Lecture 5: Scripting with Bash

CSE 374: Intermediate Programming Concepts and Tools

Administrivia

- Find partners on discord!

- Thank you for all your #feedback !
  - Self goal to post slide pre lecture
  - Poll everywhere is still being gd annoying
  - Having issues connecting to klaatu from outside us, download VM: https://www.cs.washington.edu/lab/software/linuxhomevm
  - Gradescope auto-grading shenanigans – please pay attention to the hints for formatting needs

- Homework 1 finally live
  - Calendar with deadlines

Finish redirection

- `cmd > file` sends stdout to file
- `cmd 2> file` sends stderr to file
- `cmd 1> output.txt 2> error.txt` redirects both stdout and stderr to files
- `cmd < file` accepts input from file
  - Instead of directly putting arg in command, pass args in from given file
  - `cat file1.txt file2.txt file3.txt` or `cat < fileList.txt`
- What is the difference between | and >?
  - Pipe is used to pass output to another program or utility
  - Redirect is used to pass output to either a file or stream
  - thing1 > thing2 runs thing1 and then sends the stdout stream to thing2, if these are files thing2 will be overwritten
  - thing1 > tempFile && thing2 <tempFile sends stdout of thing1 to stdin of thing2 without overwriting files
    - Equivalent to thing1 | thing2 much more elegant!

https://askubuntu.com/questions/172982/what-is-the-difference-between-redirection-and-pipe

Transferring files between local and remote

- tar – tape archive – compresses directory of files for easy transfer (like zip or archive)
  - `tar -c <directory to compress>`
    - `tar -c -v -f myTarFile.tar /home/champk/`
    - `-c` – creates new .tar archive file
    - `-v` – Verbosely show the tar process
    - `-f` - to decide name of tar file
  - `tar -x <file to extract>`
    - `tar -x -v myTarFile.tar`
- wget – non-interactive download of files from the web supporting http, https and FTP
  - Non interactive means it can work in the background (helpful if the files take a while)
  - `wget http://website.come/files/file.zip`
- Scp – secure copy – uses ssh protocol to transfer files between different hosts
  - `scp user@remote.host:file.txt /local/directory` copies file.txt from remote host to local directory
  - `scp file.txt user@remote.host:/remote/directory/ copies file.txt` from local host to remote directory
- You can always use a file transfer GUI like FileZilla uses FTP or SFTP, available for all platforms

Writing Scripts

- Instead of writing commands directly into terminal save them in a file
  - Use file extension ".sh"
- Bash can run these files as executables
  - Add line at top of file to tell computer this should be run using bash

```
#! /bin/sh
```

- # by itself makes a comment
  - Always include header comment with usage instructions
- Give the file execution permissions

```
chmod u+x myscript.sh
```

- Stop bash script on first failure by adding set –e at top of script
- Bash scripts are especially helpful

Demo of making script

Bash Script Variables

- When writing scripts you can use the following default variables

$# – stores number of parameters entered

Ex: `if [$# -lt 1]` tests if script was passed less than 1 argument

$N – returns Nth argument passed to script

Ex: `sort $1` passes first string passed into script into sort command

$0 – command name

Ex: `echo "$0 needs 1 argument"` prints "<name of script> needs 1 argument"

$* returns all arguments

$@ returns a space separated string containing all arguments

"$@" prevents args originally quoted from being read as multiple args

Control Flow in bash

- Bash has loops and conditionals like most languages
- If Statements

```
if <test> then
  <commands>
fi
```

Ex:

```
if ./myprogram args; then
  echo “it works!”
else
  echo “it didn’t work”
fi
```

Executes body if ./myprogram succeeds (returns exit code 0)

- For loop

```
for <variable> in <list>
do
  <commands>
done
```

Ex:

```
for word in “list of words”
fo
  echo $word
done
```

“lists” in bash are just strings with white space separators

- while loop

```
while [test] do
  <commands>
done
```

Conditionals

- Test evaluates Boolean comparison of two arguments

```
test "$str1" == "$str2" #tests string equality
test -f result.txt #checks if file exists with -f option
test $num -eq 0 #checks integer equality with -eq option
test $# -ne 2 #checks if ints are not equal with -ne option
```

  - Other useful options: -lt –le –gt –ge

- Combine test with if by replacing “test” with []

```
if [ -f result.txt ]; then
```

  - Spaces around the brackets and semicolon are required

- Bash understands Boolean logic syntax
  - && and
  - || or
  - ! not

Common If Use Cases

- If file contains

```
if grep -q -E 'myregex' file.txt; then
  echo "found it!"
fi
```

-q option "quiet" suppresses the output from the loop

If is gated on successful command execution (returns 0)

- If incorrect number of arguments passed

```
if [ $# -ne 2 ]; then
  echo "$0 requires 2 arguments" >&2
  exit 1
fi
```

Checks if number of arguments is not equal to 2, if so prints an error message to stderr and exits with error code

Common loop use cases

- Iterate over files

```
for file in $(ls) <- All files + directories
do
  if [-f $file ]; then
     echo "$file"
  fi
done
```

- Iterate over arguments to script

```
while [ $# -gt 0 ]
do
  echo $*
  shift
done
```

Shift command moves through list of arguments

Similar to .next in Java Scanner

Exit Command

- Ends a script's execution immediately
  - Like "return"
- End scripts with a code to tell the computer whether the script was successful or had an error
- 0 = successful
  - exit without a number defaults to 0

```
exit
exit 0
```

- Non 0 = error

```
exit 1
```

Scripting demo: combine

Glob patterns

- Syntax to replace a pattern with a list of file names that all match that pattern
  - Enables you to pass multiple file names as arguments without typing them out individually
  - Pattern matches are based on location within file directory
- Wildcard – * – anything goes here
  - EX: echo src/*
  - Src/file1.txt src/file2.txt src/file3.txt
  - Example uses
    - echo * – prints every file/folder in current directory
    - echo *.txt – finds all files with that extension within directory
    - echo /bin/python* – finds all files within that path because they start with that string
    - cp src/* dest/ – copies all files from one directory to another

Regex

- Regular expressions (regex) are a set of rules for matching patterns in text
  - Used across programming languages and math
  - Different applications might have slightly different rules (yeah, it's frustrating…)
- Regex patterns can include characters, anchors and modifiers
  - Characters = the literal characters you are trying to match
  - Anchors – set the position in the line where a pattern may be found
    - ˆ anchor to front
    - $ anchor to end
  - Modifiers – modify the range of text pattern can match
    - * matches any number of characters
    - [set of chars]
- Regex basics, let P be our pattern and S be a string to match
  - P can be a single character (ex: a) to match S of the same single character
  - P_1P_2 matches S if $S=S_1S_2$ where $P_1 = S_1$ and $P_2 = S_2$
  - $P_1|P_2$ matches S if P1 or P2 matches S
- grep –e finds using regex
  - By default grep matches against .*p.*

Regex special characters

`\` – escape following character

`.` – matches any single character at least once

- `c.t` matches `{cat, cut, cota}`

`|` – or, enables multiple patterns to match against

- `a|b` matches `{a}` or `{b}`

`*` – matches 0 or more of the previous pattern (greedy match)

- `a*` matches `{, a, aa, aaa, …}`

`?` – matches 0 or 1 of the previous pattern

- `a?` matches `{, a}`

`+` – matches one or more of previous pattern

- `a+` matches `{a, aa, aaa, …}`

`{n}` – matches exactly n repetitions of the preceding

- `a{3} matches {aaa}`

`( )` – groups patterns for order of operations

`[ ]` – contains literals to be matched, single or range

- `[a-b]` matches all lowercase letters

`^` – anchors to beginning of line

`$` – anchors to end of line

Useful patterns

- [ˆabc] matches everything NOT abc
- [a-zA-Z] matches all English letters
- [0-9]* matches list of numbers

https://courses.cs.washington.edu/courses/cse374/20sp/lectures/lecture6history