Introduction to GNU/Linux

For College Students

Winter 2023

Saif Terai

Introduction to GNU/Linux
Winter 2023-S

Contents

List of Figures

List of Tables

Acknowledgement

Norman Han has provided me with notes and teaching material for this course. Some review questions have been adopted from his pool of questions. A few passages in this book are written using his set of slides. Thank you Norman.

Wen Juan Jiang and Justin Martins have been using some labs during the Fall 2021 semester. I have used some of their feedback in improving the course material. Shavit Lupo has provided feedback on the `find` utility.

On a casual visit at the Foyer Art Gallery, I met Heloise Doucet. She has kindly given permission to use her artwork titled *Lanark Winter* on the cover page. Heloise is an artist from Ontario, she paints in acrylic and watercolors. Thank you Heloise.

1

Introduction

1.1 An Operating System in Context

An Operating System manages computer hardware and software, it provides an interface for users to interact with the computer, common services, and programs. An operating system is a software component of a computer. Examples of computer hardware includes firmware (BIOS), storage devices, input and output devices, and network devices among others. Software includes application software, device drivers, utilities such as backup. Common services include maintaining system time, print spooling, access control, journaling, logging. Figure 1.1 illustrates an operating system in relation to other computing resources [1].

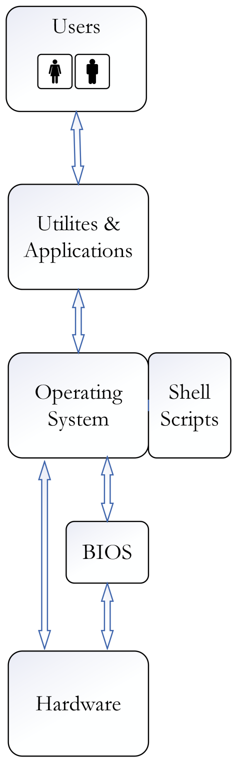


Figure 1.1: Operating System in Relation to Other Computing Resources

1.2 Components of an Operating System

Figure 1.2 illustrates some components of an operating system. The diagram has been adopted from a microkernel architecture used in some real time operating systems such as QNX [3]. The kernel is a component that shares equal importance with other components, instead being the central item. The term *software* bus is also used in early QNX documentation. A term bus is usually used in hardware; a bus is a pathway to carry data, address and control. Kernels are categorized into *monolithic* and *microkernel.* A discussion on the differences between the two architectures is not in the current scope, [3] has a short discussion on this topic.

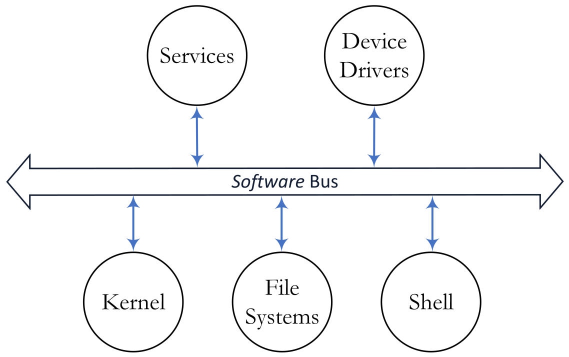


Figure 1.2: Components of an Operating System

1.3 System Information using `uname`

The vignette in figure 1.3 shows the output from `uname -a`. `uname` displays the details of the operating system, the `-a` option shows all items. It shows the kernel version, the computer architecture, it is a Symmetrical Multi-Processor SMP and the time standard as Coordinated Universal Time (UTC) .
Aside: Symmetrical Multiprocessor implies that all processors are equal. New processors can be added to the hardware and they are considered as part of the processor pool.

```
faculty@faculty-vm:~$ uname -a
Linux faculty-vm 5.11.0-34-generic #36~ 20.04.1-Ubuntu SMP
Fri Aug 27 08:06:32 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```

Figure 1.3: Output from `uname -a`

1.4 Ubuntu in the Marketplace

Ubuntu is a Linux Distribution based on Debian, [4], Page xvi. First released in 2004, for the past five years it has gained wide acceptance in cloud computing; in OpenStack and Amazon Elastic Compute Cloud (EC2). Ubuntu is named after a philosophy in the Nguni people in South Africa, it means humanity to others.

Ubuntu is released as three versions: Desktop, Server and Core. The Core edition is used for IoT and robotics. This course uses Ubuntu Desktop. Ubuntu is released every six months with Long Term Support (LTS), version 20.04 will be supported until 2025.

1.5 Linux in the Marketplace

In 2000 IBM embraced Linux [8, Pg 44-45], investing $1bn in the Linux movement and used the OS for IBM servers. Linux was soon accepted by businesses as an OS for hadoop.

1.6 Care and Maintenace of Your Ubuntu Installation

Install Updates Updates will not normally disrupt your installation; they are deployed after a testing process.

Treat your virtual installation as a regular computer. Do not close your laptop lid while VMWare is running, or close the VMWare window. Always use `shutdown -h now`, *wait* for Ubuntu to shutdown gracefully and then close the VMWare application. Closing the VMWare directly is similar to switching off the power supply to your desktop pulling without shutting down the Operating System. *Caution:* `shutdown -H` halts the system, this command is used in an emergency, on a live system. It abruptly shuts off the power, always use `shutdown -h now` on your computer.

Housing the Installation Do not move the installation files after VMWare has been installed. The location is stored in Windows registry, moving the installation files will cause errors.

1.7 Memory Management

Memory Management is a scheme that allows a computer to store and retreive data from main memory and secondary memory. A program can be run, i.e., allocated CPU cycles only if it is in main memory. Stated differently, a program cannot be run from secondary storage.
The kernel removes data from the main memory and copies this data on secondary storage in blocks, these blocks are called pages, hence the term **paging**.
Traditionally (about 1963), computers were expensive; main memory, magnetic storage, tape storage and CPU costed large sums of money. Magnetic storage was less expensive than main memory. To make the best use of main memory, processes could be taken out temporarily from main memory and stored on a magnetic device, say *Process A* was taken away. In the newly created space another program would be loaded and allocated CPU time, say *Process B*. After a certain time, *Process A* would be brought back to the main memory. This procedure swaps one process with another, hence the term **swapping**. This swapping procedure gives the illusion that the computer has more memory than it physically has, therefore the term **virtual memory**.

Swap Space

Swapping is one component of Memory Management. Swapping requires a dedicated space on secondary storage. It can be accomplished in two ways; allocating a dedicated partition on the disk, even a dedicated disk or allocating `swap files`. A swap partition has an advantage, it can provide contiguous allocation, i.e., space allocation without intervals. Having contiguous space allocation increases swapping efficiency, search time for the next page is reduced, the next page is available immediately after the previous page.

Allocating swap space, comparing techniques

Allocating a partition for swap space is done during OS installation time, this method has less flexibility when adjusting size and location, it would require computer downtime. Using files as swap space has greater flexibility in allocating size, but it is harder to get contiguous space on a drive that is in regular use. `macOS` uses multiple swap files in the root partition.

Swap Space - A Discussion

Is swap space required if main memory is large and sufficient?
There are two types of computers to consider: a personal computer and a server. It may appear that swap space is not required for a personal computer with large, and sufficient, memory. A swap space may decrease the life of your SSD drive. In a Linux server environment it is prudent to have swap space slightly larger than or equal to main memory. This allows the computer to swap less frequently used pages and free-up main memory, this will make the computer perform optimally. There are several performance monitoring utilities that allow an administrator to see swap usage. The swap size can be adjusted based on usage.

1.8 Standards

Linux belongs to a family of operating systems that comply with the IEEE Computer Society's, Portable Operating System Interface (POSIX). IEEE defines the Application Programming Interface (API), it does not specify its implementation. This ensures that each API, will function the same way on all POSIX compliant operating systems. Some examples of POSIX compliant operating systems are: AIX from IBM, HP-UX from HP, EulerOS by Huawei, and other Linux variants supported by Red Hat and other vendors.

1.9 Process Scheduling

Consider four processes P1, P2, P3 and P4 as illustrated in figure 1.4. Process P1 is CPU intensive and I/O intensive with high priority; P2 is I/O intensive; P2 is CPU intensive. Consider these processes accessing the CPU; time elapses from left to right. The CPU can accept only one process at a time. P1 is a process that requires CPU cycles for computation and it also needs access to data from storage. At first process P1 is given CPU time, while the process has CPU time it requires access to data from the hard disk, the CPU does not keep idle, it services the next available process, P2. Say, P2 is an sound player that accesses an `.mp3` file that the user wants to play in the background. It requires little CPU time, the sound is played by the sound chip, the CPU still has to initiate the process. A short time slice is given to process P2, it then accesses the sound data from storage and completes without the CPU's attention. The next process waiting is P3, it is a CPU bound process requiring no I/O, after its share of CPU cycles is complete process P1 is serviced again, after it has finished its I/O. Process P1 is a high priority process that needs to complete first, say a spreadsheet. The scheduling process continues until processes keep appearing in the queue.
The diagram shows full CPU utilization. On a *uix system with a single user, it is likely the CPU will be idle most of the time. On a *uix system with several users the CPU could be busy with several hundred processes running. Larger systems will have more than one CPU. They are referred as *multi-processor* systems. Each CPU services several processes one after the other, this phenomenon is called *multi-processing*. The two terms are similar sounding and can be confusing. The CPU *switches* its time between each process rapidly giving the *impression* that each process is getting the CPU's attention all the time. The process of switching processes is called **context-switching**. A process is often referred to as a task.

An I/O bound process, for example a large copy operation does not need the CPU's attention during the copying activity, but the task must be initiated by the CPU. After it completes the kernel will clean the process table and remove the task from the queue. Assignment of processes to the CPU is done by the **scheduler**.

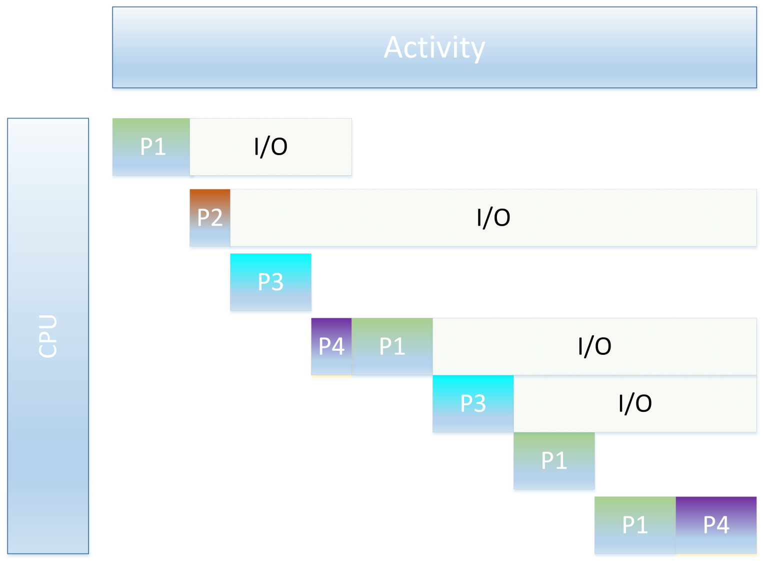


Figure 1.4: CPU Utilization

Process Table A process table is a list of processes, it tells the user, among other details, the process priority, process status (idle, running, sleeping, stopped, zombie), process dependencies, i.e., its parent process and child processes, CPU Utilization, memory usage. There are several ways to see a process table.

1. use `ps aux`
2. use `top`
3. install a curses based interactive program `htop`, written by Hisham Muhammad.

Process & Program A process is an instance of a program that is running. A program is a set of instructions residing on a storage medium —it is passive. After a program is loaded in main memory (RAM), it consumes CPU cycles it becomes —it is active. A program starts as a process. While running a process may spawn other processes. Code to spawn other processes is written within the parent process. The spawned process runs as a child process. Alternatively, a process can invoke other programs residing on the storage medium.

1.10 Preemption

The scheduler can take away a process from a resource before the process has completed its task, this action is called preemption; the resource is the CPU. There are many reasons for preemption to occur, some of them are: [1] Page 113.

- To allow other processes to access the CPU, this is done to allocate CPU resources fairly to all processes in the system, it prevents CPU starvation to processes.
- A process with the higher priority needs the CPU, this action is taken to keep the system stable.

- A process may voluntarily relinquish the CPU. For example, a background process (daemon) performing maintenance and OS accounting functions will use this mode.

Aside: In a laptop the CPU is idle most of the time, yet preemption occurs. Preemption is appreciated in a large system with limited CPU resources and several processes running and on very small systems with limited CPU resources, such as embedded system. Consider a medical device such as a pacemaker, a dialysis machine that needs to function with no failure.

1.11 Interprocess Communication

Processes rarely work in isolation, they communicate with each other. Linux provides a rich set of mechanisms that facilitate processes from communicating with other processes, thus the term Interprocess Communication (IPC). There are several types of IPC's, systems designers will use a method suited for an application. A simple IPC is a pipe, the out of one process is transferred as input to another process.
Pipes are FIFO mechanisms, the first message to be sent to the pipe is the first to be consumed by the receiver. Once the message is consumed it is not available. Think of situations where messages can be prioritized, the receiver has the option to read the message but not consume it, the receiver can inspect a message and then decide to read it based on the message type.
More sophisticated IPC's include *message passing* and *shared memory*, these mechanism are used in processes on the same computer. Processes on different computers use *sockets*, sockets can be used for processes that reside on the same computer. For example, a postgres server running on a computer will use the local host's ip address 127.0.0.1 and will listen on port 5432. Clients such as pgAdmin3, pgAdmin4, pgmodeler will communicate with the server using this IPC mechanism.
IPC's can be between two processes; three processes, one sending two receiving, or one receiving and two sending. It is possible that many processes communicate send and receive at the same time, message passing allows this level of sophistication.

1.12 Command, Command Options

Unix commands are issued and executed elegantly. Most commands are succeeded by an option. For example, `grep -i` or `ls -a`. Commands and utilities were designed and developed by several programmers at different locations at different times. The meaning of each option is not consistent. For example, in `grep -r` the `-r` is for recursion, in `ls -r`, `-r` is for reverse alphabetical order. It takes some practice to be familiar with the options. Experienced Unix users have to refer to the manual pages for rarely used options, so do not feel intimidated.

1.13 The Kernel

The kernel is the first program to be loaded into memory, this process always remains in memory. Traditionally the entire kernel remained in memory, i.e., it could not be swapped out, the computer must have at least the minimum amount of memory to run the kernel. This kernel architecture is the monolithic kernel. Some specialized operating systems uses the micro-kernel architecture, components of the kernel load in memory when required. An error, or bug, in a monolithic kernel will bring down an entire computer system. Compare this to a micro kernel architecture; an error in a kernel component will affect a small portion of a computer system and will usually be able to recover or run in a sub-functional mode. *Aside:* Do not use the

derogatory term *crippled* to refer to a partially functional mode. Micro-kernel architecture computers are more resilient compared to a monolithic kernel computer. Micro kernel architecture have an overhead when loading kernel components, compared to a monolithic architecture, the entire code in pre-loaded. Examples of micro kernel architectures are the `minix` operating system and QNX . Microkernel architecture is well suited for small computers and embedded computers.
Linux is a monolithic kernel architecture, although kernel modules can be loaded and removed when not required. These loadable kernel modules, called `LKM` is executable code that extend the functionality of a running kernel.

1.14 Review Questions

Multiple Choice

1. Identify the tasks performed by an Operating System. Select all that apply.
   A. Manage Computer Hardware.
   B. Launch and Run application programs.
   C. Provides an interface between the user and computer hardware.

2. An computer and its operating system allows more than one user to work at the same time. There is only one CPU, users is able to complete their tasks, as if each user has his/her own CPU. Identify the term used to define the above concept.
   A. time sharing
   B. single user
   C. real time
   D. batch processing

3. A computer's Main Memory is abstracted to another device which has a larger storage space than the main memory. This technique allows a user to run programs that are larger than the main memory available to a user. Identify the technique mentioned above.
   A. virtual memory
   B. cache memory
   C. solid state device
   D. firmware

4. A user presses a key, the CPU responds. Stated differently, the CPU responds only when a user, a device or a program needs attention. Otherwise the CPU will be performing other background tasks or will be idle. Identify the concept mentioned above.
   A. interrupt driven
   B. polling

5. A program can run in one of the two modes at a given time. This separation *minimize*s the chances of a malfunctioning program from crashing the operating system. Make two choices.
   A. user mode
   B. single-user mode
   C. multi-user mode
   D. kernel mode
   E. polling

6. A portion of code as part of a larger program that will count down and perform a task. This count down mechanism can be used to terminate a program if it has failed to respond because of an error. Identify the term used for this type of code.
   A. interrupt
   B. polling
   C. timer
   D. alarm

7. A kernel
   A. gets loaded into system memory and runs code in kernel space.
   B. gets loaded into system memory and runs code in user space.
   C. gets loaded in ROM BIOS and runs from the BIOS.
   D. gets loaded from flash memory then gets loaded in BIOS.

8. A user edits a document using `vim`. Identify the memory area where it runs.
   A. kernel space.
   B. user space.
   C. device driver routines.

9. Kernel space is protected because
   A. user data is confidential and should not be viewed by other users.
   B. an error in an application program is isolated and does not bring down the entire system or make it unstable.
   C. user data can be large and will not fit in kernel space.

10. Identify the software that interacts with and processes requests from peripherals such as keyboard, mouse, camera, microphone, hard disk, CD ROM, Flash Drive.
    A. Kernel
    B. application program
    C. device driver
    D. BIOS
    E. flash memory

11. A task is currently running using the CPU. It is interrupted by the kernel with the intention of resuming later. Identify the term that describes this scenario.
    A. multi user operating system
    B. single user operating system
    C. pre-emption
    D. multi processor system

12. Identify the term that matches this description. *Programs usually started at boot-up time, provides service(s) to other programs that are run by a user or by the system.*
    A. background process
    B. foreground process
    C. daemon
    D. kernel

13. Which command will restart the system immediately.
    A. `shutdown -r now`
    B. `shutdown -h now`
    C. `shutdown -P now`
    D. `shutdown -k`
    E. `shutdown -c`

14. Identify the command that will give the name of the operating system that is currently running.
    - A. `passwd`
    - B. `uname`
    - C. `hostname`
    - D. `whoami`
    - E. `su`
15. Which one of the two pathnames will work regardless of the current directory the user is in.
    - A. absolute path
    - B. relative path
16. Identify the interprocess communication mechanism that is First In First Out (FIFO)
    - A. pipe
    - B. shared memory
    - C. message passing
17. Identify the term that describes `swap` in context of operating systems
    - A. exchange of two variables in a programming language.
    - B. transfer of data from main memory to secondary storage.
    - C. removing one device and replacing it with another device.
18. The term Symmetric Multiprocessor indicates
    - A. A single operating system manages more than one processor, all processors share all I/O devices and all processors share the same memory.
    - B. A single operating system manages manages only one processor.
    - C. Some I/O devices are available to only some processors.
    - D. Main memory can be accessed by only one processor.
19. The term `context switching` implies that
    - A. there are many CPU's each one devoted to a single process.
    - B. the CPU can attend to only one process until completion.
    - C. the CPU switches from one process to another, the state of the previous process is saved.
20. The task of the `scheduler` is
    - A. to remove old pages from the RAM and store it on secondary storage.
    - B. to assign a process to the CPU.
    - C. to connect a peripheral to a computer.
21. Portable Operating System (POSIX) is
    - A. an operating system that can run on any hardware.
    - B. a standard specified by IEEE Computer Society for maintaining compatibility between operating systems.
    - C. a hardware that can run all operating systems.
22. Identify a technique that specifies a search pattern
    - A. `useradd`
    - B. `find` command in Linux
    - C. regular expression
    - D. `cut` and `paste` utility in Linux

23. A `daemon` is a
    - A. a form on interprocess communication (IPC)
    - B. another name for a device driver.
    - C. background service that provides services; examples of services are: login, printing, error logging.

24. `pre-emption` occurrs when
    - A. an operating system comes pre-installed on hardware.
    - B. a process is taken away from the CPU to allow another process access to CPU cycles.
    - C. main memory is swapped out on secondary storage.

25. Data redirected to the pseudo device file `/dev/null`
    - A. can always be recovered
    - B. cannot be recovered

Review Questions, Answer True or False

1. `rsyslogd`, `ftpd`, `httpd` are examples of daemons.

2. Multiuser is the same as multitasking. These two terms can be used interchangeably.

Written Answer

1. Differentiate between multiuser and multi-tasking operating system.
2. Compare and contrast a microkernel architecture and a monolithic kernel architecture.
3. Write a short comparison between any two IPC's.

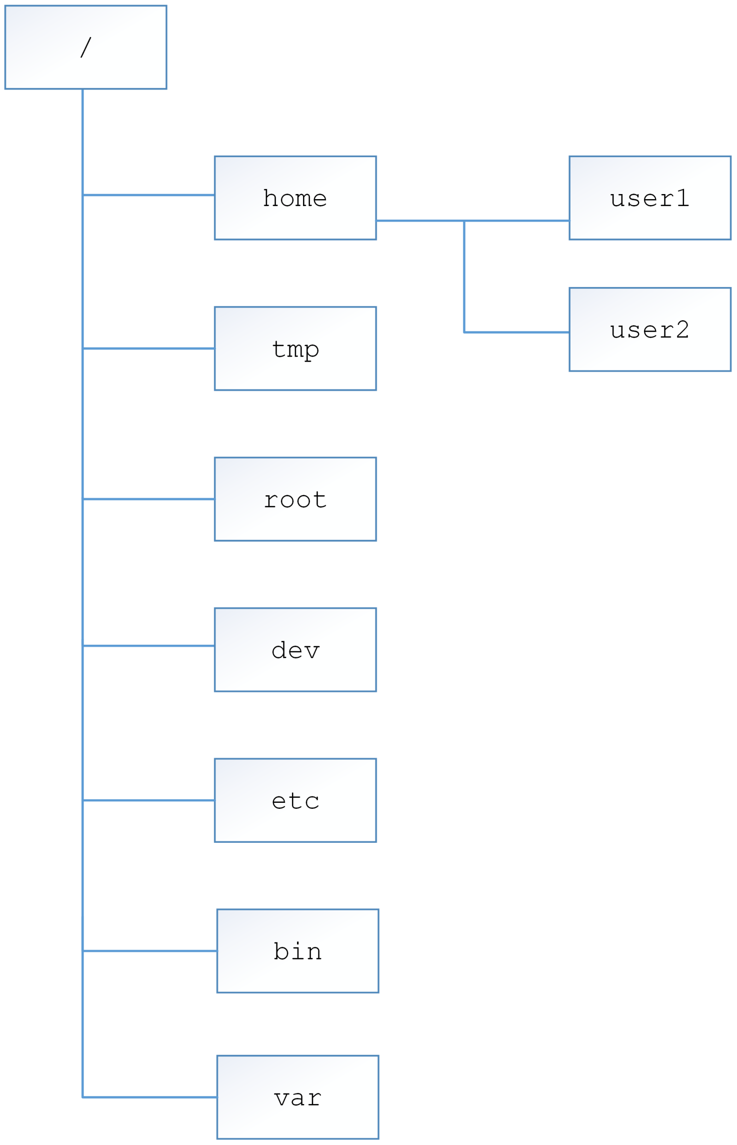


Figure 1.5: Linux File Hierarchy, Partial List

Figure 1.5 on page 24 is a representation of the Linux File System. As an example, `/sbin` has system binaries, `/bin` has binaries for user commands and utilities. For example, `/bin` has ls, grep, wc, who, mount, umount, tail, head, gcc, find, echo.
`/sbin` has fsck, fdisk, mkfs, adduser, chroot, groupdel, groupmod, groupadd, useradd, userdel, usermod.
`/etc` stores system configuration files such as passwd, gshadow, hostname, and many `.conf` files.

1. The root directory of the filesystem is represented by `/`. What is the difference between `/` and `/root`?

2. As an administrator you are creating new users. Where would the users home directory reside?

3. You insert a USB drive in your computer. Where will you see the contents of the drive in the file system,

besides seeing the drive on the Desktop.

4. What files does `/sbin` have? How is `/sbin` different from `/bin`? Name some files in each of the two directories.

5. Where are *host specific, system configuration* files stored?

6. Where are device files stored?

2

Basic Commands

Frequently used and essential commands and utilities are briefly described in this chapter. Try all suggested exercises and review questions. You are encouraged to refer to the `man` pages for additional options to the commands.

2.1 Motivation to Learn Linux Commands

There are several reasons to learn Linux commands and run them from the command line interface (CLI)). Every Graphical User Interface (GUI) option always has a corresponding command line option, but not every command line option will have an equivalent GUI option. Running utilities from the command line with its options is a necessary skill for programmers and administrators working on a server. Often servers will not have a GUI installed, it may not have a mouse either. Programmers configuration software, for example conditional compilation, these tasks are performed easily from the command line. System administrators need to automate routine tasks, writing a script is the most efficient method to perform administration tasks. Navigating the File System and knowing the properties of the files and directories is required. You will be writing code and its GUI that will create directories, copy and move files, create soft links among other tasks; to provide the user with a GUI will need to know the corresponding utilities.

2.2 `mkdir` & `rmdir`

`mkdir:` Makes a directory. `mkdir Federal` will create a directory `Federal`, `mkdir Federal/Provincial` will create a directory `Provincial` under directory `Federal`. To create a directory `Day` under `Year/Month`. The command `mkdir Year/Month/Day` will fail because the parent directories `Month` and `Year` do not exist.

To create parent directories `mkdir` needs the `-p` option. `mkdir -p Year/Month/Day` will create the parent directories if they do not exist.

`rmdir:` `rmdir Year/Month/Day` will remove the directory `Day`, if it is empty, i.e., it does not contain any files. To remove the directory `Month`, the command `rmdir Year/Month` is required. And finally to remove the directory `Year` the command `rmdir Year` is required. The three commands will remove the entire branch.

The three commands are equivalent to one command with a `-p` option. `rmdir -p Year/Month/Day`

`rmdir` accepts more than one directory. For example, `rmdir Year/Month/Day Year/Month Year` will also remove the branch. In this example first the directory `Day` is removed, then `Month` and finally `Year`. The

commands will succeed only if the directories do not contain any files.
`rm -r` removes files and directories recursively. For example, `rm -r Kingdom` will remove all files and directories under the directory `Kingdom`

Exercise: Will the command `rmdir Year Year/Month Year/Month/Day` succeed? Explain.

Exercise: Create a directory tree `Kingdom/Phylum/Class` using `mkdir`. Then remove the entire tree using `rmdir` as a single command. Create the same directory tree and remove the entire branch using `rm` as a single command.

`nice:` `nice` will perform three tasks. 1. run a program with modified scheduling priority. 2. change the priority of a running program and 3. show the current priority level the program is running at.

`nice` values range from `-20` to `19`. `-20` is the highest priority to give to a process, `19` is the lowest priority. A process running at `-20`, the highest priority, will get maximum CPU resources at the cost of denying the other processes of CPU cycles. The root, or other users with privileges, can increase the priority of the process, i.e. set a lower value say `-10`.

A `nice` value applies to processes that are about to be launched or to processes that are running; `nice` is not the same as scheduling priority. A nice value is an advice to the scheduler, the scheduler may choose to ignore the `nice` value given to a process.

By default the `nice` value of any script or shell is 10. Type `nice` at the command prompt to find the current nice value. `nice` has limited ability when running Linux in a Virtual Machine. The host OS has control over the CPU.

Exercise: Explain the limitation of `nice` when the OS is run as a virtual machine.

2.3 `touch`

The `touch` utility changes a files access time and modification time. If a files does not exist it creates it with default permissions. The access time is the time when the file was last used (accessed).
Using `touch filename` will change the access time and modification time. You can choose to change the access time with `-a` or the modification time with `-m` flag. The touch utility will take in time values as well, i.e. you can change the timestamp of a file to a value that you chose. An unauthorized user can compromise the security of a system by misusing this utility.
In a programming environment `touch` is frequently used to change the timestamp of a file or group of files to force them to recompile. The `make` utility will compile and link files only those files that have been modified by the programmer, touch forces the utility to compile all files.

Exercise: Try `touch -t [[CC]YY]MMDDhhmm[.ss] <filename>`
Verify the result using `ls -l <filename>`
Replace CC with Century, YY with Year. What do the square brackets indicate? What do the angle brackets indicate?

2.4 Remove Files and Directories

`rm` Removes files and directories.
`rm King Queen Jack` will remove the three files.
Use `rm -r` to remove directories and files recursively.
For example, `mkdir -p Federal/Provincial/Municipal` will create the tree. `rm -r Federal` will remove the entire tree.
Caution: `rm -r /` with root privileges will prune your directory.

Exercise: What does `rm -i` do?
What does `rm -f` do?

2.5 Copying Files and Directories

`cp` copies files from source to destination. `cp frog Kingdom/Phylum/Class` will copy the file frog to the directory `Class`. `cp` can accept more than one filename. For example,
`cp frog toad Kingdom/Phylum/Class`
will copy the two files frog and toad to the directory `Class`
Some options with `cp`
`-i` interactive mode, prompts before overwriting target file.
`-b` creates a backup at destination, backup will have a suffix ~
`-u` copies only if source file is newer or destination file is missing.
`-r` copies files recursively.
`--parent` appends source path in destination. For example, `cp --parent /etc/passwd ~/`

2.6 Renaming Files, moving files

`mv` moves files from source to destination. Note: to rename a file use the `mv` command. It is the same as rename, Unix filesystem has a unique feature which will be discussed in a later chapter.
`mv file /dir` moves `file` to a directory `dir`
`mv deer elk` renames `file` deer to `elk`
Options, `-i, -b, -u`

2.7 Hidden Files

Hidden Files Each directory will have atleast two hidden files. They are `.` and `..`, the single dot is the current directory, the two dots represent the parent directory. There are several other hidden files, especially in the users home directory. Each user will have a startup file called `.bashrc` the `ls -al` command lists all files, including hidden files.

2.8 Absolute and Relative Path

Absolute Path and Relative Path You are in your home directory at `/home/student005` and you want to change your current directory to `/tmp`, there are at least two way to accomplish this.

1. `cd /tmp`
2. `cd ../../tmp`

The first method uses the absolute path. You can use this path in whichever directory you are in. The command explicitly uses / the root as the starting point.
The second method uses a relative path name. By specifying `..` you refer the to parent directory, i.e. you go one level above, and again one level above, effectively you are now in the / directory, from there you traverse down to the `/tmp` directory.

Each method has its uses. In the example above, the relative path will not work if you are within another directory in your home directory. For example, if you are in the Music directory within your home directory you will need to use `../../../tmp`, i.e. you need to go three levels above, instead of `../../tmp`

2.9 Concatenating files

`cat` is the short form for concatenation i.e. to join; it displays contents of file to screen or other devices. `cat` can accept more than one file.

```
cat /etc/passwd
cat /etc/passwd /etc/fstab
```

tac prints files in reverse.

2.10 Displaying a directory structure

`tree` is a useful utility. It displays a directory listing using some graphic characters. Install it using

```
sudo apt-get install tree
```

Options: `-a -d -L`

Example: `tree -L 1 /` will list level 1 directories from the root.
From you home directory simply typing `tree` will list the directory structure of your home directory.

2.11 Redirecting Output

Redirecting Standard Output

Output from a command can be redirected to another device or file. The standatd output is the console `stdin`. The output can be redirected to another device. *Aside:* All hardware devices in Unix are treated as files; there is only one exception.
The > is the redirection operator. Examples,
Redirect output of the ls command using `ls > file1`

```
cat /etc/fstab > file2
date > today
```

If an output file does not exist it will be created, if a file exists it will be overwritten.
The append redirection operator >> will append the output to the target device (file). If a file does not exit it will be created, if a file exists it will be retained and the contents appended. Examples,

```
date >> today
ls >> today
```
Two separate outputs are redirected to a device (file).

Redirecting Standard Error

The standard error output device is `stderr` which is the console by default. The error can be redirected to another device using the 2> or 2>> notation.

2.12 Querying the Commands

Three commands provide information on commands and utilities in Linux, they are `which`, `whatis` and `whereis`.

`which`

`which` traverses your search path and looks for executable files and lists them, i.e., shows the file's location. `which` *Note 1:* Executables are binary executables and scripts.
Note 2: Scripts must have executable permission; permissions are covered in a later chapter.

Exercise: `which find`
`which grep`
The search path should ideally be `/usr/bin`, `/usr/sbin`, `/usr/local/bin` followed by your personal directory and personal `bin` directory.
Note: Unlike DOS, *nix does not search your home directory by default. Your home directory is not in the default `PATH`

`whereis`

looks for the filename and displays it. It will also search for the commands source code and its manual pages, if they exist. `whereis` does not require the file to be an executable. Use `whereis` to locate a file quickly if you remember its name. *Aside 1:* The manual page of `whereis` tells you that it lists binary, source and manual pages. In practice it lists all types of files.
Aside 2: Manual pages are stored as compressed files, hence the `.gz` extension. When a user requests the man pages, they are uncompressed on demand and displayed.

`whatis`

provides a brief explanation of Linux commands from the `man` pages. User written scripts are not recognized by `whatis`.

Exercise: Type `whereis find | head -4 | tail -1` and compare the output to `whereis find`.

`type`

provides information about a command and its full path. It tells you if the command is an alias, shell builtin, a function or a keyword.

Exercise: Try `type ls`, observe the output.
Try `type -a ls`, the `-a` option provides the path names and location of the `ls` command.
Run the *actual* `ls` command from its home directory using `/usr/bin/ls`, observe the difference between the two output of the `ls` command.
Aside: `alias` and `builtin` are discussed later in the course.

2.13 Command History

To display a list of commands you have typed, type in the `history` command. `history -c` will clear the history. Commands are stored in the file. The name of this file is defined by a shell variable `HISTFILE`. Type in `echo $HISTFILE`, by default the history file is a hidden file `.bash_history`
By default 1000 commands are stored; this number is stored as a variable `$HISTSIZE`
The last command can be re-run using `!!`
The history command lists the commands with its number, type in `!n` for a command to run; replace `n` with the number of the command in history.
`!-n` runs a command with an offset of `n`. For example if you want to run a command that has been issued three commands earlier type in `!-3`

2.14 Login Scripts

Table 2.1 lists three scripts that are run when a user logs in Linux.

Script	**Description**
`.bash_profile`	Individual user's login shell initialization. Executed each time a user logs in.
`.profile`	This file is created by default in the users home directory. If `.bash_profile` does not exist then `.profile` is run. If `.bashrc` exists it will run `.bashrc`. If a `bin` directory exists in the users home directory, `/home/bin` is added to the path.
`.bashrc`	Contains an individual users interactive non-login shell initialization script. Executed each time a new shell is spawned. Stores configuration parameters and default user preferences, such as prompt and color scheme.

Table 2.1: Login Scripts

2.15 Pipes

Unix has a sophisticated mechanism to manage inter-process communication (IPC). IPC allows processes to communicate with each other. Choosing the correct IPC allows a software designer to communicate with processes within the computer and on another computer.

One simple IPC mechanism is a pipe. There are two types of pipes; unnamed pipe and named pipe. Pipes are FIFO mechanisms, the earliest input is out first. An unnamed pipe is commonly used at the command line, for example, in `ls -al | less` the pipe symbol directs output from one command `ls -al` as input to another command `less`.
A named pipe, as the same suggests has a name, i.e., it is a file. Output from one command is directed to this pipe, later another program will consume the data.

Exercise:

1. Create a pipe using `mkfifo baton`
2. Verify the read write privileges and the file type of `baton` using `ls -al baton`. Notice the file size.
3. Send some data to `baton`. The next three statements will direct the output to the pipe. The command prompt will not appear after each statement. Keep typing the commands one after the other followed by a return.
   ```
   echo $( date ) > baton
   echo $( whoami ) > baton
   tail /etc/passwd > baton
   ```
4. Keeping this terminal, open a fresh terminal, preferably side by side.
5. Verify the size of `baton`, is it sill zero bytes. Where did the data go?
6. Consume the data from the pipe by typing `cat < baton`, three times. Each time data is retrieved from the pipe.

This mechanism has several advantages compared to redirecting the output to a file. For large files, the data storage is managed by the filesystem, the space in the users directory is not used. Pipe consumes the data, i.e., the pipe is emptied once the data is used; in a file data is persistent, it has to be removed after consumption. Several programs such as postgres will read data from a pipe and insert it into a database.

2.16 Word Count

The `wc` utility prints the number of newline characters, words and will count the number ot bytes in a file.
Options: `-w` counts words
`-l` counts lines
`-c` counts characters (bytes) in a file.
By default `wc` will count all three items.
Examples:

```
wc -l /etc/passwd
wc ~/*
```

2.17 Prompt

A prompt is set of terminal, host and computer details that you see at the command line. There are four prompts in Linux, this section discusses the first two prompts; the primary and the secondary prompt.

Primary Prompt

The most common prompt is the primary prompt, the shell identifies it as `PS1`. On the terminal type:
`echo $PS1`
bash displays a string of characters and color codes that are used to construct your current prompt. Some common codes are:
`\d` for date
`\h` for hostname
`\u` for username
`\n` for newline
`\s` for shell
`\t` for time
`\w` for ~
Experiment with `PS1="\w"`
The original prompt will reappear in a new terminal. Later we will discuss saving the primary prompt, recovering it and keeping it permanent for each new terminal session.

Try to change the primary prompt using:
`PS1="\u@\h:\w$"`

Secondary Prompt

The secondary prompt is represented by `>`, it is used to continue an unfinished command line, i.e., to type in a long command that continues on more than one line. *Caution:* Do not confuse the secondary prompt with the redirection operator. The same symbol is used but in a different context. Observe the secondary prompt by typing: `echo $PS2`

2.18 `head` and `tail`

The two utilities `head` and `tail` extracts specified number of lines from text files.

`head`

`head` displays a specified number of lines from a text file. By default it will display the first 10 lines from a file. For example,
`head Line_Data.txt`
Aside: The file Line_Data.txt is available on One Drive at
`Course Content → Scripts → 00-Misc`
`head -20 Line_Data.txt` displays the first 20 lines of the text file.

`tail`

works similar to `head` but from the bottom of the file.
`tail Line_Data.txt` will display the last 10 lines from `Line_Data.txt`

Exercise: Use `head` and `tail` in combination to display lines 26 to 35 from the file `Line_Data.txt`
Tip: You will need another tool that you learnt earlier.

2.19 A Short Tutorial on `touch` and `cp`

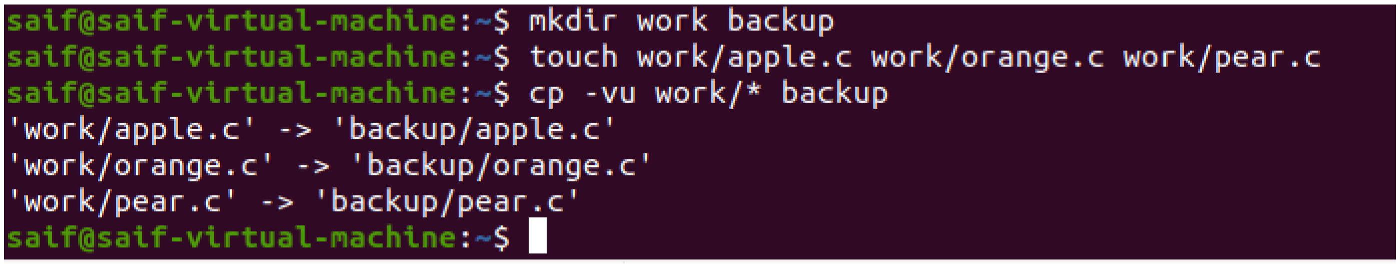
Objective

This short tutorial is intended to familiarize you with `cp` and `touch`. It demonstrates some options with `cp` and the usage of the `touch` utility in programming environment. *Aside:* Observe how the commands `rm` and `mkdir` are used. Pay attention to the use of the / and /* after the directory name.

Procedure

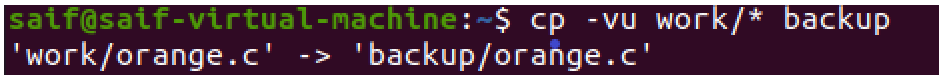
`cp` has several options; this short tutorial will review two options `-v` verbose and `-u` update. The verbose option provides a short narration of the files copied. The update option copies only those files that have changed, i.e. the source file is newer; it will also copy the file if the destination file is missing.
From your `home` directory try the following exercise. The commands you need to type are in `courier font`.

1. Create two directories.
   `mkdir work backup`
   Think of the `work` directory as your main *working* directory for editing source code files. The `backup` directory is used to store the older version of the working directory.
2. Create three empty files in the `work` directory.
   `touch work/apple.c work/orange.c work/pear.c`
3. Take a backup of the three files in the `backup` directory using the verbose option in `cp`.
   `cp -vu work/* backup`
   Confirm the copy using `ls -l work/ backup/` Observe the date and time of the files in both the directories.



```
saif@saif-virtual-machine:~$ mkdir work backup
saif@saif-virtual-machine:~$ touch work/apple.c work/orange.c work/pear.c
saif@saif-virtual-machine:~$ cp -vu work/* backup
'work/apple.c' -> 'backup/apple.c'
'work/orange.c' -> 'backup/orange.c'
'work/pear.c' -> 'backup/pear.c'
saif@saif-virtual-machine:~$ 
```

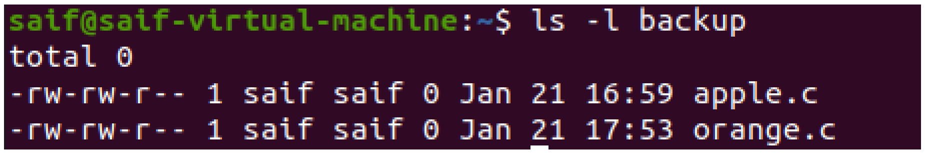
4. Wait a full minute for the time to change. Using the `touch` command on a file after the time lapse, will give the file a new timestamp even though the file has not been modified. After a full minute, alter the modification time of the file using the touch command.
   `touch work/orange.c`
5. Take a backup (copy) of only the file that you have changed.
   Use `cp` with the options `uv`.
   `cp -uv work/* backup`
   Do you see only one file being copied?



```
saif@saif-virtual-machine:~$ cp -vu work/* backup
'work/orange.c' -> 'backup/orange.c'
```

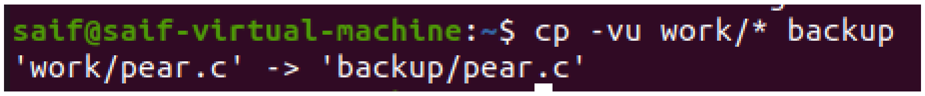
6. Now type in the same command once again. You should not see any files being copied, your backup directory is upto date.

7. Delete `pear.c` from the backup directory.
   `rm backup/pear.c`
   Verify that there are only two files in the backup directory using `ls -l backup`.



```
saif@saif-virtual-machine:~$ ls -l backup
total 0
-rw-rw-r-- 1 saif saif 0 Jan 21 16:59 apple.c
-rw-rw-r-- 1 saif saif 0 Jan 21 17:53 orange.c
```

8. Backup your `work` directory using.
   `cp -uv work/* backup`
   Only one file `pear.c` should be copied. This is the file that does not exist in the target folder.



```
saif@saif-virtual-machine:~$ cp -vu work/* backup
'work/pear.c' -> 'backup/pear.c'
```

9. Confirm that the deleted file is copied.
   `ls -l backup`
10. Remove all files and directories.
   `rm -r work backup`

2.20 Review Questions

1. Identify the root user's home directory in Linux.
   A. `/`
   B. `/root/home`
   C. `/home/root`
   D. `/root`
2. The Linux command that allows for deleting a directory, but only if it is empty:
   A. `mv -u`
   B. `rm`
   C. `rmdir`
3. The Linux command that will display the current working directory
   A. `pwd`
   B. `cd`
   C. `mkdir`
   D. `ls`
4. Identify the command that will delete directory `~/Images` and files stored in it.
   A. `rmdir ~/Images`
   B. `rm ~/Images`
   C. `rmdir -p ~/Images`
   D. `rm -r ~/Images`
   E. `rm -r /Images`

5. Identify the command that is used to change directory to `test1`, a sub-directory of `user1`'s home directory, `/home/user1`, given the command prompt listed below:
   `user1@localhost:/etc$`
   A. `cd /test1`
   B. `cd ./test1`
   C. `cd ../home/user1/test1`
   D. `cd home/user1/test1`

6. Identify the redirection operator, that will not overwrite an existing file but will append it.
   A. `>`
   B. `>>`
   C. `|`

7. Identify the operator that will stream the output of the first command and send it as input to the second command.
   A. `>`
   B. `>>`
   C. `|`
   D. `;`

8. What will the command `echo $SHELL` do?
   A. it will run the default shell
   B. it will show the default shell as specified in the environment variable `SHELL`
   C. it will show all shells currently in the system

9. What does the command `cat file1 file2` do?
   A. redirects output of `file1` to `file2`
   B. concatenates files `file1` and `file2` and prints to standard output, (i.e. default - screen)
   C. takes each line from `file1` and appends it to corresponding line in `file2`.

10. What does the command `touch file1` do?
    A. creates `file1` with zero bytes, gives an error if `file1` exists.
    B. creates `file1` with zero bytes if it does not exist, if `file1` exists `touch` will update the time stamp of the file.
    C. creates `file1` with zero bytes, if `file1` exists `touch` overwrites and creates a file with zero bytes.

11. Identify the command that will rename a file in Linux.
    A. `ren`
    B. `rename`
    C. `cp`
    D. `mv`

12. Select one correct choice.
    The `history | tail` command
    A. will list the last 10 commands issued at the prompt
    B. will list all commands issued at the prompt
    C. will give an error

2.21 Lab 1 - Ubuntu Installation

Objective

1. Install VMWare.
2. Install Ubuntu.
3. Open a Linux Terminal and run a few simple commands.
4. Gracefully shutdown the Linux installation.

Reference

1. www.ubuntu.com
2. www.vmware.com

Submission

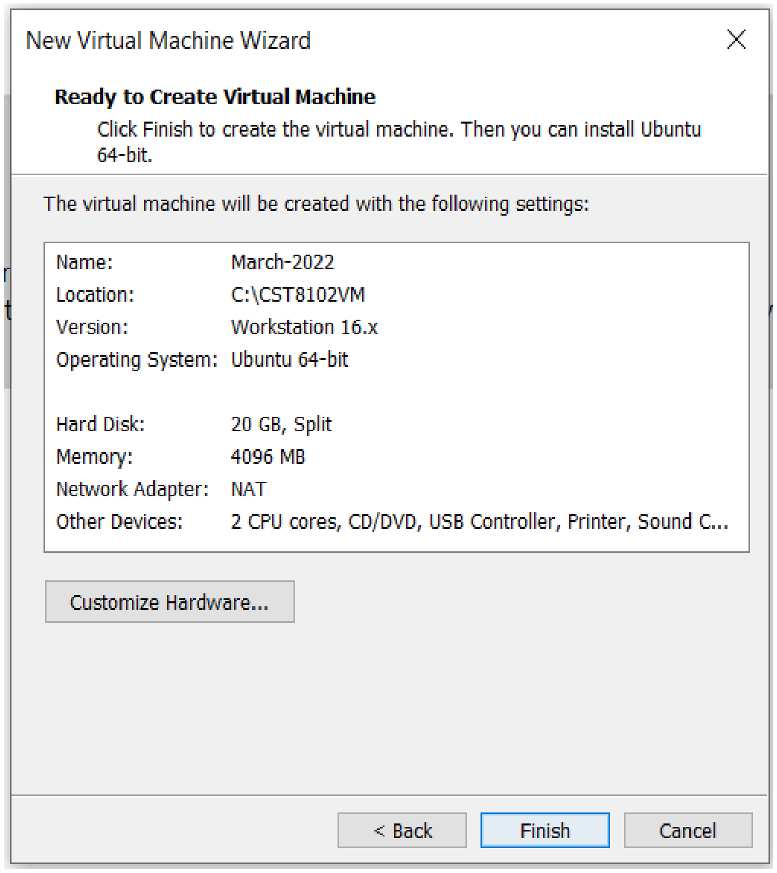
Upload a screen shot of your Ubuntu Installation to Brightspace. It should show your login name. Refer sample submission.

Requirements

1. VMWare Workstation 16 Pro or Later version
2. Ubuntu ISO last LTS version.
3. Software Resources Portal from Algonquin College, for VMWare Licence Key.

Procedure

1. Create a New Virtual Machine.
   (a) Start a VMWare Workstation.
   (b) Select File → New Virtual Machine.
   (c) Type of Configuration: Select Typical (Recommended) → Next
   (d) Guest Operating System Installation: Choose `I will install the Operating System Later` → Next. *Caution:* Do not use the option `Installer disc image file (.iso)`
   (e) From the four choices for OS, Select `Linux`, and `Ubuntu 64-bit` default.
   (f) If you prefer, rename the Virtual Machine, preferably with the month and year of the image.
   (g) **Important:** Selecting a location to store the Virtual Machine Files. Go to `Windows C:` or `D:`, create a folder using `Make New Folder` call it `CST8102VM`. Do not use the `Desktop` or `My Documents`. Microsoft's One Drive will try to sync these folders, the Virtual Machine will not work correctly. Choose a folder on the drive that One Drive does not synchronize.
   (h) On the next window, select Default Disk Capacity, and `Split virtual disk into multiple files`. This option will not use all 20 GB at once, but will use your disk space as your needs grow.
   (i) Compare your installation with figure 2.1



Ready to Create Virtual Machine

Click Finish to create the virtual machine. Then you can install Ubuntu 64-bit.

The virtual machine will be created with the following settings:

Name:	March-2022
Location:	C:\CST8102VM
Version:	Workstation 16.x
Operating System:	Ubuntu 64-bit
Hard Disk:	20 GB, Split
Memory:	4096 MB
Network Adapter:	NAT
Other Devices:	2 CPU cores, CD/DVD, USB Controller, Printer, Sound C...

Figure 2.1: New Virtual Machine Wizard

2. Editing the Virtual Machine. Compare your screen with figure 2.2

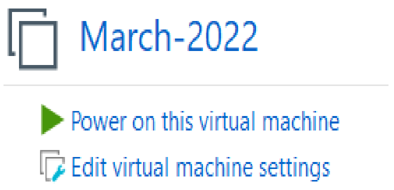


Figure 2.2: Editing the VM

   (a) Visit `www.ubuntu.com`, in the Download page, select Ubuntu Desktop 22.04 LTS. *LTS*, means Long Term Support.
   (b) Download the image. The image may be available on Brightspace, this could save you time, downloading from Brightspace will be faster.
   (c) With figure 2.2 as reference, select `Edit virtual machine settings`.
   (d) Select `CD/DVD (SATA)` from the Hardware tab. Note: There are two tabs, `Hardware` and `Options`.
   (e) Select `Use ISO image file:` and locate the `.iso` image on your hard disk. Click `OK`.

3. Start the Virtual Machine.
   (a) Select Power on this `virtual machine`. The installation will start.
   (b) Select `Install Ubuntu` on the welcome screen.
   (c) Select `Continue`, `Preparing to install Ubuntu`

   (d) Select `Erase disk and install Ununtu` for `Installation type`
   (e) Select `Continue` and`Write changes to disks?`
   (f) Select the correct city, keyboard layout, and time zone.
4. Enter user information.
   (a) Enter your information, set the username as your login ID, you will need this for the screen shot. It should match your ID.
   (b) For now, choose `algonquin` as the password.
   (c) After the installation completes, select `Restart Now`.
   (d) The `.iso` file is treated as installation media, it may prompt you to *remove the media.*
   (e) Login with your new user name and password.
   (f) User `Ctrl-Alt-T` to open a new terminal.
   (g) Shutdown the system using `shutdown -h now`. This is an important step. Always shutdown the system gracefully, otherwise you will have a corrupted filesystem.
   (h) Change the root password using `sudo passwd root`. For now choose `algonquin`. In the workplace this password is always different.
5. Installing utilites.
   (a) Install the character based editor using `sudo apt install vim`
   (b) Install VMWare tools using `sudo apt install open-vm-tools-desktop`
6. Take a screen shot
   (a) `clear` will clear your screen.
   (b) Type in `uname -a`
   (c) Press the windows key and type in `screenshot`. Select `Area to Grab.` Select the section that shows the details of `uname -a`. This is the `.png` file you need to upload to Brightspace.
7. Cloning the VM
   (a) Select VM → Manage → Clone from the menu.
   (b) Select `Full Clone` from the dialog box.

2.22 Lab 2 - Basic Commands

Objective

1. Launch `man` pages and navigate.
2. Change Directory, return to home directory.
3. Create a new directory.
4. Create a file in a new directory.
5. Differentiate between absolute path and relative path.
6. Change directories.
7. Remove Directories in different scenarios.

Reference

1. Chapter on Basic Commands

Submission

Complete the online quiz, weightpage and due date as defined in LMS

Requirements

VMWare, Ubuntu last LTS version, default student account

Procedure

Tips on using the terminal. To discard a command at the prompt press Ctrl-C, you will get a new prompt, the command will remain on the line but will be discarded. Ctrl-C will also terminate a running a script or utility. The tab key fills the remaining letters of the directory after the initial unique letters are typed by the user. The `courier` font represents commands and syntax. Instructions are written in regular font.

Exercise 1. Launch `man` pages From the Ubuntu Desktop start a new terminal. One method to start a terminal is by pressing `Ctrl-Alt-T`. At the command prompt type in `man pwd` and press enter. `man` is a short for manual pages. *Note:* All commands should be followed by the enter key to run them. The Subsection `NAME` in the `man` pages describes the command. In the space provided write the description. You will need this information in the online quiz.
Navigating `man` pages. Press the `space bar` to forward one screen, press `b` to scroll back one screen, press `q` to exit `man` pages. Try scrolling up and down, then quit.
Notes:

Exercise 2. Change Directory. The cd command changes directories. Try the commands as a regular user, i.e. *not* as `root` user.
Type `cd /tmp`. Notice the change in the prompt. Type `pwd`, write the result.
Now type `cd` and press return. The `cd` command brings you back to your home directory. Type `pwd` to confirm.
Now type in `cd /tmp` once more. *Aside:* You can either type it in or use the up arrow key to scroll through the

previous commands. You are in the `/tmp` directory. Now type in `cd ~`. You will be taken back you your home directory. In this example, `cd` and `cd ~` gives the same result, we will see the difference at a later time.
Answer:

Exercise 3. `ls` command, and using the tab key. Check the man pages for `ls`, write its usage, note some useful options such as `-a`, `-l`, `-A`. `cd` to your home directory.
Type the `ls` command on the following directories.
`ls /bin` move to another directory, for example, `cd /etc`, and try `ls ~`. The contents of the home directory are displayed.
Type in `ls -al /ho` and then press tab. The command line should automatically fill the word `home`.
Answer:

Exercise 4. Create a directory and remove it. From your home directory type in `mkdir animal`. Change your directory to `animal`, notice the change in the prompt. Go up one directory. Remove the directory animal by typing `rmdir animal`. *Aside:* A directory, or file, can be created within the `animal` directory in the same way.

Exercise 5. Create a parent and child directory and remove it. From your home directory type in `mkdir plant/maple`. You should get an error. The parent directory plant does not exist. Write the error.

Repeat the command with the `-p` option, `mkdir -p plant/maple`. Verify that the directories are created. Now type in `rmdir plant/maple`. Note that only the child directory `maple` is deleted. Verify the result. You can recreate the child directory using `mkdir plant/maple`, this time the command will succeed, the parent directory exists.
Answer:

Exercise 6. Removing Parent and Child Directories. Create a parent and child directories as parent: `fungi`, child: `mushroom`. Write your answer below. Verify that the directories have been created. *Aside:* Both parent and child directories can be deleted using the command, `rmdir -p fungi/mushroom`

Answer:

Exercise 7. using `more`, `less` and pipe `|`. Type in `ls /etc`, the contents will scroll, before you can read it. Type in `ls /etc | less`. The screen will pause, the space bar will advance it by one screen, the `b` key will scroll back one screen, `q` will quit the command output. The `|` symbol is shared with the backslash key on your keyboard. A pipe allows output from one command as input to another command. In this example the output from `ls` is the input to the `less` command. In Linux, the `more` command is being used instead of the `less`, `more` does not have the reverse scroll option.
Comments and Observations:

Exercise 8. Using the `touch` utility. Create a directory and it's sub-directory as `strings/violin` using a single command. Use the touch command to create a file in the violin directory, using the following syntax.
```
touch strings/violin/strad
```

Check the result of the command using. `ls -l strings/violin`. You should see a file titled `strad` with zero bytes.

Notes:

Exercise 9. Removing a directory and its contents using the `rm` command. `rmdir` cannot remove a directory if it has files within it. Try the following command `rmdir -p strings/violin`. It will give an error. You need to remove the file first using the `rm` command, the syntax is `rm strings/violin/strad`. Now the directories can be removed using `rmdir`. Remove the directories `strings` and `violin` using a single `rmdir` command, assume there are no files in the directory `violin`. **Answer:**

Exercise 10. Create a directory using Relative Path and Absolute Path.
`mkdir -p /home/<user>/strings/cello` will create a directory using absolute path. *Note:* Do not type in `<user>`, replace `<user>` with your username. Absolute path will work under all conditions, you can be in any directory within the system to get a successful result from the command. The command will fail only of you do not have write permission to the directory. Absolute path has its advantages.
`mkdir -p strings/viola` is an example of relative path. It takes the current path into consideration. The command is run *relative* to its current path. If the command is run from the `/tmp` directory it will create the directories within the `/tmp` directory. `cd` to the `/tmp` directory and try the `mkdir` command in the `/tmp` directory.
Remove all directories you created in this exercise.

Answer:

Hybrid Exercise 1 - Basic Commands

Objective

1. Redirect output using >.
2. Differentiate between > and >>.
3. Copy files and directories using `cp`.
4. Rename files using `mv`.
5. Concatenate files using `cat`.
6. See a pictorial view of files and directories using `tree`
7. Redirect output using >.
8. Differentiate between > and >>.
9. Display common environment variables.
10. Prevent overwriting files using `noclobber`.
11. Customise the `bash` command prompt.
12. Word count using `wc`.
13. Using `tac` and `tail`, and their relevance in system administration.
14. Remove and merge sections of files using `cut` and `paste`.
15. List previously used commands with `history`, display contents of `HISTFILE` and `HISTSIZE`. Run previous commands using `!`
16. Display information about a command using `type`.
17. Locate a command using `which`.
18. Locate the binary, source and manual pages for a command using `whereis`.

Submission

Complete online quiz related to this exercise.

Background

Refer Chapter 2

Requirements

1. Ubuntu Installed.
2. Run all commands in the user mode. Do not run the commands with root privileges.
3. Run all commands from your default home directory `/home/<user>`, `user` is your account and group.
4. Commands are typed in the terminal.
5. Complete previous exercises and Labs before attempting these exercises.

Exercise 1. Use the `date` utility to display the date. Type date at the prompt, you should see an output similar to

```
Tue 21 Sep 2021 07:33:57 PM EDT
```

Date has several options, you are encouraged to browse through the `man` pages. Here are some parameters the `date` command accepts.

`date +%d` displays the day of the month, `%m` the month and `%Y` the year with the century.

Combine the above parameters to construct a customised date format.

```
date +%d-%m-%Y
```
What do the options and %B and %b display?

Exercise 2. Using redirection. The redirection operator sends the output to another device or file. Type the following command.
`date +%d-%m-%Y > today.` A file called `today` is created, it contains the output from the `date` command.

If a file called today already exists, the redirection operator will overwrite the existing file. Try displaying the date with a format of your choice and redirect it to the same file and confirm that its contents are overwritten. **Tip:** Try `%A` and `%a`.

Exercise 3. Double redirection operator. > > The operator >> also redirects the output to a file or device, it will not overwrite the existing file but will append it to the end of the file. If the files does not exist it will create a new file.
In this example we shall write the date in a different format and append it to the end of the file. Type in the following command:
```
echo 'Today is:'  $( date '+%A %d %B %Y' )
```
PS1and redirect the output to the previous file `today`, using >>
```
echo 'Today is:'  $( date '+%A %d %B %Y' )  >> today
```
Aside: The syntax `$( )` is explained in a later chapter, it is command substitution. You can deduce that the output from command `date` is substituted.
Check the contents of the file using the concatenate utility
```
cat today
```

Exercise 4. Redirecting Errors using 2> Errors can be redirected to a file or a device, such as a printer. Try to simulate an error with the `ls` command, for example `ls -zz`. We know `zz` is not an option in the `ls` command. Observe the results. Now redirect the error to a log file using the command.
```
ls -zz 2> error.log
```
Observe the contents of using `cat error.log` The 2> notation can be used in conjunction with the regular >. For example, `ls -l > files.lst 2>>error.log`.

Exercise 5. Copy files using `cp` This exercise uses `cp` in its basic form. Make a copy of the file `today` using the command
```
cp today today.bak
```
verify the result using `ls`.

Exercise 6. Copying directories and its contents using `cp` The option `-r` copies files and directories. First create a directory and then create five files within the directory.

```
mkdir instrument
cd instruments
touch erhu bagpipe guqin pipa yiqin
```

return to the home directory, `cd`

Copy the five files in the `instruments` directory to the `backup` directory using the command

```
cp -r instrument backup
```

Note: The backup directory is created if it does not exist.

Exercise 7. Rename files using `mv` Create a file using the touch command.

```
touch barytno
```

rename the file to its correct name using `mv barytno baryton`

Exercise 8. Move a file to another location using `mv` Move the file `baryton` to the `/tmp` directory using the command

```
mv baryton /tmp
```

verify the result using

```
ls -l /tmp/b*
```

Exercise 9. Getting familiar with `vim` Type in the following lines into a file. Call the file `province`.

```
1:Nunavut
2:Alberta
3:Saskatchewan
4:Manitoba
5:Ontario
6:Quebec
7:Yukon
```

Type in the following lines into a file called `capital`, use `vim`.

```
1:Iqaluit
2:Edmonton
3:Regina
4:Winnipeg
5:Toronto
6:Quebec City
7:Whitehorse
```

Exercise 10. Using `cut` and `paste` Try the following commands and observe the results.

```
cut -d: -f2 province
cut -d: -f1 province
```

Try

```
paste province capital
```

Exercise 11. Verify `PS1` and `PS2`, change `PS1` Verify the environment variable `PS1` by typing `echo $PS1`. Type `echo $PS2`

Change the prompt using `PS1="\u@\h:\w$ "`

For this exercise, you can `exit` the terminal. A new terminal will give you the original prompt. In a later exercise you will be able to restore your original prompt on the same terminal.

3

File Permission

This chapter discusses the file permissions, changing file permissions, managing default file permissions. The Unix file system has a unique feature each object can have two links; this topics and its uses is discussed.

3.1 The Unix File System

Every object in the Linux System is a file, with two exceptions. Even devices are represented as files, this is evident by observing the `/dev` directory. Your terminal is represented as a file. See the contents of the `/dev/pts` directory.

The Linux file structure is hierarchical. All files and directories are associated under the root directory represented as `/`. Files are organized in directories. Directories are in reality *files*. Many cryptic messages inherent in Unix are still present. Try out this `mkdir` command using, `mkdir a a` and observe the (cryptic) error message which says.

```
mkdir:  cannot create directory 'a':  File exists.
```

The last two words is trying to tell you that, in Unix, every object is a file. Remove the directory using `rmdir a`.

User's home directory Each user has a home directory. All files for the user is stored in the users home directory. A users home directory is represented as `~`, the tilde symbol. `cd ~` or simply `cd` will take you to your home directory. By default your home directory is directly under the root directory, it is called `/home/username`, where `username` is your login name. The `~` replaces `/home/username`. If you want to run a script which is in your home directory, while you are in another directory, you could run it in two ways.

```
/home/username/.my_script.sh
~/.my_script.sh
```

Other Directories A user can write to a few other directories such as `/tmp`. An administrator may not want to see certain output appear on the screen, this could be redirected to `/dev/null`. Loosely speaking `/dev/null` is referred as a black hole, any redirection to `/dev/null` will disappear. **Exercise:** Determine the read and write permission of `/dev/null`. Determine the filetype of `/dev/null`; is it character or block device.

Filename Theoretically a filename can be 255 characters in length. Two files with the same name cannot reside in the same directory, and you cannot have a file and a directory with the same name. Avoid using a space in a filename, if you have accidently used a space in a filename you will have to use double quotes to access it.

Directory A directory is a file that contains a list of files and other directories.

`root` **user** Traditionally, a root user has *all* the system privileges. This level of permission is undesirable. A root user who has all the privileges should not reading other users' email. Modern installations have several levels of administrative roles such as *backup*, *restore*, system shutdown and restart among others. A root user in modern installations will have specific privileges usually associated with administrative tasks.

File Permissions Consider the partial output from `ls -l` shown in figure 3.1

```
drwxr-xr-x 2 wei01  Mktg   4096  Jan 24 12:10 Music
drwxrwxr-x 2 wei01  Mktg   4096  Jan 25 13:30 Pictures
-rw-rw-r-- 2 wei01  Mktg   1565  Nov 03 14:23 Budget
-rw-rw-r-- 1 wei01  Mktg   3244  Nov 03 14:23 Schedule
```

Figure 3.1: Partial File Listing

The first character in the first group indicates the type of file.
`-` indicates a regular file.
`d` indicates a directory.
There are several other file types some of them are discussed later, they are `c`, `b`, `l` and `s`.
The next group of 9 characters `rwxrwxr-x` indicate the file permissions. The first three indicate the permission of the user. The user has `rwx` permission in the `Music` directory.
`r` implies read permission to the directory.
`w` implies write permission to the directory and
`x` implies execute permission to the directory.
The next three characters in the directory listing indicate the same read-write-execute permission for the group and the last three group of characters indicate the permission for the `other` users in the system; `other` users include temporary users and guest users.

File Permissions

The three types of file permission that relate to files and directories are listed in table 3.1.

read (r) File - can view file content (with application). Dir - can view directory content.
write (w) File - can modify file content. Dir - can add & remove (delete) files and directories. To add and remove files from a directory, it must have Write and Execute permissions.
execute (x) File - can be executed. Note: Binary files require only the execute permission; script files require both read and execute permissions. Dir - change into a directory with the `cd` command.

Table 3.1: File Permissions

3.2 Change mode, Assign permission

The owner of a file can change its permission using the `chmod` command. By changing permission of a file the owner determines its access to other users, groups or even to the owner herself. The `chmod` utility is used to `change mode`.
Note:

- The term *file* is used in a broader context, it includes directories, which are files, and devices. Physical devices connected to the computer are represented as files in `/dev` directory.
- A superuser, i.e., root can access all files on the system, and can change permissions and modes for all files in the system.

FileType	*Permission*	*Links*	*Owner*	*Group*	*Size*	*Date/Time*	*Filename*
d	rwxrwxr-x	2	wei01	Mktg	4096	Jan 25 13:30	Pictures
-	rw-rw-r-	2	wei01	Mktg	1565	Nov 03 14:23	Budget

Table 3.2: Directory Listing Description

Figure 3.2 explains the description of each item in a long listing. The size of a directory will initially be 4096 bytes, in large directories that hold many files the size will be more than 4096 bytes.

The time displayed is the time the file was last *modified*. This time is different from the time the file was last *accessed*. To display the time the file was accessed use `ls -ul`.

Exercise: Create a file using touch. Check the timestamp using `ls -l`. Wait a few minutes, open the file in `vim`, do not change the contents, simply open and close the file, i.e., quit `vim` without saving the file. Now check the listing using `ls -ul` and `ls -l`, compare the two results.

3.2.1 Changing Permission - Symbolic Mode

File permission can be modified with the `chmod`, change mode command. `chmod` can be used in two ways (i) Octal Mode and (ii) Symbolic Mode :
Syntax for symbolic mode is `chmod [ugoa][+-=][rwx] [object]`
Who
u user/owner
g group
o other
a all (ugo=all)
Operator
+ set
= set explicitly and remove all other
- remove
Permission
r read
w write
x execute

Other file types Some of the other file types besides dash - and d are:

c character device file.
b block device file. Block and character device files are located in the `/dev` directory. Run `ls -l /dev` and see how many files you can identify as a device.
l Symbolic Link, also called soft link.
p A named pipe.
`s` A socket
a few others, such as S

Users Users are grouped into

`Owner`, creator of the file
`Group`, a group of users. A group user can own a file and can assign access permission.
`Others`, all other users who are usually with a *guest* account. They do not belong to a group and are assigned a login and password to perform minimal tasks on the system.

`rwx` File permissions are represented by 9 characters, in groups of three. The first three `rwx` or `rw-` represent permission for the owner, the next three for the group and the last three represent permission for all other other users on the system. Internally there are 12 bits are used to determine file access privilege, only 9 are visible.

Execute Permission for files Executable permission for a *regular* file has no significance; for example, files such as a word processing file, spread sheet, `.mp3`, or an image file.
For a shell script to be executable it must have a minimum `read` and `execute` permissions. A *shell script* is a set of commands and programming structures that are written in a file that can be run in a Linux shell.

Changing file permission File permissions are changes using the `chmod` utility. A user must be the owner of the file or belong to a group that has ownership of the file. There are two ways to change the file permissions using this utility.

using absolute arguments

using symbolic arguments

With three positions there are eight possibilities of file permissions, i.e. $2^3 = 8$.

r	w	x	Octal Value	Permission
0	0	0	0	No permission
0	0	1	1	Execute only
0	1	0	2	Write only
0	1	1	3	Write and Execute
1	0	0	4	Read only
1	0	1	5	Read and Execute
1	1	0	6	Read and Write
1	1	1	7	Read, Write and Execute

Table 3.3: File Permission

Exercise: You will create two terminals, redirect the output from one terminal and observe it on the second terminal. Find out your terminal's file by typing `ps`. Below the `TTY` column, you should see `pts/0`, `pts/1` or a similar terminal type. Open two terminals and determine the assignment for each. In `pts/0` type in `ls -l` the output should display on the screen. Next from the same terminal, i.e., `pts/0` type in `ls -l > /dev/pts/1`. What do you observe? Go to the second terminal i.e., `/dev/pts/1` and type in `ls -l > /dev/pts/0`. The redirection symbool > sends the output to a file which is a representation of a terminal.

3.3 Minimum File Permission

To `ls` to a directory the minimum permission required is `read` and `write`. For `cd` to work on a directory the minimum permission required is `read` and `execute`.

Without `read` and `execute` permission on a directory

- a user cannot `cd` to a directory.
- a user can `ls` to the directory and get a partial file listing.
- The command `ls -l` will not show the permission of the file nor its size and ownership.
- The `find` utility will list the file.
- The `tree` utility will not show the directory.
- Ubuntu shows the directory with a different color scheme.

3.4 `umask`

A mask can be defined as *Selective permeability.* A mask in computing is a pattern of bits, hence it is called a bitmask.
When a file or directory is created it has default permissions. These permissions are determined by `umask`. When a user logs in a default umask is set, for example in Ubuntu it is `0002`. When a user creates files and

directories, this `umask` is used to set default permissions. For files the permissions are `-rw-rw-r--` and for directories `drwxrwxr-x`. The `umask` cYohannan be changed by the user during a login session. If it is changes, all *subsequent* files will inherit the new permissions determined by the new `umask`. To change the permission of *existing* files and directories the user needs to use `chmod`.

To display the `umask` in symbolic mode type
`umask -S`
the display will be `u=rwx,g=rwx,o=rx`, it is the default `umask` in Ubuntu, it translates to `002`.
`umask` without arguments will display the mask in octal. The octal mode gives 4 digits, the least significant 3 digits are used to determine file permission. For example, for a `umask` of `0002` use `002`.
A 0 value indicates that the resultant bit is permitted, i.e., a 1 and a 1 in the mask indicates that the resultant bit should be blocked. The resultant bit is obtained by using the one's complement.
For example, for a `umask` of `002` octal, the binary value is `000000010`. Take the ones complement of this `umask`.
Aside: One's complement is the XOR operation with 1's.

```
111 111 111
000 000 010 umask
-----------
111 111 101 result
```

Translate this result into file permissions to get `rwx rwx r-x`.
Unix removes the execute permission for files. Stated differently, execute permissions for a file cannot be assigned using `umask`, they must be assigned explicitly by the user.
For the above example, the permission for a file would be `rw- rw- r--`.
If the object is a directory, the execute permissions are not taken away, they remain.
`umask` can be changed by `umask <octal value>`.
For example, in octal mode `umask 033` will set the `umask` value to `033`.
`umask` can be used in symbolic mode, it has three symbols.

= to set the permission.
+ to add a permission if it does not exist.
- to remove a permission if it exists.

For example, `umask u+x` will add the execute permission for files that are created.
`umask g=rw` will set the group's permission to read and write.

There are two methods to set and display `umask`, octal and symbolic. To set `umask` in octal mode the user has to specify the *entire* pattern. To set the `umask` in symbolic mode it is possible to incrementally assign or remove permission from the existing permission.
For example, if the default `umask` is `002` the group has read and write permisson to a file. To remove the write permission from the group the user has to use `umask 133`. To achieve the same result using symbolic mode, the command is `umask g-w`; specifying only a subset of the permission leaves the other permissions intact.

Exercise 1: Remove write permission to the user using symbolic mode. First confirm the existing `umask -S`. Create a file. Does the file have write permission for the user? If yes, then type `umask u-w`, to remove it. Notice only the user's permission has changed without affecting group and others.

Exercise 2: Type in `umask a=rwx` and answer the following questions and verify the results.

1. What is the octal value of the `umask`?
2. How will you remove the write and execute permission from `others` using `umask` in symbolic mode?
3. What is the value of the new `umask` in octal mode and in symbolic mode?

4. Create a directory with the new `umask`.
5. Does the execute permissions remain for `others` for this new directory you created?

Exercise 3: A `umask` of `022` and a `umask` of `133` will give the same permissions for a file but different permissions for a directory. Explain.

3.5 Review Questions

3.5.1 Multiple Choice Questions

1. Identify the account that has the most access to system resources.
   A. `root`
   B. Manager
   C. regular user

2. Identify the two choices that describes a directory in Unix.
   A. The current directory is represented as a single dot **.**
   B. A directory is a file that contains a list of files and other directories
   C. The current directory is represented as two dots **..**

3. To run a shell script a user must have a *minimum* of
   A. `execute` permission
   B. `read` and `execute` permission
   C. `write` and `execute` permission

4. To run a binary file a user must have a *minimum* of
   A. `execute` permission
   B. `read` and `execute` permission
   C. `write` and `execute` permission

5. Identify the *minimum* permission required, for a directory, to delete a file from that directory?
   A. read
   B. read and write
   C. read, write and execute
   D. read and execute
   E. write and execute

6. User `PrjMgr` copies the file `Plan.Q01` to directory
   `/archive/Mgt/Q01`
   using the following command:
   `cp␣/home/PrjMgr/Plan.Q01␣/archive/Mgt/Q01/`
   What are the *minimum* permissions required for the directories `PrjMgr` and `Mgt`
   A. source directory `r`, `w` and `x`, for target directory `r` and `w`
   B. source directory `r` and `w`, for target directory `r` and `w`
   C. source directory `x`, for target directory `r` and `x`
   D. source directory `x`, for target directory `w` and `x`
   E. source directory `w`, for target directory `w` and `x`

7. To delete a file a user must have write and execute permission in the directory, but does not need any permission for the file itself.
   A. True
   B. False

8. To move a file a user must have appropriate permission in the source and target directories, but does not need any permission for the file itself.
   A. True
   B. False

9. `user1` moves the file `budget.2015` using the following command.
   `mv /home/user1/budget.2015 /home/accounts/`
   The *minimum* file permission required for source and target directories are
   A. `read` permission for `/home/user1` directory and `/home/accounts` directory
   B. `write` permission for `/home/user1` directory and `/home/accounts` directory
   C. `read` and `write` permission for `/home/user1` directory and `/home/accounts` directory
   D. `read` permission for `/home/user1` directory and `write` permission for `/home/accounts` directory
   E. `write` and `execute` permission for both `/home/user1` and `/home/accounts` directory

10. What is the *minimum* permission required for a file to be copied?
    A. `read` and `write`
    B. `read` and `execute`
    C. `write` and `execute`
    D. `read`
    E. `write`

11. What is the permission, in octal mode, of the file `MySQL.log`, given the file listing:
    ```
    -rwxrw--w-   2   maotse   students   15  Jan 25 12:37 MySQL.log
    ```
    A. `652`
    B. `651`
    C. `762`
    D. `732`

12. Given the file permission `-rw-rw-r--` for a file `report.draft` in current directory, identify the `chmod` command that adds execute permissions for owner and removes read permission from others. Permission for group should remain unchanged.
    A. `chmod 740 report.draft`
    B. `chmod 750 report.draft`
    C. `chmod 760 report.draft`
    D. `chmod 770 report.draft`

13. Identify the Linux command used by `root` to give everyone full access permission to directory `/public`. [Select all that apply]
    A. `chmod 777 /public`
    B. `chmod ugo=rwx /public`
    C. `chmod a+rwx /public`
    D. `chmod ugo+rwx /public`

The user has set the `umask` to `022`. Answer the next two questions based on this value of `umask`.

14. A file is created. What will be the file's permission?
    A. `----rw-rw-`
    B. `-rw-r--r--`
    C. `-rw-rw-rw-`
    D. `----r--r--`
    E. `-rwxr-xr-x`

15. A directory is created. What will be the directory's permission?
    - A. `d---rw-rw-`
    - B. `drw-r--r--`
    - C. `drw-rw-rw-`
    - D. `drwxr-xr-x`
    - E. `d----w--w-`

16. What `umask` should be set so the resultant file permission is `-r--r--r--`
    - A. `044`
    - B. `033`
    - C. `022`
    - D. `011`
    - E. `333`

17. Identify the correct `umask` to set the resultant file permission to `-r--r--r--`. Select two answers.
    - A. `umask a-r`
    - B. `umask a=r`
    - C. `umask ugo=r`
    - D. `umask u-r`

18. The `root` user wants to change the owner to `cfo` and group to `ExecCmt` for file `Strategy.Q03`. Identify the command that will achieve the result.
    - A. `chown cfo root Strategy.Q03`
    - B. `chown Strategy.Q03 cfo root`
    - C. `chgrp Strategy.Q03 cfo ExecCmt`
    - D. `chgrp cfo.ExecCmt Strategy.Q03`
    - E. `chown cfo.ExecCmt Strategy.Q03`

19. Select one correct statement. What is the outcome of changing the `umask`
    - A. `umask` will affect file permissions for all files that are created after the `umask` has changed, the files that were already created by the same user will also automatically change.
    - B. `umask` will change file permissions for all files that are created after the `umask` has changed, the files that were already created must be changed using `chmod`.

20. `fibbonaci.sh` is a script which is user runs from the command prompt as `$:fibbonaci.sh` where `$:` is the prompt. Select one correct answer with reference to `$PATH` and the method of execution of the script.
    - A. Linux will search for the file in the `$PATH` variable, after the search is exhausted Linux will flag an error. If the file is found in the path, it will execute.
    - B. Linux like MSDOS will first search for the file in the current directory, if the file exists, it will run.

21. Select one correct statement with reference to text files in DOS and Linux.
    - A. DOS terminates each line in a text file with a line feed (LF).
    - B. Linux terminates each line in a text file with a line feed (LF) and carriage return (CR).
    - C. Linux terminates each line in a text file with a line feed (LF).

22. Identify the echo statement that will display the current PID.
    - A. `echo $?`
    - B. `echo $$`
    - C. `echo $#`
    - D. `echo $?`

23. Given the assignment
    `plant=orchid`,
    identify the output from `echo $plant`
    A. `orchid`
    B. `plant`
    C. `$plant`

24. Given the assignment
    `plant=orchid`,
    identify the output from `echo '$plant'`
    A. `'$plant'`
    B. `plant`
    C. `orchid`

3.6 Reading Assignment

Reference: Operating System Concepts, Silberschatz et al., 9e.

- Introduction: Pages 3 and 4
- Multiprocessor Systems: Page 14.
- Operating System Structure: Section 1.5, Page 19 to 21.
- Summary: Pages 47 to 49.
- Questions: Answer questions 1.1, 1.2 and 1.5 on page 49 and 50.
- Research Question: Read question 1.27 page 52 and attempt to answer it.
- Virtual Memory: Pages 397 to 398.

Lab 3 - File Permissions

Objective

1. Verify read, write and execute permissions for file.
2. Verify read, write and execute permissions for directories.
3. Change `umask` and verify its effect of default permission.
4. Calculate default permissions for a given `umask`.
5. Determine an appropriate `umask` for a desired level of access control.

Reference

1. Chapter 3

Submission

Complete the online quiz on file permission.

Background

The file names used in this exercise such as `vanda`, `cattleya`, `laelia`, `oncidium`, `lycaste` and `dendrobium` are orchid genera.
The emphasis is on minimum permission required to perform a task. Given full read, write and execute permission a user can perform any tasks, the objective is to give the *minimum* permission.

Requirements

A users account on Ubuntu.

Procedure

Run all commands in user mode, do not use the root privileges unless explicitly specified in the exercise. Using a root account without a real need will limit your understanding of the concepts. If you get a permission denial error try to resolve the issue, instead to running the command with root privileges.
This lab has short exercises that grant and revoke permissions to your own files and directories. In a production system another user, such as an administrator will perform these tasks on files and directories that are used by other users and groups. You are assuming a role of an administrator.

Exercise 1. Determine the `umask` for the account. Type in `umask`, write the 4 digit octal number. Write the number in the space below. Discard the leading octal digit, you now have three octal digits. This will be the subtrahend. Subtract this number from `777`, the minuend. You will be using octal arithmetic. Convert the difference from octal to binary. Put them into groups of three. Associate each group into `rwx rwx rwx`. Remove the execute permission. This will be your default file permission.

Exercise 2. Confirm the default file permission using `touch`. In your home directory create a file called `phal` using `touch phal`. Verify the file's permission using `ls -l phal`. Does the permission match the expected permission from the arithmetic you performed in the previous exercise?

Exercise 3. Perform a write operation and confirm the files permission. You can add some sample data to this file. You may use `vim` or redirect a command to `phal`.
For example, `date > phal`
Confirm that you have the read permission by typing `cat phal`, it should display the files contents.

Exercise 4. Remove the write permission for the user Remove the write permission from the file `phal`. Use the `chmod` command. Such as `chmod 464 phal` in absolute mode or `chmod u-w phal` in symbolic mode.
Verify the change in permission using `ls -l phal`.
Add data to `phal` using a double redirection operator. `date` >> `phal`
Write down the error in the space provided. Verify that the earlier data is still readable, use `cat phal` to verify.

Put the write permission back using `chmod 664 phal` in absolute mode or `chmod u+w phal`.

Confirm that the write permission is restored and that you can now write to the file.
Aside: `vim` can override the write permission as we will see later. You can prevent an accidental overwrite in `vim` using `noclobber`.

Exercise 5. Remove the read permission, without removing the write permission. Confirm that you have restored the files permission to `rw-rw-r--`. Remove the write permission using chmod in absolute mode `chmod 264 phal` or use symbolic mode `chmod u-r phal`.

You will be able to write to the file but not read from it, even though you are the owner of the file. Type in `cat phal`. Write the error message in the space below.
You should be able to still write to the file using `date > > phal`.
Think of a use of such a file permission, not readable but only writable.

Restore the read file permission using `chmod 664 phal` in absolute mode or `chmod u+r phal`. Use `ls -l phal` to confirm. Now use `cat phal` to verify that the last input has been stored in the file.

Exercise 6. Directory permission - Read, no write The default permission for a directory with `umask` as `0002` is `rwxrwxr-x`. Create a directory called `orchid`. Verify the default file permission. Create a file in the `orchid` directory, call it `oncidium`. With the default permission, a file will be created with no errors.
Remove the write permission leave the read permission for `orchid`, use `chmod`. Now attempt to create a new file within `orchid` called it `laelia`. With the write permission removed, `bash` will give an error. Write the error message.
Put the write permission back, and remove the read permission. Use `chmod u+w,u-r orchid`. Create a file call `cattleya` in the `orchid` directory, use `touch`. `bash` will create a file in the directory because you have write permission but will not let you view the contents of the directory because you do not have read permission. You will be able to change to the directory using `cd`; you have execute permission. `tree orchid` will also not allow you to view the contents of the directory.

Exercise 7. Use of a directory with only write and execute permissions. Think of two real life examples of having a directory with read permission removed. Write your answer in the space below.

Exercise 8. Minimum permission to delete a file from a directory. Create a directory, call it `orchid`, create a file within this directory, call it `vanda`, use `touch vanda`. Confirm the file's existence using `tree orchid`. Change the permission of `orchid` to 355, i.e. remove the read permission, but leave the write and execute permissions for the owner. Use `tree orchid`, you cannot see the contents of the directory, the read permission has been removed.
Delete the file `vanda` from the directory using `rm orchid/vanda`. Put the read permission back using `chmod 755 orchid`, confirm that the file `vanda` has been removed.
Repeat this exercise by removing either the write or execute permission, and observe the error messages.

Exercise 9. Copying files to a directory and restricting access. To copy a file from one directory to another the minimum permission required are execute permission to the source directory, write and execute permission in the target directory. With this *minimum* permission in the source directory, a user must also know the filename in the source directory.

1. Create two directories `PrjMgr` for project manager and `archive/Mgt/Q01` for archiving the files. `PrjMgr` will be source directory and `archive/Mgt/Q01` will be the target directory.
2. Create a file in the directory `PrjMgr`, call it `Plan.Q01`, use touch.
3. Remove read and write permission from `PrjMgr` directory, and remove read permission from the target directory `archive/Mgt/Q01`.
4. Copy the file `Plan.Q01` from the source to target. To verify the operation you will need to restore the read permission to the target directory. Did the operating succeed?
5. Repeat the exercise by removing the execute permission from the source directory. You may copy this file to the current directory, since you are testing the permission of the source directory.
6. Are you able to `cd` to the directory without the execute permission?
7. Restore the permission of the source directory to `165`. Change the target directory's permission to read and write only. Are you able to perform the copy operation?
8. Write your observations in the space provided.

Exercise 10. Minimum Permissions. Circle the minimum permission requried to complete the actions.

1. To copy a file a user requires:
   (a) for source directory: `r w x`

   (b) for target directory: `r w x`
   (c) for the file: `r w x`

2. To move a file a user requires:
   (a) for source directory: `r w x`
   (b) for target directory: `r w x`
   (c) for the file: `r w x`
3. To delete a file a user requires:
   (a) for directory: `r w x`
   (b) for the file: `r w x`

4

File System

This chapter discusses the Linux File Systems, it includes using utilities that prepare the hardware for a filesystem, attaching a filesystem to a working Unix computer. It covers system maintenance utilities required to maintain a filesystem.

4.1 Reading Assignment

A Practical Guide to Ubuntu Linux, 4e, Mark G Sobell, Prentice Hall,
ISBN: 978-0-13-392731-3

Chapter 6 - The Linux Filesystem

1. Page 204-205, SetUID and SetGID Permissions
2. Page 1276, Sticky bit in glossary
3. Page 598-599, (bottom of page) Setuid file

4.2 Filesystem, an Overview

A filesystem is a data structure that is used to organize data, i.e., store and retrieve data, on a storage medium. Data is stored as files, files are organized within directories. A directory is a file that lists files and directories within it. A data structure by itself is static, a filesystem is a dynamic collection of data, therefore a filesystem when attached to a Linux computer is a *method* and a *dynamic data structure* that is used by a computer to organize data.

A filesystem is hierarchical, the root directory represented as / is the topmost directory in the hierarchy. All other filesystems are branches from this root directory. Usually, the root filesystem has its own partition. A partition is a section of a storage medium. Large storage devices are partitioned for easier management. Filesystems can be local or remote; a remote filesystem is accessed by the network. A remote filesystem has a protocol that is used to access the files. Examples for remote filesystems are `nfs`, `cifs`.

Aside: There are two utilities, among many, that will display the filesystems in Linux.
`df -kh` and `mount -lt ext4`.

Try these commands on your system. Details are discussed in the later sections in this chapter.

A Linux server usually has more than one filesystem and more than one *type* of filesytem. The storage medium determines the type of filesystem, for example, `iso9660` is used for optical devices, an `ext` filesystem is commonly used for magnetic media and solid state drives (SSD). A single filesystem can spread across several physical storage media, e.g., hard disk. A single storage medium can have more than one filesystem.

A file is a stream of bytes. Linux maps external hardware devices as files. A network card, hard disk drive, printer, mouse and keyboard are represented as files. These device files are stored in the `/dev` directory.

Windows and Linux A large unix computer will have several hundred user accounts. Filesystems get attached to the computer on a need basis. Attaching filesystems on a need basis improves the performance and enhances the security of the system. The utility to attach a filesystem in Linux is `mount`. In Windows all filesystems are attached to the server during bootup. Filesystems in Windows are represented as `C:`, `D:`.

There are four utilities that are commonly used when preparing a storage medium and the filesystem for use. All four must be run with root user privileges. `fdisk` is a utility that adds and removes disk partitions. `fdisk` can do several other disk organization tasks. After the disk is prepared, a file system needs to be created, `mkfs` is the utility to do this. Section 4.4.1 shows some examples to use `mkfs`. A magnetic hard disk drive usually has a few sectors that are not usable, these are marked and data is not stored in these sectors. `fsck` checks and repairs the filesystem, i.e., marks the bad sectors or even move the data stored on bad sectors to good sectors. Finally, the filesystem is attached to the existing computer using the `mount` utility. The corresponding utility to `mount` is `umount`; it detaches the filesystem from the computer. The following sections provide examples for these utilities.

4.3 Examples of Filesystems

Common filesystems are listed with their advantages and limitations. Older filsystems are rarely deployed for new applications, a practitioner will need to be aware of these filesystems. Older storage media will still these older filesystems with data stored in them.

4.3.1 General Parallel File System (GPFS)

[8, Pg 46] is a high performance POSIX compliant shared-disk clustered filesystem that runs on a Storage Area Network (SAN). GPFS is used by supercomputers, DB2 purescale and Oracle Real Application Clusters (RAC) implementations. GPFS provides concurrent high speed file access using data striping.

4.4 Procedure to Create and Use a Filesystem

`fdisk` operations are often non-reversible. Before performing any tasks on a filesystem, you should make a clone of the current OS in the virtual machine.

Cloning a Virtual Machine (VM)

A clone is a backup of a functional VM. Perform all partition operations on a clone. The following steps list the process of cloning a VM in VMWare.

1. Shutdown the existing VM. A Clone cannot be created when a guest OS is running. Use the command `shutdown -h now`
2. In VMWare select `VM -> Manage -> Clone`. Read the message on the first window. Click Next.
3. Choose the first radio button. "The Current state in the virtual machine". Go to the next screen
4. Select the second radio button. Choose "Create a full clone". Select Next
5. Give a name to the Virtual Machine. Write the Location of the virtual machine and its date in the space below.
6. Click Finish. The process will take 2-5 minutes.

Adding a Virtual Disk

Virtual Disks are represented as files on the host computer. This procedure will create a filesystem called `/dev/sdb`. The default filesystem on a Linux Computer is `/dev/sda`. Dot use `/dev/sda` for any activity in these exercises. Linux will not boot from a damaged `sda`.

1. Select VM -> Settings, At the bottom of the screen select "Add..."
2. Select Hard Disk, then Next
3. Select SCSI
4. Select the Radio Button, "Create a new virtual disk".
5. Change the "Maximum disk size" to 2.0GB
6. Select the check box "Allocate all disk space now"
7. Choose "Split virtual disk into multiple files"
8. Read the short comment. There are other details in selecting split virtual disks.
9. Choose the file name. Write the name in your workbook with the date of creation.
10. Note the extension for virtual disks is `.vmdk`
11. Select Finish.

4.4.1 Building a Filesystem with `mkfs`

`mkfs` is short for *make file system*. Linux man pages call it *build a Linux filesystem*. Build implies formatting the partition to a required filesystem. `mkfs` is located in `/sbin`
The syntax is: `mkfs [options] [-t type] [fs-options] device [size]`
For example:

To format `/dev/sdb1` as `ext4` filesystem you would use the following command as root.

```
mkfs -t ext4 /dev/sdb1
```

to run `mkfs` on a single use basis

```
sudo mkfs -t ext4 /dev/sdb1
```

`mkfs` returns an exit code as `0` for success and `1` for failure.
Immediately after mkfs completes type in `echo $?`, a `0` indicates success. Errors if any are displayed on the screen. You would check for success or failure if running `mkfs` as part of a shell script.

`mkfs` has been deprecated, it is replaced by a filespecific utility `mkfs.<filetype>`. For example, `mkfs.ext4`.
`mkfs -t ext4 /dev/sdb5` is equivalent to `mkfs.ext4 /dev/sdb5`
Observe the output from the utility. It tells you the number of block created, the size of each block and the number of inodes available. It also creates a superblock and filesystem accounting information, ext4 is a journaling filesystem.

> A superblock is a record of the characteristics of a filesystem, including its size, the block size, the empty and the filled blocks and their respective counts, the size and location of the inode tables, the disk block map and usage information, and the size of the block groups. [6]

Verify the FileSystem

The `fsck` utility performs a check on the filesystem. It returns an errorcode, the errorcode is the *sum* of the errors. A list of errors are [7]:

```
0      No errors
1      Filesystem errors corrected
2      System should be rebooted
4      Filesystem errors left uncorrected
8      Operational error
16     Usage or syntax error
32     Checking canceled by user request
128    Shared-library error
```

`e2fsck` is used for the `ext` family of filesystems. It checks `ext2, ext3` and `ext4` filesystems.

Exercise

Simulate an error by typing `sudo fsck /dev/sdx5`, and then checking the errocode by `echo $?`.

Mounting a Filesystem

Logically attaching a filesystem to a Linux computer is called mounting a filesystem, the `mount` utility performs this task. We say logically attach; although the hardware may be powered and physically attached to the computer either internally using a bus or port, or externally using cables, it cannot be used unless it is mounted.

Usually filesystems are mounted at a mount point in `/mnt`, there are exceptions. Inserting a USB drive will automatically mount the device at `/media/<username>` The mount point is a directory.

First create a directory within `/mnt`, for example `mkdir /mnt/folkMusic`.

To mount `/dev/sdb5` use the command:

`mount /dev/sdb5 /mnt/folkMusic`

The filesystem is now attached and visible.

Detach the filesystem using
`sudo umount /mnt/folkMusic`. Detaching a filesystem will not delete the mount point. Usually the mount point is kept in the directory by the system administrator. A filesystem can be verified either way, before mounting it and after mounting it. If a hardware error is suspected, it is usually taken offline and checked.

`umount` is documented in The Linux Information Project at http://www.linfo.org/umount.html

Mounting can be automated during the system bootup process, or as a separate procedure especially if computer has a large number of filesystems. Information about a filesystems is stored in a file at `/etc/fstab`, in the form of a table. This file is read by other programs and utilities and maintained by the system administrator. The mount command references this table at the time of mounting filesystems.

Each row in `/etc/fstab` has six fields and it represents one filesystem, Example of an entry is

`/dev/sdb1 /media/blank auto rw,user,noauto,exec,utf8 0 0`
`/dev/sdb1` - the filesystem to be mounted.
`/mnt/folkMusic` - the mount point.
`auto` - indicates that the filesystem should be mounted automatically. `noauto` will need the administrator to mount the filesystem.
`rw,user,noauto,exec,utf8` - mount options, `rw`, for read and write. `noauto` will not mount the filesystem when `mount -a` option is given. A filesystem can be assigned to an owner, and only the owner can be be permitted to mount the filesystem. All options separated by a comma are read as a single field.
`0` - The fifth fields indicate the backup procedure that is applicable to the filesystem.
`0` - The last field determines the order `fsck` is performed.
Another example of an entry is
/dev/sdb1 /media/blank auto defaults 0 0
There are no commands in `/etc/fstab`. Do not insert mount commands in `/etc/fstab`, it is a file system lookup table.

Exercise

Insert a USB thumb drive. Observe the mount point it will mount at `/media`, not `/mnt`

4.5 Mounting a CD ROM Drive

This procedure lists the steps to mount a device at bootup, in this example a CD ROM drive.

Prerequisites:

1. If you are performing this exercise for the first time, make sure you are familiar with the process to mount the drive manually.
2. A Ubuntu installation
3. Access to super user account.
4. Use of `blkid`, `grep` to identify the `UUID`
5. Use of vim
6. Use of mount and mount -a

Background: Universal Unique Identifier (UUID) is a 128 bit number that uniquely identifies an object on the Internet. Each `ext2`, `ext3` or `sr0` device has this number. It is represented in hexadecimal. Ref: Pg 1280, Sobell 4e.

Procedure: Take a full clone of your Ubuntu installation using VMWare; do not proceed without a clone. If the system does not boot up, do not panic, you will have to use the clone. There are other recovery methods.

1. Ensure that a mount directory is created in `/mnt`. Use `mkdir /mnt/cdrom`
2. Identify the UUID of the device, using `blkid` and `grep` as shown: `blkid | grep iso`

   `blkid` is a utility that lists information on block devices, it is run with root privileges, `grep` will list only those rows that matches the string `iso`.
   A sample output from `blkid` is shown below:
   ```
   /dev/sr0: UUID="2019-07-31-16-51-12-00" LABEL="Ubuntu 20.04.1 LTS amd64"
   TYPE="iso9660" PTUUID="56E48570" PTTYPE="dos"
   ```
   Take note of the UUID. Note: PTUUID is a different ID.
3. This UUID needs to be added to the `/etc/fstab` file with other details. `/etc/fstab` is the file that is accessed during startup. There are several ways to do this. First take a backup of `/etc/fstab` by using `sudo cp /etc/fstab /etc/fstab.bak`
4. Using vim add the following line at the end of `/etc/fstab`, replace the `UUID` with the `UUID` for ISO on your computer.
   ```
   UUID=2019-07-31-16-51-12-00 /mnt/cdrom auto user 0 0
   ```
   Alternatively, you can add the line using `echo` as in

   ```
   echo "UUID=2019-07-31-16-51-12-00 /mnt/cdrom auto user 0 0" >> /etc/fstab
   ```

   Mount options are explained below:
5. Before restarting the system, use `mount -a` to verify. Use root privileges.

   Aside: If the mount directory does not exist, the error will be.
   ```
   mount: /mnt/cdrom: mount point does not exist.
   ```
   Create the mount directory as indicated in step 1. If the UUID is typed incorrectly the error will be.
   ```
   mount: /mnt/cdrom: can’t find UUID=2018-07-31-16-51-12-00.
   ```
   A warning such as:
   ```
   mount: /mnt/cdrom: WARNING: device write-protected, mounted read-only
   ```
   is acceptable, it is telling you that the device is read only.

   Do not proceed to restart until you have cleared all errors from `mount -a`.

Mount Options:
UUID : The Universal Unique Identifier.
`/mnt/cdrom` : The mount point.
`auto`: Indicates the filesystem should be mounted during bootup. `user`: A root user can mount and umount (unmount) this filesystem. Aside: A regular should be able to `mount` or `umount` with this option but currently only a root user can do this operation. *Caution:* The command to unmount (dismount) a filesystem is `umount`.
`0`: Field 5, determines backup preference. `0`: Field 6, determines the order of filesystem checkup using `fsck`.

Using `UUID` is the preferred method to identify filesystems. Device names, for example `/dev/sdb1` or `/dev/sr0` are also accepted in /etc/fstab.

`mount` has 8 error codes. After the command is run its success, or failure can be tested using: `echo $?` the error code will display. `echo $?` shows the error code of the last command.

For example: Immediately after `mount -a` type `echo $?` Here are three of the eight error codes you will need. For more error codes read the man listing for `mount`.

0 success 1 incorrect permission, mount must be run as root 32 mount failure

Mounted filesystems can be checked using the `findmnt` utility. Use `findmnt` on its own to list all filesystems or `findmnt -t iso9660` to list a particular filesystem, in this case `iso9660`.

4.6 Mounting an ISO image

Mounting an ISO image lets you see the contents of the image. The image is made to simulate a CD ROM disk. The image you are mounting should be on the host OS, i.e., Win 10. Use the Ubuntu Image file, if you have deleted it, then get it from the link on LMS. Read all instructions first before attempting the procedure. Read the note at the end.

Procedure to Mount an ISO image.

1. Locate your image file, for example, `ubuntu-20.04.1-desktop-amd64`, on your host (Win 10) OS's folder.
2. Start VM Ware and Ubuntu.
3. In the VMWare screen, on the top left hand corner on the tab Ubuntu 64-bit, right click the tab. From the popup select `Settings...`
4. On the left panel, from the Virtual Machine Settings window, Select CD/DVD (SATA).
5. On the right panel select the radio button "Use ISO image file:"
6. Browse to the location and select the `ubuntu-20.04.3-desktop-amd64.iso` file
7. In the device status check box. Check Connected, uncheck Connect at power on. Click OK.
8. The CDROM icon at the bottom of the screen should light up, indicating that CD ROM is active.
9. If the computer does not have a physical CD ROM drive the icon will not light up. VMWare will give an error because the drive did not connect during the Ubuntu bootup process.
10. Use the Add feature from the left panel, and attempt to add a CD ROM drive.
11. Open a terminal window in Ubuntu, login, change status to root; `su` or `su -`
12. Create a mount directory `/mnt/cdrom`. Change permission to 777 using `chmod 777 /mnt/cdrom`
13. mount the CD ROM drive using the command:
    `mount -t iso9660 /dev/cdrom /mnt/cdrom`
    You will get a warning from Ubuntu
    `mount: /mnt/cdrom: WARNING: device write-protected, mounted read-only.`
    Though you have given write permission to the directory it mounts it as a read only device. The physical characteristics override the device permissions.
14. Display the contents of the CD ROM `ls -l /mnt/cdrom`. A partial listing is shown below.

```
dr-xr-xr-x 1 faculty faculty  2048 Jul 31  2020 boot
dr-xr-xr-x 1 faculty faculty  2048 Jul 31  2020 casper
dr-xr-xr-x 1 faculty faculty  2048 Jul 31  2020 dists
dr-xr-xr-x 1 faculty faculty  2048 Jul 31  2020 EFI
```

15. Reverse the process. Unmount the image using the command, `umount /mnt/cdrom`
16. Gracefully close your virtual machine using: `shutdown -h now`.

    This will ensure all files are closed and the `init` process terminates.
17. The mounted drive is effective for this instance of startup. After Ubuntu is rebooted the mount will not function, you will have to mount it again. To mount the ISO9660 image automatically during startup, you need to modify the `/etc/fstab` file.

Notes:

1. In some instances the CDROM will not connect during startup. Perform steps 2 to 8 and restart the Ubuntu virtual machine.
2. CD ROM is mounted as a read only device. Ubuntu will give you a warning `mount: /mnt/cdrom: WARNING: device write-protected, mounted read-only`
3. Similar to mounting a filesystem, only a Superuser can `umount` a mounted device. Using `umount` with inadequate privileges will give you the following message.
   ```
   umount: /home/faculty/data: umount failed: Operation not permitted.
   ```
4. A device cannot be mounted twice. The previous mount must be un-mounted first.
5. A mount point does not have to be at `/mnt`, although it is a convention to mount external devices at this point.
6. Each command given using account can also be given using `sudo`. For example, from a regular users account, use : `sudo mount -t iso9660 /dev/cdrom /mnt/cdrom`

4.7 Lab 4 - Mounting a DOS Folder in Ubuntu

Objective

1. Create a folder in Win 10 and assign Sharing Privileges to the folder.
2. Check connectivity between guest OS, i.e., Ubuntu and host OS, i.e., Windows 10
3. Mount the folder in DOS in the `/mnt` directory in Ubuntu

Requirements

1. `net-tools` in Ubuntu.
2. `cifs` file system on Ubuntu.
3. Windows Password.

Submission

Show your instructor the connection and complete the quiz.

Requirements

VMWare, Ubuntu last LTS version, default student account, Win10 as host OS

Procedure

Creating a folder in Win 10. In the host OS create a folder on the desktop, call it `Ubuntu-Share`.

Allow Network File Sharing. At the bottom right hand corner of the Windows desktop, right click on the Network icon. Select `Open Network and Internet Settings`. Select `Status` from the left panel. From the right panel select `Network and Sharing Center`. In the new window, select `Change Advanced Sharing Settings`. In the `Network discovery` section, ensure that `Turn on network discovery` radio button is selected. The `Turn on automatic setup of network connected devices` should be checked. In the `File and printer sharing` section, `Turn on file and printer sharing` radio button should be selected. Close the `Status` window

Assigning Permissions. Right click on the folder and select `Properties`. Choose the `Sharing` tab. Select the `Share...` button. You are the administrator of your Win 10 computer. From the drop down menu select `Everyone` and select `Add`. The lower panel with `Name` and `Permission Level` should have `Everyone` as an entry. Select `Share`, select `Done`. Next choose `Advanced Sharing...`. Check the `Share this folder` check box. In the lower panel select `Permissions`. Allow `Full Control`, `Change` and `Read` by selecting the check boxes. By default only `Read` is checked. Select `Apply` and `OK`. Close the `Folder Properties` window.

Determine User Name in Win10. Open a command window. Press the Start Button in Win. In the search bar, type `cmd`. A DOS window should appear. At the prompt type in `whoami`, this is your username in Win10. Write it.
Win 10 Username:

Determine IP address of host computer. In the same window type `ipconfig` Look for the Ethernet adapter. Write the IP address (besides the IPv4 Address).
Host IP's Address:

Determine the Guest's IP address. You will need to have `net-tools` to determine the IP address of the Ubuntu computer. Install it by typing `sudo apt install net-tools`. After the tools are installed, type in `ifconfig`. Write the IP adress, it should be after `inet` and before `netmask`. *Aside:* Check can check if net-tools is installed by using `apt list | grep net-tools`, an entry should appear, possibly in red from `grep`.

Check Connectivity with DOS. Type `ping -c 4 <IP Address>`. Replace `<IP Address>` with the host computer's IP address you wrote in the earlier step. Confirm that you have 4 packets transmitted. This step will determine that you have a socket connection with the host.

Create a mount point. in Ubuntu, create a mount point in the /mnt directory using

```
sudo mkdir /mnt/Win-Share
```

Install cifs filesystem. In Ubuntu, install `cifs` by typing

```
sudo apt install cifs-utils
```

Mount the DOS folder. Type in

```
sudo mount.cifs //<host IP Address>/Ubuntu-Share /mnt/Win-Share -o username=<Win 10 username>
```

Type in your Windows 10 password.

Test the connection. Using `sudo touch <filename>` in the `/mnt/Win-Share` folder you should be able to see the file in the Windows Folder.

Caveat The folder will disconnect if your computer is in sleep mode.

4.8 Review Questions

Multiple Choice Questions

1. Identify the directory that stores device files, i.e., files that interface with hardware and filesystem.
   A. `/etc`
   B. `/dev`
   C. `/root`
   D. `/var`
   E. `/bin`

2. Identify the directory that stores log files and administrative files.
   A. `/etc`
   B. `/dev`
   C. `/lib`
   D. `/var`
   E. `/bin`

3. Which directory stores libraries and executables needed to run programs that usually reside in `/bin` and `/sbin`
   A. `/etc`
   B. `/dev`
   C. `/lib`
   D. `/var`
   E. `/bin`

4. `ln -s oldfilename slinkname` creates a
   A. symbolic link (soft link) to a file
   B. hard link to a file
   C. lists the directory structure in short form
   D. removes the link from oldfile and places it in slinkname

   Answer the following five questions based on the output of `fdisk -l /dev/sdc`

```
Disk /dev/sdc: 2147 MB, 2147483648 bytes
255 heads, 63 sectors/track, 261 cylinders, total 4194304 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x97e8fc8b

Device Boot      Start         End      Blocks   Id  System
/dev/sdc1         2048      206847      102400   83  Linux
/dev/sdc2       206848      309247       51200   82  Linux swap / Solaris
/dev/sdc3       309248      514047      102400    5  Extended
/dev/sdc5       311296      362495       25600   83  Linux
/dev/sdc6       364544      415743       25600   83  Linux
```

5. How many logical drives are created?
   A. 1
   B. 2
   C. 3
   D. 4

6. How many primary partitions have been created?
   A. 0
   B. 2
   C. 3
   D. 4
   E. 5
7. How may additional primary partitions can be created?
   A. 0
   B. 1
   C. 2
   D. 4
   E. 5
8. Identify the extended partition
   A. `/dev/sdc1`
   B. `/dev/sdc2`
   C. `/dev/sdc3`
   D. `/dev/sdc5`
   E. `/dev/sdc6`
9. Identify the maximum number of primary partitions are allowed in a drive?
   A. 0
   B. 1
   C. 2
   D. 3
   E. 4
10. How many maximum extended partitions are allowed in a drive?
    A. 1
    B. 2
    C. 3
    D. 4
    E. 5
11. Logical drives can reside only on
    A. a primary partition
    B. a swap partition
    C. an extended partition
12. Which of the following command will display all active swap partitions?
    A. `swapoff`
    B. `mkfs`
    C. `swapon -s`
    D. `fdisk -l`
13. Select one correct option with reference to `inodes`
    A. A hard link has the same `inode` number as the original file.
    B. A soft link has the same `inode` number as the original file.
    C. It is possible for two files to have the same `inode` number.

4.9 Links

A link is an alternate file name to an existing file. A symbolic link contains the name of the underlying file, the size of the symbolic link will be the number of characters in the name of the underlying file.

Reference: Pages 211-217, Sobell. Table 4.1 shows a comparison between the features of soft links and hard links.

Links in the root directory Run `ls -l /`, it will list the root directory. How many soft links do you observe? What is the use of these soft links? Think of a URL that needs to change and users are still using the previous URL.

Exercise: This exercise demonstrates the permissions, changes in permission to hard links and soft links.

1. Create a directory, call it `ln_practice`. Change to this directory and run these commands in this directory, the cleanup will be easy.
2. Create a file, `touch queen`
3. Create a symbolic link to the file `queen`. `ln -s queen queen.sl`
   `queen.sl` is the symbolic link.
4. Observe the inode numbers and the file permission of the underlying file and the symbolic link using
   `ls -il queen*`
5. Change the permission of the underlying file by changing the permission of the symbolic link, the permission of the symbolic link will remain the same, i.e., `rwxrwxrwx`, but the permission of the underlying file will change.
6. Create a hard link using `ln queen queen.hl`
   `Note: -s` option is not required to create a hard link, the notation `.hl` is used for hard link as a convenience.
7. Observe the inode numbers and the permission of the underlying file and the hard link, use `ls -li que*`. Observe the number of links reported by `ls`, this is the number after the file permission.
8. Create another hard link to the underlying file, `ln queen queen.hl2`, create another hard link using the previous hard link
   `ln queen.hl2 queen.hl3`
9. Observe the inode numbers, permission and the number of links displayed by the `ls` command.
10. Change the permission of any hard link and observe the change in all hard links.
11. Add some data to any hard link or the underlying file, for example `date >> queen.hl2`
12. Observe the size of the links and the underlying files, observe the number of links. Will all the links occupy the same amount of disk space?
13. Delete the underlying file or any hard link of your choice.
14. Observe the results using `ls`
    The user needs to remove all the hard links and the underlying file to release the inode number.
15. Remove all softlinks and hard links, remove the directory `ln_practice`.

Hard Link	**Soft Link** / `Symbolic Link`
The term `link` refers to a hard link.	A soft link is also called a symbolic link.
`inode` number of the link(s) and the underlying file are same.	A new `inode` number is assigned to the link.
Cannot create a hard link to a directory.	Softlinks were developed to overcome this limitation and have links to directories.
A hard link can be created only to an existing file.	A soft link must first be created using an existing file, it can be retained after the file is deleted. This feature does have practical uses in log files.
Links must be on the *same* filesystem, because `inodes` are the same.	A softlink can reside on different filesystems. Compare to a Windows shortcut - you can have a shortcut to a file (or folder) on your desktop or another drive.
Hard links are phased out; you still have to know its function, they exist in older scripts and systems. Mac OS X uses hard links for backups.	A file or directory can have more than one soft link.
A file can have more than one hard link.	More than one soft link and more than one hard link can be created for a file.
Syntax: `ln file hlink` `link file hlink`	Syntax: `ln -s file slink`
A file cannot be deleted until all hard links are removed.	If the original file is removed the link still exists. After the file is recreated, the link becomes active. *Aside:* If the file to the link does not exist, Ubuntu will change the color of the soft link. This behaviour is the same in Windows. A shortcut can be created for a file or folder. If the file is deleted the shortcut remains, but will not function. When the file is recreated the link is activated.
Hard links display as regular files, with the same permissions as the original file.	Soft links display as `lrwxr-xr-x ....  slink -> Readme.txt` where slink is the softlink and `Readme.txt` is the file.
Files can be moved within the filesystem, the hard links will automatically point to the file.	Same property as soft links.
Use `unlink` or `rm` to remove soft link and hard links.	

Table 4.1: Links a Comparison

4.10 Review Questions

1. Information about a file, a directory, the directory's parent and each of its children is stored in a data structure called
   A. `inode`
   B. `metadata`
   C. `timestamp`
   D. `linkcount`
   E. hardlink

2. Identify the command that will create a hardlink called `box_office_hit` for file `movie.mp4`.
   A. `ln box_office_hit movie.mp4`
   B. `ln movie.mp4 box_office_hit`
   C. `ln -s movie.mp4 box_office_hit`

3. Identify the link that has the same inode number as the file.
   A. hard link
   B. soft link

4. After the underlying file is deleted, a softlink remains in place but becomes inactive.
   A. True
   B. False

5. A hard link cannot be created for a directory.
   A. True
   B. False

6. There can be either one hard link or one softlink for a file.
   A. True
   B. False

7. Soft links and hard links can be removed using the `rm` command.
   A. True
   B. False

5

User Management

A user's activity is determined by her role and the group she belongs to. User's information is stored in the `/etc/passwd` file, among other files in the `/etc` directory. This chapter shows examples to create a user, modify user's environment, the shell, account expiry date among other login details.

5.1 Roles and Users

A sample entry in the `/etc/passwd` file is shown:

```
user1:x:1003:1003:User Name,,,,:/home/user1:/bin/bash
```

The fields are separated by a colon, a short description of each field is listed.

1. login name, this is the users login name. *Note:* Many entries in the `/etc/passwd` file are roles, i.e., they specify tasks, for example `backup`. Not all accounts are intended to be login accounts. For example, `bin`, `daemon` have a different purpose.
2. the users password is represented by a `x`. The encrypted password is stored in `/etc/shadow`, this file has restricted permissions, although `/etc/passwd` is visible to all users.
3. The user ID, each user has a unique ID, ideally. A unique ID is provided by the system.
4. group ID, by default Linux creates a group ID with the same number as the user ID. This is the default action to facilitate user creation on a Linux Desktop version. In a production system, a group must exist before a user is created and each new user is associated with the group.
5. Additonal user's information, for example, Full Name, Office Room Number, Phone Number.
6. The user's home directory.
7. The user's default shell, `bash` remains the most commonly used shell.

5.2 `/etc/passwd` and `/etc/shadow`

Compare the permission's of the two files.

```
-rw-r--r-- 1 root root 2901 Mar 23 11:53 /etc/passwd
```

and `-rw-r----- 1 root shadow 1658 Mar 24 12:45 /etc/shadow`

Although the root has read and write permission to the `/etc/passwd` file, an individual user can modify some of her details. This is done by a special procedure in Unix systems. `/etc/shadow` is much more restrictive.

Exercise: Display the contents on the `/etc/shadow` file, you will need to use `sudo`.
An example of a users entry is shown:
`jsbach:6adUJKduXZCDiAG5W:19024:365:1000:30::20088:`
The colon is the field separator. The second field is the encrypted password. The length of the encrypted password does not correspond to the length of the actual password. You will also see `!!`, `!` and `*`. The `!` means the user's password has not been set. Often you will see `!` in front of an encrypted password, it means the account has been locked.

5.3 `/etc/shadow`

The third field in the `/etc/shadow` file indicates the last password change, the number indicates the days since 1 Jan 1970. Use an spreadsheet to verify the number.
The forth field is the number of days required between password changes, the the example it is 365 days.
The fifth field is the maximum number of days the password remains valid, i.e., the user must change the password every 1000 days.
The sixth field is the number of days for a warning to appear before the password is set to expire.
The seventh field is the number of days (grace days) after which the account is disabled, after the password is set to expire.
The eight field is the number of days when the account cannot be used. This is the number of days since 1 Jan 1970.
The last field is the number of previous passwords that are not permitted to be reused.

Exercise: Try the cut utility on `/etc/shadow` `sudo cut -d:  -f1-3 /etc/shadow`

Question: If two users have the same password, will the encrypted password be the same?

5.4 Adding Users

New users can be added using the `useradd` command. The format is:
`useradd [options] [username]`

In its simplest form a user user can create a user account as:
`sudo useradd gverdi`
without any options. This will create an account with entries in:
`/etc/passwd`, `/etc/shadow` and `/etc/group`.
It will create a unique `userid` and also a group id with the same number.

Options for `useradd`

Some common options are:
`-d` Define a home directory for a new user.
`-g` Specify a group name, the group must exist. Without the `-g` a group is created with the same user id.
`-G` Supplementary group. A user can belong to more than one group.
`-c` Additional user information; full name, room number, phone number.
`-N` A generic group called `users` with group id `100` exists on the system. Specifying the `-N` option will not create a new group but will add the new user to the generic group `users`.

`-e` Sets the account expiration date, the date format is `YYYY-MM-DD`.
`-s` Login shell, the default is `sh`, found in `/bin/sh`. To specify `bash` as the default shell for the new user use `-s /bin/bash`.
`-m` Create a home directory, and copy the files from `/etc/skel` directory to the new users home directory. There are three files in the `/etc/skel` directory, `.bash_logout`, `.bashrc` and `.profile`. These three files are copied to the home directory.
`-D` The command `useradd -D` will display the default values. Using the `-D` option will create the new user with the details specified in the command line and the defaults from the `-D` option.
Aside: Use the `groups` command to determine your group associations.
Note: `/etc/group` is a file, `groups` is a command.

An example for user Giuseppe Verdi is shown below:
Note: Groups `composer`, `computer` and `staff` must exist for the command to work, read section 5.5 to add groups.

```
useradd -c "Giuseppe Verdi" -d /home/verdig -m
-g composer -G baroque,staff -e 2030-12-31
-s /bin/bash verdig
```

5.5 Adding and Deleting a Group

Adding and removing groups is done using root privileges.
The command `groupadd` adds groups, and `groupdel` removes a group.
Usage: `groupadd [groupname]`
Examples: `groupadd composer`
`groupadd baroque`
`groupadd staff`
The results can be checked using `cat /etc/group`
`groupdel groupname` will remove a group.
A group cannot be deleted if it has users that belong to it. The users must be moved to another group or the users must be removed first.

5.6 Removing a User's Account

The command `userdel` removes a user.
The usage is: `userdel [options] [username]`
Example : `userdel -r gverdi`
will remove the user `gverdi`, his home directory and all files inside the home directory. It will also remove the entries from the `/etc/passwd` file and `/etc/shadow` file.
`userdel gverdi`, without the `-r` option will not remove the users home directory but will remove entries from the `/etc/passwd` and `/etc/shadow` files.

5.7 Modifying a User's Account

The command `usermod` modifies several items in the `/etc/passwd` file for the user.
Usage: `usermod [options] username`
Examples of options:

`-c` adds comments
`-d` changes the home directory.
`-m` creates a home directory, if it does not exist, `-m` will also copy the contents of the existing home directory to the new directory.
`-g` changes the initial group of the user.
`-G` changes the supplementary group. Provide a list of groups separated by a comma do not use spaces to separate group names. If the user is not a member of a group it will add the user to the group. *Caution:* The user is associated to only those groups that are specified in the list, i.e., the user is removed from a group if the group name is not specified. To add a user to a group and maintain the users existing groups, use `-a` with `-G`.
`-s` changes the login shell in the `/etc/passwd` file, the next time a user logs in the new shell will be used.
`-e` changes the expiry date of the account. After this date the account will be disabled.The date format is `YYYY-MM-DD`. For example, `usermod -e 2025-12-31 user1` will change the expiry date to 31 Dec 2025 for user1
`-l` changes the login name.
`-L` will lock the users account, it will prefix the encrypted password in the `/etc/shadow` file with `!`.
`-U` will unlock the users account. It will remove the `!` as a prefix from the from the encrypted password.
Note: The primary group of the user is specified in the `/etc/passwd` file.

Example: `usermod -c "Sherpa Tenzing, WT218, ext 4529, 613-787-2813" tenzs`

Changing a users shell with `chsh`

An administrator may not want to give permission to use `usermod` if the user simply wants to change the shell. `chsh` is a specific command to change the users shell, it performs only one task.
Usage: `chsh -s /bin/bash gverdi`

5.8 Changing Group Association

`newgrp` will change the users current group for the login session. The syntax is:
`newgrp [-] [group]`

The `-` is optional, as indicated by square brackets. If the `-` is used, the user's environment is initialized and the directory is changed. Note that the parameter `group` is optional too. It implies that the command can be used without any arguments; in this case `newgrp` will revert the user to the default group as specified in the `/etc/passwd` file.

Exercise: At your command prompt type.
`groups` Observe the first group, it should be the user's default group.

Type: `newgrp cdrom`
This will change the users group to `cdrom`. Confirm by typing `groups`. Observe the command prompt, especially the directory, alternatively use `pwd` to confirm the working directory.
Create a file using `touch`. Observe the group ownership of the newly created file. Does it show the group owner as `cdrom`?
Remove this file.

Revert to the default group, type `newgrp`
Confirm by typing `groups`.

Change the working directory to `/tmp`

Type: `newgrp - cdrom`
Observe the command prompt and the working directory. Did you notice the change? Revert to the original group, type in `newgrp`.

5.9 Password Information

`chage` manages account password parameters. It updates account expiration date, defines the warning period to alert the user of password expiration, and other details related to a users account. This utility is owned by `root` and the its group `shadow`
The syntax is: `chage [options] LOGIN`, replace `LOGIN` with your account login.
In its simplest form `sudo chage LOGIN` will change the account password parameters.

Exercise: To list all password parameters type:
`sudo chage -l LOGIN`
The seven parameters listed in the above command line are stored in `/etc/shadow`.
Check the details using, for example, `sudo tail /etc/shadow`
Change the expiration date using `chage -E YYYY-MM-DD`, replace `YYYY-MM-DD` with a future date, for example, the end of the year.
Note the changes in the `/etc/shadow` file.

5.10 Miscellaneous Topics on User Management

Files in `/etc/skel`

Files in the directory `/etc/skel` are copied to the home directory when a new user is created.

`useradd -D`

`useradd -D` displays default values that are used during `useradd`, this option does not require `sudo` privileges.

CPU information

Review the file `/proc/cpuinfo`, it gives details of the CPU used.

Adding and Removing Users

`gpasswd` will add and remove users from a group.

Lists the groups to which a user belongs

`groups <username>` will list the groups to which a user belongs.

5.11 Administering `/etc/group` and `/etc/shadow`

Review the man pages on `gpasswd` for details.

5.12 Review Questions

Multiple Choice Questions

1. Identify the file that stores encrypted passwords for users
   A. `/etc/fstab`
   B. `/etc/passwd`
   C. `/etc/group`
   D. `/etc/shadow`
   E. `/etc/gshadow`

2. `user1` is logged in. Identify the command that will change the default shell for `user1` to `/bin/sh` the next time `user1` logs in?
   A. `useradd -s /bin/sh user1`
   B. `usermod -s /bin/sh user1`
   C. `sudo usermod -s /bin/sh user1`
   D. `$SHELL=/bin/sh`

3. With reference to Question 2 above. After the user's default shell has been changed, how will you verify the default shell? The user is still not logged out and continues to use the existing shell.
   A. send an email to the user and ask him to confirm
   B. check the `/etc/passwd` file
   C. check the shell variable `$SHELL`

4. Identify the command that will allow the superuser to delete the account and home directory of `user2`
   A. `userdel user2`
   B. `userdel -r user2`
   C. `usermod -r user2`
   D. `usermod -d user2`

5. How will you verify that `user2` has been deleted from the system?
   A. `ls /home/user2`
   B. `ls /root`
   C. `cat /etc/fstab | grep user2`
   D. `cat /etc/passwd | grep user2`

6. Identify the command that is used to delete a group in Linux.
   A. `gpasswd`
   B. `groupmod`
   C. `groupdel`
   D. `newgrp`

7. Identify the command that is used to add a group in Linux.
   A. `gpasswd`
   B. `groupmod`
   C. `groupadd`
   D. `newgrp`

8. How will you check the groups in a Linux System.
   A. `cat /etc/passwd`
   B. `cat /etc/shadow`
   C. `cat /etc/group`

9. Identify the command that will display lines from a file based on a given pattern.
   A. `cat`
   B. `type`
   C. `grep`
   D. `wc`

Short Answer

The `root` user runs the following commands. Based on the outcome answer the following 5 questions.

```
groupadd account
groupadd finance
groupadd audit
useradd -g account woqag
useradd -g account gavop
useradd -g finance turoz
useradd -g finance nibeg
gpasswd -a turoz finance
gpasswd -d nibeg finance
usermod -G account,finance,audit woqag
groups woqag
```

10. List the member(s) of group `account`

11. List the members of group `finance`

12. What is the output of the command `groups woqag`

13. What is the outcome of the command
    `usermod -G audit woqag`? Which groups will user `woqag` belong to after this command is run?

14. The command `useradd user1` will
    A. add a user called `user1` to the group `root`
    B. add a user called `user1` to the first group on the list
    C. create a group called `user1` and add a user called `user1` in the newly created group
    D. give an error

15. The command `useradd -G account,audit user2` will
    A. add `user2` to groups `account` and `audit`.
    B. remove user2 from group `account` and add it to group `audit`.
    C. give an error.

16. The command `useradd -g audit user3` will
    A. add `user3` to group `audit`
    B. will make `user3` an administrator to group `audit`
    C. give an error

17. The command `gpasswd -a user1 audit` will
    A. add `user1` to group `audit`
    B. make `user1` the administrator to group `audit`
    C. delete `user1` from group `audit`
    D. give an error

18. The command `gpasswd -d user1 audit` will
    A. will give an error
    B. add `user1` to group `audit`
    C. will make `user1` the administrator to group `audit`
    D. will delete `user1` from group `audit`

5.13 Additional Topics

This section illustrates some exceptions to topics on user management. It is not part of the course and is not tested. You may read this in your own interest.

Creating two users with the same ID

```
While it is possible to create two users with the
same UID, this practice is not recommended. The system is vulnerable to
an exploit.

A non-unique UID is implemented using the -o option

The example below shows two users that have the same UID, 4500
The -u option specifies the UID.

# adds two users benb and petp
useradd -u 4500 -o -c "Ben Benjamin" benb
useradd -u 4500 -o -c "Pete Peters" petp
```

```
# This command will fail because it does not have the -o option
# but has the -u option with the same UID as the previous two
# users.
useradd -u 4500 -c "Tas Tasker" tast

# the following command will succeed
# The system allocates a UID to user tast
useradd -c "Tas Tasker" tast
```

Group Administrator not in the Group It is possible for a user to be a group administrator and not belong to the group.

The example below demonstrates the feature: Login as root and use create a group, three users and one admininitrator The administrator does not belong to the group

```
# run the following commands as root
# create two groups lacrosse and solitaire
groupadd lacrosse
groupadd solitaire

# add three users to group lacrosse
useradd -c "Sil Silken" -g lacrosse sils
useradd -c "Ned Nedam"  -g lacrosse nedn
useradd -c "Qus Quset"  -g lacrosse qusq

# add one user to group solitaire
useradd -c "Nas Naset" -g solitaire nasn

# set passwd for each user
passwd sils
passwd nedn
passwd qusq
passwd nasn

# add another user
useradd -c "Lacrosse Admin" lacadmin

# set password for user lacadmin
passwd lacadmin

# make user lacadmin the group adminintrator for group lacrosse
gpasswd -A lacadmin lacrosse

# confirm GID and UID for user lacadmin
id lacadmin

# at this stage you will be user lacadmin
# login as lacadmin
login lacadmin
```

```
# use lacadmin to add user nasn to group lacrosse
gpasswd -a nasn lacrosse

# check crededtials of user nasn
# nasn should belong to groups lacrosse and solitaire
id nasn

# reverse the operation by first deleting the users and then the groups

# Aside: users initial group can be changed using the command usermod
# the following command changes the group of user sils from
# lacrosse to cdrom
usermod -G sudo,cdrom sils

# Aside: if a user belongs to a group and if that group is the users primary group
# then the group cannot be removed

3. While it is possible to check the log for users created, deleted or
modified it is not part of the course outline.
Hence it is not considered a valid answer in the quizzes.

# With the root account, the log file can be checked
# using the command
tail -20 /var/log/auth.log
```

6

Shell Script

6.1 Writing Shell Scripts

Some common commands such as `echo` and `read` with common options are discussed. Selection and Iteration are then covered in this chapter.

Considerations when writing `bash` scripts.

(i) `bash` is positional, unlike `C` or `SQL`. End of an expression or evaluation must be terminated by a semicolon or a new line. The statement
`if [ $balance -eq 0 ] then`
will give an error.
The correct syntax is
`if [ $balance -eq 0 ]; then`
or, write it in two lines as:
```
if [ $balance -eq 0 ]
then
```
(ii) The space after the square bracket in the sample code in (i)is required, for example:
`while [ $dow -le 7 ]` is the correct syntax
`while [ $dow -le 7]` will give an error.

(iii) Create shell scripts in the `bin` directory in your home directory, i.e., `/home/student/bin`. By default the directory is included in the path. Check by using `$PATH`. If you do not use the bin then include the path by using `PATH=$PATH:/home/students/scripts`, where `scripts` is the directory that stores the `bash` scripts. If the path is not added to the environment variable then run each script by explicitly providing the current directory, for example, `./tax.sh` when you are in the directory that holds the script.

(iv) You need read and execute permission to run a `bash` script. Each file must be given execute permission using (say) `chmod 764 tax.sh` to the shell script. Without the permission a script can be run as a subshell using `bash <filename.sh>`

(v) Use `vim` to edit the script. *Aside:* If you use `vi` your employer will think you are over 50. There are several other editors, but for now use `vim`, it will help you answer test and exam questions on vim. `vi` is available on all `Unix` systems, `vim` is available on all `Linux` systems.

(vi) Text files on Linux are different than those on DOS/Windows. In Linux each line ends with a line feed, `\r`. In DOS each line ends with two characters a carriage return and a line feed. `\r followed by \n`. The `dos2unix` utility will convert dos files to Unix files. Read `man dos2unix` for details.

(vii) Do not create or edit bash scripts in Notepad on Windows and copy it to Linux. End of line characters are different in Unix and Windows/DOS.

Debugging Tips These tips will help you locate errors and make editing easier.

(i) Create a file `.vimrc`, insert the line `set number`. This will display the line numbers in the shell script when editing the file in `vim`.
(ii) Take note of the line numbers and the error messages after running the script.
(iii) To go to the line number in `vim`. In the command mode, type in the line number followed by G, this will take you to the line number; you can now begin editing. For example, to go to line 23, type in `23G` in the command mode.

6.2 bash

`Reference: https://www.gnu.org/software/bash/manual/bash.html#Introduction`

`bash` is a shell; a shell is a command interpreter, it accepts commands from the user, parses it and runs the command; this is an example of a shell used in interactive mode. There are commands in bash that support selection and iteration, thus bash is a command interpreter and a programming language. Instructions written in a Unix shell, such as `bash`, are called shell scripts. `bash` scripts provide variables, flow control constructs, quoting and functions.

Aside: `bash` is an acronym for 'Bourne-Again SHell', a pun on its author Stephen Bourne. bash is a derivative of sh the first Unix shell.
`bash` is compatible with sh and has features from the Korn shell, `ksh` and the C shell `csh`; `bash` interactive and programming improvements over sh.
`bash` is intended to be a compliant with IEEE POSIX Shell and Tools portion of the IEEE POSIX specification, IEEE Standard 1003.1. Details are at

`https://standards.ieee.org/project/1003\_1.html#Standard`

The GNU operating system provides several other shells, including a version of `csh`, `bash` is the default shell. `bash` is portable, it runs on Unix and other operating systems. Independently-supported `bash` ports are available for MS-DOS, OS/2, Windows.

A shell is a macro processor; a macro processor expands text and symbols to create expressions.
As a command interpreter, the shell provides the user interface to GNU utilities. The programming language features in bash allow these utilities to be combined. Commands can be grouped with programming features and they become commands themselves. These new commands have the same status as system commands in directories such as `/bin`, allowing users or groups to establish custom environments to automate their administrative, programming and software application tasks.

As a command interpreter `bash` offers command history, command line editing, job control and command line editing.

bash has a small set of commands that are coded within the shell and are part of the shell, these commands are appropriately called builtin's. Commands that are used most often are coded as builtin's. Examples: of builtin's are `echo`, `pwd`, `exec`, `read`.

6.3 Exercise: Writing Your First Script

In this exercise you will write a simple script, change its permissions and run it from your home directory. You will then move it to the `bin` directory within your home direcotry and run it.

Create a simple script:

Write the following commands in a script, call it `first.sh`. Write the script in your home directory.

```
#!/bin/bash
date
cd /
ls
pwd
```

Change the permission of script to make it executable, i.e., it should have at least `r-x` for the owner of the file. Run the file from your home directory using `./first.sh`. Observe the working directory before and after the script runs.

Check the `PATH`

Verify that you have the bin directory in your home directory.
Check the `PATH` variable using `echo $PATH`. Do you see the `bin` directory in the path? *Note:* The `/usr/bin` directory is different from `bin` in your home directory.

move the script

Move `first.sh` to your bin directory use `mv`. Did you use the absolute path or relative path?
Run the script without the `./` preceding the script. Write your present working directory after the script finishes execution.

6.4 Display with `echo`

echo is a built-in and a utility. It *prints* the argument to the standard output, the default output is the display. `echo` is located at `/user/bin`.

> *Aside:* Try the commands `whereis echo` and `which echo`
> Now make a copy of the echo command to your local directory
> `cp /usr/bin/echo .`
> Type: `which echo`
> Observe the results. Delete the newly copied file `rm echo`

Its simplest form is:

```
echo "Which is the best college in Ontario?"
echo "Algonquin College"
```

In both the above cases a single quote will give the same result. to include an apostrophe double quotes is required. `echo "Chinua Achebe's Choice?"`

`echo` can be used expand filenames, for example `echo b*.sh` will list all filenames that begin with `b` and end in `.sh`.
Output of echo can be redirected to a file, for example

```
echo "Today is :" + $( date +"%A" ) > DayOfWeek
```

Interpreting Backslash

Using the `-e` option in `echo`.
`\n` is a newline character in Unix.

Try the command `echo "\n"`
`echo` will output the literal `\n`. We really want a new line, i.e., the interpretation of `\n`.
We need `echo -e "\n"`.
`echo`, now displays an empty line, i.e., it has interpreted the backslash, `\`. There are several special characters that can be used with the echo command.
Try `echo -e "\a"`
The bell will sound.

Suppress the trailing newline

Using the -n option in `echo`. `echo` completes the output with a newline. Using echo at the command prompt will show the prompt at a new line. In a shell script the echo command's new line feature could be suppressed to get the user input on the same line. Try the following two options and observe the difference.

```
echo    "This month is :" $( date +"%B" )
echo -n "This month is :" $( date +"%B" )
```

6.5 Get User Input with `read`

The `read` command accepts user input in a variable and stores it. The `read` command will accept one line at a time. For example,

```
echo "What month is this?";  read this_month
echo $this_month
```

the `-p` option

An automatic variable called `REPLY` is used if a variable is not assigned. For example, the statement

```
read -p "What month is this?"
```

will display a prompt and wait for the users input. The string input typed in by the user is stored in the variable called `REPLY`. Type `echo $REPLY` to confirm.

The `echo` and `read` have been combined into one command using `read -p`.

the `-s` option

Use the silent option `-s` to enter text that is not displayed. For example, to enter passwords.

6.6 Expressions

An expression is a code fragment that is processed and returns a value. A code fragment consists of constants, variables, operators - arithmetic and logical.

6.7 Arithmetic Evaluation

There are at least five ways to perform an arithmetic evaluation. The most robust and compact syntax uses double parenthesis. Examples are shown below.

```
(( TotIncome = DivIncome + Income ))
(( TotalDeduction = TaxPaid + RRSP ))
```

Double parenthesis does not require a $ sign before variable names. It allows use of spaces between operators, this improves readability.
Aside: The assignment `Age=35` is equivalent to `(( Age = 35 ))`. `Age = 35`, without the brackets, will give an error, there cannot be a space before and after an assignment operator.

Exercise: Arithmetic Evaluation & Arithmetic Expansion

At the command prompt type

```
a=7
echo $a
```

You should see the value 7 stored in the variable `a`, echo will display the value.
Try, `b = 7`, although this syntax improves readability, it will give an error.
`(( b = 8 ))` will assign the value 8 to the variable b.
Aside: `echo b` will display the literal `b`, `echo 'b'` and `echo "b"` will also display the literal `b`.
To perform an arithmetic operation on `a` and `b`, say an addition, you would type,
`(( c = a + b ))`
this statement will perform an arithmetic evaluation or
`c=$(( a + b ))` Note: spaces are not allowed before and after the assignment operator.
`(( c = $((a + b ))`, Note: spaces allowed, redundant syntax, but can be used for a lengthy evaluation.
verify the variable `c`, by `echo $c`
An older syntax is using the evaluation expression `expr`
``d=`expr $a + $b` ``, Note: spaces are not allowed before and after the assignment operator.

6.8 Evaluating Conditions

A simple decision construct is an `if` statement. The general syntax is:

```
if [ expression ]
   then
   <action>
fi
```

A default action needs an `else` statement.

```
if [ expression ]
   then
   <action>
else
   <default action>
fi
```

To test multiple conditions; a compact form using `elif` is shown below:

```
if [ expression ]
   then
   <action>
elif [ expression ]
   <action>
else
   <default action>
fi
```

Exercise : Run the two scripts `balance.sh` and `balance_1.sh` and observe the results.

```
#!/bin/bash
# Filename: balance.sh
# Author  : S I'Aret
# Date    : 4 Mar 2021

echo "Enter Balance :"
   read balance

if [ $balance -eq 0 ]
    then
    echo "Sleep well, you don't owe any amount"
fi

if [ $balance -gt 0 ]
    then
    echo "You have a Credit Balance"
fi

if [ $balance -lt 0 ]
    then
    echo "You have a Debit Balance"
fi

# eof: balance.sh
```

```
#!/bin/bash
# Filename: balance_1.sh
# Author  : S I'Aret
# Date    : 4 Mar 2021
# Comment : Using if-elif-fi

echo "Enter Balance :"
read balance

if [ $balance -eq 0 ]
   then
   echo "Sleep well, you don't owe any amount"
elif [ $balance -gt 0 ]
   then
   echo "You have a Credit Balance"
   # this check is redundant, can be replaced with else
elif [ $balance -lt 0 ]
   then
   echo "You have a Debit Balance"
fi

# eof: balance_1.sh
```

Figure 6.1 illustrates the simple flow of `if then else fi` construct. These `else` statement is used as a default outcome.

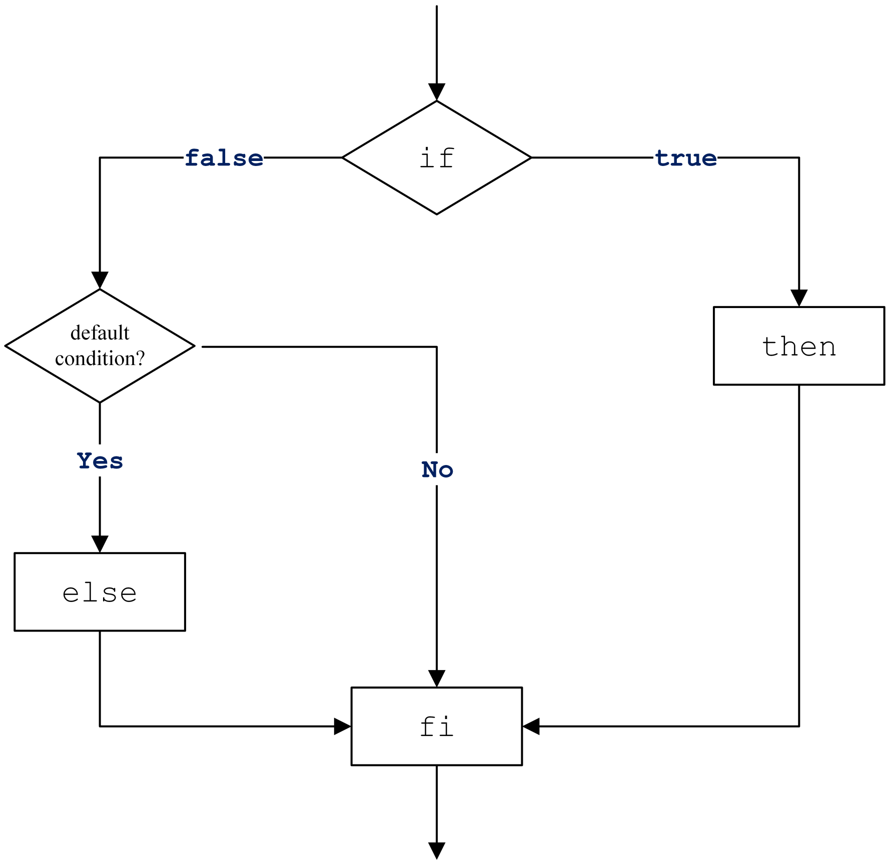


Figure 6.1: basic `if then else` construct

6.9 Using Quotations

What do each of the quotes mean in `bash`?

1. `echo pwd`
2. `echo $pwd`
3. `echo $(pwd)`
4. `echo "$((pwd))"`
   Note: This question has three possible answers.
5. `echo '$(pwd)'`
6. `echo "$(pwd)"`

6.10 Tutorial - Read & Selection Statements

Objective

— This tutorial covers three `bash` programming constructs.

  - `read`
  - `echo`
  - `if, then, else` decision, and logical operations.

— Accepts numerical values from the user.

— Performs simple arithmetic using arithmetic evaluation operators in `bash`

— Makes decisions based on the results using control structures and logical operators.

— Displays the results.

Background

Based on a taxpayers age, income, savings and other numbers there are some conclusions. The user enters a few simple numbers and the program shows a simple outcome. The intent is to get familiar in writing a `bash` script to perform these tasks. *Aside:* The calculation serve as examples and do not reflect actual financial outcomes.

Requirements

(i) The program should run as a `bash` script. Specify this requirement clearly in the script using `#!/bin/bash`

(ii) Write your Name, Student Number, Section and Semester as comments in the script. End the script with a comment indicating the end of file.

(iii) Accept the following numerical values from the user, the variable names are shown after the value.

– Age, `Age`

– Income, `Income`

– Dividend Income, `DivIncome`

– Tax Paid, `TaxPaid`

– RRSP Deposited in the accounting year, `RRSP`

(iv) Add Dividend Income and Income, call it `TotalIncome`

(v) Add Tax Paid and RRSP, call it `Deduction`

(vi) If Age is greater than 65 and `TotalInc` is greater than `40,000` print `Happy Retirement`

(vii) If Age is more than 60 years and RRSP is less than $500, print `Financial Advice Suggested`

(viii) If Age is less than 25 and RRSP more than $1000, print `Prudent Saver`

(ix) Exit the program with 0 as success

Sample Script

```
#!/bin/bash
# FileName: tax.sh
# Date : 1 Mar 2021
# Author: Vixen Taar
# Student ID: 010-375-188
# Semester: Winter 2021
# Comments:
# Accepts four numerical values from user
# performs simple arithmetic opeartions
# prints outcomes
# Updates Required: Validate user inputs
# Explore other avenues to happy retirement besides $$$

# get values from user
echo "Enter Age ( 0 to 120 ):"
   read Age
echo "Enter Annual Income ( 0 to 70000 ):"
   read Income
echo "Enter Dividend Income ( 0 to 5000 ):"
   read  DivIncome
echo "Tax Paid (0 to 15000):"
   read TaxPaid
echo "Enter RRSP Contribution ( 0 to 3000 ):"
   read RRSP

# simple arithmetic calculations
(( TotalIncome = Income + DivIncome ))
(( TotalDeduction = TaxPaid + RRSP ))

echo "Total Income    :" $TotalIncome
echo "Total Deduction :" $TotalDeduction

if [ $Age -ge 65 ] && [ $TotalIncome -ge 45000 ]
then
   echo "Happy Retiree"
elif [ $Age -le 25 ] && [ $RRSP -ge 1000 ]
then
   echo "Prudent Saver"
elif [ $Age -ge 65 ] && [ $RRSP -le 500 ]
then
   echo "Suggest Financial Planner"
else
   echo "All’s Well That Ends Well"
fi

# eof: tax.sh
```

6.11 Iteration

bash provides three constructs for iteration, for, while and until. Each of the three constructs has its advantages.

6.12 Example - while statement

Figure 6.12 illustrates the while loop. The expression must be TRUE for the control to enter the loop.

```
#!/bin/bash
# Filename : iteration_1.sh
# Date     : 14 Mar 2021
# Adapted from A Practical Guide to Ubuntu, 4e
# Page 1022
# Displays numbers from 1 to 7
# The loop is run while the condition is TRUE

dow=1   # Day of Week
while [ $dow -le 7 ]
do
echo -n "$dow" " "
(( dow += 1 )) # increment by 1
done
echo

# eof: iteration_1.sh
```

Figure 6.2: A while loop

6.13 Examples - until statement

Figure 6.13 illustrates an incorrect until condition. The expression must evaluate to FALSE, the control then enters the loop. Simply changing while to until does not work. *Aside:* The bash Reference Manual indicates the until loop will work for the condition as not zero, i.e. FALSE.

Aside: The until construct in bash is different from the until construct in other languages such as Pascal. In Pascal, using until for iteration will guarantee at least one iteration regardless of the condition, the test is made at the end of the loop. In bash the test is done at the beginning as in while.

The correct until construct is illustrated in Figure 6.13.

```
#!/bin/bash
# Filename : iteration_2.sh
# Date     : 14 Mar 2021
# Adapted from A Practical Guide to Ubuntu, 4e, Page 1022
# Illustrates difference between while & until
# simply changing while to until does not
# work.
# The expression in until must evaluate to FALSE
# for the control to enter the loop.
# allows entry to the loop

dow=1   # Day of Week
until [ $dow -le 7 ]
do
echo -n "$dow" " "
(( dow += 1 )) # increment by 1
done
echo

# eof: iteration_2.sh
```

Figure 6.3: An incorrect `until` loop

```
#!/bin/bash
# Date     : 14 Mar 2021
# Filename : iteration_3.sh
# Illustrates iteration - until

dow=1   # Day of Week
until [ $dow -gt 7 ]
do
echo -n "$dow" " "
(( dow += 1 )) # increment by 1
done
echo

# eof: iteration_3.sh
```

Figure 6.4: Correct version of `until` loop

6.13.1 `break`

The `break` statement will exit the inner-most loop, the control will return to the next outer loop. The `break 2` statement will exit two levels of while loops. Use of break statement is an indication of quick coding practices suitable for prototyping, it should be avoided in production code. Uncomment `break` and run the script, next uncomment `break 2` and observe the results.

```
#!/bin/bash
# Date     : 07 Apr 2021
# Filename : while_break.sh
# Illustrates break
# Displays 52 weeks and 7 days in a week.
# Notes: uncomment break and break 2 statements and run the code.

(( dow = 1 ))
(( week = 1 ))

while [ $week -le 52 ]
do
while [ $dow -le 7 ]
do
echo "Week $week, Day $dow"
(( dow += 1 )) # increment by 1
# break will exit only the inner loop
# uncomment break and test the script.
# break
# break 2 will exit two levels of nested while loops
# break 2
done
(( dow = 1 ))
(( week += 1 )) # increment by 1
done
echo

# eof: while_break.sh
```

Figure 6.5: The `break` Statement in Iterations

6.13.2 Review Questions - Iteration

1. How many times will the `echo` statement execute when the following script is run?
   Aside: List the output of the `echo` statement.
   ```
   for num in {1..10..2}
   do
      echo $num
   done
   ```
   A. 0
   B. 2
   C. 3
   D. 4
   E. 5

2. Study the following script. How may times will the `echo` statement execute
   ```
   (( a = 3 ))
   while [ $a -ge 3 ] || [ $a -le 5 ]
   do
      echo "Everest"
      (( a++ ))
   done
   ```
   A. 0
   B. 3
   C. 4
   D. infinite

3. How many times will the echo statement execute?
   ```
   for (( count = 3; count < 7; count++ ))
   do
     echo $count
   done
   ```
   A. Infinite
   B. 0
   C. 3
   D. 4
   E. 7

6.14 Review Questions

Multiple Choice Questions

1. Which of the following uses the bash shell as the command interpreter in a script?
   A. `/bin/bash`
   B. `#/bin/bash`
   C. `!/bin/bash`
   D. `#!/bin/bash`

2. What special variable will indicate the exit status of the last command?
   A. `$#`
   B. `$0`
   C. `$?`
   D. `$9`

3. What special variable will indicate the number of positional parameters?
   A. `$#`
   B. `$0`
   C. `$?`
   D. `$9`

4. Which symbol is used to indicate the beginning of a comment in a shell script.
   A. `#`
   B. `$`
   C. `!`
   D. `>`
   E. `<`

5. Which operator is used when you want the second command to execute only when the preceeding command succeeds?
   A. `;`
   B. `&&`
   C. `||`
   D. `>>`

6. What is the output of the command
   `echo "$(pwd)"`
   A. The command will give a syntax error
   B. The current working directory will be displayed
   C. `$(pwd)`
   D. `pwd`

7. `ls -l log.03052015` gives the following output.
   `-rw-r-r- 1 user2 faculty 0 Mar 5 19:59 log.03052015`
   What is the output of the following command
   `echo "$(ls -l log.03052015 | cut -d" " -f4)"`
   A. `user2`
   B. `faculty`
   C. `0`
   D. `Mar`
   E. The command will give a syntax error

8. What is the output of the command
   `echo 'We pay $32 per month for Internet'`
   A. `'We pay $32 per month for Internet'`
   B. `We pay $32 per month for Internet`
   C. `We pay 32 per month for Internet`

   Answer the following three questions based on the following declaration in `bash`
   `groups=(audit:2001 finance:2002 account:2003 sales:2004 purchase:2005)`

9. What is the output of the following command
   `echo ${groups[1+2]}`
   A. `account:2003`
   B. `finance:2001`
   C. `sales:2004`

10. What is the output of the command
    `echo ${#groups[*]}`
    A. 0
    B. *
    C. 4
    D. 5

11. What is the output of the command
    `echo ${groups[*]}`
    A. `audit:2001`
    B. There is no output from the command
    C. `audit:2001 finance:2002 account:2003 sales:2004 purchase:2005`

12. What is the output after the following statements are executed
    ```
    x=20; y=30
    { x=300; y=400; }
    (x=30; y=40)
    echo $x $y
    ```
    A. `20 30`
    B. `300 400`
    C. `30 40`

13. In bash shell, what is the operation done by braces `{    }`?
    A. arithmetic evaluation
    B. parameter expansion
    C. logical evaluation
    D. command substitution
    E. arithmetic expansion

14. In bash shell, what is the operation done by double brackets `((     ))`?
    A. arithmetic evaluation
    B. parameter expansion
    C. logical evaluation
    D. command substitution
    E. arithmetic expansion

15. In bash shell, what is the operation done by a dollar sign and double brackets `$((     ))`?
    A. arithmetic evaluation
    B. parameter expansion
    C. logical evaluation
    D. command substitution
    E. arithmetic expansion

16. In bash shell, what is the operation done by a dollar sign and brackets `$(     )`?
    A. arithmetic evaluation
    B. parameter expansion
    C. logical evaluation
    D. command substitution
    E. arithmetic expansion

Exercise

1. Study the following script to understand its task. How can you enhance the script
   (a) to recognize a directory
   (b) to check if a file has execute permission

```
#!/bin/bash
# filename: is_file.sh
# adopted from page 957, Sobell, 4e
if test $# -eq 0
   then
   echo "Need at least one argument"
   exit 1
fi

if test -f "$1"
   then
   echo "$1 is an regular file"
else
   echo "$1 is not an regular file"
fi
# eof: is_file.sh
```

Based on the following shell script answer the three questions that follow

```
#!/bin/bash
# adapted from pg 1021, Sobell, 4e
# filename: inc_dec.sh
n=10
echo $n
echo $(( --n + 3 ))
echo $n
echo $(( n++ - 3 ))
echo $n

#eof: inc_dec.sh
```

Note: `--n` takes precedence over `+`, `-` takes precedence over the post increment/decrement operator.

2. Write the output of each of the five echo statements

3. Explain the behavior of preoperator in `--n`

4. How is it different from postoperator `n++`?

5. What is the output of `echo s{o,u,i}n`

6. What is the purpose of `$( )`

Additional Questions

1. `echo $((5+7))` will display
   - A. The string `5+7`
   - B. The string `$((5+7))`
   - C. `12`
2. `echo 9+3` will display
   - A. The string `9+3`
   - B. The string `echo 9+3`
   - C. `12`
3. `echo "$((4+8))"` will display
   - A. The string `4+8`
   - B. The string `echo 4+8`
   - C. The string `$((4+8))`
   - D. `12`
4. `echo '$((1+11))'` will display
   - A. The string `1+11`
   - B. The string `echo 1+11`
   - C. The string `$((1+11))`
   - D. `12`

Study the shell script `ascript.sh` and answer the two questions that follow:

```
#!/bin/bash
# filename: ascript.sh
echo $$
cd /
ls
pwd
# eof: ascript.sh
```

5. The shell script is executed as `bash ascript.sh` from `/home/user5`. What will be the output? Choose all that apply.

   A. display the `pid` of the new shell that in which the script will run
   B. display the `pid` of the current users shell
   C. change directory to the root directory
   D. display the contents of the root directory
   E. display the working directory, i.e. `/`
   F. display the users home directory, `/home/user5`
   G. After the script exits the working directory will be `/`
   H. After the script exits the working directory will be `/home/user5`

6. The shell script is executed as `source ascript.sh` from `/home/user5`. What will be the output and the working directory after the script completes? Choose all that apply.

   A. displays the `pid` of the new shell that in which the script will run
   B. displays the `pid` of the current users shell
   C. changes directory to the root directory
   D. displays the contents of the root directory
   E. displays the working directory, i.e. `/`
   F. displays the users home directory, `/home/user5`
   G. After the script exits the working directory will be `/`
   H. After the script exits the working directory will be `/home/user5`

7. The statement `if [ $# -eq 0 ]`

   A. Will check if the number of files in the directory is more than zero
   B. If there is atleast one command line arguement
   C. The process id `pid` is greater than zero
   D. The exit status of the previous command is equal to zero

6.15 Review Questions - Selection

1. What is the output of the script?
   ```
   #!/bin/bash
   (( a = 4 ))
   (( b = ( 5 * a )/2 ))
   if [ $a -le $b ] || [ $a -gt 10 ]
   then
      echo $a $b
   else
      echo $b $a
   fi
   ```
   A. `4 4`
   B. `4 10`
   C. `10 4`
   D. `10 10`
2. What is the output of the script? `#!/bin/bash`
   ```
   (( p = 7 )); (( q = 20 ))
   if [ $p -eq 5 ]
   then
   echo 'p is' $p
   elif [ $q -le 20 ]
   then
   echo 'q is' $q
   else
   echo "No condition satisfied"
   fi
   ```
   A. `p is 7`
   B. `q is 20`
   C. `No condition satisfied`
3. Select one correct choice regarding iteration in `bash`. For control to enter the loop,
   A. the condition in the `while` loop must be true and the condition in the `until` loop must be false.
   B. the condition in the `while` loop must be true and the condition in the `until` loop must also to true.
   C. the condition in the `while` loop must be false and the condition in the `until` loop must be true.

6.16 Running a Shell Script

A shell script can be run in two ways. For example, a script called `calc.sh` can be run

1. in the current shell, using the command
   `. ./calc.sh` Note the space after the period.
   or
   `source ./calc.sh`
2. in a separate newly spawned shell as `./calc.sh`

Each of the two options has different effects on environment variables. Running the script as

```
source calc.sh
```

will run the script in the current shell. Any changes made to current environment variables will be affected.
In the second option, running the script as a regular script, will have a different effect. First `bash` will spawn another process and then execute the script in that process. After the script completes the newly spawned shell, it terminates and control is return to the current shell. Any environment variables altered by the script will not affect the current environment variables. A script run as `source`, i.e. in the current shell, will alter the environment variables.
There is another difference, running the script in the current shell as `source` does not require execute permission.

6.16.1 Exercise

This exercise demonstrates the execution of a shell script in two modes. It attempts to change the current shell prompt `PS1` by running a script in each of the two modes. It demonstrates that in only one of the two modes the desired change is made. The exercise also illustrates the use of command substitution `$( )`.

1. Save the current prompt `PS1` to a file called `prompt.org` (`org` for original), using the command `echo $PS1 > prompt.org`
2. Create a file called `prompt.sh` with two lines as shown below:
   ```
   #/bin/bash
   PS1="$(hostname -f:  )"
   ```
3. Try to run the script with the default permission.
   `./prompt.sh`
   It will fail.
4. Change the permission; give the script `execute` permission.
   `chmod 764 prompt.sh` and then run the script.
   `./prompt.sh`
   There is no change, although the script would have run successfully. This is because the prompt changed in the newly spawned shell, the shell exited after completion and the control returned to the current shell.
5. Run the script as `source prompt.sh`. The prompt should have changed.
6. Return `PS1` to the original prompt using the command.
   `PS1=$(cat prompt.org)`

6.16.2 Exercise

Remove execute permission from the script using, `chmod 644 prompt.sh`
Repeat 5, the script should run without execute permission.
Bring the prompt back using Step 7.
Clean up the residual file `prompt.org`, using `rm prompt.org`

6.16.3 Review Questions

1. Consider running a script called `add.sh`. Which of the following options will run the script in the current shell? Select two choices.
   A. `.  add.sh`

   B. `add.sh`

   C. `source add.sh`
2. A script can run without execute permission in the current shell.
   A. True

   B. False
3. A script must have write permission to run.
   A. True

   B. False

6.17 Exercise: Sourcing the Script

In this exercise you will run the script in the current shell, i.e., you will source the script.
Modify `first.sh`, insert
`echo "PID:" $$`
at the last line in the script.
Check your current process id using `echo $$`

Run the script the usual way by typing `first.sh`. Observe (i) the present working directory before running the script and after the script completes and (ii) the Process ID.

Sourcing the Script

Run the script using `source first.sh`
Alternatively, you can also run the script replacing the keyword `source` with a dot, as shown: *Note the space after the dot.*
`.  first.sh`
Observe (i) the present working directory before running the script and after the script completes and (ii) the Process ID.
Return to your home directory.

Write your observations.

Note: `bash ./x.sh` will take precedence over `#!/bin/bash`, i.e., you can override the shell specification from the command line.

6.18 Builtin's

Overview Builtin commands run faster than shell scripts because they are stored in memory. Functions run in the environment of the shell that calls it, a shell script is usually run in a sub-shell. Commands in a function are pre-processed - therefore they run faster.

A full external version of the some builtin's commands exists in `/bin`. Many external commands have a builin. The intention of a builtin is to provide efficiency. Not all builtin's have a corresponding external command, examples of builtin's without an external command are `exec`, `read`, `trap`, `eval`, `cd`, `break`, `continue`. Examples of builtin with an external command are: `echo`, `pwd`, `kill`.

Command's such as `cd`, `break`, `continue`, and `exec` must be implemented as builtin's because they manipulate the shell itself.

Caution: There is a disadvantage to a builtin. In situations where there are two versions of the same command, in some situations you will get two separate results. At this stage in programming you will not come across this anomaly.

Review Questions - Functions

1. What is the output of the echo statements?

```
#!/bin/bash
(( s = 12 ))
function func_print()
{
  echo $s
  (( s = 14 ))
  echo $s
}

echo $s
func_print
(( s = s + 10 ))
echo $s
```

   A. The numbers 12, 14, 14, 22 are printed on each line
   B. The numbers 12, 12, 14, 24 are printed on each line
   C. The numbers 12, 12, 14, 22 are printed on each line

2. What is the output of the echo statements? *Hint:* Carefully analyse the line `local s=14`.

```
    #!/bin/bash
    (( s = 12 ))
    function func_print()
    {
        echo $s
        local s=14
        echo $s
    }
    echo $s
    func_print
    (( s = s + 10 ))
    echo $s
```

   A. The numbers 12, 14, 14, 22 are printed on each line
   B. The numbers 12, 12, 14, 24 are printed on each line
   C. The numbers 12, 12, 14, 22 are printed on each line

3. Identify one correct statement with reference to functions in bash
   A. A function can have global variables, local variables, and can return values.
   B. A function can accept only global variables, it cannot have local variables.
   C. A function can accept only local variables, it cannot have global variables and cannot return values.

6.19 Process ID

Each process in Linux has a unique id called process id, `pid` for short. A process is a program that is running, it has CPU cycles allocated to it and it occupies space in main memory. A process can spawn other processes, therefore a each process will have a parent process id and its own process id. After a process completes the id

is discarded and recycled at a later time.
The command `echo $$` will display the process id of the current process. Verify by typing `ps` at the command prompt. `echo $PPID` will display the parent process id. The current shell that a user is logged in is a process.

Each shell script that runs in a new shell that is spawned by the current shell, after the script has completed execution the control returns to the current shell. A shell script can be forced to run in the current shell, i.e not run in a new shell. The script in figure 6.6 displays the process id and parent process id. Run the script in the current shell by typing `source pid.sh` or `. pid.sh` and then by typing `pid.sh`. The second option will run it in a spawned shell.

```
#!/bin/bash
# Filename : pid.sh
# Date     : 15 Mar 2021
# displays the current process id and
# the parent process id.
# run the script
# 1. on its own
# 2. using either source pid.sh or . pid.sh

echo "Parent  Process ID is " $PPID
echo "Current Process ID is " $$

# uncomment the exit command below and
# run the script as . pid.sh or
# source  pid.sh
# observe the results

#exit

# eof: pid.sh
```

Figure 6.6: Displying Process ID & Parent Process ID

6.20 Command Substitution

Command Substitution allows the result of a command to be stored in a variable. For example, the assignment `today=date` will store the literal `date` to a variable `today`. `echo $date` will display the literal string `date`.
The result of the date utility is stored in the variable `today` using the **command substitution** operator, `today=$( date )`. In this example, the date command is executed and its result is stored in the variable today.
Note: There is no space between `$(`.
This is the newer more robust and predictable operator used in `bash`. The older syntax used backquotes. The symbol below the ~ sign on the keyboard, is the symbol `` ` ``.
The assignment ``today=`date` `` will give the same result as the previous operation. This syntax is found in older scripts that a programmer may be required to maintain. The use of `` ` `` in new scripts is discouraged.

Lab 7 - Shell Scripting

Objective

1. Write basic `bash` shell commands as a script.
2. Use shell definition in a script.
3. Alter the permission of the script to make it executable.
4. Use path name to execute the script. Store the script in the correct location for execution without a path qualifier.
5. Change the `PATH` variable.
6. Document the script.
7. Use the selection constructs in `bash`, `if-then-else-elif`.
8. Accept user input with prompt using `echo`, `read` with options.
9. Perform simple arithmetic operations on user input.
10. Convert user input into an equivalent value using selection statements.

Reference

1. Chapter 6
2. Study and run the sample scripts.

Submission & Weightage

Upload the tested and working shell script on LMS. Rename the script with a `.txt` extension.
Weight: 6% of Final Grade.

Requirement

Ubuntu 20.04.1 or later running on a virtual machine. Class notes. Write the script with the specifications described below.

Specification

1. Use the first line of the script to specify that the script runs in the `bash` shell.
2. You may change the `PATH` variable and place the script in the `bin` directory in your home directory. Alternatively, you may run the script from your home directory without modifying the `PATH` variable. Change the file permission of the script.
3. The name of the script should be `mygrade`.
4. Comment the code with Student Name, Student Number, Semester, Course Code, Course Section, Instructors Name. Write date of creations, document all changes to the script as you test it and correct errors.
5. In 2 to 3 sentences, write a brief purpose of the script. Use your own words.
6. Script should run on a clean screen. Use the appropriate command to start the program on a fresh screen.

7. Prompt the use to enter four integer values. The user will not enter decimal values, only positive integers. They are `Assignment`, `Test 1`, `Test 2` and `Final Exam Grade`. Store these values in appropriate variables.
8. Do not test for valid input, assume the user will enter the values within the range. The acceptable range is shown in the sample screen.
9. Add all four values, this sum will be the final numerical grade.
10. Convert the numerical grade into letter grade, using the following values.
    ```
    90-100, A+.  85-89, A. 80-84, A-.
    77-79, B+.  73-76, B. 70-72, B-.
    67-69, C+.  63-66 C. 60-62, C-.
    57-59, D+.  53-56, D. 50-52, D-.  0-49, F.
    ```
11. Refer the sample input session and the output.

Sample Input Session

```
Script to Convert Numerical Grade into Letter Grade
Enter Assignment mark (0 - 40):  39
Enter Test 1 mark (0 - 15):  14
Enter Test 2 mark (0 - 15):  15
Enter Final exam mark (0 - 30):  29
Your final numeric grade is 97, and your final letter grade is A+
```

Rubric

Sr No	**Requirement**	**Maximum**	**Earned**
1	Code Documented according to specification.	2	
2	Formatting & Presentation, Header and Footer correctly used.	2	
3	Script has correct output.	2	
4	Code runs without errors.	2	
5	All decisions in `if-then-else-elif-fi` correctly handled.	2	
6	Demonstration given on Instructors request.	2	
	Total	**12**	

7

Functions

Functions are blocks of code that can be invoked with a name. Functions in bash have the same properties as other programming languages. They accept values and return values. Variables have local and global scope. Functions can be be recursive. Functions reduce repetition of code, facilitate code reuse, ease in maintenance and support structured coding.

7.1 Functions- Some Properties

Functions can be written in two ways, on the command line and in a shell script. The general format is

```
function <function name>()
{
statements
}
```

Functions are written at the beginning of the script. A function must be defined first before it is called, otherwise `bash` will give a `command not found` error. When a function is called the statements inside the function are executed, after all statements are completed, control is returned to the next line in the main program. A function can call another function. Call a function by using its name without the brackets.

Passing values Values are positional in bash functions. The script in figure **??** demonstrates the way parameters are displayed based on the order they are passed. Run the program `DisplayIt.sh` with two parameters, for example `DisplayIt 5 7`, and observe the order they are displayed.

```
#!/bin/bash
# Filename : DisplayIt.sh
# Purpose: Demonstrate parameter positions
#          swaps the parameters in the function
function DisplayIt()
{
    echo $1 $2
}

# function DisplayIt is called
DisplayIt $2 $1

# eof: DisplayIt.sh
```

Figure 7.1: Passing Parameters, positional

```
#!/bin/bash
# Filename: compare.sh
compare ()
{
    if [ $1 -eq $2 ]
    then
    echo $1 "equals" $2
    elif [ $1 -gt $2 ]
    then
    echo $1 "is greater than " $2
    else
    echo $1 "is less than " $2
    fi
}
compare $1 $2

# eof: compare.sh
```

Figure 7.2: Passing Parameters, Comparison example

7.2 Functions in `bash`

Variables in `bash` functions have scope. Scope is the boundary within which the variables can accessed. Two scopes are local and global.

Local Variables A variable with local scope can be accessed at all places in the script. A variable with local scope can be accessed only by the function, after the statements in function are run,the local variables are discarded. The value of a local variable cannot be altered by an assignment statement that is outside the function. A local variable can have the same name as a global variable, the function will treat them as separate variables. Study the script in figure 7.2. The local and global variable is intentionally kept as `x`, to illustrate the scope. The keyword `local` can only be used in a function. *Aside:* The assignment statement `(( local x = 5 ))` does not work, you will have to use the conventional syntax `local x=5` without spaces.

```
#!/bin/bash
# Filename : local_var.sh
# Purpose : Demonstrates the scope of a variable
# Date : 7 Nov 2021

# declare function check_x
check_x() {
    local x=5  # declare variable x as local
    echo "Within function check_x.    : " $x
}

(( x = 10 ))
echo "Outside function check_x. 1.: " $x

check_x

echo "Outside function check_x. 2.: " $x

# eof: local_var.sh
```

Figure 7.3: Scope of Local Variable

Global Variables Global variables can be declared anywhere in the script; they can be declared in a function too. Variables declared inside a function without a `local` keyword are treated as global variables. *Aside:* In the previous example in figure 7.2 remove the keyword local and run the script, observe the difference in the output.

Return Values The `return` statement returns a value to the calling function. Conventionally `0` is used for success, `1` for failure. Additionally, any number between `1` and `127` can be used to denote a specific error. The script in table 7.2 illustrates the action taken by the calling statement to take action based on the return value from the function. *Aside:* Note the command substitution `$(ls -l $1)`.

```
#!/bin/bash
# Filename: FileExist.sh
# Objective: Demonstrate return values from a function
# Date : 8 Nov 2021
function FileExist () {
    if [ -e $1 ]
    then
    return 0
    else
    return 1
    fi
}

if FileExist $1
then
echo $1 "File exists: "
echo $( ls -l $1 )
else
echo $1 "File not found."
fi

# eof: FileExist.sh
```

Figure 7.4: Return Value from Functions.

Functions at the command line A temporary function can be written at the command line. At the prompt type the code from figure 7.2, notice the secondary prompt `PS2`:

```
function add() {
(( sum = $1 + $2 ))
echo $sum }
```

Figure 7.5: Command Line Function

The function needs to be exported before using it. Type in
`export -f add`
Next, call the function as `add 4 5`.
At the command prompt, type in `env` to see the environment variables, the function is listed as
`BASH_FUNC_add%%`.

Lab 8 - Shell Scripting - 2

Objective

1. Use a Function to write a simple menu.
2. Demonstrate previously used User Management utilities.
3. Call Utilities from Menu options.
4. Document the script.
5. Use the iteration constructs in `bash`, `while-do-done`.
6. Accept user input with prompt using `echo`, `read` with options.

Reference

1. Chapter 7
2. Study and run the sample scripts. Use sample scripts as basis to write this script.
3. Use `temp_conv.sh`, shown in figure 7.2 as a basis to write this script.
4. Use class notes and slides for reference on User Management.

Submission & Weightage

Upload the tested and working shell script on LMS. Rename the script with a `.txt` extension.
Weight: As specified in the Course Section Information Document (CSI).

Requirement

Ubuntu 20.04.1 or later running on a virtual machine. Class notes. Write the script with the specifications described below.

Specification

1. Use the first line of the script to specify that the script runs in the `bash` shell.
2. The name of the script should be `myscript`.
3. Comment the code with Student Name, Student Number, Semester, Course Code, Course Section, Instructors Name. Write date of creation, document all changes to the script. Document all test and debug tasks.
4. In 2 to 3 sentences, write a brief purpose of the script. Use your own words.
5. Script should run on a clean screen. Use the appropriate command to start the program on a fresh screen.
6. Write a menu to provide the user with a choice of six items. Give each item a letter to be chosen, `A` to `F`, the final option is `Q` to quit the program.
7. A Sample menu is shown in figure 7.2. Use the program `temp_conv.sh` as an example.
8. Use a `while` loop. Allow the user to selection one menu item.
9. Parses (Parse)the users choice using selection statements. Use `if-then-else-elif-fi`. Alternatively, you may use `case-esac`.
10. Accept upper and lower case input from the user. Use the `||` operator. An example is given in `temp_conv.sh`

11. For Choice A. Prompt the user for **username**, **home directory**, **default login shell**. Create a user account using this information.
12. For Choice B. Accept **username** from the user. Delete the user account and the home directory.
13. For Choice C. Accept **username** and **group name** from the user. Add the group as a supplementary group for the user.
14. For Choice D. Accept **username** and **group name**. Change the initial group name for this user.
15. For Choice E. Accept **username** and **shell name**. Change the default shell for this user.
16. For Choice F. Accept **username** and **expiration date**. Change the expiration date for the user account.
17. For Choice Q. Exit the script with exit code 0.
18. Use sleep 3 after each menu item is selected, then display the menu option again.

Sample Input Session

```
Program to Manage User Accounts.
Author            : Vixen Taar
Student Number    : 041-999-999
Semester          : Fall 2021
Course - Section  : CST 8102, Section 531

A. Create User Account
B. Delete User Account
C. Change Spplementary Group for a User Account
D. Change Initial Group for a User Account
E. Change default login shell for a User Account
F. Change account expiration date for a User Account
Q. Quit
```

Figure 7.6: Sample Menu Listing

Rubric

Sr No	Requirement	Maximum	Earned
1	Code Documented according to specification.	2	
2	Formatting & Presentation, Header and Footer correctly used.	2	
3	Script has correct output.	2	
4	Code runs without errors.	2	
5	Iterations and choices from A to F. correctly handled.	6	
6	Demonstration given on Instructors request.	2	
	Total	**16**	

Sample Script to Demonstrate Menu, Functions and Iteration.

```
#!/bin/bash
# Filename : temp_conv.sh
# Date     : 15 Mar 2021
# Updated  : 20 Mar 2021, exit statement removed.
#            Quit option corrected.
#            initial value of choice corrected.
#          : 7 Nov 2021, spelling corrected
# Objective:
# demonstrate functions
# convert temperature from
#    - celsius to fahrenheit
#    - fahrenheit to celsius

display_menu() {
    clear
    echo ' Temperature Converter '
    echo ' --------------------- '
    echo 'C. Convert Celsius to Fahrenheit'
    echo 'F. Convert Fahrenheit to Celsius '
    echo 'Q. Quit'
    echo ''
    echo -n 'Select C, F or Q: '
}

cels_to_farh() {
    # echo 'inside c to f'
    # uses automatic variable REPLY
    read -p "Enter Temperature "
    (( temperature = REPLY ))
    (( farh = ( temperature * 9 / 5 ) + 32 ))
    echo "$temperature Celsius is $farh Fahrenheit"
    sleep 3
}

farh_to_cels() {
    # echo 'inside f to c'
    read -p "Enter Temperature " farh
    (( cels = ( farh - 32 ) * 5 / 9 ))
    echo "$farh Fahrenheit is $cels Celsius"
    sleep 3
}
```

```
choice=n   # initialize choice to a dummy value

while [ $choice != "Q" ] && [ $choice != "q" ]
do
display_menu
read choice

if [ $choice = ’C’ ] || [ $choice = ’c’ ]
   then
   cels_to_farh
elif [ $choice = ’F’ ] || [ $choice = ’f’ ]
   then
   farh_to_cels
elif [ $choice != ’Q’ ] && [ $choice != ’q’ ]
   then
   echo ’Valid options are C, F or [Q]uit’
   sleep 3
fi

done

# eof: temp_conv.sh
```

Figure 7.7: Script to Convert Temperature from Farhenheit to Celsius, Celsius to Farhenheit

Lab 9 - Calculator

Objective

1. Parse command line parameters.
2. Write a simple menu to process user choice and perform arithmetic.

Reference

1. Study the sample script `mul_div.sh`. Use this as a reference and then implement the lab's requirements.
2. Review iteration, selection, functions and command line parameters.

Submission Upload your working script to LMS. Rename it with a `.txt` extension and then upload.

Procedure

1. Write a script, call it `mycalc`, use `vim` to edit the file.
2. Use the first line of the script to specify that the script runs in the `bash` shell.
3. Comment the code with Student Name, Student Number, Semester, Course Code, Course Section, Instructors Name. Write date of creation, document all changes to the script. Document all test and debug tasks.
4. In 2 to 3 sentences, write a brief purpose of the script; use your own words.
5. Script should run on a clean screen. Use the appropriate command to start the program on a fresh screen.
6. The script should perform two arithmetic operations, addition and subtraction.
7. Use functions to accept two operands and an operator, a total of three parameters. If the number of parameters are more than three then display a usage message.
8. Check the operator, if it is not `+` or `-` then display an error message.
9. Based on the arithmetic operator call the appropriate function to perform the arithmetic operation.
10. Check the number of parameters. If the script is called without parameters then prompt the user for three inputs, operand, operator and another operand; perform the arithmetic operation. Display the results.

Rubric

Sr No	Requirement	Maximum	Earned
1	Code Documented according to specification.	2	
2	Formatting & Presentation, Header and Footer correctly used.	2	
3	Script has correct output.	2	
4	Code runs without errors.	2	
5	Arithmetic functions correctly written.	4	
6	Demonstration given on Instructors request.	2	
	Total	**14**	

8

Regular Expression

A regular expression is a sequence of characters, numbers, special characters and wildcards that specifies a pattern. The pattern is used in to search text that matches the pattern; pattern matching is used in a commands such as `grep`, `find`, `ls`, `vim` among others. Pattern matching is also used to validate text. For example, text entered by a user in a data field is validated using regex as a part of a constraint. A regular expression is often referred as *regex* in short. The concept of regex precedes `UNIX`, it was first proposed by Stephen Cole Kleene in 1950. Regex is used in databases such as Oracle and postgres, in spreadsheets and search engines for searching and pattern matching.

Meta Character	**Match String**
`*`	Matches zero, one or more characters in a string.
`?`	Matches a single character in a string.

Table 8.1: Regex Pattern Matching

Meta Character	**Match String**
`^`	Matches a character at the beginning of a line.
`$`	Matches a character that is at the end of a line.

Table 8.2: Regex beginning and end of line

Exercise Try the following commands and observe the results, use `sample.txt` provided.

1. `grep ^The sample.txt`
2. `grep end sample.txt`
3. `grep end^ sample.txt`

The square brackets `[ ]` represent a set of characters. To match all numerals in the search pattern try. The special character (caret) ^changes its meaning when placed inside square brackets. It negates the contents inside the brackets.

Exercise Try the following examples on the file `sample.txt`

1. `grep [0-9] The sample.txt`
2. `grep [^0-9] The sample.txt`

The second example will list all lines that have letters, lines that have only numbers will not display.

Quantification

A quantifier specifies the number of times the element can repeat in a search pattern. An element is a token, character or a group of characters. Table 8.3 lists the quantifiers and their usage.

Quantifier	**Match String**
`?`	A single occurrence or no occurrences of the preceeding element.
`*`	Zero or more occurrences of the preceeding element.
`+`	One or more occurrences of the preceeding element.
`{n}`	The preceeding element is matches by exactly n characters.
`{min,}`	The preceding element is matched `min` number of times or more.
`{,max}`	The preceding element is matched a maximum of `max` times.
`{min,max}`	The preceding character is matched `min` times, and not more than `max` times.

Table 8.3: Quantifiers

Exercise Examples with the braces require `egrep`, alternatively use `grep -E` . Run the examples and observe the results.

1. `egrep [88] sample.txt`
2. `egrep [8]{2} sample.txt`
3. `grep -E [8]{2} sample.txt`

8.1 Review Questions

1. The user enters the two commands at the `bash` command line:
   ```
   user@localhost:$ touch aaa cbd bbc bda cdd ab ad
   user@localhost:$ ls [a-d]?[!ac]
   ```
   What is the output of the above command?

   A. `bbc`

   B. `aaa cdd`

   C. `cbd cdd`

   D. `bda`

2. Write a regular expression to list files `ab` and `ad`.

3. Identify the command that will remove all files that begin with either `a`, `b` or `c` in the current directory?

   A. `rm [abc]*`

   B. `rm [abc]?`

   C. `rm [a][b][c]*`

   D. `rm [a][b][c]?`

9

Text Editor `vim`

`vi` has been the traditional character-based text editor developed by Bill Joy in 1976, released in 1978. `vim` is a contraction of **vi im**proved, it can be used in command line mode and as an graphical user interface. Released in 1991, `vim` is developed by Bram Moolenar , and available under GNU General Public License.

9.1 `.vimrc`

`.vimrc` is the default initialization file for `vim`. As a script writer you need to have the line numbers displayed. Edit `.vimrc` and insert the line `set number`. When debugging a bash script, the interpreter gives the line number before the error. In the `vim` editor you can jump to the line number by entering the number followed by `G`, this will take you directly to the line.

9.2 Copy-Paste, Cut-Paste

To copy (say) 4 lines, move the cursor to the line. To copy 4 lines to the buffer type in `4yy` in command mode. You should see a message `4 lines yanked`. This means the next 4 lines are now in the buffer. To paste the lines move your cursor to the line number you want the 4 lines to be pasted. In the command mode type in `P`, the lines will be pasted below the cursor, lower case `p` will paste the lines above the cursor.

Cut and paste is similar to copy, instead of `yy` type in `dd` for delete. The paste procedure remains the same.

9.3 swap file in `vim`

When you edit a file in `vim` for example, `vim mycalc.sh`, `vim` creates a hidden swap file with a name `.mycalc.sh.swp`. If you attempt to edit the same file in another Ubuntu terminal `vim` will give an error message `E325`. The same message will display when the Ubuntu session was not shutdown gracefully, or if VMWare crashed. Do not edit the same file on more than one terminal.

9.4 Review Questions

Short Answer Questions

1. You have made changes to a file in `vim`, you decide not to save the changes and quit. What command will you use?

2. Write the command to save changes and quit.

3. Write the command you will use to copy one line in the buffer.

4. Write the command you will use to copy 4 line in the buffer.

5. Write the command you will use to paste data from the buffer before cursors location.

6. Write the command you will use to paste data from the buffer after cursors location.

7. To run an external program in `vim` you should be in command mode. Write the command you will use to run the program.

8. Write the command to search for a string of characters in the forward direction, i.e. from the cursors position until the end of the file.

9. Write the command to search for a string of characters in the reverse direction, i.e. from the cursors position until the beginning of the file.

10. Write the command replace a character at the cursors location.

11. Write the command to delete the contents from the cursors position until the end of the line.

12. Write the command to delete the contents from the cursors position until the beginning of the line.

13. Write the command to delete the entire line at the cursor.

14. Write the command to undo all changes on a line.

15. Write the command to undo the last action.

16. How will you insert text before the cursor.

10

Search Files and Directories using `find`

In its simplest form the `find` utility searches for files. Tt can be used in conjunction with several other UNIX utilites to perform tasks on the file after it finds it. This section lists some exercises and examples that illustrate the use of the `find` utility.

Preparation: First, create a few files and directories. We will use the `/tmp` directory, but you can replace it with your `/home` directory. *Note:* Files in the `/tmp` directory are not guaranteed to be preserved. On your next Linux Boot, it is likely all your files will be erased. Use the `/tmp` directory to store files on a temporary basis.

You may create the files yourself as you proceed through the exercise, or use the script `find_exercise.sh` in the `find_exercise` directory to create the files.

```
mkdir -p /tmp/kingdom/{protista,plantae,animale}
```

Next we create a few files within these directories.

```
mkdir -p /tmp/kingdom/plantae/orchidacea/{oncidium,laelia,dendrobium,vanda}
```

Create files in the `vanda` directory.

```
 touch /tmp/kingdom/plantae/orchidacea/vanda/{aliceae,alpina,barnesii,furva,punctata}
```

Test the results with `tree -L 4 /tmp/kingdom`

Exercise 1. Find the file with a specific name.

Find the file `furva`

```
find /tmp -name "furva" 2>/dev/null
```

`2>/dev/null` will redirect the errors. Listing of directories with permission denied will not display, instead the output will be redirected to `/dev/null` Try the command with / instead of `/tmp`, it will take longer, because it has to search the entire filesystem.

Question: What does `-iname` do?

Hint: Try, `find /tmp -name "FURVA" 2>/dev/null`

and observe the result.

Exercise 2. Find the file using wildcards.

```
find /tmp -name "*icea*" 2>/dev/null
```

The file `aliceae` should display in the search result.

Exercise 3. Search for older and newer files.

Change the timestamp of file `alpina` to 11 December 1980, say more than 90 days.

```
touch -t 198012110000 /tmp/kingdom/plantae/orchidacea/vanda/alpina
```

Now find this file using `-mtime +90`,

```
find /tmp/kingdom -mtime +90 2>/dev/null
```

for recent files, say files modified less than say 10 days ago use,

```
find /tmp/kingdom/ -mtime -10 2>/dev/null
```

`alpina` should be missing in the list.

Exercise 4. Find directories.

```
find /tmp/kingdom/ -type d
```

Question: Write a command using `find` to display links.

Exercise 5. Find files with size as a search criterion.

Add some data to `barnesii`

```
date > /tmp/kingdom/plantae/orchidacea/vanda/barnesii
```

Search for files more than 10 characters in size.

```
find /tmp/kingdom -type f -size +10c
```

`barnesii` should display.

For files less than 10 characters use `find /tmp/kingdom -type f -size -10c`

We use `-type f`, otherwise directories will also show in the result.

Explore: A few other options to specify file size are `b,k,M,G`

Exercise 6. Search for files that contain a string.

Add data to a few files.

```
echo "blue" > /tmp/kingdom/plantae/orchidacea/vanda/aliceae
echo "red" > /tmp/kingdom/plantae/orchidacea/vanda/punctata
echo "green" > /tmp/kingdom/plantae/orchidacea/vanda/alipina
echo "blue" >> /tmp/kingdom/plantae/orchidacea/vanda/alipina
```

Find files that contain the string `blue`

`find /tmp/kingdom/ -type f -exec grep -Hi bl {} \;` the `-exec` option runs a command, in this case `grep`. In this example only a partial string `bl` is used. `-Hi` option prints the file name (`H`), and `i` will ignore case.

```
Try:  find /tmp/kingdom/ -type f -exec grep -Hi RED {} \;
```

Exercise 7. Moving files to another folder.

Create a directory `.recycle`

The following command will move files with zero bytes to the `.recycle` directory.

```
find ./kingdom/ -type f -size +0c 2>/dev/null -exec mv {} .recycle/ \;
```

Question: Write a command that will remove all files with zero bytes.

11

Variables and Parameters

This chapter discusses variables, their assignment in bash, reversing the assignment. Single quoting, i.e., strong quoting, and double quoting. Positional parameters and usage of the shift operator are covered in a separate section.

11.1 Variables and Parameters

A Variable is a *label* to data. It is a name of a location in memory that stores the data. A variable name is a reference to a memory location where data is stored. Data is numeric or a string. A string is an sequence of characters and numerals. Calculations are (usually) done on numeric variables.

Name and Value For example, a variable's name can be represented as `plant`. `plant` can be assigned a value, `orchid`. The data value `orchid` is *assigned* to `plant` using the notation.

`plant=orchid`

The assignment operator is =.

Aside: In bash, the assignment operator does two tasks, assignment and comparison. This is different from C language, the assignment operator is different from a comparison operator.

`echo plant` will display

`plant`

`echo $plant` will display

`orchid`

The $ before a variable refers to its value.

In the next examples a numeric value is assigned to a variables, `boiling` and `freezing`.

`boiling=100`

`freezing=0`

`echo $boiling` will display `100` `echo boiling` will display `boiling`

A variable can be declared and assigned at the same time, the previous examples show that. The reverse of declaring is `unset`.

`unset boiling` will release the variable `boiling`, the memory that stored the value `100` is now available to

store other data.
`unset freezing` will release the variable and its data value.
after releasing the variable `freezing`, `echo` will display a blank.
`echo $freezing`
Caution: `echo freezing` will display `freezing`. It treats the string `freezing` as a literal.
There are instances when the variable's value is referred to without using the $ symbol, this can be confusing. These instances are in the list below:

1. when used in a for loop, e.g., `for day in 1 3 5 7`
2. when using `unset`
3. when using `export`
4. when using the variable in double parenthesis

Declaring a variable in the above example does not use a space between the variable the assignment operator = and the value.
Using `boiling =100` will give an error.

The formal declaration The formal declaration of a variable is by using the
`let boiling=100`
without any spaces before and after the assignment operator.

Arithmetic operations Adding a numeric value of `10` to the variable `boiling` is done using
`let hot=$boiling+10`
Test it using `echo $hot`
It would be desirable to use `hot=$boiling+10`, but this assignment will assign the string `100+10` to `hot`.

Double Parenthesis allows spaces and makes declarations and arithmetic operations more readable. It also eliminates the need to use the $ sign to dereference variables.

Adding a value to the variable `boiling` is done using
`(( very_hot = boiling + 30 ))`
A new variable `very_hot` is declared and assigned a value `130`, i.e. `100 + 30`. Notice, the variable boiling within the double parenthesis is not preceded by the $ sign.

Full Quoting Using single quotation marks ' 'allows use of a variable without substituting its value. For example
`echo '$boiling'` will display `$boiling`
Using single quotation marks is also called *strong quoting*.

Partial Quoting Using double quotation marks " " allows the use of a variable by substituting its value. For example `echo "$boiling"` will display `100`. This is also called *weak quoting*.

Try the following two examples and observe the results:

```
echo 'boiling + $boiling'
echo "boiling + $boiling"
```

The first example will display the *literal* string `boiling + $boiling` Hence the term strong quoting for single quotation marks.

The second example will display `boiling = 100`, the double quotation marks allows substitution of the variables.

Finally try the command `echo "$boiling's"`. It will display `100's`

Double quoting preserves the spacing in string variables.

Assign the values shown below to a variable `four_days`, use two spaces between `Tue`, `Wed` and three spaces between `Wed`, `Thu` as shown

```
four_days="Mon Tue   Wed    Thu"
```

Type `echo $four_days` the spaces are removed.

Type `echo "$four_days"` the spaces are preserved.

11.2 Exporting Variables

Variables that are exported are visible to shell scripts that run in a spawned shell. Variables can be exported in atleast two ways:

1. using the `export` keyword. For example, `export CST8102="It is today, my favorite day"`
2. using the declare keyword. For example, `declare -x CST8102="It is today, my favorite day"`

A variable can be made local to the shell by using `declare +x CST8102`

Exercise:

Assign a variable `CST8102` to a value of your choice. The run the script in 11.1 in two separate modes. As a *sourced* script as in `source ./check_export.sh` and then as a subshell `./check_export.sh`. Observe the difference. Now remove the export attribute on the variable using `declare +x CST8102` and rerun the script in the two modes. Observe the difference.

```
#!/bin/bash
# Filename : check_export.sh
# Date     : 6 Dec 2022
# check status of exprt variables.

echo "Content of exported variable is: $CST8102"
echo "PID is : $$"
echo "Parent PID is : $PPID"

# eof : check_export.sh
```

Figure 11.1: Demonstrating Exported Variables

11.3 Positional Parameters

Refer Slides on Positional Parameters

11.4 Exercise - Path Name and Environment Variable

Reference Refer usage of the following command and utilities from your reference book: `PATH`, `export`, `paste` and `tar`.

Objective

1. Use environment variables to evaluate pathname.
2. Check directory and file before executing the command.
3. Write an install script and uninstall script.

Submission Show your work to the lab instructor during lab hours. Upload your script, datafiles and result to LMS.

Background Executables and data reside in different directories. This segregation of executables and data files facilitates maintenance and debugging. The objective is to start the script from the command prompt, let it evaluate the paths from environment variables before executing the binary. A basic shell script `merge.sh` is provided with two data files. You may study the script, but do not edit it. Two data files are `CourseCode.dat` and `CourseName.dat`. You may add data to these file, but do not remove existing data. The script merges each line from the two data files and displays the merged output.

Requirements Copy all three files `merge.sh`, `CourseCode.dat` and `CourseName.dat` to your home directory. Write an install script, call it `install.sh` that will perform the following tasks.

1. Create and export two environment variables called `BINPATH` and `DATAPATH`. For example, you may call `BINPATH` as `/home/user1/bin` and `DATAPATH` as `/home/user1/dat`.
2. Create the two directories.
3. Move the script `merge.sh` to `BINPATH` *using* the environment variable.
4. Move the two datafiles to `DATAPATH` *using* the environment variable.

After the installation, the file `merge.sh` should run from any directory, say your home directory `/home/user1`.

Uninstalling. Write an script to uninstall the binary and data files. Call the script `uninstall.sh`. Delete the script `merge.sh` and the two datafiles using the environment variables. Remove the two directories using the environment variable. Remove the environment variable using the `unset` command.

Error Reporting Before the script is executed check if directories and files exist. If they already exist do not create them. Trap the condition and exit the install program. All error messages should be defined by you, the user should not see any errors by the `bash` shell.

Appendix

```
#!/bin/bash
error_code=1
while [ ! ]
do
  if [ -f fileX.txt ]
     then
     echo "Attempting to copy..."
     cp fileX.txt fileY.txt > /dev/null
     error_code=$?
  else
     error_code=7
  fi

  if [ $error_code -eq 0 ]
  then
     echo "Copy successful"
  elif [ $error_code -ne 0 ]
     then
     echo "Copy failed"
     exit $error_code
  fi
  sleep 1
  break
done
```

Figure 11.2: Filename: cp1.txt

The code above attempts to clarify the error in slide 12. It is not a production quality script, but it attempts to answer a few questions during Monday's theory class. Some comments are written below. Send me any further questions you have.

1. Line 3 `while [ ! ]` is an infinite loop. Empty square brackets will result in a false, negating this will return true, this creates the condition for an infinite loop. The exclamation sign negates the condition. This loop needs a break statement, as shown in line 23.
2. Line 5 tests if the source file `fileX.txt` exists. It then copies `fileX.txt fileY.txt`
3. The error code is set to 7 if the source file does not exist; 7 is an arbitary number chosen. If the program exists without copying the file, the exit code will be 7.
4. Why do we need error codes. Think about a diagnostic tool for a car. The value it provides for different diagnostic tests with a code. Similarly each script should have a series of possible errors and corresponding error messages logged in a file for diagnostics. Currently our scripts are small with a few specific tasks. The error codes are shown for demonstration purposes.
5. The script is titles `cp1.txt`, it is available on One Drive at `Scripts > Misc-00`
6. Runt the script with `fileX.txt` created and then without `fileX.txt` and observe the difference.
7. Check the exit code with `echo $?` when the script exits with an error.

11.5 Change `hostname`

`hostname` is stored at two locations, in `/etc/hostname` and `/etc/hosts`. Edit these files and reboot for the new `hostname` to take effect. Alternatively, use the stream editor `sed` to change the files. Two `sed` commands are shown below to change the `hostname`.

```
sudo sed -i 's/old_name/new_name/g' /etc/hostname
sudo sed -i 's/old_name/new_name/g' /etc/hosts
shutdown -r now
```

The new prompt should display the new `hostname`, the environment variable `HOSTNAME` will also show the change. Confirm by typing `echo $HOSTNAME`.

Aside: The command `hostname` will also change the hostname temporarily, i.e., only for the current session. To change the name in the OS you need to edit the two files listed above.

Bibliography

[1] Stallings, William, Operating Systems, Internals and Design Principles, 9e, Pearson Education Inc., 2018, 978-0-13-467095-9

[2] Open Source, Various Authors, Bash Reference Manual, http://www.gnu.org/software/bash, 2020

[3] Blackberry Limited, Blackberry Limited, Ultimate Guide RTOS, Blackberry Limited, 2020

[4] Joseph, E. K. et al, Common OpenStack Deployment, Real World Examples for Systems Administrators and Engineers, Prentice Hall, 2017, 978-0-13-408623-1

[5] John Muster, Introduction to Linux and Linux, McGraw Hill/Osborne, 2003, 0-07-222695-1

[6] The Linux Information Project, `http://www.linfo.org/superblock`

[7] The `fsck` man pages.

[8] Eaton C, et al, Understanding Big Data, McGraw Hill, 2012, 978-0-07-179053-6

[9] Lamb L, Learning the vi editor, O'Reilly & Associates, 1992, 0-937175-67-6

Glossary

bus A path that transfers data, address and control signals in a computer system. 14

computer A device with a Central Processing Unit (CPU), memory and input and output (I/O) components. 13

Coordinated Universal Time (UTC) A time standard for regulating time. UTC is not adjusted for Daylight Savings Time. UTC replaces Greenwich Mean Time (GMT) 14

data striping Data, such as a file is divided into logical parts, each part is stored in a separate physical disk. Data Striping is a storage technique. Compare it to a file stored sequentially on a single physical device. 66

file A linear arrangement of binary data stored in memory with a name. 16

hadoop A collection of open source utilities that attempts to solve problems that require large data and large computing resources. Hadoop handles hardware failures automatically. A concept developed by the Apache Group. 15

Parse To resolve (split) a statement into its smaller components with the intention to analyse, interpret or describe it. 123

POSIX Portable Operating System Interface (POSIX) standards defined by IEEE Computer Society to maintain compatibility between operating systems. POSIX defines API's, shells and interfaces for utilities. 66

protocol Rules that define data transmission and reception across a network. A protocol must have at least two entities. 65

SMP Symmetrical Multi Processor (SMP) A computer architecture that uses more than one identical processor. All processors can access the main memory, have access to I/O devices and are managed by a single operating system. 14

special character A character that is not a alphbetic character, not a number but a character that is used to denote a meaning to characters and numerals. Examples of special characters in UNIX are square brackets, braces, exclamation mark. Wildcards are subsets of special characters. A special character cannot directly represent itself; a special character must be preceeded with a backslash to represent itself. 129

SSD Solid State Device. A non-volative data storage mechanism. Unlike a magnetic disk, which is also non volatile storage, a SSD has no moving parts. 16

supercomputer A large computer reaching peta FLOP performance. Compare this to a general purpose (desktop) computer with a performance in the giga FLOP range. The worlds top 500 supercomputers run Linux. 66

swap The transfer of data from main memory to secondary memory. 16

swap space Storage space used to move data from main memory (RAM) to (slower) secondary storage. Secondary storage could be magnetic storage or SSD. 16

wildcard a character that is used to represent one or more character. Examples of a wildcard characters are `*` and `?`. In regex a period `.` is treated as a wildcard. The use of a wildcard is specific to the environment and the utility. Generally, the asterisk and the question mark are represented as wildcards. 129

Index