Lecture A for foreign students

Lecture 3: Shell scripts

Norbert Pozar (npozar@se.kanazawa-u.ac.jp)

May 17, 2018

Today's plan

- Shell scripts for automation

Shell scripts for automation

Shell script

- File containing a sequence of shell commands that can be run automatically.

Example. Create a file `hello.sh` containing

```
#!/bin/bash

# print something
echo "Hello, $1!"
```

```
$ bash ./hello.sh name
```

Or make it executable directly

```
$ chmod +x hello.sh
$ ./hello.sh name
```

Pro and cons of shell scripts

Pros

- Works on any Unixy system without setup, directly on the command line.
- Simple and quite powerful when using with other system commands.

Cons

- Limited. For more general scripts use Python or similar.
- For building large projects, use `make`. See my “Quick Intro to make”:

http://polaris.s.kanazawa-u.ac.jp/~npozar/intro-to-make.html

Example: Build script for simple projects

- Good for making sure that a program is build correctly.

Create a file `build`[1]:

```
#!/bin/bash
gcc optimization2.c -o optimization2 \
    -O3 -march=native
```

and make it executable

```
$ chmod +x build
```

Building and running is as easy as

```
$ ./build && ./optimization2
```

[1] Scripts do not need `.sh` extension.

Variables

- Variables can contain arbitrary strings.

```
s=Hello
echo $s
echo ${s}
```

- Supports string operations:

```
echo ${s%llo}y
```

  - Strips the trailing `llo` from the value in `$s`.

See more at Manipulating strings.

`s=$(seq 10)` Run `seq 10` command, collect its output and store it in variable `s`.

Special variables

`$0` Name of the current script it was invoked with.

`$1, ..., $9` Command line arguments that the script was called with.

`$?` The exit code of the last executed command.

if condition

```
#!/bin/bash
if [ $1 -gt 10 ]
then
    echo Number $1 is bigger than 10!
    if (( $1 % 2 != 0 ))
    then
        echo It appears to be odd?!
    fi
else
    echo $1 is way too small...
fi
```

Functions

```
#!/bin/bash

# function that computes the length of
# the input string
function len {
    echo -n -e "$1" | wc -c | tr -d '[:space:]'
}

# call it as a command
l=$(len Hello)

# careful with spaces
input="my string"
l=$(len "$input")
echo "I got: $l = ${#input}"
```

`for` loops

- Run code with different inputs

```
for i in {1..10}
do
    echo $i
done
```

Debugging tip

`set -x` Prints every command executed.

`for` loop over strings

- List files in the current directory and their basenames:

```
for f in *
do
    echo "$f"
    echo "Basename:" ${f%.*}
done
```

Exercise: Build all `.cpp` files in the directory

Using a `for` loop, run `g++` for each `.cpp` file in the current directory and produce a binary with the same name as the file without the extension.

Solution: Build all `.cpp` files in the directory

```
#!/bin/bash

set -e      # stop on error

for f in *.cpp
do
    g++ "$f" -o "${f%.*}"
done
```

Exercise

Write a script that changes the extension of all `.dat` files in the current directory to `.txt`.

`mv <source> <dest>` move a file `<source>` to `<dest>`

Exercise: Script that removes compiled binary files

Write a script that automatically removes all compiled binary files from the current directory[2].

Useful commands:

`file <file>` print information about the given file
`grep -q <needle>` test if input from stdin contains `<needle>`
`rm <file>` removes the given file

[2] This is useful if you want to clean up the directory and keep only the source files.

Parallel execution of commands

Adding & after the command executes it in the background. (`parallel.sh`)

```
for i in {1..10}
do
      echo "Working on $i ..."\
          && sleep $i\
          && echo " done with $i" &
done

# wait for all the background tasks to finish
wait
```

Really simple parallelization of your code for **free**!

Exercise

1. Write a C program `len.c` that takes one command line argument and prints the number of characters in it.

```
$ len test
4
```

2. Write a bash script `namelengths.sh` that uses `len.c` to print the length of each filename in the current directory.

```
$ ./namelengths.sh
12 filename.txt
4 test
```

3. Use `sort -n` to sort the list of files according to the filename length.

Learn more

- Shotts, The Linux Command Line

http://linuxcommand.org/tlcl.php

- Bash Scripting Tutorial for Beginners

https://linuxconfig.org/bash-scripting-tutorial-for-beginners

- Shell Scripting Tutorial

https://www.shellscript.sh/