Chapter 14

Stream Editing

Most computer users are familiar with editing an electronic document using an interactive word-processor or text editor. *Stream editors* are non-interactive editors that automatically process a given input according to a user-specified set of editing instructions. A stream editor can be used, for example, to automatically transform a document from British spelling to American spelling. Stream editors are especially useful when processing large numbers of documents — such as a series of files encoding some musical repertory. In this chapter we will introduce two stream editors: **sed** and **humsed**.

The *sed* and *humsed* Commands

The **humsed** command is simply a Humdrum version of the common **sed** stream editor. The syntax and operation of **sed** and **humsed** are virtually identical. However, **humsed** will modify only Humdrum data records, whereas **sed** will modify any type of record, including Humdrum comments and interpretations. Both stream editors provide operations for *substitution*, *insertion*, *deletion*, *transliteration*, *file-read* and *file-write*. When used in combination, these operations can completely transform an input stream or document.

Simple Substitutions

The most frequently used stream-editing operation is substitution. Both **humsed** and **sed** designate substitutions by the lower-case letter `s`. Substitutions require two strings: the *target string* to be replaced, and the *replacement string* to be introduced. The syntax for substitutions is as follows:

s*<delimiter><target string><delimiter><replacement string><delimiter><options>*

No spaces are permitted between these elements. The delimiter can be any character; however, the same delimiter character must be used throughout the operation. The following substitution command causes occurrences of the letter 'A' to be replaced by the letter 'B':

```
s/A/B/
```

Since the slash character (/) appears immediately following the `s`, it becomes the delimiter for the rest of the operation. In this case no option has been given at the end of the substitution. Since the delimiter can be any character, the above command is functionally identical to the following:

```
sxAxBx
```

If it is necessary to use the delimiter character (as a literal) within either the target string or the replacement string it can be escaped using the backslash character.

There are two ways to execute a substition operation such as given above. One way is to give the substitution as a command-line argument to **sed** or **humsed**:

```
humsed s%A%B% filename
```

Alternatively, the operation can be placed in a file (for example, named `revise`):

```
s%A%B%
```

Then the stream editor can be invoked to execute the operations contained in this file using the **-f** option:

```
humsed -f revise inputfile
```

By default the output will be displayed on the screen. Using file-redirection (>) the output can be placed in some other file. Note that you should never redirect the output to the same file as the input — this will destroy the original input file. If necessary, send the output to a temporary file, and then use the UNIX **mv** command to rename the output.

Suppose that you had encoded a musical work in the `**kern` representation. Having finished the encoding, you realize that what you thought were *pizzicato* marks are really *spiccato* marks. In the `**kern` representation, pizzicatos are indicated by the double quote (`"`) whereas spiccatos are represented by the lower-case letter `s`. We can change all pizzicato marks to spiccato marks using the following command:

```
humsed 's/"/s/g' inputfile
```

Since the double quote is interpreted as a special character by the UNIX shell, we have escaped the entire substitution operation by placing it in single quotes. (Alternatively, we could place a backslash immediately before the double-quote character.) Note also the presence of the `g` option at the end of the string. Permissible options include any positive integer or the letter `g`. Without any option, the **sed** and **humsed** substitute (s) operation will replace only the *first* occurrence of the string in each data record. The `g` option specifies a "global" substitution, in that all occurrences on a given data record are replaced. If the option consisted of the number '3', then only the third instance of the target string would be replaced on each line.

Selective Elimination of Data

The *target string* in substitution operations is actually a regular expression. This means that we can specify patterns using the full power of regular expression syntax. Frequently, it is useful to eliminate certain kinds of information from a file. For example, we can eliminate all sharps and flats from a `**kern`-format file as follows:

```
humsed s/[#-]//g inputfile
```

Suppose we wanted to eliminate all beaming information in a score. In the `**kern` representation, open and closed beams are represented by `L` and `J` respectively; partial beams are represented by `K` and `k`.

```
humsed s/[JLkK]//g inputfile
```

Alternatively, we might want to eliminate all data except for the beaming information:

```
humsed s/[^JLkK]//g inputfile
```

Sometimes we need to restrict the circumstances where the data are eliminated. For example, we might want to eliminate all measure numbers. Eliminating all numbers from a `**kern` file will have the undesirable consequence of eliminating all note durations as well. Most **humsed** operations can be *preceded* by a regular expression delineated by slashes. This tells **humsed** to execute this substitution only if the data record matches the leading regular expression. For example, the following command eliminates measure numbers but not note durations:

```
humsed /^=/sX[0-9]*XXg inputfile
```

The operation may be interpreted as follows: look for lines that match a pattern where the first character in the line is an equals sign; if you find this pattern look for zero or more instances of any number between zero and nine, and replace that by an empty string; do this substitution for all numbers on the current data record.

Incidentally, Humdrum provides a **num** command that can be used to insert numbers in data records. The **num** command supports an elaborate set of options, but is not used often, so we won't describe it here. The following command renumbers all of the barlines in an input so that the first measure begins with the number 72. (Refer to the *Humdrum Reference Manual* for details regarding **num**.)

```
humsed /^=/sX=[0-9]*X=Xg inputfile | num -n ^= -x == -p = -o 72
```

Suppose we wanted to eliminate all octave numbers from a `**pitch` representation. In this case we want to delete all numbers except when they occur in conjunction with a barline. Our substitution should occur only when the current record does not match a leading equals sign:

```
humsed /^[^=]/s%[0-9]%%g inputfile
```

Suppose we wanted to determine which of two MIDI performances exhibits more dynamic range — that is, which performance has a greater variability in key-down velocities. Recall from Chapter 7 that MIDI data tokens consist of three elements separated by slashes (/). The third element is the key velocity. First, we want to eliminate key-up data tokens. These tokens can be distinguished by the minus sign associated with the second data element. An appropriate substutition is:

```
s%[0-9][0-9]*/-[0-9][0-9]*/[0-9]*  *%%g
```

(That is, replace by nothing any data that matches the following: a numerical digit followed by zero or more digits, followed by a slash, followed by a minus sign, followed by a digit, followed by zero or more digits, followed by a slash, followed by zero or more digits, followed by zero or more spaces.)

Having isolated only the key-down data tokens, we now need to eliminate everything but the third data element, the MIDI key-down velocities:

```
s%[0-9][0-9]*/[0-9][0-9]*/%%g
```

The *stats* Command

We can determine the range or variance of these velocity values by piping the output to the **stats** command. The **stats** command calculates basic statistical information for any input consisting of a column of numbers. A sample output from **stats** might appear as follows:

```
n:     124
total: 5700
mean:  45.9677
min:   9
max:   102
S.D.:  232.37
```

The value `n` indicates the total number of numerical values found in the input; the `total` specifies the sum of these numbers; the `mean` identifies the average; the `min` and `max` report the minimum and maximum values encountered, and the `S.D.` represents the standard deviation. The standard deviation provides a useful way of characterizing which performance has greater variability in key-down velocities.

Assuming that the above two stream-editing substitutions are kept in a file called `revise` we can compare the dynamic range for the two performances as follows:

```
extract -i '**MIDI' perform1 | grep -v ^= | humsed -r revise \
    | rid -GLId | stats
extract -i '**MIDI' perform2 | grep -v ^= | humsed -r revise \
    | rid -GLId | stats
```

The **extract** command has been added to ensure that we only process `**MIDI` data; the **grep** command ensures that possible barlines are eliminated, and the **rid** command eliminates comments and interpretations prior to passing the data to the **stats** command.

Eliminate Everything But ...

A common use for **humsed** is to eliminate signifiers that are not of interest. Stream editors like **sed** and **humsed** can be used to dramatically simplify some representation.

Did Monteverdi use equivalent numbers of sharps and flats? Or did he favor one accidental over the other? A simple way to determine this is to throw away everything but the sharps and flats. We can generate an inventory of just sharps and flats:

```
humsed 's/[^#-]//g' montev* | rid -GLId | sort | uniq -c
```

In some tasks, we might wish to transform a `**kern`-format file so that only pitch-related information is preserved:

```
humsed 's/[^a-gA-G#-]//g' inputfile
```

In extreme cases, we may wish to eliminate all Humdrum data from an input. The following command replaces all data tokens by null tokens:

```
humsed 's/[^	][^	]*/./g' inputfile
```

(That is, globally substitute all instances of the string not-a-tab followed by zero or more instances of not-a-tab characters, by a single period character.) This sort of command can be useful in generating a file that maintains the *structure* but not the *content* of some document. Incidentally, neither the **sed** nor the **humsed** commands support extended regular expressions, so we are not able to use the + metacharacter in the above substitution.

Deleting Data Records

Sometimes it is useful to delete entire data records rather than simply eliminating certain kinds of information. The `d` operation causes lines to be deleted. Normally, it is preceded by a regular expression that identifies which records should be eliminated. Deleting barlines can be done using the following command:

```
humsed /^=/d inputfile
```

Note that this is functionally equivalent to:

```
grep -v ^= inputfile
```

In the general case, **humsed /.../d** is preferable to **grep -v**. Remember that **humsed** only manipulates Humdrum data records; it never touches comments or interpretations. The **grep** command has no such restriction. Consider, for example, the following command to eliminate grace notes (acciaccaturas) from a `**kern`-format file.

```
humsed '/q/d' inputfile
```

By contrast, the command:

```
grep -v q inputfile
```

would also eliminate any comments or interpretation records containing the letter 'q'.

Suppose that we wanted to know whether a melody still evokes a certain key perception even if we eliminate all the tonic pitches. First we translate the representation to scale degree and assemble this file with the original `**kern` representation for the melody.

```
deg input > temp
assemble input temp | humsed '/1$/d' | midi | perform
```

Of course deleting all of the tonic notes will disrupt the original rhythm. An alternative is to replace all tonic pitches by rests:

```
deg input > temp
assemble input temp | humsed '/1$/s%[A-Ga-g#-]*%r%' | midi \
    | perform
```

Perhaps we might want to eliminate all the pitch information, and simply listen to the rhythmic structure of a work. That is, we might change all of the pitches in a work to a single pitch — in the following case, middle C:

```
humsed 's/[A-Ga-g#-]*/c/' | midi | perform
```

Adding Information

The substitute command can also be used to add information to points in a Humdrum input. For example, we might wish to add an explicit breath-mark (`,`) to the end of each phrase in a `**kern`-format input:

```
humsed s/}/},/g inputfile
```

Any occurrence of the ampersand (&) in the replacement string of a substitution is a standard stream-editing convention which means "the matched string." Suppose we want to add a tenuto mark to every quarter-note in a work. The following substitution seeks the number '4' followed by any character that is not a digit or period. This pattern is replaced by itself (&) followed by a tilde (~), the `**kern` signifier for a tenuto mark:

```
humsed s/4[^0-9.]/&~/g inputfile
```

Multiple Substitutions

Some tasks may require more than one substitution command. Multiple operations can be invoked by separating each operation by a semicolon. In the following example, we change all `**kern` quarter-notes to eighth-note durations:

```
humsed 's/4[A-Ga-g]/8&/g; s/84/8/g' inputfile
```

The first substitution finds strings that match the number '4' followed by an upper- or lower-case letter from A to G. The matched string is then output preceded by the number '8'. This operation will change all quarter notes and rests to eighty-fourth durations. The ensuing substitution operation changes '84' to '8' and so completes the transformation.

Switching Signifiers

In some situations, we will want to switch two or more signifiers — make all A's B's and all B's A's. These sorts of tasks require three substitutions and involve creating a unique temporary string. For example, the following command changes all `**kern` up-bows to down-bows and vice versa.

```
humsed 's/u/ABC/g; s/v/u/g; s/ABC/v/g' inputfile
```

The first substitution changes down-bows ('u') to the unique temporary string `ABC`. (In the `**kern` representation `ABC` is an illegal pitch representation, so it is bound to be a unique character string.) The second substitution changes up-bows (`v`) to down-bows. The third substitution changes occurrences of the temporary string `ABC` to up-bows.

Executing from a File

When several instructions are involved in stream editing, it can be inconvenient to type multiple operations on the command line. It is easier to place the editing instructions in a file, and use the **-f** option (with either **sed** or **humsed**) to execute from the file. Consider, for example, the task of rhythmic diminution, where the durations of notes are halved. We might create a file called `diminute` containing the following operations:

```
s/[0-9][0-9]\*/&XXX/g
s/64XXX/128/g
s/32XXX/64/g
s/16XXX/32/g
s/8XXX/16/g
s/4XXX/8/g
s/2XXX/4/g
s/1XXX/2/g
s/0XXX/1/g
```

Each substitution command is applied (in order) to every line or data record in the file. The first substitution adds the unique string `XXX` to every number. The ensuing substitutions transform these numbers to appropriate diminution values. We can execute these commands as follows:

```
humsed -f diminute inputfile
```

Writing to a File

A useful feature of **humsed** is the "write" or `w` operation. This operation causes a line to be written to the end of a specified file. Suppose, for example, we wanted to collect all seventh chords into a separate file called `sevenths`. With a `**harm`-format input, the appropriate command would be:

```
humsed '/7/w sevenths' inputfile.hrm
```

Each line containing the number 7 wll be written to a file named `sevenths`.

Similarly, we could copy all sonorities containing pauses to the file `pauses`.

```
humsed '/;/w pauses' inputfile
```

Of course there are other ways of achieving the same goal:

```
yank -m ';' 0 inputfile > pauses
```

Or even:

```
grep ';' inputfile | grep -v '^[!*]' > pauses
```

In some cases, a stream editor can be used to eliminate or modify data that will confound subsequent processing. For example, suppose we wanted to count the number of phrases that begin on the subdominant and the number of phrases that end on the subdominant. The **deg** command will allow us to identify subdominant pitches (via the number '4'). Since we would like to maintain the phrase indicators, we will avoid the **-x** option for **deg**. However, the **-x** option will pass *all* of the non-pitch related signifiers, including the duration data which encodes numbers. Hence, we will not be able to distinguish the subdominant ('4') pitch from a `**kern` quarter-note ('4'). The problem is resolved by first eliminating all of the duration information (numbers) from the original input:

```
humsed 's/[0-9.]//g' input.krn | deg | egrep -c '({.*4)|4.*{)'
humsed 's/[0-9.]//g' input.krn | deg | egrep -c '(}.*4)|4.*})'
```

In texts for vocal works, identify the number of notes per syllable.

```
extract -i '**kern' input | humsed 's/X//g' > tune
extract -i '**silbe' input | humsed 's/[a-zA-Z]*/X/' > lyrics
assemble tune lyrics | cleave -i '**kern,**silbe' -o '**new' \
   > combined
context -b X -o '[r=]' combined | rid -GLId | awk '{print NF}'
```

Identify the number of notes per word rather than per syllable.

```
extract -i '**kern' input > tune
extract -i '**silbe' input | humsed 's/^[^-].*[^-]$/BEGIN_END/;
s/-.*[^-]$/END/; s/^[^-].*-/BEGIN/' > lyrics
assemble tune lyrics | cleave -i '**kern,**silbe' -o '**new' \
   > combined
context -b BEGIN -e END -o '[r=]' combined | rid -GLId \
   | awk '{print NF}'
```

Reading a File as Input

Another useful feature is the **humsed** "read" or `r` operation. Whenever a leading regular expression is matched, a file is read in at that point. Suppose, for example, that we want to annotate a file with Humdrum comments identifying the presence of cadential 6-4 chords. First, we might create a file — `comment.6-4` — containing the following Humdrum comment:

```
!! A likely cadential 6-4 progression.
```

We can use the Humdrum **pattern** command (to be described in Chapter 21), as follows:

File `template`:

```
  .*
Ic
^\.  *
=    *
V[^I]
```

Command:

```
pattern -f template inputfile > output
humsed 'cadential-64/r comment.6-4' output > commented.output
```

Reprise

The **sed** and **humsed** commands provide stream editors that can automatically edit a data stream. We've seen that multiple operations can be carried out, either from the command line or from a file containing editing instructions. It should be noted that the **sed** and **humsed** commands provide many more editing facilities than those discussed in this chapter. Some 25 operations are provided by **sed** and **humsed**. For example, segments of text can be stored in various buffers, the contents of these buffers modified, and the results placed anywhere in the output text. Markers can be set at particular points and conditional branch statements executed. Stream-editing scripts have been written to execute programs of considerable complexity. However, for most tasks, the simple substitute (**s**) and delete (**d**) operations are the most useful. For further information about stream editing using **sed**, refer to the book on **sed** and **awk** written by Dale Dougherty (listed in the bibliography).