CSCI 2132
Software Development

Lecture 33:

Shell Scripting

Instructor: Vlado Keselj

Faculty of Computer Science

Dalhousie University

Previous Lecture

- File Manipulation in C:
- Opening a file, closing a file
- Formatted I/O, character I/O
- Block reading and writing
- File positioning
- Example with file writing

Shell Scripting

- Unix shells provide programming-language-like features
- Referred to as *shell programming* or *scripting*
- Useful for system administration
- No overhead in terms of compilation
- Very close to the use of command line
- Close to the Unix philosophy of breaking projects into sub-tasks
- Reading: [Glass and Ables] Chapter 8: bash

Shell Program Example

- Using emacs create a file named `current.sh` with the following contents:

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

- Save it, make user-executable, and execute using:

```
./current.sh
```

Variables

- Similar to the shell variables in the command line
- Example:

```
i=1
echo $i
```

- Some special variables:
  - `$0` is the pathname of the script
  - `$n` is the n-th command arguments. We can use `$1`, `$2`, `...`, `$9`, `${10}`, `${11}`, `...`
  - `$#`: the number of command-line arguments, excluding `$0`

Arithmetic Operations

- To use arithmetic expressions, use double parentheses:

```
(( expressions ))
```

- Arithmetic operators: `=`, `+`, `-`, `++`, `--`, `*`, `/`, `%`, and `**`.
- `**` is exponentiation
- Example:

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

Conditional Expressions

- The syntax for arithmetic tests:

```
(( expressions ))
```

- Operators: `<=`, `>=`, `<`, `>`, `==`, `!=`, `!`, `&&`, and `||`
- The sytax for string tests:

```
[ expression ]
```

- Note: Spaces after `[` and before `]` are mandatory
- Operators: `==` and `!=`
- Additional operators: `-n` *string*, and `-z` *string* (nonzero and zero length)

Control Structures

- ‘If’ statement: similar to C, but different syntax:

```
if condition1; then
  commands
elif condition2; then
  commands
else
  commands
fi
```

- The `elif` and `else` parts are optional

Example with ‘If’ Statement

- Example:

```
#!/bin/bash

if (( $# != 2 )); then
  echo usage: ./add.sh num1 num2
  exit
fi

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

Example with Arithmetic for-Loop

```
#!/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
```

The Standard Bash for-Statement

- This use of for-loop is also a loop statement, but quite different syntax than C or Java:

```
for var in word {word}*
do
  commands
done
```

For-loop Examples

- Example:

```
#!/bin/bash
for file in *.txt
do
  sort $file > $file.sorted
done
```

- Another way:

```
#!/bin/bash
for file in *.txt; do
  sort $file > $file.sorted
done
```

- or

```
#!/bin/bash
for file in *.txt; do sort $file > $file.sorted; done
```

Alternative Solution

- Use *command substitution* (`` `command` ``)
- Backquotes can be used to replace a command result in another command; example:

```
echo There are `ls | wc -l` files in the current\
 directory
```

- Alternative solution to previous task:

```
#!/bin/bash
for file in `ls *.txt`
do
  sort $file > $file.sorted
done
```

Case Statement

- Similar to the switch statement in C or Java; syntax:

```
case var in
  word{|word}*)
    commands
    ;;
  ...
esac
```