SYSTEMS DEVELOPMENT FOR COMPUTATIONAL SCIENCE

LECTURE 1

Fabian Wermelinger
Harvard University
CS107 / AC207

Thursday, September 1st 2022

TODAY

- CS107/AC207 introduction
  - Teaching staff
  - Important information
- History of Unix/Linux
  - Bell Labs
  - Why is Unix/Linux useful
  - Operating system interface
- Introduction to the command line
  - Overview of basic commands you should know
  - The `ls` command

COMPUTATIONAL SCIENCE AND LEARNING OBJECTIVES

Computational Science and Engineering (CSE) can be seen as a third mode of discovery next to theory and experiment. Computer simulation is what drives CSE and an important component in research and industry today. *This class teaches the necessary requirements for successful development of software systems that are the foundation of computer simulation.*

After successful completion of this course, the student will be able to:

- Use Python, including its advanced features to write scientific programs.
- Understand the Python data model and how it impacts the code you write.
- Write programs with good software engineering practices. These practices include: working on remote machines, version control, continuous integration, documentation and testing.
- Utilize data management techniques to store data, starting from a good understanding of data structures to databases.

CLASS COMPONENTS

Main class website: https://harvard-iacs.github.io/2022-CS107/

- ***Canvas:*** main hub for grades, class discussion forum and other sensitive content.
- ***Class Git repository:*** all class handouts are provided via Git. You submit your assignment solutions via your private Git repository as well.
- ***Discussion forum:*** we use an EdStem discussion board that has been installed on canvas. The platform is used for augmented discussion of lecture topics and assignments.

Grading:

Homework:	35%
Project:	35%
Quizzes:	15%
Pair-programming:	10%
Communal Contributions: Via the class communication platforms.	5%

- CS107/AC207 does not have standard midterm or final exams.
- Quizzes are out-of-class via canvas, open book and www accessible.

IMPORTANT INFORMATION

You must be a member of the CS107/AC207 organization on https://code.harvard.edu to clone and pull from this repository. You can request membership by sending an email to cs107-staff@g.harvard.edu (include your Harvard NetID in the body of the email. ***Note:*** your NetID is also your username on https://code.harvard.edu). *Please use your* `.harvard.edu` *email address when sending email to the address above.*

Let's do the first two tasks:

1. Login at https://code.harvard.edu
2. Get your username from your profile (NetID) and send it to cs107-staff@g.harvard.edu. Please use an appropriate subject.
3. Please read https://harvard-iacs.github.io/2022-CS107/#important. *(You will setup your private class repository in the first pair-programming lab.)*

SYLLABUS

- ***Syllabus:*** https://harvard-iacs.github.io/2022-CS107/pages/syllabus.html
  You can find all details for the class at the link above.
- ***Schedule and deadlines:*** *All deadlines are listed in the schedule, it is your responsibility to meet them.*
- ***Textbooks:*** The class does not follow a specific textbook. Some examples are taken from *Fluent Python: Clear, Concise, and Effective Programming*, by Luciano Ramalho (O'Reilly Media 2015). See https://harvard-iacs.github.io/2022-CS107/pages/resources.html for other useful resources.
- ***Prerequisites:*** You should have some basic familiarity with programming (functions, variables, constants, differences between integer and floating point, etc.) at the level of CS50. Some comfort with a tool to edit text files is beneficial. Any text editor or IDE will suit this purpose. The project will require the use of the chain rule and some basic calculus. The lectures will review the necessary fundamentals required to succeed with the project.
- See Homework 0 for a self-assessment (you do not need to solve it perfectly).

HOMEWORK

- There are a total of 7 homework assignments.
- Workflow for homework integrates the Git version control tool. It will require some steps you must take care of. See the homework tutorial on the class website with an example for a fictitious Homework 3.
- Homework is graded using Gradescope. It is integrated in the class Canvas, you can find it in the left panel for quick access.
- Your homework solutions must be uploaded to Gradescope (unfortunately Git is not yet supported for GitHub Enterprise editions). See step 4 in the homework example above.
- You will receive feedback for your work via the Gradescope platform.

  Regrade request must be issued via Gradescope within 2 days after the grades are published. ***Assignments will be regraded as a whole at most once.***
- You can use a total of ***3 late days*** for late submissions without losing points. For any assignment you can use *at most 2 days*.

PAIR–PROGRAMMING (PP)

- Pair-programming labs (sections) help internalize concepts from class ***by practice*** (practice is key).
- Exercises are released on Fridays and due 2 weeks after release. In the first week you will join your PP section to work on the problem. ***PP credit is based on attendance and completion.***
- Work in teams of about 3 students, using interactive sharing of terminal sessions with `tmate`.
- Using the class Docker container is recommended. See the pair-programming tutorial on the class website for a step-by-step example.
- Please choose your best match lab sections on the my.harvard platform (they will be available shortly). ***You will be asked to select multiple which allows us to optimize the sectioning process.***
- Deadline for selecting preferences is ***Tuesday September 6th***.

PAIR–PROGRAMMING (PP)

We have 6 zoom sessions for the first week:

Tuesday 09/06; Wednesday 09/07; Thursday 09/08

Noon sessions: 12:30pm - 1:45pm

- https://harvard.zoom.us/j/92821477585?pwd=V3NQbXpXYkt2bEVWMS9oL0FNYzBVUT09

Evening sessions: 5:00pm - 6:15pm

- https://harvard.zoom.us/j/93701361785?pwd=bnNLUEJKc0ZueURzOVFPVXM4V3NiUT09

Pair-programming exercise 1 could be solved individually as well.
Use the sessions above for help, discussions and to work with your peers.

FINAL PROJECT

- A substantial part of the grade in CS107/AC207 consists of a group project.
- The topic of the project is automatic differentiation (AD), a method to compute and evaluate derivatives of arbitrary functions up to machine precision.
- Your task is to develop an AD library in Python that involves the software development techniques discussed in class.
- You work in groups of 4-5 students (you can choose your partners).
- The project runs side-by-side the class and is divided into 7 milestones in total (see the schedule for deadlines).

https://harvard-iacs.github.io/2022-CS107/pages/project.html

QUIZZES

- We will have 4 quizzes via Canvas held off-class.
- You can find the schedule at https://harvard-iacs.github.io/2022-CS107/pages/syllabus.html#quizzes
- Quizzes will take 15 minutes and cover lecture content with 10 to 12 mostly multiple choice type of questions.
- Quizzes are open book and open `www`.

ED FORUM AND CONTRIBUTION CRITERIA

- The discussion forum is integrated on the Canvas page and can be found at the top in the left panel.
- Its purpose is to exchange knowledge and to serve as a hub for help with homework questions and other class related discussions.
- ***You are not allowed to share code related to assignments***, but the problem statement and approaches how to solve it can be discussed.
- The 5% communal contribution is based on contribution on this discussion platform. Factors accounted for are:
  - Posing questions
  - Giving answers
  - Commenting and liking posts
  - Creating posts
  - Frequency of visits
- See https://harvard-iacs.github.io/2022-CS107/pages/syllabus.html#contributions

CLASS POLICIES

- ***Attendance:*** It is expected that you attend lectures as well as labs. These are core parts of the class and therefore mandatory to attend. Please see the Attendance Policy section on the class website for further details. The Harvard Code of Honor is implied.
- ***Collaboration:*** You are welcome to discuss the course material and homework with others in order to better understand it, but the work you turn in must be your own (with exception of the project where collaborative work is permitted). Any work that is not your own, without properly citing the original author(s), is considered plagiarism. Failure to follow the academic integrity and dishonesty guidelines outlined in the Harvard Student Handbook will have an adverse effect on your final grade. Please see the Collaboration Policy section on the class website for further details.

C/C++ PRIMER (OPTIONAL)

- You can join a voluntary C/C++ primer class (recommended if you plan on taking CS205).
- The class is intended to strengthen basic C/C++ programming *(built-in types, arrays, pointers, memory allocation, operators, functions, object oriented design, inheritance and polymorphism, operator overloading, generic programming, STL and writing C++ modules for Python)*.
- Some basic knowledge of compiled programming languages is helpful to fully benefit.
- ***Everybody is welcome!*** (even if you are not taking CS107/AC207 or intending to take CS205). *If you know somebody who you think can benefit from this primer, please invite them*.
  - 10 lectures at 75 minutes each
  - ***September 19 to 30 at 5:30pm - 6:45pm weekdays*** in SEC LL2.224 SU Family Classroom

  See https://harvard-iacs.github.io/2022-CS107/pages/cpp_primer.html for details and registration.

SCHEDULE

Legend: shell scripting, version control, CI, development tools, Python, Python (advanced), data structures and databases.

Wk	Tuesday	Thursday	Labs	Events
1(35)		***Lecture 1:*** *2022-09-01* • Class introduction/organization • History of Bell Labs, Unix and Linux • Command line introduction	***Sign-up:*** Select one of the offered pair-programming lab session days according to your schedule ***PP1:*** *(2022-09-02)* Setup private class repository, `tmux`	***Note:*** Handouts are typeset in green and deadlines in red. All deadlines are due 11:59 pm. 1. HW1 release (2022-09-01)
2(36)	***Lecture 2:*** *2022-09-06* • Manual pages and help for Linux commands • Unix philosophy • Regular expressions and `grep` • File attributes and permissions • Short and unbiased journey into text editors	***Lecture 3:*** *2022-09-08* • Shell customization • I/O redirection • Process management • Environment variables and shell scripting	***PP2:*** *(2022-09-09)* Git workflow and `bash` scripting	1. Select your preferred lab sections in my.harvard (2022-09-06)
3(37)	***Lecture 4:*** *2022-09-13* • Introduction to version control systems (VCS) • Centralized and distributed VCS • Design and inner workings of Git	***Lecture 5:*** *2022-09-15* • Git blobs, trees and commits • Git status of working tree and index • Git remote repositories • Git branches	***PP3:*** *(2022-09-16)* Git branches and merge conflicts	1. HW2 release (2022-09-13) 2. HW1 due (2022-09-13) 3. PP1 due (2022-09-16)

SCHEDULE

Legend: shell scripting, version control, CI, development tools, Python, Python (advanced), data structures and databases.

Wk	Tuesday	Thursday	Labs	Events
4(38)	***Lecture 6:*** *2022-09-20* • Introduction to Python • Nested environments and closures • Decorators	***Lecture 7:*** *2022-09-22* • Object Oriented Programming (OOP) • Classes in Python • Inheritance and polymorphism • ***Quiz 1***	***PP4:*** *(2022-09-23)* Fully connected neural networks and Python closures	1. Project M1A due (2022-09-22) 2. PP2 due (2022-09-23) 3. C/C++ primer class 09/19 - 09/23
5(39)	***Lecture 8:*** *2022-09-27* • Duck typing • The Python data model • Special methods (dunder methods) • *Aside:* software licenses and open source	***Lecture 9:*** *2022-09-29* • Python class methods, static methods, instance methods • Python class attributes and instance attributes • Python modules • Python packages and PyPI	***PP5:*** *(2022-09-30)* Python classes and dunder methods	1. HW3 release (2022-09-27) 2. HW2 due (2022-09-27) 3. PP3 due (2022-09-30) 4. C/C++ primer class 09/26 - 09/30
6(40)	***Lecture 10:*** *2022-10-04* • Preliminary automatic differentiation (AD) • Derivatives and the Jacobian • Newton's method • Numerical approximation of derivatives (finite-difference method)	***Lecture 11:*** *2022-10-06* • Review of chain rule • Evaluation trace of a function and computational graph • Forward mode AD • Forward mode in higher dimensions	***PP6:*** *(2022-10-07)* Forward mode automatic differentiation	1. Project M1B due (2022-10-04) 2. PP4 due (2022-10-07)

SCHEDULE

Legend: shell scripting, version control, CI, development tools, Python, Python (advanced), data structures and databases.

Wk	Tuesday	Thursday	Labs	Events
7(41)	***Lecture 12:*** *2022-10-11* • Dual numbers and complex numbers • Implementation of forward mode AD: operator overloading	***Lecture 13:*** *2022-10-13* • Reverse mode AD • Some examples AD applications and extensions • *Quiz 2*	***PP7:*** *(2022-10-14)* Forward mode AD, Jacobian, seed vector	1. HW4 release (2022-10-11) 2. HW3 due (2022-10-11) 3. PP5 due (2022-10-14)
8(42)	***Lecture 14:*** *2022-10-18* • Continuous integration (CI) • Testing Python code (`pytest` and `unittest`) • Code coverage • Documenting code (docstrings, shpinx, readthedocs)	***Lecture 15:*** *2022-10-20* • Python virtual environments • Introduction to docker containers and `Dockerfiles`	***PP8:*** *(2022-10-21)* Python virtual environments and deploying Python packages	1. Project M1 due (2022-10-20) 2. PP6 due (2022-10-21)
9(43)	***Lecture 16:*** *2022-10-25* • Building your own docker containers • Integration of custom containers in CI workflows	***Lecture 17:*** *2022-10-27* • Abstract data types • Linked lists • Iterators	***PP9:*** *(2022-10-28)* Binary trees, binary search trees	1. HW5 release (2022-10-25) 2. HW4 due (2022-10-25) 3. PP7 due (2022-10-28)
10(44)	***Lecture 18:*** *2022-11-01* • Trees and binary trees • Binary search trees (BST) • Priority queues and heaps	***Lecture 19:*** *2022-11-03* • Python generators • Python coroutines • Introduction to Python internals	***PP10:*** *(2022-11-04)* Python generators	1. Project M2A due (2022-11-01) 2. PP8 due (2022-11-04)

SCHEDULE

Legend: shell scripting, version control, CI, development tools, Python, Python (advanced), data structures and databases.

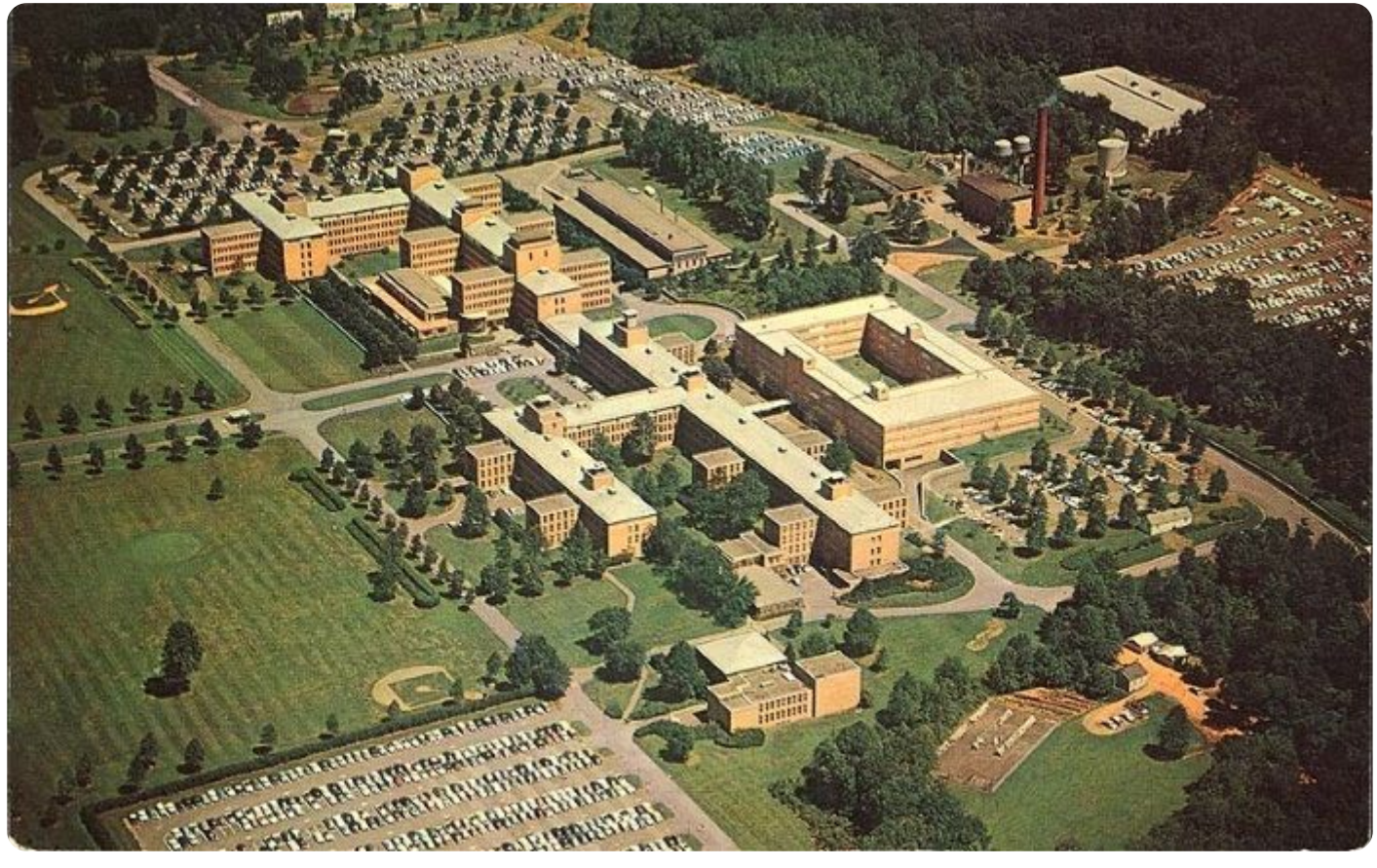
Wk	Tuesday	Thursday	Labs	Events
11(45)	***Lecture 20:*** *2022-11-08* • Code objects and Python bytecode • The Python interpreter and the evaluation loop • Memory allocation in Python • Performance of pure Python lists and NumPy arrays	***Lecture 21:*** *2022-11-10* • Data models and databases • Structured query language (SQL) • SQLite in Python • ***Quiz 3***	***PP11:*** *(2022-11-11)* Databases	1. HW6 release (2022-11-08) 2. HW5 due (2022-11-08) 3. Project M2B due (2022-11-10) 4. PP9 due (2022-11-11)
12(46)	***Lecture 22:*** *2022-11-15* • Databases • In-class exercise with SQL and Python SQLite	***Lecture 23:*** *2022-11-17* • Databases • In-class exercise with SQL and Python SQLite • Table joins • Pandas	***PP12:*** *(2022-11-18)* Finish in-class database exercises	1. Project M2 due (2022-11-17) 2. PP10 due (2022-11-18)
13(47)	***Lecture 24:*** *2022-11-22* • Debugging in Python • Generating profiles for performance analysis • Bytecode instructions and performance	***Thanksgiving break:*** *2022-11-24*		1. HW7 release (2022-11-22) 2. HW6 due (2022-11-22) 3. PP11 due (2022-11-25)

SCHEDULE

Legend: shell scripting, version control, CI, development tools, Python, Python (advanced), data structures and databases.

Wk	Tuesday	Thursday	Labs	Events
14(48)	***Lecture 25:*** *2022-11-29* • Project work • *Quiz 4*	***Lecture 26:*** *2022-12-01* • Project work		1. PP12 due (2022-12-02) 2. HW7 due (2022-12-04)
15(49)	***Reading period:*** *2022-12-06*	***Exam period:*** *2022-12-08*		1. Project final milestone due (2022-12-10)
16(50)	***Exam period:*** *2022-12-13*	***Exam period:*** *2022-12-15*		

HISTORY OF UNIX: WHERE IT ALL BEGAN...



Bell Labs, Murray Hill, NJ, 1961

BELL TELEPHONE LABORATORIES

...was what Google is today (somehow)...

...but **WAY** more influential!

- **9** Nobel Prizes have been awarded for work completed at Bell Labs
- **5** Turing Awards went to Bell Labs, *one was for Unix*
- Transistors have been invented there
- C and C++ originate from Bell Labs
- Information theory (Claude Shannon)
- The Bourne shell (Steve Bourne)
- Error-correcting-code (Richard Hamming)
- The list goes on...

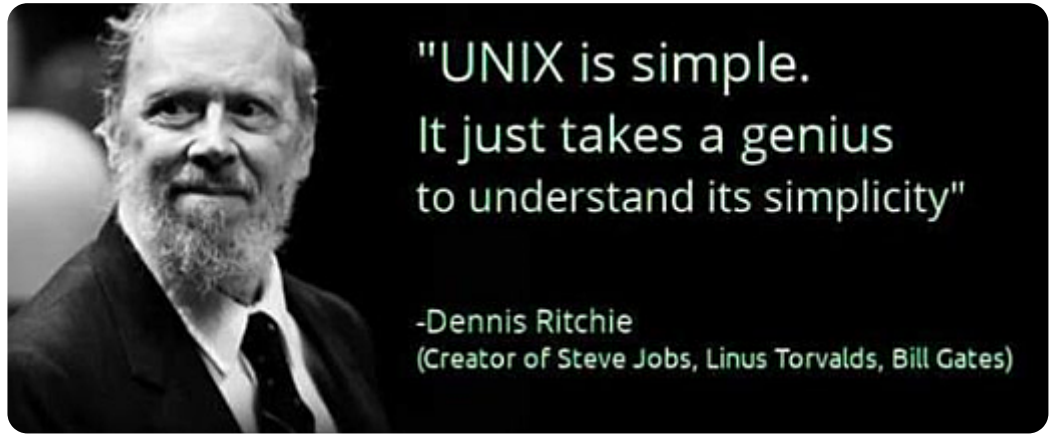
UNIX

- Is based on the ideas of MULTICS (developed at MIT)
- **Time-Sharing Operating System (OS):** many users *share* the same computing resources at the same time
- **The kernel** is responsible to manage the available resources (hardware) and coordinates the different user requests to grant them computing resources
- Some **interface** is required to communicate with the kernel

UNIX

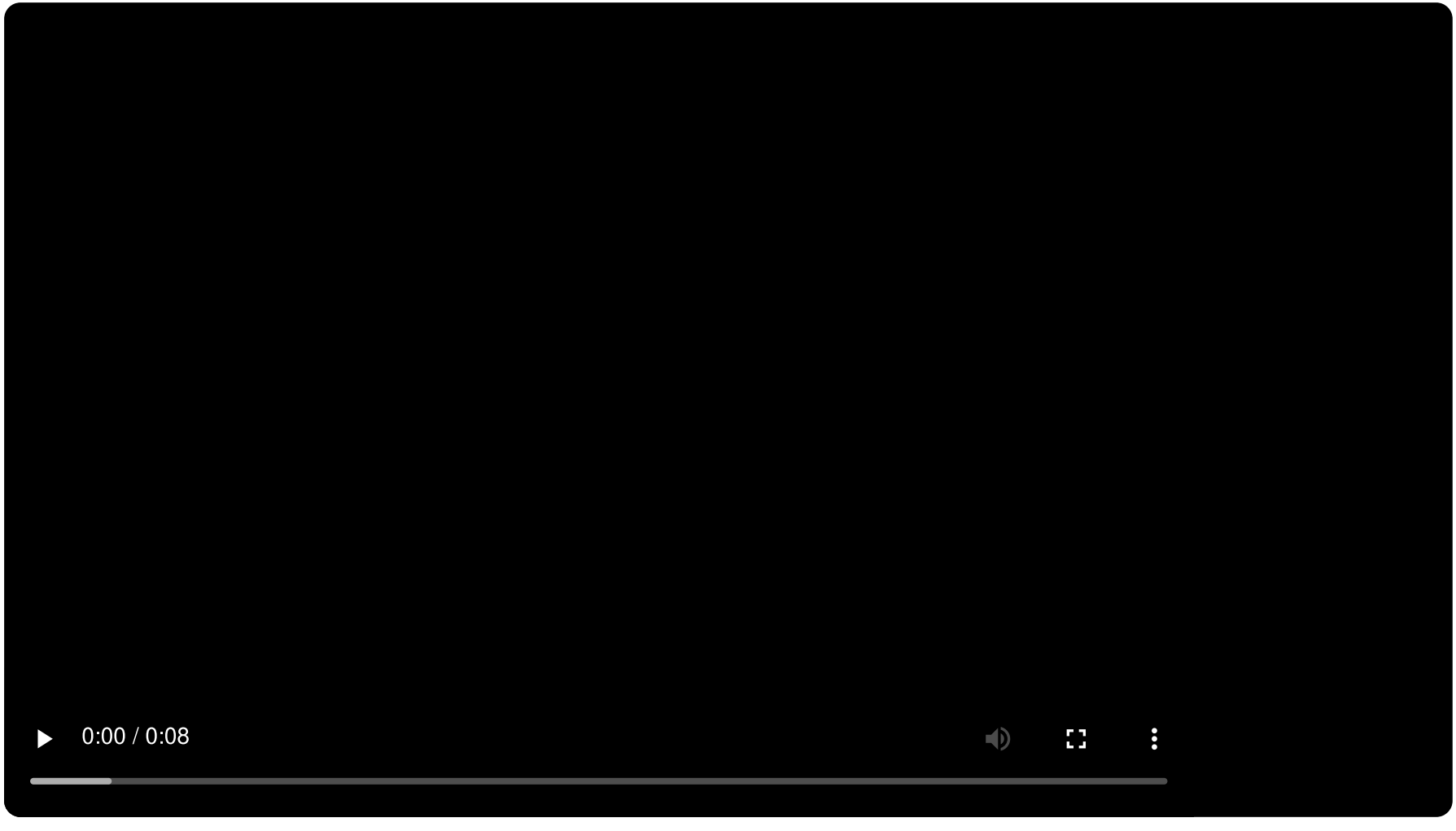
Dennis Ritchie (standing) and Ken Thompson operating a PDP-11 computer with Unix at Bell Labs (1972)

SO DO WE STILL USE UNIX TODAY?



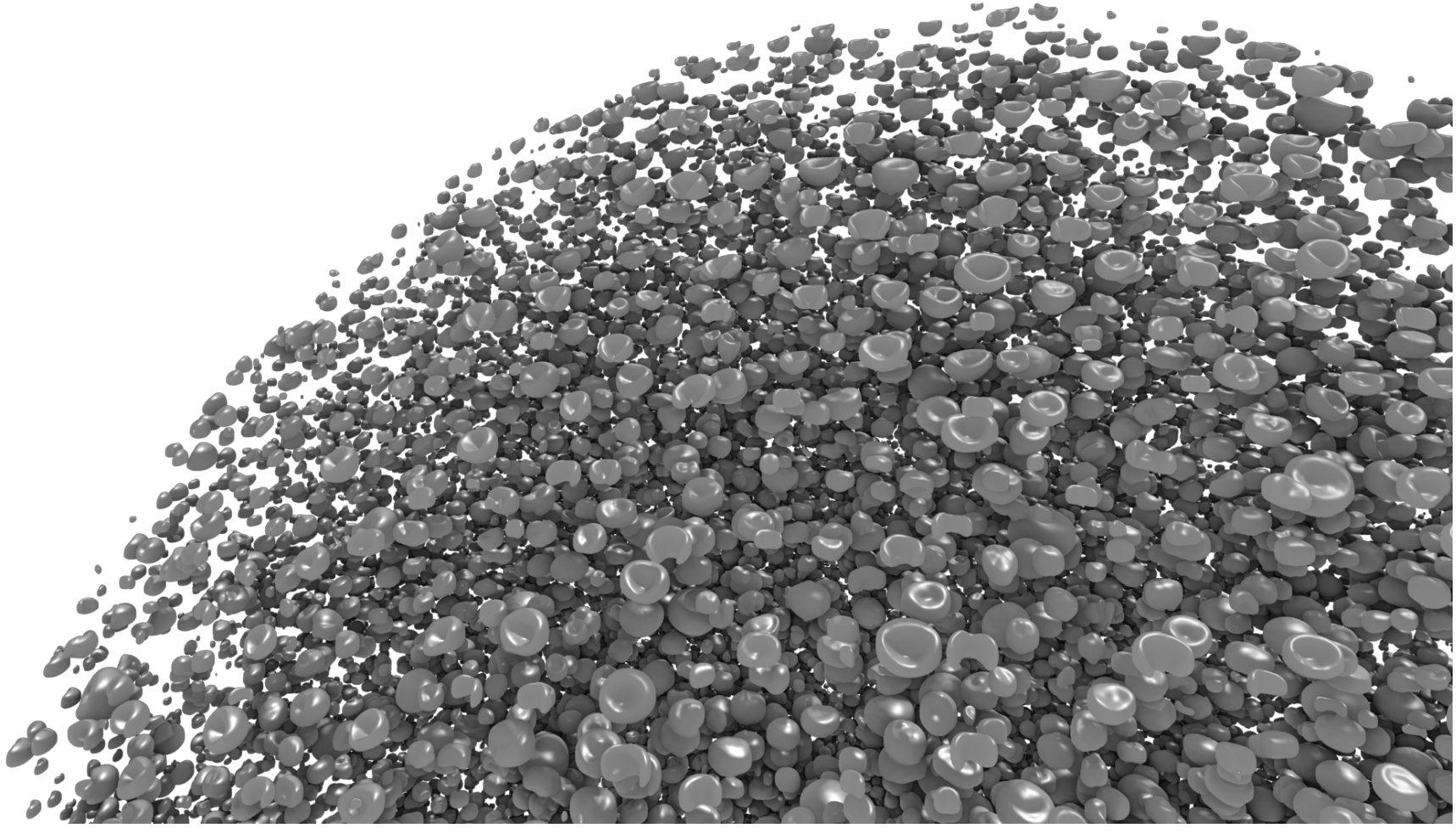
- Do you have an Android or maybe iPhone? You use Unix
- MacBook's OSX is based on Unix
- Stream Netflix? Not without Unix...
- Your research?

MOTIVATION



Compressible turbulent channel flow with air bubbles

MOTIVATION



Cloud cavitation collapse with 50'000 air bubbles

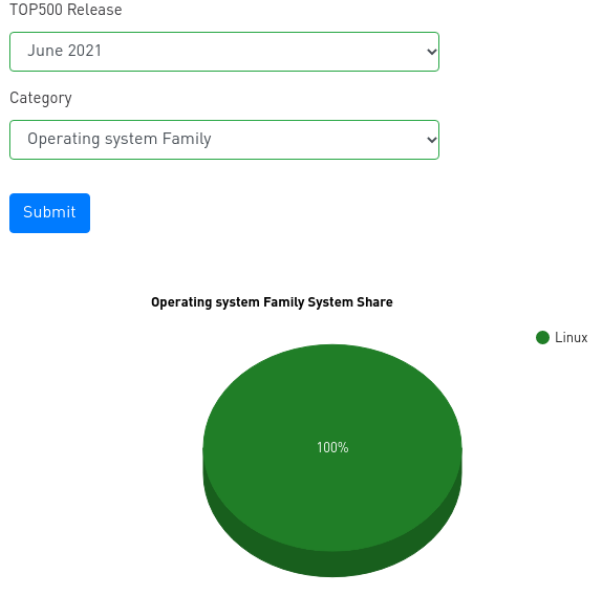
WHY UNIX/LINUX?

- Many research codes are developed and maintained in Unix/Linux systems (including the previous examples)
- Almost all servers that offer some user service (email, websites, data processing, etc.) run on a Linux operating system (OS).
- Unix/Linux is an ideal development platform (it was designed by Bell Labs for this purpose as well as necessity for time-sharing)
- Very stable and reliable due to its long existence

WHY UNIX/LINUX?

- The 500 most powerful supercomputers in the world use a Unix-like OS (https://top500.org/)
- Remote resources at companies like Google, Facebook or Nvidia use a Unix-like OS for their computing platforms.

Summit Supercomputer (ORN)



Linux OS share for top500

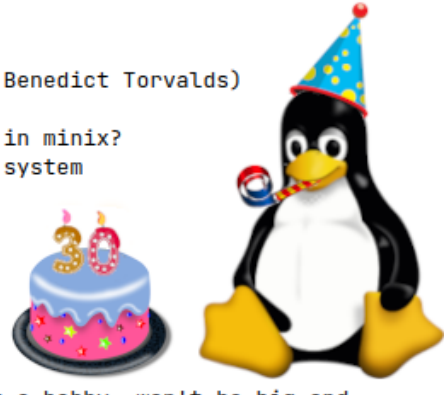
UNIX VS. LINUX

- **Unix** is licensed and you actually have to pay to use it.
- **Linux** was first developed by Linus Torvalds and first released in 1991.
- Linux is based on the same ideas of Unix, but it *does not* contain any code from Unix. It is licensed under the GNU General Public License and therefore **free software**.
- You often see GNU/Linux - GNU stands for "GNU is not Unix".

LINUX

Birthday Party was last week!

Happy 31th birthday Linux



From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID:
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

LINUX

Linux is a broad term. In most cases it refers to the **kernel** which is the code responsible to manage hardware resources. This happens in the so called *kernel space*.

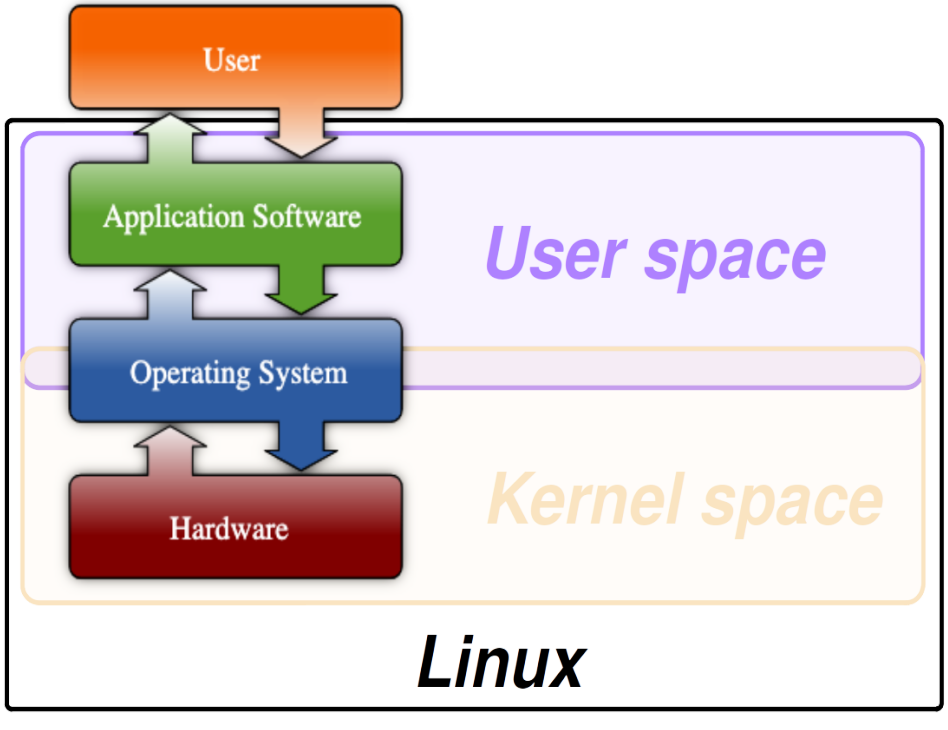
On top of the kernel space, there is a *user space* where applications run (with less permissions). Many different **Linux distributions** exist, they combine kernel, libraries and programs to make the system usable. Examples are:

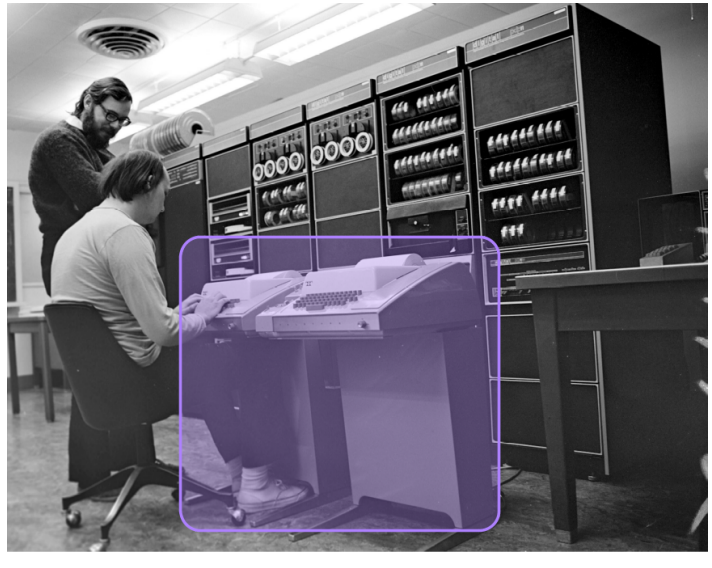
- Ubuntu (easy to get started with Linux)
- Debian (completely free, i.e. no proprietary hardware drivers)
- CentOS (often used on servers or HPC systems)
- Arch Linux (for the advanced user)

LINUX

How do you interface with the system?

User *input* and system *output*





Ken and Dennis in 1972

Fun fact: Unix/Linux commands have short names because you had to apply quite some force to type on the Teletype Model 33 terminal...

LINUX

How do you interface with the system?

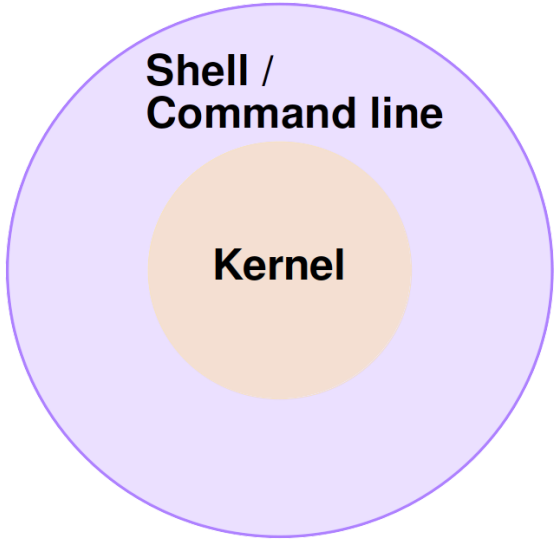
- Today mostly Graphical User Interface (GUI) dominated by **mouse** and **keyboard** input.
- The classical Textual User Interface (TUI) still exists and is dominated by **keyboard** input primarily.

Discussion:

- Which interface do you prefer? GUI or textual?
- Which of the two do you think is more efficient in terms of navigation through files, for example?
- Can you think of advantages a GUI might have over a TUI or vice versa?

LINUX

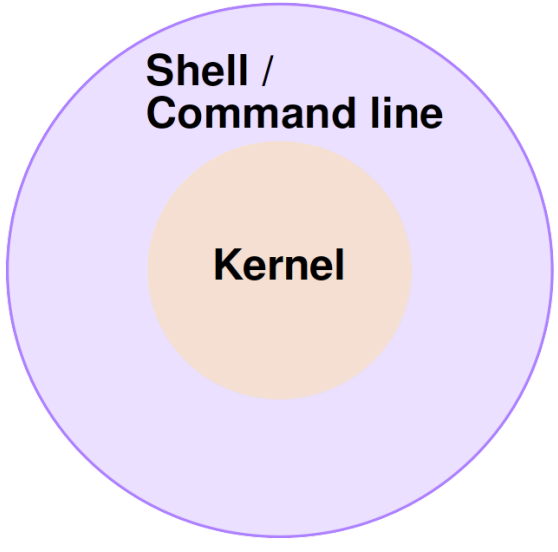
How do you interface with the system?



- *Applications* that run in user space communicate with the kernel.
- These applications can be isolated programs (e.g. allocating memory in C++ using the `new` operator involves the kernel) or a textual interface where you can enter commands *interactively*.
- This textual interface is called *command line*, the application where you enter commands is called a *shell*.

LINUX

How do you interface with the system?



- Everything in Unix/Linux is either a *process* or a *file*
- A process is a running application
  - Each process has a unique process ID (PID)
  - Processes may have different priorities and can live for a short time or run indefinitely
- A file is a sequence of bytes in memory
  - It stores data (long-term)
  - Files can be created by users or processes
  - Text files (ASCII) or binary files
  - Executable applications/programs are files itself

THE SHELL

The shell basically does four things repeatedly:

1. Display the prompt and command output
2. Read commands
3. Process commands (can be a sequence of many)
4. Execute commands

Example for listing the contents of a directory:

```
$ ls -la                                    # ls is the list command, -la are options
total 16
drwxr-xr-x 4 fabs fabs 4096 Aug 23 12:33 .      # the '.' means current directory
drwxr-xr-x 4 fabs fabs 4096 Aug 23 12:32 ..     # the '..' means parent directory
drwxr-xr-x 2 fabs fabs 4096 Aug 23 12:33 dir1   # a directory inside current dir '.'
drwxr-xr-x 2 fabs fabs 4096 Aug 23 12:33 dir2   # another directory
-rw-r--r-- 1 fabs fabs    0 Aug 23 12:33 file1  # a file
-rw-r--r-- 1 fabs fabs    0 Aug 23 12:33 file2  # another file
```

THE SHELL

- All user interaction with the system is through the shell
  - E.g. create files or directories, list all contents of current directory, ...
- There are different kinds of shells, the two main families are:
  - **Bourne shell:** `bash`, `zsh` (on Mac OSX) or `ksh`
  - **C shell:** `csh`, `tcsh`
- To remotely access a shell session you can use `ssh` (secure shell, more on it later)
- To access your Git repositories on https://code.harvard.edu/ *without* entering your password all the time, you can use a `ssh`. See this tutorial as well: https://harvard-iacs.github.io/2022-CS107/pages/tutorials.html#tutorial-repo-ssh

COMMON LINUX TERMINOLOGY

Time-sharing introduces *accounts*, which are associated with:

1. A username and password
2. A user and group ID (`uid/gid`)
3. A home directory (`$HOME`)
4. A preference for your login shell

Example: who am I and what are my ID's?

```
$ whoami
fabs
$ id
uid=1000(fabs) gid=1000(fabs) groups=1000(fabs),985(video),986(uucp),991(lp),995(audio),998(wheel)
```

FILES AND DIRECTORIES

- A file simply is a sequence of bytes in memory and it stores your data
- Every file has a *filename* associated to it
- Filenames (or directory names) are *case-sensitive* in Linux:

```
$ ls -l
-rw-r--r-- 1 fabs fabs    0 Aug 23 12:33 file1  # a file
-rw-r--r-- 1 fabs fabs    0 Aug 23 12:33 File1  # not the same file as 'file1'
```

- *Directories* are a special kind of files (they hold information about other files inside the directory)
- Think of a directory as a container for other files. *(On Mac or Windows they are often called folders and are synonymous to directories.)*

THE LINUX FILESYSTEM

- Unix (and Linux) uses a *hierarchical* system of files and directories
- The top level in the hierarchy is called the *root*, denoted by a "/" (forward slash)

```
$ ls /
bin   dev  etc   lib    lost+found  opt   root  sbin  sys  usr
boot  efi  home  lib64  mnt         proc  run   srv   tmp  var
```

  - `/bin`: contains system critical executable programs
  - `/etc`: contains system configuration files
  - `/root`: home directory of the system administrator
  - `/usr`: contains applications accessible to all users
  - `/home`: contains the home directories of all users
- The full *pathname* of a file includes all directories up to the root of the file system:

```
$ ls /home/fabs/harvard/CS107/file1
/home/fabs/harvard/CS107/file1
```

ABSOLUTE AND RELATIVE PATHS

- ***Absolute*** pathnames start at the root of the file system. In the following `/home/fabs` is an ***absolute path***:

```
$ pwd          # pwd: print working directory
/home/fabs
```

- ***Relative*** pathnames are specified in relation to the current working directory. Here " . . " denotes the *parent* directory and it is a ***relative path*** (relative to the current directory):

```
$ ls ..        # list the contents of the parent directory which is /home.
fabs
```

SPECIAL DIRECTORY NAMES

- The placeholder for the *current* directory is a dot " . ":

```
$ pwd        # print the path of the current working directory
/home/fabs/harvard/CS107
$ ls .       # list the contents of the current directory
dir1  dir2  file1  file2
```

- The placeholder for the *parent* directory is " . . ":

```
$ ls ..      # list the contents of the parent directory
CS107
```

- The tilde "~" will expand to your *home* directory:

```
$ ls ~       # list the contents of the home directory
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

- The tilde is similar to the $HOME *environment variable*

```
$ ls $HOME   # list the contents of the home directory
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

OVERVIEW OF BASIC LINUX COMMANDS

UNIX / LINUX CHEAT SHEET

FILE SYSTEM

- `ls` — list items in current directory
- `ls -l` — list items in current directory and show in long format to see perimissions, size, and modification date
- `ls -a` — list all items in current directory, including hidden files
- `ls -F` — list all items in current directory and show directories with a slash and executables with a star
- `ls dir` — list all items in directory dir
- `cd dir` — change directory to dir
- `cd ..` — go up one directory
- `cd /` — go to the root directory
- `cd ~` — go to to your home directory
- `cd -` — go to the last directory you were just in
- `pwd` — show present working directory
- `mkdir dir` — make directory dir
- `rm file` — remove file
- `rm -r dir` — remove directory dir recursively
- `cp file1 file2` — copy file1 to file2
- `cp -r dir1 dir2` — copy directory dir1 to dir2 recursively
- `mv file1 file2` — move (rename) file1 to file2
- `ln -s file link` — create symbolic link to file
- `touch file` — create or update file
- `cat file` — output the contents of file
- `less file` — view file with page navigation
- `head file` — output the first 10 lines of file
- `tail file` — output the last 10 lines of file
- `tail -f file` — output the contents of file as it grows, starting with the last 10 lines
- `vim file` — edit file
- `alias name 'command'` — create an alias for a command

SYSTEM

- `shutdown` — shut down machine
- `reboot` — restart machine
- `date` — show the current date and time
- `whoami` — who you are logged in as
- `finger user` — display information about user
- `man command` — show the manual for command
- `df` — show disk usage
- `du` — show directory space usage
- `free` — show memory and swap usage
- `whereis app` — show possible locations of app
- `which app` — show which app will be run by default

COMPRESSION

- `tar cf file.tar files` — create a tar named file.tar containing files
- `tar xf file.tar` — extract the files from file.tar
- `tar czf file.tar.gz files` — create a tar with Gzip compression
- `tar xzf file.tar.gz` — extract a tar using Gzip
- `gzip file` — compresses file and renames it to file.gz
- `gzip -d file.gz` — decompresses file.gz back to file

PROCESS MANAGEMENT

- `ps` — display your currently active processes
- `top` — display all running processes
- `kill pid` — kill process id pid
- `kill -9 pid` — force kill process id pid

SEARCHING

- `grep pattern files` — search for pattern in files
- `grep -r pattern dir` — search recursively for pattern in dir
- `grep -rn pattern dir` — search recursively for pattern in dir and show the line number found
- `grep -r pattern dir --include='*.ext` — search recursively for pattern in dir and only search in files with .ext extension
- `command | grep pattern` — search for pattern in the output of command
- `find file` — find all instances of file in real system
- `locate file` — find all instances of file using indexed database built from the updatedb command. Much faster than find
- `sed -i 's/day/night/g' file` — find all occurrences of day in a file and replace them with night - s means substitude and g means global - sed also supports regular expressions

PERMISSIONS

- `ls -l` — list items in current directory and show permissions
- `chmod ugo file` — change permissions of file to ugo - u is the user's permissions, g is the group's permissions, and o is everyone else's permissions. The values of u, g, and o can be any number between 0 and 7.
- `7` — full permissions
- `6` — read and write only
- `5` — read and execute only
- `4` — read only
- `3` — write and execute only
- `2` — write only
- `1` — execute only
- `0` — no permissions
- `chmod 600 file` — you can read and write - good for files
- `chmod 700 file` — you can read, write, and execute - good for scripts
- `chmod 644 file` — you can read and write, and everyone else can only read - good for web pages
- `chmod 755 file` — you can read, write, and execute, and everyone else can read and execute - good for programs that you want to share

NETWORKING

- `wget file` — download a file
- `curl file` — download a file
- `scp user@host:file dir` — secure copy a file from remote server to the dir directory on your machine
- `scp file user@host:dir` — secure copy a file from your machine to the dir directory on a remote server
- `scp -r user@host:dir dir` — secure copy the directory dir from remote server to the directory dir on your machine
- `ssh user@host` — connect to host as user
- `ssh -p port user@host` — connect to host on port as user
- `ssh-copy-id user@host` — add your key to host for user to enable a keyed or passwordless login
- `ping host` — ping host and output results
- `whois domain` — get information for domain
- `dig domain` — get DNS information for domain
- `dig -x host` — reverse lookup host
- `lsof -i tcp:1337` — list all processes running on port 1337

SHORTCUTS

- `ctrl+a` — move cursor to beginning of line
- `ctrl+f` — move cursor to end of line
- `alt+f` — move cursor forward 1 word
- `alt+b` — move cursor backward 1 word

COMMANDS YOU SHOULD GET FAMILIAR WITH

- `ls` — list items in current directory
- `ls -l` — list items in current directory and show in long format to see perimissions, size, and modification date
- `ls -a` — list all items in current directory, including hidden files
- `ls -F` — list all items in current directory and show directories with a slash and executables with a star
- `ls dir` — list all items in directory dir
- `cd dir` — change directory to dir
- `cd ..` — go up one directory
- `cd /` — go to the root directory
- `cd ~` — go to to your home directory
- `cd -` — go to the last directory you were just in
- `pwd` — show present working directory
- `mkdir dir` — make directory dir
- `rm file` — remove file
- `rm -r dir` — remove directory dir recursively
- `cp file1 file2` — copy file1 to file2
- `cp -r dir1 dir2` — copy directory dir1 to dir2 recursively
- `mv file1 file2` — move (rename) file1 to file2
- `ln -s file link` — create symbolic link to file
- `touch file` — create or update file
- `cat file` — output the contents of file
- `less file` — view file with page navigation
- `head file` — output the first 10 lines of file
- `tail file` — output the last 10 lines of file
- `tail -f file` — output the contents of file as it grows, starting with the last 10 lines
- `vim file` — edit file
- `alias name 'command'` — create an alias for a command

THE LIST COMMAND

The list command "`ls`" displays the contents of directories:

```
$ man ls  # get the manual page for ls
LS(1)                                User Commands                                LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).  Sort entries
       alphabetically if none of -cftuvSUX nor --sort is specified.
...
```

Some `ls` examples:

`ls`	List files in the current directory
`ls .`	List files in the current directory
`ls ..`	List files in the parent directory
`ls ~`	List files in your home directory
`ls /`	List files in the root directory
`ls /usr`	List files in the `/usr` directory

COMMAND LINE OPTIONS

- Almost all commands use *options* to customize their behavior.
- There are many options for the `ls` command, for example:
  - `-l`: *long* format
  - `-a`: *all*, shows hidden files in addition to regular files

```
$ ls
dir1  dir2  file1  file2
$ ls -a  # note: hidden filenames start with a '.'
.  ..  dir1  dir2  file1  file2  .hidden_file
$ ls -la
total 16
drwxr-xr-x 4 fabs fabs 4096 Aug 23 16:05 .
drwxr-xr-x 4 fabs fabs 4096 Aug 23 12:32 ..
drwxr-xr-x 2 fabs fabs 4096 Aug 23 12:33 dir1
-rw-r--r-- 1 fabs fabs    0 Aug 23 12:33 file1
-rw-r--r-- 1 fabs fabs    0 Aug 23 16:05 .hidden_file
```

- The `drwxr-xr-x` or `-rw-r--r--` describe the file type and permissions relative to the file owner, group and everybody else.
  Why do we care about permissions? Because of time-sharing system → there are other users on the system simultaneously *(with good and bad intentions)*

GENERAL COMMAND LINE FOR THE LIST COMMAND

- The general form is always given in the `man` page:

```
$ man ls  # get the manual page for ls
LS(1)                                User Commands                                LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...
```

- The arguments in `[ . . . ]` brackets are *optional*. If arguments are required, they will not be enclosed in such brackets.
- Options can be combined, e.g., "`ls -l -a`" is the same as "`ls -la`".
- The ellipsis " . . . " mean that this argument may occur multiple times. For example

```
$ ls -l . ~ /usr
```

lists the current, home and `/usr` directories with *long format* output.

RECAP

- CS107/AC207 Introduction
- History of Bell Labs and Unix
- Linux and different ways to interface with the system
- Looked at common Unix/Linux terminology
- Intro to the list command and its options and arguments
- All Linux commands are documented in manual pages (more next week)

Further reading (and listening):

- B. W. Kernighan, *UNIX: A History and a Memoir*, Independently published, 2019
- Check out Episode 1 and 2 of Season 1 from the Command Line Heroes podcast