Faculty of Computer Science, Dalhousie University *29-Nov-2018*

CSCI 2132 — Software Development

Lab 10: Shell Scripting

Location: Teaching Labs   Instructor: Vlado Keselj
Time: Thursday

Lab 10: Shell Scripting

Lab Overview

– Practice in Shell scripting

If you have not finished the current assignment, you can work on it. If you finished it, you should work on the lab material below.

The objective of this lab is to introduce you to `bash` shell scripting. You should already have some experience throughout the course with the `bash` command line. Many of the common sequences of commands that we do in the `bash` command-line can be automated by writing them in a script file, which we can execute later.

Step 1: Login and Lab Setup. Login to your bluenose account as before.

Note: This lab is not a part of an assignment, so it will not be marked. Still, if you want, you can submit it using SVN.

Go to your checked out copy of your course SVN repository. Based on the previous suggestions, it should probably be in the following directory: `~/csci2132/svn/`*CSID* where *CSID* is your CS user id.

Create and add `lab10` directory to the SVN. You will work in this directory. Commit the change to SVN.

Step 2: Starting with Shell Scripting. Make sure that your current directory is the `lab10` directory in your SVN working copy. In this step you will implement a simple Bash shell script learned in class, or will be learned in class. Using emacs, edit the file `current.sh` and enter the following contents:

```
#!/bin/bash
# Print current status
whoami
pwd
ls
```

Save the file, grant it the user-executable permission (`chmod u+x current.sh`), and execute it using the command:

```
./current.sh
```

If you correctly did the above operations, running the file should print out your userid, your current working directory, and the contents of your directory.

The output of this script is quite terse, and it can be cryptic if you forgot what exactly the output mean. We will address this issue by adding some commands. Edit the file, and change it to the following contents:

```
#!/bin/bash
# Print current status
echo "Your username is:"
whoami
echo "Your current directory is:"
pwd
echo "The contents of your current directory are:"
ls
```

If you save and run the file now, it will produce a more informative output.

SVN submission: Add the file `current.sh` to SVN and commit your changes to SVN.

Step 3: Simple "Add" Script. The Bash shell provides evaluation of arithmetic expressions between double parentheses. Edit the file named `add.sh` with the following contents:

```
#!/bin/bash
(( sum = $1 + $2 ))
echo the sum of $1 and $2 is $sum
```

Save the file, make it user-executable, and try using the following command:

```
./add.sh 5 8
```

The variables `$1` and `$2` are the command-line parameters. The word `sum` is a shell variable, which we can set in a arithmetic expression, but if we need its value we need to prefix it with `$`, as shown above.

If we try to run the above script without any arguments, it will report a syntax error. To prevent this, open the script in an editor and enter the following contents:

```
#!/bin/bash

if (( $# != 2 )); then
    echo usage: $0 num1 num2
    exit
fi

(( sum = $1 + $2 ))
echo the sum of $1 and $2 is $sum
```

Try now to execute it with no arguments:

```
./add.sh
```

as well as with two numerical arguments. We can learn here a couple other Bash scripting features:

- `$#` denotes the number of arguments
- we used the 'if' command in the script (described in more detail in the class)
- `$0` is the name of the script as called in the command line
- and we can use the command `exit` to exit the script

You can read now more about the arithmetic evaluation in `bash` using the following commands:

```
man bash
```

This will open the “man bash” page. Type
`/arithmetic`
and Enter. This will find the first occurrence of the word “arithmetic” in the page, which is not the one you are looking for. You can search for next occurrences by typing:
`n`
several times, until you get to a place that starts as follows:

```
ARITHMETIC EVALUATION
       The shell allows arithmetic expressions to be evaluated, under
       certain circumstances (see the let  and  declare builtin
       commands and Arithmetic Expansion).  Evaluation is done in
       fixed-width integers with no check for overflow, though
       division by 0 is trapped and flagged as an error. The operators
       and their precedence, associativity,  and  values are the same
       as in the C language.  The following list of operators is
       grouped into levels of equal-precedence operators.  The levels
       are listed in order of decreasing precedence.
...
```

By searching through the same page you can read about many features of `bash` scripting. You can exit the browsing of the “man” page by typing:
`q`

SVN submission: Add the file `add.sh` to SVN and commit your changes to SVN.

Step 4: Using for-Loop. The `bash` shell allows several forms of the for loops. One, with double parentheses, is arithmetic, and very similar to the C for-loop. Edit the file `gen-files.sh` and enter the following contents:

```
#!/bin/bash

if (( $# != 1 )); then
    echo usage: $0 num1
    exit
fi

for (( i = 1; $i <= $1; i = $i + 1 )) do
  f=tmpfile-$i.txt
  echo "Appending file $f"
  echo Updated on `date` >> $f
done
```

Save the script, make it executable, and run it using the command:

```
./gen-files.sh 10
```

You can notice that several new files are created: `tmpfile-1.txt`, `tmpfile-2.txt`, ..., and `tmpfile-10.txt`.

You can notice the use of backquotes for command substitution. The string `` `date` `` is replaced by the output of running the command `date`. The command substitution works also within strings delimited with double quotes.

SVN submission: Add the file `gen-files.sh` to SVN and commit your changes to SVN.

Step 5: Another form of for-Loop. We saw one, arithmetic format of the for loop. Actually, the ‘for’ loop is more commonly used in an another format in a `bash` shell, which is less similar to the ‘for’ loop in C or Java.

Since we generated in the previous step a set of `.txt` files, let us now write a script named `report-lines.sh` to report the number of lines in all `.txt` files in the current directory:

```
#!/bin/bash

for file in *.txt
do
    lines=`wc -l $file| cut -d" " -f1`
    echo "The file $file contains $lines lines."
done
```

- Save it, make executable, and run it

It is important not to have any space around the equal sign (=) when assigning a value to the variable `lines`.

SVN submission: Add the file `report-lines.sh` to SVN and commit your changes to SVN.

Step 6: Using the case-Statement. Let us now write a shell script for one of the textbook exercises (Glass and Ables, Ex. 5.2, page 209). It is a utility named `junk`, which is a safer alternative to the `rm` utility. Rather than removing files, it moves the into the subdirectory `.junk` of the home directory. If the `.junk` subdirectory does not exist, it is automatically created. If the option `-l` is used with the script, it prints the contents of the `.junk` directory; and if the option `-p` is used, it purges the `.junk` subdirectory.

You can implement the script as the following example. Write it and test it.

```
#!/bin/bash
case $1 in
  -l)
    ls ~/.junk
    ;;
  -p)
    rm ~/.junk/*
    ;;
  *)
    if [ ! -d ~/.junk ]; then
      mkdir ~/.junk
    fi
    for file in $@
    do
      mv $file  ~/.junk/$file
    done
esac
```

We can see here the syntax of the case statement, as well as another form of the for loop. We can also see how the square brackets are used for a conditional expression:

```
[ ! -d ~/.junk ]
```

where we test whether the directory `~/.junk` exists. In an expression like this, with square brackets, it is important to have at least one space after the opening bracket (`[`) and at least one before the closing bracket (`]`).

The variable `$@` contains the list of all command-line arguments.

SVN submission: Add the file `junk` to SVN and commit your changes to SVN.

Step 7: Save Example. Your task in this step is to write a script named `save.sh`, which can be used to save the current version of a file or multiple files. If you are working on a file, it can be useful to save a snapshot of the current file for later reference. The script will save the files in a directory named `saved.d` and it will assign a timestamp to their names. If the directory `saved.d` does not exist, the script needs to create it. For example, if a file named `f` is saved at 9h 30min 0sec, on Nov 26, 2013, it would be copied to the file: `saved.d/f-2013-11-26-093000`
You should first try to solve this problem by yourself, and if you want some hints, you can take a look at the sample solution provided further.

Sample Solution

– Edit the file `save.sh` by inserting:

```
#!/bin/bash

if [ ! -d saved.d ]; then
  mkdir saved.d
fi

for file in $@
do
  cp $file saved.d/$file-`date +%Y-%m-%d-%H%M%S`
done
```

– Try the script by saving all `.txt` files:

```
./save.sh *.txt
```

– Check the contents of the directory `saved.d`

SVN submission: Add the file `save.sh` to SVN and commit your changes to SVN.

Step 8: End of Lab. By now, you have finished the required work of this lab.