

**NAME**

`rcalc` - record calculator

**SYNOPSIS**

**rcalc** [ **-b** ] [ **-l** ] [ **-p** | **-P** ] [ **-n** ] [ **-w** ] [ **-u** ] [ **-tS** ] [ **-i format** ] [ **-o format** ] [ **-in M** ] [ **-on M** ] [ **-f source** ] [ **-e expr** ] [ **-s svar=sval** ] file ..

**DESCRIPTION**

*Rcalc* transforms "records" from each *file* according to the given set of literal and relational information. By default, records are separated by newlines, and contain numeric fields separated by tabs. The **-tS** option is used to specify an alternate tab character.

A **-i format** option specifies a template for an alternate input record format. *Format* is interpreted as a specification string if it contains a dollar sign '\$'. Otherwise, it is interpreted as the name of the file containing the format specification. In either case, if the format does not end with a newline, one will be added automatically. A special form of the **-i** option may be followed immediately by a 'd' or an 'f' and an optional count, which defaults to 1, indicating the number of double or float binary values to read per record on the input file. If the input is byte-swapped, the **-iD** or **-iF** options may be substituted. If binary input is specified, no format string or file is needed.

A **-o format** option specifies an alternate output record format. It is interpreted the same as an input specification, except that the special **-od** or **-of** options do not require a count, as this will be determined by the number of output channels in the given expressions. If byte-swapped output is desired, the **-oD** or **-oF** options may be substituted.

The **-p** option specifies "passive mode," where characters that do not match the input format are passed unaltered to the output. If the **-P** option is given instead, then valid input records that do not yield a positive *cond* value are similarly passed to the output. (See paragraph below.) These options require that the **-i** input format and **-o** output format also be present. With both input and output formats, the passive mode can effectively substitute information in the middle of a file or stream without affecting the rest of the data.

If a **-in** option is given with a positive integer argument, *rcalc* will stop processing input once it has loaded this number of records. Similarly, if a **-on** option is present, *rcalc* will stop after producing this many records, which may be a smaller number if the *cond* variable is used. If multiple input files are given, these counts are continuous over the input and do not reset on each file.

The variable and function definitions in each **-f source** file are read and compiled from the RADIANCE library where it is found. The **-e expr** option can be used to define variables on the command line. Since many of the characters in an expression have special meaning to the shell, it should usually be enclosed in single quotes. The **-s svar=sval** option can be used to assign a string variable a string value. If this string variable appears in an input format, only records with the specified value will be processed.

The **-b** option instructs the program to accept only exact matches. By default, tabs and spaces are ignored except as field separators. The **-l** option instructs the program to ignore newlines in the input, basically treating them the same as tabs and spaces. Normally, the beginning of the input format matches the beginning of a line, and the end of the format matches the end of a line. With the **-l** option, the input format can match anywhere on a line.

The **-w** option causes non-fatal error messages (such as division by zero) to be suppressed. The **-u** option causes output to be flushed after each record. The **-n** option tells the program not to get any input, but to produce a single output record. Otherwise, if no files are given, the standard input is read.

Format files associate names with string and numeric fields separated by literal information in a record. A numeric field is given in a format file as a dollar sign, followed by curly braces enclosing a variable name:

This is a numeric field: \${vname}

A string variable is enclosed in parentheses:

This is a string field: \$(sname)

The program attempts to match literal information in the input format to its input and assign string and

numeric fields accordingly. If a string or numeric field variable appears more than once in the input format, input values for the corresponding fields must match (ie. have the same value) for the whole record to match. Numeric values are allowed some deviation, on the order of 0.1%, but string variables must match exactly. Thus, dummy variables for "don't care" fields should be given unique names so that they are not all required to take on the same value.

For each valid input record, an output record is produced in its corresponding format. Output field widths are given implicitly by the space occupied in the format file, including the dollar sign and braces. This makes it impossible to produce fields with fewer than four characters. If the *-b* option is specified, input records must exactly match the template. By default, the character following each input field is used as a delimiter. This implies that string fields that are followed by white space cannot contain strings with white space. Also, numeric fields followed but not preceded by white space will not accept numbers preceded by white space. Adjacent input fields are advisable only with the *-b* option. Numeric output fields may contain expressions as well as variables. A dollar sign may appear in a literal as two dollar signs (\$\$).

The definitions specified in *-e* and *-f* options relate numeric output fields to numeric input fields. For the default record format, a field is a variable of the form \$N, where N is the column number, beginning with 1. Output columns appear on the left-hand side of assignments, input columns appear on the right-hand side.

A variable definition has the form:

var = expression ;

Any instance of the variable in an expression will be replaced with its definition.

An expression contains real numbers, variable names, function calls, and the following operators:

+ - \* / ^

Operators are evaluated left to right, except '^', which is right associative. Powers have the highest precedence; multiplication and division are evaluated before addition and subtraction. Expressions can be grouped with parentheses. All values are double precision real.

A function definition has the form:

func(a1, a2, ..) = expression ;

The expression can contain instances of the function arguments as well as other variables and functions. Function names can be passed as arguments. Recursive functions can be defined using calls to the defined function or other functions calling the defined function.

The variable *cond*, if defined, will determine whether the current input record produces an output record. If *cond* is positive, output is produced. If *cond* is less than or equal to zero, the record is skipped (or passed to the output if *-P* is specified) and no other expressions are evaluated. This provides a convenient method for avoiding inappropriate calculations. The following library of pre-defined functions and variables is provided:

- \$N**            Return the value for input column *N*. If an input format is given, using a channel number generates an error.
- in(n)**        Return the value for input column *n*, or the number of columns available in this record if *n* is 0. This is an alternate way to get a column value instead of using the \$N notation, and is more flexible since it is programmable. This function is disabled if an input format is used.
- if(test, then, else)**  
               if test is greater than zero, then is evaluated, otherwise else is evaluated. This function is necessary for recursive definitions.

**select(N, a1, a2, ..)**  
return aN (N is rounded to the nearest integer). This function provides array capabilities. If N is zero, the number of available arguments is returned.

**rand(x)** compute a random number between 0 and 1 based on x.

**min(a1, a2, ..)**  
return the minimum value from a list of arguments.

**max(a1, a2, ..)**  
return the maximum value from a list of arguments.

**floor(x)** return largest integer not greater than x.

**ceil(x)** return smallest integer not less than x.

**sqrt(x)** return square root of x.

**exp(x)** compute e to the power of x (e approx = 2.718281828).

**log(x)** compute the logarithm of x to the base e.

**log10(x)** compute the logarithm of x to the base 10.

**PI** the ratio of a circle's circumference to its diameter.

**recno** the number of records recognized thus far.

**outno** the number of records output thus far (including this one).

**sin(x), cos(x), tan(x)**  
trigonometric functions.

**asin(x), acos(x), atan(x)**  
inverse trigonometric functions.

**atan2(y, x)** inverse tangent of y/x (range  $-\pi$  to  $\pi$ ).

**EXAMPLE**

To print the square root of column two in column one, and column one times column three in column two:

```
rcalc -e '$1=sqrt($2);$2=$1*$3' inputfile > outputfile
```

**ENVIRONMENT**

RAYPATH the directories to check for auxiliary files.

**AUTHOR**

Greg Ward

**BUGS**

String variables can only be used in input and output formats and  $-s$  options, not in definitions.

Tabs count as single spaces inside fields.

The  $-P$  option buffers up to 16 Kbytes of data per record. Longer records passed because the "cond" variable evaluates to  $\leq 0$  will be partial; a warning will be generated and the string "\*\*\* MISSING DATA \*\*\*" will appear at the break. The only workaround is to use the  $-p$  option instead, which does not pass rejected records.

**SEE ALSO**

cnt(1), ev(1), getinfo(1), icalc(1), rcollate(1), rlam(1), rsplit(1), tabfunc(1), total(1)