# ICS 45c Assignment 0

There is only one program to write for this assignment. This does not have to be turned in for credit. You can find excellent cheat-sheets or tutorials on-line using Google. Example searches include “vim command summary” and “linux command summary” if you want to find quick reference sheets. Here is one for vim. Here is another one for vim. Here is one for linux bash commands. Make tutorial.

Do the following to set-up your account for writing and compiling C++ programs on openlab.ics.uci.edu:

1. Login to openlab.ics.uci.edu using putty (you should see it listed under applications). If you have trouble, see a tutor, TA, Instructor, or ask the support group on the 3rd floor of the old ICS building.
2. You are now able to give commands to the “bash” command processor. It prints a prompt and you type a command followed by enter to cause it to be executed.
3. Make a directory named 45c with this command:
   ```
   mkdir 45c
   ```
4. Connect to 45c with cd and make a directory named lab00 then connect to it:
   ```
   cd 45c
   mkdir lab00
   cd lab00
   ```
5. Run vim (the program/text editor) on main.cpp to create a new file:
   ```
   vim main.cpp
   ```
6. Enter the following program, by first entering insert mode with the command i, then type the program, then finish with <esc> to leave insert mode:
   ```
   #include <iostream>
   using namespace std;
   int main()
   {
           cout << “Hello World” << endl;
           return 0;
   }
   ```
7. Get out of insert mode with <esc> (which takes you back to command mode) then write the file and exit vim with **ZZ**
8. Compile the program with the following command:
   ```
   g++ main.cpp
   ```
9. Give the command ls to see the file a.out:
   ```
   ls
   ```
10. Try to run the program a.out, but you will probably get an error “command not found”:
    ```
    a.out
    ```
11. pushd to your home directory ~:
    ```
    pushd ~
    ```
12. Run vim on .bash_profile:
    ```
    vim .bash_profile
    ```
13. Find the string PATH using:
    ```
    /PATH<enter>
    ```

14. Add **:.** to the end of this command with **A:.<esc>**
15. Go to the end of this file (with G) and add these two lines (o for open a new line after current line):

```
source    /opt/Modules/init/bash
module   load    gcc/5.2.0
module   load    gdb/7.11
```

16. <esc> to get back to command mode, then write and exit the file (with ZZ)
17. Source the commands in .bash_profile so they take effect in the current session:
    ```
    source .bash_profile
    ```
18. echo $PATH to ensure that . is in your command path:
    ```
    echo $PATH
    ```
19. Use pushd with no arguments to go back to your lab00 directory:
    ```
    pushd
    ```
20. Use which a.out to find out which command will be executed when you give the command a.out, it should be ./a.out:
    ```
    which a.out
    ```
21. Use which g++ to see what command will run when you type g++
    ```
    which g++
    ```
22. It should say something like /pkg/gcc/5.2.0/bin/g++   (If you don't see the 5.2.0, something is wrong)
23. Run your program with the command a.out (It should run correctly)
24. Now shift gears and recompile with different command line options
    ```
    g++ -ggdb -std=c++11 main.cpp
    ```
25. Run gdb on your program with the command gdb a.out
26. It should print some version information about gdb
27. You can run your program with the command run:
    ```
    run
    ```
28. Now print a list of the commands with the command list:
    ```
    list
    ```
29. You will see line numbers next to the C code lines
30. Set a break on line containing the call to printf (assume it is on line 5 for example)
    ```
     break 5
    ```
31. Run your program again, it will stop before the call to printf:
    ```
    run
    ```
32. Now give the step command and it will execute the printf then stop again:
    ```
    step
    ```
33. Now edit your program to add a declaration of a variable i and initialize it to 40:
    ```
    #include <iostream>
    using namespace std;
    int main()
    {
            int i = 40;
            cout << "Hello\n";
            return 0;
    }
    ```
34. Now recompile your program with g++ -ggdb and run gdb on a.out again

35. List your program and break on the call to operator << on line 6
36. Run the program
37. Now print out the value of i using print (or p for short)
    ```
    p i
    ```
38. Go back to the bash prompt, type up arrow and see what happens. When you get to a safe command (like vim main.cpp), press enter and see what happens. Then find another command (like g++ main.cpp) and press enter to redo that command. Note you can edit the commands before you press enter.
39. Run valgrind on your program:
    ```
    valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --track-origins=yes a.out
    ```
40. Remove the a.out file with the rm command then list the contents of the current working directory with ls
    ```
    rm a.out:
    ls
    ```
41. Create a Makefile for your program (using vim, put the text below in a file named Makefile in your program directory, note the indentation before g++ must be one tab character):
    ```
    main: main.cpp
    	echo ----------------compiling main.ccp to create executable program main--------------------
    	g++ -ggdb -std=c++11 main.cpp -o main
    ```
42. Give the command make to build your program (it should execute the g++ command to build main):
    ```
    Make
    ```
43. Now run your program called main:
    ```
    main
    ```
44. Now add a new rule to your Makefile for clean to clean up the files you no longer need, add this to the bottom of your Makefile (again the indentation must be done with one tab character):
    ```
    clean:
    	echo ----------------removing executable program main--------------------
    	/bin/rm main
    ```
45. Now make clean to remove the executable:
    ```
    make clean
    ```
46. Copy the bash script create_script in my directory into your directory with the cp command (note the dot at the end means the current working directory, so you must be sure to put it in your command)
    ```
    cp   ~klefstad/public_html/public/45c/create_script   .
    ```
47. Next change the protection of this file with the chmod command,
    ```
    chmod a+x create_script
    ```
48. Now add a new rule to the end of your Makefile for test, to build your main program and run it
    ```
    test:
    	create_script main.cpp main
    ```
49. Now make test, then view the file main.txt (either with more, less, or vim)
    ```
    less main.txt
    ```