Faculty of Science

| | |
|---|---|
| **Course**: | CSCI 3020U: Operating System Concepts |
| **Lab:** | #1 |
| **Topic:** | Introduction to Linux CLI |

## Instructions

This is an introductory lab covering some basic Linux CLI commands which will be helpful for programming in a Unix based environment. Following are some tutorials along with explanations which you should practice and at the end of this handout are some deliverable tasks which you need to complete and submit at the end of the lab.

## Linux Commands:

### HELP

```
$ man [command name]
```

DISPLAYS SHORT MANUAL PAGE DESCRIPTIONS

```
$ whatis read
```

**Purpose:** Each manual page has a short description available within it. whatis searches the manual page names and displays the manual page descriptions of any name matched.

### Date Time

```
$ date
```

**Purpose:** Prints the system date and time.

### Calendar

```
$ cal
```

**Purpose:** Prints an ASCII calendar of the current month.

## Current Working Directory

```
$pwd
```

**Purpose:** Displays the absolute path to the current working directory.

## Writing simple text

```
$echo 'Hello World!'
```

**Purpose:** it prints the statement in single quotes.

```
$echo $(ls)
```

**Purpose:** The $() operator allows you to use other Linux commands to print out as strings.

```
$echo $((2+2+2))
```

**Purpose:** The $(()) operator allows for mathematical operations to be done.

## Changing Directories

```
$ cd
```

**Purpose:** Changes directories.

```
$ cd /home/ucp/cs
```

**Purpose:** To an absolute path.

```
$ cd project /docs
```

**Purpose:** To a relative path.

```
$ cd ..
```

**Purpose:** To a directory one level up.

```
$ cd ~
```

**Purpose:** To your home directory.

```
$ cd –
```

**Purpose:** To your previous working directory

## Listing Directory Contents

```
$ ls [options] [files_or_ directories]
```

**Purpose:** Lists the contents of the current directory or a specified directory.

```
$ ls
```

**Purpose:** Without arguments lists the files and directories names in the current directory.

```
$ ls /
```

**Purpose:** Lists the contents of the directory given as an argument.

```
$ ls –a/home /student
```

**Purpose:** Includes so-called “hidden” files and directories whose names begin with a dot (.):

```
$ ls –all
```

**Type man ls** to see all the available options.

## Copying files and directories:

```
$ cpfile destination
```

**Purpose:** Copy files and directories. More than one file may be copied at a time if the destination is a directory.

### *Note:*

If the destination exists and is a directory, the copy is placed there with the same name.

If the destination exists and is a file, the copy overwrites the destination file.

## Moving files and directories:

```
$ mv file destination
```

**Purpose:** Move files and directories from one place to another. More than one file may be moved at a time if the destination is a directory.

## Removing and creating files

```
$ rm filename
```

**Purpose:** Remove files

```
$ mkdir dirname
```

**Purpose:** Create a new directory with the given name.

```
$ rmdir dirname
```

**Purpose:** Remove an empty directory.

## Creating a file

```
$ touch file1.txt
```

**Purpose:** This creates a new file by the name “file1.txt” in your current directory.

## Viewing an Entire Text File

```
$ cat [option] [file …]
```

**Purpose:** A Unix/Linux command that can read, modify or concatenate text files. The cat command also displays file contents. Contents of the files are displayed sequentially with no break. Also files display “Concatenated”

```
$ cat filename
```

**Purpose:** Display the contents of the file filename.

```
$cat>file1
<enter text>
[Ctrl+d]
```

**Purpose:** The above command creates the file called file1 and you can enter the text there only. After finishing your work, press Ctrl+d (Press Enter after the last line of your character to denote the end of the file).If file1 already exists then it over writes the contents of the file1.

">" is called Redirection Operator.

```
$cat>>file1
<enter text>
[Ctrl+d]
$ cat file1
```

**Purpose:** The above command is used to append more text to already existing file.

```
$cat file1 file2 >file3
```

**Purpose:** The above command is used to write contents of the file1 and file 2 into file3

```
$head [option] File
```

**Purpose:** "head" displays the top part of a file. By default, it shows the first 10 lines. -n allows you to change the number of lines to be shown.

```
$head [options]filename
```

**Example:**

```
$head –n50 file.txt
```

**Purpose:** Displays the first 50 lines of the file.txt

```
$head -18 filename
```

**Purpose:** Displays the first 18 lines of the file called filename.

```
$tail option File
```

**Purpose:** Displays last 10(by default) lines of a file.

Same as head command.

```
$tail filename
```

**Purpose:** Displays the last 10 lines from the ending

```
$tail -12 filename
```

**Purpose:** Displays the last 12 lines from the ending

## Editing files

There are 2 default text editors in Linux. Vi and Nano

```
$ vi filename
```

Or

```
$ nano filename
```

**Purpose:** Opens the file in the text editor.

**Note:** To save and exit from VI, press Esc then “wq”. If you want exit without saving then press Esc then “q”.

**Note:** To exit from Nano, Press Ctrl+x, it will give a prompt you if you want to save before exiting, you can press “y” or “n”.

## File Permissions:

Each file in UNIX/LINUX has an associated permission level. This allows the user to prevent others from reading/writing/executing their files or directories.

Use “ls –l filename” to find permission level of that file.

The permission levels are:

“r” means “read only” permission.

“w” means “write” permission.

“x” means “execute” permission.

In case of directory, “x” grants permission to list directory contents.

**Command: chmod**

If you own a file, you can change its permissions with “chmod”.

```
$ chmod [user/group/others/all] + [permission] filename
```

Example 1:

```
$ chmod 777 filename
```

**Purpose:** Gives user, group and others r, w, x permissions

```
$chmod 750 filename
```

**Purpose:** Gives the user read, write and execute. Gives group members read and execute. Give others no permissions.

Using numeric representations for permissions:

**r = 4; w = 2; x = 1; total = 7**

**4 stands for "read",**

**2 stands for "write",**

**1 stands for "execute", and**

**0 stands for "no permission."**

## Redirection of Input and Output

Mostly all command gives output on screen or take input from keyboard, but in Linux (and in other OSs also) it's possible to send output to file or to read input from file.

For e.g.

$ ls command gives output to screen; to send output to file of ls command give command

```
$ ls > filename
```

It means put output of ls command to filename.

There are three main redirection symbols >, >>, <

**> Redirector Symbol (Overwrite)**

Syntax:

```
[Linux-command] > filename
```

Purpose: To output Linux-commands result (output of command or shell script) to file filename. Note that if file already exist, it will be overwritten else new file is created. For example, to send output of ls command write:

```
$ ls > myfiles
```

Now if 'myfiles' file exist in your current directory it will be overwritten without any type of warning.

**>> Redirector Symbol (Append)**

Syntax:

```
[Linux-command] >> filename
```

**Purpose:** To output Linux-commands result (output of command or shell script) to END of file filename. Note that if filename exist, it will be opened and new information/data will be written to END of file, without losing previous information/data, And if file is not exist, then new file is created.

For example, to send output of date command to already existing file, give command:

```
$ date >> myfiles
```

**< Redirector Symbol**

Syntax:

```
[Linux-command] < filename
```

**Purpose:** To take input to Linux-command from file instead of keyboard. For example, to take input for cat command give:

```
$ cat < myfiles
```

You can also use above redirectors simultaneously as follows

Create text file sname as follows:

```
$cat > sname
Brown
Ali
Chang
Collins
```

```
[Ctrl+d]
```

Now issue following command.

```
$ sort < sname > sorted_names
$ cat sorted_names
```

The output will be:

```
Ali
Brown
Chang
Collins
```

In above example sort ($ sort < sname > sorted_names) command takes input from sname file and output of sort command (i.e. sorted names) is redirected to sorted_names file.

## Writing Functions and Scripts in CLI

The Linux CLI is a powerful tool and can also be used to write functions and scripts which can be executed within the CLI.

Example:

```
$ function sum()
{
Echo $(($1+$2));
}
```

**Purpose:** The above command will create a function called sum, which you can execute to get the sum of 2 numbers:

```
$sum 2 4
```

Output:

```
6
```

## Compiling C++ code in Linux

In order you compile your C++ code you need to use the Gcc or G++ compiler.

Make sure your system has a working copy of the compiler install:

```
$ gcc -- verison
```

```
$ g++ -- version
```

These should return a valid version of the compiler installed on your system.

In order you compile your C++ code:

```
$ g++ test.cpp –o testApp
```

Purpose: This command takes in “test.cpp” file and compiles it and outputs an executable code “testApp” after the “-o”.

## Running C++ code in Linux

Once you have compiled the code, you’ll get the executable file. You just need to run it in the following manner:

To run your code:

```
$ ./testApp
```

**Purpose:** This will run the executable file of your code.

## Passing Command Line Arguments to your Code

You can pass arguments to your code when executing it. This is done using the Command Line Arguments.

```
#include <iostream>
using namespace std;

int main (int argc, char** argv)
{
    cout << "You have entered " << argc
         << " arguments:" << "\n";
```

```
    for (int i = 0; i < argc; ++i)
        cout << argv[i] << "\n";

    return 0;
}
```

Type this code above in a ".cpp" file and try running it. The two variables passed to the main function are "argc" and "argv"

***Argc:*** *tells us the number of arguments passed*
***Argv[i]:*** *gives us the values of the variables passed*

## Lab Tasks

**Q1-** Create a folder with complete permissions for all users, inside it place a file with read-only permission for all users.

**Note:** You need to submit your CLI commands for this task in a simple text format.

**Q2-** Display the system date in following format:

Today is Tuesday, 15-Jan-2019

**Hint:** Look up "echo" and "date" command formatting.

**Note:** This is what your output must look like. You need to submit your CLI command(s) for this task in a simple text format.

**Q3-** Write a Cpp code which takes the User's Name as a Command Line Argument and prints:

Hello [NAME]!

as output.

**Note:** You need to submit your .cpp file.

**Q4-** Write a Cpp code which takes in 2 Command Line "Integer" Arguments and outputs their sum.

**Hint:** You might need to include <cstdlib>

**Note:** You need to submit your .cpp file.

### How to Submit

To prove your completion of this lab assignment, display the code for all parts of this lab assignment to your TA before you leave, and submit the text files (Q1+Q2) and .cpp files (Q3+Q4) to the drop box corresponding with this lab assignment on Blackboard. The TA can provide you with feedback when you show her/him your work, if requested. The Blackboard submission is just in case somehow your lab mark is misplaced; you will have proof that your lab was submitted. Comments on Blackboard are not guaranteed and are at the discretion of the TA, however, TAs are expected to provide feedback in one form or another.

***Note**: Comments are mandatory. Getting into the habit of properly documenting your program will also help you understand your own program, after a break between programming sessions.*