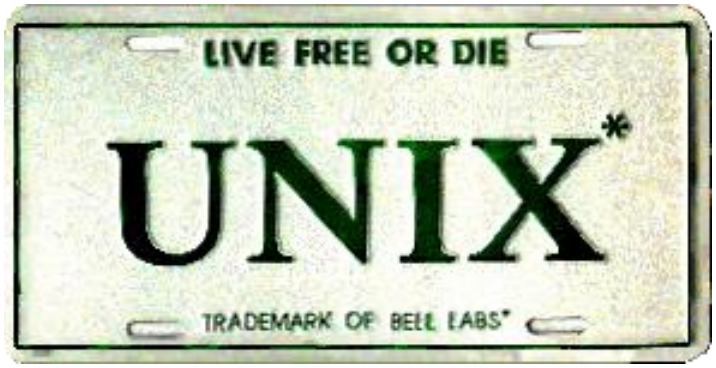

Shell Scripting

Slides adapted from Prof. Andrzej (AJ) Bieszczad

Read command

- The read command allows to prompt for input and store it in a variable.

- Example:

```
$ cat exp1
#!/bin/bash
echo -n Enter name of file to delete:
read file
echo "Type 'y' to remove it, 'n' to change your mind ... "
rm -i $file
echo That was YOUR decision!
```

Arithmetic Evaluation

- The let statement can be used to do mathematical functions:

```
$ let X=10+2*7
$ echo $X
24
$ let Y=X+2*4
$ echo $Y
32
```

- An arithmetic expression can be evaluated by $[expression] or $((expression))

```
$ echo $((123+20))
143
$ VALORE=$[123+20]
$ echo $[123*$VALORE]
17589
```

Arithmetic Evaluation

- Available operators: +, -, /, *, %

- Example

```
$ cat exp2
#!/bin/bash
echo -n Enter the first number: ; read x
echo -n Enter the second number: ; read y
add=$(($x + $y))
sub=$(($x - $y))
mul=$(($x * $y))
div=$(($x / $y))
mod=$(($x % $y))
# print out the answers:
echo Sum: $add
echo Difference: $sub
echo Product: $mul
echo Quotient: $div
echo Remainder: $mod
```

Conditional Statements

- Conditionals let us decide whether to perform an action or not, this decision is taken by evaluating an expression. The most basic form is:

```
if [ expression ];
then
        statements
elif [ expression ];
then
        statements
else
        statements
fi
```

- the elif (else if) and else sections are optional

- Put spaces after [and before], and around the operators and operands.

Expressions

- An expression can be: String comparison, Numeric comparison, File operators and Logical operators and it is represented by [expression]:

- String Comparisons:

=	compare if two strings are equal
!=	compare if two strings are not equal
-n	evaluate if string length is greater than zero
-z	evaluate if string length is equal to zero

- Examples:

[$s1 = $s2]	(true if s1 same as s2, else false)
[$s1 != $s2]	(true if s1 not same as s2, else false)
[$s1]	(true if s1 is not empty, else false)
[-n $s1]	(true if s1 has a length greater than 0, else false)
[-z $s1]	(true if s1 has a length of 0, otherwise false)

Expressions

- Examples: String Comparisons:

```
$ cat exp3
#!/bin/bash
s1= ; s2=UG1;s3=UG1
[ "$s1" = "$s2" ]; echo $?    #(true if s1 same as s2, else false)
[ "$s2" = "$s3" ]; echo $?    #(true if s2 same as s3, else false)
[ "$s1" != "$s2" ]; echo $?   #(true if s1 not same as s2, else false)
[ "$s2" != "$s3" ]; echo $?   #(true if s2 not same as s3, else false)
[ "$s2" ]; echo $?            #(true if s2 is not empty, else false)
[ -n "$s2" ]; echo $?         #(true if s2 has a length greater than 0, else false)
[ -z "$s2" ]; echo $?         #(true if s2 has a length of 0, otherwise false)
[ "$s1" ]; echo $?            #(true if s1 is not empty, else false)
[ -n "$s1" ]; echo $?         #(true if s1 has a length greater than 0, else false)
[ -z "$s1" ]; echo $?         #(true if s1 has a length of 0, otherwise false)
```

Examples

```
$ cat exp4
 #!/bin/bash
  echo -n Enter your login name:
  read name
  if [ $name ]; then
      if [ $name = $USER ]; then
              echo Hello, $name. How are you today?
      else
              echo You are not $USER, so who are you?
      fi
  else
      echo Username can not be empty!
  fi
```

Expressions

- Number Comparisons:

-eq	compare if two numbers are equal
-ge	compare if one number is greater than or equal to a number
-le	compare if one number is less than or equal to a number
-ne	compare if two numbers are not equal
-gt	compare if one number is greater than another number
-lt	compare if one number is less than another number

- Examples:

[n1 -eq n2]	(true if n1 same as n2, else false)
[n1 -ge n2]	(true if n1greater then or equal to n2, else false)
[n1 -le n2]	(true if n1 less then or equal to n2, else false)
[n1 -ne n2]	(true if n1 is not same as n2, else false)
[n1 -gt n2]	(true if n1 greater then n2, else false)
[n1 -lt n2]	(true if n1 less then n2, else false)

Examples

```
$ cat exp5
#!/bin/bash
  echo -n 'Enter a number 1 <= x <= 10:'
  read num
  if [ $num -le 10 ]; then
            if [ $num -ge 1 ]; then
                      echo $num*$num=$(($num*$num))
                      echo $num*$num=$[$num*$num]
            else
                      echo 'Wrong insertion! You have entered x < 1'
                      exit 1
            fi
  else
            echo 'Wrong insertion! You have entered x > 10'
            exit 1
  fi
```

Expressions

- Files operators:

-d	check if path given is a directory
-f	check if path given is a file
-e	check if file name exists
-s	check if a file has a length greater than 0
-r	check if read permission is set for file or directory
-w	check if write permission is set for a file or directory
-x	check if execute permission is set for a file or directory

- Examples:

[-d fname]	(true if fname is a directory, otherwise false)
[-f fname]	(true if fname is a file, otherwise false)
[-e fname]	(true if fname exists, otherwise false)
[-s fname]	(true if fname length is greater then 0, else false)
[-r fname]	(true if fname has the read permission, else false)
[-w fname]	(true if fname has the write permission, else false)
[-x fname]	(true if fname has the execute permission, else false)

Expressions

- Examples: File operators:

$ cat exp6

```
#!/bin/bash
touch xz1;mkdir xz2
[ -d xz1 ]; echo -n $?; [ -d xz2 ]; echo $?
[ -f xz1 ]; echo -n $?; [ -f xz2 ]; echo $?
[ -e xz1 ]; echo -n $?; [ -e xz2 ]; echo -n $?; [ -e xz3 ]; echo $?
[ -s xz1 ]; echo $?
echo do not give up > xz1
[ -s xz1 ]; echo $?
[ -r xz1 ]; echo $?
[ -w xz1 ]; echo $?
[ -x xz1 ]; echo $?
```

Example

$ cat exp7 (Copy the a file to a directory)

```
#!/bin/bash
echo -n Enter the file name:; read f1
echo -n Enter the directory name:; read d1
if [ -f "$f1" ]; then
    if [ ! -d "$d1" ]; then
        echo The $d1 directory does not exist, creating it.
        mkdir "$d1"
    fi
    cp "$f1" "$d1"
    echo Done.
 else
     echo This file does not exist.
     exit 1
 fi
```

Expressions

- Logical operators:

!	negate (NOT) a logical expression
-a	logically AND two logical expressions
-o	logically OR two logical expressions

Example:

$ cat exp8 (Compute the square of any number from 1 to 10)

```
#!/bin/bash
 echo -n 'Enter a number 1 < x < 10:' ; read num
 if [ $num -ge 1 -a $num -le 10 ];
 then
        echo $num*$num=$(($num*$num))
 else
        echo Wrong insertion!
        exit 1
 fi
```

Expressions

- Logical operators:

&&	logically AND two logical expressions
||	logically OR two logical expressions

Example:

$ cat exp9 (Compute the square of any number from 1 to 10)

```
#!/bin/bash
 echo -n 'Enter a number 1 < x < 10:' ; read num
 if [ $num -gt 1 ] && [ $num -lt 10 ];
 then
        echo $num*$num=$(($num*$num))
 else
        echo Wrong insertion!
        exit 1
 fi
```

Case Statement

- Used to execute statements based on specific values. Often used in place of an if statement if there are a large number of conditions.

- Value used can be an expression
- each set of statements must be ended by a pair of semicolons;
- a *) is used to accept any value not matched with list of values

```
case $var in
  val1)
        statements;;
  val2)
        statements;;
  *)
        statements;;
  esac
```

Example (case.bash)

```
$ cat exp10
#!/bin/bash
  echo -n Enter a number 1 \< x \< 10:
  read x
  case $x in
            1) echo “Value of x is 1.”;;
            2) echo “Value of x is 2.”;;
            3) echo “Value of x is 3.”;;
            4) echo “Value of x is 4.”;;
            5) echo “Value of x is 5.”;;
            6) echo “Value of x is 6.”;;
            7) echo “Value of x is 7.”;;
            8) echo “Value of x is 8.”;;
            9) echo “Value of x is 9.”;;
            10 | 0) echo “wrong number.”;;
            *) echo “Unrecognized value.”;;
  esac
```

Iteration Statements

- The for structure is used when you are looping through a range of variables.

```
for var in list
  do
    statements
  done
```

- statements are executed with var set to each value in the list.

Iteration Statements: Examples

```
$ cat exp11
#!/bin/bash
sum=0
for num in 1 2 3 4 5
   do
      sum=$[$sum + $num]
   done
echo $sum
```

Iteration Statements: Examples

```
$ cat exp12
#!/bin/bash
 for x in paper pencil pen
  do
    echo Value of variable x is: $x
    sleep 1
  done
```

Iteration Statements

- if the list part is left off, var is set to each parameter passed to the script ($1, $2, $3,…)

```
$ cat exp13
#!/bin/bash
 for x
  do
    echo Value of variable x is: $x
    sleep 1
  done
```

Bash Shell

Example (old.bash): Move the command line arg files to old directory

```
$ cat exp14
#!/bin/bash
if [ $# -eq 0 ] #check for command line arguments
then
  echo "Usage: $# file …"
  exit 1
fi
if [ ! -d "$HOME/old" ]
then
  mkdir "$HOME/old"
fi
echo The following files will be saved in the old directory:
echo $*
for file in $* #loop through all command line arguments
do
  mv $file "$HOME/old/"
  chmod 400 "$HOME/old/$file"
done
ls -l "$HOME/old"
```

Using Arrays with Loops

```
pet[0]=dog
pet[1]=cat
pet[2]=cow
```

or

```
pet=(dog cat cow)    #1024 elements can be used.
```

- To extract a value, type ${arrayname[i]}

```
$ echo ${pet[0]}
 dog
```

- To extract all the elements, use an asterisk as:

```
echo ${arrayname[*]}
 dog cat cow
```

Using Arrays with Loops: Example

```
$ cat exp15
#!/bin/bash
pet[0]=dog
pet[1]=cat
pet[2]=cow
echo Your pet animal is: ${pet[0]}
echo Your pet animal is: ${pet[1]}
echo Your pet animal is: ${pet[2]}
sleep 1
for x in ${pet[*]}
  do
    echo My pet animal is: $x
    sleep 1
done
```

Using Arrays with Loops: Example

```
$ cat exp16
#!/bin/bash
pet=(dog cat cow)
echo Your pet animal is: ${pet[0]}
echo Your pet animal is: ${pet[1]}
echo Your pet animal is: ${pet[2]}
sleep 1
for x in ${pet[*]}
  do
    echo My pet animal is: $x
    sleep 1
done
```

A C-like for loop

- An alternative form of the for structure is

```
for (( EXPR1 ; EXPR2 ; EXPR3 ))
 do
        statements
 done
```

- First, the arithmetic expression EXPR1 is evaluated.
- EXPR2 is then evaluated repeatedly until it evaluates false.
- Each time EXPR2 is evaluates to true, statements are executed and EXPR3 is evaluated.

A C-like for loop: Example

```
$ cat exp17          #add first x numbers
#!/bin/bash
echo -n Enter a number: ; read x
sum=0
for (( i=1 ; $i<$x+1 ; i=$i+1 )) ; do
   sum=$[$sum + $i]
done
echo The sum of the first $x numbers is: $sum
```

While Statements

- The while structure is a looping structure.
- Used to execute a set of commands while a specified condition is true.
- The loop terminates as soon as the condition becomes false.
- If condition never becomes false, loop will never exit.

```
while expression
 do
        statements
 done
```

While Statements

```
$ cat exp18          #add first x numbers
#!/bin/bash
echo -n Enter a number: ; read x
sum=0; i=1
while [ $i -le $x ]; do
   sum=$[$sum + $i]
   i=$[$i+1]
done
echo The sum of the first $x numbers is: $sum
```

Example: Menu

```
$ cat exp19
#!/bin/bash
  loop=y
  while [ "$loop" = y ] ;
  do
    echo "Menu";  echo "===="
    echo "D: print the date"
    echo "W: print the users who are currently log on."
    echo "P: print the working directory"
    echo "Q: quit."
    echo ;echo -n Enter your choice:
    read choice                    # silent mode: no echo to terminal
    case "$choice" in
              D | d) date ;;
              W | w) whoami ;;
              P | p) pwd ;;
              Q | q) loop=n ;;
              *) echo "Illegal choice." ;;
    esac
    echo
  done
```

Continue Statements

- The continue command causes a jump to the next iteration of the loop, skipping all the remaining commands in that particular loop cycle.

Example: Continue

$ cat exp20 Printing Numbers 1 through 20 (but not 3 and 11)

```
#!/bin/bash
  LIMIT=20
  echo
  echo "Printing Numbers 1 through 20 (but not 3 and 11)"
  a=1
  while [ "$a" -le "$LIMIT" ]; do
    if [ "$a" -eq 3 ] || [ "$a" -eq 11 ]
    then
        a=$(($a+1))
        continue
    fi
    echo "$a"
    a=$(($a+1))
  done
```

Break Statements

- The break command terminates the loop (breaks out of it).

```
$ cat exp21
#!/bin/bash
 echo
 echo "Printing Numbers; Stop at 20 … "
 a=1
 while TRUE
 do
   if [ "$a" -gt 20 ]
   then
        a=$(($a+1))
        break
   fi
   echo "$a"
    a=$(($a+1))
done
```

Until Statements

- The until structure is very similar to the while structure. The until structure loops until the condition is true. So basically it is “until this condition is true, do this”.

```
until [expression]
 do
        statements
 done
```

Until Statements

Example: countdown

```
$ cat exp22
#!/bin/bash
echo -n "Enter a number: "; read x
echo ; echo Count Down
until [ "$x" -le 0 ]; do
    echo $x
    x=$(($x-1))
    sleep 1
done
echo ; echo GO !
```

Manipulating Strings

- Bash supports a number of string manipulation operations.

${#string} gives the string length
${string:position} extracts sub-string from $string at $position
${string:position:length} extracts $length characters of sub-string from $string at $position

- Example

```
$ cat exp23
st=0123456789
echo ${#st}
echo ${st:6}
echo ${st:6:2}
```

Functions

- Functions make scripts easier to maintain. Basically it breaks up the program into smaller pieces. A function performs an action defined by you, and it can return a value if you wish.

```
$ cat exp24
#!/bin/bash
hello()
{
      echo "You are in function hello() "; sleep 2
}
echo "Calling function hello()… "; sleep 2
hello
echo "You are now out of function hello()"
```

- In the above, we called the hello() function by name by using the line: hello . When this line is executed, bash searches the script for the line hello(). It finds it right at the top, and executes its contents.

Debugging

- Bash provides two options which will give useful information for debugging

-x : displays each line of the script with variable substitution and before execution
-v : displays each line of the script as typed before execution

- Usage:

#!/bin/bash –v or #!/bin/bash –x or #!/bin/bash –xv

```
$ cat exp25
#!/bin/bash -x
echo -n Enter a number: ; read x
sum=0
for (( i=1 ; $i<$x+1 ; i=$i+1 )) ; do
   sum=$[$sum + $i]
done
echo The sum of the first $x numbers is: $sum
```

Debugging (Output)

```
$ exp25
+ echo -n Enter a number:
Enter a number:+ read x
5
+ sum=0
+ (( i=1  ))
+ (( 1<5+1  ))
+ sum=1
+ (( i=1+1  ))
+ (( 2<5+1  ))
+ sum=3
+ (( i=2+1  ))
+ (( 3<5+1  ))
+ sum=6
+ (( i=3+1  ))
+ (( 4<5+1  ))
+ sum=10
+ (( i=4+1  ))
+ (( 5<5+1  ))
+ sum=15
+ (( i=5+1  ))
+ (( 6<5+1  ))
+ echo The sum of the first 5 numbers is: 15
The sum of the first 5 numbers is: 15
```