Chapter 1B, Filtering, Redirection, and Data Munging

John M. Morrison

January 4, 2022

Contents

0 Redirection

UNIX treats everything in your system as a file; this includes all devices such as printers, the screen, and the keyboard. Things put to the screen are generally put to one of two files, `stdout`, or *standard output* and `stderr`, or standard error. You will see that it is very easy to redirect `stdout` and `stderr` to different places.

The keyboard, by default, is represented by the file `stdin`, or *standard input*. It is also possible to redirect standard input and take standard input from a file.

UNIX filters, such as `cat` and `more` have as their default input `stdin` and as output `stdout`. This section will show you how to redirect these to files.

The examples here are based on the files `animalNoises.txt`; make them and follow along.

```
miao
bleat
moo
```

and `physics.txt`

```
snape
benettron
stephan
```

First we show how `cat` puts files to `stdout`.

```
unix> cat animalNoises.txt physics.txt
miao
bleat
moo
snape
benettron
stephan
```

Now let us capture this critical information into the file `stuff.txt` by redirecting `stdout`. We then use `cat` to display the resulting file to `stdout`.

```
unix> cat animalNoises.txt physics.txt > stuff.txt
unix> cat stuff.txt
miao
bleat
moo
snape
benettron
stephan
```

The `cat` command has a second guise. It accepts a file name as an argument, but it will also accept standard input; this is no surprise since `stdin` is treated as a file. At the UNIX command line enter

```
unix> cat
```

The `cat` program is now running and it awaits word from `stdin`. Enter some text and then hit the enter key; cat echoes back the text you type in. To finish, hit control-d (end-of-file).

```
unix> cat
me too
```

```
me too
ditto
ditto
unix>
```

The control-d puts no response to the screen. You can also put a file to the screen with

```
unix> cat < someFile
```

This is, in fact what happens behind the scenes when you use a file as an argument to a UNIX filter. It treats that file as `stdin`. In this example, the file someFile becomes `stdin` for the cat command.

This phenomenon is shown in the man page for cat. Under the description of the command it says, " Concatenate FILE(s), or standard input, to standard output."

Let us now come back to our examples. Next create a new file named `sheck.txt` with these contents.

```
roach
stag beetle
tachnid wasp
```

Were we to invoke the command

```
unix> cat animalNoises.txt physics.txt > sheck.txt
```

we would clobber the file `sheck.txt` and lose its valuable contents. This may be our intent; if so very well. If we want to add new information to the end of the file we use the `>>` *append operator* to append it to the end of the receiving file. If we do this

```
unix> cat animalNoises.txt physics.txt >> sheck.txt
```

we get the following result if we use the original file `sheck.txt`.

```
unix> cat sheck.txt
roach
stag beetle
tachnid wasp
miao
bleat
moo
```

```
snape
benettron
stephan
```

The >> redirection operator will automatically create a file for you if the file to which you are redirecting does not already exist.

0.1 An error of Omission

But what about that Cinderella `stderr`? Watch this.

```
unix> mkdir tossme
unix> cd tossme
unix> ls
unix> touch a b c d e
unix> ls f* 2> errors.txt
unix> ls
a          b          c          d          e          errors.txt
unix> cat errors.txt
ls: f*: No such file or directory
```

You can redirect `stderr` by using the two-funnel `2>`. Here is another cool trick.

```
unix> ls -l a b g >items.txt 2> dumb.txt
unix> cat items.txt
-rw-r--r--  1 morrison  1671898719     0B Jan  4 14:19 a
-rw-r--r--  1 morrison  1671898719     0B Jan  4 14:19 b
unix> cat dumb.txt
ls: g: No such file or directory
unix>
```

We separated the strems `stderr` and `stdout` into separate files.

Programming Exercise What does `2»` do?

1 More Filters

It is very common to want to use `stdout` from one command to be `stdin` for another command. This will grant us the ability to chain the actions of the existing filters we have `cat`, `more` and `less` with some new filters to do a wide variety of tasks To achieve this tie, we use a device called a *pipe*. Pipes allow you to chain the action of various UNIX commands. We shall add to our palette

of UNIX commands to give ourselves a larger and more interesting collection of examples. These commands are extremely useful for manipulating files of data.

1.1 The sort filter

Bring up the man page for the command sort. This command accepts a file (or stdin) and it sorts the lines in the file.

This begs the question: how does it sort? It sorts alphabetically in a case-insensitive manner, and it "alphabetizes" non-alphabetical characters by ASCII value. The sort command several four helpful options.

-b	-ignore-leading-blanks	ignores leading blanks
-d	-dictionary-order	pays heed to alphanumeric characters and blanks and ignores other characters
-f	-ignore-cases	ignores case
-r	-reverse	reverses comparisons

Here we put the command to work with stdin; use a control-d on its own line to get the prettiest format. Here we put the items moose, jaguar, cat and katydid each on its own line into stdin. Without comment, a sorted list is produced.

```
unix> sort -f
moose
jaguar
cat
katydid        (now hit control-d)
cat
jaguar
katydid
moose
unix>
```

You should try various lists with different options on the sort command to see how it works for yourself. You can also run sort on a file and send a sorted copy of the file to stdout. Of course, you can redirect this result into a file using > or ».

Programming Exercises

1. Enter sort -n at the command prompt. Enter some numbers, one to a line, then hit control-d. What happens?

2. Enter `sort -r` at the command prompt. Enter some names, one to a line, then hit control-d. What happens?
3. Can you effectively combine the action of the last two items?

1.2 The Filters `head`, `tail`, and `uniq`

The filters `head` and `tail` put the top or bottom of a file to `stdout`; the default amount is 10 lines. To show the first 5 lines of the file foo.txt, enter the following at the UNIX command line.

```
unix> head -5 foo.txt
```

You can do exactly the same thing with tail with an entirely predictable result. The command `uniq` weeds out consecutive duplicate lines in a file, leaving only the first copy in place. These three commands have many useful options; explore them in the man pages.

1.3 The `grep` Filter

This command is incredibly powerful; here we will just scratch the surface of its protean powers. You can search and filter files using `grep`; it can be used to search for needles in haystacks. In its most basic form grep will inspect a file line-by-line and put all lines to `stdout` containing a specified string. Here is a sample session.

```
unix> grep gry /usr/share/dict/words
angry
hungry
unix>
```

The file `/usr/share/dict/words` is a dictionary file containing a list of words, one word to a line in (mostly) lower-case characters. Here we are searching the dictionary for all lines containing the character sequence gry; the result is the two words angry and hungry. There are options `-f` and `-ignore-case` to ignore the case of alphabetical characters.

1.4 Serving up Delicious Data Piping Hot

Pipes allow you to feed `stdout` from one command into `stdin` to another without creating any temporary files yourself. Pipes can be used along with redirection of `stdin` and `stdout` to accomplish a huge array of text-processing chores.

Now let us do a practical example. Suppose we want to print the first 5 lines alphabetically in a file named `sampleFile.txt`. We know that sort will sort the file asciicographically; we will use the `-f` option to ignore case. The command `head -5` will print the first five lines of a file passed it or the first five lines of `stdin`. So, what we want to do is sort the file ignoring case, and pass the result to head -5 to print out the top five lines. You join two processes with a pipe; it is represented by the symbol `|` , which is found by hitting the shift key and the key just above the enter key on a standard keyboard. Our command would be

```
unix> sort -i sampleFile.txt | head -5
```

The pipe performs two tasks. It redirects the output of `sort -f` into a temporary buffer and then it feeds the contents of the buffer as standard input to `head -5`. The result: the first five lines in the alphabet in the file `sampleFile.txt` are put to `stdout`.

Suppose you wanted to save the results in a file named `results.txt`. To do this, redirect `stdout` as follows

```
unix> (sort -i sampleFile.txt | head -5) > results.txt
```

Note the use of defensive parentheses to make our intent explicit. We want the five lines prepared, then stored in the file `results.txt`.

Programming Exercises Here are two more filters, `wc` and a command `echo`. You will use the `man` pages to determine their action and to use them to solve the problems below.

1. Tell how to put the text "Cowabunga, Turtle soup!" to `stdout`.
2. Tell how to get the text "This is written in magic ink" into a text file without using a text editor of any kind.
3. The `ls` command has an option `-R`, for "list files recursively." This lists all of the sub-directories and all of their contents within the directory being listed. Use this command along with grep to find a file containing a specified string in a file path.
4. Put a list of names in a file in lastName, firstName format. Put them in any old order and put in duplicates. Use pipes to eliminate duplicates in this file and sort the names in alphabetical order.
5. Find the word in the system dictionary occupying line 10000.
6. How do you count all of the words in the system dictionary containing the letter `x`?
7. Find all words in the system dictionary occupying lines 50000-50500.
8. Tell how, in one line, to take the result of the previous exercise, place it in reverse alphabetical order and store in in a file named `myWords.txt`.