CSC / CPE 357

Systems Programming

Lecture Topic

- UNIX Overview
- The Runtime Stack & Memory
- Git

UNIX Terminology

- User
  - Individual user accounts each have their own identity and ownership.
- Shell
  - The shell executes commands and keeps track of your environment
- File System
  - Hierarchical arrangement of directories and files
- Command
  - A program, tool, or script
- Process
  - Instance of a running program

https://homepages.uc.edu/~thomam/Intro_Unix_Text/Glossary.html

UNIX Users

- Administrative superuser: **root**
  - User ID 0
- System-level users
- "Regular" users

The UNIX Shell

The UNIX **shell** allows you to:

- Run programs
- Provide input
- Inspect output

Many different shell implementations:

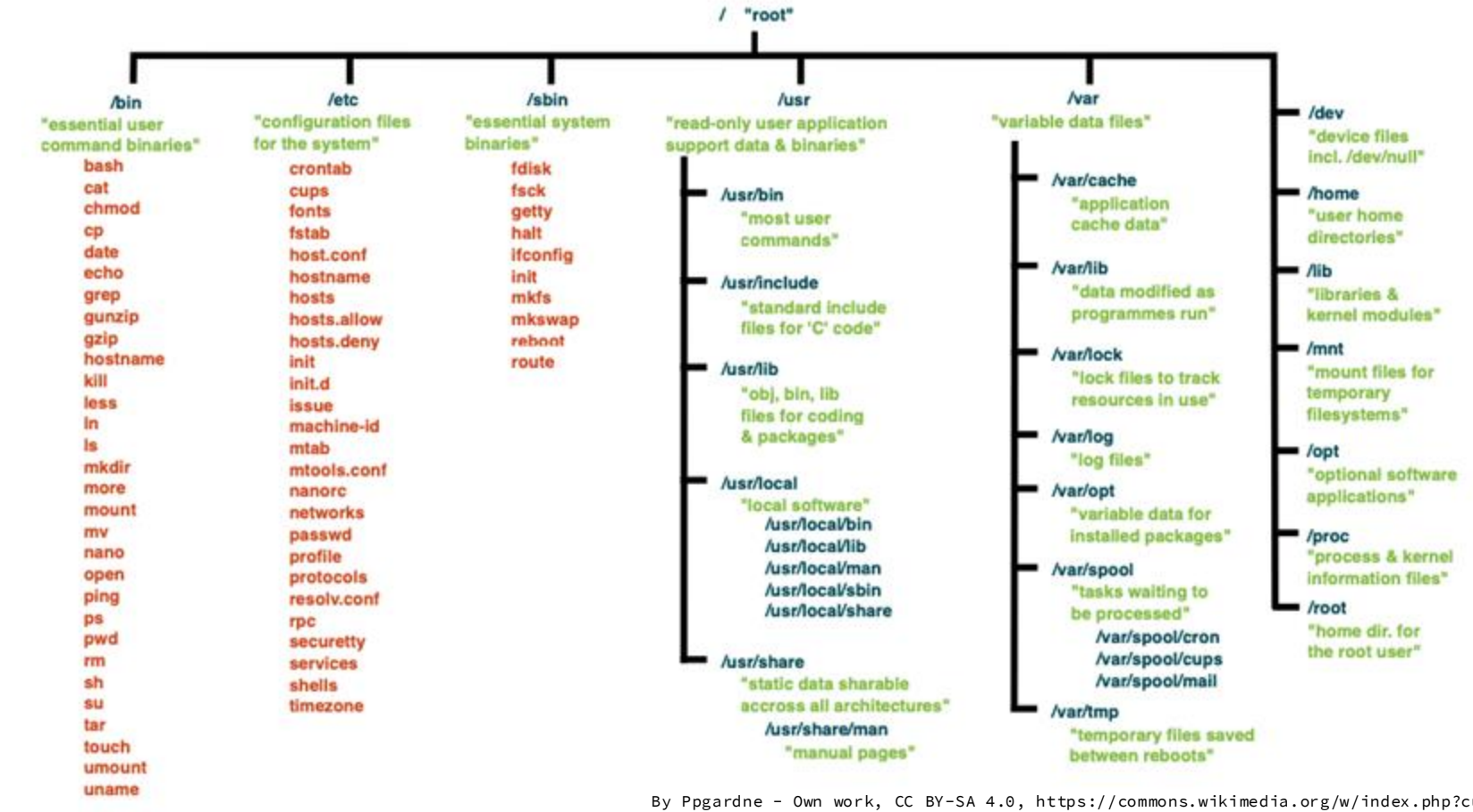
- `sh`
- `bash`
- `zsh`
- `tcsh`

UNIX File System

- The UNIX file system is a hierarchical arrangement of directories and files.

- Everything starts in the directory called **root**, whose name is the single character: **/**

UNIX File System



UNIX Processes

- A **process** is an instance of a running program
  - Contrast with: *program*, an executable file residing on disk in a directory

- Whenever a command is issued in UNIX, it creates/starts a new process.
  - System tracks: user ID, group ID, process ID, working directory

- Record of services/resources used:
  - Memory
  - CPU time
  - I/O streams

Signals

- **Signals** are a technique to notify a process that some condition has occurred.
  - Example: divide by zero / `SIGFPE` (floating point exception)

- Three choices for dealing with signals:
  - Ignore the signal
  - Let the default action occur (for `SIGFPE`, terminate the process)
  - Provide a function that is called when the signal occurs ("catch" the signal)

UNIX Paths

A path is a reference to a file or directory:

```
/usr/local/bin/

/home/fkhan19/csc357/recipes.txt
```

Absolute (starting with `/`) or **relative** to the current working directory.

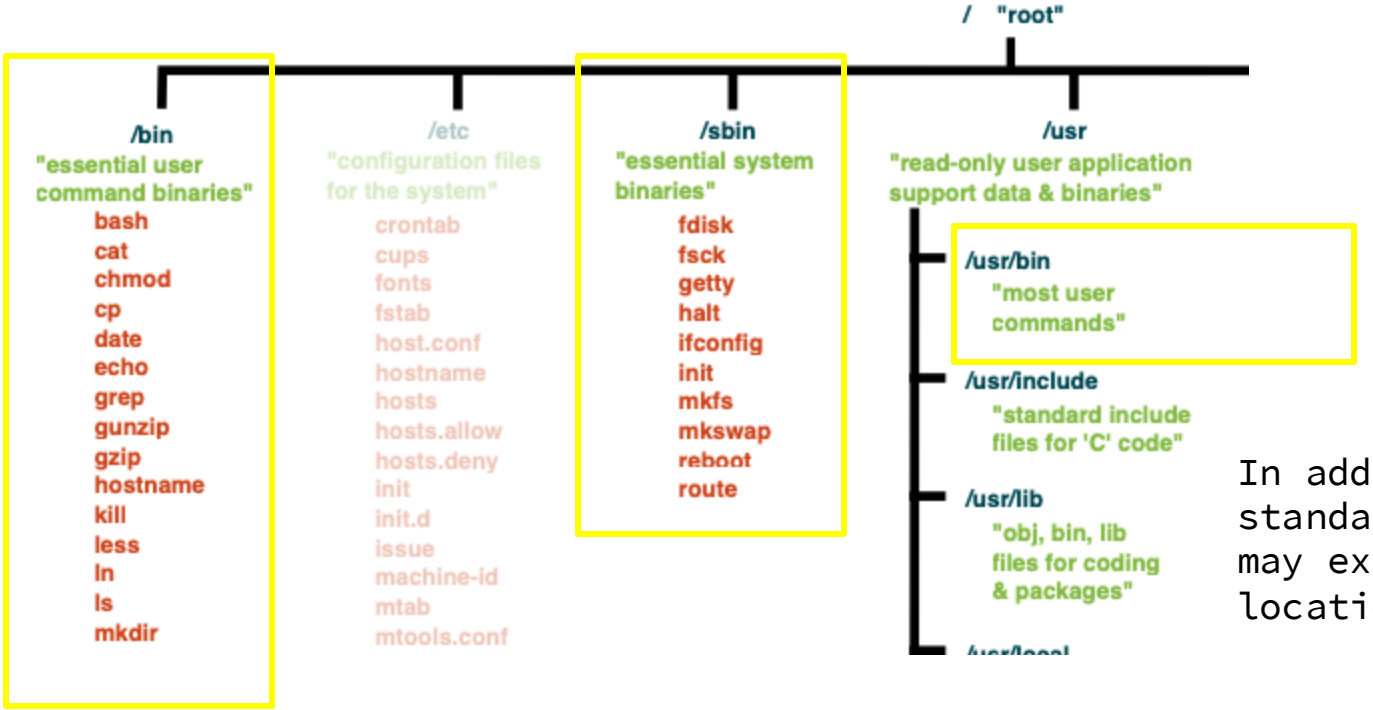
`~` (short for "home directory")

PATH Environment Variable

- PATH is an environment variable specifying a set of directories where executable programs are located

- When you execute a command, the shell searches through each directory defined in PATH, one by one, until it finds a directory where the executable exists.

UNIX Commands



In addition to the standard paths, commands may exist in other locations.

A Few Useful UNIX Tools

- `man`: view reference manuals for commands, library functions, or system calls
- `wc`: word, line, character, and byte count
- `ps`: Used to see what processes are running
- `who`: Used to see if other people are on this same machine
- `top`: Used to see active processes
- `which`: locate a program file in the user's path
- `grep`: text match based on regular expression
- `head` / `tail`: list start or end of a file
- `tr` and `sed`: two “editors” controlled by command-line options

UNIX Streams

- Input stream (`STDIN` / file descriptor 0)
  - Keyboard by default
- Output stream (`STDOUT` / file descriptor 1)
  - Terminal by default
- Error stream (`STDERR` / file descriptor 2)
  - Terminal by default

```
command < input.txt >output.txt 2>error.txt
```

Redirection of Standard Input/Output

Append output of the command into output.txt:

```
command >>output.txt
```

Redirect stdout (to `out.txt`) and stderr (to `error.txt`):

```
command >output.txt 2>error.txt
```

Redirect stderr and stdout to file `output.txt`:

```
command >output.txt 2>&1
```

UNIX Pipes

“We should have some ways of coupling programs like a garden hose — add in another segment when it becomes necessary”

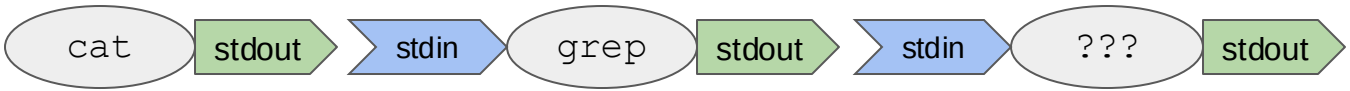
M. D. McIlroy

October 11, 1964

http://doc.cat-v.org/unix/pipes/

Composing UNIX Tools

```
cat | grep | ??? …
```



UNIX Text Processing Tools (Linux Documentation Project)

GNU coreutils Documentation

Composing UNIX Tools: Pipes

UNIX pipes allow you to send the output from one program to the input of another

```
# sorted roster

cat roster.csv | sort


# count of files last modified in September

ls -l | grep Sep | wc -l
```

Shell Scripts

- File containing a sequence of shell commands
- Support for variables, loops, conditional tests, simple math, string manipulations, pipe program output into a variable, etc.
- Combine built-in programs in many ways

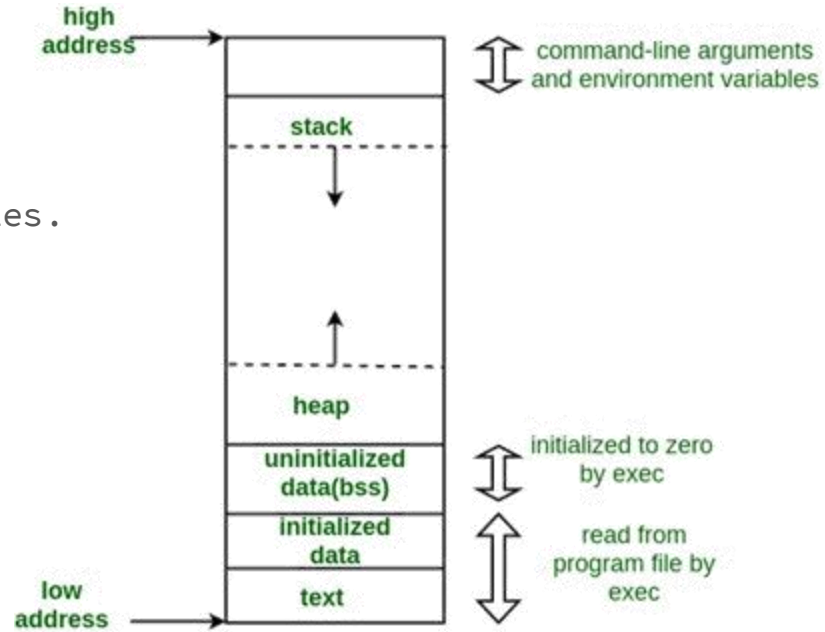
The Runtime Stack & Memory

- What is the runtime stack?
- How memory is structured for program execution.
- Stack frames and their components.

Memory Layout of a Program

Main Memory Segments:

- Text (Code) Segment: Contains the compiled program instructions.
- Data Segment:
  - Static/Global Variables: Fixed-size, allocated at compile time.
  - BSS (Block Started by Symbol): Uninitialized global and static variables.
- Heap: Dynamic memory for objects and data structures (managed at runtime).
- Stack: Stores function call information, local variables, and control data.



https://www.geeksforgeeks.org/memory-layout-of-c-program/

What is the Runtime Stack?

- A data structure in memory that stores information about active subroutines (functions/methods) in a program.

- Role of the Stack:
  - Keeps track of function calls, parameters, return addresses, and local variables.
  - Helps in managing scope and lifetime of variables.

- Key Properties:
  - LIFO (Last In, First Out) structure.
  - Automatically managed during function calls and returns.

Stack Frames

- What is a Stack Frame?
  - A block of data on the stack containing information about a single function call.

- Components of a Stack Frame:
  - Return Address: Address to return to when the function call is finished.
  - Parameters: Arguments passed to the function.
  - Local Variables: Variables declared inside the function.
  - Saved Registers: Copies of registers used by the function.