Linux Basics 1: Filesystem Intro

To work effectively on a Linux system, it is necessary to understand a number of the basic aspects of Linux:

- **The Filesystem (directories and files)**
  - **the logical filesystem tree (directory structure)**
  - **pathnames**
  - **mounting and mount points**
  - **filenames**
  - **key commands**
  - standard directories
  - regular files vs. special files
  - physical filesystems
  - additional commands
- Users, Groups, and File Permissions (security mechanisms)
- Processes (running programs and their attributes)

Logical Filesystem

The **logical filesystem** is the set of *directories* (*"folders"*) and *files* that are made available by an operating system.

Alternative terms for the logical filesystem include:
filesystem hierarchy and **directory structure**.

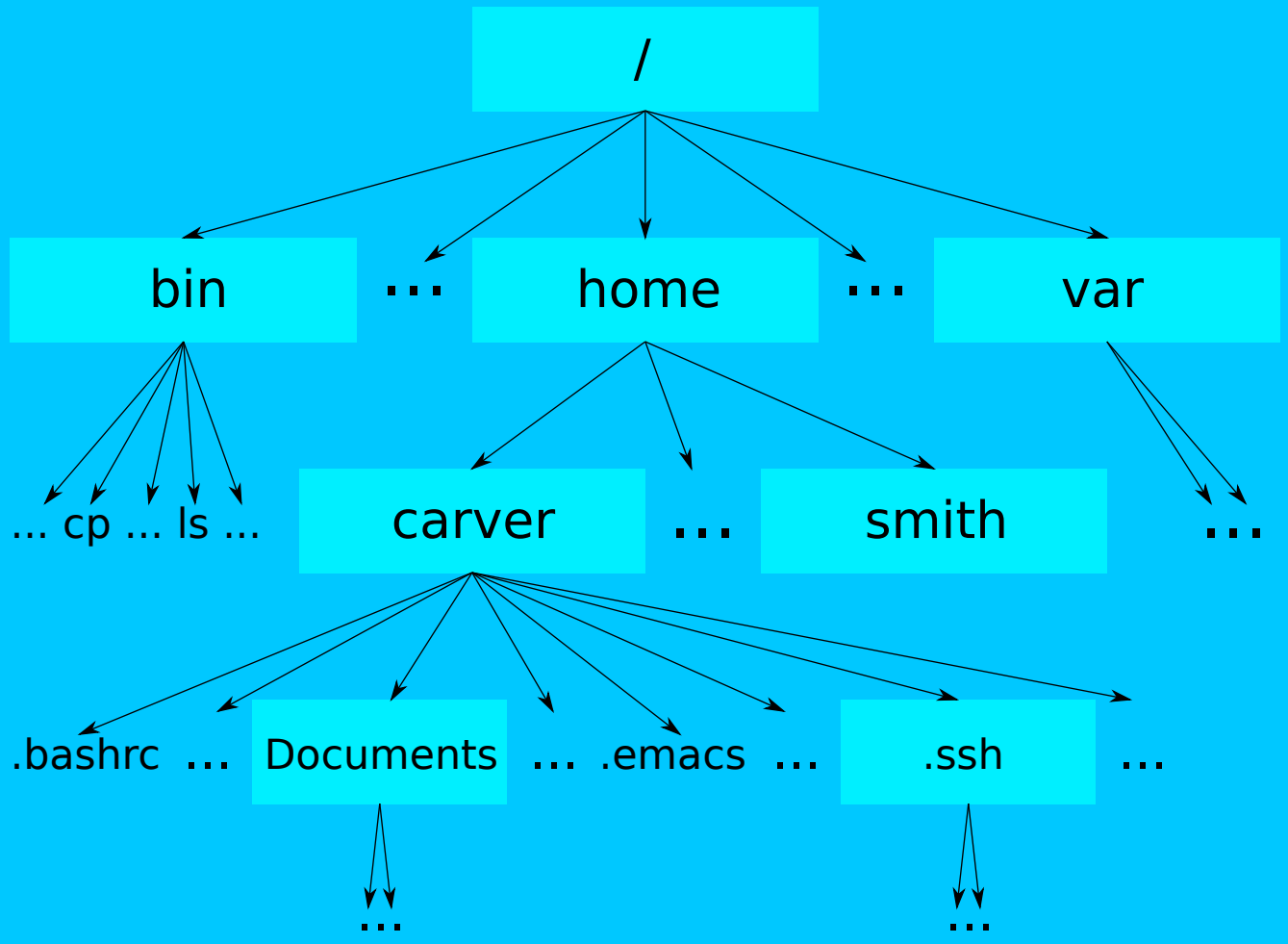
In Linux, the directories and files within the filesystem form a *single* **tree** structure (i.e., a **hierarchical** structure).

In CS, we draw trees with their **roots** at the top.

Thus, the *top-level directory* of the logical filesystem tree is called "**the root directory**."

It is denoted as: / (a single *forward slash*)

Logical Filesystem (contd.)



Pathnames

A particular file or directory within the filesystem structure can be denoted by a **pathname** (**path**) specification.

Note: files and directories are specified in exactly the same way, so one *cannot* tell from a pathname which is being specified.

There are *two types* of pathname specifications:

- **absolute** pathnames
- **relative** pathnames

Pathnames (contd.)

An **absolute pathname/path**:

- completely/absolutely specifies a directory/file
- begins with / (denoting *the root directory*)
- includes all subdirectories on path to directory/file
- these path subdirectories are separated by /'s
- e.g., `/home/carver/Documents/test.text`
- gives the *sequence* of directories to traverse in the filesystem tree to reach the directory/file:
  / $\rightarrow$ `home` $\rightarrow$ `carver` $\rightarrow$ `Documents` $\rightarrow$ `test.text`
- note that containing subdirectory would be:
  `/home/carver/Documents`

Pathnames (contd.)

To understand **relative pathnames**, one must know that every **process** (including each shell command line) has an associated directory known as the **current working directory** (**CWD**) or just **working directory**.

This is the default directory for opening/creating files if only a file/directory *name* is specified (e.g., `test.text`).

The CWD can be explicitly denoted as: **.** ("**dot**").

So a file can be explicitly denoted as being in the CWD as:
`./test.text`

Pathnames (contd.)

A **relative path/pathname**:

- specifies a file/directory *relative to the CWD*
- does *not* start with /
- instead starts with a directory/file in the CWD

Example:

- CWD is: `/home/carver/Documents`
- relative path: `temp/talk.pdf`
- equivalent to absolute path:
  `/home/carver/Documents/temp/talk.pdf`
- (“`/home/carver/Documents`” + “/” + “`temp/talk.pdf`”)

Pathnames (contd.)

The **parent directory** (next higher-level directory) of the CWD can be denoted as: **..** ("**dot-dot**" or "**double dot**").

The dot-dot notation can be used in relative path specifications to *ascend* from the CWD, and can be combined as "`../..`"

Example:

- CWD is: `/home/carver/Documents/temp`
- relative path: `../../talks/talk.pdf`
- equivalent to absolute path: `/home/carver/talks/talk.pdf`

Mounting and Mount Points

The single Linux filesystem tree can be constructed from *multiple* **partitions** on *multiple* **storage devices**.

In fact it is standard practice to spread the filesystem among multiple partitions.

One storage partition must be designated as the **root partition**, with its files and directories appearing under /.

Every other **storage device** must be **mounted** at some *directory* within the single filesystem tree for its files to be accessible.

The directory at which a device/partition is mounted is called its **mount point**.

Mounting and Mount Points (contd.)

For example, user *home directories* may be stored in a separate disk partition (from /), which is normally mounted at `/home`.

E.g., user carver's home directory would be the top-level directory `carver` in the home directories partition.

Once that partition is mounted at `/home`, user carver's home directory would have path `/home/carver`.

Removable devices like CD-ROM drives will have their contents mounted when media are inserted.

E.g., a CD-ROM may be mounted as `/mnt/cdrom`, so a file on the CD-ROM would have a path like: `/mnt/cdrom/fav1.mp3`.

Windows Logical Filesystem

Windows uses a fairly similar scheme for its logical filesystem, but with some key differences:

- each mounted device gets its own separate filesystem tree, identified by a so called "**drive letter**"
- directory separators are "backwards" (backslashes)
- e.g., `C:\WINDOWS\Fonts\Vera.ttf`

Filenames

Linux **filename** scheme:

- much more general than *8.3 scheme* from Windows/DOS
- same naming scheme applies to files, directories, devices, etc.
- cannot tell file's *type* from its filename
  (e.g., `linux.save` could be text file, MP3 file, directory, etc.)
- "**extensions**" do not play a special role with the OS
  (a dot is a character that can appear anywhere in a filename)
- some *applications* do expect certain extensions however
  (e.g., GCC expects C source files to end in "`.c`")

Filenames (contd.)

Linux filename scheme (contd.):

- **extensions** often used very differently than in Windows:
  - files need *not have any extension*
    (e.g., `ls`)
  - extensions often *not 3-letters*
    (e.g., `.html` vs. `.htm`)
  - virtually never use `.exe` for executables
    (files are executable if their **permissions** say so)
  - can have *multiple* "extensions"
    (e.g., `test.text.save.120115`)

- filenames that *begin with a dot* are considered "**hidden files**" and may not be listed by default by some programs
  (e.g., `/home/carver/.bashrc`)

Filenames (contd.)

Linux filename scheme (contd.):

- most hidden files/directories are configuration-related (e.g., `/home/carver/.bashrc` and `/home/carver/.ssh`)
- filenames are *case sensitive*:
  - this includes commands, e.g., `ls` $\neq$ `LS` $\neq$ `Ls`
  - also applies to extensions, e.g., `foo.mp3` $\neq$ `foo.MP3`
- POSIX/SUS says to be **portable**, filenames can include only *alphanumerics* plus *dots*, *dashes*, and *underscores*.
- however, most Linux filesystems allow the use of many non-alphanumeric characters, such as spaces, `~`, `?`, `*`, etc.
- the forward slash (/) is *never allowed* in filenames, as there could then be ambiguity in interpreting pathnames

Filenames (contd.)

Linux filename scheme: (contd.)

- it is common to use *dashes* or *underscores* to separate "words" in filenames, since if spaces are used, the filenames have to be *quoted* on the command line:

  e.g., `linux-books` or `linux_books`
  vs.
  `"linux books"` or `'linux books'` or `linux\ books`

Filenames (contd.)

Example filenames:

- `test1.text`
- `Test1.text` (completely different file)
- `test1.text.save3`
- `ls` (command/executable)
- `.bashrc` (hidden file)
- `.ssh` (hidden directory)
- `a_long_name`
- `alternative-long-name`
- `a name with spaces . even in extension`

Key Filesystem Commands

cd – change working directory:

- `cd`
- `cd /home/carver/temp`
- `cd temp`
- `cd ..`

pwd – print working directory:

- `pwd`

mkdir – make a new directory:

- `mkdir cs306`
- `mkdir /backup/mp3s`

ls – list directory contents or file(s) info:

- `ls -lA`
- `ls -l *.c`

Key Filesystem Commands (contd.)

mv – move and/or rename file(s)/directory:

- `mv test.c /home/carver/cs306`
- `mv test.c ~/cs306`
- `mv test.c lab1.c`

cp – copy file(s)/directory:

- `cp cs306/*.c ~/cs306-backup`
- `cp -pr cs306 /backup`

rm – remove/delete files or directories:

- `rm *.old`
- `rm -r test-dir`