Advanced SUSE Linux Enterprise Server Administration (Course 3038)

Chapter 6

Create Shell Scripts

Objectives

- Use Basic Script Elements
- Use Variable Substitution Operators
- Use Control Structures
- Use Advanced Scripting Techniques
- Learn About Useful Commands in Shell Scripts

Use Basic Script Elements

- Objectives
  - Flow Charts for Scripts
  - The Basic Rules of Shell Scripting
  - How to Develop Scripts That Read User Input
  - How to Perform Basic Script Operations with Variables
  - How to Use Command Substitution
  - How to Use Arithmetic Operations

Flow Charts for Scripts

- Programming elements of a script
  - Often visualized by using program flow charts
- Flow charts benefits
  - Force author to lay down the steps the script should perform
  - Provide a clear symbolic outline of the algorithm

Flow Charts for Scripts (continued)

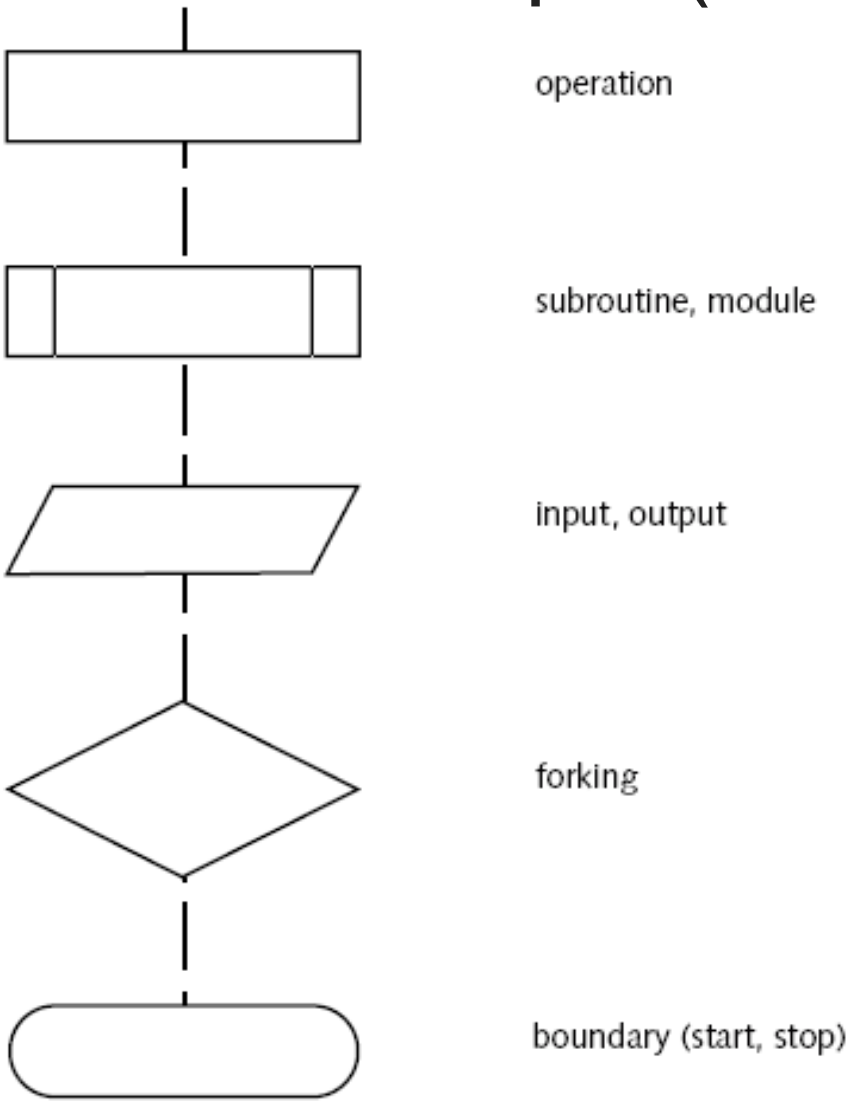


Figure 6-1

The Basic Rules of Shell Scripting

- Shell script
  - ASCII text file
  - Contains commands to be executed in sequence
  - Permissions for script file must be set to “r” and “x”
    - chmod +x script.sh
- Run shell script with sh script.sh
- Create a /bin directory for scripts under each user’s home directory
  - Add this directory to the user’s search path
    - export PATH=$PATH:~/bin

The Basic Rules of Shell Scripting (continued)

- Add an .sh extension to the script filename
- Make sure script filename is not identical to existing commands
- Elements of a script
  - Start
  - Commands
  - Stop

The Basic Rules of Shell Scripting (continued)

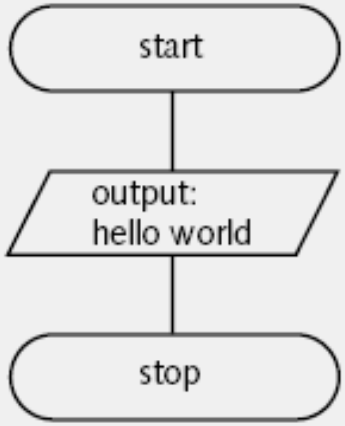
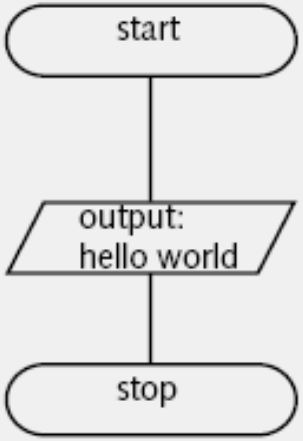


Figure 6-2




```
#!/bin/bash

# This script …

echo "Hello world!"

exit 0
```

Figure 6-3

Exercise 6-1 Produce Output from a Script

- In this exercise, you will produce output from a script using the echo command

How to Develop Scripts That Read User Input

- Command read
  - Used to create scripts that read user input
  - Takes a variable as an argument
    - And stores the read input in the variable
  - Variable can then be used to process the user input

How to Develop Scripts That Read User Input (continued)

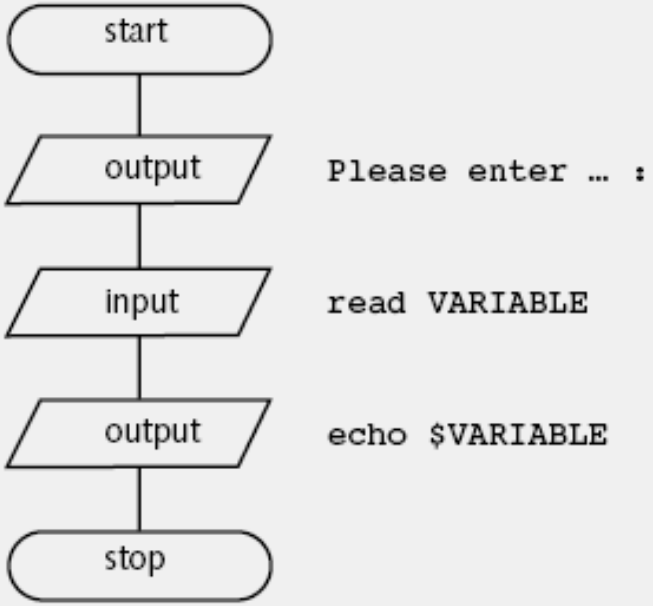


Figure 6-4

Exercise 6-2 Read User Input

- In this exercise, you will use the read command in a shell script to accept user input

How to Perform Basic Script Operations with Variables

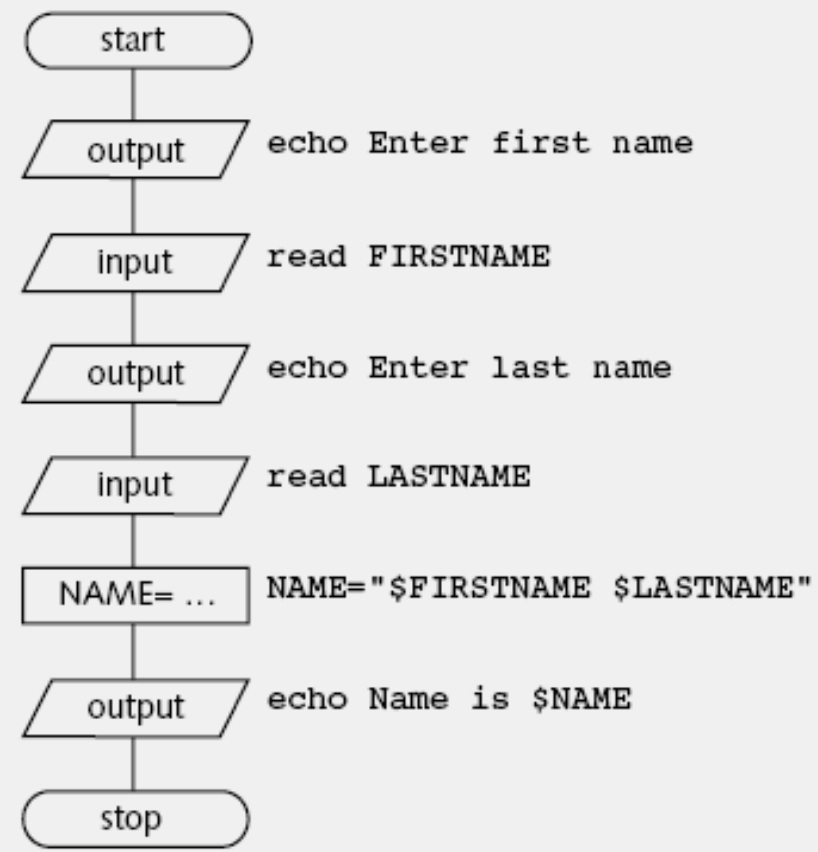


Figure 6-5

Exercise 6-3 Simple Operations with Variables

- In this exercise, you will practice performing simple operations with variables

How to Use Command Substitution

- Command substitution
  - Output of a command is used in a shell command line or a shell script
- Example 1: printing output

```
echo "Today is `date +%m/%d/%Y`"
```

- Example 2: assigning output to a variable

```
TODAY=`date +%m/%d/%Y`
echo "Today is $TODAY"
```

Exercise 6-4 Use Command Substitution

- In this exercise, you will practice using command substitution

How to Use Arithmetic Operations

- Bourne shell is limited in this regard
  - Can perform operations by relying on external commands (such as expr)
- Bash shell
  - Comes with built-in support for arithmetic operations
  - Limited in the following ways
    - Only supports operations with whole numbers
    - All values are signed 64-bit values
  - Needs to use external commands, such as bc, for floating-point calculations

How to Use Arithmetic Operations (continued)

- Use the external command expr
  - `A=\`expr $B + 10\``
- Use the Bash built-in command let
  - `let A="$B + 10"`
- Use arithmetic expressions inside parentheses or brackets
  - `A=$((B + 10)) or A=$[B + 10]`
- Use the built-in command declare
  - `declare -i A`
  - `declare -i B`
  - `A=B+10`

Exercise 6-5 Use Arithmetic Operations

- In this exercise, you will practice using arithmetic operations

Use Variable Substitution Operators

- Variable substitution operators
  - Used to assign different values to variables
    - Without having to rely on external commands

Use Variable Substitution Operators (continued)

Table 6-1

Substitution Operator	Description
${variable-value}	Returns value if the variable does not exist.
${variable=value}	Assigns value to the variable and returns value if the variable does not exist.
${variable+value}	Returns value if the variable exists.
${#variable}	Returns the number of characters in the value of variable.
${variable#pattern}	Deletes the shortest part matched by pattern from the beginning of the variable's value and returns the rest.
${variable##pattern}	Deletes the longest part matched by pattern from the beginning of the variable's value and returns the rest.
${variable%pattern}	Deletes the shortest part matched by pattern from the end of the variable's value and returns the rest.
${variable%%pattern}	Deletes the longest part matched by pattern from the end of the variable's value and returns the rest.

Exercise 6-6 Use Variable Substitution

- In this exercise, you will practice using variable substitution

Use Control Structures

- Objectives
  - Create Basic Branches with the if Command
  - Build Multiple Branches with a case Statement
  - Create Loops Using the while and until Commands
  - Process Lists with the for Loop
  - Interrupt Loop Processing

Create Basic Branches with the if Command

- Basic usage of the if command

```
if condition
then
  commands
fi
```

- Optional else statement

```
if condition
then
  command1
else
  command2
fi
```

Create Basic Branches with the if Command (continued)

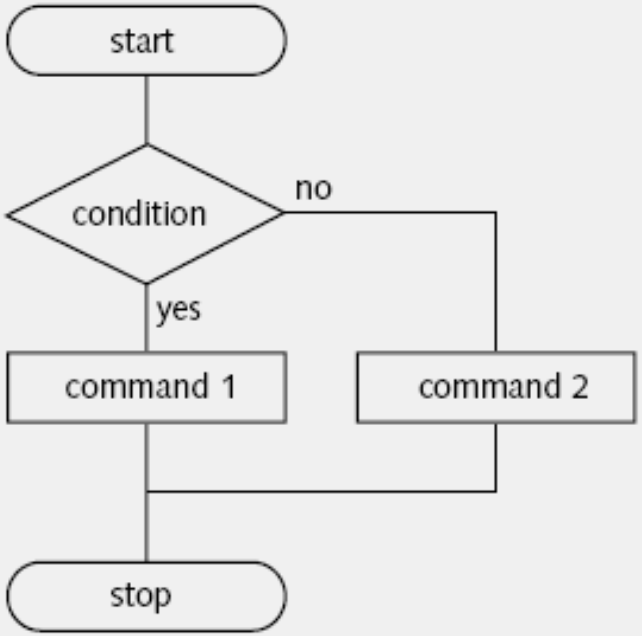


Figure 6-7

Create Basic Branches with the if Command (continued)

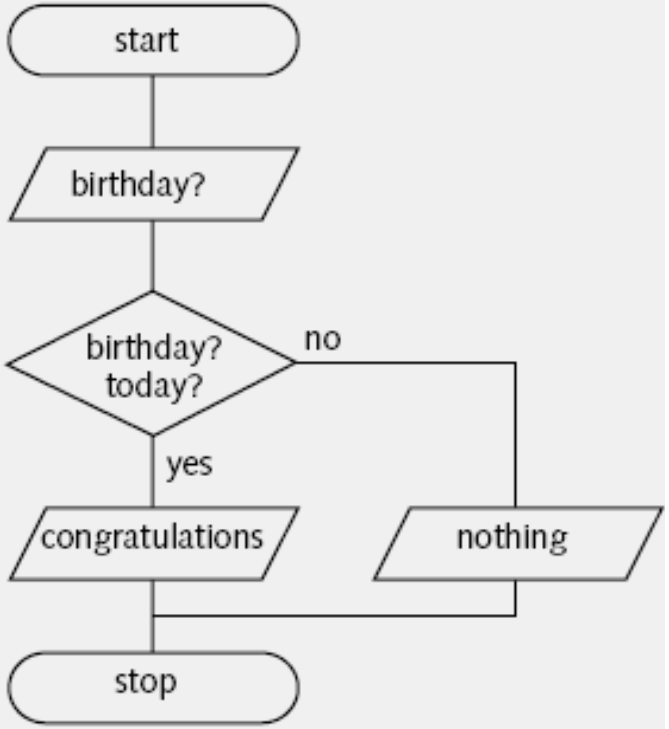


Figure 6-8

Create Basic Branches with the if Command (continued)

- Example: birthday script

```
echo "Please enter your date of birth (YYYY-MM-DD, for
instance 1978-06-21): "
read BIRTHDAY
BIRTHDAY=${BIRTHDAY#*-}
TODAY= date + %m-%d
if test "$BIRTHDAY" = "$TODAY"
then
  echo "Tada! Happy birthday to you! Nice presents
  awaiting you ..."
else
  echo "Sorry to disappoint you, no presents today ..."
fi
```

Create Basic Branches with the if Command (continued)

- elif command

```
if condition1
then
  command1
elif condition2
  command2
else
  command3
fi
```

- Logical separators
  - && and ||

Create Basic Branches with the if Command (continued)

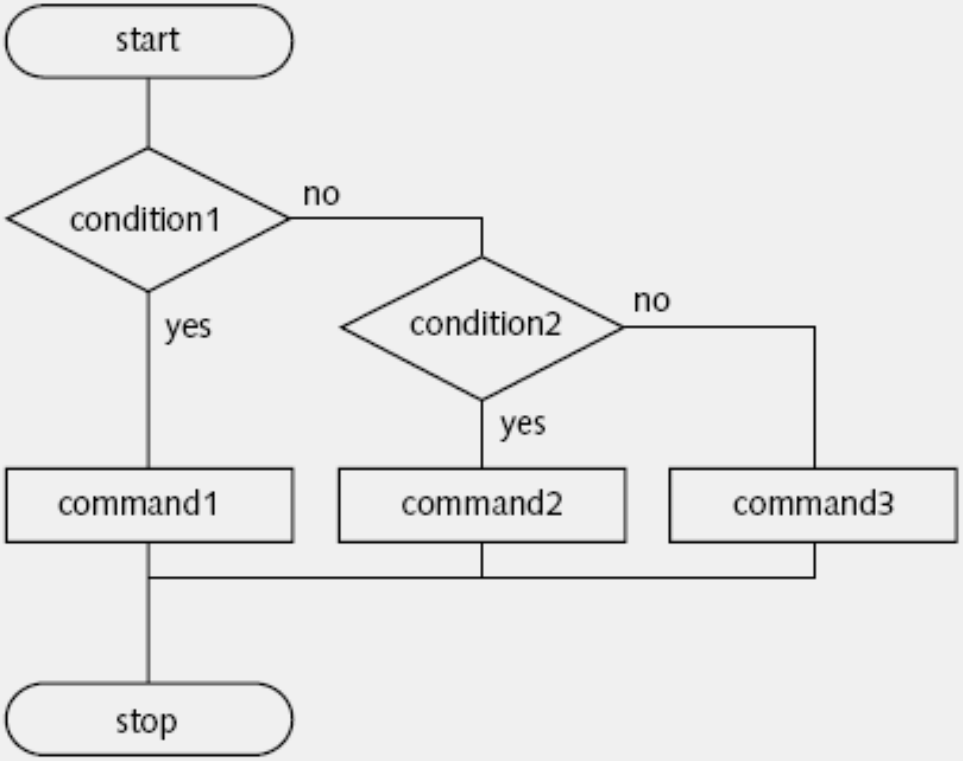


Figure 6-9

Exercise 6-7 Use the if Command

- In this exercise, you will practice using the if command

Build Multiple Branches with a case Statement

- case statement structure

```
case $variable in
  expression1) command1;;
  expression2) command2;;
esac
```

- Example

```
#!/bin/bash
cat << EOF
Name me an animal and I will tell you how many
  legs it
has!
EOF
```

Build Multiple Branches with a case Statement (continued)

- Example (continued)

```
read CREATURE
case "$CREATURE" in
    dog | cat | mouse ) echo "A $CREATURE has 4 legs."
    ;;
    bird | human | monkey ) echo "A $CREATURE has 2 legs."
    ;;
    spider ) echo "A $CREATURE has 8 legs."
    ;;
    fly ) echo "A $CREATURE has 6 legs."
    ;;
    * ) echo "I haven t the faintest idea how many
    legs a(n) $CREATURE has."
    ;;
esac
exit 0
```

Build Multiple Branches with a case Statement (continued)

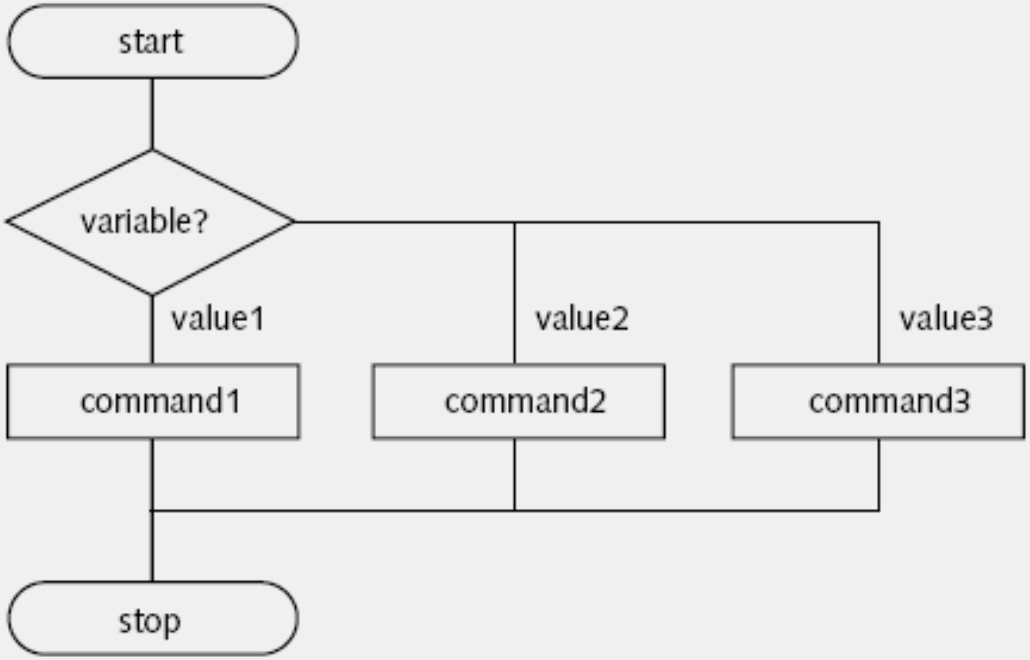


Figure 6-10

Exercise 6-8 Use the case Command

- In this exercise, you will practice using the case command

Create Loops Using the while and until Commands

- while loop

```
while condition
do
  commands
done
```

- until loop

```
until condition
do
  commands
done
```

Create Loops Using the while and until Commands (continued)

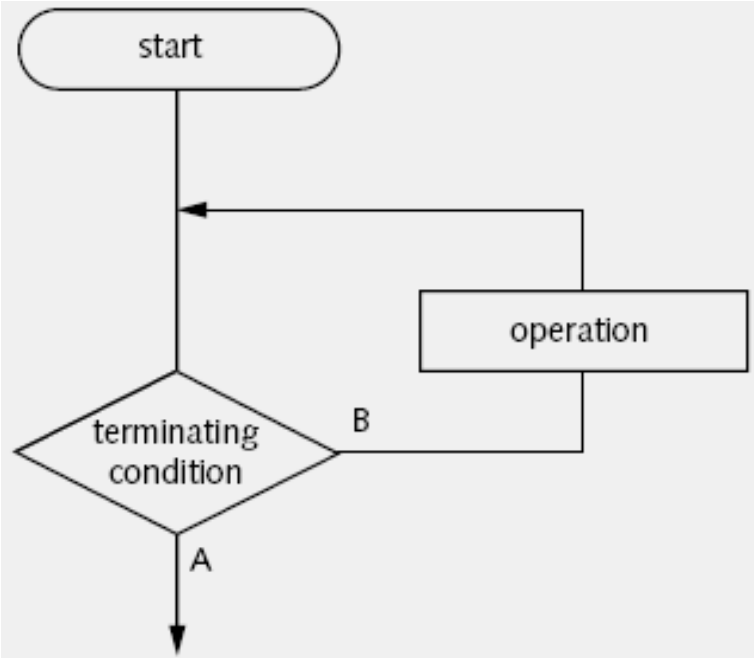


Figure 6-11

Exercise 6-9 Use the while and until Command

- In this exercise, you will practice using the while and until commands

Process Lists with the for Loop

- for loop

```
for variable in element1 element2 element3
do
  commands
done
```

- Example

```
LIMIT=10
for ((a=1; a <= LIMIT ; a++))
do
  echo -n "$a "
done
```

Exercise 6-10 Use the for Loop

- In this exercise, you will practice using the for loop

Interrupt Loop Processing

- continue command
  - Exits from the current iteration of a loop
  - Resumes with the next iteration of the loop
- Example

```
for FILE in ls *.mp3
do
   if test -e /MP3/$FILE
   then
    echo "The file $FILE exists."
    continue
   fi
   cp $FILE /MP3
done
```

Exercise 6-11 Interrupt Loop Processing

- In this exercise, you will practice using the continue command to interrupt loop processing

Use Advanced Scripting Techniques

- Options
  - Use Shell Functions
  - Read Options with getopts

Use Shell Functions

- Use functions
  - To perform a task multiple times
- Shell functions
  - Normally defined at the beginning of a script
  - Syntax

```
functionname () {
  commands
  commands
}
```

Use Shell Functions (continued)

- Generate a function

```
function functionname {
   commands
   commands
}
```

- Example

```
# mcd: mkdir + cd; creates a new directory and
# changes into that new directory right away
mcd (){
   mkdir $1
   cd $1
}
...
mcd directory
...
```

Exercise 6-12 Use Shell Functions

- In this exercise, you will practice using shell functions

Read Options with getopts

- getopts
  - Extracts options supplied to a script on the command line
- Shell command-line arguments
  - Command options prefixed with a -
- Syntax
  - getopts optionstring variable
    - *optionstring* describes all options to be recognized
    - Option string is followed by a *variable*

Read Options with getopts (continued)

- Example

```
while getopts abc: variable
do
  case $variable in
   a ) echo "The option -a was used." ;;
   b ) echo "The option -b was used." ;;
   c ) option_c="$OPTARG"
       echo "Option c has been set." ;;
  esac
done

echo $option_c
```

Exercise 6-13 Use the getopts Command

- In this exercise, you will practice using the getopts command

Learn About Useful Commands in Shell Scripts

- Objectives
  - Use the cat Command
  - Use the cut Command
  - Use the date Command
  - Use the echo Command
  - Use the grep and egrep Commands
  - Use the sed Command
  - Use the test Command
  - Use the tr Command

Use the cat Command

- Combined with the << (here) operator
  - Outputs several lines of text from a script
- Interactive use
  - Mostly run with a filename as an argument
  - Prints the file contents on standard output

Use the cut Command

- Cuts out sections of lines from a file
  - Specified section is printed on standard output
  - Use cut -f to cut out text fields
  - cut -c works with the specified characters
- You can specify single sections (characters or fields)
  - Or several sections
- Default delimiter to separate fields from each other is a tab
  - Specify a different field separator with the -d option

Use the date Command

- Used to obtain a date or time string
  - For further processing by a script
- Examples

```
date -I
2007-09-03
date +%m-%d %H:%M
09-03 14:19
date +%D, %r
09/03/07, 02:19:58 PM
date +%A, %e. %B %Y
Friday, 3. September 2007
```

Use the echo Command

- Prints text lines on standard output
  - Line break is inserted automatically after each line
  - With the -e option, echo accepts a number of additional options
- Special sequences
  - \a – outputs an alert
  - \c – do not add a new line
  - \n – add a new line

Use the grep and egrep Commands

- Used to search files for certain patterns
- Syntax
  - grep searchpattern filename ...
- Commands support various options
- Example

```
tux@DA1:~> egrep (b|B)lurb file*
bash: syntax error near unexpected token |

tux@DA1:~> egrep "(b|B)lurb" file*
file1:blurb
file2:Blurb
```

Use the sed Command

- Sed is a stream editor
  - Editor used from the command line rather than interactively
- Syntax
  - sed *editing-command filename*
- Available editing commands
  - d: Delete
  - s: Substitute (replace)
  - p: Output line
  - a: Append after

Use the sed Command (continued)

- Command-line options include
  - -n, --quiet, --silent
  - -e *command1* -e *command2* ....
  - -f *filename*
- Supports regular expressions

Use the sed Command (continued)

Table 6-2

Command	Example	Editing Action
d	sed 10,$d *file*	Delete line.
a	sed 'a*text**text*' *file*	Insert text before the specified line.
i	sed 'i*text**text*' *file*	Replace specified lines with the text.
c	sed '2000,$c*text* ' *file*	Replace specified lines with the text.
s	sed s/*x*/*y*/*option*	Search and replace. The search pattern *x* is replaced with pattern *y*. The search and the replacement pattern are regular expressions in most cases and the search and replace behavior can be influenced through various options.
y	sed y/abc/xyz/	(yank) Replace every character from the set of source characters with the character that has the same position in the set of destination characters.

Use the test Command

- Used to compare values
  - And to check for files and their properties
- If a tested condition is true
  - test returns an exit status of 0
  - Otherwise, test returns an exit status of 1
- Syntax
  - test *condition*
- Testing whether a file exists
  - See Table 6-3

Use the test Command (continued)

Table 6-3

Option	Description
-e	File exists
-f	File exists and is a regular file
-d	File exists and is a directory
-x	File exists and is an executable file

Use the test Command (continued)

- Comparing two files
  - See Table 6-4
- Comparing two integers
  - See Table 6-5
- Testing strings
  - See Table 6-6

Use the test Command (continued)

Table 6-4

Option	Description
-nt	Newer than
-ot	Older than
-ef	Refers to the same inode (such as in the case of a hard link)

Use the test Command (continued)

Table 6-5

Option	Description
-eq	Equal
-ne	Not equal
-gt	Greater than
-lt	Less than
-ge	Greater than or equal
-le	Less than or equal

Use the test Command (continued)

Table 6-6

Option	Description
test -z *string*	Exit status is 0 (true) if the string has zero length (is empty).
test *string*	Exit status is 0 (true) if the string has nonzero length (consists of at least one character).
test *string1* = *string2*	Exit status is 0 (true) if the strings are equal.
test *string1* != *string2*	Exit status is 0 (true) if the strings are not equal.

Use the test Command (continued)

- Combined tests
  - See Table 6-7

Use the test Command (continued)

Table 6-7

Option	Description
test ! *condition*	Exit status is 0 (true) if the condition is not true.
test *condition1* -a *condition2*	Exit status is 0 (true) if both conditions are true.
test *condition1* -o *condition2*	Exit status is 0 (true) if either condition is true.

Use the tr Command

- Used to translate (replace) or delete characters
- Reads from standard input
  - Prints the result on standard output
- Syntax
  - tr *set1 set2*
- Examples
  - cat text-file | tr a-z A-Z
  - tr -d set1
  - VAR=‘echo $VAR | tr -d %’
  - tr -s *set1 char*

Summary

- Shell scripts contain Linux commands that execute in the shell
- Before creating a shell script
  - Use flow chart symbols to plan its flow control
- Shell scripts start with a shebang
  - Denotes the shell that is called to run the commands
- User input may be obtained using the read command
  - echo command sends output to the terminal screen
- Backticks are used to perform command substitution

Summary (continued)

- Arithmetic operations performed with the expr, declare, and let commands
- Value of a variable may be modified using a variable substitution operator
- Flow of a program may be modified using one of several control structures
  - if, case
  - while, until, for
  - continue, break
  - &&, || operators

Summary (continued)

- You can create reusable subroutines within shell scripts called functions
- getopts command
  - May be used within a shell script to obtain the options specified on the command line
- Most common Linux commands used within a shell script include:
  - cat, cut, date, echo, grep, egrep, sed, test, and tr