# Unix and Linux

*Still More Shell Script Programming*

# Flow Control Statements

Other flow control statements in addition to if statements,:

The `while` loop—Executes a block of code repeatedly as long as the conditional statement is true.

The `until` loop—Executes a block of code repeatedly as long as the conditional statement is false. Essentially the opposite of a while loop.

The `case` statement—Similar to an if statement but provides an easier branching method for multiple situations.

The `for` loop—Executes a block of code for each item of a list of values.

## for loop

The IFS environment variable is the internal field separator

Create the script `path.sh` with this code and test it:

```
#! /bin/bash
# path.sh
IFS=:

for dir in $PATH
do
      ls -ld $dir
done
```

## Another for loop

Create the file users.sh in your bin directory containing:

```
#! /bin/bash
#users

for USERS in john ellen tom becky eli
do
    echo $USERS
done
```

Make it executable and test it.

## Case logic syntax

```
case expression in
     pattern1 )
          statements ;;
     pattern2 )
          statements ;;
     ...
esac
```

## `case` statement key points

- `case` statement first expands the expression and tries to match it against each pattern.
- When a match is found all of the associated statements until the double semicolon (;;) are executed.
- After the first match, case terminates with the exit status of the last command that was executed.
- If there is no match, exit status of case is zero.

## `case` example

Create the script colors.sh and test it

```
#! /bin/bash
# colors
echo -n "Enter your favorite color: "; read color
 case "$color" in
    "blue") echo "As in My Blue Heaven.";;
    "yellow") echo "As in the Yellow Sunset.";;
    "red") echo "As in Red Rover, Red Rover.";;
    "orange") echo "As in Autumn has shades of Orange.";;
    *) echo "Sorry, I do not know that color.";;
 esac
```

## Shell Variables

- Basic guidelines for handling shell variables:
  - Omit spaces when you assign a variable without using single/double quotation marks around value
  - To assign a variable that must contain spaces, enclose value in “” or ‘’
  - To reference a variable, use a $ in front of it or enclose it in { }
  - Use [ ] to refer to a specific value in an array
  - Export a shell variable to make the variable available to other scripts
  - To make a shell variable read-only: *readonly fname*

## Shell Variables (continued)

- Sample guidelines for naming shell variables:
  - Avoid using dollar sign in variable names
  - Use descriptive names
  - Use capitalization appropriately and consistently
  - If a variable name is to consist of two or more words, use underscores between the words

## Create a variable and assign a value

```
DOG=Poodle
```

What is the difference:

```
    echo DOG
    echo $DOG
    echo ‘$DOG’
    echo “$DOG”
```

## The backquote operator

The backquote is on the key in the upper left corner of your keyboard, under the ~

```
TODAY=`date`

echo $TODAY
```

## Backquote example

Create the file backquote.sh and test it

```
#! /bin/bash
#backquote.sh

lines=`cat $1 | wc -l`
echo The number of lines in the file $1 is $lines
```

To test this script, type the name of the script and the name of another file on the same line.

## Backup your bin directory

First create a backup directory named “~/binbackups”

Create the script binbackup.sh and test it

```
#!/bin/bash
# binbackup.sh

date=`date +%F`
mkdir ~/binbackups/$date
cp -R ~/bin ~/binbackups/$date
echo “Backup of bin directory completed”
```

## Quiz 4 and review quiz

If you have created and successfully tested all the programs, take Quiz #4 until you get a perfect score.

After you get a perfect score on Quiz #4, take the Review Quiz as many times as you can.