Chapter 40

Conclusion

In Chapter 1, we noted that computers are quite limited in what they are able to do. We noted in particular that computers are not good at *interpreting* data. Because so much music scholarship hinges on interpretations, this would seem to preclude computers from being of much use. However, as we have seen, there are some things that computers do well, and when a skilled user intervenes to make a few crucial interpretations, the resulting computer/human interaction can lead to pretty sophisticated results.

The lucid use of Humdrum depends on understanding how each of the tools works and how it is possible to connect the tools to perform particular tasks. The power and creativity of Humdrum truly lies in the hands of the user.

As we've seen, there are two parts to Humdrum: the *Humdrum syntax* and the *Humdrum toolkit.* The Humdrum syntax simply provides a formal framework for representing any kind of sequential symbolic data. A number of representation schemes are pre-defined in Humdrum, but you are free to construct your own representations as demanded by the tasks. In several of the tutorial examples in the previous chapters we saw various *ad hoc* representations that were concocted as temporary or intermediate representations. You should now feel comfortable with the possibilities of devising your own representations to assist you in whatever task you are interested.

The *Humdrum Toolkit* is a set of inter-related software tools. These tools manipulate text data conforming to the Humdrum syntax. If the data represents music-related information, then we can say that the Humdrum tools manipulate music-related information. Each Humdrum tool carries out a fairly modest process.

By way of review, we can group the various Humdrum (and UNIX) tools according to the type of operation. There are roughly a dozen or so classes of tasks:

1. **Displaying things.** The **ms** command can be used to print or display musical notation.
2. **Auditing.** MIDI sound output can be generated using the **midi** and **perform** commands.
3. **Searching for things.** When searching for things within specified files, appropriate commands include: **grep**, **egrep**, **yank -m**, **patt**, **pattern**, **correl**, **simil** and **humdrum -v**. When searching for files that meet certain conditions, appropriate commands include **grep -l**, **egrep -l** and **find**. Most of these search tools rely extensively on regular expressions to define patterns of characters.

4. **Counting things.** Appropriate commands include: **wc**, **wc -l**, **grep -c**, **egrep -c**, **census** and **census -k**. Eliminating unnecessary or confounding information can be achieved using **rid**, **extract**, **grep -v**, **sed** and **humsed**.
5. **Editing things.** Manual editing may be done using any text editor, such as **emacs** or **vi**. Automated (or "stream") editing may be done using **sed** and **humsed**.
6. **Editorializing.** E.g., add an editorial footnote to a specified note or passage; indicate that a passage differs from the composer's autograph.
7. **Transforming or translating between representations.** Appropriate commands include: **cents**, **deg**, **degree**, **dur**, **freq**, **hint**, **humsed**, **iv**, **kern**, **mint**, **pc**, **pf**, **pitch**, **reihe**, **semits**, **solfa**, **solfg**, **text**, **tonh**, **trans** and **vox**.
8. **Arithmetic transformations of representations.** Manipulating numerical values can be done using **xdelta**, **ydelta**, **recode** and **awk**.
9. **Extracting or selecting information.** Appropriate commands include: **extract**, **yank**, **grep**, **egrep**, **yank -m**, **thru**, **strophe**, **rend** and **csplit**.
10. **Linking or joining information.** Appropriate commands include: **assemble**, **cat**, **cleave**, **timebase**, **context**, **ditto** and **join**.
11. **Generating inventories of things.** Appropriate commands include: **sort** and **uniq**. Once again, unnecessary or confounding information can be eliminated using **rid**, **extract**, **grep -v**, **yank -m**, **sed** or **humsed**.
12. **Classifying things.** Numerical values can be classified using **recode**; non-numerical data can be classified using **humsed**.
13. **Labelling things.** Appropriate commands include: **patt -t**, **recode**, **humsed** and **timebase**.
14. **Comparing whether things are the same or similar.** Appropriate commands include: **diff**, **diff3**, **cmp**, **correl** and **simil**.
15. **Capturing data.** MIDI data can be input via **encode** and **record.**
16. **Trouble-shooting.** Appropriate commands include: **humdrum**, **humdrum -v**, **proof**, **proof -k**, **veritas**, **midi** and **perform**.

Not all of the existing Humdrum Tools were covered in this book. Nor were all of the available options described for all of the tools discussed. Exploring the *Humdrum Reference Manual* is recommended for readers interested in continuing to develop facility with Humdrum.

Pursuing a Project with Humdrum

If you have made it this far in the book, you will now have a fairly sophisticated knowledge of Humdrum. With this background, we might review some general principles and specific tips for making effective use of Humdrum when pursuing some musicological project. The following seven questions provide useful guidelines:

1. *What do I want my final output to look like?*
   Do I want a count or inventory?

```
115 instances of ...
28 instances of ...
```

Do I want to output a found pattern?

```
pattern found in line ...
pattern not found ...
```

Do I want a comparison?

```
file X is the same as file Y
X is similar to Y
X and Y are different
```

2. *What materials are available for processing?*

   Use **find** and **grep** to locate useful materials.

3. *What materials do I need to create?*

   Use **encode** to create new data. Use **humdrum** and **proof** to check the data. If necessary, define your own Humdrum representation for a given purpose.

4. *How do I transform my data so it is easier to process?*

   Use **recode** and **humsed** to classify data into various classes — such as *up*, *down*, *leap*, *long*, *short*, *difficult*, *easy*, *clarion register*, *dominant*, etc.

   Use translating/transforming commands such as **mint**, **ydelta**, **pcset**, etc to translate the data to a different representation.

5. *What data should I eliminate?*

   Use **rid**, **extract**, **yank**, **sed**, **humsed**, **uniq**, **uniq -d** and **grep -v** to eliminate selective materials.

6. *What data do I need to coordinate?*

   Use **context** to generate contextual information. Use **assemble**, **rend** and **cleave** to link information together.

7. *How do I know my results are worthwhile?*

   Use comparative tests whenever you can. Use **scramble -r**, **scramble -t**, **tac** and **reihe -s** to generate control groups.