Systems Programming Laboratory, Spring 2023

Basic Unix commands

Abhijit Das
Bivas Mitra

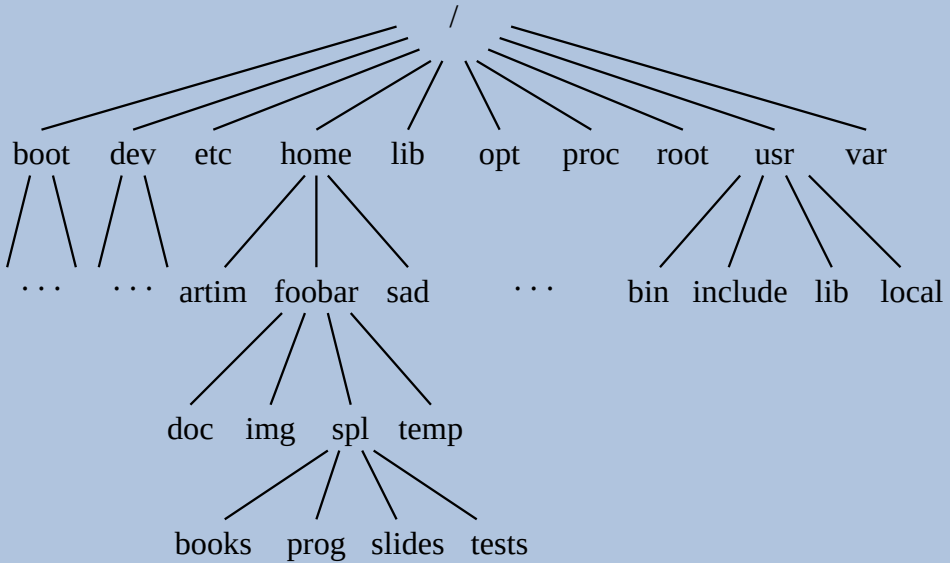
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

January 12, 2023

Why Unix?

- Unix was developed in 1970's by the AT&T Bell Labs.
- Good and transparent design philosophy.
- Multi-user multi-tasking OS since near the beginning.
- Proprietary implementations started being developed.
- Richard Stallman and Linus Torvalds developed the *free* and open-source GNU-Linux.
- Eventually, Linux has become popular and widespread in academia and industry.
- MacOS is another derivative of Unix. Proprietary. Hardware-dependent.
- Initially targeted to serve servers, Linux is now commonplace in personal computers.
- You need to use Linux quite often in your several courses.
- Currently, all Unix versions in use are Unix-like (Linux is no exception).

The Unix directory tree



Contents of a directory

- A directory consists of:
  - subdirectories
  - text files (C source, text documents, program outputs, scripts, …)
  - binary files (compiled executable files, images, printer files, …)
  - special files (sockets, symbolic links, …)
- Every directory contains two subdirectories: . (pointing to itself), .. (pointing to the parent directory).
- Files/directories with names starting with . are called hidden files/directories.
- When you log in, you enter your home directory (like **`/home/foobar`**).
- In Unix philosophy, everything is a file. Even directories are.

Viewing the contents of a directory

- The basic command is **`ls`**. The listing is sorted with respect to the content names.
- Some options
  - –l Long listing
  - –a Show the hidden files also
  - –R Recursively list the subdirectories, the subsubdirectories, and so on
  - –t The sorting is with respect to last modification times (newest first)
  - –r Reverse the sorting order
  - –d Do not expand the directory contents
- Example: **`ls -lart`** shows a long listing of all files (including the hidden ones) sorted in the reverse order of modification times (oldest first)
- You may supply one or more directory or file names after the options in order to see the the listing of that/those file(s) or director(ies).
- Example: **`ls -lR /`** makes a long listing of the entire directory tree (excluding the hidden files).

Directory and file names

Absolute names

You specify the exact path starting from the root **/**. Examples:

```
/usr/local/lib/
/usr/local/lib/libstaque.so
/home/foobar/spl/prog/assignments/A1/src/
/home/foobar/spl/prog/assignments/A1/src/Makefile
```

Relative names

- Relative to the current directory. Examples (assume that you are in **/home/foobar**):

  ```
  spl/prog/assignments/A2/myprog.c
  ./spl/prog/assignments/
  ../artim/SPL/tests/T1/questions.pdf
  ```

- Relative to the home directory. Examples:

  ```
  ~/spl/prog/assignments/A3/
  ~sad/SPL/doc/T1soln.pdf
  ```

Permissions

- Three types of users
  - The user who owns the file (u)
  - Other members of the same group as the owner (g)
  - All other users (o)
- Three types of permission
  - Read permission (r)
  - Write permission (w)
  - Execute permission (x)
- Straightforward meaning for files.
- For directories, the permissions mean:
  - Read permission: You can read the contents of the directory (by **`ls`**). With only read permission, you cannot access the files in the directory.
  - Write permission: You can create new files in the directory.
  - Execute permission: You can go to the directory, and open and/or execute files in the directory (provided you know the names). With only execute permission, you cannot see the directory content.

Examples of permissions

- User sad of group faculty is the owner of the directory **`/home/sad/spl/prog/libstaque`**
- You are foobar belonging to the group student. artim is a user in the group faculty.

```
$ ls -ld /home/sad/spl/prog/libstaque
drwxr-xr-x 4 sad faculty 4096 Jan 11 19:55 /home/sad/spl/prog/libstaque/
$ ls -l /home/sad/spl/prog/libstaque
-rwx------ 1 sad faculty 16744 Jan 11 20:04 a.out
-rw-rw-r-- 1 sad faculty   170 Dec 28 19:56 Makefile
-rw-r--r-- 1 sad faculty   357 Dec 20 17:36 Makefile.txt
drwxr-xr-- 2 sad faculty  4096 Dec 28 20:03 shared/
drwxr-x--x 2 sad faculty  4096 Dec 28 20:03 static/
$
```

- Only sad can execute a.out.
- You can only read Makefile and makefile.txt. artim can read and modify Makefile, but can only read Makefile.txt.
- You can **`ls /home/sad/spl/prog/libstaque/shared`** to see its content, but cannot access any file in that directory.
- You cannot see the directory listing of **`/home/sad/spl/prog/libstaque/static`**, but if you know a file name in that directory and have read permission for that file, you can view that file.

Changing permissions of files

- Only the owner (and the root) can change the permission of a file/directory.
- The command for that is **`chmod`**.
- Symbolic change: Add (+) or remove (–) a permission (r, w, x) for user (u), group (g), others (o) or all (a).

```
chmod g+x /home/sad/spl/prog/libstaque/a.out
chmod o-rwx /home/sad/spl/prog/libstaque/static
chmod a+w /home/sad/spl/prog/libstaque/shared
```

- Numeric change: Set the permission bits as a three-digit octal number.

```
chmod 755 /home/sad/spl/prog/libstaque/a.out
chmod 666 /home/sad/spl/prog/libstaque/Makefile.txt
chmod 700 ~sad/spl/prog/libstaque/shared ~sad/spl/prog/libstaque/static
```

Moving around in the directory tree

- Use **`cd <dirname>`** to go to the directory **`<dirname>`**. The name may be absolute or relative. You should have execute permission to go to the directory.
- **`cd`** without any argument lets you go to your home directory.
- **`mkdir <newdirname>`** lets you create a new directory **`<newdirname>`**. You should have write permission in the directory where this new directory is created.
- **`rmdir <dirname>`** lets you remove the directory **`<dirname>`** provided that
  - you have write permission in the parent of **`<dirname>`**, and
  - **`<dirname>`** is empty.
- Use **`rm -r <dirname>`** to remove the entire subtree rooted at **`<dirname>`** (provided that you have permission to do so).

File utilities

- **`cp <file1> <file2>`** copies **`<file1>`** to **`<file2>`**.
- **`mv <file1> <file2>`** moves (renames) **`<file1>`** to **`<file2>`**.
- **`mv <file> <dir>`** moves **`<file>`** to directory **`<dir>`**.
- You can copy or move more than one files, but then the last argument must be a directory.
- You can copy or move an entire subtree with **`cp -r`** or **`mv -r`**.
- You can delete a file (or multiple files) using **`rm <file1> <file2> ...`**.
- Use these commands with the option **`-i`** to see warning messages (like when something is overwritten).
- **`wc <file1> <file2> ...`** gives the individual counts of characters, words, and lines in the files, and the sums of these counts (if there are multiple files). Meaningful for text files only.

Viewing text files

- You can open a text file using an editor (in the read-only mode if you only have read permission).
- **`cat <textfile>`** prints the file content.
- **`head <textfile>`** prints the first few lines of **`<textfile>`**.
- **`tail <textfile>`** prints the last few lines of **`<textfile>`**.
- Use **`less`** (or **`more`**) for a page-by-page display of the file. Some **`less`** commands:

Up or down arrow	One line up or down
Space or f	One page down
b	One page up
d	Half page down
u	Half page up
g	Go to the first page
G	Go to the last page
/pattern	Search for a pattern
n	Go to the next match
N	Go to the previous match
q	Quit the viewer

Redirection and pipes

- Three file descriptors: stdin (for reading), stdout (for writing output), stderr (for writing error messages)
- **`command < file`** redirects the command's stdin to the given file.
- **`command > file`** redirects the command's stdout to the given file.
- **`command 2> file`** redirects the command's stderr to the given file.
- **`command > outfile 2> errfile`** redirects the command's stdout to outfile and stderr to errfile.
- Use **`>>`** if you want to append (**`>`** overwrites existing files).
- **`command1 <cmd1args> | command2 <cmd2args>`** short-circuits command1's stdout to command2's stdin.

```
ls -l | wc
cat myprog.txt | less
```

Locating commands

- Commands are searched in some default directories (like /bin, /usr/bin, /usr/local/bin).
- . (the current directory) may be absent in the default search path
- You can set the environment variable PATH for setting/updating the search path

```
export PATH="$PATH:newpath1:newpath2:newpath3:..."
export PATH="$PATH:."
```

- Paths are searched from beginning to end. Search stops as soon as the command is found.
- **which** tells you the command first found. **whereis** gives additional details. **whatis** gives a short description. **man** opens the detailed manual page.

```
$ which cat
/usr/bin/cat
$ whereis cat
cat: /usr/bin/cat /usr/share/man/man1/cat.1.gz
$ whatis cat
cat (1)              - concatenate files and print on the standard output
$ man cat
```

Users and system information

```
$ uname
Linux
$ uname -a
Linux FOOBAR-SERVER 5.11.0-44-generic #48 20.04.2-Ubuntu SMP Tue Dec 14 15:36:44 UTC 2021 x86_64 x86_64 x86_64
GNU/Linux
$ who
abhij     :0           2022-01-11 09:40 (:0)
abhij     pts/0        2022-01-11 17:35 (:0)
artim     pts/1        2022-01-11 17:27 (:0)
foobar    pts/2        2022-01-11 18:47 (:0)
$ w
 22:11:37 up 12:33,  4 users,  load average: 0.65, 0.55, 0.57
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
abhij    :0       :0               09:40   ?xdm?   1:40m  0.00s /usr/lib/gdm3/gdm-x-session --run-s
abhij    pts/0    :0               17:35   11.00s  1:26   0.38s -bin/tcsh
artim    pts/1    :0               17:27   17.00s 23.66s 23.57s gedit appointments.txt
foobar   pts/2    :0               18:47    0.00s  0.20s  0.01s more SherlockHolmes.txt
$
```

Practice exercises

[Experiment on unimportant files and directories (create those as you need). Apply to important files after you gain sufficient confidence.]

0. Grow familiarity with Linux shell and Unix commands.

1. Explain the different behaviors of `ls -l /usr` and `ls -l /root`.

2. Enter the commands `man printf` and `man 3 printf`. Why do you get different outputs?

3. How can you enter multiple commands in a single line?

4. How can you enter one command in multiple lines (like `ls` in one line and `-l` in the second)?

5. Enter the command `wc` without any arguments. Write a few lines, and then hit `control-d` (with the `control` button pressed, hit `d`) at the beginning of a new line. See what happens. Explain the output. What does `control-d` do here?

6. Repeat the last exercise with `cat` (without any arguments).

7. What happens if you press `control-c` instead of `control-d`?

8. Enter `ls -l | wc | wc` as a command. Explain the output.

9. [*Disk usage*] Go to a directory that contains both regular files and subdirectories. Type the following commands and explain the differences: `du`, `du -a`, `du -s`, `du -sk`, `du -sm`, and `du -sh`. Explain the outputs.

Practice exercises

10. Try the command **`ls -l /dev`**. What kind of files do you see (look at the first character of each line)? What are these files?

11. Explain why the count of links to a directory is always $\geqslant 2$ (look at the number appearing immediately after the permissions, in each line of **`ls -l`**).

12. [*Symbolic links*] Create a non-empty text file **`testfile.txt`**. See the directory listing using **`ls -l`**. Then type the command **`ln -s testfile.txt T`**. Do **`ls -l`** again. What is the permission of **`T`**? Try changing the permission of **`T`** as **`chmod 000 T`**. What happens? Why? Remove **`T`**. What happens? Create another symbolic link **`TT`** to **`testfile.txt`**. Remove **`testfile.txt`**. What happens?

13. [*Hard links*] Create a text file **`abc.txt`**. See the directory listing. Then enter the command **`ln abc.txt ABC.txt`**. Again see the directory listing. What are the differences? Explain. Add some extra lines to **`abc.txt`**. Again see the directory listing. Explain the changes. Remove the original file **`abc.txt`**. Explain what the directory listing shows.

14. Study the commands **`pwd`**, **`chmod -R`**, **`chown`**, **`chgrp`**, **`date`**, **`time`**, **`strings`**, and **`exit`**.

15. Enter the command **`cal 1752`**, and look at September. Can you explain?