Lecture 4: Bash Scripts

CSE 374: Intermediate Programming Concepts and Tools

Administrivia

- Course webpage live!
- HW1 actually posted
  - after this lecture you now have everything you need
  - klaatu accounts created
    - set up SSH
    - ssh yourUWNetId@klaatu.cs.washington.edu
    - password: tempPassword
    - passwd #will let you change your password
    - issues fill out this form: https://forms.gle/sFojyRU4eC8kJT4T9
  - Grading scripts are still being adjusted

Terminal Text Editing

- While working with the text interface for the OS often it is helpful to use the text interface editors
  - You do not have to use any of these in this course
  - Editors are a matter of preference

PICO

```
~ — ssh champk@klaatu.cs.washington.edu — 80×24
  UW PICO 5.09                         File: PeterPan.txt

Chapter 1 PETER BREAKS THROUGH

All children, except one, grow up. They soon know that they will grow
up, and the way Wendy knew was this. One day when she was two years old
she was playing in a garden, and she plucked another flower and ran with
it to her mother. I suppose she must have looked rather delightful, for
Mrs. Darling put her hand to her heart and cried, "Oh, why can't you
remain like this for ever!" This was all that passed between them on
the subject, but henceforth Wendy knew that she must grow up. You always
know after you are two. Two is the beginning of the end.

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

EMACS

```
~ — ssh champk@klaatu.cs.washington.edu — 80×24
File Edit Options Buffers Tools Help
Chapter 1 PETER BREAKS THROUGH

All children, except one, grow up. They soon know that they will grow
up, and the way Wendy knew was this. One day when she was two years old
she was playing in a garden, and she plucked another flower and ran with
it to her mother. I suppose she must have looked rather delightful, for
Mrs. Darling put her hand to her heart and cried, "Oh, why can't you
remain like this for ever!" This was all that passed between them on
the subject, but henceforth Wendy knew that she must grow up. You always
know after you are two. Two is the beginning of the end.

-UUU:----F1  PeterPan.txt    All L1      (Text) ----------------------------------
PeterPan.txt has auto save data; consider M-x recover-this-file
```

VIM

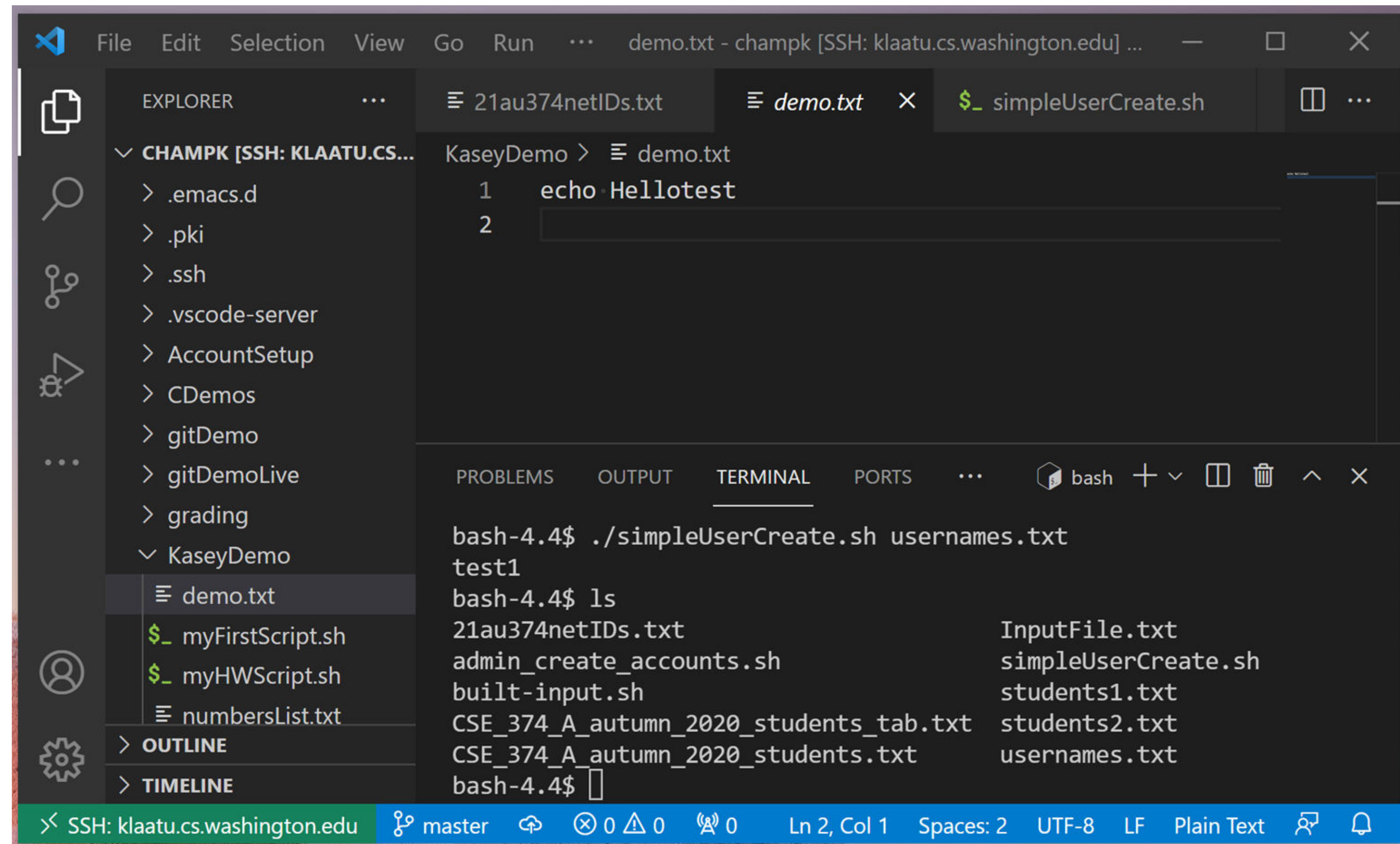
```
~ — ssh champk@klaatu.cs.washington.edu — 80×24
Chapter 1 PETER BREAKS THROUGH

All children, except one, grow up. They soon know that they will grow
up, and the way Wendy knew was this. One day when she was two years old
she was playing in a garden, and she plucked another flower and ran with
it to her mother. I suppose she must have looked rather delightful, for
Mrs. Darling put her hand to her heart and cried, "Oh, why can't you
remain like this for ever!" This was all that passed between them on
the subject, but henceforth Wendy knew that she must grow up. You always
know after you are two. Two is the beginning of the end.

~
~
~
~
~
~
~
~
~
~
~
~
"PeterPan.txt" 11L, 592C                                      1,1           All
```

Working in VS Code

1. [Install VS Code]
   a. it's free!
2. ctrl + shift + P or F1 to open SSH dialog
   a. enter in klaatu account info
3. open folder to see your files
4. use terminal on bottom



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 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

Script Error Checking

- If incorrect number of arguments passed

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

  - Check if file exists

```
if [ ! -f $1 ]; then
  echo "File does not exist" <& 1
  exit 1
fi
```

Exit Codes

The command “exit” exits a shell and ends a shell-script program

ctrl+c aborts a process

When a process exits you can use a number to indicate either a success or failure

0 = success

1 or any non-zero positive number = error

grep

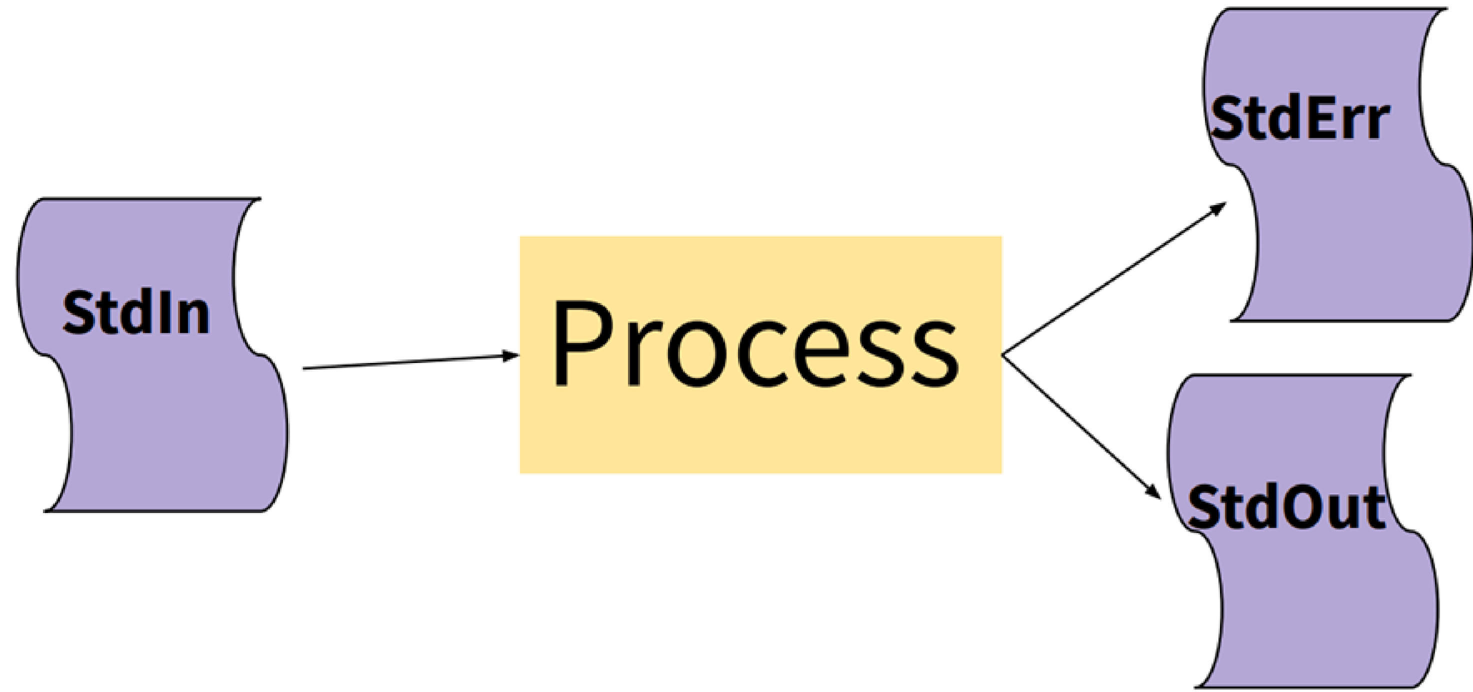
- Search for a given string within a given file
  - `grep [options] pattern [files]`
  - EX: `grep “computer” /usr/share/dict/words`
- Helpful Options
  - `-c` : prints count of lines with given pattern
  - `-h` : display matched lines (without filenames)
  - `-i` : ignore case when matching
  - `-l` : display list of filenames with matches

```
$ grep 'computer' /usr/share/dict/words
computer
computerese
computerise
computerite
computerizable
computerization
computerize
computerized
computerizes
computerizing
computerlike
computernik
computers
microcomputer
microcomputers
minicomputer
minicomputers
multicomputer
multimicrocomputer
supercomputer
supercomputers
telecomputer
```

https://www.geeksforgeeks.org/grep-command-in-unixlinux/

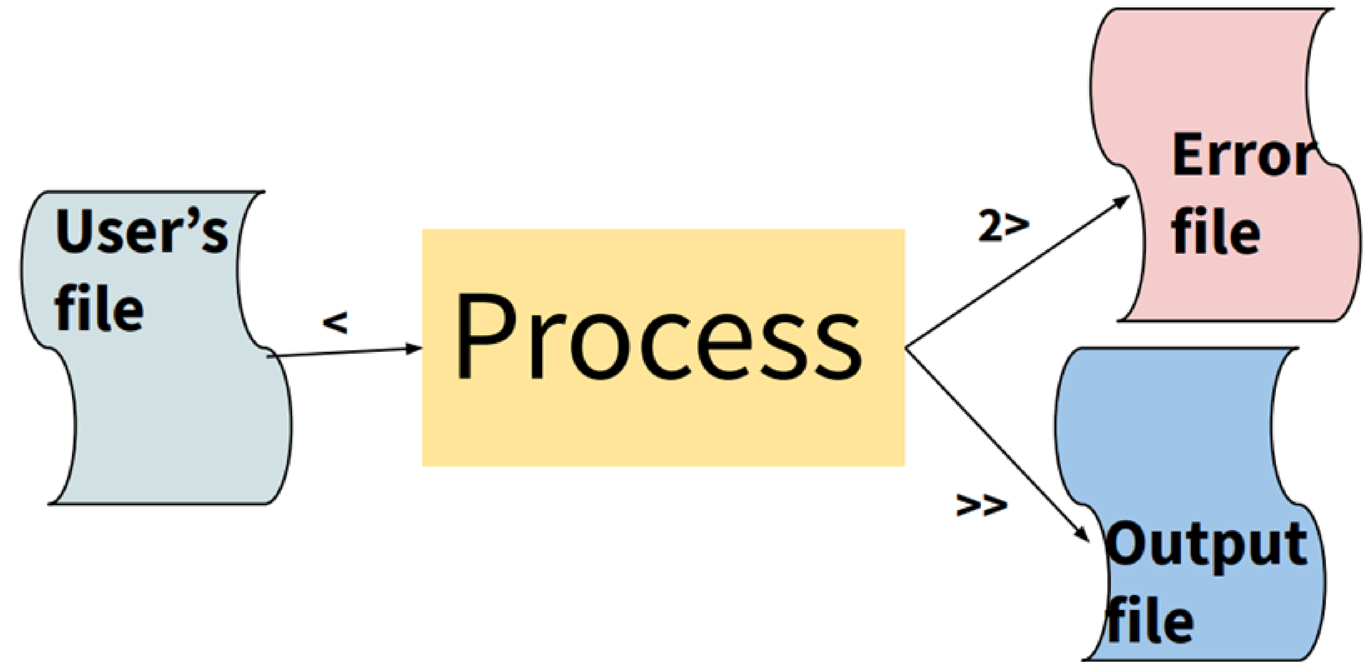
Demo: Grep

Processes all can take INPUT from one source, the default being StdIn.



Processes have two OUTPUT destinations, the default being StdOut and StdErr. You can think of these as two potential files to which a processes can write.

But, instead of using StdIn you can use any file, and ‘redirect’ it in by using the ‘<’ symbol (pointing towards process).



You can also write to different files instead of StdErr or StdOut. The ‘>’ symbol means to put in an new file, while ‘>>’ means to append to the end of a file. The ‘2’ specifies that you want iostream ‘2’, or the error stream.

Input / Output Streams

- stdin – Standard in (File Descriptor = 0)
  - Keyboard input typed into the terminal
- stdout – Standard out (File Descriptor = 1)
  - Results of a process after it has completed printed to the screen of the terminal
- stderr – Standard error (File Descriptor = 2)
  - Results of a process if it exits in error printed to the screen of the terminal

Redirecting Streams

Redirection Syntax:

- < yourInput
- > yourOutput
- >> appendYourOutput
- 2> yourError
- &> yourOutputAndError
- Stdout & stderr default to terminal

Examples

- 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`

https://www.gnu.org/software/bash/manual/html_node/Redirections.html

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***

I/O Piping

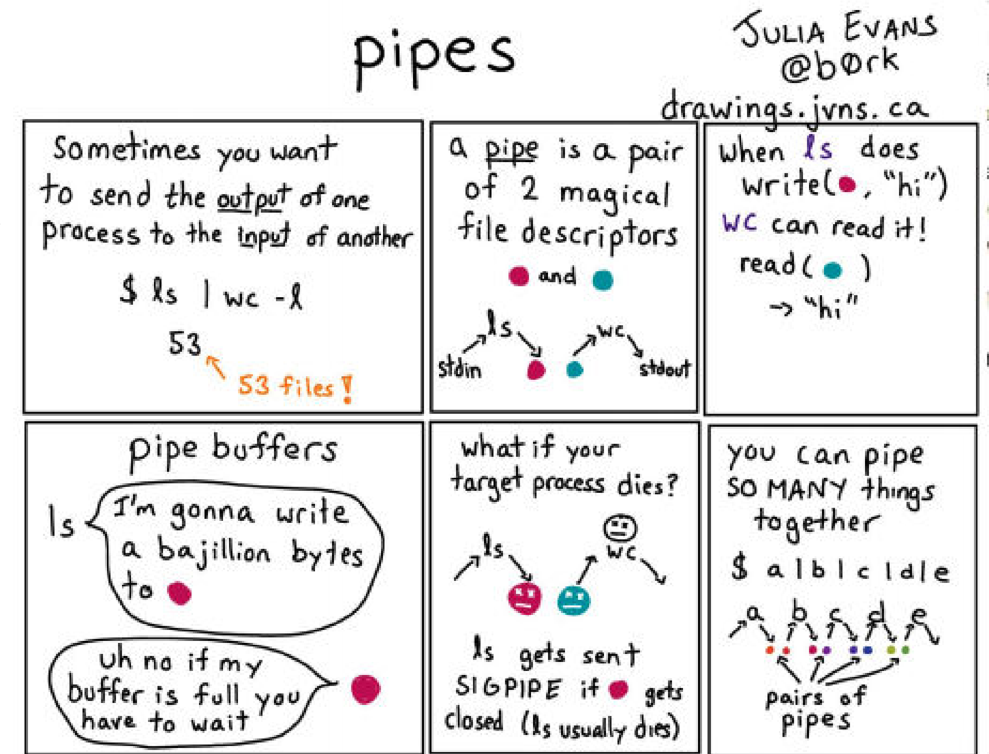
We can feed the stdout of one process to the stdin of another using a pipe ("|")

- Data flows from process to the other through multiple transformations seamlessly
- Similar to redirection, but specifically passes streams into other programs instead of their defaults

Example:

- Instead of:
  - du –h –d 1 . > sizes.txt
  - grep 'M' sizes.txt
- We can use piping
  - du – h –d 1 . | grep 'M'

- Piping is effective when you have one set of data that needs to be transformed multiple times
  - Cmd1 | cmd2 – pipe output of cmd1 into input of cmd2



[Video: The Magic of Piping]

What is the difference between | and >?

- 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

Demo: Stream Redirection