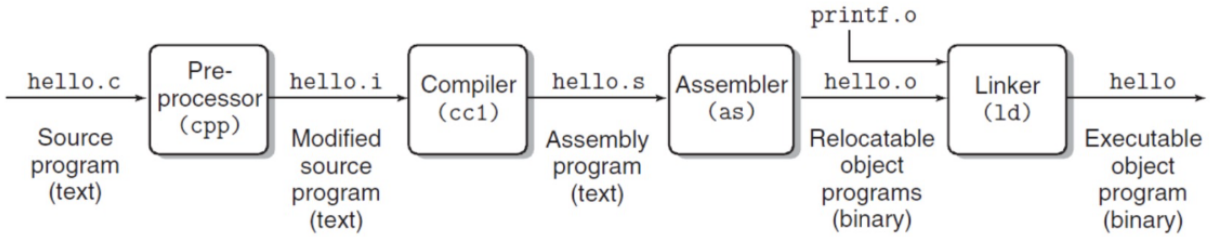
C Programming & Basic Unix

Compilation in C Programming

gcc: GNU Complier Collection (Compiler for C programming language)




```
gcc hello.c
gcc -E hello.c -o hello.i
gcc -S hello.c
gcc hello.s

-ggdb (add debug information to help gdb)
-O# (set 'optimization level of colde, #=0-3 (use 0 for debug)

gcc –c –O0 –ggdb –o bigprocmain.o bigprocmain.c
gcc –c –O0 –ggdb –o bigprocutils.o bigprocutils.c
gcc -ggdb –o bigprog bigprocmain.o bigprocutils.o
```

1) Pre-processor (cpp): it processes source file's #include and #define part → '.i' file (not saved in the disk)

2) Compiler (cc1): it creates a '.s' assembly program from a '.i' file. (not saved in the disk)

3) Assembler (as): it creates a '.o' object file from a '.s' file.

4) Linker (ld): it connects library functions and object files

(link: http://seamless.tistory.com/2)

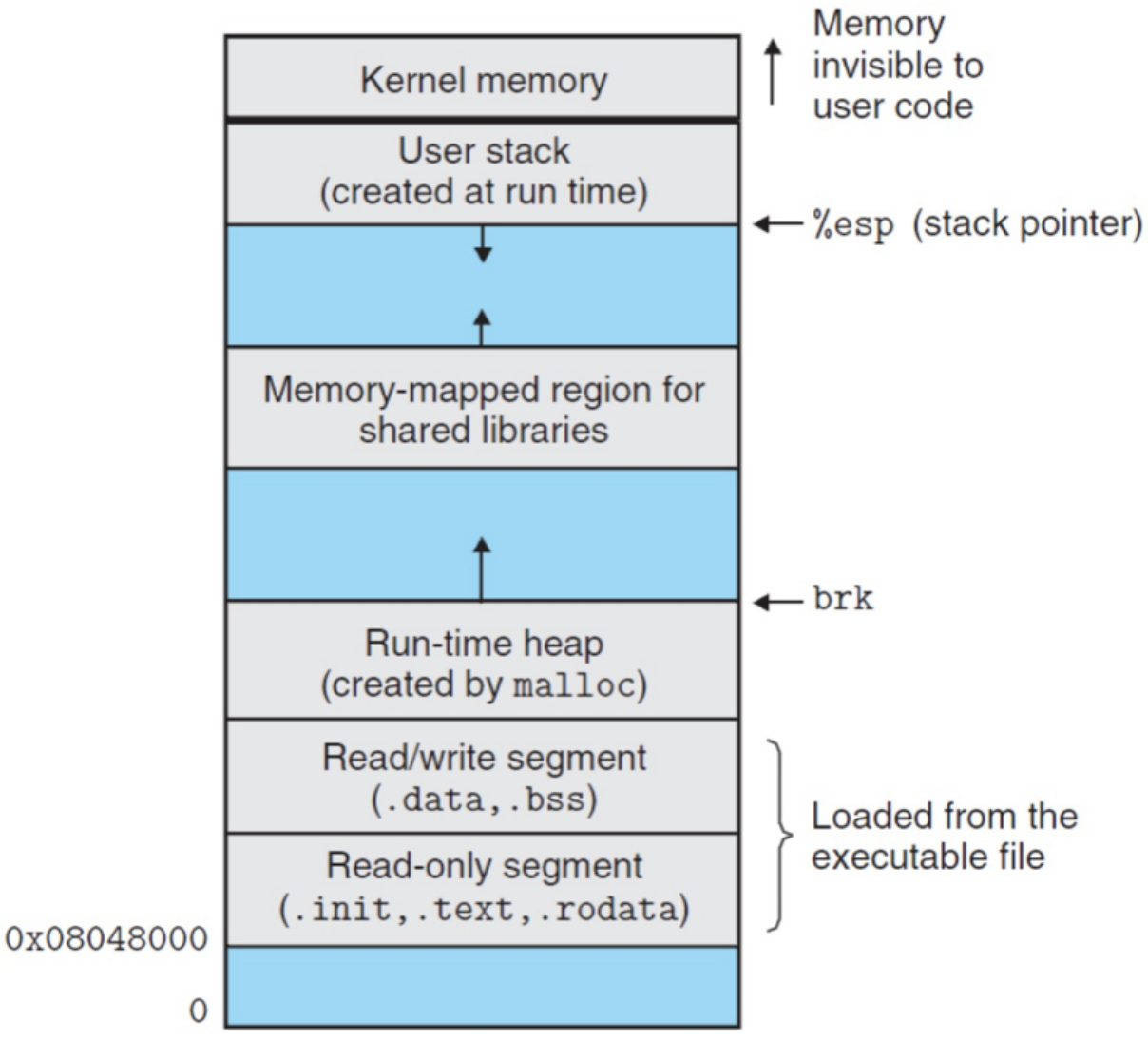
-S: generate an assembly file

-c: do not link

-O0: no optimization (will make the generated assembler code easier to follow)

-fverbose -asm: add verbose comments

C Programming & Run-time Environments



(Variables, Flow control, Function calls, Structures, Pointers, Dynamic memory allocation, Runtime env.)

Next Chapters

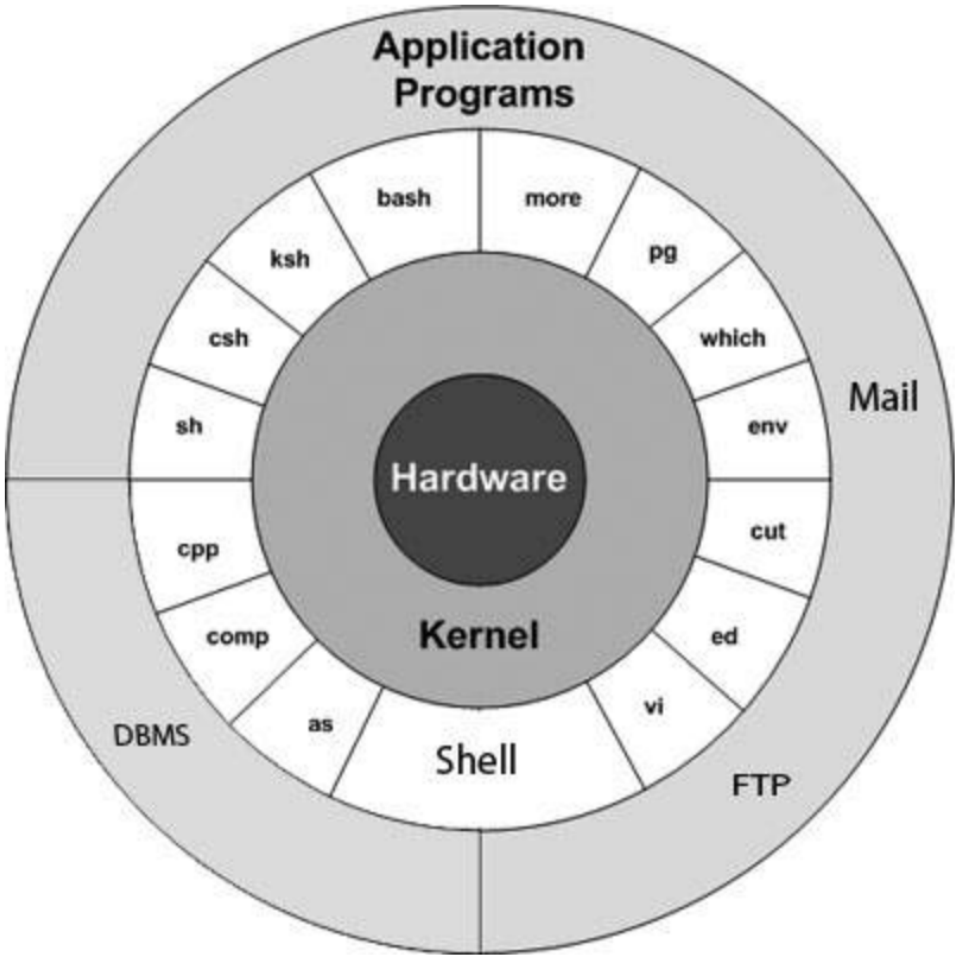
→ Assembly Language, System Calls, Threads and Locks, Performance, Memory Hierarchy, Locality, Virtual memory, Memory Mapping and Linking

Basic Unix Commands

Unix is a multiuser, multi-tasking Operating System (OS) written in C programming language.

- Time Sharing System
- Tree-structural file system

<Unix Architecture>

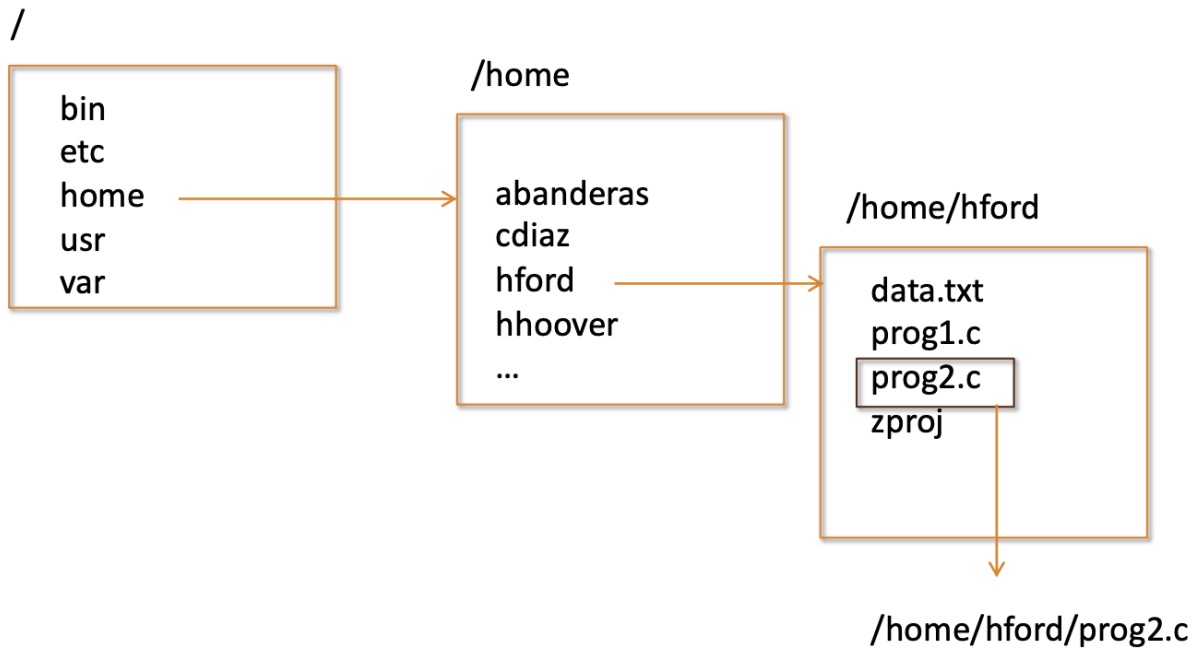


<Unix Commands (cont')>

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

Commands provide information on files, manipulate files and data from files, and run arbitrary utilities available on the system.

The unix filesystem has '/' for the root of the filesystem. It separates a series of directories which a path contains.



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

pwd: print working directory (link: https://phoenixnap.com/kb/pwd-linux)

cd <new_working_directory>: connect to a new working directory ['-' indicates the last working directory]

('.' is for current directory and '..' is for parent 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

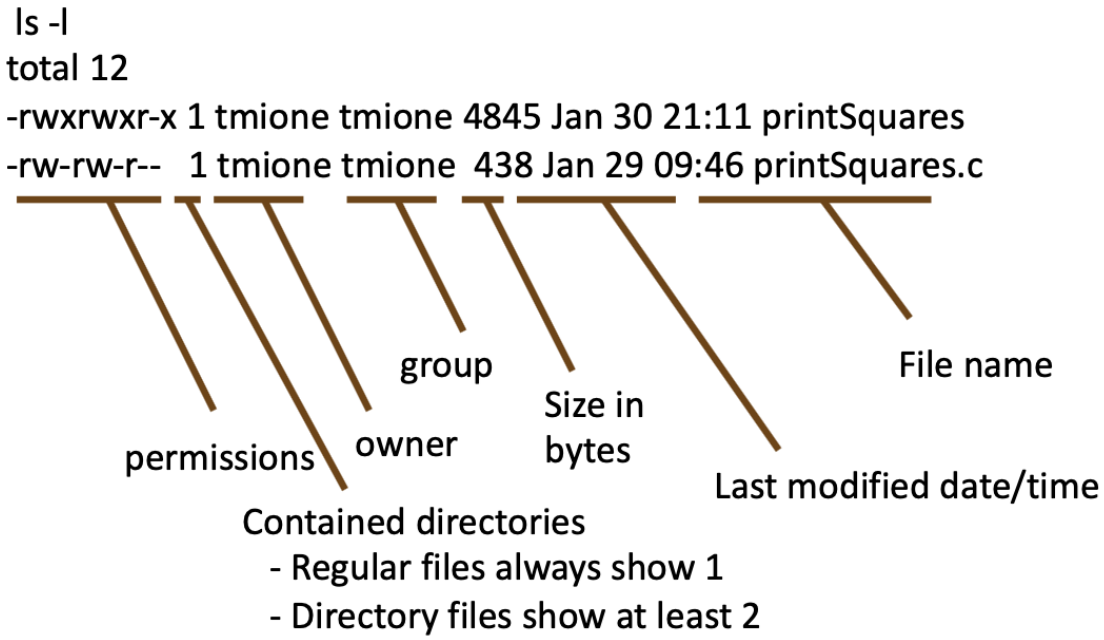
- Absolute: specifies all directories from '/' to desired directory (begins with '/')

- Relative: specifies the directory relative to current working directory (begins with a name or '..')

ls [options] <path>: list directory

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

ls —l: lists files in a directory with extra information ('long')



(Exercise) Connect to /repository/common and use 'ls —l' to look at detailed information on files in the directory

- list owner accounts for various files
- list some of the 'groups' owning files
- are any protected against group and/or world access
- are there any very large files or very small files?
- what is the oldest file in the directory

```
cd /repository/common
ls --l
```

<File ‘globbing’>

- ‘*’ match any string of 0 or more characters (ex) ls *.c _ all files with a .c extension
- ‘?’ match any single character (ex) ls notes??.txt _ files starting with the word ‘notes’ and ending with two other characters
- ‘[<characters>]’ is any character in the list within [] (ex) ls [a-c]*.txt _ all files beginning with ‘a’, ‘b’, or ‘c’ and contain only ‘a-c’

(ex) ls */*.c _ match all files with a .c extension in a directory one level down from the connected directory

<Activity>

1. connect to /usr/local/bin
2. using a relative path, connect to /usr/include
3. list files with extensions of .h
4. list files with extensions of .h in all directories below your current working directory
5. connect to your home directory once again
6. type ^D to exit the script application and stop recording

```
cd /usr/local/bin
cd ../../include
ls *.h
ls ./*.h
cd ~
^D
```

touch <filename>: creates an empty file with the given name

- any editor such as vi or emacs can be used to create a file

mkdir [options] <dirname>

- creates a directory
- -p: create the whole path to the directory
- (ex) mkdir -p project/src/include

rm <filename>: delete the named file(s)

rmdir <dirname>: deletes the named directory(ies), directory must be empty (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'.)

(link: https://withcoding.com/111)

cat <filename>: 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 the Unix Systems

head -# <filename>: prints the first few lines of a file

tail -# <filename>: prints the last few lines of a file

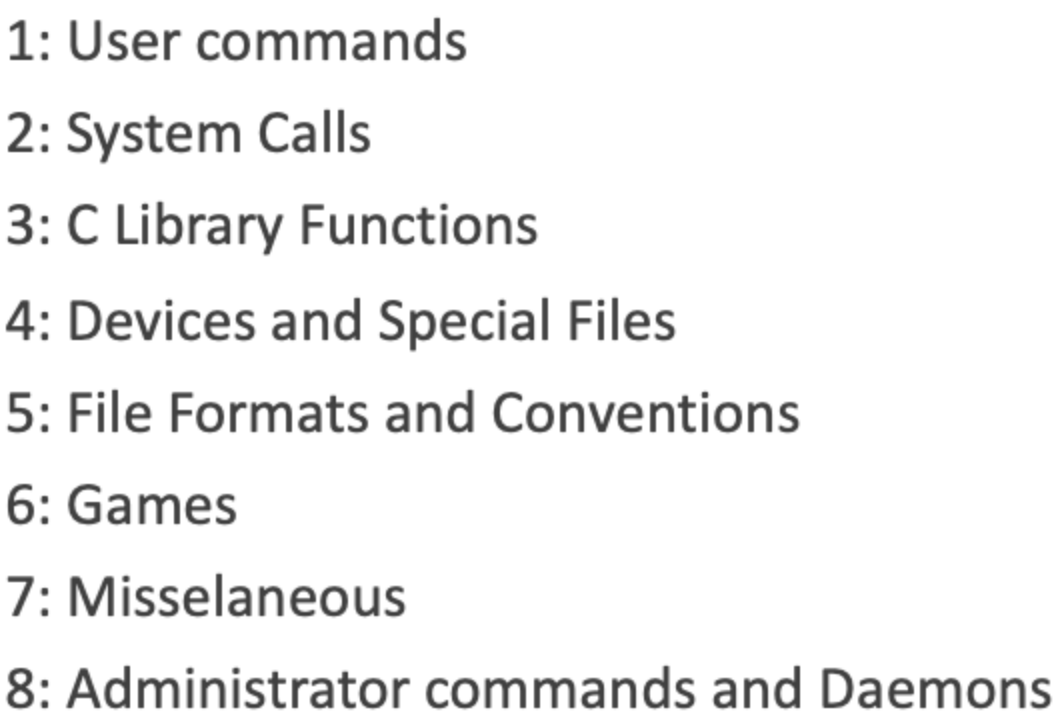
cp <oldname> <newname>: copy file <oldname> to <newname>. File names can include absolute or relative paths

mv <oldname> <newname>: move or 'rename' a file from <oldname> to <newname>. Filenames can include absolute or relative paths

(link: https://withcoding.com/90)

man <section> <text>

- displays a manual page about the command or program <text>
- <section> is optional (1-8), which specifies the ‘section’ of the manual (names may appear in 2+ sections



(link: https://gregorio78.tistory.com/254)

apropos <text>

- searches man page titles and descriptions for given <text>
- lists man entries that match the text
- same as ‘man --k’

<Learning more>

(link: https://m.blog.naver.com/jjjhygo91/221649634916)

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

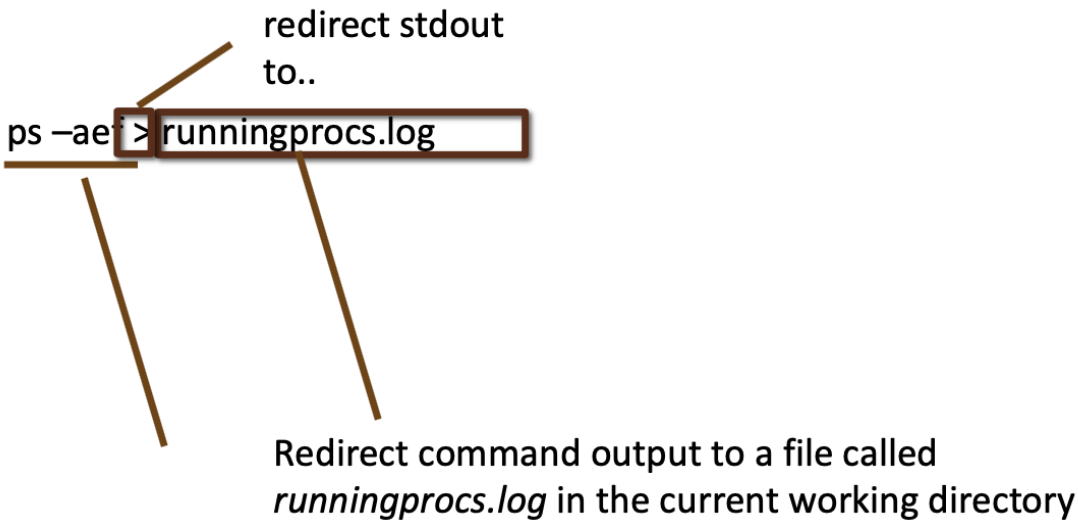
Shells allow command output and input to be 'redirected' with special characters

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

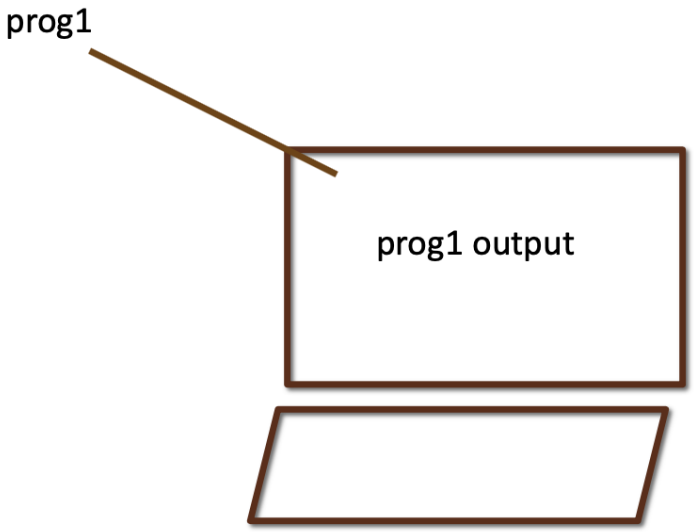


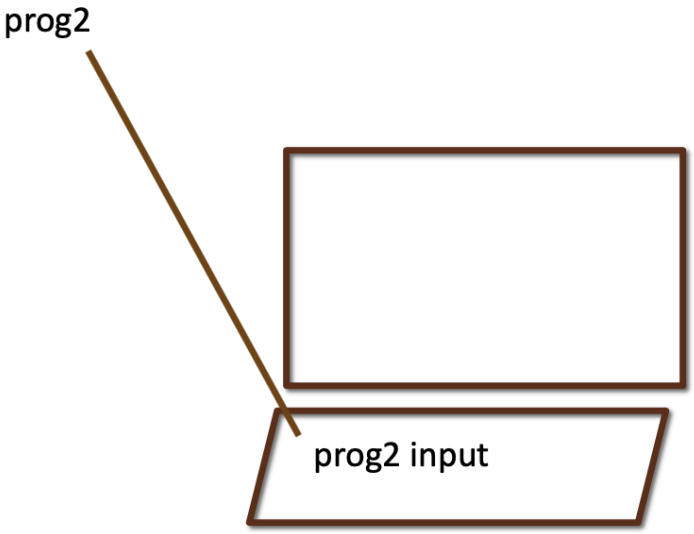
(link: https://www.geeksforgeeks.org/piping-in-unix-or-linux/)

Pipes

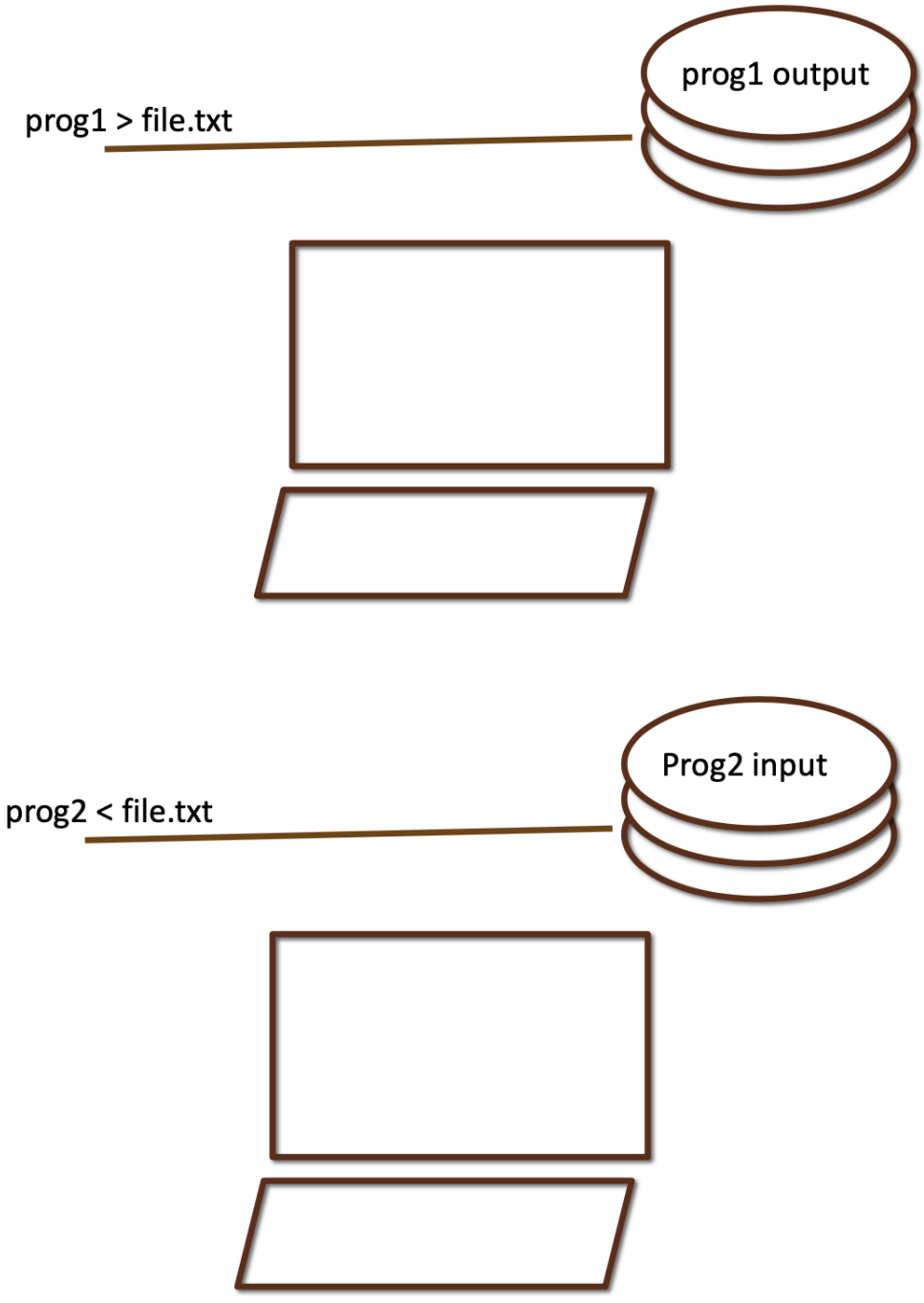
Redirect output from one command/program directly to the input of another

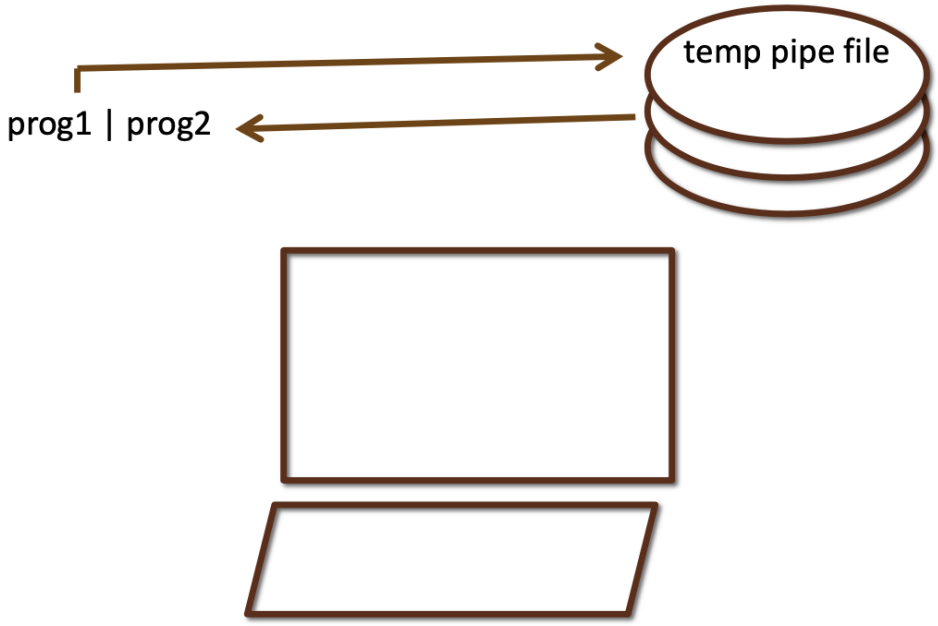
Use '|' between programs or commands


prog1
prog1 output


prog2
prog2 input

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

<Command Options>

Unix options communicate specific customizations to a program or command

- '-' followed by a single character
- '-a -e -f' can be specified 'aef'
- ordering after '-' doesn't matter

- '—' follwed by a string (it is used to avoid confusion with multiple other options)
- both option types may take an 'argument'. Argument must be next on the command line (i.e. -o -filename) and sometimes '=' is used between the option and argument)

<Environment Variables>

Environment Variables store information about the user and process and help provide needed context to applications

Important variables:

- HOME – This is the user's home directory
- PATH – This contains a list of directories to be searched for commands and programs. Directories are ':' separated
- USER – This contains the name of the logged in user
- PWD – This holds the current working directory. The value can be changed but doing so will not affect the working directory.

Manipulating Environment Variables

printenv
- Prints the values of all environment variables

echo ***$VARNAME***
- Displays contents of VARNAME

unset ***VARNAME***
- Deletes the named environment variable

export ***VARNAME='string'***
- Sets VARNAME to the provided value

<More Unix Commands>

→ Process Status and Control

ps [options]: provides information about running processes

^C

- stops running program
- process is deleted
- process resources (memory, devices) are freed

^Z

- suspends running program
- process and resources are still present

jobs

- lists the processes in the immediate 'process tree'

- are shown as [jobnum]<+/-> command (where job number is a small integer assigned by the OS. + indicates it is the top backround job that will be pulled to foreground with ‘fg’. ‘command’ shows the entire command line originally used to start the job)
- -l option adds process id to displayed information

bg <jobnum>: continues running program indicated by <jobnum> in the background. keyboard commands do not affect background process

fg <jobnum>: continues running program indicated by <jobnum> in the foreground

kill <pid>: stops program running in process indicated by process id <pid> and deletes process. Use -9 option if kill does not work

<File ownership>

chown <username> <filename>: changes the owner of the file to the user

chgrp <groupname> <filename>: changes the group of files to the group

file permissions (9 characters)

- Format : tuuugggooo
  - ‘t’ is type (‘-’=file, ‘d’=directory, ‘b’=block device, ‘c’=char device)
  - Each triplet represents read,write,execute (rwx) permissions for a user or subset of users
    - uuu – ‘owning user’
    - ggg – ‘group’ permissions
    - ooo – ‘other’ permissions
  - ‘-’ in a position means ‘not permitted’

chmod (Change Mode): changes file permissions

- chmod <perms> <filename>
- <perms> can be given
  - As a combination of sets and permissions:
    - o+x – Add execution permission for ‘others’
    - o-rwx – Remove read, write, and execute from ‘others’
  - As a 3 digit octal number where each octal digit is the permissions for a class of user:
    - 700 – Provide read, write, and execute for the owning user, no permissions for anyone else
    - 660 – Provide read and write for the owning user and group. No access is provided for others

File permissions examples

drwxr-xr-x
  Directory file
  owner can read, write, and execute (search)
  group can read and execute (search)
  ‘other’ users can read and execute (search)

-rwxrw-rw-
  regular file
  owner can read, write, and execute
  group can read and write (but not execute)
  ‘other’ users can read and write (but not execute)

(link: https://coding-factory.tistory.com/804)

Locating files

'ls' is not good at locating files in a large directory subtree

'find' searches a subtree for files with matching criteria

find ***<top_path>*** *[options] [expression]*

- <top_path> should be the top directory where the search will start
- [options] – Provide details to restrict scope of search (maxdepth, mindepth, etc)
- [expression] – These provide search parameters

'find' options

- -maxdepth # - Do not descend past '#' directory levels
- -mindepth # - Do not test for files above '#' directory levels

'find' expression predicates

- -name '<filespec>'
  - <filespec> can contain wildcards (*, ?, [character class], etc.
  - Must be in ''!
- -iname '<filespec>' – like -name but match is case insensitive
- -empty – Empty files (or directories)

‘find’ expression predicates

- -amin # - Files that were
  - # - accessed ‘#’ minutes ago
  - -# - accessed less than ‘#’ minutes ago
  - +# - accessed more than ‘#’ minutes ago
- -cmin # - Files that were changed ‘#’ minutes ago (see –amin)
- -readable, -executable, -writable
- -type ‘t’ – (d=directory, c=char special, b=block special, f=regular file, l=symbolic link, etc)
- -size # - Files that are (>,<,=) ‘#’ units large (Supports +/-)
  - #k - # * 1024
  - #M - # * 1024 * 1024
  - #G - # * 1024 * 1024 * 1024

<Activity>

Locate all directories under /usr/include

Locate executable files in /usr/include that are not directory files

Locate executable files in /usr/bin that are not directory files

Locate files in the /usr/include tree whose names end with ‘ab.h’

<Command History>

Unix will save recent commands

Environment Variables (place definitions in .bashrc)

HISTSIZE: number of commands to save

HISTFILESIZE: number of commands to save across sessions

HISTFILE: file in which to save history

history: lists entire saved command history

!#: recalls command '#' from list

!-#: recalls '#'th previous command

!!: recalls previous command

!<string>: recalls most recent command starting '<string>'

':' after recall command specifies what part of command to recall

^: recall first argument

$: recall last argument

#: recall '#'th argument

n-m: recall 'n'th through 'm'th argument

<Custom Commands>

alias <commandname≥'<command and arguments>'

- creates a command (<commandname>) that translates to the remaining text
- can place these in .bashrc so they are present each time you start a shell

(ex)

• alias direct='ls –l | grep "^d"'
• alias tmpfiles='ls –l *.tmp'
• alias dgcc='gcc –O0 –ggdb'

<Activity>

Create a couple of custom commands with the 'alias' command

Try those commands out and see that they work

Edit your .bashrc file to add the alias commands to it. Doing this will cause those special commands to be defined each time you log in.