Introduction to 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.

WHAT IS A SHELL?

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.

WHAT IS A SHELL?

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 course use bash.

SHELL TYPES

csh. The original C shell isn't much used on Linux, but if a user is familiar with csh, tcsh makes a good substitute.

tcsh. This shell is based on the earlier C shell (csh). It's a fairly popular shell in some circles, but no major Linux distributions make it the default shell.

ksh. The Korn shell (ksh) was designed to take the best features of the Bourne shell and the C shell and extend them further.

zsh. The Z shell (zsh) takes shell evolution further than the Korn Shell, incorporating features from earlier shells and adding still more. zsh is the new default shell on macOS.

This course is about bash. This is the default shell in Linux and is the one we will use in this course. For now on the terms "shell" and "bash" refer to the same.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

There are different ways to access a Linux system from Windows:

- ➔PuTTY
- ➔Windows PowerShell
- ➔Secure Shell for Google Chrome
- ➔OpenSSH for Cygwin Terminal

ACCESS

Windows

PuTTY:

- Search for “putty download” on Google.
- Select this result: www.chiark.greenend.org.uk (This is the original...).
- Follow link to download putty
- Install putty
- Run putty

ACCESS

Windows

Windows PowerShell:

Windows PowerShell has slowly been taking over from the Windows Command Prompt app since it was introduced in Windows 7. More recently, support for OpenSSH has been added, which you can incorporate in PowerShell as follows:

- Press WIN + I to open Settings.
- Open Apps > Apps & features
- Click Optional features
- Click +Add a feature
- Browse the list to find OpenSSH Client
- Select and click Install
- When this has completed reboot Windows 10

With OpenSSH added, you can use it by opening Windows PowerShell (right-click Start > PowerShell) and typing a connection command.

ACCESS

Windows

- Windows PowerShell:

  With OpenSSH added, you can use it by opening Windows PowerShell (right-click Start > PowerShell) and typing a connection command.

```
ssh username@192.168.1.10
```

ACCESS

Windows

Secure Shell for Google Chrome:

Google provides an SSH client called Secure Shell App, that can be added to the Chrome browser. Just install the Secure Shell app from the Chrome Web store. Although it runs in the Chrome browser, it runs completely offline so you don't need internet access to use it. So it works as well with devices on your local network as it does with remote servers.

Secure Shell App opens as a browser tab. Simply enter your credentials and the hostname (IP address) of the remote SSH server. You can also append additional SSH command-line arguments, if necessary.

As with other Chrome web apps, the Secure Shell App can open in a dedicated window to separate it from your main browser.

As Secure Shell is a Chrome web app, it's also available for macOS, Linux, and even Chrome OS.

ACCESS

Windows

OpenSSH for Cygwin Terminal:

You can download Cygwim from: http://cygwin.com/install.html

Cygwin is a large installation package so you may prefer to install just OpenSSH.

To do this, run the installer and when you're prompted to Select Packages, search for OpenSSH. Expand Net and in the New column, click Skip so it displays the version to download.

Click Next to proceed, review the packages to be installed, then Next again.

After the installation process finishes, launch Cygwin's Terminal application from the Start menu. To start an SSH connection, use the same ssh command that you'd run on Linux and other UNIX-like operating systems.

ACCESS

Mac Instructions:

Run terminal

ACCESS

Mac Instructions:

Run terminal

Is that easy?!

Yes, it's Mac!

Detailed instructions:

To find "terminal":

- Press command+space
- Type "terminal"
- Move icon to the dock (you will be using it on a daily basis from now on until the end of ages)

Click on it to open "terminal".

ACCESS

Mac or Linux (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 one page at a time	**head**	Show head of a file
pwd	Print Working Directory	**less**	Show files one page at a time	**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 (Rename)	**grep**	Find a pattern in a file	**chmod**	Change permissions
rm	Remove	**date**	Show/Modify system date	**chown**	Change owner of an object
mkdir	Make directory	**time**	Time execution of a command	**chgrp**	Change group of an object
rmdir	Remove directory	**vi (vim)**	Powerful text editor	**wc**	Word count
cat	Concatenate	**echo**	Send to stdout		

COMMON COMMANDS

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
```

pwd Print Working Directory:

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

COMMON COMMANDS

date Show/modify date. Print or set the system date and time:

```
$ date
Mon Jun 12 10:27:28 EDT 2023
```

echo send to stdout:

```
$ echo 'Hello World!'
Hello World!
```

echo can be used to expand variables:

```
$ echo $USER
mts
$ echo $HOME
/home/mts
```

mkdir make a directory:

```
$ mkdir linux_course
$ ls
command.out  directory  histcontrol.txt  linux_course  linux_script  newdir  test
$ cd linux_course/
$ pwd
/home/mts/linux_course
```

NAVIGATING THE FILE SYSTEM

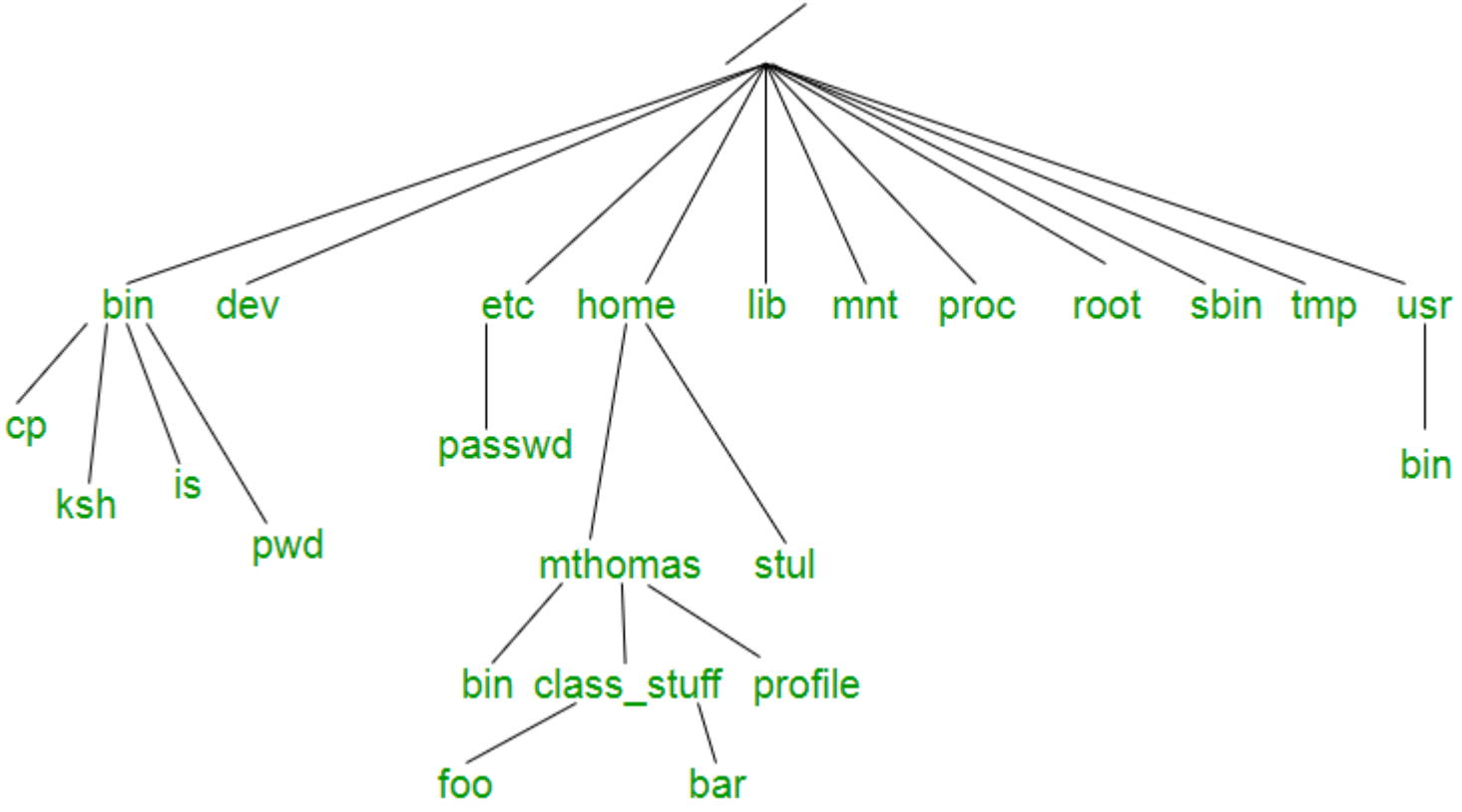
Where is my C: drive?!

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.

Introduction to Linux Shell

Exploring Your Linux Shell Options

PATH

PATH is a shell variable, also called an environment variable. It holds the search path for commands. It is a colon-separated list of directories in which the shell looks for commands. A common value is:

```
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
```

PATH is slightly different for the root user:

```
# echo $PATH
/sbin:/usr/sbin:/usr/local/sbin:/usr/local/bin:/usr/bin:/bin:/usr/games:/root/bin
```

Exploring Your Linux Shell Options

PATH

When you type a command in the command-line, the shell looks for a file in the directories listed in the PATH variable, in the same order as they are in the variable. When the shell finds the first, it runs it.

If you want to know where the executable file is located, you can run a command called **which**:

```
$ which grep
/bin/grep
$ which nocommand
which: no nocommand in (/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin)
```

Exploring Linux command-line tools

Performing Some Shell Command Tricks

Command completion

Many users find typing commands to be tedious and error prone. This is particularly true of slow or sloppy typists. For this reason, Linux bash shell include various tools that can help speed up operations

Exploring Linux command-line tools

Performing Some Shell Command Tricks

Command completion

The first of these is command completion: type part of a command or (as an option to a command) a filename, and then press the Tab key. The shell tries to fill in the rest of the command or the filename. If just one command or filename matches the characters you've typed so far, the shell fills it in and places a space after it. If the characters you've typed don't uniquely identify a command or filename, the shell fills in what it can and then stops. Depending on the shell and its configuration, it may beep. If you press the Tab key again, the system responds by displaying the possible completions. You can then type another character or two and, if you haven't completed the command or filename, press the Tab key again to have the process repeat.

Exploring Linux command-line tools

Performing Some Shell Command Tricks

history

This is, by far, the most powerful tool of the command-line. The history keeps a record of every command you type (stored in ~/.bash_history). If you've typed a long command recently and want to use it again, or use a minor variant of it, you can pull the command out of the history.

The simplest way to do this is to press the Up arrow key on your keyboard; this brings up the previous command. Pressing the Up arrow key repeatedly moves through multiple commands so you can find the one you want. If you overshoot, press the Down arrow key to move down the history.

Exploring Linux command-line tools

Performing Some Shell Command Tricks

history (continued)

Frequently, after finding a command in the history, you want to edit it. The bash shell, provides editing features modelled after those of the Emacs editor

Editing the command line

Delete text Pressing

The Delete key deletes the character under the cursor, whereas pressing the Backspace key deletes the character to the left of the cursor. Pressing Ctrl+K deletes all text from the cursor to the end of the line. Pressing Ctrl+X and then Backspace deletes all the text from the cursor to the beginning of the line. Pressing Ctrl-U deletes all text from the cursor to the beginning of the line.

Transpose text

Pressing Ctrl+T transposes the character before the cursor with the character under the cursor. Pressing Esc and then T transposes the two words immediately before (or under) the cursor.

Exploring Linux command-line tools

Performing Some Shell Command Tricks

history (continued)

Editing the command line (continued)

Change case

Pressing Esc and then U converts text from the cursor to the end of the word to uppercase. Pressing Esc and then L converts text from the cursor to the end of the word to lowercase. Pressing Esc and then C converts the letter under the cursor (or the first letter of the next word) to uppercase, leaving the rest of the word unaffected.

Exploring Linux command-line tools

Performing Some Shell Command Tricks

history (continued)

These editing commands are just the most useful ones supported by bash history; consult its ***man page*** to learn about many more obscure editing features. In practice, you're likely to make heavy use of command and filename completion, the command history, and perhaps a few editing features.

The history command provides an interface to view and manage the history.

Typing history alone displays all the commands in the history (typically the latest 500 commands); adding a number causes only that number of the latest commands to appear. Typing history -c clears the history, which can be handy if you've recently typed commands you'd rather not have discovered by others (such as commands that include passwords).

Exploring Linux command-line tools

Performing Some Shell Command Tricks

Exercise

Editing Commands

To experiment with your shell's completion and command-line editing tools, follow these steps:

1) Log in as an ordinary user.
2) Create a temporary directory by typing mkdir test.
3) Change into the test directory by typing cd test.
4) Create a few temporary files by typing touch one two three. This command creates three empty files named one, two, and three.
5) Type ls -l t, and without pressing the Enter key, press the Tab key. The system may beep at you or display two three. If it doesn't display two three, press the Tab key again, and it should do so. This reveals that either two or three is a valid completion to your command, because these are the two files in the test directory whose filenames begin with the letter t.

Exploring Linux command-line tools

Performing Some Shell Command Tricks

Exercise

Editing Commands (continued)

6) Type h, and again without pressing the Enter key, press the Tab key. The system should complete the command (ls -l three), at which point you can press the Enter key to execute it. (You'll see information on the file.)
7) Press the Up arrow key. You should see the ls -l three command appear on the command line.
8) Press Ctrl+A to move the cursor to the beginning of the line.
9) Press the Right arrow key once, and type es (without pressing the Enter key). The command line should now read less -l three.
10) Press the Right arrow key once, and press the Delete key three times. The command should now read less three. Press the Enter key to execute the command. (Note that you can do so even though the cursor isn't at the end of the line.) This invokes the less pager on the three file. (The less pager is described more fully later, in "Getting Help.") Because this file is empty, you'll see a mostly empty screen.
11) Press the Q key to exit from the less pager.

Exploring Linux command-line tools

Shortcuts

aliases

A bash alias is essentially nothing more than a keyboard shortcut, an abbreviation, a means of avoiding typing a long command sequence. If, for example, we include:

```
alias lm="ls -l | more"
```

in the ~/.bashrc file, then each ***lm*** typed at the command-line will automatically be replaced by a ***ls -l | more***. This can save a great deal of typing at the command-line and avoid having to remember complex combinations of commands and options.

Setting alias **rm="rm -i"** (interactive mode delete) may save a good deal of grief, since it can prevent inadvertently deleting important files.

Exploring Linux command-line tools

Getting Help

apropos

apropos - search the manual page names and descriptions.

Each manual page has a short description available within it. apropos searches the descriptions for instances of a keyword.

```
$ apropos grep

bzgrep (1)           - search possibly bzip2 compressed files for a regular expression
egrep (1)            - print lines matching a pattern
fgrep (1)            - print lines matching a pattern
grep (1)             - print lines matching a pattern
grep (1p)            - search a file for a pattern
grep-changelog (1)   - print ChangeLog entries matching criteria
msggrep (1)          - pattern matching on message catalog
pgrep (1)            - look up or signal processes based on name and other attributes
pm-utils-bugreport-info.sh (8) - Print pm-utils bug report
xzegrep (1)          - search compressed files for a regular expression
xzfgrep (1)          - search compressed files for a regular expression
xzgrep (1)           - search compressed files for a regular expression
zgrep (1)            - search possibly compressed files for a regular expression
zipgrep (1)          - search files in a ZIP archive for lines matching a pattern
```

Exploring Linux command-line tools

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

Exploring Linux command-line tools

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.

Exploring Linux command-line tools

FILE DESCRIPTORS

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.

Exploring Linux command-line tools

FILE DESCRIPTORS

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.

Exploring Linux command-line tools

Using Streams, Redirection, and Pipes

Exploring Types of Streams

In short, these three are the most important ones for this topic:

Standard input
Programs accept keyboard input via standard input, or stdin. In most cases, this is the data that comes into the computer from a keyboard.

Standard output
Text-mode programs send most data to their users via standard output (a.k.a. stdout), which is normally displayed on the screen, either in a full-screen text-mode session or in a GUI window such as an xterm.

Standard error
Linux provides a second type of output stream, known as standard error, or stderr. This output stream is intended to carry high-priority information such as error messages. Ordinarily, standard error is sent to the same output device as standard output, so you can't easily tell them apart. You can redirect one independently of the other, though, which can be handy. For instance, you can redirect standard error to a file while leaving standard output going to the screen so that you can interact with the program and then study the error messages later.

Exploring Linux command-line tools

Redirection

Redirecting Input and Output

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

Exploring Linux command-line tools

Redirection

Redirecting Input and Output (continued)

To redirect input or output, you use symbols following the command, including any options it takes. For instance, to redirect the output of the echo command, you would type something like this:

```
$ echo $HISTCONTROL > histcontrol.txt
```

The result is that the file *histcontrol.txt* contains the output of the command. Redirection operators exist to achieve several effects, as summarized in the next slide:

Exploring Linux command-line tools

Redirection

Common Redirection Operators

Redirection Operator	Effect
>	Creates a new file containing standard output. If the specified file exists, it's overwritten.
>>	Appends standard output to the existing file. If the specified file doesn't exist, it's created.
2>	Creates a new file containing standard error. If the specified file exists, it's overwritten.
2>>	Appends standard error to the existing file. If the specified file doesn't exist, it's created.
&>	Creates a new file containing both standard output and standard error. If the specified file exists, it's overwritten.
<	Sends the contents of the specified file to be used as standard input
<<	Accepts text on the following lines as standard input.
<>	Causes the specified file to be used for both standard input and standard output.

Exploring Linux command-line tools

Redirection

PIPE (ALSO CALLED PIPELINE)

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

Exploring Linux command-line tools

Redirection

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.

Exploring Linux command-line tools

Redirection

Piping Data Between Programs (continued)

For instance, suppose that ***first*** generates some system statistics, such as system uptime, CPU use, number of users logged in, and so on. This output might be lengthy, so you want to trim it a bit.

You might therefore use ***second***, which could be a script or command that echoes from its standard input only the information in which you're interested. The grep command is often used in this role.

Pipes can be used in sequences of arbitrary length:

```
$ first | second | third | fourth | fifth | sixth [...]
```

Introduction to Linux Shell

Exploring Linux command-line tools

Processing Text Using Filters

Many simple commands are available to manipulate text. These commands accomplish tasks of various types, such as combining files, transforming the data in files, formatting text, displaying text, and summarizing data.

Combining Files with *cat*

The cat command's name is short for ***concatenate***, and this tool does just that: It links together an arbitrary number of files end to end and sends the result to standard output.

By combining ***cat*** with output redirection, you can quickly combine two files into one:

```
$ cat first.txt second.txt > combined.txt
```

Although ***cat*** is officially a tool for combining files, it's also commonly used to display the contents of a short file. If you type only one filename as an option, ***cat*** displays that file. This is a great way to review short files; but for long files, you're better off using a full-fledged pager command, such as *more* or *less*.

Exploring Linux command-line tools

Processing Text Using Filters

File-Transforming Commands (continued)

Sorting Files with *sort*

Sometimes you'll create an output file that you want sorted. To do so, you can use a command that's called, appropriately enough, ***sort***. This command can sort in several ways, including the following:

Ignore case Ordinarily, ***sort*** sorts by ASCII value, which differentiates between uppercase and lowercase letters. The -f or --ignore-case option causes sort to ignore case.

Month sort The -M or --month-sort option causes the program to sort by three-letter month abbreviation (JAN through DEC).

Numeric sort You can sort by number by using the -n or --numeric-sort option.

Exploring Linux command-line tools

Processing Text Using Filters

File-Transforming Commands (continued)

Sorting Files with *sort* (continued)

Reverse sort order The -r or --reverse option sorts in reverse order.

Sort field By default, sort uses the first field as its sort field. You can specify another field with the -k field or --key=field option. (The field can be two numbered fields separated by commas, to sort on multiple fields.)

As an example, suppose you wanted to sort listing1.1.txt by first name. You could do so like this:

```
$ sort -k 3 listing1.1.txt

555-2397 Beckett, Barry
555-5116 Carter, Gertrude
555-9871 Orwell, Samuel
555-7929 Jones, Theresa
```

The sort command supports a large number of additional options, many of them quite exotic. Consult sort's ***man*** page for details.

Exploring Linux command-line tools

Processing Text Using Filters

File-Viewing Commands

Sometimes you just want to view a file or part of a file. A few commands can help you accomplish this goal without loading the file into a full-fledged editor.

Viewing the start of files with *head*

Sometimes all you need to do is see the first few lines of a file. This may be enough to identify what a mystery file is, for instance; or you may want to see the first few entries of a log file to determine when that file was started. You can accomplish this goal with the ***head*** command, which echoes the first 10 lines of one or more files to standard output.

You can modify the amount of information displayed by head in two ways:

Specify the number of bytes The -c num or --bytes=num option tells head to display num bytes from the file rather than the default 10 lines.

Specify the number of lines You can change the number of lines displayed with the -n num or --lines=num option.

Exploring Linux command-line tools

Processing Text Using Filters

File-Viewing Commands (continued)

Viewing the end of files with *tail*

The ***tail*** command works just like head, except that tail displays the last 10 lines of a file. (You can use the -c/--bytes and -n/--lines options to change the amount of data displayed, just as with head.) This command is useful for examining recent activity in log files or other files to which data may be appended.

The tail command supports several options that aren't present in head and that enable the program to handle additional duties, including the following:

Track a file The -f or --follow option tells tail to keep the file open and to display new lines as they're added.

Some additional options provide more obscure capabilities. Consult tail's ***man*** page for details.

Exploring Linux command-line tools

Processing Text Using Filters

File-Viewing Commands (continued)

Paging through files with *less*

The ***less*** command's name is a joke; it's a reference to the ***more*** command, which was an early file pager. The idea was to create a better version of ***more***, so the developers called it ***less***.

The idea behind ***less*** (and ***more***, for that matter) is to enable you to read a file one screen at a time. When you type ***less filename***, the program displays the first few lines of filename. You can then page back and forth through the file:

- Pressing the **spacebar** moves forward through the file one screen at a time.
- Pressing **B** moves backward through the file one screen at a time.
- Pressing **D** moves forward through the file half a screen at a time.
- Pressing **U** moves backward through the file half a screen at a time.
- The **Up** and **Down** arrow keys move up or down through the file one line at a time.

Exploring Linux command-line tools

Processing Text Using Filters

File-Viewing Commands (continued)

Paging through files with *less* (continued)

- You can search the file's contents by pressing the slash (/) key followed by the search term. Typing **n** alone repeats the search forward, while typing N alone repeats the search backward.
- You can move to a specific line by typing g followed by the line number, as in **50g** to go to line 50.
- You can move to an approximate percentage position of the file by typing g followed by the line number, as in **50p** to go to the 50% of the file.
- **g** will take you to the beginning of the file, while **G** will take you to the end of the file.
- When you're done, type **q** to exit from the program.

Exploring Linux command-line tools

Processing Text Using Filters

File-Viewing Commands (continued)

Extracting Text with *cut*

The ***cut*** command extracts portions of input lines and displays them on standard output. You can specify what to cut from input lines in several ways:

By byte The -b list or --bytes=list option cuts the specified list of bytes from the input file. (The format of a list is described shortly.)

By character The -c list or --characters=list option cuts the specified list of characters from the input file.

By field The -f list or --fields=list option cuts the specified list of fields from the input file. By default, a field is a tab-delimited section of a line, but you can change the delimiting character with the -d char, --delim=char, or --delimiter=char option option, where char is the character you want to use to delimit fields.

Exploring Linux command-line tools

Processing Text Using Filters

File-Viewing Commands (continued)

Extracting Text with *cut* (continued)

The ***cut*** command is frequently used in scripts to extract data from some other command's output. For instance, suppose you're writing a script and the script needs to know the hardware address of an Ethernet adapter. This information can be obtained from the ifconfig command:

```
$ ifconfig eno2
eno2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.168.254  netmask 255.255.255.0  broadcast 192.168.168.255
        inet6 fe80::a6ba:dbff:fee1:4be7  prefixlen 64  scopeid 0x20<link>
        ether a4:ba:db:e1:4b:e7  txqueuelen 1000  (Ethernet)
        RX packets 23450459  bytes 2775486516 (2.5 GiB)
        RX errors 0  dropped 4  overruns 0  frame 0
        TX packets 2779622  bytes 365258758 (348.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 17
```

Exploring Linux command-line tools

Processing Text Using Filters

File-Viewing Commands (continued)

Extracting Text with *cut* (continued)

Unfortunately, most of this information is extraneous for the desired purpose. The hardware address is the 6-byte hexadecimal number following HWaddr. To extract that data, you can combine ***grep*** with ***cut*** in a pipe:

```
$ ifconfig eno2 | grep ether | cut -d" " -f10
a4:ba:db:e1:4b:e7
```

Of course, in a script you would probably assign this value to a variable or otherwise process it through additional pipes.

Exploring Linux command-line tools

Processing Text Using Filters

File-Viewing Commands (continued)

Obtaining a Word Count with *wc*

The ***wc*** command produces a word count (that's where it gets its name), as well as line and byte counts, for a file:

```
$ wc file.txt
308 2343 15534 file.txt
```

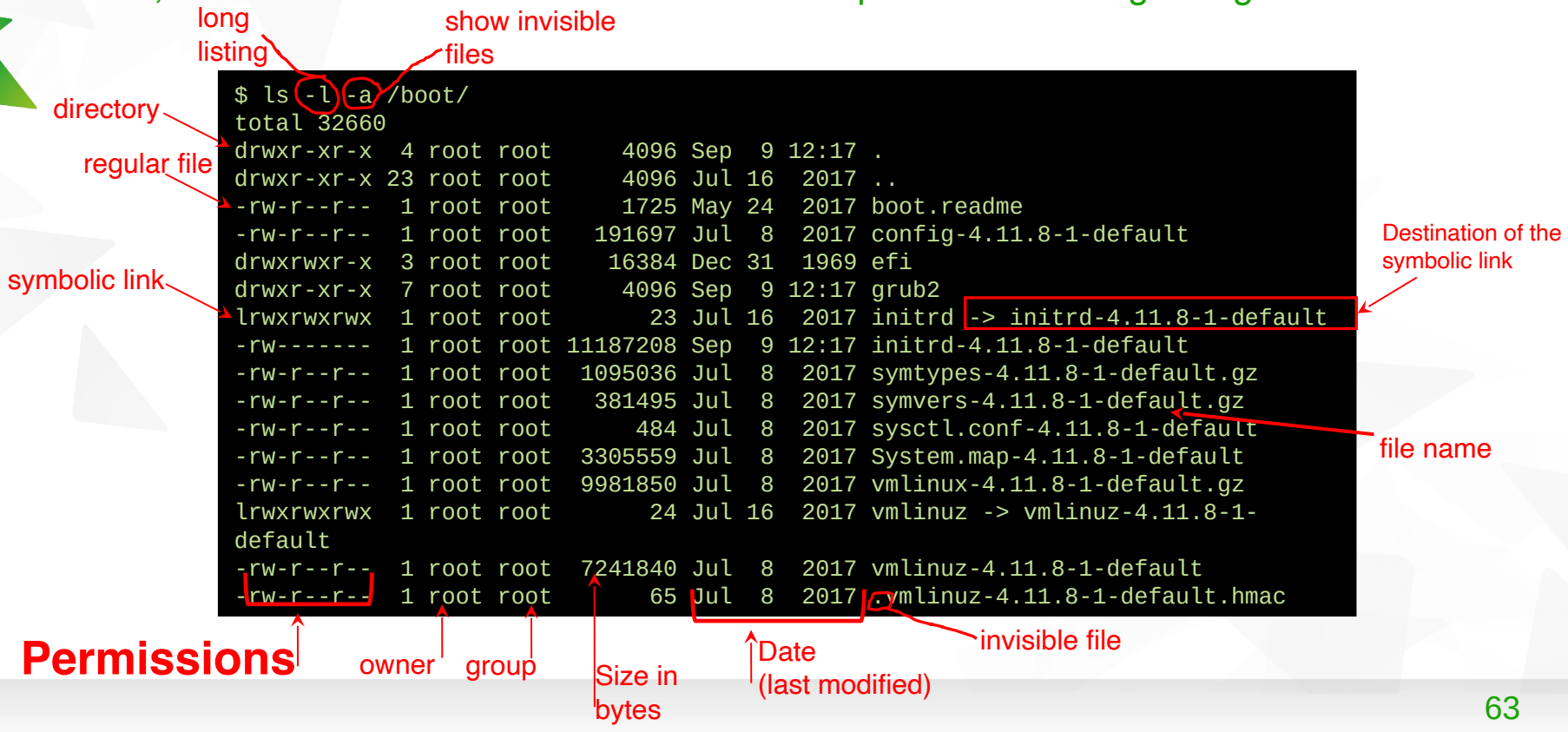
This file contains 308 lines (or, more precisely, 308 newline characters); 2,343 words; and 15,534 bytes. You can limit the output to the newline count, the word count, the byte count, or a character count with the --lines (-l), --words (-w), --bytes (-c), or --chars (-m) option, respectively. You can also learn the maximum line length with the --max-line-length (-L) option.

Introduction to Linux Shell

Exploring Linux command-line tools

Permissions

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



Exploring Linux command-line tools

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--

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

rwx

Exploring Linux command-line tools

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)

Exploring Linux command-line tools

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

Exploring Linux command-line tools

Permissions

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

Questions?

Introduction to Linux Shell