Lecture 8

Shell Programming – Control Constructs

COP 3353 Introduction to UNIX

Command Line Arguments Cont.

- $# contains the number of command line arguments.
- $@ will be replaced by a string containing the command line arguments
- Example script echo.sh

```
#!/bin/sh
echo "The" $# "arguments entered:" $@
```

- Usage:

```
echo.sh alpha beta gamma
```

- Output:

```
The 3 arguments entered: alpha beta gamma
```

Testing Conditions

- There are two ways to test for conditions. The two general forms are:

  test <condition>

  or

  [<condition>]

- The latter method is easier to read. Remember to include a space before and after the bracket
- A condition can be reversed with a ! before the condition (this is the same as not condition)

  [!<condition>]

- A ‘:’ command in place of condition always returns true

Testing File Attributes

- To test if a file is readable

  [-r prog.txt]

  [-r $1.c]
- To test if a file is writeable

  [-w specialfile.txt]
- To test if a file is executable

  [-x prog4.sh]
- To test if a file exists

  [-f temp.text]
- Testing for the negation - use ! (eg. not writeable)

  [! -w nochange.txt]

Numeric Tests

- The following operators can be used for numeric tests:

  { -eq, -ne, -gt, -ge, -lt, -le }

- Examples

  [$1 –lt $2]

  [$1 –gt 0]

  [$# -eq 2]

  [$# -lt 3]

Simple If Statement

- General Form:

  if <condition>

  then

  one-or more commands

  fi

- Example:

  if [-r tmp.text]

  then

  echo “temp.text is a readable file”

  fi

General If Statement

- General form:

  if <condition>
  then
  one-or-more-commands
  elif <condition>
  then
  one-or-more-commands
  …
  else
  one-or-more-commands
  fi

- Note that you can have 0 or more elif statements and that the else is optional.

Testing Strings

- Performing string comparisons. It is a good idea to put the shell variable being tested inside double quotes.

  [“$1” = “yes”]

  [“$2” != “no”]

- Note that the following will give a syntax error when $1 is empty since:

  [$1 != “no”]

- becomes

  [!= “no”]

Testing with Multiple Conditions

- && is the *and* operator
- || is the *or* operator
- checking for the *and* of several conditions

  [“$1” = “yes”] && [-r $2.txt]

  [“$1” = “no”] && [$# -eq 1]
- checking for the *or* of several conditions

  [“$1” = “no”] || [“$2” = “maybe”]

Quoting Rules

- Using single quotes

  ‘xyz’ disables all special characters in xyz

- Using double quotes

  “xyz” disables all special characters in xyz except $, `, and \.

- using the backslash

  \x disables the special meaning of character x

Quoting Examples

```
var1=“alpha”          #set the variable
echo $var1            #prints: alpha
echo “$var1”          #prints: alpha
echo ‘$var1’          #prints: $var1
cost=2000
echo ‘cost:$cost’     #prints: cost:$cost
echo “cost:$cost”     #prints: cost:2000
echo “cost:\$cost”    #prints: cost:$cost
echo “cost:\$$cost”   #prints: cost:$2000
```

More on String Relational Operators

- The set of string relational operators are:

  { =, !=, >, >=, <, <= }

- The { >, >=, <, <= } operators assume an ASCII ordering (for example “a” < “c”). These operators are used with the expr command that computes an expression. The backslash has to be used before the operators so that they are not confused with I/O redirection

Using Exit

- The exit command causes the current shell script to terminate. There is an implicit exit at the end of each shell script. The exit command can set the *status* at the time of exit. If the status is not provided, the script will exit with the status of the last command.
- General form:

  exit
- or

  exit <status>
- $? is set to the value of the last executed command
- Zero normally indicates success. Nonzero values indicate some type of failure. Thus, exit 0 is normally used to indicate that the script terminated without errors.

If Statement Examples

```
if [ “$1” != “” ] || [ -r $1 ]
then
    echo “the file” $1 “is not readable”
fi

if [ $var1 –lt $var2 ]
then
    echo $var1 “is less than” $var2
elif [ $var1 –gt $var2 ]
then
    echo $var1 “is greater than” $var2
else
    echo $var1 “is equal to” $var2
fi
```

Case Statement

- Compares stringvalue to each of the strings in the patterns. At a match, it does the corresponding commands. ;; indicates to jump to the statement after the esac (end of case). *) means the default case.
- Form:

```
case stringvalue in
pattern1) one or more commands;;
pattern2) one or more commands;;
…
        *) one or more commands;;
esac
```

Case Statement Example

```
echo “do you want to remove  file $1?”
echo “ please enter yes or no”
read ans
case $ans in
“yes”) rm $1
      echo “file removed”
      ;;
“no”) echo “file not removed”
      ;;
      *) echo “do not understand your request”
esac
```

while and until statements

- while form:

  while <condition>
  do
  one or more commands
  done

- until form:

  until <condition>
  do
  one or more commands
  done

while and until examples

```
read cmd
while [ $cmd != “quit” ]
do
    …
    read cmd
done

read cmd
until [ $cmd = “quit” ]
do
    …
    read cmd
done
```

Using for in a directory

- Use the for loop to iterate through every file in a directory

```
for filename in *
 do
  echo $filename
done;
```

- You can replace * with *.doc to iterate through only .doc files in the current directory.

Good reference on scripting

- http://steve-parker.org/sh/sh.shtml