Viewing the Contents of Files

Commands to view but not modify file contents

Copyright 2009 Stewart Weiss

Viewing files

- Several commands let you view all or part of one or more files, including: **`cat`**, **`more`**, **`less`**, **`head`**, and **`tail`**.
- The most useful of these are **`more`** and **`less`**, more or less. They display files a screen-full at a time, and wait for you to enter a command before they show you more of the file.
- In contrast, the **`cat`** command displays the entire file, at lightning speed, so that all you really end up seeing is the last screenful of the file.
- **`head`** and **`tail`** are specialized and you'll see they can be very useful too.

Viewing files with **more**

- The **more** command displays the files whose pathnames are given as its arguments, one page (page = screen) at a time. For example

  more /etc/passwd /etc/services

  displays the **passwd** file followed by the **services** file, both in **/etc**. Each file is displayed approximately one screen at a time. **more** uses the **TERM** environment variable to determine how many columns and lines are in the terminal.
- **more** is useful primarily for viewing large files.

Filtering output through `more`

- `more` is also useful as the last command in a pipe, when a command has lots of output. For example, the `ps -ef` command displays a list of all processes (running programs) in the system. It can produce many more lines of output than can fit on a screen. Piping its output through `more` lets us read the output more carefully, since

  ```
  ps -ef | more
  ```

  produces a page at a time.

Control within `more`

- While you are viewing a file with **`more`**, some keystrokes do useful things. There are many, and you should read the **`more`** man page if you want to know more than just the few mentioned here.

spacebar	displays next page of text
return key	displays one more line
`b`	skips backwards one screen of text
`/`*pattern*	searches forward for next occurrence of ***pattern***
`?`*pattern*	searches backward for next occurrence of ***pattern***

More commands within **more**

n	repeat search for ***pattern*** already given
q	quit (before reaching end)

- There are various command line options to **more**, and there are other more advanced features. I do not recommend learning these, because **more** has been replaced by a more powerful command called **less**.

Doing more with `less`

- The **`more`** command does not let you go backwards in a file very easily, and it also takes a long time to start up if the file is very large.
- The **`less`** command does more than **`more`**: it allows forward and backward motion through a file very easily.
- **`less`** works the same way as **`more`**; you give it a list of file names on the command line, or you can use it as the right hand side of a pipeline, as in:

```
less bigfile /etc/passwd /etc/services
ps -ef | less
```


Basic operations in `less`

- The **`less`** command includes most of the commands in **`more`**. All of the ones that I listed for **`more`** work the same way in **`less`** as they do in **`more`**.
- **`less`** also includes many other ways to navigate through a file. The next slide lists some of the basic, but useful commands not described already.

Doing more with `less`

- The **`more`** command does not let you go backwards in a file very easily, and it also takes a long time to start up if the file is very large.
- The **`less`** command does more than **`more`**: it allows forward and backward motion through a file very easily.
- **`less`** works the same way as **`more`**; you give it a list of file names on the command line, or you can use it as the right hand side of a pipeline, as in:

```
less bigfile /etc/passwd /etc/services
ps -ef | less
```


Commands within `less`

- In the following, *N* is a number you type before the letter. If you omit the number, `less` uses its default value.

***N*d**	scroll forward *N* lines (½ page by default)
***N*u**	scroll backward *N* lines (½ page by default)
***N*g**	go to line *N* in the file (top by default)
***N*%**	go to the position that is *N*% of the file (top)
:n	go to next file, if more than one
:p	go to previous file, if more than one

Viewing the beginning of a file: **head**

- The **head** command displays the first 10 lines of a file.

  ```
  head Hamlet
  ```

  will display the first 10 lines of **Hamlet**. You can change the number of lines to *N* by typing **head -*N***, as in

  ```
  head -20 Hamlet
  ```

  which will display the first 20 lines instead.
- **head** has other options, which you can read about in the its man page.

Viewing the end of a file: `tail`

- The **`tail`** command displays the ***last*** 10 lines of a file.

  ```
  tail TwoCities
  ```

  will display the last 10 lines of **`TwoCities`**. You can change the number of lines to ***N*** by typing **`tail -`*N***, as in

  ```
  tail -20 TwoCities
  ```

  which will display the first 20 lines instead.
- **`tail`** has other options, which you can read about in the its man page.

Printing a range of lines with **sed**

- The **sed** command is one of many UNIX filter programs, which we will learn about soon. In this context though, one simple use of it is to display a range of consecutive lines in a file:

```
sed -n '5,10p' TwoCities
```

will display the lines 5 through 10 of **TwoCities**. In general,

```
sed -n 'N,Mp' filename
```

which will display lines **N** through **M** of the file named **filename.**

The `cat` command

- The **`cat`** command displays the contents of the files listed on its command line without any modification.

  `cat verse1 refrain verse2 refrain`

  displays **`verse1`** then **`refrain`** then **`verse2`** then **`refrain`** on the screen.
- **`cat`** *concatenates* the files onto the screen. Unlike **`more`** and **`less`**, it does not pause per screen full, so all you see is the last page of the last file.
- Why, then, would you want to use it?
- It is most useful when using I/O redirection, as you will now see.

Using `cat` without Arguments

- If `cat` has no command line arguments, it takes its input from the keyboard. Thus, if I type `cat` by itself, it will echo each line that I type:

```
$ cat
hello
hello
bye
bye
^D
$
```

- This is not very interesting in itself, right?

Using `cat` to create a file

- What if I redirect its output to a file?

```
$ cat > instantnotes
Some thoughts I had just now --
before it's too late.
^D
$
```

- Nothing appears on the screen other than what I type. If I examine the file **`instantnotes`**, I will see what I typed.

```
$ cat instantnotes
Some thoughts I had just now --
before it's too late.
$
```

More about `cat`

- `cat` has one other important use, but it will be discussed after we learn about other forms of I/O redirection. If you are interested now, try to find something about "here documents".

Comparing files: `diff`

- **`diff`** is a command that can compare two files, or even two directories. In its simplest form, you can compare two text files like this:

```
diff file1 file2
```

and it they are identical it will produce NO output. If they are different, it will produce a set of instructions that can be used to make them the same. If you do not care about making them the same, just use

```
diff -q file1 file2
```

which will report only that they are different if they are, or nothing if they are not.

More about `diff`

- `diff` can be given options to ignore case (`-i`), blank lines (`-B`), the amount of white space (`-b`), and more:

```
diff -ibB file1 file2
```

will report that the two files are the same ignoring blank lines, changes in case, and the amount of white space between words. Read the man page for more interesting options.

The `file` command

- The **`file`** command attempts to guess the type of a file. For example, you may want to know if a file is a shell script, or an executable file, or a device file, or some other kind of special file. If you type

  ```
  file filename
  ```

  the **`file`** command will display information about the file whose name is supplied.

- For example,

  ```
  $ file .bashrc
  .bashrc:  ASCII text
  ```

More `file` examples

- These are some of the different types of descriptions that the file command will provide

```
mbox:              ASCII mail text
misc:              directory
perlstuff:         POSIX tar archive
/usr/lib/liby.a:   current ar archive
/dev/tty0:         character special (4/0)
```

- It classifies files by looking at their content, so it needs read access to the files.

More about the `file` command

- The **`file`** command can also determine the form of an executable file in UNIX.

```
$ file chtime
```

```
chtime:   ELF 32-bit LSB executable, Intel 80386,
version 1 (SYSV), for GNU/Linux 2.2.5, dynamically
linked (uses shared libs), not stripped
```

- This shows that **`chtime`** can run on an Intel 80836 or compatible ship, that it is a 32-bit executable, among other things. You will need to know something about computer architecture to understand most of this.