

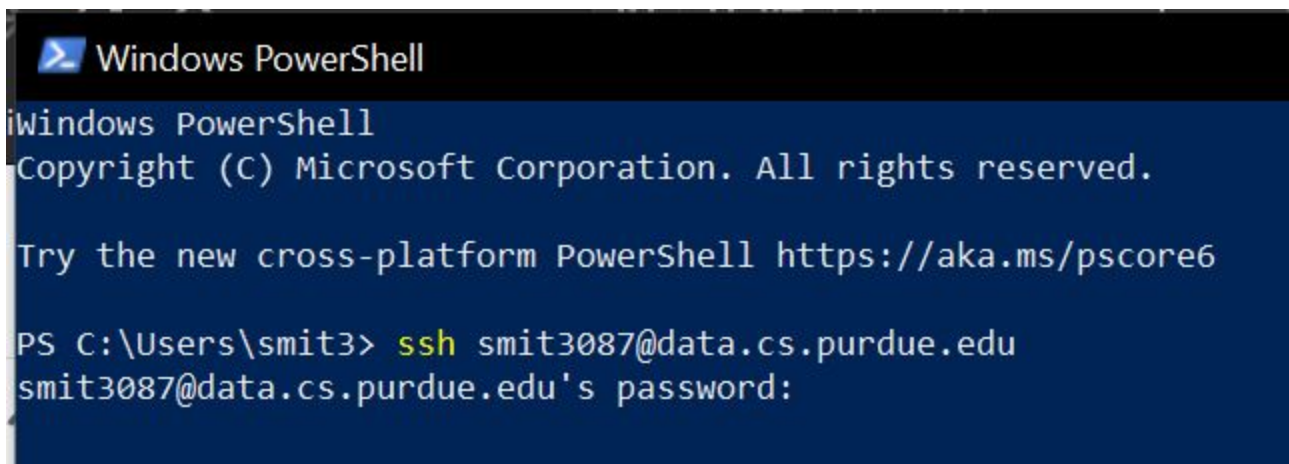
# CS193HW2: Command Line Instructions

## Step 1: Clone this repository

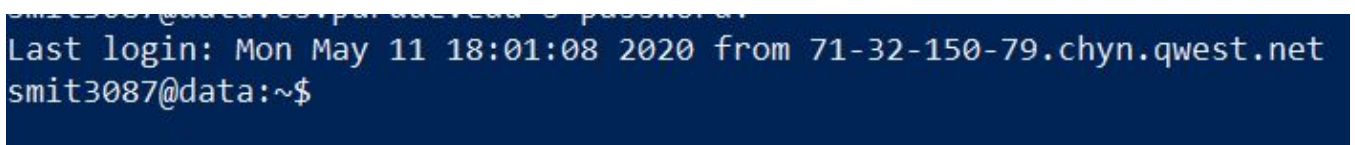
Open the command prompt on your local machine. If you are using your personal laptop, type the command:

ssh [your\\_purdue\\_username@data.cs.purdue.edu](https://github.com/Purdue-CS193/CS193HW2.git)

Into the command prompt. Hit enter to run the command. You will be presented with a password prompt:

A screenshot of a Windows PowerShell terminal window. The title bar says "Windows PowerShell". The text inside the terminal shows the standard PowerShell startup messages: "Windows PowerShell", "Copyright (C) Microsoft Corporation. All rights reserved.", and "Try the new cross-platform PowerShell https://aka.ms/pscore6". Below this, the user has entered the command "ssh smit3087@data.cs.purdue.edu". The prompt now shows "smit3087@data.cs.purdue.edu's password:".

Enter your Purdue password in the prompt. No text will appear after the prompt as you type; this is normal! Hit enter when you're done, and if you've entered it incorrectly you'll be prompted for a password again. If not, you'll see something like this:

A continuation of the terminal screenshot. It shows the login process completing. The text "Last login: Mon May 11 18:01:08 2020 from 71-32-150-79.chyn.qwest.net" is displayed. The prompt now shows "smit3087@data:~\$".

From here, you can create or navigate to a folder where you'd like to put the homework document using the mkdir or cd commands, respectively. Once you've reached a place in your directory where you'd like to put the homework, run the command:

git clone <https://github.com/Purdue-CS193/CS193HW2.git>

If the clone is successful, you'll see the following:

```
smit3087@data:~$ cd cs193-2019
smit3087@data:~/cs193-2019$ git clone https://github.com/Purdue-CS193/CS193HW2.git
Cloning into 'CS193HW2'...
remote: Enumerating objects: 112, done.
remote: Counting objects: 100% (112/112), done.
remote: Compressing objects: 100% (96/96), done.
remote: Total 112 (delta 37), reused 59 (delta 12), pack-reused 0
Receiving objects: 100% (112/112), 5.01 MiB | 29.49 MiB/s, done.
Resolving deltas: 100% (37/37), done.
smit3087@data:~/cs193-2019$
```

And if you run the command **ls**, you should see a directory called CS193HW2. Cd into that directory using the command **cd CS193HW2** (the directory name is case-sensitive!). In that directory, you'll see multiple files and subdirectories. The **README** file contains the instructions for this assignment (the document you're looking at right now!). The **src** subdirectory contains all of the Java files you'll need to edit/debug for this assignment. Navigate to this directory using **cd src** and inspect the java files using whichever text editor you prefer (nano, vim, etc). The **Questions.java** file is the one you'll be editing; the **TestCases.java** contains code to test the code you edit.

## Step 2: Running and debugging JUnit testcases

### Compiling

Download the file named 'junit-platform-console-standalone-1.7.0-M1.jar' at the link [here](#).

To compile the test cases, the file you just downloaded must be located in the same repository where the test cases are found. To do this, you'll need to access the file on your local machine and send it to your remote machine.

If you ssh'd into a remote machine via your local command line, run the command '**exit**' with no arguments to exit to your local machine.

If you've ssh'd via a tool (like PuTTY) instead of your command line, open your command line (Powershell or Command Prompt on Windows, and Command Line on macOS/Linux).

Once you've accessed your local computer's command line, you need to copy the downloaded file to your remote machine. To get the file from your local to your remote computer, run the command:

```
scp <local_filepath> <your-purdue-username>@data.cs.purdue.edu:<your_testcase_filepath>
```

For instance, if your local file is stored in your Downloads folder, the local file path would be <Downloads/junit-platform-console-standalone-1.7.0-M1.jar>. Your test cases will be located at CS193HW2/src. If you've cloned the repo into a subdirectory, make sure to include the full path name, starting from your home directory.

If the scp command has run correctly, you will be prompted to login to your account on the data server (just like when you ssh'd!). After logging in, the file should be copied to the directory which contains

your test cases. Navigate to that directory and run 'ls' to ensure that the file exists in the correct location.

Once the file is in the correct location, run the command

**javac -d out -cp out:junit-platform-console-standalone-1.7.0-M1.jar TestCases.java Questions.java**

To compile the test cases. After compiling, a new executable named 'out' should exist in your directory. If there are no error messages, run the command:

**java -jar junit-platform-console-standalone-1.7.0-M1.jar --class-path out --scan-class-path**

To run the test cases. The output may look confusing, but the most important information is located at the bottom. It should look something like:

```
Test run finished after 38 ms
[      3 containers found      ]
[      0 containers skipped    ]
[      3 containers started    ]
[      0 containers aborted    ]
[      3 containers successful ]
[      0 containers failed     ]
[     10 tests found           ]
[      0 tests skipped         ]
[     10 tests started         ]
[      0 tests aborted         ]
[      0 tests successful      ]
[     10 tests failed          ]
```

The bottom two lines show the number of tests succeeded and failed. To further inspect how each test has behaved, scroll up:

```
[...]
JUnit Vintage:TestCases:testSumBetween193
MethodSource [className = 'TestCases', methodName = 'testSumBetween193', methodParameterTypes = '']
=> java.lang.AssertionError: expected:<5> but was:<0>
    org.junit.Assert.fail(Assert.java:89)
    org.junit.Assert.failNotEquals(Assert.java:835)
    org.junit.Assert.assertEquals(Assert.java:647)
    org.junit.Assert.assertEquals(Assert.java:633)
    TestCases.testSumBetween193(TestCases.java:66)
    java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    java.base/java.lang.reflect.Method.invoke(Method.java:566)
    org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:59)
[...]
```

This output exists for each test. The first line displays the name of the test, and the third line shows the expected and the actual output. In the example above, the expected output was 5 (expected:<5>), while the actual output was 0 (was:<0>).

You can use the output above to logically debug your code, or you can use resources given below to debug failed tests.

## Step 3: Fix those bugs

You can use print statement debugging by manually editing the **Questions.java** file and adding print statements. Use whichever text editor you're most familiar with.

You can also use the Java command-line debugger, JDB, to debug by setting breakpoints and inspecting variables as you step through your code. There is a link below to JDB documentation, which will guide you through its usage.

You might also want to try to understand the algorithms, such as binary search, as they commonly show up in technical interviews (although you're not required to do this). **You are NOT allowed to make any changes to the test case file. If you feel there is a need to make changes, please email your TA or post on Piazza. It will be an automatic 0 if changes are made!**

[Click here for JDB documentation](#)

## Step 4: Push your changes to GitHub!

When you're ready to push your changes to your main repository (GitHub), navigate to your **src** directory, or whichever directory contains your **TestCases** and **Questions** files. Run the command **git add .** to stage all of your changes before committing - DO NOT FORGET the period at the end! If you want to ensure all the changed files have been staged, you can run **git status** to see the status of the added files.

After staging the files, run **git commit -m "Your commit message here"**, and include your unique, descriptive commit message between the quotes of the final argument. PSA: Your TAs and instructors can see your commit messages.

After committing the files, run the command **git push origin your-branch**, where **your-branch** is the name of the branch you're currently on. To verify your current branch name, run the command **git branch** - this will list all of the branches, and your current branch will be marked with a **\***.

As with every other homework, if your code isn't on GitHub we won't be able to grade it! Once you're happy with the fixes you've made to the code (likely when all the test cases pass), save your work to GitHub.

