Quotes for the day:

"Software stands between the user and the machine" - Harlan D. Mills

Software can help the user in their daily endeavors or stand in the way.

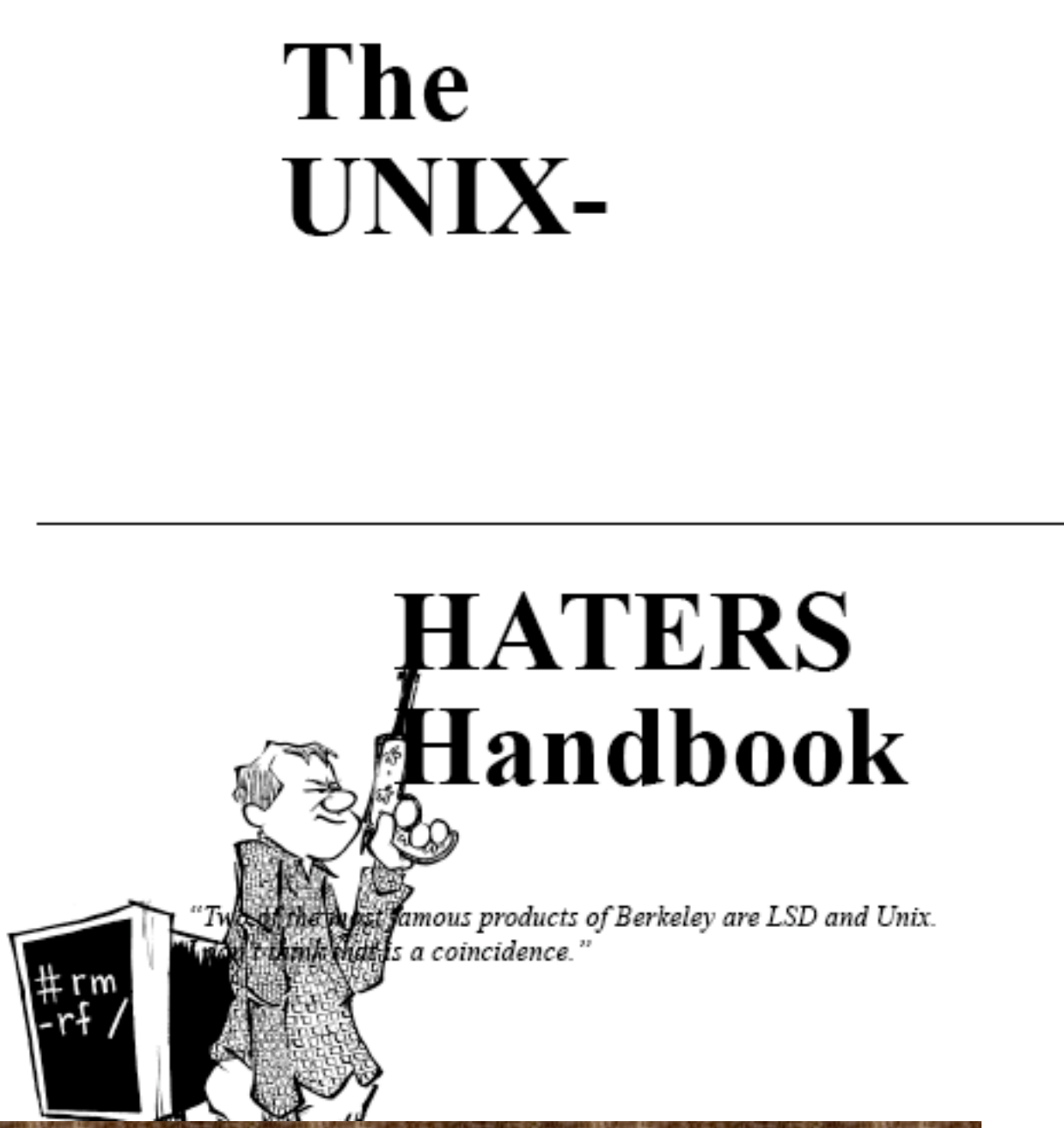
"the UNIX operating system, a unique computer operating system in the category of help, rather than hindrance."
Introducing the UNIX System, McGilton and Morgan, 1983.

or

The trouble with UNIX: The user interface is horrid, Norman, D. A. Datamation, 27, No. 12, 139-150.

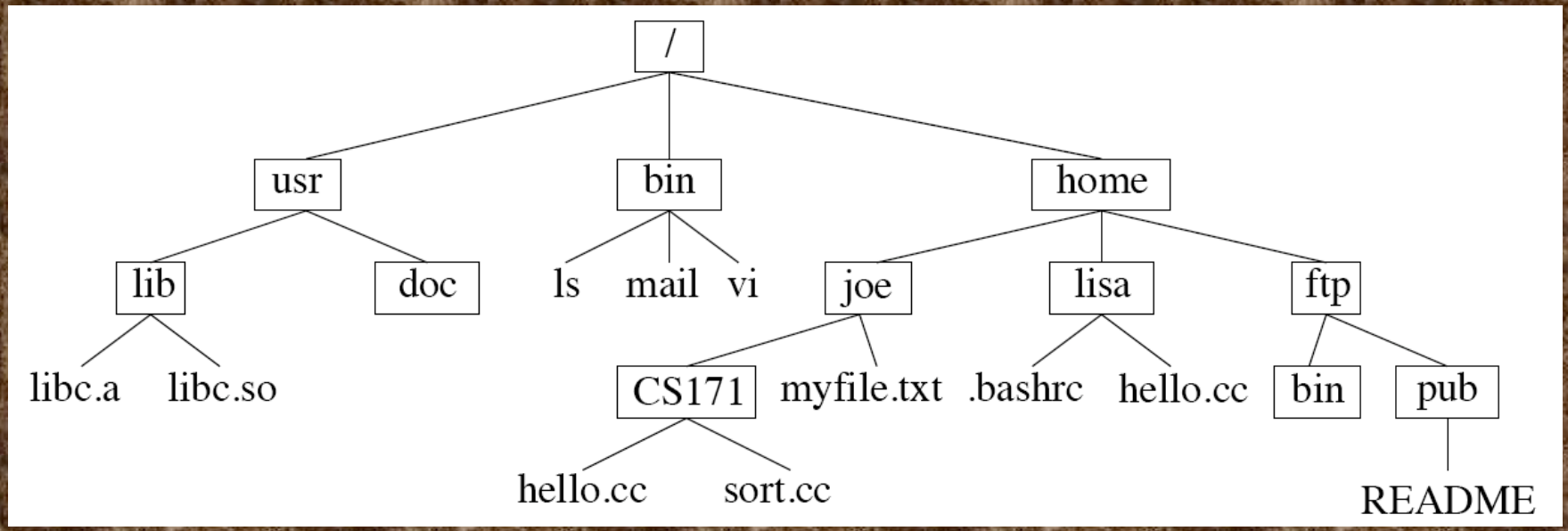
"Two of the most famous products of Berkeley are LSD and Unix. I don't think that this is a coincidence."

Anonymous



Before looking at more Unix commands, we will first look at the FILE STRUCTURE (how files [documents on Mac and Windows] are stored/organized).

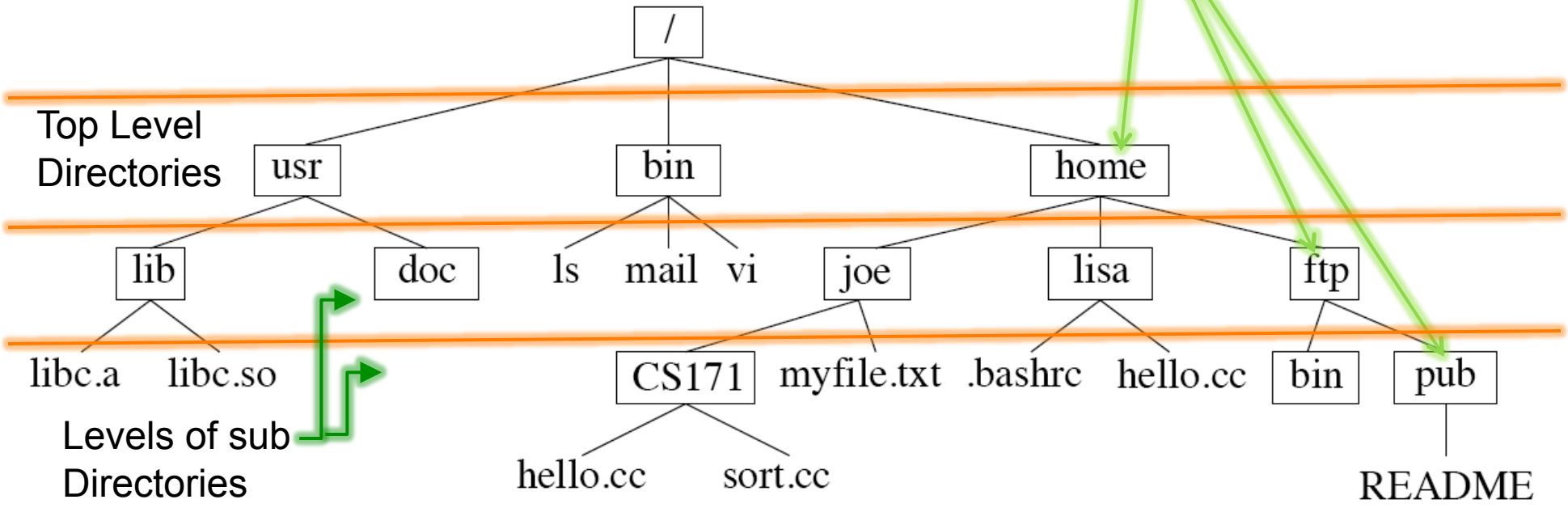
Unix uses a hierarchical file system (as does Mac and Windows).



Looks like an upside down tree.

Starts at top with "/", called "root".

Uses "/" to separatedirectories (known as folders on Mac or Windows)



So this file is

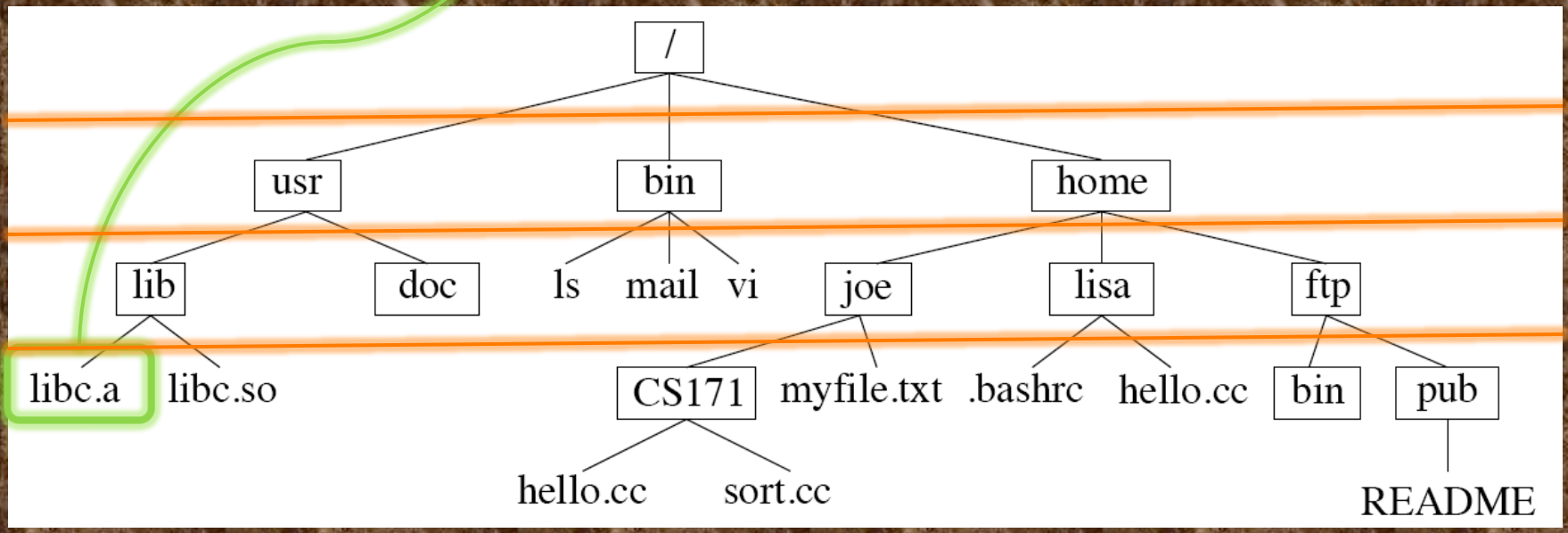
root

separators ("/")

/usr/lib/libc.a

filename

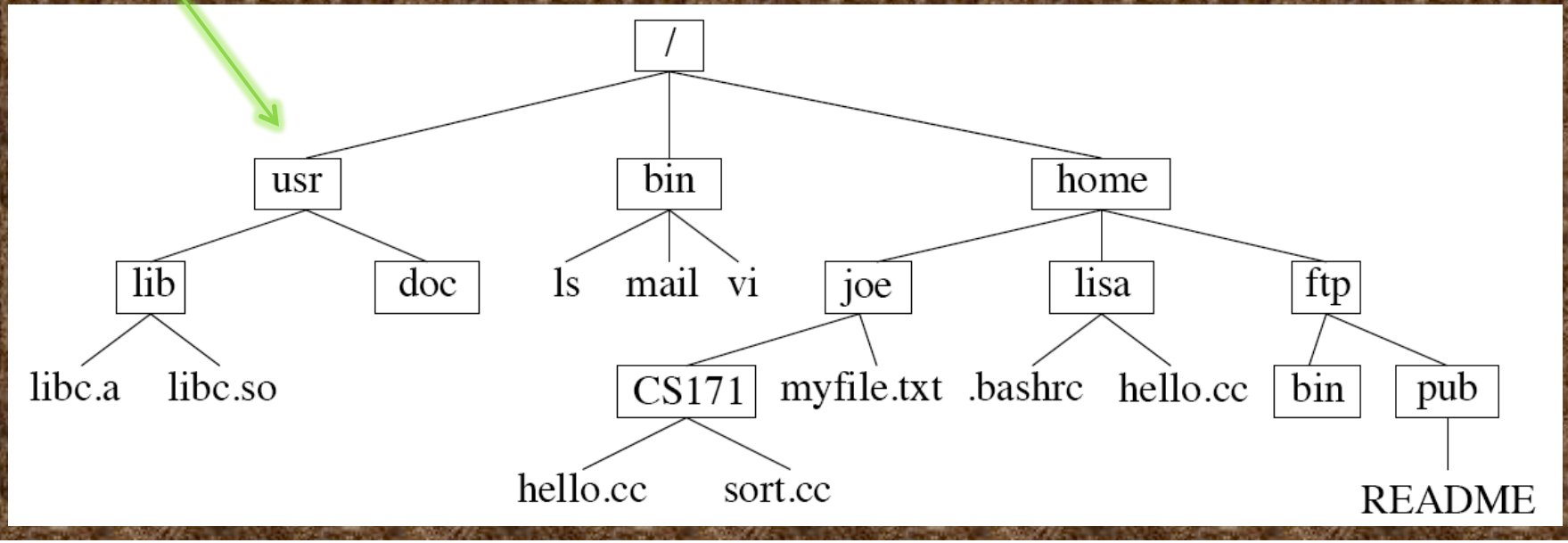
path



Some commands:

pwd – print working directory – tells us where we are. (notice logical command name.)

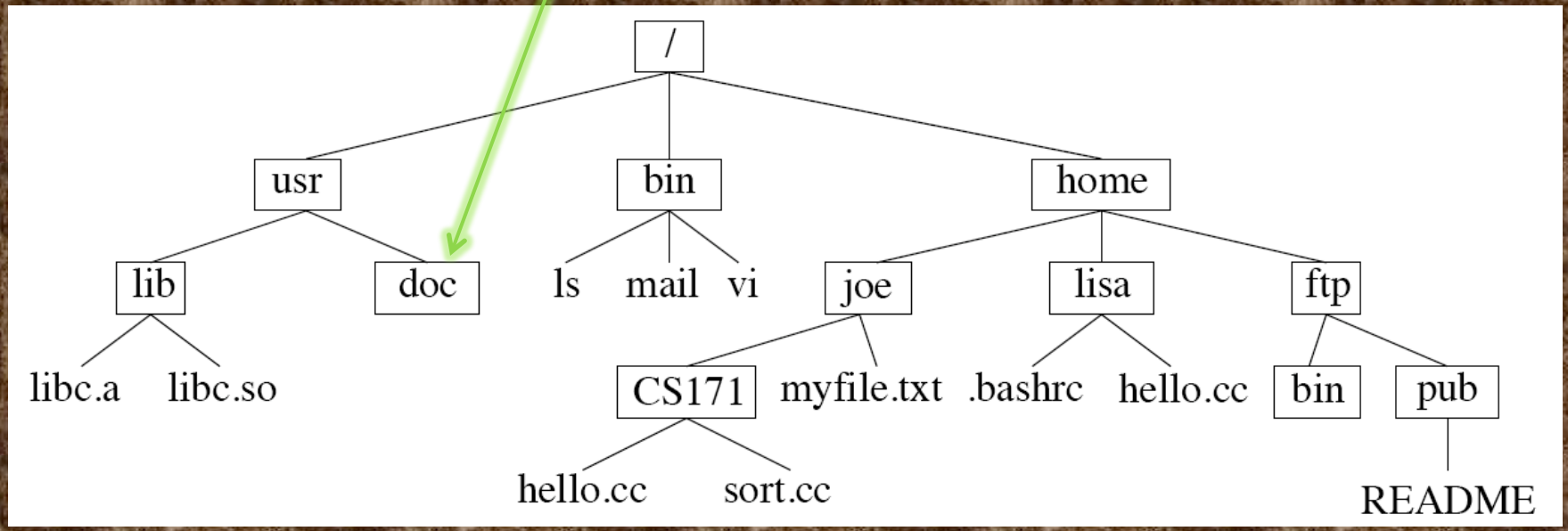
```
smalleys-imac-2:usr smalley$ pwd<CR>
/usr
```



How to move between directories - going up and down the directory tree-

To go down to the directory doc we "change directory" = "cd"

```
smalleys-imac-2:usr smalley$ cd doc<CR>
smalleys-imac-2:doc smalley$
```



Aside:

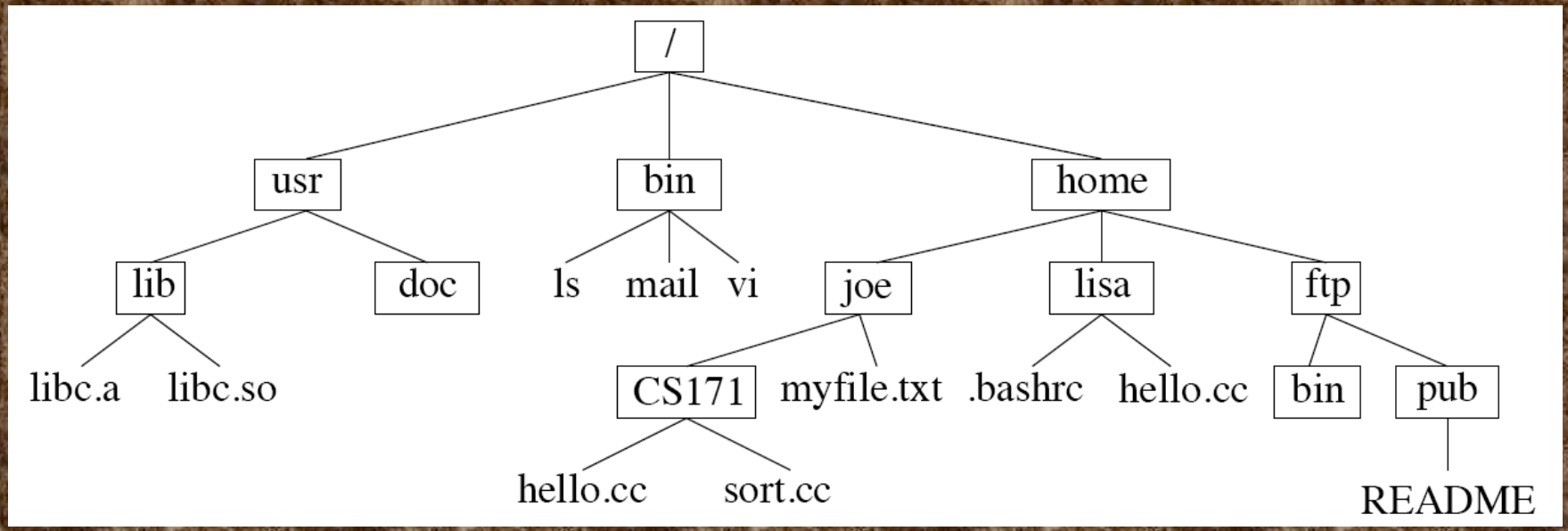
Unix sub philosophy –

Minimize typing (on teletype) – so use short (2, in extreme cases 3 character) command names constructed from description of the command.

(this is a "feature" of Unix, compared to other OSes)

Notice that you have to know where you are and what subdirectories are contained there to navigate around.

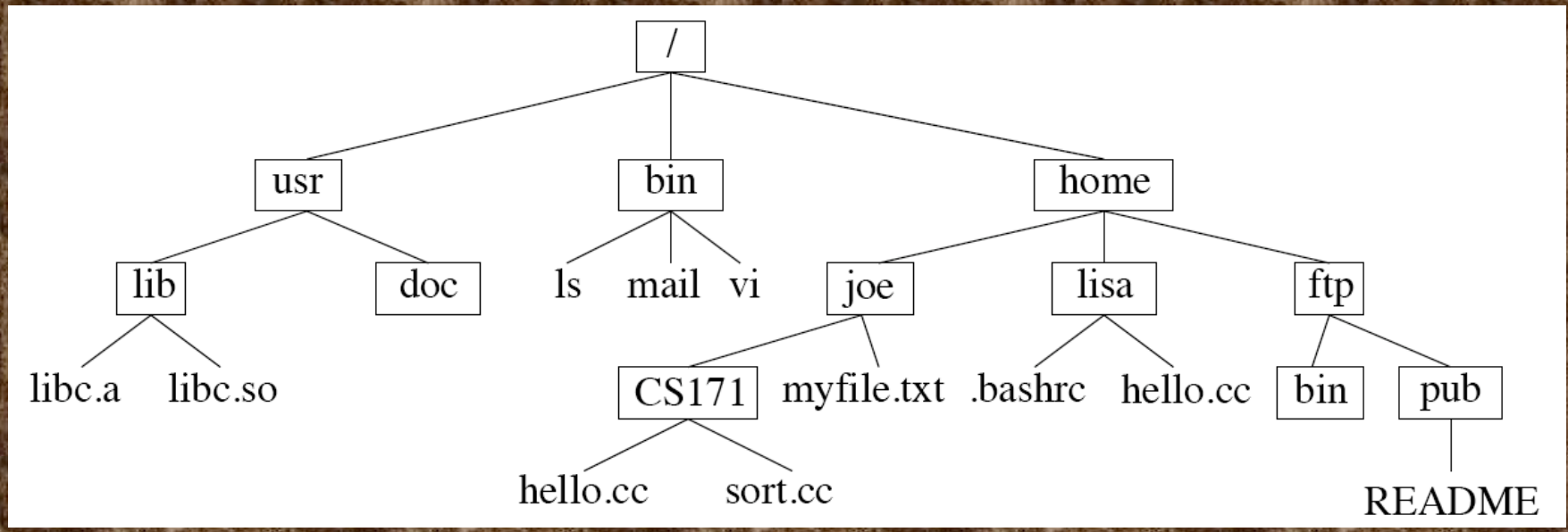
Unix does not provide a display of the picture below.



We can also go up the directory structure.

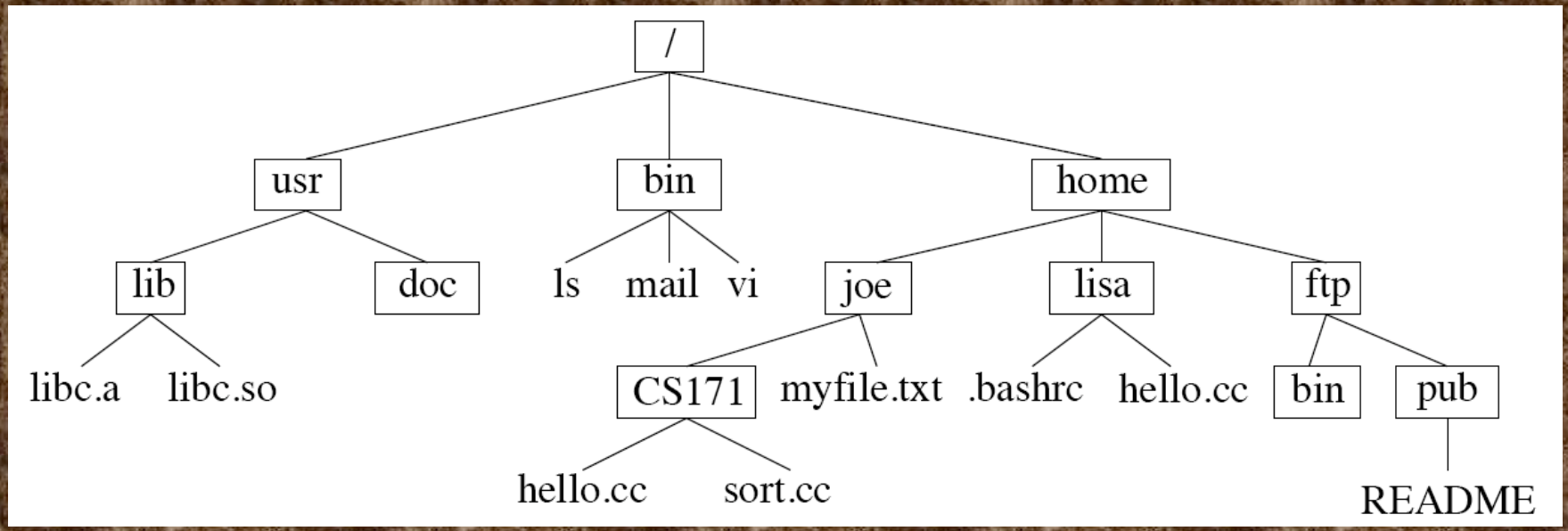
To return to usr

```
smalleys-imac-2:doc smalley$ cd ..<CR>
smalleys-imac-2:usr smalley$
```

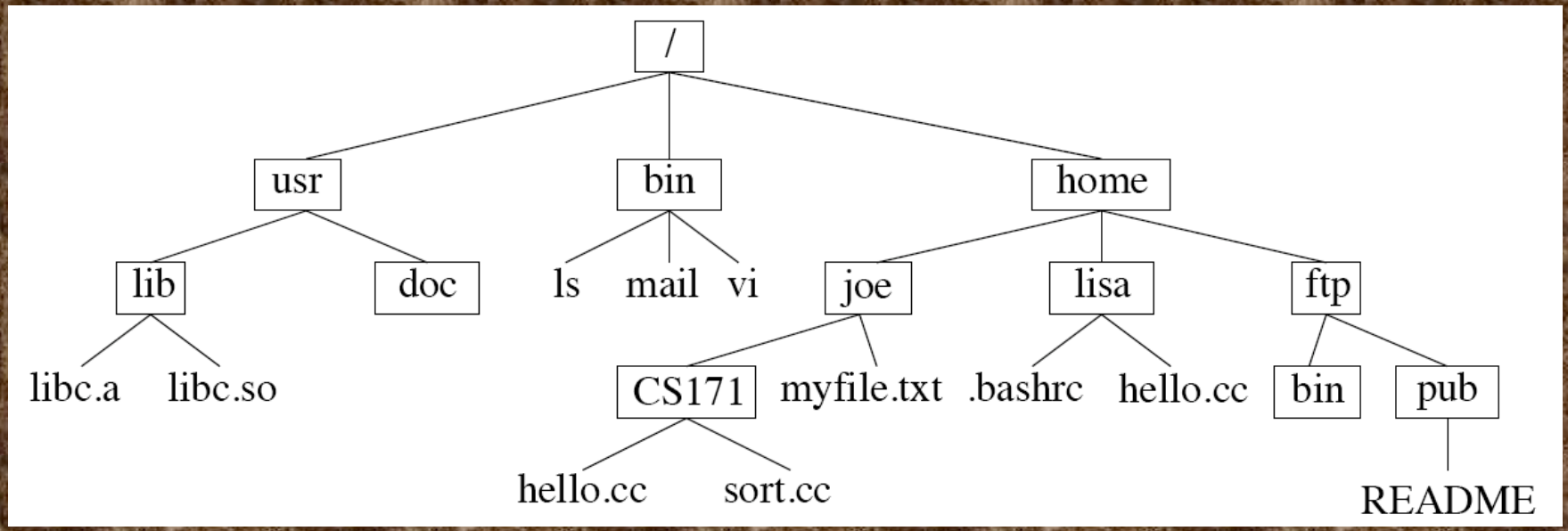


This is a little strange ---

The double dot ("..") signifies the directory directly above you (up) in the directory structure (tree).



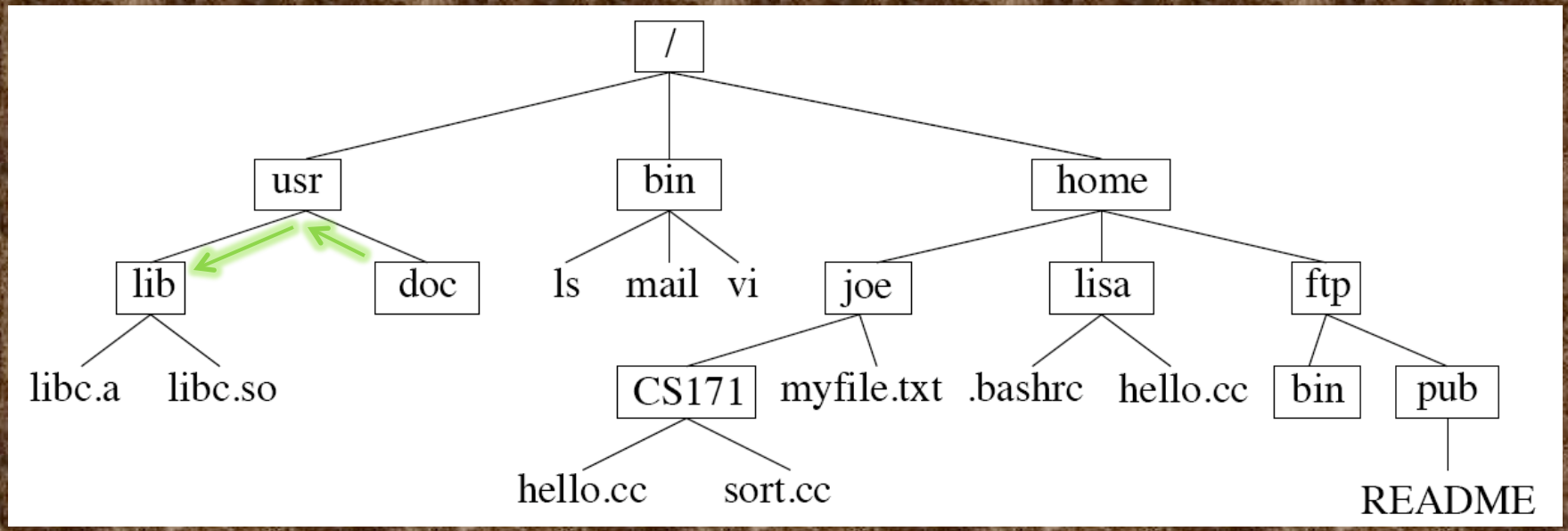
For use later - a single dot (".") signifies the current or working directory.



How do we go from doc to lib?

We have to go up one level; then down one level. This can be done with one command.

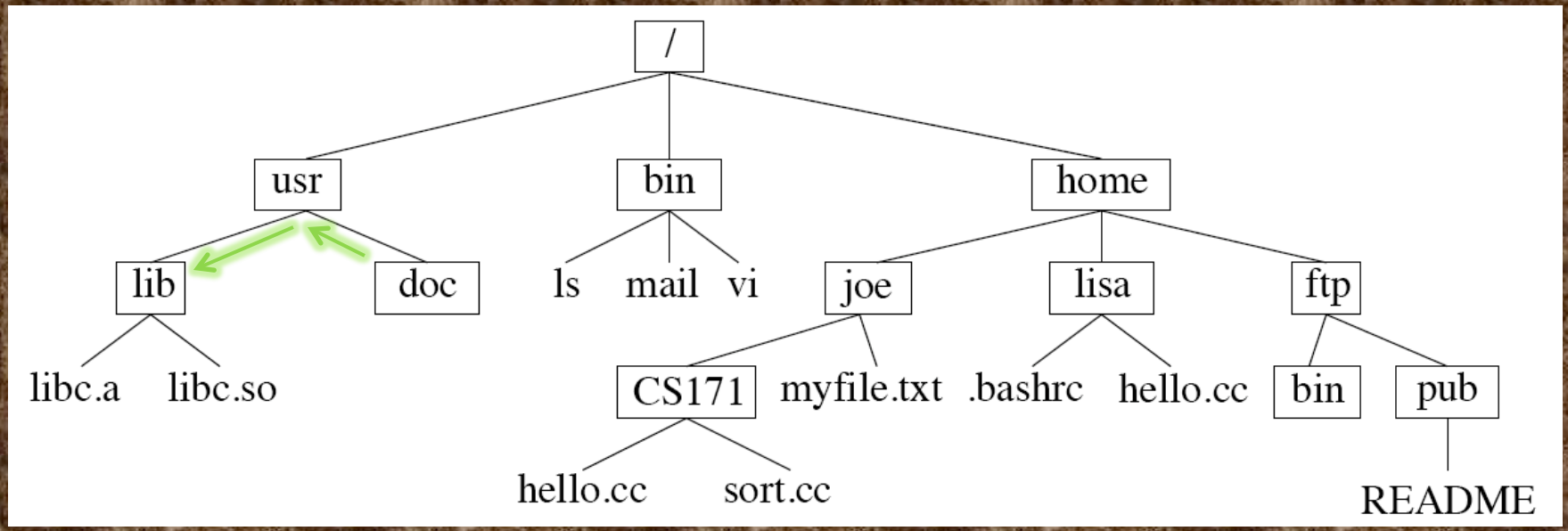
```
smalleys-imac-2:doc smalley$ cd ../lib<CR>
smalleys-imac-2:lib smalley$
```



How do we go from doc to lib?

We could also have done this.

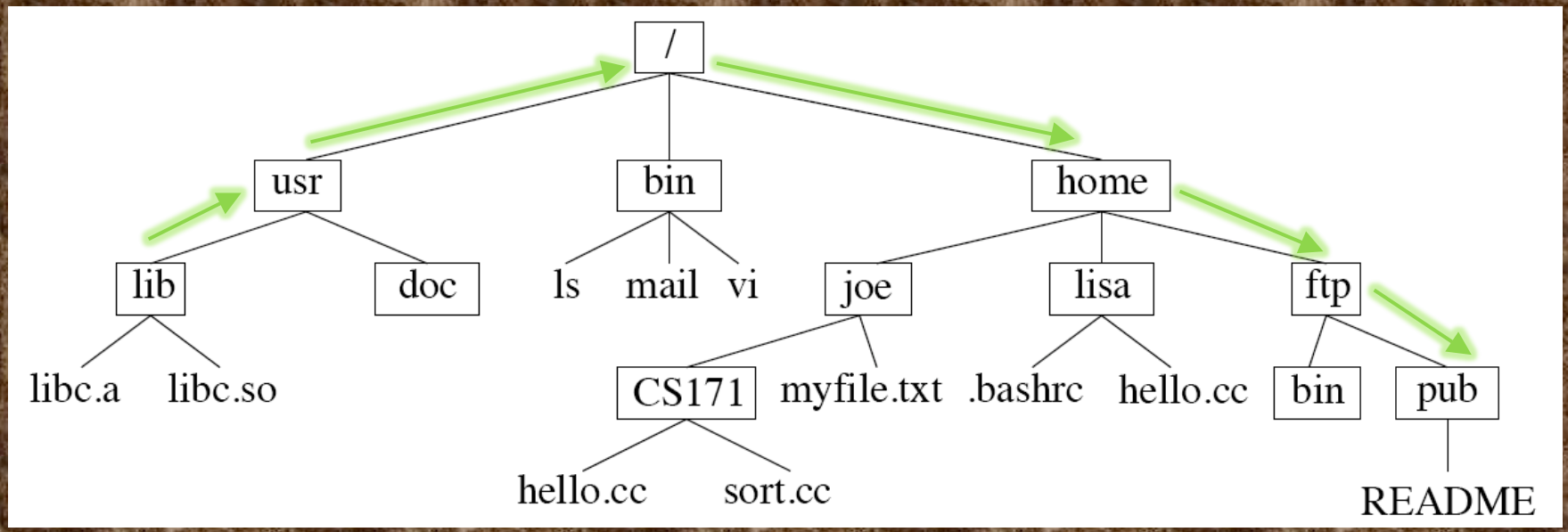
```
smalleys-imac-2:doc smalley$ cd /usr/lib<CR>
smalleys-imac-2:lib smalley$
```



Say we want to go to "pub"

```
smalleys-imac-2:lib smalley$
cd ../../home/ftp/pub<CR>
```

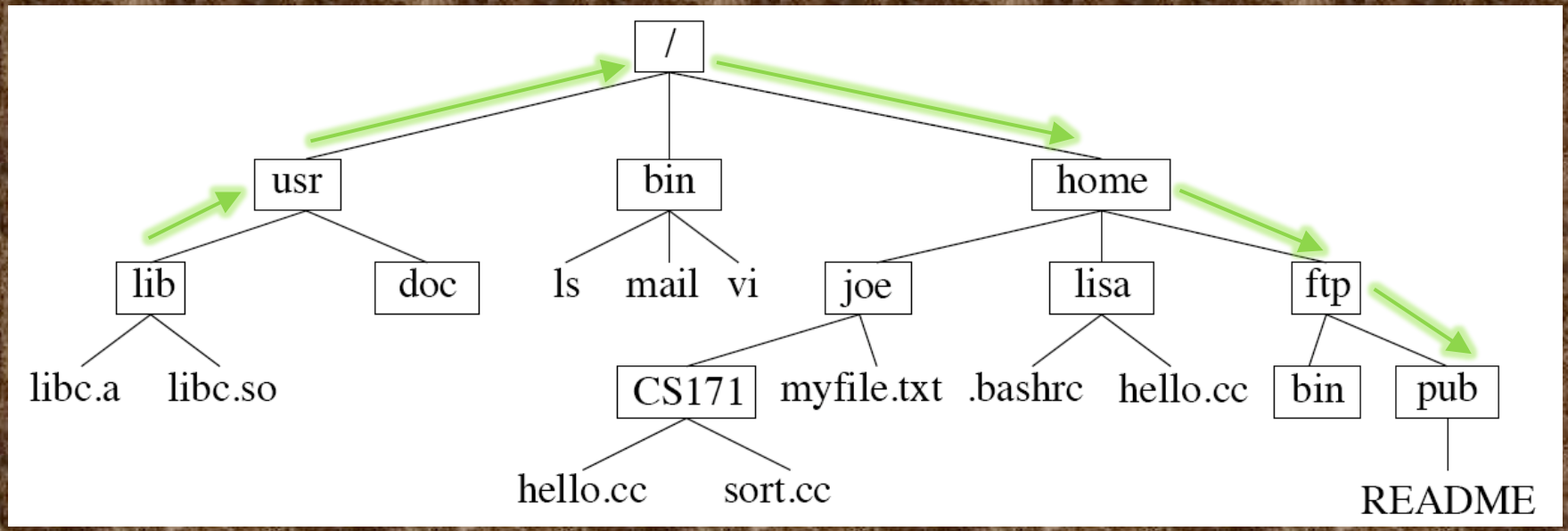
We went up two, then down three.



Say we want to go to "pub"

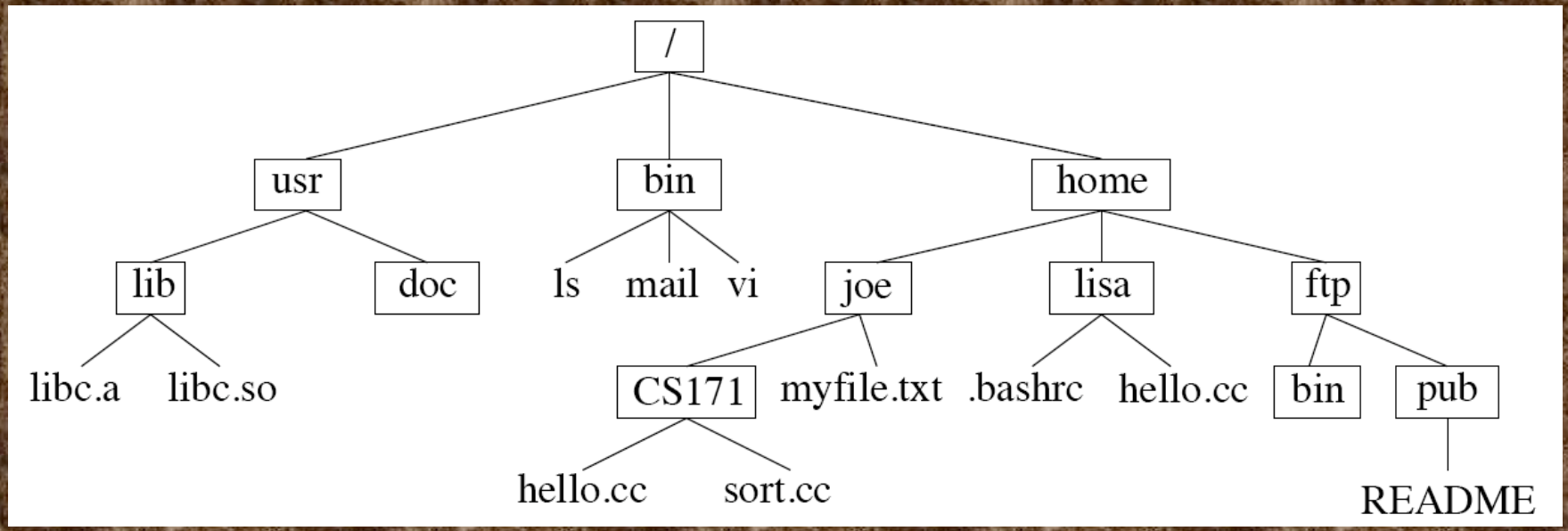
We could (actually would) also have done (and is simpler).

```
smalleys-imac-2:pub smalley$
cd /home/ftp/pub<CR>
```



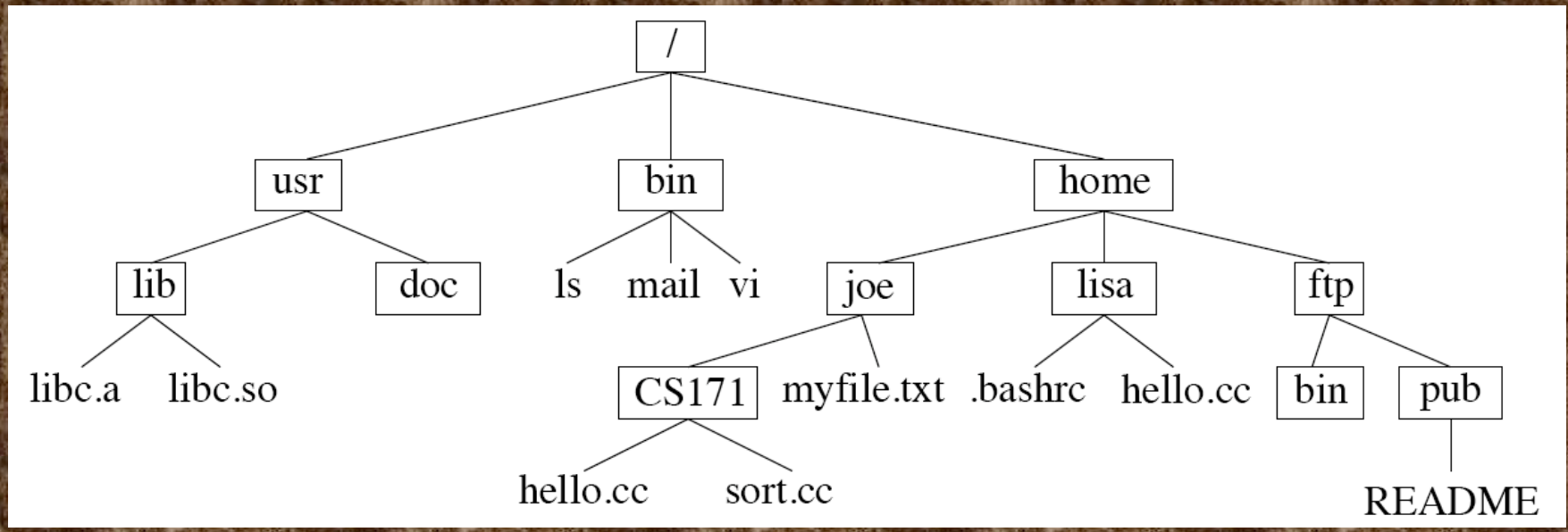
I'm doing this on a Mac – notice that the prompt has been programmed to tell us where we are! (power of unix.)

```
smalleys-imac-2:lib smalley$
cd /home/ftp/pub<CR>
smalleys-imac-2:pub smalley$
```



Go directly to "root" directory ("/")

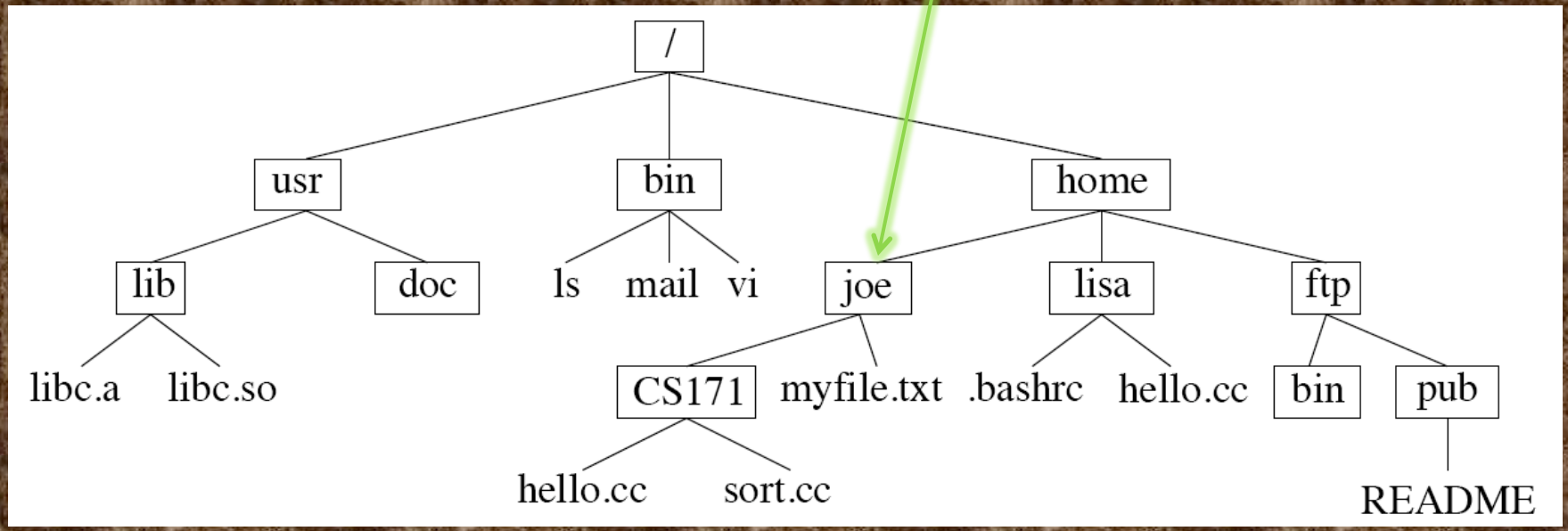
```
smalleys-imac-2:lib smalley$ cd /<CR>
smalleys-imac-2:/ smalley$
```



Go from anywhere directly to your "home" directory (assume I'm "joe").

```
smalleys-imac-2:/ smalley$ cd ~<CR>
smalleys-imac-2:/home/joe smalley$
```

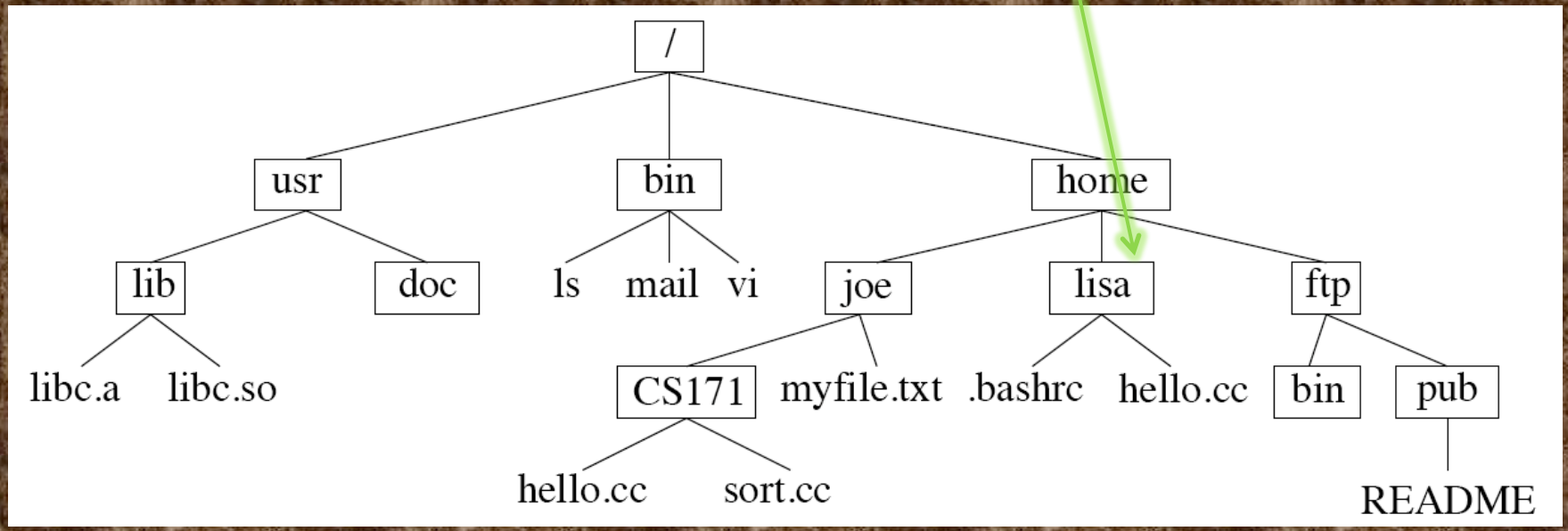
uses tilde "~"



Go from anywhere directly to someone else's home directory

```
smalleys-imac-2:/ smalley$ cd ~lisa<CR>
smalleys-imac-2:/home/lisa smalley$
```

also uses tilde "~"



The tilde character "~"

-refers to your home directory when by itself,

- or that of another user when used with their home directory name (the same as their user name).

(The shell expands the "~" into the appropriate character string "/home/joe" or "/home/lisa")

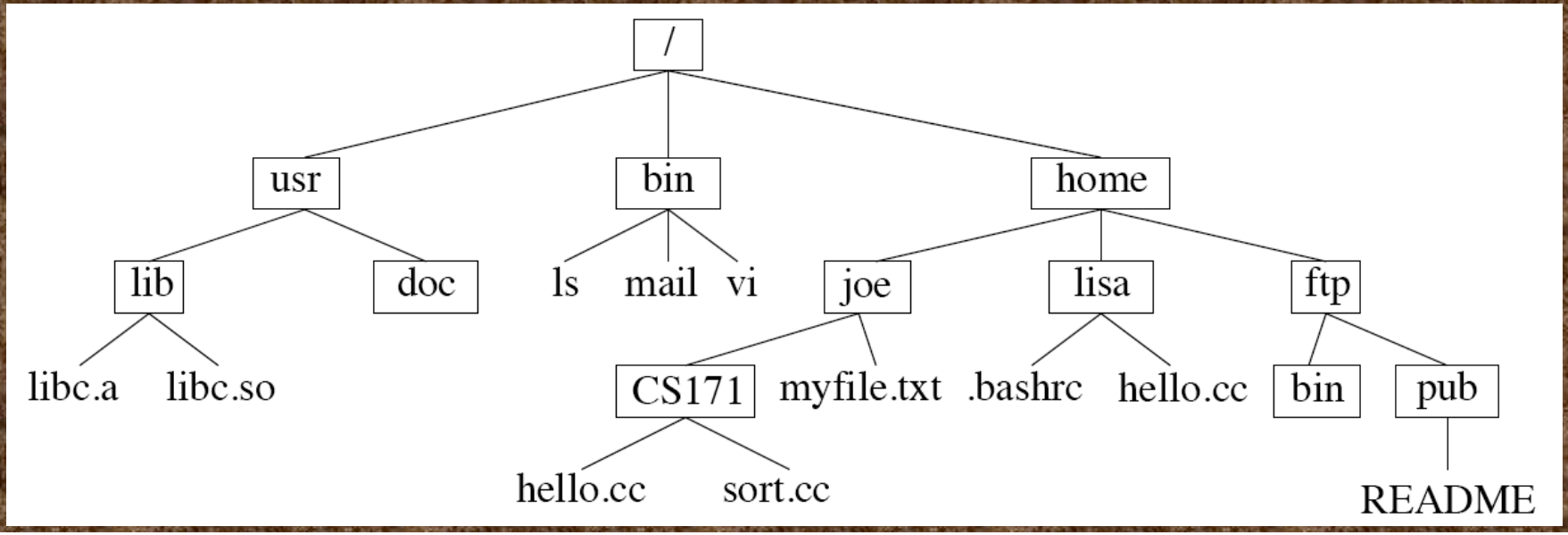
Specifying file names

full path /usr/lib/libc.a

relative path

(if in directory lib) libc.a

(if in another directory ../lib/libc.a

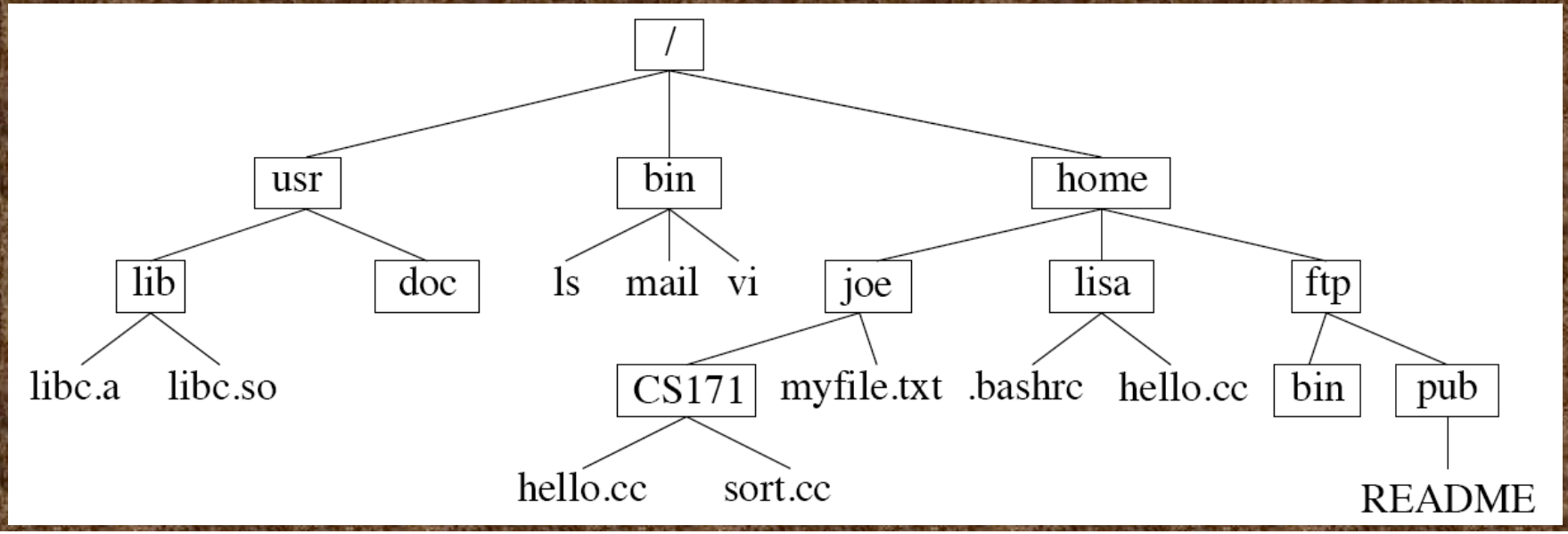


Specifying file names

abbreviations

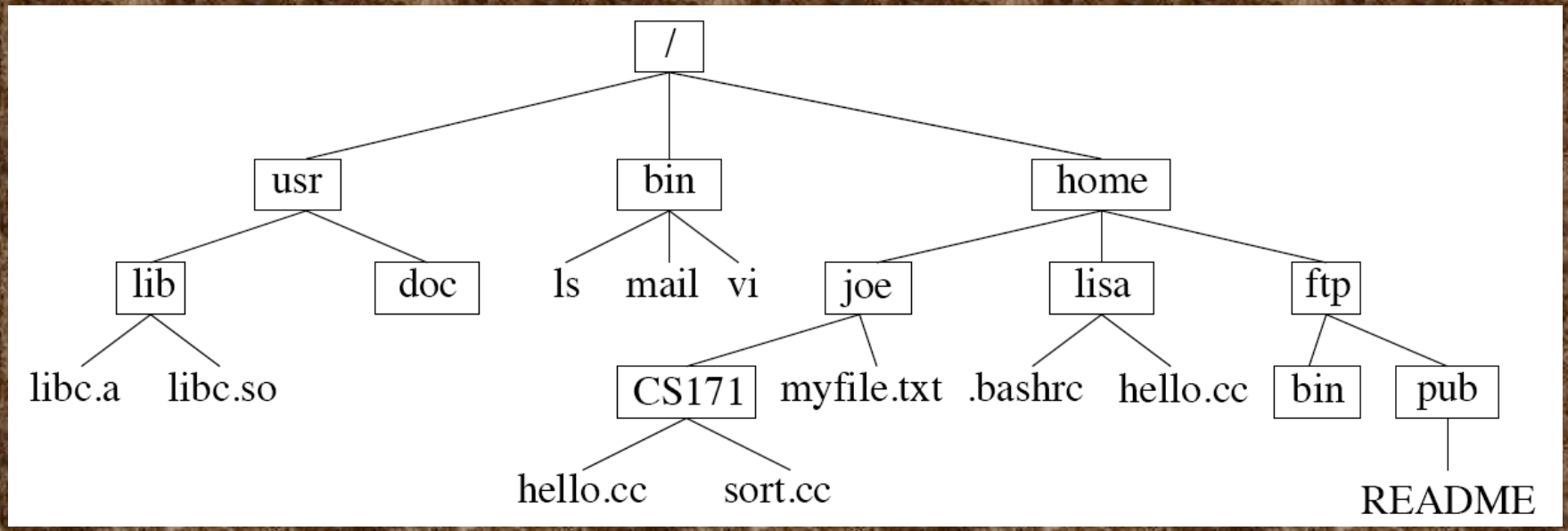
(if I am joe) ~/CS171/hello.cc

(if I am not joe) ~joe/CS171/hello.cc



You have to keep track of the file structure in your head

or have a way to find out what files are in the working directory.



What files are in working directory?

Use the "list" command, which is actually "ls".

(Compare this to VAX-VMS which uses "directory" – much longer)
(but Unix supporters forget to tell you that it can be shortened, by dropping letters off the back, to

"director", "directo", "direct", "direc", "dire", or "dir"

at which point continued shortening stops as "di" is non-unique as another command also begins with the letters "di". This means you can write "com" files – same as shell scripts or batch files – to be readable using "directory" or cryptic using "dir".)

Listing working directory (where we are) contents with "ls" command.

```
smalleys-imac-2:~ smalley$ ls<CR>
Adobe SVG 3.0 Installer    vel.dat
Desktop                                    heflen_web.dat
Documents                                  isc0463.dat
...
Downloads                                  nuvel-1a.dat
gpsplot.dat
smalleys-imac-2:~ smalley$
```

Note – this example is from a Mac, where spaces are allowed in file names. The file

`Adobe SVG 3.0 Installer`

has spaces in the name.

On Unix this is somewhat of a problem.

Spaces are allowed in filenames in Unix (all characters but the "/" are allowed in filenames!) , but spaces, and special characters

!@#$%^&*()_+|?><`[]{}\'":;

are not handled nicely as most of them mean something special to the shell.

The problem with spaces is that the command interpreter of the shell parses (breaks) the command line up into tokens (individual items) based on the spaces.

So our file name gets broken into 4 small distinct character strings ("Adobe", "SVG", "3.0", and "Installer") which causes confusion.

So we have to "protect" the spaces from the interpreter.

This is done with quotes.

We refer to this file using

```
"Adobe SVG 3.0 Installer"
```

or

```
'Adobe SVG 3.0 Installer'
```

(We will see the difference between single, ', and double, ", quotes later.)

ls: lists files and subdirectories of the specified path.

```
%ls /gaia/home/rsmalley<CR>
bin    src usr world.dat

%ls<CR
```

lists everything in the current directory

```
%ls ~/bin<CR>
```

lists everything in your directory /bin (not the system /bin).

ls: getting more information than just file name.
Tells kind of file - list directories with '/' & executables with '*'

```
smalleys-imac-2:~ smalley$ ls -F<CR>
Adobe SVG 3.0 Installer  vel.dat
Desktop/                                heflen_web.dat
Documents      /                           isc0463.dat
mymap.sh*                                  a.out*
. . .
Downloads/                              nnr-nuvel-1a.dat
gpsplot.dat
smalleys-imac-2:~ smalley$
```

This example introduces the switch, or flag, "-F", which modifies the output.

The output now identifies if the file is a

"regular file" (nothing appended), a "directory" (slash appended), or an

"executable file" (asterisk appended, = program, application).

More switches

list entries beginning with the character dot, '.', which is normally hidden or invisible, using the -a flag & show the listing in long format using the -l flag.

```
smalleys-imac-2:~ smalley$ls -alF<CR>
-rwx------  1 rsmalley rsmalley       1201 Jul 10 15:03 .cshrc*
drwx------  1 rsmalley rsmalley      16384 Aug  1 13:50     bin/
-rw-------  1 rsmalley rsmalley  186668405 Jul 31  2007     world.dat
```

Can combine flags as above (-alF) or put individually (-a -l -F).

```
-rwx------  1 rsmalley rsmalley       1201 Jul 10 15:03 .cshrc*
drwx------  1 rsmalley rsmalley      16384 Aug  1 13:50 bin/
-rw-------  1 rsmalley rsmalley  186668405 Jul 31  2007 world.dat
```

What is the extra information

First character, "d" for directory, "-" for regular file, plus about 10 other things for other types of files.

The next 9 characters show read/write/ execute privileges for owner, group, and all (or world or other).

```
-rwx------  1 rsmalley rsmalley       1201 Jul 10 15:03 .cshrc*
drwx------  1 rsmalley rsmalley      16384 Aug  1 13:50 bin/
-rw-------  1 rsmalley rsmalley  186668405 Jul 31  2007 world.dat
```

If have read, write or execute privileges has "r", "w", or "x" respectively. If not, has a "-".

So the owner has read and write privileges on all the files or directories, and execute privileges on the executable file (indicated by the "*") .cshrc and the directory bin (although one cannot execute a directory). Group and world or other have no privileges.

```
-rwx------  1 rsmalley rsmalley       1201 Jul 10 15:03 .cshrc*
drwx------  1 rsmalley rsmalley      16384 Aug  1 13:50 bin/
-rw-------  1 rsmalley rsmalley  186668405 Jul 31  2007 world.dat
```

Privileges can also be specified or displayed in OCTAL (base 8) with each bit of the octal value representing the permission/privilege.

rwx=111=7
rw-=110=6
r--=100=4
--x=001=1

etc. for owner, group, world.

```
-700  1 rsmalley rsmalley       1201 Jul 10 15:03    .cshrc*
d700  1 rsmalley rsmalley      16384 Aug  1 13:50    bin/
-600  1 rsmalle yrsmalley  186668405 Jul 31  2007    world.dat
```

This is “much better” as it uses fewer characters

```
-rwx------  1 rsmalle yrsmalley        1201   Jul 10 15:03   .cshrc*
drwx------  1 rsmalley rsmalley       16384   Aug  1 13:50   bin/
-rw-------  1 rsmalley rsmalley   186668405   Jul 31  2007   world.dat
```

Temporarily skipping the next 3 columns, we then have the file size in bytes, the date the file was last modified, and the file name.

Switches/flags and manual pages:

Most Unix commands have switches/flags that can be specified to modify the default behavior of the command.

How do we find what switches are available and what they do?

But first – a little more about files on Unix.

Files on Unix are "flat"

Just strings of bytes with the information contained in the file.

What do we mean by this?

The files do not have headers or tailers with metadata about the file, icons, etc.

Unix does not provide

Indexed

or relational database

files.

(but you can write a program to provide them! Oh the power of Unix.).

To Unix

EVERYTHING is a file, which is a string of bytes.

All equal.

Returning to the question of finding out about available switches.

The developers of Unix thought of this and provided documentation through the manual command - "man". To read the man page for the list command.

```
alpaca.ceri.memphis.edu/rsmalley 160:> man ls
Reformatting page.  Please Wait... done

User Commands                                                ls(1)

NAME
ls - list contents of directory

SYNOPSIS
     /usr/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]

     /usr/xpg4/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]

DESCRIPTION
     For each file that is a directory, ls lists the contents  of
     the  directory.  For  each file that is an ordinary file, ls
     repeats its name and any other  information  requested.  The
```

This goes on for quite a while. Note the --More-- (9%) at the bottom - says we are 9% done (oh joy on a teletype!)

output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format for output directed to a terminal is multi-column with entries sorted down the columns. The -1 option allows single column output and -m enables stream output format. In order to determine output formats for the -C, -x, and -m options, ls uses an environment variable, COLUMNS, to determine the number of character positions available on one output line. If this variable is not set, the terminfo(4) database is used to determine the number of columns, based on the environment variable, TERM. If this information cannot be obtained, 80 columns are assumed.

The mode printed under the -l option consists of ten characters. The first character may be one of the following:

--More--(9%)

continuing

```
     d     The entry is a directory.

     D     The entry is a door.

     l     The entry is a symbolic link.

     b     The entry is a block special file.

     c     The entry is a character special file.

     p     The entry is a FIFO (or "named pipe") special file.

     s     The entry is an AF_UNIX address family socket.

     -     The entry is an ordinary file.

     The next 9 characters are interpreted as three sets of three
     bits  each. The first set refers to the owner's permissions;
     the next to permissions of others in the user-group  of  the

SunOS 5.9           Last change: 19 Nov 2001                    1

User Commands                                              ls(1)

     file; and the last to all others. Within each set, the three
     characters indicate permission to read,  to  write,  and  to
     execute  the  file  as a program, respectively. For a direc-
     tory, ``execute'' permission is interpreted to mean  permis-
     sion to search the directory for a specified file. The char-
     acter after permissions is ACL indication. A  plus  sign  is
     displayed if there is an ACL associated with the file. Noth-
     ing is displayed if there are just permissions.

     ls -l (the long list) prints its output as follows  for  the
     POSIX locale:

--More--(16%)
```

This goes on for several pages.

Try the manual command on a number of commands.

Now for some more commands.

Making & removing directories

mkdir: make directory

```
% mkdir  bin  src  Projects  Classes<CR>
```

Makes 4 directories: bin, src, Projects, and Classes in the working directory.

Making & removing directories

rmdir: remove directory - only works with empty directories so is safe.

```
% rmdir bin  src  Projects  Classes
```

Removes the 4 directories bin, src, Projects, and Classes in the working directory -- if they are EMPTY.

(from here on, will drop the <CR> at end).

Removing files and directories

rm: remove files or directories

A very straightforward and potentially dangerous command.

There is no trash can on a unix machine.

Once you hit the <CR> it is GONE.

Removing files

rm: remove files or directories

CERI accounts are set up so that rm is aliased to rm -i (more on aliases later), which means the computer will ask you if you really want to remove the file(s) one at a time

```
% which rm
rm:    aliased to /bin/rm –i
```

Removing files

rm: remove files or directories

```
%rm f1
remove f1? yes<CR>
%
```

and bye-bye file.

Removing files

```
%rm f1
remove f1?
```

Valid answers.

Yes, yes, Y, y – to accept and erase.

No, no, N, n – to not erase.

<CR> - does not erase.

Something it does not understand – does not erase.

Removing files

Remember that Unix is lean and mean.

It is a multi-user system and once the disk space associated with your file is released, the system can write somebody elses file into it immediately.

There is NO RECOVERING removed files.

Removing files

Without the *-i* option set - this is what we would get.

```
%rm f1
%
```

and bye-bye file.

So if you made a typo - tough.

Removing files

If the *-i* option was not set - you can get it by typing *-i* yourself (but sooner or later you will mess up and forget on one!).

```
%rm –i f1
remove f1? yes<CR>
%
```

and bye-bye file.

So if you made a typo - tough.

Removing files

Say you are 100% sure and don't want to have to answer the question. You can return to the original definition of rm using the "\".

```
%\rm f1
%
```

and bye-bye file without prompting.

So if you made a typo - tough.

General Unix behavior.

The "\" undoes an alias and gives you the default version of the command.

Removing files

We will see more potential rm disasters when we get to wildcards.

(If you have sufficient privilages, it is possible to accidently erase the whole operating system!!!)

Making & removing files and directories

rm -d: also removes directories
(rm does not otherwise remove directories.)

rm -r: removes directory and all subdirectories and files; implies -d

can be very dangerous... one typo could remove months of files

```
% rm -r  Classes
```

Making & removing files and directories

```
% rm -r  Classes
```

With the CERI alias for rm to rm -i, this command will prompt you for each file!

Gets tedious - and makes you want to do

```
%\rm -r  Classes
```

Which is VERY, VERY DANGEROUS.

Manipulating files

cat: concatenate files and sends output to Standard OUT.

If you want the concatenated files in another file – you have to redirect the output.

Manipulating files

cat: Since it dumps the entire file contents to the screen

- we can use it to "print out" or "type out" a file.

Manipulating files

cat: Since it dumps the entire file contents to the screen

- we can use it to "print out" or "type out" a file.

Another Unix philosophy issue - use of side effects.
We don't need another command to print or type the contents of a file to the screen as it is a side effect of the cat command.

Manipulating files

cat: make one file out of file1, file2 and file3 and call it alltogether.

```
%cat file1 file2 file3 > alltogether
```

This command (does not need input redirection, exception to regular rule that input comes from Standard IN) takes files file1, file2, and file3 and puts them into file alltogether.

Manipulating files

OK, what does this do?

```
%cat > myfile
```

Manipulating files

OK, what does this do?

```
%cat > myfile
```

It takes Standard IN (the keyboard) and puts it into the file myfile.

Looking at files

OK, what does this do?

```
%cat > myfile
```

How does one get it to stop?

Enter "^D" or "^Z", where "^" is the control (ctrl) key and you hold it down and then press the D or Z.

Notice the logic associated with the input, output, and use of the command.

This type of thinking, or logic, permeates Unix.

When you cat a long file it flies by on the screen (and off the top).

On newer GUIs there are scroll bars and you can scroll up and down.

On the older interactive terminals the text disappeared off the top.

Not good.

This problem was fixed by another Unix program that takes Standard IN and puts it to Standard OUT a screenful at a time. (has to know about screens).

(This way, following the Unix philosophy, the cat program could be lean and mean. It did not have to figure out if it was going to the screen, etc., it just sends stuff to Standard OUT.)

So we pipe the output into another program that handles the screen display.

This program is called more.

```
%cat myfile | more
```

The program more puts up a screens worth of text and then waits for you to tell it to continue (using the space bar for a new page worth and <CR> for a new lines worth of the file. ^Z to quit more.)

Looking at files

more can also be used directly (and should be, there is not need for the cat on the last slide, which was only to demonstrate pipes).

```
%more myfile
```

Or

```
%more < myfile
```

(more was written outside UNIX "club" and borrowed by UNIX, does not strictly follow UNIXphilosophy.)

Looking at files

less: same as more but allows forward and backward paging.

(in OSX, more is aliased to less because less is more with additional features.)

(We will discuss aliases later.)

Manipulating files

paste: concatenate files with each file a new column; when used on a single file, it dumps the entire file contents to the screen

Looking at files

head -nX: prints the first X number of lines to the screen; default is 10 lines if -n is not specified.

tail -nX: prints the last X number of lines to the screen; default is 10 lines if -n is not specified.

Piping and Redirect

Input and output on the command line are controlled by the |, >, <, and ! Symbols.

| : pipe function; sends the output from command on left side as input to the command on the right side.

(We have seen these actions already.)

Piping and Redirect

Example pipe

```
% ls | head -n5
1002
1019
1023
1045
1046
```

Piping and Redirect

">" redirects standard output (screen) to a specific file*

```
% ls | head -n5 > directory.list
% more directory.list
1002
1019
1023
1045
1046
```

* In tcsh, this will not overwrite (clobber) a pre-existing file with the same name. In the bash shell, the > overwrites (clobbers) any pre-existing file with no warning

Piping and Redirect

>! : redirects standard output (screen output) to a specific file and overwrite (clobber) the file if it already exists *

```
%ls | head –n5 >! directory.list
%more directory.list
1002
1019
1023
1045
1046
```

*This syntax is specific to tcsh, your default CERI shell (in bash this will put the output into a file named "!", what you told it to do, not what you wanted it to do.)

Piping and Redirect

>> : redirects and concatenates standard output (screen output) to the end of a specific file

```
%ls | head -n2 >! directory.list
%ls | tail -n2 >>directory.list
%more directory.list
1002
1019
Tmp
tmp.file
```

Piping and Redirect

< : redirects input from Standard input to the file on right of the less-than sign to be used as input to command on the left

```
% head –n1 < suma1.hrdpicks
1  51995    31410273254     30870    958490
```

Copying files & directories

cp: copy files

cp -r: copy directory and all files & subdirectories within it

Copying files & directories

```
% cp file1 ESCI7205/homework/HW1
```

Makes a copy with a new name - "HW1"

```
% cp file1 ESCI7205/homework/.
```

Makes a copy with same name (file1) in the new directory - specified by the dot "." (period) to save typing.

Moving files & directories

mv: move files or directories

```
% mv file1 file2 ESCI7205/HW/.
```

Moves file1 and file2 to new directory (relative) ESCI7205/HW with same names.

Move differs from copy in that it erases the original file, you only have 1 copy of it when done (what move does actually depends on whether or not the destination is on the same file system – if so, it just changes the directory entry, it does not actually do anything to the file itself).

Moving files & directories

mv: move files or directories

- `% mv file1 ESCI7205/HW/HW1`
- `% mv file2 ESCI7205/HW/HW2`

If you want to change the names when you move them, you have to do them one at a time

Renaming files & directories

Uses a side-effect of move!!!

```
% mv file1 HW1
% mv file2 HW2
```

There is NO RENAME command. One uses the side-effect of move!!!!

(We keep seeing this kind of logic in Unix.)

Linking files & directories

ln -s: creates a symbolic link between two files.

This makes the file show up in the new directory (the target) listing, but the file really only exists in the original place.

```
% ln –s f1 ESCI7205/HW/.
```

Linking files & directories

Doing an ls command in the directory ESCI7205/HW produces the following

```
% ls
f1      f2

% ls -1
-rw-r--r-- 1 smalley  smalley  10 Sep 20:26 f1
lrw-r-xr-x 1 smalley  smalley   2 Sep 20:42 f2->f2
```

The leading "l" (lower case L) in the long ls output says it is a link.

Linking files & directories

This allows us to "have" the file in more than one place.

We can therefore access it locally from the directory where it is a symbolic link.