Overview

What Does This Course Offer?

- How to be a Unix user—in fact *power*user.

What Does This Course Offer?

- How to be a Unix user—in fact *power*user.
- How to be a Unix programmer, using C to call up system services (system calls), e.g., I/O, file operations, launch programs, talk to another program, talk to another program on another computer. . .

What Does This Course Offer?

- How to be a Unix user—in fact *power*user.
- How to be a Unix programmer, using C to call up system services (system calls), e.g., I/O, file operations, launch programs, talk to another program, talk to another program on another computer. . .
- Gateway ~~drug~~ course to C69 (operating systems), D58 (computer networks),. . .

What Does This Course Offer?

- How to be a Unix user—in fact *power*user.
- How to be a Unix programmer, using C to call up system services (system calls), e.g., I/O, file operations, launch programs, talk to another program, talk to another program on another computer. . .
- Gateway ~~drug~~ course to C69 (operating systems), D58 (computer networks),. . .
- This course uses Linux, a Unix-like OS (though not a descendent). There are some differences. There are also many Unix descendents and other Unix-like OSes.

  (“Descendent” means source code traceable to Unix. The others started independently but implement almost the same features.)

Why Study Unix?

Unix has a yesteryear design but:

- Contains many ideas and techniques worth learning.

Why Study Unix?

Unix has a yesteryear design but:

- Contains many ideas and techniques worth learning.
- Home of many tools useful for all programmers.
  We will look at some.
  (Unix was made to support programmers.)

Why Study Unix?

Unix has a yesteryear design but:

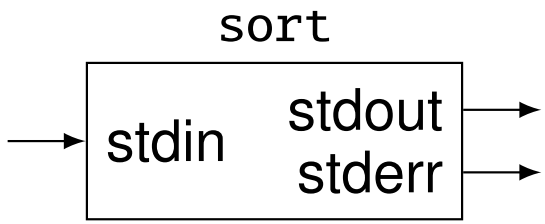
- Contains many ideas and techniques worth learning.
- Home of many tools useful for all programmers.
  We will look at some.
  (Unix was made to support programmers.)
- Unix-like OSes ubiquitous though you don't see them:
  Most network equipments (home and industry), Kindles, Android, iOS, PS4, Steam Console.
  Favourite choice on Raspberry Pi, servers (those in data centres, "cloud").

What Is It Like to Use Unix?

Video clip: Kernighan's explanation.

How is it possible? I'll show some block diagrams:

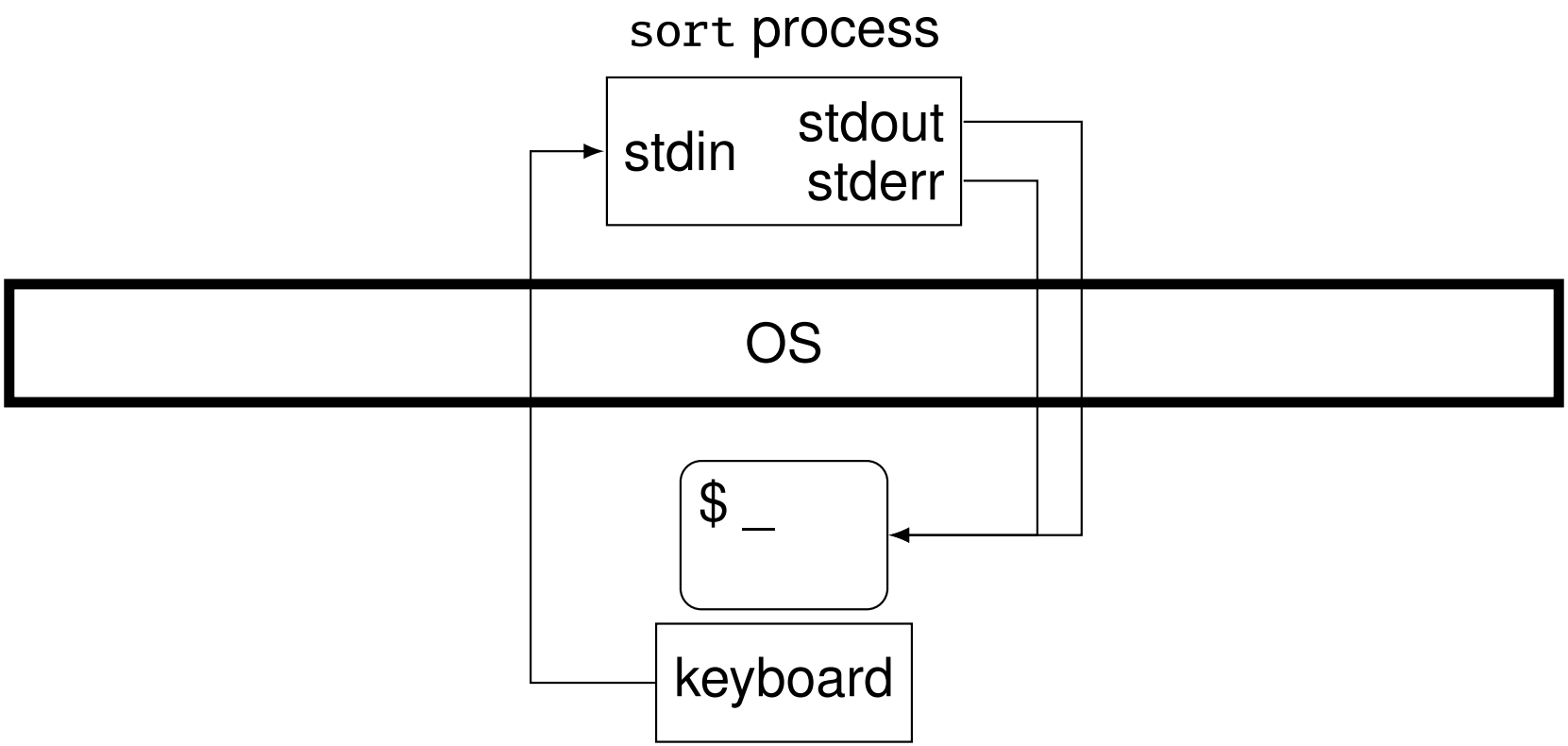
A program has access to these I/O data streams:
stdin = standard in[put]
stdout = standard out[put]
stderr = standard error (for error messages)



Actually: *process*, not program. Process = what happens when you run a program.

What It Is Like to Use Unix

Default setup: Connected (via OS) to terminal.

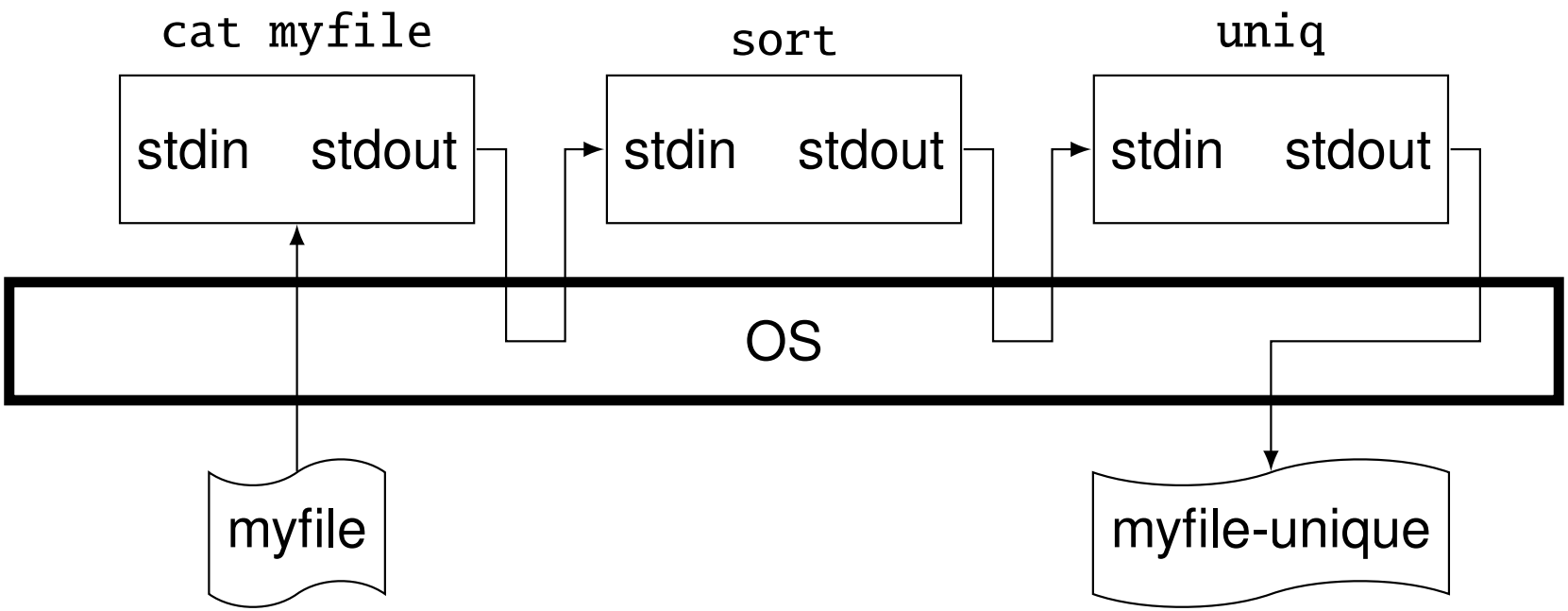


Exercise: run `sort` alone, enter a few lines, use Ctrl-D on its own line to end, see what happens.

What It Is Like to Use Unix

But configurable to connect (via OS) to files (redirection) or other programs (pipelining).

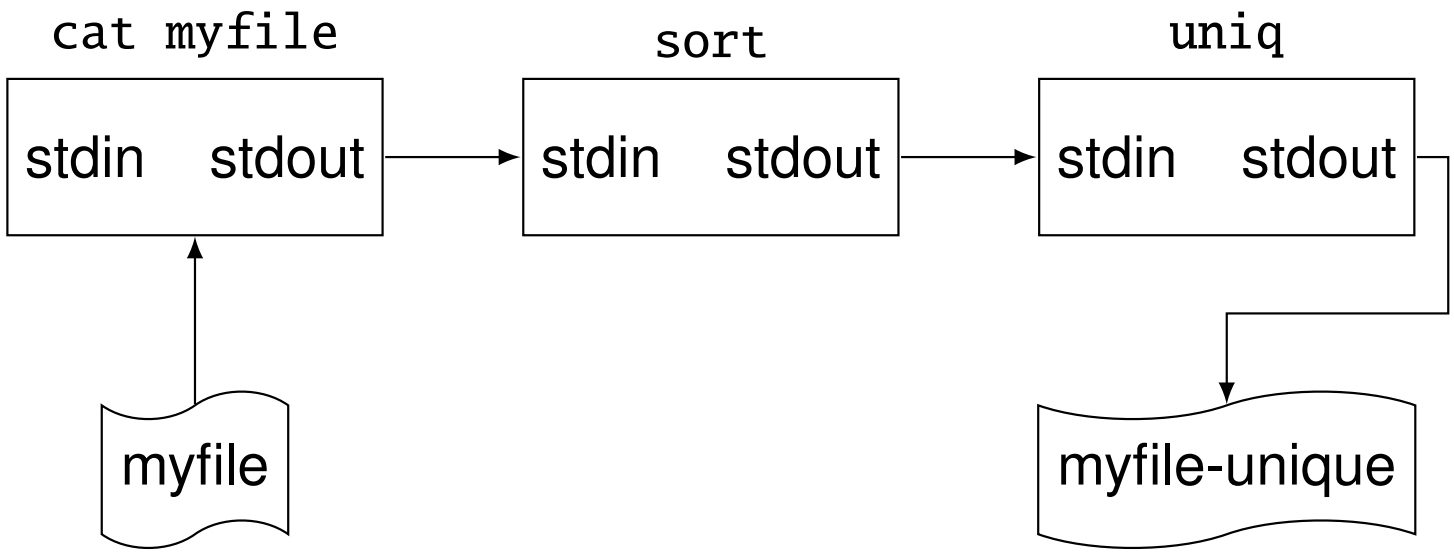
```
cat myfile | sort | uniq > myfile-unique
```



(stderrs to terminal, not shown. Exercise: Why stderr≠stdout?)

What It Is Like to Use Unix

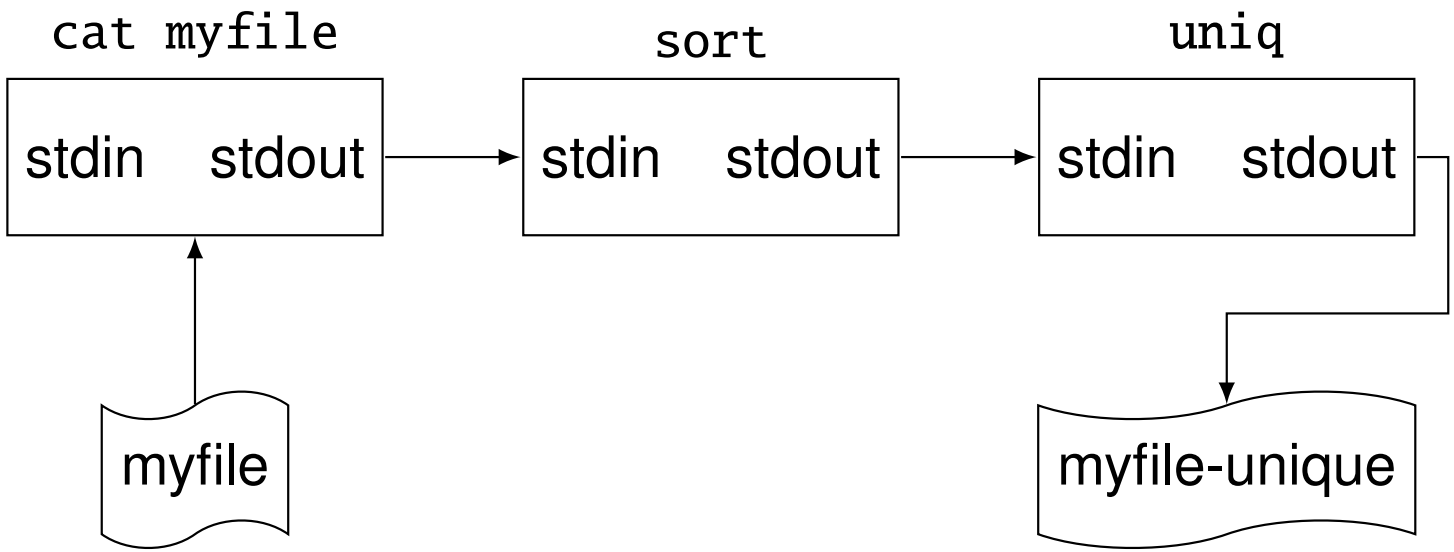
Sometimes I draw this simpler, higher-level picture when the spotlight is not on the OS (so omit it):



(stderrs to terminal, not shown.)

What It Is Like to Use Unix

Sometimes I draw this simpler, higher-level picture when the spotlight is not on the OS (so omit it):



(stderrs to terminal, not shown.)

In this course, we begin as users of pipelining and redirection; later on, we learn how to use system calls to implement it (and more).

Scripting

The command line interface is also scriptable.

Instead of 3 similar commands:

```
cat mywords | sort | uniq > mywords-unique
cat yourwords | sort | uniq > yourwords-unique
cat badwords | sort | uniq > badwords-unique
```

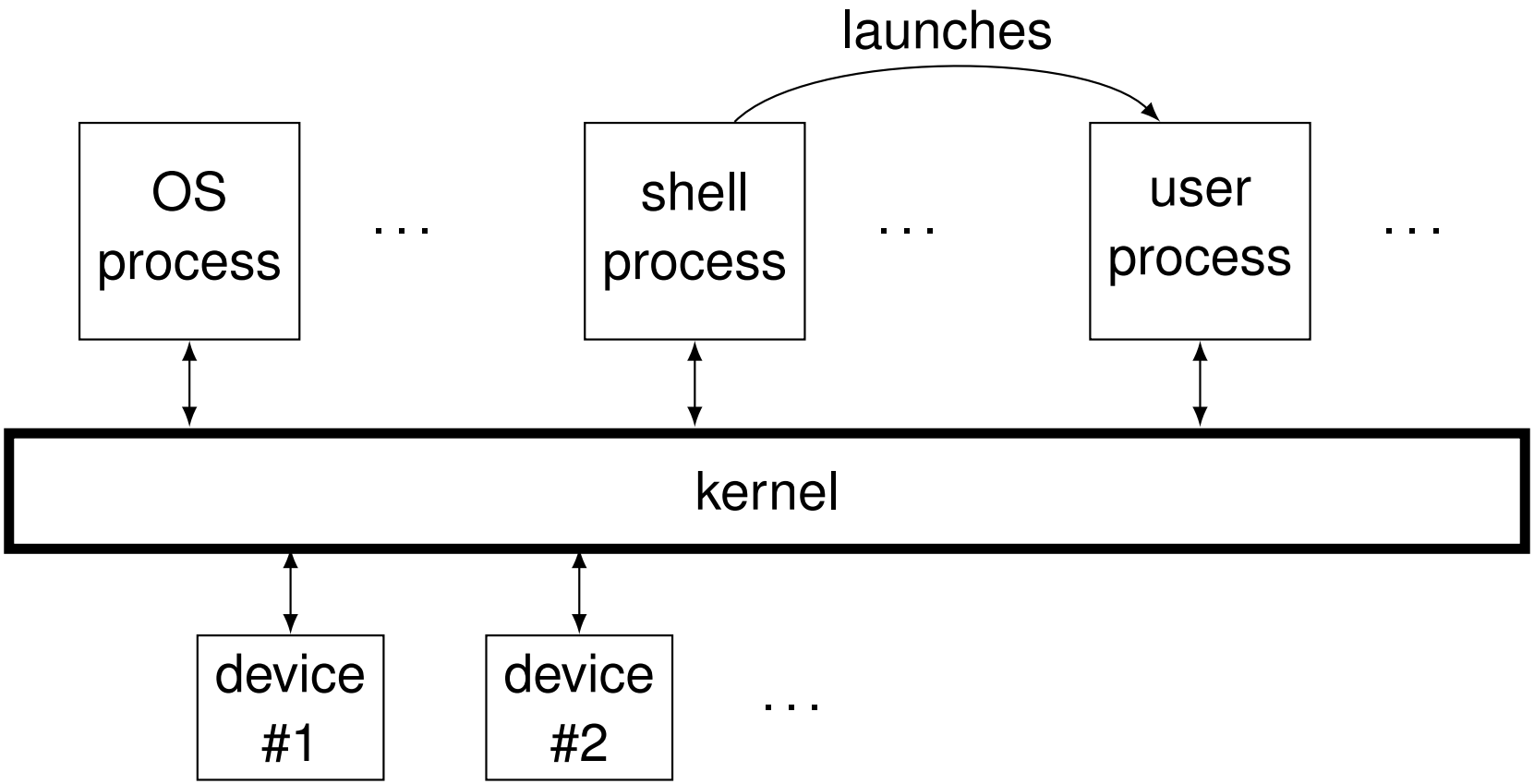
how about a for-loop!

```
for w in mywords yourwords badwords; do
  cat $w | sort | uniq > $w-unique
done
```

Can enter that at the command line.
Can also put that in a file and run that file (“shell script”).

System Structure

Block diagram to keep in mind throughout the course:



Next slide briefs you on the vocabulary.

Terminology

Kernel: arbitrator and service provider: decides which process to run and when, what it may access or not, how to access.

Process: what happens when you run a program.

OS processes: More services, features, and background monitoring. Because a lot of services don't have to live in the kernel.

Shell: That 70s text-mode command-line user interface. (Modern graphical desktops are also called shells, e.g., GNOME shell, Windows shell.)

User processes: your processes.

Special Files for Devices and Services

Unix presents devices and some services and info as files.

Of course not real files, the kernel makes up filenames and emulate file operations (open, read, write, close). We say “special files”.

(For real files as you know them, “regular files”.)

Examples:

- /dev/sda: Hard disk, the whole hard disk. (Clearly, restricted access (why?).)
- /dev/urandom: Crytographically secure random bytes.
- /dev/null: Discards written data. Empty when read.
- /proc: Info about processes and system stuff.

Unix {Philosophy, Style, Spirit}

Small, focused programs that use I/O data streams.

Combine them in a shell command/script for complex tasks.

OS fakes a file-like interface for accessing peripherals and services, to support that technique. (Some complication and imperfection in practice.)

E.g., process reads stdin like any file, even when it's terminal. The terminal even has fake filenames.

(Unfortunately) Terse culture, e.g., the program for copying files is called `cp`.

Incompleteness Guarantee

This course cannot possibly go over all utilities, all system calls, all special files, all scripting tricks, all C techniques. . .

This course will only provide orientation and cover selected topics.

From them, you must learn the underlying skill (system-level thinking) so you are ready to pick up the rest upon demand.

Assignments, labs, term test, and exam can require you to learn uncovered topics (but short) on the spot and apply them.

Survival Guide

Ctrl-C can abort most processes.

Many programs support `--help` as a command line argument, e.g., try `uniq --help`

Many programs (also C library functions and system calls) have detailed doc via the `man` program (short for “manual”), e.g., try `man uniq`

Those “man pages” have professionals in mind. It is normal if you can’t understand at the beginning. But learn to pick out the parts you need as you go.

Seek help early, seek help often. Think up and perform experiments, build toys to test ideas. Treat this as a *tinkering* course (esp. not just “knowledge transfer”).

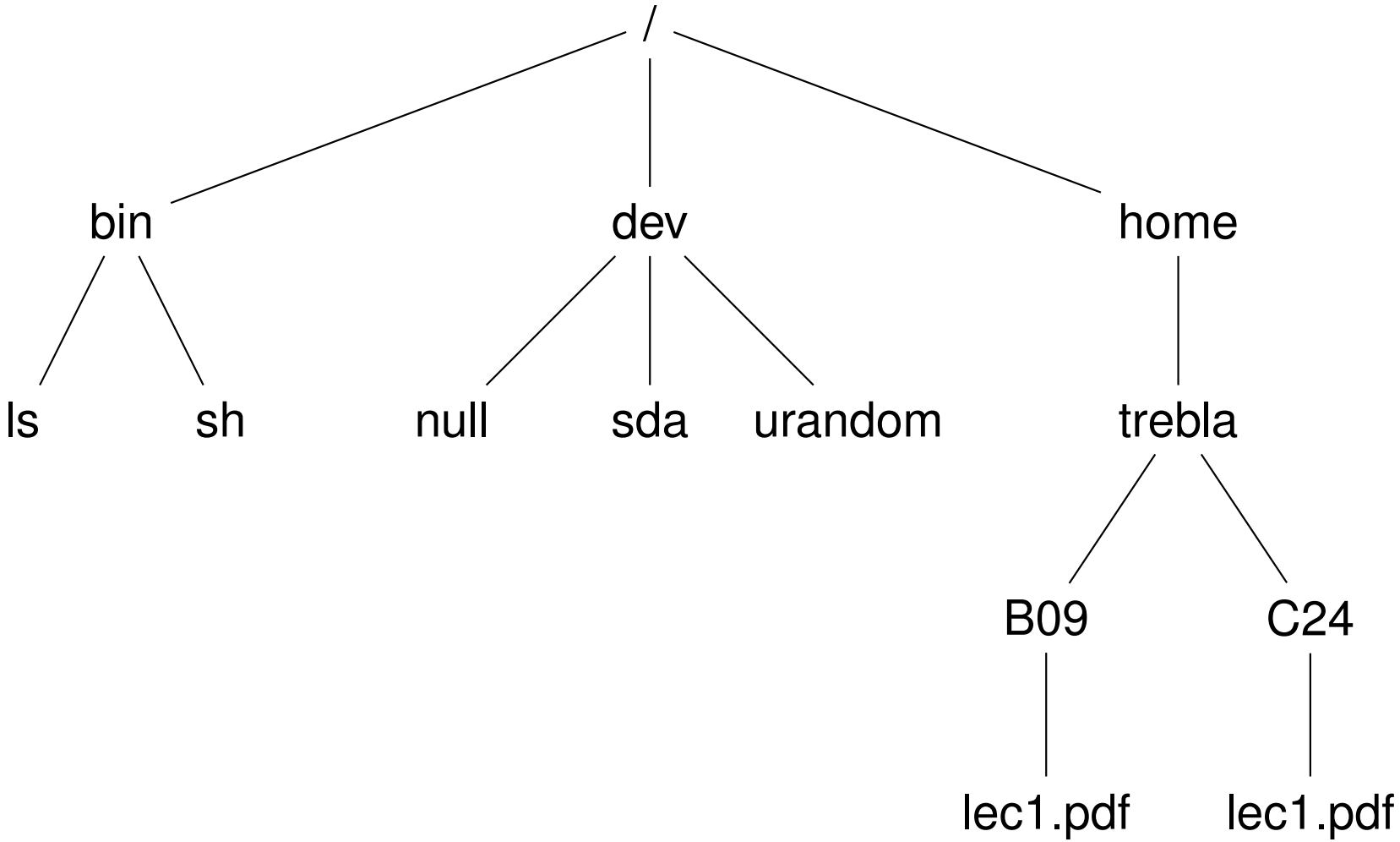
Hope you will get the hang of it!

Tour of file management

Or: How we survived without File Explorer.

Directory Tree Model

Partial, but starts from system-wide root. Also, “tree” is an approximation.



Path(name)s

How to refer to a file or directory in the tree.

- Absolute path: start from root.
  /home/trebla/B09/lec1.pdf
- Relative path: start from current directory.
  B09/lec1.pdf
  (Makes sense if current directory is /home/trebla.)

“Current directory” is part of the current state of a process.

Path(name)s

Pathnames may also include:

- parent directory: `..`
- the directory itself: `.`

Examples: If current directory is /home/trebla, then these two both refer to /bin/ls:

```
../../bin/ls
../../bin/./ls
```

Why is `.` useful: Some commands want a directory name, and you want to name the current directory.

`pwd` and `cd`

`pwd` (print working directory): Output absolute path of current directory.

`cd` (change directory): Set current directory.

Example: `cd B09`

Example: `cd ../C24`

Example: `cd /dev`

`ls` (list)

List filenames. Default: in the current directory (folder), alphabetical order.

Given directory name(s): in those directory(es).

Given filename(s): list those filenames. (Why useful? See '`-l`' below.)

Some options:

- `-l`: More information, e.g., access permissions, size, modification time. (Next slide.)
- `-d`: Directories themselves, not files inside.
- `-t`: Order by modification time, new to old.
- `-r`: Reverse order.
- `-R`: Recurse into subdirectories—whole tree.

`ls -l` information

```
-rw-r--r-- 1 laialber cmsusers 63 May  6 20:28 myfile
```

`-rw-r--r--`	access permissions (later slides)
`1`	hard-link reference count (future lecture)
`laialber`	owning user
`cmsusers`	owning group (later slides)
`63`	file size, bytes
`May 6 20:28`	last modification time
`myfile`	file name

Directories have a leading “`d`”:

```
drwxr-xr-x 2 laialber cmsusers 4 May 19 18:49 mydir
```

`ls -a`, `ls -A`, Dot Files

‘`ls -a`’: include filenames starting with ‘`.`’ (“dot files”)

‘`ls -A`’: like ‘`-a`’ but exclude ‘`.`’ and ‘`..`’

‘`..`’ stands for parent directory
‘`.`’ stands for the directory itself

Convention: “dot files” contain user settings, would be annoying to be listed all the time.

Example: .nanorc has nano settings.

`cat` (dump file(s))

Dump file content or stdin to stdout.

Example: `cat myfile`

Handy for viewing a short text file. For long files, see next slide.

More generally, dump one or more files consecutively to stdout, “concatenate”, hence the name “cat”.

Example: `cat file1 file2 file3`

`less` (view a text file)

View a text file with nice scrolling and searching.

Example: `less myfile`

action	key
scroll	down, up, pgdn, pgup
goto line 42	42g
search “foo”	/foo <enter>
search next	n
search prev	N
unhilight	<esc> u
help	h
quit	q

Trivia: Old limited viewer called “more”. New better viewer called “less” for irony and proverb “less is more”.

`mkdir` (make directory)

Create new directory(es). Names are from the arguments you provide, e.g.,

```
mkdir lab02 ../C24/lab02 /tmp/foo
```

Exercise: Read up about the option '`-p`' and test it.

`cp` (copy)

Copy files.

Copy a file to a new pathname:

```
cp myfile newname
```

Copy file(s) to a directory:

```
cp file1 file2 B09
```

Copy recursively:

```
cp -R /home/trebla /tmp/mystuff
```

Be careful: Can overwrite existing files.

`mv` (move)

Can Rename. Can move to another directory.

Rename:

```
mv myfile mycoolfile
```

Move file(s) and/or directory(s) to another directory:

```
mv myfile B09 /tmp
```

Be careful: Can replace existing files.

`rm` and `rmdir`

`rmdir` (remove directory): Delete directory(es). Precondition: they are empty.

`rm` (remove): Delete file(s). Does not delete directories unless:

Recursive delete:

```
rm -r /home/trebla
```

Be careful: They don't enjoy a "recycle bin", i.e., you won't be able to restore.