SCHOOL OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

UNIX Bash Shell Scripting

Week 2

Agenda

- Unix Shell
- Intro to Shell Scripting
  - I/O commands
  - Variables
  - Shell Arithmatics

What is Shell

- The **shell** is the interface between the command prompt user and the operating system.
  - It interprets the user’s commands and invokes the appropriate systems calls so that the commands are executed.
- Type of shells:
  - The first shell was the **Bourne shell (sh)** by Steven Bourne of AT&T Bell Labs.
  - The **C shell (csh)** was developed for BSD Unix, to offer better programming, but much of the syntax was changed. It has a few ancestor shells, which add additional features (tcsh).
  - The **Korn shell (ksh)**, by David Korn (AT&T Bell Labs) offered better programming like C shell, but followed the Bourne shell syntax. It is often called the K shell.
  - The **BASH shell (bash)** is quite similar to the Korn shell, but was released by the GNU project with an open source license.

Shell Script

- A **shell script** is a computer **program** designed to be run by the Unix **shell**, a command-line interpreter.
  - The various dialects of **shell** scripts are considered to be **scripting** languages. (.bash, .sh, .ksh, …)
- Typical operations performed by **shell** scripts include file manipulation, **program** execution, and printing text.

Shell Scripting

- **Shell scripting** is scripting in ***any* shell**
  - Bash scripting is scripting specifically for Bash.
- In practice, however, "shell script" and "bash script" are often used interchangeably, unless the shell in question is not Bash.

Shell Scripting

- Main building blocks:
  - Variables
  - Output commends
    - echo, printf
  - Input
    - read
  - Control structures
    - Loops, conditional statements,…
  - Sub-programs
    - functions, arguments, return values,…

Shell script : Shebang

- Most Linux shell and Perl / Python script starts with Shebang
  - *#!/bin/bash*
- The #! syntax used in scripts to indicate an interpreter for execution under UNIX / Linux operating systems
  - It is nothing but the absolute path to the Bash interpreter.
  - This ensures that Bash will be used to interpret the script, even if it is executed under another shell
  - Use the which utility to find out path to use: **`which bash`**

echo command

- Displays messages to the terminal followed by a newline
  - Use the **–n** option to suppress the default newline
- Output can be redirected or piped Arguments can be quoted to preserve spaces, double quotes to allow variable substitution or single quotes to disable variable substitution

echo command

- **echo** is a command that outputs the strings it is being passed as arguments
  - `echo hello world`
  - `echo ‘hello world’`
  - `echo “hello world”`
- You may use echo to list the files!
  - `echo *.txt`
    - Show all files in current folder with .txt extension
  - `echo .*`
    - Show all hidden files in current folder

Shell Variables

- Variables can be read/write or read-only
- Name of a variable can be any sequence of letters and numbers, but it must not start with a number

Variables

There are three general categories of variables

– Environment Variables
  - Variables that have been assigned by the Linux OS.
  - This variables are easy to remember and are commonly used.
  - Some of these variables cannot be changed by the user.

– User-Defined Variables
  - Variables set within the shell script to be used within the script.
  - The user can set and change these variables for their own purpose.

– Positional Parameters
  - These variables can be used 2 ways:
    – Assigned **inside** the shell script by the set command
    – Assigned when issuing a shell script with arguments

Example: ./myShellScript.bash arg1 arg2 arg3

Variables

Environment Variables

- Environment variables are used by the shell and many of these variable have values already assigned to them.
- Environment variables are usually identified as UPPERCASE letters.
- The user can see the values assigned to these variables by issuing the set command without an argument.
- Some of these variables can be changed by the user, some are assigned by the system and cannot be changed.

Common (Environment) Shell Variables

- Shell environment variables shape the working environment whenever you are logged in
- Common shell variables include:

– PS1	– primary prompt
– PWD	– present working directory
– HOME	– absolute path to user's home
– PATH	– list of directories where executables are
– HOST	– name of the host
– USER	– name of the user logged in
– SHELL	– current shell

- * use **env** to see the list of all environment variables!

Variables

Environment Variables

- Keyword shell variables can be used in the shell script with Unix commands to “customize” the script for the particular user.

- Examples:
  - **echo “Hi there, $USER”** # Displays current user’s name.
  - **echo $PWD** # Displays user’s current directory.
  - **mkdir $HOME/dir1** # Creates a directory called dir1 that is # contained in user’s home directory.

The PATH variable

- PATH is an environment variable present in Unix/Linux operating systems, listing directories where executable programs are located
- Multiple entries are separated by a colon (:)
- Each user can customize a default PATH
- The shell searches these directories whenever a command is invoked in the sequence listed
- In case of multiple matches use the which utility to determine which match has a precedence
- On some systems the present working directory may not be included in the PATH by default
- Use ./ prefix or modify the PATH as needed

User defined variables

- Syntax: `name=value`
- For example: `course=UNX510`
- If variable values are to contain spaces or tabs they should be surrounded by quotes
- For example: phone="1 800 123-4567“
- User defined variables are global and their lifetime is during the current-user-session.

Variable Substitution

- Whenever you wish to use the value of a variable (its contents), use the variable name preceded by a dollar sign ($)
- This is called variable substitution
  - Example:

```
name=Bob
echo $name
```

Read-Only Variables

- Syntax: `readonly variable = value`
- For example: *readonly phone="123-4567"*
- After a variable is set, it can be protected from changing by using the *readonly* command
- If no variable name is supplied a list of defined read only variables will be displayed

Removing Variables

- For example:
- `course=`
- OR
- `unset course`
- Read-only variables cannot be removed – you must log out for them to be cleared

read commend

- The read command allows obtaining user input and storing it into a variable
  - Everything is captured until the Enter key is pressed
- Example:
- echo –n "What is your name? "
- read name
- echo Hello $name

Creating Shell Scripts

Here is an example of a simple shell script:

Contents of Shell Script

```
cat askAge.bash
# Start of Shell Script
# Prompt user for age and store result in a variable

echo -n "Please enter your age (in years): "
read age

# Print empty line, then print text using value
# of age stored in that variable...
echo
echo "You are $age years old"

# End of Shell Script
```

Execution of Shell Script

```
./askAge.bash
Please enter your age (in years): 44

You are 44 years old
```

Arithmetic expression

- A *Bash* and *Korn* shell built-in command for math is **let**.
  - let z=5
  - let z=$z+1
- With the *BASH* shell, whole arithmetic expressions may be placed inside double parenthesis.
  - ((e=5))
  - ((e = e + 3))

Variables

Positional Parameters

- Positional Parameters have the feature that if your shell script is run with arguments, those arguments can be used as variables within your running shell script!

- This makes your shell script work like a “real” Linux OS command that accepts arguments.

- You can use the set command to assign values to these read-only shell variables inside the script as well..

Variables

Positional Parameters

- Parameter Parameters range from $1 to $9. To access higher numbers (command arguments), you must contain number in braces (eg. ${10}, ${25}, etc…)

- $0 is script name or is shell name if $0 used from shell prompt.

- Positional parameters are assigned values two ways:
  - Using the set command within the shell script. For Example:

```
set one two three
echo "First: $1, Second: $2, Third: $3"
```

  - Using the set command within the shell script. For Example:

```
./myShellScript.bash one two three
# Can use $1, $2, $3 in script…
```

Variables

Positional Parameters

- The shift command is used to move the positional parameters (i.e. arguments) one position to the left.
- As a result, the leftmost positional parameter is lost.
- A number as an argument after the shift command indicates how many positions to the left to shift.

```
Eg.     set one two three
        echo $1 $2 $3
        one two three
        shift
        echo $1 $2 $3
        two three
```

Variables: Special Parameters

Special Parameters

- There are some special symbols that can be used to represent positional parameter information and other useful information such as process ID, exit status, etc…

$#	**number of positional parameters**
$*	**All positional parameters**
$@	**All positional parameters (each contained in**
$?	**Exit Status of previous command**
$$	**Current process ID Number**
$!	**Previous background process ID Number**