Humdrum Command Reference

Command Documentation Style

This section of the *Reference Manual* provides detailed information for all of the commands in the Humdrum Toolkit. These commands allow various Humdrum representations (described in Section 2) to be manipulated or transformed in musically useful ways. The commands can be classified into two broad groups: basic tools and specialized tools. *Basic tools* can be applied to any type of Humdrum input. An example of a basic tool is the **pattern** command — which is able to locate user-defined patterns for any type of Humdrum input. *Specialized tools* are more restricted in their operation. For example, the **cents** command changes pitch representations into hundredths of semitones. The tables on the following pages list all the basic and specialized commands included in Release 1.0 of the Humdrum Toolkit.

Note that all Humdrum commands assume that inputs conform to the Humdrum representation syntax. If a command fails to perform in the expected fashion, the user should first ensure that the input is valid by invoking the **humdrum** command. Most commands do not check the syntax of their inputs, so invalid inputs will produce invalid outputs.

Each command is described in a separate manual entry. Reference information is organized according to the standard UNIX documentation style. For users unfamiliar with this style, the following description should prove useful in understanding common conventions and abbreviations.

In general, commands are invoked according to the following conventional order of elements:

command name *options & arguments* *input files* *output file*

The *command name* is the name of the executable program to be invoked; *options* modify the operation of the command in distinctive ways; *arguments* provide special information required by some options; when more than one option is specified, each option may be followed by its own argument; *input files* are the names of pre-existing files of information to be processed; *output file* is the name of some file that will store the result.

The syntax by which a given command is invoked is specified in the **SYNOPSIS** section of the command documentation. **Bold** typeface indicates material to be typed literally by the the user. *Italic* typeface indicates text to be supplied by the user — such as file names, regular expressions, or numerical values. For example, the word *inputfile* should be replaced by the name of the file intended by the user as input. The italic letter *n* refers to an integer number (such as the number '23') to be supplied by the user. The italic *n.n* refers to a real number (such as the number '5.31') to be supplied by the user. Ellipses (...) are used to indicate that the preceding item may be repeated any number of times. Thus

inputfile ...

means that the user can specify one or more files as input. Square brackets [ ] are used to indicate items that are optional and may be omitted.

BASIC TOOLS	
assemble	paste together Humdrum files
census	determine general properties of a Humdrum input
cleave	join tokens from two or more spines into a single spine
context	congeal data records to form a contextual frame
correl	measure the numerical similarity between two spines
encode	interactive Humdrum encoding from MIDI input
extract	select input spines for output
fields	trace changes in spine structure
fill	replace null tokens with previous non-null data token
humdrum	test conformance to Humdrum syntax
humsed	stream editor for Humdrum files
info	calculate information flow
num	number selected records according to user-defined criteria
patt	locate and output user-defined patterns in a Humdrum input
pattern	exhaustively locate user-defined patterns in a Humdrum input
recode	recode numeric tokens in selected Humdrum spines
rend	split tokens in a single spine into two or more spines
rid	eliminate specified record types from the input
scramble	randomize order of either Humdrum data records or data tokens
simil	measure the similarity between two Humdrum spines
strophe	selectively extract strophic data
thru	expand repeats to through-composed form
xdelta	calculate numeric differences for successive tokens within a spine
yank	extract passages from a Humdrum input
ydelta	calculate numeric differences for concurrent spines

Basic Humdrum commands

The dash character (-) is used to designate command options — and should be typed literally when an option is invoked. Options are usually indicated by a single alphabetic letter, such as the **-b** option. In many cases, the option is followed by an *argument* that specifies further information pertaining to the invoked option. For example, the syntax: **-f** *filename* means that the name of a file must be provided as an argument if the **-f** option is specified. Options and their accompanying arguments must be separated by blank space (i.e. one or more spaces and/or tabs). If more than one option is invoked, and none of the invoked options require an argument, then the option-letters may be concatenated together. For example, the **-a** and **-b** options might be invoked as **-ab** (or as **-ba**) — provided neither option requires an argument.

By way of example, consider the syntax for the transposition command, **trans:**

trans **-d** [±]*n* [**-c** [±]*n*] [*inputfile* ...] [> *outputfile*]

Since the word **trans** is in bold typeface, the user must type it literally in order to invoke the command. The **-d** is an "option"; however, since this option is not surrounded by square brackets it is actually required. The user must type, literally:

```
trans -d
```

SPECIALIZED TOOLS	
cbr	calculate the critical band rate corresponding to some frequency
cents	translate pitch-related representations to cents
deg	translate pitch-related representations to relative scale degree
degree	translate pitch-related representations to absolute scale degree
diss	calculate the degree of sensory dissonance for successive spectra
freq	translate pitch-related representations to frequency
hint	determine harmonic intervals between concurrent pitches
iv	determine interval vectors for successive vertical sonorities
humver	display Humdrum toolkit version and copyright information
kern	translate pitch-related representations to kern
key	estimate the key (tonic and mode) of a passage
melac	calculate melodic accent values for successive pitches
metpos	assign metric position indicators to sonorities
mint	determine sequential diatonic interval between successive pitches
nf	determine normal form for successive vertical sonorities
pc	translate numeric semit values to numeric pitch-class
pcset	convert pitch-class to set-theoretic representation
pitch	translate pitch-related representations to American standard pitch notation
proof	check syntax of kern file
regexp	interactive regular-expression tester
reihe	output specified row variant for a given prime row
semits	translate pitch-related representations to semits
solfg	translate pitch-related representations to French solfège notation
spect	assemble total spectral content for individual sonorities
synco	measure degree of metric syncopation
timebase	reformat kern score with constant time-base
tonh	translate pitch-related representations to German pitch notation
trans	transpose pitch representations
urrhythm	characterize the rhythmic prototypes in a passage
vox	determine number of concurrently active voices

Specialized Humdrum commands

Since the letter *n* is in italics, it means that an integer must be specified. The integer is preceded by "±" in square brackets; this means that the number may be preceded by an optional plus or minus sign, such as:

```
trans -d +3
```

or

```
trans -d 6
```

or

HUMDRUM MIDI TOOLS	
encode	interactively encode Humdrum data from MIDI input
midi	convert from `**kern` to Humdrum `**MIDI` format
midreset	reset MIDI controller card
perform	play Humdrum `**MIDI` files
record	record MIDI activity in Humdrum `**MIDI` data format
smf	create *Standard MIDI File* from Humdrum `**MIDI` input
tacet	reset MIDI channels; silence

Humdrum MIDI commands

```
trans -d -12
```

Another option — the **-c** — option is also presented in square brackets. The entire expression is:

[**-c** [±]*n*]

This whole expression may be omitted. However, if the user chooses to use this option, the syntax *within* the square brackets must be followed. The **-c** is in bold typeface, so the user must type it literally if the option is selected. The letter 'c' is followed by blank space which can be followed by an optional plus or minus sign, followed by a required integer number. Syntactically correct invocations of the **trans** command include the following:

```
trans -d 13 -c 2
trans -d +7
trans -d -1 -c -5
trans -d -4 -c +8
```

If desired, the user may then specify one or more input files. Each file must be separated by blank space. Finally, an optional *outputfile* may be specified — however the name of the file must be preceded by the file redirection symbol ('>').

Inputs and Outputs

Most commands support several input and output modes *not explicitly indicated in the command* **SYNOPSIS**. Input to a command may come from three sources. In many cases the input will come from an existing file. Apart from existing files, input may also come from text typed manually at the terminal, or from the output of preceding commands. When input text is entered manually it must be terminated with an end-of-file character (control-D) on a separate line. (On IBM PCs the end-of-file character is control-Z.) When input is received from preceding commands, the output is sent via a UNIX pipe ('|').

The different ways of providing input to a command are illustrated in the following examples. In the first example, the input (if any) is taken from the terminal (keyboard). In the second example, the input is explicitly taken from a file named `input`. In the third example, the input is implicitly taken from a file named `input`. In the fourth example, the input to `command2` comes from the output of `command1`.

```
command
command < input
command input
command1 | command2
```

Outputs produced by commands may similarly be directed to a variety of locations. The default output from most commands is sent to the terminal screen. Alternatively, the output can be sent to another process (i.e. another command) using a pipe (|). Output can also be stored in a file using file redirection operator ('>') or *added* to the end of a (potentially) existing file using the file-append operator ('>>'). In the first example below, the output is sent to the screen. In the second example, the output is sent to the file `outfile`; if the file `outfile` already exist, its contents will be overwritten. In the third example, the output is appended to the end of the file `outfile`; if the file `outfile` does not already exist, it will be created. In the fourth example, the output is sent as input to the command `command2`.

```
command
command > outfile
command >> outfile
command1 | command2
```

When two or more commands have their inputs and outputs linked together using the pipe operator (|), the entire command line is known as a *pipeline*. Pipelines occur frequently in Humdrum applications.

Tee

A special UNIX command known as `tee` can be used to clone a copy of some output, so that two identical output streams are generated. In the first example below, the output is piped to `tee` which writes one copy of the output to the file `outfile` and the second copy appears on the screen. In the second example, the output from `command1` is split: one copy is piped to `command2` for further processing, while an identical copy is stored in the file `outfile1`; if the file `outfile1` already exist, its contents will be overwritten. In the third example, the append option (**-a**) for `tee` has been invoked — meaning that the output from `command` will be added to the end of any existing data in the file `outfile`. If the file `outfile` does not already exist, it will be created.

```
command | tee outfile
command1 | tee outfile1 | command2 > outfile2
command | tee -a outfile
```

The `tee` command is a useful way of recording or diverting some intermediate data in the middle of a pipeline.

Quotation Marks in the Command Line

Commands are recognized and executed by a command-line interpreter, known as a *shell*. Several characters and keywords hold special meanings for the shell. As we have learned, the characters > and | are normally used to mean *file redirection* and *pipe* respectively. In addition, the characters < & ( ) ; ` ' " * ? . # % = $ [~ and \ all have special meanings. For example, the characters * and ? are treated as pattern *metacharacters;* the asterisk means "match all instances" whereas the question-mark means "match any single letter." Hence, the command:

```
command *a?
```

will apply the `command` action to all files in the current directory having the letter 'a' as the second last letter in their filenames.

These special characters can be a nuisance when the user wants them to be treated literally, rather than according to their special meanings. The simplest way to defeat these meanings is by preceding the special character by a backslash character (\). The backslash is refered to as an "escape character" since it releases other characters from their special designations.

An alternative way of directing the shell command interpreter to disregard the meanings of special characters is by using quotation marks. Both single quotes (apostrophes) and double quotes can be used, however, single quotes are *stronger* in their effect. Consider the following commands:

```
echo $a
echo "$a"
echo '$a'
```

The dollar-sign ($) is interpreted by the command interpreter as specifying a shell variable. In the first and second commands, the shell looks for a variable named `$a` and attempts to echo the contents of this variable on the display. Unless `$a` happens to be an active shell variable, only an empty line will be displayed. In the third command, the string `$a` is treated literally, and echoed back to the display. There are circumstances where the double quotes are more useful, but for most casual users, the single quotes provide the best means for disengaging the meanings of special characters.

For a more complete discussion of command-line quoting, refer to any standard UNIX manual.

Help

Nearly all of the Humdrum commands provide a **-h** option that causes a "help screen" to be displayed. Each help screen simply provides a brief statement of the purpose of the command, the types of permissible input formats (if applicable), a brief summary of the available options, and a synopsis of the command syntax.

On-line Manual

More detailed information concerning various commands is normally available using the UNIX **man** command. Currently, there is no on-line help available for the Humdrum Toolkit so this *Reference Manual* will need to be a constant companion.