CE350
Lecture9

PROGRAMMING THE BASH SHELL
PART V

by İlker Korkmaz and Kaya Oğuz

Programming the bash

- As pointed in syllabus document, there are 5 lecture weeks to cover the programming with bash scripts.
- This lecture is the last part on programming the bash shell.

References

- The contents of this lecture are prepared with the help of and based on:
  - the textbook, UNIX Shells by Example (Chapter15)
  - the tutorial, Advanced Bash-Scripting Guide at http://tldp.org/LDP/abs/html/

Contents of Lecture9

- Debugging shell scripts

Why to cover the debugging subject ?

- Chapter15 of the textbook, UNIX Shells by Example, aims to provide the debugging concept for finding, fixing, and understanding some types of errors that cause shell scripts to misbehave.
- Although they have a successful script, the system administrators will usually want to make it better to maintain. In this sense, tracing the script may be helpful.
- Therefore, a good script is not only a successful one, but owns the good style issues as well.

Lecture design

- At first, the style issues are introduced.
- Secondly, the types of errors are categorized.
- Finally, tracing options for debugging the running codes are presented.
- At the end of the lecture, your common mistakes on styling issues are listed according to the observation on your HW and LAB works submitted.

Style Issues

- To be written in a good style is an important issue for any program in deed.
- A good style in script design can be helpful to quickly find possible bugs in the scripts.
- Even the script does not include any bug, which is an unrealistic property according to the philosophy of program-bug relation, it could be easily readable and also maintainable if it is written in a good style.
- So, what are the style issues for a script?

Commenting

- As the first style issue, put helpful comments in your scripts to maintain them later in the future without spending any extra time to understand the objectives of them.

Variable naming

- Define variables with meaningful names and put them at the beginning point of their scope, which is generally the top of the script.
- Pay attention to case sensitivity.
- Assure that your script does not include any variable with an identifier name among the reserved words.

Indentation

- Use indentation in the code.
- Whenever you use a conditional or looping command, indent the block of statements that follows, at least one tab stop.

Echoing

- Use the echo command in areas where you suspect of any syntax error to trace the program execution.

Logic correctness

- The programs may contain some logic errors, even a program runs without any syntax error.
- The operators are the possible logic error sources. Some operators have different use in different shells.

Robustness

- To make your program robust test it carefully.
- Check your script for any possible human error, such as bad input, insufficient arguments, nonexistent files, and so on.

Simplicity

- Try to implement the script in a simple way.
- Also when testing your script, keep simplicity. As an illustration, a test on the syntax of the function may be handled via trying it within a short script to check whether the result is similar to the expected one or not.

Command know-how

- Know the commands of the OS.
- Know the commands of the shell.
- If you are new to a command, try it on the command line before using in a script.
- By the way, try to understand the behavior of the command, such as what it returns, what the exit code of its is.
- Also try to understand the OS behavior while issuing the commands, such as how the variables are interpreted in your OS, how to redirect output and errors in your OS.

System administrator responsibility

- If you are an **admin**istrator of a system, test your any script carefully before taking it to the system level. An unexpected error could bring a whole system to its knees.

Types of errors

- Runtime errors
- Logical errors

Runtime errors

- Syntactical errors
  - Mismatched quotes
  - Misspelled errors
- Bad script name,
- Permission issues,
- Path problems,
- ...

Naming conventions

- If you have a script with a name of *ls*, when you try *ls* in the command line which ls will be executed?
  - It depends on which ls is found in the path first.
- **which** command tells you the path where the named program is found.
- So, avoid using the following command names as to be filenames.
  - test
  - script
  - ls
  - ...
- You can also try the script with a leading ./ to run the script in the current directory.

Insufficient permissions

- Check the permissions of the files.
- The scripts need an execute permission to be run.
- $ ls -l myScript
- chmod ...

Path problems

- If you have a superuser account, it is recommended, for security reasons, that you do not include a dot in the search PATH variable.
- There are two alternatives:
    - Give the explicit path:
        - $ ./myScript
    - Precede the name of the script with the name of the shell
        - $ bash myScript
        - # The shell automatically checks the current directory for myScript

The shbang line

- Shbang (#!) line as the first line of a script.
- The path following the shbang notation is the location of the shell that will be invoked to interpret the script.
- If the shbang is put at any top line except the first line of the script, the line will be ignored to be interpreted.

Sneaky alias

- Remember any alias you defined previously.
- You may delete an alias if you do not need it anymore.
    - unalias ...

Two reminders:

- 1. Check page 985 of the textbook for what you need to know about **quotes**.
- 2. Check page 999 for **common bash error messages**.

Tracing the script execution

- By using the -n option to the bash command, you can check the syntax of your scripts without really executing any command.
    - If there is a syntax error in the script, error will be reported, nothing will be displayed otherwise.
    - $ bash -n myScript
        - interprets but does not execute commands
- The two usual debugging methods of any script are as follows:
    - $ bash -x myScript
        - displays each line of myScript after variable substitutions and before execution
    - $ set -x
        - turns on echo

Demos for debugging:

```
#!/bin/bash
total=0
cnt=0
for i in $(cat myNumbers.dat)
do
        let total=$total+$i
        let cnt=$cnt+1
        echo $total
done
echo "cnt:  $cnt"
declare -i avg
avg=$total/$cnt
echo "avg: " $avg
```

- # myScript.sh file contains the code above. Try followings:
- $ bash -n myScript.sh
- $ bash -x myScript.sh
- $ bash myScript.sh

Suggestions according to your HW-LAB answers

- 1- You may use a text editor that provides coloring the reserved words of the shell. (By this way, you may prevent using the conflicting identifiers that are introduced as to be keywords within the editor; you may also see any syntax error while writing your scripts if the editor has the feature to color any errors.)
- 2- Indent your scripts. (By this way, you may at least see the blocks clearly.)
- 3- Test your scripts many times. (By this way, you may catch some unexpected runtime errors.)