Filtering Tools

Filtering Tools and a Bit More About I/O Redirection

Input redirection operator **<**

- The ***input redirection operator*** '**<**', replaces the standard input of a command by the file that follows it. It is not very useful for many commands because many commands let you put the names of files on the command line as their arguments, so there is no need to use this operator for most commands.

```
cat < myfile
```

- for example, is the same as

```
cat myfile
```

Other redirection operators

- There are two other redirection operators, '**<<** ' and '**>>**'. We will start with '**>>**':
- The ***output append operator***, **>>**, which you saw in an earlier lesson, forces standard output to be ***appended*** to a file instead of replacing the file's contents. This is very useful; it gives you the means to add new lines to files.

An example using >>

- To illustrate, suppose that **story** is a file with a single line of text as shown below. We'll use **>>** to lengthen the story.

```
$ cat story
Once upon a time
$ echo "there was no UNIX." >> story
$ cat story
Once upon a time
there was no UNIX.
$
```

- You can see that the output of **echo** was added as a new line in **story**, instead of replacing its contents.

Another example of **>>**

- There are many system programs that maintain log files, i.e., files that record the program's activities. Many of these programs are shell scripts. When they need to append information to their log files, they use the append operator.
- I could create a log file to record whenever I started a bash shell by adding this to my **`.bashrc`** file:

```
echo "bash started on `tty` at `date`" >> ~/.mylog
```

- This would add a line to the file **`.mylog`** in my home directory each time I ran a bash shell. The expression **`` `...` ``** is explained next.

And another example

- I keep track of my own logins by adding the following line to my .**bash_profile**:

```
echo "Login into `hostname` on `date`" >> .loghist
```

- The **.bash_profile** file is executed only on logins, not when a non-login shell runs, so this does not add a line each time I start a shell.
- This line appends the name of the host and the time and date of login to my **.loghist** file. The ***backquoted expressions*** **`hostname`** and **`date`** cause the **hostname** and **date** commands to run and their output to be substituted for their backquoted names.

The pipe operator `|`

- We have been using the ***pipe operator***, '`|`', already so this is a review. The pipe operator connects the output of the command on its left to the input of the command on its right. It is a very powerful tool. For example

```
  ypcat passwd | awk -F: '{print $5}'
```

displays the real names of everyone with an account on the computer.

```
  man -k | grep ' file ' | grep ' write '
```

looks for commands with the words **`file`** and **`write`** in the description.

Pipes connecting filters

- Commands can be composed upon each other using the pipe operator. The power of pipes was demonstrated in the first lesson with the following example:

```
who | awk '{print $1}' | sort | uniq
```

- In this example, the output of **`who`** is the input of **`awk`**, whose output is the input of **`sort`**, whose output is the input of **`uniq`**. The output is a list of usernames of people currently logged in, one per line. (There is a system command, **`users`**, that creates this list also, so we really didn't need to do all that work!)

Filters

- The **`awk`**, **`sort`**, and **`uniq`** commands are examples of a class of UNIX programs called ***filters***.
- ***A filter is a UNIX command whose input and output are ordinary text, and that expects its input from standard input and puts its output on standard output.***
- Filters transform their input in some way, such as by sorting it, removing words or lines based on a pattern or on their position in the line or file (e.g., remove every 3rd word in a line, or remove every 4th line, or remove any line that has a curse word.)

Filtering standard input

- Filters may have file name arguments on the command line, but when they have no arguments, they read from standard input (the keyboard) instead:

```
grep  'a clue'  thehouse
```

searches for '**`a clue`**' in **`thehouse`**, whereas in

```
cat  thehouse | grep 'a clue'
```

`grep` searches through its standard input stream for lines with '**`a clue`**'.

Some useful filters (1)

- Some of the most useful filters are:

`grep` family: **`grep, egrep, fgrep`**	global regular expression parsers
`sort`	sorts based on several criteria
`uniq`	removes adjacent identical lines
`awk`	full-fledged programming language for field-oriented pattern matching
`cut`	removes pieces of each line based on positions

More filters

- These are filters too:

`head, tail`	display just top or bottom lines of files
`cat`	null filter -- shows everything in order
`tac`	shows lines in reverse order
`fold -w<N>`	display output in width of N columns
`sed`	stream editor -- very powerful filter
`wc`	not exactly a filter, display count of chars, words, and lines

Selected filters

- In the next chapter you will learn about patterns (called regular expressions). Then you will learn how to use the family of filters that use these patterns.

- First we will look a few of the filters that do not require patterns as arguments.

Selected filters: **`sort`**

- The **`sort`** program can be used for sorting one or more text files in sequence. It can also merge files.

  In its simplest form,

  ```
  sort filename
  ```

  will sort **`filename`** using the first field of the line (first chars up to the first white space) in ASCII collating order, displaying the sorted file on the screen (standard output).
- **`sort`** will ignore case by default in some versions of UNIX, whereas in others, uppercase and lowercase letters will be treated as different.

Selected filters: **`sort`**

- The **`sort`** program will not sort numbers properly unless you tell it to sort numerically. It treats them like letters, by default. For example, if **`scores`** contains the two lines

```
5
10
```

then we get

```
$ sort scores
10
5
```

because "**`1`**" precedes "**`5`**", so "**`10`**" precedes "**`5`**".

More on Sorting

- To sort numerically, you give **`sort`** the **`-n`** flag:

```
sort -n filename
```

as in:

```
$ sort -n scores
5
10
```

- To reverse the order of the sort use the **`-r`** flag

Sorting by fields

- To sort using the second field of the line,

```
sort +1 filename
```

- To sort using the third field of the line,

```
sort +2 filename
```

and so on.

- To use the first field as the primary key, then the second field as secondary key, use

```
sort +0  +1  filename
```

Selected filters: `uniq`

- This is the last filter in that pipeline I showed you earlier. The **`uniq`** command removes a line from a file if it is identical to the one preceding it.
- If you sort a file that has duplicate lines, and pipe it through **`uniq`**, the duplicates will be removed.
- You could do the same thing by using the **`-u`** option with **`sort`** though, so this is not why **`uniq`** is unique.
- Sometimes there are files that are not sorted but have "runs" of the same lines. You could sort them using **`sort -u`**, but it is fast to just run **`uniq`** on them.

Selected filters: `fold`

- The **`fold`** filter breaks each line at a fixed number of characters, so that each line is at most a certain length.

```
fold -w8  myfile
```

will break each line of **`myfile`** into 8-character wide lines except those with fewer characters.

- Suppose **`dnastring`** has the line

```
agatggcggc
```

`fold -c4 dnastring` produces

```
agat
ggcg
gc
```

Selected Filters: `wc`

- The `wc` command, by default, displays the numbers of lines, words, and characters in one or more files given on the command line, or in its input stream if not given any arguments.

```
$ wc /etc/passwd
  55 99 2678 /etc/passwd
```

  tells me how many lines, words, and characters are in the passwd file.
- I can give it `-m`, `-w`, or `-l` for chars, words, and lines to restrict its output.

Uses of **wc**

- For example

```
$ who | wc -l
 2
```

displays the number of users currently logged in, and

```
$  ps -ef | grep bash | wc -l
```

displays how many people are running bash at the moment.

Things to try

- Read the man pages for **`sort`**, **`fold`**, and **`wc`**. Familarize yourself with them because they are very useful.
- Read about **`cut`**; it is also useful.
- There is a file in the **`cs132/data`** directory named **`directors`**. Sort it by movie title, then by director last name. Print out just the movie titles. Print out just the director names.
- Download any spreadsheet and save as a CSV (comma separated values) file, using tabs to separate the fields. Experiment with sort to see how you can sort this file in different ways.