

# Overview of the UNIX File System

Navigating and Viewing Directories



# The UNIX file system

- The most distinguishing characteristic of the UNIX file system is the nature of its files.

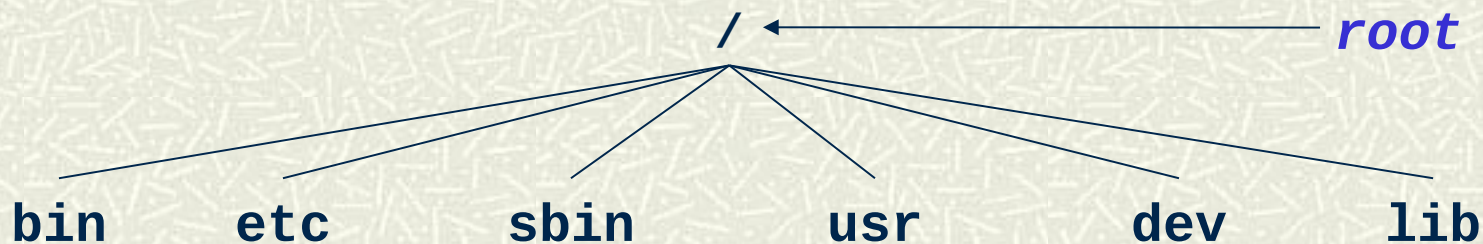
✿ *In UNIX, everything is a file: **directories** (folders), **devices** (like keyboards, monitors, and printers), **documents**, **programs**, **network connections**, and even **random access memory**.*

- By making everything a file, UNIX's designers simplified the operating system, making it easy to extend and maintain.



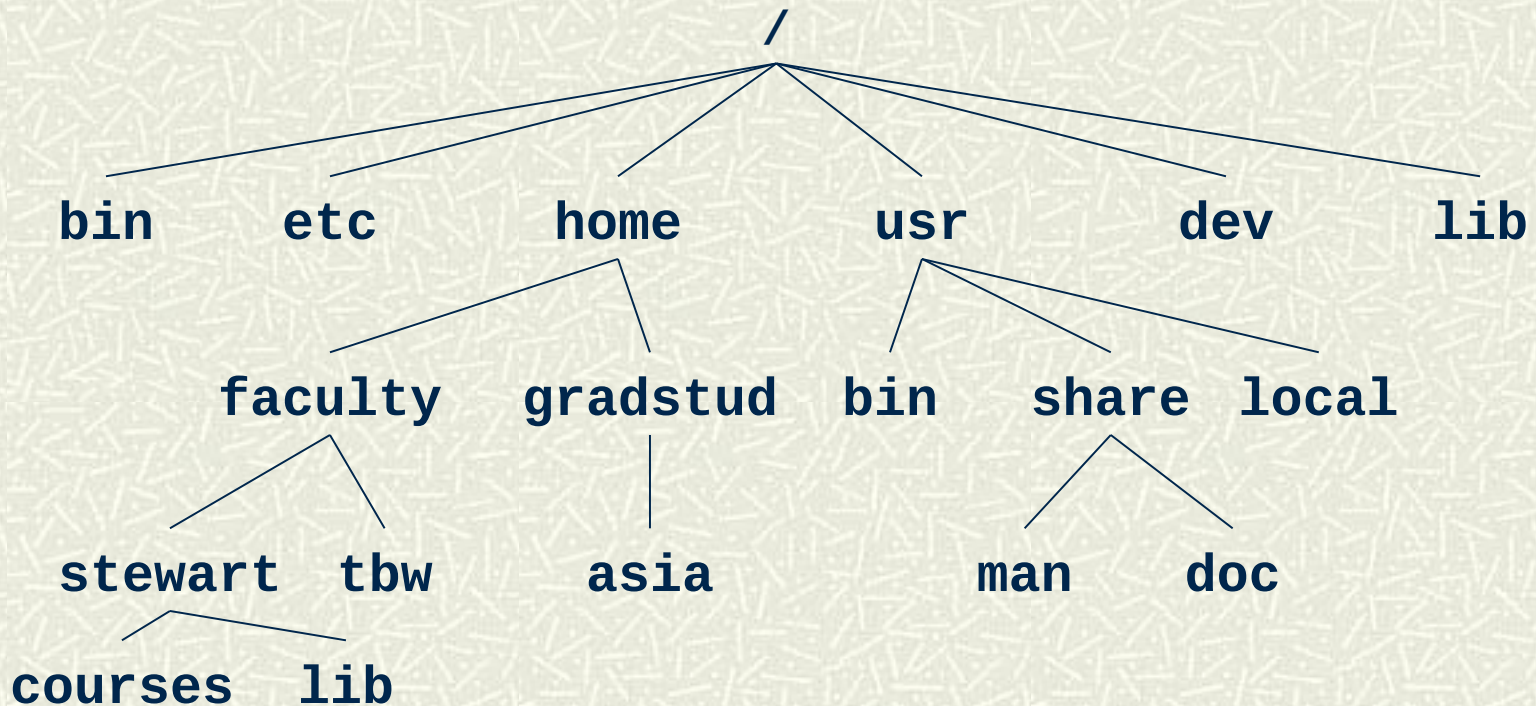
# The UNIX file system hierarchy

- # The UNIX file system is organized into a hierarchy whose top level is called the **root** ("/") directory.
- # The root directory always has certain specific **sub-directories**, which in turn have specific sub-directories of their own. Below are some of the directories present in every UNIX system:



# The UNIX file system hierarchy

- Every directory is either empty, or it has other directories or non-directory files. This is a fragment of a fictitious file system:



# Special directories

- The top level sub-directories keep UNIX organized. Some contain executable programs, some contain administrative files, some contain special *library* files, and others contain *device drivers*. Some contain important data.
- Every UNIX system uses these directories in the same way, no matter what "flavor" of UNIX it is.
- Ordinary users cannot modify any of the files in these special directories.
- Following are descriptions of a few of these directories.



# The `/bin` subdirectory

- # Originally UNIX was small enough that all programs could be placed into a single directory, `/bin`.
- # Programs are *executable* files, consisting not of words and letters, but sequences of *binary digits* (i.e., *bits*), 0 and 1. They are called *executables*, because they are *executed* (run) by the computer.
- # Executables are also called *binaries*.
- # `bin` is short for binaries. In UNIX all words were abbreviated, to make it compact and faster.



# The `/etc` subdirectory

- # The `/etc` directory was created as a place to store configuration and administrative files. The type of stuff you'd find in a Windows registry or in the Mac's System folder is found in `/etc`.
- # So much is stored in `/etc` these days that it has many subdirectories of its own.



# The `/dev` subdirectory

- # The `/dev` directory is a marvel to behold. It embodies the ingenuity of UNIX's creators.
- # Every file in `/dev` represents some physical or logical device. Each actually contains a device driver; i.e., an executable program. The interesting part is that you never "run" these files. Instead you read from them or write to them as if they were ordinary text files. We will visit this directory in future lessons.





# Other important top-level directories

<b>/usr</b>	usually contains executables
<b>/var</b>	more administrative files
<b>/lib</b>	library files
<b>/tmp</b>	temporary work space for programs
<b>/boot</b>	start up files
<b>/home</b>	usually the top level of user home directories
<b>/proc</b>	images of running programs



# Identifying files by pathnames

- # In UNIX, files are referred to by *pathnames*.
- # A *pathname* specifies the location of the file in the file system by its position in the hierarchy.
- # In UNIX, a forward slash "/" separates successive directory and file names in the pathname. This is the pathname from the root of the file system to the directory named *courses* in the directory named *stewart* in the directory named *faculty* in the directory named *home*:

**/home/faculty/stewart/courses**



# Absolute pathnames

- ⌘ There are two kinds of pathnames: ***absolute pathnames***, and ***relative pathnames***.

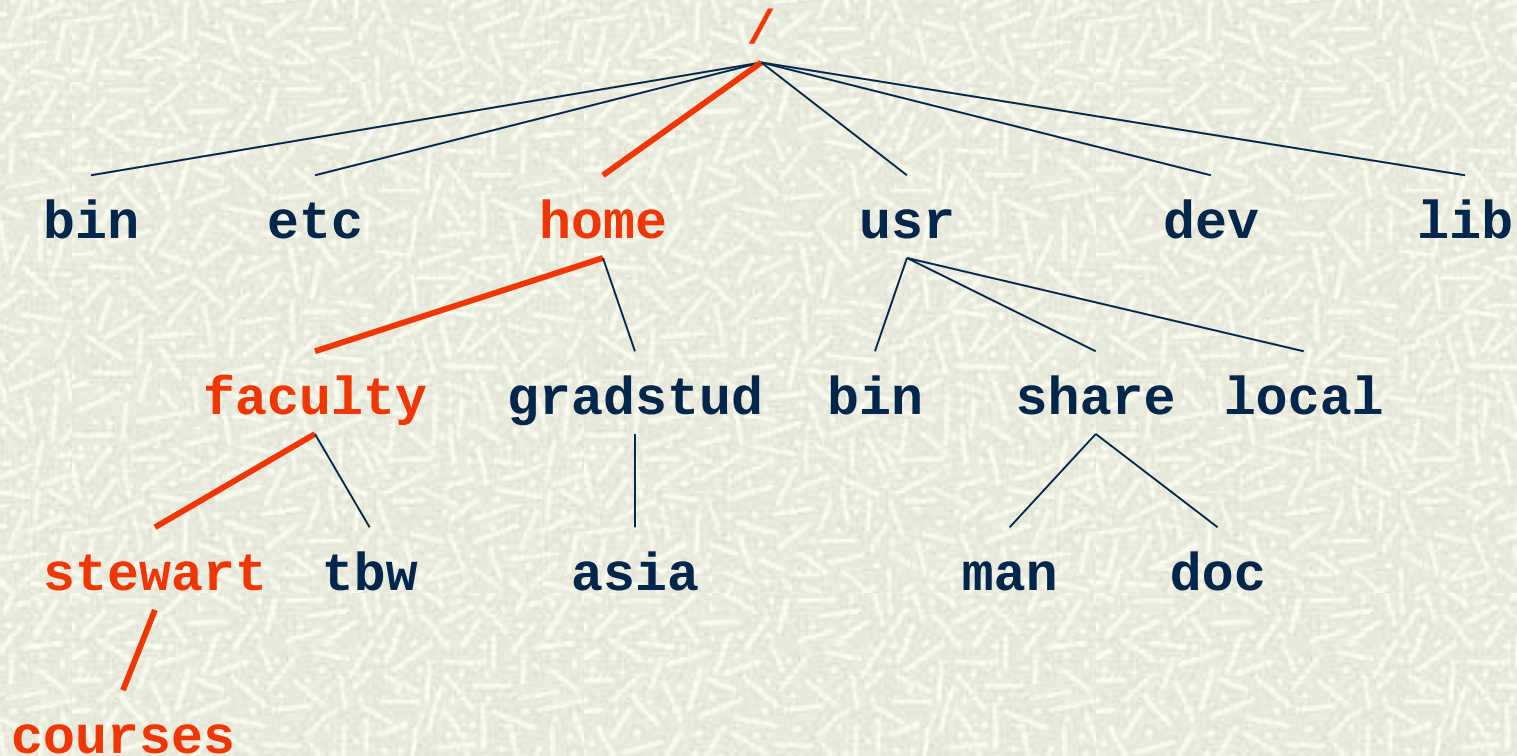


*The absolute pathname is the unique name for the file that starts at the root of the file system.*

- ⌘ ***It always starts with a forward slash.*** The next slide illustrates.



# Example of absolute pathname



The absolute pathname of the **courses** directory is **/home/faculty/stewart/courses**



# Your current working directory

- Whenever you are logged into a UNIX system, you have a unique, special directory called your **current** or **present working directory** (**PWD** for short).
- The **present working directory** is **where you "are"** in the file system at the moment, i.e., the directory that you feel like you are currently "in".
- UNIX keeps track of your **PWD** as you move about the file system. (We'll see how soon.)



# Your current working directory (2)

- # Many commands operate on the *PWD* if you do not give them an argument. We say that the *PWD* is their **default argument**. (Defaults are fall-backs – what happens when you don't do something.)
- # For example, when you enter the "**ls**" command (list files) without an argument, it displays the contents of your *PWD*.
- # The dot "**.**" is the name of the *PWD*:  
**ls .**  
and  
**ls**  
both display the files in the *PWD*.



# Relative pathnames

- A *relative pathname* is a pathname to a file relative to the *PWD*. *It never starts with a slash!*
  - When I login, my *PWD* is my home directory, **/home/faculty/stewart**. The relative pathname of **courses** is just **courses** or **./courses**.
  - If I change my *PWD* (with the **cd** command) to **/home/faculty**, the relative pathname of **courses** is **stewart/courses** (or **./stewart/courses**.)
  - The pathname of **courses** relative to **/home** is **faculty/stewart/courses**.



# The home directory

- # When you first login, your present working directory is set to your *home directory*.
- # Your *home directory* is a directory created for you by the system administrator. It is the top level of the part of the file system that belongs to you. You can create files and directories within your home directory, but usually nowhere else.

*Usually your home directory's name is the same as your username.*





# The home directory's names

- # The *home directory* can be referred to by the tilde `~` (located above the backquote ``` on the keyboard). For example,

```
ls ~
```

means list the files in your home directory.

- # Every user's home directory can be referenced using the form, `~username`, where *username* is the person's login name, as in

```
cd ~sweiss
```

which is the same as

```
cd ~
```



# What is a directory, anyway?

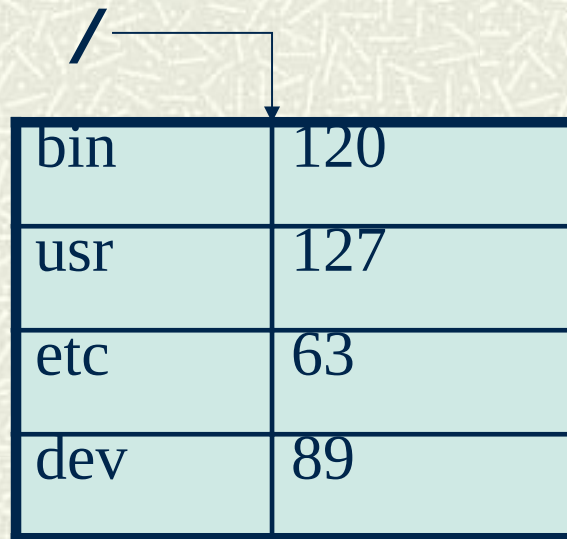
- # How do directories "contain" files?
- # A directory is a file with a specific structure: it is a table with as many rows as the number of files it contains. *Each row represents a file* and is called a *directory entry*.
- # The entry has two parts: *the name of the file* and *an index number that UNIX uses to find the file's contents on the disk*:

name of file	index number UNIX uses to find file
--------------	-------------------------------------



# Example

# Part of the root directory might look like this:



A diagram illustrating a portion of the root directory structure. A root directory symbol (/) is shown at the top left, with a line extending from it and an arrow pointing to the top-left corner of a table. The table has four rows and two columns. The first column contains the directory names: bin, usr, etc, and dev. The second column contains their corresponding index numbers: 120, 127, 63, and 89.

bin	120
usr	127
etc	63
dev	89

Notice that each name has a unique index number. The numbers above are fictitious.



# Displaying the files in a directory

- # In UNIX the **ls** command displays the files within a directory (like the **dir** command does in DOS.)
- # For example, if a directory named **stuff** looked like this:

pictures	120
letters	127
private	63
school	89

typing **ls stuff** would display  
**letters pictures private school**



# Hidden files in UNIX

- A file is called a *hidden file* in UNIX if its name begins with a dot "." because the **ls** command by default does not display files whose names start with ".".
- You can request **ls** to display them by giving it the "**-a**" option.
- In the next slide the first use of **ls** display files whose names do not start with a dot. Then with **-a** it displays the ones that start with a dot.



# How **ls** "hides" Files

```
$ ls
```

```
courses          lib              research         temp  
$
```

```
$ ls -a
```

```
. . . .bash_history  .bash_profile  
.bashrc  courses  .dir_colors  lib  
research  
$
```



# Why hide files?

- # In Windows, user-specific data and customizations are kept in a central place, the *Documents and Settings* folder, and in the registry. All applications create files there.
- # In UNIX, all user-specific data are kept in files in the user's home directory. These files are hidden because usually the user does not need to see them.
- # Every shell creates a hidden file called something like **.cshrc** or **.bashrc** or **.profile** that stores your customizations of the shell.



# Configuration files

- # Many applications create hidden directories in your home directory, where they store many files related to how they are configured for you.
- # Mozilla does this, for example. You will see directory named **.mozilla** in your home directory if **firefox** has been installed properly. **SSH** installs **.ssh**. In general, the files in these directories configure the application to work the way you like (sometimes.)





# Two entries in every directory

- # Did you notice that the example directory had two files named "." and ".."? (pronounced *dot* and *dot-dot*)
- # Every directory has these two entries, even if it has no other files. Each of them is a directory.
- # Dot is a shorthand name for the directory in which it occurs. It means "the current directory."



# The `.` directory

- # If my *PWD* is `/home/faculty/stewart` and I type  
`ls ./courses`  
I will see the same files as if I typed  
`ls courses`
- # Wherever I am in the file system, "`.`" means "here".



# The `..` directory

- # Dot-dot is a shorthand name for the *parent* directory. The *parent* directory is the one in which the directory is contained. It is the one that is one level up in the file hierarchy.
- # If my *PWD* is currently `/home/cs132/data`, then
  - `../projects/project1`  
is the directory `/home/projects/project1` and
  - `../../usr/share`  
is the directory `/usr/share`.



# More about `..` entries

- # You can use multiple `..` references in a relative pathname. If my *PWD* is `/usr/share/man/man1`, then
  - `../man2`  
is `/usr/share/man/man2`  
since `..` is `/usr/share/man`, and
  - `../../../../local`  
is `/usr/local`  
since `../..` is `/usr/share` and `../../../../..` is `/usr`



## Useful related commands: **cd**

- # To change your *PWD*, use the **cd** command.

```
$ cd ..
```

changes the *PWD* to the parent directory

```
$ cd /
```

makes the *PWD* the root directory.

- # The **cd** command with no argument

```
$ cd
```

changes your *PWD* to be your home directory



# Useful related commands: **pwd**

- # To display the absolute pathname of your *PWD*, use the **pwd** command.

```
$ pwd  
/home/faculty/stewart
```

Notice that **pwd** is a command but *PWD* is not. With **pwd** you can not get lost in the file system – you always know where you are.



# Creating directories: **mkdir**

- ✦ UNIX provides a single command to create a new directory: **mkdir** pronounced "make dir" which has a single argument:  
**mkdir my\_newdir**  
will create a new directory named **my\_newdir** in the *PWD*.
- ✦ Directory names can have almost any character in them, including spaces and newlines, but it is best to avoid using any characters other than letters, digits, and punctuation marks.



# More about `mkdir`

- # If you type a name but the directory already exists, `mkdir` will not replace it. It will warn you instead.
- # `mkdir` can be given any pathname for an argument, either relative or absolute, and it will create the directory in the place you specify in the pathname. For example,  
`mkdir /tmp/mydir`  
will create the directory `mydir` in the `/tmp` directory.
- # We will learn more about the `mkdir` command in Lesson 7.





# Output redirection

- # All commands in UNIX display their output, i.e., their results, on your terminal. You do not have to do anything special for this to happen. For example, when you type **ls**, it displays the list of files in your terminal window.
- # UNIX provides a way to **redirect that output to a file** instead. (In fact, so does DOS, and the method is the same most likely because DOS borrowed this from UNIX when it was designed.)



# The output redirection operator

- # To force a command's *standard output* (the output ordinarily written to the screen) to be written into a file named **gotcha**, in the PWD you append the following to the end of the command:

```
> ./gotcha
```

- # For example,

```
$ last > ./gotcha
```

puts the output of **last** into a file named **gotcha** in the PWD. (Type **last** and see what it outputs.) In general, the form is

```
$ command > filename
```



# Creating files Using >

- It should be obvious that this is a convenient way to create small files, especially with the **echo** command. Assuming that the file **file1** does not exist in the PWD,

```
$ echo 'This is a test.' > file1
```

puts the text **'This is a test.'** in **file1** (without the quotes):

```
$ cat file1
This is a test.
$
```



# Your environment

- # UNIX keeps track of your actions by saving various items of information about you in specially named variables called *environment variables*.
- # A *variable* is just like a mathematical variable – it is a symbol that can have a single value at any time. In UNIX, variables are named using ordinary words; environment variables are usually in all UPPERCASE.

*The collection of environment variables and their values is called your **environment**.*



# Viewing your environment

- There are several commands to display the current "state" of your environment. Without arguments, the safest to use are:

**env**

**printenv**



# Using **printenv**

- # The output of **printenv** may be many lines long and contain many variables whose purpose is unclear. Ignore them for now. Here is a fragment of what it could look like:

```
USER=sweiss  
LOGNAME=sweiss  
HOME=/data/biocs/b/cshome/sweiss  
PWD=/data/biocs/b/cshome/sweiss  
SHELL=/bin/bash  
PATH=/bin:/usr/local/bin:/usr/bin:
```



# Basic environment variables

<b>USER</b>	<b>User's username</b>
<b>LOGNAME</b>	<b>User's username also</b>
<b>HOME</b>	<b>User's login (home) directory</b>
<b>PWD</b>	<b>The present working directory</b>
<b>SHELL</b>	<b>The file name of the user's shell.</b>
<b>PATH</b>	<b>A sequence of directories searched by shells and some other programs when you enter a command (and sometimes when you do other things too.)</b>



# Other ways to view variables

- # When you put a '\$' in front of any variable in a command, it replaces the variable by its value. (\$ is called a *substitution operator*.)
- # To see how this works, let me introduce a new command: **echo**.
- # The **echo** command displays its argument on standard output:

```
$ echo hello
hello
$ echo "hello to you."
hello to you.
$
```





# Echoing environment variables

- If you give **echo** an argument with the **\$** operator, it does the substitution first, and then it echoes:  
**\$ echo \$HOME**  
**/data/biocs/b/cshome/sweiss**
- So **echo** is another way to view the value of an environment variable.



# Things to try

- # Use the **cd** and **pwd** commands to explore the file hierarchy.
- # Use **printenv** to see what the values of your environment variables are.
- # Each time you change directory with **cd**, type **printenv PWD**
- # Read the man page for the **ls** command and see what the **-l, i, t**, and **m** options do.

