Chapter 29

Differences and Commonalities

In Chapter 25 we introduced the problem of similarity via the Humdrum **simil** and **correl** commands. This chapter revisits the problem of similarity by focussing on differences and commonalities. Specifically, this chapter introduces three additional tools, the UNIX **cmp**, **diff** and **comm** commands. Although these commands are less sophisticated than the **simil** and **correl** commands, they nevertheless provide convenient tools for quickly determining the relationship between two or more inputs.

Comparing Files Using *cmp*

The **cmp** command does a character-by-character comparison and indicates whether or not two files are identical.

```
cmp file1 file2
```

If the two files differ, **cmp** generates a message indicating the first point where the two files differ. E.g.,

```
file1 file2 differ: char 4, line 10
```

If the two files are identical, **cmp** simply outputs nothing ("silence is golden").

Sometimes files differ in ways that may be uninteresting. For example, we may suspect that a single work has been attributed to two different composers. The encoded files may differ only in that the `!!!COM:` reference records are different. We can pre-process the files using **rid** in order to determine whether the scores are otherwise identical.

```
rid -G file1 > temp1
rid -G file2 > temp2
cmp temp1 temp2
```

Of course one of the works might be transposed with respect to the other. We can circumvent this problem by translating the data to some key-independent representation such as `solfa` or `deg`:

```
rid -GL file1 | solfa > temp1
rid -GL file2 | solfa > temp2
cmp temp1 temp2
```

Two songs might have different melodies but employ the same lyrics. We can test whether they are identical by extracting and comparing any text-related spines. Since there may be differences due to melismas, we might also use **rid -d** to eliminate null data records.

```
extract -i '**silbe' file 1 | rid -GLd > temp1
extract -i '**silbe' file 2 | rid -GLd > temp2
cmp temp1 temp2
```

Similarly, two works might have identical harmonies:

```
extract -i '**harm' file 1 | rid -GLd > temp1
extract -i '**harm' file 2 | rid -GLd > temp2
cmp temp1 temp2
```

By further reducing the inputs, we can focus on quite specific elements, such as whether two songs have the same rhythm. In the following script, we first eliminate bar numbers, and then eliminate all data except for durations and barlines.

```
extract -i '**kern' file 1 | humsed '/=/s/[0-9]//; \
   s/[^0-9.=]//g' | rid -GLd > temp1
extract -i '**kern' file 1 | humsed '/=/s/[0-9]//; \
   s/[^0-9.=]//g' | rid -GLd > temp2
cmp temp1 temp2
```

For some tasks, we might focus on just a handful of records. For example, we might ask whether two works have the same changes of key.

```
grep '^*[a-gA-G][#-]*:'  file 1 > temp1
grep '^*[a-gA-G][#-]*:'  file 2 > temp2
cmp temp1 temp2
```

In the extreme case, we might compare just a single line of information. For example, we might identify whether two works have identical instrumentation:

```
grep '^!!!AIN:'  file 1 > temp1
grep '^!!!AIN:'  file 2 > temp2
cmp temp1 temp2
```

Comparing Files Using *diff*

The problem with **cmp** is that it is unable to distinguish whether the difference between two files is profound or superficial. A useful alternative to the **cmp** command is the UNIX **diff** command. The **diff** command attempts to determine the minimum set of changes needed to convert one file to another file. The output from **diff** entails editing commands reminiscent of the **ed** text editor. For example, two latin texts that differ at line 40, might generate the following output:

```
40c40
< es quiambulas
---
> es quisedes
```

Let's consider again the question of whether two works have essentially the same lyrics. Many otherwise similar texts might differ in trivial ways. For example, texts may differ in punctuation or in the use of upper- and lower-case characters. The **diff** command provides a **-i** option that ignores distinctions between upper- and lower-case characters. Punctuation marks can be eliminated by adding a suitable **humsed** filter.

```
extract -i '**silbe' file1 | text | humsed 's/[^a-zA-Z ]//g' \
    | rid -GLId > temp1
extract -i '**silbe' file2 | text | humsed 's/[^a-zA-Z ]//g' \
    | rid -GLId > temp2
diff -i file1 file2
```

Every time **diff** encounters a difference between the two files, it will output several lines identify the location of the difference and showing the conflicting lines in the two files. The **diff** command is line-oriented. Two lines need only differ by a single character in order for **diff** to generate an output.

When there are more than a dozen or so differences, the output becomes cumbersome to read. A useful alternative is to avoid looking at the raw output from **diff**; instead, we might simply count the number of lines of output (using **wc -l**). When compared with the total length of the input, the number of output lines can provide a rough estimate of the magnitude of the differences between the two files. A suitable revision to the last line of the above script would be:

```
diff -i file1 file2 | wc -l
```

One problem with this approach is that it assumes that we know which two files we want to compare. A more common problem is looking for *any* work that is somewhat similar to some given work. We can automate this task by embedding the above script in a loop so that the comparison (second) file cycles through a series of possibilities. A simple **while** loop will enable us to do this. Since our script may process a large number of scores, we ought to format our output for ease of reading. The **echo** command in our script outputs each filename in turn with the a count of the number of output lines generated by **diff**.

```
extract -i '**silbe' $1 | text | humsed 's/[^a-zA-Z ]//g' \
    | rid -GLId > temp1
shift
while [ $# -ne 0 ]
do
    extract -i '**silbe' $1 | text | humsed 's/[^a-zA-Z ]//g' \
        | rid -GLId > temp2
    CHANGES=`diff -i temp1 temp2 | wc -l`
    echo $1 ":  " $CHANGES
    shift
done
```

```
    rm temp[12]
```

Of course this same approach may be applied to other musical aspects apart from musical texts. For example, with suitable changes in the processing, one could identify works that have similar rhythms, melodic contours, harmonies, rhyme schemes, and so on.

Comparing Inventories — The *comm* Command

The **diff** command is sensitive to the order of data. Suppose that texts for two songs differ only in that one song reverses the order of verses 3 and 4. Comparing the "wrong" verses will tend to exaggerate what are really minor differences between the two songs. In addition, the above approach is too sensitive to word or phrase repetition. Many works — especially polyphonic vocal works — use extensive repetitions (e.g., "on the bank, on the bank, on the bank of the river"). Short texts (such as for the *Kyrie* of the Latin mass) are especially prone to use highly distinctive repetition. How can we tell whether one work has pretty much the same lyrics as another?

Fortunately, most texts tend to have unique word inventories. Although words may be repeated or re-ordered, phrases interrupted, and verses re-arranged, the basic vocabulary for similar texts are often much the same. A useful technique is to focus on the similarity of the word inventories. In the following script, we simply create a list of words used in both the original and comparison files.

```
    extract -i '**silbe' file1 | text | humsed 's/[.,;:!?]//g' \
       | rid -GLId | tr A-Z a-z | sort -d > inventory1
    extract -i '**silbe' file2 | humsed 's/[.,;:!?]//g' | tr A-Z
    a-z | text \
       | rid -GLId | sort | uniq -c | sort -nr > inventory2
```

Suppose that our two vocabulary inventories appear as follows:

Inventory 1:	**Inventory 2:**
domine	a
et	coronasti
eum	domine
filio	et
gloria	eum
in	filio
jerusalem	gloria
orietur	honore
patri	manuum
sancto	oper
spiritui	patri
super	sancto
te	spiritui
videbitur	super
	tuarum

Notice that a number of words are present in both texts, such as *domine*, *et*, *eum*, *filio*, and so on. Identifying the common vocabulary items is easily done by the UNIX **comm** command; **comm** compares two sorted files and identifies which lines are shared in common and which lines are unique to one file or the other.

The **comm** command outputs three columns: the first column identifies only those lines that are present in the first file, the second column identifies only those lines that are present in the second file, and the third column identifies those lines that are present in both files. In the case of our two Latin texts, the command:

```
comm inventory1 inventory2
```

will produce the following output. The first and second columns identify words unique to `inventory1` and `inventory1`, respectively. The third column identifies the common lines:

```
                a
                coronasti
                                domine
                                et
                                eum
                                filio
                                gloria
                honore
in
jerusalem
                manuum
                oper
orietur
                                patri
                                sancto
                                spiritui
                                super
te
                tuarum
videbitur
```

In the above case, five words are unique to `inventory1`, six words are unique to `inventory2` and nine words are common to both.

The **comm** command provides numbered options that suppress specified columns. For example, the command **comm -13** will suppress columns one and three (outputing column two). (Empty lines are also suppressed with these options.) A convenient measure of similarity is to express the shared vocabulary items as a percentage of the total combined vocabularies. We can do this using the word-count command, **wc**. The first command counts the total number of words and the second command counts the total number of shared words:

```
comm inventory1 inventory2 | wc -l
comm -3 inventory1 inventory2 | wc -l
```

An important point about **comm** is that the order of materials is important in the input files. If the word *filio* occurs near the beginning of `inventory1` but near the end of `inventory2` then **comm** will not consider the record common to both files. This is the reason why we used an alphabetical sort (**sort -d**) in our original processing.

On the other hand, there are sometimes good reasons to order the vocabulary lists non-alphabetically. For example, suppose we created our inventories according to the frequency of occurrence of the words. That is, suppose we use **uniq -c | sort -nr** to generate a vocabulary list ordered by how common each word is. Our inventory files might now appear as follows:

Inventory 1:

```
3   et
2   te
2   gloria
1   videbitur
1   super
1   spiritui
1   sancto
1   patri
1   orietur
1   jerusalem
1   in
1   filio
1   eum
1   domine
```

Inventory 2:

```
4   et
2   gloria
2   eum
1   tuarum
1   super
1   spiritui
1   sancto
1   patri
1   oper
1   manuum
1   honore
1   filio
1   domine
1   coronasti
1   a
```

Comparing these two inventories will produce little in common due to the presence of the numbers. For example, the records "`3        et`" and "`4        et`" will be deemed entirely different. However, we can eliminate the numbers using an appropriate **sed** command leaving us with vocabulary lists that are ordered according to the frequency of occurrence of the words. If we apply the

comm command to these lists then the commonality measures will be sensitive to the relative frequency of words within the vocabularies.

Reprise

In this chapter we have introduced the UNIX **cmp**, **diff** and **comm** commands. The **cmp** command determines whether two files as are the same or different. The **diff** command identifies how two files differ. The **comm** command identifies which (sorted) lines two files share in common; **comm** also allows us to identify which lines are unique to just one of the files.

The value of these tools is amplified when the inputs are pre-processed to eliminate unwanted or distracting data, and when post-processing is done (using **wc**) to estimate the magnitude of the differences or commonalities.

Together with the **simil** and **correl** commands discussed in Chapter 25, these five tools provide a variety of means for characterizing differences, commonalities, and similarities.