Working with commands & the filesystem

OS Practice CSC 420 Spring 2024

README

- You can code along with the file tinyurl.com/3-codealong.org

  You can save it to an Org-mode file `3-shell.org`

- Upload the finished codealong file [to Canvas]
- You can get the Emacs configuration file for Pi OS from `tinyurl.com/EmacsLyonPi`
- The solutions and results herein were all obtained on a Lenovo laptop running Linux Mint 21.3 with 4 Intel Core i3-6006U CPUs.
- This lab is based on chapter 5 (pp. 39-48) of Shotts (2019).
- You can get the completed file from `tinyurl.com/3-shell-org`

Command summary

COMMAND	MEANING	EXAMPLE
type	how a cmd name is interpreted	type ls
which	which cmd is executed	which ls
help	help for shell built-ins	help cd
man	full manual page	man ls
apropos	searches matching manuals	apropos –exact apt
info	info entry (hyperlinked)	info ls
whatis	one-line manual extract	whatis ls
alias	create command alias	alias ll='ls -alF'

The linux file system ("everything is a file")

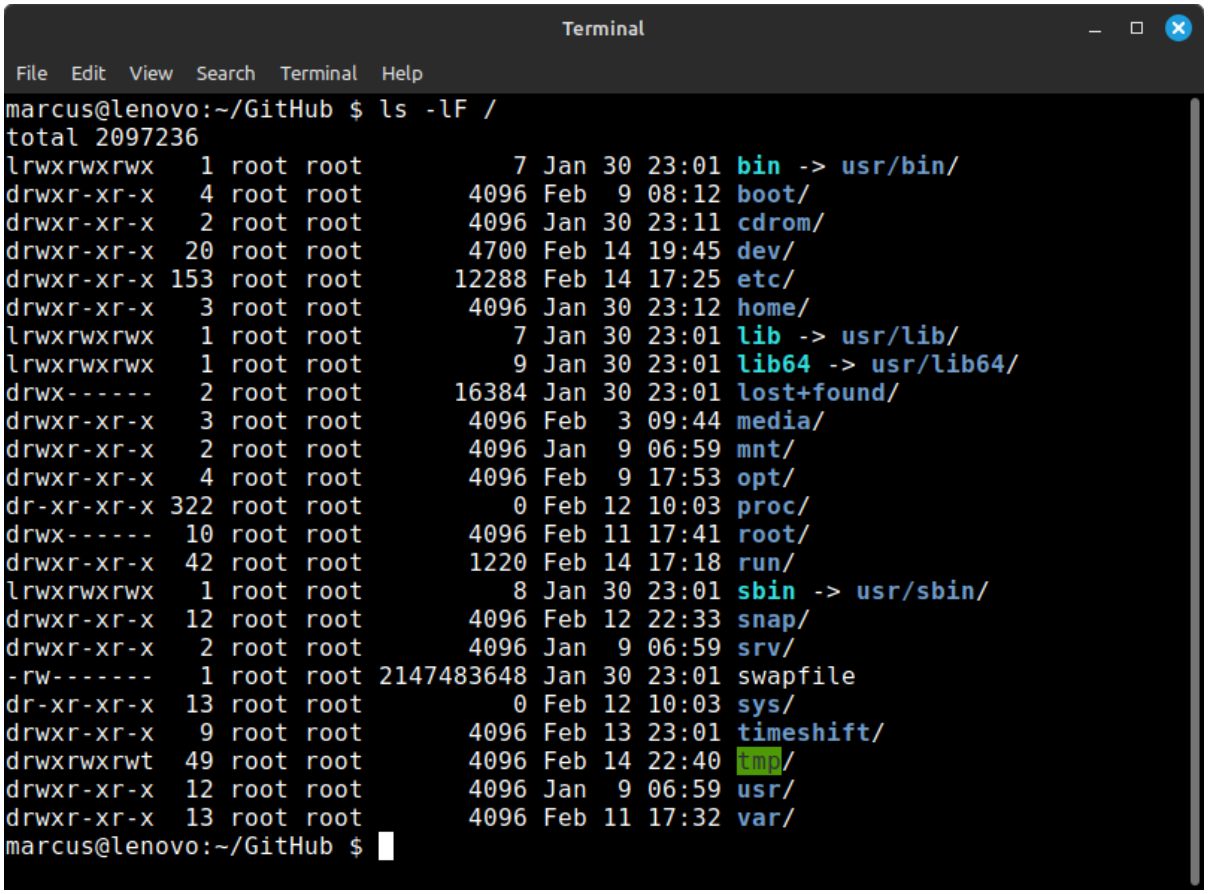
Open a terminal (or a shell in Emacs) and look at the file system

```
ls -lF /
```

```
total 2097236
lrwxrwxrwx   1 root root          7 Jan 30 23:01 bin -> usr/bin/
drwxr-xr-x   4 root root       4096 Feb  9 08:12 boot/
drwxr-xr-x   2 root root       4096 Jan 30 23:11 cdrom/
drwxr-xr-x  20 root root       4780 Feb 17 21:40 dev/
drwxr-xr-x 154 root root      12288 Feb 16 21:52 etc/
drwxr-xr-x   3 root root       4096 Jan 30 23:12 home/
lrwxrwxrwx   1 root root          7 Jan 30 23:01 lib -> usr/lib/
lrwxrwxrwx   1 root root          9 Jan 30 23:01 lib64 -> usr/lib64/
```

```
drwx------   2 root root      16384 Jan 30 23:01 lost+found/
drwxr-xr-x   3 root root       4096 Feb  3 09:44 media/
drwxr-xr-x   2 root root       4096 Jan  9 06:59 mnt/
drwxr-xr-x   4 root root       4096 Feb  9 17:53 opt/
dr-xr-xr-x 329 root root          0 Feb 16 21:04 proc/
drwx------  10 root root       4096 Feb 11 17:41 root/
drwxr-xr-x  42 root root       1220 Feb 19 22:41 run/
lrwxrwxrwx   1 root root          8 Jan 30 23:01 sbin -> usr/sbin/
drwxr-xr-x  15 root root       4096 Feb 15 11:08 snap/
drwxr-xr-x   2 root root       4096 Jan  9 06:59 srv/
-rw-------   1 root root 2147483648 Jan 30 23:01 swapfile
dr-xr-xr-x  13 root root          0 Feb 16 21:04 sys/
drwxr-xr-x   9 root root       4096 Feb 19 23:01 timeshift/
drwxrwxrwt  46 root root       4096 Feb 20 07:42 tmp/
drwxr-xr-x  12 root root       4096 Jan  9 06:59 usr/
drwxr-xr-x  13 root root       4096 Feb 11 17:32 var/
```

You'll see something like this:



DIRECTORY	CONTENT
`/`	Root directory where everything begins
`/bin`	Executable binaries for the OS to boot and run
`/boot`	Linux kernel, initial RAM disk image to boot
`/dev`	List for kernel with all known devices

DIRECTORY	CONTENT
`/etc`	System configuration files (e.g. `/etc/passwd`)
`/home`	Directory for user directories (e.g. `/home/pi`)
`/lib`	Shared library files (like Windows DLLs)
`/lost+found`	Panic room for each formatted disk partition
`/media`	Mount points for removable media (e.g. USB stick)
`/mnt`	Mount points for manually mounted removable media
`/opt`	Optional commercial software (e.g. browser)
`/proc`	Virtual FS for the kernel (e.g. `/proc/cpuinfo`)
`/root`	`$HOME` directory of the `root` super-user
`/sbin`	System binaries for system tasks (`sudo shutdown`)
`/tmp`	Holding bay for temp files, emptied at reboot
`/usr`	Programs and support files for regular users
`/usr/bin`	Executable programs of the distro (e.g. `cat`)
`/usr/lib`	Shared libraries for `/usr/bin` programs
`/usr/local`	Programs not included in your distro
`/usr/sbin`	More system administration programs
`/usr/share`	Shared data for `/usr/bin` programs (e.g. sound files)
`/usr/share/doc`	Man pages and other package documentation
`/var`	Databases, spool files, user mail (volatile files)
`/var/log`	Records of system activity (e.g. `/var/log/syslog`)

What is a command?

There are four types of commands: binaries/executables, built-ins, shell functions, and aliases.

1. An **executable** program e.g. in `/usr/bin` - could be compiled from source (e.g. from C), or scripted (e.g. from bash)
2. A **builtin**, a command built into the shell itself, like `cd`:

```
type cd
```

```
cd is a shell builtin
```

3. A **shell function**, scripts incorporated into the environment like `~/.bashrc`, which is a configuration file.
4. An `alias`, commands that we can define from other commands.

Identify commands with `type` and `which`

- `type` is a built-in (check that?), which displays the kind of command executed by the shell. Try it on `ls` and `type`, and then try `type -a grep`:

```
type ls
type type
type -a grep # all locations of the command
```

```
ls is /usr/bin/ls
type is a shell builtin
grep is /usr/bin/grep
grep is /bin/grep
```

- If you try `type -a grep` in a real terminal (not in Emacs), you get an additional answer, `grep is aliased to `grep --color=auto``.
- To find out `which` of perhaps many different commands with the same name is executed, use `which`.

```
which ls
```

```
/usr/bin/ls
```

- `which` only works for executable programs (not aliases, not builtins). Try it on a builtin command:

```
which cd
```

From a time before Google: getting help with `help`

- `bash` has a built-in help facility for each of the shell builtins. Try it for `cd`:

```
help cd
```

```
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

    Change the current directory to DIR.  The default DIR is the value of the
    HOME shell variable.

    The variable CDPATH defines the search path for the directory containing
    DIR.  Alternative directory names in CDPATH are separated by a colon (:).
    A null directory name is the same as the current directory.  If DIR begins
    with a slash (/), then CDPATH is not used.

    If the directory is not found, and the shell option `cdable_vars' is set,
    the word is assumed to be  a variable name.  If that variable has a value,
    its value is used for DIR.

    Options:
      -L        force symbolic links to be followed: resolve symbolic
```

```
                links in DIR after processing instances of `..'
      -P        use the physical directory structure without following
                symbolic links: resolve symbolic links in DIR before
                processing instances of `..'
      -e        if the -P option is supplied, and the current working
                directory cannot be determined successfully, exit with
                a non-zero status
      -@        on systems that support it, present a file with extended
                attributes as a directory containing the file attributes

    The default is to follow symbolic links, as if `-L' were specified.
    `..' is processed by removing the immediately previous pathname component
    back to a slash or the beginning of DIR.

    Exit Status:
    Returns 0 if the directory is changed, and if $PWD is set successfully when
    -P is used; non-zero otherwise.
```

- In all documentation, `[ ]` indicates optional items, like here:

```
cd [-L|[-P [-e]] [-@]] [dir]
```

- Meaning: `cd` can be followed by either `-L` or `-P`, and if `-P` is specified, then the `-e` option can be included followed by `dir` with the default `$HOME`.

- Show that `cd` can be run without option or argument, and that this use of `cd` defaults to going `$HOME`.

```
pwd
cd
pwd
```

```
/home/marcus/GitHub/os24/org
/home/marcus
```

- Many executable programs support the `--help` optoin that gives an overview of syntax and options: try this option for `type`.

```
type --help
```

```
type: type [-afptP] name [name ...]
    Display information about command type.

    For each NAME, indicate how it would be interpreted if used as a
    command name.

    Options:
      -a        display all locations containing an executable named NAME;
                includes aliases, builtins, and functions, if and only if
                the `-p' option is not also used
      -f        suppress shell function lookup
      -P        force a PATH search for each NAME, even if it is an alias,
                builtin, or function, and returns the name of the disk file
                that would be executed
      -p        returns either the name of the disk file that would be executed,
                or nothing if `type -t NAME' would not return `file'
```

```
      -t        output a single word which is one of `alias', `keyword',
                `function', `builtin', `file' or `', if NAME is an alias,
                shell reserved word, shell function, shell builtin, disk file,
                or not found, respectively

    Arguments:
      NAME      Command name to be interpreted.

    Exit Status:
    Returns success if all of the NAMEs are found; fails if any are not found.
```

- Try the option `--help` for `help`.

```
help --help
```

```
help: help [-dms] [pattern ...]
    Display information about builtin commands.

    Displays brief summaries of builtin commands.  If PATTERN is
    specified, gives detailed help on all commands matching PATTERN,
    otherwise the list of help topics is printed.

    Options:
      -d        output short description for each topic
      -m        display usage in pseudo-manpage format
      -s        output only a short usage synopsis for each topic matching
                PATTERN

    Arguments:
      PATTERN   Pattern specifying a help topic

    Exit Status:
    Returns success unless PATTERN is not found or an invalid option is given.
```

Getting help from the `man` page & your `fortune`

- Executable programs that can be run on the command line have a manual or man page. The `man` program is used to view them. Try it on `ls` first using a code block, and then display it in a separate buffer with `M-x man RET ls`.

```
man pwd
```

```
PWD(1)                          User Commands                          PWD(1)

NAME
       pwd - print name of current/working directory

SYNOPSIS
       pwd [OPTION]...

DESCRIPTION
       Print the full filename of the current working directory.

       -L, --logical
              use PWD from environment, even if it contains symlinks
```

```
       -P, --physical
              avoid all symlinks

       --help display this help and exit

       --version
              output version information and exit

       If no option is specified, -P is assumed.

       NOTE:  your shell may have its own version of pwd, which usually super-
       sedes the version described here.  Please refer to your  shell's  docu-
       mentation for details about the options it supports.

AUTHOR
       Written by Jim Meyering.

REPORTING BUGS
       GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
       Report any translation bugs to <https://translationproject.org/team/>

COPYRIGHT
       Copyright  ©  2020  Free Software Foundation, Inc.  License GPLv3+: GNU
       GPL version 3 or later <https://gnu.org/licenses/gpl.html>.
       This is free software: you are free  to  change  and  redistribute  it.
       There is NO WARRANTY, to the extent permitted by law.

SEE ALSO
       getcwd(3)

       Full documentation <https://www.gnu.org/software/coreutils/pwd>
       or available locally via: info '(coreutils) pwd invocation'

GNU coreutils 8.32              January 2024                            PWD(1)
```

- Try `man` on a shell builtin, e.g. `help`:

```
man help
```

- The format of a manual page is
  1. title (page name including the command section)
  2. synopsis of the syntax
  3. description of the purpos
  4. listing and description of each of the options
- Man pages are not vignettes like you might know them from R or from the Python standard library, with examples. They are not tutorials but only reference pages.
- `man` uses `less` to display its information.

- The Unix manual is broken into sections:

  1. run these in a terminal or inside Emacs. When you address a section, you can prefix the number, e.g. `man 3 printf`.
  2. find out `which` command is executed when you run it.

Section	Contents	Example
1	User commands (`/usr/`)	bash(1)
2	Interface to kernel system calls	write(2)

Section	Contents	Example
3	Interface to C library	printf(3)
4	Special device files (`/dev`)	/dev/null
5	File formats	/etc/passwd
6	Games	fortune
7	Miscellaneous	inode(7)
8	System admin commands/daemons	cron(8)

- Where are these commands located?

```
which bash
which write # see later in `redirection`
type -a printf # see example below
ls -l /dev/null # notice the file type `c`
ls -l /etc/passwd # check out `man 5 passwd`
ls -l fortune
ls -l /usr/share/man/man7/inode*
which cron # essential for scheduling backups, updates etc.
```

```
/usr/bin/bash
/usr/bin/write
printf is a shell builtin
printf is /usr/bin/printf
printf is /bin/printf
crw-rw-rw- 1 root root 1, 3 Feb 16 21:05 /dev/null
-rw-r--r-- 1 root root 3030 Feb  9 16:21 /etc/passwd
-rw-r--r-- 1 root root 4871 Jun 10  2021 /usr/share/man/man7/inode.7.gz
/usr/sbin/cron
```

- Some functions have the same name but are different programs. E.g. there is a user-command `printf(1)` to print stuff from the terminal:

```
printf "Hello, I'm printf(1)\n"
```

```
Hello, I'm printf(1)
```

  And there's `printf(3)` which refers to the standard library function in `stdio.h` that you use in C programs:

```
printf("Hello, I'm printf(3)\n");
```

```
Hello, I'm printf(3)
```

- Let's play the `fortune` game:

```
fortune
```

```
This was the most unkindest cut of all.
                -- William Shakespeare, "Julius Caesar"
```

- Hey, the command and its man page are not found. Open a fully functional terminal and run these commands:

```
$ sudo apt install fortune -y
$ fortune
$ man fortune
$ which fortune
```

- Can you generate a "potentially offensive" fortune cookie?

```
fortune -o > offensive_fortune
ls -l offensive_fortune
```

```
-rw-rw-r-- 1 marcus marcus 0 Feb 20 07:42 offensive_fortune
```

- The shell, `bash(1)` has one of the longest man pages (80). It's essentially a booklet. `GCC(1)` the C compiler beats this, it's a book. Don't print these out.

Display appropriate commands with `apropos`

- `apropos` is a search function that exists in many programs - e.g. Emacs has such a help (try `C-h a man`), and R does, too: in a terminal outside of Emacs, enter R, and on the console, enter:

```
R> ??Nile
R> ?datasets::Nile
```

- For a shell example, try `apropos` on `fortune`, `inode` and `cron`:

```
apropos fortune
apropos inode
apropos cron
```

```
fortune (6)          - print a random, hopefully interesting, adage
ecryptfs-find (1)    - use inode numbers to match encrypted/decrypted filenames
inode (7)            - file inode information
ioctl_iflags (2)     - ioctl() operations for inode flags
strmode (3bsd)       - convert inode status information into a symbolic string
anacrontab (5)       - configuration file for anacron
anacron (8)          - runs commands periodically
cron (8)             - daemon to execute scheduled commands (Vixie Cron)
crontab (1)          - maintain crontab files for individual users (Vixie Cron)
crontab (5)          - tables for driving cron
DateTime::Locale::en_FM (3pm) - Locale data examples for the English Microones...
```

- The `man` function with the `-k` flag performs the same job as `apropos`:

```
man -k fortune
```

```
fortune (6)          - print a random, hopefully interesting, adage
```

`whatis` in a command?

- `whatis` displays the name and a one-line description of a man page:

```
whatis ls whatis printf whatis regex whatis fortune
```

```
ls (1)               - list directory contents
whatis (1)           - display one-line manual page descriptions
printf (3)           - formatted output conversion
printf (1)           - format and print data
regex (3)            - POSIX regex functions
regex (7)            - POSIX.2 regular expressions
fortune (6)          - print a random, hopefully interesting, adage
```

Display a commands `info` entry

- Info is a GNU project. You can use it as a standalone manual or use it for example inside Emacs: try `C-h i` and you're in it.
- Info files are created from `.texi` TeXinfo files using the TeX typesetting system created by Donald Knuth (who also invented literate programming), generated with `texinfo`.
- Info pages are hyperlinked - this is in fact where Tim Berners-Lee (and Steve Jobs) very likely got the idea from. The creator of Emacs, Richard Stallman created the info system in the early 1980s.
- The `info` program reads info files, which are tree structured into nodes. Each node contains a single topic. Much like many Emacs modes, `info` navigation works with single letter commands like p,n,U,?.

- Print only the first 13 lines of the `info` for `head`:

```
info head | head -n 13
```

```
File: coreutils.info,  Node: head invocation,  Next: tail invocation,  Up: Output

5.1 ‘head’: Output the first part of files
==========================================

‘head’ prints the first part (10 lines by default) of each FILE; it
reads from standard input if no files are given or when given a FILE of
‘-’.  Synopsis:

     head [OPTION]... [FILE]...

   If more than one FILE is specified, ‘head’ prints a one-line header
consisting of:
```

  - Find the info for `head` inside the Emacs Info reader.
  - Most of the mentioned command line help programs are part of the GNU coreutils suite of programs: enter `info coreutils` in the terminal to see the info, and exit with `q`.

You must `README`

- You probably noticed the `README` headline that all my scripts start with: this is a Unix and developer tradition.
- Software distributions usually contain a README file that lists the main changes and the history of changes as well as anything that you need to know before starting to use the software.
- On GitHub, when you create a repo of your own, GitHub will create a README file for you.

Creating your own commands with `alias`

- Check out `~/.bashrc` (if you have it, which you should) and search (`C-s`) for `alias`.
- Use one of these aliases: open a terminal or a shell in Emacs (`M-x shell`) and enter `ll`, which is aliased for `ls -alF` (remember what this means? How can you find out?)

- You can put more than one command on one line separated by `;`

```
echo "hello"; echo "world"
```

```
hello
world
```

- On **one line**, change to `/usr`, list all files, and go back to `$HOME` again, then make sure you're `$HOME`.

```
cd /usr;ls;cd ~;pwd
```

```
bin
games
include
lib
lib64
libexec
local
sbin
share
src
/home/marcus
```

- We want to create a new command using `alias` called `test`. First, find out if `test` already exists:

```
type test
```

```
test is a shell builtin
```

- It does! Let's try `foo` (see foobar) in the format `alias='string'` - you have to do this in a fully functional shell.

```
$ alias foo='cd /usr;ls;cd ~;pwd' # defines the new command
$ foo # runs the command sequence
$ alias # shows all aliases
$ type foo # displays the character of the command
```

- To remove the alias, use `unalias` on the name:

```
$ unalias foo
$ type foo
```

- Why can we not do this in Emacs?

  > Because aliases are temporary and vanish when the session ends, and because the Emacs terminal is only a simulatin (`M-x term` would work though but it's hard to get rid of - try it: you have to `exit` to kill it.)

References

Shotts W (2019). The Linux Command Line (2e), NoStarch Press.

Author: [yourname]

Created: 2024-02-20 Tue 07:42

Validate