CS107 / AC207

SYSTEMS DEVELOPMENT FOR COMPUTATIONAL SCIENCE

LECTURE 0

Thursday, September 2nd 2021

Fabian Wermelinger
Harvard University

OUTLINE

- Teaching Staff
- Course Policies
- History of Unix and Class Motivation
- Linux

TEACHING STAFF

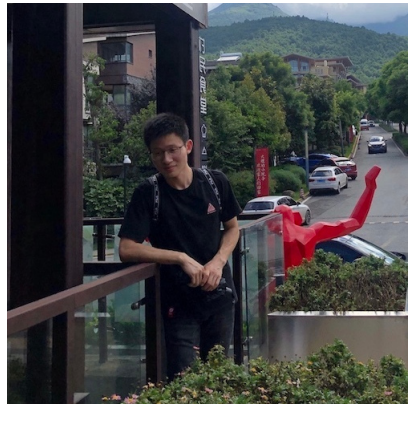
- **Head instructor:** Fabian Wermelinger (PhD, ETH Zürich)
- Lecturer in Computational Science
- **Research interests:** Fluid Mechanics, compressible multiphase flows, high performance computing, software development
- **Hobbies:** Vinyl records, cooking (love the wok!), reading, ice hockey

TEACHING FELLOWS

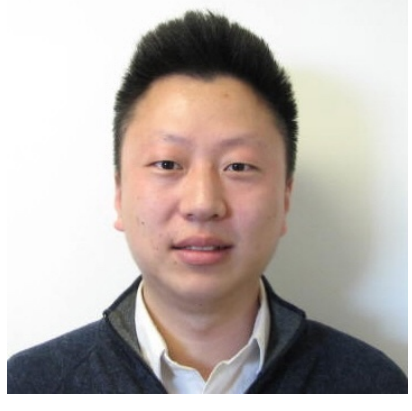
- Sehaj Chawla (Head TF)
- sehajchawla@g.harvard.edu

- Connor Capitolo
- connorcapitolo@g.harvard.edu

- Erick Ruiz
- eruiz@g.harvard.edu



- Yang Xiang
- yangxiang@fas.harvard.edu



- Johnathan Jiang
- johnathan_jiang@harvard.edu

COURSE POLICIES

- **Sign up for Piazza:** http://piazza.com/harvard/fall2021/cs107
- **Sign up for GitHub:** https://www.github.com
  - Add the teaching staff GitHub user `cs107-sys-dev` as a collaborator on your course repository when you create it. Click on "Settings" and then "Manage Access" in the left panel.
- **Understand roles of main course sites:**
  - Main class site: https://harvard-iacs.github.io/2021-CS107/
  - Canvas: Grades
  - Piazza: All course announcements and discussions
  - GitHub: All assignment submissions (homework, project, pair-programming exercises)
- **Helpline:** cs107-sys-dev@lists.fas.harvard.edu
- **Homework re-grading requests:** Send them to cs107-sys-dev@lists.fas.harvard.edu. See the Course Flow section on the main class site.

We will be working in the command line. If you are a Windows user, please install the "Windows Subsystem for Linux" on your OS (choose Ubuntu if you are not sure about which Linux). Alternatively we provide an Ubuntu based `docker` image. You can obtain it with

```
$ docker pull cs107sysdev/ubuntu
```

CLASS COMPONENTS

- 7 Homeworks (25%)
- Participation (25%)
- Final project (50%)

PARTICIPATION

- Pair-programming sections account for 20% (you must attend one section per week)
- Posting on Piazza (questioner and/or responder)
- Engaging in class discussions

PAIR-PROGRAMMING SECTIONS

- We practice coding and command-line usage with pair-programming sections
- There are multiple sections per week, you must attend **one**
- We are using Zoom for these sections
- Pair-programming exercises are not difficult and designed to be completed during the section
- If there were issues and you can not finish during the section, TFs can make a note and you are allowed to hand in the completed exercise *one* week after the last section of the cycle. Make use of the office hours for more help.
- The deadline for handing in PP exercises is one week after the last PP-section of a cycle. The start of a PP cycle is Friday morning (new PP-exercise will be available by then). The last section of that cycle is on the following Thursday morning. *The hand-in deadline is the Thursday in the following week*. Every PP-cycle we have sections on **Friday**, **Monday**, **Wednesday** and **Thursday**.
- See "Pair-Programming Sections" on our class website for more details
- The tool we are using to connect with programming mates is `tmate`

PROJECT

- You will work in groups of 3 to 4 people (assigned by teaching staff)
- You will add to your library throughout the semester
- The project is guided by two milestones
- Project topic: automatic differentiation
  - Evaluate derivatives of single-variate (or multi-variate) scalar (or vector) functions
  - Computes the result to *machine precision*
  - Applications: neural networks and back-propagation, Hamiltonian Monte Carlo methods, compute Jacobians (e.g. for coordinate transformations), ...

PROGRAMMING AND HOMEWORK 0 REFRESHMENT

- The homework, pair-programming exercises and project will be using the Python programming language
- Please spend a few minutes on Homework 0 to refresh your mind (will not be graded)

C/C++ PROGRAMMING PRIMER

I will be offering a voluntary C/C++ mini-class in calendar **week 42 and 43** if you are interested to dig into the very basics of a widely used programming language. See the main class site for more info.

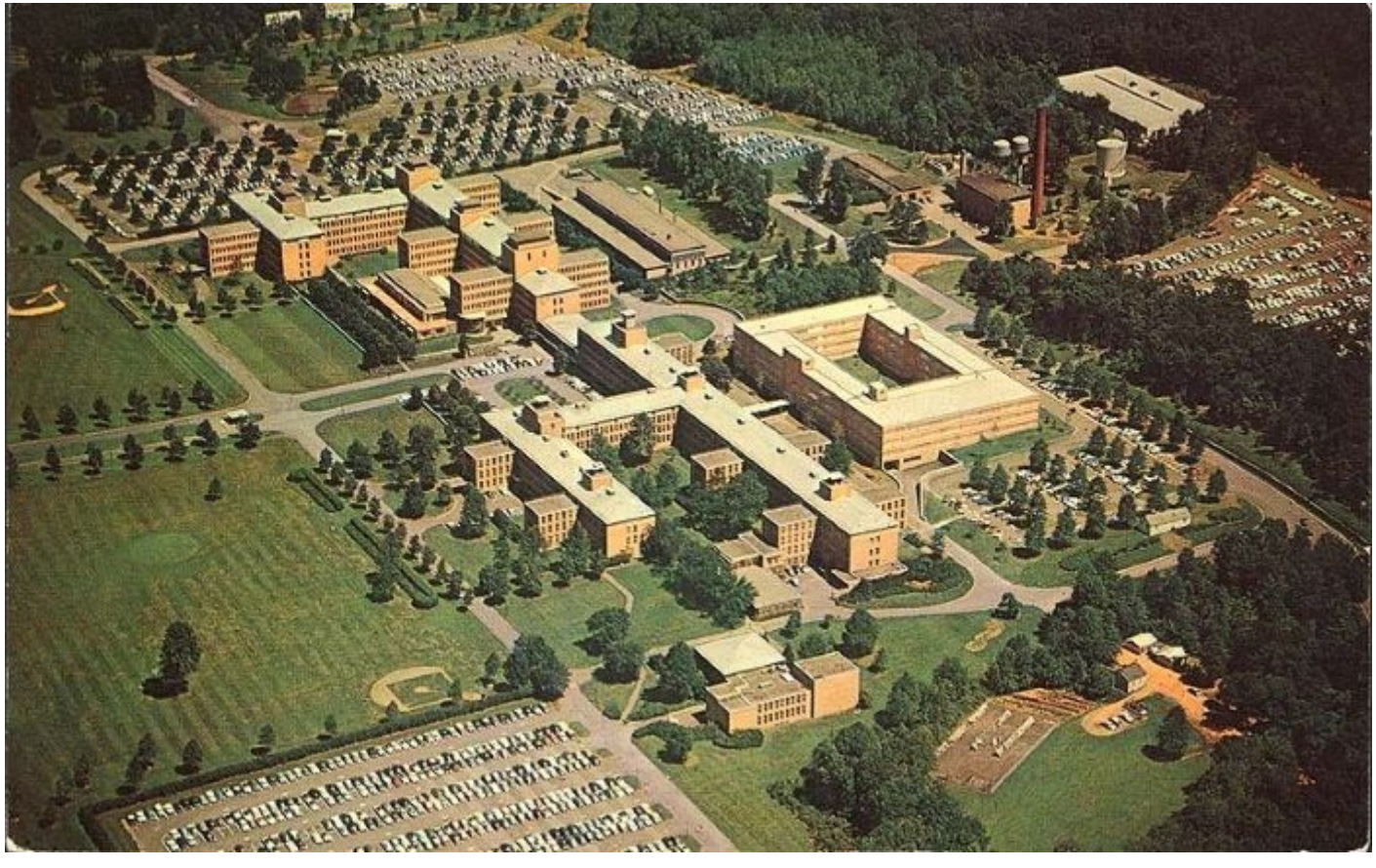
CS205: *If you plan on taking this class you must be familiar with the basics of C or C++ at the beginning of the class.*

DO YOU HAVE QUESTIONS ABOUT THE COURSE POLICIES?

- **Class syllabus:** https://harvard-iacs.github.io/2021-CS107/pages/syllabus.html
- **Coursework:** https://harvard-iacs.github.io/2021-CS107/pages/coursework.html

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...

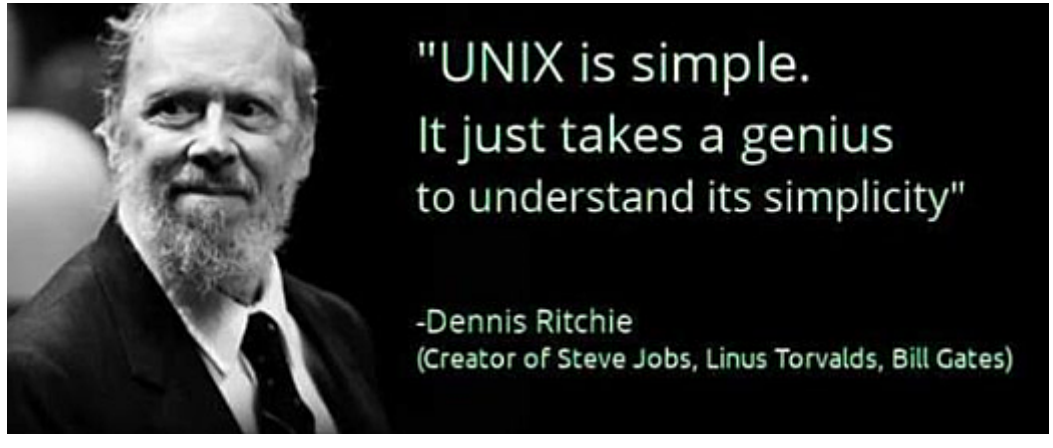
UNIX

- 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

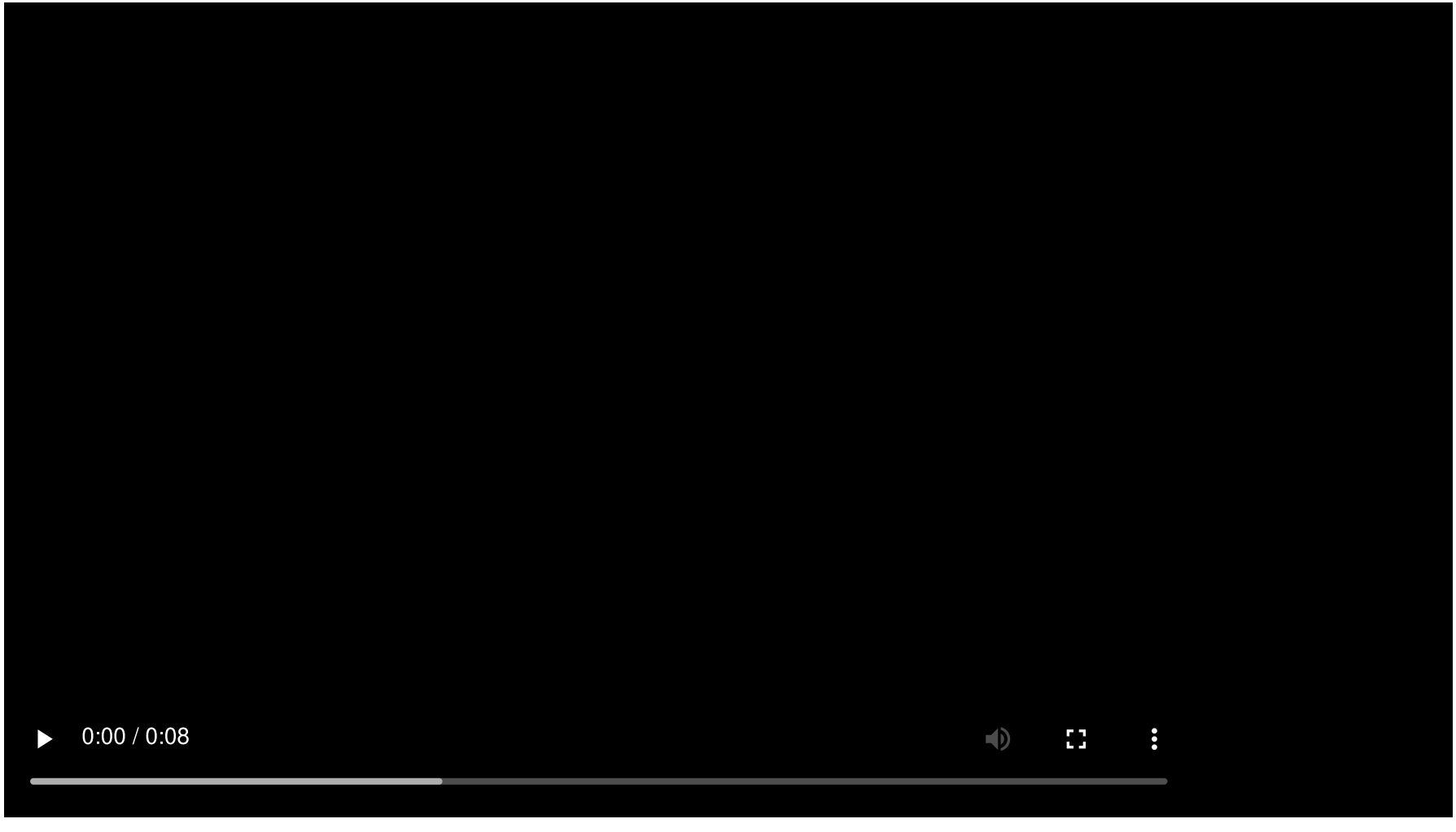
Dennis Ritchie (standing) and Ken Thompson operating a PDP-11 computer with Unix at Bell Labs (1972)

SO DO WE STILL USE UNIX TODAY?



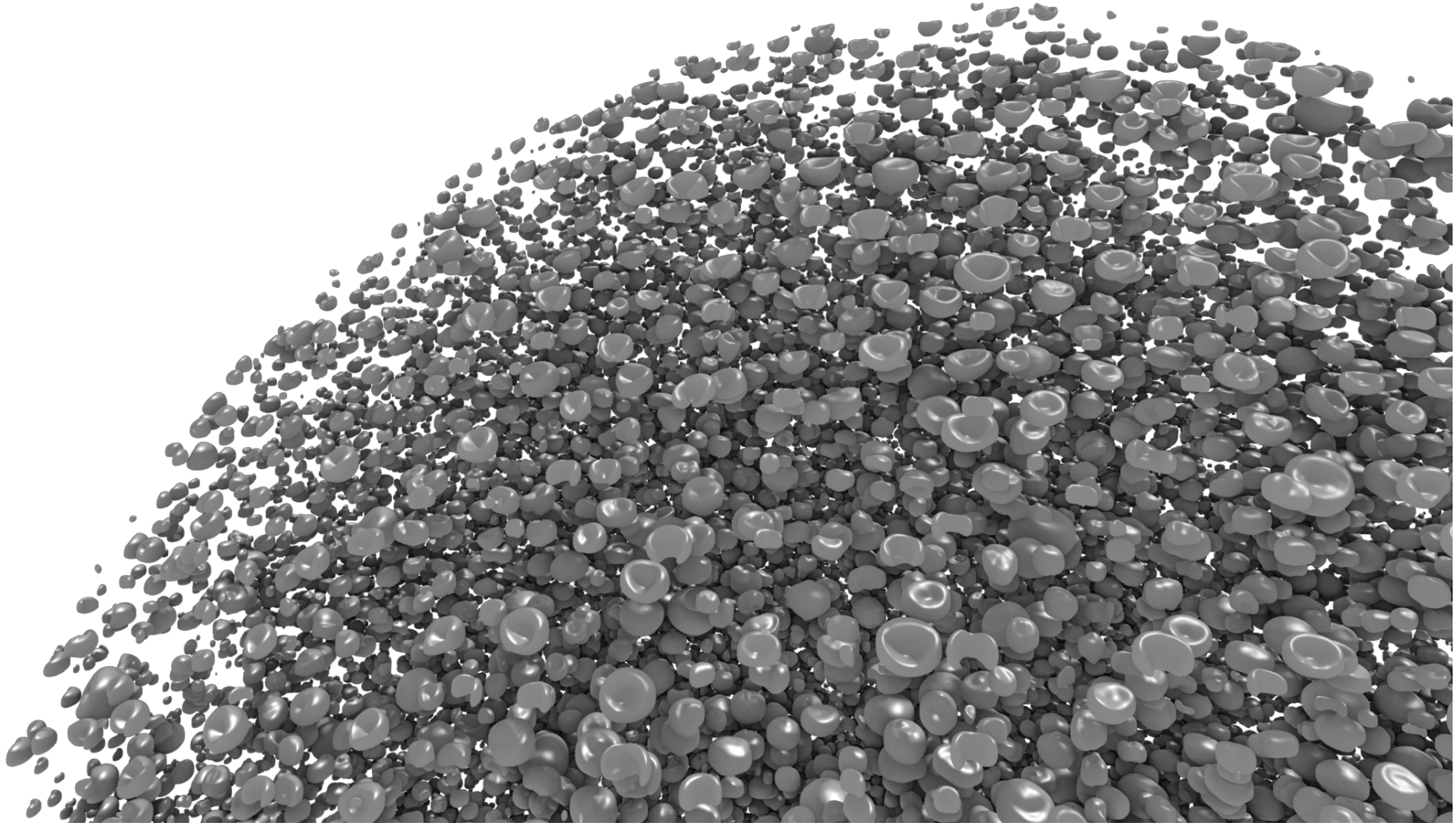
- Do you have an Android or maybe iPhone? You use Unix
- MacBook's OSX is based on Unix
- Stream Netflix? Not without Unix...
- Your research?

MOTIVATION



Compressible turbulent channel flow with air bubbles

MOTIVATION



Cloud cavitation collapse with 50'000 air bubbles

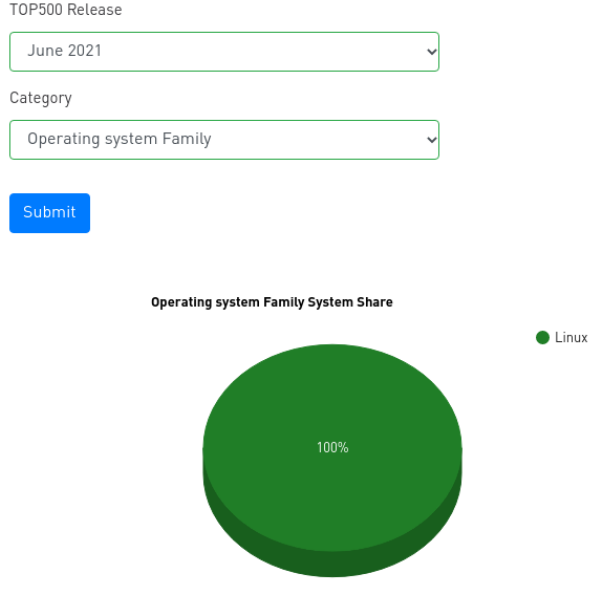
WHY UNIX/LINUX?

- Many research codes are developed and maintained in Unix/Linux systems (including the previous examples)
- 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 almost certainly

Summit Supercomputer (ORN)



Linux OS share for top500

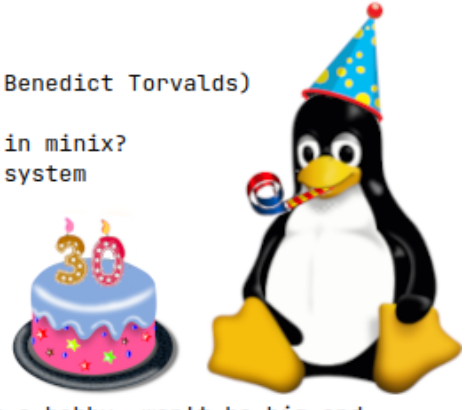
UNIX VS. LINUX

- **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

Big Birthday Party last week!

Happy 30th 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"*.

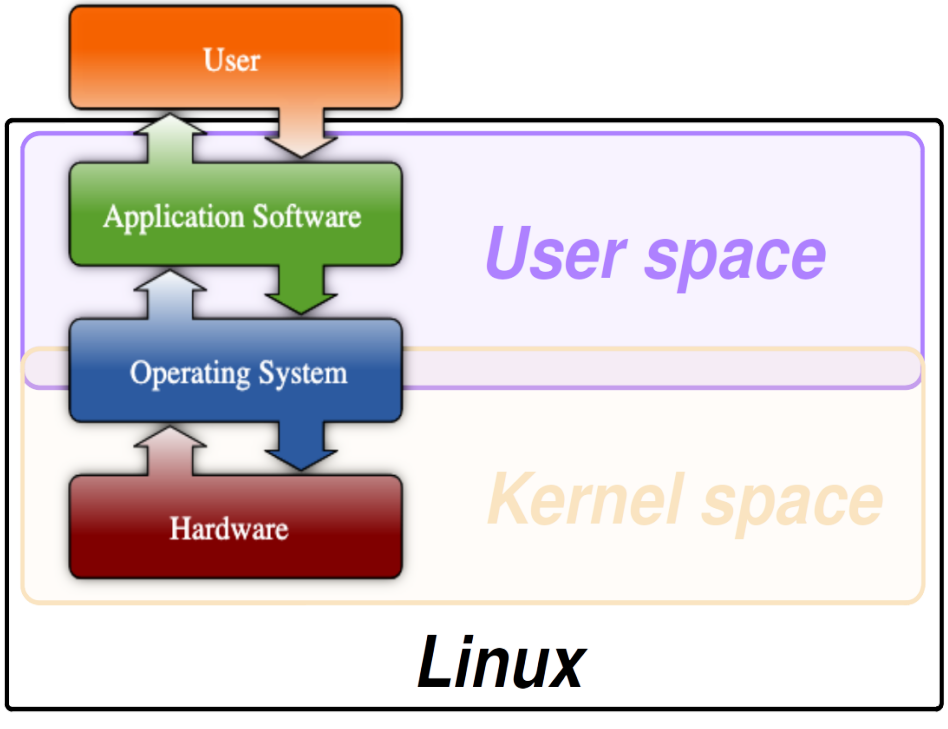
On top of the kernel space, there is a *"user space"* where applications run (with less permissions). Many different **Linux distributions** exist, they combine kernel, libraries and programs to make the system usable. Examples are:

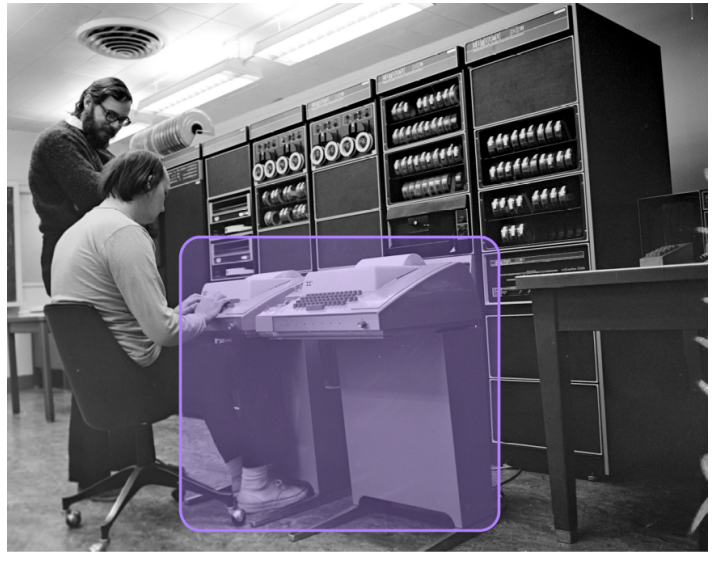
- Ubuntu (easy to get started with Linux)
- Debian (completely free, i.e. no proprietary hardware drivers)
- CentOS (often used on servers or HPC systems)
- Arch Linux (for the advanced user)

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?

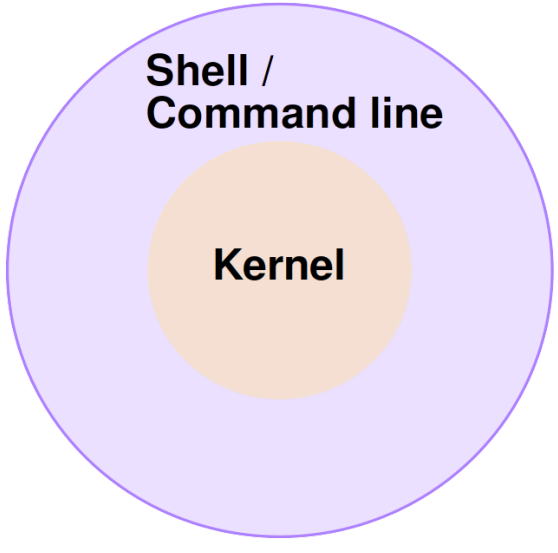
- Today mostly Graphical User Interface (GUI) dominated by **mouse** and **keyboard** input.
- The classical Textual User Interface (TUI) still exists and is dominated by **keyboard** input primarily.

Discuss with your neighbors:

- 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

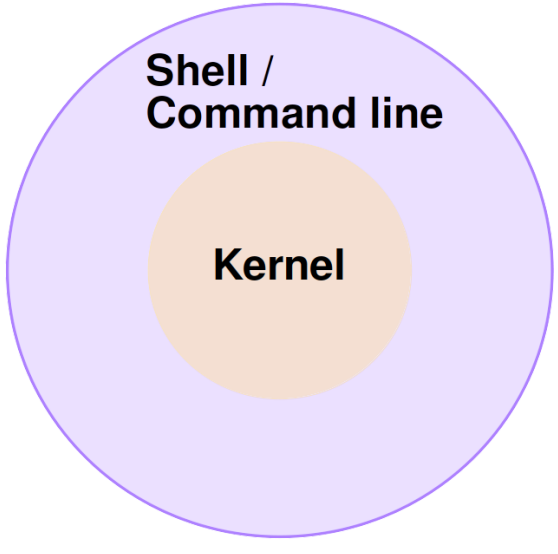
How do you interface with the system?



- *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)

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*

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:

```
$ 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 " . . " (note that 'parent' implies relative):

```
$ 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
```

OVERVIEW OF BASIC LINUX COMMANDS

UNIX / LINUX CHEAT SHEET

Tweet 89 Like Share

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 too...

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 in long format.

RECAP

- Course intro
- 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)

SUGGESTED OPTIONAL READING/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

UPCOMING SEMINAR SERIES AT IACS

The IACS hosts Seminar Series with interesting talks from researchers in Computational Science!

- Checkout the upcoming events here: https://iacs.seas.harvard.edu/
- They are free to join and will be held via Zoom
- You need to register to attend a series

Upcoming talk is by Katherine Yelick of UC Berkeley