CHAPTER 8

Making New Commands by Joining Commands

We have freedom to create new commands by joining the existing UNIX commands. Of course, we can mix up our program also which are developed from C or C++. Evidently, alias, piping and tee commands are used for this purpose.

8.1 THE ALIAS COMMAND

Essentially*, Aliases* allow a string to be substituted for a word when it is used as the first word of a simple command. The shell maintains a list of aliases that may be set and unset with the alias and unalias builtin commands. For example, in some Linux versions we can execute dir and feel like as if we are working under DOS. This became possible by declaring an alias in /etc/profile file (see the content of the file /etc/profile).

The first word of each simple command, if unquoted, is checked to see if it has an alias. If so, that word is replaced by the text of the alias. The characters '/', '$', '`', '=' and any of the shell metacharacters or quoting characters listed above may not appear in an alias name. The replacement text may contain any valid shell input, including shell metacharacters. The first word of the replacement text is tested for aliases, but a word that is identical to an alias being expanded is not expanded a second time. This means that one may alias ls to "ls -F", for instance, and Bash does not try to recursively expand the replacement text. If the last character of the alias value is a space or tab character, then the next command word following the alias is also checked for alias expansion.

Aliases are created and listed with the alias command, and removed with the unalias command. This gets expanded in interactive shells only. In C shell, aliases can be made to take arguments.

The rules concerning the definition and use of aliases are somewhat confusing. Bash always reads at least one complete line of input before executing any of the commands on that line. Aliases are expanded when a command is read, not when it is executed. Therefore, an alias definition appearing on the same line as another command does not take effect until the next line of input is read. The commands following the alias definition on that line are not affected by the new alias. This behavior is also an issue when functions are executed. Aliases are expanded when a function definition is read, not when the function is executed, because a function definition is itself a compound command. As a consequence, aliases

defined in a function are not available until after that function is executed. To be safe, always put alias definitions on a separate line, and do not use alias in compound commands.

We can define new commands with alias facility. For example, the following instruction at the shell prompt makes an alias to be defined for ls –l command.

alias dir='ls –l'

Now, if type dir at the $ prompt (as shown below), we may get ls –l to be executed.

dir

That is, alias command format is

alias name = text

Similarly, let us have defined the following aliases :

alias NF='ls'

alias NN='ls –al'

alias h=history

If we run simply alias command with out any arguments, it displays all the aliases defined in this system currently. For example,

alias

give output as :

alias dir='ls –l'

alias N='ls'

alias NN='ls -al|wc -l'

alias h='history'

We have an option –p with alias command which also displays all the aliases available currently.

In C shell, to define an alias we need not required to use =. For example the following command defines alias N in C shell

alias N ls

If we execute alias command followed by an already defined alias name, it displays that alias name and value. For example,

alias N

gave

alias N='ls'

As alias PN is not defined by us, we get error for the following command.

alias PN

We have N, PN and MN aliases are defined as follows. That is, we can have one alias to another and vice versa.

alias
alias MN='PN'
alias N='ls'
alias PN='N'

N

```
! F3 aa abc bbb employee nbv pp simham testac testaf
F1 a1 aaa abcc bbbb file1 pal ppp testaa testad testag
F2 a2 aaaa bb bbbbb mmmm passwd pppp testab testae xx
```

$ PN

```
! F3 aa abc bbb employee nbv pp simham testac testaf
F1 a1 aaa abcc bbbb file1 pal ppp testaa testad testag
F2 a2 aaaa bb bbbbb mmmm passwd pppp testab testae xx
```

$ MN

```
! F3 aa abc bbb employee nbv pp simham testac testaf
F1 a1 aaa abcc bbbb file1 pal ppp testaa testad testag
F2 a2 aaaa bb bbbbb mmmm passwd pppp testab testae xx
```

The unalias command

We can remove an alias by executing unalias command. For example, if we assume that we happened have defined dir, N, NN, and h aliases as above and want to remove alias NN then we can execute

unalias NN

After this if we run alias command, we may get the following output :

alias dir='ls –l'
alias N='ls'
alias h='history'

If we want to remove all the aliases which are currently available, we can execute

unailas –a

For example, the following aliases we use commonly.

alias nooflines="wc –l"
alias nooffiles="ls –l | wc-l"
alias directories="echo $PATH|tr : \\\n"
alias noofemptydir="find . –type d –empty | wc –l "

8.2 INTRODUCTION TO PIPES

UNIX operating system supports a unique approach through which we can join two commands and generate new command with the help of pipe concept.

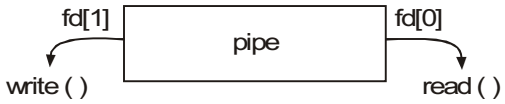
For example

command1|command2

Here, whatever output which first command generates becomes standard input for the second command. We can develop complex UNIX command sequences by joining many commands while maintaining this input output relation ships. Whenever left hand side of piping symbol does not generate any standard output, we may get an error "broken pipe".

OS Background

In operating systems point of view, a special file known as FIFO (First In First Out) is created with two descriptors by calling a system call known as pipe(). The left hand side command becomes as a process and sends its standard output through writing descriptor to this pipe or FIFO file. Also, the right hand side command also becomes another process and reads from the same pipe file using reading descriptor as shown in the following Figure.



For example

ls -l|grep "^d"

This command displays details of only the directories of the current working directory. That is output of ls -l command becomes input to grep command which displays only those lines which starts with d (they are nothing but details of files).

ls -l|grep "^d"|wc -l

This command displays number of directories in the given directory.

grep "bash$" /etc/passwd|wc -l

This command displays number of users of the machine whose default shell is bash.

cut -t":" -f 3 /etc/passwd|sort -n|tail -1

This command displays largest used UID number in the system. Here, cut command first extract UID's of all the users in the system from the /etc/passwd file, and the same becomes input to sort; which sorts these numbers in numerical order and sends to tail command as input which in turn displays the largest number (last one).

8.2.1 Tee command

The tee command is used to save intermediate results in a piping sequence. It accepts a set of filenames as arguments and sends its standard input to all these files while giving the same as standard output also. Thus, use of tee in piping sequence will never break the pipe.

For example if we want to save details of the directories of current working directory while knowing their number through the above piping sequence, we can use tee as follows. Here, the file xyz will have the details of directories.

ls -l|grep "^d"|tee xyz|wc -l

The following piping sequence writes the number of directories into the file pqr while displaying the same on the screen.

ls -l|grep "^d"|tee xyz|wc -l|tee pqr

8.3 SOME OTHER MEANS OF JOINING COMMANDS

UNIX supports some other means of joining command such as the following.

Command1&&Command2

Here, if first command is successfully executed then second command is executed.

For example :

ls x1&&cat x1

Here, 'ls x1' command first checks whether x1 file is available or not. If it succeeds, *i.e.,* if x1 exists then the second command executes. That is, we will see the output of the file x1.

Command1||Command2

Here, if first command is failed then second command is executed.

For example :

ls x1||echo File x1 not found

Here, 'ls x1' command first checks whether x1 file is available or not. If it fails, *i.e.,* if x1 does not exist then the second command is executed. That is, we will see that x1 file is not found

We can also enclose a set of commands which has to be executed one after another in between parenthesis. For example :

(ls; cat /etc/passwd)

We can send output of a group of file into a file or directory.

(ls; cat /etc/passwd) > outputfile

We can also enclose a set of commands which has to be executed one after another in between curly braces. For example, the following executes ls command first and then displays the content of /etc/passwd file. However, remember that there should be a semicolon after the last command.

{ ls; cat /etc/passwd; }

{ ls; cat /etc/passwd; }>outputfile

Placing a list of commands between curly braces causes the list to be executed in the current shell context. No sub-shell is created. The semicolon (or newline) following *list* is required.

In addition to the creation of a sub-shell, there is a subtle difference between these two constructs due to historical reasons. The braces are reserved words, so they must be separated from the *list* by blanks or other shell metacharacters. The parentheses are operators, and are recognized as separate tokens by the shell even if they are not separated from the *list* by whitespace.

The following piping command will grab the last 10,000 lines from /var/log/exim_mainlog, find all occurrences of domain.com (the period represents 'anything', — comment it out with a so it will be interpreted literally), then send it to your screen page by page.

tail -10000 /var/log/exim_mainlog |grep domain.com |more

The following piping sequence will show how many active connections there are to apache (httpd runs on port 80)

netstat -an |grep :80 |wc -l

The following command displays value of the environment variable $PATH.

$ echo $PATH

/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/cygdrive/c/WINDOWS/system32:/cygdrive/c/WINDOWS:/cygdrive/c/WINDOWS/System32/Wbem:/cygdrive/c/Program Files/Cepstral/bin:/cygdrive/c/ProgramFiles/java/jdk15/bin:/cygdrive/c/javaexamples

Now, we wanted to separate the directories from this variable. Thus, output of the same is passed as input to the command tr which replaces : with new line. As newline (\n) is a special character, we are escaping the same by prepending with \.

$ echo $PATH| tr : \\n

Output :

/usr/local/bin
/usr/bin
/bin
/usr/X11R6/bin
/cygdrive/c/WINDOWS/system32
/cygdrive/c/WINDOWS
/cygdrive/c/WINDOWS/System32/Wbem
/cygdrive/c/Program Files/Cepstral/bin
/cygdrive/c/Program Files/java/jdk15/bin
/cygdrive/c/javaexamples

Similarly, the following commands displays the directories of the environment variable which ends with /bin.

$ echo $PATH| tr : \\n|grep /bin

/usr/local/bin
/usr/bin
/bin
/usr/X11R6/bin
/cygdrive/c/Program Files/Cepstral/bin
/cygdrive/c/Program Files/java/jdk15/bin

Usually, the alias commands are active for the current session only. If we want either alias commands or piping commands to be available for every session then we can enter them either in .profile or .basher file.

8.4 CONCLUSIONS

In most of the OS's, it is common to create new commands by joining the existing commands. Here, we explain about creating new commands through alias, piping, tee, etc.,. Live examples are included for the purpose of experimentation.