Unix Shell Programming

Introduction to Unix/Linux

Roshan Chitrakar, PhD
roshanchi@gmail.com

Initial Commands

- login
- logout
- exit
- date
- passwd
- last
- who
- id
- su
- sudo
- less
- init
- reboot
- shutdown
- dmesg

Finding a Program: which, type, whence

- To check if a program is available is to use the ***which*** command

which date

which date less vi emacs

- Here is the output for the first command:

/bin/date

- If you use Bash, there is an alternative to the ***which*** command, ***type*** , for example:

type date

- If you use Bash, try:

type which

which type

type type

- If you use the Korn shell, you can use the ***whence*** command:

whence date

- The ***type*** and ***whence*** commands will sometimes display more detailed information than ***which***

Displaying the Date and Time: date

- Displays the current time and date. Examples: -

date

Fri Apr 15 15:15:24 NPT 2016

date -u

Fri Apr 15 09:30:56 UTC 2016

Note: there is a command time that does not display the time.

Displaying a Calendar : cal

- To display the calendar for the current month, enter:

cal

- To display a calendar for a particular year, just specify the year. For example:

cal 1952

- If you want a specific month, you must always specify both the month and the year. For instance, if you want a calendar for July 2009, you must enter:

cal 7 2009

- If, instead of displaying dates, you want to see the numbers of the days of the year, from 1 to 365 (Jan 1 = 1; Jan 2 = 2; and so on), the cal program can oblige. Just type -j after the name date. Here is an example.

cal -j 12 2009

The Reminder Service: calendar

- The calendar program offers a reminder service based on a fi le of important days and messages that you create yourself.
- All you need to do is make a file named calendar. The cal program will look for this file in the current directory. Within this file, you put lines of text in which each line contains a date, followed by a tab character, followed by a reminder note. For example:

October 21<Tab>Tammy’s birthday

November 20<Tab>Alex’s birthday

December 3<Tab>Linda’s birthday

December 21<Tab>Harley’s birthday

- When you enter ***calendar*** command, the program will check your calendar file and display all the lines that have today’s or tomorrow’s date. If “today” is a Friday, calendar will display three days worth of reminders.

Info about your system: uptime, hostname, uname

- the ***uptime*** command displays information about how long your system has been up

15:33:51 up 7:16, 3 users, load average: 0.04, 0.09, 0.13

- To find out the name of your computer, use the ***hostname*** command.
- The uname command shows you the name of your operating system. For example, you might enter:

uname and see ***Linux***

- To fi ndout more details about your operating system, use -a (all information):

uname -a

- Here is some sample output:

Linux roshan-ubuntu 4.2.0-35-generic #40-Ubuntu SMP Tue Mar 15 22:15:45 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux

Info about you: whoami

- The ***whoami*** command displays the userid you used to log in.

whoami

- A sample output may look like

roshanchi

- You can also try

who am i

- A sample output may be

roshanchi pts/1 2016-04-15 14:42 (:0)

Info about other user : users, who, w

users

- shows a list of all the userids that are currently logged in, for example:

alex casey harley root tammy

- ***who*** command shows more information than does ***users***.
- If you want to find out even more information about the userids on your system, you can use the ***w*** command. The name stands for “Who is doing what?” Here is some sample output:

w

```
 15:49:14 up  7:31,  4 users,  load average: 0.04, 0.11, 0.13
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
testuser tty1                     15:45    3:23   0.07s  0.07s -bash
roshanch :0       :0              08:20   ?xdm?  13:32   0.26s mate-session
roshanch pts/0    :0              08:20    7:28m  0.03s  0.03s /bin/bash
roshanch pts/1    :0              14:42    0.00s  0.18s  0.00s w
```

Locking your terminal : lock

- lock command tells Unix that you want to lock your terminal temporarily. The terminal will remain locked until you enter a special password. By default, lock will freeze a terminal for 15 minutes.

lock

lock -5

Asking Unix to remind you when to leave: ***leave***

- As the name implies, you can use leave to remind you when it is time to leave. You can also use it to remind you when it is time to take a break.

leave

leave 1030

leave +15

A built-in Calculator : bc

- ***bc*** is a fully programmable mathematical interpreter, which offers extended precision. Each number is stored with as many digits as necessary, and you can specify a scale of up to 100 digits to the right of the decimal point. Numeric values can be manipulated in any base from 2 to 16, and it is easy to convert from one base to another.

bc

***bc* -l** {activates built-in library)

122152 + 70867 + 122190

10+10; 20+20

quit

man bc

- {Also try ***dc***}

OPERATOR	MEANING
+	addition
-	subtraction
*	multiplication
/	division
%	modulo
^	exponentiation
sqrt(*x*)	square root

Library Functions in bc

***bc* -l** {library)

- When you use this command, bc a
  20. When you start bc, the value of
- To ask for three digits to the right o

scale=3

- If at any point you want to check what the scale factor is, simply enter:

scale

- calculations without setting a scale factor: -
- ***150/60*** yields 2
- ***35/60*** yields 0

FUNCTION	MEANING
`s(`x`)`	Sine of x; x is in radians
`c(`x`)`	Cosine of x; x is in radians
`a(`x`)`	Arctangent of x; x is in radians
`ln(`x`)`	Natural logarithm of x
`j(`n`,`x`)`	Bessel function of integer order n of x

Using variables with bc

- Within ***bc***, variable names consist of a single lowercase letter; that is, there are 26 variables, from a to z.
- To set the value of the variable x to 100, enter:

x=100

- To display the value of a variable, just enter its name:

x

- The sequence of computations below

w=160

r=(w*2)*1000

d=(w/3)*2000

r+d

- yields 426000

Using ***bc*** with different bases

- ***bc*** allows you to specify different bases for input and for output. To do so, there are two special variables that you can set: ***ibase*** is the base that will be used for input; ***obase*** is the base that will be used for output.
- For example, if you want to display answers in base 16, enter:

obase=16

- If you want to enter numbers in base 8, use:

ibase=8

- To work with bases larger than 10, bc represents the values of 10, 11, 12, 13, 14 and 15 as the uppercase letters A, B, C, D, E and F, respectively. Even if you are working in base 10, the expression A+1 will have the value 11.
- ***obase=16; obase*** , you will see 10
- if things become confusing, you can always reset the bases by entering:

obase=A; ibase=A

The stack-based calculator: dc

- Originally, the bc program was based on a program called dc (desk calculator). dc is among the oldest Unix programs, even older than the C programming language. In fact, the original version of dc was written in 1970 using the programming language B, the ancestor of C.
- bc worked by converting its input to Reverse Polish Notation (RPN) and then calling upon dc to do the actual work. In other words, bc was a “front-end” to dc. This allowed people to use whichever system they preferred: postfi x notation with dc or infi x notation with bc. Years later, as part of the GNU Project, bc was completely rewritten as an independent program.
- Example:
- ***34 25 + 15 ****{displays nothing}
- ***p***
- displays 885.
- ***f*** shows the entire stack.
- For more, ***man dc***

Getting Help: man, whatis, apropos, info

- man is the help built into Unix
- man -f is equivalent to whatis
- man -k is equivalent to apropos
- info is a separate help from man

- RTFM !

Working with Files

- Listing Files: The ls Command
  - To see what files you have stored in your directory, you can type the ls command:

$ ls

$ ls –l

$ ls -l programs

- The -l option (the letter l) provides a more detailed description of the files in a directory.

total 2

drwxr-xr-x 5 steve DP3725 80 Jun 25 13:27 documents

drwxr-xr-x 2 steve DP3725 96 Jun 25 13:31 programs

- Displaying the Contents of a File: The cat Command
  - You can examine the contents of a file by using the cat command. The argument to cat is the name of the file whose contents you want to examine.

$ cat names

Files (contd.)

- Counting the Number of Words in a File: The wc Command
  - With the wc command, you can get a count of the total number of lines, words, and characters of information contained in a file. The name of the file is needed as the argument to this command:

$ wc names

5 5 27 names

  - The first number represents the number of lines contained in the file (5), the second the number of words contained in the file (in this case also 5), and the third the number of characters contained in the file (27).
  - To count just the number of lines contained in a file, the option -l (that's the letter l) is given to the wc command:

$ wc -l names

5 names

  - To count just the number of characters in a file, the -c option is specified:$ wc -c names

27 names

  - Finally, the -w option can be used to count the number of words contained in the file:

$ wc -w names

5 names

Files (contd.)

- Making a Copy of a File: The cp Command
  - To make a copy of a file, the cp command is used. The first argument to the command is the name of the file to be copied (known as the source file), and the second argument is the name of the file to place the copy into (known as the destination file). You can make a copy of the file names and call it saved_names as follows:

$ cp names saved_names

- Renaming a File: The mv Command
  - A file can be renamed with the mv command. The arguments to the mv command follow the same format as the cp command. The first argument is the name of the file to be renamed, and the second argument is the new name. So, to change the name of the file saved_names to hold_it, for example, the following command would do the trick:

$ mv saved_names hold_it

- Removing a File: The rm Command
  - To remove a file from the system, you use the rm command. The argument to rm is simply the name of the file to be removed:

$ rm hold_it

- You can remove more than one file at a time with the rm command by simply specifying all such files on the command line. For example, the following would remove the three files wb, collect, and mon:

$ rm wb collect mon

Working with Directories

- Displaying Your Working Directory: The pwd Command
  - The pwd command is used to help you "get your bearings" by telling you the name of your current working directory.

$ pwd

- Changing Directories: The cd Command
  - You can change your current working directory by using the cd command. This command takes as its argument the name of the directory you want to change to.
  - $ cd /users/steve/documents
  - cd ..
  - cd ../..
  - cd

Directories (contd.)

- Creating a Directory: The mkdir Command
  - To create a directory, the mkdir command must be used. The argument to this command is simply the name of the directory you want to make. For example, assume that you are still working with the directory structure depicted in Figure 2.7 and that you want to create a new directory called misc on the same level as the directories documents and programs. If you were currently in your home directory, typing the command mkdir misc would achieve the desired effect:

$ mkdir misc

- Linking Files: The ln Command
  - In simplest terms, the ln command provides an easy way for you to give more than one name to a file. The general form of the command is

ln from to

- This links the file from to the file to.

Directories (contd.)

- Removing a Directory: The rmdir Command
  - You can remove a directory with the rmdir command. The stipulation involved in removing a directory is that no files be contained in the directory. If there are files in the directory when rmdir is executed, you will not be allowed to remove the directory. To remove the directory misc that you created earlier, the following could be used:

$ rmdir /users/steve/misc

rm -r dir

  - where dir is the name of the directory that you want to remove. rm removes the indicated directory and all files (including directories) in it.

Filename Substitution

- The Asterisk (*)

cat *

echo *

echo * : *

cat chapt*

echo *t1

echo *t*

echo *x

- Matching Single Characters
- echo ?
- echo a?
- echo ??
- echo ??*

Matching Single Characters

- Another way to match a single character is to give a list of the characters to use in the match inside square brackets [ ].
- For example, [abc] matches one letter a, b, or c.
- The specification [0-9] matches the characters 0 through 9.
- The first character must be alphabetically less than the last character, so that [z-f] is not a valid range specification.
- [a-np-z]* matches all files that start with the letters a through n or p through z (or any lowercase letter but o).
- If the first character following the [is a !, the sense of the match is inverted. That is, any character is matched except those enclosed in the brackets.
- [!a-z] matches any character except a lowercase letter, and
- *[!o] matches any file that doesn't end with the lowercase letter o.
- ls [a-z]*[!0-9] lists files that begin with a lowercase letter and don't end with a digit.

Standard Input and Standard Output

- Most Unix system commands take input from your terminal and send the resulting output back to your terminal.
- A command normally reads its input from a place called standard input, which happens to be your terminal by default.
- Similarly, a command normally writes its output to standard output, which is also your terminal by default.

Standard I/O Examples

```
$ sort
Tony
Barbara
Harry
Dick
Ctrl+d
Barbara
Dick
HarryTony
$
```

```
$ wc -l
This is text that
is typed on the
standard input device.
Ctrl+d
3
$
```

Output Redirection

- The output from a command normally intended for standard output can be easily diverted to a file instead.
- This capability is known as output redirection.
- If the notation > file is appended to any command that normally writes its output to standard output, the output of that command will be written to file instead of your terminal:

```
$ who > users
$
```

O/P Redirection Examples

```
$ cat users
oko tty01 Sep 12 07:30
ai tty15 Sep 12 13:32
ruth tty21 Sep 12 10:10
pat tty24 Sep 12 13:07
steve tty25 Sep 12 13:03
$
```

```
echo line 1 > users
$ cat users
line 1
$ echo line 2 >> users
$ cat users
line 1
line 2
$
```

O/P Redirection More Examples

- By using the redirection append characters >>, you can use cat to append the contents of one file onto the end of another:

$ cat file1

This is in file1.

$ cat file2

This is in file2.

$ cat file1 >> file2

$ cat file2

This is in file2.

This is in file1.

$

$ cat file1

This is in file1.

$ cat file2

This is in file2.

$ cat file1 file2

This is in file1.

This is in file2.

$ cat file1 file2 > file3

$ cat file3

This is in file1.

This is in file2.

$

Input Redirection

- To redirect the input of a command, you type the < character followed by the name of the file that the input is to be read from.

```
$ wc -l users
2 users
$
$ wc -l < users
2
$
```

Pipes

- The Unix system allows you to effectively connect two commands together. This connection is known as a pipe, and it enables you to take the output from one command and feed it directly into the input of another command.
- A pipe is effected by the character |, which is placed between the two commands.

```
$ who | wc -l
5
$
$ ls | wc -l
10
$
```

Filters

- The term filter is often used in Unix terminology to refer to any program that can take input from standard input, perform some operation on that input, and write the results to standard output.
- More succinctly, a filter is any program that can be used between two other programs in a pipeline.
- So in the previous pipeline, wc is considered a filter. ls is not because it does not read its input from standard input.

Standard Error

- There is place known as standard error, where most Unix commands write their error messages. And standard error is associated with your terminal by default.

$ ls n*

n* not found

$

- Here the "not found" message is actually being written to standard error and not standard output by the ls command.

$ ls n* > foo

n* not found

$

- So, you still get the message printed out at the terminal, even though you redirected standard output to the file foo.

- You can also redirect standard error to a file by using the notation

command 2> file

- No space is permitted between the 2 and the >. Any error messages normally intended for standard error will be diverted into the specified file, similar to the way standard output gets redirected.

$ ls n* 2> errors

$ cat errors

n* not found

$

Typing More Than One Command on a Line

- You can type more than one command on a line provided that you separate each command with a semicolon. For example, you can find out the current time and also your current working directory by typing in the date and pwd commands on the same line:

```
$ date; pwd
Sat Jul 20 14:43:25 EDT 2002
/users/pat/bin
$
```

Sending a Command to the Background

- If you type in a command followed by the ampersand character &, that command will be sent to the background for execution. This means that the command will no longer tie up your terminal, and you can then proceed with other work.

$ sort data > out &

- sends the sort to the background

[1] 1258 Process id

- Your terminal is immediately available to do other work

$ date

Sat Jul 20 14:45:09 EDT 2002

$

vi Text Editor

Introduction to vi

- The two principal Unix text editors are vi and Emacs (there are kedit, gedit, Pico and Nano also).
- vi is part of the two principal Unix specifications - POSIX and the Single Unix Specification (the standards that must be met to call something “Unix”). Thus, vi must be available on every Unix system.
- The vi editor was created by Bill Joy , at U.C. Berkeley in the late 1970s.

vim: An Alternative to vi

- In the late 1980s, an open source vi-clone named STvi (often written as STevie) was created for non-Unix systems.
- In 1988, a Dutch programmer named Bram Moolenaar took STvi and used it to create a new program he called ***vim***, the name meaning “vi imitation”. For several years, Moolenaar worked on vim, fixing bugs and adding new features until, in 1992, he released the first Unix version of the program.
- By now, there were so many enhancements, that Moolenaar changed the meaning of the name. Although the program was still called vim, Moolenaar declared that, from now on, the name should stand for “vi improved”.

Starting vi

- To start vi, the basic syntax is:

vi [-rR] [file...]

- where file is the name of a file you want to edit.

vi message

vi

Starting vim

- On some systems vi has been replaced by Vim. Use vim as if it were vi.
  - Later, once you are comfortable with vi, you can teach yourself how to use the extended features offered by vim.
- How to start Vim so it acts like vi?
  - Starting Vim is just like starting vi. The basic syntax is:

vim -C [-rR] [file...]

- The -r and -R options work the same way as with vi. When you start vim with -C, it changes the settings so as to make vim act as much like vi as possible. (C stands for Compatibility mode).
- For Example: -

vim -C essay

vim -C

Command Mode and Input Mode

- vi to work in two different modes.
  - In COMMAND mode, whichever keys you type are interpreted as commands. For example, in command mode, the single letter x is the command to delete a character; the combination dd is the command to delete an entire line. There are many such 1- and 2-character commands.
  - The second mode is INPUT mode. In this mode, everything you type is inserted directly into the editing buffer. For example, in input mode, if you type “Hello Harley”, these 12 characters are inserted into the editing buffer. If you press the x key, an “x” is inserted; if you press dd, the letters “dd” are inserted.

Command and Input Mode (contd.)

- When vi starts, you are in command mode. As vi starts, it does three things: -
  - First, it copies the contents of schedule to the editing buffer.
  - Next, it positions the cursor at the beginning of the first line of the buffer.
  - Finally, it puts you in command mode.
- To change to input mode, you use one of several commands (e.g. a colon : followed by i for insert texts). Once you are in input mode, changing back to command mode is easy: just press the <Esc> (Escape) key. If you are already in command mode and you press <Esc>, vi will beep.
- You begin by using the appropriate commands to move the cursor to the place where you want to add the new data. You then type a command to change to input mode and start typing. At this point, everything you type is inserted directly into the editing buffer. When you are finished typing, you press <Esc> to change back to command mode. You then save the contents of the editing buffer back to the original file and quit the program.

Starting vi as a READ-ONLY editor: view , vi -R

- First, you can start the program with the -R (read-only) option. This tells vi that you do not want to save data back into the original file.
- Secondly, you can start the program by using the view command.

vi -R importantfile

view importantfile

- (Note; the small letter r option is to recover file from system failure. i.e. vi -r filename)

Stopping vi

- To save your work and then quit, the command is ZZ. Hold down the <Shift> key and press <Z> twice. You do not need to press <Return>:

ZZ

- To quit without saving your work, the command is :q!. After you type this command, you do need to press <Return>:

:q!<Return>

Writing Data to a File

- When you stop vi using the ZZ command, it automatically saves your data.
- It also allows you to save data to a different file. The commands are:

:w----------write data to original file

:w file ----------write data to a new file

:w! file----------overwrite an existing file

:w>> file----------append data to specified file

A Strategy for Learning vi commands

- The vi editor has a large variety of commands. For convenience, we can group them as follows:
  - Commands to move the cursor
  - Commands to enter input mode
  - Commands to make changes
- Once you learn the basic vi commands, you will find that there are a variety of ways to implement any particular strategy. How you choose to do it depends on the specific situation and your level of skill. No one needs to know 12 ways to enter input mode or 40 ways to move the cursor.
- So, RTFM !