Seneca | SCHOOL OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

Unix Scripting

Lecturer: Shahdad Shariatmadari

July 2020

Agenda

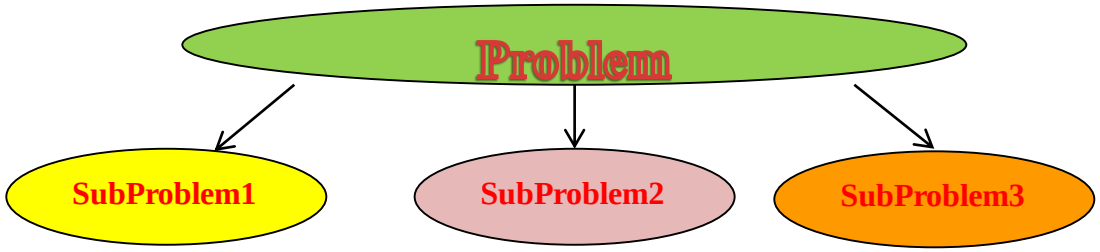
- Function/Procedure
  - How to declare
  - How to use

Modular programming

- When a program becomes very large and complex, it becomes very difficult task for the programmer to design, test and debug such a program.
- Therefore, a long program can be divided into a smaller program called **modules**.
- Programs are composed of one or more independently developed **modules**

Modular programming

- Modular programming is a software design technique that emphasize separating the functionality of a program into independent, interchangeable modules.
- The main idea is to divide the complex problem into small tasks and solve each task, then combine these solutions.



Modular Programming

- What is Module?
  - Section of code that performs a specific task
- What the module does:
  - It is look like a program (input, processing,output)
- Other names which refer to Module are:
  - Sub-program
  - Function
  - Procedure
  - Method

Subprograms in UNIX (Example)

```
#Declare subprogram
Read_Data(){
        #code appear here
}
Process_Data(){
        #code appear here
}
Display_Result(){
        #code appear here
}
```

```
#Main program
#Call Modules
Read_Data
Process_Data
Display_Result
```

Function vs Procedure

- In programming , both functions and procedures are a sub-program which consist of set of commands.
- Both have
  - Input data (we call it parameter)
  - Processing commands (process data)
- However, the difference is only in the returning a value part.

Function vs Procedure

- A procedure just executes commands (no return value)
  - `proc_name(){`
    - `statement(s)`
  - `}`
- A function must return a value
  - `func_name(){`
    - `statement(s)`
    - **`return(value)`**
  - `}`

Example(Procedure)

```
#declare the sub-program or module
addEmUp(){
   echo 'What is the first number? '
   read one
   echo 'What is the second number? '
   read two
   result = $((one + two))
   echo  $result
 }
#main program
addEmUp
```

Example

```
#!/bin/sh
# Define your subprogram here
Hello () {
echo "Hello World"
}
# Invoke your function
Hello
```

Function

- Bash functions, unlike functions in most programming languages do not allow you to return a value to the caller.
- When a bash function ends its return value is its status: zero for success, non-zero for failure.

Pass Parameters to a subprogram

- You can define a function that will accept parameters while calling the function. These parameters would be represented by **$1**, **$2** and so on.

```
#!/bin/sh
# Define your subprogram
Hello () {
echo "Hello World $1 $2"
}

# Invoke your function
Hello John Mary
```

Variables/Parameters

- Using $1, $2,… as passing parameters
  - Use shift command
  - Use $@
- No scoping!
  - Other than $1, $2,…

Activity: Explain how does the following script work? Develop a script and demonstrate it

```
#!/bin/sh

myfunc()
{
  echo "\$1 is $1"
  echo "\$2 is $2"
  # cannot change $1 - we'd have to
say:
  # 1="Goodbye Cruel"
  # which is not a valid syntax.
However, we can
  # change $a:
  a="Goodbye Cruel"
}
```

```
### Main script starts here

a=Hello
b=World
myfunc $a $b
echo "a is $a"
echo "b is $b"
```

Activity: Explain how does the following script work? Develop a script and demonstrate it

```
add_a_user()
{
  USER=$1
  PASSWORD=$2
  shift; shift;
  # Having shifted twice, the rest is now comments ...
  COMMENTS=$@
  echo "Adding user $USER ..."
  echo useradd -c "$COMMENTS" $USER
  echo passwd $USER $PASSWORD
  echo "Added user $USER ($COMMENTS) with pass $PASSWORD"
}
```