What do you think?

Can you make an alias called ‘bestclass’ that automatically changes into you cse374 director whenever you call it?

Can you make it so that this alias is called every time you log into Cancun?

Go ahead - see if you and your neighbor solve this problem the same way.

CSE 374 Lecture 4

Shell Variables and Scripting

Feel free to ask questions until lecture starts...

Alias

Defines a shortcut or ‘alias’ to a command.

(Essentially a really easy script)

Also, ‘alias’

.bashrc

.bash_profile

- Executed for login shells
- Use for commands run once
  - Changing $PATH

.bashrc

- Executed for non-login shells
- Use for commands that are re-run
  - Aliases & functions

```
echo ‘alias greet="echo hello $USER"’ >> ~/.bashrc
greet
source ~/.bashrc
greet
```

Today

Scripting -Source / executable, exit codes, math

30 09:30-10:20 Lecture CSE2 G10 *I/O Redirection and alias* Slides Shell history, Alias demo History shortcuts Emacs motivation, Emacs demo 11:00-12:30 OH (Hazen) CSE1 212	01 12:00-13:00 OH (Bagaria) CSE2 274 13:00-15:00 OH (Luo) CSE1 403	02 09:30-10:20 Lecture CSE2 G10 *Introduction to scripting* Slides shiftdemo script, dcdls script, lectfour demonstration script Redirection Demo 16:00-17:30 OH (Zhao) CSE1 220 23:59 PRACTICE HW0 due; Shell Access Spec	03 14:30-16:00 OH (Chu) TBA	04 09:30-10:20 Lecture CSE2 G10 *Scripting Continued* Slides fibo script, sdel script Review **before** class: Exercises for this class session Extras: Emacs motivation, Emacs demo ,

October

Monday	Tuesday	Wednesday	Thursday	Friday
07 09:30-10:20 Lecture CSE2 G10 *RegEx, Grep* 11:00-12:30 OH (Hazen) CSE1 212 23:59 HW1 due; Bash Spec	08 12:00-13:00 OH (Bagaria) CSE2 274 13:00-15:00 OH (Luo) CSE1 403	09 09:30-10:20 Lecture CSE2 G10 *Regex and sed* 16:00-17:30 OH (Zhao) CSE1 220	10 14:30-16:00 OH (Chu) TBA	11 09:30-10:20 Lecture CSE2 G10 *Version Control* 23:59 HW2 due; Shell Script Spec

HW0 & HW1

Please remember that these two assignments should be done using Cancun.

If your homework passes the autograder this is sufficient….

To move files from seaside, the ‘scp’ command works well. Remember the command is `scp <copyfrom> <copyto>`, and pay attention to which computer you are executing it on.

Passwords, and managing Passwords

Linux systems have consistent password management.

- `/etc/passwd` file contains user info
  - Username
  - Password
  - Userid, groupid
  - Shell
  - Home directory
- `/etc/shadow` stores encrypted passwords

Change your password on Linux:

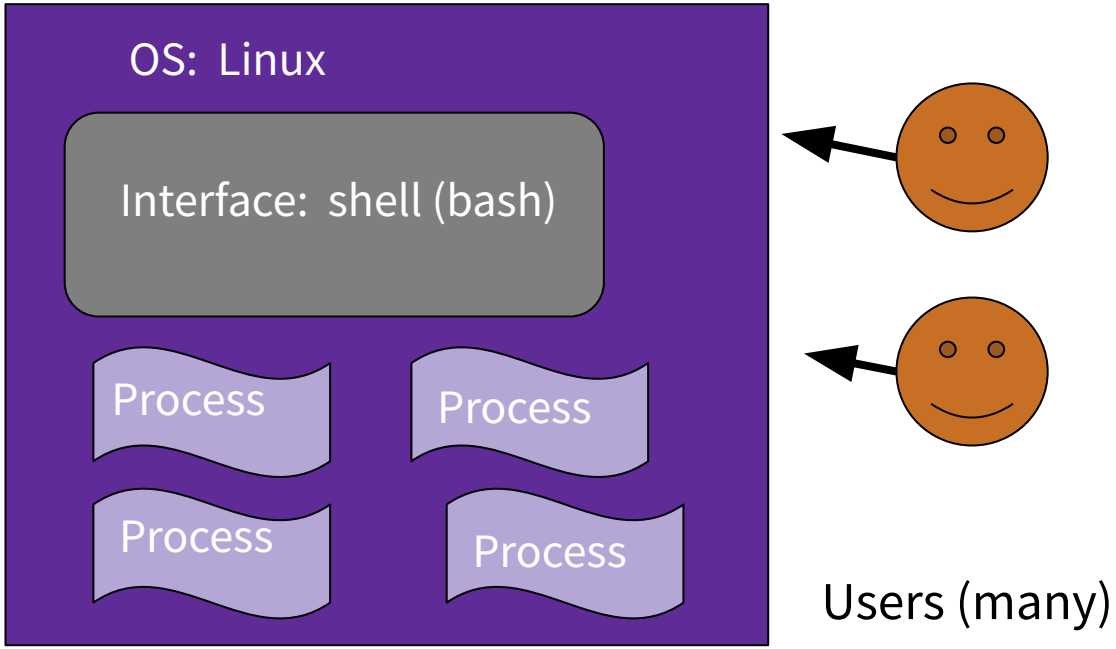
```
> passwd
```

Prompts for previous password, then new password

Passwd also has facilities for those with sudo access to update other user accounts and password management

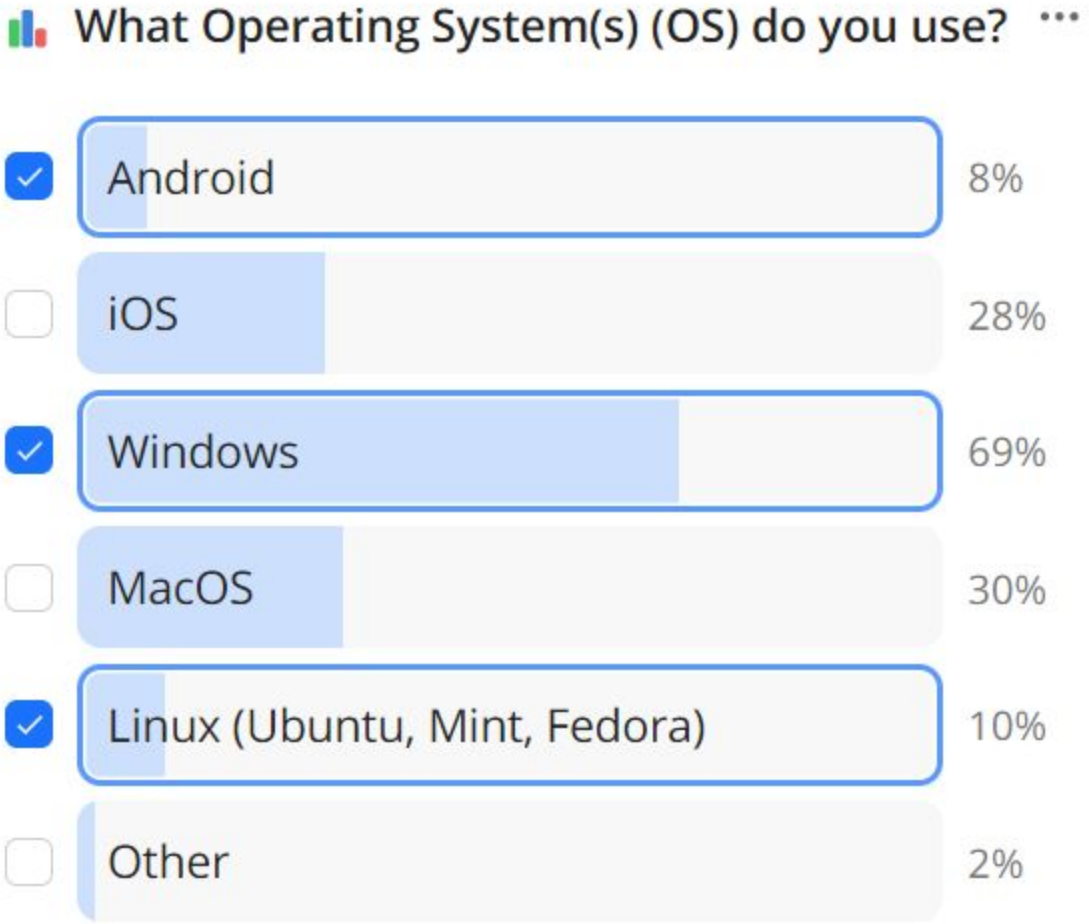
Cancun is a little different - passwords are obtained from the UWNetID servers (no /etc/passwd entries). Passwd will work, and propagate changes through UWNetID servers.

Computer Model



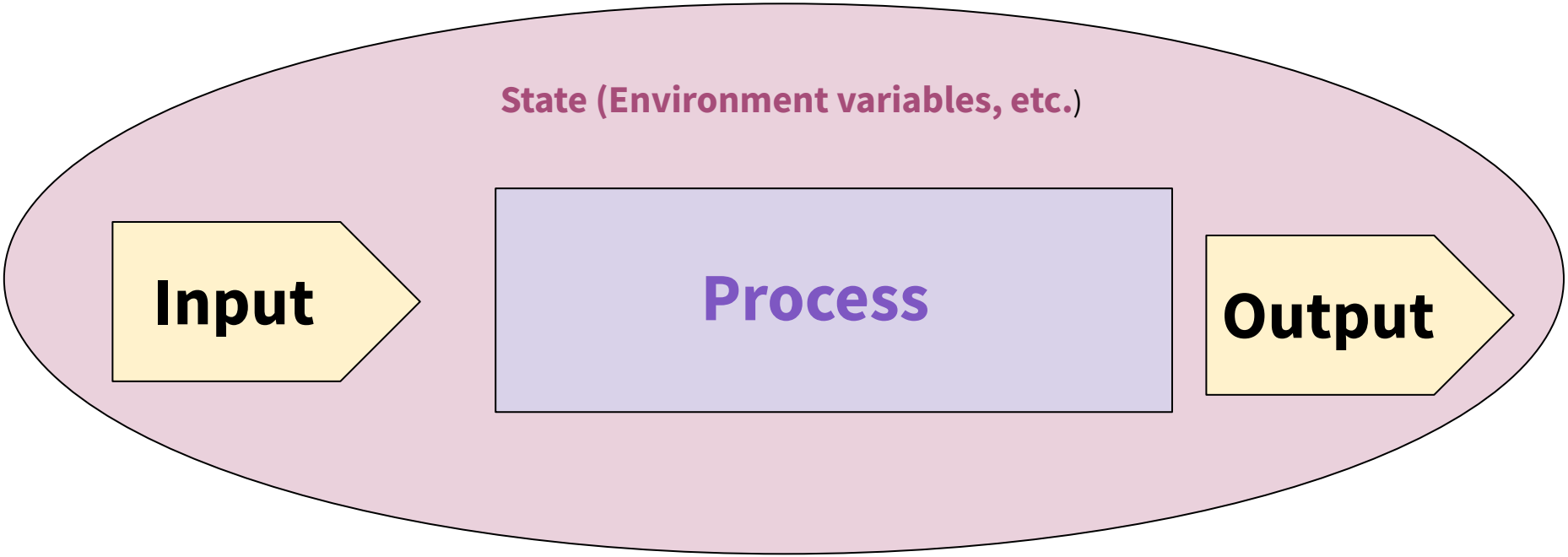
- Computers do two things
  - Store data (filesystem)
  - Manipulate data (processes)
- Shell is a process that allows the user to interact with the above.
- But, the shell also allows programming in its special language.

Aside


What Operating System(s) (OS) do you use?
Android
8%
iOS
28%
Windows
69%
MacOS
30%
Linux (Ubuntu, Mint, Fedora)
10%
Other
2%

Bash (shell) 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’

Special Characters

! > < & | * ~ [] “ ‘ ` $ /

What do they all mean?

\ is escape character

Would substitute things like $VAR

“string”

Suppresses substitutions

‘string’

Shell Behavior

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

Command only sees resulting I/O streams.

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

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)

Bash (shell) 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

Towards Scripts

- Shell has a state (working directory, user, aliases, history, streams)
- Can expand state with variables
- ‘Source’ runs a file and changes state

```
Printenv
echo $PS1
echo $PWD
echo $PATH
```

Special Variables

Common variables which set shell state:

$HOME - sets home directory. $HOME=~/CSE374 would reset your home directory to always be CSE374

$PS1 - sets prompt

$PATH - tells shell where to look for things. Often extended: $PATH=$PATH:~/CSE374

Show current state: `printenv`

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.
- Allows for repeatable processes and actions

Variables useful in a script

$# stores number of parameters (strings) entered

$0 first string entered - the command name

$N returns the Nth argument

$? Returns state of last exit

$* returns all the arguments

$@ returns a space separated string with each argument

(* returns one word with spaces, @ returns a list of words)

Variables

Shell has a state, which includes shell variables

All variables are strings (but can do math, later)

White space matters - not spaces around the ‘=’

Create: myVar= or myVar=value

Set: myVar=value

Use: $myVar

Remove: unset $myVar

List variables (use ‘set)

Export Variables

Use: export myVar

To make variable available in the initial shell environment.

If a program changes the value of an exported variable it does not change the value outside of the program

: export -n remove export property

Variables act as though passed by value

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

Exit Codes

Command ‘exit’ exits a shell, and ends a shell-script program.

Exit with no error:

Use `exit` or `exit 0`

Exit with error:

User `exit 1` or.. {1-255}

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)

Quoting Variables

In order to retain the literal value of something use ‘single quotes’

In order to retain all but $, ` , \ use “double quotes”

Put $* and $@ in quotes to correctly interpret strings with spaces in them.

Arithmetic

Variables hold strings, so we need a way to tell the shell to evaluate them numerically:

`K=$i+$j` does not add the numbers

Use the shell function ((

`k=$(( $i+$j ))`

Or `let k="$i+$j"`

The shell will automatically convert the strings to the numbers

What do you think?

Try it:

Download the lectfour demonstration script. Can you modify it so that it prints out the sum of two arguments?

Functions and local variables

Yes, possible

Generally, a script's variables are global

```
name () compound-command [ redirections ]
or
function name [()] compound-command [
redirections ]

Ex:

func1()
{
    local var='func1 local'
    func2
}
```

Stuff to watch out for

White space: spacing of words and symbols matters

Assign WITHOUT spaces around the equal, brackets are WITH SPACES

Typo on left creates new variable, typo on right returns empty string.

Reusing variable name replaces the old value

Must put quotes around values with spaces in them

Non number converted to number produces ‘0’

Conditionals

Binary operators: -eq -ne -lt -le -gt -ge

Can use the [[shell command to use < , > , ==

Syntax is a little different, but commands works as expected

if *test*; then
    commands
fi

while *test*; do
    commands
done

for *variable* in *words*; do
    commands
done

Flow control

test *expression* or [*expression*]

```
if [ -f .bash_profile ]; then
    echo "You have a .bash_profile.
Things are fine."
else
    echo "Yikes! You have no
.bash_profile!"
fi
```

http://linuxcommand.org/lc3_man_pages/testh.html

Shell-scripting Notes

Bash Scripting	Java Programming
Interpreted	Compiled
Esoteric variable access	Highly structured, Strongly typed
Everything is a string	Strings have library processing
Easy access to files and program	Data structures and libraries
Good for quick & interactive programs	Good for large complex programs

Scripting Style Guide

Scripts should generally be <200 lines

Do one thing and do it well.

Always use spaces, not tabs (indent line with two spaces)

Comment code with ‘#’

https://google.github.io/styleguide/shell.xml

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.