CSE 374 Lecture 3

Emacs and I/O Redirection

HWo

- Due on Monday
- Goal: ensure that you are able to use the fundamental tools we need for this course
  - If you have problems, follow up to correct them ASAP.
- Debugging: in this course you need to work more independently. Use all the resources you have to find answers.
  - Ex: scp - how could you figure out that command?
- If you added the course late: You will have two days from when you get your klaatu account to complete the exercise before it uses a ‘late day’.

Bash Language

- Bash acts as a language interpreter
  - Commands are subroutines with arguments
  - Bash interprets the arguments & calls subroutine
  - Bash also has its own variables and logic

BASH applies its own processing to the I/O text - ‘globbing’

Shell Behavior

All redirection & string expansion or substitutions are done by the shell, before the command.

Command only sees resulting I/O streams.

Special characters

- Directory shortcuts
  - ~uname, ~
  - ./ or ../
- Wildcards (globbing)
  - 0 or more chars: *
  - Exactly one char: ?
  - Specified chars: [A-Z]
- History, or ‘!’

Special Characters

! > < & | * ~ [] “ ‘ ` $ /

What do they all mean?

\ is escape character

Would substitute things like $VAR

“string”

Suppresses substitutions

‘string’

I/O Streams

- All bash commands have three streams
  - 0- stdIn [keyboard]
  - 1- stdOut [screen]
  - 2-stdErr [screen]
- Can redirect streams
  - < yourInput
  - > yourOutput
  - >> appendYourOutput
  - 2> yourError
  - &> yourOutput&Error
  - And more…
- Special File /dev/null
  - Is EoF if input
  - Data is discarded if output
- Can combine one cmd to the next
  - Cmd1 | cmd2 - pipe output of cmd1 into input of cmd2
  - Cmd1; cmd2 - do one after another
  - Cmd1 `cmd2` - use output of cmd2 as argument to cmd1
- Can use cmd logic
  - Cmd1 || cmd2 - do cmd2 if cmd1 fails
  - Cmd1 && cmd2 - do cmd 2 if cmd1 succeeds

3.5.4 Command Substitution

Command substitution allows the output of a command to replace the command itself. Command substitution occurs when a command is enclosed as follows:

```
$(command)
```

or

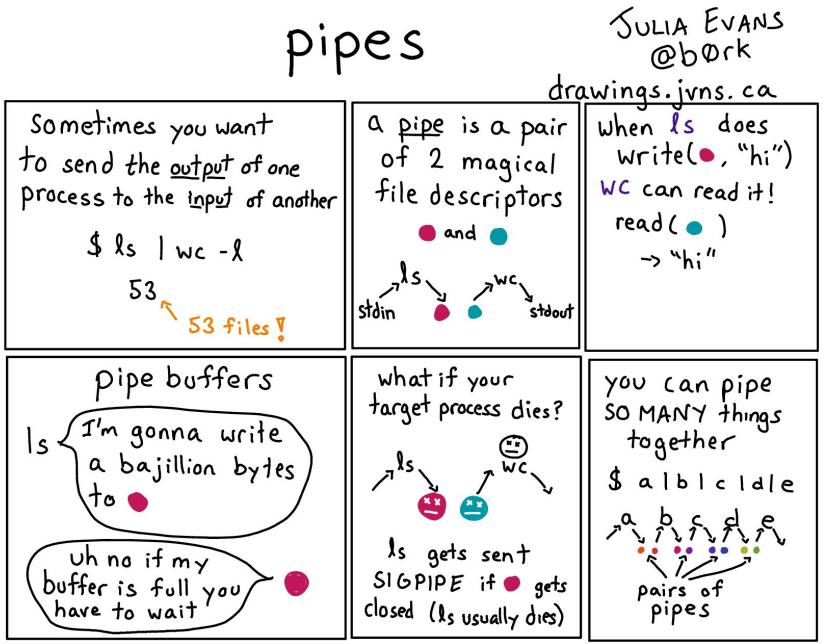
```
`command`
```

by executing *command* in a subshell environment and replacing the command substitution with the
nd, with any trailing newlines deleted. Embedded newlines are not deleted, but they may be removed
mmand substitution `$(cat file)` can be replaced by the equivalent but faster `$(< file)`.

e form of substitution is used, backslash retains its literal meaning except when followed by '$', '`', or '\'.
ded by a backslash terminates the command substitution. When using the `$(command)` form, all characters
e up the command; none are treated specially.

be nested. To nest when using the backquoted form, escape the inner backquotes with backslashes.

thin double quotes, word splitting and filename expansion are not performed on the results.



Some Bash redirection syntax

redirect stdout to a file →	*command* ***> output***
redirect stderr to a file	*command* ***2> output***
redirect stdout to stderr	*command* ***1>&2 output***
redirect stderr to stdout	*command* ***2>&1 output***
redirect stderr and stdout to a file	*command* ***&> output***

Reading: Bash Redirections (spec), bash hackers redirections (examples)

Alias

Defines a shortcut or ‘alias’ to a command.

(Essentially a really easy script)

Also, ‘alias’

.bashrc

Towards Scripts

- Shell has a state (working directory, user, aliases, history, streams)
- Can expand state with variables
- ‘Source’ runs a file and changes state
- Can run a file without changing state by running script in new shell.

Emacs (text editor)

C-x C-s #save

C-x C-c # quit

C-e # go to end of line

C-a # go to beginning of line

C-x C-f # find a file

C-g #exit menu

C-x C-k # kill a buffer

You can use any text editor you like. Emacs is amazingly powerful, and highly customizable with lisp scripts. It is probably worth learning.

Okay, lets make a script!

1. First line of file is #!/bin/bash (specifies which interpreter to execute)
2. Make file executable (chmod u+x)
3. Run a file ./myNewScript
4. Shell sees the shell program (/bin/bash) and launches it to run the script
5. Can include
    a. String tests (string returns true if non-zero length, string < string, etc.)
    b. Logic (&&,||,!) - use double brackets
    c. File tests (-d : is directory, -f: is file, -w: file has write permission etc.)
    d. Math - use double parens

Script Arguments & Errors

Script refers to i^{th} argument at $i ; $0 is the program

Use ‘shift’ to move arguments towards left ($i become $i-n)

Exit your shell with 0 (normal) or 1 (error)

Variables & Alias

Define variable

i=15

Access variable

$i

Undefined variable is empty string

Alias cheer=”echo yahoo\!”