## NAME

**recode** — recode numeric tokens in selected Humdrum spines

## SYNOPSIS

**recode** -f *reassign-file* **-i** '*******interpretation*' [**-s** *regexp*] [**-x**] [*inputfile* ...]

## DESCRIPTION

The **recode** command is used to recode numeric components of data tokens in selected input spines. Typically, **recode** is used to reassign a range of numerical values into a finite set of classes or categories. For example, **recode** could be used to reassign all numerical values less than zero to the value -1, and to assign all values greater than or equal to zero to the value +1. A typical use of **recode** might be to reassign melodic intervals (represented in semitones) to one of five categories: (1) unison [0 semits], (2) up step [plus 1 or 2 semits], (3) up leap [plus 3 or more semits], (4) down step [minus 1 or 2 semits], (5) down leap [minus 3 or more semits]. Similarly, duration information might be rhythmically "justified" so that all durations near 0.5 seconds are recoded as precisely 0.5 seconds.

Note that **recode** will modify only those input spines matching the exclusive interpretation specified in the command line.

The manner by which numeric values are reassigned is specified by the user in a separate *reassignment file*. Reassignment files consist of one or more reassignment records; each record specifies a *condition* and a resulting replacement *string*. When the condition is satisfied, the numerical data is replaced by the associated string. A simple reassignment file is:

```
==0    zero
!=0    other
```

This file contains two reassignment records. *Conditions* are given in the left column and the associated replacement *strings* are given in the right column. Conditions and strings are separated by a single tab. Given the above reassignment, when a numerical value in an input token is equal to zero, the output replaces the input number by the alphabetic string "`zero`." The second condition (!= means not-equals) indicates that if a numerical value not equal to zero is encountered in an input token, the output replaces the number by the alphabetic string "`other`."

Permissible relational operators are listed in the following table.

| | |
|---|---|
| == | equals |
| != | not equals |
| < | less than |
| <= | less than or equal |
| > | greater than |
| >= | greater than or equal |
| else | default relation |

*Relational operators for* **recode**

Permissible replacement strings include any combination of printable ASCII characters with the exception of the tab.

Conditions are tested in the order given in the reassignment file. Thus if a numeric value satisfies more than one condition, only the first string replacement is made. Consider, for example, the following reassignment file:

```
<=0     LOW
>100    HIGH
>0      MEDIUM
```

In this case, all numeric values are replaced by one of three strings: `HIGH`, `MEDIUM`, or `LOW`. The order of specification is important in the above file. If the `MEDIUM` condition was specified prior to the `HIGH` condition, then all values greater than one hundred would be categorized as `MEDIUM` rather than as `HIGH`.

The *else* relation can be used to specify the default string output for numeric input values that satisfy none of the preceding conditions in the reassignment file. If no *else* condition is specified and none of the other conditions are satisfied, **recode** outputs the original input token without any modification.

Substitutions are made even when a number is embedded in non-numeric data. For example, given the above reassignment file, an input token `foo200bar` would be output as `fooHIGHbar`. That is, the numeric portion of the input string (200) would be deemed to satisfy the condition (>100) and so would be replaced by the string ("HIGH").

An important property of the **recode** command is that string replacements are limited to the **first** occurrence of numeric data within each data token. Subsequent numeric data within the token remains untouched. Thus, using the above reassignment file, the input token `foo200bar300` would be output as `fooHIGHbar300`.

In the case of multiple-stops (data tokens having two or more parts separated by spaces), **recode** processes the first occurrence of numeric data for each part of the token. For example, the double-stop token `foo200 bar300` would be output as `fooHIGH barHIGH`.

The **recode** command provides options to identify which data tokens may be excluded (skipped) in processing (**-s**), plus an option that suppresses the echoing of unprocessed

signifiers in the output (**-x**). See OPTIONS for further information.

## OPTIONS

The **recode** command provides the following options:

| | |
|---|---|
| **-f** *reassign* | use reassignments given in file *reassign* |
| **-h** | displays a help screen summarizing the command syntax |
| **-i** *'**interp'* | process only **interp* spines |
| **-s** *regexp* | skip; completely ignore tokens matching *regexp*; (echo in output only) |
| **-x** | (exclude) do not echo unprocessed data signifiers in the output |

Options are specified in the command line.

The user can suppress the echoing of non-numeric data within a token by specifying the **-x** option on the command line. When this exclude option is selected, only the replacement strings are output. For example, given the following reassignment file:

```
<=0     LOW
>100    HIGH
>0      MEDIUM
```

The input token `foo200bar` would be output as `HIGH`. If a data token contains no numeric component, then the **-x** option causes a null token to be output.

The **-x** option also suppresses the echoing of unprocessed numerical components. (Recall that string replacements made by **recode** are limited to the first occurrence of numerica data within a data token.) For example, with the **-x** option, the input data token `foo200bar17` would be output as `HIGH`.

Processing of certain types of data tokens may be avoided by invoking the **-s** (skip) option. This option must be accompanied by a user-define regular-expression (see **regexp (6)**). Input data tokens matching this expression are not processed and are simply echoed in the output. This option may be useful, for example, in avoiding the processing of barlines, or other types of data.

## EXAMPLES

The operation of the **recode** command can be illustrated by referring to the following hypothetical Humdrum file named `patrie`.

```
**kern    **abc
16g       0
8.g       00
16g       1
=1        =1
4cc       2.0
4cc       +3.
4ee       4
4ee       -1
=2        =2
4.gg      22.
8ee       1.1
8.cc      .1
16cc      x1X
8.ee      x1x2x
16cc      1 2
=3        =3
4a        .
4r        r
*-        *-
```

Consider also the following "reassignment" file, named `reassign`.

```
==0     zero
==1     one
==2     two
<0      negative
<=3     <=3
>4      >4
else    ???
```

The command:

```
recode -s = -i '**abc' -f reassign patrie
```

would produce the following output:

```
**kern    **abc
16g       zero
8.g       zero
16g       one
=1        =1
4cc       two
4cc       <=3
4ee       ???
4ee       negative
=2        =2
4.gg      >4
8ee       <=3
8.cc      <=3
16cc      xoneX
8.ee      xonex2x
16cc      one two
=3        =3
4a        .
4r        r
*-        *-
```

Notice the following: (1) the measure numbers `1` and `2` have remained unchanged due to the skip option `-s =`, (2) the input `x1X` has been replaced by the output string `xoneX` (non-numeric data remain in the same relative position), (3) the input `x1x2x` has been replaced by the output string `xonex2x` (only the first numerical value in each token is modified), (4) the double-stop input `1 2` has been replaced by the output string `one two`, and (5) both the null token (.) and the non-numeric token (`r`) have been echoed in the input unchanged.

Note that with the **-x** option, all of the non-numeric signifiers in `**abc` spine would be suppressed in the output. The single non-numeric token (`r`) would be output as a null token.

## PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

## SEE ALSO

**humsed** (4), **rend** (4), **regexp** (4), **regexp** (6), **sed** (UNIX)