csce215 — UNIX/Linux Fundamentals
Spring 2022 — Lecture Notes: Automating More Stuff

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with some supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

(9.1) *Last time*

Last time, we pulled together many of the ideas from throughout the semester to see how to create **shell scripts**.

- What are shell scripts?
- How can shell scripts be created?
- How can shell scripts be executed?
- Shell variables and environment variables

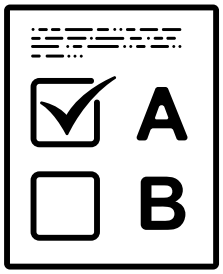
Today, we'll wrap things up with a few final shell features that are particularly useful for shell scripts.

- Shell options.
- Conditionals.
- Loops.

(9.2) *Final exam*

As a reminder, the **final exam**, covering concepts marked with , will be **in person** in **Amoco Hall** on **April 30** at **4:00pm**.

Sample questions are in the lecture notes.



(9.3) *Shell options*

There are several **shell options** that change the behavior of the shell for the rest of the session.

`set`
Enable shell options.
`-e` stop the script when an error occurs
`-v` print each command as given
`-x` print each command after expansion

Example: Using `-v` and `-x` to see what is being executed.

```
$ cat $(which cvlc)
#! /bin/sh
exec /usr/bin/vlc -I "dummy" "$@"
$ set -v
$ cat $(which cvlc)
cat $(which cvlc)
#! /bin/sh
exec /usr/bin/vlc -I "dummy" "$@"
$ set -x
set -x
$ cat $(which cvlc)
cat $(which cvlc)
++ which cvlc
+ cat /usr/bin/cvlc
#! /bin/sh
exec /usr/bin/vlc -I "dummy" "$@"
```

(9.4) *Conditionals*

We can execute code *conditionally*.

`if ... then ... fi` 
Execute commands if a condition is met.

The simplest form of if statement uses a normal command as the test, and check its return code. 

- Recall the details about return codes from Chapter 4.
- In general: Running without an error is considered 'true'.

In a directory with these files:

```
$ ls
a.txt
b.txt
cond1
cond2
cond3
cond4
cond5
cond6
loop1
loop2
loop3
```

True / success / zero return code:

cond1

```
#!/bin/bash

if ls
then
  echo Yes
else
  echo No
fi
```

```
$ ./cond1
a.txt
b.txt
cond1
cond2
cond3
cond4
cond5
cond6
loop1
loop2
loop3
Yes
```

False / failure / non-zero return code:

cond2

```
#!/bin/bash

if ls *.py
then
  echo Yes
else
  echo No
fi
```

```
$ ./cond2
ls: cannot access '*.py': No such file or directory
No
```

(9.5) *Primary expressions*

Instead of a command, we can use a **primary expression**, marked with square brackets `[]` and spaces, as the condition of an if statement.

- File exists? `-a`

cond3

```
#!/bin/bash

file="$1"

if [ -a "$file" ]
then
  echo $file exists
else
  echo $file does not exist
fi
```

```
$ ./cond3 a.txt
a.txt exists
```

```
$ ./cond3 c.txt
c.txt does not exist
```

- One file is newer than another? `-nt` 

cond4

```
#!/bin/bash

file1="$1"
file2="$2"

if [ "$file1" -nt "$file2" ]
then
  echo $file1 is newer
else
  echo $file2 is newer
fi
```

```
$ ./cond4 a.txt b.txt
b.txt is newer
```

- Strings are equal? ==

cond5

```
#!/bin/bash

opt="$1"

if [ "$opt" == "-v" ]
then
  echo Got -v
else
  echo Nope
fi
```

```
$ ./cond5
Nope
```

```
$ ./cond5 -v
Got -v
```

- String has zero length? `-z`

cond6

```
#!/bin/bash

if [ -z "$1" ]
then
  echo Argument is missing.
  exit
fi

echo Ready!
```

```
$ ./cond6 a.txt
Ready!
```

```
$ ./cond6
Argument is missing.
```

There are many other primary expressions, including boolean operators (and, or, not) for combining expressions.[1]

(9.6) *Loops*

We can use **for loops** to repeat things.

[1] https://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html

`for ... in ... do ... done`

Repeat commands for each element in a list

A `for` command needs:

- A variable
- A list of values that variable should take
- A commands to execute for each value

Example:

loop1

```
#!/bin/bash

for i in a b c
do
  echo $i
done
```

```
$ ./loop1
a
b
c
```

How to generate the list of values to iterate over? You already know at least three ways!

- List values explicitly. 

  (See `loop1` example above.)

- Use wildcards. 

loop2

```
#!/bin/bash

for i in *.txt
do
  echo File $i has $(cat $i | wc -l) lines.
done
```

```
$ ./loop2
File a.txt has 1 lines.
File b.txt has 2 lines.
```

- Use command substitution.

loop3

```
#!/bin/bash

for pid in $(ps -o pid=)
do
  cat /proc/$pid/status | grep -E 'Name|State'
  echo
done
```

```
$ ./loop3
Name:   bash
State:  S (sleeping)

Name:   evince
State:  S (sleeping)

Name:   evince
State:  S (sleeping)

Name:   python3
State:  S (sleeping)

Name:   bash
State:  S (sleeping)

Name:   bash
State:  S (sleeping)

Name:   loop3
State:  S (sleeping)

cat: /proc/1785832/status: No such file or directory
```

(9.7) *When to make a shell script?*

Shell scripts can be very helpful to save time and avoid repeating mistakes.

- There are enough features in bash to think of it like a programming language. Some we've seen (variables, conditionals, loops) and others we've left for you to explore on your own (functions, arrays, arithmetic, etc).
- There's no limit to the complexity of problems that shell scripts can solve.

But...

- The syntax can be painful.
- Data structures are limited.
- Debugging can be tricky.

My conclusion: When your script gets more complex than a handful of conditionals or a loop or two, it's time to 'upgrade' to a more complete language like Python or Perl.

(9.8) *Sample final exam questions*

1. The purpose of the `-x` shell option, enabled with `set -x`, is to _____.

   A. print each command as given
   B. stop the script when an error occurs
   C. print each command after expansion
   D. prevent overwriting of files by redirection

2. The conditional

```
if [ "$a" == b ]; then
```

tests whether _____.

   A. Two processes are running the same program.
   B. Two files have the same modification date.
   C. Two files have the same contents.
   D. Two strings are equal.

3. The conditional

```
if [ -a "$file" ]; then
```

tests whether _____.

   A. A file is a hidden file.
   B. A file has been accessed.
   C. A file is readable by all users.
   D. A file exists.

4. The conditional

```
if [ "$file1" -nt "$file2" ]; then
```

tests whether _____.

   A. One file is neater than another.
   B. One file depends on another.
   C. One file is newer than another.
   D. One file is larger than another.

5. The purpose of the `-v` shell option, enabled with `set -v`, is to _____.

   A. print each command after expansion
   B. stop the script when an error occurs
   C. print each command as given
   D. prevent overwriting of files by redirection

6. The purpose of the `-e` shell option, enabled with `set -e`, is to _____.

   A. print each command after expansion
   B. print each command as given
   C. prevent overwriting of files by redirection
   D. stop the script when an error occurs

7. In a directory containing the files `a.txt`, `b.txt`, `c.java`, and `d.cpp`, how many iterations will the loop

```
for f in *.txt; do
```

execute?

A. 4

B. 2

C. 3

D. 1

8. In a directory containing the files `a.txt`, `b.txt`, `c.java`, and `d.cpp`, how many iterations will the loop

```
for f in $(find -name *java); do
```

execute?

A. 1

B. 2

C. 4

D. 3

9. The purpose of the `set` command is to _____.

A. enable shell options

B. assign values to shell variables

C. assign values to environment variables

D. set permissions on files

10. The conditional

```
if command; then
```

tests whether _____.

A. The *command* runs successfully, with a zero return code.

B. The *command* is the name of the currently-running script.

C. The *command* is running in the background.

D. The *command* exists.

11. The conditional

```
if [ -z "$a" ]; then
```

tests whether _____.

A. A jobs list is empty.

B. A file is empty.

C. A string is empty.

D. A process list is empty.