UNIT – 3

ENVIRONMENT VARIABLES

Environment variables defined as an affect of multiple utilities, functions and applications. Environment variables that apply to a single utility only are defined as a part of the utility description.

Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer.

They are part of the operating environment in which a process runs. For example, a running process can query the value of the TEMP environment variable to discover a suitable location to store temporary files, or the HOME or USERPROFILE variable to find the directory structure owned by the user running the process.

In Unix, each process has its own separate set of environment variables. By default, when a process is created it inherits a duplicate environment of its parent process, except for explicit changes made by the parent when it creates the child. A user can change environment variables for a particular command invocation by indirectly invoking it using

Environment_variable = value<command> notation.

All Unix operating system flavors DOS and Microsoft Windows have environment variables. However, they do not all use the same variable names. A running program can access the values of environment variable for configuration purpose.

ENVIRONMENT VARIABLES INCLUDE

A. HOME: Unix and UserProfile (Microsoft Windows) – indicate where a user's home directory is located in the file system.

HOME / {App Name}: {App Name} and App data \ {developer name \ App Name} {Microsoft Windows} – for storing application setting. Many applications incorrect use USERPROFILE for application setting in window.

B. LANG: this variable determines the locale category for native language, local custom character set in the absence of the LC_ALL and other LC_* (LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_NUMERIC, LC_TIME) environment variables.

This can be used by applications to determine the language to use for error message and instructions, collating sequences, date formats.

a) **LC_ALL**: this variable determines the values for all locale categories. The value of the LC_ALL environment variable has precedence over any of the other environment variable starting with LC and the Lang environment variable.

b) **LC_CTYPE**: this variable determines the locale category for character handling functions such as tolower(), toupper() and isalpha(). The classification of character (for example, alpha, digit, graph) and the behaviour of character class.

c) **LC_NUMERIC**: this variable determines the locale category for numeric formatting (for example, thousand separator and radix characters) information in various utilities as well as the formatted I/O operation in printf() and scanf().

d) **LC_TIME**: this variable determines the locale category for date and time formatting information. It affects the behaviour of the time function in strftime().

C. **SHELL**: a pathname of the user's preferred command language interpreter. If this interpreter does not conform to the XSI shell command language in the XCU specification, shell command language, utilities may behave differently from those described in this specification set.

D. **USER**: the name of the currently logged in user. This variable is set by the system. You probably should not change its value manually.

E. **DISPLAY**: it is used by the X window system to identify the display server that will be used for input and output. It must be defined if we connect remotely to another machine. In principle the ssh command should take care of defining it correctly, but sometimes it does not. So, you may need to set it manually for instance: set env DISPLAY:0.0

F. **VISUAL**: contains the path to full-fledged editor that is used for more demanding tasks, such as editing mail: eg. vi, vim, emacs. Similar to the "editor" environment variable applications typically try the value in this variable first before falling back to "editor" if it is not set.

SOME EXAPMLES OF COMMONLY USED ENVIRONMENT VARIABLES:

1. **IDENTITY**

- USER, HOME, LOGNAME, HOST

2. **DEFAULT PROGRAMS**

- SHELL, EDITOR, VISUAL

3. INTERNATIONALIZATION (EX. TO FOLLOW)

- Lang, LC_ALL, LC_MONETARY etc.

4. COMMUNICATION WITH SOFTWARE TOOLS

- CC, CFLAGS, LDFLAGS, MAKEFLAGS

5. COMMAND SEARCH PATH, DIRECTORY LIST FOR FILE SEARCH

- PATH (usually connected to the shell variable path or PATH)

TYPES OF VARIABLES

GLOBAL VARIABLES

Global variables or environment variables are available in all shells. The env print env commands can be used to display environment variables. These programs come with the sh-utilities package.

$ print env

global – access global variables. Global variables are variables in the global namespace.

EXAMPLE OF GLOBAL:

global var = value

global varname

LOCAL VARIABLES

Local variables are only available in the current shell. Using set built in command without any options will display a list of all variables (including environment variables) and functions. The output will be sorted according to the current locale and displayed in a reusable format.

$ diff set.sorted print env.sorted | grep"<" | awk '{print $2}'

1. By default, all variables are global.
2. Modifying a variable in a function changes it in the whole script.
3. This can be result into problem.
4. You can create a local variable using the local command and syntax is: local var = value
   local varnam
5. Local command can only be used within a function.
6. It makes the variable name have a visible scope restricted to that function and its children only.

CONCEPT OF SET ENVIRONMENT VARIABLE

1. Set env X_ROOT /some/specified/path
2. Export classpath = ${ClassPath}: ${ravi_dir} /ravi.jar

SOME DEFINITION ABOUT EXPORT

1. View all the current exported variable

$ export -p

2. View a specific exported variable

$ echo $ PATH

3. Set an environment variable

You can add a new environment variable. The following creates a new environment variable called “myapp”.

$ export myapp = 1

$ echo $myapp

4. Append a value to an environment variable

$ export PATH = $ PATH: /home/himanshu/practice/

$ echo $PATH

5. Variables without export

$ myapp = 1

$ cat myapp.sh

#! /bin/bash

echo “MyApp = $MyAppi”

MyAppi = 2

echo “MyApp = $ MyAppi”

6. Now, execute the above script as shown below:

$ ./myapp.sh

MyApp = MyAppi = 2

Set env PATH "~/mydir: $PATH"

A. CONCEPT OF /etc/passwd

etc/passwd file stores essential information, which is required during login i.e. user account information etc/passwd is a text file, that contains a list of the system's accounts, giving for each account some useful information like userID, groupID, home directory, shell etc. it should have general read permission as many utilities, like is use it to map userID to user names, but write access only for super user (root).

UNDERSTANDING FIELDS IN /etc/passwd:

The /etc/passwd contains one entry per line for each user (or user account) of the system. All fields are separated by a colon (:) symbol. Total seven fields as follows:

Generally, passwd file entry looks as follow by the example:

1	2	3	4	5	6	7
Oracle:	X:	1021:	1020:	Oracle user:	/data/network/oracle:	/bin/bash

1. **USER NAME**: It is used for logs in. It should be between 1 and 32 characters in length.
2. **PASSWORD**: An X character indicated that encrypted password is stored in /etc/shadow file.
3. **USER ID (UID)**: Each user must be assigned a user ID (UID). UID 0 (Zero) is reserved for root and UID 1-99 are reserved for other predefined accounts. Further UID 100-999 are reserved by system for administrative and system accounts/groups.
4. **GROUP ID (GID)**: The primary group ID (stored in /etc/group file).
5. **USER ID INFORMATION**: The command field. It allows you to add extra information about the users such as user's full name, phone no., etc. This field use by finger command.

6. **HOME DIRECTORY**: The absolute path to the directory by the user will be in when they log in. if this directory does not exist then users directory becomes /.
7. **COMMAND/SHELL**: The absolute path of a command or shell(/bin/bash). Typically, this is a shell.

SEE USER LIST:

/etc/passwd is only used for local users only. To see list of all users:

$ cat /etc/passwd

To search for a user name called Xguest enter:

$ grep Xguest /etc/passwd

/etc/passwd FILE PERMISSION

The permission on the /etc/passwd file should be read only to users (-rw-r—r–) and the owner must be root:

$ ls -l /etc/passwd

OUTPUT:

-rw-r—r–	1	root	root	2659	Sep 17	01:46	/etc/passwd

READING /etc/passwd FILE:

You can read /etc/passwd file using the while loop and IFS Separator as follows:

Example:

```
while IF =: read -r f1 f2 f3 f4 f5 f6 f7
do
echo “user $f1 use $f7 shell and stores files in $f6 directory”.
done < /etc/passwd
```

B. CONCEPT OF /etc/shadow file

/etc/shadow file stores actual password in encrypted format for user's account with additional properties related to user password i.e. it stores secure user account information. All fields are separated by a colon (:) symbol. It contains one entry per line for each user listed in /etc/passwd file generally. Shadow file entry looks as follows:

/etc/shadow File Fields

1	2	3	4	5	6	7	8
vivek:	1fnfffc$PGteyHdicpGofffxx40w#5:	13064:	0:	99999:	7:	:	:

1. **USER NAME**: it is your login name.
2. **PASSWORD**: it is your encrypted password. The password should be minimum 6-8 characters long including special characters/digits.
3. **LAST PASSWORD CHANGE (LAST CHANGED)**: days since Jan1, 1970 that password was last changed.
4. **MINIMUM**: the minimum no. of days required between password changes i.e. the no. of days left before the user is allowed to change his/her password.
5. **MAXIMUM**: the maximum no. of days the password is valid (after that user is forced to change his/her password).
6. **WARN**: the no. of days before password is to expire that user is warned that his/her password must be changed.
7. **INACTIVE**: the no. of days after password expires that account is disabled.
8. **EXPIRE**: day since Jan 1, 1990 that account is disabled i.e. an absolutes date specifying when the login may no longer be used.

The last 6 fields provide password aging and account lockout features (you need to use change command to set password aging). According to man page of shadow – the password field must be filled. The encrypted password consists of 13 to 24 characters from the 64 characters alphabet a through z, A through Z, 0 through 9, \ & /. Optionally, it can start with a '$' character. This means the encrypted password was generated using another (not DES) algorithm. For example, if it starts with "1" it means the MDS – based algorithm was used.

C. CONCEPT /etc/group

/etc/group is a text file which defines the groups to which users belong under Linux and Unix operating system. Under Unix/Linux multiusers can be categorized into groups. Unix file system permissions are organized into three classes: users, group and others. The use of groups allows additional abilities to be delegated in an organized fashion. Such as access to disks, printer and other peripherals. This method amongst others, also enables the super user to delegate some administrative tasks to normal users.

/etc/group FILE:

It stores group information or defines the user group i.e. it defines the group to which users belong. There is one entry per line and each line has the following format (all fields are separate by a colon (:)).

EXAMPLE:

1	2	3	4
Cdrom:	X:	24:	Priya, student11, Jeet

Where,

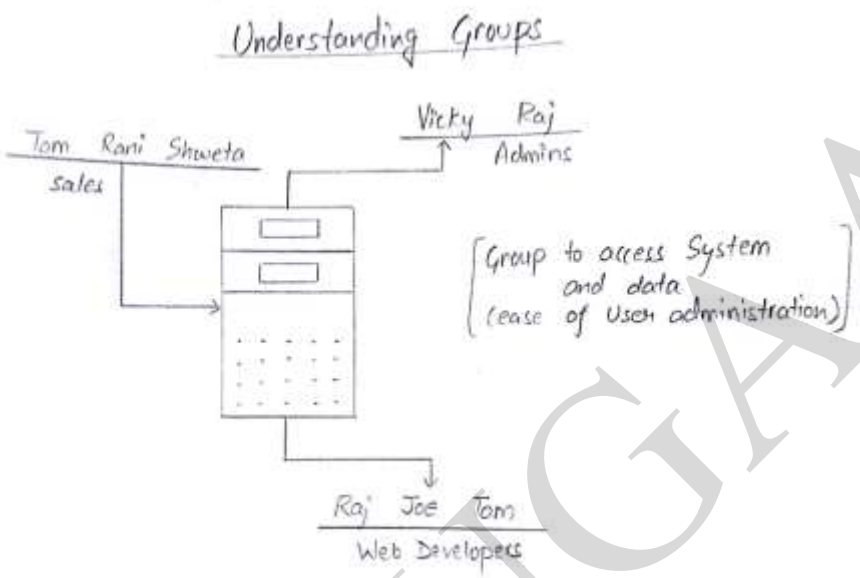
1. **GROUP NAME**: It is the name of group if you run ls -l command. You will see this name printed in the group field.
2. **PASSWORD**: Generally, passwd is not used. Hence it is empty/blank. It can store encrypted password. This is useful to implement privileged groups.
3. **GROUP ID (GID)**: Each user must be assigned a group ID. You can see this no. in your /etc/passwd file.
4. **GROUP LIST**: It is a list of user names of users who are members of the group. The user names, must be separated by commas.

More about user groups

Users on Linux and Unix systems are assigned to one or more groups of the following reasons:

1. To share files or other resource with a small no. of users.
2. Ease of user management.
3. Ease of user monitoring.

4. Group membership is perfect solution for large Linux (Unix installation).
5. Group membership gives you or your user special access to files and directories or devices which are permitted to that group.



User Tom is part of both 'web developers' and 'sales' group. So, Tom can access files belongs to both groups.

View current Groups Setting

Any one of the following command:

$ less /etc/group

Or

$ more /etc/group

Find out the groups a user is in

Type the following command:

$ groups {username}

$ groups

$ groups vivek

OUTPUT:

vivek: vivek adm dialout cdrom plugdev IPadmin netdev admin sambashare libvirtd

Print User/Group Identity

Use the id command to display information about the given user.

Display only the groupID other:

$ id -g

$ id -g user

$ id -g vivek: return groupID

Or

$ id -gn vivek: return group name

Display only the groupID and the supplementary groups enter:

$ id -G

$ id -G user

$ id -G vivek

Or

$ id -Gn vivek: return groupID

D. SU COMMAND

The su command makes it possible to change a login session's owner (i.e., the user who originally created that session by logging on to the system) without the owner having to first log out of that session.

su can be used to change the ownership of a session to any user, it is most commonly employed to change the ownership from an ordinary user to the root user, thereby providing access to all parts of and all commands on the computer or system. It is also sometimes called the switch user command. It is usually the simplest and most convenient way to change the ownership of a login

session to root or to any other user. su is that a record is kept of its usage in system logs, whereas no such record is kept if one logs in directly as root.

SYNTAX

A simplified expression of the syntax of the su command is:

su [options] [commands] [-] [username]

EXAMPLE

1. su

OUTPUT:

Password: root@localhost:/home/John #exit

Logout

2. su Jane

Jane@localhost:/home/John
$exit

SPECIAL PERMISSION

There are lot of good security features built into Linux-based systems, one very important potential vulnerability can exist when local address is granted that is file permission-based issues resulting from a user not assigning the correct permissions to files to files and directories. Some basic permission group are:

1. Owner (u)
2. Group (g)
3. All users (o or a)

Permission Types

1. Read (r)
2. Write (w)
3. Execute (x)

Advanced Permissions

The special permission flag can be marked with any of the following:

–	No special permissions
D	Directory
L	The file or directory is a symbolic link
S	This indicated the setuid/setgid permissions
T	This indicated the sticky bit permission

A. SUID FOR AN EXECUTABLE

This command is used for set user ID which means that you are assigning a special permission on a user owner of any particular file or directory.

Assigning a user SUID means that you are giving him additional permission i.e. user owner permission over the normal permission which he already has to run only executable file inside some directory on with the SUID is applied.

Some common check the super user permission:

$ ls -l /bin/ping

-rwsr-xr-x	1	Root	root	36892	Jul 19 2011	/bin/Ping

$ ls -l /bin/mount

-rwsr-xr-x	1	Root	root	73996	Dec 8 2011	/bin/mount

Suppose you want the extra special permission then we use the SUID Command.

Assigning SUID permission

There are two ways to assign SUID:

1. Octal (u)
2. Symbolic (u+s)

OCTAL METHOD

$ chmod 4755 /m.sh

$ ls -l

-rwsrw-rw-	1	root	root	0	Oct 16	11:33	/m.sh

SYMBOLIC METHOD

$ chmod u+s /m.sh

$ ls -l

-rwsrw-rw-	1	root	root	0	Oct 16	11:34	/m.sh

Removing SUID

OCTAL METHOD

$ chmod 0755 /m.sh

SYMBOLIC METHOD

$ chmod u-s /m.sh

Before applying SUID without executable permission on user owner.

$ chmod 655 /m.sh

$ ls -l

-rw-rw-rw-	1	root	root	0	Oct 16	11:35	/m.sh

After applying SUID without executable permission on user owner.

$ chmod 4655 /m.sh

$ ls -l

-rwsrw-rw-	1	root	root	0	Oct 16	11:35	/m.sh

Finding all the executable files with SUID

$ find / -perm +4000

where, +4000 is the ID we use for assigning permission in octal method.

B. SGID FOR AN EXECUITABLE AND DIRECTORY

This command is used for set group ID. This is a permission assigned to any file or directory to give normal group members additional authority of running that file with a privilege of group owner.

For example, you have some executable file and you want to all the group members of sysadmin to be able to execute it but that file can only be run as root so you assign a SGID over that file and now all the members of sysadmin team will be able to run the file with the permission of root.

Assigning the SGID permission

There are two ways to assign SGID:

1. SGID (2)
2. Symbolic (g+s)

OCTAL METHOD

$ chmod 2755 /m.sh

$ ls –l

-rwxr-sr-x	1	root	root	0	Oct 16	11:33	/m.sh

SYMBOLIC METHOD

$ chmod g+s /m.sh

$ ls -l

-rwxr-sr-x	1	root	root	0	Oct 16	11:33	/m.sh

Removing SGID

OCTAL METHOD

$ chmod 0755 /m.sh

SYMBOLIC METHOD

$ chmod g-s /m.sh

C. STICKY BIT FOR A DIRECTORY

The special permission becomes very useful in most of the cases. This is used when you are the owner of a particular file and you have give full permission to that file for all others but still you don't want any one of them to delete that file apart from the user and group owner. In that case, sticky bit plays a very important role as once you assign this permission to some file or directory. No one else apart from the user and group owner will be able to delete that file or directory. There are two ways to assign sticky bit:

1. Octal (1)
2. Symbolic (t)

OCTAL (1) METHOD

$ chmod 1XXX /m.sh

Here, 1 means assigning sticky bit and XXX means the permission to be applied.

EXAMPLE

$ chmod 1775 /m.sh

Here I am assigning full permission to user and group owner and read and execute permission to others including a sticky bit given be 1 at the beginning of permission.

SYMBOLIC (t) METHOD

If you want to assign sticky bit using symbolic way then this will be the,

SYNTAX

$ chmod +t /m.sh

EXAMPLE

$ chmod 0+t /m

Here I am not meshing with any other existing permission instead additionally I am assigning a sticky bit permission for all others for m directory.

Removing sticky bit

$ chmod 0775 /m

$ ls -l

drwxrwxr-x	3	root	root	4096	Oct 17	07:07	/m

Now, symbolic way of using command:

$ chmod -t /m

TAIL COMMAND

The tail command reads the final few lines of any text given to it as an input and writes them to standard output (which, by default, is the monitor screen).

Unix tail command is a standard way of checking the last lines as well as dynamically watching log files in Unix. It is capable of remembering the last line of the file viewed so only new lines are displayed.

SYNTAX

tail [options] <file_name>

EXAMPLE

Print the last N lines:

To view the last N no. of lines from file. Just pass the file name with -n option as shown below:

$ tail -n hello.txt

Here supposes n = 5 then,

tail – 5 hello.txt

OUTPUT:

b

c

z

e

f

Numbering having leading minus (-) sign or no explicit sign are relative to the end of the input. For example, -n 2 displays the last two lines of the input. The no. starting dash (-) is the synonym to that and also specifies a line offset relative to the end of the file.

Number having a leading plus (+) sign are relative to the beginning of the input, for example -c +2 starts the display at the second line of the input and -c +2 from the second byte.

$ tail +5 hello

/*

HEAD COMMAND

The external head command displays the first few lines of a file. The head command is useful for verifying what type of data is in a file. It can be used to view the first few lines of an ASCII file to visually decide what data is contained in the file. It displays however many lines you specify as an option. If you don't specify the no. of lies to display, head defaults to ten. Its cousins is the tail command.

$ head [-n] file

$ head -5 hello

Here n is the no. of lines to display starting from the beginning of each file. */

WC COMMAND

The wc command by defaults counts the no. of lines, words and characters in text.

wc defines a word as a set of contiguous letters and/or symbols which are separated from other characters by one or more spaces, tabs and/or newline characters (which are generated when the return key is pressed). When counting the no. of characters, all characters are counted not only letters, numbers and symbols, but also spaces, tabs and newline characters. A line is only counted if it ends with a newline character.

wc syntax is:

wc [options] [filename]

EXAMPLE

wc myfile.txt

OUTPUT:

5 13 57 myfile.txt

Where 5 is the number of lines, 13 is the number of words, and 57 is the number of characters.

OPTIONS

W	Gives only the word count
L	Gives only the line count
C	Gives only the byte
M	Gives only the character count
L	Gives only the length of the longest line
S	Gives only the symbol

EXAMPLE: wc -wlm hello

OUTPUT:

hello –

a

b

c

d

e

f

g

h

Where, 8 is the number of lines, 8 is the number of words, and 8 is the number of characters.

SORT COMMAND

SORT command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ASCII. Using options in sort command, it can also be used to sort numerically. Let us discuss it with some examples:

Let the following examples:

$ cat file1

Unix

Linux

Solaris

AIX

LINUX

HPUX

1. Sort simply sorts the file in Alphabetical order.

$ sort file1

AIX

HPUX

LINUX

Linux

Solaris

Unix

2. Sort removes the duplicates using -u option.

$ sort -u file1

AIX

HPUX

Linux

Solaris

Unix

FILE WITH NUMBERS

$ cat file2

20

19

5

49

10

$ sort file2

5

10

19

20

49

3. Sort a file numerically.

$ sort -n file2

5

19

20

49

4. Sort file numerically in reverse order.

$ sort -nr file2

49

20

19

10

5

MULTIPLE FILES

$ cat file 4

20

19

5

49

50

$ cat file 5

18

24

32

56

5

5. Sort can sort multiple files.

```
$ sort -n file4 file5
5
5
18
19
20
24
32
49
50
56
```

6. Sort, merge and remove duplicates.

```
$ sort -nu file4 file5
5
18
19
20
24
32
49
50
56
```

FILES WITH MULTIPLE FIELD AND DELIMITER

$ cat file6

Linux, 20

Unix, 30

AIX, 25

Linux, 31

Solaris, 14

HPUX, 11

7. Sort file6

AIX, 25

HPUX, 11

Linux, 20

Linux, 31

Solaris, 14

UNIQ COMMAND

Uniq command is helpful to remove or detect duplicate entries in a file. Uniq command discard all the successive identical lines except one from the input and writes the output. If duplicate lines in a file are not adjacent to one another, uniq will not treat them as duplicates:

SYNTAX

uniq [option] filename

The options of uniq command are:

C	Count of occurrence of each line
D	Prints only duplicate lines

D	Prints all duplicate lines
F	Avoid comparing first N fields
I	Ignore case when comparing
S	Avoid comparing first N characters
U	Prints only unique lines
W	Compare no more than N characters in lines

EXAMPLE

Cat examp1.txt

Unix operating system

unix operating system

unix dedicated server

linux dedicated server

1. Suppress duplicate lines:

The default behaviour of the uniq command is to suppress the duplicate line. You have to pass sorted input to the uniq, as it compares only successive lines.

EXAMPLE

uniq examp1.txt

unix operating system

unix dedicated server

linux dedicated server

COUNT OF LINES

The -c option is used to find how many times each line occurs in the file. It prefixes each line with the count.

$ uniq -c examp1.txt

2 unix operating system

1 unix dedicated server

1 linux dedicated server

Display only duplicate lines:

You can print only the lines that occur more than once in a file using -d option.

EXAMPLE

uniq -d examp1.txt

unix operating system

Print only unique line:

You can skip the duplicates lines and print only unique lines using the -u option.

uniq -u examp1.txt

unix dedicated server

linux dedicated server

CUT COMMAND

cut command in Unix (Linux) is used to select sections of text from each lines of files. You can use the cut command to select fields or columns from a line by specifying a delimiter or you can select a portion of text by specifying the range or characters. Basically, the cut command slices a line and extracts the text. Remove sections from each line of files.

SYNTAX

cut [options] [file]

OPTIONS

-b	Output only the bytes from each line as specified in LIST.
-c	Output only the characters from each line as specified in LIST.

-d	Use character DELIM instead of a tab for the field delimiter.
-f	Output only these fields on each line; also print any line that contains no delimiter character, unless the -s option is specified.

EXAMPLE

cat file1.txt

unix or linux OS

is unix good OS

is linux good OS

1. Write a unix/linux cut command to print characters by position?

The cut command can be used to print characters in a line by specifying the position of the characters. To print the characters in a line, use the -c option in cut command

cut -c4 file.txt

x

u

l

The above cut command prints the fourth character in each line of the file. You can print more than one character at a time by specifying the character positions in a comma separated list as shown in the below example

cut -c4,6 file.txt

xo

ui

ln

This command prints the fourth and sixth character in each line.

2. Write a unix/linux cut command to print characters by range?

You can print a range of characters in a line by specifying the start and end position of the characters.

cut -c4-7 file.txt

x or

unix

linu

The above cut command prints the characters from fourth position to the seventh position in each line. To print the first six characters in a line, omit the start position and specify only the end position.

TR COMMAND

tr is an abbreviation of translate or transliterate, indicating its operation of replacing or removing specific characters in its input data set. In Unix tr is a utility for translating or deleting or squeezing repeated characters. It will read from STDIN and write to STDOUT.

EXAMPLE

tr command use as a lower to upper:

$ echo 'welcome to the land of linux' | tr "[:lower:]" "[:upper:]"

OUTPUT

WELCOME TO THE LAND OF LINUX

Deleting specific character

The "-d": parameter can be used to delete character with ease:

$ echo "This is another test" | tr -d 'a'

OUTPUT

This is nother test.

White space to tabs

$ echo "This is a test" | tr [:space:] 'It'

OUTPUT

This is a test

DIFF COMMAND

In the simplest case, diff compares the contents of the files from -file and to -file. It is used to differences between two files. It is typically used to show the changes between one version of a file and a former version of the same file. It displays the changes made per line for text files.

SYNTAX

diff [options..] from file to file

OPTIONS

-a	Treat all files as text and compare them line by line
-b	Ignore changes in amount of white spaces
-w	Ignore white space when comparing lines

File1.txt

HIOXTEST

hscripts.com

with friendship

hioxindia.com

File2.txt

HIOXTEST

HSCRIPTS.com

with friendship

1. Compare files ignoring white spaces:

diff -w file1.txt file2.txt

This command will compare the file file1.txt with file2.txt ignoring white/blank space and it will produce the following output:

2c2

< hscripts.com – – – > HSCRIPTS.com

ASPELL COMMAND

aspell is a utility program that connects to the Aspell library so that it can function as an ispell -a replacement, as an independent spell checker, as a test utility to test out Aspell library features, and as a utility for managing dictionaries used by the library.

The Aspell library contains an interface allowing other programs direct access to its functions and therefore reducing the complex task of spell checking to simple library calls.

EXAMPLE

List spelling mistakes in a file using aspell command:

cat Linux1

linux kernel is an open source software.

aspell list < linux1

where, list is the software.

BASIC SHELL SCRIPTS

GREP COMMAND

grep is a command-line utility for searching plain-text data sets for lines that match a regular expression. Grep was originally developed for the Unix operating system, but later available for all Unix-like systems. grep was created by Ken Thompson as a standalone application adapted from the regular expression parser he had written for ed.

Selecting all lines containing the self-standing word apple i.e. surrounded by white space or hyphens, may be accomplished with the option flag w.

Extract line match is performed with the option flag x. Lines only containing exactly and solely apple are selected with a line -regexp instead of word -regexp.

EXAMPLE

cat fruitlist.txt

apple

apples

pineapple

apple-

apple-fruit

fruit-apple

grep -x apple fruitlist.txt

apple

The v option reverses the sense of the match and prints all lines that do not contain apple. As in this example:

grep -v apple fruitlist.txt

banana

pear

peach

orange

SED COMMAND

SED command in UNIX is stands for stream editor and it can perform lots of function on file like, searching, find and replace, insertion or deletion. Though most common use of SED command in UNIX is for substitution or for find and replace. By using SED, you can edit files even without opening it, which is much quicker way to find and replace something in file, than first opening that file in VI Editor and then changing it.

The one we use below is g which means “replace all matches”.

EXAMPLE

cat file

I have three dogs and two cats.

sed -e ‘s/dog/cat/g’ -e ‘s/elephants/g’ file

I have three elephants and two elephants

-e option to sed tells it to use the next item as the sed command.

DELETE COMMAND

$ cat file

http://www.abcde.com/my1page.html

$ sed -e ‘s@http://www.abcde.com@http://www.baaar.net@’ file

http://www.baaar.net/my1page.html

$ cat file1

The black cat was chased by the brown dog.

$ sed -e ‘s/black/white/g, file1

The white cat was changed by the brown dog.

AWK COMMAND (Basic usage)

Awk is one of the most powerful tools in Unix used for processing the rows and columns in a file. Awk has built in string functions and associative arrays. Awk supports most of the operators, conditional blocks, and loops available in C language. Some of the key features of Awk are:

- Awk views a text file as records and fields.
- Like common programming language, awk has variables, conditions and loops.
- Awk has arithmetic and string operators
- Awk can generate formatted reports.

EXAMPLE

Let's create a file file1.txt and let it have the following data:

Data in file1.txt

14	15	16
15	15	11
5	56	6
5	25	1

1. To print the second column data in file1.txt

awk '{print $2}' file1.txt

this command will manipulate and print second column of text file (file1.txt). the output will look like:

15
15
56
25

2. To multiply the column -1 and column -2 and redirect the output to file2.txt:

awk '{print $1, $2, $1 * $2}' file1.txt > file2.txt

COMMAND EXPLANATION

$1	Prints 1st column
$2	Prints 2nd column
$1 * $2	Prints result of $1 * $2

file1.txt	Input file
>	Redirection symbol
fFile2.txt	Output file

The above command will redirect the output to file2.txt and it will look like.

14	15	210
15	15	225
5	56	280
5	25	125