

# CSE320 System Fundamentals II

---

TONY MIONE

# Acknowledgements

---

Some slides provided by Dr Yoon Seok Yang

# 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

# What is UNIX?

---

Unix -- stands for UNiplexed Information Computing System -- is a multiuser, multitasking operating system (OS) designed for flexibility and adaptability. Originally developed in the 1970s, Unix was one of the first OSes to be written in the C programming language.

In the late 1960s, Bell Labs (later AT&T), General Electric and the Massachusetts Institute of Technology attempted to develop an interactive time-sharing system called Multiplexed Information and Computing Service ([Multics](#)) that would enable multiple users to access a mainframe simultaneously.

Disappointed with the results, Bell Labs pulled out of the project, but Bell computer scientists Ken Thompson and Dennis Ritchie continued their work, which culminated in the development of the Unix OS. As part of this effort, Thompson and Ritchie recruited other Bell Labs researchers, and together, they built a suite of components that provided a foundation for the operating system. The components included a hierarchical file system, a command-line interface ([CLI](#)) and multiple small utility programs. The OS also brought with it the concepts of computer processes and device files.

\* <https://www.techtarget.com/searchdatacenter/definition/Unix>

# What is UNIX?

---

Prior to 1973, Unix was written in assembler language, but the fourth edition was rewritten in C. In the late 1970s and early '80s, Unix amassed a strong following in academia, which led commercial startups, such as Solaris Technologies and Sequent, to adopt it on a larger scale. Between 1977 and 1995, the Computer Systems Research Group at the University of California, Berkeley developed Berkeley Software Distribution ([BSD](#)), one of the earliest Unix distributions and the foundation for several other Unix spinoffs.

In 1991, Linus Torvalds, a student at the University of Helsinki, created a Unix-based OS for his PC. He would later call his project Linux and make it available as a free download, which led to the growing popularity of Unix-like systems. Today, a variety of modern servers, workstations, mobile devices and embedded systems are driven by Unix-based OSes, including macOS computers and Android mobile devices.

\* <https://www.techtarget.com/searchdatacenter/definition/Unix>

# UNIX OS

---

Unix consists of 3 components:

- **A Shell** that acts as an interface between the kernel and the user. Every user has to undergo authentication checks before they are allowed to access a shell.
- The complete shell constitutes either a file or a **program**. A program, under execution, features a unique process identifier (PID) that is also used to identify it.
- **The Kernel** is responsible for time and memory allocation given to programs. It is also responsible for file storage, communication, and responding to the system calls.

# Features of UNIX

---

**Portable:** The UNIX system is written using a High level language, so it is easy to understand, make modifications and transfer it to other machines. This feature of the UNIX OS enables the user to not only modify, but also to store the code on a new machine.

**Multitasking:** More than one process can be operated at the same time. That is, if a process is functioning, then another process can also be run in the background.

**Shell:** A simple interface that allows the user to complete certain tasks. Meanwhile, the Shell hides all the intricacies of the hardware and low level kernel software from the user.

**Library is extensive:** The OS has the support of an extensive library that makes the OS very useful.

# Pros of UNIX

---

- Portability that allows for the OS to function on multiple devices
- Multiple tasks are functioning simultaneously with minimal usage of physical memory
- Complete complex tasks efficiently
- Hierarchical File system is supported, and it helps organize files for easy maintenance and efficiency
- Secure UNIX System due to strong validation and authentication



# Cons of UNIX

---

- Command line based UNIX OS makes it difficult for common users to operate due to a steep learning curve
- Only programmers who understand the command line commands thoroughly can run it
- Documentation for the several UNIX tools is not that easily accessible
- Cryptic commands are applied using special characters, thus making it difficult for new users to comprehend

# 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)

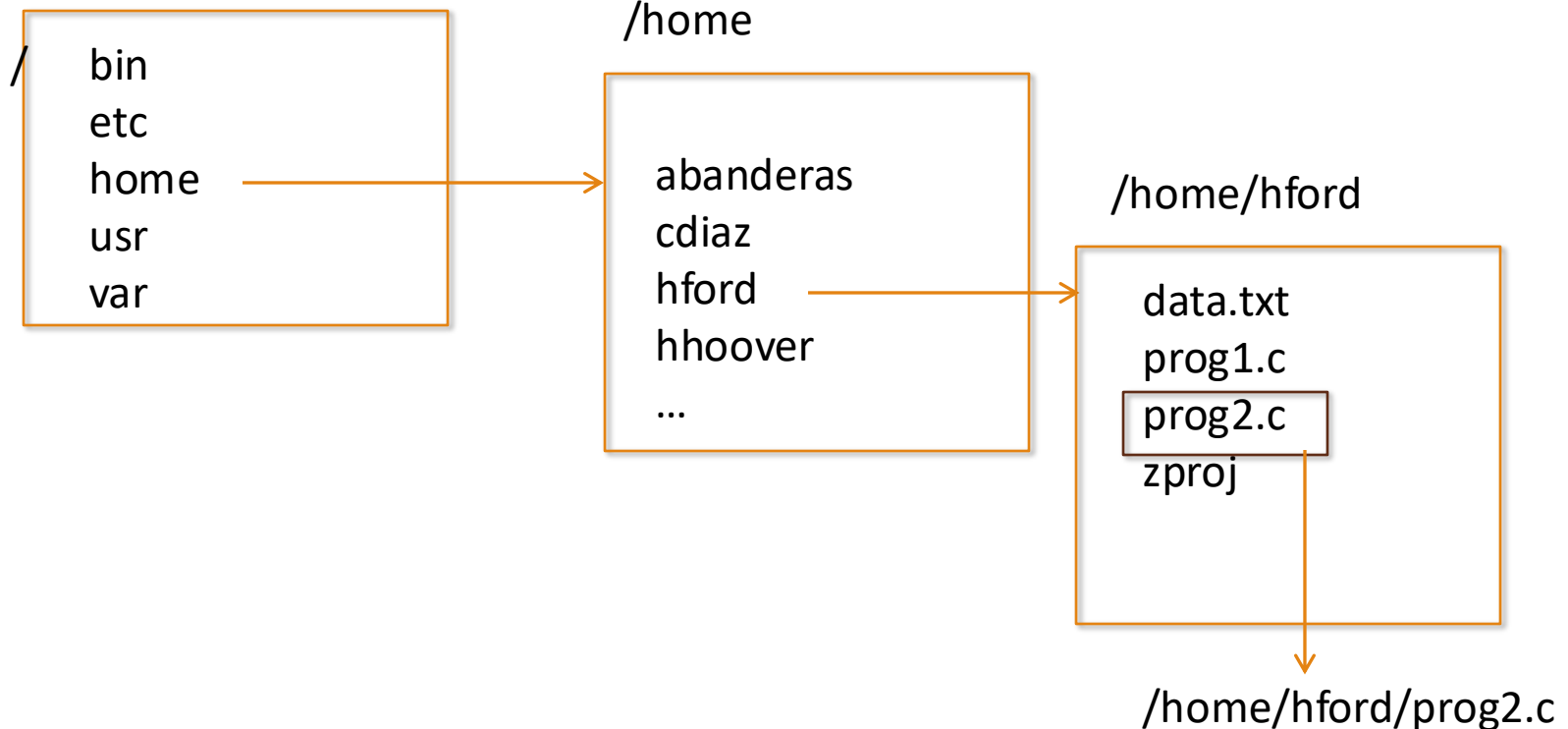
- `script` – Saves to the file *typescript*
- `script filename` – Saves to the *filename* provided by user

# The unix filesystem

A hierarchical collection of directories and files

The *root* of the filesystem is '/'

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



# 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'.

## *rm -r <dirname>*

- This will remove directories and their contents recursively
- Note: *rm -rf <dirname>* does the same but does NOT confirm before taking the action (-f => force). This command should be used with extreme caution!

# Displaying file contents

---

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

- Sends *<filename>*'s contents to terminal

**less** *<filename>*

- Sends *<filename>*'s contents to terminal with pauses when screen is full
  - *<spacebar>* will continue listing
  - *'q'* will terminate the output
- **more** command does the same thing on many (older) 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 #)*

- *Examples:*
  - *tail -400 # print last 400 lines of a file.*
  - *tail -25 # print last 25 lines of a file.*

# 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*

# Find, grep, du, and top commands

---

***find*** <directory: where to start search from> <expression> [options] <what to find>

- ***find*** ./cfgdir -name “\*cfg\*”

***grep*** [options] <pattern> <files>

- Searches for the pattern in files
- ***grep*** -r Yoon ./\*

***du*** [options] [directory/file]

- Run this in your home directory to see how much space you are using
- ***du*** -hs ./

***top***

- Displays all the processes running on the machine, and shows available resources
- Many options => ordering by cpu usage (default). Use -o <key> to change order (mem, threads, etc.)

# 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 (kernel) Calls
    - 3: C Library Functions
    - 4: Devices and Special Files
    - 5: File Formats and Conventions
    - 6: Games
    - 7: Miscellaneous
    - 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 with titles or descriptions that match the <text> argument
- 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

- **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

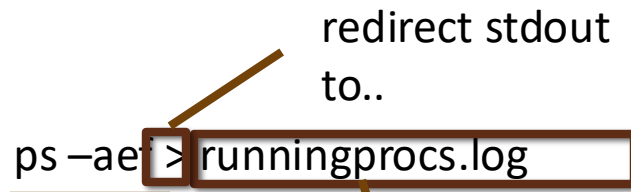
Most shells use:

- ‘<’ to redirect stdin [read from a file]
- ‘>’ to redirect stdout [write to a file]
- ‘>>’ redirect stdout and append [append a file]
- ‘2>’ to redirect stderr
- ‘|’ connects output of one command to input of the next

# I/O Redirection examples

---

`ps -aef > runningprocs.log`



redirect stdout  
to..

**IMPORTANT:** Note that this entirely overwrites `runningprocs.log`. If you were to do this a second time, the results of the first `ps` command would be lost!

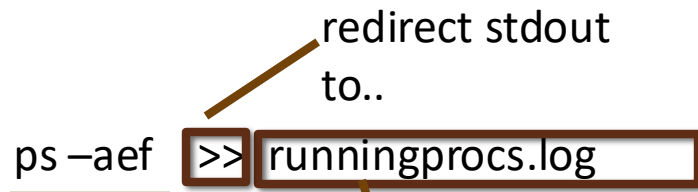
Redirect command output to a file called *runningprocs.log* in the current working directory

Unix process status command with options

# I/O Redirection examples

---

`ps -aef` `>>` `runningprocs.log`



redirect stdout  
to..

Redirect command output to a file called  
*runningprocs.log* in the current working directory

Unix process status command with options

**IMPORTANT:** Note that here we are appending to the file `runningprocs.log`. If you were to do this a second time, the results of the first `ps` command would still be in the file and the new output would be written after that!

# Pipes

---

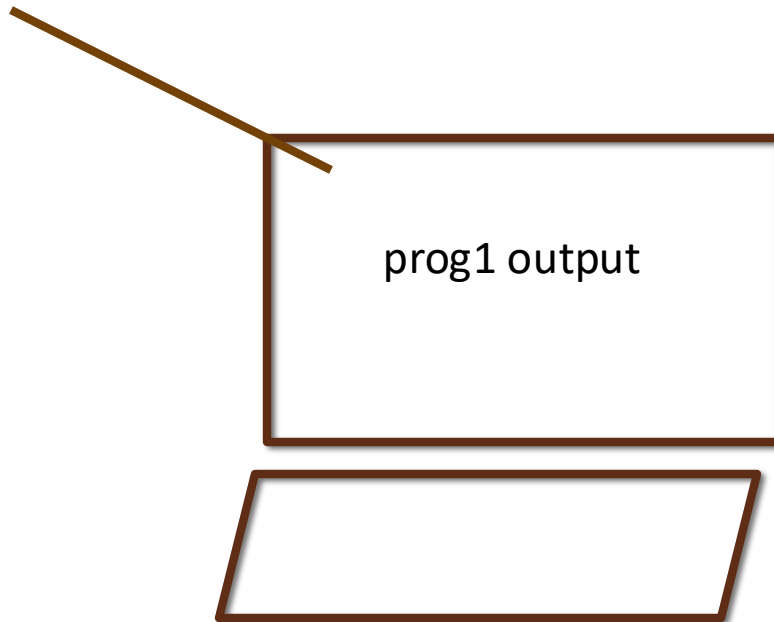
Redirect output from one command/program directly to the input of another

Use '|' between programs or commands

# Pipes

---

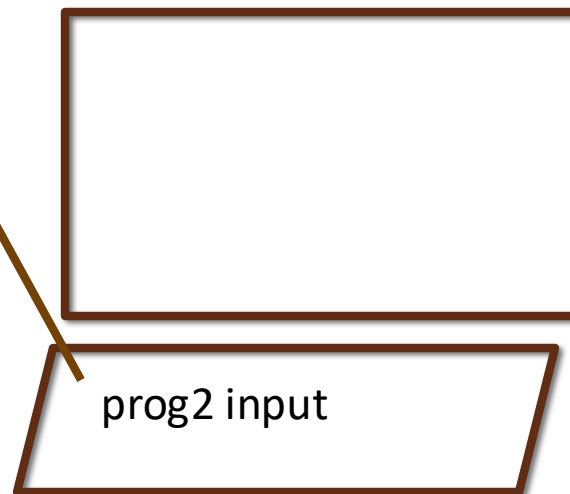
prog1



# Pipes

---

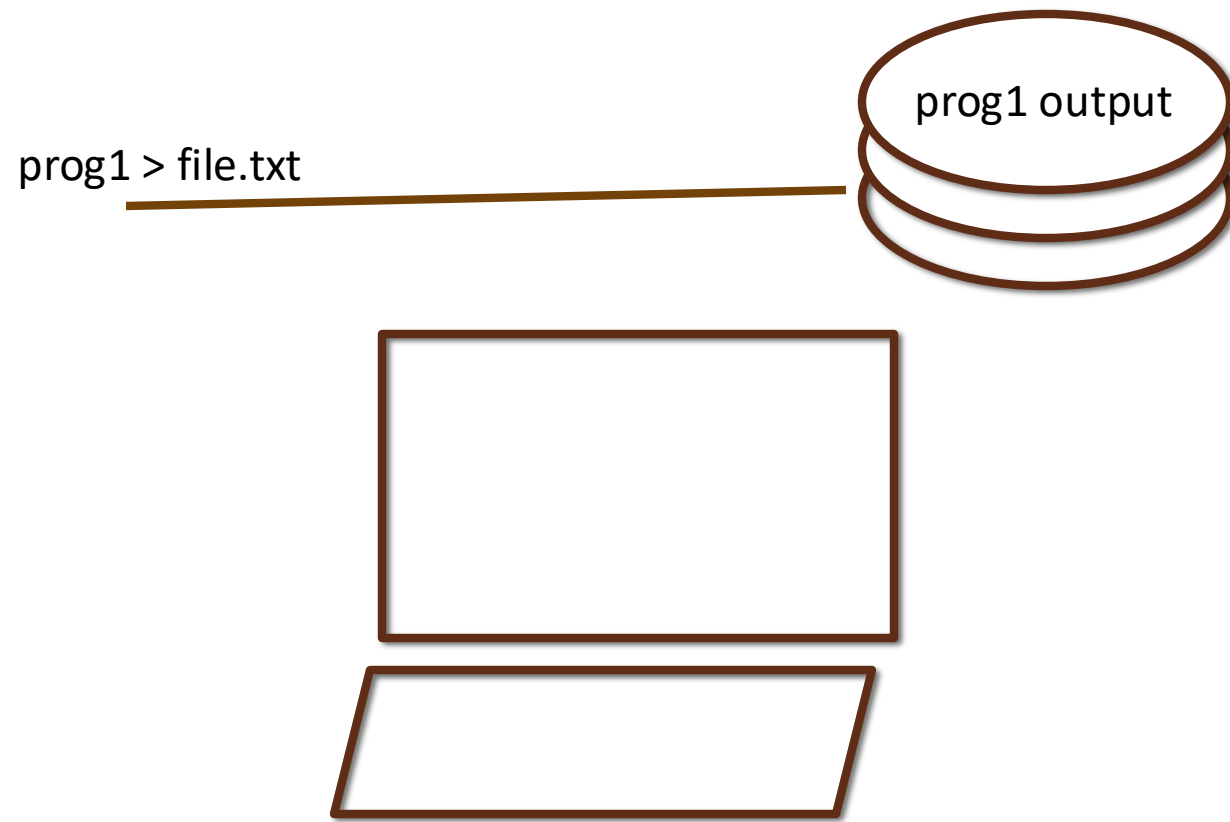
prog2



# Pipes

---

prog1 > file.txt



The diagram illustrates the execution of the command 'prog1 > file.txt'. A horizontal line connects the command to a stack of three ovals labeled 'prog1 output'. Below this, there are two empty shapes: a rectangle and a parallelogram, representing the output file and its alternative representation.

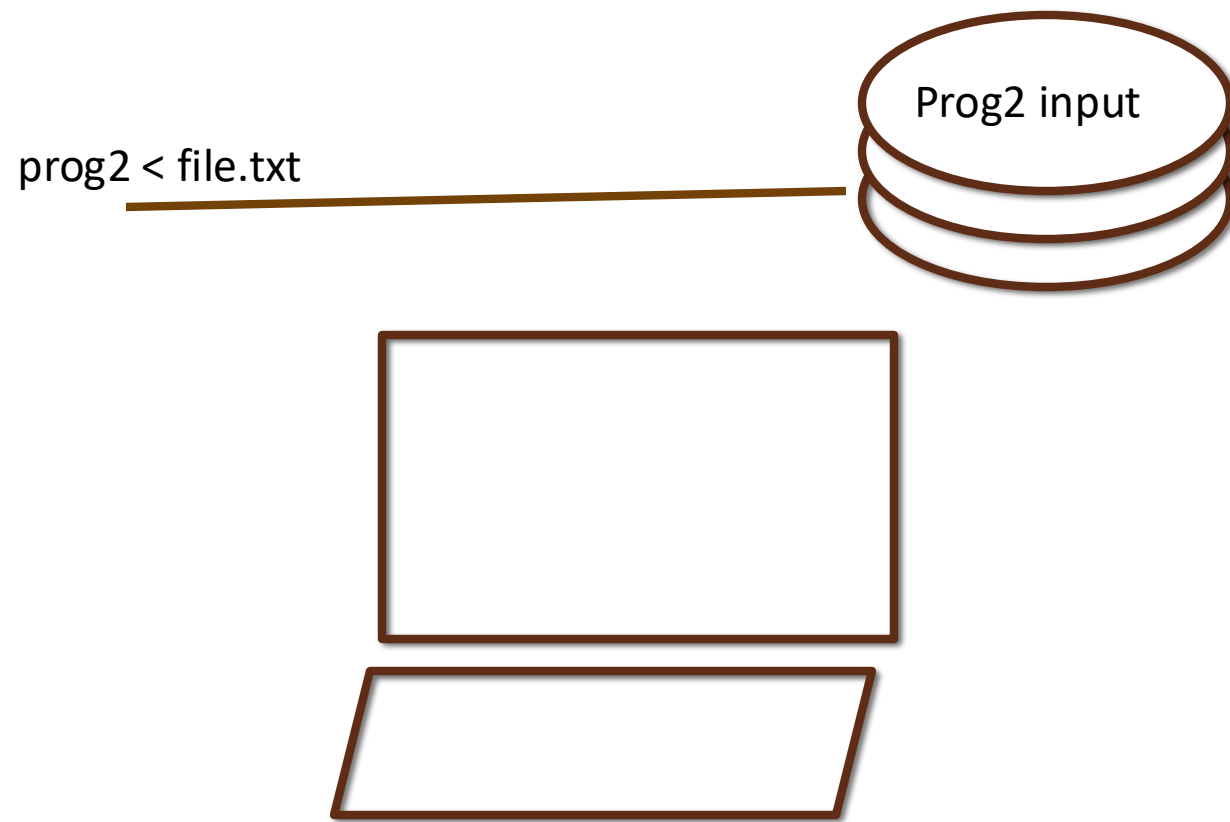
prog1 output



# Pipes

---

prog2 < file.txt

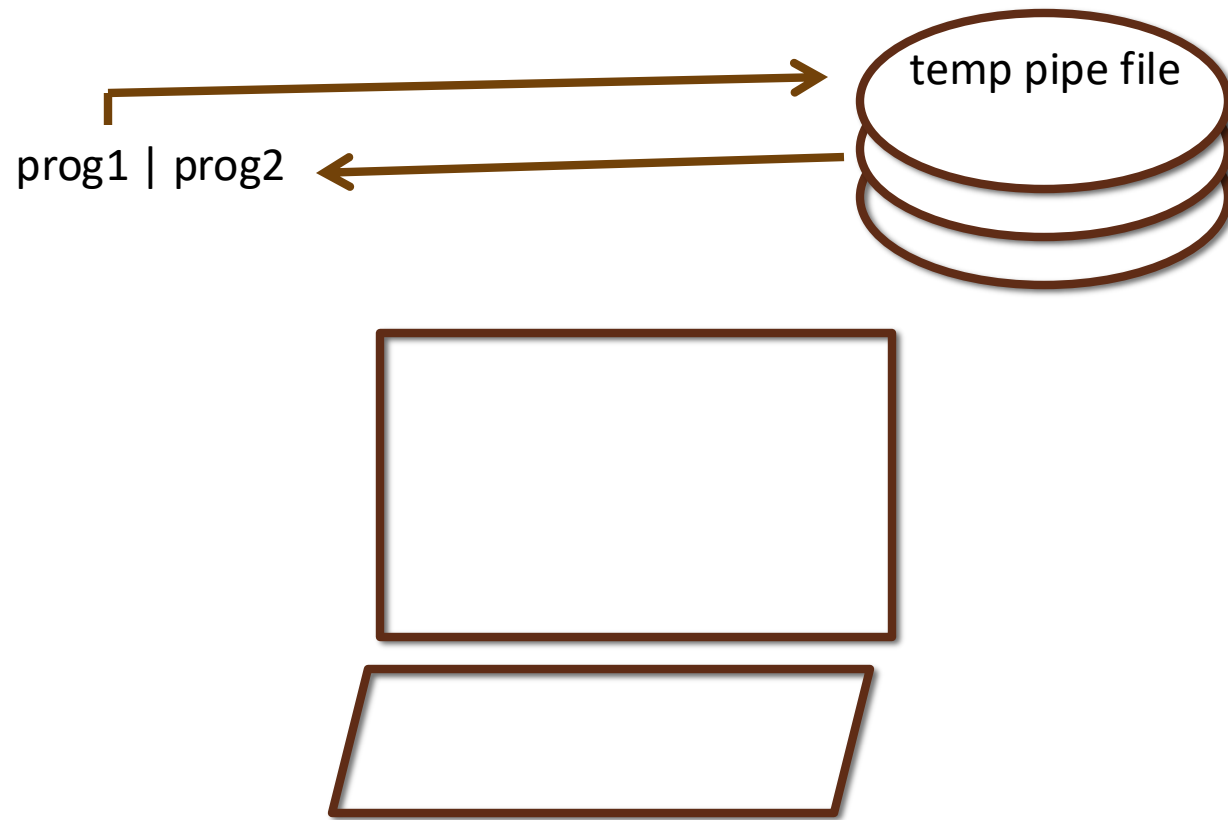


The diagram illustrates the execution of the command `prog2 < file.txt`. A horizontal line connects the command text to a stack of three ovals. The top oval is labeled "Prog2 input". Below this stack are two empty shapes: a rectangle and a parallelogram, representing the output of the program.

Prog2 input

# Pipes

---



---

# Questions?