CSE 374 Lecture 4

Shell Variables and More Scripting

Feel free to ask questions until lecture starts...

Today

1. Quick review
2. Script necessities
3. Script examples
4. Arguments & Exit conditions

Office hours this week:

Office Hours are subject to change. Check your calendar.

1/9/2022: All office hours on using Zoom until further notice.

Tuesdays, 2:30-3:30pm, Zoom	Mohit
Tuesdays, 3:30-4:30pm, Zoom	Diana
Wednesdays, 1:00-2:00pm, Zoom	Mohit
Wednesdays, 1:30-2:30pm, Zoom	Dixon
Thursdays, 11:00-12:00, Zoom	Yitian
Thursdays, 1:30-2:30pm, Zoom	Maxim
Fridays, 4:00-5:00pm, Zoom	Yitian

HW0 & HW1

Please remember that if you used a system other than klaatu or the CSE virtual machine you will want to check your homework on one of these systems before submitting.

If your homework passes the autograder this is sufficient….

Getting files from the VM to Gradescope: There are options, but, I use the GUI and drag a file from my VM to my Windows desktop, and then upload from a windows browser. Can also just open a browser in the VM, go to gradescope, upload directly.

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

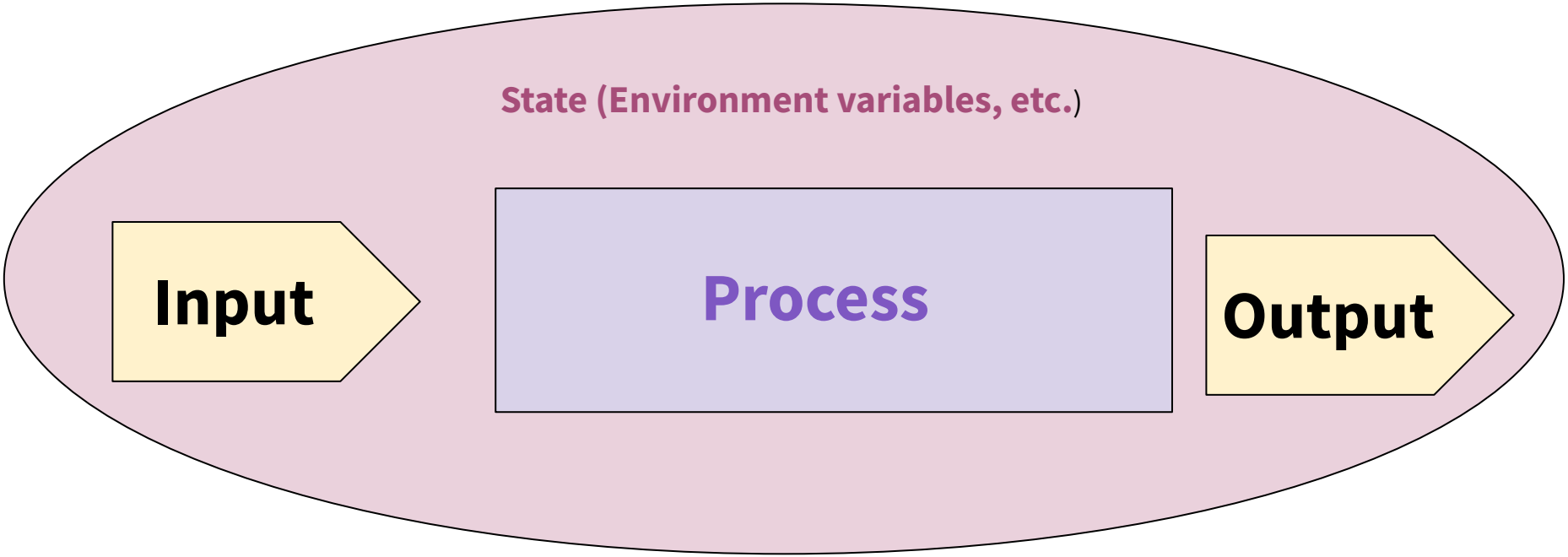
Change your password on Linux:

```
> 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.



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

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)

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`

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

Special Characters

! > < & | * ~ [] “ ‘ ` $ /

What do they all mean?

\ is escape character

Would substitute things like $VAR

“string”

Suppresses substitutions

‘string’

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.

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)

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}

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)

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

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

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