Basic Shell Usage, part 1

ComS 252 — Iowa State University

Andrew Miner

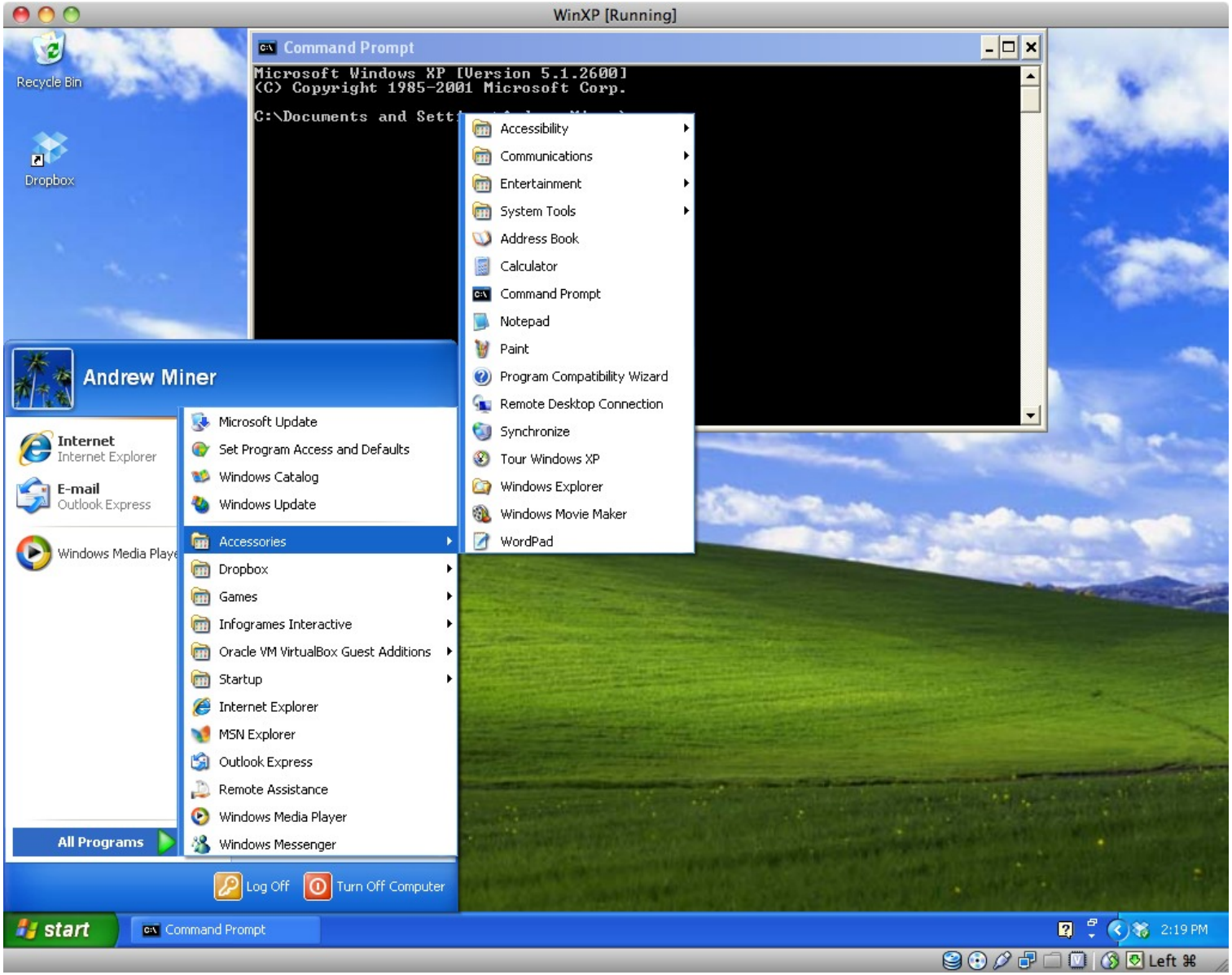
Graphical User Interfaces (GUIs)

- Mainstream OS's provide GUIs to manage files and folders
  - Browse folders
  - Open files
  - Copy, move, rename, and delete files
  - Copy, move, rename, and delete folders
- In Windows: Windows Explorer (since Windows 95)
- In Mac OS X: Finder
- In Linux: default one depends on your desktop; e.g
  - Nautilus in GNOME
  - Konqueror in KDE
  - PCManFM in LXDE

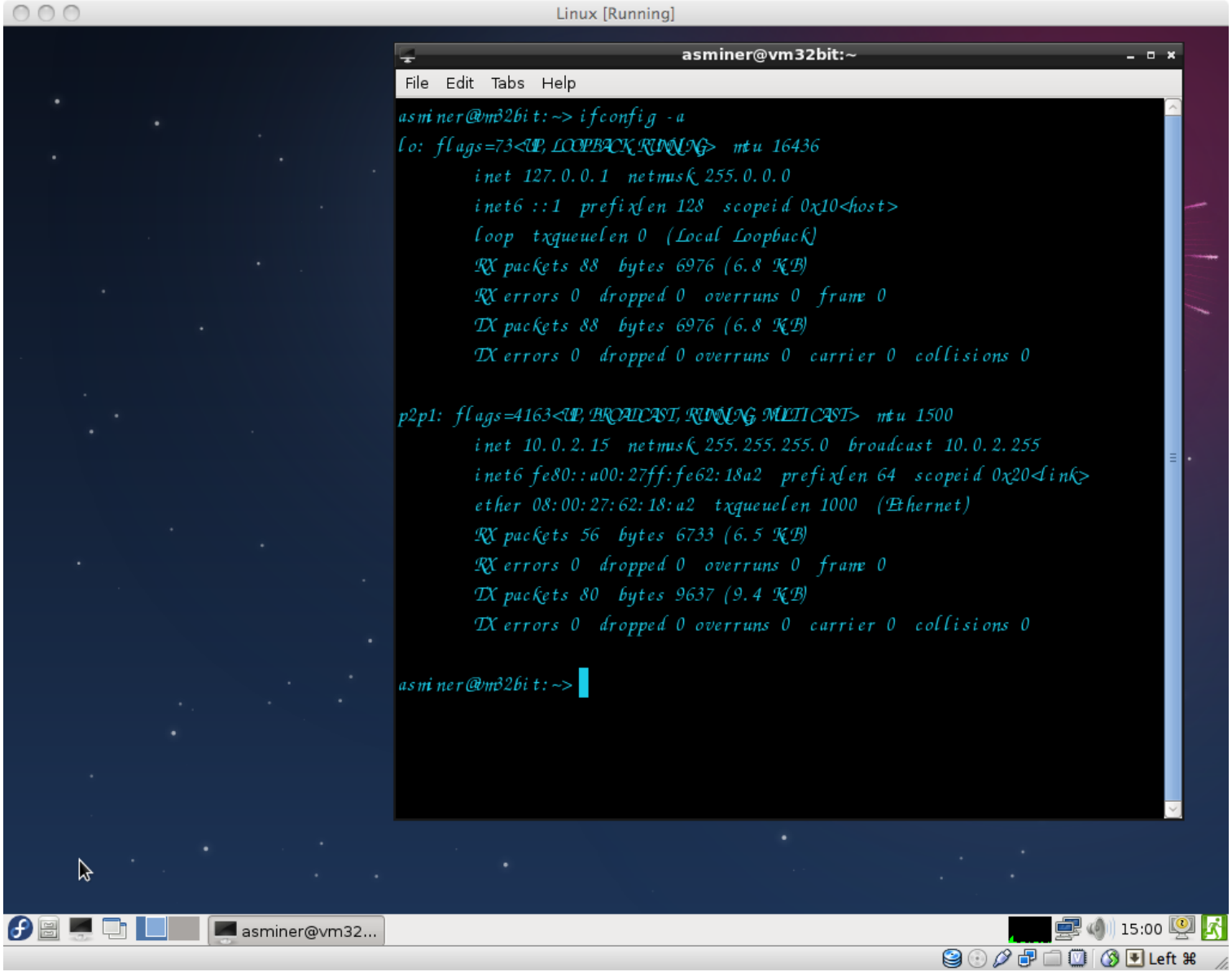
  All of these assume you are running X, of course
- These are easy to use — I assume you are already proficient

Command Line Interfaces (CLIs)

- Mainstream OS's *still* provide CLIs
- Proper term is command shell or simply shell
  - A shell is a thin wrapper of software around the OS kernel
  - A terminal window is a thin, GUI wrapper around a shell
- In Windows: Command Prompt
  - Very thin wrapper around a DOS shell
- In Mac OS X: Terminal (under `Applications/Utilities`)
  - Wrapper around a "UNIX" shell
- In Linux:
  - Many graphical terminals available
  - Around 6 text–based virtual terminals
    - Usually Alt-F*n* or Ctrl-Alt-F*n* to switch to console *n*
- These are tricker to use and are not as "portable"
  - Unix shells and DOS shells are different



Finding the command prompt in Windows XP



Linux terminal window running a shell; the terminal is responsible for the horrendous font

Linux [Running]

```
Fedora release 17 (Beefy Miracle)
Kernel 3.4.0-1.fc17.i686 on an i686 (tty1)

vm32bit login: asminer
Password:
Last login: Wed Jul 25 14:50:25 on tty1
asminer@vm32bit:~> ifconfig -a
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 16436
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 88  bytes 6976 (6.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 88  bytes 6976 (6.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

p2p1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe62:18a2  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:62:18:a2  txqueuelen 1000  (Ethernet)
        RX packets 135  bytes 13645 (13.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 159  bytes 15997 (15.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

asminer@vm32bit:~> _
```

Left ⌘

Linux virtual terminal ("pure text") running a shell

Why are CLIs still relevant after 40 years?

- *Sometimes* faster or easier to use CLI
- Interface has not changed much in *decades*
- Shells can be run remotely — efficiently
  - Remote shell – transfers just the text
  - Remote desktop – transfers the GUI
- Some machines (especially servers) do not run a GUI
- Shells are more powerful than GUIs
  - Remember— power of UNIX is
    - Simple utilities
    - Mechanism to combine these
  - Can do things that are tedious or impossible to do with GUI
- Great for system administration
  - Can automate repetative tasks

What is a shell

- A program to parse and execute commands
- Usually runs interactively
  - You get a prompt, e.g. (style may be changed)

    ```
    user@machine >
    [user@machine ]$
    ```

  - Type a command, press `return`
  - Command runs; then you get the prompt again
- The shell is just another user application
  - Usually lives in `/bin`
  - You can write your own shell in C/C++.

Basic shell usage

- A command has the form: `cmd arg1 arg2 ... argn`
  - `cmd` is a utility or command
  - `arg1 ... argn` are arguments
- If `cmd` is "built in", it is handled by the shell directly:
  1. Check arguments for correctness
  2. Execute command
- If `cmd` is not "built in", then it is a program
  1. Shell locates the program (if not found, you get an error)
  2. Shell runs the program and passes the arguments to it
  3. The program checks the arguments for correctness
  4. The program executes itself
- The built-in `type` will tell you if a `cmd` is built in or not:

```
prompt$ type type
type is a shell builtin
prompt$
```

Which shell?

- Some common shells in Linux:

`ash`	BSD version of `sh`
`bash`	Bourne Again SHell, default in Linux
	(The shell covered in lecture)
`sh`	original Bourne shell, c. 1977
	(before that – original shell by Ken Thompson)
`csh`	C shell (has C–like syntax)
`tcsh`	TENEX C shell
`ksh`	Korn shell
`zsh`	Enhanced ksh

- You can change your default login shell

Shells are fairly similar except for the more "advanced" features

The Shell so far...

To use the shell effectively, you need to know

1. Appropriate commands for what you want to do
   - There are *many* of these
   - We will cover some throughout the semester
2. For each command, its arguments and what they mean

The Shell so far...

To use the shell effectively, you need to know

1. Appropriate commands for what you want to do
   - There are *many* of these
   - We will cover some throughout the semester
2. For each command, its arguments and what they mean

Important!

- Yes, we will see lots of shell examples, but...
- The best way to learn is by doing

System manual

man : for "manual"

- Gives the manual pages about a command, utility, configuration file, system call, etc.
- Use: give the commands you want to know about, as arguments.
- To learn about the C function, `fopen`:

```
prompt$ man fopen
```

- To learn about `man` itself:

```
prompt$ man man
```

info : like `man`, but fancier

- Opens manual page in a viewer
- Has hyperlinks

Do I really need `man`? (Probably)

- Tells you what the arguments mean for a command
  - E.g., to copy a file, which do I use:

    `copy source destination`

    `copy destination source`
- More importantly: tells you about switches and options
  - Switches allow you to adjust behavior of a command
  - Typically have one of the following forms

    `-a`, `-B`: "short" switches

    `--longswitch`: "long" switches

    `-n10`, `-n 10`, `-n=10`: a switch with an argument
  - Order may or may not be important

    `mycmd -f -b` $\stackrel{?}{=}$ `mycmd -b -f`
  - Switches may sometimes be grouped

    `mycmd -a -l -F` $\stackrel{?}{=}$ `mycmd -alF`
  - Switches may or may not be case sensitive

    `mycmd -r` $\stackrel{?}{=}$ `mycmd -R`
  - Switches may be different on different systems!

```
man(1)                                                    man(1)

NAME
    man - format and display the on-line manual pages

SYNOPSIS
    man [-acdfFhkKtwW] [--path] [-m system] [-p string]
    [-C config_file] [-M pathlist] [-P pager] [-B
    browser] [-H htmlpager] [-S section_list] [section]
    name ...

DESCRIPTION
    man formats and displays the on-line manual pages.
    If you specify section, man only looks in that
:
```

Example: `man man`. Note:

```
man(1)                                                    man(1)

NAME
    man - format and display the on-line manual pages

SYNOPSIS
    man [-acdfFhkKtwW] [--path] [-m system] [-p string]
    [-C config_file] [-M pathlist] [-P pager] [-B
    browser] [-H htmlpager] [-S section_list] [section]
    name ...

DESCRIPTION
    man formats and displays the on-line manual pages.
    If you specify section, man only looks in that
:
```

Example: `man man`. Note:

(1) This is in section 1 of the manual.

```
man(1)                                                    man(1)

NAME
    man - format and display the on-line manual pages

SYNOPSIS
    man [-acdfFhkKtwW] [--path] [-m system] [-p string]
    [-C config_file] [-M pathlist] [-P pager] [-B
    browser] [-H htmlpager] [-S section_list] [section]
    name ...

DESCRIPTION
    man formats and displays the on-line manual pages.
    If you specify section, man only looks in that
:
```

Example: `man man`. Note:

(2) Anything in brackets is optional.

```
man(1)                                                    man(1)

NAME
    man - format and display the on-line manual pages

SYNOPSIS
    man [-acdfFhkKtwW] [--path] [-m system] [-p string]
    [-C config_file] [-M pathlist] [-P pager] [-B
    browser] [-H htmlpager] [-S section_list] [section]
    name ...

DESCRIPTION
    man formats and displays the on-line manual pages.
    If you specify section, man only looks in that
:
```

Example: `man man`. Note:

(3) These are short switches, and they can be grouped.

```
man(1)                                                    man(1)

NAME
    man - format and display the on-line manual pages

SYNOPSIS
    man [-acdfFhkKtwW] [--path] [-m system] [-p string]
    [-C config_file] [-M pathlist] [-P pager] [-B
    browser] [-H htmlpager] [-S section_list] [section]
    name ...

DESCRIPTION
    man formats and displays the on-line manual pages.
    If you specify section, man only looks in that
:
```

Example: `man man`. Note:

(4) This is a long switch.

```
man(1)                                                    man(1)

NAME
    man - format and display the on-line manual pages

SYNOPSIS
    man [-acdfFhkKtwW] [--path] [-m system] [-p string]
    [-C config_file] [-M pathlist] [-P pager] [-B
    browser] [-H htmlpager] [-S section_list] [section]
    name ...

DESCRIPTION
    man formats and displays the on-line manual pages.
    If you specify section, man only looks in that
:
```

Example: `man man`. Note:

(5) These switches take arguments.

```
man(1)                                                    man(1)

NAME
    man - format and display the on-line manual pages

SYNOPSIS
    man [-acdfFhkKtwW] [--path] [-m system] [-p string]
    [-C config_file] [-M pathlist] [-P pager] [-B
    browser] [-H htmlpager] [-S section_list] [section]
    name ...

DESCRIPTION
    man formats and displays the on-line manual pages.
    If you specify section, man only looks in that
:
```

Example: `man man`. Note:

(6) Output is fed through a pager; use arrows to scroll, q to quit

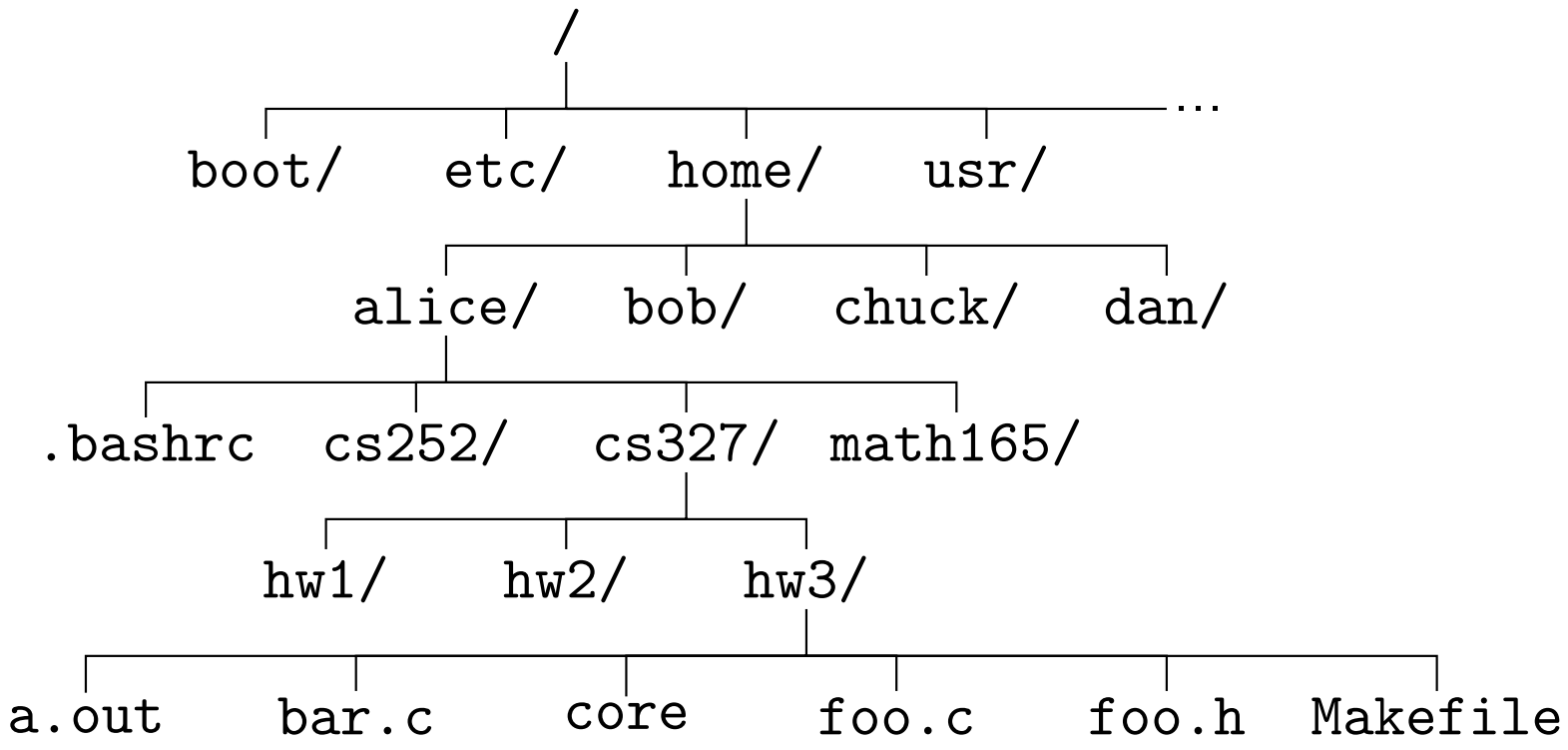
Useful switches for man

- -S : Specify a list of sections to check
  - Default is to check all sections
- -a : Print all matching pages (e.g., in multiple sections)
  - Default: only the first matching page is shown
- -k : Check all pages for keywords
  - Use this when you don't know which command

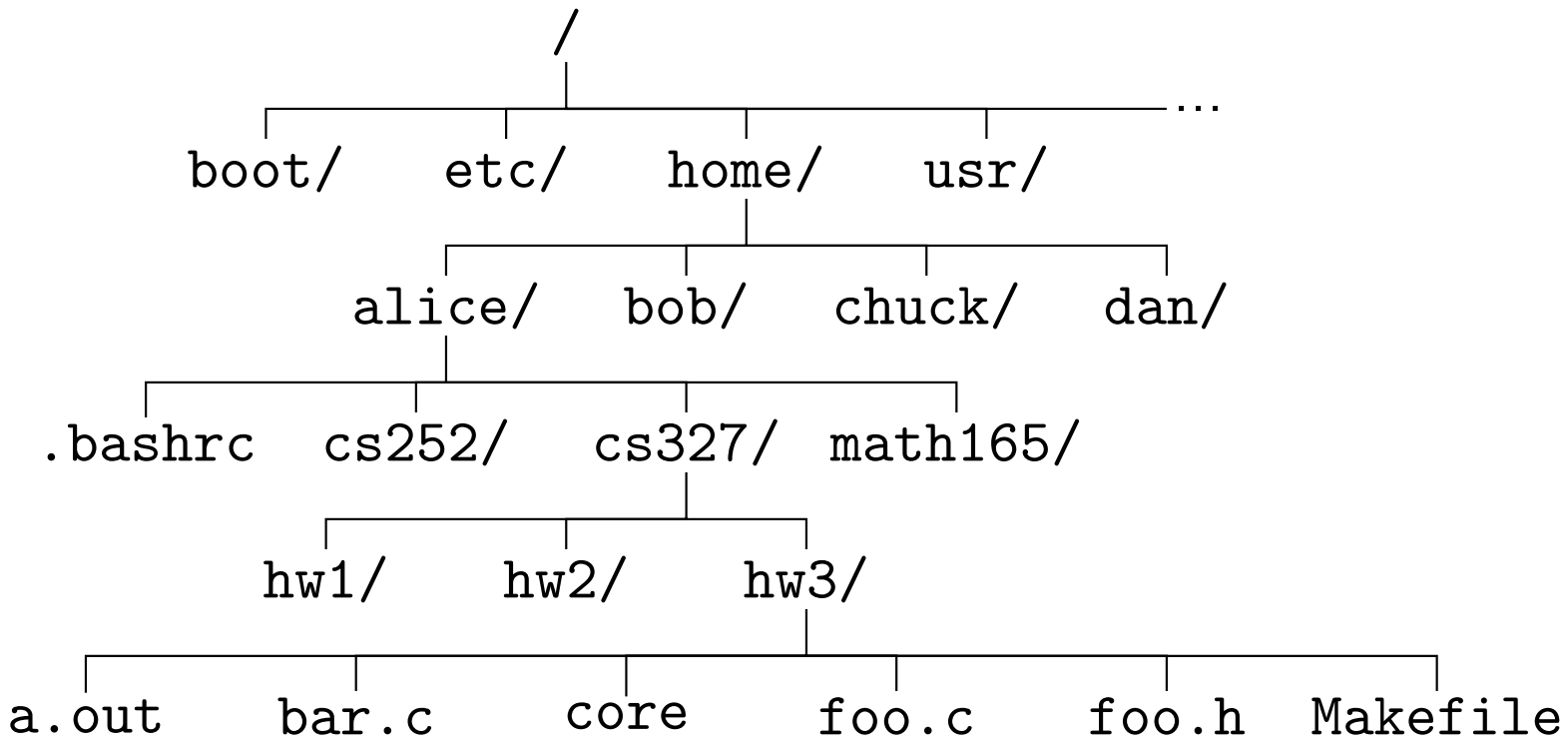
E.g., want to know something about a floppy drive:

```
prompt$ man -k floppy
fdformat (8)    - low-level format a floppy disk
floppy (8)      - format floppy disks
mbadblocks(1)   - tests a floppy disk, and marks the bad
mformat (1)     - add an MSDOS filesystem to a low-level
prompt$
```

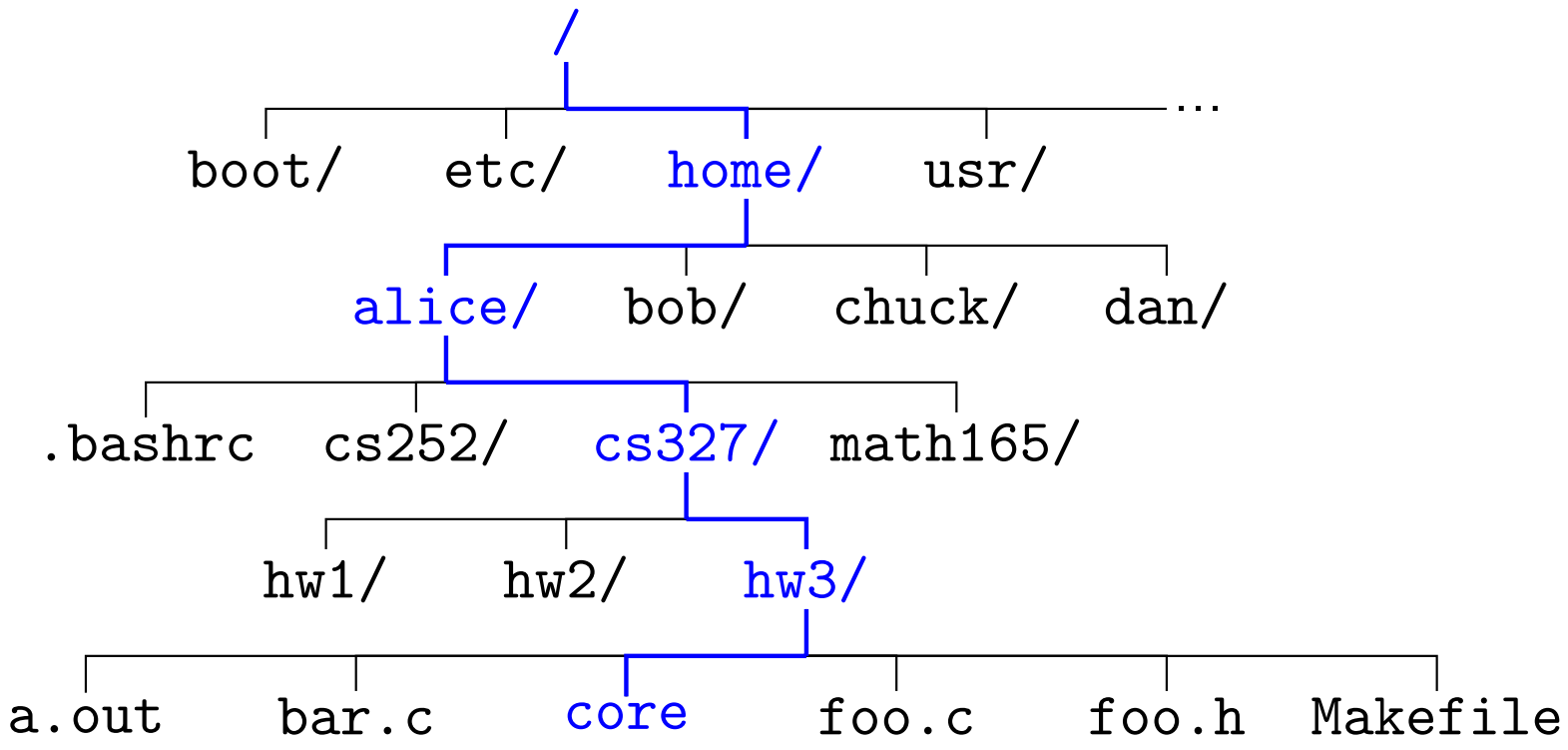
For more information, read the man page



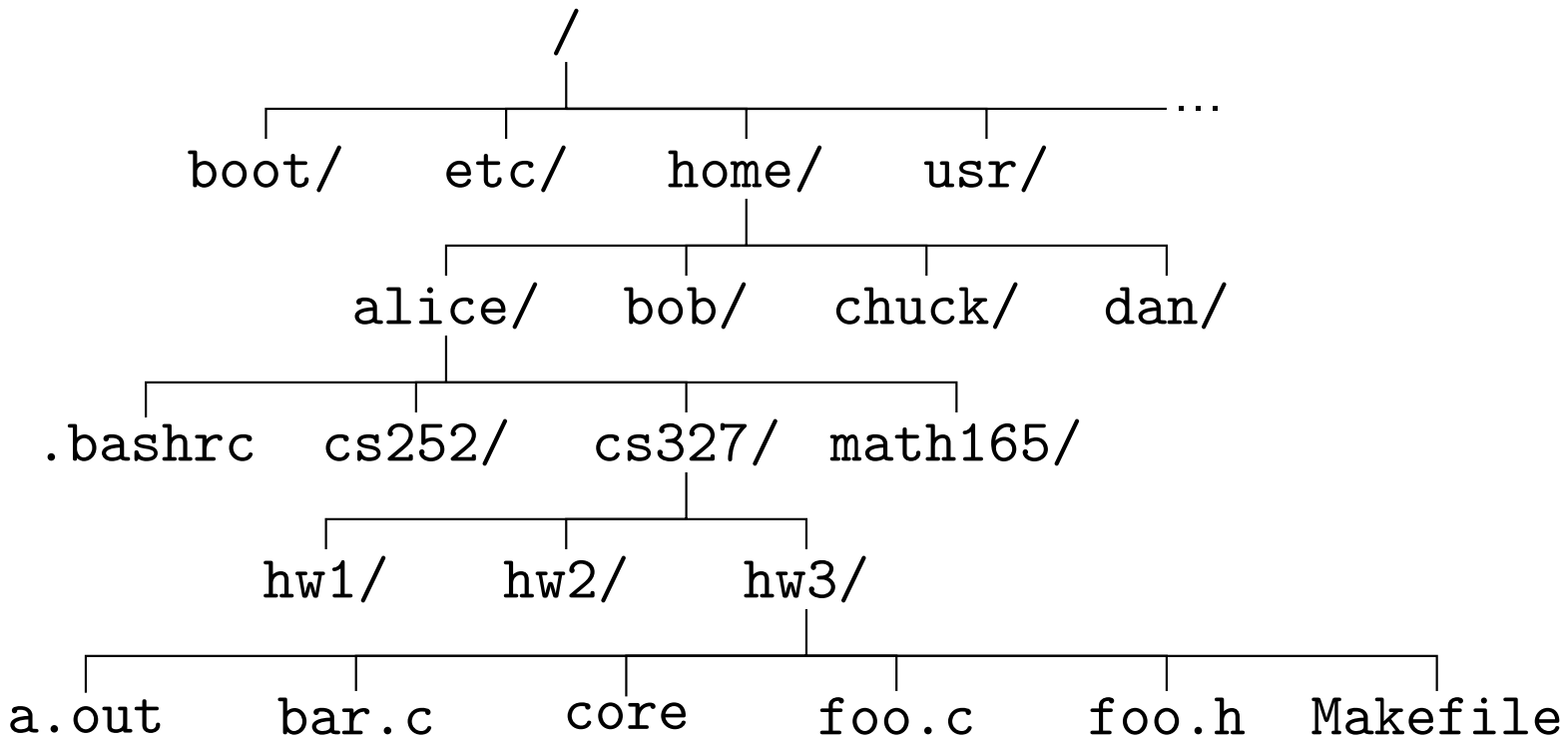
- Linux files are organized as a single tree



- Linux files are organized as a single tree
- Downward path in the tree → a file or folder



- Linux files are organized as a single tree
- Downward path in the tree → a file or folder
  - Example: `/home/alice/cs327/hw3/core`



- Linux files are organized as a single tree
- Downward path in the tree → a file or folder
  - Example: `/home/alice/cs327/hw3/core`
- Absolute pathname: path starting with /

Relative pathnames

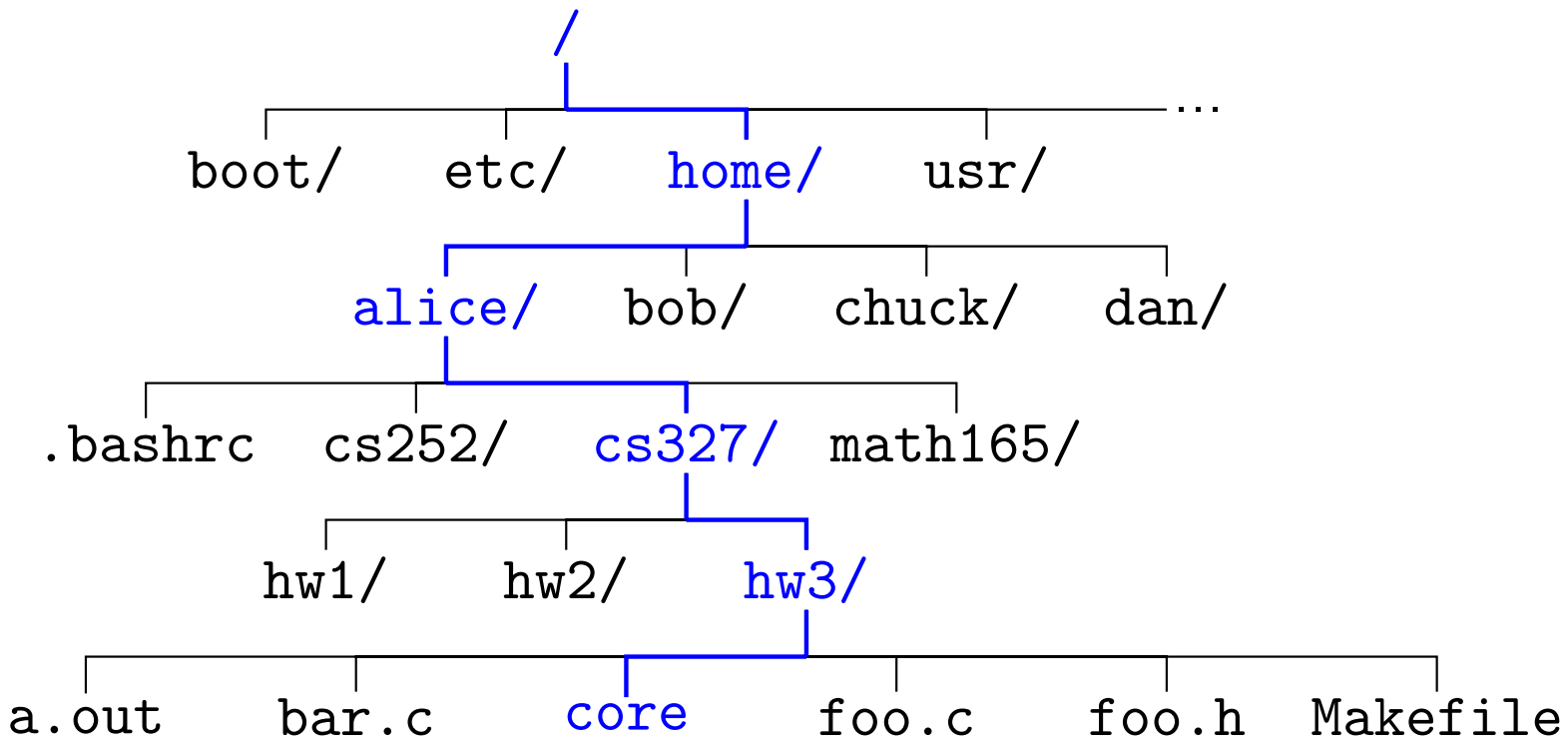
- Typing absolute pathnames is tedious
- Can instead use relative pathnames
  - Paths that do not start at the root
  - Pathname does not start with /
  - Paths are "relative" to some point
- How to specify the starting point?

Relative pathnames

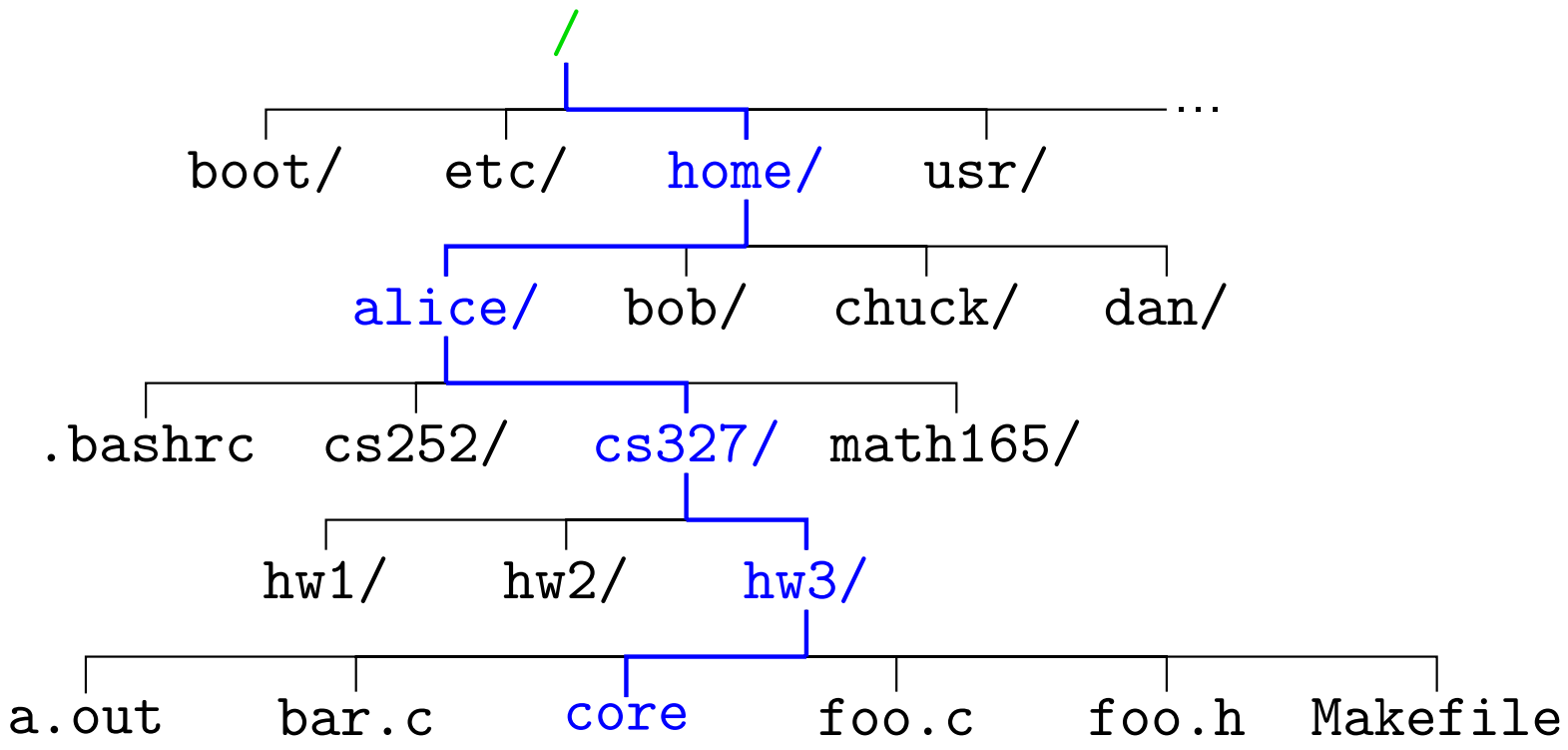
- Typing absolute pathnames is tedious
- Can instead use relative pathnames
  - Paths that do not start at the root
  - Pathname does not start with /
  - Paths are "relative" to some point
- How to specify the starting point?

Working directory

- The "starting point" for relative paths
- Can be changed in the shell...

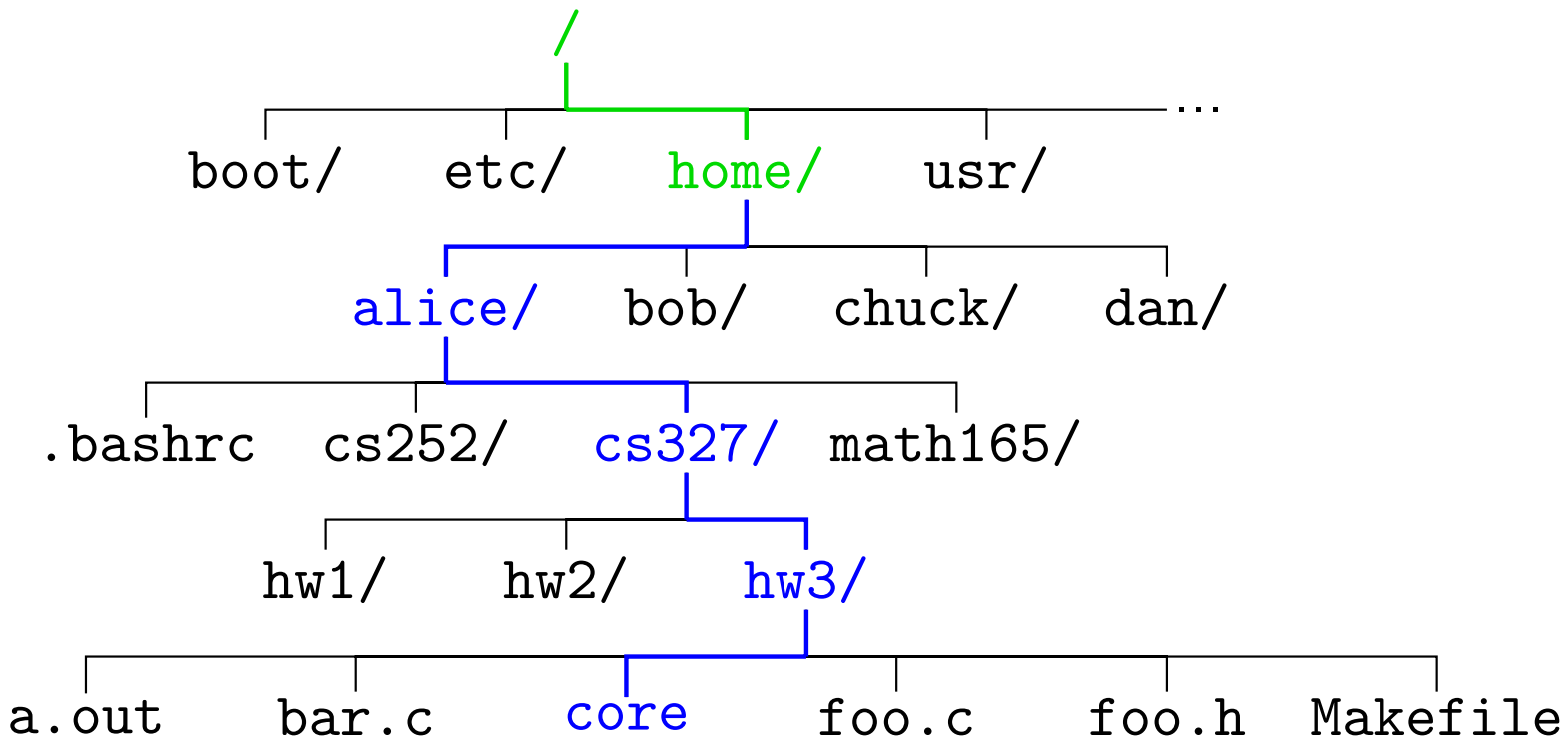


Absolute path: `/home/alice/cs327/hw3/core`



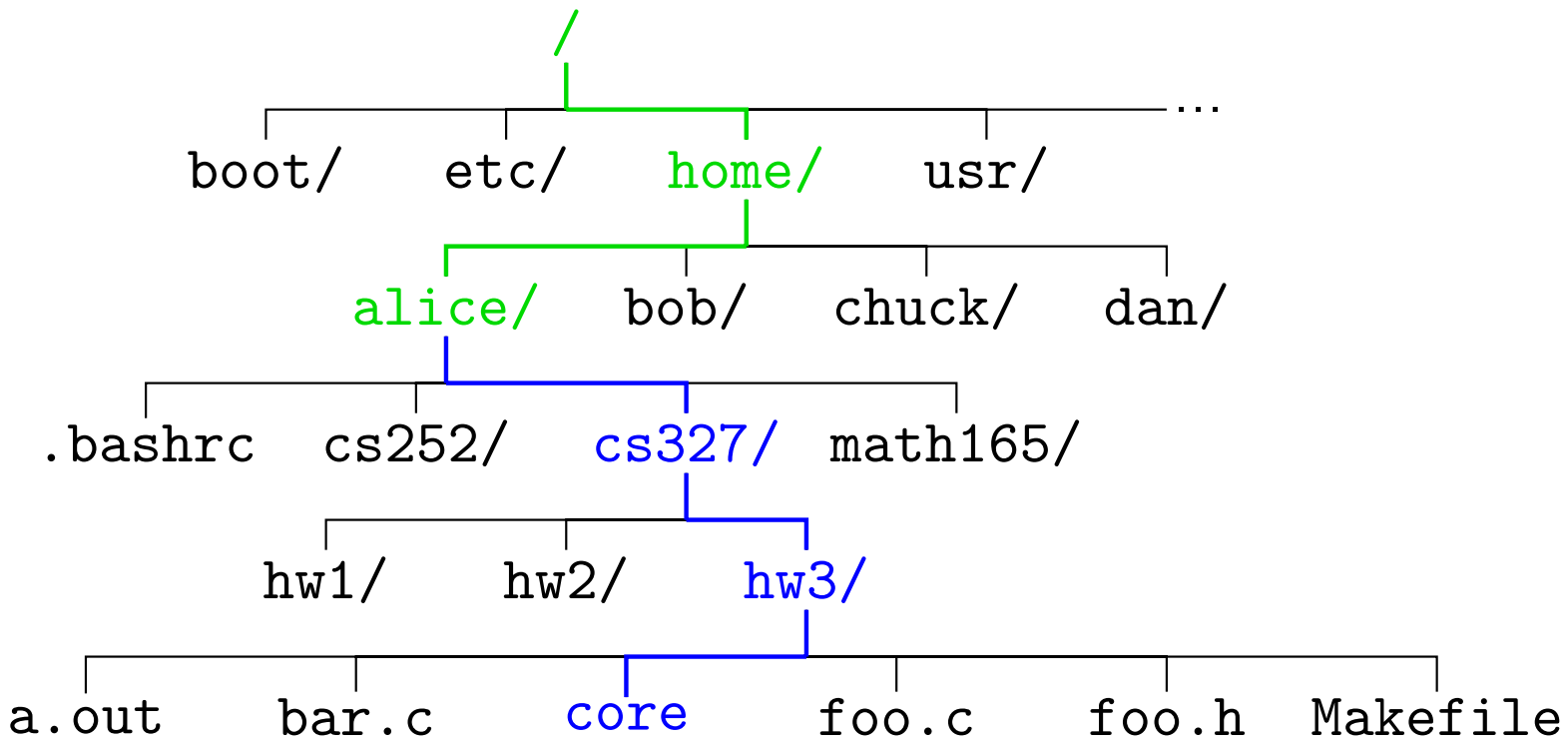
Absolute path: `/home/alice/cs327/hw3/core`

Working directory	Relative pathname
/	home/alice/cs327/hw3/core



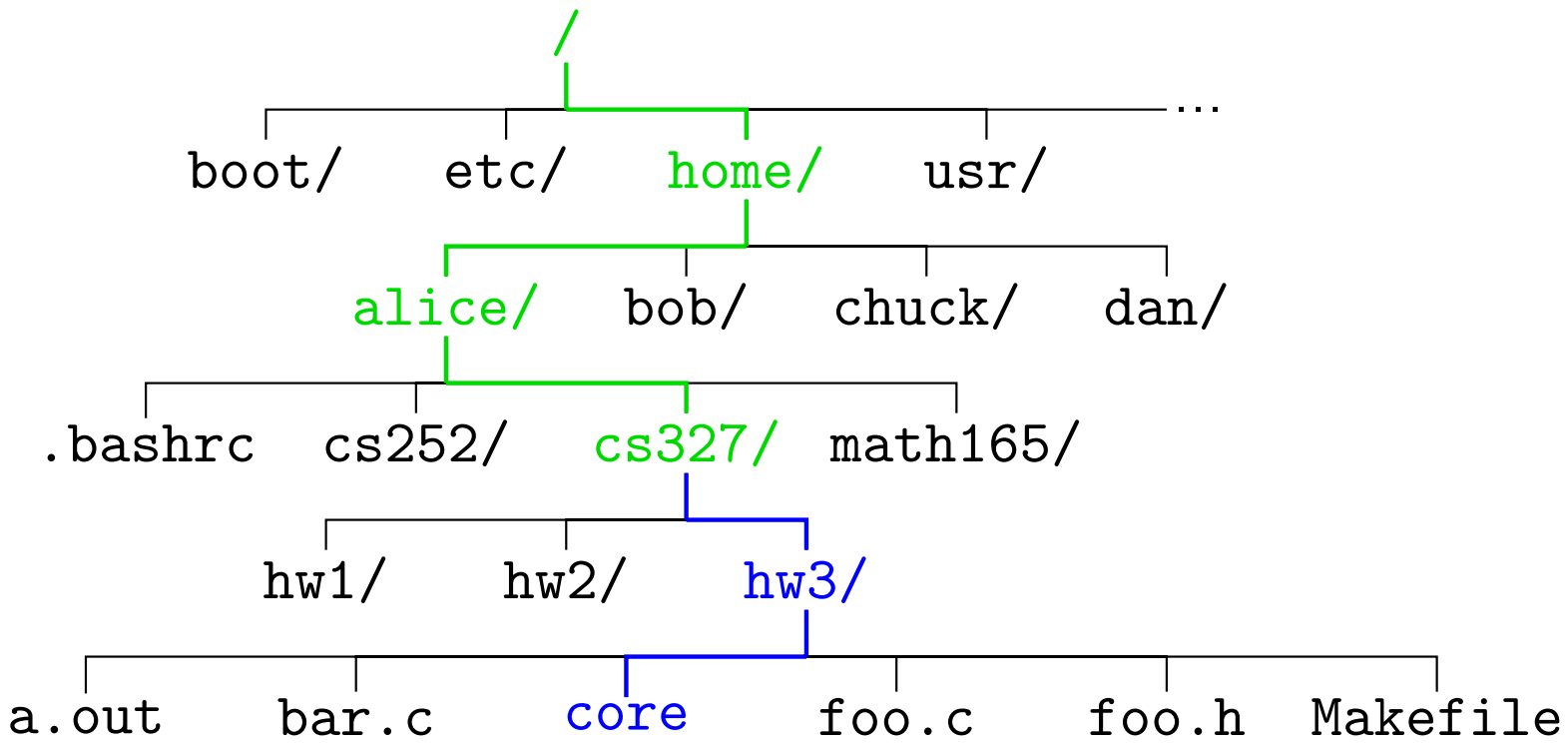
Absolute path: `/home/alice/cs327/hw3/core`

Working directory	Relative pathname
`/home/`	`alice/cs327/hw3/core`



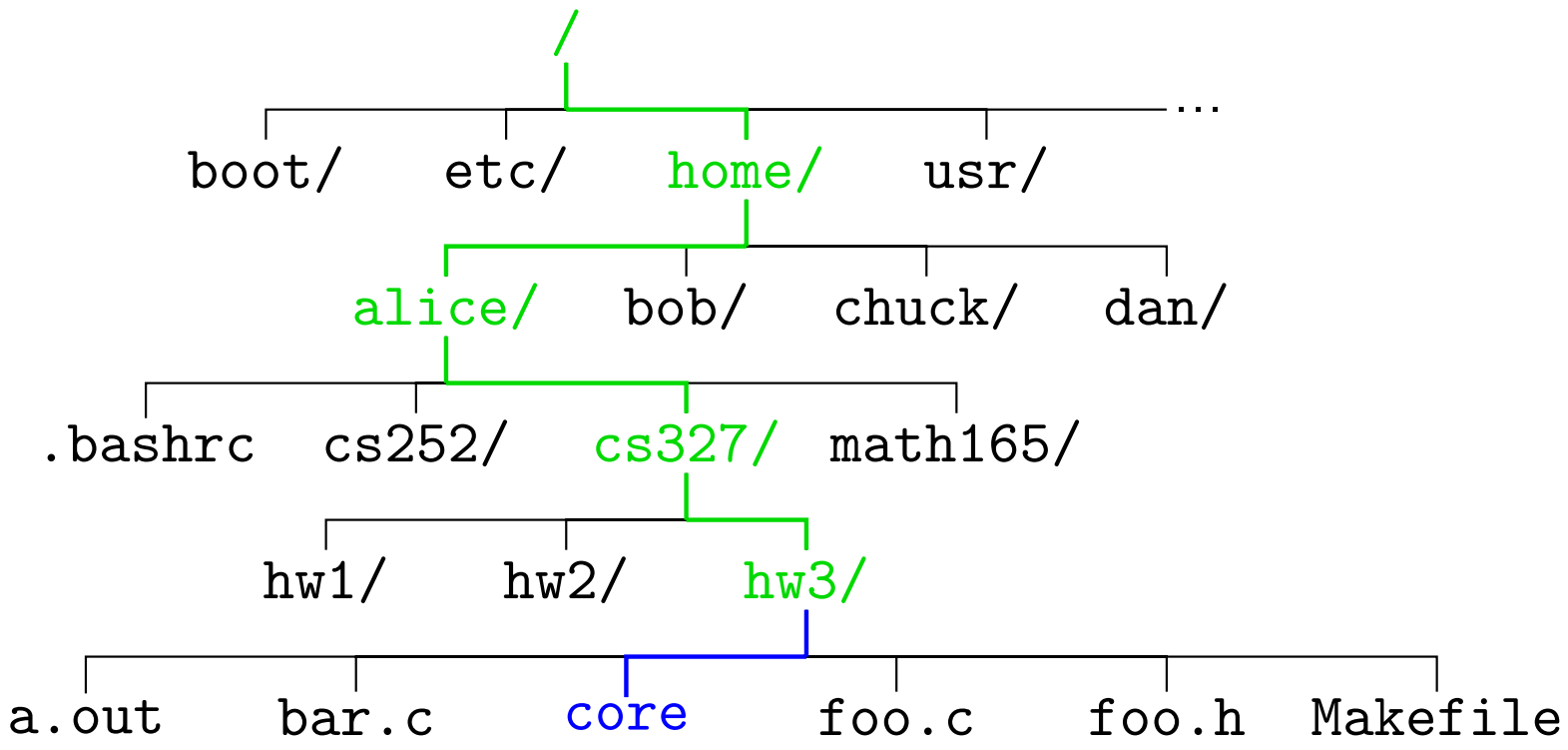
Absolute path: `/home/alice/cs327/hw3/core`

Working directory	Relative pathname
`/home/alice/`	`cs327/hw3/core`



Absolute path: `/home/alice/cs327/hw3/core`

Working directory	Relative pathname
`/home/alice/cs327/`	`hw3/core`

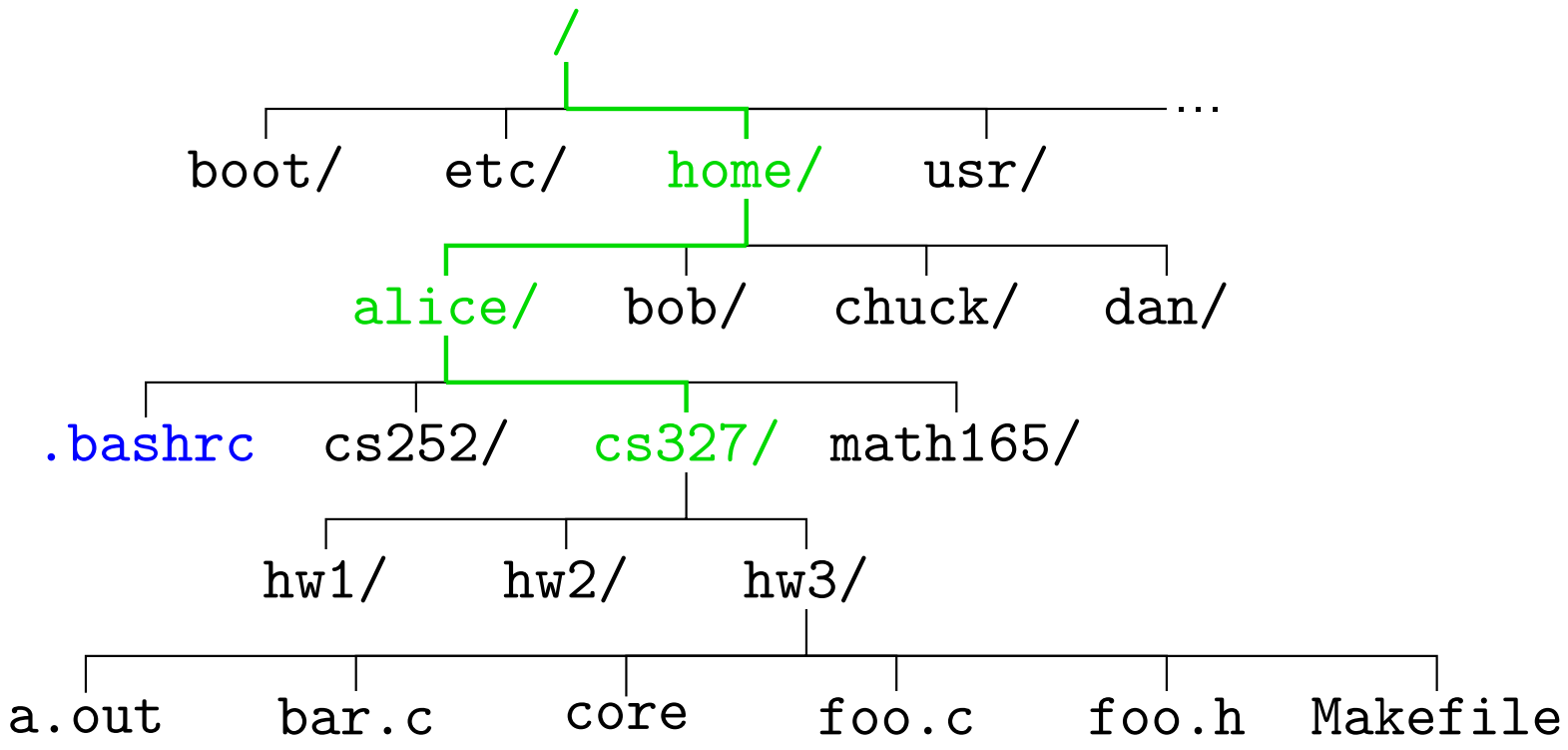


Absolute path: `/home/alice/cs327/hw3/core`

Working directory	Relative pathname
`/home/alice/cs327/hw3`	`core`

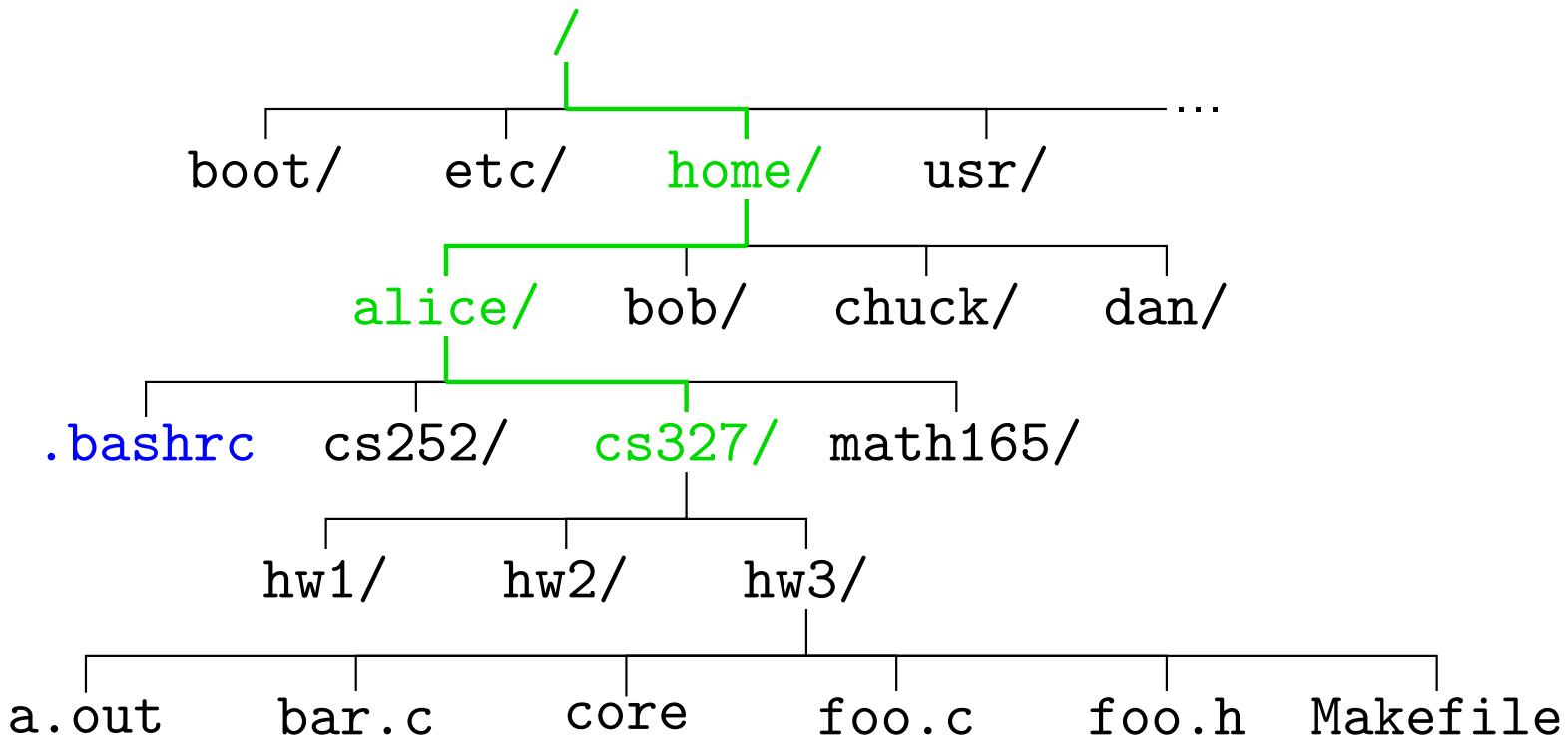
Special directories

- `.` : current directory
  - Useful for relative paths
  - May appear in absolute paths
- `..` : parent directory ("up one")
  - Useful for relative paths
  - May appear in absolute paths
  - In root directory, acts like "."
- `~` : current user's home directory
  - Only valid at the start of a path
  - Expanded by the shell
- `~user` : another user's home directory
  - Only valid at the start of a path
  - Expanded by the shell



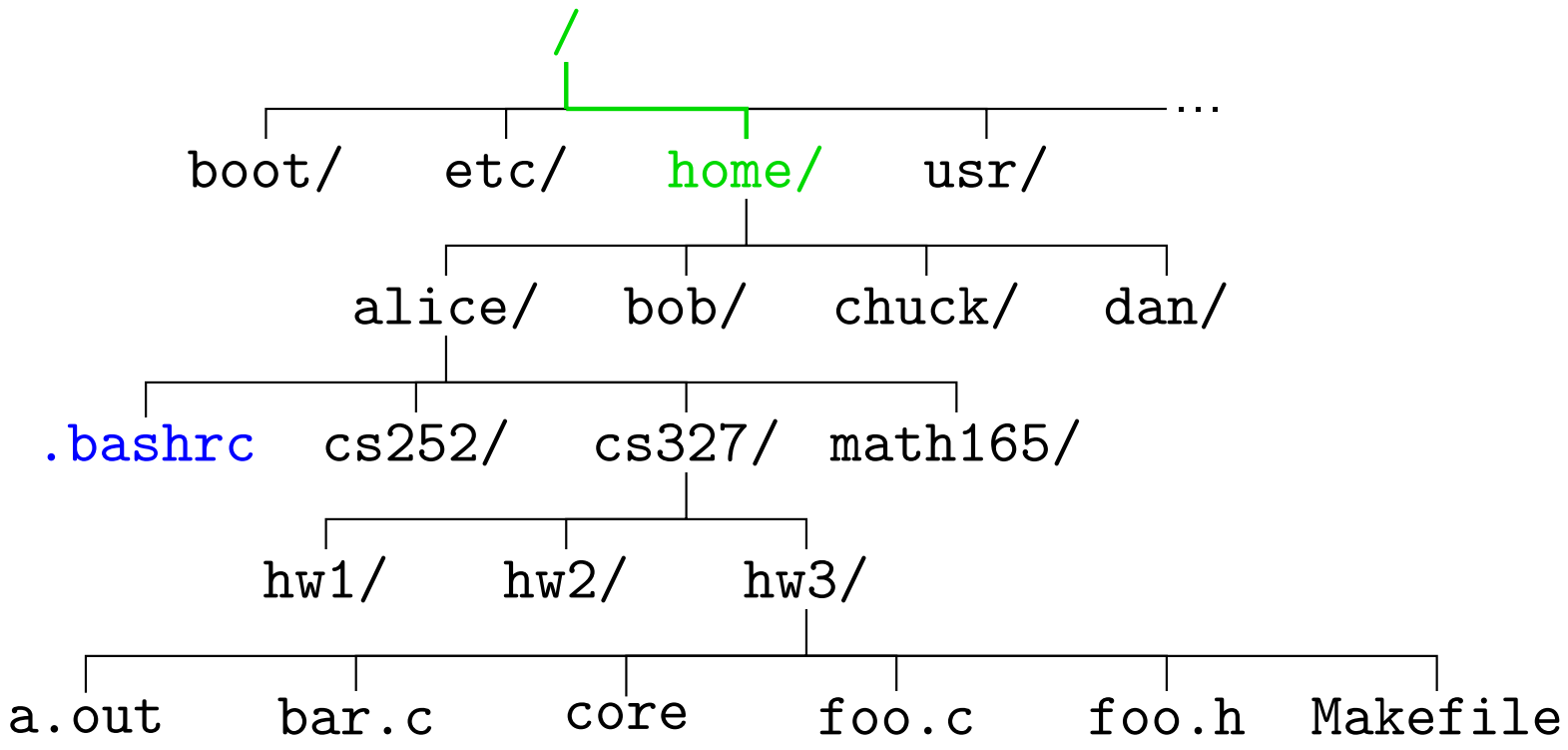
Some pathnames for user `alice` in `w.d.` /home/alice/cs327

1. `/home/alice/.bashrc`
2. `../.bashrc`
3. `~/.bashrc`
4. `hw3/../../../alice/././.bashrc`



Some pathnames for user bob in w.d. /home/alice/cs327

1. /home/alice/.bashrc
2. ~/../alice/.bashrc
3. ~alice/.bashrc
4. ../cs252/../.bashrc



Some pathnames for user `alice` in w.d. `/home`

1. `/home/alice/.bashrc`
2. `alice/.bashrc`
3. `~alice/.bashrc`
4. `../../../../home/./alice/./.bashrc`

Fun digression

What does the pathname `/.` refer to?

Fun digression

What does the pathname `/.` refer to?

- The root directory of the filesystem

Fun digression

What does the pathname `/.` refer to?

- The root directory of the filesystem
- Thus the name, slashdot
  - For those who read `slashdot.org`

Commands for the working directory

pwd: print working directory

```
prompt$ pwd
/home/alice
prompt$
```

Commands for the working directory

pwd: print working directory

```
prompt$ pwd
/home/alice
prompt$
```

cd: change working directory

- Single argument: pathname of new directory
  - Can be absolute or relative pathname
    - This is the case for most utilities

Home directory

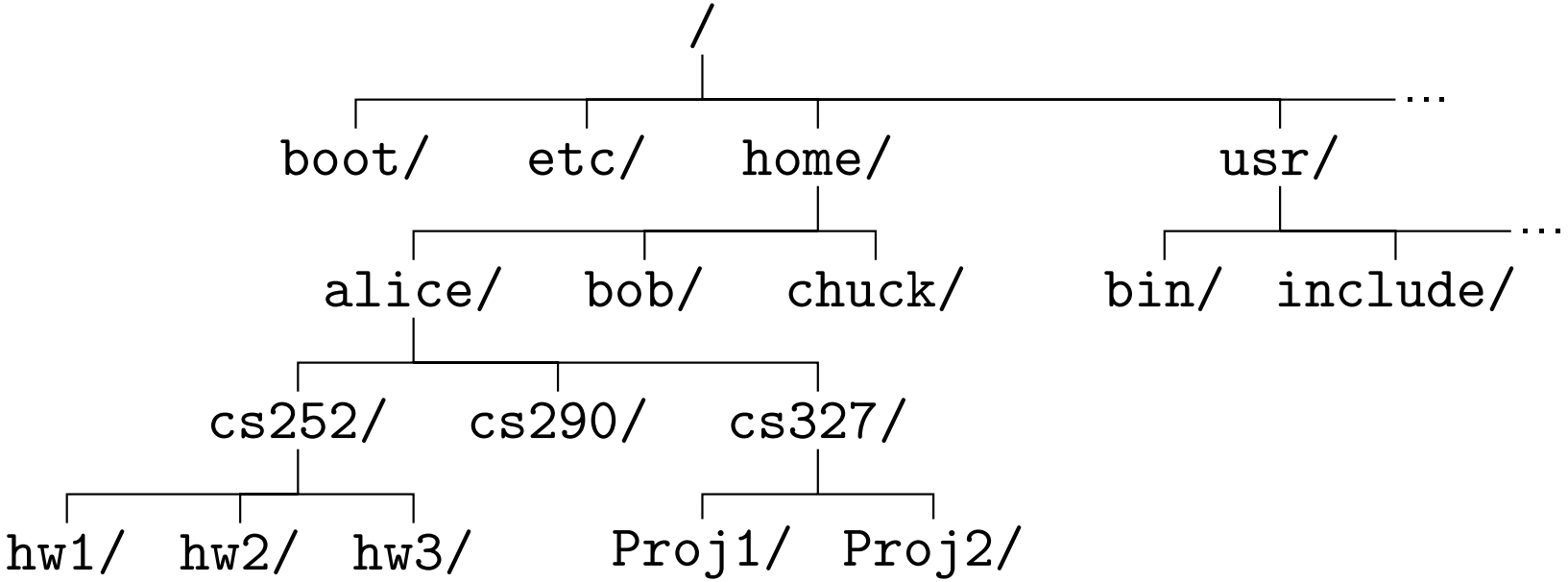
- Initial working directory when you
  - Login (remotely or in a text terminal)
  - Open a terminal window
- Preferred place for a user's files
  - Or in subdirectories...
- To get back home:

```
prompt$ cd ~
```

- Usually, this works too:

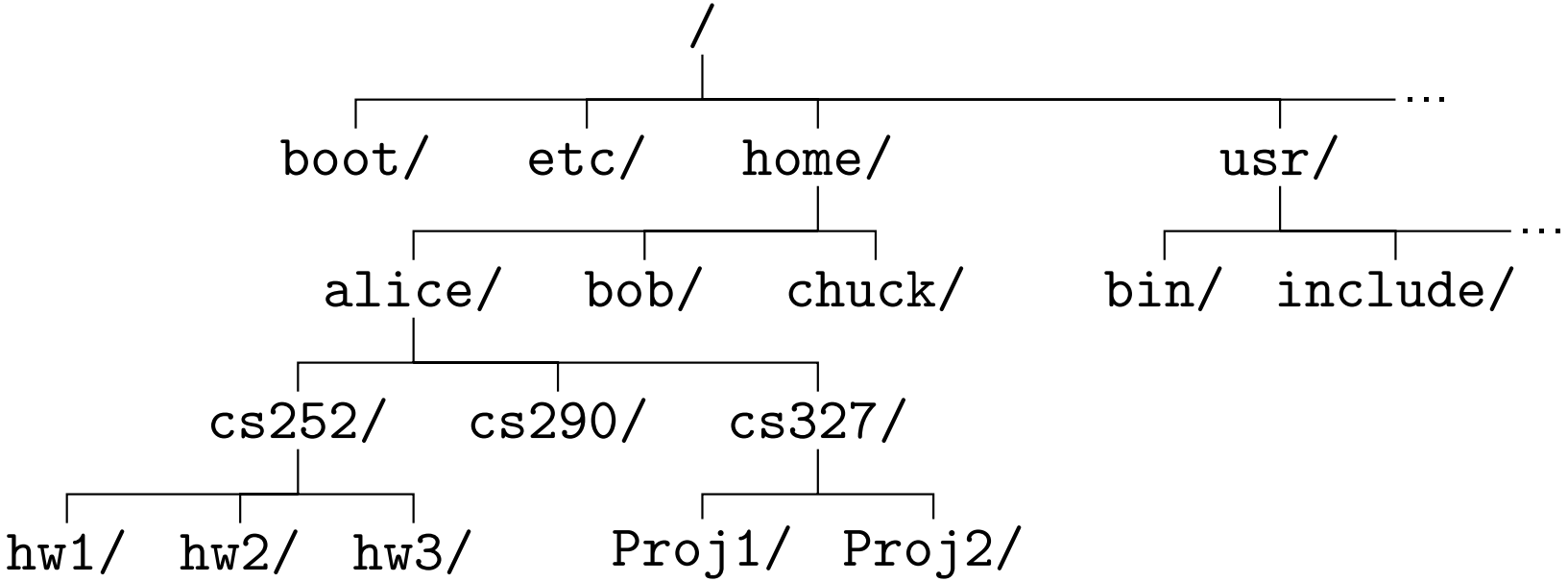
```
prompt$ cd
```

Examples: cd and pwd



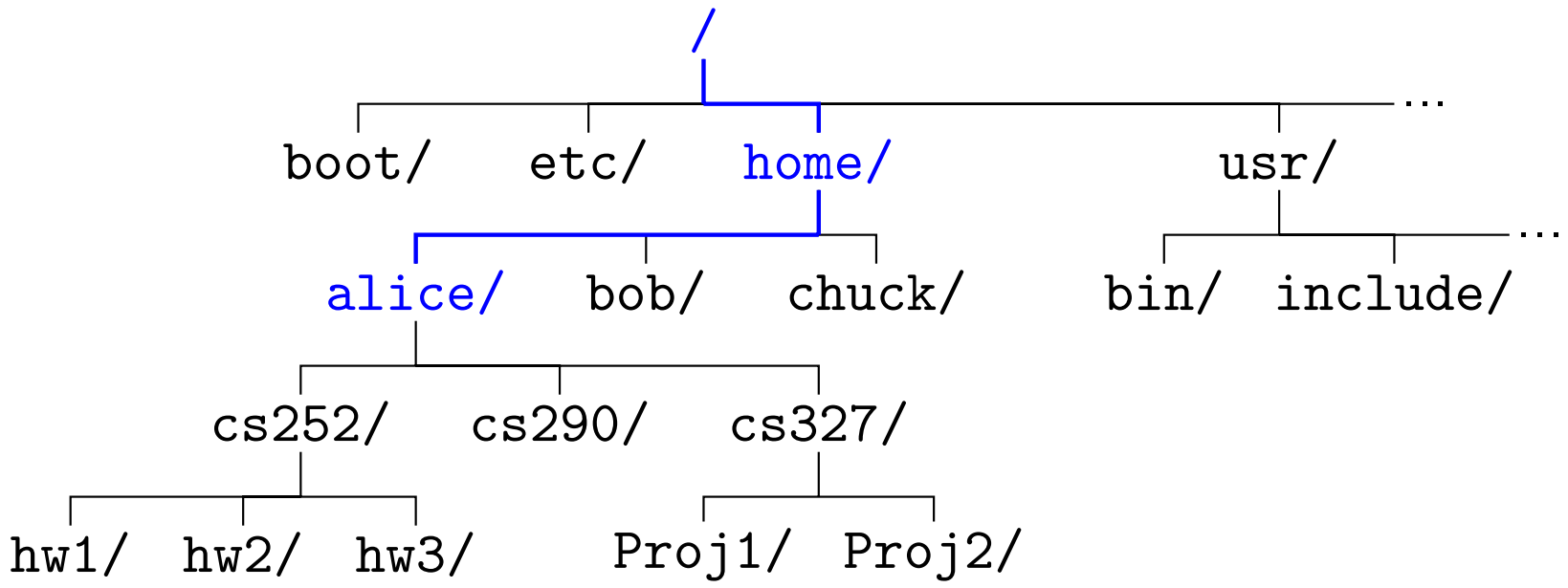

```
prompt$
```

Examples: cd and pwd



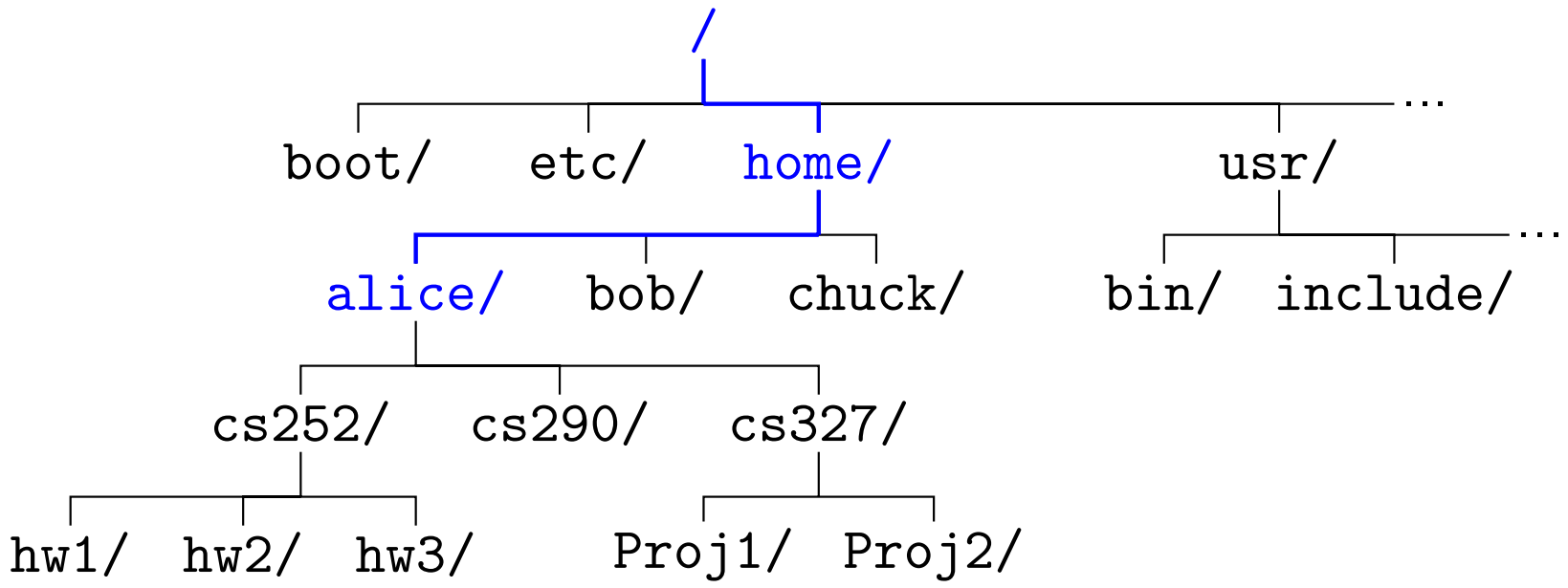

```
prompt$ pwd
```

Examples: cd and pwd



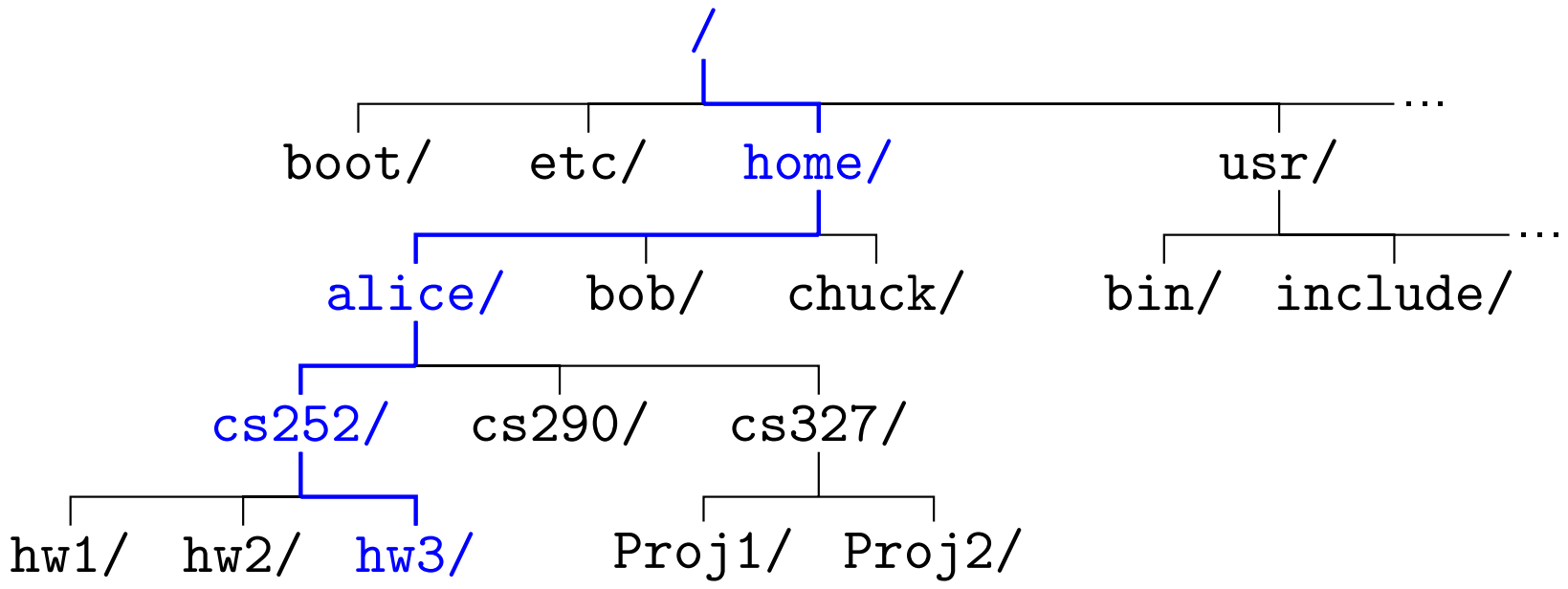

```
prompt$ pwd
/home/alice
prompt$
```

Examples: cd and pwd



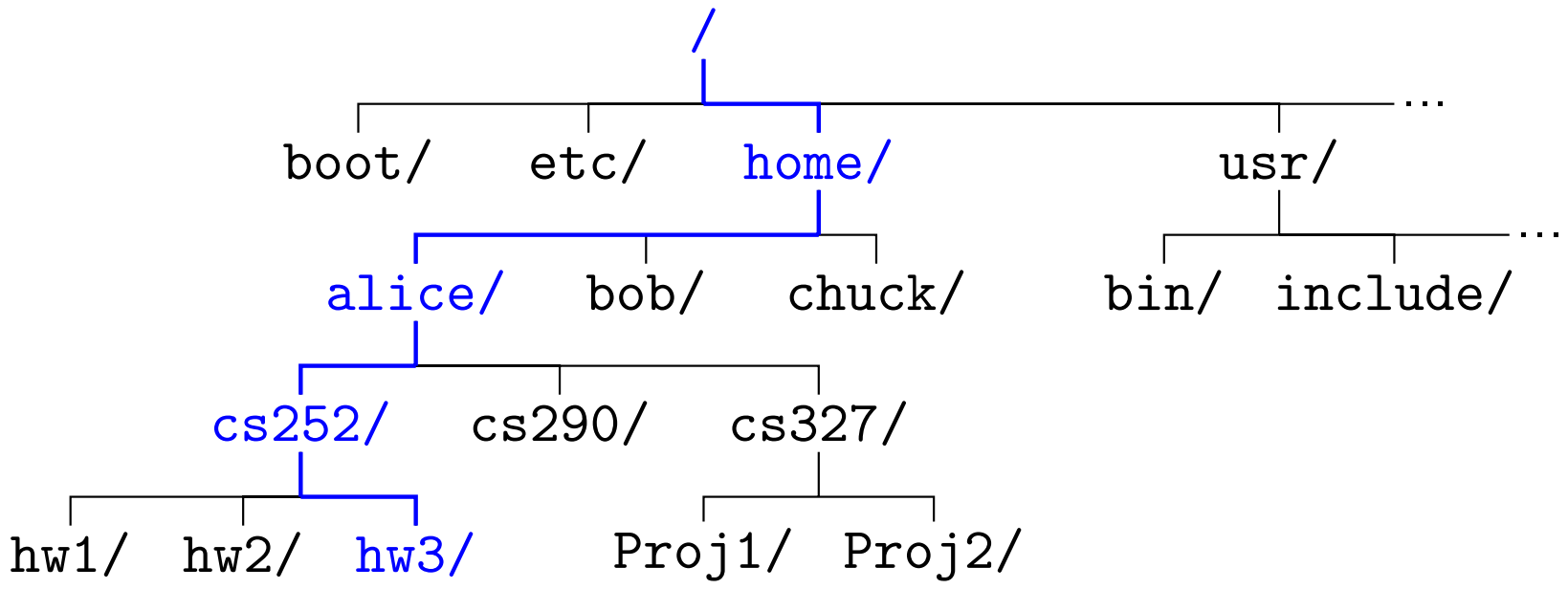

```
prompt$ pwd
/home/alice
prompt$ cd cs252/hw3
```

Examples: cd and pwd



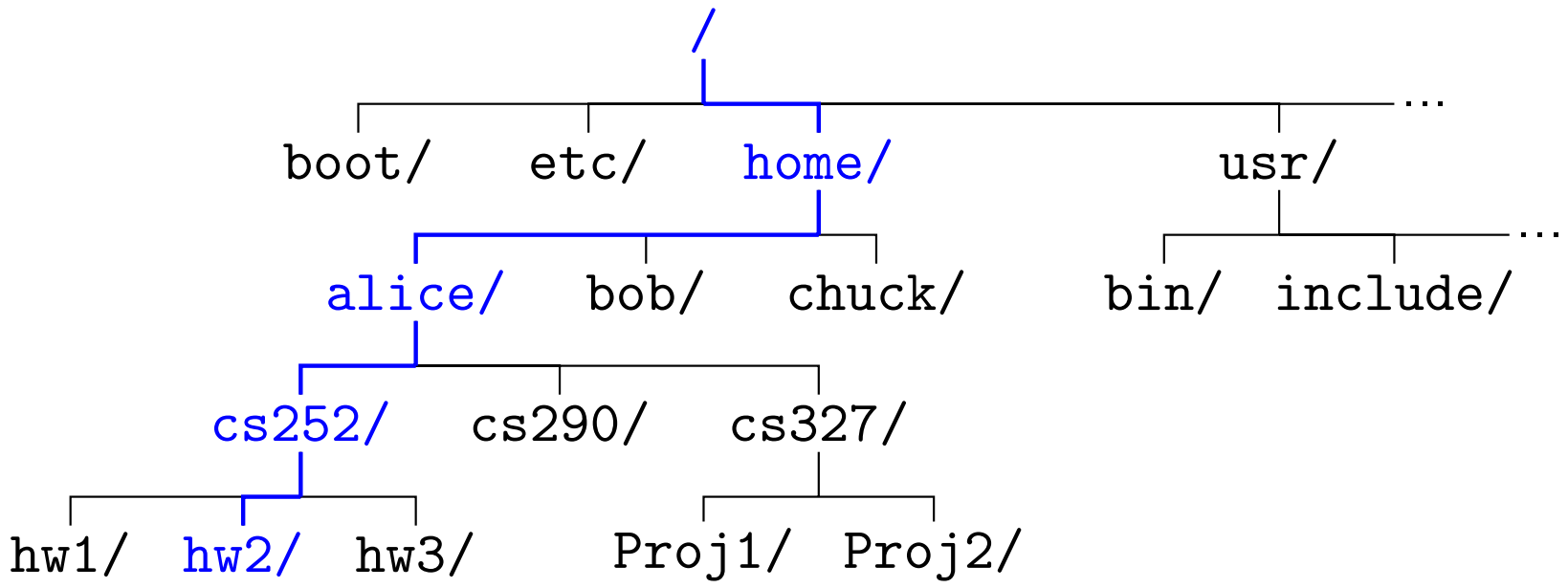

```
prompt$ pwd
/home/alice
prompt$ cd cs252/hw3
prompt$
```

Examples: cd and pwd



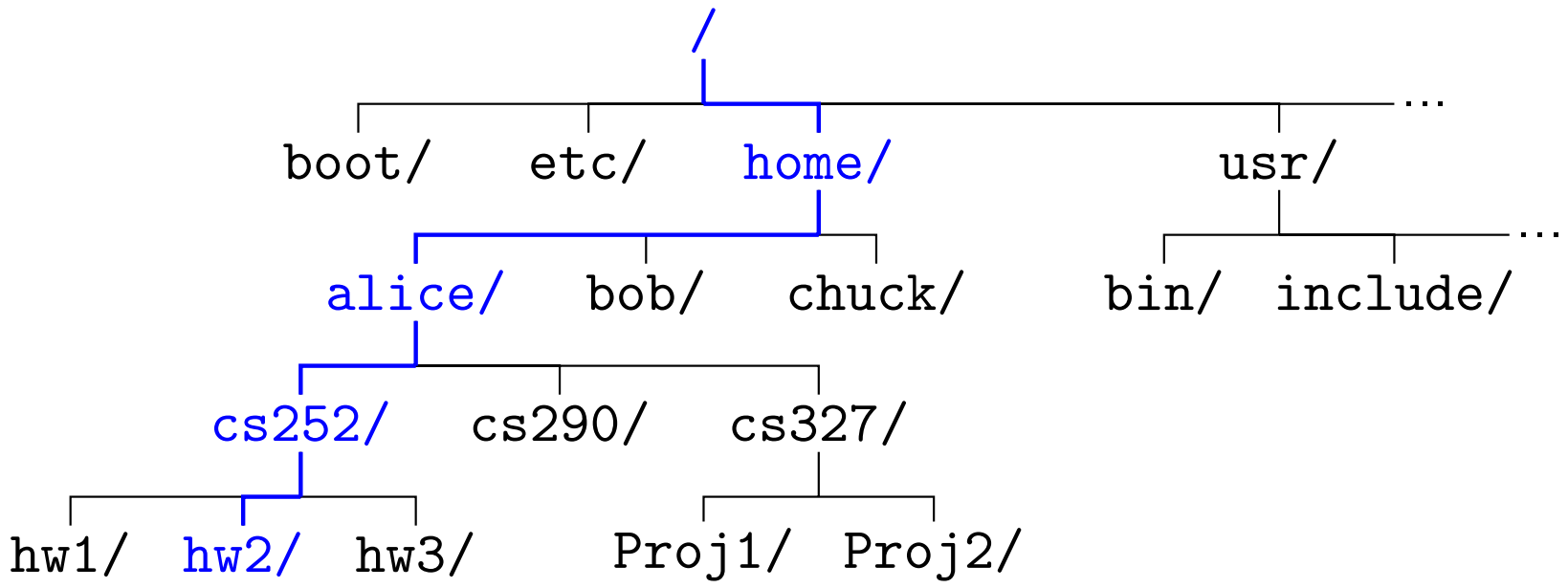

```
prompt$ pwd
/home/alice
prompt$ cd cs252/hw3
prompt$ cd ../hw2
```

Examples: cd and pwd



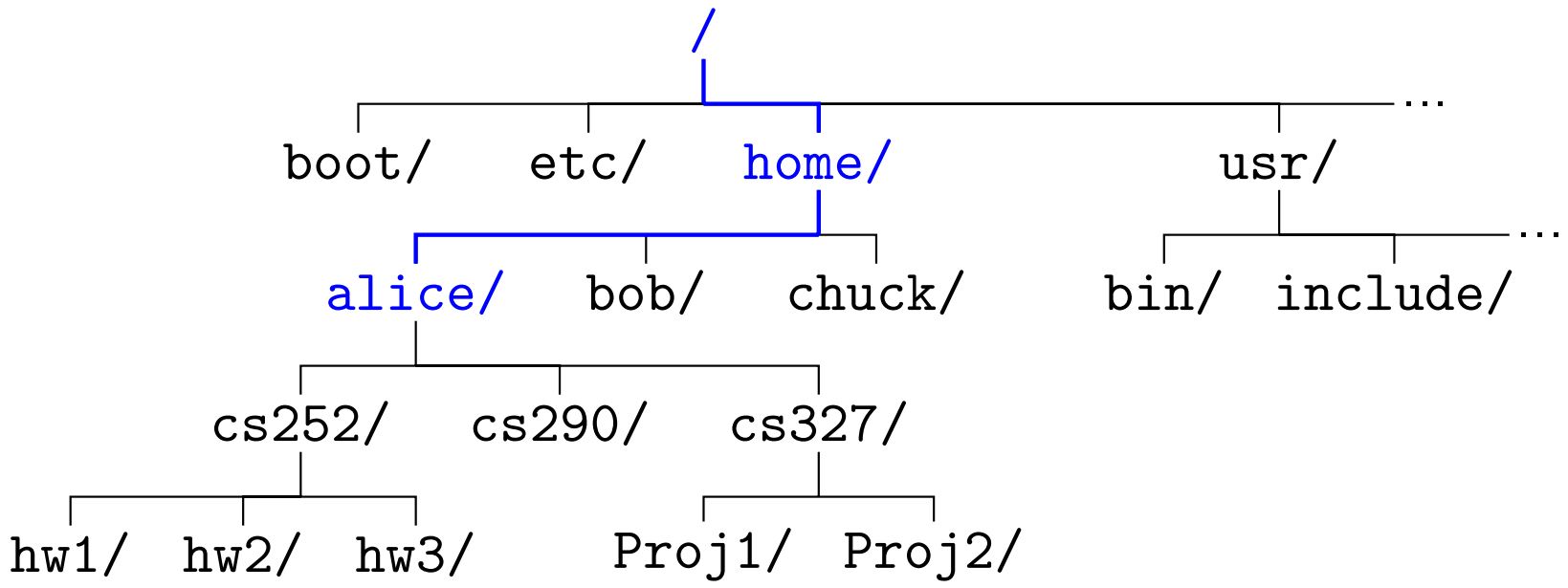

```
prompt$ pwd
/home/alice
prompt$ cd cs252/hw3
prompt$ cd ../hw2
prompt$
```

Examples: cd and pwd



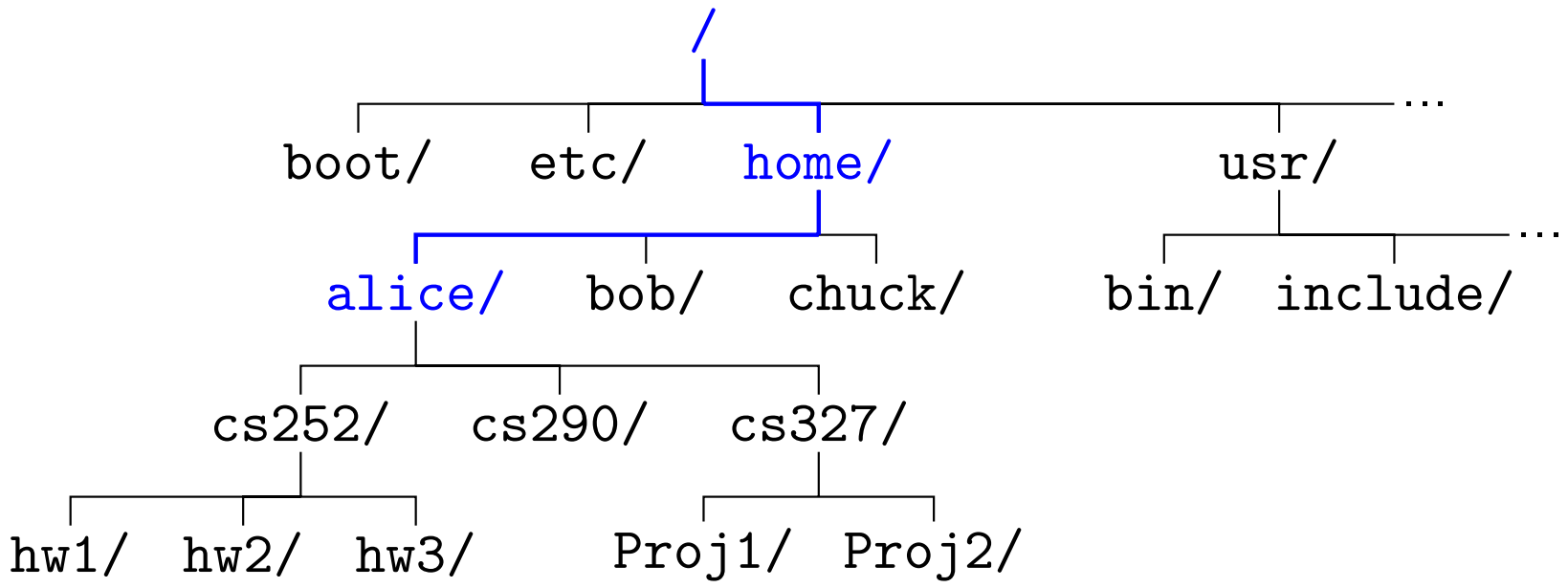

```
prompt$ pwd
/home/alice
prompt$ cd cs252/hw3
prompt$ cd ../hw2
prompt$ cd ../..
```

Examples: cd and pwd



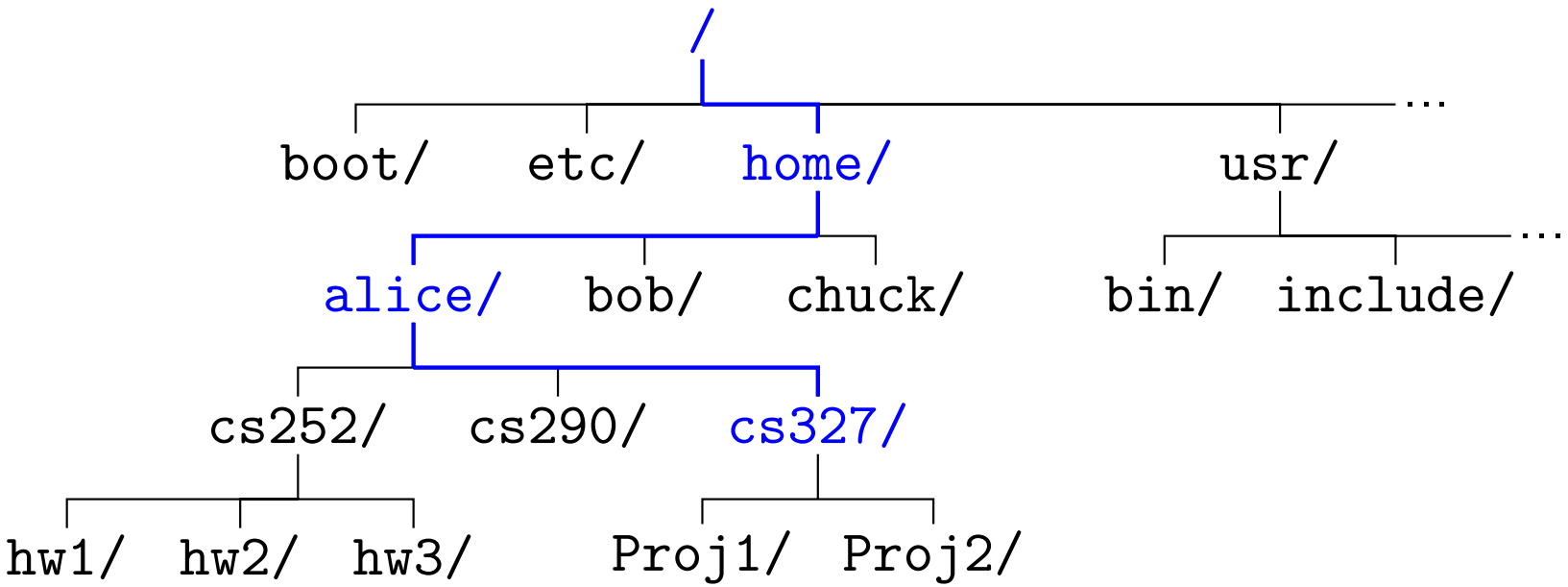

```
prompt$ pwd
/home/alice
prompt$ cd cs252/hw3
prompt$ cd ../hw2
prompt$ cd ../..
prompt$
```

Examples: cd and pwd



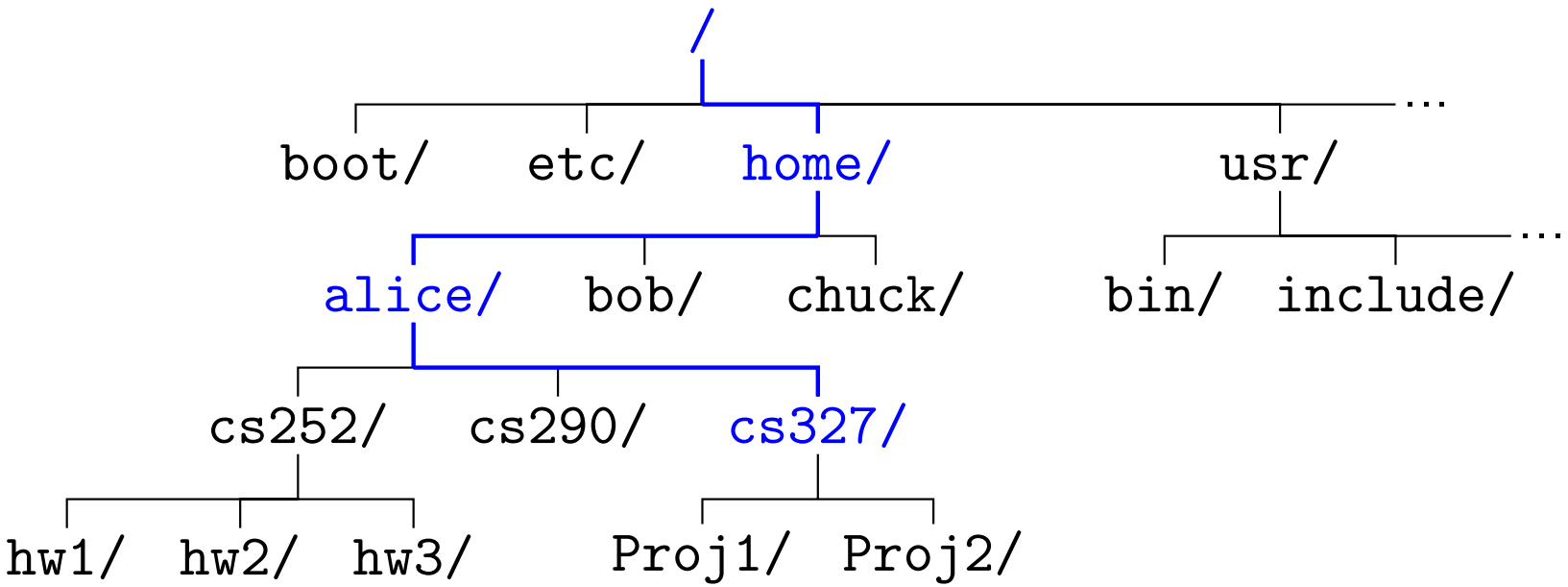

```
prompt$ pwd
/home/alice
prompt$ cd cs252/hw3
prompt$ cd ../hw2
prompt$ cd ../..
prompt$ cd cs327
```

Examples: cd and pwd



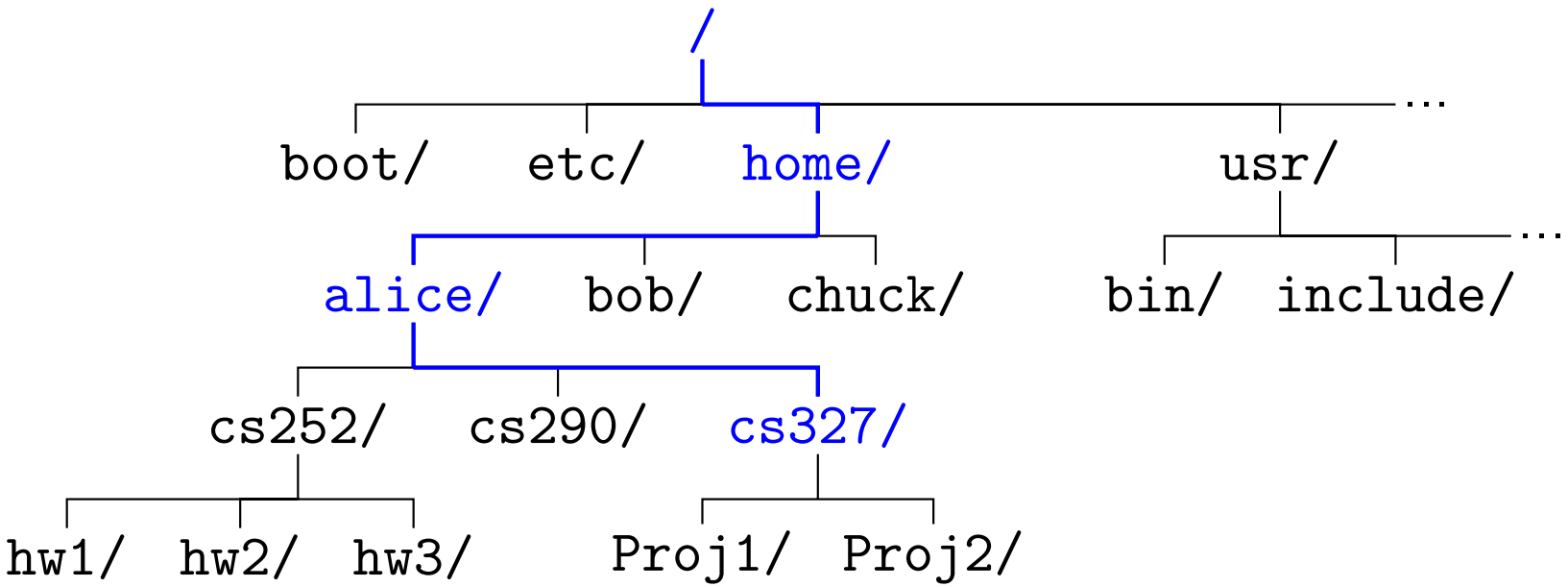

```
/home/alice
prompt$ cd cs252/hw3
prompt$ cd ../hw2
prompt$ cd ../..
prompt$ cd cs327
prompt$
```

Examples: cd and pwd



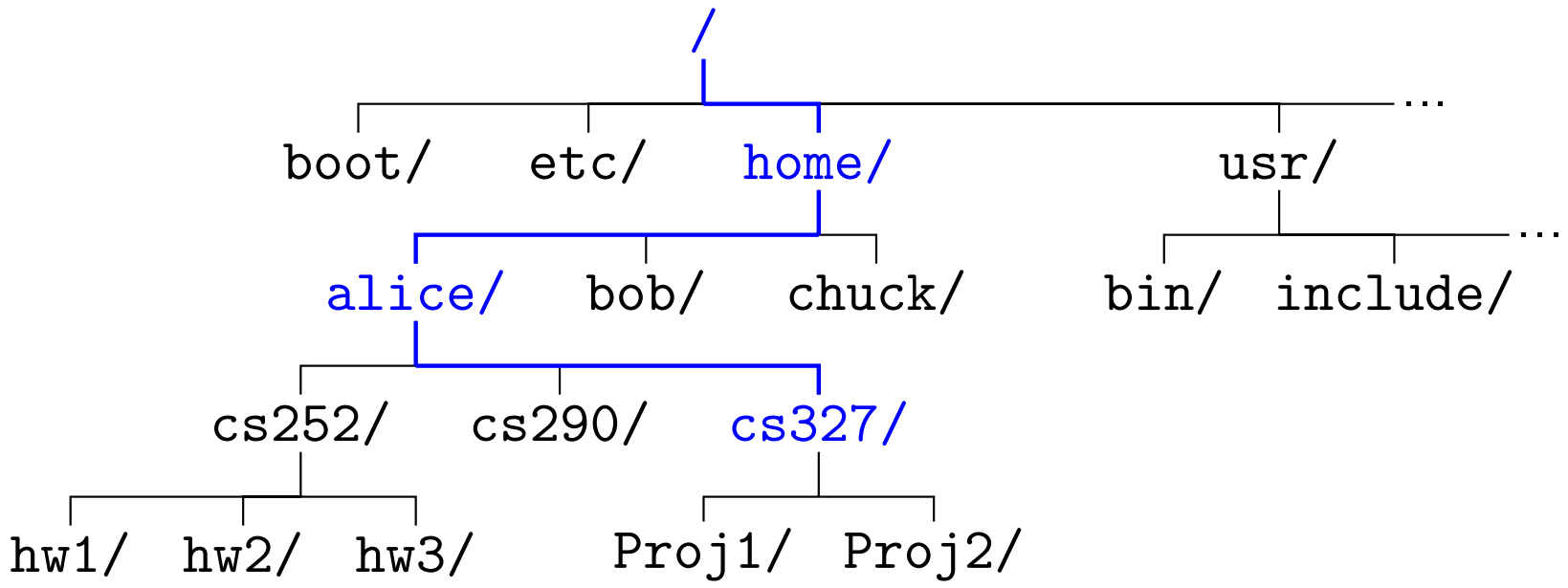

```
/home/alice
prompt$ cd cs252/hw3
prompt$ cd ../hw2
prompt$ cd ../..
prompt$ cd cs327
prompt$ pwd
```

Examples: cd and pwd



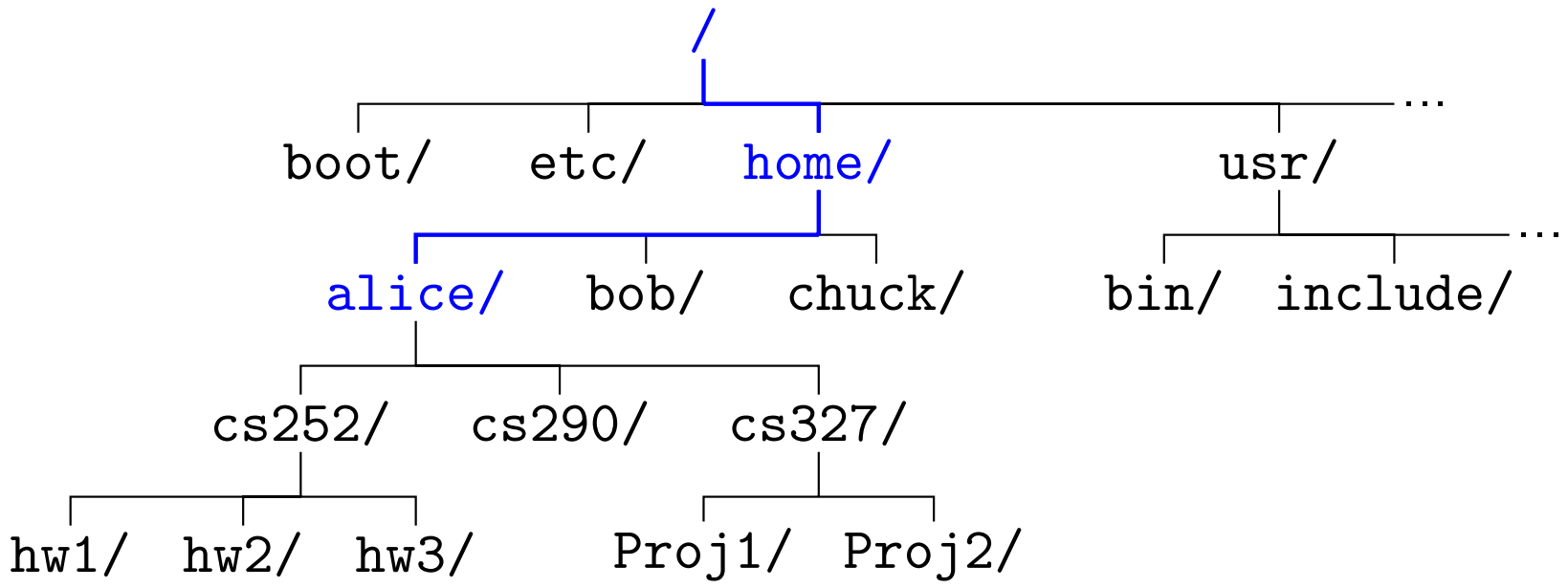

```
prompt$ cd ../hw2
prompt$ cd ../..
prompt$ cd cs327
prompt$ pwd
/home/alice/cs327
prompt$
```

Examples: cd and pwd



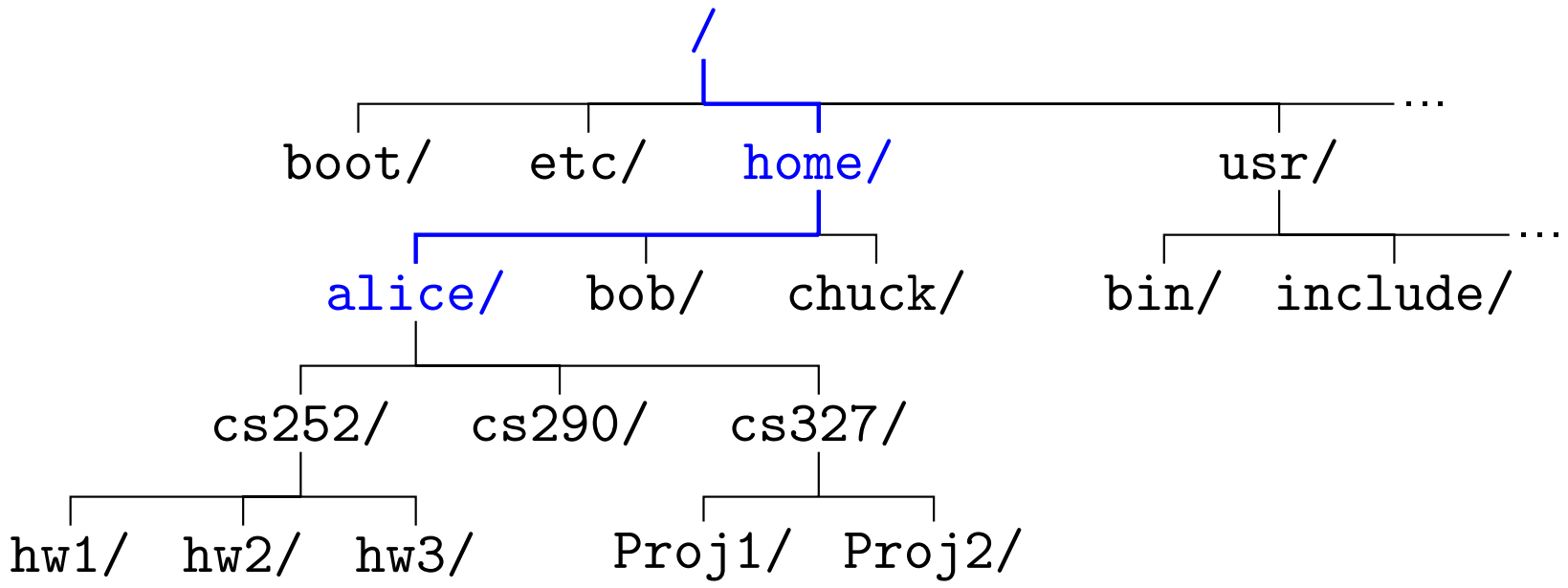

```
prompt$ cd ../hw2
prompt$ cd ../..
prompt$ cd cs327
prompt$ pwd
/home/alice/cs327
prompt$ cd
```

Examples: cd and pwd



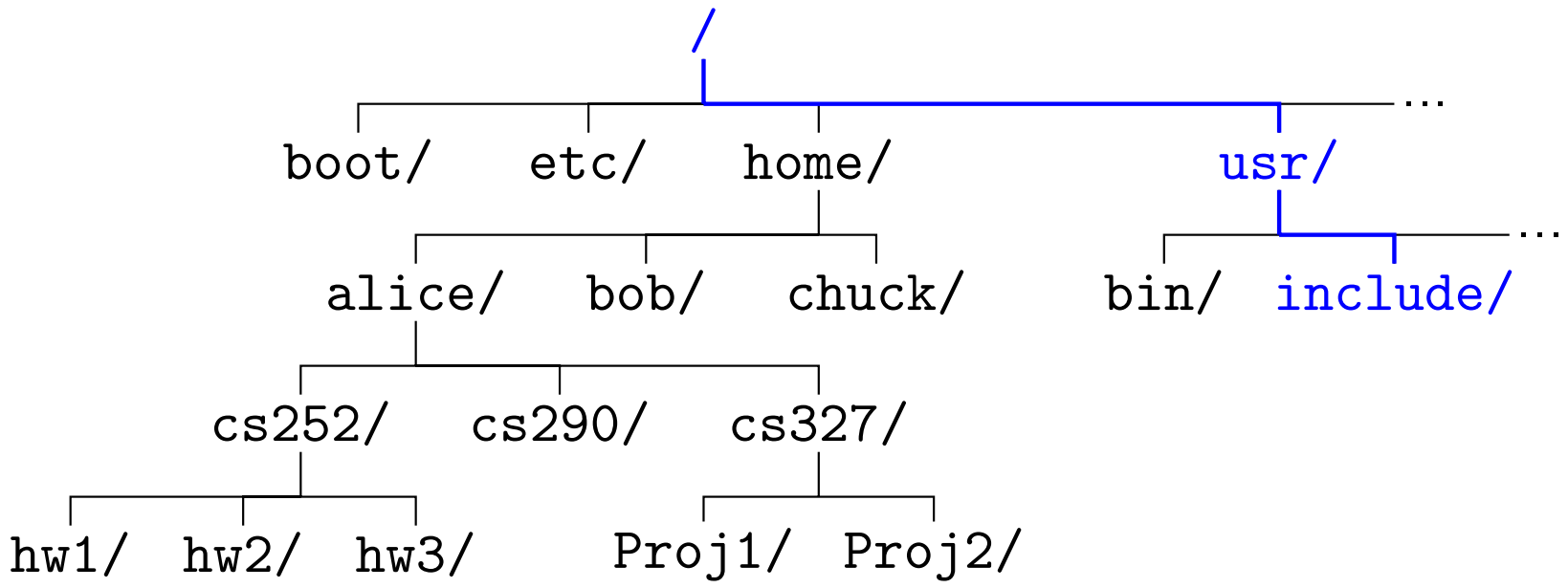

```
prompt$ cd ../..
prompt$ cd cs327
prompt$ pwd
/home/alice/cs327
prompt$ cd
prompt$
```

Examples: cd and pwd



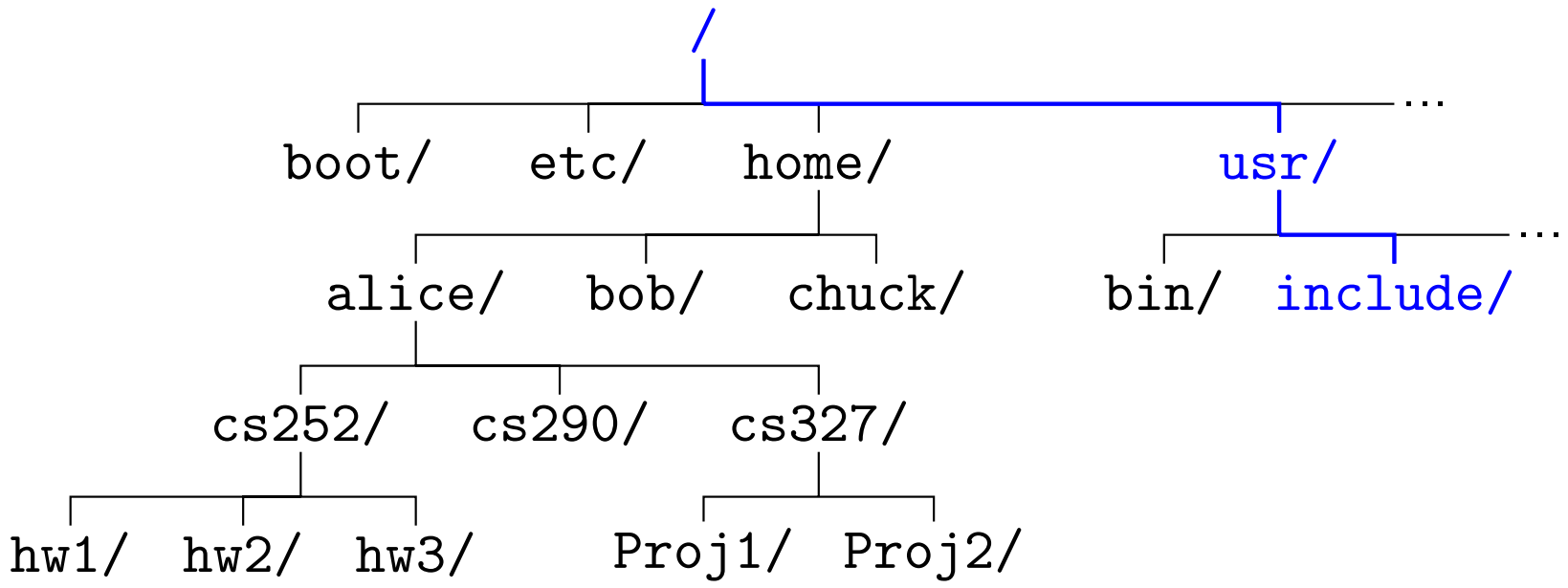

```
prompt$ cd ../..
prompt$ cd cs327
prompt$ pwd
/home/alice/cs327
prompt$ cd
prompt$ cd /usr/include
```

Examples: cd and pwd



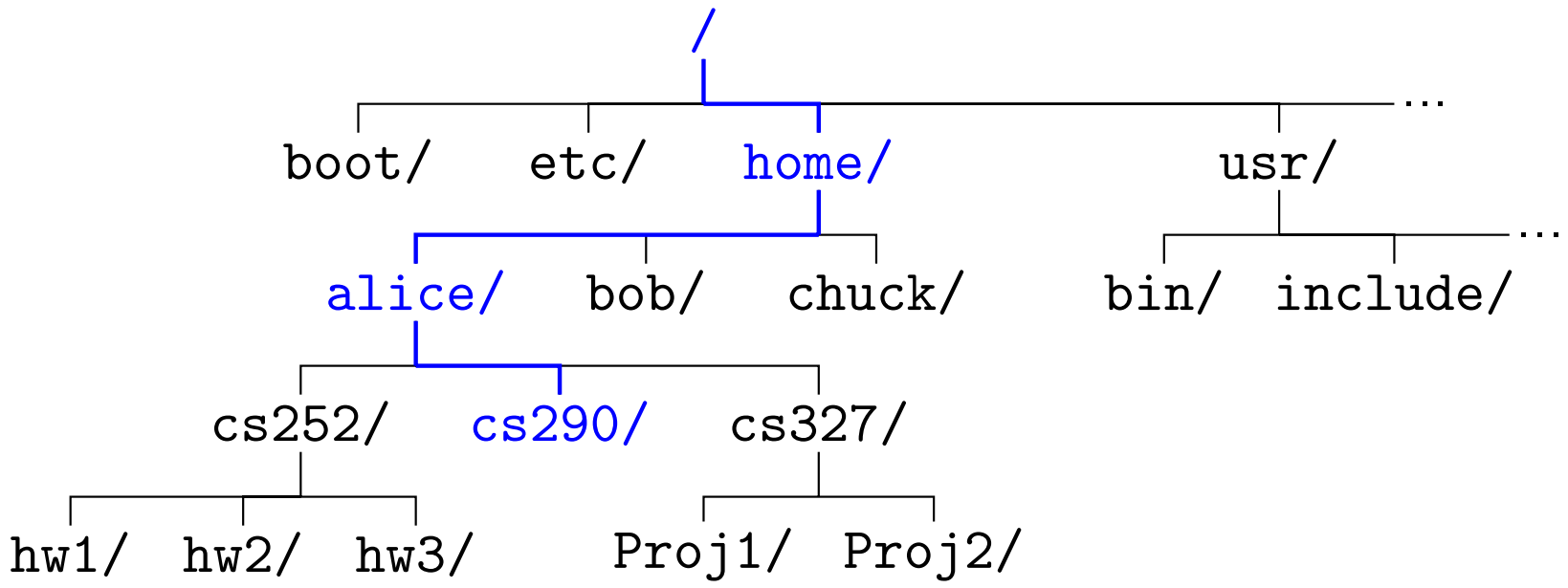

```
prompt$ cd cs327
prompt$ pwd
/home/alice/cs327
prompt$ cd
prompt$ cd /usr/include
prompt$
```

Examples: cd and pwd




```
prompt$ cd cs327
prompt$ pwd
/home/alice/cs327
prompt$ cd
prompt$ cd /usr/include
prompt$ cd ~/cs290
```

Examples: cd and pwd




```
prompt$ pwd
/home/alice/cs327
prompt$ cd
prompt$ cd /usr/include
prompt$ cd ~/cs290
prompt$
```

Creating and deleting directories

`mkdir`: make (empty) directories

- Arguments: (absolute or relative) pathnames

-p : will create intermediate directories if necessary

Creating and deleting directories

`mkdir`: make (empty) directories

- Arguments: (absolute or relative) pathnames

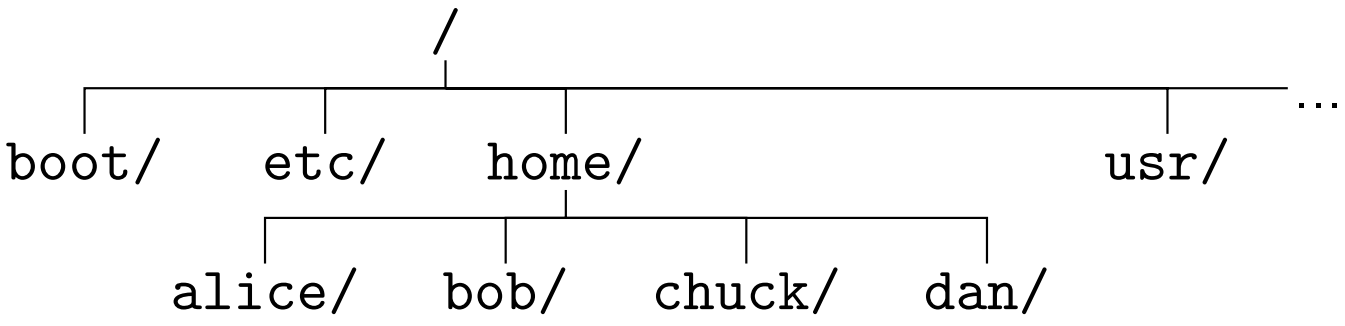
`-p` : will create intermediate directories if necessary

`rmdir`: remove empty directories

- Arguments: (absolute or relative) pathnames
- Will fail if the directory is not empty

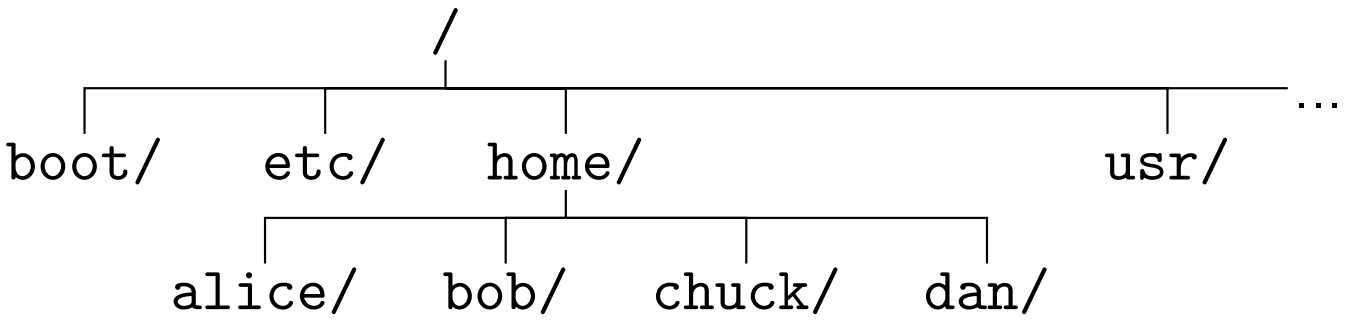
`-p` : will remove parent directories also

Examples: mkdir and rmdir



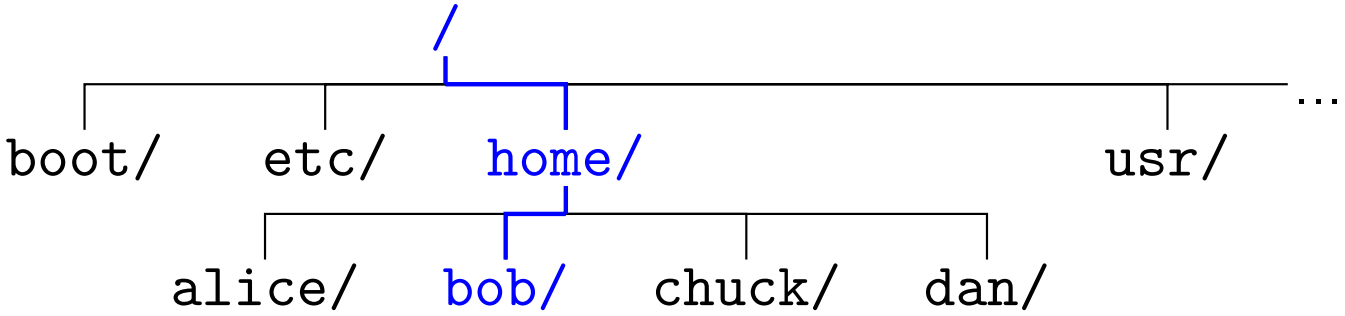

```
prompt$
```

Examples: mkdir and rmdir



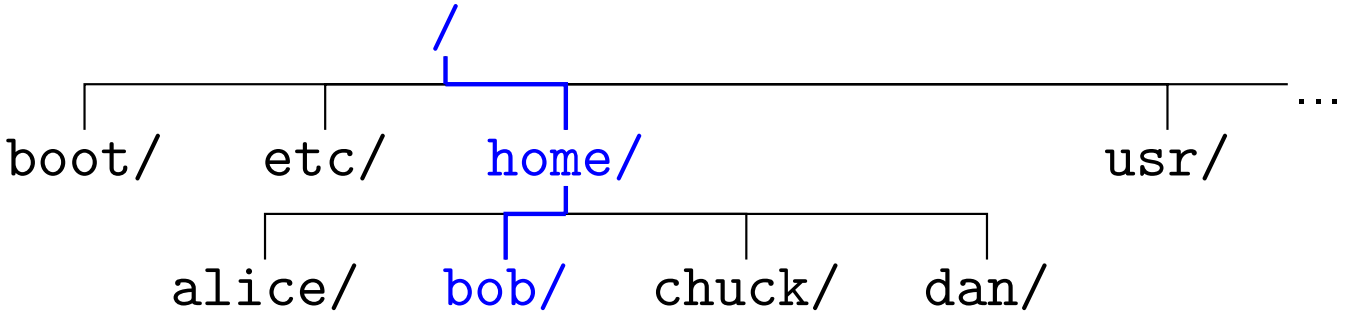

```
prompt$ pwd
```

Examples: mkdir and rmdir



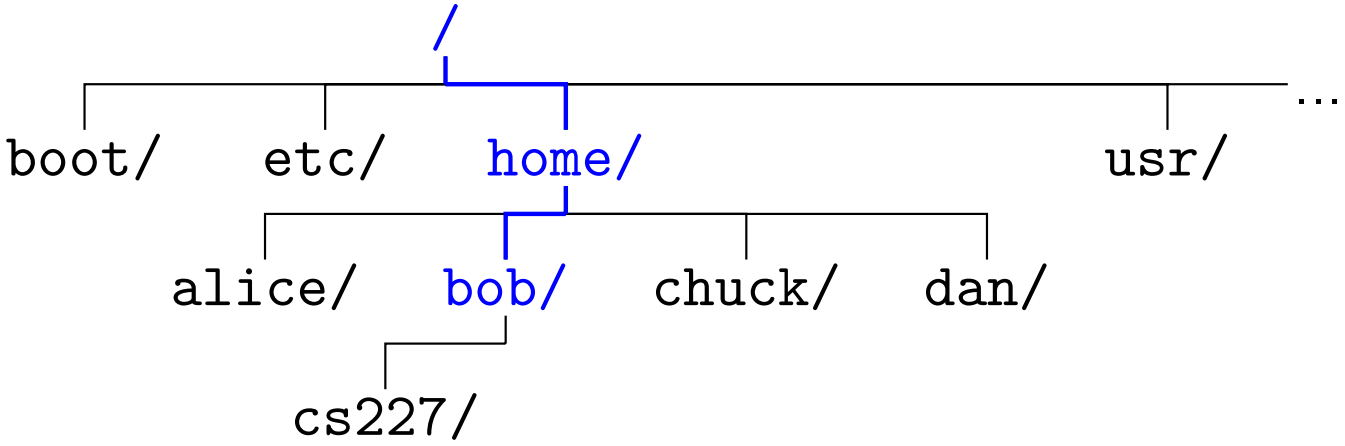

```
prompt$ pwd
/home/bob
prompt$
```

Examples: mkdir and rmdir



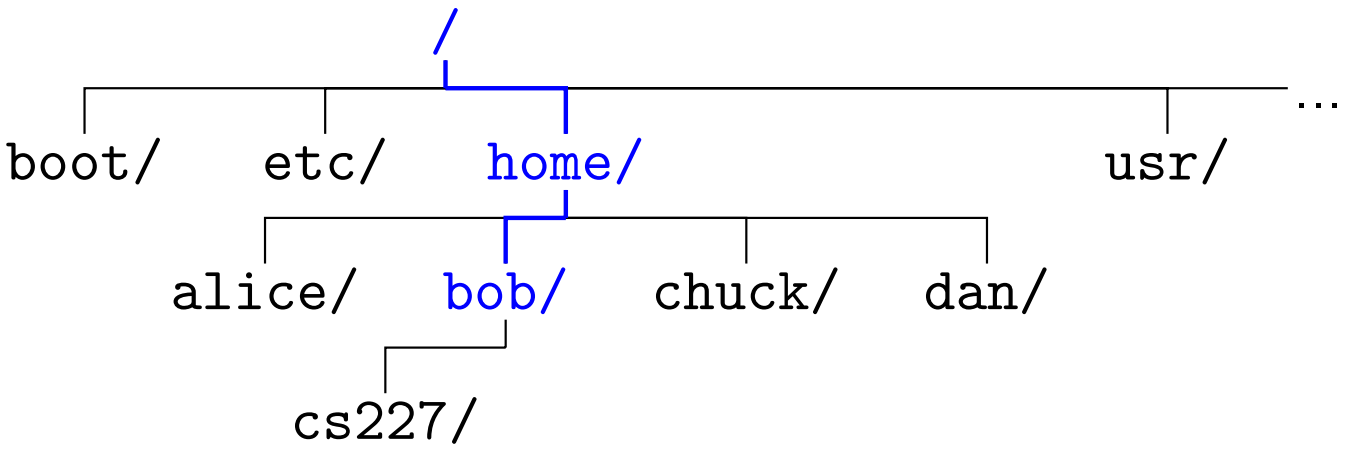

```
prompt$ pwd
/home/bob
prompt$ mkdir cs227
```

Examples: mkdir and rmdir



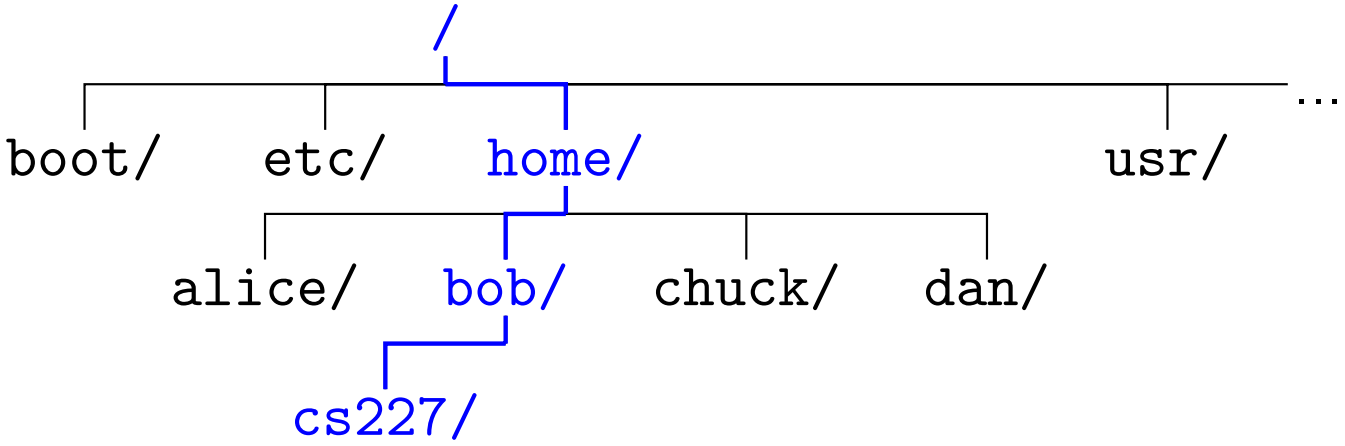

```
prompt$ pwd
/home/bob
prompt$ mkdir cs227
prompt$
```

Examples: mkdir and rmdir



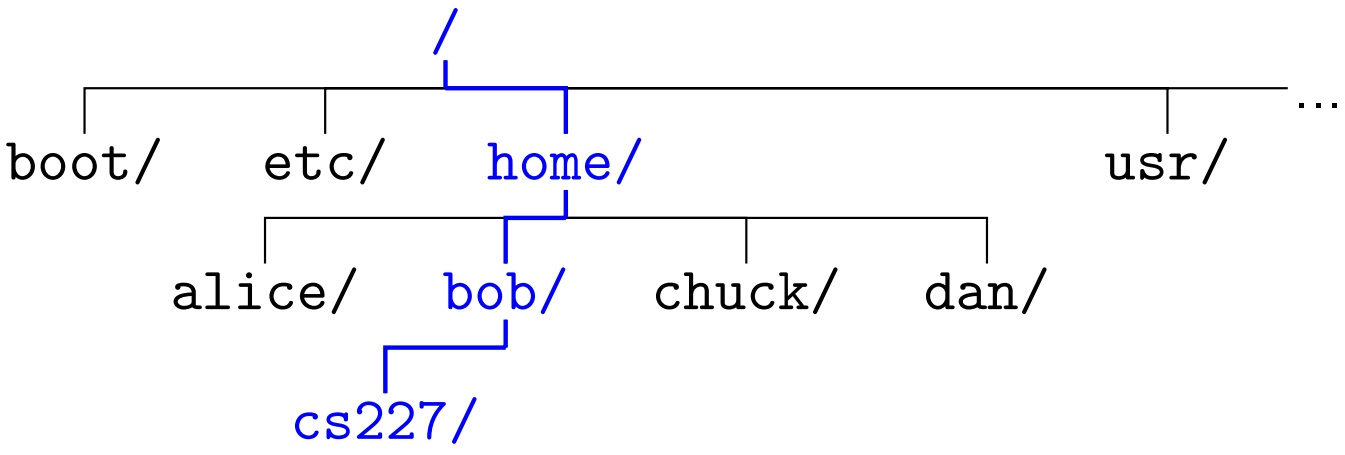

```
prompt$ pwd
/home/bob
prompt$ mkdir cs227
prompt$ cd cs227
```

Examples: mkdir and rmdir



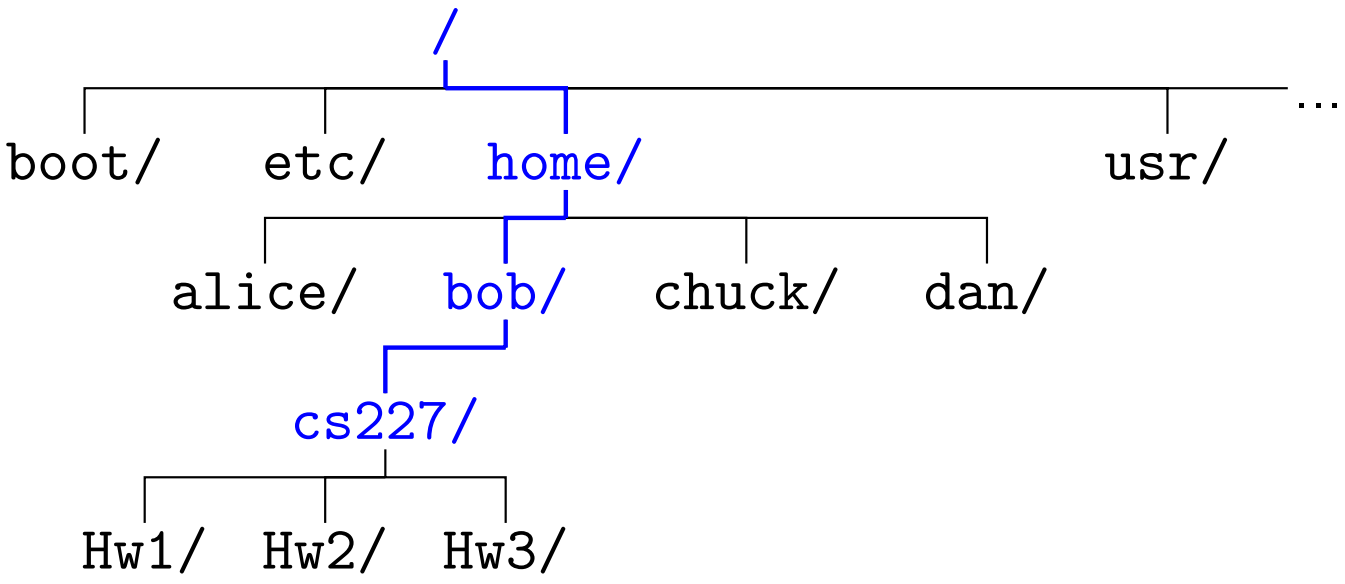

```
prompt$ pwd
/home/bob
prompt$ mkdir cs227
prompt$ cd cs227
prompt$
```

Examples: mkdir and rmdir



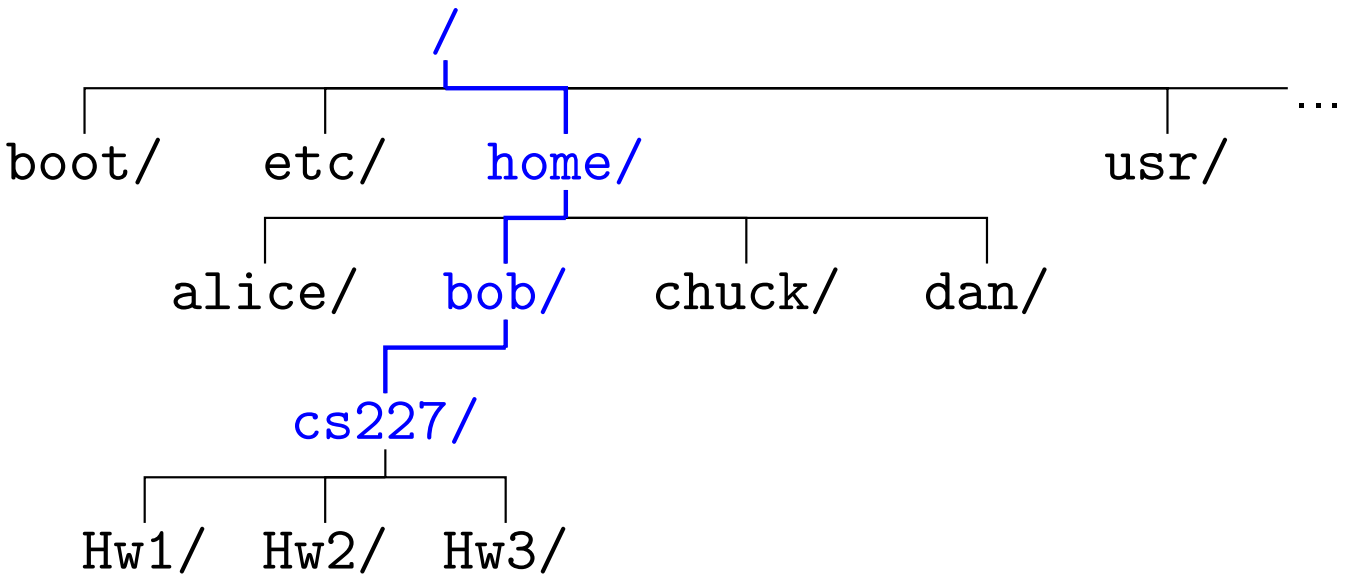

```
prompt$ pwd
/home/bob
prompt$ mkdir cs227
prompt$ cd cs227
prompt$ mkdir Hw1 Hw2 Hw3
```

Examples: mkdir and rmdir



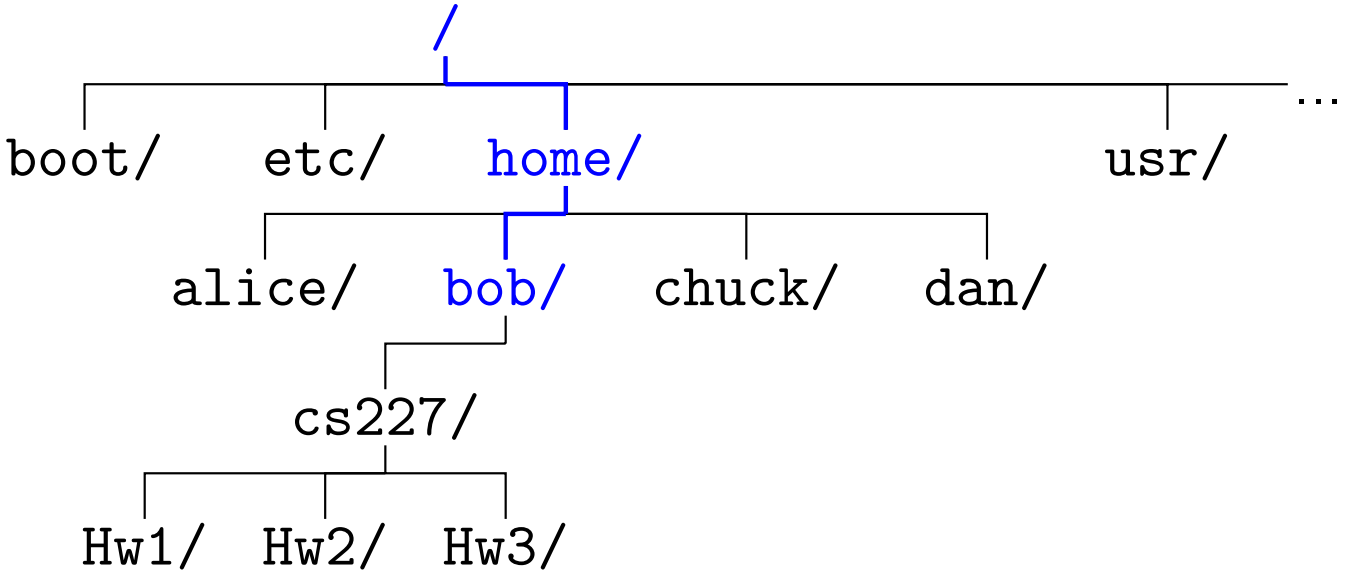

```
/home/bob
prompt$ mkdir cs227
prompt$ cd cs227
prompt$ mkdir Hw1 Hw2 Hw3
prompt$
```

Examples: mkdir and rmdir



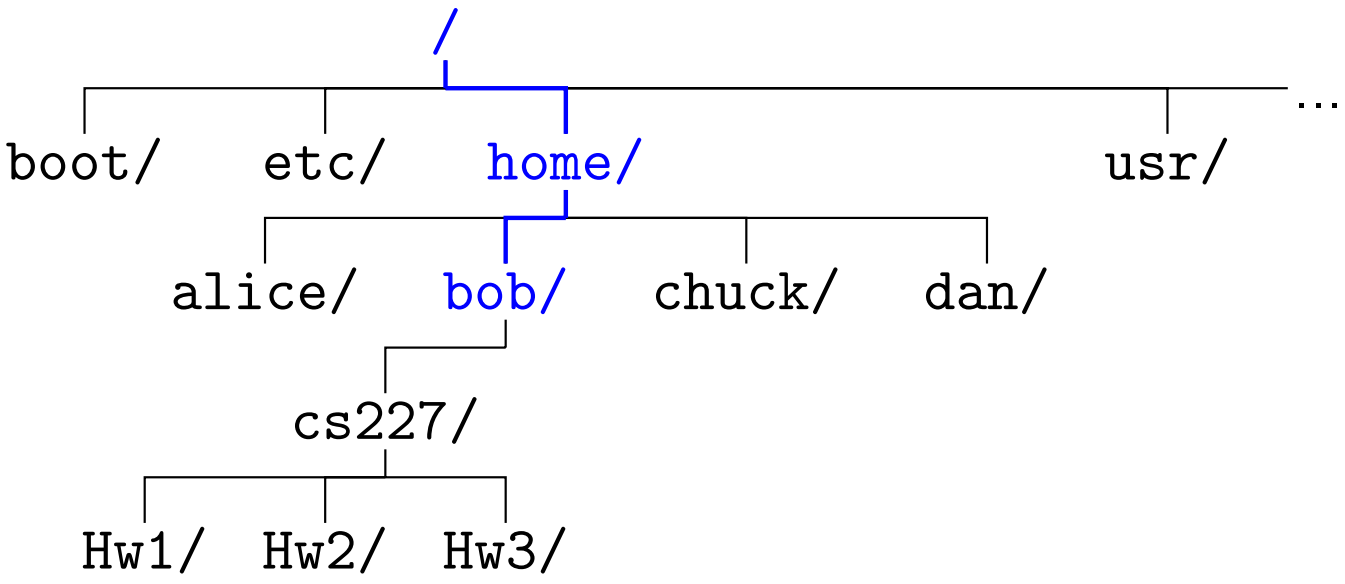

```
/home/bob
prompt$ mkdir cs227
prompt$ cd cs227
prompt$ mkdir Hw1 Hw2 Hw3
prompt$ cd ..
```

Examples: mkdir and rmdir



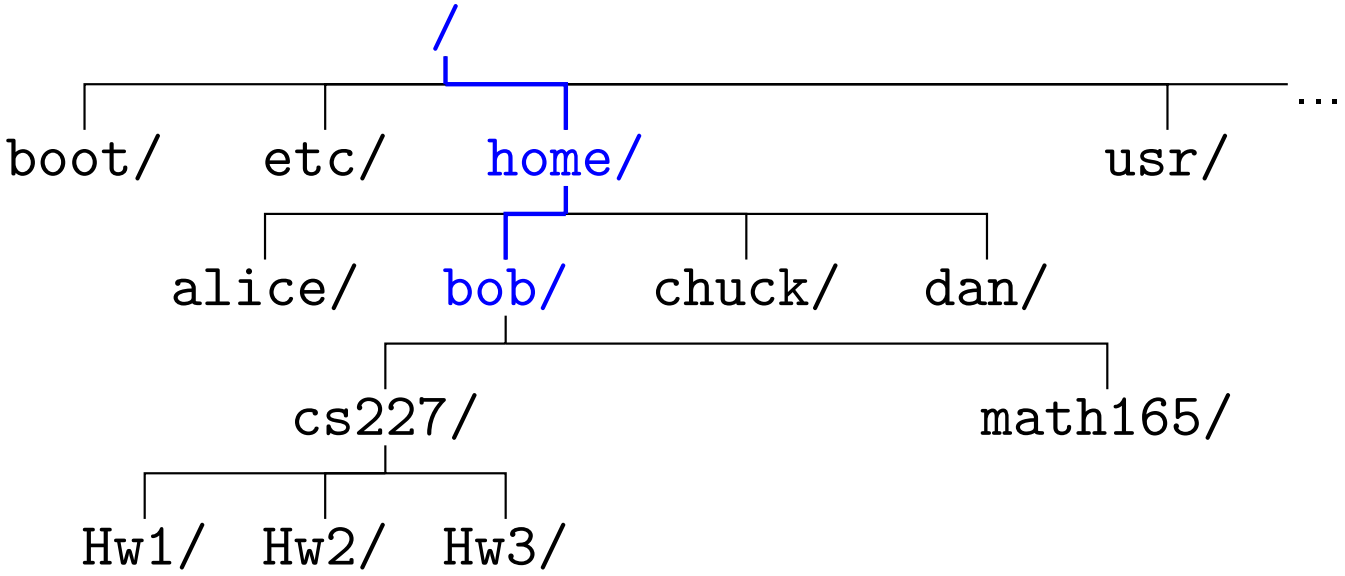

```
prompt$ mkdir cs227
prompt$ cd cs227
prompt$ mkdir Hw1 Hw2 Hw3
prompt$ cd ..
prompt$
```

Examples: mkdir and rmdir



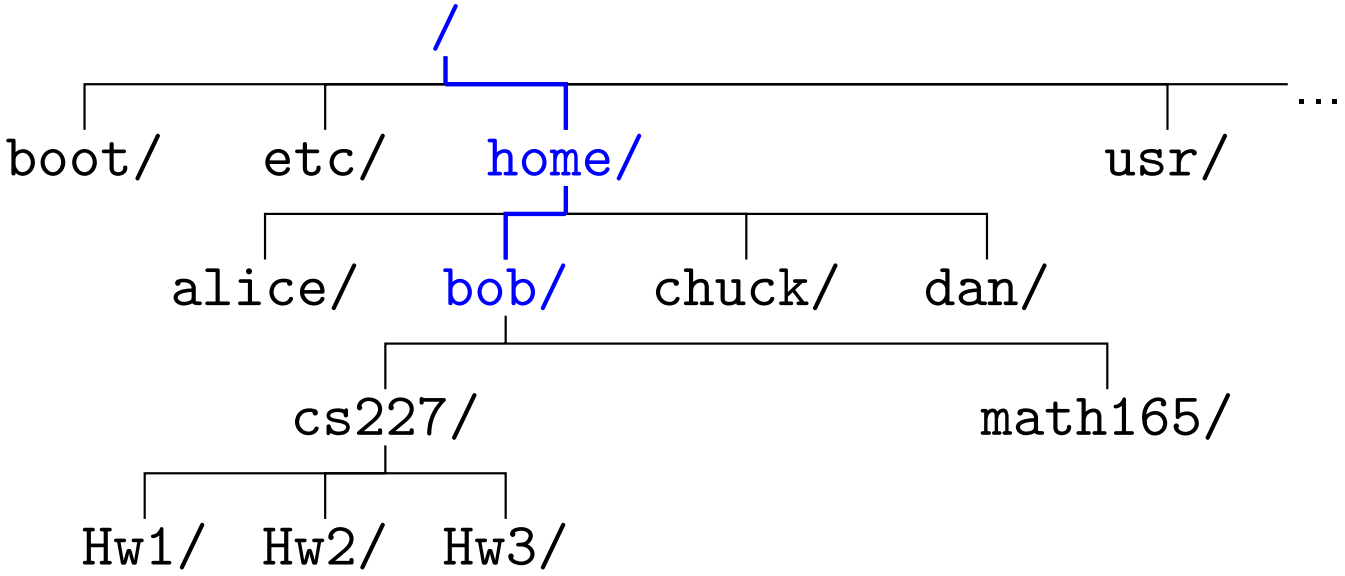

```
prompt$ mkdir cs227
prompt$ cd cs227
prompt$ mkdir Hw1 Hw2 Hw3
prompt$ cd ..
prompt$ mkdir math165
```

Examples: mkdir and rmdir



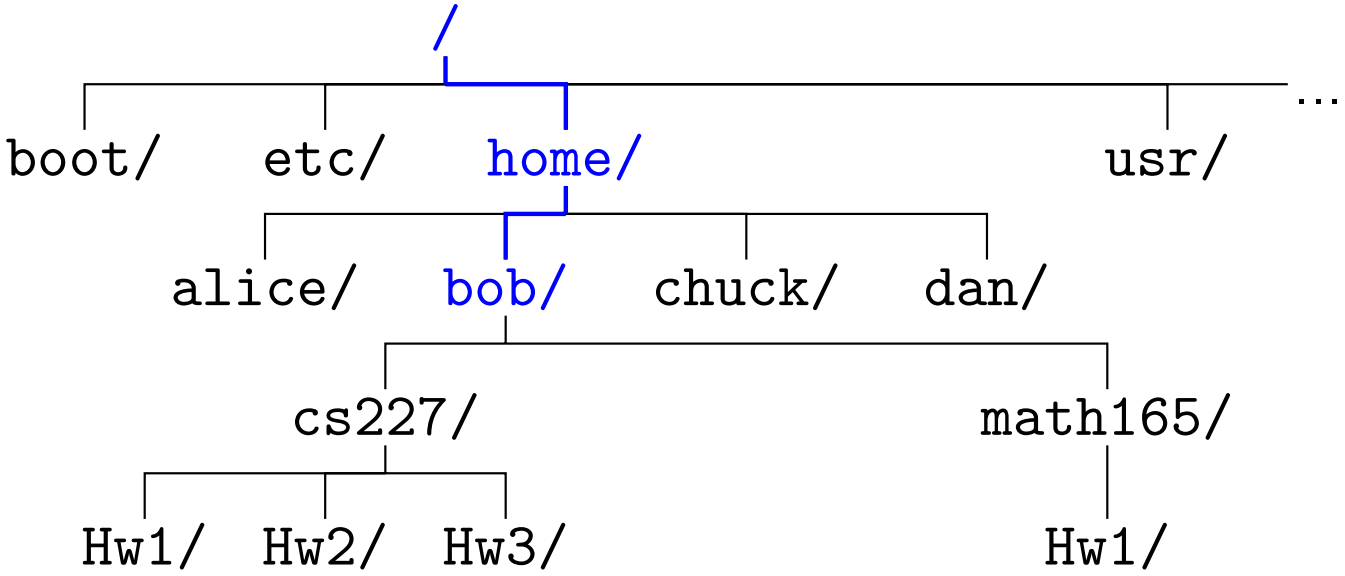

```
prompt$ cd cs227
prompt$ mkdir Hw1 Hw2 Hw3
prompt$ cd ..
prompt$ mkdir math165
prompt$
```

Examples: mkdir and rmdir



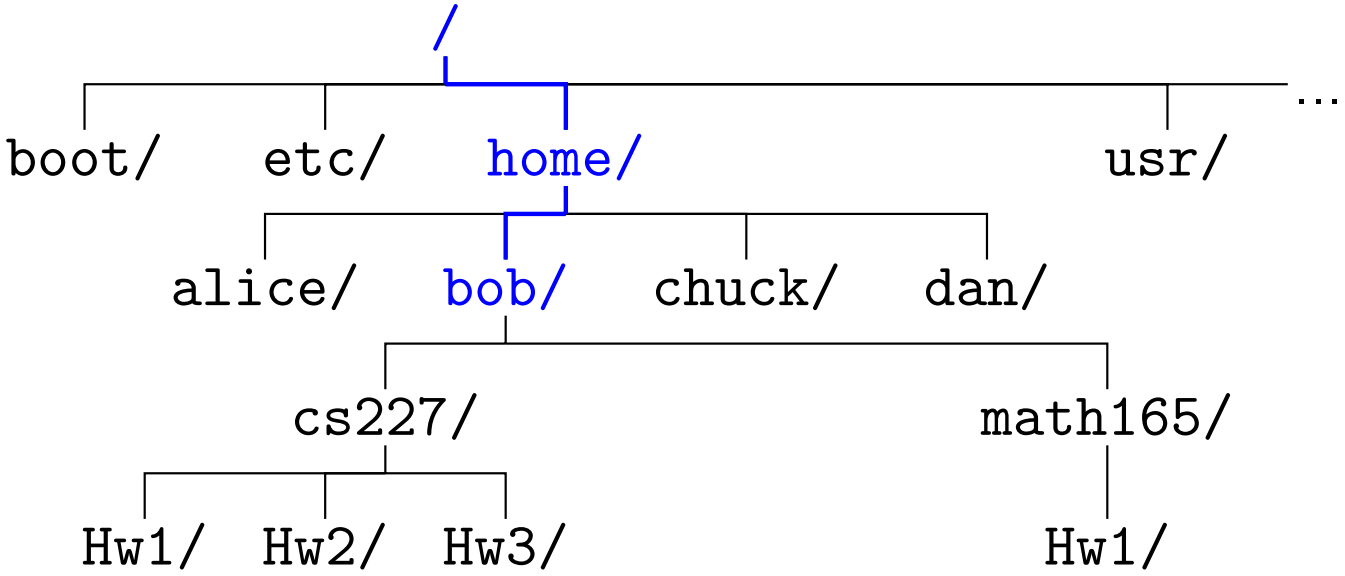

```
prompt$ cd cs227
prompt$ mkdir Hw1 Hw2 Hw3
prompt$ cd ..
prompt$ mkdir math165
prompt$ mkdir math165/Hw1
```

Examples: mkdir and rmdir



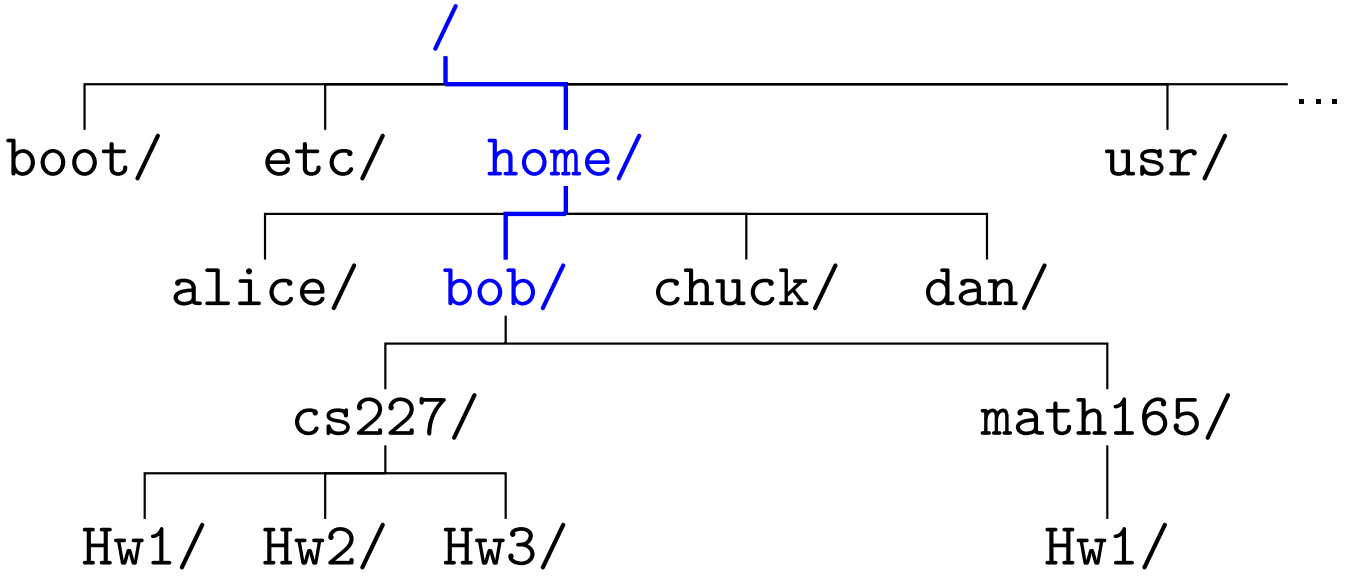

```
prompt$ mkdir Hw1 Hw2 Hw3
prompt$ cd ..
prompt$ mkdir math165
prompt$ mkdir math165/Hw1
prompt$
```

Examples: mkdir and rmdir



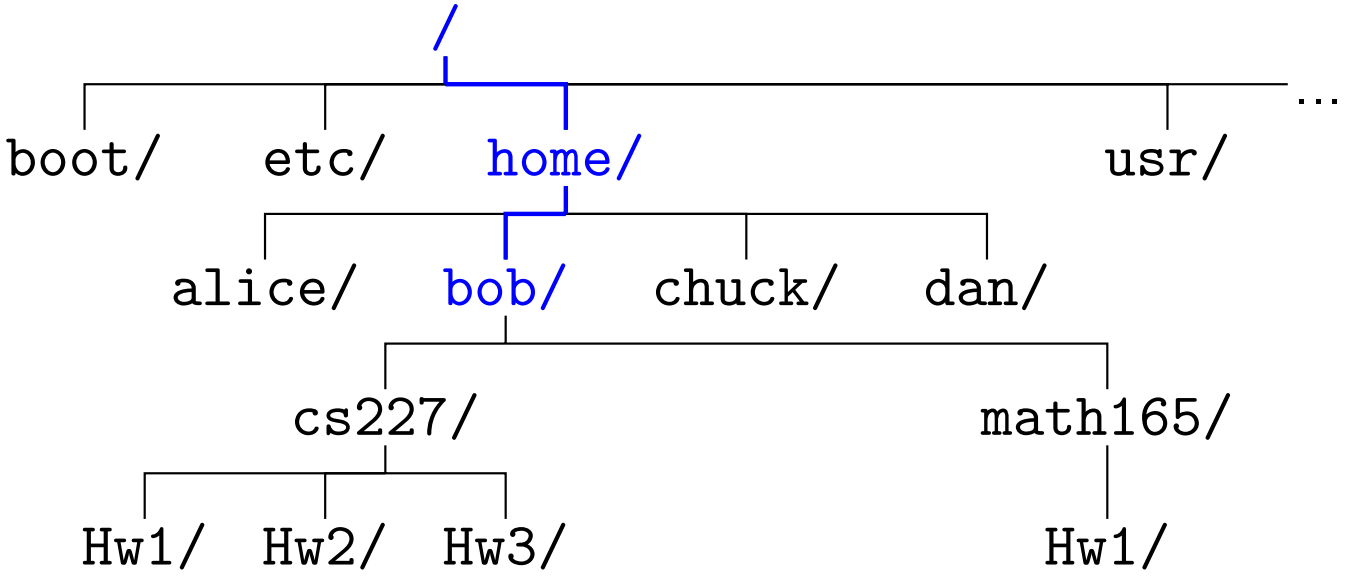

```
prompt$ mkdir Hw1 Hw2 Hw3
prompt$ cd ..
prompt$ mkdir math165
prompt$ mkdir math165/Hw1
prompt$ mkdir ds201/Proj1/src
```

Examples: mkdir and rmdir



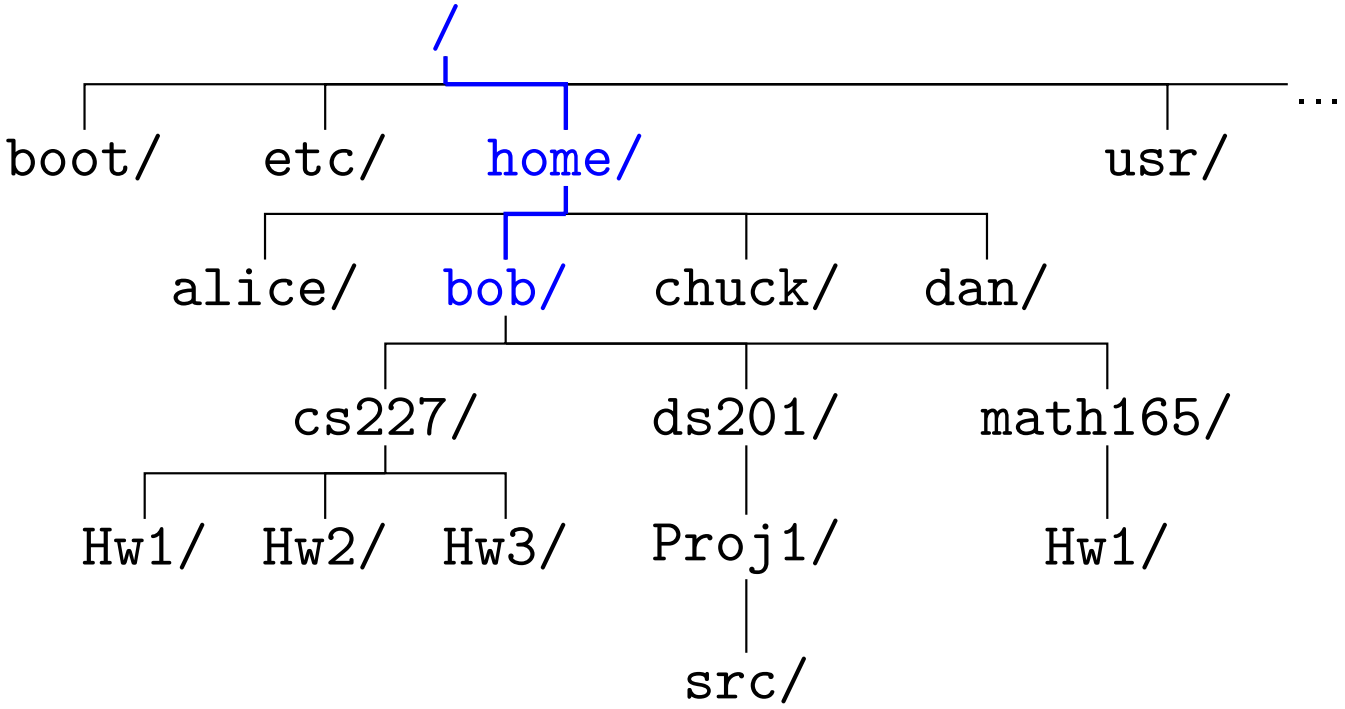

```
prompt$ mkdir math165
prompt$ mkdir math165/Hw1
prompt$ mkdir ds201/Proj1/src
ds201: No such file or directory
prompt$
```

Examples: mkdir and rmdir



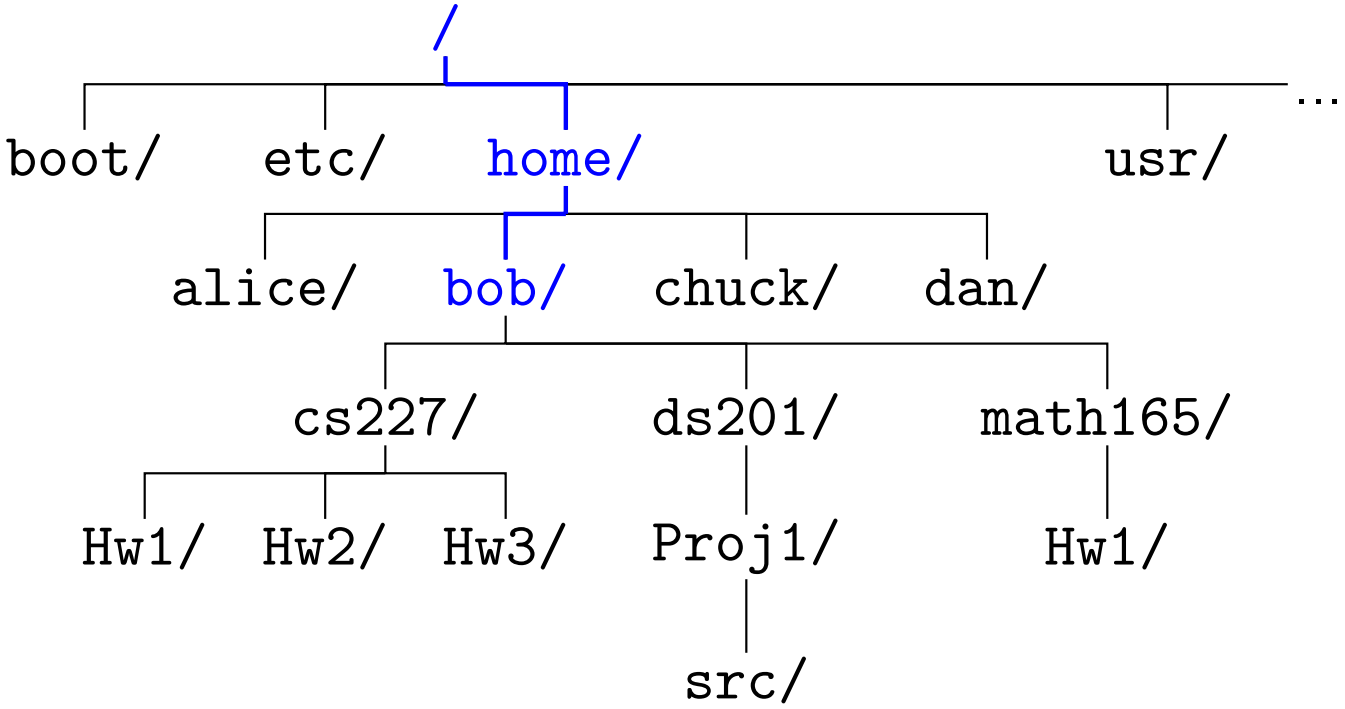

```
prompt$ mkdir math165
prompt$ mkdir math165/Hw1
prompt$ mkdir ds201/Proj1/src
ds201: No such file or directory
prompt$ mkdir -p ds201/Proj1/src
```

Examples: mkdir and rmdir



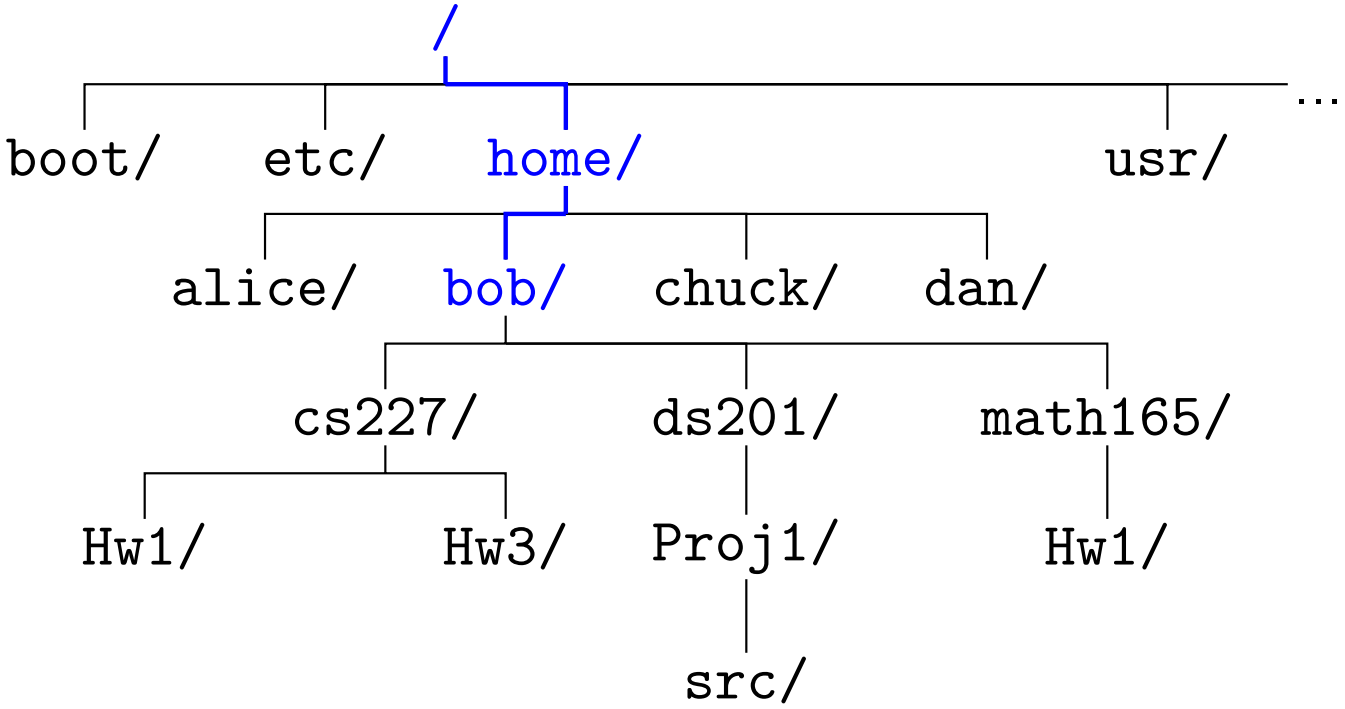

```
prompt$ mkdir math165/Hw1
prompt$ mkdir ds201/Proj1/src
ds201: No such file or directory
prompt$ mkdir -p ds201/Proj1/src
prompt$
```

Examples: mkdir and rmdir



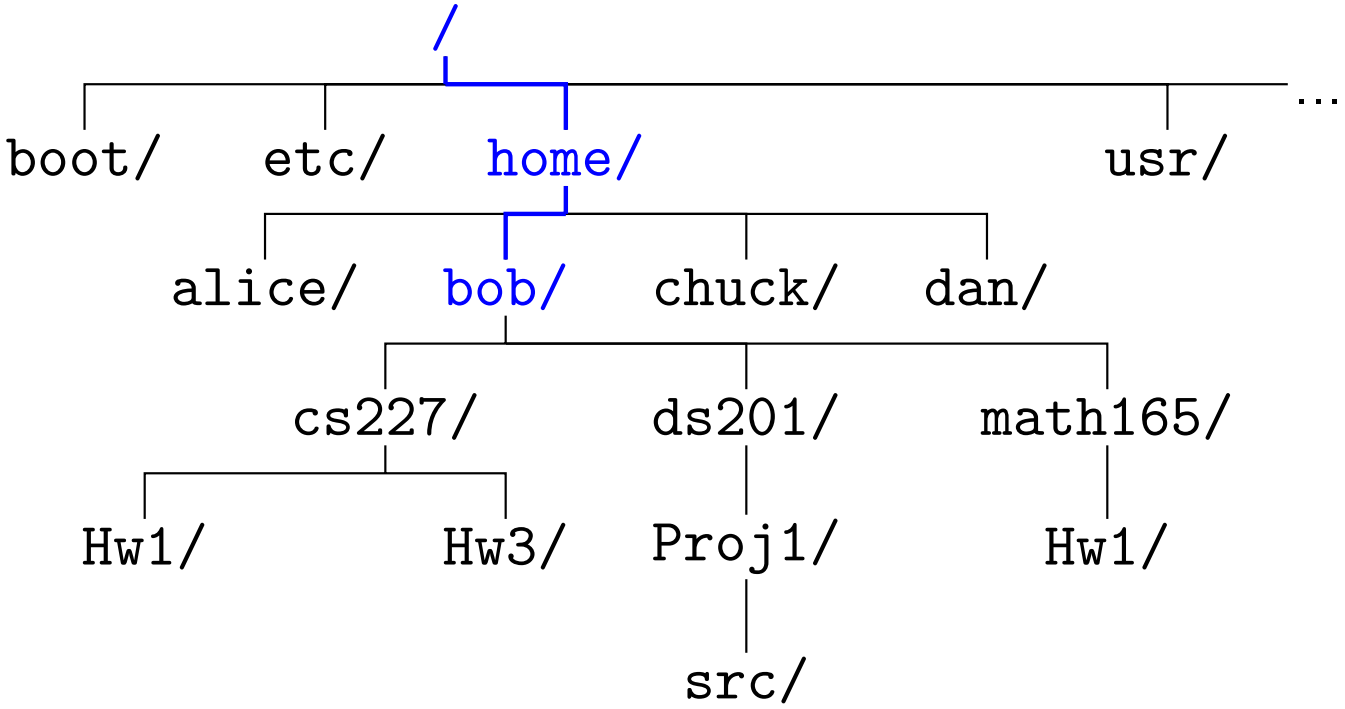

```
prompt$ mkdir math165/Hw1
prompt$ mkdir ds201/Proj1/src
ds201: No such file or directory
prompt$ mkdir -p ds201/Proj1/src
prompt$ rmdir cs227/Hw2
```

Examples: mkdir and rmdir



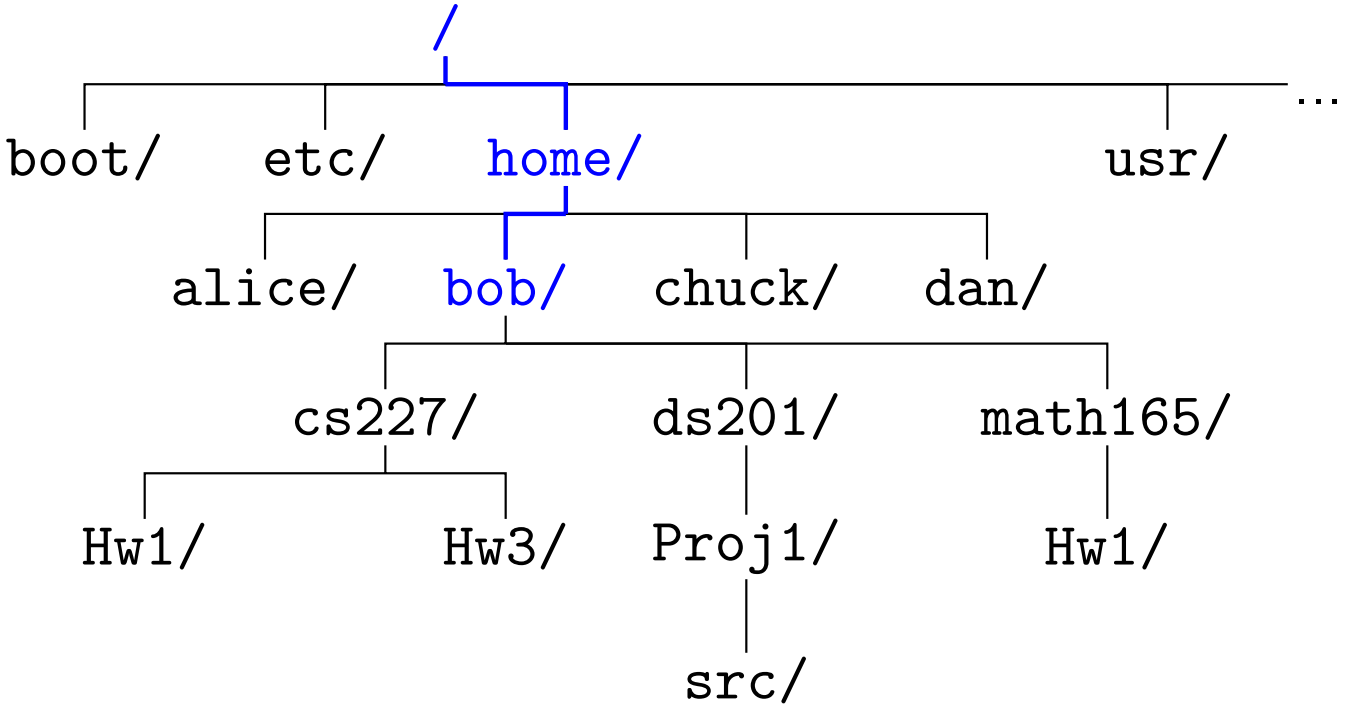

```
prompt$ mkdir ds201/Proj1/src
ds201: No such file or directory
prompt$ mkdir -p ds201/Proj1/src
prompt$ rmdir cs227/Hw2
prompt$
```

Examples: mkdir and rmdir



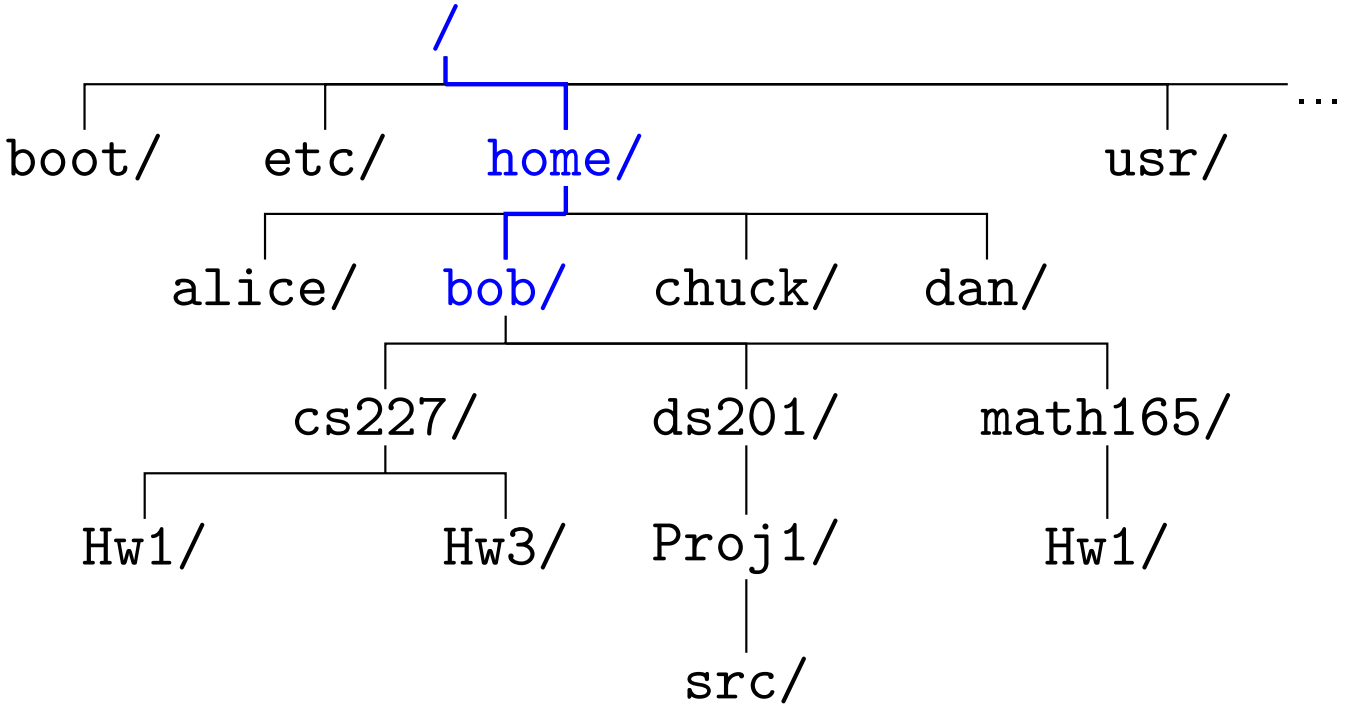

```
prompt$ mkdir ds201/Proj1/src
ds201: No such file or directory
prompt$ mkdir -p ds201/Proj1/src
prompt$ rmdir cs227/Hw2
prompt$ rmdir cs227
```

Examples: mkdir and rmdir



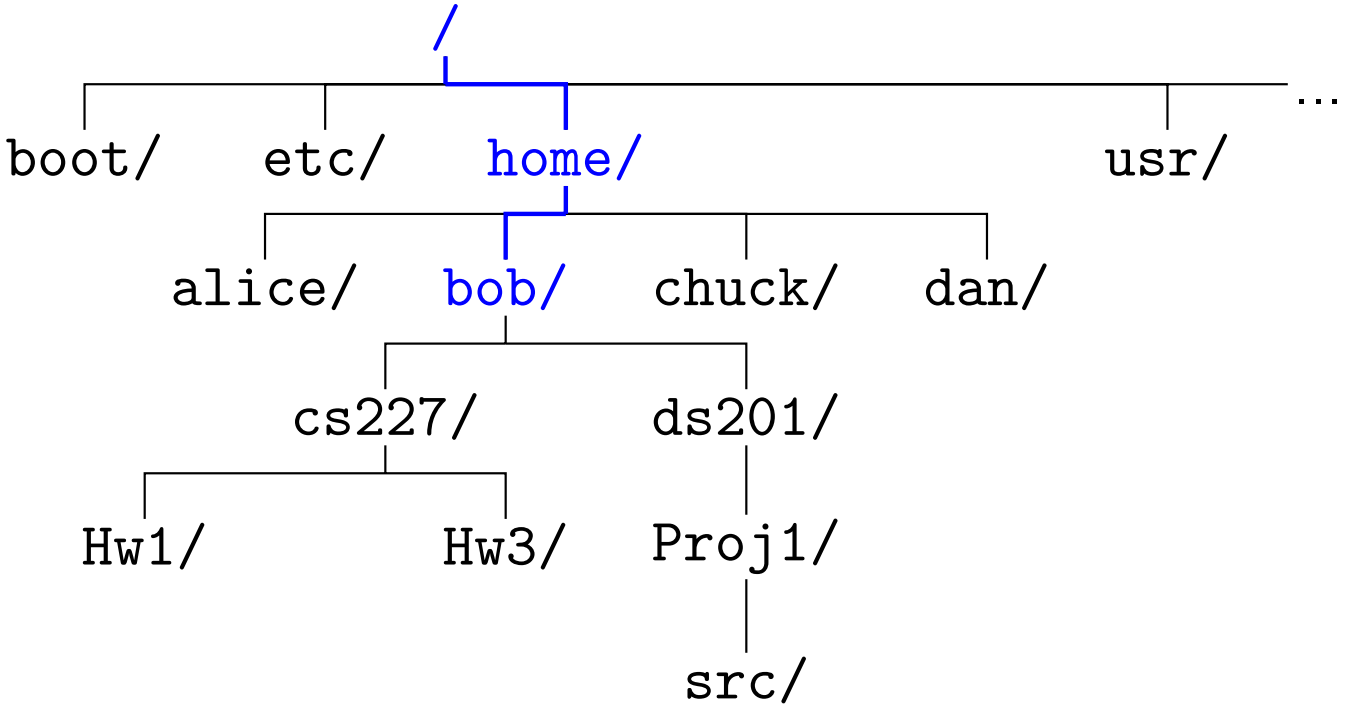

```
prompt$ mkdir -p ds201/Proj1/src
prompt$ rmdir cs227/Hw2
prompt$ rmdir cs227
Directory not empty
prompt$
```

Examples: mkdir and rmdir




```
prompt$ mkdir -p ds201/Proj1/src
prompt$ rmdir cs227/Hw2
prompt$ rmdir cs227
Directory not empty
prompt$ rmdir -p math165/Hw1
```

Examples: mkdir and rmdir




```
prompt$ rmdir cs227/Hw2
prompt$ rmdir cs227
Directory not empty
prompt$ rmdir -p math165/Hw1
prompt$
```

Listing contents

`ls`: list files (and folders)

- Without arguments: show files in working directory
- With arguments
  - File: show listing for the file
  - Directory: show files in the directory

```
prompt$
```

Listing contents

`ls`: list files (and folders)

- Without arguments: show files in working directory
- With arguments
  - File: show listing for the file
  - Directory: show files in the directory

```
prompt$ pwd
```

Listing contents

`ls`: list files (and folders)

- Without arguments: show files in working directory
- With arguments
  - File: show listing for the file
  - Directory: show files in the directory

```
prompt$ pwd
/home/alice
prompt$
```

Listing contents

`ls`: list files (and folders)

- Without arguments: show files in working directory
- With arguments
  - File: show listing for the file
  - Directory: show files in the directory

```
prompt$ pwd
/home/alice
prompt$ ls
```

Listing contents

`ls`: list files (and folders)

- Without arguments: show files in working directory
- With arguments
  - File: show listing for the file
  - Directory: show files in the directory

```
prompt$ pwd
/home/alice
prompt$ ls
a.out     core     cs252    hello.c  hello.o  se101
bar.cc    cs229    foo.cc   hello.h  math168
prompt$
```

Listing contents

`ls`: list files (and folders)

- Without arguments: show files in working directory
- With arguments
  - File: show listing for the file
  - Directory: show files in the directory

```
prompt$ pwd
/home/alice
prompt$ ls
a.out     core     cs252    hello.c  hello.o  se101
bar.cc    cs229    foo.cc   hello.h  math168
prompt$ ls cs252
```

Listing contents

`ls`: list files (and folders)

- Without arguments: show files in working directory
- With arguments
  - File: show listing for the file
  - Directory: show files in the directory

```
prompt$ pwd
/home/alice
prompt$ ls
a.out     core     cs252    hello.c  hello.o  se101
bar.cc    cs229    foo.cc   hello.h  math168
prompt$ ls cs252
hw1    hw2    hw3    hw4
prompt$
```

Useful `ls` switches

- `-a` : show all files
  - Filenames starting with `.` are normally hidden
- `-l` : long listing
  - Default is "short listing": ordered names in columns
- `-F` : extra character displayed for the file type
  - `*` : executables
  - `/` : subdirectory
  - `@` : symlink (will discuss later)
  - `=` : socket (discussed in ComS 352)
- `--color` : Like "`-F`" but uses color (Linux only)
- `-G` : Like "`-F`" but uses color (Mac OS only)

Example: `ls` with switches

```
prompt$
```

Example: `ls` with switches

```
prompt$ pwd
```

Example: `ls` with switches

```
prompt$ pwd
/home/alice
prompt$
```

Example: `ls` with switches

```
prompt$ pwd
/home/alice
prompt$ ls
```

Example: `ls` with switches

```
prompt$ pwd
/home/alice
prompt$ ls
a.out      core      cs252     hello.c   hello.o   se101
bar.cc     cs229     foo.cc    hello.h   math168
prompt$
```

Example: `ls` with switches

```
prompt$ pwd
/home/alice
prompt$ ls
a.out      core      cs252     hello.c   hello.o   se101
bar.cc     cs229     foo.cc    hello.h   math168
prompt$ ls -a
```

Example: `ls` with switches

```
prompt$ pwd
/home/alice
prompt$ ls
a.out      core      cs252     hello.c   hello.o   se101
bar.cc     cs229     foo.cc    hello.h   math168
prompt$ ls -a
.          .bashrc   foo.cc    .history  .viminfo
..         core      hello.c   math168
a.out      cs229     hello.h   se101
bar.cc     cs252     hello.o   .ssh
prompt$
```

Example: `ls` with switches

```
prompt$ pwd
/home/alice
prompt$ ls
a.out      core      cs252     hello.c   hello.o   se101
bar.cc     cs229     foo.cc    hello.h   math168
prompt$ ls -a
.          .bashrc   foo.cc    .history  .viminfo
..         core      hello.c   math168
a.out      cs229     hello.h   se101
bar.cc     cs252     hello.o   .ssh
prompt$ ls -aF
```

Example: `ls` with switches

```
prompt$ pwd
/home/alice
prompt$ ls
a.out      core      cs252     hello.c   hello.o   se101
bar.cc     cs229     foo.cc    hello.h   math168
prompt$ ls -a
.          .bashrc   foo.cc    .history  .viminfo
..         core      hello.c   math168
a.out      cs229     hello.h   se101
bar.cc     cs252     hello.o   .ssh
prompt$ ls -aF
./         .bashrc   foo.cc    .history  .viminfo
../        core      hello.c   math168/
a.out*     cs229/    hello.h   se101/
bar.cc     cs252/    hello.o   .ssh/
prompt$
```

Example: `ls` with switches

```
prompt$ pwd
/home/alice
prompt$ ls
a.out      core      cs252     hello.c   hello.o   se101
bar.cc     cs229     foo.cc    hello.h   math168
prompt$ ls -a
.          .bashrc   foo.cc    .history  .viminfo
..         core      hello.c   math168
a.out      cs229     hello.h   se101
bar.cc     cs252     hello.o   .ssh
prompt$ ls -aF
./         .bashrc   foo.cc    .history  .viminfo
../        core      hello.c   math168/
a.out*     cs229/    hello.h   se101/
bar.cc     cs252/    hello.o   .ssh/
prompt$ ls -aF --color
```

Example: `ls` with switches

```
prompt$ ls -a
.          .bashrc   foo.cc    .history  .viminfo
..         core      hello.c   math168
a.out      cs229     hello.h   se101
bar.cc     cs252     hello.o   .ssh
prompt$ ls -aF
./         .bashrc   foo.cc    .history  .viminfo
../        core      hello.c   math168/
a.out*     cs229/    hello.h   se101/
bar.cc     cs252/    hello.o   .ssh/
prompt$ ls -aF --color
./         .bashrc   foo.cc    .history  .viminfo
../        core      hello.c   math168/
a.out*     cs229/    hello.h   se101/
bar.cc     cs252/    hello.o   .ssh/
prompt$
```

Long vs. short listing

The short listing:

- Divides output into columns
- Just displays the name
- Goal: use least space possible

The long listing:

- Displays one item per line
- Displays lots of information for each item, in columns

Long listing columns

1. 10 "mystery characters"
   - First character specifies the file type

     `-` : Ordinary file
     `b/c` : block/character device (to be discussed)
     `d` : directory
     `l` : symlink (to be discussed)
     `s` : socket
   - In UNIX, everything is a file
   - Other 9 characters are for permissions (to be discussed)
2. an integer; we will get to that
3. user who owns the file
4. group of the file
   - Allows groups of people to work together
   - Details when we discuss "permissions"

Long listing columns, ctd.

5. Size of the file, in bytes.
   - For devices: the major and minor number
     - Don't worry about what that means

6,7,8. Date and time of last modification.
   - If the file is old, the time is replaced by the year

9. Name of the file.

Example long listing

```
prompt$
```

Example long listing

```
prompt$ ls -lF
```

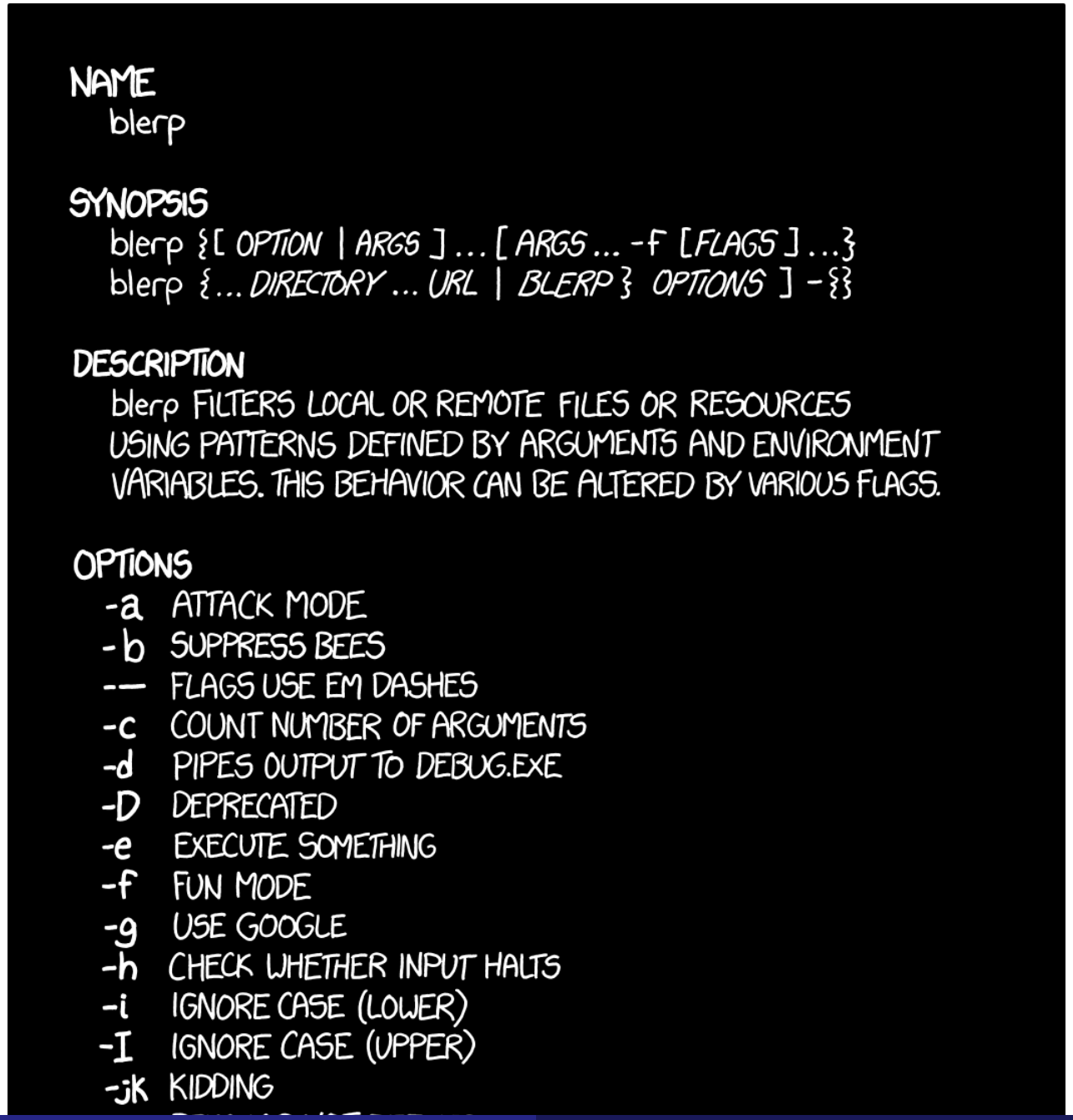
Example long listing

```
prompt$ ls -lF
total 96
-rwx------ 1 alice hackers  6425 May  4 09:47 a.out*
-rw-r----- 1 alice hackers   392 Jan 16  2010 bar.cc
-rw------- 1 alice hackers 56438 Oct  5  2007 core
drwxr-xr-x 2 alice hackers  4096 Apr 28 11:05 cs229/
drwxr-xr-x 2 alice hackers  4096 Dec  3  2011 cs252/
-rw-r----- 1 alice hackers   937 Jan 16  2010 foo.cc
-rw-r----- 1 alice hackers    88 May  4 09:43 hello.c
-rw-r----- 1 alice hackers   104 May  3 18:23 hello.h
-rw------- 1 alice hackers  3584 May  4 09:46 hello.o
drwxr-xr-x 2 alice hackers  4096 Apr 26 14:44 math168/
drwxr-xr-x 2 alice hackers  4096 Dec  1  2009 se101/
prompt$
```

Summary of today's commands

- `cd` : Change working directory.
- `info` : Fancy browsing of the online manual.
- `ls` : List the contents of a directory.
- `man` : Browse the online manual.
- `mkdir` : Create a directory.
- `pwd` : Print working directory.
- `rmdir` : Remove a directory.
- `type` : Is a command built-in, or not?

An appropriate xkcd comic: http://www.xkcd.com/1692



End of lecture