Lecture – 16

SECTION -C

Getting Started with UNIX

Introduction

- Pipes
- UNIX Redirection : the three standard files
- Processes : An introduction

PIPES

- Standard input and standard output constitute two separate streams that can be individually manipulated by the shell.
- If that be so, cant the shell connect these streams so that one command take input from the other?
- You know the who command produces a list of users, one user per line. Lets use redirection to save this output in a file:

```
 $ who > user.txt
$ cat user.txt
  root              console      Aug 21 07:51          (:0)
  projectpts/8        Aug 30 07:51          (pc125.heavens.com)
 xyz                pts/14       Aug 11 07:51           (pc125.heavens.com)
 abc                pts/12       Aug 1 07:51           (pc125.heavens.com)
$ wc – l <user.txt
        4                       Counts the number of user.
```

- Here, who's standard output was redirected, and so was wc's standard input, and both used the same disk file. The shell can connect these streams using a special operator, the | (pipe) , and avoid creation of the disk file.
- You can make who and wc work in combination so that one takes input from the other:

$ who | wc-l **no intermediate files created.**

5

- **Here the output of the who has been passed directly to the input of wc, and who is said to be piped to wc.**
- **When multiple commands are connected this way, a pipeline is said to be formed.**
- **It's the shell that set up this connection and the commands have no knowledge of it.**
- **The pipe is the third source and the destination of the standard input and standard output, respectively.**
- **You can now use one to count the number of the files in the current directory:**
- **$ ls | wc – l**

15

Redirection : THE THREE STANDARD FILES

TERMINAL: The terminal is a generic name that represents the screen, display or keyboard. Just as we refer to a directory as file; we'll also sometimes refer to the keyboard as terminal.

- We see a command output and error messages on the terminal(display) , and we sometimes provide the command input through the terminal (keyboard).
- The shell associates three files with the terminal – two for display and one for the keyboard.
- Even though our terminal is also represented by specific device name (/dev/tty), commands don't usually read from or write to this file. They perform all terminal related activity with the three files that the shell makes available to every command.

- These special files are actually a stream of characters which many commands see as Input & Output.
- A stream is simply a sequence of bytes. When a user logs-in, the shell makes available three files representing three streams.

(1) **Standard Input** – the file or stream representing Input, which is connected to the keyboard.

(2) **Standard Output –** the file or stream representing output; which is connected to the display.

(3) **Standard Error** - the file (or stream) representing error messages that originate from the command or shell. This is also connected to the display.

Standard Input

- We've used the cat & wc command to read disk files. These commands have an additional method of taking input. When they are used without arguments , they read the file representing the standard input.
- This file is indeed special; it can represent three input sources –
  - ✓ The keyboard, the default source
  - ✓A file using redirection with the < symbol.
  - ✓Another program using a pipeline.
- When you use wc without any argument & have no special symbols like < and | in the command line, wc obtain its input from the default source. You've to provide this input from the keyboard & mark the end of the input with [Ctrl – d]

$ wc

Standard input can be redirected.

It can come from a pipeline or from a file.

[Ctrl – d]

3 (l) 14 (w) 71(c)

- It can reassign the standard input file to a disk file. This meant it can redirect the standard input to originate from a file or a disk.
- This reassignment or redirection requires the < symbol.

$ wc < sample.txt (file containing the above three lines)

3 14 71

- The file name is missing once again, which means that “wc” doesn’t open sample.txt . It read the standard input file as a stream but only after the shell reassigned this

Standard Output

- All commands displaying output on the terminal actually write to the “standard output” file as a stream of characters, and not directly to the terminal as such.
- There are three possible destinations of this stream :
  - ✓ The terminal, the default destination
  - ✓ A file using the redirection symbol > and >>
  - ✓ As input to another program using a pipe.
- The shell can effect redirection of this stream when it sees the > or >> symbols in the command line.
- You can replace the default destination (the terminal) with any file by using > (right chevron) operator followed by the file name :

$ wc sample.txt > newfile

$ cat newfile

3 14 71 sample.txt

- The first command sends the word count of the sample.txt to newfile; nothing appears on the terminal screen. If the output doesn't exist; the shell creates it before executing the command. If it exists the shell overwrites it, so use this operator with caution.
- The shell also provides the >> symbol (the right chevron used twice) to append to a file.

$ wc sample.txt>> newfile (doesn't disturb existing content)

Standard Error

- Each of the three standard files is represented by a number called – file descriptor. A file is opened by refering to its pathname, but subsequent read & write operations identify the file by this file descriptor.
- The kernel maintains the table of the file descriptors for every process running in the system. The first three slots are generally allocated to the three standard streams in this manner

  0 – Standard Input

  1 – Standard Output

  2 – Standard Error

- These descriptors are implicitly prefixed to the redirection symbols. For instance > and >1 mean the same thing to the shell ; while < and <0 also are identical.
- We need explicitly use one of these descriptors when handling the standard error stream. When you enter an incorrect command or try to open an nonexistent file, certain diagnostic messages show up on the screen. This is the standard error stream whose default destination is the terminal Trying to “cat” a nonexistent file produces the error stream:

$ cat file1

cat : cannot open file1

Cat fails to open the file & writes to standard error.

Processes

- 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.
- After execution is complete the process is said to **die**.
- A process also has a name, usually the name of the program being executed.
- The kernel is responsible for the management of processes. It determines the time and priorities that are allocated to processes so that multiple processes are able to share CPU resources.
- It provides a mechanism by which a process is able to execute for a finite period of time and then relinquish control to another process.

- Files have attributes and so do processes. Some attributes of every process are maintained by the kernel in memory in a separate structure called the **process table.** We can say that process table is the inode for processes.

  Two important attributes of a process are:

- The Process –id (PID) : Each process is uniquely identified by a unique integer called the **Process –id (PID)** that is allocated by the kernel when the process is born. We need this PID to children separately.
- The Parent PID (PPID) : The PID of a parent is also available as a process attribute. When several processes have the same PPID, it often makes sense to kill the parent rather than all its children separately.

The shell Process

- When you log on to a UNIX system, a process is immediately set up by the kernel. This process represents a UNIX command which may be **sh(**Bourne shell), **ksh** (Korn shell), **csh** (C shell) or **bash** (Bash). Any command that you key in is actually the standard input to the shell process. This process remains alive until you logout, when it is killed by the kernel.
- The shell's pathname is stored in SHELL, but its PID is stored in a special "variable", **$$.** To know the PID of your current shell, type :

$ echo $$ ***The PID of the current shell.***

291

The PID of your login shell can't obviously change as long as you are logged in. A low PID indicates that the process was initiated quite early. When you log out and log in again, your login shell will be assigned a different PID. Knowledge of the

Process Status

- Let's use **ps** command to display some process attributes. The ps command can be seen as the process counterpart of the file system's **ls** command.
- The command reads through the kernel's data structures and process tables to fetch the characteristics of process.
- By default, **ps** displays the process owned by the user running the command. If you execute the command immediately after logging in, what you see could look like this:

$ ps

PID TTY TIME CMD

291 console 0:00 bash *The login shell of this user*

- Ignoring the header, each line (here, only one) shows PID, the terminal (TTY) with which the process is associated (the controlling terminal), the cumulative processor time (TIME) that has been consumed since the process has been started, and the process name (CMD).
- You can see that your login shell (**bash**) has the PID 291, the same number echoed by the special variable

Options to ps:

Process Options	Significance
- f	Full listing showing the PPID (Parent Process ID)
- e or – A	All processes including user and system processes
- u usr	Processes of user usr only
- a	Processes of all users excluding processes not associated with terminal.
- l	Long listing showing memory related information.
-t term	Processes running on terminal term (say , /dev/console)

System Processes : (- e or – A)

- Apart from the processes a user generates, a number of system processes keep running all the time.
- Most of them are not associated with any terminal (having no controlling terminal .
- They are initiated during system start up and some of them start when the system goes to the multiuser state.
- To list all processes running on your machine, use the –e or –A option to **ps** . On a busy system, this list could be very long ; we produce below a short list showing some important system processes.

$ ps –e

PID	TTY	TIME	CMD	
0	?	0:01	sched	*Takes care of swapping*
1	?	0:00	init	*Parent of all shells*
2	?	0:00	pageout	*Part of the kernel – not executed*
3	?	4:36	fsflush	*Part of the kernel – not executed*
194	?	0:00	sendmail	*Handles all your mails*
2931	?	0:00	in.telne	*Serves your TELNET requests.*

System processes are easily identified by the ? In the TTY column: they generally don't have a **controlling terminal.** This means that the standard input and standard output of

Applications

To the great process pool in the sky

- Some processes live forever (such as init), and some processes reincarnate themselves into a new form (such as your shell). Ultimately, most processes die of natural causes -- a program runs to completion.
- Additionally, you can place a process in a kind of suspended animation, where it waits to be reanimated. And as the previous example shows, you can terminate a process prematurely with kill.
- If a command is running in the foreground and you want to suspend it, press **Control-Z**:

```
$ sleep 10
(Press Control-Z)
[1]+ Stopped sleep 10
 $ ps
 PID PPID USER COMMAND S STIME TIME 18195 16351 mstreic sleep 10 T
```

Research : More magic demystified

- UNIX has many moving parts. It has system services, devices, memory managers, and more. Luckily, most of these complex machinations are hidden from view or are made convenient to use through user interfaces, such as the shell and windowing tools. Better yet, if you want to dive in, specialized tools, such as top, ps, and kill, all are readily available.
- Now that you know how processes work, you can become your own one-person band. Just one request: *Freebird!*
- Resources
- **Learn**
- Speaking UNIX: Check out other parts in this series.
- AIX and UNIX: The AIX and UNIX developerWorks zone provides a wealth of information relating to all aspects of AIX systems administration and expanding your UNIX skills.
- New to AIX and UNIX?: Visit the New to AIX and UNIX page to learn more about AIX and UNIX.
- AIX 5L™ Wiki: A collaborative environment for technical information related to AIX.
- Check out other articles and tutorials written by Martin Striecher:
  - Across developerWorks and IBM
- Search the AIX and UNIX library by topic:
  - System administration
  - Application development
  - Performance
  - Porting
  - Security
  - Tips
  - Tools and utilities
  - Java™ technology
  - Linux
  - Open source
- Safari bookstore: Visit this e-reference library to find specific technical resources