CSE 374: Programming Concepts and Tools

Spring 2026
Instructor: Megan Hazen

Today's Goals

Subject Matter

- Dive into Bash
  - Special characters
  - I/O & Redirection
  - Variables
- Alias
- Set up for scripts

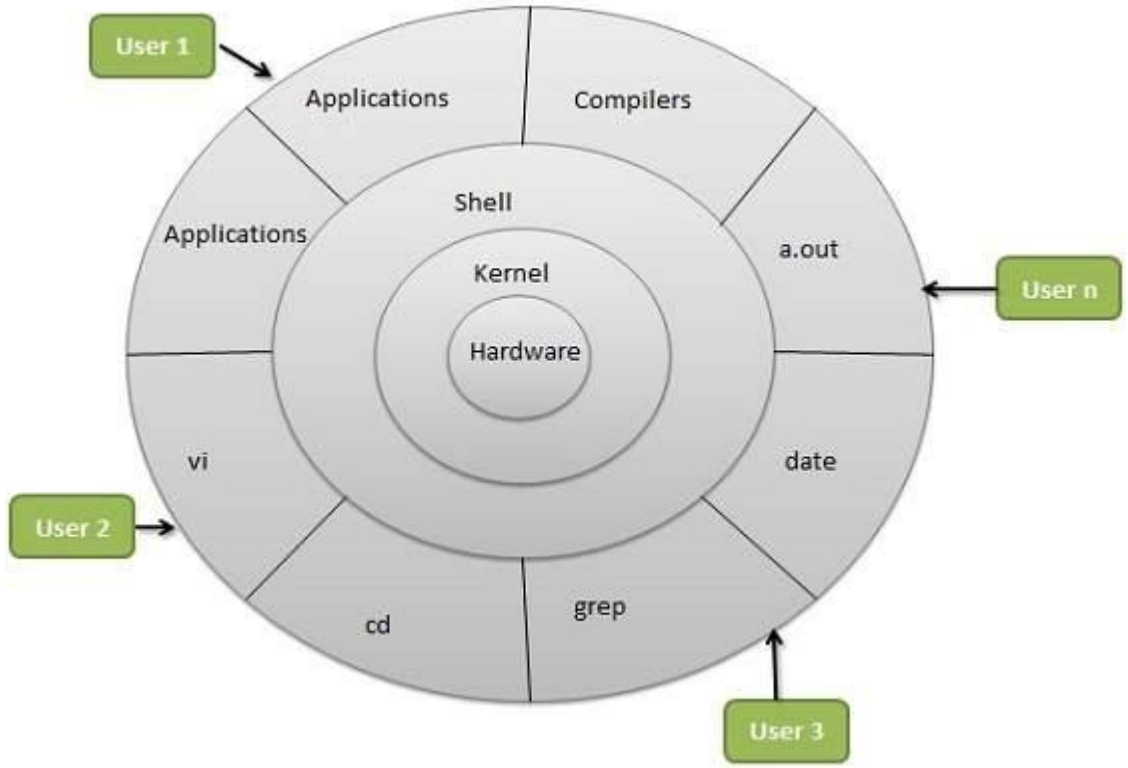
Your Goals

- HWo
  - Find description on webage
  - Complete HWo
  - Find Gradescope to turn it in
- Second practice problem on Gradescope

Aside: Text Editors

- Editors
  - emacs intro
  - VI reference
  - choosing
  - Using Visual Studio remotely

What is Linux?



Linux is an operating system

Portable, open source, built with C

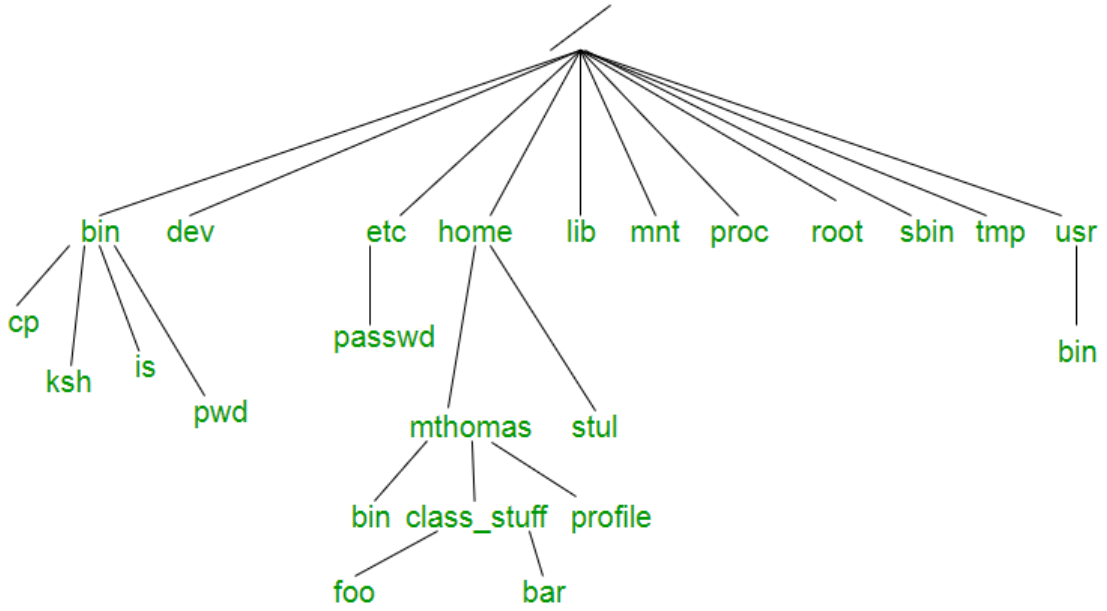
Kernel – does all the hardware interaction

Shell – provides interface for users

Users – control and run various processes (applications)

File Systems

- Data is stored in files
- Files are organized in a file system
- File systems are trees
- Absolute file address starts from the top (`/`)
- Relative file address from current position
  (`./`) – this directory
  (`../`) – one directory up
- Home directly has short-cut (`~`)



https://tldp.org/LDP/intro-linux/html/sect_03_01.html
- Linux layout, but windows is similar

Demo: `whoami, pwd, ls, mkdir, cd, cp, mv, rm, cat`

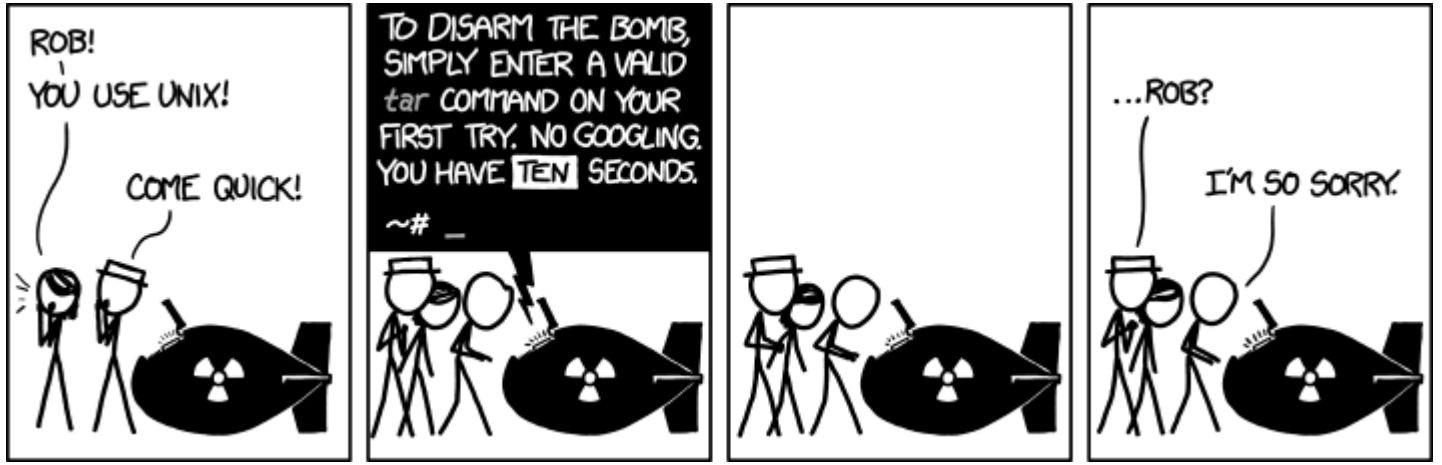
Processes

"On a UNIX system, everything is a file; if something is not a file, it is a process."

- Shell essentially runs programs, or processes. Shell *is* a process and has a state.
- Usually launch a process and return to shell when done.
- Each process has own memory stream and I/O
- Stdin (keyboard), stdout (console), stderr
- Many processes have options

- '&' runs process in the background
- `fg, bg, top, kill`
- Step through a script with built-in `source`
- Can redirect input and output ('<', '>', '>>')
- Redirect output, `cat, more/less`

Getting Help




```
[mh75@calgary ~]$ man date
[mh75@calgary ~]$ date –help
[mh75@calgary ~]$ man –k date
[mh75@calgary ~]$ apropos date
```

- https://tldp.org/LDP/intro-linux/html/sect_02_03.html

Bash as a 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

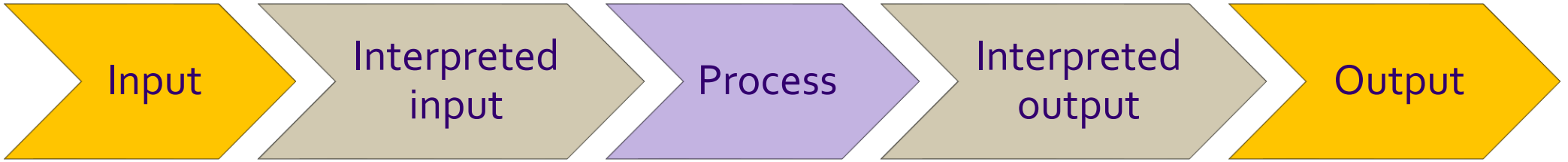
Bash Globbing

Bash applies its own processing (***globbing***) to the I/O (Input / Output)


```
[mh75@calgary cse374]$ ls hw0.script
hw0.script
```

Globbing at work

Bash does redirection, string-expansion, & substitutions
Process only sees the interpreted I/O




```
[mh75@calgary cse374]$ ls *.script
hw0.script  hw1.script
```

Bash Special Characters

Directory Shortcuts
- ~uname or ~
- ./ or ../

Wildcards - Globbing
- 0 or more chars: *
- Exactly 1 char: ?
- Specified chars: [a-f]

History or `!`

! > < & | * ~ [] “
‘ ` $ /

\ is escape character

“string”
- Glob within the string

‘string’
- Use the string as-is

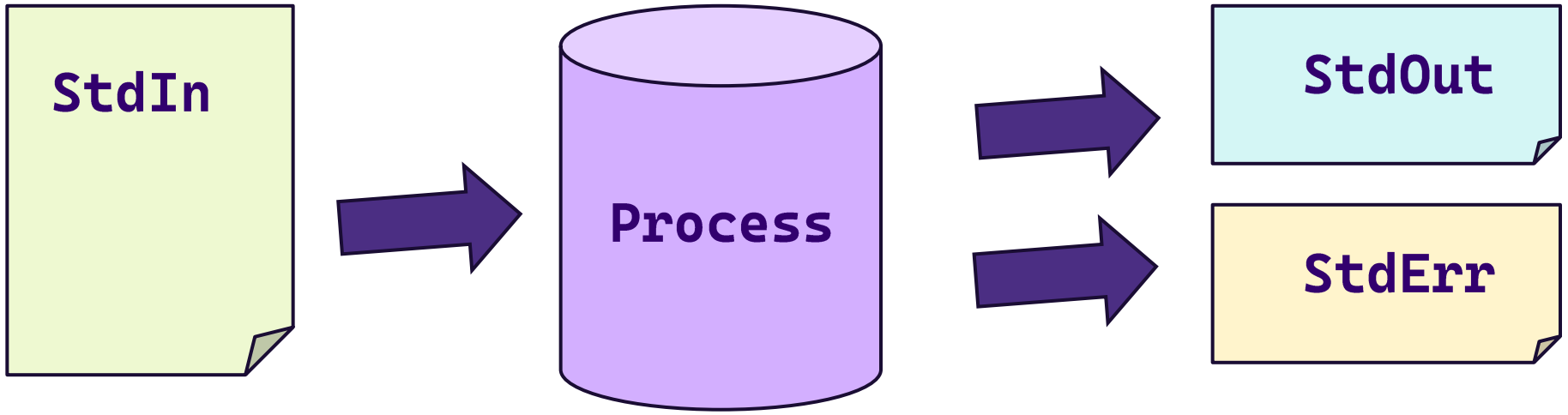
Bash Variables

- Define variable: **`i=15`**
- Access variable: **`$i, ${i}`**
- Undefined variable is empty string
- Global scope within environment
- See your variables:
  - **`printenv`**
  - **`env`**
  - **`set`**
  - **`declare -p`**

```
[mh75@calgary ~]$ i=5
[mh75@calgary ~]$ j=
[mh75@calgary ~]$ echo $i
5
[mh75@calgary ~]$ echo $ith

[mh75@calgary ~]$ echo ${i}th
5th
[mh75@calgary ~]$ echo $j
```

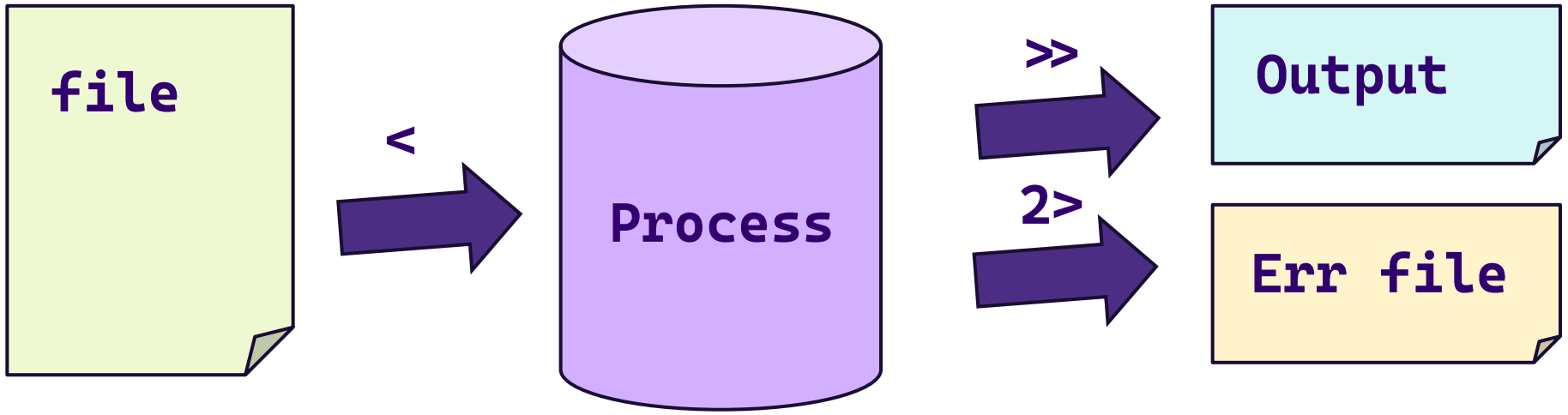
Input-Output (IO) streams



- All processes have these three streams
  - StdIn: keyboard
  - StdOut = StdErr: terminal output

IO Redirection

Special file: /dev/null
Is EoF if input
Data is discarded if output



- All processes have these three streams, but, you can redirect
  - Use '<' to redirect a file in
  - Use '>' to redirect out. '>>' appends, '2>' says redirect the second stream, or error

IO Redirection Syntax

To redirect:	Use this 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**

Combining Commands

- Can also redirect commands as input to other commands
  - `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
- And do logic with commands
  - `Cmd1 || cmd2` - do cmd2 if cmd1 fails
  - `Cmd1 && cmd2` - do cmd 2 if cmd1 succeeds

```
[mh75@calgary cse374]$ ls | wc
      2       2      22
[mh75@calgary cse374]$ wc `ls`
   19    99   913 hw0.script
  268  1320 32528 hw1.script
  287  1419 33441 total
[mh75@calgary cse374]$ wc $(ls)
   19    99   913 hw0.script
  268  1320 32528 hw1.script
  287  1419 33441 total
[mh75@calgary cse374]$ ls hw0.script &&
wc hw0.script
hw0.script
 19  99 913 hw0.script
[mh75@calgary cse374]$
^hw0.script^hw3.script
ls hw3.script && wc hw0.script
ls: cannot access 'hw3.script': No such
file or directory
```

Alias

- `alias`
  - Returns any aliases you have set
  - Defines a 'short-cut' or alias to a command
  - `unalias`

Aside - `.bashrc / .bash_profile`

'preferences' files that include your personal environment settings

`.bash_profile` is sourced every time you log-in

`.bashrc` is sourced every time you open a new interactive shell (usually called by `.bash_profile`)

Can put aliases or environment variables in `.bashrc` if you want to always use them.

Shell states & Scripting

- Shell **has a state**
  - Working directory, aliases, variables, history, streams
- Can change your state with command line operations (such as defining a new alias)
- Can use `source` to execute all the commands in a file, in your current shell, and **change your state**.
- You can also run commands in a file **without** changing state – a SCRIPT!

```
[mh75@calgary cse374]$ echo
"alias six='echo 6 7'" >>
party
[mh75@calgary cse374]$ six
-bash: six: command not
found
[mh75@calgary cse374]$
source party
[mh75@calgary cse374]$ six
6 7
```

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

Source & Executable

We demonstrated source, but what about running a script?

- Want to run in a separate shell
- Need to specify what shell / interpreter
  - `#/bin/bash`
- Need to make the file executable
  - `chmod u+x`

```
CHMOD(1)

NAME
       chmod - change file mode bits

SYNOPSIS
       chmod [OPTION]... MODE[,MODE]... FILE...
       chmod [OPTION]... OCTAL-MODE FILE...
       chmod [OPTION]... --reference=RFILE FILE...

[mh75@calgary cse374]$ chmod 744 hello.sh
```

DEMO

wget https://courses.cs.washington.edu/cse374/26sp/lectures/hello.sh

Happy Weekend

HW 0, start HW1

Practice problems for Lectures 2&3