

## Review

- Conditions
- The test, or '[' Command
- Control Structures
  - if statement
  - if-else-if statement
  - for loop statement
  - while loop statement
  - until loop statement
  - case statement

COSC350 System Software, Fall 2020  
Dr. Sang-Eon Park

1

## Preview

- Functions
  - Function with local variable
  - Function with return value
  - Bash recursive function
- Other Commands
  - break Command
  - continue Command
  - eval
  - exit
  - export
  - expr
  - printf
  - set
  - shift

COSC350 System Software, Fall 2020  
Dr. Sang-Eon Park

2

## Functions

- You can define functions in a shell script.
- SYNTAX:
 

```
function_name ()
{
    statement1
    statement2
    ...
}
```
- Function prototypes cannot be placed for calling a function before function definition.
- A function must be defined before the function call.

COSC350 System Software, Fall 2020  
Dr. Sang-Eon Park

3

## Functions

- When a function is called, the positional parameter the script \$\*, \$@, \$#, \$0, \$1, \$2, .. and so on are replaced by the parameters to the function.
- When the function finishes, the positional parameters are restored to their previous values.

COSC350 System Software, Fall 2020  
Dr. Sang-Eon Park

4

## Functions with Local Variables

```
#!/bin/sh
# function.sh: an example for a function definition

function_hello()
{
    local yourname
    echo -n "What is your name?"
    read yourname
    echo " Hello $yourname "
    echo "Parameter variables for the function_hello are $@"
}
echo "Parameters variables for this shell script are $@"
function_hello you your you yours
function_hello I mine me mine
echo "your name is $yourname"
echo "Parameters variables for this shell script are $@"
exit 0
```

COSC350 System Software, Fall 2020  
Dr. Sang-Eon Park

5

## Functions with Local Variables

- We can declare local variables within a function by using **local** keyword which in only in the function scope.
- If the local variable has same name as a global variable, it overlays that variable, but only within the function

COSC350 System Software, Fall 2020  
Dr. Sang-Eon Park

6

## Functions with Local Variables

```
#!/bin/sh
#local.sh for testing local variable
# local variable is in the function scope

yourlocation()
{
    local mylocation
    echo -n "where are you now?"
    read mylocation
    echo "He is now in $mylocation "
}

mylocation = "Salisbury"
yourlocation
echo "I am still in $mylocation"

exit 0
```

COSC350 System Software, Fall 2020  
Dr. Sang-Eon Park

7

```
#!/bin/bash
# GlobalLocal.sh
# Global and local variables inside a function.

LocalGlobal ()
{
    local loc_var=23      # Declared as local variable.
    echo echo "\"loc_var\" in function = $loc_var"
    global_var=999      #global variable
    echo "\"global_var\" in function = $global_var"
}

LocalGlobal

echo "\"loc_var\" outside function = $loc_var"
echo "\"global_var\" outside function = $global_var"

exit 0
```

COSC350 System Software, Fall 2020  
Dr. Sang-Eon Park

8

## Function with Return Value

- In bash, we can define a function with a return value.
- A function can **return 0 (true)** or **1(false)** as a result.

COSC350 System Software, Fall 2020  
Dr. Sang-Eon Park

9

```
#!/bin/sh
# myname.sh: demonstrate a function with return value
yes_or_no(){
    echo "Is your name $* ?"
    while true
    do
        echo -n "Enter yes or no"
        read x
        case "$x" in
            y | yes ) return 0;;
            n | no ) return 1;;
            *) echo "Answer yes or no"
        esac
    done
}

echo "Original parameter are $*"
if yes_or_no "$*"
then
    echo "Hi $*, nice name"
else
    echo "Never mind"
fi
exit 0
```

COSC350 System Software, Fall 2020  
Dr. Sang-Eon Park

10

## Recursive Function

- Does bash permit recursion?
- Yes, but it's so slow since it use big memory space.
- Running a script with recursion could possibly lock up your system!

COSC350 System Software, Fall 2020  
Dr. Sang-Eon Park

11

## Recursive Function

```
#!/bin/bash
# fact.sh - Shell script to find factorial of given command line arg
factorial()
{
    local i=$1
    local f
    # factorial() is called until the value of $f is returned and is it is <= 2
    # This is called the recursion
    if [ $i -le 2 ]; then
        echo $i
    else
        let f=$(( i - 1 ))
        let f=$(( factorial $f ))
        let f=$(( f * i ))
        echo $f
    fi
}

# display usage
if [ $# -eq 0 ]; then
    echo "Usage: need a number for input"
    exit 1
fi

# call factorial
factorial $1
exit 0
```

COSC350 System Software, Fall 2020  
Dr. Sang-Eon Park

12

```
#!/bin/bash
# recursive.sh
# Demonstration of recursion.

RECURSIONS=9 # of times to call function recurse().
r_count=0

recurse ()
{
  var="$1"

  while [ "$var" -ge 0 ]
  do
    echo "Recursion count = '$r_count'  ++ \ $var = '$var'"
    let var--
    let r_count++
    recurse "$var"
  done
}

recurse $RECURSIONS

exit 0
```

## Break command

- Exit from a for, while or until loop

### SYNTAX

break [*n*]

- If *n* is supplied, the *n*th enclosing loop is exited. *n* must be greater than or equal to 1.

## Break Command

```
for myloop in 1 2 3 4 5 do
  echo "$myloop"
  if [ "$myloop" -eq 3 ]
then
  break
fi
done
```

## Continue Command

- Resume the next iteration of an enclosing for, while, until, or select loop.

### SYNTAX

continue [*n*]

- If *n* is supplied, the execution of the *n*th enclosing loop is resumed.
- n* must be greater than or equal to 1.

## Continue

```
for myloop in 1 2 3 4 5
do
  if [ "$myloop" -eq 3 ]
  then
    continue # Skip rest of loop iteration.
  fi
  echo "$myloop"
done
```

## Other Commands: eval

- Indirect Variable References**
- eval: allows you to evaluate argument

```
foo=10
x=foo
y=\$x #it is same as y=''$x
echo $y
$foo
```

```
foo=10
x=foo
eval y=\$x
echo $y
10
```

```
foo=10
x=foo # x=foo=10
eval echo \$x
10
```

```
a=letter
letter=z # a=letter=z
echo "Now a = $a"
Now a = letter
eval a=\$a
echo "Now a = $a"
Now a = z
```

## Other Commands: exit n

- exit command cause the script to exit with exit code.
- In shell programming,
  - exit 0 : exit with success
  - exit 1 ~125 : exit with an error
  - exit 126 : reserved code the file was not executable
  - exit 127 : A command was not found
  - exit 128 : a signal occurred

## Other Commands: export

- Set an environment variables.
- Mark each *name* to be passed to child processes in the environment

## Other Commands: export

```
#!/bin/sh
# export.sh: export variables to subscripts
export gr1="Happy"
export gr2="New"
export gr3="Year"
./import.sh
./import1.sh.
exit 0
```

#!/bin/sh # import.sh gr1, gr2, gr3 might be # imported from parent script  echo "\$gr1 \$gr2 \$gr3" exit 0	#!/bin/sh # import1.sh: variable gr1 might be # imported from it's parent  echo "\$gr1 Birthday!" exit 0
--	---

## Other Commands: expr

- Evaluate expressions, evaluates an expression (arithmetic, logical) and writes the result on standard output.
- Syntax
  - `expr expression...`

```
#!/bin/sh
# exprArith.sh: expr example with arithmetic operators

a=5+3
echo "a is $a"

echo "Arithmetic Operators"
echo "-----"
a=`expr 5 + 3`
echo "now a = 5 + 3 = $a"

a=`expr $a + 1`
echo
echo "a + 1 = $a"

a=`expr 5 % 3` #same as let a=5%2
# modulo
echo
echo "5 mod 3 = $a"

exit 0
```

```
#!/bin/sh
# exprlogic.sh: expr with Logical Operators
# Returns 1 if true, 0 if false,
# opposite of normal Bash convention.

echo "Logical Operators"
echo "-----"
x=24
y=25
echo "x = $x and y = $y"
echo "Compare with = operator"
b=`expr $x = $y`
echo "b = $b means x == y is false"
echo "Compare with < operator"
b=`expr $x < $y`
echo "b = $b means x < y is true"

echo "There are more operators such as <, <= >=, ...."
exit 0
```

```
#!/bin/sh
# exprstring.sh : expr with string operators
a=987abCD1234RTab
echo "The string being operated upon is \"$a\"."

# length: length of string
b=`expr length $a`
echo "Length of \"$a\" is $b."

# index: position of first character in substring that matches a character in string
b=`expr index $a 2`
echo "Numerical position of first \"2\" in \"$a\" is \"$b\"."

# substr: extract substring, starting position & length specified
b=`expr substr $a 2 6`
echo "Substring of \"$a\", starting at position 2, \
and 6 chars long is \"$b\"."

# The default behavior of the 'match' operations is to search for the specified match at
the # beginning of the string. uses Regular Expressions
b=`expr match "$a" '[0-9]*'`      # Numerical count.
echo "Number of digits at the beginning of \"$a\" is $b."
b=`expr match "$a" '([0-9]*)'`   # Note that escaped parentheses
#                               + trigger substring match.
echo "The digits at the beginning of \"$a\" are \"$b\"."

exit 0
```

## Other Commands: printf

### Syntax

#### printf "Format strings", parameter list

- Format string : sequence of conversion specifier (%d, %c, %s)
- Parameter list and conversion specifier must be matched.

Ex)

```
printf "%d, %s, %c\\a" 2 "Hi" 'a'
```

## Other Commands: printf

- Escape Sequence in printf
- \b :backspace
- \f :form feed
- \n :Newline
- \r : carriage return
- \t : tap
- \v : vertical tap

```
#!/bin/sh
# printf.sh
#Print the decimal number 5 followed by a newline (\n)
printf "%d\n" 5

#Print as float (default 6 decimal places)
printf "%f\n" 5

#Print text followed by variable $USER
printf "Hello, $USER.\n\n"

#Display variables distance=15
distance=15
printf "Distance is %d Miles\n" $distance

#Use \n anywhere to start a new line:
printf "Two separate\nlines\n"

#Print decimal numbers interspersed with text
printf "There are %d orders valued at over %d euros.\n" 64 1500

#Convert a hex number to decimal
printf "%d\n" 0xf

#Convert a decimal number to Hex
printf "%x\n" 15

#Convert a decimal number to Octal
printf "%o\n" 8

#Convert an Octal number to decimal
printf "%d\n" 010

exit 0
```

## Other Commands: set

- The set command sets the parameter variables for the shell script.
- Position parameter can change inside a script
- We can use the result of a command as a input to other commands

```
#!/bin/sh

# script "set-test1.sh"

# Invoke this script with three command line parameters,
# for example, "./set-test one two three".

echo
echo "Positional parameters before set \"uname -a\" :"


echo "Command-line argument #1 = $1"



echo "Command-line argument #2 = $2"



echo "Command-line argument #3 = $3"



set `uname -a` # Sets the positional parameters to the output
                # of the command `uname -a`



echo $_      # last positional parameter



# Flags set in script.



echo "Positional parameters after set \"uname -a\" :"



# $1, $2, $3, etc. reinitialized to result of `uname -a`



i=1



for ARG in "$@" ; do



echo "Field \"$i\" of 'uname -a' = $ARG"



let i++



done



exit 0


```

## Other Commands: set

```
./set-test.sh one two three
```

```
$0=./set-test.sh
$1=one
$2=two
$3=three
```

```
set `uname -a`
```

```
$1=Linux
$2=claude
$3=2.6.32.12-0.7-pae
$4=#1
$5 = SMP
..
..
$_=GNU/Linux
```

## Other Commands: shift

- The shift command shift all positional parameter variable down by one. \$4 become \$3, \$3 become \$2..
- The previous value of \$1 is discarded \$0 (name of script) remains.

## Other Commands: shift

```
#!/bin/sh
# shft.sh: Using 'shift' to step through all the
# positional parameters
# need invoke shft.sh merry Christmas and happy new year

echo "arguments before shift "$*"
echo
until [ -z "$1" ] # Until all parameters used up . . .
do
    echo "$* "
    shift
done
exit 0
```