

CSc 352

Shell Scripting

Benjamin Dicken

Shell Scripts

Many cases where it is useful to string together multiple bash commands to complete a task

This: NBA and Currency examples from last class!

Can write **bash scripts**.

Programs written in bash

Shell Scripts

- Shell scripts typically use the .sh extension, but ultimately, as with many file types, they're just text file behind-the-scenes
- The first line should always be of the form:

```
#! /path/to/interpreter
```

More specifically:

```
#! /bin/bash
```

See: <https://stackoverflow.com/questions/8967902/> and <https://linux.die.net/man/2/execve>

Shell Scripts

```
1 #! /bin/bash
2
3 wget https://www.nba.com/suns/roster 2> /dev/null
4
5 cat roster | sed 's/[{}]/\n/g' > roster2
6
7 echo "Suns player names sorted:"
8
9 sed -n -E 's/Person", "name": "([A-Za-z ]+)/\1/p' roster2 | cut -d '"' -f 4
```

Shell Scripting Variables

Shell scripts support variables

```
name="Ben"  
occupation=Lecturer  
echo "${name} is a ${occupation}"
```

By default, the “type” of all variables are basically just strings

- There are attributes, but for now just expect that every variables is just a string
- <https://stackoverflow.com/questions/29840525>

Command Line Arguments

Special variables for the command line arguments:

```
#!/bin/bash
```

```
echo "Your name is: ${1}"
```

```
echo "Your occupation is: ${2}"
```

```
echo "The command line arguments: ${@}"
```

Modify the script

How could this script be modified to allow the user to specify the team to get the roster for as a command line argument?

```
1 #! /bin/bash
2
3 wget https://www.nba.com/suns/roster 2> /dev/null
4
5 cat roster | sed 's/[{}]/\n/g' > roster2
6
7 echo "Suns player names sorted:"
8
9 sed -n -E 's/Person", "name": "([A-Za-z ]+)/\1/p' roster2 | cut -d '"' -f 4
```

Command Substitution

Storing the standard out that a command produces in a variable is useful when scripting with bash

Use **command substitution** with `$(command)`

```
temp_files=$(ls /tmp/)
```

```
username=$(whoami)
```

```
search_results=$(cat roster.txt | grep [A-Z])
```


Loops

Can loop through a sequence of tokens with a for loop

```
for VARIABLE in X Y Z;  
do  
    echo ${VARIABLE}  
done
```

Conditions

- Use `-eq`, `-gt`, `-lt` for numbers
- Use `==`, `!=`, etc for non numeric strings

```
if test "${num1}" -eq "${num2}"
then
    echo "equal"
else
    echo "unequal"
fi
```

Rewrite SBT as a shell script (simplified)

Re-implement the SBT script as a shell script

The script should:

- Run make to build a program
- Iterate through test directories
- Check if output matches expected
- Run make clean at the end

```
#!/bin/bash
```

```
code_dir=${1}
```

```
test_dir=${2}
```

```
pushd ${code_dir}
```

```
make > /dev/null
```

```
popd
```

```
test_dir_names=$(ls ${test_dir})
```

```
for directory in ${test_dir_names};
```

```
do
```

```
    echo "Testing the directory ${directory}"
```

```
    cat ${test_dir}/${directory}/input.txt | ${code_dir}/a.out > /tmp/actual.txt
```

```
    diff ${test_dir}/${directory}/output.txt /tmp/actual.txt
```

```
done
```

Improve SBT

Have the program say “Text case X passed” or “text case X failed” depending on the results of the call to the `diff` command.

Use a bash if-statement

```
/tmp/sbtsh
```

```
#!/bin/bash
```

```
code_dir=${1}
```

```
test_dir=${2}
```

```
pushd ${code_dir}
```

```
make > /dev/null
```

```
popd
```

```
test_dir_names=$(ls ${test_dir})
```

```
for directory in ${test_dir_names};
```

```
do
```

```
    echo "Testing the directory ${directory}"
```

```
    cat ${test_dir}/${directory}/input.txt | ${code_dir}/a.out > /tmp/actual.txt
```

```
    result=$(diff ${test_dir}/${directory}/output.txt /tmp/actual.txt)
```

```
    if test "${result}" == ""
```

```
    then
```

```
        echo "passed test case for ${test_dir}"
```

```
    else
```

```
        echo "failed test case for ${test_dir}"
```

```
    fi
```

```
done
```

Kill Processes

- Write a bash script that accepts one command-line argument
- Script should search for all processes that match the argument, and kill each one
- Kill by PID using the **kill** command

```
$ ./kill_processes.sh hithere
```

```
2 Processes Killed
```

```
$
```

Do not use **killall**

```
#!/bin/bash
```

```
to_kill=${1}
```

```
ps_results=$(ps -e | grep ${to_kill})
```

```
pids=$(printf "${ps_results}" | sed -n -E 's/^[ ]*(.+)/\1/p' | cut -d ' ' -f 2)
```

```
counter=0
```

```
for pid in ${pids} ;
```

```
do
```

```
    kill -9 ${pid}
```

```
    counter=$((counter+1))
```

```
done
```

```
echo "${counter} processes were killed"
```