

Lab 2: Unix, GDB, Valgrind

cat lab_readme.txt|

Welcome!

Welcome to Lab 2 (which will henceforth be referred to as hw0). Before we begin this lab, please make sure you have read through the [Lab 0: General Lab Guide](#). This is your guide to navigating all labs and making sure you get the most out of every lab (and all the credit you deserve as well!).

The purpose of this assignment is to acquaint you with some of the tools in the programming lab, and with the standard procedures and guidelines for working on the homework assignments. This assignment walks you through using the GDB debugger, valgrind memory management tool, and some more advanced UNIX tools.

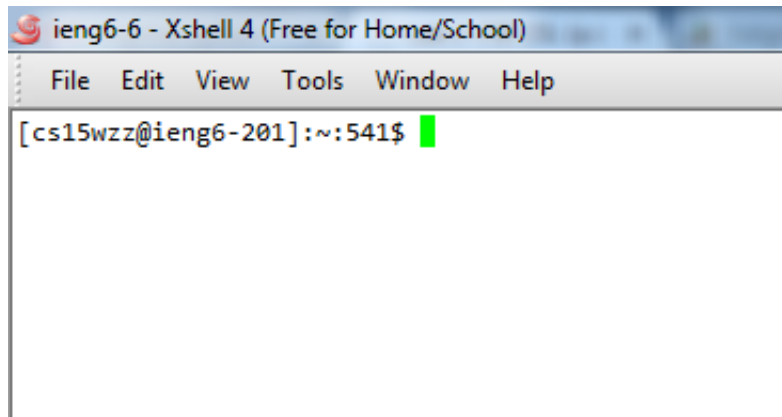
For this lab, please record the answers to all questions in a README file (see the "Getting Started" section), which will be used for the checkoff at the end of this lab.

Goals of Completing this Lab

- Learn how to use the GDB debugger for help debugging your C and C++ code.
- Learn how to use Valgrind to detect memory leaks in your code.
- More practice on Unix commands that will help you to better

understand your system, manipulate/edit/compress files, and view/stop running processes.

Getting Started



Log in to a lab computer, under Linux using your cs15fa19x username. If you do not know your cs15fa19xx login, it can be found with the ACS account lookup tool. Your password is the same as your personal ACS account password.

Open a new terminal by right-clicking on the desktop and then choosing Open Terminal.

Acquire The Lab Files

Acquire the assignment from the public directory. To avoid clobbering your work when repeating installation steps, assignments contain one or more .empty files that need to be renamed to remove the .empty file extension. Those will contain the code that you will write for an assignment. There will be some output displayed to the terminal while you run these commands, so please don't be alarmed by it.

```
$ cd ../../public/hw0
$ make directories
$ make install
$ cd ~/hw0/c
$ mv hw0.c.empty hw0.c
$ cd ../cpp
$ mv hw0.cpp.empty hw0.cpp
```

Prepare the README file

Back in the terminal window, rename the README.empty file, and open it for editing. You will use this file to record your answers to the lab questions.

```
$ cd ~/hw0
$ mv README.empty README
$ gvim README
```

If you're new to vim/gvim, it may be helpful to use a search engine, and search for "vim commands". Here are a few.

- :w - Save
- :q - Quit
- i - Enter Insert Mode
- - Enter Command Mode

Build all Files

```
$ cd ~/hw0
$ make
```

What is 'make'?

Make is a build automation tool that automatically builds executable programs and libraries from source code by reading files called Makefiles which specify how to derive the target program.

If the make command failed, you may have made a mistake when renaming your files. Please review your directories to ensure the expected files are present. If you get a compiler warning "cast from pointer to integer", you can safely ignore it. We will have a lab on it soon!



```
[cs15x] cs15x19@its-cseb250-10.ucsd.edu:/home/linux/ieng6/cs15x/cs15x19/hw0
File Edit View Search Terminal Help
[cs15x19@its-cseb250-10]:hw0:45$ make
make -C c
make[1]: Entering directory `/home/linux/ieng6/cs15x/cs15x19/hw0/c'
gcc -Wall -pedantic -g -O0 -c hw0.c
hw0.c: In function 'main':
hw0.c:167:17: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
    math = argc + (int)argv;
                   ^
gcc -Wall -pedantic -g -O0 -o a.out hw0.c
make[1]: Leaving directory `/home/linux/ieng6/cs15x/cs15x19/hw0/c'
make -C cpp
make[1]: Entering directory `/home/linux/ieng6/cs15x/cs15x19/hw0/cpp'
g++ -Wall -pedantic -g -c hw0.cpp
g++ -Wall -pedantic -g -o a.out hw0.o
make[1]: Leaving directory `/home/linux/ieng6/cs15x/cs15x19/hw0/cpp'
make -C java
make[1]: Entering directory `/home/linux/ieng6/cs15x/cs15x19/hw0/java'
javac -g hw0.java
echo 'java hw0 $*' > a.out
chmod u+x a.out
make[1]: Leaving directory `/home/linux/ieng6/cs15x/cs15x19/hw0/java'
[cs15x19@its-cseb250-10]:hw0:46$
```

Goal: Debug C Code by analyzing the code during program execution.

This exercise will teach you how to use the gdb debugger. It will cover

break, next, step, print, continue where finish quit

the break, next, step, print, continue, where, finish, and quit commands.

Change directories to the c subfolder of the hw0 assignment and build.

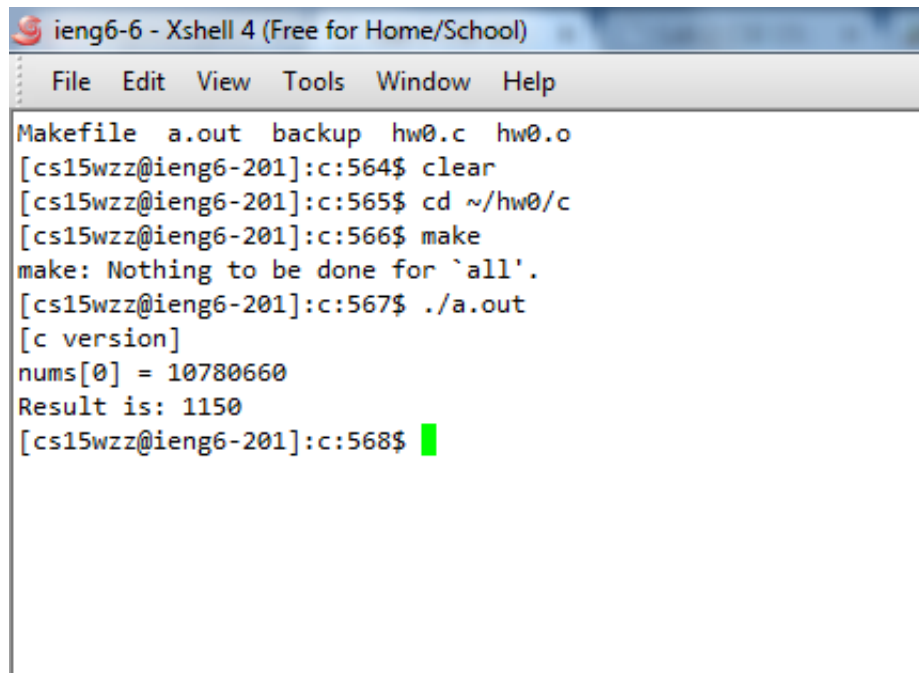
```
$ cd ~/hw0/c
```

```
$ make
```

If you followed the last step on the previous page, make should say "Nothing to be done for 'all'."

An executable file named `a.out` will be created. Run the program by typing:

```
$ ./a.out
```



```
ieng6-6 - Xshell 4 (Free for Home/School)
File Edit View Tools Window Help
Makefile a.out backup hw0.c hw0.o
[cs15wzz@ieng6-201]:c:564$ clear
[cs15wzz@ieng6-201]:c:565$ cd ~/hw0/c
[cs15wzz@ieng6-201]:c:566$ make
make: Nothing to be done for `all'.
[cs15wzz@ieng6-201]:c:567$ ./a.out
[c version]
nums[0] = 10780660
Result is: 1150
[cs15wzz@ieng6-201]:c:568$
```

NOTE: For some of you, you will get `nums[0] = 0` instead of a random number. This is fine.

Open hw0.c in gvim, or your favorite text editor.

lol

```
$ gvim hw0.c
```

```
ieng6-6 - Xshell 4 (Free for Home/School)
File Edit View Tools Window Help
*****
Name(s):
cs15w account(s):
CSE 15L, Winter 2014

Lab 2 (hw0)

File Name: hw0.c
Description: This program's purpose is to be an ideal specimen for
debugging. It does basic string manipulation, integer math,
dynamic memory allocation, and simulated object-oriented
programming with the DataHolder "class." This program also
contains some common memory errors that can be easily
detected with valgrind.
*****/
#include <string.h>
#include <malloc.h>
#include <stdio.h>
#include <ctype.h>

/* struct DataHolder
 *
 * Description: A simple "class" for demonstrating object-oriented
 * programming in C.
 *
 * Data members:
 * data - a dynamically allocated string
 * id - a unique id for this DataHolder
 *
 * Public functions:
 * new_DataHolder
 * mutate_DataHolder
 * DataHolder_delete
 */
typedef struct {
    char *data;
    int id;
} DataHolder;
/*-----
typedef struct {
char *data
int id;
} Data Holder;
-----*/
```

Start a.out in the gdb debugger.

```
$ gdb a.out
```

NOTE: If you are getting an "'import site' failed" error, that's okay. This should not affect your program, so just continue)

All command line C/C++ programs start execution in `main()`. We can set breakpoints using the `break` command to stop execution at certain places in our program. Type in the following command to learn more about `break`:

```
(gdb) help break
```

Question 1: What gdb command would you run to set a breakpoint at line 42?

Set a breakpoint at `main()`, then run your program.

```
(gdb) break main
```

```
(gdb) run
```



```
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/linux/ieng6/cs15w/cs15wzz/hw0/c/a.out...done.
(gdb) break main
Breakpoint 1 at 0x80486b2: file hw0.c, line 144.
(gdb) run
Starting program: /home/linux/ieng6/cs15w/cs15wzz/hw0/c/a.out

Breakpoint 1, main (argc=1, argv=0xbfffed84) at hw0.c:144
144      nums[1] = 5;
(gdb) █
```

(NOTE: If you get a "Missing separate debuginfos" error, you can ignore this as well)

Execution will stop at the breakpoint, and the next line to be executed will be displayed. To see more lines nearby, type

```
(gdb) list
```

```

GNU gdb (GDB) 7.6.1
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/linux/ieng6/cs15w/cs15wzz/hw0/c/a.out...done.
(gdb) break main
Breakpoint 1 at 0x80486b2: file hw0.c, line 144.
(gdb) run
Starting program: /home/linux/ieng6/cs15w/cs15wzz/hw0/c/a.out

Breakpoint 1, main (argc=1, argv=0xbfffed84) at hw0.c:144
144      nums[1] = 5;
(gdb) list
139
140      /* declare a DataHolder on the stack */
141      DataHolder myHolder;
142
143      /* populate the nums array */
144      nums[1] = 5;
145      nums[2] = 6;
146      nums[3] = 10;
147
148      printf("[c version]\n");
(gdb) print/d argc
$1 = 1
(gdb) 

```

Notice the variables `argc` and `argv`. `argc` is the number of command line arguments passed to the program, and `argv` is an array of strings containing those command line arguments.

Question 2: How many command line arguments were passed to `a.out`?

Print out `argc` as a signed decimal integer:

```
(gdb) print/d argc
```

Funny, I don't remember passing any arguments to a.out, do you? Let's examine the value of the first argument as a string:

```
(gdb) x/s argv[0]
```

x/s arg'

Question 3: What does the first argument represent?

Execute the next few lines of code in your program:

```
(gdb) next  
(gdb) print/d nums[2]
```

Shortcut: You can use 'p' in place of 'print' to accomplish the same task. In our previous example instead of 'print/d nums[2]', you can simply input 'p/d nums[2]'.

Question 4: What is the value of nums[2]? Why?

Execute the next line of code.

```
(gdb) next
```

Question 5: Now what is the value of nums[2]? Which line of code just got executed?

Set a breakpoint on the first call to doMath() and continue execution. The break command, b for short, also takes line numbers.

```
(gdb) break 153
```

```
(gdb) continue
```

Execute the next line of code, stepping into function calls.

```
(gdb) step
```

Find out which function is currently executing:

```
(gdb) where
```

Question 6: What function is currently executing?

Finish executing that function.

```
(gdb) finish
```

Question 7: What value was returned by that function?

```
(gdb) continue
```

```
(gdb) quit
```

Goal: Detect memory leaks in your code.

There are some not-so-obvious problems with this program. Valgrind is a memory mismanagement detector that can help track down memory issues such as memory leaks, access violations and

uninitialized memory reads.

Run your program using valgrind build of your program.

```
$ make valgrind
```

```

[cs15wzz@ieng6-201]:c:531$ make valgrind
valgrind --read-var-info=yes --leak-check=yes --show-reachable=yes ./a.out
==2980== Memcheck, a memory error detector
==2980== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==2980== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==2980== Command: ./a.out
==2980==
[c version]
==2980== Use of uninitialised value of size 4
==2980==    at 0x8F4E36: _itoa_word (_itoa.c:195)
==2980==    by 0x8F8A3E: vfprintf (vfprintf.c:1640)
==2980==    by 0x9004AF: printf (printf.c:35)
==2980==    by 0x80486E9: main (hw0.c:152)
==2980==
==2980== Conditional jump or move depends on uninitialised value(s)
==2980==    at 0x8F4E3E: _itoa_word (_itoa.c:195)
==2980==    by 0x8F8A3E: vfprintf (vfprintf.c:1640)
==2980==    by 0x9004AF: printf (printf.c:35)
==2980==    by 0x80486E9: main (hw0.c:152)
==2980==
==2980== Conditional jump or move depends on uninitialised value(s)
==2980==    at 0x8F69E4: vfprintf (vfprintf.c:1640)
==2980==    by 0x9004AF: printf (printf.c:35)
==2980==    by 0x80486E9: main (hw0.c:152)
==2980==
==2980== Conditional jump or move depends on uninitialised value(s)
==2980==    at 0x8F6A08: vfprintf (vfprintf.c:1640)
==2980==    by 0x9004AF: printf (printf.c:35)
==2980==    by 0x80486E9: main (hw0.c:152)
==2980==
nums[0] = 10780660
Result is: 1150
==2980==
==2980== HEAP SUMMARY:
==2980==    in use at exit: 14 bytes in 1 blocks
==2980==    total heap usage: 1 allocs, 0 frees, 14 bytes allocated
==2980==
==2980== 14 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2980==    at 0x4007282: malloc (vg_replace_malloc.c:270)
==2980==    by 0x80485C1: new_DataHolder (hw0.c:68)
==2980==    by 0x8048771: main (hw0.c:161)
==2980==
==2980== LEAK SUMMARY:
==2980==    definitely lost: 14 bytes in 1 blocks
==2980==    indirectly lost: 0 bytes in 0 blocks
==2980==    possibly lost: 0 bytes in 0 blocks
==2980==    still reachable: 0 bytes in 0 blocks

```

Valgrind should have reported four (or possibly a few more) problems of two types: a use of an uninitialized value (also known as a UMR, or

Uninitialized Memory Read) and a memory leak. Answer the following questions by referring to the output of valgrind.

**Question 8: What line of code caused the UMR?
Accessing which value appears to cause the UMR?**

**Question 9: How many bytes were leaked? From
which function in your program was this memory
allocated?**

Exercise 1:

Eliminate the valgrind warnings by adding code to hw0.c. You need to add 2 lines of code;

Look for `/* YOUR CODE GOES HERE */` in `main()`. (Hint: look around hw0.c for a certain helpful method).

After your edits, you should get output similar to the following. Ignore the ld-2.17.so conditional jump warnings:

```
[cs15x] cs15x19@its-cse250-10.ucsd.edu:/home/linux/ieng6/cs15x/cs15x19/hw0/c
File Edit View Search Terminal Help
make[1]: Entering directory `/home/linux/ieng6/cs15x/cs15x19/hw0/c'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/linux/ieng6/cs15x/cs15x19/hw0/c'
valgrind --read-var-info=yes --leak-check=yes --show-reachable=yes ./a.out
==14280== Memcheck, a memory error detector
==14280== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==14280== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==14280== Command: ./a.out
==14280==
==14280== Conditional jump or move depends on uninitialised value(s)
==14280==    at 0x40190AE: index (in /usr/lib64/ld-2.17.so)
==14280==    by 0x4007A03: expand_dynamic_string_token (in /usr/lib64/ld-2.17.so)
==14280==    by 0x400852E: _dl_map_object (in /usr/lib64/ld-2.17.so)
==14280==    by 0x400135D: map_doit (in /usr/lib64/ld-2.17.so)
==14280==    by 0x400F363: _dl_catch_error (in /usr/lib64/ld-2.17.so)
==14280==    by 0x4001DE9: handle_ld_preload (in /usr/lib64/ld-2.17.so)
==14280==    by 0x40038A8: dl_main (in /usr/lib64/ld-2.17.so)
==14280==    by 0x40171A5: _dl_sysdep_start (in /usr/lib64/ld-2.17.so)
==14280==    by 0x4001BC0: _dl_start (in /usr/lib64/ld-2.17.so)
==14280==    by 0x4001177: ??? (in /usr/lib64/ld-2.17.so)
==14280==
==14280== Conditional jump or move depends on uninitialised value(s)
==14280==    at 0x40190B9: index (in /usr/lib64/ld-2.17.so)
==14280==    by 0x4007A03: expand_dynamic_string_token (in /usr/lib64/ld-2.17.so)
==14280==    by 0x400852E: _dl_map_object (in /usr/lib64/ld-2.17.so)
==14280==    by 0x400135D: map_doit (in /usr/lib64/ld-2.17.so)
==14280==    by 0x400F363: _dl_catch_error (in /usr/lib64/ld-2.17.so)
==14280==    by 0x4001DE9: handle_ld_preload (in /usr/lib64/ld-2.17.so)
==14280==    by 0x40038A8: dl_main (in /usr/lib64/ld-2.17.so)
==14280==    by 0x40171A5: _dl_sysdep_start (in /usr/lib64/ld-2.17.so)
==14280==    by 0x4001BC0: _dl_start (in /usr/lib64/ld-2.17.so)
==14280==    by 0x4001177: ??? (in /usr/lib64/ld-2.17.so)
==14280==
==14280== Conditional jump or move depends on uninitialised value(s)
==14280==    at 0x40190C4: index (in /usr/lib64/ld-2.17.so)
==14280==    by 0x4007A03: expand_dynamic_string_token (in /usr/lib64/ld-2.17.so)
==14280==    by 0x400852E: _dl_map_object (in /usr/lib64/ld-2.17.so)
==14280==    by 0x400135D: map_doit (in /usr/lib64/ld-2.17.so)
==14280==    by 0x400F363: _dl_catch_error (in /usr/lib64/ld-2.17.so)
==14280==    by 0x4001DE9: handle_ld_preload (in /usr/lib64/ld-2.17.so)
==14280==    by 0x40038A8: dl_main (in /usr/lib64/ld-2.17.so)
==14280==    by 0x40171A5: _dl_sysdep_start (in /usr/lib64/ld-2.17.so)
==14280==    by 0x4001BC0: _dl_start (in /usr/lib64/ld-2.17.so)
==14280==    by 0x4001177: ??? (in /usr/lib64/ld-2.17.so)
==14280==
[c version]
nums[0] = 69
Result is: 1150
==14280==
==14280== HEAP SUMMARY:
==14280==    in use at exit: 0 bytes in 0 blocks
==14280==    total heap usage: 1 allocs, 1 frees, 14 bytes allocated
==14280==
==14280== All heap blocks were freed -- no leaks are possible
==14280==
==14280== For counts of detected and suppressed errors, rerun with: -v
==14280== Use --track-origins=yes to see where uninitialised values come from
==14280== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
[cs15x19@its-cse250-10]:c:120$
```

** What did you add, and where? (Put this in your README as well.) **

Question 10: How do you know that your code has no

more memory leaks?

Goal: Debug C++ code by analyzing the code during program execution.

Change directories to the cpp subdirectory of hw0, compile, and run.

```
$ cd ~/hw0/cpp  
$ make  
$ ./a.out
```

Open hw0.cpp for editing:

```
$ gvim hw0.cpp
```

Start a.out in the gdb debugger:

```
$ gdb a.out
```

Set a breakpoint on the declaration of myHolder() (line 117) and run.

```
(gdb) break 117  
(gdb) run
```

Step once.

```
(gdb) step
```

Question 11: What function are you in? What are the names and values of the parameters to this function? (Hint: You might want to use print/s)

Finish executing the current function.

```
(gdb) finish
```

You should now be back in main.

Question 12: What is the address of myHolder in hexadecimal? To print out a hexadecimal value, use print/x instead of print/d. To get an address, use &myHolder instead of myHolder.

Continue execution and let the program exit normally, then quit gdb.

```
(gdb) continue
```

```
(gdb) quit
```

Exercise 2:

Given that you can print a string in C++ with the following:

```
cout << "print me!";
```

** Add a line of code that will print myHolder after the call to myHolder.mutate(). **

Look for `/* YOUR CODE GOES HERE */`

C++ can be a very difficult language to follow because there are so many implicit function calls and conversions. While it may not be obvious, the line of code you added actually called a function in `hw0.cpp`. You will now use `gdb` to figure out which function was called after you've **recompiled** with

```
$ make
```

** What code did you add, and where? (Add this to your README) **

Restart the `gdb` debugger.

```
$ gdb a.out
```

Set a breakpoint on the line of code that you added in exercise 2 and run. If you don't remember how to do this, review **Goal: Debug C Code by analyzing the code during program execution**. Begin execution with `run`. The program should stop at the breakpoint. Use `step` to execute the very next line of code.

```
(gdb) run
```

```
(gdb) step
```

See which function is currently executing by typing

```
(gdb) where
```

Question 13: Which function is currently executing?

Exit GDB by typing "quit" or "q" at the prompt.

Goal: Know where you are in your system.

After logging in, open a terminal window. Type the following command:

```
$ history
```

Question 14: What gets printed out?

In your home directory, create a directory named Lab2, and `cd` to it. You will work within that directory when doing this lab:

```
$ mkdir Lab2
$ cd Lab2
```

Copy the Lab2 directory contents to your account, then change current directory to Lab2/sectionA:

```
$ cp -r ~/../public/Lab2/* .
$ cd ~/Lab2/sectionA
```

Question 15: What command would you use to verify that you are in the subdirectory sectionA?

Goal: Grab information from files without

needing to manually sort through them.

The `diff` command

The `diff` command displays differences between two files on a line-by-line basis. It displays the differences as instructions that you can use to edit one of the files (using the `vi` editor) to make it the same as the other. When you use `diff`, it produces a series of lines containing Append `a`, Delete `d`, and Change `c` instructions. Each of these lines is followed by the lines from the file that you need to append, delete, or change. A less than symbol `<` precedes lines from file1. A greater than symbol `>` precedes lines from file2.

You will now need four files: `telnos`, `telnos2`, `telnos3`, `telnos4`. These files are all short files that contain names, departments, and telephone numbers. This is what they look like.

telnos	telnos2
Hale Elizabeth Bot 744-6892	Hale Elizabeth Bot 744-6892
Harris Thomas Stat 744-7623	Harris Thomas Stat 744-7623
Davis Paulette Phys 744-9579	Davis Paulette Phys 744-9579
Cross Joseph MS 744-0320	Holland Tod A&S 744-8368
Holland Tod A&S 744-8368	
telnos3	telnos 4
Hale Elizabeth Bot 744-6892	Hale Elizabeth Bot 744-6892
Harris Thomas Stat 744-7623	Smith John Comsc 744-4444
Smith John Comsc 744-4444	Davis Paulette Phys 744-9579
Davis Paulette Phys 744-9579	Cross Joseph MS 744-0320

Cross Joseph MS 744-0320	Holland Tod A&S 744-8368
Holland Tod A&S 744-8368	

Verify that you have these four files in your sectionA directory.

In order to see how `diff` works, type in:

```
$ diff telnos telnos2
```

What was the result?

The difference between these two files (telnos and telnos2) is that the 4th line in telnos is missing from telnos2. The first line that `diff` displays `4d3` indicates that you need to delete the 4th line from file telnos to make the two files match. The 4 is the line number and the (d) is delete. The line number to the left of each of the a, c, or d instructions always pertains to file1. Numbers to the right of the instructions apply to file2. The `diff` command assumes that you are going to change file1 to file2. The next line that `diff` displays starts with a less than < symbol indicating that this line of text is from file1.

Next type in:

```
$ diff telnos telnos3
```

What was the result?

In this case, the second line has the > greater than sign which means that the extra line is in file2. The a means you must append a line to the file telnos after line 2 to make it match telnos3. Append means to

add on to the end.

Next is an example of the change feature. Type in the following command:

```
$ diff telnos telnos4
```

What was the result?

There is another command called `vimdiff` (or `gvimdiff`) that allows you to easily visualize the differences between files. Type the following command:

```
$ vimdiff telnos telnos4
```

Alternatively, type

```
$ gvimdiff telnos telnos4
```

Question 16: What lines do you need to change in order to make the `telnos` and `telnos4` files alike?

Notice that the three hyphens indicate the end of the text in the first file that needs to be changed and the start of the second file that needs to be changed. **Next, copy `telnos` to `telnos5`.**

Question 17: What command did you use to do this?

Next, type in:

```
$ diff telnos5 telnos2
```

Question 18: What is printed?

Use the vi editor to change telnos5 to match the file telnos2. Then check to see if they are now alike.

Question 19: How would you know when two files contain exactly the same contents?

The uniq command.

The `uniq` command displays a file, removing all but one copy of successive repeated lines. If the file has been sorted, `uniq` ensures that no two lines that it displays are the same. Sort telnos and telnos3 and then send them to a new file called tel2.

Type in the following:

```
$ sort telnos telnos3 > tel2
```

Question 20: Look at the file, tel2. What does it contain?

Next issue the command:

```
$ uniq tel2
```


Question 21: What was the result?

The `uniq` command has three options. These are:

`-c` Causes to precede each line with the number of consecutive occurrences of the line in the input file

`-d` only lines that are repeated

`-u` only lines that are not repeated

Issue the command:

```
$ uniq -u tel2
```

Next, issue the command:

```
$ uniq -d tel2
```

Question 22: Given a dictionary file containing words seperated by newlines, what command would you use to check to make sure there are no duplicate words?

The grep command.

The `grep` command searches one or more files for a specified pattern. Normally each matching line is copied to the standard output. Two options that can be used with `grep` are:

- `-i` ignore case of alphabetic characters

- `-n` precede each line printed by its relative line number in the input file

Use the line numbers in the output of `grep` to answer the questions in the following sections.

Issue the command:

```
$ grep -n H telnos
```

Note what is printed.

Issue the command:

```
$ grep -ni m telnos
```

Question 23: What changed between the two previous commands?

These are the files that should be in your sectionA folder.

- books
- dept
- employee
- name
- salary
- telnos
- telnos2
- telnos3
- telnos4

- telnos5
- tel2

Question 24: What is the output of the following commands?

Option `-v` will display all the lines except the match. In the example below, it displays all the records from `telnos3` that doesn't match Joseph.

```
$ grep -v Joseph ./telnos3
```

What is the output?

How many lines matched the text pattern in a particular file?

In the example below, it displays the total number of lines that contains the text Joseph in `telnos3` file.

```
$ grep -c Joseph ./telnos3
```

What is the output?

You can also get the total number of lines that did not match the specific pattern by passing option `-cv`.

```
$ grep -cv Joseph ./telnos3
```

What is the output?

How to search a text by ignoring the case?

Pass the option `-i` (ignore case), which will ignore the case while searching.

```
$ grep -i joseph ./telnetos3
```

What is the output?

Goal: Compress and combine files together.

The `tar` command.

Ever wondered how to create a single backup file of all files and subdirectories under the home directory?

The following command creates a single archive backup file called `mycs15Lxxdirectory.tar` under `/tmp`. This archive will contain all the files and subdirectories under `./`

Option `c`, stands for create an archive.

Option `v` stands for verbose mode, displays additional information while executing the command.

Option `f` indicates the archive file name mentioned in the command.

Replace `cs15Lxx` with your login account name to ensure the filename is unique:

```
$ tar -cvf /tmp/my_cs15fa19xx_directory.tar ./
```

How do I view all the files inside the tar archive?

Option t will display all the files from the tar archive.

```
$ tar -tvf /tmp/my_cs15fa19xx_directory.tar
```

How do I extract all the files from a tar archive?

Option x will extract the files from the tar archive as shown below. This will extract the content to the current directory location from where the command is executed.

```
$ tar -xvf /tmp/my_cs15fa19xx_directory.tar
```

Note: Does the 'tar' command remind you of anything? Tar is to Unix as zip is to Windows!

Please remove your file from /tmp folder.

Question 25: What command would you use to to remove your file from /tmp?

Question 26: How would you use tar to compress a directory called `files` located in the root directory?

Goal: Be able to handle processes running in your environment.

Have you ever had a program freeze up on you? Have you accidentally put a process to sleep using `[ctrl]-z`? This next section will help you learn ways to deal with issues like those.

The `ps` command.

The `ps` command will display the current processes that are running. It stands for "process status". To use it, simply type `ps`:

Let's add two more processes by running them in the background. One useful trick with vim is pressing `[ctrl]-z` to pause vim and interact with the console (such as to compile code) and typing `fg` (stands for foreground) in the console to go back to vim.

```
$ vim foo
# add some text and press ctrl-z to put vim into the background
$ vim bar
# add some other text and press ctrl-z
$ ps
```

Now we can see that there are more processes in the background. However, we can't distinguish between the two vim processes, we need more information. The `-f` option shows more info.

```
$ ps -f
```

Question 27: Which columns were added?

Now type `fg` in the console to open bar back up in vim. Save and quit.

Type `fg` again to open up `foo`. Save and quit again. Another useful option in `ps` is `-e`, which displays the processes for all users.

```
$ fg # This brings up bar. Save and quit
$ fg # This brings up the foo. Save and quit
$ ps -ef
```

The `kill` command.

The `kill` command kills unresponsive programs. It's syntax is:

```
$ kill -9 [pid]
```

Exercise 3: Practice killing a unresponsive program.

The `-9` is the kill signal, to make sure we completely terminate it's execution. We need a process to kill. Try out the `yes` command.

```
$ yes
```

It doesn't really do anything but print a bunch of "y"s! Let's put it in the background by pressing `[Ctrl]-Z`. Another way to run processes in the background is by adding a `&` at the end of the command. Try it:

```
$ yes &
```

Great now you have an unresponsive terminal. Don't worry this is

intended. We need to some how stop this process. Don't close the terminal, let's solve this more humanely by killing the process instead.

We need to open up a new terminal, find the process, and kill it.

```
# in a new terminal...
$ ps -ef | grep your_username | grep yes
# look for the pid of the yes process
```

Find the pid of the processes. The ps should return two "yes" processes. The pid is the first number after your username.

```
cs15wxx  30214 29103 21 23:05 pts/0    00:00:14 yes
cs15wxx  31080 29103 21 23:07 pts/0    00:00:16 yes
```

Now you can kill the two processes.

```
$ kill -9 [PID_of_yes_process]
$ kill -9 [PID_of_other_process]
# use ps again to make sure its dead
$ ps -ef | grep your_username
```

If you have problems killing the process, ask a tutor.

Question 28: What command would you use to open up gvim as a background process?

Note: If the command does not work try to ssh into ieng6.

UNIX Fun

Try this: `cat /dev/urandom | tr -dc 'a-z'`

You can press ctrl-c to stop the continuous output.

- In Unix-like operating systems, `/dev/random`, `/dev/urandom` and `/dev/arandom` are special files that serve as pseudorandom number generators.
- `tr` is the translate command. `tr` reads a byte stream from its standard input and writes the result to the standard output. As arguments, it takes two sets of characters (generally of the same length), and replaces occurrences of the characters in the first set with the corresponding elements from the second set.
- `-d` : delete characters in the first set from the output.
- `-c` : complements the set of characters in string. Operations apply to characters not in the given set.

Using `cat /dev/urandom | tr -dc 'a-z'` will filter out all characters other than `a-z`. The resulting output will contain only characters `a-z`.

Example

```
echo hello | tr ho HO
```

- Notice that all occurrences of "h" and "o" were replaced with "H" and "O". output: `He11o`

Feedback

If you have any feedback about this lab, please [leave it here!](#) we will check it and respond.

Checkoff

Before you leave, please make sure that you are checked off and that you receive an email confirming that you have been checked off (remember, this is your receipt!). To get checked off, please raise your hand and show the following to a lab tutor:

1. Lab 2 README with the 28 questions above answered.