Chapter 31

Repertories and Links

In initiating a research project, we often begin by selecting a suitable repertory for study. A common approach is to focus on a particular composer, period, style, or culture. For example, a researcher might focus on 17th century canons, or on Beethoven's compositions prior to Opus 20, or on Ojibway songs transcribed in the 1900s. Depending on the research task, the user may wish to locate works that conform to highly complex criteria — such as solo Baroque flute works written in compound meters, or slow Russian symphonic movements that don't include any wind instruments and are written in minor keys.

In this chapter, we discuss how to search entire file-systems for Humdrum scores that conform to user-specified criteria. With large musical databases, automated methods for locating specific repertories become important. As we will see, in most cases, one or two commands are all that is necessary to assemble a repertory list of works conforming to complex selection criteria.

The *find* Command

The UNIX **find** command traverses through a file hierarchy, and finds all files that match certain conditions. The **find** command takes the following syntax:

```
find <PATH> <OPTIONS> <ACTIONS>
```

The *PATH* is a directory from which the search *commences*. All files in the specified directory are examined including all files in the subdirectories, sub-subdirectories, and so on.

The path

```
/
```

means the "root" directory containing all files on a computer system. Even single-user systems are apt to have several thousand files subsumed under the root directory.

The path

```
/scores
```

means all files under the `scores` directory, whereas

```
/scores/bach
```

means all files under the `scores/bach` directory. The period character:

```
.
```

tells **find** to commence searching from the current directory.

Since **find** searches all files under the given path, its operation may be quite slow when there are thousands of files to search. It's wise to restrict the search by choosing a reasonable starting point. For example, specifying the path `/scores/bach/chorales` may save a great deal of time compared with the path `/scores`. Although we won't discuss them in this book, the **find** command provides a number of options that help to restrict the depth of searches or otherwise "prune" the search. When first trying **find** it's a good idea to limit the searches to small segments of the file system.

When searching through the specific *PATH*, **find** is able to carry out a wide variety of possible tests on each file. One simple action is to test whether the file-name conforms to a given regular expression. Consider, for example, the goal of identifying all files representing pitch-class (`**pc`) information. The Humdrum convention is to identify these files by adding the `.pc` extension to the filename — such as `opus24.pc`. The following **find** command will traverse through the `/scores` directory (and all sub-directories) searching for files that contain the `pc` file extension:

```
find /scores -name *.pc
```

The above command uses the **-name** option followed by the appropriate regular expression. This command is unusual in that it has no explicit *action*. In such cases, the implied action is to print or display the name of all files whose names match the regular expression.

Note that regular expressions may be literal strings. This means we can locate a specific named file. For example, the following command will locate all files named `findme`:

```
find / -name findme
```

An example of an explicit *ACTION* might be to delete files conforming to a particular criterion. For example, the following command searches the `/scores` path for files whose names contain the `.tmp` extension. Any matching file is then deleted using the UNIX **rm** command:

```
find /scores -name *.tmp -exec rm "{}" ";"
```

This command illustrates a number of features of the **find** command. The search begins from the *path* `/scores`. The *option* `-name *.tmp` identifies the search condition. The `-exec` flag identifies the *action* as that of executing a command. The arguments between `-exec` and the semi-colon are treated as the command to be executed. Each time a file is found with the `.tmp` extension, the `-exec` action is executed. The paired curly braces {} have a special significance to **find.** The braces are replaced by the filename found to match the regular expression. For example, if the first file found is named `xyz.tmp` the braces will be replaced by the string `xyz.tmp`.

The quotation marks around the braces and around the semi-colon are necessary in order to prevent the UNIX shell interpreter from substituting inappropriate information before passing the command line to **rm**.

Note that the **-name** option defaults to filenames only; it does not apply to directory names. The above command is equivalent to the more explicit form:

```
find /scores -type f -name *.tmp -exec rm "{}" ";"
```

The **-type** option can be used to match regular files (**f**), directories (**d**), or network files (**n**). By way of example, the following command deletes all directories whose names have the `.tmp` extension.

```
find /scores -type d -name *.tmp -exec rmdir "{}" ";"
```

Content Searching

For most music research applications, we are interested in identifying files on the basis of their contents. That is, we'd like to know what's inside the file before we take any action.

The **grep** command is especially useful in determining whether certain items of information are present in a file. For example, the following command identifies all files in the path `/scores` that contain passages in 7/8 meter:

```
find /scores -type f -exec grep -l '\*M7/8' "{}" ";"
```

Recall that the **-l** option for **grep** causes the output to consist only of names of files that contain the sought regular expression. Note that the **-type f** option has been specified in order to ensure that the **grep** command is only executed for files.

The structure of the above command can be used to search for all sorts of pertinent musical information. For example, recall that the `*IC` tandem interpretation is used to encode instrument classes such as strings, voice, percussion, etc. The following command searches all files in the path `scores` and generates a list of those files that encode scores containing one or more brass instruments:

```
find scores -type f -exec grep -l '\*ICbras' "{}" ";"
```

The following command identifies all files in the path `/scores`, that contain passages in the key of C major:

```
find /scores -type f -exec grep -l '\*C:' "{}" ";"
```

The following command identifies all files in the path `/scores`, that contain passages in any minor key:

```
find /scores -type f -exec grep -l '\*[a-g][#-]*:' "{}" ";"
```

Humdrum reference records are ideal targets for such searches since reference records encode information such as the composer's name, composer's dates, title of work, date of composition, movement number, instrumentation, meter classification, and so on. For example, the following command identifies all files in the path `/scores` that are composed by Franck:

```
find /scores -type f -exec grep -l '!!!COM.*Franck' "{}" ";"
```

The following command identifies all files in the path `/scores` that are written in compound meters:

```
find /scores -type f -exec grep -l '!!!AMT.*compound' "{}" ";"
```

The following command identifies all files beginning from the current directory that are rondos:

```
find . -exec grep -il '!!!AFR.*rondo' "{}" ";"
```

Recall that the **-i** option for **grep** makes the pattern-match insensitive to upper- or lower-case.

The following command identifies all files in the path `non-western` that have been designated as having heterophonic textures:

```
find non-western -exec grep -il '!!!AST.*heterophony' "{}" ";"
```

In the path `/scores/jazz`, we might want to identify all files that contain the style-designation "bebop:"

```
find /scores/jazz -exec grep -il '!!!AST.*bebop' "{}" ";"
```

The following command identifies all files in the path `18th-century`, that include French horns and oboes:

```
find 18th-century -exec grep -il '!!!AIN.*cor.*oboe' "{}" ";"
```

Of course, more complex regular expressions can be also be defined. For example, the following command identifies all works composed between 1805 and 1809:

```
find / -exec grep -l '!!!ODT.*180[5-9]' "{}" ";"
```

There is no restriction on the complexity of the regular expression. The following command identifies all works composed between 1812 and 1840:

```
find / -exec egrep -l '!!!ODT.*18(1[2-9])|([23][0-9])|(40)' \
   "{}" ";"
```

Often the **find** command can be used to answer research questions more directly. Suppose we wanted to determine whether German drinking songs more likely to be in triple meter. There are over four thousand German folksongs encoded in Helmut Schaffrath's *Essen Folksong Collection.* These works contain genre-related tags encoded as "AGN" reference records. One of the genres distinguished is "Trinklied" (drinking song).

In order to answer our question, we need to search the file system for all works that have the "Trinklied" designation, and then generate an inventory of meter classifications (available in "AMT" records).

```
find /scores -type f -exec grep -l '!!AGN.*Trinklied' "{}" \
   ";" | grep '!!!AMT.*' | sort | uniq -c
```

For the entire database, the output is as follows:

```
  1 !!!AMT: compound duple
  4 !!!AMT: irregular
 14 !!!AMT: simple quadruple
  5 !!!AMT: simple triple
```

There are just 24 drinking songs in the Essen collection and only five are in triple meters. The proportion of drinking songs in triple meters turns out to be no different than the distribution of triple meters in general for German folksongs. In other words, according to the *Essen Folksong Collection*, it is not the case that German drinking songs are more likely to be in triple meters.

Using *find* with the *xargs* Command

As we saw in Chapter 10, the **xargs** command can be used to propagate file names from command to command within a pipeline. Using **xargs** in conjunction with **find** provides a powerful means for finding works that conform to highly complex criteria. For example, the following command identifies all files in the path `/corelli` that contain a change of meter signature:

```
find /corelli -type f -name '*' | xargs grep -c '^\*M[0-9]' \
   | grep -v ':[01]$'
```

The output specifies each filename followed by a colon, followed by the number of meter signatures in the corresponding file. For example, in the following output, the third movement from Opus 1, No. 5 by Corelli is identified as containing 6 meter signatures at different points in the score:

```
/corelli/opus1n5c.krn:6
/corelli/opus1n9a.krn:3
/corelli/opus1n9b.krn:2
/corelli/opus1n9d.krn:2
```

Similarly, the following command identifies all works that contain a change of key signature:

```
find /scores -type f -name '*' | xargs grep -c '^\*k\[' \
   | grep -v ':[01]$'
```

As a further example of the use of **xargs**, consider the following extension of the above pipeline. The **grep -v** command causes only those files containing more than one key signature to be passed. The **sed** command eliminates the colon and the number appended to the filenames. The ensuing **grep -c** counts the number of meter signatures in each file. The final **grep -v** passes only those

filenames containing 2 or more meter signatures.

```
find / -type f -name '*' | xargs grep -c '^\*k\[' | \
   grep -v ':[01]$' | sed 's/:.*$//' | \
   xargs grep -c '^\*M[0-9]' | grep -v ':[01]$'
```

In summary, the above pipeline identifies all scores that contain both a change of key signature as well as a change of meter signature.

The **xargs** command can also be used to process a list of files — where the list has been stored in a file. For example, suppose we used the **find** command to locate all scores in compound meters written for woodwind quintet:

```
find . -name '*' | xargs grep -l '!!!AMT:.*compound' \
| xargs grep -l '!!!AIN: clars cor fagot flt oboe' > scorelist
```

The resulting list of files can be used for further processing. For example, we might search these files for any scores containing changes of key:

```
cat scorelist | xargs grep -c '^\*[A-Ga-g][#-]*:' | grep -v
':[01]$'
```

The output identifies all scores in compound meters written for woodwind quintet that contain changes of key.

Repertories As File Links

Rather than applying commands to files stored in a list, it is often helpful to have all of the files accessible in one location. That is, we might create a directory containing only those score-files that meet our selection criteria. it is often helpful to have all of the files accessible in one location. We might simply make copies of the files in a special directory. However, UNIX systems make it possible to create "links" to files in other directories without having to make duplicate copies of already existing files.

Suppose you wanted to make a directory of all scores containing vocal parts. The following command creates a file (`vocalfiles`) listing all files in the path `/scores` that contain one or more vocal parts:

```
find /scores -exec grep -l '!!!AIN.*vox' "{}" ";" > vocalfiles
```

The contents of `vocalfiles` might look like the following:

```
/scores/bach/cantatas/cant140.krn
/scores/bach/chorales/chor217.krn
/scores/bach/chorales/midi/chor368.hmd
etc.
```

We can create an appropriate new directory using the **mkdir** command.

```
mkdir vocal
```

Next, edit the file containing the list of filenames as follows. Insert **ln -s** prior to each filename, and append the directory name `vocal` at the end of each line.

```
ln -s /scores/bach/cantatas/cant140.krn vocal
ln -s /scores/bach/chorales/chor217.krn vocal
ln -s /scores/bach/chorales/midi/chor368.hmd vocal
etc.
```

(The **-s** option for **ln** is used to create a so-called "symbolic" link.)

Using the **chmod** command, we can make this file executable, and then we can execute it:

```
chmod +x vocalfiles
 ./vocalfiles
```

We now have a new directory whose files contain scores with vocal parts.

Reprise

The **find** command provides a convenient way to traverse through an entire file-system looking for files that conform to specific criteria. In musicological tasks, the **find** command is especially well suited to assembling a repertory of scores that exhibit some characteristic(s) of interest. Multiple selection criteria can be accommodated by using one or more pipes in conjunction with the **grep** command.

For convenience, it is often helpful to create a new directory that holds all of works selected for a study repertory. On UNIX systems, file "links" can be created, so that there is no need to make multiple copies of the same score. This means that several concurrent directory structures can be created without duplicating files. For example, a given score may be accessed in one directory structure via *composer*, in another directory via *instrumentation*, in a third directory via *genre*, and so on.