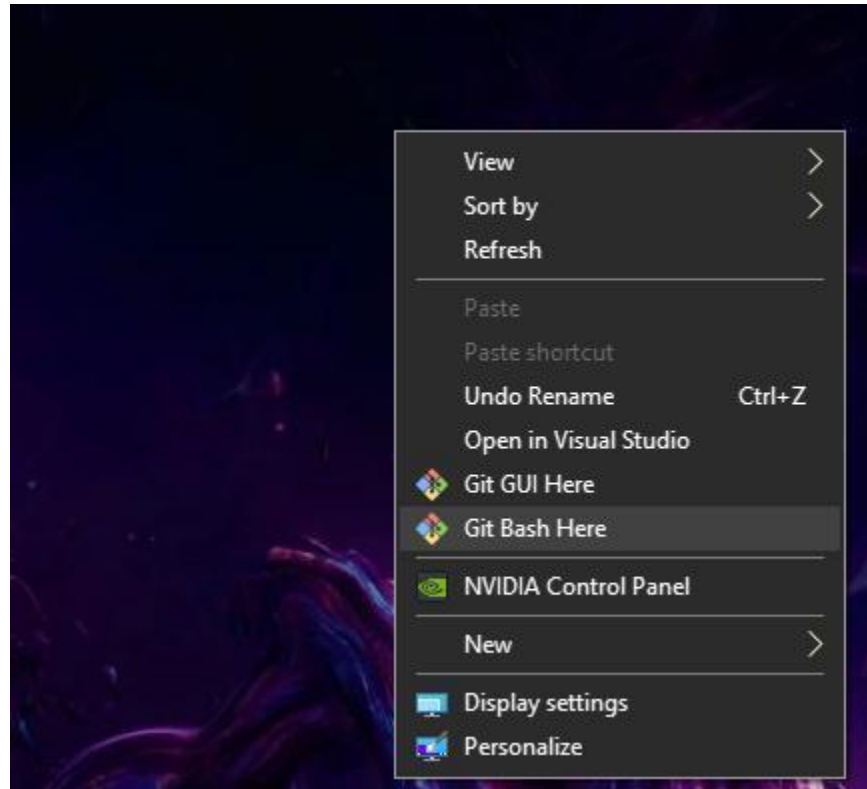
You can invite a new member to **CptS321-Fall2019** or invite another group.

A screenshot of the 'Project members' page in GitLab. The page has two tabs: 'Invite member' (active) and 'Invite group'. Under the 'Invite member' tab, there's a form with the following fields: 'GitLab member or Email address' (containing 'varnaoud'), 'Choose a role permission' (a dropdown menu with 'Maintainer' selected), and 'Access expiration date' (with an 'Expiration date' input field). At the bottom of the form, there's a green 'Add to project' button and a grey 'Import' button. Below the form, there's a section titled 'Existing members and groups' with a search bar and a 'Sort by' dropdown menu.

From there, you can search the usernames. All our usernames are listed in the syllabus on Canvas. You must add **ALL TAs and the instructor** and set us as “**Maintainer**” so we can download projects as necessary for grading. You can set a proper expiration date (As in a date a few weeks after the semester ends) or leave it blank. Once you click Add to project for all of us, we’re all good there.

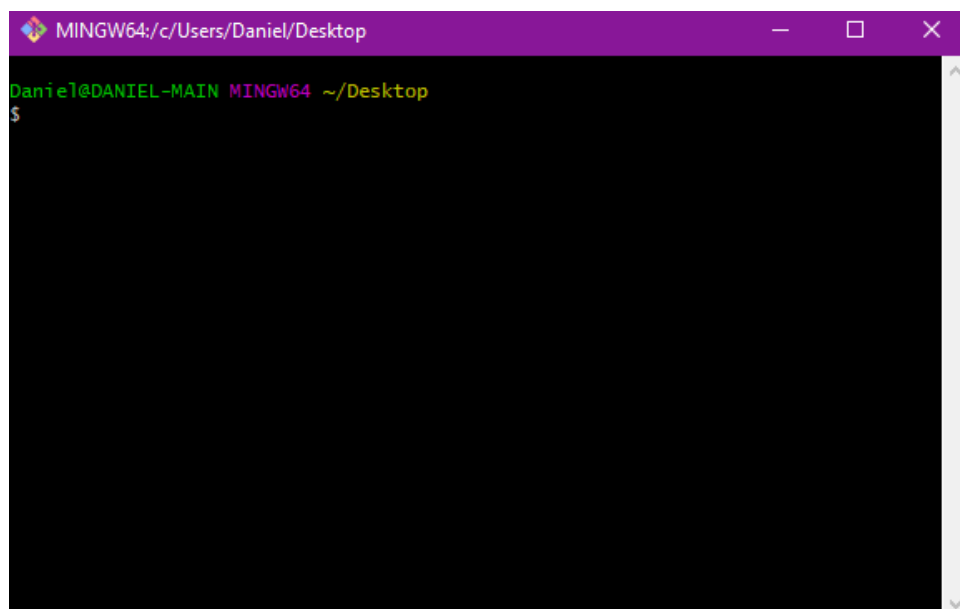
Next thing we need to do is to clone the repository into our local computer. For simplicity, we will do it on the desktop, but it can be done anywhere. Keep the web browser open on the page above. In the following, we do this using GitBash but any other version control tool can be used.

Cloning repository to local machine using

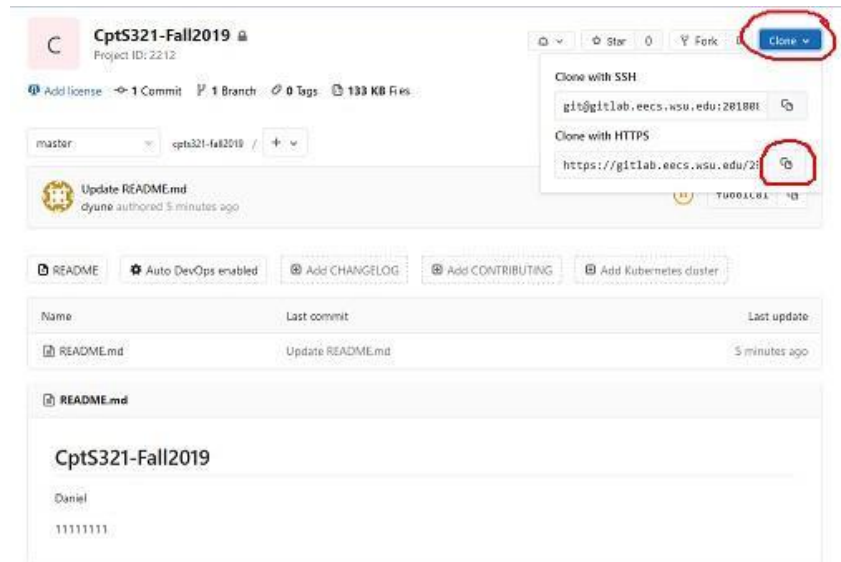


GitBash:

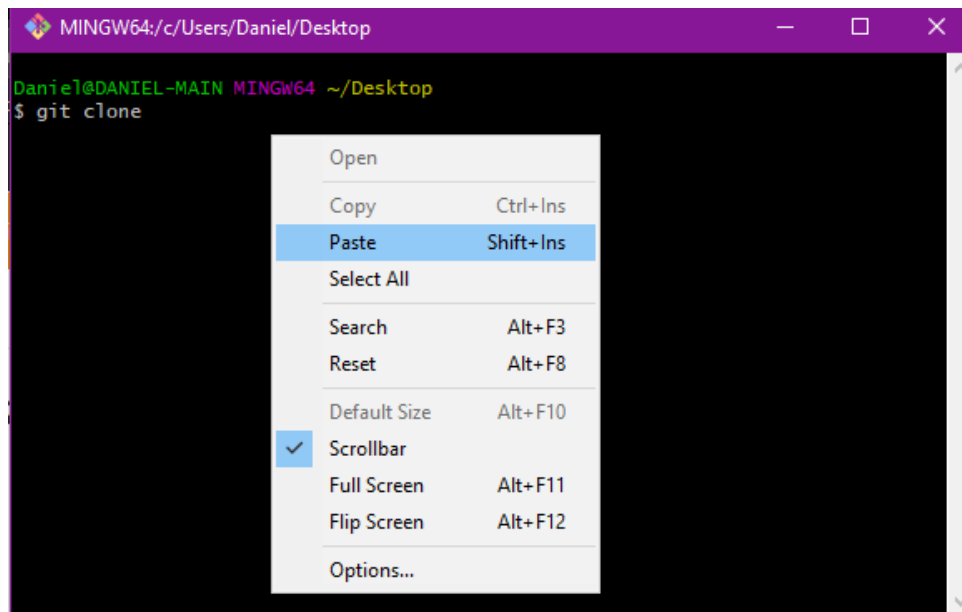
Right click the desktop (or any folder), click “Git Bash Here”. A Git Bash interface will come up as shown below. It is normal to not have anything yet.



Once you can see that, go back to your browser and find that gitlab repository you should've kept open.



Click Clone, then click the “Copy to clipboard” button for the “Clone to HTTPS” one, which will be the bottom one. You can also manually copy the text to the left of the lower red circle with Ctrl-C. Now go back to the git bash interface you opened earlier.



Be aware, paste is not Ctrl-V in this interface, but “Shift+Ins”. You can alternatively just right click and click Paste.

Full command should be “git clone <pasted URL>”


```
MINGW64:/c/Users/Daniel/Desktop

Daniel@DANIEL-MAIN MINGW64 ~/Desktop
$ git clone https://gitlab.eecs.wsu.edu/[REDACTED]/cpts321-fall2019.git
Cloning into 'cpts321-fall2019'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.

Daniel@DANIEL-MAIN MINGW64 ~/Desktop
$ |
```

The erased part will differ for everyone, it is your unique gitlab identifier number, everyone has a different one, so don't be alarmed if yours is short or longer than the blank area.

Now you can enter the directory you just made from cloning:

```
MINGW64:/c/Users/Daniel/Desktop

Daniel@DANIEL-MAIN MINGW64 ~/Desktop
$ git clone https://gitlab.eecs.wsu.edu/[REDACTED] cpts321-fall2019.git
Cloning into 'cpts321-fall2019'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.

Daniel@DANIEL-MAIN MINGW64 ~/Desktop
$ ls

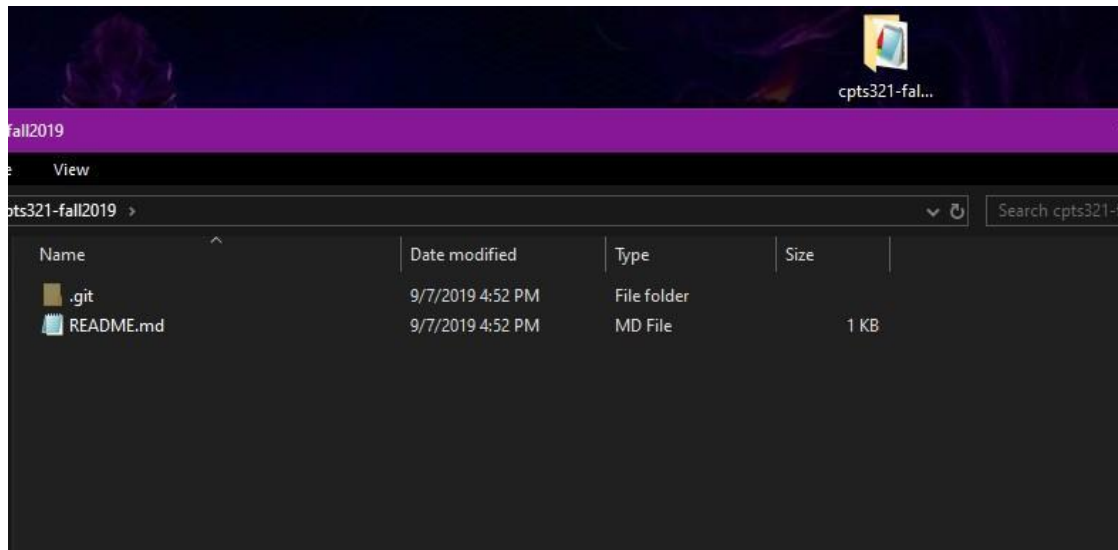
cpts321-fall2019/

Daniel@DANIEL-MAIN MINGW64 ~/Desktop
$ cd cpts321-fall2019/

Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (master)
$ |
```

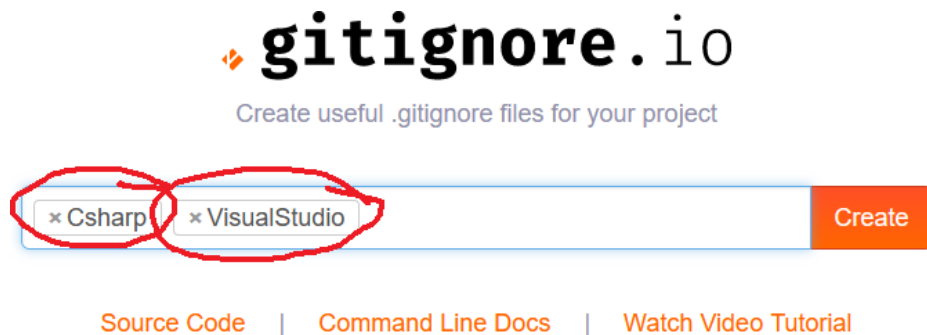
"ls" just lists the directories and shortcuts on the desktop. You can confirm the clone succeeded as a new folder/directory will be made matching the project name. You can type "cd <name>", which you can also hit Tab after you type the first few letters and the Bash interface will auto-fill the rest.

Setup Gitignore:



You can open the directory and see the contents. It should be very empty in this case. The readme.md in here is the one you edited on gitlab's site in earlier steps. The first thing we need is a proper gitignore.

You can go to www.gitignore.io and generate one real quick.



For this course, you will want to type those two filters into the field, and hit Create

```
# Created by https://www.gitignore.io/api/csharp,visualstudio
# Edit at https://www.gitignore.io/?templates=csharp,visualstudio

### Csharp ###
## Ignore Visual Studio temporary files, build results, and
## files generated by popular Visual Studio add-ons.
##
## Get latest from https://github.com/github/gitignore/blob/master/VisualStudio.gitignore

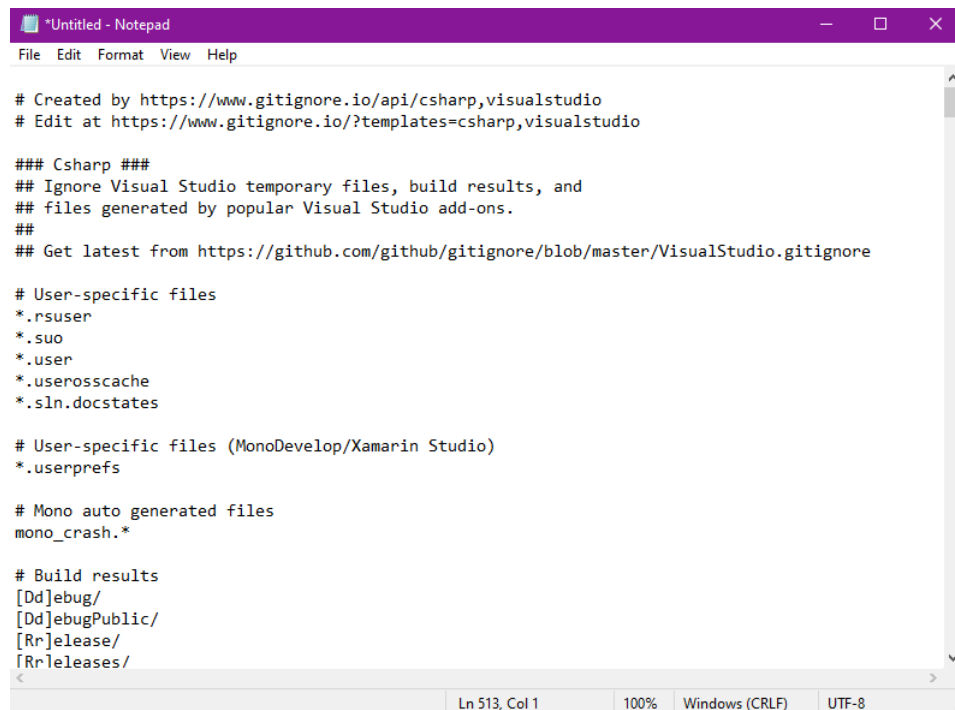
# User-specific files
*.rsuser
*.suo
*.user
*.useroscachecache
*.sln.docstates

# User-specific files (MonoDevelop/Xamarin Studio)
*.userprefs

# Mono auto generated files
mono_crash.*

# Build results
[Dd]ebug/
[Dd]ebugPublic/
[Rr]elease/
[Rr]eleases/
x64/
x86/
```

You should see a text only page that starts like this, this is normal. You will want to copy all this text and paste it in a text editor.



```
# Created by https://www.gitignore.io/api/csharp,visualstudio
# Edit at https://www.gitignore.io/?templates=csharp,visualstudio

### Csharp ###
## Ignore Visual Studio temporary files, build results, and
## files generated by popular Visual Studio add-ons.
##
## Get latest from https://github.com/github/gitignore/blob/master/VisualStudio.gitignore

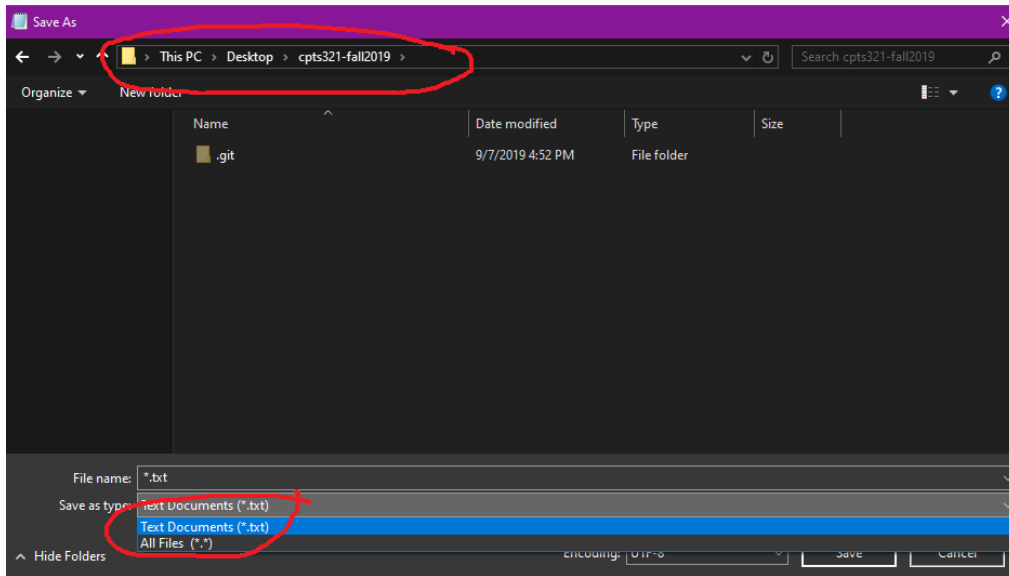
# User-specific files
*.rsuser
*.suo
*.user
*.useroscachecache
*.sln.docstates

# User-specific files (MonoDevelop/Xamarin Studio)
*.userprefs

# Mono auto generated files
mono_crash.*

# Build results
[Dd]ebug/
[Dd]ebugPublic/
[Rr]elease/
[Rr]eleases/
```

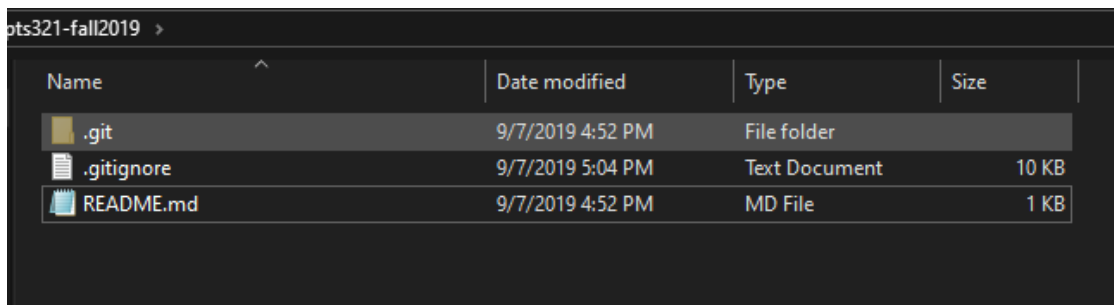
From here you can go to File->Save As



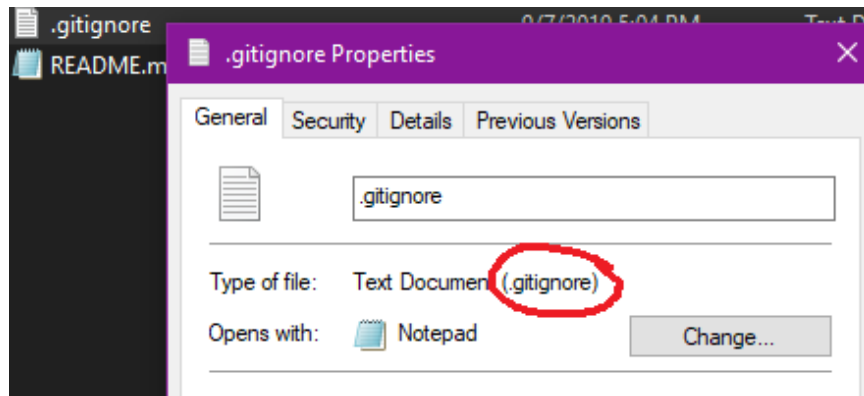
Find the folder you created the git clone, then click “Save As type”, then click “All Files(*.*)”



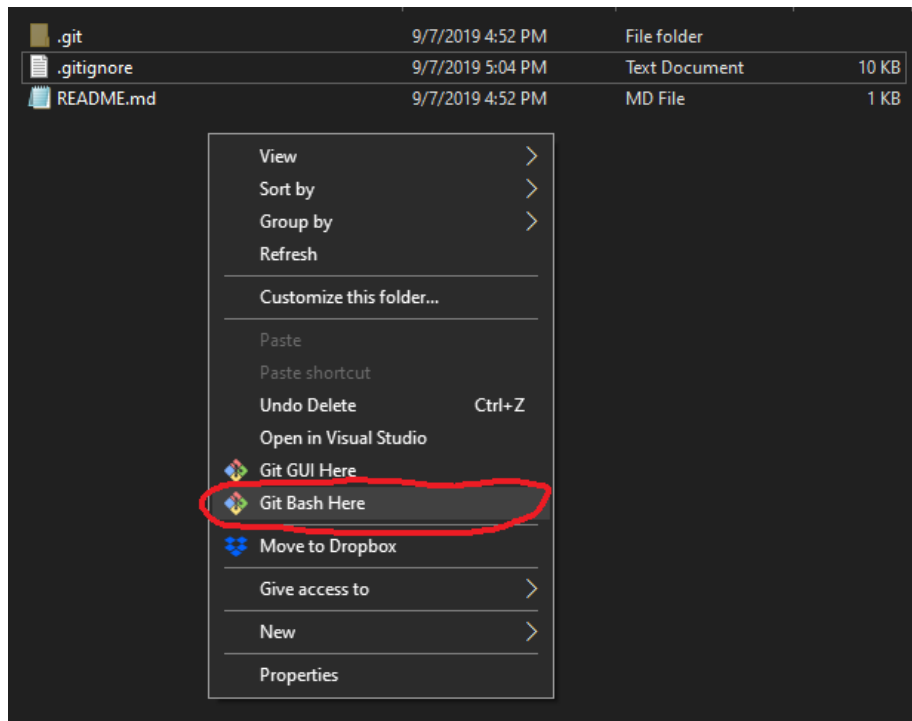
You’ll want to just call it “.gitignore”. No extension or anything, that is the entire name. Once you save, you can close the notepad and go back to the folder to confirm that .gitignore was created



In the folder, you should see that there is no extension. To confirm, you can right click the .gitignore and click “Properties”. It should say “Type of File:_____(.gitignore)”



As shown above, means we're good on the gitignore creation and can now push it to the Repository before we do anything else, that way all of the projects won't push files/folders that we do not want to upload. Go back to the git bash interface (if you closed it, you can simply right click within the folder and click "Git Bash Here")



This should open the Git bash interface.

```
MINGW64:/c/Users/Daniel/Desktop/cpts321-fall2019
Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (master)
$ 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)

        .gitignore

nothing added to commit but untracked files present (use "git add" to track)
Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (master)
$ git add .
Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (master)
$ git commit -m 'Adding gitignore'
[master ab3774b] Adding gitignore
1 file changed, 512 insertions(+)
create mode 100644 .gitignore
Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 3.18 KiB | 3.18 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://gitlab.eecs.wsu.edu/[redacted]/cpts321-fall2019.git
fd661c8..ab3774b master -> master
Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (master)
$ |
```

Once it is open, you can do the sequence of commands to push it to the repository.

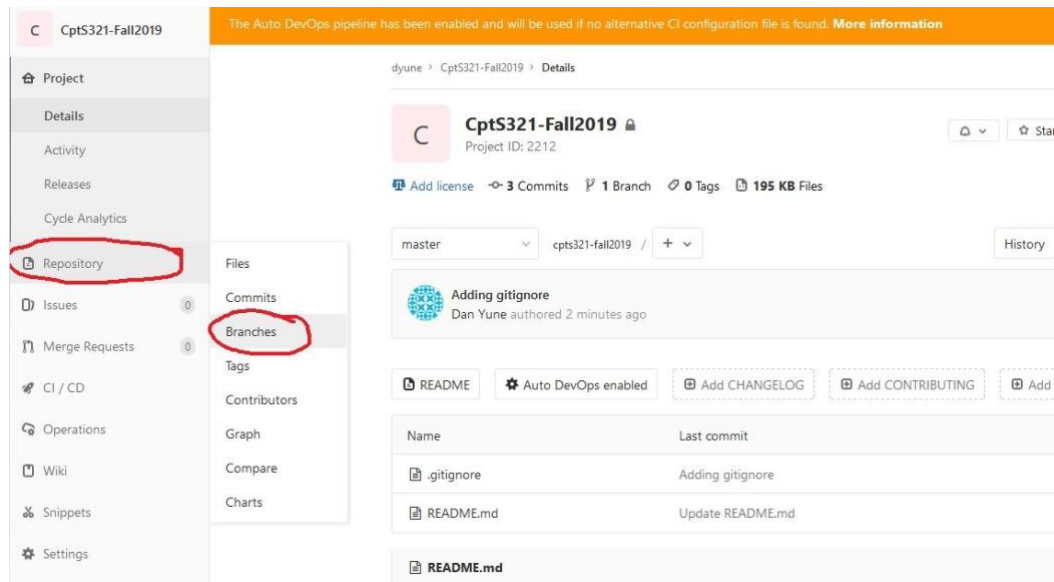
- “git status” (checks to see what files have changed since last pull/push)
- “git add .” (the . denotes to add all files that have changed, created, or deleted to the commit)
- “git commit –m ‘Adding gitignore’” (denotes a message for this push for documentation)
- “git push” (actually pushes the content you have added in step 2 onto the repository)

Name	Last commit	Last update
 .gitignore	Adding gitignore	2 minutes ago
 README.md	Update README.md	29 minutes ago

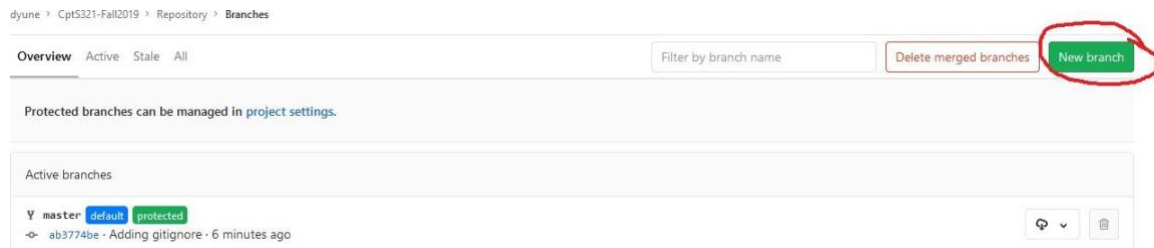
Going back to the repository, refresh the page and you should now see .gitignore there. As you become more familiar with git you can edit the ignore to ignore more file types, or to specifically not ignore something (whether it is one file or a type of file).

Creating a Branch for a Homework (via Gitlab Method):

Now that we have successfully put gitignore, we can now create a branch. There are two ways to do it. The gitlab way is easiest so I'll demonstrate that first. The Bash (second) way is technically faster and still easy, but not as user friendly to someone unfamiliar with git.



On the gitlab site, hover over “Repository” on the left and click “Branches”.



On the top right, you will see “New Branch”, click that

dyune > CptS321-Fall2019 > New Branch

New Branch

Branch name

HW1

Create from

master

Existing branch name, tag, or commit SHA

Create branch

You can name it anything you want, but for this we will call it “HW1”. After you name it, click “Create branch”

dyune > CptS321-Fall2019 > Repository

You pushed to HW1 just now

Create merge request

HW1

cpts321-fall2019

+

History

Find file

Web IDE

⌵



Adding gitignore

Dan Yune authored 9 minutes ago



ab3774be



Name	Last commit	Last update
.gitignore	Adding gitignore	9 minutes ago
README.md	Update README.md	36 minutes ago

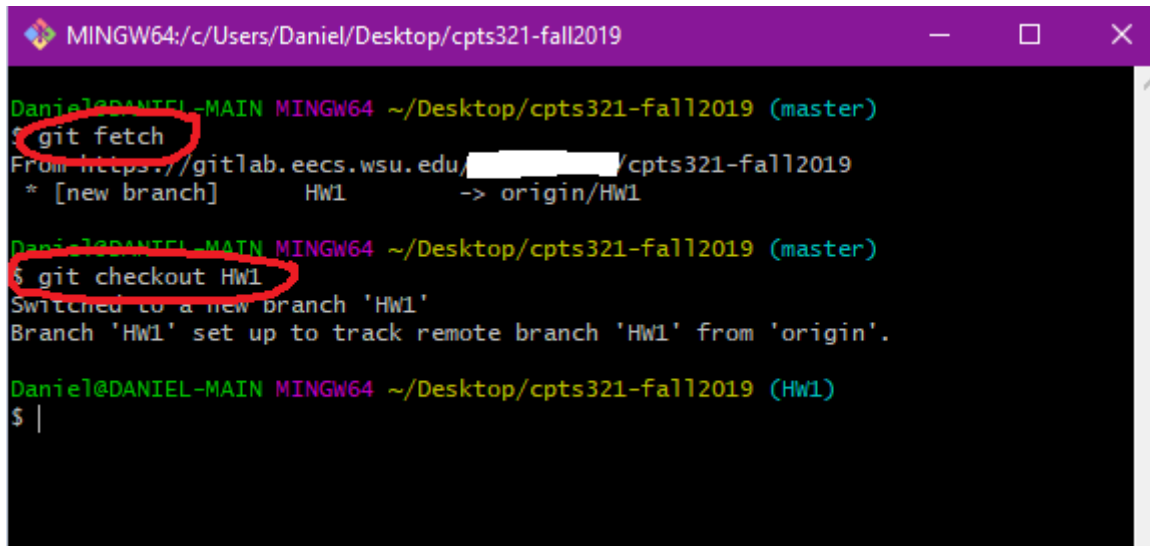
README.md

CptS321-Fall2019

Daniel

11111111

Result should be very familiar. The only difference is that in the top left (circled), it should say “HW1” (or whatever name you used). This means you are in the branch now and it has copied master, so the readme and gitignore have been copied over. You can edit the Readme to say (“Homework 1”) or anything you like, not required though.

A screenshot of a Windows terminal window with a purple title bar. The title bar text is "MINGW64:/c/Users/Daniel/Desktop/cpts321-fall2019". The terminal shows a user named Daniel at a machine named DANIEL-MAIN, in a directory ~/Desktop/cpts321-fall2019, on the master branch. The user enters the command "git fetch", which is circled in red. The output shows a new branch "HW1" being created from "origin/HW1". The user then enters "git checkout HW1", also circled in red. The output shows the user has switched to the new branch "HW1". The prompt then changes to "(HW1)".

```
Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (master)
$ git fetch
From https://gitlab.eecs.wsu.edu/[redacted]/cpts321-fall2019
* [new branch]      HW1      -> origin/HW1

Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (master)
$ git checkout HW1
Switched to a new branch 'HW1'
Branch 'HW1' set up to track remote branch 'HW1' from 'origin'.

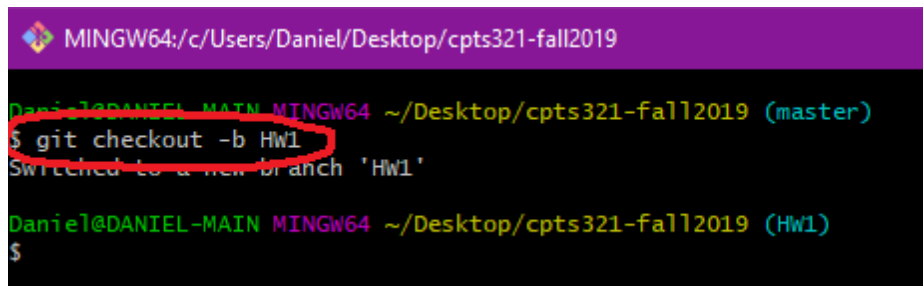
Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (HW1)
$ |
```

Now go back to the bash interface. If you closed it, instructions above on how to open it. Use the following two commands:

- “git fetch” (will retrieve new information regarding the repository that differs from local)
- “git checkout HW1” (this will switch from master to the newly created branch “HW1”, if you used a different name, use that name instead of HW1)

Creating a Branch for Homework (via Bash Method):

**** If you have already made a branch via Gitlab method, you should skip this section! ****

A screenshot of a Windows command prompt window. The title bar is purple and contains the text 'MINGW64:/c/Users/Daniel/Desktop/cpts321-fall2019'. The terminal shows the following text: 'Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (master)' followed by '\$ git checkout -b HW1'. The command 'git checkout -b HW1' is circled in red. Below it, the text 'Switched to a new branch 'HW1'' is displayed. The prompt then changes to 'Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (HW1)' followed by '\$'.

From the bash interface, you can type the following command to create the branch:

- “git checkout -b HW1” (shorthand to create the branch and ‘checkout’ the branch)
 - checkout means to switch to the branch

As per usual, HW1 can be interchanged with any name.

Do note that the branch is only created locally, it will not be shown on gitlab’s website as of yet. We will get to that in a bit.

****Disclaimer: There will be an additional command using this method, will demonstrate as we go on****

After Branch Creation:

```
Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (HW1)
```

However you did it, your git bash should say (HW1) on the right, demonstrating that you are in the HW1 branch.

Configure your new project

Console App (.NET Framework) C# Windows Console

Project name
SampleProgram

Location
C:\Users\Daniel\source\repos

Solution name ⓘ
SampleProgram

☐ Place solution and project in the same directory

Framework
.NET Framework 4.7.2

Now, you can just Browse for the location of your git folder and create a new project directly there.

Name	Date modified	Type	Size
.git	9/7/2019 5:51 PM	File folder	
HW1	9/7/2019 5:50 PM	File folder	
.gitignore	9/7/2019 5:43 PM	Text Document	10
README.md	9/7/2019 5:43 PM	MD File	1

```
MINGW64:/c:/Users/Daniel/Desktop/cpts321-fall2019
Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (HW1)
$ git status
On branch HW1
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  HW1/

nothing added to commit but untracked files present (use "git add" to track)
Daniel@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (HW1)
$
```

Once you make a new project or copy the original project into the folder, you can type “git status” and see that it should be written in Red, meaning “this is new.”

```
MINGW64:/c/Users/Daniel/Desktop/cpts321-fall2019
Danie1@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (HW1)
$ git push --set-upstream origin HW1
Enumerating objects: 25, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 16 threads
Compressing objects: 100% (22/22), done.
Writing objects: 100% (24/24), 10.43 KiB | 1.49 MiB/s, done.
Total 24 (delta 2), reused 0 (delta 0)
remote:
remote: To create a merge request for HW1, visit:
remote: https://gitlab.eecs.wsu.edu/2018080827/cpts321-fall2019/merge_requests/new?merge_request%5Bsource_branch%5D=HW1
remote:
To https://gitlab.eecs.wsu.edu/[redacted]/cpts321-fall2019.git
* [new branch] HW1 -> HW1
Branch 'HW1' set up to track remote branch 'HW1' from 'origin'.
Danie1@DANIEL-MAIN MINGW64 ~/Desktop/cpts321-fall2019 (HW1)
$ |
```

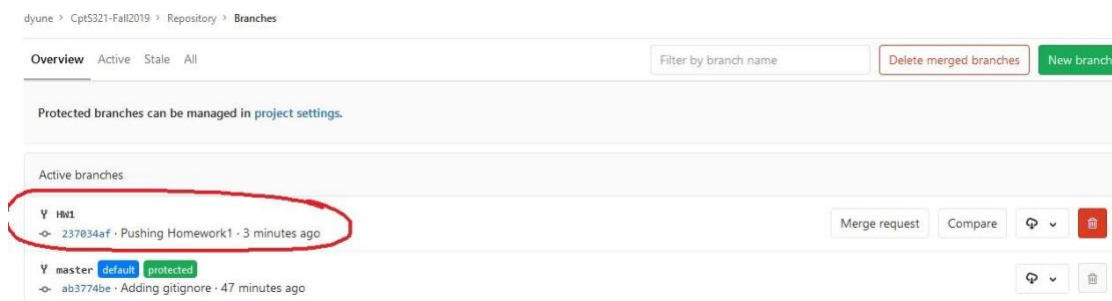
We can then go through the regular procedure of:

- “git add .”
- “git commit -m ‘your commit message goes here’”

This is where the next command differs. If you did method 1 (via gitlab), you can simply do “git push”

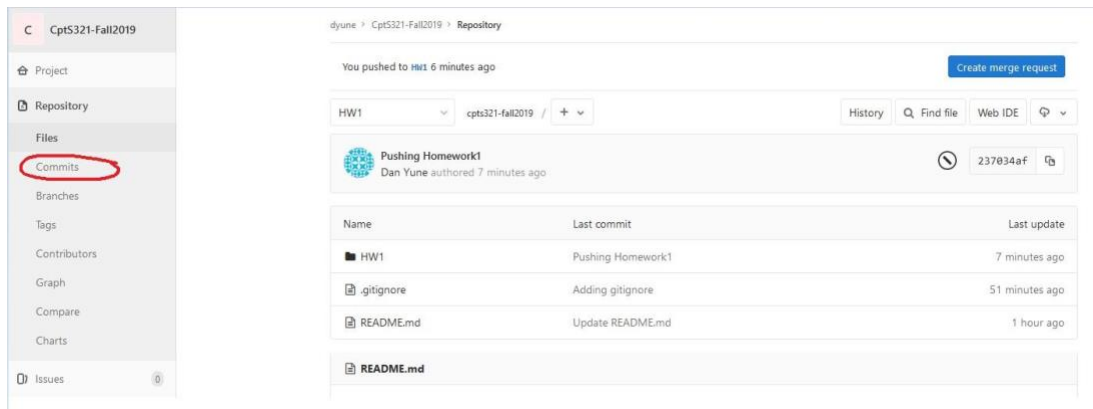
If you did method 2 (via Bash), you need to set an upstream, which is just typing a little extra

- git push --set-upstream origin HW1 (basically telling HW1 who the big boss is)
 - There are two dashes before “set”



Should now be able to see that there is an active branch called HW1. Regardless of which method you used, the result will be the same, a newly created branch with the proper .gitignore and the project of yours.

Tagging: your code is considered submitted if it is tagged and a link to the tag is submitted on Canvas!

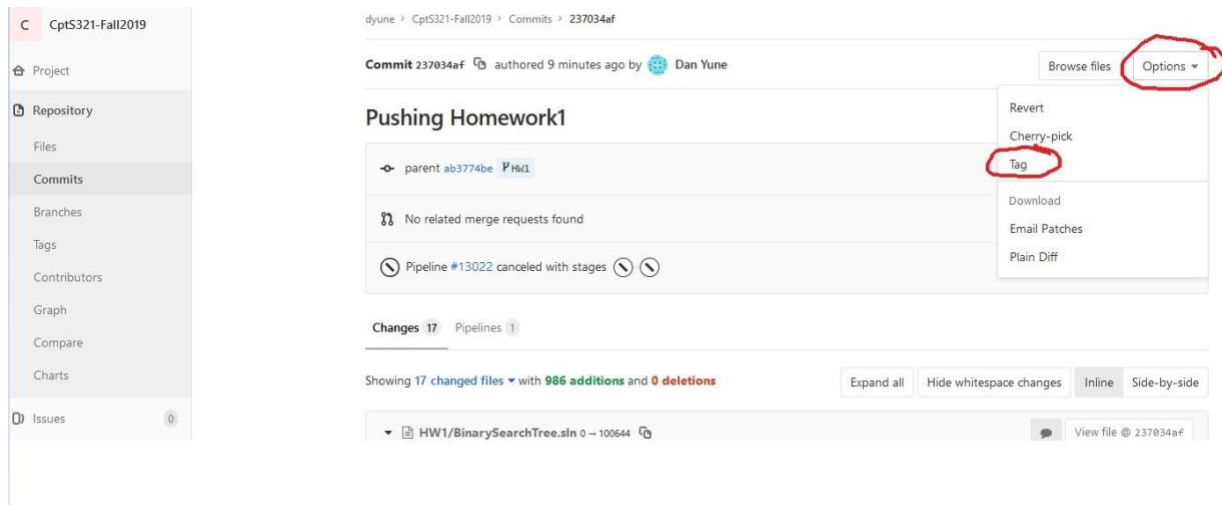


For simplicity, I'm only going to show tagging via Gitlab. You can look up how to do it via Bash, which is very simple.

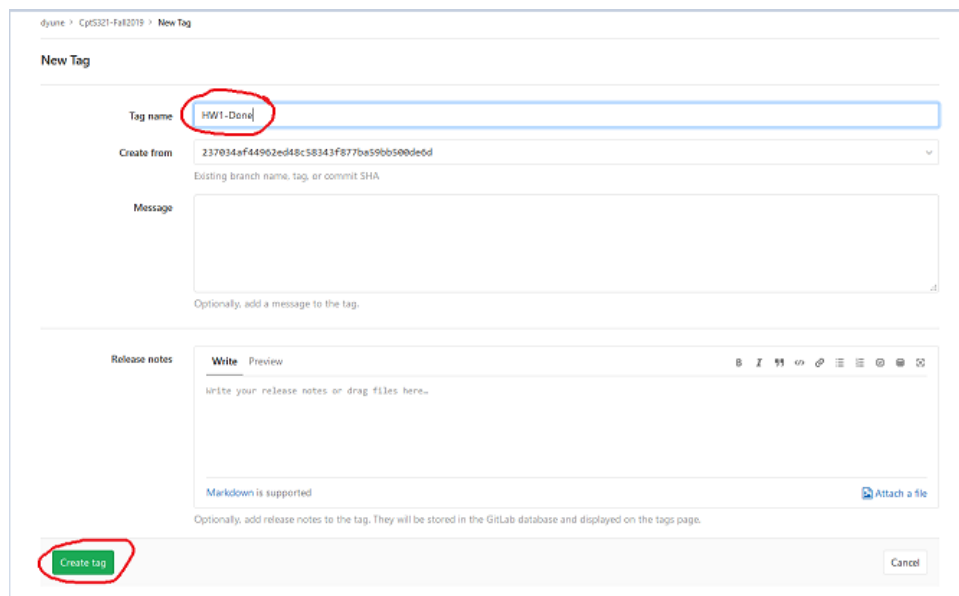
Click the "Commits" on the left toolbar.



Click the homework commit you want graded, typically it will be the latest one. There's probably a reason or another that you'd tag a different one, such as you are experimenting with the HW and the last stable copy was a previous commit, just in case you forget to revert the experimenting changes.

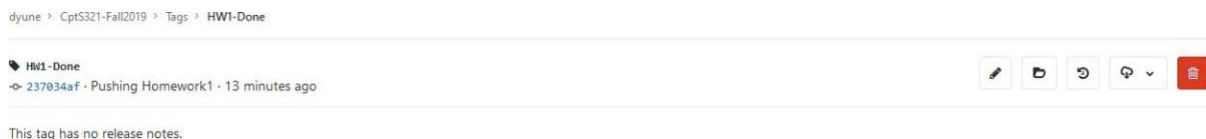


Once you click the link, you can see this interface. Click the top right “Options”, and click “Tag”



Name the tag something very obvious (e.g., “HW1-Done”) so that we know what to grade. The hyphen is put in because tags must be without spaces.

Click “Create tag”



You should now see that it was tagged as “HW1-Done”, and that is how we TAs will find which homework to grade.

Ending:

Thank you for taking the time to go over this and I hope that this was informative. This will be a nice little starting point to using git at the most basic level, and as you become more familiar, you can explore the other options.

We TAs typically have about 100 students' assignments to grade, so we would appreciate it if you could go through this procedure for every homework, it just makes it easier for us. The majority of us are also students with our own assignments to do, so the easier this is for us, the better it is.

If you are experiencing any git related issues, then please attend office hours, as listed in the Syllabus, post a question on the discussion board on Canvas, or send Venera and all the TAs an email. There are some very dangerous git commands that can permanently break your repo if you do not know what you are doing and cause a loss of work.