CSE320 System Fundamentals II

TONY MIONE

Topics

Unix Commands

- Saving session commands and output
- Using the file system
  - Changing ‘current directory’
  - Listing files in a directory
  - Creating/deleting files and directories
- Files
  - Displaying file contents
  - Copying/renaming file

Finding Commands and how to use them

- apropos
- man

I/O redirection

Unix commands

Logging in from to a terminal window (or xterm) provides the user with a ‘shell’

Commands:

- Provide information on files
- Manipulate files
- Manipulate data from files
- Run arbitrary utilities available on the system

Saving a session

script – Saves a session including commands and output until user enters ^D (eof)

- script – Saves to the file *typescript*
- script *filename* – Saves to the *filename* provided by user

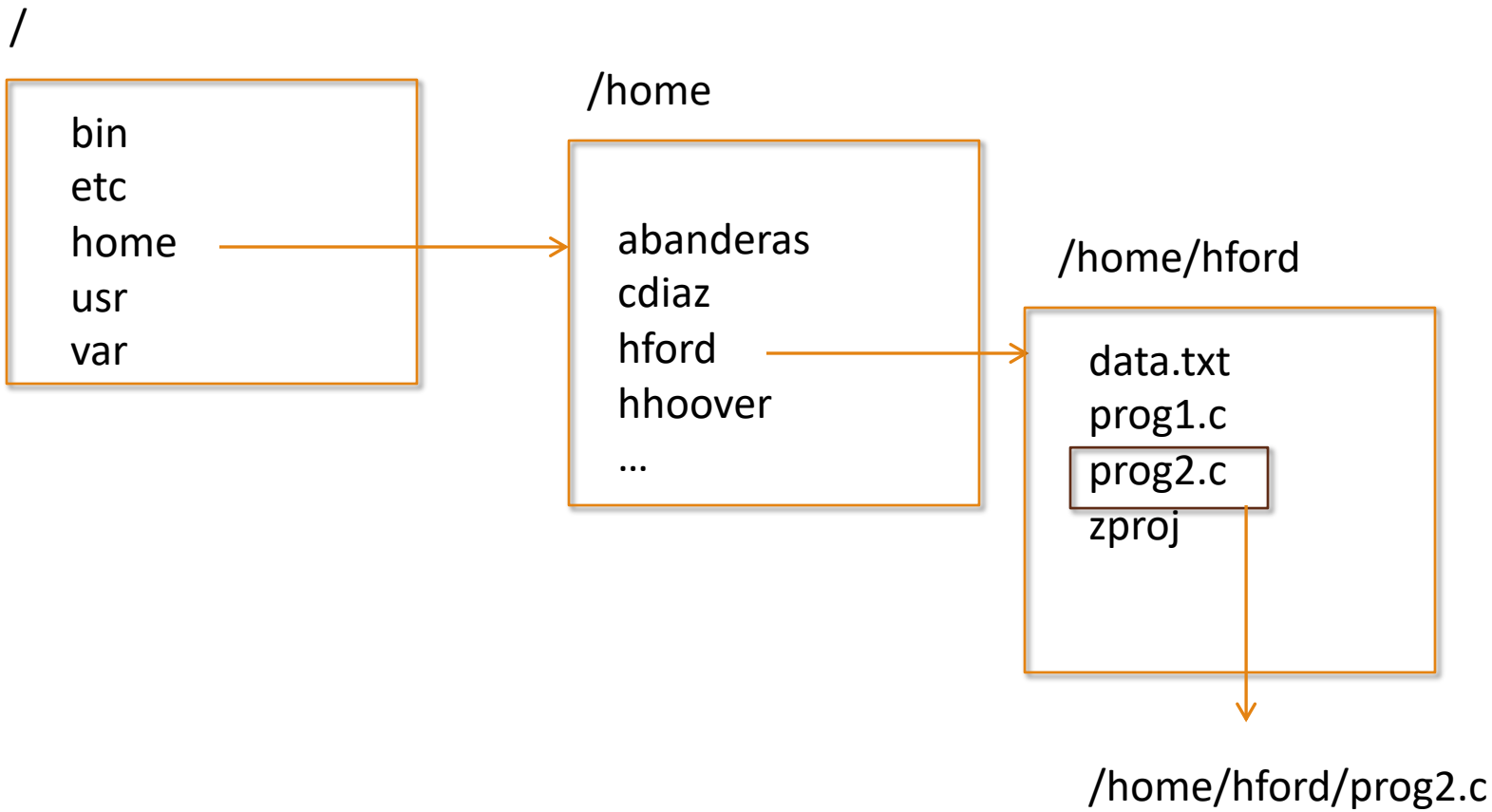
The unix filesystem

A hierarchical collection of directories and files

The *root* of the filesystem is '/'

A *path* contains a series of directories seperated by '/'

Unix filesystem



Navigating the filesystem

A user process has a current *working directory*

pwd – Print Working Directory

Commands to change working directory:

- **cd *<new_working_directory>*** – Connect to a new working directory [‘-’ indicates the last working directory]
- **pushd *<new_working_directory>*** – Connect to directory and save old directory on a *directory stack*
- **popd** – Change working directory to the one on the top of the directory stack
- **dirs** – List directories in the directory stack

Directory paths

- **Absolute** – specifies all directories from ‘/’ to desired directory [Must begin with ‘/’]
- **Relative** – specifies the directory relative to current working directory [begins with a name or ‘..’]

Navigating the filesystem

Directory paths

- . – Current directory
- .. – Parent directory

Examples:

- cd /usr/include/os
- pushd ../../bin
- cd
- cd –

Directory Listing

ls [options] ***<path>*** – List directory

- Options provide details about information requested and displayed
- <path> can be absolute, relative, or absent (for current working directory)
- <path> can include just directory or directory+file name

Relative paths:

- Exclude leading '/'
- '.' is current directory
- '..' is parent directory
- Example: ls ../../usr/bin – go up two directory levels, then down into usr/bin

File ‘globbing’

File names can contain wildcards:

- * – match any string of 0 or more characters
- ? – match any single character
- [<characters>] – Any character in the list within []

Examples

- ls *.c
  - All files with a .c extension (C source code files)
- ls notes??.txt
  - All (ascii text) files starting with the word ‘notes’ and ending with two other characters.
- ls [a-c]*.txt
  - All .txt files beginning with ‘a’, ‘b’, or ‘c’
- ls */*.c
  - Match all files with a ‘.c’ extension in a directory one level down from the connected directory.

Activity

Use ‘script’ to record the following session:

- Connect to /usr/local/bin
- Using a relative path, connect to /usr/include
- List files with extensions of .h
- List files with extensions of .h in all directories below your current working directory
- Connect to your home directory once again
- Type ^D to exit the script application and stop recording

Creating files/directories

touch ***<filename>***

- Creates an empty file with the give name

Any editor can be used to create a file:

- **vi** – standard unix editor
- **emacs** – epic multi-platform editor

mkdir [options] ***<dirname>***

- Creates a directory
- -p – create the whole path to the directory
- Example: mkdir –p project/src/include

Removing files/directories

rm ***<filename>***

- Delete the named file(s)

rmdir ***<dirname>***

- Deletes the named directory(ies)
- Directory **must** be empty
  - Note: Files starting with '.' are 'invisible' by default. If rmdir reports 'directory not empty' but you do not see any files with ls, try 'ls –a'.

Displaying file contents

cat ***<filename>*** (conCATenate)

- Sends <filename>'s contents to terminal

less ***<filename>***

- Sends <filename>'s contents to terminal with pauses when screen if full
- <spacebar> will continue listing
- 'q' will terminate the output
- *more* command does the same thing on my Unix systems

head -# ***<filename>*** *- This prints the first few lines of a file (count given by value of #)*

tail -# ***<filename>*** *- This prints the last few lines of a file (count given by value of #)*

Copying and renaming files

cp ***<oldname> <newname>***

- *Copy file <oldname> to <newname>*
- *Filenames can include absolute or relative paths*

mv ***<oldname> <newname>***

- *Move or 'rename' a file from <oldname> to <newname>*
- *Filenames can include absolute or relative paths*

Learning more – ‘man’

man <section> <text>

- Displays a manual page about the command or program (<text>)
- <section> is optional (1-8) - Specifies the ‘section’ of the manual
  - Names may appear in 2+ sections (‘open’ shell command versus ‘open()’ library call)
  - Man page sections:
    1: User commands
    2: System Calls
    3: C Library Functions
    4: Devices and Special Files
    5: File Formats and Conventions
    6: Games
    7: Misselaneous
    8: Administrator commands and Daemons
- More a ‘reference’ than a ‘tutorial’

Learning more – ‘apropos’

apropos <text>

- Searches man page titles and descriptions for given <text>
- Lists man entries that match the text
- Same as ‘man –k’

Learning more – info

info brings up an emacs buffer with a menu of topics

Topics are marked with a leading '*'

Topics (tutorials) can be opened by moving to the text (with arrow keys) and hitting the enter key

Other info commands:

- 'p' – move to previous topic
- 'u' – move up to earlier menu in hierarchy
- Space bar – page down in text
- Delete key – page up in text
- 's' – search for text. Type the text at prompt. Enter key searches for next string occurrence

I/O Redirection

Shells allow command output and input to be ‘redirected’ with special characters

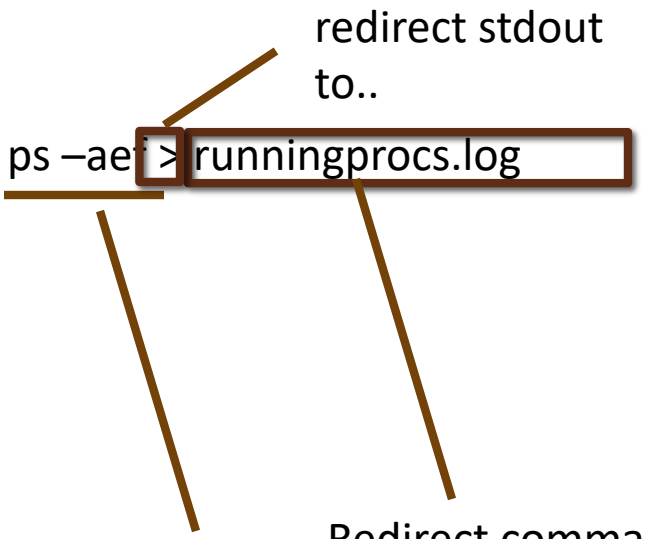
3 ‘Standard’ files are created for each process (these can be redirected as well)

- stdin – By default, this is input from the keyboard
- stdout – By default, this is output to the screen
- stderr – By default, this is output to the screen but is used for ‘error’ text

Most shells use:

- ‘<‘ to redirect stdin
- ‘>’ to redirect stdout
- ‘2>’ to redirect stderr
- ‘|’ connects output of one command to input of the next

I/O Redirection examples

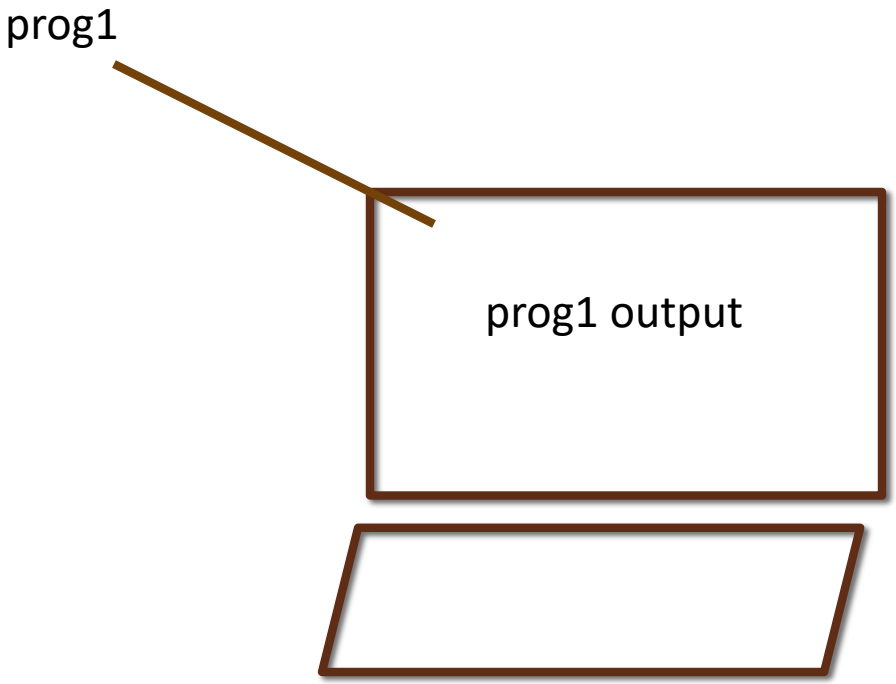


Pipes

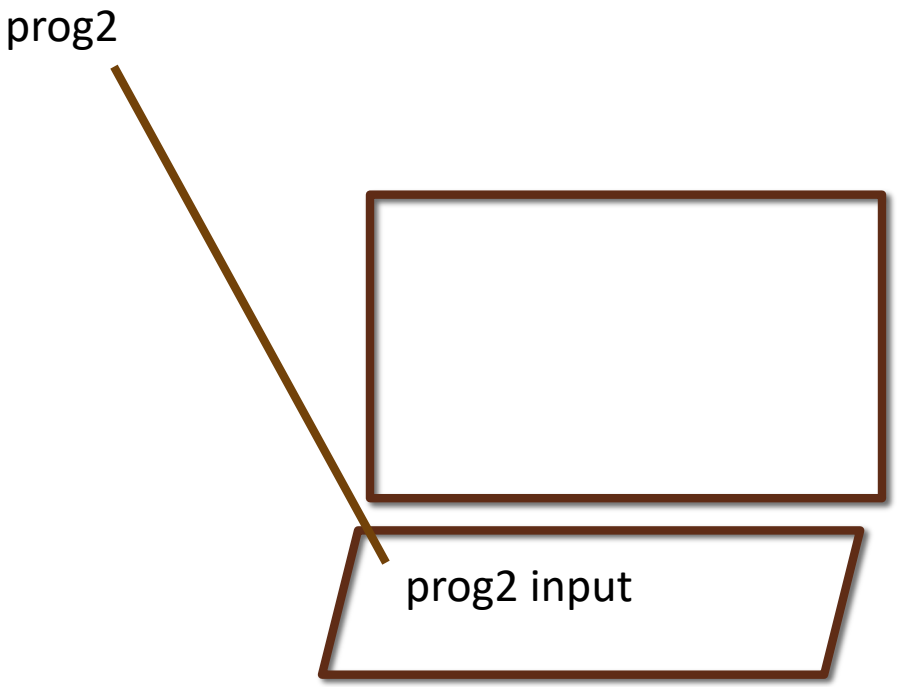
Redirect output from one command/program directly to the input of another

Use ‘|’ between programs or commands

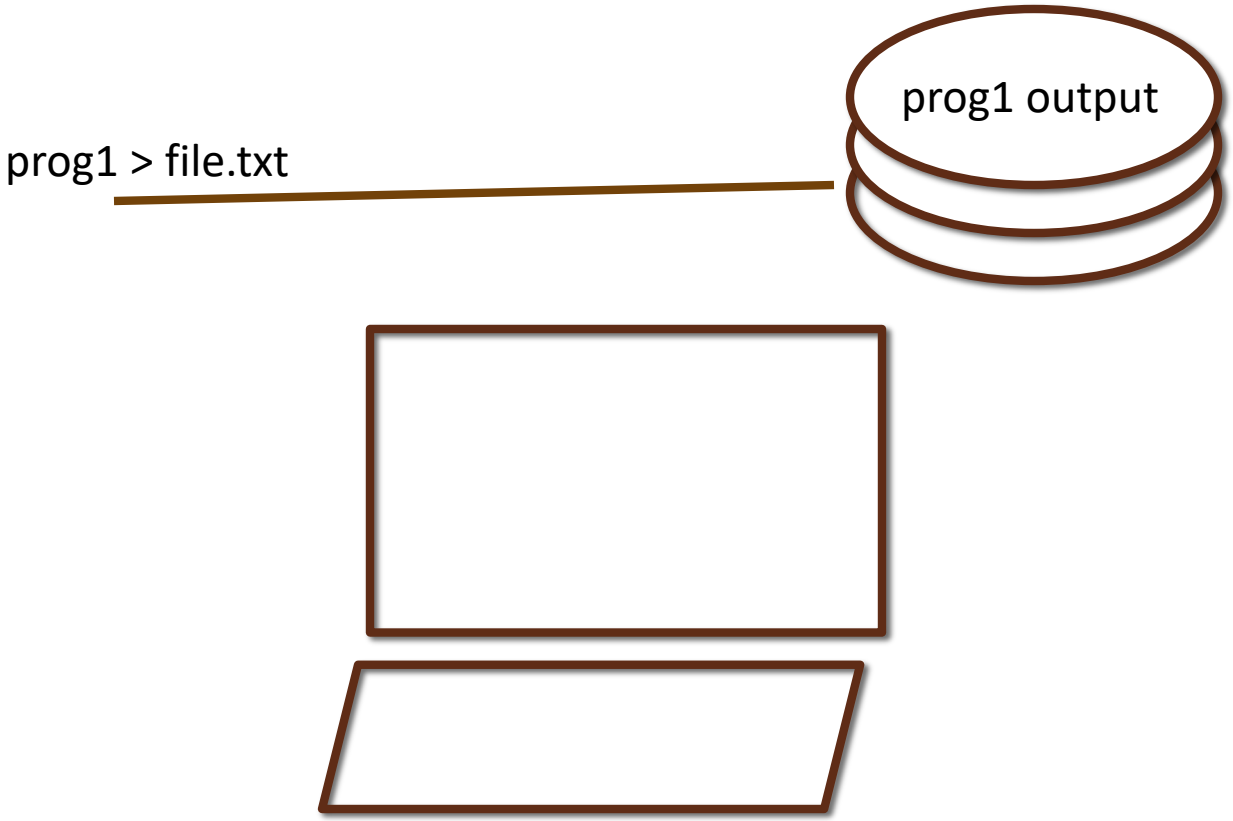
Pipes



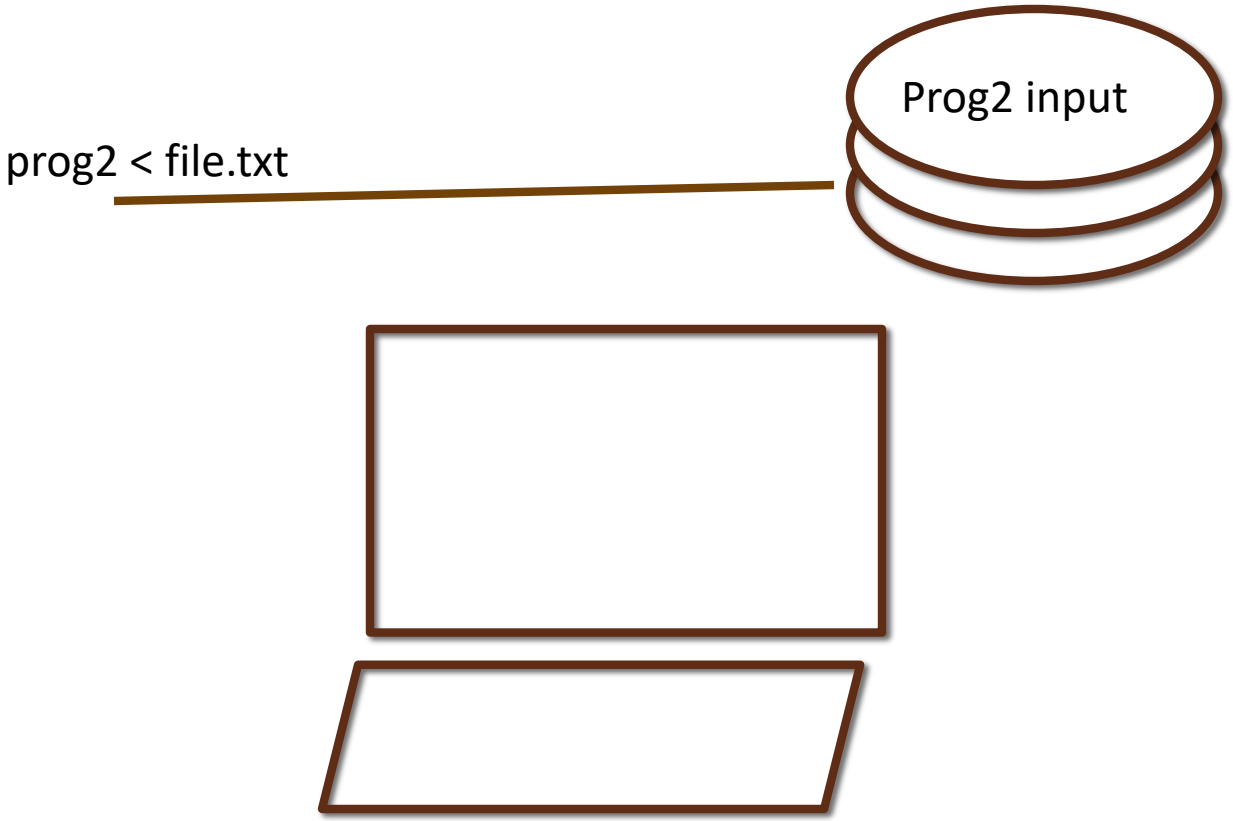
Pipes



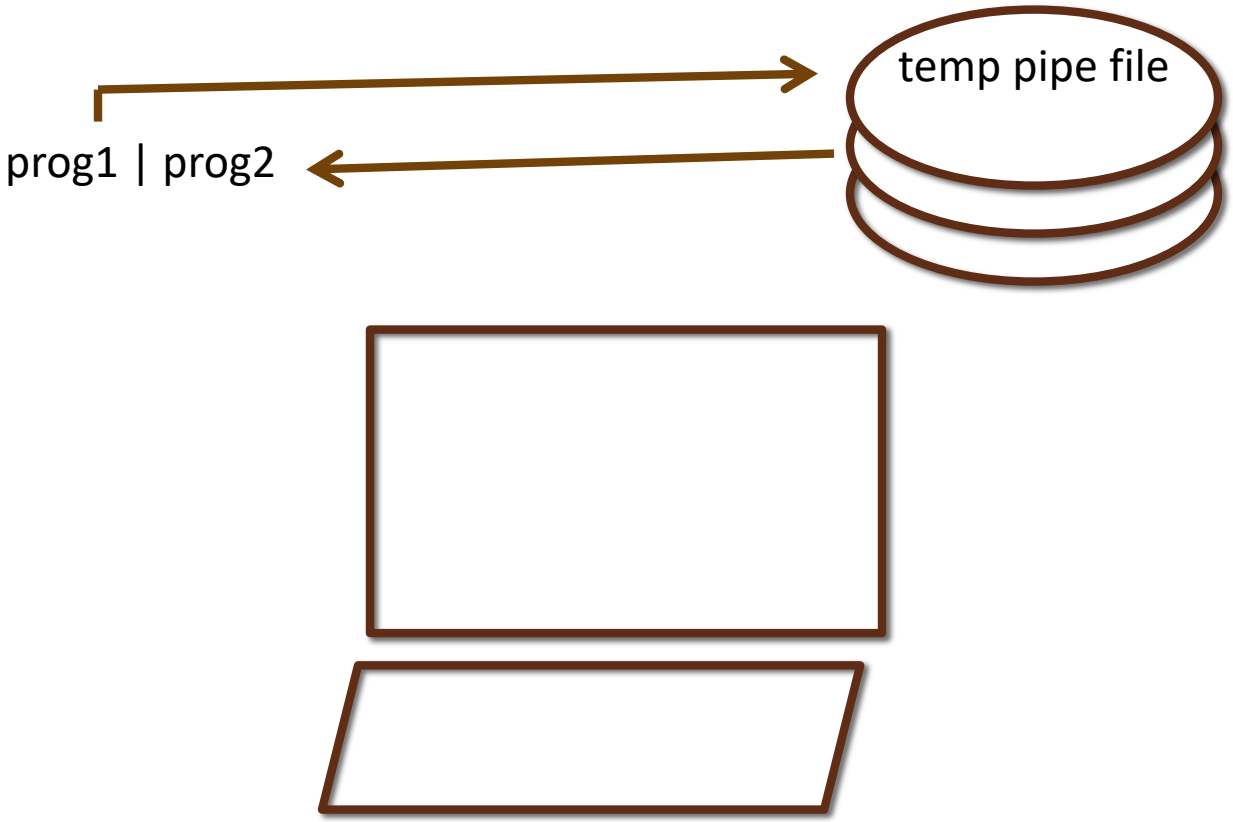
Pipes



Pipes



Pipes



Activity

Given:

- ps –aef # returns process information for all processes on the system
- grep <text> # searches for the string <text>

Pipe the output of the ps command into grep and try to find:

- All the processes owned by yourself
- All the processes owned by ‘root’
- All the processes running an application called ‘ypbind’

Try using I/O redirection to store the output into a file

Try using a pipe to paginate the output so it doesn’t scroll faster than you can read

Questions?