./

GENESIS – Basic Linux and OS Programming Learning Report



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **To be Approved** | **Remarks/Revision Details** |
| V1.1 | 10/01/2021 | Shriram M S |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Details**

Contents

[Contents 3](#_Toc61211765)

[1 Basic Linux 4](#_Toc61211766)

[1.1 System 4](#_Toc61211767)

[1.2 Search 4](#_Toc61211768)

[1.3 File Commands 4](#_Toc61211769)

[1.4 Process Related 5](#_Toc61211770)

[1.5 File Permission 5](#_Toc61211771)

[1.6 Shortcuts 5](#_Toc61211772)

[2 Assignments 5](#_Toc61211773)

[2.1 BASIC COMMANDS: 5](#_Toc61211774)

[2.2 COMMANDS AND ARGUMENTS: 7](#_Toc61211775)

[2.3 CONTROL OPERANDS: 9](#_Toc61211776)

[2.4 FILE LINKS: 10](#_Toc61211777)

[2.5 STANDARD FILE PERMISSIONS: 11](#_Toc61211778)

[2.6 FILE SYSTEM: 13](#_Toc61211779)

[2.7 FILTERS: 14](#_Toc61211780)

[2.8 SHELL PATTERN MATCHING: 15](#_Toc61211781)

[2.9 SHELL HISTORY: 17](#_Toc61211782)

[2.10 SHELL VARIABLES: 18](#_Toc61211783)

[2.11 WORKING WITH DIRECTORIES: 19](#_Toc61211784)

[2.12 WORKING WITH FILES: 21](#_Toc61211785)

[2.13 WORKING WITH FILE CONTENTS: 22](#_Toc61211786)

[3 OS Programming Concepts 23](#_Toc61211787)

[3.1 SYSTEM CALLS 23](#_Toc61211788)

[3.2 PROCESSES 24](#_Toc61211789)

[3.3 THREADS 24](#_Toc61211790)

[3.4 Semaphore 25](#_Toc61211791)

[3.5 Message Queue 26](#_Toc61211792)

[3.6 Pipes 26](#_Toc61211793)

[3.7 Shared Memory 27](#_Toc61211794)

# Basic Linux

## System

* uname => Displays Linux system information
* uname -r => Displays kernel release information
* uptime => Displays how long the system has been running including load average
* hostname => Shows the system hostname
* hostname -i => Displays the IP address of the system
* last reboot => Shows system reboot history
* date => Displays current system date and time
* timedatectl => Query and change the System clock
* cal => Displays the current calendar month and day
* w => Displays currently logged in users in the system
* whoami => Displays who you are logged in as
* finger username => Displays information about the user

## Search

grep ‘pattern’ files => Search for a given pattern in files

grep -r pattern dir => Search recursively for a pattern in a given directory

locate file => Find all instances of the file

find /home/ -name “index” => Find file names that begin with ‘index’ in /home folder

find /home –size +10000k => Find files greater than 10000k in the home folder

## File Commands

ls -al => Lists files - both regular & hidden files and their permissions as well.

pwd => Displays the current directory file path

mkdir ‘directory\_name’ => Creates a new directory

rm file\_name => Removes a file

rm -f filename => Forcefully removes a file

rm -r directory\_name => Removes a directory recursively

rm -rf directory\_name => Removes a directory forcefully and recursively

cp file1 file2 => Copies the contents of file1 to file2

cp -r dir1 dir2 => Recursively Copies dir1 to dir2. dir2 is created if it does not exist

mv file1 file2 => Renames file1 to file2

ln -s /path/to/file\_name link\_name => Creates a symbolic link to file\_name

touch file\_name => Creates a new file

cat > file\_name => Places standard input into a file

more file\_name => Outputs the contents of a file

head file\_name => Displays the first 10 lines of a file

tail file\_name => Displays the last 10 lines of a file

gpg -c file\_name => Encrypts a file

gpg file\_name.gpg => Decrypts a file

wc => Prints the number of bytes, words and lines in a file

xargs => Executes commands from standard input

## Process Related

* ps => Display currently active processes
* ps aux | grep ‘telnet’ => Searches for the id of the process ‘telnet’
* pmap => Displays memory map of processes
* top => Displays all running processes
* kill pid => Terminates process with a given pid
* killall proc => Kills / Terminates all processes named proc
* pkill process-name => Sends a signal to a process with its name
* pgrep firefox => Find Firefox process ID
* pstree => Visualizing processes in tree model

## File Permission

* chmod 777 /data/test.c => Set rwx permissions to owner, group and everyone
* chmod 755 /data/test.c => Set rwx to the owner and r\_x to group and everyone
* chmod 766 /data/test.c => Sets rwx for owner, rw for group and everyone

## Shortcuts

* ctrl+c =>Halts the current command
* ctrl+z =>Stops the current command, resume with fg in the foreground or bg in the background
* ctrl+d =>Logout the current session, similar to exit
* ctrl+w =>Erases one word in the current line
* ctrl+u =>Erases the whole line
* ctrl+r =>Type to bring up a recent command
* !! =>Repeats the last command
* exit =>Logout the current session

# Assignments

## BASIC COMMANDS:

# Q) Explain the difference between these two commands.

* find /data -name "\*.txt"
* find /data -name \*.txt
* When the \*.txt is quoted then the shell will not touch it. The find tool will look in the /data for all files ending in .txt.
* When \*.txt is not quoted then the shell might expand this “when one or more files that ends in .txt exist in the current directory”. The find might show a different result, or can result in a syntax error.

# Q) Explain the difference between these two statements. Will they both work when there are 200 .odf files in /data? How about when there are 2 million .odf files?

* find /data -name "\*.odf" > data\_odf.txt
* find /data/\*.odf > data\_odf.txt
* The first find will output all .odf filenames in /data and all subdirectories. The shell will redirect this to a file.
* The second find will output all files named .odf in /data and will also output all files that exist in directories named \*.odf (in /data).
* With the two millions of files the command line would be expanded beyond the maximum that the shell can accept. Then the last part of the command line would be lost.

# Q) Write a find command that finds all files created after January 30th 2010.

* touch -t 201001302359 marker\_date
* find . -type f -newer marker\_date

# Q) Write a find command that finds all \*.odf files created in September 2009.

* touch -t 200908312359 marker\_start
* touch -t 200910010000 marker\_end
* find . -type f -name "\*.odf" -newer marker\_start ! -newer marker\_end

# Q) Count the number of \*.conf files in /etc and all its subdirs.

* find /etc -type f -name '\*.conf' | wc –l

# Q) Here are two commands that do the same thing: copy \*.odf files to /backup/ . What would be a reason to replace the first command with the second?

* cp -r /data/\*.odf /backup/
* find /data -name "\*.odf" -exec cp {} /backup/ \;
* The first might fail when there are too many files to fit on one command line

# Q) Create a file called loctest.txt. Can you find this file with locate? Why not? How do you make locate find this file?

* You cannot locate this with locate because it is not yet in the index.
* By using “updatedb” we can make locate find this file.

# Q) Use find and -exec to rename all .htm files to .html.

* find . -name '\*.htm'\
* find . -name '\*.htm' -exec mv {} {}l \;
* find . -name '\*.htm\*'

# Q) Issue the date command. Now display the date in YYYY/MM/DD format.

* date +%Y/%m/%d

# Q) Issue the cal command. Display a calendar of 1582 and 1752. Notice anything special ?

* cal

Shows today’s date

* cal -y 1582

Displays 1582 calendar.

* cal -y 1752

Displays 1752 calendar.

## COMMANDS AND ARGUMENTS:

# Q) How many arguments are in this line (not counting the command itself).

• touch '/etc/cron/cron.allow' 'file 42.txt' "file 33.txt

* Three (3)

# Q) Is tac a shell builtin command?

* Tac command in linux is used to concatenate and print files in reverse. This command will write each file to standard output, the last line first. When no file is specified then this command will read the standard input.
  + Tac is /usr/bin/tac

# Q) Is there an existing alias for rm?

* There is no inbuit rm in alias
* bash:alias:rm:not found
* if we need we can create using “alias rm”

# Q) What is -i option of rm. Create and remove a file to test the -i option.

* ‘-I’ option in rm command will prompt before deleting a file.
* man rm

touch testfile

rm –i testfile

# Q) Execute: alias rm='rm -i'. Test your alias with a test file. Does this work as expected?

* When we need to be promted ,just use rm -I in the alias.
* touch testfile rm testfile

# Q) List all current aliases.

* For this we need to type alias at the promt and any active aliases will be listed.
* Alias

# Q) Create an alias called 'city' that echoes your hometown and use your alias to test that it works.

* alias city= ‘echo bvrm’
* alias
* alias city=’echo bvrm’
* alias egrep=’egrep

color=auto’

* alias fgrep=’egrep

color=auto’

* alias grep=’egrep

color=auto’

* alias l=’ls -CF’
* alias la=’ls -A’
* alias ll=’ls –alF’
* alias ls=’ls

color=auto’

# Q) Execute set -x to display shell expansion for every command.

* city (it should display bvrm) bvrm

# Q) Test the functionality of set -x by executing your city and rm aliases.

* set –x

# Q) Execute set +x to stop displaying shell expansion.

* ~$ set -x
* ~$ city
* + echo bvrm
* bvrm

# Q) Remove your city alias.

* unalias city
* unalias:removes the city name bvrm

# Q) What is the location of the cat and the passwd commands ?

* cat: /bin/cat
* passwd: /user/bin/passwd

# Q) Explain the difference between the following commands:

* echo
* /bin/echo
* The echo command will be interpreted by the shell as the built-in echo command. The /bin/echo command will make the shell execute the echo binary located in the /bin director

# Q) Explain the difference between the following commands:

* echo Hello
* echo -n Hello
* The -n option of the echo command will prevent echo from echoing a trailing newline.

echo Hello will echo six characters in total, echo -n hello only echoes five characters.

# Q) Display A B C with two spaces between B and C.

* echo "A B C"

# Q) Display (do not use spaces) exactly the following output:

* 4+4=8
* 10+14=24
* echo -e "4+4\t=8" ;
* echo -e "10+14\t=24"

# Q) Use echo to display the following exactly :??\\ Find two solutions with single quotes, two with double quotes and one without quotes.

* echo '??\\'
* echo -e '??\\\\'
* echo "??\\\\"
* echo -e "??\\\\\\"
* echo ??\\\\

# Q) Use one echo command to display three words on three lines.

* echo -e “firstword \nsecondword \nthirdword" “\n” ->takes to next line

## CONTROL OPERANDS:

# Q) When you type passwd, which file is executed?

* which passwd

# Q) What kind of file is that?

* file /usr/bin/passwd

# Q) Execute the pwd command twice. (remember 0.)

# pwd; pwd

# Q) Execute ls after cd /etc, but only if cd /etc did not error.

* cd /etc && ls

# Q) Execute cd /etc after cd etc, but only if cd etc fails.

* cd /etc || cd /etc

# Q) Echo it worked when 'touch test42' works, and echo it failed when the touch failed. All on one command line as a normal user (not root). Test this line in your home directory and in /bin/ .

* ~$ cd ; touch test42 && echo it worked || echo it failed

It worked

* ~$ cd /bin; touch test42 && echo it worked || echo it failed

touch: cannot touch `test42` : Permission denied

It failed

# Q) Execute sleep 6, what is this command doing ?

* Pausing for six seconds

# Q) Execute sleep 200 in background (do not wait for it to finish).

* sleep 200 &

# Q) Write a command line that executes rm file55. Your command line should print 'success' if file55 is removed, and print 'failed' if there was a problem.

* rm file55 && echo success || echo failed

# Q) Use echo to display "Hello World with strange' characters \ \* [ } ~ \\ ." (including all quotes)

* echo \”Hello World with strange\’ characters \\ \\* \[ \} \~ \\\\ \. \” OR
* echo \”Hello World with strange\’ characters \ \* [ } ~ \\ . “\”

## FILE LINKS:

# Q) Create two files named winter.txt and summer.txt, put some text in them.

* echo cold > winter.txt ; echo hot > summer.txt

# Q) Create a hard link to winter.txt named hlwinter.txt.

* ln winter.txt hlwinter.txt

# Q) Display the inode numbers of these three files, does the hard links have the same inode?

* ls -li winter.txt summer.txt hlwinter.txt

# Q) Use the find command to list the two hardlinked files

* find . inum xyz #replace xyz with the inode number

# Q) Everything about a file is in the inode, except two things : name them!

* The name of the file is in a directory,and the contents is somewhere on the disk

# Q) Create a symbolic link to summer.txt called slsummer.txt.

* ln -s summer.txt slsummer.txt

# Q) Find all files with inode number 2. What does this information tell you ?

* It tells you there is more than one inode table (one for every formatted partition + virtual file systems)

# Q) Look at the directories /etc/init.d/ /etc/rc2.d/ /etc/rc3.d/ ... do you see the links ?

* ls -l /etc/init.d
* ls -l /etc/rc2.d
* ls -l /etc/rc3.d

# Q) Look in /lib with ls -l...

* ls -l /lib

# Q) Use find to look in your home directory for regular files that do not(!) have one hard link.

* find ~ ! -links 1 -type f

## STANDARD FILE PERMISSIONS:

# Q) As normal user, create a directory ~/permissions. Create a file owned by yourself in there.

* mkdir ~/permissions ; touch ~/permissions/myfile.txt

# Q) Copy a file owned by root from /etc/ to your permissions dir, who owns this file now ?

* cp /etc/hosts ~/permissions/ (The copy is owned by the user)

# Q) As root, create a file in the users ~/permissions directory.

* (become root)# touch /home/username/permissions/rootfile

# Q) As normal user, look at who owns this file created by root.

* ls -l ~/permissions The file is created by root and is owned by root

# Q) Change the ownership of all files in ~/permissions to yourself.

* chown user ~/permissions/\* Cannot become owner of the file that belongs to root

# Q) Make sure you have all rights to these files, and others can only read.

* chmod 644 (on file)
* chmod 755 (on directories)

# Q) With chmod, is 770 the same as rwxrwx--- ?

* Yes

# Q) With chmod, is 664 the same as r-xr-xr-- ?

* No

# Q) With chmod, is 400 the same as r-------- ?

* Yes

# Q) With chmod, is 734 the same as rwxr-xr-- ?

* No

# Q) A) Display the umask in octal and in symbolic form

# B) Set the umask to 077, but use the symbolic format to set it. Verify that this works.

* umask ; umask -S
* umask -S u=rwx, g0=

# Q) Create a file as root, give only read to others. Can a normal user read this file ? Test writing to this file with nano.

* (become root)

# echo hello > /home/username/root.txt

# chmod 744 > /home/username/root.txt

(become user)

vi ~/root.txt

# Q) A) Create a file as normal user, give only read to others. Can another normal user read this file ? Test writing to this file with vi.

# B) Can root read this file ? Can root write to this file with vi ?

* echo hello > file ; chmod 744 file Yes, others can read this file
* Yes, root can read and write to this file. Permissions do not apply to root

# Q) Create a directory that belongs to a group, where every member of that group can read and write to files, and create files. Make sure that people can only delete their own files.

* mkdir home/project42 ; groupadd project42
* chgrp project42 /home/project42 ; chmod 755 /home/project42

## FILE SYSTEM:

# Q) Put a sorted list of all bash users in bashusers.txt.

* grep bash /etc/passwd | cut -d: -f1 | sort > bashusers.txt

# Q) Put a sorted list of all logged on users in onlineusers.txt.

* who | cut -d' ' -f1 | sort > onlineusers.txt

# Q) Make a list of all filenames in /etc that contain the string conf in their filename.

* ls /etc | grep conf

# Q) Make a sorted list of all files in /etc that contain the case insensitive string conf in their filename.

* ls /etc | grep -i conf | sort

# Q) Look at the output of /sbin/ifconfig. Write a line that displays only ip address and the subnet mask.

* /sbin/ifconfig | head -2 | grep 'inet ' | tr -s ' ' | cut -d' ' -f3,5

# Q) Write a line that removes all non-letters from a stream.

* ~$ cat text

This is, yes really! , a text with ?&\* too many str$ange# characters ;-)

* ~$ cat text | tr -d ',!$?.\*&^%#@;()-'

This is yes really a text with too many strange character.

# Q) Write a line that receives a text file, and outputs all words on a separate line.

* ~$ cat text2

it is very cold today without the sun

* ~$ cat text2 | tr ' ' '\n'

it

is

very

cold

today

without

the

sun

# Q) Write a spell checker on the command line(There may be a dictionary in /usr/share/dict/.)

* ~$ echo "The zun is shining today" > text
* ~$ cat > DICT

is

shining

sun

the

today

* ~$ cat text | tr 'A-Z ' 'a-z\n' | sort | uniq | comm -23 - DICT

zun

## FILTERS:

# Q) Put a sorted list of all bash users in bashusers.txt.

* grep bash /etc/passwd | cut -d: -f1 | sort > bashusers.txt

# Q) Put a sorted list of all logged on users in onlineusers.txt.

* who | cut -d' ' -f1 | sort > onlineusers.txt

# Q) Make a list of all filenames in /etc that contain the string conf in their filename.

* ls /etc | grep conf

# Q) Make a sorted list of all files in /etc that contain the case insensitive string conf in their filename.

* ls /etc | grep -i conf | sort

# Q) Look at the output of /sbin/ifconfig. Write a line that displays only ip address and the subnet mask.

* /sbin/ifconfig | head -2 | grep 'inet ' | tr -s ' ' | cut -d' ' -f3,5

# Q) Write a line that removes all non-letters from a stream.

* ~$ cat text

This is, yes really! , a text with ?&\* too many str$ange# characters ;-)

* ~$ cat text | tr -d ',!$?.\*&^%#@;()-'

This is yes really a text with too many strange characters

# Q) Write a line that receives a text file, and outputs all words on a separate line.

* ~$ cat text2

it is very cold today without the sun

* ~$ cat text2 | tr ' ' '\n'

it

is

very

cold

today

without

the

sun

# Q) Write a spell checker on the command line(There may be a dictionary in /usr/share/dict/.)

* ~$ echo "The zun is shining today" > text
* ~$ cat > DICT

is

shining

sun

the

today

* ~$ cat text | tr 'A-Z ' 'a-z\n' | sort | uniq | comm -23 - DICT

Zun

## SHELL PATTERN MATCHING:

# Q) Create a test directory and enter it.

* mkdir testdir; cd testdir

# Q) Create the following files :

file1

file10

file11

file2

File2

File3

file33

fileAB

filea

fileA

fileAAA

file(

file 2

(the last one has 6 characters including a space)

* touch file1 file10 file11 file2 File2 File3
* touch file33 fileAB file fileA fileAAA
* touch "file("
* touch "file 2"

# Q) List (with ls) all files starting with file

* ls file\*

# Q) List (with ls) all files starting with File

* ls File\*

# Q) List (with ls) all files starting with file and ending in a number.

* ls file\*[0-9]

# Q) List (with ls) all files starting with file and ending with a letter

* ls file\*[a-z]

# Q) List (with ls) all files starting with File and having a digit as fifth character.

* ls File[0-9]\*

# Q) List (with ls) all files starting with File and having a digit as fifth character and nothing else.

* ls File[0-9]

# Q) List (with ls) all files starting with a letter and ending in a number.

* ls [a-z]\*[0-9]

# Q) List (with ls) all files that have exactly five characters.

* ls ?????

# Q) List (with ls) all files that start with f or F and end with 3 or A.

* ls [fF]\*[3A]

# Q) List (with ls) all files that start with f have i or R as second character and end in a number.

* ls f[iR]\*[0-9]

# Q) List all files that do not start with the letter F.

* ls [!F]\*

# Q) Can echo replace ls ? How can you list the files in the current directory with echo ?

* echo \*

# Q) Is there another command besides cd to change directories ?

* pushd popd

## SHELL HISTORY:

# Q) Issue the command echo

* echo

The answer to the meaning of life, the universe and everything is 42

# Q) Repeat the previous command using only two characters (there are two solutions!)

* !!

OR

!e

# Q) Display the last 5 commands you typed.

* ~$ history 5

52 ls -l

53 ls

54 df -h | grep sda

55 echo The answer to the meaning of life, the universe and everything is 42

56 history 5

# Q) Issue the long echo from question 1 again, using the line numbers you received from the command in question 3.

* ~$ !55

echo The answer to the meaning of life, the universe and everything is 42

The answer to the meaning of life, the universe and everything is 42

# Q) How many commands can be kept in memory for your current shell session?

* echo $HISTSIZE

# Q) Is the current session history stored to ~/.bashrc\_history?

* Yes

# Q) Where are these commands stored when exiting the shell?

* echo $HISTFILE

# Q) How many commands can be written to the history file when exiting your current shell session?

* echo $HISTFILESIZE

# Q) Make sure your current bash shell remembers the only 10 commands you type.

* HISTSIZE=10

# Q) When is command history written to the history file?

* For each terminal execution.

## SHELL VARIABLES:

# Q) Use echo to display Hello followed by your username. (use a bash variable!)

* echo Hello $USER

# Q) Create a variable answer with a value of 42.

* answer=42

# Q) Copy the value of $LANG to $MyLANG.

* MyLANG=$LANG

# Q) List all current shell variables.

* set
* set|more on Ubuntu/Debian

# Q) List all exported shell variables.

* env
* export
* declare -x

# Q) Do the env and set commands display your variable ?

* env | more
* set | more

# Q) Destroy your answer variable.

* unset answer

# Q) Create two variables, and export one of them.

* var1=1; export var2=2

# Q) Display the exported variable in an interactive child shell.

* bash
* echo $var2

# Q) Create a variable, give it the value 'Dumb', create another variable with value 'do'. Use echo and the two variables to echo Dumbledore.

* varx=Dumb; vary=do
* echo ${varx}le${vary}re
* solution by Yves from Dexia : echo $varx'le'$vary're'
* solution by Erwin from Telenet : echo "$varx"le"$vary"re

# Q) Find the list of backslash escaped characters in the manual of bash. Add the time to your PS1 prompt.

* PS1='\t \u@\h \W$ '

## WORKING WITH DIRECTORIES:

# Q) Display your current directory.

* pwd

# Q) Change to the /etc directory.

* cd /etc

# Q) Now change to your home directory using only three key presses.

* cd (and the enter key)

# Q) Change to the /boot/grub directory using only eleven key presses.

* cd /boot/grub (use the tab key)

# Q) Go to the parent directory of the current directory.

* cd .. (with space between cd and ..)

# Q) Go to the root directory.

* cd /

# Q) List the contents of the root directory.

* ls

# Q) List a long listing of the root directory.

* ls -l

# Q) Stay where you are, and list the contents of /etc.

* ls /etc

# Q) Stay where you are, and list the contents of /bin and /sbin.

* ls /bin /sbin

# Q) Stay where you are, and list the contents of ~.

* ls ~

# Q) List all the files (including hidden files) in your home directory.

* ls -al ~

# Q) List the files in /boot in a human readable format.

* ls -lh /boot

# Q) Create a directory testdir in your home directory.

* mkdir ~/testdir

# Q) Change to the /etc directory, stay here and create a directory newdir in your home directory.

* cd /etc ; mkdir ~/newdir

# Q) Create in one command the directories ~/dir1/dir2/dir3 (dir3 is a subdirectory from dir2,and dir2 is a subdirectory from dir1 ).

* mkdir -p ~/dir1/dir2/dir3

# Q) Remove the directory testdir.

* rmdir testdir

# Q) Open manual page for bash and read about pushd and popd by searching in manpage.

* man bash # opens the manual

/pushd # searches for pushd

n # next (do this two/three times)

## WORKING WITH FILES:

# Q) List the files in the /bin directory

* ls /bin

# Q) Display the type of file of /bin/cat, /etc/passwd and /usr/bin/passwd.

* file /bin/cat /etc/passwd /usr/bin/passwd

# Q) Use files LFS.png dummy.pdf

* Display the type of file of wolf.jpg and dummy.pdf
* Rename LFS.png to wolf.pdf
* Display the type of file of wolf.pdf and dummy.pdf.
* file wolf.jpg dummy.pdf
* mv LFS.png wolf.pdf
* file wolf.pdf dummy.pdf

# Q) Create a directory ~/touched and enter it.

* mkdir ~/touched ; cd ~/touched

# Q) Create the files today.txt and yesterday.txt in touched.

* touch today.txt yesterday.txt

# Q) Change the date on yesterday.txt to match yesterday's date.

* touch -t 200810251405 yesterday.txt (substitute 20081025 with yesterday)

# Q) Copy yesterday.txt to copy.yesterday.txt

* cp yesterday.txt copy.yesterday.txt

# Q) Rename copy.yesterday.txt to kim

* mv copy.yesterday.txt kim

# Q) Create a directory called ~/testbackup and copy all files from ~/touched into it.

* mkdir ~/testbackup ; cp -r ~/touched ~/testbackup/

# Q) Use one command to remove the directory ~/testbackup and all files in it.

* rm -rf ~/testbackup

# Q) Create a directory ~/etcbackup and copy all \*.conf files from /etc into it. Did you include all subdirectories of /etc ?

* cp -r /etc/\*.conf ~/etcbackup

Only \*.conf files that are directly in /etc/ are copied.

# Q) Use rename to rename all \*.conf files to \*.backup .

* On RHEL: touch 1.conf 2.conf ; rename conf backup \*.conf
* On Debian: touch 1.conf 2.conf ; rename 's/conf/backup/' \*.conf

## WORKING WITH FILE CONTENTS:

# Q) Display the first 12 lines of /etc/services.

* head -12 /etc/services

# Q) Display the last line of /etc/passwd.

* tail -1 /etc/passwd

# Q) Use cat to create a file named count.txt that looks like this:

One

Two

Three

Four

Five

* cat > count.txt

One

Two

Three

Four

Five (followed by Ctrl-d)

# Q) Use cp to make a backup of this file to cnt.txt.

* cp count.txt cnt.txt

# Q) Use cat to make a backup of this file to catcnt.txt.

* cat count.txt > catcnt.txt

# Q) Display catcnt.txt, but with all lines in reverse order (the last line first).

* tac catcnt.txt

# Q) Use more to display /etc/services.

* more /etc/services

# Q) Display the readable character strings from the /usr/bin/passwd command.

* strings /usr/bin/passwd

# Q) Use ls to find the biggest file in /etc.

* ls -lrS /etc

# Q) Use cat to create a file named tailing.txt that contains the contents of tailing.txt followed by the contents of /etc/passwd.

* cat /etc/passwd >> tailing.txt

# Q) Use cat to create a file named tailing.txt that contains the contents of tailing.txt preceded by the contents of /etc/passwd.

* mv tailing.txt tmp.txt ; cat /etc/passwd tmp.txt > tailing.txt

# OS Programming Concepts

## SYSTEM CALLS

The system call is the fundamental interface between an application and the Linux kernel. System call provides the services of the operating system to the user programs via Application Program Interface(API).

Services by system calls:

* Process creation and management
* Main memory management
* File Access, Directory and File system management
* Device handling(I/O)
* Protection
* Networking, etc.

Types of System Calls

There are 5 different categories of system calls:

* Process control: end, abort, create, terminate, allocate and free memory.

Example: fork(), exit(), wait()

* File management: create, open, close, delete, read file etc.

Example**:** open(), read(), write(), close()

* Device management

Example: ioctl(), read(), write()

* Information maintenance

Example: getpid(), alarm(), sleep()

* Communication

Example: pipe(), shmget(), mmap()

## PROCESSES

A program/command when executed, a special instance is provided by the system to the process. This instance consists of all the services/resources that may be utilized by the process under execution.

A process has a series of characteristics, which can be viewed with the **ps** command:

* The process ID or PID: a unique identification number used to refer to the process.
* The parent process ID or PPID: the number of the process (PID) that started this process.
* Terminal or TTY: terminal to which the process is connected.

A process can be run in two ways:

* Foreground Process.
* Background Process.

Types of Processes:

* **Parent and Child process**: All the processes in operating system are created when a process executes the fork() system call except the startup process. The process that used the fork() system call is the **parent** process. In other words, a parent process is one that creates a child process. A **child** process is a process created by a parent process in operating system using a fork() system call. If a child process exits or is interrupted, then a SIGCHLD signal is send to the parent process.
* **Zombie and Orphan process**:At times when the parent process is killed before the termination of the child process, the child processes becomes **orphan** processes, with the parent of all processes “init” process, becomes their new ppid.  
  A process which is killed but still shows its entry in the process status or the process table is called a **zombie** process, they are dead and are not used.
* **Daemon process**:A **daemon** process is a process which runs in background and has no controlling terminal. Daemon processes are used to provide services that can well be done in background without any user interaction. For example a process that runs in background and observes network activity and logs any suspicious communication can be developed as a daemon process.

## THREADS

A thread is a path of execution within a process. A process can contain multiple threads. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

Types of Threads

* User Level Thread
* Kernel Level Thread

The primary difference between thread and process is that threads within the same process run in a shared memory space, while processes run in separate memory spaces.

Advantages of Thread over Process:

*1.* **Responsiveness**: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

*2.* **Faster context switch**:Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

*3.* **Effective utilization of multiprocessor system**:If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

*4.* **Resource sharing**:Resources like code, data, and files can be shared among all threads within a process. Stack and registers can’t be shared among the threads. Each thread has its own stack and registers.

*5.* **Communication**:Communication between multiple threads is easier, as the threads shares common address space. While in process we have to follow some specific communication technique for communication between two processes.

*6.* **Enhanced throughput of the system**:If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

Header file

* “pthread.h”

Initialize thread

* pthread\_mutex\_t M1 = PTHREAD\_MUTEX\_INITIALIZER
* pthread\_mutex\_init(&M1)

Lock thread

* pthread\_mutex\_lock(&M1)

Unlock thread

* pthread\_mutex\_unlock(&M1)

Destroy thread

* pthread\_mutex\_destroy (&M1)

## Semaphore

Semaphore is a data handling technique which is very useful in process synchronization and multithreading.

The POSIX system in Linux presents its own built-in semaphore library. To use it:

Include “**semaphore.h”** header file

Compile the code by linking with **-lpthread –lrt**.

To lock a semaphore, we can use the **sem\_wait** function.

* int sem\_wait(sem\_t \*sem);

To release or signal a semaphore, we use the **sem\_post** function**.**

* int sem\_post(sem\_t \*sem);

A semaphore is initialised by using **sem\_init** (for processes or threads).

* sem\_init(sem\_t \*sem, int pshared, unsigned int value);

To destroy a semaphore, we can use **sem\_destroy**

* sem\_destoy(sem\_t \*sem);

## Message Queue

POSIX message queues allow for an efficient, priority-driven IPC mechanism with multiple readers and writers. The message-queue structures are found in the “**mqueue.h**” header file.

Compilation of the message queue programs aew by linking with **–lrt**.

**mq\_close() -** Close a message queue

* int mq\_close (mqd\_t mqdes);

**mq\_notify() -** notify the calling process when the queue becomes nonempty

* int mq\_notify (mqd\_t mqdes, const struct sigevent \*sevp);

**mq\_open() -** open or create a message queue

* mqd\_t mq\_open (const char \*name, int oflag);
* mqd\_t mq\_open (const char \*name, int oflag, mode\_t mode, struct mq\_attr \*attr);

**mq\_receive() -** receive a message from a queue

* ssize\_t mq\_receive (mqd\_t mqdes, char \*msg\_ptr, size\_t msg\_len, unsigned int \*msg\_prio);

**mq\_send() -** put a message into a message queue

* int mq\_send (mqd\_t mqdes, const char \*msg\_ptr, size\_t msg\_len, unsigned int msg\_prio);

**mq\_unlink() -** unlink (i.e. delete) a message queue

* int mq\_unlink(const char \*queue\_name);

## Pipes

 Pipe is a communication medium between two or more related or interrelated processes. It can be either within one process or a communication between the child and the parent processes. Communication can also be multi-level such as communication between the parent, the child and the grand-child, etc.

Communication is achieved by one process writing into the pipe and other reading from the pipe.  Pipe has the functions create, connect and delete and functions similar to a device driver (open, write, read, close). Pipe is unidirectional. One thread or task inserts into it and other one deletes from it.

Limitations:

 As a channel of communication a pipe operates in one direction only. Pipes cannot support broadcast i.e. sending message to multiple processes at the same time.  The read end of a pipe reads any way. It does not matter which process is connected to the write end of the pipe. Therefore, this is very insecure mode of communication.

Pipe

* mkfifo mypipe, tail -f mypipe

Write to pipe

* ssize\_t write(int fd, void \*buf, size\_t count)

Read from pipe

* ssize\_t read(int fd, void \*buf, size\_t count)

Close pipe

* int close(int fd)

## Shared Memory

Shared memory system is one of the fundamental models of inter-process communication. In the shared memory system, the cooperating processes communicate with each other by establishing the shared memory region, in its address space. Shared memory model allows the fastest inter-process communication.

In Shared Memory system, the cooperating processes communicate, to exchange the data or the information with each other. For this, the cooperating processes establish a shared region in their memory.

The processes share data by reading and writing the data in the shared segment of the processes.

POSIX Shared Memory Calls

A POSIX shared memory object is a memory-mapped file. POSIX shared memory files are provided from a tmpfs filesystem mounted at /dev/shm. The individual shared memory files are created using the shm\_open system call under /dev/shm.

There are just two specialized POSIX shared memory system calls, shm\_open and shm\_unlink, which are analogous to open and unlink system calls for files.

* int shm\_open (const char \*name, int oflag, mode\_t mode);
* int shm\_unlink (const char \*name);

Other operations on POSIX shared memory are done using the ftruncate, mmap and munmap system calls for files.

* void \*mmap (void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset);
* int munmap (void \*addr, size\_t length);

A program using POSIX shared memory calls needs to be linked with -lrt.