Chapter 1B, Filtering, Redirection, and Data Munging

John M. Morrison

January 9, 2022

Contents

0 Introduction

Now the rubber will meet the road. We will begin to develop some very powerful tools for cleaning up and making sense of data and for finding needles in gigantic haystacks.

We will begin with the redirection of `stdin`, `stdoout`, and `stderr` to files. This will give us a place to store things as we process them and it will keep error messages from cluttring up our data files.

You will then meet an array of UNIX filters that manipulate files in various ways. You will learn how to chain their actions to achieve some extremely useful effects. All of this can be done with surprisingly little code.

1 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`. 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 recall 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.

1.1 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?

2 Pipes and `sort`

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.

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?

3 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.

3.1 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.

3.2 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`.

4 Supercharging `grep`

We are going to learn a little language for specifying patterns in text that you wish to search for or filter into or out of collection. The first step in this process is to understand *character classes*, which represent a wildcard for a single character.

Here are examples of classes of characters we might like to represent with a wildcard.

- an alphabetical character
- a punctuation mark
- a decimal digit
- lower–case letters
- upper–case letters
- any old list of characters you'd like to create

- any whitespace character

The simplest character class is just a character by itself. For example, the character `a` is the character class standing for the character `a`. This is called a *literal* character class.

The the next simplest character class shows an explicit list of characters. Oddly enough, these are known as *list character classes* For example

You can put any list of characters inside of `[....]` and use it to to create a match object. For instance, `[aWybe05]` will match any of the characters a, W, y, b, e, 0 or 5. Be warned, however that Characterclassese has some magic (meta-) characters. We shall address this issue here so you know what to do.

```
[aeiou]
```

represents any of the lower–case letters a, e, i, o or u. Notice how this character class lives in a "house" made of `[]`. The characters `]` and `[` metacharacters. They play the role of being the exterior walls of a character class's house. We will make a complete list of metacharacters in Characterclassese later in this section.

Drive Time! Make a file called oneTwo.txt (one to a line) and place this text in it.

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S t U V W X Y Z
0 1 2 3 4 5 6 7 8 9
! @ # $ % ^ & * ( )
- + = , .  < < > / ?
```

Then, put each character on its own line and make sure there is no space after it. You will use this file to learn abbout character classes.

First, let's look at a literal character class.

```
unix> grep q oneTwo.txt
q
```

Boring. And it is just what we expect. Let's try it on a list character class.

```
unix> grep [ab7^5] oneTwo.txt
a
b
5
7
^
```

4.1 Ranges

The metacharacter - creates a range of characters. The ordering of characters is determined by ASCII values. A character class can contain zero or more ranges. Let's begin with a simple example.

```
[a-z]
```

represents the lower case alphabetical characters. Range is keyed on ASCII value. If you are unsure about the ASCII value of a character, you can use Python's ord function to look it up, or you can look in an ASCII table. Here is a little Python session using the `ord` function..

```
>>> ord("a")
97
>>> ord("z")
122
>>> ord("A")
65
```

You can represent any alphabetical character with

```
[A-Za-z]
```

and any hex digit with

```
[0-9A-Fa-f]
```

Any octal digit may be represented with

```
[0-7]
```

You can include the - character in a character class by using

```
[\- ({\it and any other characters you want})]
```

Now try out a range character class.

```
MAC:Sat Jan 08:15:09:S1> grep [m-q] oneTwo.txt
m
n
o
p
q
```

Let's get case-insensitive.

```
unix> grep -i [m-q] oneTwo.txt
m
n
o
p
q
M
N
O
P
Q
```

Programming Exercise Use `grep` on the range characters classes with `oneTwo.txt` to see the characters they match.

A Metacharacter Moment This table shows characters that are "magic" inside of the [...] of a character class. You can defang their magic by preceding them with a whack:
.

`[`	begin a character class formed with a list of characters
`]`	end a character class formed with a list of characters
-	produces a range (see below)
\	defangs any magic character (ex: `\[` for a `[` character)
`^`	special "not" character

Let's be exclusionary and sNOTty! We can write "negative" character classes with the exclusion operator.

The not character `^` must appear FIRST, right after `[`. This example represents any character that is not a lower case alpha.

```
[^a-z]
```

4.2 Escape now!

We can toggle magic with the escape character

\

This turns magic off for magic characters and turns it on for certain non-magic characters. For example, `n` is not magic, but `\n` is the newline metacharacter.

Sometimes raw is better Recall that python has raw strings, which supresses the action of magic characters, including \. To make a raw string, you just put an `r` in front of the string. This will save your regex strings from having tons of irksome double backslashes.

4.3 Special Character Classes

Some character classes exist that take the form *someCharacter*. Here we present a short table of the most useful ones.

`\s`	This stands for any single whitespace character.
`\S`	This stands for any single non-whitespace character.
`\d`	This stands for any single decimal numeral.
`\D`	This stands for any character that is NOT a single decimal numeral.
`\w`	This stands `[a-zA-Z_0-9]`; this represents the characters allowed for Python identifiers in the positions beyond the first.

If you use these, enclose them in quotes when using `grep`. Here is an example.

4.4 Regexese

Be warned: The rules for Regexese are different from Characterclassese! This is because Regexese is a whole new language. When in character classes, speak Characterclassese, when outside, speak Regexese. Context is everything! Let us begin with the metacharacters of Regexese.

Basic Metacharacters (One Keystroke)	
Metacharacter	Action
`[`	begin delimiting a character class
`]`	end delimiting a character class
`^`	beginning-of-line charcter
`$`	end-of-line charcter
`|`	or
`\`	escape character The escape character can make other characters into metacharacters, or it can remove magic from a metacharacter.
`(`	left delimiter
`)`	right delimiter
`.`	any single character except for a newline
`* + ?`	repetition metacharacters (later)

To turn off any magic character, precede it with the escape character \. This rule is exactly the same as the corresponding rule in Characterclassese.

4.5 "And then immediately"

We shall now see our first regex for matching a sequence of characters. Juxtaposition in a regex means "and then immediately". The the regex

```
[a-iA-I][1-9]
```

matches a string that contains of a letter a-i and then a digit 0-9. The digit must immediately follow the letter. For example, baaa5 matches and Q3 does not.

Battleship! In the game of Battleship, we specify coordinates with letters a-i and digits 0-9. The regex

```
^[a-iA-I]\d$
```

matches any string that is a legit Battleship coordinate.

For example it will match a5, A9 or B3, but not Q4. This regex demands the following: begining-of-line andthenimmediately a-i or A-I andthenimmediately a decimal digit andthenimmediately an end-of-line.

String Literals The character 'a' is the same as the character class `[a]`. The regex

```
CUSIP[0-9]
```

contains the string literal "CUSIP"; this portion of the string requires an exact match of a the substring "CUSIP". Therefore the regex here matches CUSIP5, but not CUSIp5, CUSIP55 or CUSIP. Read it as CUSIP andthenimmediately a digit.

4.6 Repetition Operators

There are repetitions operators for regular expressions. These are all postfix operators. They all have higher precedence than juxtaposition.

Repetition Operators	
Operator	Action
?	expression appears 0 or 1 times
+	expression appears at one or more times consecutively
*	expression appears at zero or more times consecutively
{n}	expression appears exactly n times
{m,n}	expression appears at least m but not more than n times

To match a social security number, you needs three digits, followed by a dash, two more digits and then a dash and then four digits. This is an easy job when you use the repetition operators.

```
^[0-9]{3}-[0-9]{2}-{0-9}{4}$
```

Observe that - is *not* a magic character in Regexese. Now we will bring the delimiters (...) into the picture. The regex

```
^([a-c][0-9])+$
```

matches any string containing a character a-c followed by a digit any number of times. Here are some matches

```
>>> import re
>>> seeker = re.compile("^([a-c][0-9])+$")
>>> bool(seeker.search("a3b2c5")
True
>>> bool((seeker.search(""))
True
>>> bool(seeker.search("b4"));
False
```

Notice that the multiplicity operators have a higher order of precedence than juxtaposition. Hence the need for parentheses when having a regex with more than one character class being acted on by a multiplicity operator.

4.7 Using or

The operator | means "or". When using it, ALWAYS enclose the things you are "orring" in parens! This is a strict style expectation; adhere to it. It protects you from all manner of stupidity. The or operator is piggy and if you do not use parens, you do not control its ardor.

Let's plunge in with an example. Notice how we escape the magic character . to defang its magic (any character).

```
>>> import re
>>> seeker = re.compile("^(Morrison|Sheck) is a nut\.$")
>>> bool(seeker.search("Morrison is a nut"));  ##no period.
False
>>> bool(seeker.search("Morrison is a nut."));
True
>>> bool(seeker.search("Sheck is a nut."));
True
```

Two-Keystroke Metacharacters There are some characters that can be preceded by a to give a special interpretation. Here is table of some of them.

Two-Keystroke Metacharacters	
Metacharacter	Matches
\d	any decimal integer
\D	Any character not a decimal integer
\s	any whitespace character
\S	any non-whitespace character

For example the regex

```
^\s*-?[0-9]+\s*$
```

matches any string that has an integer in it that may or may not be surrounded by whitespace.

Turbo! Repetition operators can be applied to regexes, not just character classes. This is accomplished by using parentheses.

This regex will do a case-insensitive (note i after /) check and return true if the string passed it alternates a letter a-c followed by a digit zero or more times. Note: it must start with a letter and end with a digit. Enter this into a file named `reg.py`.

```
import re
seeker = re.compile("^([a-c]\d)*$", re.IGNORECASE)
print (bool(seeker.search("c4")), ", expected:  True")
print (bool(seeker.search("poop")),  ", expected:  False")
print (bool(seeker.search("A5b4")),  ", expected:  True")
```

Notice the second argument to `re.compile`; it causes the case of the string being scanned to be ignored. Running this program we see

```
unix> python reg.py
True , expected:  True
False , expected:  False
True , expected:  True
unix>
```

Programming Exercises

1. Write a function called `xWordCheat` which accepts as an argument a string of the form `--te--au-` and which returns all words in the Scrabble dictionary in a list that match the string with a single character for each dash. In this case, CATERWAUL should be in your list
2. Write a function called `containsInOrder` which takes a comma-separated list of at least one string as arguments and which retuns `True` if the second and subsequent strings are find in order inside of the first string Examples

   ```
   containsInOrder("asciicographical", "ci", "cog", "ica") -> True
   containsInOrder("zither", "th", "er", "zi") -> False
   ```

3. A valid IPv4 number is a string of the form xxx.xxx.xxx.xxx, where each xxx is a decimal number 0-255. Write a function that accepts a string as an argument and whic returns `True` when the string passed in is a valid IP4.
4. A valid hex code for a color is a six-digit hex number, which might be preceded by a prefix of `0x` or a `#`. Write a function that determines if a string is a valid hex color code.

5 Transforming with `tr`

This filter allows you to translate a file character by character. Create this file.

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!@#$%^&*(),.<>/?
```

We will now use the `tr` filter to change all lower-case letters to upper-case letters.

```
unix> cat sampler.txt | tr "abcdefghijklnopqrstuvwxyz" "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
ABCDEFGHIJKLmMNOPQRSTUVWXY
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!@#$%^&*(),.<>/?
```

Here is a nifty shortcut.

```
unix> cat sampler.txt | tr "[:lower:]" "[:upper:]"
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!@#$%^&*(),.<>/?
```