HOMEWORK 5
Due Monday 02/13 at 11:45pm
This homework has 2 parts, written homework and scripting.

Part1

CSE 15L Homework 5

1. You work for a software company, and before you leave to go home you pushed your latest code to the remote repository. Assuming there were people still working and pushing code to the remote repository after you leave, what single git command do you need to enter in order to update your local repository (on the master branch) when you return in the morning?

2. What are the three steps of the git workflow to perform when you have a new file you want to include as part of your repository?

3. True or False. Your code on your local repository is visible to other programmers.

4. True or False. Only you can access the branches that you created in the remote repository.

5. Charly and Danny are working in parallel on a coding project from different computers, but sharing the same remote repository. Charly just changed several files and performed a 'git commit' in his working directory. Danny then performs a 'git pull'. Can Danny see the changes that Charly did? Explain your reasonings.

6. While programming in java, whenever you compile your code you create a massive amount of ".class" files. You want your repository to be as maintained and clean as possible, so you want to avoid ".class" files being added to your repository, while ensuring your 'git status' will not consider them when displaying files to be added or comitted.

   (a) What file do you need to edit?

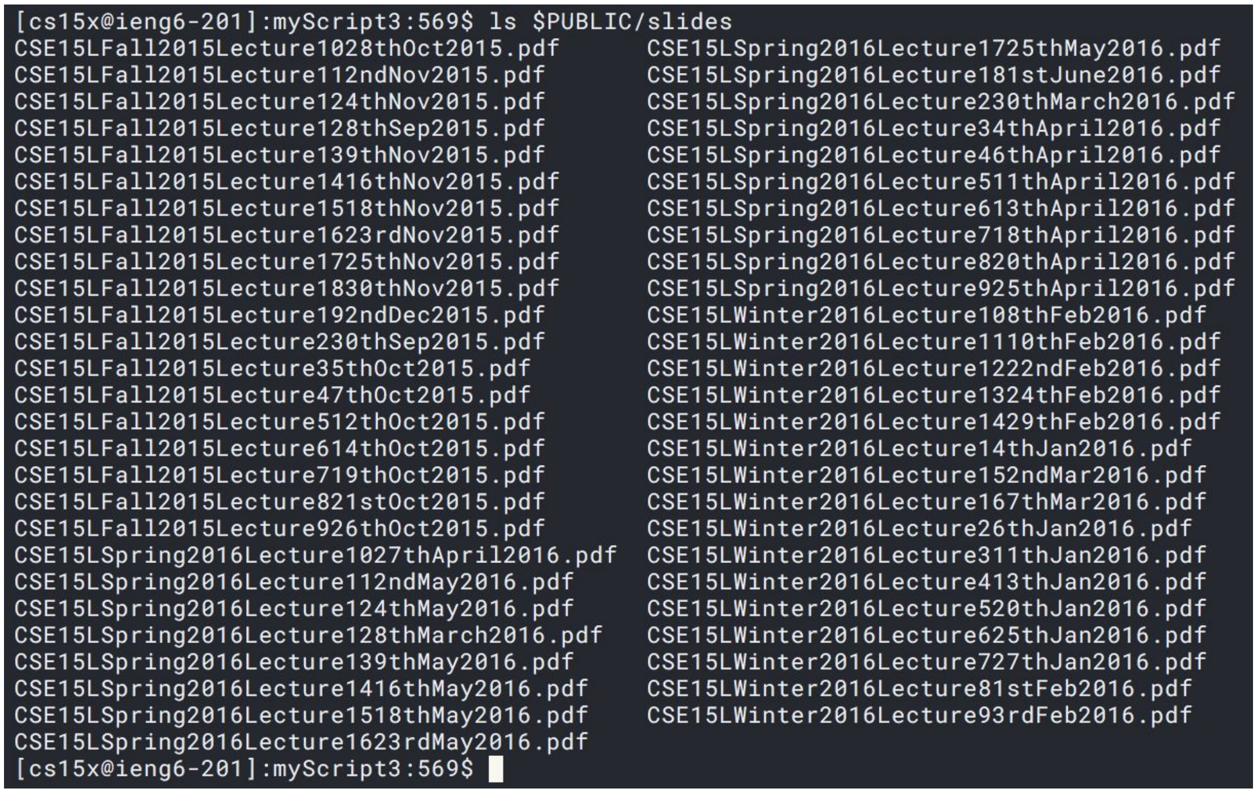
   (b) What is the one line you need to add to that file to ignore all the ".class" files?

Part2

Scripting 3 Instructions

The Premise:

Imagine you are the instructor of CSE 15L at UCSD, and you have just finished the school year of 2015-2016. It's that time of year when you wrapped up your year's work and keep all the old work files archived in an organized fashion. But when you take a look at your working directory, you have a bad feeling about this:



```
[cs15x@ieng6-201]:myScript3:569$ ls $PUBLIC/slides
CSE15LFall2015Lecture1028thOct2015.pdf      CSE15LSpring2016Lecture1725thMay2016.pdf
CSE15LFall2015Lecture112ndNov2015.pdf       CSE15LSpring2016Lecture181stJune2016.pdf
CSE15LFall2015Lecture124thNov2015.pdf       CSE15LSpring2016Lecture230thMarch2016.pdf
CSE15LFall2015Lecture128thSep2015.pdf       CSE15LSpring2016Lecture34thApril2016.pdf
CSE15LFall2015Lecture139thNov2015.pdf       CSE15LSpring2016Lecture46thApril2016.pdf
CSE15LFall2015Lecture1416thNov2015.pdf      CSE15LSpring2016Lecture511thApril2016.pdf
CSE15LFall2015Lecture1518thNov2015.pdf      CSE15LSpring2016Lecture613thApril2016.pdf
CSE15LFall2015Lecture1623rdNov2015.pdf      CSE15LSpring2016Lecture718thApril2016.pdf
CSE15LFall2015Lecture1725thNov2015.pdf      CSE15LSpring2016Lecture820thApril2016.pdf
CSE15LFall2015Lecture1830thNov2015.pdf      CSE15LSpring2016Lecture925thApril2016.pdf
CSE15LFall2015Lecture192ndDec2015.pdf       CSE15LWinter2016Lecture108thFeb2016.pdf
CSE15LFall2015Lecture230thSep2015.pdf       CSE15LWinter2016Lecture1110thFeb2016.pdf
CSE15LFall2015Lecture35thOct2015.pdf        CSE15LWinter2016Lecture1222ndFeb2016.pdf
CSE15LFall2015Lecture47thOct2015.pdf        CSE15LWinter2016Lecture1324thFeb2016.pdf
CSE15LFall2015Lecture512thOct2015.pdf       CSE15LWinter2016Lecture1429thFeb2016.pdf
CSE15LFall2015Lecture614thOct2015.pdf       CSE15LWinter2016Lecture14thJan2016.pdf
CSE15LFall2015Lecture719thOct2015.pdf       CSE15LWinter2016Lecture152ndMar2016.pdf
CSE15LFall2015Lecture821stOct2015.pdf       CSE15LWinter2016Lecture167thMar2016.pdf
CSE15LFall2015Lecture926thOct2015.pdf       CSE15LWinter2016Lecture26thJan2016.pdf
CSE15LSpring2016Lecture1027thApril2016.pdf  CSE15LWinter2016Lecture311thJan2016.pdf
CSE15LSpring2016Lecture112ndMay2016.pdf     CSE15LWinter2016Lecture413thJan2016.pdf
CSE15LSpring2016Lecture124thMay2016.pdf     CSE15LWinter2016Lecture520thJan2016.pdf
CSE15LSpring2016Lecture128thMarch2016.pdf   CSE15LWinter2016Lecture625thJan2016.pdf
CSE15LSpring2016Lecture139thMay2016.pdf     CSE15LWinter2016Lecture727thJan2016.pdf
CSE15LSpring2016Lecture1416thMay2016.pdf    CSE15LWinter2016Lecture81stFeb2016.pdf
CSE15LSpring2016Lecture1518thMay2016.pdf    CSE15LWinter2016Lecture93rdFeb2016.pdf
CSE15LSpring2016Lecture1623rdMay2016.pdf
[cs15x@ieng6-201]:myScript3:569$
```

Now you wish you had named your slides a little more friendly, don't you?

"No worries!" You say to yourself, "I will just use my CSE 15L knowledge to make my life easy!"

Get Started:

Make sure you are logged into your cs15x?? Account.

```
cd ~
mkdir myScript3
cp ~/../public/scripting/script3/script3.sh ~/myScript3
cp ~/../public/scripting/script3/Makefile ~/myScript3
cd ~/myScript3
```

We have provided you with a string array. Use this for file matching and creating directories:

```
MONTHS=(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec)
```

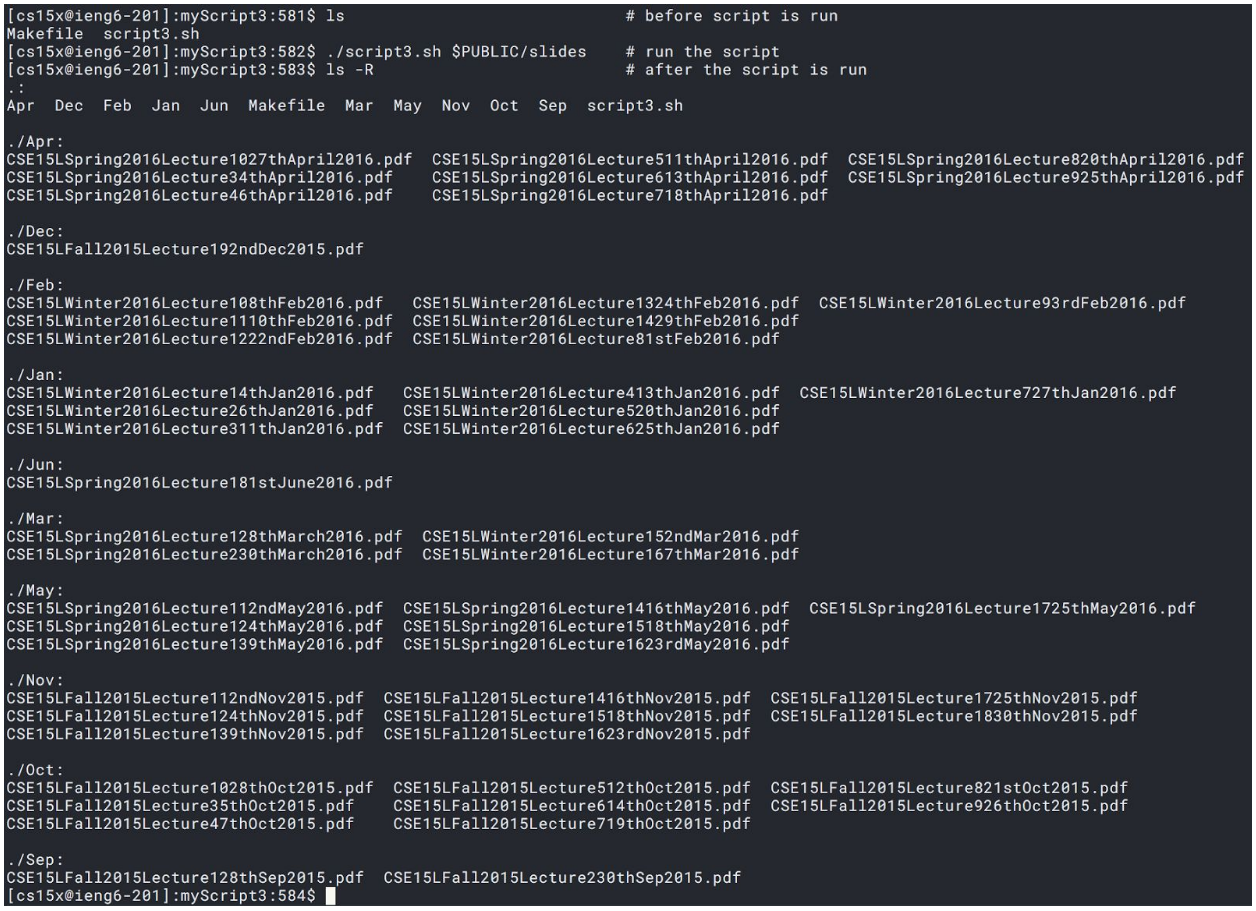
Your objective is to write a shell script that allows yourself to pass an argument that represents the directory where the slides are, and copy all the slides into your current directory where your script is. While doing so, categorize each slide into a directory of corresponding month. Namely, after your script is run, you should be able to find a slide named **`CSE15LFall2015Lecture1028thOct2015.pdf`** under a directory named **`Oct/`**, and so on.

Notice that if a directory of a month does not exist, you should create it with **`mkdir`**. Otherwise, you should not attempt to make this directory again. Your script should not have any error output as long as it's run correctly.

DO NOT ATTEMPT TO HARD CODE YOUR ANSWER. It will be very obvious and sad if you hard code the files you need to copy, and on top of that, **YOU WILL RECEIVE NO CREDIT**.

Sample Result:

After successful execution of your script, all the slides should be organized inside your directory.



```
[cs15x@ieng6-201]:myScript3:581$ ls                            # before script is run
Makefile  script3.sh
[cs15x@ieng6-201]:myScript3:582$ ./script3.sh $PUBLIC/slides    # run the script
[cs15x@ieng6-201]:myScript3:583$ ls -R                         # after the script is run
.:
Apr  Dec  Feb  Jan  Jun  Makefile  Mar  May  Nov  Oct  Sep  script3.sh

./Apr:
CSE15LSpring2016Lecture1027thApril2016.pdf  CSE15LSpring2016Lecture511thApril2016.pdf  CSE15LSpring2016Lecture820thApril2016.pdf
CSE15LSpring2016Lecture34thApril2016.pdf    CSE15LSpring2016Lecture613thApril2016.pdf  CSE15LSpring2016Lecture925thApril2016.pdf
CSE15LSpring2016Lecture46thApril2016.pdf    CSE15LSpring2016Lecture718thApril2016.pdf

./Dec:
CSE15LFall2015Lecture192ndDec2015.pdf

./Feb:
CSE15LWinter2016Lecture108thFeb2016.pdf   CSE15LWinter2016Lecture1324thFeb2016.pdf  CSE15LWinter2016Lecture93rdFeb2016.pdf
CSE15LWinter2016Lecture1110thFeb2016.pdf  CSE15LWinter2016Lecture1429thFeb2016.pdf
CSE15LWinter2016Lecture1222ndFeb2016.pdf  CSE15LWinter2016Lecture81stFeb2016.pdf

./Jan:
CSE15LWinter2016Lecture14thJan2016.pdf   CSE15LWinter2016Lecture413thJan2016.pdf  CSE15LWinter2016Lecture727thJan2016.pdf
CSE15LWinter2016Lecture26thJan2016.pdf   CSE15LWinter2016Lecture520thJan2016.pdf
CSE15LWinter2016Lecture311thJan2016.pdf  CSE15LWinter2016Lecture625thJan2016.pdf

./Jun:
CSE15LSpring2016Lecture181stJune2016.pdf

./Mar:
CSE15LSpring2016Lecture128thMarch2016.pdf  CSE15LWinter2016Lecture152ndMar2016.pdf
CSE15LSpring2016Lecture230thMarch2016.pdf  CSE15LWinter2016Lecture167thMar2016.pdf

./May:
CSE15LSpring2016Lecture112ndMay2016.pdf  CSE15LSpring2016Lecture1416thMay2016.pdf  CSE15LSpring2016Lecture1725thMay2016.pdf
CSE15LSpring2016Lecture124thMay2016.pdf  CSE15LSpring2016Lecture1518thMay2016.pdf
CSE15LSpring2016Lecture139thMay2016.pdf  CSE15LSpring2016Lecture1623rdMay2016.pdf

./Nov:
CSE15LFall2015Lecture112ndNov2015.pdf  CSE15LFall2015Lecture1416thNov2015.pdf  CSE15LFall2015Lecture1725thNov2015.pdf
CSE15LFall2015Lecture124thNov2015.pdf  CSE15LFall2015Lecture1518thNov2015.pdf  CSE15LFall2015Lecture1830thNov2015.pdf
CSE15LFall2015Lecture139thNov2015.pdf  CSE15LFall2015Lecture1623rdNov2015.pdf

./Oct:
CSE15LFall2015Lecture1028thOct2015.pdf  CSE15LFall2015Lecture512thOct2015.pdf  CSE15LFall2015Lecture821stOct2015.pdf
CSE15LFall2015Lecture35thOct2015.pdf    CSE15LFall2015Lecture614thOct2015.pdf  CSE15LFall2015Lecture926thOct2015.pdf
CSE15LFall2015Lecture47thOct2015.pdf    CSE15LFall2015Lecture719thOct2015.pdf

./Sep:
CSE15LFall2015Lecture128thSep2015.pdf  CSE15LFall2015Lecture230thSep2015.pdf
[cs15x@ieng6-201]:myScript3:584$ 
```

Testing:

```
cd ~/myScript3                      # navigate to your script
make clean                          # start with clean slate
./script3.sh ~/../public/slides     # run your script
ls -R                               # make sure that output matches above
```

`ls -R` recursively prints out all the content within the current directory. You may use this command to check if you script is working correctly.

Doing this will likely produce a bunch of directories named after months, to clean them up before running a test, you can simply type

```
make clean
```

Useful Tips:

1. The **`for...in`** Loop:
   If you used the C-style for loop to draw the chessboard in the last assignment, you will find that the same technique does not apply very easily when you wish to iterate through a list of files, or an array. The **`for...in`** loop comes in very handy for that purpose. For xample:

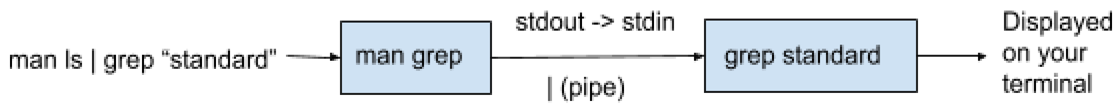
   ```
   for cat in {Garfield,Tom,Felix,Scratchy}
   do
        echo $cat
   done
   ```

2. Piping and grepping
   Piping or ‘ | ’ allow you to redirect the standard output of a command into the standard in of another command, and **`grep`** is a very useful command line utility that filters text with a matching pattern. For example:

   ```
   man ls | grep standard
   ```

   `man ls` will display the manual of the command **`ls`** to your standardout.
   `man ls | grep standard` will take the standardout from, and instead of displaying it on your terminal, it will feed it into grep as the input or standard-in. This command will return any lines inside the manual that has the word “standard”.

Here is a visualization



3. **`test`** and conditions

Conditions in shell scripting may appear unpredictable at first, but once you get it, it's quite simple. The structure looks like this

```
if condition-cmd; then
    if-branch-cmd
else
    else-branch-cmd
fi
```

Where **`if`** will check the **`condition-cmd`**'s exit code. If the exit code is 0, then the script will take the **`if`** branch; otherwise it will take the **else** branch.

Conditions in shell script is often used with **`test`**, which based on its command line inputs, either exits with 0 when condition is true, or 1 when condition is false (kind of counter-intuitive, but that's how shell scripting works).

Spend some time reading the man page for **`test`**, and play with it for some time, and see which configurations are applicable to the task that you need to do for this assignment.

Because test is so often used with **`if`**, square brackets are often used in place of **`test`**. For example:

```
if [ 1 -lt 2 ]; then
    echo TRUE
fi
```

Will be equivalent to

```
if test 1 -lt 2; then
    echo TRUE
fi
```

Beware that when you are using the square brackets notation, there needs to be spaces between the brackets and **`test`**'s arguments.

How to Submit

When submitting your homework, make sure the part1 homework is completed separately in its own file. Do not complete part1 within the script.

When you are ready to submit, make sure both part1 and part2 files stored within the **`myScript3`** directory and run the following script:

```
cd ~/myScript3
turnin
```