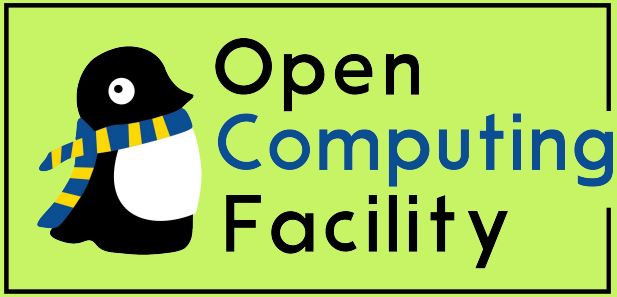


Lecture 2

Shell Scripting

Joe Wang

(slides courtesy of joe, aly, laksith, trinityc)

Logistics

- **Lab 02** released, due Saturday, 2/15.
- Lab 1 should now be graded!
- Lab will all be posted at decal.ocf.io per usual.
- Attendance: required

Course Resources

- Your facilitators!
- Ed, Gradescope
- OCF Slack (ocf.io/slack) or Discord (ocf.io/discord) #decal-general
- All materials available at decal.ocf.io
- Ask questions / work on lab with us during live sessions! (8-9 p.m. after live lecture)

Engaging with this lecture

- Connect to the shell (follow along!)
  - `ssh $OCF_USERNAME@ssh.ocf.berkeley.edu`
  - Open a (unix) shell in your terminal locally
- Ask questions!
  - During live sessions
  - On #decal-general

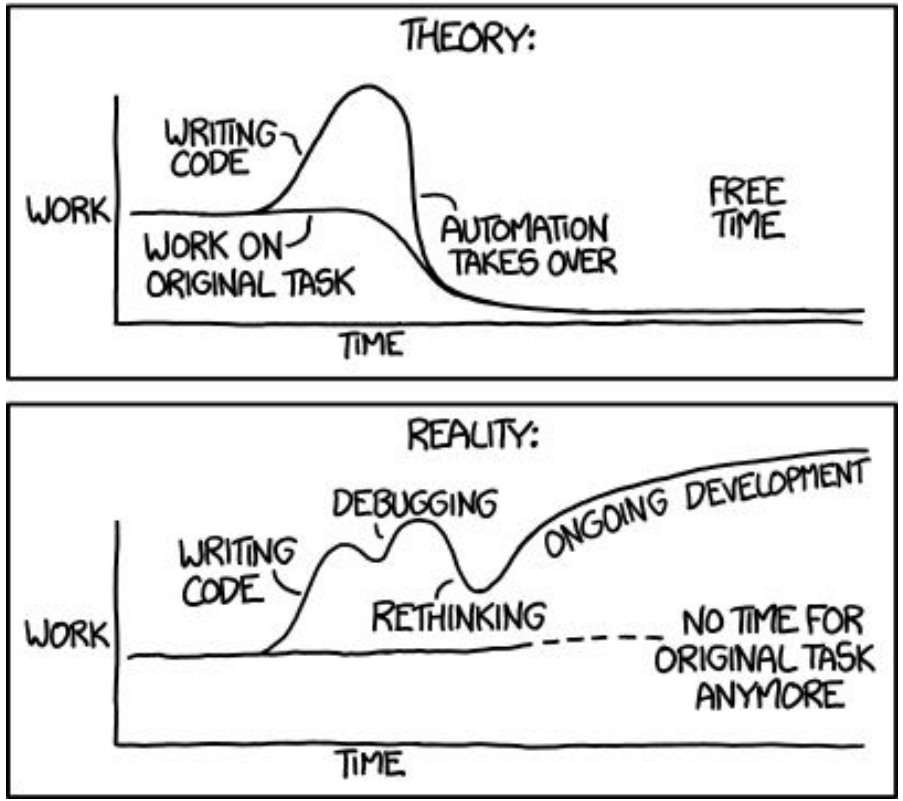
Topics

1. Bash
2. Variables
3. Conditionals
4. Loops
5. Functions
6. Streams

But WHY should I learn to script?

- You’re a sysadmin
- You have to run some commands all the time
- But you want to ~~be lazy~~ DRY (Don’t Repeat Yourself)
- Describe your task as a step-by-step set of instructions so that a computer can do it for you!

"I SPEND A LOT OF TIME ON THIS TASK.
I SHOULD WRITE A PROGRAM AUTOMATING IT!"



Topics

1. **Bash**
2. Variables
3. Conditionals
4. Loops
5. Functions
6. Streams

Bash: 1989

A shell...

Expanded version of `sh` (also a shell)

And also a programming language!

Bash

To run bash scripts:

- `bash /path/to/script` (usually `.sh` file)
- OR:
  - `chmod+x /path/to/script`, `/path/to/script` (requires shebang)
  - `./path/to/script`
  - (see demo)

Shebang

Special comment, specifies that the file is a script and calls a certain interpreter (i.e., bash, sh, python)

```
#!/bin/bash

#!/bin/sh

#!/usr/bin/env python

#!/usr/bin/python
```

Comments

Use a pound/sharp/hashtag **without a !** to write comments

```
# This is a comment
```

Topics

1. Bash
2. **Variables**
3. Conditionals
4. Loops
5. Functions
6. Streams

shell variables

- Whitespace matters!
- Use $VAR to output the value of variable VAR
- Display text with echo

```
NAME="value"
echo "$NAME"
```

shell variables

- Types? What types?
- Bash variables are untyped
- Operations are contextual

```
FOO=1
$FOO + 1
```

shell variables

- Types? What types?
- Bash variables are untyped
- Operations are contextual
- Everything is string

```
FOO=1
$FOO + 1
error!
```

shell variables

- Use the expr command to evaluate expressions
- Part of coreutils

```
FOO=1
expr $FOO + 1
2
```

User input

- Use the read command get user input
- “-p” is for the optional prompt

```
read -p “send: ” FOO
# type “hi” and enter

echo “sent: $FOO”
sent: hi
```

subshell

- $(**cmd**) evaluates the command **cmd** inside, and substitutes the output into the script.

```
FOO=$(expr 1 + 1)
echo "$FOO"
2
```

Topics

1. Bash
2. Variables
3. **Conditionals**
4. Loops
5. Functions
6. Streams

test

conditional checks

- Evaluates an expression

  Also synonymous with []
- Sets exit status to
  - 0 (true)
  - 1 (false)

(Yup you read that right)

test

conditional checks

(result stored in $?)

```
test zero = zero; echo $?
0 # 0 means true
test zero = one; echo $?
1 # 1 means false
```

test

conditional checks

```
-eq ==
-ne !=
-gt >
-ge >=
-lt <
-le <=
```

“boolean” ops

*careful with space!

&& and || for shell

```
[ 0 -lt 1 ] && [ 0 -gt 1 ];
echo $?
1

[ 0 -lt 1 || 0 -gt 1 ]; echo
$?
0
```

if

What if...?

```
if [ “$1” -eq 79 ];
then
   echo “nice”
fi
```

if-else

...And what ifn't

```
if [ “$1” -eq 79 ];
then
   echo “nice”
else
   echo “darn”
fi
```

elif

...And what ifn't but if

```
if [ "$1" -eq 79 ];
then
   echo "nice"
elif [ "$1" -eq 42 ];
then
   echo "the answer!"
else
   echo "wat r numbers"
fi
```

case

No one likes long if statements...

```
read -p "are you 21?" ANSWER
case "$ANSWER" in
  “yes”)
    echo "i give u cookie";;
  “no”)
    echo "thats illegal";;
  “are you?”)
    echo “lets not”;;
  *)
    echo "please answer"
esac
```

Topics

1. Bash
2. Variables
3. Conditionals
4. **Loops**
5. Functions
6. Streams

for loops

for all your stuff in stuffs

```
SHEEP=("one" "dos" "tre")
for S in $SHEEP
do
   echo "$S sheep..."
done
```

for loops

supports ranges too

```
n=0
for x in {1..10}
do
   n=$(expr $x + $n)
done
echo $n
```

while loops

nightmare nightmare
nightmare nightmare
nightmare nightmare
nightmare nightmare
nightmare nightmare
nightmare nightmare
nightmare nightmare
nightmare nightmare

```
while true
do
   echo "nightmare "
done
```

Time to Exercise!

Let’s write a script that copies files in our current directory into new files with “new” prepended to the contents (hint: cp file newfile)

```
> ls
a.txt b.txt c.txt
> ./mycoolscript.sh
> ls
a.txt b.txt c.txt
new_a.txt new_b.txt
new_c.txt
```

Time to Exercise!

Let's write a script that copies files in our current directory

```
#! /bin/sh

FILES=$(ls *)
for FILE in $FILES
do
   cp $FILE new_$FILE
done
```

Topics

1. Bash
2. Variables
3. Conditionals
4. Loops
5. **Functions**
6. Streams

functions

fun

```
function greet() {
    echo "hey there $1"
}
greet “sysadmin decal”

hey there sysadmin decal
```

functions

script args are stored the same way as with functions

```
# in terminal
ls .
b.txt a.txt c.txt

# script.sh
ls $1 | sort

# in terminal
./script.sh .
a.txt b.txt
c.txt
```

Topics

1. Bash
2. Variables
3. Conditionals
4. Loops
5. Functions
6. **Streams**

Redirection

Use > to output to somewhere else, like a text file!

```
echo "hello" > out.txt
```

Append

Use >> to append output to a file. If file is empty, works the same as >

```
echo "hello" >>
out.txt
```

Redirection

Use < to take input from a file!

```
sort < file
```

Pipes

Take output of first command and “pipe” it into the second one, connecting stdin and stdout

```
command1 | command2
```

Additional Notes

- Python
  - **`argparse`**: easy CLI
  - **`fabric`**: easy deployment
  - **`salt`**: generally useful for infrastructure-related tasks
  - **`psutil`**: monitor system info
- Use **`bash`** when the functionality you want is easily expressed as a composition of command line tools
  - Common file manipulation operations
- Use **Python** when you need “heavy lifting” with complex control structures, messy state, recursion, OOP, etc.

Other Shells

- You might've heard of **zsh**, **fish**, **ksh**.
  - These shells are alternative derivatives of **sh**, but may have some differences in syntax.
  - For example, subshells in **fish** don't use the $(**cmd**) notation.
- For our purposes (labs, etc) we will be using **bash** specifically.

Other Resources

- AT&T Archives: The UNIX Operating System
- Knuth and McIlroy Word Count
- Linux Documentation Project: Bash Guide for Beginners
- Honestly, Google is your best friend
- `man bash`

Open
Computing
Facility

OCF logo assets

Copy-paste these!!!
DO NOT MODIFY

Individually cropped sticker assets

Copy-paste these!!!
DO NOT MODIFY