Displaying Information about Files

Commands for Viewing File Properties

Copyright 2006 Stewart Weiss

Viewing file properties

- A file has two components:
  - Contents
  - Properties
- Contents are the data contained in the file. Properties are characteristics of the file such as its size, ownership, security tags, and type.
- Here we concentrate on how to display the properties of files.

`ls` revisited

- The `ls` command lists the files in a given directory.

  (If `ls` is given a non-directory file as an argument, it just lists or displays information about that file.)

- `ls` has several flags that control what information is displayed, or that change the format or order in which the information is presented. You already know, for example, that `-a` lists hidden files. There are other useful flags. These slides cover `-F`, `-R`, `-d`, and `-l`.

Options for `ls`

`-F` appends a trailing character to the file name to indicate its type ("**`/`**" for directory, "**`*`**" for executable, or "`@`" for symbolic link.)

`-R` lists the entire hierarchy rooted at the given directory, ***recursively***. This means, informally, that if any file is a directory, its contents are also listed, and if any of its files is a directory, its contents are also listed, and so on. A specific order is used; try it and see what the order is.

`-d` displays information about the directory itself instead of its contents.

All options can be combined, in various ways. Examples:

```
ls -Fd
ls -F -d -R
```

Long listings

- The most important and most frequently used flag is `-l`, which generates a ***long listing*** of the directory. For example:

```
$ ls -l homeworks
-rw-------   1 sweiss faculty   3170 Apr 21 17:29 hwk4
```

- For each file, 7 fields are displayed; each of these is a specific property. They are defined in the next slide.

Interpreting a long listing

```
-rw-------  1  sweiss faculty  3170 Apr 21 17:29 hwk4
```

Some of these fields are self-explanatory: ***user*** is the username of the owner; ***filesize*** is the number of ***bytes*** in the file; ***time last modified*** is the time that the file was last changed in some way, not just read or opened, but changed. I'll discuss ***links*** and ***groups*** another time. First a word about ***filesize***.

Measuring storage capacity

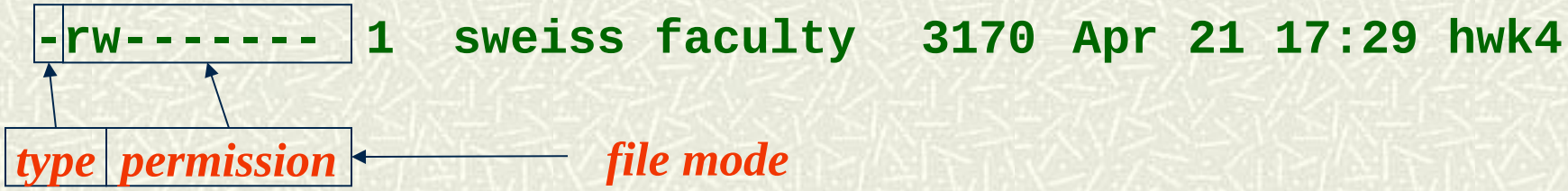
Just in case you did not know, a ***bit*** is the smallest unit of information content. It is either 1 or 0 (on or off, true or false, etc). Building on bits:

1 ***byte***	= 8 bits	
1 ***kilobyte*** (KB)	= 1024 bytes	
1 ***megabyte*** (MB)	= 1024 kilobytes	~ 10^6 bytes
1 ***gigabyte*** (GB)	= 1024 megabytes	~ 10^9 bytes
1 ***terabyte*** (TB)	= 1024 gigabytes	~ 10^{12} bytes

File size

- Even though ***a bit is the smallest unit of information***, files are measured in bytes because ***bytes are the smallest unit of storage***.
- A single character such as a letter or digit or punctuation mark currently requires a byte of storage on older computer systems (and will someday require two bytes on all systems.)

File mode



- The ***file mode*** is a string of 10 letters that describes what kind of file it is, and who has permission to do what to the file. The first letter identifies the ***type*** of file: "**d**" for directory, "**-**" for regular file, "**c**" or "**b**" for device files, "**l**" for symbolic link, and so on. (There are some other types not worth mentioning now.)

Permissions

- The file mode lets you see the permissions on a file. In UNIX, the universe is divided into three sets of users:

  user the owner of the file

  group the UNIX group that "owns" the file

  others everyone not in either of the first two sets

- The permission string has 3 ***bits*** for each set, a read-bit, a write bit, and an execute-bit:

`r w x` `r w x` `r w x`

user **group** **others**

Types of file access

- A user can access a file to do one of three things:
  - ***read*** it (view its contents)
  - ***write*** it (change or modify it)
  - ***execute*** it (run it if it is a program)
- If the user has permission to read, the '`r`' bit is set; if to write, the '`w`' bit is set, and if to execute, the '`x`' bit is set. If a bit is not set, a dash '`-`' appears instead of `r`, `w` or `x` in the permission string.

Example

- Given file mode

  `-rwxrw-r--`

  break it up into the type and 3 groups of user bits:

  `- rwx rw- r--`

  The leading dash '`-`' indicates that this is a regular file. The owner (user) has read, write, and execute permission (`rwx`); the group has read and write permission but not execute permission (`rw-`), and everyone else can only read the file (`r--`).

Creating empty files

- The **`touch`** command was designed to update ***the time of last modification*** of a file; if you **`touch`** a file, it is like you have made it current. (*You made it **dirty** by touching it, and so it is marked dirty, as if you wrote in it.)* (The usefulness of this will be apparent later.)
- If you **`touch`** a file that does not exist, UNIX creates it.

```
$ touch museumpiece
```

creates **`museumpiece`** if it did not exist, or it makes it dirty if it did.

The `file` Command

- The **`file`** command is a useful command for determining the type of a file. It tries to use information in the file properties or the file contents to classify the file.
- Examples:

```
$ file .bashrc
.bashrc: ASCII text
$ file /etc
/etc: directory
$ file myhwk_sweiss
myhwk_sweiss: Zip archive data
```

Checking Disk Capacity and Free Space

- The **`df`** command, intended to report on the status of ***mounted file systems***, provides an easy way to check the amount of free space and the capacity of all the storage available on the UNIX system.
- A ***file system*** is like an apartment building. It is an organized collection of units available for occupancy (by data), with a means of identifying them, finding them, determining which are occupied and which are not, and retrieving or placing the occupants.
- A file system is ***mounted*** if it is connected to the single file hierarchy rooted at **`/`**.

The `df` Command

- When you type the **`df`** command, it outputs lines containing 6 fields, including
  - the name of the (mounted) file system,
  - its total capacity in kilobytes,
  - the kilobytes currently used by this file system,
  - the kilobytes available for use in this file system,
  - the percent of this file system in use,
  - the pathname of its ***mount point***
- ***The mount point is the directory on which the file system is attached.***

The `df` Command's Output

- This is an example of the output of the **`df`** command on a system I once used:

```
$ df
Filesystem     1K-blocks    Used   Available Use% Mounted on
/dev/hda2       15116868  6394252   7954712  45% /
/dev/hda1         101086     8601     87266   9% /boot
/dev/hda5        6736100   734924   5658996  12% /data/eniac
mars:/scratch   17413280 11185280   6053856  65% /scratch
moon:/data/aa  210023264 97939072 109983936  48% /data/moon/aa
$
```

From this, we see that the disk **`hda5`** has about 5.7 million Kbytes free, which is about 5.8 Gigabytes.

About mount points

- On Windows machines, when you have several disks, they have separate file systems that have separate roots, with names like `C:`, `D:`, and `E:`, called volume or drive letters.
- In UNIX, all disks and disk partitions are attached to the file hierarchy at directories called ***mount points***. There are no volume letters, no drive letters.

Illustration

- Suppose that the root file system looks partly like this:

Illustration (continued)

- Suppose a CD has a file system on it that looks partly like this:

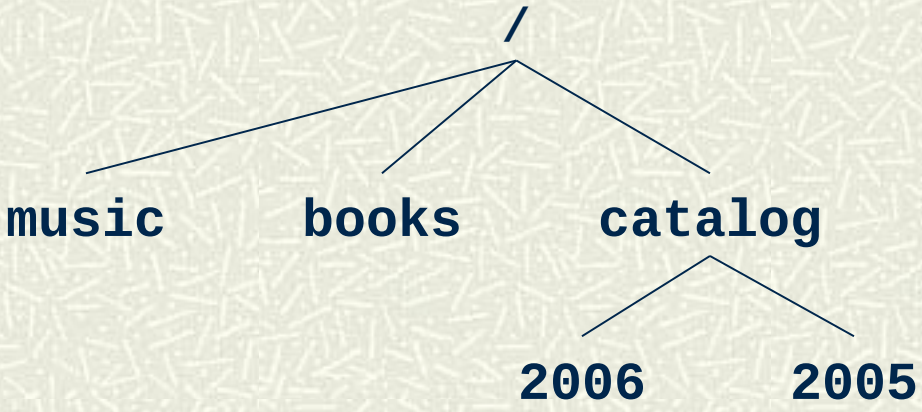
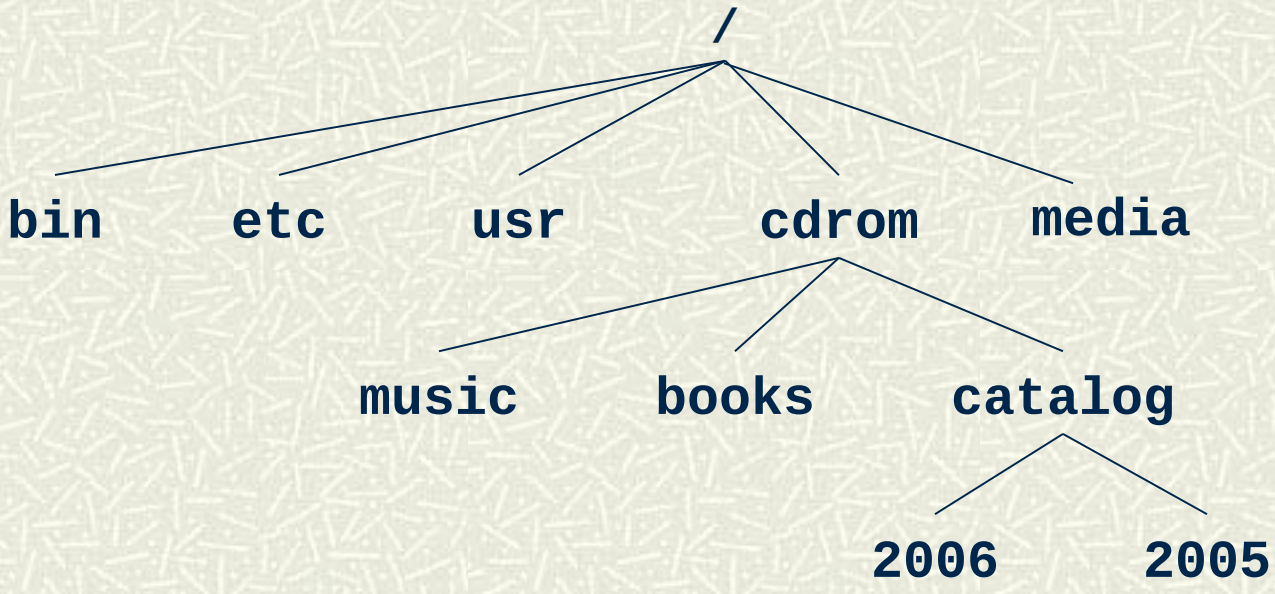


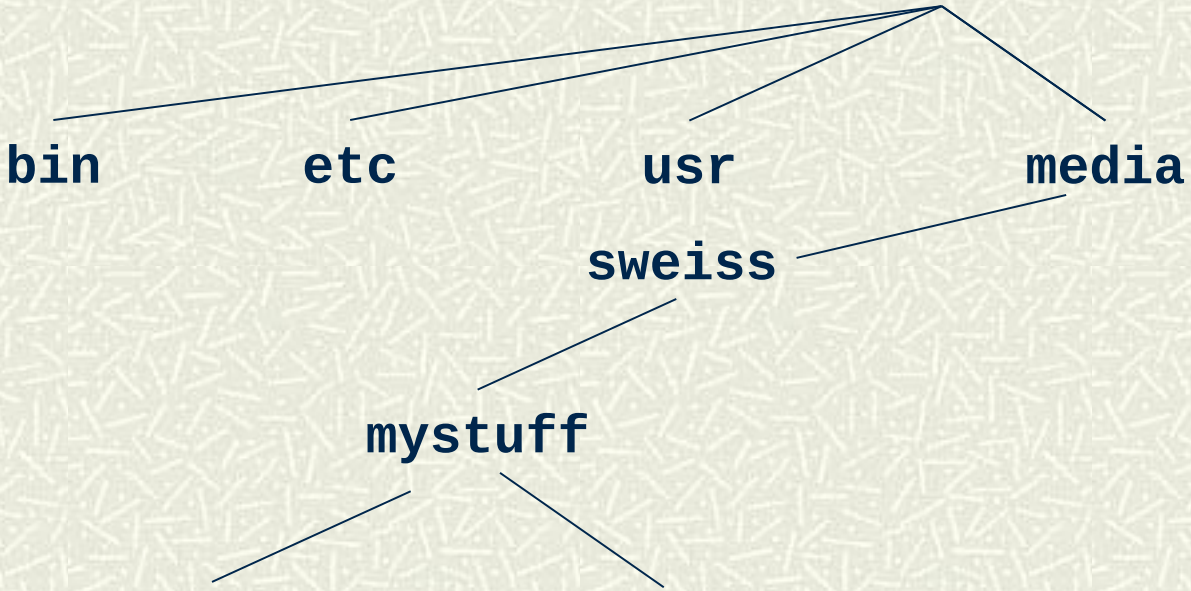
Illustration (continued)

- When you insert this CD into the CDROM drive, if the driver is configured to ***automount***, then the root of the CD file system will be attached where the **`cdrom`** directory is in the tree, and renamed **`cdrom`**:



More About Mounting

- Computers sometimes automount CD's, flash drives, and other external storage devices into the **`media/`** directory. In Ubuntu, it is **`media/username`** (e.g. **`/media/sweiss`**.) For a flash drive named **`mystuff`**, the mounted system would be:



Things to try

- How much free space is available in **`/data/biocs/b`**?
- Is your home directory physically located on **`eniac`**?
- There are a number of files in the directory

  `/data/biocs/b/student.accounts/cs132/data`
- Some of these are protein data files (PDB files). Find them and determine how many bytes they contain. How many atoms do they contain? How can you figure this out (remember **`grep`**?) Read about the **`wc`** command and think about using a pipe.