UNIT - III

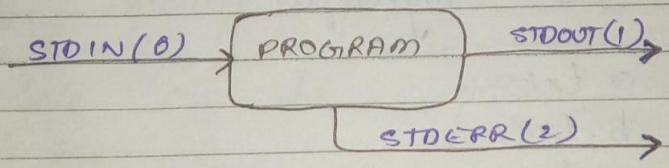
I/O Redirection

Every program we run on the command line automatically has 3 data streams connected to it:—

(i) STDIN(0) - standard input

(ii) STDOUT(1) - standard output

(iii) STDERR(2) - standard error message



Piping and redirection is a way by which we can connect these streams between programs and files to direct data in interesting and useful ways.

Redirecting to a file

Normally we get output on the screen. But if we want to save op as a file to use as a record or want to send that to someone else, then redirection is useful

→ The > (greater than operator) indicates to the command line that we want to save output in a file,

for eg:- ls > myoutput

```
1. $: ls
2   a.txt- b.txt- c.txt- d.txt-
3 $: ls > myoutput-
4 $: ls
5   a.txt- b.txt- c.txt- d.txt-

6 $: cat- myoutput-

7 $: a.txt-
8.   b.txt
9.   c.txt-
10.  d.txt-
11. $:
```

Line 1:- Let's see what our current directory contains with the help of ls command.

Line 2:- o/p of ls command on terminal.

Line 3:- Here we are running ls command with redirection

operator >, to save the o/p of ls command in myoutput file.

→ note :- we don't need to create file, the terminal will create it automatically.

Lin 4 :- our new file has been created.

Line 6 :- o/p of myoutput file.

NOTE during piping and redirection the actual data will be same but formatting of data can be slightly different.

→ The mechanism works as follows -

① the file is created first (if the file is not exist)

② then the program is run and o/p saved into the file.

Saving to an existing file

If we save the o/p to an existing file then the contents of the file will be cleared and new o/p saved in that file.

for eg:-

$: cat- myoutput-

a.txt-
b.txt-
c.txt-
d.txt-

$: wc -l a.txt- > myoutput-

$: cat- myoutput-

6 a.txt-

(3) instead of getting new data we can append to the file by using double greater than operator (>>)

1. $: cat- myoutput-

6 a.txt-

2. $: ls >> myoutput-

3. $: cat- myoutput-

6 a.txt-
a.txt-
b.txt-
c.txt-

Redirecting from a file

If we use the less than operator (<) then we can read data in other way.

We will read data from the file and feed it into the program via its STDIN stream.

for eg :- $: wc -l a.txt

O/P 8 a.txt

$: wc -l < a.txt

O/P 8

combination of < and >

If required we can combine the two forms of redirection to a single command.

for eg :- $: wc -l < a.txt > my output

O/P $: cat myoutput

O/P 7.

Redirecting "STDERR"

The third stream is STDERR.

The three streams have numbers associated with them. (0 - STDIN, 1 - STDOUT 2 - STDERR).

We can use these numbers to identify the streams.

If we place a number before > operator then it will redirect that streams.

If we don't use any number then by dfault it is stream 1.

for eg :-

1. $: ls -l a.txt- b.txt-

No such file or directory ~~exist~~
-rwxr--r--1 ryan user 6 sep 17 8.10 a.txt-

2. $: ls -l a.txt- b.txt- 2> e.txt

3. $: cat- e.txt-

o/p No such file or .....

- may be some time we want to save both error message and normal o/p into a single file.

→ This can be done by redirecting the stderr stream to stdout stream and redirecting stdout to output file.

for eg :- ls -l a.txt > myoutput 2>&1 (files)

$: cat files.

for eg / command > out.txt 2> error.txt

command $> out.txt

Both standard error and standard o/p of command will be saved in out.txt.

Q what is piping in linux?

Ans.

- PIPE :- pipe is a mechanism in which the output of one command can be redirected as input to another command.

The general format is -

command1 | command2

The pipe command feed the output of command1 to the input of command2.

for eg :-

$ ls | more

The output of command ls is sent to the 'more' command as input.

we may pipe as many programs together as we like.

Examples

① bash: ls -l /etc | tail -n +2 | sort

② $: ls -l /etc | less

③ identify all files in your home directory which the group has write permission

$: ls -l ~ | grep '^....w'

UNIT - III

simple filter commands

These are some Linux commands that accept input from std input as file and produces o/p to the std output. Since these commands perform some filtering operation on data, they are called filters.

These filters are used for different - different tasks like -

a) Display the contents of file in sorted order

b) Merge contents of 2 files

c) Extract lines from a specified file

d) Handle duplicate lines in a file etc.

Some filters commands are as follows -

[1] GREP

grep command is used to search a specified pattern in a specified file. The command will display all the lines on o/p screen having specified pattern.

general format is -

grep [-option] pattern filename

Note enclose the pattern in single quote if pattern contains any shell special characters.

where options can be -

-b → Ignores spaces, tabs

-i → Ignores case (small/capital)

-v → Display only those lines that do not match specified pattern

c → displays total no. of occurrence of the pattern in the file

n → Displays resultant lines with their line no.

<number> → displays the matching lines along with <number> of lines above and below.

We can use regular expression characters with grep command. like -

* → Matches zero or more

. → matches single character

[r1-r2] Matches a single character within the Ascii range represented by the characters.

[^ abcd] Matches a single character which is not abcd

^<character> Matches are ~~characters~~ lines that are beginning

<character>$ → ending with.

with the character specified in <character>

for eg:-

$ grep "com*" file.dat

$ grep "B$" file.dat

$ ls -l | grep "^d"

egrep :- extended global search for regular expression.

→ Multiple pattern can be search by using pipe symbol (|)

→ for example -

$ egrep "raj|shyam" file

12 raj computer
14 shyam commerce

fgrep :- (fixed grep)
Regular expressions are not allowed.

for eg :-

$ fgrep "cs" file.dat] ✓

$ fgrep "c*" file.dat] ✗

***

[2] HEAD command

→ This command is used to display the top of the specified file.

→ General syntax is -

head [-n] <file name>

→ If -n option is used then the first n lines of specified file are displayed.

→ Default value for n is 10.

for example.

$ head -3 file1

or

$ head file1.

[3] Tail :- displays the end of specified file.

general syntax is -

tail ±n <filename>

+n → nth line to end of the file

-n → last n line of the file.

by default - last 10 lines are displayed.

for example :-

$ tail +4 file1

$ tail -5 file1

$ tail file1

[4] cut command

→ used to cut the column / field of a specified file -

→ the general syntax is -

cut [-option] <filename>

→ where options can be -

c <columns> cuts the column specified in <columns>.

→ seperate the column no. by using comma.

f <field> cuts the field specified in <field>.

- seperate fields no by using commas

for example :-

$ cut -f 3-5 file1

$ cut -f 1-3,5 file1

$ cut -c 1-4 file1

$ cut -c 6-10, 15, 17 file1

[5] PASTE :- → Paste command merges the files.

→ This command concatenates of the specified files into a single file vertically.

→ general format is -

paste <filename1> <file2>

```
$ paste file1 file2
```

[6] SORT :- sort the contents of a given file based on ASCII values of characters.

General Syntax is -

sort [options] <filename>

where options can be -

-m <filelist>
merges sorted files specified in <filelist>

-o <filename>
stores output in the specified <file name>

-r sorts the contents in reverse order (reverse alphabetical order)

-u Removes duplicate lines and display sorted content

-n Numeric sort

-t "char" uses the specified "char" as delimiter to identify fields

-c checks if the file is sorted or not

+pos starts sort after skipping the pos^th field

-pos stops sorting after pos^th field

+pos.n starts sort after the n^{th} column of the $(pos+1)^{th}$ field

-pos.n stops sort on the n^{th} column of the $(pos+1)^{th}$ field.

[6] uniq → This unique command is used to handle duplicate lines in a file.

If this command is used without any option it displays the lines by eliminating duplicate lines.

general format is -

uniq [-option] filename

where options can be -

u Displays only the non repeated lines

d Displays only the duplicated lines

c Displays each line by eliminating duplicate lines and prefixing the number of times it occurs.

* o *

Process :-

A process is simply an instance of a running program. A process is said to be born when the program starts execution, and remains alive as long as the program is active.

The process is identified by the process-id (PID), and every process has a parent whose PID is also available (PPID). The login shell is the only user process that keeps running as long as the user logged in. Its PID is stored in the parameter $$.

[A] THE sh process

When we log in to a system, a process is immediately set up by the kernel.

This is technically a command which may be sh. Any command which we type at the command prompt is input to the shell (sh) process.

and this process remains alive until we log out.

* $ echo $$

It will give process numbes of the current shell.

The PID of your login shell does not change as long as we are logged in.

when you log out and logged in again, your login shell will be assigned a different PID.

Back Ground Process :-

Linux provides facility for background processing.

It means when one process is running in the foreground, another process can be executed in the background.

To run a command

in background simply put & (ampersand) symbol at the end of a command.

for example:-

$ sort a.c &

when we execute this command a number is displayed. This is called PID no.

In Linux PID can range from 0 to 32767.

The command "sort a.c" will run in background. we can execute another command in foreground.

Premature termination of process:

If we want to terminate a command pre-maturely then press [ctrl+c].

This type of interrupt does not affect background

processes, because the background processes are protected by the shell from these interrupt signals.

The "kill" command is used to terminate a background process.

General format is -

```
kill [-signal Number] <PID>
```

The PID is the process identification number of the process that we want to terminate.

→ use PS command to know the PIDs of the current processes.

By default this kill command uses the signal number 15 to terminate a process.

→ for sure killing use signal no: 9.

for example:-

$ kill 120

$ kill -9 130

[3] NOHUP command :

If a user wants that a process must not die even when he/she logged-out from the system, then we can use nohup command.

general format is

nohup <command>$

for example -

$ nohup sort a.c$

The command sort a.c will continue after we logged out from the system.

The o/p of this command is saved to nohup.out file.

[4] Process Priorities :-

when we talk about process priority, it is all about managing processor time.

In Linux we can set guidelines for the CPU to follow when it is looking at all the task it has to do.

a) checking the priority of Running Pro.

```
ps -o pid, comm, nice, -p 594
```

This will output the process ID command, command and nice value.

b) setting priority on new processes

To change the priority when issuing a new command we do.

nice -n [nice value] [command]:

nice -n 10 apt-get upgrade

this will increment the defau-

nice value by a positive 10 for the command 'apt-get upgrade'.

c) Setting Priority on Existing Process

To change the priority of an existing process, use renice.

Syntax :-

renice [nice value] -p [process_id]

renice 10 -p 101

This will increment the priority of the process with an id of 101 to 10.

(5) Process Scheduling

Process scheduling and

process priority is same.

[5] compiling c prog. in linux environment :-

To compile a c prog. on any linux distribution like ubuntu, Red Hat, Fedora, Debian and other, we need to install -

1) GNU C and C++ Compiler collection
2) Development tools
3) Development libraries
4) IDE to text editor to write prog.

→ create a file called a.c using a text editor for eg vi or gedit -

→ use following syntax to compile and run -

cc prog. source code.c -o executable
↳ o/p file

for eg:

```
cc a.c -o b
```