HOP09 – BASH – Parameters and Variable

IS 340 – Operating Systems

School of Technology & Computing (STC) @ City University of Seattle (CityU)

> 7/16/2019 Developed by Kevin Wang
> 6/17/2020 Reviewed by Kim Nguyen
> 02/14/2022 Reviewed by Ken Ling
> 09/06/2022 Modified by Thaddeus Thomas

Before You Start

- This exercise assumes that the user is working with the Ubuntu 18.04 distribution. If you are working with a different Linux distribution, the set of shell commands may vary from those available in Ubuntu 18.04.
- Students will use the EC2 Ubuntu virtual machine that they created in the module 1 exercise.
- All commands and code discussed in this exercise will run in the Ubuntu console.
- The directory path shown in screenshots may be different from yours.
- Some steps are not explained in the tutorial. If you are not sure what to do:
    1. Consult the resources listed below and experiment in the Ubuntu console and try to solve the problem yourself.
    2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes

1. Students will be able to:
    - Understand the scope of variables
    - Use the parameter expansion
    - Use arrays

Resources

- Linux command line: bash + utilities
- Nano/Basics Guide

Preparation

1. Connect to your Ubuntu instance
    - Open a command prompt
    - Syntax: ssh -i LOCATION_OF_YOUR_KEY ubuntu@PUBLIC_DNS
    - Example:

```
ssh -i key.pem ubuntu@ec2-33-222-101-222.us-west-
2.compute.amazonaws.com
```

2. `git pull https://github.com/cityuseattle/IS340-Fall-2020-Assignment.git` (to get the most updated content from source repository). If you are prompted to type a message, you can skip this by typing `:wq + Enter`
3. !!! NOTE: FOR EACH WEEK, YOU NEED TO DO THIS STEP BEFORE YOU START CODING!!!
4. Change directory to the corresponding folder of each week.
    - For example: Your work for module 1 should be stored under Module 1 folder;
    - your work for module 2 should be stored under Module 2, and so on:

```
cd "Module 7"
```

5. Now, follow the instructions provided in each folder to complete your Hands-on Practice

Variable Scope

1. The variable is defined in the shell will be just visible in the shell.
2. No script that is called by the shell can see it. Test this by following the instructions below:
    - Type the following command to define a variable in the shell:

```
name=Thaddeus
```

    - Type the following command to create a script file:

```
nano NameTest.sh
```

    - Type the script in the file as below to print out the variable name:

```
#! /bin/bash
printf "%s\n" "$name"
```

3. Hit [CTRL+X] key to quit and save the file
4. Test the script by typing the following commands:

```
bash NameTest.sh
```

Note: you are supposed to see an empty output since the name variable is defined in the shell instead of in the script file (The script knows nothing about it).

5. Export the variable to the environment by typing the following commands:

```
export name=Thaddeus
```

   Note: we export the name variable to the environment.

6. Try to run the script again:

```
bash NameTest.sh
```

   Now our script can read the variable name we defined in the shell.

7. Use unset command to remove a variable from the environment. You will see the empty output again.

```
unset name
bash NameTest.sh
```

8. The variables are defined in the subshell will not be visible to the shell that called it. This is a common problem when we use the pipe functionality. Type the following command to create a file to test it:

```
nano SubshellTest.sh
```

9. Type the following script in the file:

```
#! /bin/bash
sum=0
printf "Add sum to these numbers:\n"
printf "%s\n" ${RANDOM}{,,,} |
while read num
do
    printf "The current number is %d\n" "$num"
    ((sum=sum+num))
    printf "The current sum number is %d\n" "$sum"
done
printf "Now the loop is finished.\n\nThe final sum is %d\n" "$sum"
```

   Note: we generate several random numbers and use the pipe to pass them to a while loop. The while loop adds them together and put the total value to the sum variable.

10. Hit [CTRL+X] key to quit and save

11. Type the following command to run the script:

```
bash SubshellTest.sh
```

> As you can see from the output, the sum variable is visible inside of the while loop but not visible after the while loop. This is because we are using the pipe operation | to run the while loop, which causes it runs in a subshell and the variable in the subshell is not visible in the caller environment.

Parameter Expansion

1. Copy the `ShowArguments.sh` file we made in the module7 by typing the follow command:

```
cp ../Module7/ShowArguments.sh ./
```

```
#! /bin/bash
index=1
for arg in "$@"
do
    printf "Argument %d: %s\n" "$index" "$arg"
    index=$(( $index + 1 ))
done
```

2. Type the following command to test how to give a default value to a parameter:

```
arg=
bash ShowArguments.sh "${arg:-defaultArg}"
```

Note: we set an empty arg as the argument and call the script with a default value "defaultArg". So, the script will take the defaultArg as the parameter value.

```
arg=value
bash ShowArguments.sh "${arg:-defaultArg}"
```

> This time we defined a value for arg variable. So, when we run the script, it will use that value instead of the default value.

3. If we omit the colon, the script will just check whether the variable is unset. Test it by typing the following commands:

```
arg=
bash ShowArguments.sh "${arg-defaultArg}"
```

> Even the arg is no value, the default value will not be applied since the script knows it is not unset.

4. `${var:=default}` and `${var=default}` can define a default value for a variable. Test it by typing the follow command:

```
unset arg
echo $((1+${arg:=100}))
```

5. `${var:?message}` and `${var?message}` will show the message when the variable is empty or unset. Test it by typing the follow command:

```
arg=
echo $((1+${arg:?EmptyVariable}))
```

6. `${#var}` can show the length of a variable. Test it by typing the follow command:

```
var=text
echo ${#var}
```

7. `${var%PATTERN}` will remove the shortest match from the end. Test it by typing the follow command:

```
var=testtexttext
echo ${var%text*}
```

8. `${var%%PATTERN}` will remove the longest match from the end. Test it by typing the following command:

```
echo ${var%%text*}
```

9. `${var#PATTERN}` and `${var##PATTERN}` will remove the shortest and longest match from the beginning. Test them by typing the follow command:

```
var=testtesttext
echo ${var#*test}
echo ${var##*test}
```

10. `${var//PATTERN/STRING}` can help to replace the match pattern with the given string. Test it by typing the follow command:

```
var=teabct
echo ${var/abc/s}
```

11. `${var:OFFSET:LENGTH}` can return a substring of a variable. Test it by typing the following command:

```
var=testtext
echo ${var:4}
```

Note: the length is optional. If a length parameter is not supplied, the command will return all string from the offset position.

Using arrays

1. Create an array by typing the follow command:

```
unset name
name[0]=Thaddeus
name[1]=Strong
echo ${name[0]}
echo ${name[1]}
```

2. You also can set an array with all value in one command:

```
unset name
name=( Thaddeus R Strong )
printf "%s\n" "${name[@]}"
```

3. Associative Arrays allows developers to use strings as index. Test it by typing the follow commands:

```
declare -A strArr
strArr[Thaddeus]="Thaddeus's address"
strArr[R]="R's address"
echo ${strArr[Thaddeus]}
echo ${strArr[R]}
```

Note: the associative arrays that use string as indexes must be declare before use.

Submit your Work to Brightspace

1. Please upload all your files for this hands-on practice to the HOP assignment on Brightspace.