

Assignment 0

1. Learning Goals

1. Learn some basic unix/linux commands.
2. Start using a text editor of your choice.
3. Become familiar with the build process.
4. Modify a simple C program.
5. Learn how to submit your assignments.

2. Logistics

1. All work for this assignment is to be done on one of the department's instructional Unix/Linux machines. You are welcome to remotely login using ssh, putty, etc., but you will have to figure out those details on your own. It is not that hard really.
2. All assignments in this course will be graded only on CS departmental machines (e.g. Galapagos lab at CS first floor, Room No: 1366) running Linux Operating System. It is your responsibility to make sure that your code runs on these machines correctly.

3. General Advice

If you are not using the CS departmental machines with Linux Operating System and would like to code using your laptop then:

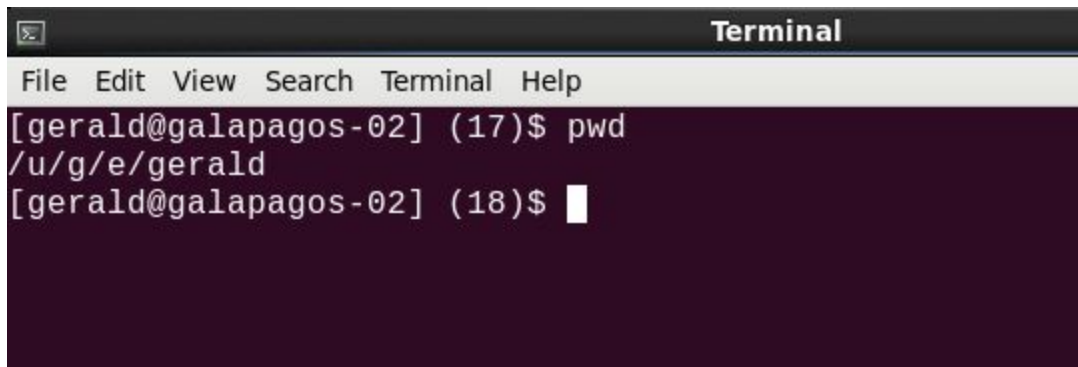
1. Please DON'T use an Integrated Development Environment (IDE) for learning to code in C. It'll hide many of the low-level details from you. You can become a good C programmer only if you understand all these low-level details.
2. Avoid using Windows Operating System for writing C code since the compiler that we'll use for grading your assignments is gcc (GNU C Compiler) which is a Unix/Linux based C compiler.
3. If you still want to use a computer which already has a Windows OS, then you should have a virtual machine (VMware or Virtualbox) to run a Linux OS or Cygwin (which contains tools like gcc) installed on your Windows machine.

Please note that it is your responsibility to make sure that your code runs on CS departmental machines.

4. Unix/Linux commands

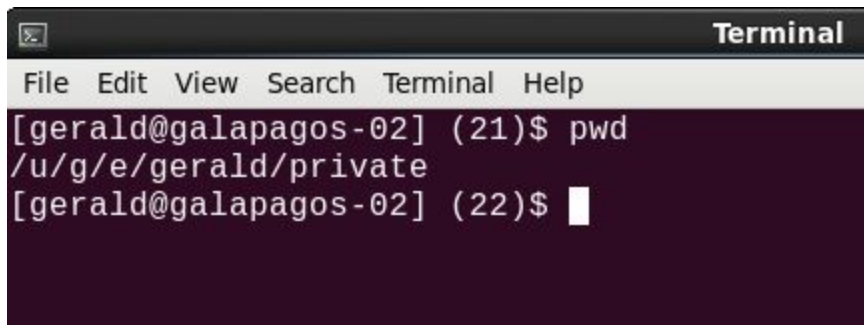
The following set of linux commands will help you to create some directories(folders) on your CS Linux machines. You'll be creating the files needed for this assignment inside these directories.

1. Open the terminal (Applications -> System Tools -> Terminal).
2. Check if you are in your home directory(a directory with your CS username inside which all your files will be stored). The command used for finding out where you are in the directory tree is: **pwd** (print working directory). Type this command at the command prompt (the line in the terminal which ends with a \$ sign) and press enter.



```
Terminal
File Edit View Search Terminal Help
[gerald@galapagos-02] (17)$ pwd
/u/g/e/gerald
[gerald@galapagos-02] (18)$
```

3. List all the files and directories under your home directory. Try to find out the linux command to perform this operation. See if you can find a directory named **private** among the files and directories listed. Notice there is a directory named 'Public' and another named 'public'. Unlike in Windows, filenames are case-sensitive. One useful reference: http://www.comptechdoc.org/os/linux/usersguide/linux_ugbasics.html
4. Change your current directory to **private**. Find the appropriate linux command to change between directories.
5. Check if you are inside the private directory using pwd as shown below:



```
Terminal
File Edit View Search Terminal Help
[gerald@galapagos-02] (21)$ pwd
/u/g/e/gerald/private
[gerald@galapagos-02] (22)$
```

6. Create a new directory named **354** inside your **private** directory.
7. Change your current directory to **354**.
8. Create a new directory named **p0** (for project 0) inside your **354** directory.

9. Change your current directory to **p0**.
10. Copy the file **hello.c** from the location `/p/course/cs354-common/public/spring16/code/hello.c` to your **p0** directory. (Find out the linux command used for copying files).
11. Open the file **hello.c** using any of the text editors of your choice as shown below.

5. Text editor

Some popular text editors used in the Unix/Linux OS environment are the following:

1. vim
2. emacs
3. gedit

If you are transitioning from Windows to Linux probably gedit is the easiest test editor to use but remember learning to use vim or emacs will definitely help you in the longer run to become an awesome programmer!

Use the text editor of your choice to open the file **hello.c**.

6. Build Process

The purpose of this part of the assignment is for you to understand the different steps involved in building an executable file from a C program. You are provided with a very famous C program written by Brian W. Kernighan and Dennis M. Ritchie in their well-known book “The C Programming Language”. Their book is popularly known as K&R C. Your task is to use the gcc C compiler to generate the following files as explained below.

6.1. Preprocessing Phase

Build the intermediate file after preprocessing and store it in a file named **hello.i**. Command used to build the intermediate file after preprocessing is:

```
gcc -E -o hello.i hello.c -m32 -Wall
```

1. Try to understand the meaning of the gcc’s command line option **-E**. You can use the manual (man) page for gcc to understand this. The command to see the man page for gcc is: **man gcc**. Manual pages is a useful way to understand how linux command work. You can find the man pages for all the command on the internet too.
2. The option **-m32** is used to generate code for a 32-bit environment since we’ll be using this environment to study Assembly Language.

3. The **-Wall** gcc option is suggested so that you are aware of the warnings in your programs.
4. Open the file **hello.i** in your text editor and see how much extra code has been added. Try to find out the declaration of the function **printf** that we have used in our C program. The declaration should look something similar as shown below:

```
extern int printf (__const char *__restrict __format, ...);
```

5. Don't worry about what this line exactly means. Just try to understand that the reason for including the file **stdio.h** is mainly to let the compiler know that the function **printf()** that we are using in our program **hello.c** is actually declared in the file **stdio.h**. The definition of **printf** can be seen in the GNU C Library (glibc). Try to understand the differences between a declaration and a definition in C programming.

6.2. Compilation Phase

Now, we are going to stop our compilation after the compiler generates the assembly file. There is an option to let gcc know that it should stop the build process after the compilation phase (named as "compilation proper" in the man page). The output of the compilation phase is a file named **hello.s**.

The following steps will help you generate the assembly code:

1. Run the following command at the command prompt:

```
gcc <option> hello.c -m32 -Wall (OR)  
gcc <option> hello.i -m32 -Wall
```
2. Your task is to find the correct <option> to stop the build process after compilation.
HINT: gcc's man page! Also note that we can give the file **hello.c** or the file **hello.i** as an input to gcc to generate the file **hello.s**.
3. Open the generated **hello.s** file in a text editor and see if you can find something similar as shown below. Note that only a part of this file is shown here. Your **hello.s** file may not look exactly similar as the sample output shown below and you shouldn't worry about it. Just check if you can see the string "hello, world" (not shown below) and the label

“main:” (as shown below).

```
 8 main:
 9 .LFB0:
10  .cfi_startproc
11  leal  4(%esp), %ecx
12  .cfi_def_cfa 1, 0
13  andl  $-16, %esp
14  pushl -4(%ecx)
15  pushl %ebp
16  .cfi_escape 0x10,0x5,0x2,0x75,0
17  movl  %esp, %ebp
18  pushl %ecx
19  .cfi_escape 0xf,0x3,0x75,0x7c,0x6
20  subl  $4, %esp
21  subl  $12, %esp
22  pushl $.LC0
23  call  puts
```

4. As of now, don't worry about understanding the contents of this file. Just get a feel of how Assembly Language Code looks like. By the end of this semester, you'll be able to understand what these lines mean. Wow! Isn't that cool? :-)

6.3. Assembling Phase

You are going to stop the build process after the assembling phase and before the linking phase. The output generated will be the object file for the C source file hello.o. The steps you need to do for this part of the assignment are shown below:

1. Execute the following command to create the object file **hello.o**:
gcc -c hello.c -m32 -Wall (OR) **gcc -c hello.s -m32 -Wall**
Again note that the input to gcc can either be the C source file (hello.c) or the Assembly Code file (hello.s) that was generated from the previous step.
2. Try opening the binary file (hello.o) in your text editor and see what happens.
3. View the contents of the object file (hello.o) using a tool named **objdump** (object dump) as shown below. objdump is a disassembler which converts the machine code (in binary) to assembly code (human readable mnemonic form). A disassembler does the inverse

operation of an assembler which converts the the assembly code into machine code.

```
[gerald@francisco] (205)$ objdump -d hello.o

hello.o:      file format elf32-i386

Disassembly of section .text:

00000000 <main>:
 0:  8d 4c 24 04      lea    0x4(%esp),%ecx
 4:  83 e4 f0        and    $0xffffffff0,%esp
 7:  ff 71 fc        pushl  -0x4(%ecx)
 a:  55             push  %ebp
 b:  89 e5          mov    %esp,%ebp
 d:  51             push  %ecx
 e:  83 ec 04        sub    $0x4,%esp
11:  83 ec 0c        sub    $0xc,%esp
14:  68 00 00 00 00  push  $0x0
19:  e8 fc ff ff ff  call   1a <main+0x1a>
1e:  83 c4 10        add    $0x10,%esp
21:  8b 4d fc        mov    -0x4(%ebp),%ecx
24:  c9             leave
25:  8d 61 fc        lea   -0x4(%ecx),%esp
28:  c3             ret

[gerald@francisco] (206)$ █
```

4. Understand the use of the command `objdump` and the meaning of the option “-d” by looking at the man page for **objdump** or typing “`objdump --help`” at the terminal.
5. Save the disassembled output (shown in the image above) of the object file `hello.o` in a file named **objfile_contents.txt**. One easy way to do this is to redirect the output of the command “`objdump -d hello.o`” to a file named `objfile_contents.txt` as follows:
objdump -d hello.o > objfile_contents.txt (The symbol ‘>’ writes the output of the `objdump` command to the specified text file)
One other way to do this is to simply copy the contents of the output from the terminal and paste it in your text file.

6.4. Linking Phase

This is the final phase of the build process where your object file will be linked with some files in the standard C library to create the final executable file. Follow the steps below to create the final executable file.

1. Execute the following command to complete the build process:

```
gcc -o hello hello.c -m32 -Wall (OR)
gcc -o hello hello.o -m32 -Wall
```

As you might have already guessed, you can provide the original C file or the object file (that we generated in the previous phase) as an input to gcc to generate the final executable file.

2. Execute the generated executable file (hello) using the following command in the terminal. You should be able to see the string “hello, world” printed on your screen.
./hello
3. Use objdump to view the disassembled contents of the executable file (which is also a binary file) as we did for the object file hello.o.
4. Redirect the disassembled output that you got from step 3 to a file named **exefile_contents.txt**. This file should be much larger than the disassembled output of the hello.o file since **hello** is an executable file which has information combined from hello.o and printf (binary file having the definition of the function printf).

Note: Even though we have seen each and every intermediate files that are created while we try to compile a C program, the two files that you’ll most often use in your real life are the following:

1. C Source File (hello.c)
2. Executable File (hello)

For the purposes of this course and whenever you are debugging at the assembly level, you’ll also use the assembly file (hello.s) often.

Although we have multiple variations of using gcc, the most commonly used command in real life to compile the C program and to build the executable file are the following:

1. **gcc -o hello hello.c -Wall** (in real life for a 64-bit machine or a 32-bit machine)
2. **gcc -o hello hello.c -m32 -Wall** (in CS 354 since we’ll be studying mainly 32-bit machines)

You’ll be using these commands a lot over the course of this semester and even after that whenever you are working with some C code in the future. Welcome to the World of C Programming! :-)

7. Modifying the C program

Now let’s get your hands dirty by writing some very simple C code on your own. You’ll be modifying the same C program **hello.c** to make it more sophisticated. Follow the instructions below to make the necessary modifications.

1. Declare an integer variable named **myage** and initialize it to your age.
2. Declare a C style string (an array of characters) named **myname** and initialize the value of the string to be your name.
3. Declare a char variable named **myalpha** and initialize it to your favourite English alphabet (A - Z).
4. Print your name, your age, and your favourite alphabet using the variables **myage**, **myname**, and **myalpha** each on a separate line as shown below. Use the correct format specifiers for the three variables (e.g. %d for integer variable). Find out the format specifiers for char and string in C and use them in your code.

```
[gerald@francisco] (6)$ ./hello
hello, world
My name is Gerald
My age is 27
My favourite alphabet is A
[gerald@francisco] (7)$ █
```

8. Submitting the assignment

The files you need to submit are the following:

1. hello.c (modified file which prints your name, age and favourite alphabet).
2. hello.i (the intermediate file after preprocessing)
3. hello.s (the assembly file after compilation proper)
4. hello.o (the object file after assembling)
5. hello (the executable file after linking with standard libraries)
6. objfile_contents.txt (disassembled output of hello.o object file)
7. exefile_contents.txt (disassembled output of hello executable file)

Copy all your 7 files to your handin directory using the command prompt in the terminal as shown below:

```
cp hello.c /p/course/cs354-common/public/spring16.handin/<your_CS_login_id>/p0/
```

```
cp hello.i /p/course/cs354-common/public/spring16.handin/<your_CS_login_id>/p0/
```

Similarly copy the other files to the same directory as shown above.