Lesson 2: Exploring Linux Command-Line tools

Objectives covered

- 103.1 Work on the command line (weight: 4)
- 103.2 Process text streams using filters (weight: 2)
- 103.4 Use streams, pipes, and redirects (weight: 4)
- 103.7 Search text files using regular expressions (weight: 3)
- 103.8 Basic file editing (weight: 3)

Process text streams using filters & Basic file editing

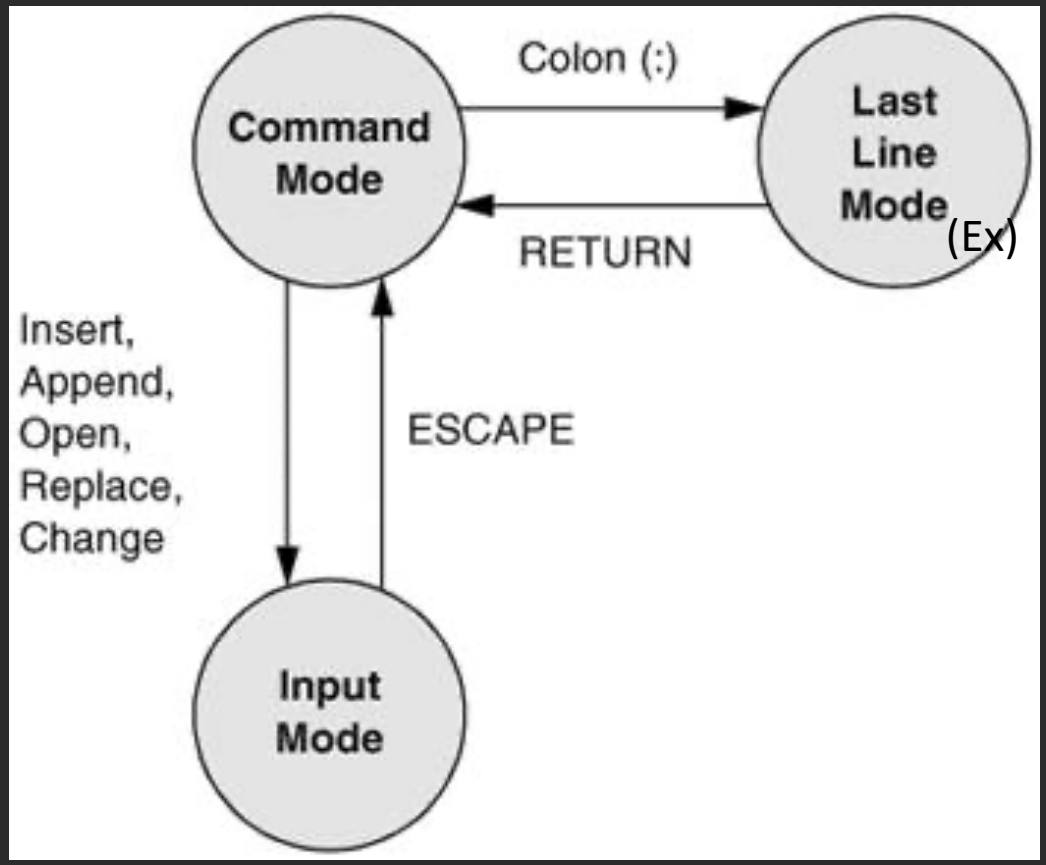
Editing text file:

emacs

nano

vim

Understanding vim:



Understanding vim:

Keystroke(s)	Description
h	Move cursor left one character.
l	Move cursor right one character.
j	Move cursor down one line (the next line in the text).
k	Move cursor up one line (the previous line in the text).

Keystroke(s)	Description
w	Move cursor forward one word to front of next word.
e	Move cursor to end of current word.
b	Move cursor backward one word.
^	Move cursor to beginning of line.
$	Move cursor to end of line.
gg	Move cursor to the file's first line.
G	Move cursor to the file's last line.
*n*G	Move cursor to file line number *n*.
Ctrl+B	Scroll up almost one full screen.
Ctrl+F	Scroll down almost one full screen.
Ctrl+U	Scroll up half of a screen.
Ctrl+D	Scroll down half of a screen.
Ctrl+Y	Scroll up one line.
Ctrl+E	Scroll down one line.

Understanding vim:

Keystroke(s)	Description
a	Insert text after cursor.
A	Insert text at end of text line.
dd	Delete current line.
dw	Delete current word.
i	Insert text before cursor.
I	Insert text before beginning of text line.
o	Open a new text line below cursor, and move to insert mode.
O	Open a new text line above cursor, and move to insert mode.
p	Paste copied text after cursor.
P	Paste copied (yanked) text before cursor.
yw	Yank (copy) current word.
yy	Yank (copy) current line.

Understanding vim:

Keystrokes	Description
`:! command`	Execute shell *command* and display results, but don't quit editor.
`:r! command`	Execute shell *command* and include the results in editor buffer area.
`:r file`	Read *file* contents and include them in editor buffer area.

Mode	Keystrokes	Description
Ex	:x	Write buffer to file and quit editor.
Ex	:wq	Write buffer to file and quit editor.
Ex	:wq!	Write buffer to file and quit editor (overrides protection).
Ex	:w	Write buffer to file and stay in editor.
Ex	:w!	Write buffer to file and stay in editor (overrides protection).
Ex	:q	Quit editor without writing buffer to file.
Ex	:q!	Quit editor without writing buffer to file (overrides protection).
Command	ZZ	Write buffer to file and quit editor.

Processing text file utilities:

cat [option] [file name]

Short	Long	Description
-A	--show-all	Equivalent to using the option -vET combination.
-E	--show-ends	Display a $ when a newline linefeed is encountered.
-n	--number	Number all text file lines and display that number in the output.
-s	--squeeze-blank	Do not display repeated blank empty text file lines.
-T	--show-tabs	Display a ^I when a tab character is encountered.
-v	--show-nonprinting	Display nonprinting characters when encountered using either ^ and/or M- notation.

Processing text file utilities:

od [option] [file name]

od can display file's content in octal (base 8), hexadecimal (base 16), decimal (base 10), and ASCII.

od can display content of non-text file

Processing text file utilities:

split [option] [Input [prefix]]

tr [option] SET1 [SET2]

sort [option] [file name]

nl [option] [file name]

Viewing text file utilities:

"less is more"

Viewing text file utilities:

Text file summary utilities:

wc [option] [file name]

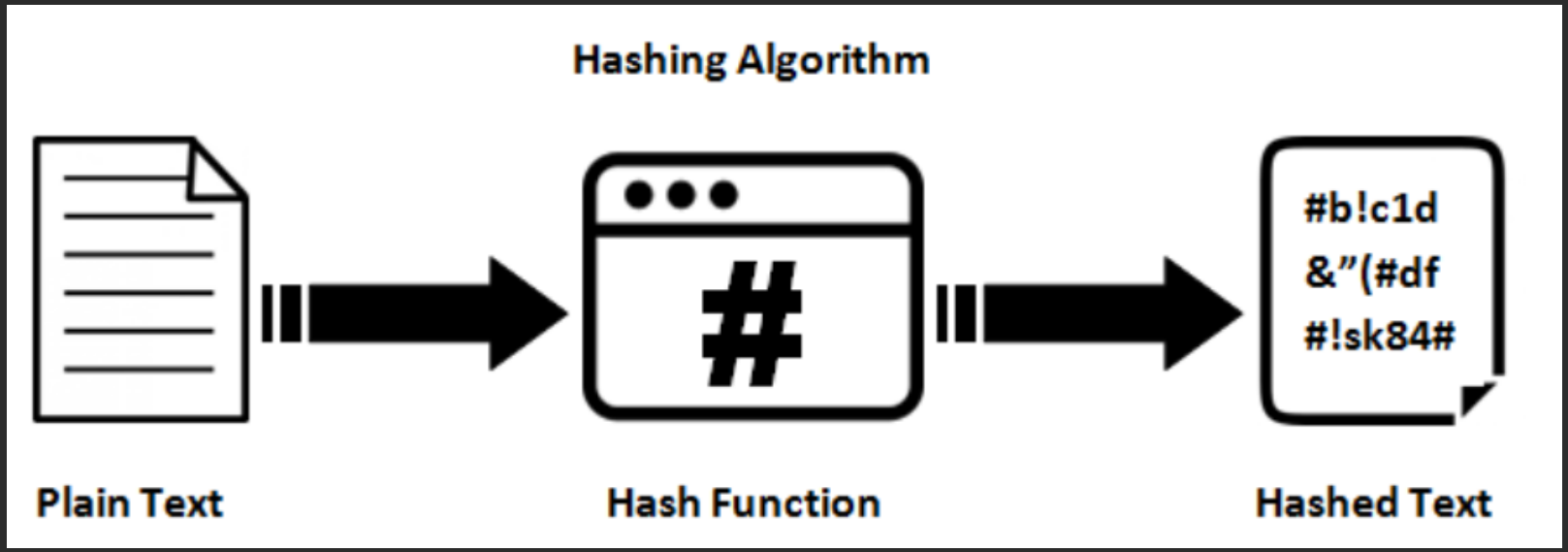
Short	Long	Description
-c	--bytes	Display the file's byte count.
-L	--max-line-length	Display the byte count of the file's longest line.
-l	--lines	Display the file's line count.
-m	--chars	Display the file's character count.
-w	--words	Display the file's word count.

Text file summary utilities:

cut [option] [file name]

Short	Long	Description
-c *nlist*	--characters *nlist*	Display only the record characters in the *nlist* (e.g., 1–5).
-b *blist*	--bytes *blist*	Display only the record bytes in the *blist* (e.g., 1–2).
-d *d*	--delimiter *d*	Designate the record's field delimiter as *d*. This overrides the Tab default delimiter. Put *d* within quotation marks to avoid unexpected results.
-f *flist*	--fields *flist*	Display only the record's fields denoted by *flist* (e.g., 1,3).
-s	--only-delimited	Display only records that contain the designated delimiter.
-z	--zero-terminated	Designate the record end-of-line character as the ASCII character NUL.

Hashing utilities:



Common hash tool: md5sum, sha256sum, sha512sum…

Search text files using regular expressions

Search for text with grep

grep [option] Pattern [file]

Short	Long	Description
-c	--count	Display a count of text file records that contain a *PATTERN* match.
-d action	--directories=action	When a file is a directory, if *action* is set to read, read the directory as if it were a regular text file; if *action* is set to skip, ignore the directory; and if *action* is set to recurse, act as if the - R, -r, or --recursive option was used.
-E	--extended-regexp	Designate the *PATTERN* as an extended regular expression.
-i	--ignore-case	Ignore the case in the *PATTERN* as well as in any text file records.
-R, -r	--recursive	Search a directory's contents, and for any subdirectory within the original directory tree, consecutively search its contents as well (recursively).
-v	--invert-match	Display only text files records that do *not* contain a *PATTERN* match.

Regular Expressions (RegEx)

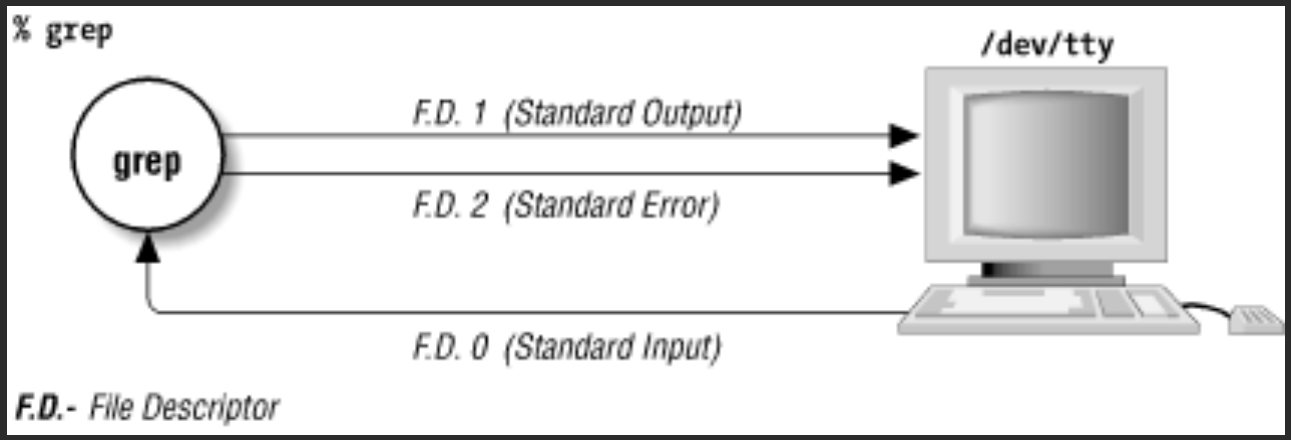
Character	Meaning	Example
*	Match **zero, one or more** of the previous	`Ah*` matches "Ahhhhh" or "A"
?	Match **zero or one** of the previous	`Ah?` matches "Al" or "Ah"
+	Match **one or more** of the previous	`Ah+` matches "Ah" or "Ahhh" but not "A"
\	Used to **escape** a special character	`Hungry\?` matches "Hungry?"
.	Wildcard character, matches **any** character	`do.*` matches "dog", "door", "dot", etc.
( )	**Group** characters	See example for |
[ ]	Matches a **range** of characters	`[cbf]ar` matches "car", "bar", or "far" `[0-9]+` matches any positive integer `[a-zA-Z]` matches ascii letters a-z (uppercase and lower case) `[^0-9]` matches any character not 0-9.
|	Matche previous **OR** next character/group	`(Mon)|(Tues)day` matches "Monday" or "Tuesday"
{ }	Matches a specified **number of occurrences** of the previous	`[0-9]{3}` matches "315" but not "31" `[0-9]{2,4}` matches "12", "123", and "1234" `[0-9]{2,}` matches "1234567..."
^	**Beginning** of a string. Or within a character range `[]` negation.	`^http` matches strings that begin with http, such as a url. `[^0-9]` matches any character not 0-9.
$	**End** of a string.	`ing$` matches "exciting" but not "ingenious"

Character classes

Class	Description
[:alnum:]	Matches any alphanumeric characters (any case), and is equal to using the [0-9A-Za-z] bracket expression
[:alpha:]	Matches any alphabetic characters (any case), and is equal to using the [A-Za-z] bracket expression
[:blank:]	Matches any blank characters, such as tab and space
[:digit:]	Matches any numeric characters, and is equal to using the [0-9] bracket expression
[:lower:]	Matches any lowercase alphabetic characters, and is equal to using the [a-z] bracket expression
[:punct:]	Matches punctuation characters, such as !, #, $, and @
[:space:]	Matches space characters, such as tab, form feed, and space
[:upper:]	Matches any uppercase alphabetic characters, and is equal to using the [A-Z] bracket expression

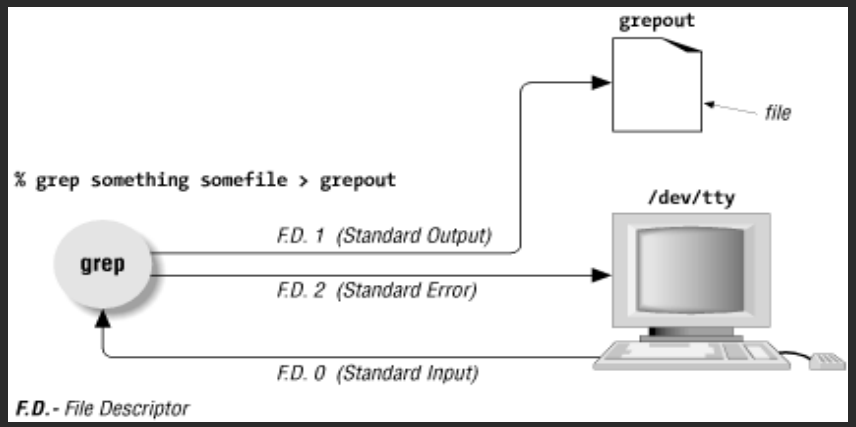
Use streams, pipes and redirects

Standard I/O file descriptors



Name	File descriptor	Description	Abbreviation
Standard input	**0**	The default data stream for input, for example in a command pipeline. In the terminal, this defaults to keyboard input from the user.	**stdin**
Standard output	**1**	The default data stream for output, for example when a command prints text. In the terminal, this defaults to the user's screen.	**stdout**
Standard error	**2**	The default data stream for output that relates to an error occurring. In the terminal, this defaults to the user's screen.	**stderr**

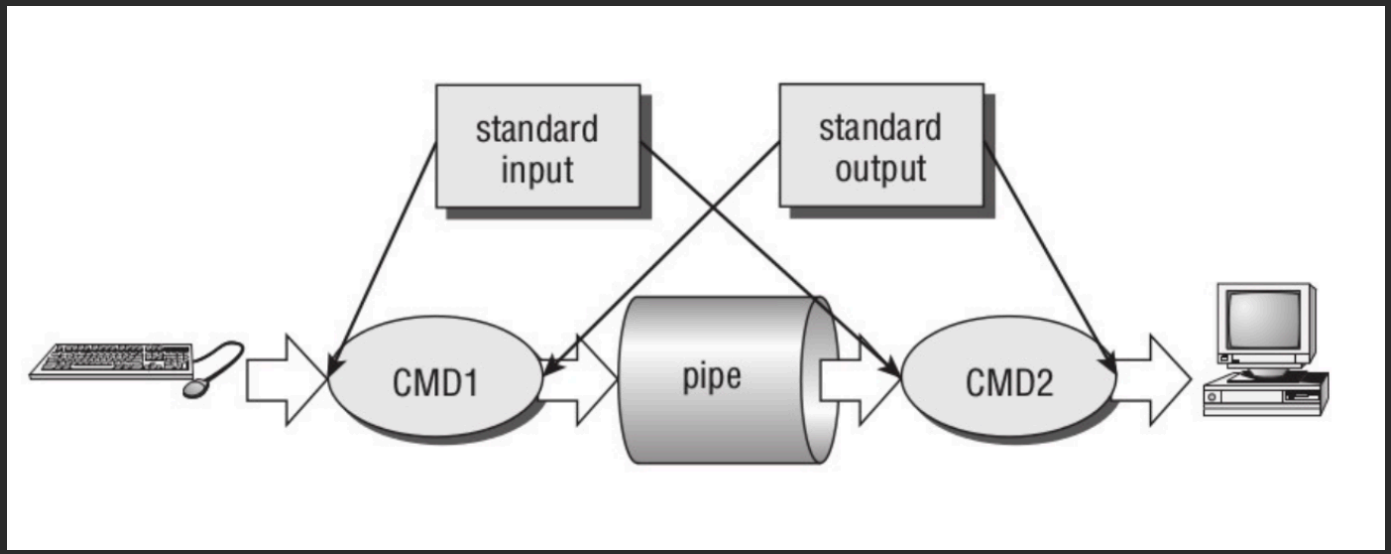
F.D. Redirection



Operator	Description
>	Redirect STDOUT to specified file. If file exists, overwrite it. If it does not exist, create it.
>>	Redirect STDOUT to specified file. If file exists, append to it. If it does not exist, create it.
2>	Redirect STDERR to specified file. If file exists, overwrite it. If it does not exist, create it.
2>>	Redirect STDERR to specified file. If file exists, append to it. If it does not exist, create it.
&>	Redirect STDOUT and STDERR to specified file. If file exists, overwrite it. If it does not exist, create it.
&>>	Redirect STDOUT and STDERR to specified file. If file exists, append to it. If it does not exist, create it.
<	Redirect STDIN from specified file into command.
<>	Redirect STDIN from specified file into command and redirect STDOUT to specified file.

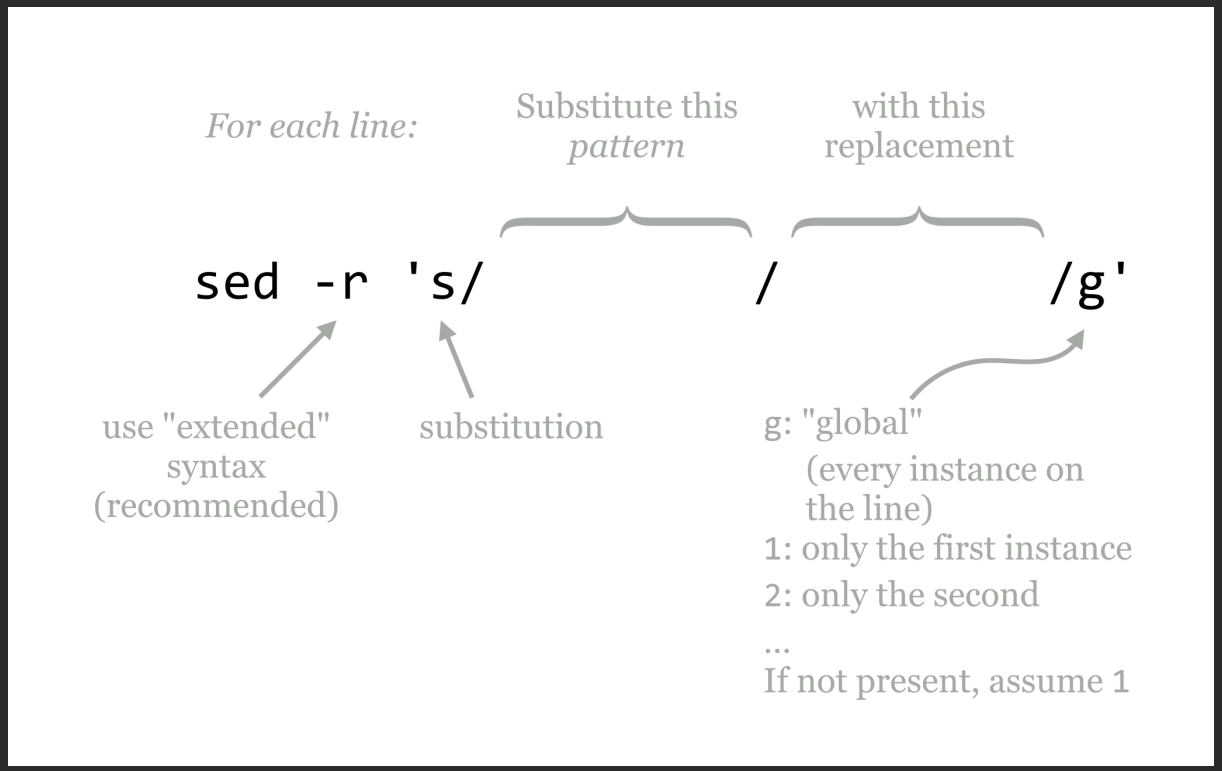
Piping data between commands

Command 1 | Command2 [| Command 3]…



Stream editor = sed

sed [option] [script] [file]…



sed' commonly used options

Short	Long	Description
-e *script*	--expression=script	Add commands in *script* to text processing. The *script* is written as part of the sed command.
-f *script*	--file=script	Add commands in *script* to text processing. The *script* is a file.
-r	--regexp-extended	Use extended regular expressions in script.

```
$ cat cake.txt
Christine likes chocolate cake.
Rich likes lemon cake.
Tim only likes yellow cake.
Samantha does not like cake.
```

```
$ sed -e 's/cake/donuts/ ; s/like/love/' cake.txt
Christine loves chocolate donuts.
Rich loves lemon donuts.
Tim only loves yellow donuts.
Samantha does not love donuts.
```

Before ***After***

Generating command line

Using xargs

```
$ touch EmptyFile1.txt EmptyFile2.txt EmptyFile3.txt
$
$ ls  EmptyFile?.txt
EmptyFile1.txt  EmptyFile2.txt  EmptyFile3.txt
$
$ ls -1 EmptyFile?.txt | xargs -p /usr/bin/rm
/usr/bin/rm EmptyFile1.txt EmptyFile2.txt EmptyFile3.txt ?...n
```

Using $()

```
$ rm -i $(ls EmptyFile?.txt)
rm: remove regular empty file ‘EmptyFile1.txt’? y
rm: remove regular empty file ‘EmptyFile2.txt’? y
rm: remove regular empty file ‘EmptyFile3.txt’? y
```

Question...