0. Contents

0. List of Tables

0. List of Figures

1. Linux SysAdmin Course

1.1. STD O I ERR

Everything in linux is treated as ordinary files. This includes data streams. Each file gets a file-descriptor assigned to it that can then be used to acces the particular type of data stream that it is carrying. A file-descriptor is basically in integer. When executing a command on linux we will mainly be concerned with the following three tupes of file descriptors :

STDIN : Standard input has file decriptor
STDOUT : Standard output has file decriptor
STDERR : Standard error has file decriptor

every process that is running in the linux kernel has three channels , which are STDIN , STDOUT and STDERR . Each one of these channels have numbers that correspond to them. If we want to redirect ouputs of certain processes into others.

STD IN / OUT / ERR ?

FIGURE 1.1.

echo “this should be in a file” 1 > somefile.txt

output redirection uses STDOUT by default so we dont even need the 1 in the above command.

echo “this should be in a file” > somefile.txt

this will however overwrite everything in the original file. so instead of overwrite , we append using >>

echo “this should be in a file” >> somefile.txt

similarly to print out error , we use the filedescriptor 2

ls thisdoesnotexist 2 > somefile.txt

to redirect input we use the <symbol

1.2. Package Management

1.2.1. pacman

1.2.2. AUR

1.2.3. Source Installation

1.3. GREP : *Global Regular Expression Print*

Grep is a pattern matching tool. It basically returns things based on your set conditions from a large ouput. Grep is most often used in conjuction with other file manipulation commands such as less , cut , cat , sort ... in order to reduce the information that we as a human need to parse through to get at what we really wanted.

Grep uses POSIX regular expressions in evaluation. This is different from PERL compatible regular expressions that are used in python and some other programming languages. So dont get that confused.

Grep use examples and flags are shown below :

grep John notes.txt
basic usecase

grep “abc pqr” *filename*
use double quotes to search for phrases with spaces

grep -w *phrase* *filename*
exact matches

grep -i *phrase* *filename*
non case sensitive matches

grep -n *phrase* *filename*
display line numbers

grep -B 4 *phrase* *filename*
display lines **before** matched word

grep -A 4 *phrase* *filename*
display lines **after** matched word

grep -win -C 4 *phrase* *filename*
display lines **context** of word **(before and after)**

grep -r *phrase* *filename*
search for phrase recursively in all subdirectories

grep -l *phrase* *filename*
display files that contain match

grep -c *phrase* *filename*
return count of the number of lines (not times) the phrase occurs in the specified files

grep -v *phrase* *filename*
Do an inverse match , i.e. return all lines that do not contain the given phrase

grep *phrase* ./*
search all files (*) in current dir (./)

grep *phrase* ./*.txt
search all .txt files (*.txt) in current dir (./)

1.4. File Permissions

Each file in linux has 9 permission bits.The first 9 bits in a file are used to describe the kinds of permissions that are allowed for a file. Traditionally there are three types of permission bits :

- **r** gives permission to read the file
- **w** gives permission to write the file , this lets you edit a file , move a file , rename a file and a bunch of other stuff.
- **x** gives permission to execute the file
- **-** means the file has no permission

The nine bits can be grouped into threes since they specify *rwx* permissions for the owner , the next three for the owners group , the last three are for everyone else respectively.

Usually when you are printing with a command like ls with the -l flag , then we see an additional 10th bit at the front of the permission bits. This bit sets the entry type, and describes the type of the file we are dealing with. The most common types you might see the first bit being are :

- **-** means it is a file
- **d** means it is a directory
- **l** means it is a link
- **b** means it is a block device type (hard disk)

Permissions		Owner	Owner Group	PID	Filename
-rw-r–r–	1	ishan	users	11707	Level.uno
drwxr-xr-x	15	ishan	users	4096	Documents

The permissions are goverened by the command *chmod*

The rwx permission also have numerical representations that are sometime used. These also follow the same orders mentioned above , i.e. , the first numerical representation will be for the user account. The second for the user group and the third for the everyone else.

These numbers are :

- 7 : r w x
- 6 : r w -
- 5 : r - x
- 4 : r - -
- 3 : - w x
- 2 : - w -
- 1 : - - x
- 0 : - - -

If you dont want to go around manually editing file permissions then edit (very very carefully) the file : /etc/login.defs in debian based systems.

1.5. File Access Control

Everything in linux is an object.

Permissions of a process are goverened by the user that started the process.

Usually these permissions are set-up not on an individual user basis , but rather at a user group basis. (you might have heard of the WHEEL group). In a production or company environment these are groups like Human Resources group , the managers group etc .... One user can be part of multiple groups which allows for easy modularity between user privilege levels.

The biggest baddest user is called root.

1.6. Root

Root in linux can mean one of three things :

1. Root User
2. Root Folder
3. Root Home

Root User

The root user is the most powerful account user , and is also known as a ***superuser***. There are many commands such as installing new packages, or mounting / unmounting filesystems that require this privelege (sudo -i -s , sudo su , sudo do).

Root Directory

The root diretory is the place where the linux filesystem starts. It is the directory that is indicated by only a / . The root directory contains all folders in the linux sub directory.

1.7. Users

A user is the person who is currently using the machine. You can see who you are in many ways. First you can just open up the shell and look at the prompt. It will look something like *username@machinename:* . You can also type in the *who* and *w* commands. If you want more information than that about just your current user account then you can see all current users by catting the file **/etc/passwd** , or the file **/etc/shadow** .

In the **/etc/shadows** file you might see two anomailes in the password field : * and ! . The root user has password status ! which means that they can only be accessed through sudo. And all processes acting as users have no ability to log in whatsoever. This is specified by *. All other passwords should look like hashes. If a user has been created , but thier password has not been set yet, then they will also have ! in the password field in this file.

To list user groups instead of just the users , we can cat : **/etc/group**

These files also contains services that are running. Therefore you will see a lot more ‘ Users ’ than you might have on the machine. This is because a lot of services also run as users within a linux machine. These services are listed here albeit with much fewer permission than those that a normal human user would have.

The feilds in **/etc/passwd** will look something like :

Username : x : USER ID : GROUP ID : Human Readable Info : Home Dir : Default Shell

The feilds in **/etc/group** will look something like :

Group Name : x : GROUP ID : Members of that Group

Adding a user

To add a user we use the *useradd* command , with the following flags

- -m create home directory
- -d /home/username
- -u create user id
- -g define group id
- -s define default shell

We can also create a bunch of new users simultaneously using the *newusers* command. After creating the user , use the *passwd* command to set the password for the user. This will remove the locked ! status from /etc/shadows mentioned earlier. If you want to relock the user then we can use the *usermod -L username* command. And to unlock him/her user *usermod -U username* .

Deleting a user

Finally use *userdel username* to delete the user. After deleting the user remember to find and clean up their home directory or anything else thay they have done.

NOTE

Emails originated from the ishan@linuxmachine sort of format , where the username would follow the machine name and that is the convention that people were used to in servers back in like the 70s.

1.8. Filesystem

So when we are storing data it is essentially stored in binary bits (0 and 1). But that is essentially just one large body of text in some storage device and there is no way for us or the computer tell when the bits from one piece of data end and where the bits of another begin. We need some sort of pre agreed on system , to say : “ *from ‘here’ to ‘there’ is one thing, and from ‘there’ to ‘another place’ is another thing* ”.

By separating the data into pieces and giving each piece a name, the data is easily isolated and identified.

Since most of the naming conventions came about when paper-based management systems and filing cabinets were still things that existed , each little group of data started being called a “file”.

And since we are still dealing with old timer terminology , what do you do with files ? You name them and organize them and put them into filing cabinets according to some sort of system. Therefore the way these are named , sorted and organized got called a file system.

So one thing that we know is that not everyone agrees on the logic of how things should be organized. Some people organize clothes by color , others by season , others by occasion and so on. Similary there are various different ways to organize and categorize the files inside our storage devices, which means there are a bunch of differnt filesystems that we can use. The most common ones are :

- FAT32
- EXT3
- EXT4
- NTFS
- ZFS
- HFS

There are also certain filesystems that are specific to certain mediums. Such as the ISO 9660 filsystem which is only for CDs.

Its been said before , and I will say it again. Everything in linux is a process and every process is manfisted as a file, so everything in linux is essentially just a file. Which means that files and filesystems are very important in understanding and dealing with any information and information manipulation inside Linux.

The filesystems inside linux can be physical / real filesystems that the machine is dealing with , or they can be virtual filesystems that the kernel / system creates to make things easier to deal with. An example is the /proc filesystem that deals with the storage and processing of inter-process signals.

1.8.1. Logical File System

The logical file system is responsible for interaction with the user application. It provides the application program interface (API) for file operations — OPEN, CLOSE, READ, etc., and passes the requested operation to the layer below it for processing. The logical file system "manage[s] open file table entries and per-process file descriptors."[8] This layer provides "file access, directory operations, [and] security and protection."[7]

1.8.2. Virtual Filesystem

The second optional layer is the virtual file system. "This interface allows support for multiple concurrent instances of physical file systems, each of which is called a file system implementation."[8]

1.8.3. Physical Filesystem

The third layer is the physical file system. This layer is concerned with the physical operation of the storage device (e.g. disk). It processes physical blocks being read or written. It handles buffering and memory management and is responsible for the physical placement of blocks in specific locations on the storage medium. The physical file system interacts with the device drivers or with the channel to drive the storage device.[7]

tmpfs

Mounting

udevd Daemon

1.9. Process

A process ia basically an application , shell command or any program that is "doing something" in linux. It can be as complex as running a browser , running a vulnerability scanner , or as simple as running a shell command or read a file. All of this will spawn a process.

A process is given its own share of memory do go about its business (it has its own call stack and memory allocated for a heap). This is usually virtual memory , so it might not be able to see absolute address that it is reading or writing to. Instead it is given virtual addresses by the kernel.

Kernel

Basically , a kernel is the thing that is sitting between a running process and the physical hardware. The kernel also keeps track of

- **Owner** : Which user is the owner of the process
- **Parent Process** : Which process spawned the current process , since all processes are born from other processes.
- **Priority** : Does it need to do work quickly (high priority) or can it chill for a while (low priority).
- **Memory Allocation** : the address spaces (files , network ports . . .) that the process is allowed to use and what it is currently using.

Each process has an ID , called the ***Process ID*** or ***PID***. This is a unique number assigned to each process. The first process (PID 1) is always *init*. It is the parent that gives birth to all other processes on the system. The kernel spawns this process during boot , and it runs all the startup scripts that setup linux. If any process dies , then all the children that it spawns are not killed , rather they are adopted by init.

What init specifically does differs according to some distributions, and is outside the scope of this text. Just remember it is the first process and it is spawned by the kernel.

NOTE

There are some virtual machines that think that they are real machines. Which means that they will have thier own *init* with PID 1, while the host machine also has a similar *init* with PID 1.

There are two kinds of process permissions , namely id based permissions or effective id based permissions. More formally UID (user id) define permissions based on user privilege level of the user that spawns the process. On the other hand , the EUID (effective user id) , gives permissions more sparingly (or just differently) based upon a foundation of the original permissions.

A good example why this is needed is if root spawns a process , and this process goes crazy , we dont want it walking around in our system doing whatever it wants with root privileges , because that pretty much means deth. So EUID gives permissions different than the ones granted by user spawn.

Similarly there are also group ids GID and effective group ids EGIDs.

1.9.1. Signals

Signals are what processes use to communicate with each other. You might also already know them as software interrupts from your operating systems class. Signals can be sent by the kernel as well when a process does something heinous like dividing by or zero, but they are more commonly the kernel uses signals to indicate the state of the system to the processes. Some

common signals are notifying the parent of the death of the child process, or notifying a process about the availability and readiness of some hardware like a CD-drive.

As has been mentioned before everything in linux is a file. So what a signal essentially is read and writing from files. A process has something to say to another process so it will write to a file, and the other process will read fomr this file. This 'process' of reading and writing is called a signal.

The following are examples of signals : [i]

SIG **HUP**	Hang up
SIG **INIT**	interrupt
SIG **QUIT**	quit
SIG **ILL**	illegal instruction
SIG **TRAP**	trace trap
SIG **ABRT**	abort process
SIG **BUS**	bus error
SIG **FPE**	floating point exception
SIG **KILL**	kill (cannot be ignored)
SIG **USR1**	custom user defined signal
SIG **SEGV**	Segmentation Fault / Violation
SIG **USR2**	custom user defined signal
SIG **PIPE**	Write to pipe fault
SIG **ALRM**	Alarm
SIG **TERM**	Terminate (can be ignored by process)
SIG **STKFLT**	Stack fault
SIG **CHLD**	Child process terminited / interrupted / resumed.
SIG **CONT**	Continue executing after stop
SIG **STOP**	Stop a process for later
SIG **TSTP**	Temporaily stop
SIG **TTIN**	tty in
SIG **TTOU**	tty out
SIG **URG**	urgent data aviable for read
SIG **XCPU**	You have overused your CPU time. This is a warning to gracefully exit before SIGKILL shows up to b
SIG **XFSZ**	X-Filesize , Grew file larger than maximum allowed size.
SIG **VTALRM**	Virtual alarm clock
SIG **PROF**	Profling alarm clock
SIG **WINCH**	window change , or when terminal / process manager changes size
SIG **IO**	Input Output now possible
SIG **PWR**	Power failure detected
SIG **LOST**	Power lost , synonym for SIGPWR
SIG **UNUSED**	Unused signal

Sending signals from the keyboard

Signals may be sent from the keyboard. Several standard defaults are listed below. Default key combinations for sending interrupt signals can be defined with the stty command.

Ctrl-C	Send SIG INT (Interrupt). By default, this causes a process to terminate.
Ctrl-Z	Send SIG TSTP (Suspend). By default, this causes a process to suspend all operation.
Ctrl-	Send SIG QUIT (Quit). By default, this causes a process to terminate immediately and dump the core.
Ctrl-T	Send SIG INFO (Info). By default, this causes the operating system to display information about the command. Not supported on all s

To send a signal we can also use the signal number assigned to each signal as opposed to typing / remebering the whole signal name. Similarly for the processes themselves. Although getting the process ID will usually require a grep. As examples :

kill 15 firefox	:	this will send SIGTERM to firefox
ps aux |grep firefox	:	Greps for the named process [ii]
kill 9 3460	:	this will send SIGKILL to process num

1.9.2. Process : States

Everything on the machine is basically some combination of calculations that need to get done, in order to get some answer. This answer is the raison d'être [iii] for a process , and to calculate it they all need CPU time. However not every process needs CPU time at every given second. Which is why we have scheduling algorithms like round robin. So for example if a process is waiting for keyboard input , or for something to be read from the hard drive or CD-Drive then it doesnt need CPU time because it can't really do much (or anything) with it anyway. So the processes can exist in the following states :

- **Runnable** : The process has all the information and input it needs , and it can just get run (or be scheduled to run) the next time the CPU has any free time.
- **Sleeping** : It can be waiting for something.
- **Zombie State** : It has finished doing whatever it needed to do , but now it is just waiting to give back the information and then be killed.
- **Stopped State** : A process that may have been in the middle of doing something but the user or another process sent it SIGSTOP so it is waiting for SIGCONT to resume where it left off.

The processes also have a ***niceness property***, which basically details how much they allows other processes to run , or does this process hog system resources or not. This is goverened by it's priority level, so the higher the priority the more of a dick the process is going to be about sharing system resources.

Linux processes have niceness ranger from -20 to 19 , where the higher the number the lower the priority. So if something is -20 nice then it is ultra critical and it needs to get run before everything else. This will also probably cause your screen freeze because as much CPU time as possible is going to this process , and the CPU wont even have enough time to refresh your screen.

1.9.3. /proc Filesystem

One of the reasons Linux is really popular as a server side operating system is that we (specifically System Admnisistrators) can change the underlying values of

[i] man page is : man 7 signal

[iii] reason or justification for existence

the operating systen while still keeping the server and the system up and running.

Since we know that all actions on linux run as an underlying process , and we also know that all things in linux are essentially files. This means that all procsses are also represented as a file 'somewhere' on the system.

This 'somewhere' is the **/proc** folder. This folder is a virtual filsystem which can be accessed by the system administrarors to view and change underlying system values.

A /proc filesystem is not a real filsystem because it resides only in the computer's memory and does not use and space on the hard disk (i.e. it is a virtual filesystem). This filesystem is a map to the running kernel processes.

It is mounted in /proc during system initialization and has an entry in the /etc/fstab file[iv]

If you ls /proc you will see a bunch of subdirs and files. Most of these have numbers , and these numbers are actually PIDs for running processes. Other directories deal with system hardware, networking settings , activity and memory usage and so on.

If you list the files under /proc, you will find that all the files have a size of 0 - this is because they are not really files and directories in the typical sense. Basically these files are beaing created and updated on the fly. This also means that if you want to 'see' one of these files you can't just cat them.

NOTE

WARNING

Do not use 'cat' on /proc/kcore as this is a special file which is an image of the running kernel's memory at that particular moment - cat'ting this file will leave your terminal unusable.

Another thing to keep in mind is that a lot of the information contained in these files is either raw data , and sometimes even just binary. So looking through these files for useful information is not always easy. This information if often meant to parsed and interpreted through other programs , but it is still good to know where the programs like top and htop are getting their information from, and if you are a madlad read it directly from source.

Within each process's (es) folder there will be some common files that will be of interest :

- **cmd** : Contains the command that the process is currently executing
- **cmdline** : Contains information about how the process was called , i.e. the command line argument that was inputted to run the current process
- **cwd** : Contains information about the directory within which the process is opertaing from
- **environ** : shows the environment variables
- **exe** : a link to the executable

Some of the key files in the top-level directory are as follows :

- **/proc/interrupts** - View IRQ settings
- **/proc/cpuinfo** - Information about the system's CPU(s)
- **/proc/dma** - Direct Memory Access (DMA) settings
- **/proc/ioports** - I/O settings.
- **/proc/meminfo** - Information on available memory, free memory, swap, cache memory and buffers. You can also get the same information using the utilities free and vmstat.
- **/proc/loadavg** - System load average
- **/proc/uptime** - system uptime and idle time. Can also be obtained using utility uptime.
- **/proc/version** - Linux kernel version, build host, build date etc. Can also be obtained by executing 'uname -a'.

Beneath the top-level /proc directory are a number of important subdirectories containing files with useful information. These include :

- **/proc/scsi** - Gives information about SCSI devices
- **/proc/ide** - information about IDE devices
- **/proc/net** - information about network activity and configuration
- **/proc/sys** - Kernel configuration parameters. The values in files in this directory are editable by root, which I will further explain below.
- **/proc/** - information about process PID.

[iv] fstab file is the operating systems system table

Commands Overview

TABLE 1.1.

Navigation & Creation & Destruction

clear	:	clear the terminal
pwd	:	Print working Directory
ls	:	list everything in current dir
	:	can also list everything the dir if path is specified
cd	:	change directory
touch	:	creates an empty file
mkdir	:	makes a directory inside current directory with given name
rmdir	:	removes directory inside current directory with given name , but only works if specified dir is empty
rm	:	remove (delete) the file , will also work with directories , even if they are non empty
mv	:	move the file to the specified destination (Regex works). Is also used to rename files. Just move the file into the same folder with a different name

Manipulation

cat	:	prints out everything in the file
grep	:	Allows STDOUT manipulation , see (§ 1.3)
less	:	allows pagination of STDOUT
cut	:	remove section from STDOUT based on args
sort	:	sort the output produced on STDOUT
wc	:	gives char / byte / newline count of STDOUT
uniq	:	removed repeated lines from STDOUT
tail	:	print the last 10 lines of each file to STDOUT
wc	:	print the number of lines / character / bytes in a file depending on args
|	:	pipes all STDOUT from the first command into STDIN for second command
&&	:	runs command 2 iff command 1 runs succesfully ,i.e. returns true

System

man	:	shows manual for command , as well as all avialable flags
ps	:	lists all the running processes
who	:	shows list of logged on users
w	:	shows list of logged on users and their processes
.	:	indicates current working directory
..	:	indicates parent of current working directory
find	:	
locate	:	
awk	:	
ack	:	
rename	:	rename files

Users

useradd	:	Add a new user
userdel	:	Delete an existing user
usermod	:	Modify existing user somehow
newusers	:	create multiple users simultaneously
passwd	:	set the password for a user

Commands Overview 2

TABLE 1.2.

Process & Signals		
kill	:	tries to terminate a process , default SIGTERM (15)
kill 9	:	sends SIGKILL to brutally murder a process where it stands
killall	:	SIGTERM all processes with the specified name
pkill	:	kill all processes based on name or other attribute
pgrep	:	kill all processes based on name or other attribute
pkill -u	:	kill all processes for a certain user
nice	:	run a process with specified niceness value
renice	:	Change the niceness of a running process [a]
strace	:	gives a detailed view of system processes
Network		
ifconfig	:	

[a] be very very careful here. If you give a process too high importance you might not even be able to kill it since your kill signal will get less CPU time than the process itself. It might even take minutes just to renice the thing again to be able to kill it. Its just not a good time.

-§-

2. Standards

-§-

3. Conventions

All file names are case sensitive. So filename vivek.txt Vivek.txt VIVEK.txt all are three different files.

You can use upper and lowercase letters, numbers, “.” (dot), and “_” (underscore) symbols.

You can use other special characters such as blank space, but they are hard to use and it is better to avoid them.

In short, filenames may contain any character except / (root directory), which is reserved as the separator between files and directories in a pathname. You cannot use the null character.

No need to use . (dot) in a filename. Some time dot improves readability of filenames. And you can use dot based filename extension to identify file. For example:

.sh = Shell file

.tar.gz = Compressed archive

Most modern Linux and UNIX limit filename to 255 characters (255 bytes). However, some older version of UNIX system limits filenames to 14 characters only.

A filename must be unique inside its directory. For example, inside /home/vivek directory you cannot create a demo.txt file and demo.txt directory name. However, other directory may have files with the same names. For example, you can create demo.txt directory in /tmp.

Linux / UNIX: Reserved Characters And Words

Avoid using the following characters from appearing in file names:

Please note that Linux and UNIX allows white spaces, ¡, ¿, —, :, (,), &, ;, as well as wildcards such as ? and *, to be quoted or escaped using symbol.

. is used to separate a filetype extension, e.g. foo.txt.

- or _ is used to separate logical words, e.g. my-big-file.txt or sometimes my_big_file.txt. - is better because you don’t have to press the Shift key (at least with a standard US English PC keyboard), others prefer _ because it looks more like a space.

So if I understand your example, backup-part2-random or backup_part2_random would be closest to the normal Unix convention.

CamelCase is normally not used on Linux/Unix systems. Have a look at file names in /bin and /usr/bin. CamelCase is the exception rather than the rule on Unix and Linux systems.

(NetworkManager is the only example I can think of that uses CamelCase, and it was written by a Mac developer. Many have complained about this choice of name. On Ubuntu, they have actually renamed the script to network-manager.)

Absolute vs. Relative Paths

Absolute path means full true path to get to the thing you want to get to. This means that all aboslute paths must start with / i.e. they must all start in the root directory. So the absolute path to get to file in your Documents directory would be :

/home/username/Documents/

Relative paths are based on where you are currently in the system. As an example from your root directory if you cd into Documents, and then run ls. The ls command will be run relative to where you are and will therefore show the contents of the folder that you are in ,i.e. of Documents. Which means to get to the same file in the Documents directory as in the absolute path explanation , the path is now (if we are sitting on home) :

Documents/

Generally you want to be using Absolute paths as much as possible. This avoids confusion and works on all linux distributions. The reason I say this is because there are some commands (like ifconfig) that work based on relative paths in some distributions like Ubuntu , but on others like Debian they will not work unless the full absolute path is specified , i.e. , /sbin/ifconfig , which would work on both systems.

There are also security reasons to use absolute paths since whenever you use relative paths to access / run some software some other process / program has to deal with figuring out the absolute path to it and the returning the relevant information. The program responsible for this translation might return the correct information or run the right software but it also has the ability to store any information that was passed as arguments to the command we are running since it is a literal man in the middle on your system.

-§-

4. References