COURSE TECHNOLOGY
CENGAGE Learning

Linux Operations and Administration

Ahmad AlRjoub
ahmadrj@ksu.edu.sa

Chapter Five

Creating Shell Scripts and Displaying File Contents

Objectives

- Identify and change Linux file permissions
- Create and run shell scripts
- Display the contents of a text file

An Overview of Linux File Permissions

- Viewing file permissions
  - `less -l` command
- User, group, and other permissions:
  - User—the file owner
  - Group—a group of users; users are divided into groups to facilitate administrative tasks
  - Other—everyone else on the Linux system

An Overview of Linux File Permissions (cont'd.)

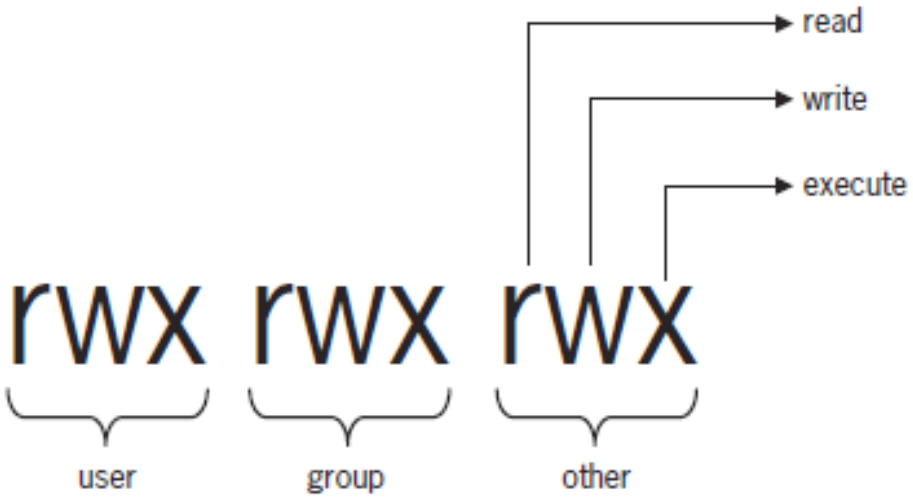

Figure 5-1 File permissions
©Engage Learning 2013

An Overview of Linux File Permissions (cont'd.)

- Read, write, and execute
  - Functions of these permissions differ, depending on whether they're applied to files or directories
- Table 5-1
  - Defines permissions for files and directories

An Overview of Linux File Permissions (cont'd.)

Permission	Permission for files	Permission for directories
`r` (read)	Gives users permission to open a file and view its contents	Allows users to list a directory's contents with commands such as `ls`
`w` (write)	Gives users permission to open a file and edit its contents	Allows users to add or remove files and subdirectories
`x` (execute)	Allows users to run the file (as long as it's a program or script)	Allows users to switch to the directory with the `cd` command; to read a directory's contents and add or remove files and subdirectories, you must have execute permission

Table 5-1 Linux file and directory permissions

An Overview of Linux File Permissions (cont'd.)

- Example:
  - `-rw-r--r-- 1 martin users 0 2012-04-11 09:20 file1`
  - User category of permissions is set to `rw-`
    - Hyphen (-) represents no permission
    - Read, write, not execute
  - Group category is set to `r--`
    - Group has read permission but not write and execute permissions
  - Other category is set to `r--`
    - Every user on the system has read permission but not write or execute

Permission Commands

- `chmod` (change mode) command
  - Change permissions on files and directories
  - Syntax: `chmod permissions file/directory`
  - Permissions argument
    - Information used to change permissions
  - File/directory argument
    - Specifies the file or directory you want to change
- Notations
  - Symbolic notation
  - Numeric notation

Permission Commands (cont'd.)

- Symbolic notation
  - Uses criteria such as categories and operators to change file permissions
  - Described in Table 5-2
  - Example: `chmod o-wx file4`
- Numeric notation
  - Uses numbers from 0 to 7 to represent file permissions
  - Shown in Table 5-3
  - Example: `chmod 774 file1`

Permission Commands (cont'd.)

Category	Operator	Permission
u (user)	+ (add to existing permissions)	r (read)
g (group)	- (remove from existing permissions)	w (write)
o (other)	= (assign absolute permissions)	x (execute)
a (all)	One of the preceding operators	One or more of the preceding permissions

Table 5-2 Symbolic notation

Permission Commands (cont'd.)

Permission	Numeric value
---	0
--x	1
-w-	2
-wx	3
r--	4
r-x	5
rw-	6
rwx	7

Table 5-3 Numeric notation

Permission Commands (cont'd.)

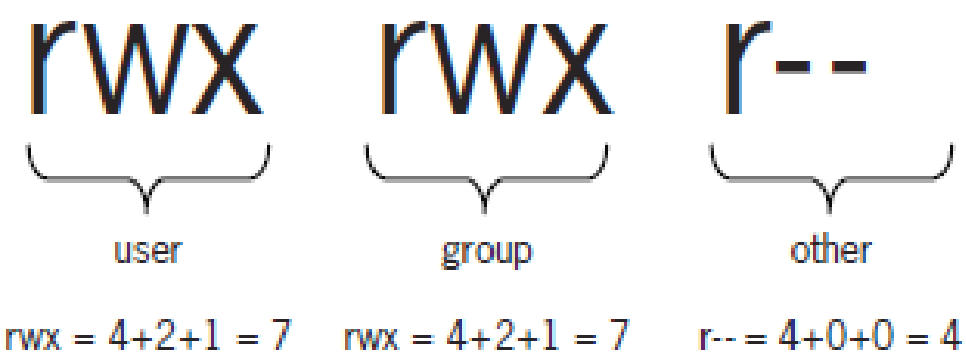

Figure 5-2 File permissions in numeric notation
©Engage Learning 2013

Creating Shell Scripts

- Shell script
  - Contains a sequence of commands to execute line by line
  - Used in troubleshooting
  - Some scripts run when the Linux system starts
    - Need to know how to manage these scripts if problems occur during the boot process
- Programming language
  - Set of rules for instructing a computer how to perform specific tasks

Creating Shell Scripts (cont'd.)

- Machine code
  - Consists of binary 1s and 0s and
  - Language a computer's CPU understands
- Scripts can be considered compiled programs or interpreted programs
  - Compiled program: all the source code is converted to machine code and stored in a binary file before the user runs the script
  - Interpreted program: source code is converted to machine code, line by line, as the user runs the script

Creating Shell Scripts (cont'd.)

- In Chapter 5: BASH shell interpreter
- Create a shell script:
  - Create a file
  - Assign execute permission for it
- Run a shell script:
  - Enter the absolute or relative path to where it's stored
  - Example: run a script called `scr1` that's stored in your current directory
    - `./scr1`

Creating Shell Scripts (cont'd.)

- Activity 5-1: Creating a Shell Script
  - Create and run a shell script
  - `#!/bin/bash` command
    - Specifies running the script in the BASH shell
  - Comment
    - Add documentation information for users and anyone else who might need to modify the script

Variables

- Environment variable
  - Placeholder for data that can change
  - Gets its value automatically from the OS startup or the shell being used
  - Each user has his or her own environment variables
- Table 5-4
  - Describes some common environment variables

Variables (cont'd.)

Variable name	Stored value
HOME	Home directory
USER	Login name
PATH	Gives the search path, which is the list of directories (separated by : symbols) the shell uses when searching for executable commands
HOST	Computer name

Table 5-4 Environment variables

Variables (cont'd.)

- `env` command
  - Display a list of all environment variables and their stored values
- `echo` command with `$` symbol before the variable name
  - Display value of particular environment variable
  - `echo $HOME` returns the value of the `HOME` variable

Variables (cont'd.)

- Shell variable
  - Similar to an environment variable
  - Value is usually assigned in a shell script
  - Related to a particular script
    - Not necessarily the global environment

Direct Assignment

- Direct assignment method
  - Specify the variable's value in the command
  - For example, `COLOR=blue`
- Activity 5-2: Using the Direct Assignment Method
  - Use the direct assignment method to store a value in a variable

Direct Assignment (cont'd.)

Option	Example	Description
`-name`	`find / -name hosts`	Starts in the root directory (/) and searches for files named hosts
`-type d`	`find . -type d`	Starts in the current directory (indicated by the .) and searches for all subdirectories
`-type f`	`find /home -type f`	Starts in the `/home` directory and searches for all files
`-type l`	`find /etc -type l`	Starts in the `/etc` directory and searches for all symbolic links
`-group`	`find . -group users`	Starts in the current directory and searches for all files belonging to the users group
`-user`	`find /home -user jasmine`	Starts in the `/home` directory and searches for all files belonging to the user jasmine
`-inum`	`find / -inum 3911`	Starts in the root directory (/) and searches for all files with the inode number 3911
`-mmin n`	`find / -mmin 10`	Starts in the root directory (/) and searches for all files that have been modified in the past 10 minutes

Table 5-5 Options for the find command

The Prompt Method

- Prompt method
  - User is asked to enter a value for the variable
- Activity 5-3: Using the Prompt Method
  - Create a script with the prompt method for storing a value in a variable

Positional Parameters

- Positional parameter method
  - Uses the order of arguments in a command to assign values to variables on the command line
  - Variables from `$0` to `$9` are available
    - Values are defined by what the user enters
- Example:
  - `./scr1 /home`
  - `$1 to be /home`

```
#!/bin/bash
clear
echo "Searching for $1"
find $1
```

Positional Parameters (cont'd.)

- Table 5-6
  - Describes positional parameters
- Activity 5-4: Using Positional Parameters
  - Create a script that uses positional parameters to assign values to variables

Positional Parameters (cont'd.)

Positional parameter	Description	Example
`$0`	Represents the name of the script	`./scr4` (`./scr4` is position 0)
`$1` to `$9`	`$1` represents the first argument, `$2` represents the second argument, and so on	`./scr4 /home` (`./scr4` is position 0 and `/home` is position 1) `./scr4 /home scr1` (`./scr4` is position 0, `/home` is position 1, and `scr1` is position 2)
`$*`	Represents all the positional parameters except 0	`/home scr1` (just `/home` and `scr1`)
`$#`	Represents the number of arguments that have a value	`./scr4 /home scr1` `echo $#` (`$*` represents positions 1 and 2, which are `/home` and `scr1`)

Table 5-6 Positional parameters

Exit Status Codes

- Exit status code is sent to the shell
  - When you quit a program or a command
- Successful commands usually return the code 0
- Failures return a value greater than 0
- Code isn't actually displayed onscreen
  - Reference it with `$?`

```
echo $?
0
cd baddir
bash: cd: baddir: No such file or
directory
echo $?
1
```

Conditions

- Tell interpreter to skip commands based on a condition
- `if` statement
  - Carry out certain commands based on testing a condition

Conditions (cont'd.)

- Common condition statements used in scripts:
  - `if` statement—starts the condition being tested
  - `then` statement—starts the portion of code specifying what to do if the condition evaluates to `true`
  - `else` statement—starts the portion of code specifying what to do if the condition evaluates to `false`
  - `fi` statement—indicates the end of the condition being tested

Conditions (cont'd.)

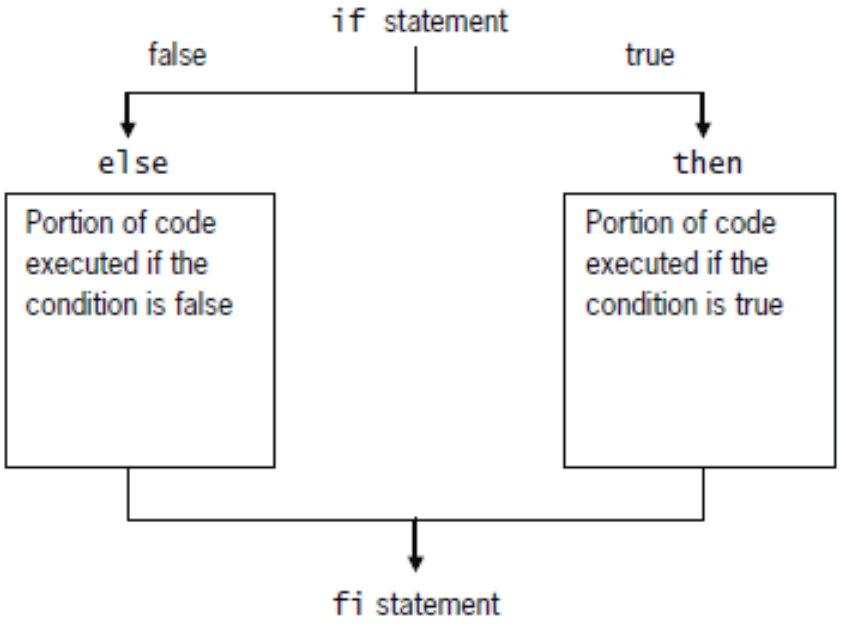

Figure 5-3 A flowchart of the if statement

©Engage Learning 2013

Conditions (cont'd.)

- Activity 5-5: Using Condition Statements
  - Create a script with `if`, `then`, and `else` statements
- Table 5-7
  - Lists file attribute operators available in the BASH shell

Conditions (cont'd.)

File attribute operator	Description
`-a`	Checks whether the file exists
`-d`	Checks whether the file is a directory
`-f`	Checks whether the file is a regular file
`-r`	Checks whether the user has read permission for the file
`-s`	Checks whether the file contains data
`-w`	Checks whether the user has write permission for the file
`-x`	Checks whether the user has execute permission for the file
`-O`	Checks whether the user is the owner of the file
`-G`	Checks whether the user belongs to the group owner of the file
`file1 -nt file2`	Checks whether `file1` is newer than `file2`
`file1 -ot file2`	Checks whether `file1` is older than `file2`

Table 5-7 File attribute operators in the BASH shell

Menu Scripts

- Menu scripts
  - Allows users to choose from a list of options
- Activity 5-6: Creating a Menu Script
  - Create a menu script with `if` and `then` statements
- `elif` statement
  - Combines the `else` and `if` statements
  - Create multiple conditions without closing each condition

The `case` Statement

- `case` statement
  - Uses one variable to specify multiple values and matches a portion of the script to each value
- Syntax:

```
case $VARIABLE in
value1) code for specified value1
;;
value2) code for specified value2
;;
valuen) code for specified valuen
;;
*)code for value not matching any
specified choices ;;
esac
```

The `case` Statement (cont'd.)

- Double semicolon (`;;`)
  - Marks the end of each code portion matching a specific value
- `*)` character
  - Runs if the value the user enters doesn't match any of the choices specified in the case statement

The `case` Statement (cont'd.)

- Activity 5-7: Using `case` Statements in a Menu Script
  - Create a menu script with `case` statements

Looping

- Perform a set of commands repeatedly
- Looping statements:
  - `while` statement—interpreter continues executing the code in the while loop portion of the script as long as the condition is true
  - `until` statement—interpreter continues executing the code in the until loop portion of the script as long as the condition is false

Looping (cont’d.)

- `for` statement—specifies the number of times to execute the portion of code
- `do` statement—indicates the beginning of the code to be repeated
- `done` statement—indicates the end of the code to be repeated

The `while` Loop

- `while` loop
  - Repeats commands between `do` and `done` statements
  - As long as the tested condition is `true`
- When the command after the `while` statement returns an exit status code greater than `0`
  - `while` statement fails
  - Program executes commands after `done` statement
- Activity 5-8: Creating a `while` Loop
  - Create a `while` loop in a script

The `while` Loop (cont'd.)

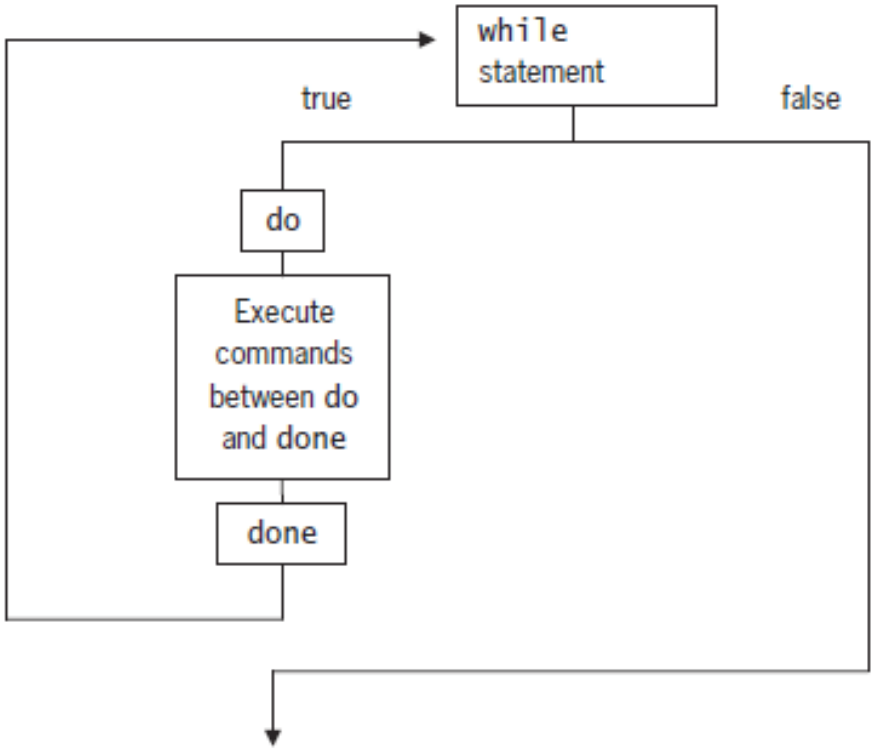

Figure 5-4 A while loop

©Engage Learning 2013

The `until` Loop

- `until` loop
  - Repeats commands between `do` and `done`
  - As long as the tested condition is `false`
- When the command following the `until` statement has the exit status code 0
  - `until` loop fails
  - Program executes commands after `done` statement
- Activity 5-9: Creating an `until` Loop in a Menu Script
  - Create a menu script that continues running until the user decides to exit

The `until` Loop (cont'd.)

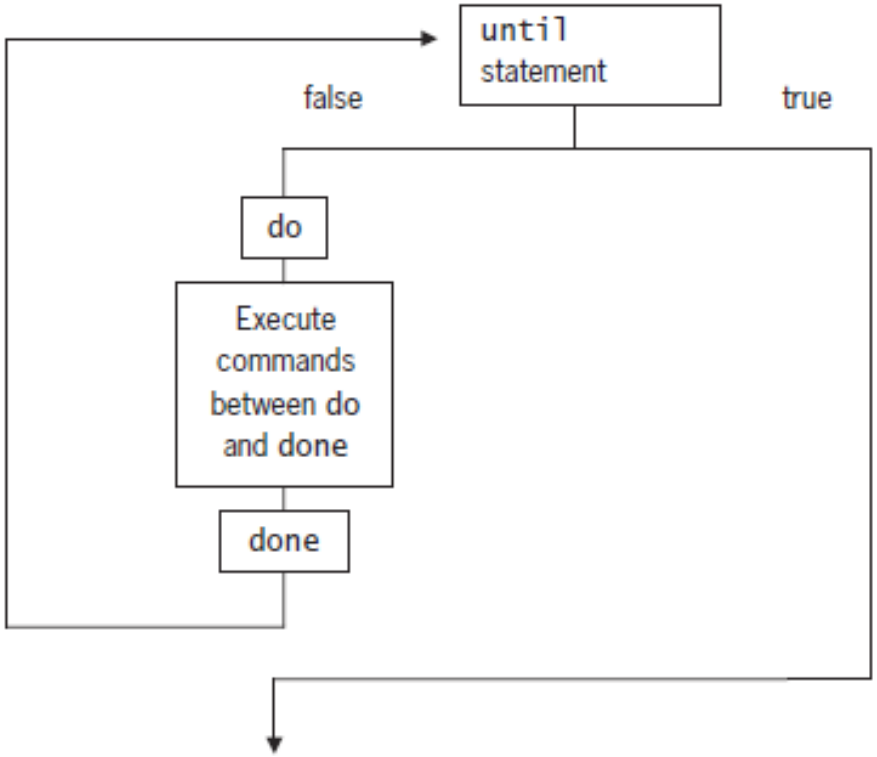

Figure 5-5 An until loop
©Engage Learning 2013

The `for` Loop

- `for` loop
  - Repeats the commands between `do` and `done` a specified number of times
  - Each time the script carries out the commands in the loop, a new value is given to a variable
  - Assign this value in the command with positional parameters
- Activity 5-10: Creating a `for` Loop
  - Create a script that repeats the commands between do and done a specified number of times

Displaying the Contents of a Text File

- List a file’s contents without actually opening the file in a text editor

The `cat` and `tic` Commands

- `cat` (concatenation) command
  - Displays an entire file's contents at once
  - Typically used to display the contents of a small text file
  - Can be used to display the contents of multiple files at once
- `-n` option
  - Display line numbers in a text file:
  - `cat -n scr1`

The `cat` and `tic` Commands (cont'd.)

- `tic` command
  - Display a text file's contents in reverse order
  - Main purpose to display log files

The `head` and `tail` Commands

- `head` command
  - Displays the first 10 lines of a text file
  - `head scr8`
  - Can display more than 10 lines: `head -15 scr8`
- `tail` command
  - Displays the last 10 lines of a text file
  - `tail scr8`
  - Can display more than 10 lines: `tail - 15 scr8`
  - `+` operator
    - Start displaying text at a specified line number all the way to the end of the file

The `more` and `less` Commands

- `more` command
  - Displays a file's contents one screen at a time
- Table 5-8
  - Lists options you can use with the `more` command
- `less` command
  - Displays a file's contents one screen at a time
  - Allows you to navigate the file by using arrow keys or the mouse wheel

The `more` and `less` Commands (cont'd.)

```
#!/bin/bash
until [ $CHOICE -eq 4 ]
do
clear
echo Please select a menu item
echo echo CHOICE is $CHOICE
echo
echo "1)Display your current working directory"
echo "2)Display your home directory"
echo "3)List the contents of your current working directory"
echo "4)Exit the program"
echo
read CHOICE
case $CHOICE in
--More--(67%)
```

Figure 5-6 Output of the more command

©Engage Learning 2013

The `more` and `less` Commands (cont'd.)

Command	Description
Spacebar	Displays the next screen
#+spacebar	Displays the next # lines
Enter	Displays the next line
q	Exits the `more` command
=	Displays the current line number
h	Displays help

Table 5-8 Options for the more command

The `more` and `less` Commands (cont'd.)

```
#!/bin/bash
until [ $CHOICE -eq 4 ]
do
clear
echo Please select a menu item
echo echo CHOICE is $CHOICE
echo
echo "1)Display your current working directory"
echo "2)Display your home directory"
echo "3)List the contents of your current working directory"
echo "4)Exit the program"
echo
read CHOICE
case $CHOICE in
1) pwd;;
2) echo $HOME;;
scr8 lines 1-16/23 72%
```

Figure 5-7 Output of the less command

©Engage Learning 2013

Summary

- Linux file permissions
  - Assigned in the user, group, and other categories
  - Changed by using the `chmod` command
- Shell scripts
  - Values are assigned to variables by direct assignment, positional parameters, or the prompt method
  - Condition statements
    - Used to run specified portions of a script matching the condition

Summary (cont'd.)

  - Loops used in shell scripts are `while`, `until`, and `for`
- Listing file contents
  - `cat` and `tic`
    - Print entire file contents
  - `head` and `tail`
    - View beginning or end of file
  - `more` and `less`
    - View file screen by screen