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

Functions

- You can define functions in a shell script.
- SYNTAX:
  ```bash
  function_name ()
  {
  statement1
  statement2
  …
  }
  ```
- Function prototypes cannot placed for calling a function before function definition.
- A function must be defined before the function call.

Functions with Local Variables

```bash
#!/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 yours
function_hello I mine me mine
echo "Parameters variables for this shell script are $#" exit 0
```

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.
- Functions with local variables:
  - Function with local variable
  - Function with return value
  - Bash recursive function
- Other Commands
  - break Command
  - continue Command
  - eval
  - exit
  - expr
  - printf
  - set
  - shift
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
    mylocation = "Salisbury"
    echo "I am now in $mylocation"
    exit 0
}
mylocation = "Salisbury"
yourlocation
```

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.

Recursive Function

- Does bash permit recursion?
  - Yes, but it's so slow since it uses big memory space.
  - Running a script with recursion could possibly lock up your system!
#!/bin/bash
# recursive.sh
# Demonstration of recursion.
RECURSIONS=9   # How many times to recurse.
r_count=0
recurse ()
  var="$1"
  while [ "$var" -ge 0 ]
    do
      echo "Recursion count = "$r_count"  + "$var"
      (( var-- )); (( 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 nth enclosing loop is exited. n must be greater than or equal to 1.

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 nth enclosing loop is resumed.
- n must be greater than or equal to 1.

Other Commands: eval

- Indirect Variable References

  - eval: allows you to evaluate argument

  ```bash
  foo=10
  x=$x # it is same as y=5'x'
  echo $y
  $$foo
  ```

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

  ```bash
  foo=10
  x=foo
  eval echo $$x
  ```

  ```bash
  ```
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: expr

- Evaluate expressions, evaluates an expression and writes the result on standard output.
- Syntax
  - `expr expression...`
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: shift

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

```
#!/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 [ $- = "$1" ] # Until all parameters used up . . .
do   echo "$* 
    shift
done
exit 0
```