./



GENESIS – Linux & OS Programming Learning Report

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **To be Approved** | **Remarks/Revision Details** |
| 1.0 | 10-1-21 | 1. Megha Annie George |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Details**

**Contents**

[List of Figures 3](#_Toc61193066)

[List of Tables 4](#_Toc61193067)

[1 LINUX COMMANDS 4](#_Toc61193068)

[1.1 BASIC COMMANDS 4](#_Toc61193069)

[1.2 COMMANDS AND ARGUMENTS 6](#_Toc61193070)

[1.3 CONTROL OPERATIONS 9](#_Toc61193071)

[1.4 FILE LINKS 11](#_Toc61193072)

[1.5 FILE PERMISSIONS 12](#_Toc61193073)

[1.6 FILE SYSTEMS 14](#_Toc61193074)

[1.7 FILTERS 16](#_Toc61193075)

[1.8 PATTERN MATCHING 17](#_Toc61193076)

[1.9 REDIRECTION 20](#_Toc61193077)

[1.10 SHELL HISTORY 20](#_Toc61193078)

[1.11 SHELL VARIABLES 22](#_Toc61193079)

[1.12 WORKING WITH DIRECTORIES 23](#_Toc61193080)

[1.13 WORKING WITH FILE CONTENTS 26](#_Toc61193081)

[1.14 WORKING WITH FILES 28](#_Toc61193082)

[2 OS PROGRAMMING 30](#_Toc61193083)

[2.1 SYSTEM CALLS 30](#_Toc61193084)

[2.2 PROCESSES 30](#_Toc61193085)

[2.3 THREADS 30](#_Toc61193086)

[2.4 SEMAPHORES 30](#_Toc61193087)

[2.5 MESSAGE QUEUE 30](#_Toc61193088)

[2.6 PIPE 30](#_Toc61193089)

[2.7 SHARED MEMORY 30](#_Toc61193090)

# List of Figures

# No table of figures entries found.

# List of Tables

# LINUX COMMANDS

## FILE COMMANDS

|  |  |  |
| --- | --- | --- |
| 1. | ls | Directory listing |
| 2. | ls -al | Formatted listing with hidden files |
| 3. | ls -lt | Sorting the Formatted listing by time modification |
| 4. | cd dir | Change directory to dir |
| 5. | cd | Change to home directory |
| 6. | pwd | Show current working directory |
| 7. | mkdir dir | Creating a directory dir |
| 8. | cat >file | Places the standard input into the file |
| 9. | more file | Output the contents of the file |
| 10. | head file | Output the first 10 lines of the file |
| 11. | tail file | Output the last 10 lines of the file |
| 12. | tail -f file | Output the contents of file as it grows,starting with the last 10 lines |
| 13. | touch file | Create or update file |
| 14. | rm file | Deleting the file |
| 15. | rm -r dir | Deleting the directory |
| 16. | rm -f file | Force to remove the file |
| 17. | rm -rf dir | Force to remove the directory dir |
| 18. | cp file1 file2 | Copy the contents of file1 to file2 |
| 19. | cp -r dir1 dir2 | Copy dir1 to dir2;create dir2 if not present |
| 20. | mv file1 file2 | Rename or move file1 to file2,if file2 is an existing directory |
| 21. | ln -s file link | Create symbolic link link to file |

## PROCESS MANAGEMENT

|  |  |  |
| --- | --- | --- |
| 1. | ps | To display the currently working processes |
| 2. | top | Display all running process |
| 3. | kill pid | Kill the process with given pid | |
| 4. | killall proc | Kill all the process named proc | |
| 5. | pkill pattern | Will kill all processes matching the pattern | |
| 6. | bg | List stopped or background jobs,resume a stopped job in the background | |
| 7. | fg | Brings the most recent job to foreground | |
| 8. | fg n | Brings job n to the foreground | |

## FILE PERMISSION

|  |  |  |
| --- | --- | --- |
| 1. | chmod octal file | Change the permission of file to octal,which can be found separately for user,group,world by adding,   * 4-read(r) * 2-write(w) * 1-execute(x) |

## SEARCHING

|  |  |  |
| --- | --- | --- |
| 1. | grep pattern file | Search for pattern in file |
| 2. | grep -r pattern dir | Search recursively for pattern in dir |
| 3. | command | grep pattern | Search pattern in the output of a command |
| 4. | locate file | Find all instances of file |
| 5. | find . -name filename | Searches in the current directory (represented by a period) and below it, for files and directories with names starting with filename |
| 6. | pgrep pattern | Searches for all the named processes , that matches with the pattern and, by default, returns their ID |

## SYSTEM INFO

|  |  |  |
| --- | --- | --- |
| 1. | date | Show the current date and time |
| 2. | cal | Show this month's calender |
| 3. | uptime | Show current uptime |
| 4. | w | Display who is on line |
| 5. | whoami | Who you are logged in as |
| 6. | finger user | Display information about user |
| 7. | uname -a | Show kernel information |
| 8. | cat /proc/cpuinfo | Cpu information |
| 9. | cat proc/meminfo | Memory information |
| 10. | man command | Show the manual for command |
| 11. | df | Show the disk usage |
| 12. | du | Show directory space usage |
| 13. | free | Show memory and swap usage |
| 14. | whereis app | Show possible locations of app |
| 15. | which app | Show which applications will be run by default |

## COMPRESSION

|  |  |  |
| --- | --- | --- |
| 1. | tar cf file.tar file | Create tar named file.tar containing file |
| 2. | tar xf file.tar | Extract the files from file.tar |
| 3. | tar czf file.tar.gz files | Create a tar with Gzip compression |
| 4. | tar xzf file.tar.gz | Extract a tar using Gzip |
| 5. | tar cjf file.tar.bz2 | Create tar with Bzip2 compression |
| 6. | tar xjf file.tar.bz2 | Extract a tar using Bzip2 |
| 7. | gzip file | Compresses file and renames it to file.gz |
| 8. | gzip -d file.gz | Decompresses file.gz back to file |

## NETWORK

|  |  |  |
| --- | --- | --- |
| 1. | ping host | Ping host and output results |
| 2. | whois domain | Get whois information for domains |
| 3. | dig domain | Get DNS information for domain |
| 4. | dig -x host | Reverse lookup host |
| 5. | wget file | Download file |
| 6. | wget -c file | Continue a stopped download |

## SHORTCUTS

|  |  |  |
| --- | --- | --- |
| 1. | ctrl+c | Halts the current command |
| 2. | ctrl+z | Stops the current command, resume with fg in the foreground or bg in the background |
| 3. | ctrl+d | Logout the current session, similar to exit |
| 4. | ctrl+w | Erases one word in the current line |
| 5. | ctrl+u | Erases the whole line |
| 6. | ctrl+r | Type to bring up a recent command |
| 7. | !! | Repeats the last command |
| 8. | exit | Logout the current session |

# ASSIGNMENT

## BASIC COMMANDS

|  |  |
| --- | --- |
| 1 | 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. |

|  |  |
| --- | --- |
| 2 | 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. |

|  |  |
| --- | --- |
| 3 | Write a find command that finds all files created after January 30th 2010. |
| touch -t 201001302359 marker\_date  find . -type f -newer marker\_date |

|  |  |
| --- | --- |
| 4 | 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 |

|  |  |
| --- | --- |
| 5 | Count the number of \*.conf files in /etc and all its subdirs. |
| find /etc -type f -name '\*.conf' | wc –l |

|  |  |
| --- | --- |
| 6 | 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 |

|  |  |
| --- | --- |
| 7 | 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. |

|  |  |
| --- | --- |
| 8 | Use find and -exec to rename all .htm files to .html. |
| find . -name '\*.htm'\  find . -name '\*.htm' -exec mv {} {}l \;  find . -name '\*.htm\*' |

|  |  |
| --- | --- |
| 9 | Issue the date command. Now display the date in YYYY/MM/DD format. |
| date +%Y/%m/%d |

|  |  |
| --- | --- |
| 10 | Issue the cal command. Display a calendar of 1582 and 1752. Notice anything special ? |
| * cal * cal -y 1582 * displays 1582 calendar:   In 1582, when Pope Gregory XIII introduced his Gregorian calendar, Europe adhered to the Julian calendar, first implemented by Julius Caesar in 46 B.C. Since the Roman emperor's system miscalculated the length of the solar year by 11 minutes, the calendar had since fallen out of sync with the seasons.   * cal -y 1752 * displays 1752 calendar:   Since the Gregorian calendar accounted more accurately for leap years, it was 11 days ahead of the Julian calendar by 1752. |

## COMMANDS AND ARGUMENTS

|  |  |
| --- | --- |
| 1 | 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) |

|  |  |
| --- | --- |
| 2 | 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 |

|  |  |
| --- | --- |
| 3 | 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” |

|  |  |
| --- | --- |
| 4 | 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 |

|  |  |
| --- | --- |
| 5 | 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 |

|  |  |
| --- | --- |
| 6 | List all current aliases. |
| * For this we need to type alias at the promt and any active aliases will be listed.   Alias |

|  |  |
| --- | --- |
| 7 | 1. Create an alias called 'city' that echoes your hometown.   B) 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’ |

|  |  |
| --- | --- |
| 8 | Execute set -x to display shell expansion for every command. |
| city (it should display bvrm) bvrm |

|  |  |
| --- | --- |
| 9 | Test the functionality of set -x by executing your city and rm aliases. |
| set –x |

|  |  |
| --- | --- |
| 10 | Execute set +x to stop displaying shell expansion. |
| * ~$ set -x * ~$ city * + echo bvrm * bvrm |

|  |  |
| --- | --- |
| 11 | Remove your city alias. |
| * unalias city * unalias:removes the city name bvrm |

|  |  |
| --- | --- |
| 12 | What is the location of the cat and the passwd commands ? |
| * cat: /bin/cat * passwd: /user/bin/passwd |

|  |  |
| --- | --- |
| 13 | 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 |

|  |  |
| --- | --- |
| 14 | 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. |

|  |  |
| --- | --- |
| 15 | Display A B C with two spaces between B and C. |
| echo "A B C" |

|  |  |
| --- | --- |
| 16 | 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" |

|  |  |
| --- | --- |
| 17 | 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 ??\\\\ |

|  |  |
| --- | --- |
| 18 | Use one echo command to display three words on three lines. |
| echo -e “firstword \nsecondword \nthirdword" “\n” ->takes to next line |

## CONTROL OPERATIONS

|  |  |
| --- | --- |
| 1 | When you type passwd, which file is executed? |
| which passwd |

|  |  |
| --- | --- |
| 2 | What kind of file is that? |
| file /usr/bin/passwd |

|  |  |
| --- | --- |
| 3 | Execute the pwd command twice. (remember 0.) |
| pwd; pwd |

|  |  |
| --- | --- |
| 4 | Execute ls after cd /etc, but only if cd /etc did not error. |
| cd /etc && ls |

|  |  |
| --- | --- |
| 5 | Execute cd /etc after cd etc, but only if cd etc fails. |
| cd /etc || cd /etc |

|  |  |
| --- | --- |
| 6 | 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 |

|  |  |
| --- | --- |
| 7 | Execute sleep 6, what is this command doing ? |
| pausing for six seconds |

|  |  |
| --- | --- |
| 8 | Execute sleep 200 in background (do not wait for it to finish). |
| sleep 200 & |

|  |  |
| --- | --- |
| 9 | 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 |

|  |  |
| --- | --- |
| 10 | 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

|  |  |
| --- | --- |
| 1 | Create two files named winter.txt and summer.txt, put some text in them. |
| echo cold > winter.txt ; echo hot > summer.txt |

|  |  |
| --- | --- |
| 2 | Create a hard link to winter.txt named hlwinter.txt. |
| ln winter.txt hlwinter.txt |

|  |  |
| --- | --- |
| 3 | Display the inode numbers of these three files, does the hard links have the same inode? |
| ls -li winter.txt summer.txt hlwinter.txt |

|  |  |
| --- | --- |
| 4 | Use the find command to list the two hardlinked files |
| find . inum xyz #replace xyz with the inode number |

|  |  |
| --- | --- |
| 5 | 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 |

|  |  |
| --- | --- |
| 6 | Create a symbolic link to summer.txt called slsummer.txt. |
| ln -s summer.txt slsummer.txt |

|  |  |
| --- | --- |
| 7 | 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) |

|  |  |
| --- | --- |
| 8 | 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 |

|  |  |
| --- | --- |
| 9 | Look in /lib with ls -l... |
| ls -l /lib |

|  |  |
| --- | --- |
| 10 | Use find to look in your home directory for regular files that do not(!) have one hard link. |
| find ~ ! -links 1 -type f |

## FILE PERMISSIONS

|  |  |
| --- | --- |
| 1 | As normal user, create a directory ~/permissions. Create a file owned by yourself in there. |
| mkdir ~/permissions ; touch ~/permissions/myfile.txt |

|  |  |
| --- | --- |
| 2 | 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 |

|  |  |
| --- | --- |
| 3 | As root, create a file in the users ~/permissions directory. |
| (become root)# touch /home/username/permissions/rootfile |

|  |  |
| --- | --- |
| 4 | As normal user, look at who owns this file created by root. |
| * ls -l ~/permissions   The file is created by root is owned by root |

|  |  |
| --- | --- |
| 5 | Change the ownership of all files in ~/permissions to yourself. |
| * chown user ~/permissions/\*   Cannot become owner of the file that belongs to root |

|  |  |
| --- | --- |
| 6 | Make sure you have all rights to these files, and others can only read. |
| * chmod 644 (on file)   chmod 755 (on directories) |

|  |  |
| --- | --- |
| 7 | With chmod, is 770 the same as rwxrwx--- ? |
| yes |

|  |  |
| --- | --- |
| 8 | With chmod, is 664 the same as r-xr-xr-- ? |
| no |

|  |  |
| --- | --- |
| 9 | With chmod, is 400 the same as r-------- ? |
| yes |

|  |  |
| --- | --- |
| 10 | With chmod, is 734 the same as rwxr-xr-- ? |
| no |

|  |  |
| --- | --- |
| 11 | 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. |
| a. umask ; umask -S  b. umask -S u=rwx, g0= |

|  |  |
| --- | --- |
| 12 | 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 |

|  |  |
| --- | --- |
| 13 | 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  b. Yes, root can read and write to this file. Permissions do not apply to root |

|  |  |
| --- | --- |
| 14 | 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 |

## FILTERS

|  |  |
| --- | --- |
| 1 | Put a sorted list of all bash users in bashusers.txt. |
| grep bash /etc/passwd | cut -d: -f1 | sort > bashusers.txt |

|  |  |
| --- | --- |
| 2 | Put a sorted list of all logged on users in onlineusers.txt. |
| who | cut -d' ' -f1 | sort > onlineusers.txt |

|  |  |
| --- | --- |
| 3 | Make a list of all filenames in /etc that contain the string conf in their filename. |
| * ls /etc | grep conf |

|  |  |
| --- | --- |
| 4 | 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 |

|  |  |
| --- | --- |
| 5 | 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 |

|  |  |
| --- | --- |
| 6 | 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 |

|  |  |
| --- | --- |
| 7 | 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 |

|  |  |
| --- | --- |
| 8 | 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 |

## PATTERN MATCHING

|  |  |
| --- | --- |
| 1 | Create a test directory and enter it. |
| mkdir testdir; cd testdir |

|  |  |
| --- | --- |
| 2 | 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" |

|  |  |
| --- | --- |
| 3 | List (with ls) all files starting with file |
| ls file\* |

|  |  |
| --- | --- |
| 4 | List (with ls) all files starting with File |
| ls File\* |

|  |  |
| --- | --- |
| 5 | List (with ls) all files starting with file and ending in a number. |
| ls file\*[0-9] |

|  |  |
| --- | --- |
| 6 | List (with ls) all files starting with file and ending with a letter |
| ls file\*[a-z] |

|  |  |
| --- | --- |
| 7 | List (with ls) all files starting with File and having a digit as fifth character. |
| ls File[0-9]\* |

|  |  |
| --- | --- |
| 8 | List (with ls) all files starting with File and having a digit as fifth character and nothing else. |
| ls File[0-9] |

|  |  |
| --- | --- |
| 9 | List (with ls) all files starting with a letter and ending in a number. |
| ls [a-z]\*[0-9] |

|  |  |
| --- | --- |
| 10 | List (with ls) all files that have exactly five characters. |
| ls ????? |

|  |  |
| --- | --- |
| 11 | List (with ls) all files that start with f or F and end with 3 or A. |
| ls [fF]\*[3A] |

|  |  |
| --- | --- |
| 12 | 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] |

|  |  |
| --- | --- |
| 13 | List all files that do not start with the letter F. |
| ls [!F]\* |

|  |  |
| --- | --- |
| 14 | Can echo replace ls ? How can you list the files in the current directory with echo ? |
| echo \* |

|  |  |
| --- | --- |
| 15 | Is there another command besides cd to change directories ? |
| pushd popd |

## SHELL HISTORY

|  |  |
| --- | --- |
| 1 | Issue the command  echo |
|  | * echo   The answer to the meaning of life, the universe and everything is 42 |

|  |  |
| --- | --- |
| 2 | Repeat the previous command using only two characters (there are two solutions!) |
| !!  OR  !e |

|  |  |
| --- | --- |
| 3 | 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 |

|  |  |
| --- | --- |
| 4 | 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 |

|  |  |
| --- | --- |
| 5 | How many commands can be kept in memory for your current shell session? |
| echo $HISTSIZE |

|  |  |
| --- | --- |
| 6 | Is the current session history stored to ~/.bashrc\_history? |
| Yes |

|  |  |
| --- | --- |
| 7 | Where are these commands stored when exiting the shell? |
| echo $HISTFILE |

|  |  |
| --- | --- |
| 8 | How many commands can be written to the history file when exiting your current shell session? |
| * echo $HISTFILESIZE |

|  |  |
| --- | --- |
| 9 | Make sure your current bash shell remembers the only 10 commands you type. |
| * HISTSIZE=10 |

|  |  |
| --- | --- |
| 10 | When is command history written to the history file? |
| For each terminal execution. |

## SHELL VARIABLES

|  |  |
| --- | --- |
| 1 | Use echo to display Hello followed by your username. (use a bash variable!) |
| * echo Hello $USER |

|  |  |
| --- | --- |
| 2 | Create a variable answer with a value of 42. |
| answer=42 |

|  |  |
| --- | --- |
| 3 | Copy the value of $LANG to $MyLANG. |
| MyLANG=$LANG |

|  |  |
| --- | --- |
| 4 | List all current shell variables. |
| * set   set|more on Ubuntu/Debian |

|  |  |
| --- | --- |
| 5 | List all exported shell variables. |
| * env * export   declare -x |

|  |  |
| --- | --- |
| 6 | Do the env and set commands display your variable ? |
| * env | more   set | more |

|  |  |
| --- | --- |
| 7 | Destroy your answer variable. |
| unset answer |

|  |  |
| --- | --- |
| 8 | Create two variables, and export one of them. |
| var1=1; export var2=2 |

|  |  |
| --- | --- |
| 9 | Display the exported variable in an interactive child shell. |
| * bash   echo $var2 |

|  |  |
| --- | --- |
| 10 | 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 |

|  |  |
| --- | --- |
| 11 | 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

|  |  |
| --- | --- |
| 1 | Display your current directory. |
| pwd |

|  |  |
| --- | --- |
| 2 | Change to the /etc directory. |
| cd /etc |

|  |  |
| --- | --- |
| 3 | Now change to your home directory using only three key presses. |
| cd (and the enter key) |

|  |  |
| --- | --- |
| 4 | Change to the /boot/grub directory using only eleven key presses. |
| cd /boot/grub (use the tab key) |

|  |  |
| --- | --- |
| 5 | Go to the parent directory of the current directory. |
| * cd .. (with space between cd and ..) |

|  |  |
| --- | --- |
| 6 | Go to the root directory. |
| cd / |

|  |  |
| --- | --- |
| 7 | List the contents of the root directory. |
| ls |

|  |  |
| --- | --- |
| 8 | List a long listing of the root directory. |
| ls -l |

|  |  |
| --- | --- |
| 9 | Stay where you are, and list the contents of /etc. |
| ls /etc |

|  |  |
| --- | --- |
| 10 | Stay where you are, and list the contents of /bin and /sbin. |
| ls /bin /sbin |

|  |  |
| --- | --- |
| 11 | Stay where you are, and list the contents of ~. |
| ls ~ |

|  |  |
| --- | --- |
| 12 | List all the files (including hidden files) in your home directory. |
| * ls -al ~ |

|  |  |
| --- | --- |
| 13 | List the files in /boot in a human readable format. |
| * ls -lh /boot |

|  |  |
| --- | --- |
| 14 | Create a directory testdir in your home directory. |
| * mkdir ~/testdir |

|  |  |
| --- | --- |
| 15 | Change to the /etc directory, stay here and create a directory newdir in your home directory. |
| * cd /etc ; mkdir ~/newdir |

|  |  |
| --- | --- |
| 16 | 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 |

|  |  |
| --- | --- |
| 17 | Remove the directory testdir. |
| rmdir testdir |

|  |  |
| --- | --- |
| 18 | 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 FILE CONTENTS

|  |  |
| --- | --- |
| 1 | Display the first 12 lines of /etc/services. |
| head -12 /etc/services |

|  |  |
| --- | --- |
| 2 | Display the last line of /etc/passwd. |
| tail -1 /etc/passwd |

|  |  |
| --- | --- |
| 3 | 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) |

|  |  |
| --- | --- |
| 4 | Use cp to make a backup of this file to cnt.txt. |
| cp count.txt cnt.txt |

|  |  |
| --- | --- |
| 5 | Use cat to make a backup of this file to catcnt.txt. |
| cat count.txt > catcnt.txt |

|  |  |
| --- | --- |
| 6 | Display catcnt.txt, but with all lines in reverse order (the last line first). |
| tac catcnt.txt |

|  |  |
| --- | --- |
| 7 | Use more to display /etc/services. |
| more /etc/services |

|  |  |
| --- | --- |
| 8 | Display the readable character strings from the /usr/bin/passwd command. |
| * strings /usr/bin/passwd |

|  |  |
| --- | --- |
| 9 | Use ls to find the biggest file in /etc. |
| ls -lrS /etc |

|  |  |
| --- | --- |
| 10 | 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 |

|  |  |
| --- | --- |
| 11 | 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 |

## WORKING WITH FILES

|  |  |
| --- | --- |
| 1 | List the files in the /bin directory |
| ls /bin |

|  |  |
| --- | --- |
| 2 | Display the type of file of /bin/cat, /etc/passwd and /usr/bin/passwd. |
| file /bin/cat /etc/passwd /usr/bin/passwd |

|  |  |
| --- | --- |
| 3 | Use files LFS.png dummy.pdf  a. Display the type of file of wolf.jpg and dummy.pdf  b. Rename LFS.png to wolf.pdf  c. Display the type of file of wolf.pdf and dummy.pdf. |
| a. file wolf.jpg dummy.pdf  b. mv LFS.png wolf.pdf  c. file wolf.pdf dummy.pdf |

|  |  |
| --- | --- |
| 4 | Create a directory ~/touched and enter it. |
| mkdir ~/touched ; cd ~/touched |

|  |  |
| --- | --- |
| 5 | Create the files today.txt and yesterday.txt in touched. |
| touch today.txt yesterday.txt |

|  |  |
| --- | --- |
| 6 | Change the date on yesterday.txt to match yesterday's date. |
| touch -t 200810251405 yesterday.txt (substitute 20081025 with yesterday) |

|  |  |
| --- | --- |
| 7 | Copy yesterday.txt to copy.yesterday.txt |
| cp yesterday.txt copy.yesterday.txt |

|  |  |
| --- | --- |
| 8 | Rename copy.yesterday.txt to kim |
| mv copy.yesterday.txt kim |

|  |  |
| --- | --- |
| 9 | Create a directory called ~/testbackup and copy all files from ~/touched into it. |
| mkdir ~/testbackup ; cp -r ~/touched ~/testbackup/ |

|  |  |
| --- | --- |
| 10 | Use one command to remove the directory ~/testbackup and all files in it. |
| * rm -rf ~/testbackup |

|  |  |
| --- | --- |
| 11 | 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. |

|  |  |
| --- | --- |
| 12 | 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 |

# OS PROGRAMMING

## 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 Provided 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 can be run in two ways:**

* **Foreground Process**: Every process when started runs in foreground by default, receives input from the keyboard and sends output to the screen.
* **Background Process**: It runs in the background without keyboard input and waits till keyboard input is required.

**Types of Processes**

* **Parent and Child process**: The 2nd and 3rd column of the ps –f command shows process id and parent’s process id number. For each user process there’s a parent process in the system, with most of the commands having shell as their parent.
* **Zombie and Orphan process**:After completing its execution a child process is terminated or killed and SIGCHLD updates the parent process about the termination and thus can continue the task assigned to it. But 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**:They are system-related background processes that often run with the permissions of root and services requests from other processes, they most of the time run in the background and wait for processes it can work along with for ex print daemon.  
  When ps –ef is executed, the process with ? in the tty field are daemon processes

## THREADS

A thread is a path of execution within a process. A process can contain multiple threads. A thread is also known as lightweight process. 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.

**Process vs Thread?**  
The primary difference 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**

* **Responsiveness**:If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.
* **Faster context switch***:*Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.
* **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.
* **Resource sharing***:*Resources like code, data, and files can be shared among all threads within a process.  
  Note: stack and registers can’t be shared among the threads. Each thread has its own stack and registers.
* **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 process.
* **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.

**Types of Threads**

* User Level Thread
* Kernel Level Thread

## SEMAPHORES

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.

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

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Definition** |
| **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); |

## PIPE

* 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.
* Some plumbing (closing of ends) is required to create a properly directed pipe.

## 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.
* Other operations on POSIX shared memory are done using the ftruncate, mmap and munmap system calls for files.
* A program using POSIX shared memory calls needs to be linked with -lrt.