INTRODUCTION TO THE (LINUX) SHELL.

WHAT IS A SHELL?

WHAT IS A SHELL?

○ In computing, a shell is a user interface for access to an operating system's services. In general, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI), depending on the computer's role and particular operation. It is named a shell because it is the outermost layer around the operating system kernel.

○ A program that interprets commands.

○ Allows a user to execute commands by typing them manually at a terminal, or automatically in programs called shell scripts, or simply scripts.

○ A shell is not an operating system. It is a way to interface with the operating system and run commands.

Alternate names: console, terminal, command line, command line interface, command prompt.

SHELL TYPES:

Just like people know different languages and dialects, your UNIX system will usually offer a variety of shell types:

○**sh** or Bourne Shell: the original shell still used on UNIX systems and in UNIX−related environments. This is the basic shell, a small program with few features. While this is not the standard shell anymore, it is still available on every Linux system for compatibility with UNIX programs.

○**bash** or Bourne Again shell: the standard GNU shell, intuitive and flexible. Probably most advisable for beginning users while being at the same time a powerful tool for the advanced and professional user. On Linux, bash is the standard shell for common users. This shell is a so−called superset of the Bourne shell, a set of add−ons and plug−ins. This means that the Bourne Again shell is compatible with the Bourne shell: commands that work in sh, also work in bash. However, the reverse is not always the case. All examples and exercises in this book use bash.

○**csh** or C shell: the syntax of this shell resembles that of the C programming language. Sometimes asked for by programmers.

○**tcsh** or Turbo C shell: a superset of the common C shell, enhancing user−friendliness and speed.

○ksh or the Korn shell: sometimes appreciated by people with a UNIX background. A superset of the Bourne shell; with standard configuration a nightmare for beginning users.

The file `/etc/shells` gives an overview of known shells on a Linux system:

```
$ cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/bin/ksh
```

WHAT IS BASH?

- BASH = Bourne Again SHell

- Bash is a shell written as a free replacement to the standard Bourne Shell (/bin/sh) originally written by Steve Bourne for UNIX systems.

- It has all of the features of the original Bourne Shell, plus additions that make it easier to program with and use from the command line.

- Since it is Free Software, it has been adopted as the default shell on most Linux systems and other Unix flavours.

ACCESS

Windows Instructions:

1. Search for "putty download" on Google.

2. Select this result: www.chiark.greenend.org.uk (It's the original….).

3. Follow link download putty

4. Install putty

5. Run putty

ACCESS

Mac Instructions:

1. Run terminal

ACCESS

Mac Instructions:

1. Run terminal

Is that easy?!

Yes, it's Mac!

Detailed instructions:

To find "terminal":

1. Press command+space

2. Type "terminal"

3. Move icon to the dock (you will be using it on a daily basis from now on until the end of ages)

4. Click on it to open "terminal".

ACCESS

Mac (Terminal):

$ ssh username@niagara.scinet.utoronto.ca

Windows (Putty):

Host Name: niagara.scinet.utoronto.ca

User: <Your username>
Password: <Your password>

COMMON COMMANDS

ls	List contents of a directory	**more**	Show files per page	**head**	Show head of a file
pwd	Print Working Directory	**less**	Show files per page	**tail**	Show tail of a file
ps	Process Status	**touch**	Touch a file	**history**	Show command history
cd	Change Directory	**man**	Linux Manuals	**find**	find ANYTHING in the filesystems
mv	Move	**grep**	Find a pattern in a file		
rm	Remove	**date**	Show/modify date	**chmod**	Change permissions
mkdir	Make directory	**time**	Time execution of a command	**wc**	Word count
rmdir	Remove directory				
cat	Concatenate	**vi**	Powerfull text editor		
		echo	Send to stdout		

COMMON COMMANDS

pwd   Print Working Directory

Print the full filename of the current working directory:

```
$ pwd
/usr/src
```

ls   List contents of a directory:

```
$ ls
iso  linux  linux-4.11.8-1  linux-4.11.8-1-obj  linux-obj  packages  vboxhost-5.1.22  vtun-
2.6.tar.gz
```

ls   List contents of a directory:

(long listing format)

```
$ ls -l
total 116
drwxrwxr-x  3 root root  4096 Nov  5 10:28 iso
lrwxrwxrwx  1 root root    14 Jul 16  2017 linux -> linux-4.11.8-1
drwxr-xr-x 24 root root  4096 Jul 16  2017 linux-4.11.8-1
drwxr-xr-x  3 root root  4096 Jul 16  2017 linux-4.11.8-1-obj
drwxr-xr-x  3 root root  4096 Jul 16  2017 linux-obj
drwxr-xr-x  8 root root  4096 Jul 16  2017 packages
lrwxrwxrwx  1 root root    34 Apr 28  2017 vboxhost-5.1.22 -> /usr/share/virtualbox/src/vboxhost
-rw-r--r--  1 mts  root 95637 Aug 23 12:49 vtun-2.6.tar.gz
```

date   Show/modify date:

print or set the system date and time:

```
$ date
Tue 26 Oct 2021 13:23:56 EDT
```
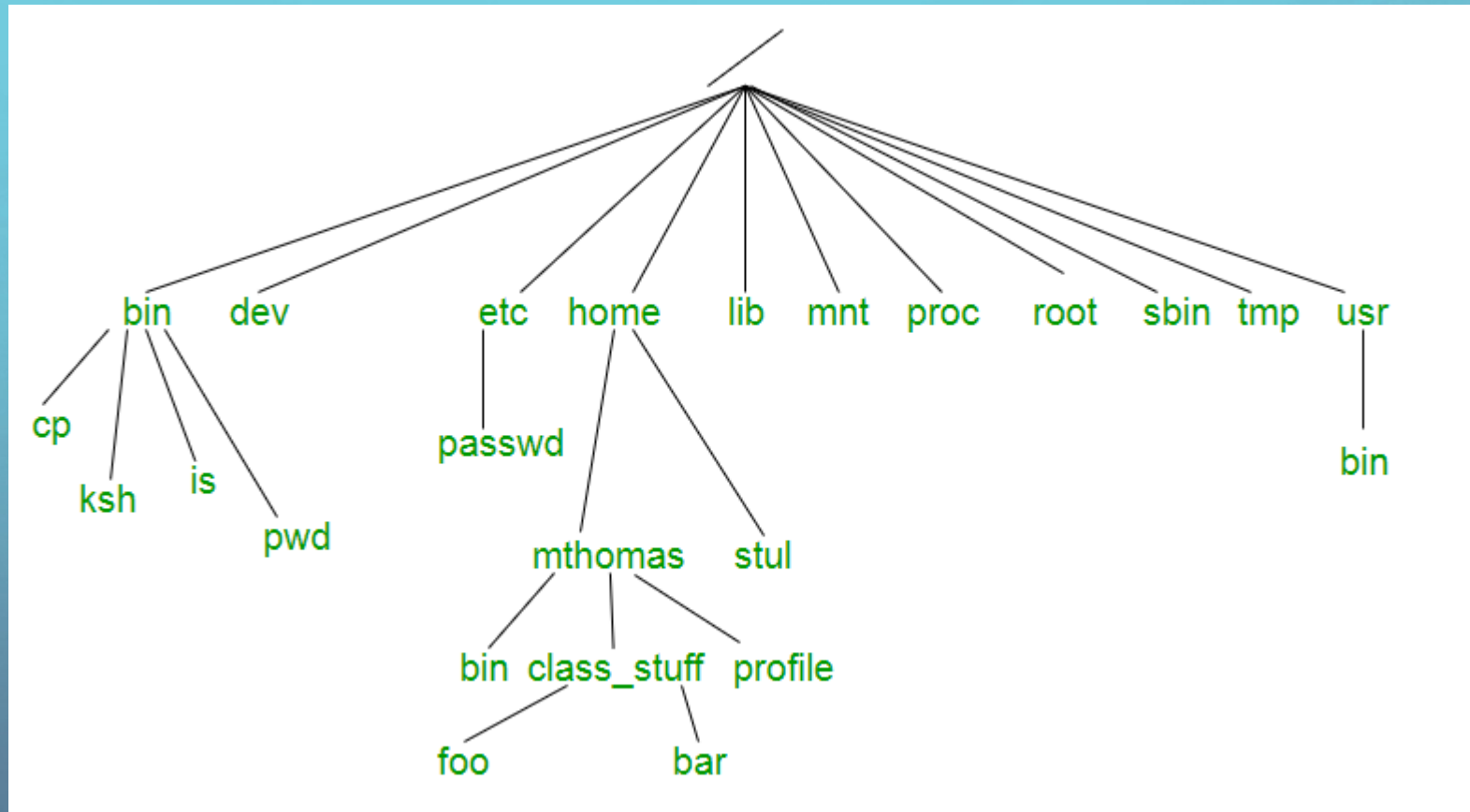
NAVIGATING THE FILE SYSTEM

Where is my C: drive?!

A filesystem organizes a computer's files and directories into a tree structure: The first directory in the filesystem is the root directory. It is the parent of all other directories and files.

The Unix filesystem is a tree-like hierarchy of directories and files. At the base of the filesystem is the "/" directory, otherwise known as the "root" (not to be confused with the root user) or "root directory". Unlike DOS or Windows filesystems that have multiple "roots", one for each disk drive, the Unix filesystem mounts all disks somewhere underneath the "/" filesystem.

NAVIGATING THE FILE SYSTEM

NAVIGATING THE FILE SYSTEM
THE LINUX DIRECTORY LAYOUT:

Directory	Description
/	The nameless base of the filesystem. All other directories, files, drives, and devices are attached to this root. Commonly (but incorrectly) referred to as the "slash" or "/" directory. The "/" is just a directory separator, not a directory itself.
/bin	Essential command binaries (programs) are stored here (bash, ls, mount, tar, etc.)
/boot	Static files of the boot loader. (Kernel, initrd, etc)
/dev	Device files. In Linux, hardware devices are accessed just like other files, and they are kept under this directory.
/etc	Host-specific system configuration files.
/home	Location of users' personal home directories (e.g. /home/susan).
/lib	Essential shared libraries and kernel modules.
/proc	Process information pseudo-filesystem. An interface to kernel data structures.
/root	The root (superuser) home directory.

NAVIGATING THE FILE SYSTEM
THE LINUX DIRECTORY LAYOUT:

Directory	Description
/sbin	Essential system binaries (fdisk, fsck, init, etc).
/tmp	Temporary files. All users have permission to place temporary files here.
/usr	The base directory for most shareable, read-only data (programs, libraries, documentation, and much more).
/usr/include	Header files for compiling C programs.
/usr/lib	Libraries for most binary programs.
/usr/sbin	Non-vital system binaries (lpd, useradd, etc.)
/usr/share	Architecture-independent data (icons, backgrounds, documentation, man pages, etc.).
/usr/src	Program source code. E.g. The Linux Kernel, source RPMs, etc.
/usr/X11R6	The X Window System.

NAVIGATING THE FILE SYSTEM
THE LINUX DIRECTORY LAYOUT:

Directory	Description
/var	Variable data: mail and printer spools, log files, lock files, etc.
/sys	Modern Linux distributions include a /sys directory as a virtual filesystem (sysfs, comparable to /proc, which is a procfs), which stores and allows modification of the devices connected to the system
/lost+found	The lost+found directory is a construct used by `fsck` when there is damage to the filesystem (not to the hardware device, but to the fs). Files that would normally be lost because of directory corruption would be linked in that filesystem's lost+found directory by inode number.
/mnt	This is a generic mount point under which you mount your filesystems or devices.
/opt	This directory is reserved for all the software and add-on packages that are not part of the default installation.

SPECIAL CHARACTERS

It is important to know that there are many symbols and characters that the shell interprets in special ways. This means that certain typed characters:

a) cannot be used in certain situations,

b) may be used to perform special operations, or,

c) must be "escaped" if you want to use them in a normal way.

SPECIAL CHARACTERS

Character	Description
\	Escape character. If you want to reference a special character, you must "escape" it with a backslash first. Example: touch /tmp/filename*
/	Directory separator, used to separate a string of directory names. Example: /usr/src/linux
.	Current directory. Can also "hide" files when this is the first character in a filename.
..	Parent directory
~	User's home directory
*	Wildcard character. Represents 0 or more characters in a filename, or by itself, all files in a directory. Example: pic*2002 can represent the files pic2002, picJanuary2002, picFeb292002, etc.
?	Represents a single character in a filename. Example: hello?.txt can represent hello1.txt, helloz.txt, but not hello22.txt
[ ]	Can be used to represent a range of values, e.g. [0-9], [A-Z], etc. Example: hello[0-2].txt represents the names hello0.txt, hello1.txt, and hello2.txt
| |&	"Pipe". Redirect the output of one command into another command. Example: ls | more

SPECIAL CHARACTERS

Character	Description
>	Redirect output of a command into a new file. If the file already exists, over-write it. Example: ls > myfiles.txt
>>	Redirect the output of a command onto the end of an existing file. Example: echo "Mary 555-1234" >> phonenumbers.txt
<	Redirect a file as input to a program. Example: more < phonenumbers.txt
;	Command separator. Allows you to execute multiple commands on a single line. Example: cd /var/log ; less messages
&&	Command separator as above, but only runs the second command if the first one finished without errors. Example: cd /var/log && less messages
||	Command separator as above, but only runs the second command if the first one finished with errors. Example: grep –q kern messages|| echo "No kernel messages"
&	Execute a command in the background, and immediately get your command line back. Example: find / -name core > /tmp/corefiles.txt &

FILE DESCRIPTORS

In Unix, a file descriptor (fd) is an abstract indicator (handle) used to access a file or other input/output resource, such as a pipe or network socket.

Each Unix process should expect to have three standard POSIX file descriptors, corresponding to the three standard streams:

Integer value	Name	<unistd.h> symbolic constant	<stdio.h> file stream
0	Standard input	`STDIN_FILENO`	`stdin`
1	Standard output	`STDOUT_FILENO`	`stdout`
2	Standard error	`STDERR_FILENO`	`stderr`

FILE DESCRIPTORS

Standard input (stdin)

Standard input is stream data going into a program. The program requests data transfers by use of the read operation. Not all programs require stream input. For example, the ls program (which display file names contained in a directory) may take command-line arguments, but perform their operations without any stream data input. Unless redirected, standard input is inherited from the parent process. In the case of an interactive shell, that is usually associated with the keyboard.

The file descriptor for standard input is 0 (zero); the POSIX <unistd.h> definition is STDIN_FILENO; the corresponding C <stdio.h> variable is FILE* stdin; similarly, the C++ <iostream> variable is std::cin.

Standard output (stdout)

Standard output is the stream where a program writes its output data. The program requests data transfer with the write operation. Not all programs generate output. For example, the file mv command is silent on success. Unless redirected, standard output is inherited from the parent process. In the case of an interactive shell, that is usually the text terminal which initiated the program.

The file descriptor for standard output is 1 (one); the POSIX <unistd.h> definition is STDOUT_FILENO; the corresponding C <stdio.h> variable is FILE* stdout; similarly, the C++ <iostream> variable is std::cout.

Standard error (stderr)

Standard error is another output stream typically used by programs to output error messages or diagnostics. It is a stream independent of standard output and can be redirected separately. The usual destination is the text terminal which started the program to provide the best chance of being seen even if standard output is redirected (so not readily observed). For example, output of a program in a pipeline is redirected to input of the next program, but errors from each program still go directly to the text terminal.

The file descriptor for standard error is defined by POSIX as 2 (two); the <unistd.h> header file provides the symbol STDERR_FILENO;[2] the corresponding C <stdio.h> variable is FILE* stderr.

REDIRECTION

Input and Output of a command may be redirected before it is executed, using a special notation, the redirection operators, interpreted by the shell.

Redirection operators:

<   Read from

>   Write to

>>   Append to

| Pipe

PIPE (ALSO CALLED PIPELINE)

This is one of the most powerful tools of bash. A pipeline is a way in which the output of one command becomes the input of a second command.

```
command1 | command2
```

The stdout of command1 is the stdin of command2

```
command1 |& command2
```

The stdout, AND the stderr, of command1 is the stdin of command2

You can use the pipeline more than once in a command line:

```
command1 | command2 | command3
```

The stdout of command1 is the stdin of command2 and the stdout of

command2 is the stdin of command3

PIPE (ALSO CALLED PIPELINE)

Example:

```
ls -la /usr/bin | more
```

In this example, we run the command "ls -la /usr/bin", which gives us a long listing of all of the files in /usr/bin. Because the output of this command is typically very long, we pipe the output to a program called "more", which displays the output for us one screen at a time.

BUILT-IN COMMANDS

Built-in commands are necessary to implement functionality impossible or inconvenient to obtain with separate utilities.

Bash supports 3 types of built-in commands:

Bourne Shell built-ins:

:, ., break, cd, continue, eval, exec, exit, export, getopts, hash, pwd, readonly, return, set, shift, test, [, times, trap, umask and unset.

Bash built-in commands:

alias, bind, builtin, command, declare, echo, enable, help, let, local, logout, printf, read, shopt, type, typeset, ulimit and unalias.

Special built-in commands:

When Bash is executing in POSIX mode, the special built-ins differ from other built-in commands.

The POSIX special built-ins are: :, ., break, continue, eval, exec, exit, export, readonly, return, set, shift, trap and unset.

PERMISSIONS

t, let see what does it mean the information provided in a long listing:

long listing

show invisible files

directory

regular file

symbolic link

```
$ ls -l -a /boot/
total 32660
drwxr-xr-x  4 root root      4096 Sep  9 12:17 .
drwxr-xr-x 23 root root      4096 Jul 16  2017 ..
-rw-r--r--  1 root root      1725 May 24  2017 boot.readme
-rw-r--r--  1 root root    191697 Jul  8  2017 config-4.11.8-1-default
drwxrwxr-x  3 root root     16384 Dec 31  1969 efi
drwxr-xr-x  7 root root      4096 Sep  9 12:17 grub2
lrwxrwxrwx  1 root root        23 Jul 16  2017 initrd -> initrd-4.11.8-1-default
-rw-------  1 root root  11187208 Sep  9 12:17 initrd-4.11.8-1-default
-rw-r--r--  1 root root   1095036 Jul  8  2017 symtypes-4.11.8-1-default.gz
-rw-r--r--  1 root root    381495 Jul  8  2017 symvers-4.11.8-1-default.gz
-rw-r--r--  1 root root       484 Jul  8  2017 sysctl.conf-4.11.8-1-default
-rw-r--r--  1 root root   3305559 Jul  8  2017 System.map-4.11.8-1-default
-rw-r--r--  1 root root   9981850 Jul  8  2017 vmlinux-4.11.8-1-default.gz
lrwxrwxrwx  1 root root        24 Jul 16  2017 vmlinuz -> vmlinuz-4.11.8-1-default
-rw-r--r--  1 root root   7241840 Jul  8  2017 vmlinuz-4.11.8-1-default
-rw-r--r--  1 root root        65 Jul  8  2017 .vmlinuz-4.11.8-1-default.hmac
```

Destination of the symbolic link

file name
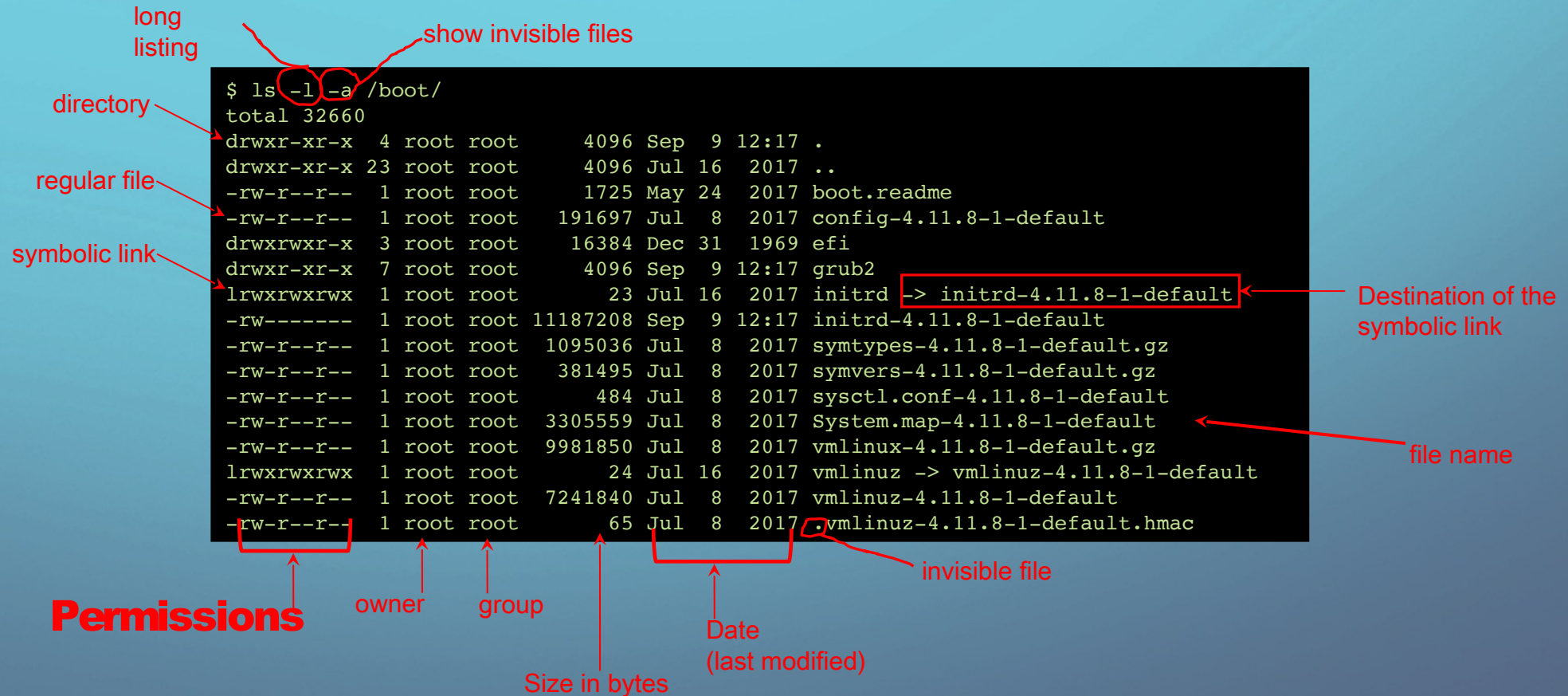
Permissions

owner

group

Size in bytes

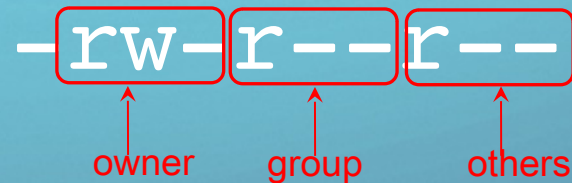Date (last modified)

invisible file

PERMISSIONS

```
$ ls -l -a /boot/vmlinuz*
lrwxrwxrwx 1 root root        24 Jul 16  2017 /boot/vmlinuz -> vmlinuz-4.11.8-1-default
-rw-r--r-- 1 root root 7241840 Jul  8  2017 /boot/vmlinuz-4.11.8-1-default
```

Permissions

There are nine permission settings (also called bits). These settings are divided in three groups of three each one:

-rw-r--r--

owner   group   others

The first "bit" is for read, the second bit if for write and the third is for execute:

rwx

PERMISSIONS

This:

rwx

can be expressed as a number. A binary number, or a decimal number.

Since these three letters are actually bits:

```
rwx is the same as 111 (binary) or 7 (decimal)
rw- is the same as 110 (binary) or 6 (decimal)
r-- is the same as 100 (binary) or 4 (decimal)
--- is the same as 000 (binary) or 0 (decimal)
--x is the same as 001 (binary) or 1 (decimal)
```

PERMISSIONS

Permissions	Decimal	Description
rwxrwxrwx	777	All permissions for everybody (Dangerous!).
rw-r--r--	644	Read and write for the owner. Read for everybody else. (Most common for regular files).
r--r--r--	444	Read only for everybody
rwxr-xr-x	755	Read and write and execute for the owner. Read and execute for everybody else. (Most common for directories).
r--------	400	Only the owner can read the file.
r--r-----	440	Only the owner and the group members can read the file.
r-xr-x---	550	Only the owner and the group members can read and execute the file.
---------	000	No permissions for anybody

CHMOD

chmod is the command for modifying permissions (change file mode bits).

Here there are some examples:

Command	Description
chmod +x <FILE>	Add execution permissions to file
chmod 777 <FILE>	Add ALL permissions for everybody (Dangerous!)
chmod 755 <FILE>	Read, write and execute for the owner, read and execute for everybody else.
chmod 644 <FILE>	Read and write for the owner, read-only for everybody else