

Homework 0: Using Linux for Programming

Due before end of Week 1

First register for Piazza using your UCI net ID, but use a different password for added security. the simplest thing to do is to add one character to the end of your other password, but I'll leave that up to you for increased security.

Next read my [advice to students](#).

Do all the following steps in order. This is to set up your account and expose you to the editor (vim), the compiler (g++), debugger (gdb), the build program (make), and the program that will catch your memory leaks (valgrind). You will be using all these programs every week.

Pay attention to what you are doing step. Do not mindlessly do the steps. These are the programs you will be using all term. There is nothing to submit for this homework and it will not be graded, but it is important for you to complete it successfully before you begin on homework 1. There is only one program to write and run for this assignment.

You can find excellent cheat-sheets or tutorials on-line with 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 or C++ programs on openlab.ics.uci.edu:

1. Be sure you have activated your ICS account http://www.ics.uci.edu/~lab/students/acct_activate.php
2. 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. If you have a Mac, you can use the **ssh** command to log into openlab.ics.uci.edu
3. 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.
4. First add a new command to .bash_profile in your login directory. run vim on .bash_profile, then go to the end of the file (with the command G) and open a new line after the current line (with the command o), then insert the line below (you can copy and paste, but in Linux you use highlight to copy and right mouse button to paste). Exit insert mode back to command mode (by pressing the <esc> key) then write the file and exit (with the command ZZ)
source ~klefstad/.bash_profile_fix
5. bash must read these commands to cause them to take effect. One way is to logout then log back in. Or you can source .bash_profile this time thusly, (but only if you stay logged in. .bash_profile will be sourced next time you login in again)
source .bash_profile
6. If you ever destroy one of your initialization files like .bash_profile, you can find out how to restore them yourself [here](#).
7. Now we will create a folder for homework 0 with a program we can compile, run, and debug. **Make a directory named hw0 with this command**
mkdir hw
8. connect to hw with cd and make a directory named hw0 then connect to it

```
cd hw
mkdir hw0
cd hw0
```

9. run vim (the program/text editor) on main.cpp to create a new file

```
vim main.cpp
```

10. 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;
}
```

11. get out of insert mode with <esc> (which takes you back to command mode) then write the file and exit vim with ZZ

12. compile the program with the following command

```
g++ main.cpp
```

13. Give the command ls to see the file a.out

```
ls
```

14. try to run the program a.out, but if you get an error "command not found", you must use ./a.out

15. Now we will use the debugger called gdb. Recompile your program with different command line options

```
g++ -ggdb main.cpp
```

16. run gdb on your program with the command gdb a.out

17. it should print some version information about gdb

18. you can run your program with the command run

```
run
```

19. now print a list of the ~~commands~~ with the command list

```
list
```

20. you will see line numbers next to the C++ code lines

21. set a break on line 5 containing the call to print "Hello World" (assume it is on line 5 for example)

```
break 5
```

22. run your program again, it will stop before the call to print out the message

```
run
```

23. now give the next command and it will execute the print then stop again

```
next
```

24. 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 World\n";
    return 0;
}
```

25. now recompile your program with `g++ -ggdb` and run `gdb` on `a.out` again

26. list your program and break on the call to operator `<<` on line 6

27. run the program

28. now print out the value of `i` using `print` (or `p` for short)

`p i`

29. quit from `gdb` and you will be back back at the bash prompt. Type up arrow and see what happens.

You should see commands that you previously typed. When you get to a safe command (like `vim main.cpp`), press enter and see what happens. This is command history. It saves your commands so you can edit and reuse them. Go back to bash and find another command (like `g++ main.cpp`) and press enter to redo that command. Note you can edit the commands before you press enter.

30. Now we will run `valgrind` to check for memory leaks and other memory access errors. Run `valgrind` on your program, below is a simple command and a complex command to run `valgrind`. Try the simple one first, then try the complex one.

`valgrind a.out`

`valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --track-origins=yes a.out`

31. Remove the `a.out` file with the `rm` command then list the contents of the current working directory with `ls` to verify that it is gone

`rm a.out`

`ls`

32. Now we are going to create a Makefile for your program so you can simply type **make** and it will automatically recompile your program using the correct command. 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, if you use spaces it will not work)

`CXXFLAGS=-ggdb -std=c++11 -Wpedantic -Wall -Wextra -Werror -Wzero-as-null-pointer-constant`

`main: main.cpp`

`g++ $(CXXFLAGS) main.cpp -o main` 

33. Give the command `make` to build your program (it should execute the `g++` command to build `main`)

`make`

34. Now run your program called `main`

`main`

35. Now add a new rule to your Makefile for `clean` to remove 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`

36. Now make `clean` to remove the executable

`make clean`

37. Test it out by making your program. It should give those options to the compile command. You may see more error messages now. Note, never ignore warnings. Fix each one in order.

38. Create a sample report and program file. In the future Homeworks, we want a PDF document for your report, but for this week, you may use the typescript you created in the previous step. The report should include **screenshots** showing you can make the program (using `make` which calls `g++`) and run the program under `valgrind` and that it produces the correct result and that there are no `valgrind` errors, but this typescript will be fine for only this week.

Submit your report and your source code to GradeScope. [Here](#) for more details about submission. Always download your submission again to verify it is correct.