M-Command Manual

NYSOL Package Current Release: Ver. 3.0

Releases:
April 30, 2017: Ver. 3.0 Release
June 30, 2016: Addition of notes about summary processing command, and modify of parameters
December 24, 2015: Ver. 2.4 Release, and addition of assert function
October 6, 2014: Ver. 2.0 Release
April 30, 2014 : Updated errors in examples
March 10, 2014 : Changes in installation method corresponding to integration of NYSOL package
February 15, 2014 : Added mxml2csv
November 2, 2013 : Added mpadding, mvnullto, mvdelnull, and minor changes
September 20, 2013 : First Release

October 13, 2017

Copyright ©2013-2017 by NYSOL CORPORATION

Contents

Chapter 1

Changes

1.1 Changes in Ver.2

1.1.1 Addition of Automatic Sorting Function

In MCMD Ver. 1.0, sorting by key field with `msortf` is necessary prior to the aggregate calculation with commands such as `msum` and joining of files such as `mjoin` command.

Key fields must be sorted for commands which require specification key field name, or else it would result in calculation error if users failed to sort the key field beforehand, this is the main draw back easily give rise to bugs in scripts.

Therefore, the new function in Ver. 2.0 automatically sorts key columns for commands where key field(s) is/are specified at `k=` option. The symbol of sort order by column is added to the field name in the CSV header (Refer to sort Sort order of columns for more information), and sorting is only carried out when necessary on required commands.

Example of Ver. 1.0

Before executing `msum k=customer` command, `customer` column must be sorted with `msortf` command.

```
$ more dat1.csv
customer,amount
A,10
B,10
A,20
B,15
B,20
$ msortf i=dat1.csv f=customer | msum k=customer f=amount:totalamount o=rsl1.csv
#END# kgsortf f=customer i=dat1.csv
#END# kgsum f=amount:totalamount k=customer o=rsl1.csv
$ more rsl1.csv
customer,totalamount
A,30
B,45
```

Example of Ver. 2.0

`msum` will perform sorting when necessary without the use of `msortf` command while achieving the correct results as shown in the previous example. The symbol `%0` is added next to the column name of `customer` after sorting is carried out in `msum` command.

```
$ more dat1.csv
customer,amount
A,10
B,10
A,20
B,15
B,20
$ msum i=dat1.csv k=customer f=amount:totalamount o=rsl1.csv
#END# kgsum f=amount:totalamount k=customer i=dat1.csv o=rsl1.csv
$ more rsl1.csv
customer%0,totalamount
A,30
B,45
```

1.1.2 Commands with Change in Specifications

Table 1.1 shows the list of commands with specification changes from Ver. 1.0 (Excludes commands with `k=` parameter where automatic sorting has been added). In addition, the `s=` parameter is added to the commands where the sort order of records will affect the processing results.

Table 1.1: Commands with Change in Specifications

Command	Description	Changes
maccum	Calculate cumulative totals	Added `s=` parameter (required)
mbest	Select specific rows	Added `s=` parameter (required)
mcombi	Compute combinations	Added `s=` parameter
mkeybreak	Keybreak point	Added `s=` parameter
mmvavg	Compute moving average	Added `s=` parameter (required)
mmvsim	Compute similarity of sliding window	Added `s=` parameter (required)
mmvstats	Compute statistics of sliding window	Added `s=` parameter (required)
mnumber	Serial number	`s=` parameter is required when `-B` is not used.
mpadding	Repair rows	`type=` parameter is not retired, repair method is specified at `f=`.
mrjoin	Join reference file by specified range	Specify sorting method at `r=` parameter.
mslide	Shift records	Added `s=` parameter (required)
mtra	Convert vert data to vector	Added `s=` parameter
mwindow	Generate sliding window	Specify sorting method at `wk=`parameter.

1.1.3 Executing Previous Scripts

When `-q` option is used, automatic sorting on columns defined at `k=` parameter is disabled. As shown in Table 1.1, when `s=` parameter is not defined for commands that takes in `s=` parameter, the commands will operate as in Ver. 1.0. For scripts using Ver. 2.0 commands where `k=` parameter and `-q` option is specified, while `s=` parameter is not defined, the results is equivalent to scripts using Ver. 1.0 commands (For mpadding command, even though `type=` parameter is removed, the specification is required at `f=` parameter).

Script Before Modification

In Ver. 2.0, `maccum` command will return an error if `s=` parameter is not specified.

```
msortf i=customer.csv f=custID,date |
maccum k=custID f=amount o=accum.csv
```

```
#ERROR# parameter s= is mandatory without -q (kgaccum); kgaccum f=amount i=test.csv k=custID
```

Modification

`s=` parameter is required for `maccum` command, when -q option is used, it is equivalent to the execution method in Ver. 1.0.

```
msortf i=customer.csv f=custID,date |
maccum -q k=custID f=amount o=accum.csv
```

1.2 Changes in Ver. 3

MCMD has been separated from the NYSOL package and registered in GitHuB. Accordingly, the version level has been raised from 2.x to 3.0. There are no major changes in the command specifications. There are some minor enhancements as described in the following sections. Several new commands have been added, including input commands.

1.2.1 Addition of the -params option

For all commands, the -params option has been added. This option is primarily intended for use by a system. Specifying the -params option sends a list of available parameters (and options) to the standard output.

```
$ mcut -params
-assert_diffSize
-assert_nullin
```

```
-nfn
-nfni
-nfno
-params
-r
-x
f=
i=
o=
precision=
tmpPath=
```

1.2.2 Addition of environment variable KG_msgTimebyfsec

When this environment variable is true, the end message time is shown in units of microseconds.

```
$ export KG_msgTimebyfsec=true
```

1.2.3 Enhancement of the mchkcsv command functions

- UTF BOM is automatically removed.
- Changes have been made so that specifying the -diag option makes the contents output in English and specifying the -diagl option makes the contents output in Japanese.

1.2.4 Enhancement of the mcal command functions

- Changes have been made so that multiple values can be specified for c= and a=.
- Changes have been made to allow values in microseconds to be entered in `$t{}`, `#t{}`, and `0t`. Microseconds are indicated by way of the decimal point (six digits). Accordingly, the following functions have been added: `diffusecond(T,T), tuseconds(T), usecond(T), useconds(T), and unow().`
- A Boolean value can be returned as in `if(B,B,B).`

1.2.5 Enhancement of the mcat command functions

- With the `-skip_zero` option specified, a 0-byte file no longer results in an error even when the `-nfn` option is not specified.
- Using the `flist=` parameter, you can specify a CSV file as the list of files to be merged. Use the following syntax: flist=filename:fldName.
- With the `kv=` parameter specified, the “ key-value ” character string is extracted from a path name and added to data as a fieldname and its value.

1.2.6 Additional commands

Five screen input commands shown below have been added. They are still under development, and their specifications may be changed in the future.

- minput : Displays the input screen.
- mminput : Displays the input screen with multiple input frames.
- mdsp : Displays a character string at a specified position on the screen.
- mseldsp : Displays a single-choice input window on the screen.
- mmseldsp : Displays a multiple-choice input window on the screen.

In addition, the following commands have been added:

- mshuffle : Partitions a file with a hush key.
- mcsvconv : Converts a CSV file into one of several other formats.
- mcsv2json : Converts a CSV file into a json file.

Chapter 2

M-Command

2.1 M-Command

M-Command (also referred to as MCMD) is a set of commands developed for high-speed processing of large-scale spreadsheet data (CSV). The M derives from the name of the inventor, Mr. Yasuyuki Matsuda. M-Command provides about 80 different commands where each command is specific to a single function (for example, sort or join tables). All commands use the same, very simple processing method they all read CSV data from standard input and write the results to standard output. The user can interconnect the single-function commands with pipes and implement sophisticated processing. M-Command enables a standard PC to process data records in the order of hundreds of millions. You will not need much time to learn the basic usage of M-Command. Through intensive practice, it will take just a matter of weeks to be fairly skillful in data processing.

2.2 Installation

2.2.1 Requirements

Operation of MCMD has been confirmed on major operating systems, including Linux, MaxOS X, and Bash on Ubuntu on Windows. It will probably work on any UNIX operating system with the help of the above tools. The following are the versions of operating systems on which MCMD has been confirmed to run:

- MacOS 10.9.5 (Marverics) or later
- CentOS 7.3 1611
- Ubuntu 16.04 LTS
- Bash on Ubuntu on Windows(Windows 10)

To compile source codes, you need the following tools:

- c++ compiler
- autotools
- boost C++ Library
- lib2xml Library

2.2.2 Compilation and installation

MacOS X

```
## Make sure autotools and boost have been installed. If not, use brew or similar to install them.
$ brew install autoconf
$ brew install automake
$ brew install libtool
$ brew install boost

## Download the source.
$ git clone https://github.com/nysol/mcmd.git
$ cd mcmd

## Compile and install MCMD.
$ aclocal
$ autoreconf -i
$ automake --add-mising
$ ./configure
$ make
$ sudo make install
```

CentOS

```
## Make sure autotools, boost, and libxml2 have been installed. If not, use yum or similar to install the
$ sudo yum update
$ sudo yum groupinstall "Development Tools"
$ sudo yum install boost-devel
$ sudo yum install libxml2-devel

## Download the source.
$ git clone https://github.com/nysol/mcmd.git
$ cd mcmd

## Compile and install MCMD.
$ aclocal
$ autoreconf -i
$ ./configure
$ make
$ sudo make install
```

Ubuntu,Bash on Windows

```
## Make sure necessary tools have been installed, including autotools, boost, and libxml2. If not, use ap
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install build-essential
$ sudo apt-get install git
$ sudo apt-get install libboost-all-dev
$ sudo apt-get install libxml2-dev

## Download the source.
$ git clone https://github.com/nysol/mcmd.git
$ cd mcmd/

## Compile and install MCMD.
$ aclocal
$ autoreconf -i
$ ./configure
$ make
$ sudo make install

# Specify the path to the shared library.
# If the path is to be specified each time upon startup, have it described in .bash_profile.
$ export LD_LIBRARY_PATH=/usr/local/lib
```

2.2.3 Checking successful installation

To check how to use each command, use the `--help` option to view a simple help (see Figure 2.1). Check that the command help is shown as described below, and the installation is complete. In MCMD Ver. 2.4 and later, the help is in English by default. To view Japanese help, specify the `--helpl` option (see Figure 2.2). To make Japanese the default output language for `--help`, specify `KG_LOCALHELP=true` in the environment variables. To view the version number of the installed MCMD, specify the `--version` option (see Figure 2.3). Note that the shown version number is for the entire MCMD and not for each command. Thus, the same version number is shown for all commands.

2.2.4 What to do when installation has failed

Some cases of MCMD installation failure have been reported regarding operating systems installed with Anaconda. You can avoid this error by changing the command search path sequence as described below. Change the sequence before installing MCMD.

```
# Correct the ~/.bash_profile path as described below.
# Before)
export PATH="/Users/username/anaconda/bin:$PATH"

# After)
export PATH="$PATH:/Users/username/anaconda/bin"
```

```
$ mcut --help

MCUT - SELECT COLUMN

Extract the specified column(s). The specified column is removed with the -r
option.

Format

mcut f= [-r] [-nfni] [i=] [o=] [-assert_diffSize] [-assert_nullin]
[-nfn] [-nfno] [-x] [-q] [tmpPath=] [--help] [--helpl] [--version]
```

Figure 2.1: Help in English (default)

```
$ mcut --helpl

mcut 項目の選択
===============
指定した項目を選択する。
-r オプションを付けると、指定した項目を削除する。

書式
----
mcut f= [-r] [i=] [o=] [-nfn] [-nfno] [-x] [--help]  [--helpl] [--version]

パラメータ
----------
  f=      抜き出す項目名
          項目名をコロンで区切ることで、出力項目名を変更することができる。
          ex. f=a:A,b:B
  -r      項目削除スイッチ
          f=で指定した項目を削除し、それ以外の項目が抜き出される。
  -nfni   入力データの１行目を項目名行とみなさない。よって項目番号で指定しなければならない。
          以下のように、新項目名を組み合わせて指定することで項目名行を追加出力することが可能となる。
          例）f=0:日付,2:店,3:数量
                  :
                  :
```

Figure 2.2: Help in Japanese

```
$ mcut --version
lib Version 1:1:0:0
```

Figure 2.3: Version

2.3 Let's start

Let's start with a simple example. Sample CSV data is required in order to proceed with the tutorial. In MCMD, the `mdata` command outputs many different kinds of data sets. See the example in Figure 2.4. The line beginning with `$` indicates the command line input, followed by the execution message and results from the command. The `mdata` command reads the data type from the parameter and directs the contents to standard output (Refer to the chapter on mdata for more details). In the following example, the data is saved to the file `man0.csv` by redirecting the contents to the standard output.

```
$ mdata man0 >man0.csv
#END# kgData -man0
$ more man0.csv
顧客,金額
A,5200
B,4000
B,3500
A,2000
B,800
```

Figure 2.4: Generate data with mdata command

All commands of MCMD return a status message starting with `#END#` when they have ended successfully. (If they have abended, they return a status message starting with `#ERROR#`.) In addition, `more` is a UNIX command to display the contents of the file one page at a time which allows forward navigation of the file. All examples in this manual are illustrated according to the above-mentioned format.

Figure 2.5 shows an example of calculating the total amount for each customer from the data in `man0.csv`. The `msum` command reads the data from the `man0.csv` file (`i=`), uses the customer field set as the key field (`k=`) to calculate the total amount (`f=`), and the results are written to the output.csv file (`o=`). In the output CSV file, the field name customer is followed by `%0`, which indicates that the msum command has used the customer field as the sorting key. The `%0` is not part of the fieldname, and you can just ignore it for the time being.

Let 's explore a more complex example. Figure 2.6 shows how to display products and their quantities for

```
$ msum k=customer f=amount i=man0.csv o=output.csv
#END# kgsum f=amount i=man0.csv k=customer o=output.csv
$ more output.csv
customer%0,amount
A,7200
B,8300
```

Figure 2.5: Total amount by each customer

each customer in a matrix format. The lines starting with a # are comments and need not be typed.

The mcut command only extracts the specified field, the mcount command counts the number of rows, and the mcross command performs cross tabulation. You do not have to pay attention to the detailed processes of each command. Instead, capture the flow of the input data, that is, how it is processed by each command. As shown in this example, M-Command has more than 80 commands that can be combined to enable a variety of data processing.

In the examples so far, the results of the commands are output to work files (`xxa, xxb, and xxc`). As shown in Figure 2.7 below, you can execute the same commands without using work files. Use pipes (`I`) to connect the commands, and the output result of a command is handed over as the input of the next command one after another.

```
$ mdata man1 >man1.csv
#END# kgData -man1
$ more man1.csv
顧客,日付,商品
A,20130916,a
A,20130916,c
A,20130917,a
A,20130917,e
B,20130916,d
B,20130917,a
B,20130917,d
B,20130917,f
$ mcut f=顧客:customer,日付:date,商品:product i=man1.csv |
$ mcut f=customer,product o=xxa
#END# kgcut f=顧客:customer,日付:date,商品:product i=man1.csv
#END# kgcut f=customer,product o=xxa
$ more xxa
customer,product
A,a
A,c
A,a
A,e
B,d
B,a
B,d
B,f
$ msortf f=customer,product i=xxa o=xxb
#END# kgsortf f=customer,product i=xxa o=xxb
$ more xxb
customer%0,product%1
A,a
A,a
A,c
A,e
B,a
B,d
B,d
B,f
# Count the number of rows by customer and product.
$ mcount k=customer,product a="number of items" i=xxb o=xxc
#END# kgcount a=number of items i=xxb k=customer,product o=xxc
$ more xxc
customer%0,product%1,number of items
A,a,2
A,c,1
A,e,1
B,a,1
B,d,2
B,f,1
# Perform a cross tabulation by the item of product. The number of the product that is not purchased give
$ mcross k=customer s=product f="number of items" v=0 i=xxc o=xxd
#END# kgcross f=number of items i=xxc k=customer o=xxd s=product v=0
$ more xxd
customer%0,fld,a,c,d,e,f
A,number of items,2,1,0,1,0
B,number of items,1,0,2,0,1
# remove extra item "fld".
$ mcut f=fld -r i=xxd o=output.csv
#END# kgcut -r f=fld i=xxd o=output.csv
$ more output.csv
customer%0,a,c,d,e,f
A,2,1,0,1,0
B,1,0,2,0,1
```

Figure 2.6: Customer-based product-quantity matrix

```
$ mdata -man1 | mcut f=customer,product | mcount k=customer,product a=number of items | mcross k=customer
mcut f=fld -r o=output.csv
#END# mdata -man1
#END# kgcut f=customer,product
#END# kgcount a=number of items k=customer,product
#END# kgcross f=number of items k=customer s=product v=0
#END# kgcut -r f=fld o=output.csv
$ more output.csv
customer%0,a,c,d,e,f
A,2,1,0,1,0
B,1,0,2,0,1
```

Figure 2.7: Using pipes to connect the commands

2.4 CSV

MCMD processes spreadsheet data in CSV format (Comma Separated Values) as illustrated in Figure 2.8. CSV is a *de facto* standard format for spreadsheet data. It is widely used as an intermediate format for data interchange between different application programs.

```
Product code, product name, classification, price
0899781,bread, food,128
8879674,orange juice, beverage,98 3244565,cheese, food,350
6711298,bowl, tableware,168
```

Figure 2.8: Example of CSV data

```
(A) file = [header CRLF] record *(CRLF record) [CRLF]
(B) header = name *(COMMA name)
(C) record = field *(COMMA field)
(D) name = field
(E) field = (escaped / non-escaped)
(F) escaped = DQUOTE *(TEXTDATA / COMMA / CR / LF / 2DQUOTE) DQUOTE
(G) non-escaped = *TEXTDATA
(H) COMMA = %x2C
(I) CR = %x0D ;as per section 6.1 of RFC 2234 [2]
(J) DQUOTE = %x22 ;as per section 6.1 of RFC 2234 [2]
(K) LF = %x0A ;as per section 6.1 of RFC 2234 [2]
(L) CRLF = CR LF ;as per section 6.1 of RFC 2234 [2]
(M) TEXTDATA = %x20-21 / %x23-2B / %x2D-7E
```

Figure 2.9: Definition of ABNF for CSV

The meaning of each line in Figure 2.9 is as follows:

- (A) File consists of a header and record of one or more lines. A header is not required. The line break (CRLF) is attached at the end of the header and at each record. The line break (CRLF) in the last row is not required.
- (B) Header consists of one or more names separated by a single comma.
- (C) Record consists of one or more fields separated by a single comma.
- (D) Name refers to field.
- (E) Field can include an escape character or a non-escape character.
- (F) Field values containing 1 or more text characters (TEXTDATA), comma(COMMA), and a newline character (CR or LF) shall have a pair of consecutive double quotes character escaped by doubling it.
- (G) Non-escape refers to 1 or more text characters (TEXTDATA).
- (H) ASCII code of comma in hexadecimal is 2C.
- (I) ASCII code of carriage return (CR) in hexadecimal is 0D.
- (J) ASCII code of double quotation (DQUOTE) in hexadecimal is 22.
- (K) ASCII code of line feed (LF) in hexadecimal is 0A.
- (L) Line break or newline is represented as a carriage return + line feed.
- (M) Text character (TEXTDATA) had the range of 20-21, 23-2B, 2D-7E in hexadecimal ASCII code.

2.4.1 Definition specific to MCMD

In addition to the above definitions, MCMD has the following restrictions regarding CSV:

- The number of fields must be the same in all the rows.
- There is a limit on the maximum length of a single row (1024000 bytes (1 MB) by default and expandable to 10 MB by changing the value of `KG_LimitRecLen` in the source code kgConfig.h).

- Line break is only marked with Line Feed (LF).
- Line break is mandatory even in the last record.
- Added the 80-FF (hexadecimal) range to text characters for handling multibyte characters.

To check if a CSV file meets the above definitions, use the mchkcsv command.

2.4.2 Common input and output process

The input and output sequence of CSV file for MCMD follows the steps listed below.

1. Read file into memory blocks.
2. Split the comma-delimited string into different fields while taking consideration of escape character.
3. Interpret escape characters and convert to original data (except DQUOTE).
4. Run the specific processing function of the command and write the results to the output buffer.
5. Add character escapes if necessary.
6. Output to a file when buffer is full.

2.4.3 Field sorting information

When the msortf command is used to sort fields or the `s=` or `k=` parameter is used in any other command, records in the specified fields will be sorted. Then, sorting information is added to the fieldnames in the output CSV file. Specifically, numerals will be added to the fieldnames in the order they are specified as sorting fields, starting with a zero: `%0`, `%1`, and so on. When the fields are sorted as numerical values, `n` will be added. When they are sorted in the descending order, `r` will be added. The added information is internally used by the commands for efficiency; the user must not specify it as part of a fieldname.

```
$ cat dat1.csv
id,item,price
2,b2,200
1,a1,100
2,b1,120
$ msortf f=id,price%n i=dat1.csv
id%0,item,price%1n
1,a1,100
2,b1,120
2,b2,200
$ msum k=id f=price i=dat1.csv
id%0,item,price
1,a1,100
2,b1,320
```

To delete the added information, run `mfldname -q` as described below.

```
$ cat dat2.csv
id%0,item,price%1n
1,a1,100
2,b1,120
2,b2,200
$ mfldname i=dat2.csv
id,item,price
1,a1,100
2,b1,120
2,b2,200
```

If the fieldname sorting information does not match the actual sequence of records (for instance, a fieldname includes `%0` although records have not been sorted), the operation of the commands is indefinite.

2.4.4 Notes

Points to note when preparing the CSV data will be described below with examples.

Data containing comma characters

Escape comma characters in data by enclosing them in double quotes. The following is a CSV file comprising of two fields `f1,f2`. The data in row 01 at column `f1` is enclosed in double quotes since it contains a comma.

```
f1,f2
"abc,def",2
xyz,2
```

[1]

Data containing double quotes

Data containing double quotes characters can be represented by a pair of consecutive double quotes. The following is the CSV file that consists of two columns `f1,f2`. Data in row 0 and 1 at column `f1` is escaped with a double quotation. The original data is written as `abc"def` and `"` respectively.

```
f1,f2
"abc""def",2
"""",2
```

Line breaks in data

Data including a line break can be a process when enclosed in double quotes. A line break is included in the data at row 0 in column `f1` after `abc`. Since the data is enclosed in double quotes, it is identified as part of the data instead of the end of the line.

```
f1,f2
"abc
def",1
```

Unnecessary double quotes

Unnecessary double quotes are removed in the output.

```
$ more data.csv
f1,f2
"abc",efg
abc,"efg"
$ mcut f=f1,f2 i=data.csv
f1,f2
abc,efg
abc,efg
```

[1]MCMD address the value of the first row as 0 (except for the field name row) consistently.

2.5 Data Type

MCMD handles plain text files in CSV format, where the data is a sequence of characters. Thus, it depends on how the specific command interprets the character string for the data type. For example, the data in the field specified at `f=` in `msum` command is converted from a character string to a number. As shown in Table 2.1, MCMD can handle six types of data including numeric, character string, date, time, boolean and vector type.

Table 2.1: Table 2.1: Six data types supported by MCMD

Data type	Notation of CSV Data type	Details of Conversion
Numerical type	10, 2.5, 1.5E+10	Double-precision real number
Character string type	abc, あいう	Character string
Date type	20130920	Date object[*1](Gregorian calendar, 8-digit fixed length)
Time type	20130920151154.123456	Date object[*1](Gregorian calendar, 8-digit fixed length)
		+POSIX time[*2](6 digits indicating hhmmss + microseconds in up to 6 digits
	151154.123456	If no date is specified, the date of the day is added internally.
Boolean type	1,0	Convert character to boolean value. ”1 ” is true and “ 0 ” is false
Vector type	a c b, 1 5 11	Character string delimited by space can be converted to any data type above.

[*1] The boost::gregorian::date class in the boost library is used.
[*2] The boost::posix time::ptime class in the boost library is used.

Further, list of data types of commonly used commands is shown in Table 2.2.

Table 2.2: Table 2.2: Data types of commonly used commands

Data type	Command	Details
Numerical type	msum	Calculate total of numeric field
	msim	Calculate the similarity between two fields
String type	mjoin	Combine fields from the reference file
	mcombi	Enumerate combination
Date type	age function of mcal	Calculate Age
	leapyear function of mcal	Determine leap year
Time type	now function of mcal	Output the current time (in seconds)
	unow function of mcal	Output the current time (in microsecconds)
	diffminute function of mcal	Calculate the time difference in minutes
Boolean type	and function of mcal	Compute the logical product
	if function of mcal	Set the value of the criteria
Vector type	mvsort	Sort vector elements
	mvuniq	Extract unique vector elements

2.6 Specify Fields

MCMD reads the field names in the first row of CSV data; the field can also be specified with a field number without the field names. There are four options for handling the row of field names, which are: -nfn,-nfno,-nfni and -x. Its usage will be illustrated with examples as follows. In addition, note that the field number starts at 0 from the left such as 0, 1, 2.

Example 1: Specify -nfn

When `-nfn` (no field name) is specified, the first row in the data will not be considered as field names. Thus, each field is specified as a number (note that the number starts from 0).

```
$ more dat2.csv
a,2
b,5
b,4
$ msum -nfn k=0 f=1 i=dat2.csv o=rsl1.csv
#END# kgsum -nfn f=1 i=dat2.csv k=0 o=rsl1.csv
$ more rsl1.csv
a,2
b,9
```

Example 2: Specify -nfno

When `-nfno` (no field name for output) is specified, the first row of input data is initialized as field names, but the field names is removed from the output data.

```
$ more dat1.csv
key,val
a,2
b,5
b,4
$ msum k=key f=val -nfno i=dat1.csv o=rsl2.csv
#END# kgsum -nfno f=val i=dat1.csv k=key o=rsl2.csv
$ more rsl2.csv
a,2
b,9
```

Example 3: Specify -nfni

The option `-nfni` (no field names or input) is only available in the mcut command. This option does the opposite of - nfno; the first row of input data is not treated as a fieldname row, but the fieldnames will be shown in the output data. Thus, you need to specify an output fieldname after a colon (:) following an input field number.

```
$ mcut f=0:key,1:val -nfni i=dat2.csv o=rsl3.csv
#END# kgcut -nfni f=0:key,1:val i=dat2.csv o=rsl3.csv
$ more rsl3.csv
key,val
a,2
b,5
b,4
```

Example 4: Specify -x

For CSV data with a field names, use the `-x` option to specify the field number.

```
$ msum -x k=0 f=1 i=dat1.csv o=rsl4.csv
#END# kgsum -x f=1 i=dat1.csv k=0 o=rsl4.csv
$ more rsl4.csv
key%0,val
a,2
b,9
```

2.6.1 Valid fieldnames

Fieldnames can contain the characters stated as follows:

- Alphabetic characters (a-z, A-Z)
- Numerals (0-9)
- Multibyte characters (such as UTF-8)
- Symbols

However, it is recommended to avoid using the following symbols. Using the symbols will not return an error; however, the symbols may be used as special characters in MCMD and the special character function may not be available if it is used as a field name.

- `,` Comma
- `:` Colon
- `%` Percent
- `*` Asterisk
- `?` Question mark
- `&` And
- `\` Backslash
- `]` Square brackets, right
- `[` Square brackets, left

2.6.2 Valid item number

When specifying field number, the field numbers can be listed with a comma delimiter. Alternatively, it is also possible to specify the number at the end of the field name (add `"L"`) or specify the range (`-`).

For example, the argument `0L` specifies the last field, and `2L` specifies the 2nd field from the end (note that field number starts from 0). Furthermore, when `0-5` is specified, six fields starting from 0 to 5 are selected, which is equivalent to `0,1,2,3,4,5`.

Example 1: Specify range

By specifying "0-4", fields 0,1,2,3,4 are specified.

```
$ more dat1.csv
brand,quantity01,quantity02,quantity03,quantity04,quantity05,quantity06,quantity07,quantity08,quantity09,
A,10,50,90,130,170,210,250,290,330,370
B,20,60,100,140,180,220,260,300,340,380
C,30,70,110,150,190,230,270,310,350,390
D,40,80,120,160,200,240,280,320,360,400
$ mcut -x f=0-4 i=dat1.csv o=rsl1.csv
#END# kgcut -x f=0-4 i=dat1.csv o=rsl1.csv
$ more rsl1.csv
brand,quantity01,quantity02,quantity03,quantity04
A,10,50,90,130
B,20,60,100,140
C,30,70,110,150
D,40,80,120,160
```

Example 2: Specify range in reverse order

By specifying " 4-0 ", fields 0,1,2,3,4 are specified.

```
$ mcut -x f=4-0 i=dat1.csv o=rsl2.csv
#END# kgcut -x f=4-0 i=dat1.csv o=rsl2.csv
$ more rsl2.csv
quantity04,quantity03,quantity02,quantity01,brand
130,90,50,10,A
140,100,60,20,B
150,110,70,30,C
160,120,80,40,D
```

Example 3: Specify Multiple ranges

By specifying “ 1-0,2-4 ”, fields “ 1,0,2,3,4 ” are specified.EOF scp=¡¡'EOF' mcut -x f=1-0,2-4 i=dat1.csv o=rsl3.csv more rsl3.csv

```
$ mcut -x f=4-0 i=dat1.csv o=rsl2.csv
#END# kgcut -x f=4-0 i=dat1.csv o=rsl2.csv
$ more rsl2.csv
quantity04,quantity03,quantity02,quantity01,brand
130,90,50,10,A
140,100,60,20,B
150,110,70,30,C
160,120,80,40,D
```

Example 4: Specified field from the end

By specifying ”2L”, the second field from the end is specified (quantity 08).

```
$ mcut -x f=2L i=dat1.csv o=rsl4.csv
#END# kgcut -x f=2L i=dat1.csv o=rsl4.csv
$ more rsl4.csv
quantity08
290
300
310
320
```

Example 5: Specify the range of fields from the end

By specifying ”5-3L”, the 5th to the 3rd item from end is specified, i.e. ”5,6,7”.

```
$ mcut -x f=5-3L i=dat1.csv o=rsl5.csv
#END# kgcut -x f=5-3L i=dat1.csv o=rsl5.csv
$ more rsl5.csv
quantity05,quantity06,quantity07
170,210,250
180,220,260
190,230,270
200,240,280
```

2.6.3 Input and output fields

The `f=` parameter is used to specify the field(s) in many commands. The format of `f=` is defined as “ input field:output field ”. If an output fieldname is not specified, the input fieldname will be used as the output fieldname. In addition, you can specify the `-x` option to combine a fieldname with a number, like `f=0:Quantity`.

Example 1: Basic Example

By specifying ”quantity:unit sales”, the field name is converted from “ quantity ” to “ unit sales ” in the output.

```
$ more dat1.csv
brand,quantity
A,10
B,20
C,30
D,40
$ mcut f=brand,quantity:salesquantity i=dat1.csv o=rsl1.csv
#END# kgcut f=brand,quantity:salesquantity i=dat1.csv o=rsl1.csv
$ more rsl1.csv
brand,salesquantity
A,10
B,20
C,30
D,40
```

Example 2: Add field name

The maccum command accumulates the values in the "quantity" field, and add the field name "cumulative quantity" in the output results. If the parameter is specified as "f=quantity", the field name of the cumulative result will remain as "quantity", thus results in error because the same field name " quantity " exists in the output.

```
$ maccum f=quantity:accumulationquantity i=dat1.csv o=rsl2.csv
#ERROR# parameter s= is mandatory without -q,-nfn (kgaccum)
$ more rsl2.csv
$ maccum f=quantity i=dat1.csv o=rsl2.csv
#ERROR# parameter s= is mandatory without -q,-nfn (kgaccum)
```

Example 3: Mixing field name and field number

The field name and field number can be specified at the same time.

```
$ mcut f=0,1:salesquantity -x i=dat1.csv o=rsl3.csv
#END# kgcut -x f=0,1:salesquantity i=dat1.csv o=rsl3.csv
$ more rsl3.csv
brand,salesquantity
A,10
B,20
C,30
D,40
```

2.6.4 Wildcard

The wildcard characters "*" and "?" can be used to specify multiple field names. The asterisk sign "*" matches 0 or more characters, and the question mark "?" matches a single character. Note that the order of evaluation of wildcard characters follows the order of the fields in the input data. For example, if the order of the fields in input data is `A5,A3,A4,A2,A1`, the parameter `f=A*` is evaluated as `f=A5,A3,A4,A2,A1`.

Example 1: Basic Example

The expression "quantity*" matches field names starting with quantity ("quantity10", "quantity11", "quantity12" and "quantity123").

```
$ more dat1.csv
brand,quantity10,quantity11,quantity12,quantity123
A,10,15,9,1
B,20,16,8,2
C,30,17,7,3
D,40,18,6,4
$ mcut f= quantity* i=dat1.csv o=rsl1.csv
#ERROR# invalid argument: quantity* (kgcut)
$ more rsl1.csv
rsl1.csv: No such file or directory
```

Example 2: Wildcard character “ ? ”

Select field names which begin with ”quantity” followed by 1, and match any single character after 1. In this case, the wildcard does not match with field name “ quantity123 ”.

```
$ mcut f= quantity 1? i=dat1.csv o=rsl2.csv
#ERROR# invalid argument: quantity (kgcut)
$ more rsl2.csv
rsl2.csv: No such file or directory
```

2.6.5 Replace the name of an output field

The special character `"&"` specified in the output field name can be replaced with the current field name. For example, the parameter `f=abc:xx&xx` returns `xxabcxx` as the output field name. The `"&"` character can be specified at any position as many times as required in the output field name. However, the ampersand is a special character in shell which is interpreted as ”background execution”. Thus, it is necessary to escape and enclose the field name in double quotes when including `"&"` in field name.

Example 1: Basic Example

In this example, `"&"` is replaced with “ `brand` ” in the input field name, which is equivalent to the expression ”`f=brand:brand code`”.

```
$ more dat1.csv
brand,quantity10,quantity11,quantity12,quantity123
A,10,15,9,1
B,20,16,8,2
C,30,17,7,3
D,40,18,6,4
$ mcut f="brand:& code" i=dat1.csv o=rsl1.csv
#END# kgcut f=brand:& code i=dat1.csv o=rsl1.csv
$ more rsl1.csv
brand code
A
B
C
D
```

Example 2: Combine with wildcard

Attach “ `&` ” after `sales&` to replace the character with input field name (e.g. ”quantity10”) in the output field name. For all input fields name beginning with “ `quantity` ”, attach “ `sales` ” as the prefix in the output field name.

```
$ mcut f="brand,quantity*:sales&" i=dat1.csv o=rsl2.csv
#END# kgcut f=brand,quantity*:sales& i=dat1.csv o=rsl2.csv
$ more rsl2.csv
brand,salesquantity10,salesquantity11,salesquantity12,salesquantity123
A,10,15,9,1
B,20,16,8,2
C,30,17,7,3
D,40,18,6,4
```

2.6.6 Notes on summation commands

A summation command sums records in specified fields for each key field. When one row is output per key field, records to be output are indefinite for non-specified fields. Take the msum command, for example, where there are four fields: customer, date, product, and amount. To sum the amounts for each customer, you would specify `msum k=customer f=amount`. Regarding the records in the date and product fields, the output is indefinite.

2.7 Working with Data Without Field Names

If fields names are included in CSV input data, the set of field names are usually included in the output corresponding to the context of data processing.

On the other hand, if the CSV input data do not include field names, and data is 0 byte file, the result will also return 0 byte file. Both the number of input and output records are 0.

Example 1: Data with field name

```
$ more dat1.csv
A,B,C
$ msetstr v="string" a=X i=dat1.csv o=rsl1.csv
#END# kgsetstr a=X i=dat1.csv o=rsl1.csv v=string
$ more rsl1.csv
A,B,C,X
```

Example 2: Data without field name

```
$ more dat2.csv
$ msetstr v="string" -nfn i=dat2.csv o=rsl2.csv
#END# kgsetstr -nfn i=dat2.csv o=rsl2.csv v=string
$ more rsl2.csv
```

2.8 Multibyte Characters

MCMD handles multibyte characters such as Chinese characters in UTF-8 encoding. Other encodings such as SHIFT_JIS can be treated as multibyte characters, however, some functions may not work correctly. The following explains how MCMD process multibyte characters.

Kanji-code is processed as multibyte characters without conversion in order to increase the processing speed when using MCMD. However, character string search and string substitution functions may result in unexpected results depending on the encoding.

For example, "陰 (shadow)" is represented as 0x8941 in SHIFT_JIS, the second byte of this character refers to "A" in single-byte characters. Thus, when " A " is substituted with " B ", "陰 " will be converted to " 隠 (hidden) " (0x8942). The UTF-8 uses an encoding system which could avoid problems with character substitution. Moreover, it is difficult to count the number of characters in strings containing multibyte characters and ASCII characters even in UTF-8.

This problem can be avoided by converting all characters including ASCII code to fixed length character, known as wide character (MCMD adopts 32-bit fixed length).

When converting wide characters, it is necessary to find out the encoding for multibyte characters in the environment variable `LANG`. Type the following at the command prompt to check the environment variable, .

```
$ echo $LANG
ja_JP.UTF-8
```

Some MCMD commands have built-in option (`-W`) to convert input data to wide characters before data processing. The list of commands which support the option is shown in Table 2.3. These commands pertain to search or replace functions, it is not necessary to use this option if encoding is set as UTF-8.

Table 2.3: List of commands with wide character conversion function

Command name	Function	Description
mchgstr	Substitution	-By specifying -W, the field data specified by f= is converted to wide characters internally.
mselstr	Search	In case of substring matching (-sub), the field data specified by f= is converted to wide characters internally.
msed	Substitution	By specifying -W, the field data specified by f= is converted to wide characters internally.
mtonull	Search	For substring matching (-sub), he field data specified by f= is converted to wide characters internally.

In addition, mcal and msel incorporated functions to handle wide characters (Table 2.4). For instance, the `lengthw` function counts the number of characters and computes the character position for data in UTF-8 encoding.

Take note of the following when handling wide-character.

- Conversion to wide character involves overhead which sacrifices the processing speed.
- Wide characters input data can be converted except for the field names.
- File name with multibyte characters can be processed as it.

Table 2.4: List of mcal functions with wide character conversion function

Name of the function	Function	Details
lengthw	Number of characters	Convert target string to wide character before processing.
midw	Substring	Convert target string to wide character before processing.
rightw	Substring	Convert target string to wide character before processing.
leftw	Substring	Convert target string to wide character before processing.
regexsw	Match regular expression	Convert target string to wide character before processing.
regexmw	Match regular expression	Convert target string to wide character.
regexrepw	Substitute by regular expression	Convert target string to wide character.
regexlenw	Match length by regular expression	Convert target string to wide character.
regexposw	Match position by regular expression	Convert target string to wide character.
regexstrw	Substring match by regular expression	Convert target string to wide character.
regexpfxw	Prefix by regular expression	Convert target string to wide character.
regexsfxw	Suffix match by regular expression	Convert target string to wide character.

2.9 Specify Parameters

The format of the parameters used in M-Command is slightly different than UNIX commands. The keyword and specified value is separated by an equal sign i.e. ”`keyword=value`”. Option type parameters precedes with a minus sign e.g. ”`-keyword`” and do not require specified value.

Many parameters share common functions in M-Command. The parameters are explained below. However, in some command, it works as a completely different function.

Keyword	Description
i=	Input file name
o=	Output file name
f=	Input and output field name
k=	Key field name
s=	Sort field name
a=	Add item name
-nfn	CSV without field name
-nfno	Output without field name
-x	Specify the field number
-q	Disable automatic sorting
[-assert_diffSize]	Compare numbers of inputs and outputs
[-assert_nullkey]	Check whether a key field contains a NULL value
[-assert_nullin]	Check NULL value in the input field specified by f= or vf=
[-assert_nullout]	Check NULL value in output fields
precision=	Number of significant figures
tmpPath=	Work file storage path name
delim=	Delimiter of vector data
bufcount=	Number of buffers
`--help`	Display help

2.9.1 i= Input file name

Specify the name of input file. Most commands only allow a single file to be specified, with the exception of `mcat` command where multiple files can be specified separated with a comma. Yet, certain commands such as `mnewnumber` and `mnewrand` do not require input data.

When this parameter is not defined, data is read from standard input by using pipeline. In the example below, `i=` parameter is not specified for `msum` command because the input data is the result of `msortf`, which is read from standard input through the pipeline.

```
$ msortf f=a i=dat.csv | msum k=a f=b o=rsl.csv
```

However, it is difficult to identify errors when results are piped directly from one command to the next. In the following example, `i=` parameter is also specified for `msum`. The results of `msortf` is sent to standard output, and msum reads input data from `dat.csv`. Since `msortf` did not add meaning to the input for `msum`, the results from this example is different from the above.

```
$ msortf f=a i=dat.csv | msum k=a f=b i=dat.csv o=rsl.csv
```

Examples

Example 1: Basic Example

Run `mcut` using `dat1.csv` as input data.

```
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
$ mcut f=customer,amount i=dat1.csv o=rsl1.csv
#END# kgcut f=customer,amount i=dat1.csv o=rsl1.csv
```

```
$ more rsl1.csv
customer,amount
A,10
A,20
```

Example 2: Specify output field name

Read standard input using redirection (”"<"”).

```
$ mcut f= customer, amount o=rsl2.csv <dat1.csv
#ERROR# invalid argument: customer, (kgcut)
$ more rsl2.csv
rsl2.csv: No such file or directory
```

Related commands

The parameter can be used in all commands except for commands such as `mnewnumber` and `mnewrand`.

2.9.2 o= Output file name

Specify the name of output file. Most commands only allow specification of a single file name, with the exception of `mtee` command where multiple files can be specified. There is also the command that does not require output data, for example, `msep`.

When this parameter is not defined, data is read from standard input through pipeline. In the following example `o=` is not specified in `msortf` because the output data is sent to standard output through pipeline.

```
$ msortf f=a i=dat.csv | msum k=a f=b o=rsl.csv
```

The example below is similar to the above. The difference is that o= parameter is specified for the `msortf` and the result of `msortf` is saved to `tmp.csv`. Even though the two commands are connected with pipeline, there is no data stream from standard output to `msum`, the receiving process could not read data from pipeline and stays idle.

```
$ msortf f=a i=dat.csv o=tmp.csv | msum k=a f=b o=rsl.csv
```

Below is a more complicated example by using `mtee` to connect the data streams between the two commands.

```
$ msortf f=a i=dat.csv | mtee o=tmp.csv | msum k=a f=b o=rsl.csv
```

The `mtee` command writes to a standard input file specified at `o=` and send the data to standard output concurrently. The results of `msortf` is written to `tmp.csv`, at the same time, `msum` receives the data stream through pipeline from `mtee`. The final result is saved to `rsl.csv`.

Examples

Example 1: Basic Example

The result of `mcut` is saved to `rsl1.csv` as specified in `o=` parameter.

```
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
$ mcut f=customer,amount i=dat1.csv o=rsl1.csv
#END# kgcut f=customer,amount i=dat1.csv o=rsl1.csv
$ more rsl1.csv
customer,amount
A,10
A,20
```

Example 2: Redirect

Write to standard input using redirection (">").

```
$ mcut f=customer,amount i=dat1.csv >rsl2.csv
#END# kgcut f=customer,amount i=dat1.csv
$ more rsl2.csv
customer,amount
A,10
A,20
```

Related commands

This parameter can be used in all commands except for certain commands such as `sep`.

2.9.3 f= Input and output field name

Specify the input and output field name for processing. For example, this parameter specifies the "field name to select" in mcut, "field name to aggregate" for magg, and "field name to merge" for mjoin. In addition, multiple field names can be specified separated by a comma in between such as `f=a,b,c`.

The output field name for every specified item from the input file can be renamed in MCMD. This can be done by defining the input field name and output field name separated by a colon in between e.g. `f=a:A,b:B`. The field name in the output remains the same if the output field name is not specified.

Examples

Example 1: Basic Example

Extract fields `val1` and `val2`.

```
$ more dat1.csv
id,val1,val2
A,1,2
B,2,3
C,3,4
$ mcut f=val1,val2 i=dat1.csv o=rsl1.csv
#END# kgcut f=val1,val2 i=dat1.csv o=rsl1.csv
$ more rsl1.csv
val1,val2
1,2
2,3
3,4
```

Example 2: Specify name of output field

Aggregate `val1,val2`, and rename the fields in the output as `sum1,sum2` respectively.

```
$ msum f=val1:sum1,val2:sum2 i=dat1.csv o=rsl2.csv
#END# kgsum f=val1:sum1,val2:sum2 i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,sum1,sum2
C,6,9
```

Related commands

mcut, msum, mcat, mjoin, etc.

2.9.4 k= Key field name

Specify the key field name. A key field uniquely identifies individual rows or an entity in the data, it is used as unit of aggregation, or used as common key for joining fields between two files.

For example, in `msum` command, aggregate computation is carried out for records with the same key (aggregate key break processing). Whereas in `mjoin` command, the size of key items in the two data files are compared (join key break processing) and joined.

When `k=` command is specified, the field(s) specified are first sorted in character string ascending order, afterwards, corresponding processing is carried out.

and is considered as the default field for sorting character strings in ascending order (except for mhashsum). Key break process refers to the processing method for every same key field with the same value assuming that the items are sorted beforehand (However, mhashsum command is an exception).

For details on key break process, please refer to Key break processing. Since frequent sorting may decrease the processing performance, understanding the need for key break processing would help reduce the instances for sorting, desirable for optimizing script performance.

Examples

Example 1: Basic Example

Compute sum on `val` column by `id`.

```
$ more dat1.csv
id,val
A,1
B,1
B,2
A,2
B,3
$ msum i=dat1.csv k=id f=val o=rsl1.csv
#END# kgsum f=val i=dat1.csv k=id o=rsl1.csv
$ more rsl1.csv
id%0,val
A,3
B,6
```

Example 2: Join Process

Use the join key “ id ” from `dat1.csv`, and join the field “ name ” from `ref1.csv`.

```
$ more dat1.csv
id,val
A,1
B,1
B,2
A,2
B,3
$ more ref1.csv
id,name
A,nysol
B,mcmd
$ mjoin k=id i=dat1.csv m=ref1.csv f=name o=rsl4.csv
#END# kgjoin f=name i=dat1.csv k=id m=ref1.csv o=rsl4.csv
$ more rsl4.csv
id%0,val,name
A,1,nysol
A,2,nysol
B,1,mcmd
B,2,mcmd
B,3,mcmd
```

Related commands

msum, mslide, mjoin, mrjoin, mcommon, etc.

2.9.5 s= Sort Field Name

Specify the field name for sorting (multiple fields can be specified).

The order of records affects the process results for some commands such as `maccum`. When `s=` parameter is specified, sorting is carried out on the specified fields before the processing command.

There are four combinations of sorting methods (order), including numeric / string, and ascending / descending order. The sorting methods can be specified by appending `%` followed by `n` or `r` after the column name. The examples are as follows.

Character string ascending order: `field` (`%` not required), character string descending order: `f=field%r`, numeric ascending order: `f=field%n`, numeric descending order:`f=field%nr`.

Example

Example 1: Basic Example

After sorting by `id`, calcuate the cumulative sum on `val` column.

```
$ more dat1.csv
id,val
A,1
B,1
B,2
A,2
B,3
$ maccum s=id k=id f=val:val_accum i=dat1.csv o=rsl1.csv
#END# kgaccum f=val:val_accum i=dat1.csv k=id o=rsl1.csv s=id
$ more rsl1.csv
id,val,val_accum
A,1,1
A,2,3
B,1,1
B,2,3
B,3,6
```

Example 2: Specify sort method

After sorting the `val` field in descending numerical order, calculate the cumulative sum on `val` column.

```
$ more dat1.csv
id,val
A,1
B,1
B,2
A,2
B,3
$ maccum s=id,val%nr k=id f=val:val_accum i=dat1.csv o=rsl1.csv
#END# kgaccum f=val:val_accum i=dat1.csv k=id o=rsl1.csv s=id,val%nr
$ more rsl1.csv
id,val,val_accum
A,2,2
A,1,3
B,3,3
B,2,5
B,1,6
```

Corresponding Commands

maccum, mbest, mmvavg, mnumber, mslide, etc.

2.9.6 a= Add field name

Add an additional field (column) according to the field name specified. Most commands add the result in 1 field, thus, only 1 field is specified at this parameter. Nevertheless, `mcombi` returns multiple fields as output, thus multiple field names are specified delimited by comma.

Examples

Example 1: Basic Example

Add a new field as " payday ".

```
$ more dat1.csv
id
A
B
C
$ msetstr v=20070101 a=payday i=dat1.csv o=rsl1.csv
#END# kgsetstr a=payday i=dat1.csv o=rsl1.csv v=20070101
$ more rsl1.csv
id,payday
A,20070101
B,20070101
C,20070101
```

Example 2: Add multiple fields

Enumerate the two combination of each item `A,B,C` in the column " id ".

```
$ mcombi f=id n=2 a=id1,id2 i=dat1.csv o=rsl2.csv
#END# kgcombi a=id1,id2 f=id i=dat1.csv n=2 o=rsl2.csv
$ more rsl2.csv
id,id1,id2
C,A,B
C,A,C
C,B,C
```

Related command

mcal, mcombi, mrand, msetstr etc.

2.9.7 -nfn CSV without field names (No Field Names)

This option reads input data without field names. When this option is specified, the field number is used instead of the field name to specify the field. The field number begins from the integer 0 and increments by 1 from the left onwards. When `--nfn` option is specified, the field name will not be included in the output file.

Examples

Example 1: Basic Example

Extract column0 and 2.

```
$ more dat1.csv
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ mcut -nfn f=0,2 i=dat1.csv o=rsl1.csv
#END# kgcut -nfn f=0,2 i=dat1.csv o=rsl1.csv
$ more rsl1.csv
```

```
A,10
A,20
B,15
B,10
B,20
```

Related command

This option can be used in all M-Commands except mchkcsv.

2.9.8 -nfno Output with field names (No Field Names for Output)

This option allow users to remove field names from the output data. Unlike `--nfn`, this option assumes that input data specified at `i=` and `m=` includes field names in the first row.

Examples

Example 1: Basic Example

Extract column0 and 2.

```
$ more dat1.csv
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ mcut -nfn f=0,2 i=dat1.csv o=rsl1.csv
#END# kgcut -nfn f=0,2 i=dat1.csv o=rsl1.csv
$ more rsl1.csv
A,10
A,20
B,15
B,10
B,20
```

Related commands

This option can be used in all commands except mchkcsv.

2.9.9 -x Specify by item number

This option allows user to specify a column with corresponding field number where input data includes field names. Users can specify the output field name(s) by adding colon right after input field, followed by the output field name.

Examples

Example 1: Basic Example

Compute the sum of all items in column 1 and 2 of the same key.

```
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ msum -x k=0 f=1,2 i=dat1.csv o=rsl1.csv
#END# kgsum -x f=1,2 i=dat1.csv k=0 o=rsl1.csv
```

```
$ more rsl1.csv
customer%0,quantity,amount
A,3,30
B,5,45
```

Example 2: Output column names

Rename column 1 and 2 as `a,b` respectively.

```
$ msum -x k=0 f=1:a,2:b i=dat1.csv o=rsl2.csv
#END# kgsum -x f=1:a,2:b i=dat1.csv k=0 o=rsl2.csv
$ more rsl2.csv
customer%0,a,b
A,3,30
B,5,45
```

Example 3: Error when using -nfn

The `-nfn` option assumes data starts from the first row when computing the sum of ”quantity” and ”amount”. However, the result will not be computed as expected since the position of first row of data is defined differently when using `-x` and `-nfn`.

```
$ msum -nfn k=0 f=1,2 i=dat1.csv o=rsl3.csv
#END# kgsum -nfn f=1,2 i=dat1.csv k=0 o=rsl3.csv
$ more rsl3.csv
customer,0,0
A,3,30
B,5,45
```

Related commands

This option can be used in all commands except mchkcsv.

2.9.10 -q Disable Automatic Sorting

Use this option to disable automatic sorting on fields specified at `k=` parameter.

The `s=` option is not required when `k=` parameter is defined at the same time, therefore, each command operates the same as MCMD Ver. 1.0.

Example

Example 1: Basic Example

Find out the cumulative value by `id` field. When `-q` option is specified, sorting by field specified at `k=` parameter will be disabled.

```
$ more dat1.csv
id,val
A,1
B,1
B,2
A,2
B,3
$ maccum -q k=id f=val:val_accum i=dat1.csv o=rsl1.csv
#END# kgaccum -q f=val:val_accum i=dat1.csv k=id o=rsl1.csv
$ more rsl1.csv
id,val,val_accum
A,1,1
B,1,1
B,2,3
A,2,2
B,3,3
```

Corresponding Commands

This function is available in all commands where `k=` parameter exists.

2.9.11 -assert_diffSize Compare numbers of inputs and outputs

Specify this option to compare the numbers of the input and output files. When the numbers do not match, the " `#WARNING# ; the number of lines is different` " message is shown.

Example

(Basic example)

Assume, for instance, that you use the mjoin command (join fields from the reference file) and wish to check whether the key fields from the input file (the fields specified by the k= parameter) match the key fields from the reference file (the fields specified by the K= parameter). When the -n or -N option for outputting NULL values is not specified for the mjoin command, the numbers of input and output date items differ. This is because only the key fields that are common to the input and reference files are joined and the values of the other key fields are disregarded. Specify the -assert_diffSize option to compare the numbers of the input and output files. When the numbers do not match, the " `#WARNING# ; the number of lines is different` " message is shown, indicating that the key fields in the input and reference files do not match completely.

```
$ more dat1.csv
item,date,price
A,20081201,100
A,20081213,98
B,20081002,400
B,20081209,450
C,20081201,100

$ more ref1.csv
item,cost
A,50
B,300
E,200

$ mjoin k=item f=cost m=ref1.csv -assert_diffSize i=dat1.csv o=rsl1.csv
#WARNING# ; the number of lines is different
#END# kgjoin -assert_diffSize f=cost i=dat1.csv k=item m=ref1.csv o=rsl1.csv; IN=5 OUT=4

$ more rsl1.csv
item%0,date,price,cost
A,20081201,100,50
A,20081213,98,50
B,20081002,400,300
B,20081209,450,300
```

Related commands

This option can be used for all commands except for the following:
marff2csv, mchkcsv, mcsv2arff, mnewnumber, mnewrand, mnewstr, msep, msep2, mtee, mxml2csv

2.9.12 -assert_nullkey Check whether a key field contains a NULL value

Specify this option to check whether a key field (a field specified by the k= or K= parameter) contains a NULL value. When a NULL value is contained, the " `#WARNING# ; exist NULL in key filed` " message is shown.

Example

(Basic example)

Assume, for instance, the msum command (sum the field values) is used. When the values in the fields specified by the f= parameter is summed for the rows that have the same value in the fields specified by the k= parameter, the value of the key field specified by the k= parameter may contain a NULL value. Specify the -assert_nullkey option to check whether a key field contains a NULL value. When a NULL value is contained, the “ `#WARNING# ; exist NULL in key filed` ” message is shown.

```
$ more dat1.csv
customer,quantity,金額
A,1,10
,1,10
B,1,15
A,2,20
B,3,10
B,1,20

$ msum k=customer f=quantity:quantityT,Amount:AmountT -assert_nullkey i=dat1.csv o=rsl1.csv
#WARNING# ; exist NULL in key filed
#END# kgsum -assert_nullkey f=quantity:quantityT,Amount:AmountT i=dat1.csv k=customer o=rsl1.csv

$ more rsl1.csv
customer%0,quantityT,AmountT
,1,10
A,3,30
B,5,45
```

Related commands

This option can be used for the following commands:
maccum, mavg, mbest, mbucket, mcal, mcommon, mcount, mcross, mdelnull, mhashavg, mhashsum, mjoin, mkeybreak, mmbucket, mmvavg, mmvsim, mstats, mnjoin, mnormalize, mnrcommon, mnrjoin, mnumber, mpadding, mrand, mrjoin, mselnum, mselrand, mselstr, msep2, mshare, msim, mslide, mstats, msum, msummary, mtra, muniq, mwindow

2.9.13 -assert_nullin -assert nullin Check NULL value in the input field specified by f= or vf=

Specify this option to check whether the input field specified by f= or vf= contains a NULL value. When a NULL value is contained, the “ `#WARNING# ; exist NULL in input data` ” message is shown.

Example

(Basic example)

Assume, for instance, the maccum command (accumulation) is used. The maccum command ignores the fields specified by the f= parameter if they contain a NULL value. Specify the -assert_nullin option to check whether a key field specified by the f= parameter contains a NULL value. When a NULL value is contained, the “ `#WARNING# ; exist NULL in input data` ” message is shown.

```
$ more dat1.csv
customer,quantity,amount
A,1,
A,2,20
B,1,15
B,3,10
B,,20

$ maccum s=customer f=quantity:quantityC,amount:amountC -assert_nullin i=dat1.csv o=rsl1.csv
#WARNING# ; exist NULL in input data
#END# kgaccum -assert_nullin f=quantity:quantityC,amount:amountC i=dat1.csv o=rsl1.csv s=customer
```

```
$ more rsl1.csv
customer%0,quantity,amount,quantityC,amountC
A,1,,1,
A,2,20,3,20
B,1,15,4,35
B,3,10,7,45
B,,20,,65
```

Related commands

This option can be used for all commands except for the following:
marff2csv, mbest, mbucket, mchkcsv, mcommon, mcount, mdelnull, mfldname, mkeybreak, mnewnumber, mnewrand, mnewstr, mnrcommon, mnullto, mnumber, mrand, msel, mselrand, msep2, msetstr, msortf, mtee, muniq, mwindow, mxml2csv

2.9.14 -assert_nullout Check NULL value in output fields

Specify this option to check whether an output field contains a NULL value. When a NULL value is contained, the “ `#WARNING# ; exist NULL in output data` ” message is shown. This option does not work on fields in which the input data is output as is, such as the calculation field.

Example

(Basic example)

Assume, for instance, the mslide command (slide rows) is used. When the values of the fields specified by the f= parameter are slid for the number of rows specified by the t= parameter for each field specified by the k= parameter, the number of slides specified by the t= parameter can be greater than the number of rows of the fields specified by the f= parameter depending on the data. If that is the case, the -n option can be specified to output a field with a NULL value in it in the absence of the next (or previous) row. Specify the -assert_nullout option to check whether an output field contains a NULL value. When a NULL value is contained, the “ `#WARNING# ; exist NULL in output data` ” message is shown.
In the example below, the option is used to check whether the syo_1 and syo_2 output fields contain a NULL value. Since the two fields contain a NULL value, the “ `#WARNING# ; exist NULL in output data` ” message is shown.

```
$ more dat1.csv
customer,date,product,quantity
A,20130406,a,1
A,20130408,b,1
A,20130416,c,1
B,20130407,k,2
C,20130408,d,1
C,20130409,e,4

$ mslide s=customer,date k=customer f=product:syo_ t=2 -n -assert_nullout i=dat1.csv o=rsl1.csv
#WARNING# ; exist NULL in output data
#END# kgslide -assert_nullout -n f=product:syo_ i=dat1.csv k=customer o=rsl1.csv s=customer,date t=2

$ more rsl1.csv
customer,date,product,quantity,syo_1,syo_2
A,20130406,a,1,b,c
A,20130408,b,1,c,
A,20130416,c,1,,
B,20130407,k,2,,
C,20130408,d,1,e,
C,20130409,e,4,,
```

Related commands

This option can be used for all commands except for the following:
mbest, mcat, mcombi, mcommon, mcount, mcsv2arff, mcut, mdelnull, mduprec, mfldname, mfsort, mnewnumber, mnewrand, mnewstr, mnrcommon, mnullto, mnumber, mproduct, mrand, msel, mselnum, mselrand, mselstr, msep, msep2, msetstr, msortf, mtee, mtonull, muniq, mvcount, mwindow, mxml2csv

2.9.15 precision= Number of significant digits

Applies sprintf format `["%.`*n*`g"]` in C language. This format converts the number of significant figures defined from normalized notation (integer bits, decimal bits: ex.`123.456`) to exponent notation (mantissa e± exponent part: ex. `1.23456e+02`). The criteria to adopt exponent notation for conversion is when the exponent bits exceed the specified number of significant digits or if it is less than or equal to -5 (i.e more than 4 zeros after decimal points).

Integers between 1 to 16 can be specified in n, the default value is 10. When $n < 1$, set $n = 1$, and when $n > 16$ set to $n = 16$.

In addition, the number of significant figures can be changed by setting the environment variable `KG_Precision`. However, changes to the environment variable will affect the execution of all commands.

Examples

Example 1: Basic Example

The exponential notation of id=1 is 1.2345678e +08, the exponent bits is more than 6 significant figures when the significant figures of mantissa is set at 6. The exponential notation of id=2 is 1.23456789e +03, the exponent bits is more than 7 significant figures when the significant figures of integer bits + decimal bits is set at 6. The exponential notation of id=4 is 1.23456789e-04, the exponent bits is less than -4 when the significant figures is set at 6. The exponential notation of id=5 is 1.23456789e-05, the exponent bits is less than -4 when the significant figures of mantissa is set at 6.

```
$ more dat1.csv
id,val
1,123456789
2,1234.56789
3,0.123456789
4,0.000123456789
5,0.0000123456789
$ mcal c='${val}' a=result precision=6 i=dat1.csv o=rsl1.csv
#END# kgcal a=result c=${val} i=dat1.csv o=rsl1.csv precision=6
$ more rsl1.csv
id,val,result
1,123456789,1.23457e+08
2,1234.56789,1234.57
3,0.123456789,0.123457
4,0.000123456789,0.000123457
5,0.0000123456789,1.23457e-05
```

Example 2: Case when precision=2

```
$ mcal c='${val}' a=result precision=2 i=dat1.csv o=rsl2.csv
#END# kgcal a=result c=${val} i=dat1.csv o=rsl2.csv precision=2
$ more rsl2.csv
id,val,result
1,123456789,1.2e+08
2,1234.56789,1.2e+03
3,0.123456789,0.12
4,0.000123456789,0.00012
5,0.0000123456789,1.2e-05
```

Example 3: Specify the environment variable

When the environment variable is set, the setting will be applied to all commands in subsequent processes.

```
$ export KG_Precision=4
$ mcal c='${val}' a=result i=dat1.csv o=rsl3.csv
#END# kgcal a=result c=${val} i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,val,result
1,123456789,1.235e+08
2,1234.56789,1235
3,0.123456789,0.1235
4,0.000123456789,0.0001235
5,0.0000123456789,1.235e-05
```

Related commands

This setting applies to all commands for calculating real numbers which is used in msum,mcal.

2.9.16 tmpPath= Path name of temporary file

Specify the name of the directory which stores the temporary files for use by the command. For example, the results from `msortf` is saved as a temporary file during partitioned sort. If the path is not specified, the file is saved in `/tmp`. The name of temporary files begins with `__KGTMP`.

The temporary files are deleted if the command terminates normally (includes termination by exit signal, or termination by signal from MCMD signal). Temporary files will be retained in the directory when the program is terminated unexpectedly by power outage or bug.

Depending on the amount of data, enormous amount of temporary data may be generated (more than 1 million files). This will significantly slow down the execution of commands, therefore, clean out the files in the temporary path on a regular basis. Currently there is no plans to implement functions for garbage collection to remove objects no longer used by the program.

The temporary directory can be changed by setting the environment variable `KG_Tmp_Path`, however, the same variable applies to the execution of all commands.

Examples

Example 1: Basic Example

Set the `tmp` directory under the current directory for temporary files.

```
$ msortf f=val tmpPath=./tmp i=dat1.csv o=rsl1.csv
#ERROR# internal error: cannot create temp file (kgsortf)
```

Example 2: Specify the environment variable

The settings of the environment variable will be applied to subsequent commands.

```
$ export KG_TmpPath=~/tmp
$ msortf f=val i=dat1.csv o=rsl1.csv
#END# kgsortf f=val i=dat1.csv o=rsl1.csv
```

Related commands

This applies to commands such as msortf and mdelnull which select records by key field, and commands such as mbucket, mnjoin, and mshare that require multiple pass scanning based on key field.

2.9.17 delim= Delimiter of vector element

Specify the delimiter for elements in vector data. The default delimiter is 1 byte space. When comma is specified as the delimiter for the vector, the vector is enclosed in double quotes to avoid confusion with the comma delimiter in CSV file.

Examples

Example 1: Basic Example

Sort the elements of the vector field “ `vec` ” with colon as a delimiter.

```
$ more dat1.csv
vec
b:a:c
x:p
$ mvsort vf=vec delim=: i=dat1.csv o=rsl1.csv
#END# kgvsort delim=: i=dat1.csv o=rsl1.csv vf=vec
$ more rsl1.csv
vec
a:b:c
p:x
```

Example 2: When delimiter is not specified

`b:a:c`, `x:p`, etc are intepreted as one element when `delim` is not specified.

```
$ mvsort vf=vec i=dat1.csv o=rsl2.csv
#END# kgvsort i=dat1.csv o=rsl2.csv vf=vec
$ more rsl2.csv
vec
b:a:c
x:p
```

Example 3: Use comma as delimiter

If comma is used as delimiter for the vector, the entire vector is enclosed by double quote to draw distinction between the delimiter of CSV and the delimiter of the vector.

```
$ more dat2.csv
id,vec1,vec2
1,a,b
2,p,q
$ mvcat vf=vec1,vec2 a=vec3 delim=, i=dat2.csv o=rsl3.csv
#END# kgvcat a=vec3 delim=, i=dat2.csv o=rsl3.csv vf=vec1,vec2
$ more rsl3.csv
id,vec3
1,"a,b"
2,"p,q"
```

Related commands

This parameter can be used in all vector related commands such as such as mvcat and mvsort.

2.9.18 bufcount= Buffer size

Specify the internal buffer size (number of blocks) to be used in commands such as mbucket, mnjoin, and mshare, for processing key units at which data requires multiple pass scanning. One buffer block contains 4MB, the default size is 10 blocks (40MB). In case of buffer overflow, data is written to a temporary file. If the key size is very large, the processing speed can be improved by adjusting this parameter if memory permits.

Examples

Example 1: Basic Example

If the key size of the reference file is less than 80MB (4MB × 20), the temporary file will not be used.

```
$ mnjoin k=id m=ref.csv f=name i=dat.csv o=rsl.csv bufcount=20
#END# kgnjoin bufcount=20 f=name i=dat.csv k=id m=ref.csv o=rsl.csv
```

Related command

Commands that require multiple pass scanning of the data to process key units, such as mbucket, mnjoin, and mshare.

2.10 Environment Variable

Shell environment variable can be set in MCMD to customize the settings for commands. The environment variables that can be set for MCMD are listed in Table 2.5.

Table 2.5: List of the environment variable to configure KGMOD

Variable name	Default value	Description
`KG_iSize`	4096000	Size of data read for each time Four times of `KG_iSize` is allocated for input buffer. However, 10 times of the command parameters is allocated for kgsortf. Refer to `KG_BlockCount` for command 's buffer size on processing by key block. The following conditions must be satisfied. `KG_iSize`=`KG_MaxRecLen`∗i (given i is an integer greater than 1) `KG_iSize`≥`KG_MaxRecLen`∗2
`KG_oSize`	2048000	Size of data written for each time The same memory size in `KG_oSize` is allocated for output buffer. `KG_oSize`=`KG_MaxRecLen`∗i (given i is an integer greater than 1) `KG_oSize`≥`KG_MaxRecLen`∗2
`KG_MaxRecLen`	1024000	The maximum number of characters per row (maximum:10240000) It must satisfy the conditions shown in `KG_iSize` and `KG_oSize`.
`KG_BlockCount`	128	The number of buffers required when processing$^{1)}$ by key. The column `KG_iSize` shows buffer size∗`KG_BlockCount` is allocated. `KG_MaxRecLen` ∗2 + 4∗ `KG_ioSize`) ∗ `KG_BlockCount`
`KG_TmpPath`	/tmp	The directory of temporary files in the default directory used by library functions.
`KG_Precision`	10	Significant figures
`KG_VerboseLevel`	4	Output error message for M-Command 0: Do not print any messages. 1: + error message output 2: + warning message output 3: + end message output 4: + msg message output (default)
`KG_msgTimebyfsec`	false	false: The end message time is shown in seconds. ture: The end message time is shown in microseconds.

Note 1) Processing by key, such as `mnjoin`, refers to reading data with same key into the memory at once. Refer to the manual of corresponding command for more details.

Below is an example of changing the settings of `KG_VerboseLevel` to control the command message.

Examples

Example 1: End message

The default setting of output message is set as `KG_Verbose=4` which displays a message when the program finished processing normally or terminates upon an error.

```
$ more dat.csv
k,v
A,1
B,2
$ mcut f=k,v i=dat.csv o=out.csv
#END# kgcut f=k,v i=dat.csv o=out.csv
$ mcut x=k,v i=dat.csv o=out.csv
#ERROR# unknown parameter x= (kgcut)
```

Example 2: Only display error messages

If `KG_Verbose=1`, the stop error message is displayed, but not the normal exit message.

```
$ export KG_VerboseLevel=1
$ mcut f=k,v i=dat.csv o=out.csv
$ mcut x=k,v i=dat.csv o=out.csv
#ERROR# unknown parameter x= (kgcut)
```

Example 3: Do not display any messages

If KG_Verbose=0, no message will be displayed.

```
$ export KG_VerboseLevel=0
$ mcut f=k,v i=dat.csv o=out.csv
$ mcut x=k,v i=dat.csv o=out.csv
```

2.11 Key Break Processing

In key break processing, it is assumed that within the column which matches the column specified, processing is executed for key fields with the same value. Key break processing is broadly divided into two type of processes. First is key break processing for aggregate calculation (referred to as "**aggregate key break processing**" below), second is key break processing for joins (referred to as "**join key break processing**" below).

Join key break processing is executed for commands such as mjoin, mcommon which contains the word "join" and "common". Aggregate key break processing is carried out on other commands with `k=` parameter.

For example, when the `msum` command triggers aggregate key break processing, it detects the change of value in the key field, and executes aggregate processing for records with the same key. That is, the records need to be sorted according to the key field beforehand (unless the input files are sorted in advance). Thus, the msum command internally sorts the records before aggregate processing. Join key break processing involves a more complicated process. For instance, the `mjoin` command takes in two data files, and compare the values in the key field. The key fields from the smaller data set is read continuously, and the records are joined when the key fields in the input file and the reference file matches. When the comparing the key field values, since the key break processing is used for join operation, the key fields from the two data files need to be sorted beforehand. Therefore, the `mjoin` command internally sorts the records in the two data files first.

Basic sorting **character string ascending order** is carried out for both key break processing, however, when joining records by numerical range in `mrjoin` command, sorting is carried out by **numeric ascending order**.

Besides the fields defined at `k=` parameter are automatically sorted, in other commands automatic sorting is pre-determined, thus users do not need to resolve whether the input files requires sorting. Even though users no longer need to initiate the sort command, note that sorting is handled within each command internally. Thus, depending on the construction of the script, sort processing may frequently take place which could reduce performance.

Example of Script

Example of script when sorting takes place frequently

Initially, `name` column is sorted and saved as xxcustomer output file, afterwards, join processing by `id` key field is carried out by `mjoin` command. In this case, `mjoin` is executed three times, and `id` column from xxcustomer inputer data is sorted at each instance of `mjoin` command.

```
mcut   i=customer.csv f=id,name |
msortf f=name o=xxcustomer

mjoin i=xxcustomer m=address.csv k=id f=address o=cust_address.csv
mjoin i=xxcustomer m=phone.csv   k=id f=phone   o=cust_phone.csv
mjoin i=xxcustomer m=age.csv     k=id f=age     o=cust_age.csv
```

Example of script to minimize sorting

When the script is modified as follows, since xxcustomer file is sorted by `id` field and saved as xxcustomer. Automatic sorting of the input file at `mjoin` commands is not carried out.

```
mcut   i=customer.csv f=id,name |
msortf f=id o=xxcustomer

mjoin i=xxcustomer m=address.csv k=id f=address o=cust_address.csv
mjoin i=xxcustomer m=phone.csv   k=id f=phone   o=cust_phone.csv
mjoin i=xxcustomer m=age.csv     k=id f=age     o=cust_age.csv
```

Chapter 3

Convenient tools

3.1 bash completion

The bash-completion tool assists command input on the command line. To use the input completion functions shown below,install the bash-completion configuration file for MCMD. It will greatly enhance the ease of using MCMD on the command line.

- Listing parameters available for each command and completing them
- Completing input/output filenames
- Completing fieldnames in the input CSV data
- Completing values of choice parameters

3.1.1 Installation

To check whether bash-completion has been installed, check whether the `complete` command can be used as described below. If it has not been installed, see the development page and install it. You can search the Internet to find simple installation procedures for different platforms.

```
$ complete --help
-bash: complete: --: invalid option
complete: usage: complete [-abcdefgjksuv] [-pr] [-o option] [-A action]
[-G globpat] [-W wordlist] [-P prefix] [-S suffix] [-X filterpat] [-F function]
[-C command] [name ...]
```

For the MCMD configuration file, find the `bash_completion_mcmd.sh` file in the root of the `mcmd/unitilies` directory of the MCMD source tree, copy it to the root of the `home` directory, and rename it.

```
$ cp mcmd/unitilies/bash_completion_mcmd.sh ~/.bash_completion_mcmd.sh
```

Then, as described below, add an instruction to run the script to `~/.bash_profile`. That way, the configuration is automatically loaded each time you log in.

```
. ~/.bash_completion_mcmd.sh
```

3.1.2 Completing parameters

After typing a command name, press the `TAB` key twice to list the parameters and options that can be specified for that command. As an example, the following shows the list for the `msum` command.

```
$ msum [Press TAB twice]
-assert_diffSize  -assert_nullkey   -n      -nfno     -q      f=      k=      precision=
-assert_nullin    -assert_nullout   -nfn    -params   -x      i=      o=      tmpPath=
```

With the above list displayed, typing `p` and pressing `TAB` completes the keyword character string "`precision=` " because it is the only parameter name that starts with a `p`. Following the completion, a space is inserted after the keyword. To enter another value, you need to move back one character.

```
$ msum p[Press TAB]
# The string will be completed as shown below.
$ msum precision=
```

Typing `-a` and pressing TAB completes the keyword character string only up to " `-assert_` " because it is common to the multiple options starting with -a. Then, pressing TAB again shows the four parameters that start with `-assert_`.

```
$ msum -a[Press TAB]
-assert_diffSize  -assert_nullin    -assert_nullkey   -assert_nullout
```

3.1.3 Completing input/output filenames

In MCMD, several parameters are used to specify a filename or a directory name, such as `i=`, `m=`, and `o=`. You can use filename completion by typing any of such parameters and pressing TAB. Typing `=` only and pressing TAB lists the files in the root of the current directory. Typing a certain character string and pressing TAB shows the one filename or multiple candidate filenames starting with that character string. Only one filename can be completed per command, except for the `mcat` command. When using the mcat command, you can enter multiple filenames by completion by pressing TAB after a comma.

```
$ msum i=[Press TAB twice]
test1.csv   test2.csv ...

$ mcat i=test1.csv,[Press TAB twice]
test1.csv   test2.csv ...
```

3.1.4 Completing fieldnames

In MCMD, many parameters are used to specify a fieldname. If an input file has already been entered, the fieldnames in the file can be entered by completion.

```
$ cat test1.csv
codeA,codeB,date,value1,value2
A,aaa,20170101,3,120
A,bbb,20170102,6,100

$ msum i=test1.csv k=[Press TAB twice]
codeA   codeB   date    value1  value2
~ $ msum i=test1.csv k=c[Press TAB]
# The string will be completed as shown below.
~ $ msum i=test1.csv k=code[Press TAB]
codeA  codeB
~ $ msum i=test1.csv k=codeA f=value[Press TAB twice]
value1  value2
~ $ msum i=test1.csv k=codeA f=value*
codeA%0,codeB,date,value1,value2
A,bbb,20170102,9,220
#END# kgsum f=value* i=test1.csv k=codeA; IN=2 OUT=1;
```

For some commands like `mjoin`, two input files are specified. If that is the case, which file to use has been predetermined for each keyword used to specify a filename. Thus, fieldnames in the file corresponding to the keyword are used for completion.

```
$ cat test1.csv
codeA,codeB,date,value1,value2
A,aaa,20170101,3,120
A,bbb,20170102,6,100

$ cat test2.csv
code,name
A,aaa
B,bbb

$ mjoin i=test1.csv m=test2.csv k=[Press TAB twice]
codeA   codeB   date    value1  value2
$ mjoin i=test1.csv m=test2.csv k=codeA K=[Press TAB twice]
codeA  name
$ mjoin i=test1.csv m=test2.csv k=codeA K=code f=[Press TAB twice]
codeA  name
$ mjoin i=test1.csv m=test2.csv k=codeA K=code f=name
codeA%0,codeB,date,value1,value2,name
A,aaa,20170101,3,120,aaa
A,bbb,20170102,6,100,aaa
#END# kgjoin K=code f=name i=test1.csv k=codeA m=test2.csv; IN=2 OUT=2;
```

3.1.5 Completing choices

Some parameters allow you to choose one of several predetermined items, like the `c=` parameter in the `mstats` command. You can enter such parameters by completion.

```
$ msim c=[Press TAB twice]
chi       confMax convMax cosine euclid  jaccard kendall oddsRatio phi      supportr yuleQ
cityblock confMin convMin covar  hamming kappa   lift    pearson   spearman ucovar   yuleY

~ $ mstats c=[Press TAB twice]
USD     cv      kurt    mean    min     qrange  qtile3  sd      sum     ukurt   uskew   var
count   devsq   max     median  mode    qtile1  range   skew    ucount  usd     uvar
```

Chapter 4

Command Reference

The format for all commands is explained in this section. The format of all command references is shown as in the following example.

Format

```
mjoin k= [f=] [K=] [-n] [-N] m=| i= [o=] [-nfn] [-nfno] [-x] [--help] [--version]
```

Parameters

```
k=   List of field name(s) to match with the input data
     [join key break processing: character string in ascending order].
     The field(s) in the input data is specified at the K= parameter.
     Field in the reference data will be combined with same field in the record.
     If the field(s) specified at K= from reference file do not match any values,
     it is treated as a NULL value.
f=   Specify the list of field name(s) to join from the reference file.
     When the f= parameter is not defined, all fields except the key field(s)
     in the reference file will be joined to the input file.
     :   :
```

Command options and parameter can be specified after the command name. The parameter takes a value and the parameter name and value is separated by an " = " sign. An option begins with one dash " - " or two dash " – ". Command options and parameters that are common in most commands include `i=,-nfn`. The link to that section is appended in a separate section. The description of each parameter is described in the format below.

Parameters enclosed in square brackets like `[f=]` means that it is optional. On the other hand, parameters that are not enclosed in square brackets like `k=` means that it is a required parameter. Parameters separated by a vertical bar enclosed in square parentheses, such as `[to=|size=]` means that the command will only read either to= or size= (e.g. mbest command). On the other hand, if the parameters separated by `"|"` is not enclosed in square brackets such as `m=|i=`, one of these parameters is required and must be specified (e.g. mjoin command. See above).

Further, parameters or options are only required when specifying certain options, there are also more complex parameter conditions which is described in the description field of each parameter.

4.1 maccum Cumulative Calculation

Calculates the cumulative value for the column specified at `f=` parameter and save the result in a new column. Cumulative calculation is carried by the key unit when `k=` is specified.

Format

`maccum f= s= [k=]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameter

`f=`	Specify the field(s) (multiple fields can be specified) for cumulative calculation. Field with NULL values is ignored. Use :(colon) to specify new field name. Example: `f=quantity:cumulativeqty`
`s=`	After sorted by specified field (multiple fields can be specified), calculate cumulative value. `s=` parameter is required if `-q` option is not specified.
`k=`	Specify the field name as unit for cumulative calculation (multiple fields can be specified).

Examples

Example 1: Basic Example

Calculate the cumulative values of "Quantity" and "Amount" fields for each "Customer", save output as new data attributes in new columns named "AccumQuantity" and "AccumlAmount".

```
$ more dat1.csv
Customer,Quantity,Amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ maccum s=Customer f=Quantity:AccumQuantity,Amount:AccumAmount i=dat1.csv o=rsl1.csv
#END# kgaccum f=Quantity:AccumQuantity,Amount:AccumAmount i=dat1.csv o=rsl1.csv s=Customer
$ more rsl1.csv
Customer%0,Quantity,Amount,AccumQuantity,AccumAmount
A,1,10,1,10
A,2,20,3,30
B,1,15,4,45
B,3,10,7,55
B,1,20,8,75
```

Example 2: Specify Calculation by Key

Calculates the cumulative value of "Quantity" and "Amount" fields for each "Customer", and save the output in new columns named "AccumQuantity" and "AccumAmount".

```
$ more dat1.csv
Customer,Quantity,Amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ maccum k=Customer s=Customer f=Quantity:AccumQuantity,Amount:AccumAmount i=dat1.csv o=rsl2.csv
#END# kgaccum f=Quantity:AccumQuantity,Amount:AccumAmount i=dat1.csv k=Customer o=rsl2.csv s=Customer
$ more rsl2.csv
Customer,Quantity,Amount,AccumQuantity,AccumAmount
A,1,10,1,10
A,2,20,3,30
B,1,15,1,15
```

```
B,3,10,4,25
B,1,20,5,45
```

Example 3: Cumulative computation with NULL values

Calculate the cumulative values of "Quantity" and "Amount" item, and save the output as new columns named "AccumQuantity" and "AccumAmount". NULL values are ignored. Records with NULL values will be retained in the output.

```
$ more dat2.csv
Customer,Quantity,Amount
A,1,10
A,,20
B,1,15
B,3,
B,1,20
$ maccum s=Customer f=Quantity:AccumQuantity,Amount:AccumAmount i=dat2.csv o=rsl3.csv
#END# kgaccum f=Quantity:AccumQuantity,Amount:AccumAmount i=dat2.csv o=rsl3.csv s=Customer
$ more rsl3.csv
Customer%0,Quantity,Amount,AccumQuantity,AccumAmount
A,1,10,1,10
A,,20,,30
B,1,15,2,45
B,3,,5,
B,1,20,6,65
```

Related Commands

mshare : Calculate composition ratio. Cumulative relative frequency can be calculated when used with `maccum`.

mcal : Cumulative total can be calculated by using the results from the previous row with `#{}`.

4.2 marff2csv - Conversion from arff to csv Format

Convert data from arff format (data format for WEKA) to csv format.

arff format data

The arff data format is described below.

```
@RELATION       Title

@ATTRIBUTE      Field name    string(character string)
@ATTRIBUTE      Field name    date(date format:format is optional.
                                If not defined, "yyyy-MM-dd'T'HH:mm:ss")
@ATTRIBUTE      Quantity    numeric(number)
@ATTRIBUTE      Product    {A,B}(category field type)

@DATA(Actual data)
No.1,20081201,1,10,A
No.2,20081202,2,20,A
No.3,20081203,3,30,A
No.4,20081201,4,40,B
No.5,20081203,5,50,B
```

Format

`marff2csv` [i=] [o=] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Examples

Example 1: Basic Example

Convert customer purchasing data in arff format to csv format.

```
$ more dat1.arff
@RELATION       Customer Purchase Data

@ATTRIBUTE      Customer    string
@ATTRIBUTE      Date    date yyyymmdd
@ATTRIBUTE      Quantity    numeric
@ATTRIBUTE      Amount    numeric
@ATTRIBUTE      Product    {A,B}

@DATA
No.1,20081201,1,10,A
No.2,20081202,2,20,A
No.3,20081203,3,30,A
No.4,20081201,4,40,B
No.5,20081203,5,50,B
$ marff2csv i=dat1.arff  o=rsl1.csv
#END# kgarff2csv i=dat1.arff o=rsl1.csv
$ more rsl1.csv
Customer,Date,Quantity,Amount,Product
No.1,20081201,1,10,A
No.2,20081202,2,20,A
No.3,20081203,3,30,A
No.4,20081201,4,40,B
No.5,20081203,5,50,B
```

Related Command

mcsv2arff

Reference

http://weka.wikispaces.com/ARFF

4.3 mavg - Calculate Average

Calculates the average values in column specified by the `f=` parameter.

Format

`mavg f= [k=] [-n]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Specify the field(s) (multiple fields can be specified) with the values to be aggregated. Use : (colon) to specify the new field name. Example: `f=`Quantity:AverageQuantity
`k=`	Specify the set of field name(s) (Multiple fields can be specified) as unit of aggregation.
`-n`	Output as NULL if the data consist at least one NULL value.

Examples

Example 1: Basic Example

Calculate the average values of "Quantity" and "Amount" fields for each "Customer", save the computed output in new columns named "AverageVolume" and "AverageAmount".

```
$ more dat1.csv
Customer,Quantity,Amount
A,1,5
A,2,20
B,1,15
B,,10
B,5,20
$ mavg k=Customer f=Quantity:AvgQuantity,Amount:AvgAmount i=dat1.csv o=rsl1.csv
#END# kgavg f=Quantity:AvgQuantity,Amount:AvgAmount i=dat1.csv k=Customer o=rsl1.csv
$ more rsl1.csv
Customer%0,AvgQuantity,AvgAmount
A,1.5,12.5
B,3,15
```

Example 2: Output consisting of NULL values

Calculate the average values of "Quantity" and "Amount" fields for each "Customer", save output in a new columns named "AverageVolume" and "AverageAmount". When specifying the `-n` option, if a NULL value is included in the input, the result will return NULL value.

```
$ mavg k=Customer f=Quantity:AvgQuantity,Amount:AvgAmount -n i=dat1.csv o=rsl2.csv
#END# kgavg -n f=Quantity:AvgQuantity,Amount:AvgAmount i=dat1.csv k=Customer o=rsl2.csv
$ more rsl2.csv
Customer%0,AvgQuantity,AvgAmount
A,1.5,12.5
B,,15
```

Example 3: Calculate sum without key field

Calculate the average values of "Quantity" and "Amount" fields, and save the outputs in columns "AvgQuantity" and "AvgAmount".

```
$ mavg f=Quantity:AvgQuantity,Amount:AvgAmount i=dat1.csv o=rsl3.csv
#END# kgavg f=Quantity:AvgQuantity,Amount:AvgAmount i=dat1.csv o=rsl3.csv
$ more rsl3.csv
Customer,AvgQuantity,AvgAmount
B,2.25,14
```

Related Commands

mhashavg : Aggregate calculation does not require prior sorting on the key field.

msum : Command to calculate sum.

mstats : Calculate a variety of statistics.

4.4 mbest - Select Rows

Select records based on the specified row numbers. Note that row number starts at 0 (the first row of data starts at row 0 excluding the row of field names). Define the row numbers at from= and to= parameters (and size= parameter in some instances).

Format

`mbest s= [R=] [from=] [to=|size=] [k=] [u=] [-r]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`s=`	After sorted by specified field(s) (multiple fields can be specified), the rows are selected. `s=` parameter is required if `-q` option is not specified.
`from=`	Define the start of row number (integers greater than 0) [default value: 0]
`to=`	Define the end of row number (integers greater than 0) [default value: 0] Where [value of `from=`] ≤ [value of `to=`] .
`size=`	Number of rows to select [default value: 1] `to=` and `size=` cannot be specified at the same time.
`R=`	Range list (accept multiple fields) 【required parameter】 * Method of specifying the range that was used in the previous versio Set the range using line numbers. Use _(underscore) to specify the range in the format of MIN (start of row)_MAX (end of row). ※ Note: Sort the data in advance based on user's preference.
`k=`	Key field (accept multiple key fields) [aggregate key break processing] Records with same key values will be selected based on the defined rows at `from=`,`to=`,`size=`. `-x`,`-nfn` options can be used to specify the field number (0〜).
`u=`	Output file of unmatched records. Unmatched records the do not match the criteria is written to the defined file.
`-r`	Reverse selection Select the rows other than the ones defined at the parameter `from=,to=(size=)`.

Examples

Example 1: Basic Example

This example assumed that the "quantity" and "amount" fields are sorted from the largest value (descending order). Records are selected from the first row (line 0) by default if `from=,to=,size=` parameters are not specified.

```
$ more dat1.csv
Customer,Quantity,Amount
A,20,5200
B,18,4000
C,15,3500
D,10,2000
E,3,800
$ mbest s=Quantity%nr,Amount%nr i=dat1.csv o=rsl1.csv
#END# kgbest i=dat1.csv o=rsl1.csv s=Quantity%nr,Amount%nr
$ more rsl1.csv
Customer,Quantity%0nr,Amount%1nr
A,20,5200
```

Example 2: Basic Example 2

After sorting by "customers", select 3 rows from the first row (line 0).

```
$ mbest s=Customer from=0 size=3 i=dat1.csv o=rsl2.csv
#END# kgbest from=0 i=dat1.csv o=rsl2.csv s=Customer size=3
$ more rsl2.csv
Customer%0,Quantity,Amount
A,20,5200
B,18,4000
C,15,3500
```

Example 3: Basic Example 3

Without sorting (in the original order), select from line 0 to line 1.

```
$ mbest -q from=0 to=1 i=dat1.csv o=rsl3.csv
#END# kgbest -q from=0 i=dat1.csv o=rsl3.csv to=1
$ more rsl3.csv
Customer,Quantity,Amount
A,20,5200
B,18,4000
```

Example 4: Reverse Selection

Select records other than customers' first visit to store. Save the records of customers' first visit to the file `fvd.csv`.

```
$ more dat2.csv
Customer,Date,Amount
A,20081201,10
A,20081207,20
A,20081213,30
B,20081002,40
B,20081209,50
$ mbest s=Customer,Date k=Customer -r u=fvd.csv i=dat2.csv o=rsl4.csv
#END# kgbest -r i=dat2.csv k=Customer o=rsl4.csv s=Customer,Date u=fvd.csv
$ more rsl4.csv
Customer,Date,Amount
A,20081207,20
A,20081213,30
B,20081209,50
$ more fvd.csv
Customer,Date,Amount
A,20081201,10
B,20081002,40
```

Related commands

msel : The `line()` function can be specified in the condition parameter to carry out similar processing functions.

muniq : Returns unique values in key field.

mselnum :Select rows within a numeric range.

4.5 mbucket - Partition Data into Uniform Buckets

Partition numerical data field(s) specified at `f=` into a number of segments specified by `n=`. There are two ways to compute the bucket intervals. The first method is to compute an uniform spread of data points for each partition (referred to as partition of uniform buckets). The second method is to compute uniform interval ranges across partitions (referred to as partition of uniform ranges). Data is partitioned into equal interval ranges when the `-rng` option is specified. Otherwise, data will be partitioned uniformly across buckets if this option is not specified. When multiple fields are defined at `f=`, the data buckets are generated separately for each field.

Format

`mbucket f= n= [-rng] [-r] [F=] [k=] [O=]` [i=] [o=] [bufcount=] [-assert_diffSize] [-assert_nullkey] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Partitioning is based on the value specified in this field (multiple fields can be specified). Target partition field name:new field name
`n=`	Number of buckets Specify the number of buckets to be partitioned for the field(s) defined in `f=` parameter(s). If 1 is defined, the command will partition by the number of items specified in `f=` parameter.
`F=`	Output format [default value: 0] Output format of bucket label. 0:Display bucket numbers 1:Display value range of buckets 2:Display both bucket numbers and value range of buckets.
`k=`	Key field(s) to retrieve rows of data incrementally for bucket partitions (multiple keys can be specified).
`O=`	Output file with values range of bucket Specify output file name with values range of bucket on the field name(s) defined in parameter `f=`.
`-rng`	Define equal value of bucket range Divide buckets by the specified value range.
`-r`	Print bucket numbers in reverse order.

Examples

Example 1: Basic Example

Partition x and y into two subsets of equal extent and save the output file as rng.csv

```
$ more dat1.csv
id,x,y
A,2,7
B,6,7
C,5,6
D,7,5
E,6,4
F,1,3
G,3,3
H,4,2
I,7,2
J,2,1
$ mbucket f=x:xb,y:yb n=2 O=rng1.csv i=dat1.csv o=rsl1.csv
#END# kgbucket O=rng1.csv f=x:xb,y:yb i=dat1.csv n=2 o=rsl1.csv
$ more rsl1.csv
id,x,y,xb,yb
A,2,7,1,2
B,6,7,2,2
C,5,6,2,2
D,7,5,2,2
E,6,4,2,2
F,1,3,1,1
```

```
G,3,3,1,1
H,4,2,1,1
I,7,2,2,1
J,2,1,1,1
$ more rng1.csv
fieldName,bucketNo,rangeFrom,rangeTo
x,1,,4.5
x,2,4.5,
y,1,,3.5
y,2,3.5,
```

Example 2: Partition by equal range

Use `-rng` option to partition the data by uniform value ranges.

```
$ mbucket f=x:xb,y:yb n=2 -rng O=rng2.csv i=dat1.csv o=rsl2.csv
#END# kgbucket -rng O=rng2.csv f=x:xb,y:yb i=dat1.csv n=2 o=rsl2.csv
$ more rsl2.csv
id,x,y,xb,yb
A,2,7,1,2
B,6,7,2,2
C,5,6,2,2
D,7,5,2,2
E,6,4,2,2
F,1,3,1,1
G,3,3,1,1
H,4,2,2,1
I,7,2,2,1
J,2,1,1,1
$ more rng2.csv
fieldName,bucketNo,rangeFrom,rangeTo
x,1,,4
x,2,4,
y,1,,4
y,2,4,
```

Example 3: Example using key field

Partition x and y into two subsets of equal extent using ”id” as the key parameter. By specifying `n=2,3`, field `x` is divided into 2 buckets, and field `y` is divided into 3 buckets. Include bucket numbers and value range of buckets in the output file (`F=2`).

```
$ more dat2.csv
id,x,y
A,2,7
A,6,7
A,5,6
B,7,5
B,6,4
B,1,3
C,3,3
C,4,2
C,7,2
C,2,1
$ mbucket k=id f=x:xb,y:yb n=2,3 F=2 i=dat2.csv o=rsl3.csv
#END# kgbucket F=2 f=x:xb,y:yb i=dat2.csv k=id n=2,3 o=rsl3.csv
$ more rsl3.csv
id%0,x,y,xb,yb
A,2,7,1:_3.5,2:6.5_
A,6,7,2:3.5_,2:6.5_
A,5,6,2:3.5_,1:_6.5
B,7,5,2:3.5_,3:4.5_
B,6,4,2:3.5_,2:3.5_4.5
B,1,3,1:_3.5,1:_3.5
C,3,3,1:_3.5,3:2.5_
C,4,2,2:3.5_,2:1.5_2.5
C,7,2,2:3.5_,2:1.5_2.5
C,2,1,1:_3.5,1:_1.5
```

Theorem of Arithmetic

Assume D is a data set of n elements $(x_1, x_2, \cdots, x_n)$. Partition D uniformly into k number of groups (referred to as buckets) and ensure that the data is partitioned uniformly in each bucket. Dispersion is used as the fundamental evaluation criteria for uniformity.

$$X = \{x_i \mid 1 \ \leq \ i \ \leq \ n \ , \ x_i \ \in \ D\}$$

Arrange each value within X in ascending order $v_1, v_2, \ldots, v_n$. Partition of the buckets X is represented by partition of intervals $\{v_1, v_2, \ldots, v_n\}$. This interval is defined as $I_1, I_2, \ldots, I_k$. Based on these elements,$D_j?n_j$, the following is defined.

$$D_j = \{x_i \mid 1 \ \leq \ i \ \leq \ n \ , \ x_i \ \in \ I_j\}$$

$$n_j = |D_j|$$

The basis of uniform dispersion is described as follows.

$$Var = \sum_{j=1}^{k} \ (n_j - \bar{n})$$

If $\bar{n} = n/k$, the equation becomes

$$Var \ = \ \sum_{j=1}^{k} \ (n_j^2 - 2n_j\bar{n} + \bar{n}^2) \quad = \ \sum_{j=1}^{k} \ n_j^2 \ - \ k\bar{n}^2$$

$k\bar{n}^2$ remains a constant regardless of how the data is partitioned. Therefore, the first term from the above equation:

$$Var' = \ \sum_{j=1}^{k} \ n_j^2$$

minimizes Var and splits the data into intervals.

Algorithm

Recursive equation for dynamic programming is used for optimization as illustrated in the following. The minimum value of $\sum_{j=1}^{h} n_j^2$ is divided into h intervals of $I_1, I_2, \ldots, I_h$ of $v_1, v_2, \ldots, v_m$, to determine $DP(n, k)$. $DP(m, h)$ is substituted in the following function.

$$DP(m, \ h) \ = \ \min_{g=h-1, \ \ldots, m-1} \ \{ \ DP(g, h-1) \ + \ |\{ \ x_i \mid v_{g+1} \ \leq \ x_i \ \leq \ v_m \ \}|^2\}$$

The above function recursively defines a sequence. Given the initial term below:

$$DP(m, \ 1) \ = \ |\{ \ x_i \mid v_1 \ \leq \ x_i \ \leq \ v_m \ \}|^2 \ , \ m = 1, \ldots, n$$

the next term of the sequence is defined as a function of the preceding term and iterated as follows: $DP(m, 2)(m = 1, \ldots, n), DP(m, 3)(m = 1, \ldots, n), \ldots, DP(m, k-1)(m = 1, \ldots, n)$.

The function ends until it searches for the term $DP(n, k)$

$$DP(m, \ k) \ = \ \min_{g=k-1, \ \ldots, n-1} \ \{ \ DP(g, k-1) \ + \ |\{ \ x_i \mid v_{g+1} \ \leq \ x_i \ \leq \ v_n \ \}|^2\}$$

Related command

mmbucket : Partition multidimensional data into uniform buckets

4.6 mcat - ConCATenate

Merge all records in the files specified at `i=` parameter according to the order of files. If a wild card is used to specify file names, the files will be merged in alphabetical order of the file name.

Format

`mcat [f=] [-skip_fnf] [-nostop|-skip|-force]` [i=] [o=] [-add_fname] [-stdin] [-assert_diffSize] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`i=`	Specify list of input file names. Read multiple CSV files separated by comma delimiter. Wild card characters can be used in file name.
`f=`	Specify the field name(s) to concatenate. If `f=` is not specified, the field names defaults to the first file defined in the `i=` parameter.
`-skip_fnf`	If a specified file in the `i=` parameter does not exist, the program will bypass the error. However, the program returns an error if all files cannot be found.
`-nostop`	`-nostop,-skip,-force` are parameters for controlling exceptions when header is not present. `-nostop` flag returns null if field name is not specified. When `-nfn` flag is used with `stop` flag, the program terminates if the number of items in the data is different than the parameter defined.
`-skip`	Files are not concatenated if field name(s) is not specified. When `-nfn` flag is used with `-skip` flag, files are not concatenated if the number of data items are different.
`-force`	Force concatenation of files using location of fields when header is not present. Print output to null if item number is not available.
`-stdin`	Merge from standard input.
`-add_fname`	Add file name in the last column. Standard input will be named as `/dev/stdin`. The field name for this option is fixed as `"fileName"`, error will be returned if input data contains the same field name.

Note

- Wild card characters ("?" and "*") can be used to specify multiple directory and file names.
- The symbol `~/`can be used to indicate home directory.
- The files are concatenated according according to the order specified in the `i=` parameter. If a wild card is used, files will be merged in alphabetical order. Standard input takes precedence when merging files.

Examples

Example 1: Concatenate files with the same header

```
$ more dat1.csv
customer,date,amount
A,20081201,10
B,20081002,40
$ more dat2.csv
customer,date,amount
A,20081207,20
A,20081213,30
B,20081209,50
$ mcat i=dat1.csv,dat2.csv o=rsl1.csv
#END# kgcat i=dat1.csv,dat2.csv o=rsl1.csv
$ more rsl1.csv
customer,date,amount
A,20081201,10
B,20081002,40
```

```
A,20081207,20
A,20081213,30
B,20081209,50
```

Example 2: Concatenate files with different header

The first file `dat1.csv` defined at `i=` contains columns ”customer,date,amount”. However, since ”amount” is not present in `dat3.csv`, it will return an error. Nevertheless, the contents in the first file `dat1.csv` is merged and saved in the output.

```
$ more dat3.csv
customer,date,quantity
A,20081201,3
B,20081002,1
$ mcat i=dat1.csv,dat3.csv o=rsl2.csv
#ERROR# field name [amount] not found on file [dat3.csv] (kgcat)
$ more rsl2.csv
customer,date,amount
A,20081201,10
B,20081002,40
```

Example 3: Concatenate files with different header2

When previous example is attached with `-nostop` option, the command will continue processing and return NULL value for the data item not found. Other options such as `skip,force` handle conditions when the field name is not found. For details, refer to the description of parameters.

```
$ more dat3.csv
customer,date,quantity
A,20081201,3
B,20081002,1
$ mcat -nostop i=dat1.csv,dat3.csv o=rsl3.csv
#END# kgcat -nostop i=dat1.csv,dat3.csv o=rsl3.csv
$ more rsl3.csv
customer,date,amount
A,20081201,10
B,20081002,40
A,20081201,
B,20081002,
```

Example 4: Concatenate specific field names from input files

Merge field names specified at `f=`.

```
$ mcat f=customer,date i=dat2.csv,dat3.csv o=rsl4.csv
#END# kgcat f=customer,date i=dat2.csv,dat3.csv o=rsl4.csv
$ more rsl4.csv
customer,date
A,20081207
A,20081213
B,20081209
A,20081201
B,20081002
```

Example 5: Merge from standard input

Read file `dat2.csv` from standard input by specifying `-stdin` option.

```
$ mcat -stdin i=dat1.csv o=rsl5.csv <dat2.csv
#END# kgcat -stdin i=dat1.csv o=rsl5.csv
$ more rsl5.csv
customer,date,amount
A,20081207,20
A,20081213,30
B,20081209,50
```

```
A,20081201,10
B,20081002,40
```

Example 6: Add file name as new column

When `-add_fname` is specified, the original file name `fileName` is added as a new column. File name of standard input is `/dev/stdin`.

```
$ mcat -add_fname -stdin i=dat1.csv o=rsl6.csv <dat2.csv
#END# kgcat -add_fname -stdin i=dat1.csv o=rsl6.csv
$ more rsl6.csv
customer,date,amount,fileName
A,20081207,20,/dev/stdin
A,20081213,30,/dev/stdin
B,20081209,50,/dev/stdin
A,20081201,10,dat1.csv
B,20081002,40,dat1.csv
```

Example 7: Specify wild card

Specifying wild card `dat*.csv` to concatenate the three CSV files `dat1.csv,dat2.csv,dat3.csv` in the current directory.

```
$ more dat1.csv
customer,date,amount
A,20081201,10
B,20081002,40
$ more dat2.csv
customer,date,amount
A,20081207,20
A,20081213,30
B,20081209,50
$ more dat3.csv
customer,date,quantity
A,20081201,3
B,20081002,1
$ mcat -force i=dat*.csv o=rsl7.csv
#END# kgcat -force i=dat*.csv o=rsl7.csv
$ more rsl7.csv
customer,date,amount
A,20081201,10
B,20081002,40
A,20081207,20
A,20081213,30
B,20081209,50
A,20081201,3
B,20081002,1
```

Example 8: Concatenate the same file multiple times

Same file can be specified more than one time.

```
$ mcat i=dat1.csv,dat1.csv,dat1.csv o=rsl8.csv
#END# kgcat i=dat1.csv,dat1.csv,dat1.csv o=rsl8.csv
$ more rsl8.csv
customer,date,amount
A,20081201,10
B,20081002,40
A,20081201,10
B,20081002,40
A,20081201,10
B,20081002,40
```

Related command

msep : Reverse the operation mentioned above and separate data files.

4.7 mchgnum - Substitute Values within Numerical Range

The field name for encoding is specified at `f=` parameter, number and range criteria is specified at the `R=` parameter, the substitution string specified in the `v=` parameter replaces the value in the defined field.

Format

`mchgnum f= R= [O=|-F] [v=] [-A] [-r]` [i=] [o=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Replace the specified field (multiple fields can be specified) according to the replacement string list at `v=` parameter and the numerical ranges list at `R=` parameter.
`R=`	Specify the numerical range to be replaced (multiple fields can be specified) (`1.1,2.5` : more than 1.1 and less than 2.5). Use `MIN` for minimum value, `MAX` for maximum value (MIN, 2.5 : 2.5 or less).
`O=`	Out of range character strings Specify the replacement string when values fall outside the numeric range list at the `R=` parameter (returns NULL values in output when this parameter is not specified).
`-F`	Display out of range values in the column. Retain the out of range values in the output even though the values fall outside the specified numeric range defined in the `R=` parameter.
`v=`	Specify the replacement character string corresponding to the numerical range in the `R=` parameter. More than 1 argument must be defined at `R=`.
`-A`	This option adds output to a new column instead of replacing the specified item.
`-r`	The range defined at `R=` parameter deals with ' greater than less than'. For example, `1.1_2.5` represents "greater than 1.1 ¡ less than 2.5".

Examples

Example 1: Basic Example

Encodes the numeric values in `quantity` column to character strings where values of less than but not equal to 10 are treated as `low`, 10 or more but less than 20 are treated as `middle`, values of 20 or more is treated as `high`.

```
$ more dat1.csv
customer,quantity
A,5
B,10
C,15
D,2
E,50
$ mchgnum f=quantity R=MIN,10,20,MAX v=low,middle,high i=dat1.csv o=rsl1.csv
#END# kgchgnum R=MIN,10,20,MAX f=quantity i=dat1.csv o=rsl1.csv v=low,middle,high
$ more rsl1.csv
customer,quantity
A,low
B,middle
C,middle
D,low
E,high
```

Example 2: Equal to paramter range

Replace the numeric values in `quantity` column to character strings where 10 or below is treated as `low`, more than 10 but less than or equal to 20 is treated as `middle`, more than 20 is treated as `high`.

```
$ mchgnum f=quantity R=MIN,10,20,MAX v=low,middle,high -r i=dat1.csv o=rsl2.csv
#END# kgchgnum -r R=MIN,10,20,MAX f=quantity i=dat1.csv o=rsl2.csv v=low,middle,high
$ more rsl2.csv
customer,quantity
A,low
B,low
C,middle
D,low
E,high
```

Example 3: Replace values out of the list of numeric range

Replace numeric values in `quantity` column to character strings where 10 or above and less than 20 is coded as `low`, 20 or above and less than 30 is coded as `middle`, 30 or more is coded as `high`, values that are less than 10 is coded as `OutOfRange`.

```
$ mchgnum f=quantity R=10,20,30,MAX v=low,middle,high O="OutOfRange" i=dat1.csv o=rsl3.csv
#END# kgchgnum O=OutOfRange R=10,20,30,MAX f=quantity i=dat1.csv o=rsl3.csv v=low,middle,high
$ more rsl3.csv
customer,quantity
A,OutOfRange
B,low
C,low
D,OutOfRange
E,high
```

Example 4: Add a new column

Replace the numeric values in `quantity` column to character strings where values less than 10 is treated as `low`, 10 or above but less than 20 is treated as `middle`, 20 or above is treated as `high`. Store the output of replacement strings in a new column as `evaluate`.

```
$ mchgnum f=quantity:evaluate R=MIN,10,20,MAX v=low,middle,high -A i=dat1.csv o=rsl4.csv
#END# kgchgnum -A R=MIN,10,20,MAX f=quantity:evaluate i=dat1.csv o=rsl4.csv v=low,middle,high
$ more rsl4.csv
customer,quantity,evaluate
A,5,low
B,10,middle
C,15,middle
D,2,low
E,50,high
```

Example 5: Display original values in column if out of defined range

Replace the numeric values in `quantity` column to character strings where values of 10 or above but less than 20 is coded as `low`, 20 or above but less than 30 is coded as `middle`, 30 or above is coded as `high`. Retain original values in the output if the value is less than 10.

```
$ mchgnum f=quantity R=10,20,30,MAX v=low,middle,high -F i=dat1.csv o=rsl5.csv
#END# kgchgnum -F R=10,20,30,MAX f=quantity i=dat1.csv o=rsl5.csv v=low,middle,high
$ more rsl5.csv
customer,quantity
A,5
B,low
C,low
D,2
E,high
```

Related Commands

mchgstr : Substitute string.

msed : Replace text with regular expressions.

4.8 mchgstr - Substitute String

Replace the field(s) specified in the `f=` parameter with the string according to the replacement criteria specified in the `c=` parameter.

Format

`mchgstr c= f= [O=] [-A] [-F] [-sub] [-W]` [i=] [o=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`c=`	Specify the replacement character string to be replaced.
`f=`	Specify the field(s) (multiple fields can be specified) where the character string is replaced according to the replacement criteria specified in the verb—c=— parameter.
`O=`	Specify the replacement string if the values does not match any substitution criteria specified in the `c=` parameter (returns NULL values when this parameter is not specified).
`-A`	This option adds output in a new column instead of replacing the specified item.
`-F`	Retain values in output even though values fall outside the specified numeric range defined in the R= parameter.
`-sub`	Compare using substring match rather than exact match. Search the values specified in the `f=` parameter, and replace the string with the criteria specified in `c=`.
`-W`	Match wide-character substring with `-sub` option.

Examples

Example 1: Basic Example

Replace values in the column from `"01"` to `"A"`, `"03"` to `"B"`, `"04"` to `"C"`. Other values that do not match the criteria are returned as NULL values in the output.

```
$ more dat1.csv
id,item
1,01
2,02
3,03
4,04
5,05
$ mchgstr f=item c=01:A,03:B,05:C i=dat1.csv o=rsl1.csv
#END# kgchgstr c=01:A,03:B,05:C f=item i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,item
1,A
2,
3,B
4,
5,C
```

Example 2: Replace values outside the criteria

Use the `O=` parameter to replace character string that do not match the substitution criteria to `"out of range"` in the output.

```
$ mchgstr f=item c=01:A,03:B,05:C O="out of range" i=dat1.csv o=rsl2.csv
#END# kgchgstr O=out of range c=01:A,03:B,05:C f=item i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,item
1,A
2,out of range
3,B
```

```
4,out of range
5,C
```

Example 3: Example 3: Add new column in output

Define the name of new column (`item info`) in output with `-A` option.

```
$ mchgstr f=item:"item info" c=01:A,03:B,05:C O="out of range" -A i=dat1.csv o=rsl3.csv
#END# kgchgstr -A O=out of range c=01:A,03:B,05:C f=item:item info i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,item,item info
1,01,A
2,02,out of range
3,03,B
4,04,out of range
5,05,C
```

Example 4: Display original value in field falls outside the criteria

When `-F` option is specified, output value of the field remains the same if it does not match any of the substitution criteria.

```
$ mchgstr f=item c=01:A,03:B,05:C -F i=dat1.csv o=rsl4.csv
#END# kgchgstr -F c=01:A,03:B,05:C f=item i=dat1.csv o=rsl4.csv
$ more rsl4.csv
id,item
1,A
2,02
3,B
4,04
5,C
```

Example 5: Replace matching substrings

Replace substring with `-sub` option specified. In following example, values where `item` field contains `"01"` will be substituted with `"A"`.

```
$ more dat2.csv
id,item
1,0111
2,0121
3,0231
4,0241
5,0151
$ mchgstr f=item c=01:A -sub i=dat2.csv o=rsl5.csv
#END# kgchgstr -sub c=01:A f=item i=dat2.csv o=rsl5.csv
$ more rsl5.csv
id,item
1,A11
2,A21
3,
4,
5,A51
```

Example 6: Wide character substring match

Use the option `-W` to replace wide-characters strings. However, if you are using UTF-8 encoding, it is not necessary to define `-W`. Refer to the section ”Multibyte characters” for details.

```
$ more dat3.csv
id,city
1,奈良市
2,下市町
3,十津川村
4,五條市
5,山添村
```

```
$ mchgstr f=city c=市:01,町:02,村:02 -sub -W i=dat3.csv o=rsl6.csv
#END# kgchgstr -W -sub c=市:01,町:02,村:02 f=city i=dat3.csv o=rsl6.csv
$ more rsl6.csv
id,city
1,奈良01
2,下0102
3,十津川02
4,五條01
5,山添02
```

Related Commands

mchgnum : Substitution based on numeric range.

msed : Replace string by regular expression.

4.9 mchkcsv - Check CSV Data

Automatically repair (standardize the number of fields for all rows) comma-separated-values (CSV) data that do not meet the f[sect:csv]CSV format for MCMD. Use the `-diag` option to perform checking on CSV data.

Format

`mchkcsv [a=] [-diag] [-r]` [i=] [o=] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`i=`	Input file name The CSV data defined here is checked for incomplete lines. Issues identified in this file is automatically repaired. If this parameter is omitted, the standard input is used.
`a=`	Substitute the field name(s) in the input data with the field name(s) specified at this parameter. If the number of fields specified is less than the number of fields in the header, the output will only yield portion of columns from the input data starting from the left. Conversely, if the number of fields specified here is greater than the number of fields in input data, data items in the extra fields are expressed as NULL values in the output.
`-diag`	Execute check. When this option is specified, write result to standard output.
`-r`	Exclude control characters This defines the control character as ASCII character code 0x00-0x1f,0x7f (remove 0x09,0x0a,0x0d). Control code is automatically converted to the string &#x when this option is not specified.

Examples

Example 1: Repair data

This data contains different number of columns in all records. For instance, only 3 records have 4 columns. Use M-Command to repair and standardize 3rd and 5th rows to 4 columns.

```
$ more dat1.csv
product,date,quantity,amount
A,20081201,1,10
A,20081202,2,
A,*,3
B,20081201,4,40
B,20081203,50
$ mchkcsv i=dat1.csv o=rsl1.csv
#END# kgchkcsv i=dat1.csv o=rsl1.csv
$ more rsl1.csv
product,date,quantity,amount
A,20081201,1,10
A,20081202,2,
A,*,3,
B,20081201,4,40
B,20081203,50,
```

Example 2: Change field name after repairing the data

This data contains different number of columns in all records. For instance, only 3 records have 4 columns. Use M Command to repair and standardize 3rd and 5th rows to 4 columns. At the same time, label the field names from the input data as ¥verb—“ product,date,quantity,amount ”— starting from the left.

```
$ more dat2.csv
fld1,fld2,fld3,fld4
A,20081201,1,10
A,20081202,2,
A,*,3
B,20081201,4,40
B,20081203,50
$ mchkcsv a=product,date,quantity,amount i=dat2.csv o=rsl2.csv
#END# kgchkcsv a=product,date,quantity,amount i=dat2.csv o=rsl2.csv
$ more rsl2.csv
product,date,quantity,amount
A,20081201,1,10
A,20081202,2,
A,*,3,
B,20081201,4,40
B,20081203,50,
```

Example 3: Check data integrity and output diagnostic results

It merely checks for incomplete data structure in the CSV data, and save the diagnosis result in CSV file.

```
$ mchkcsv -diag i=dat1.csv o=rsl3.csv
#END# kgchkcsv -diag i=dat1.csv o=rsl3.csv
$ more rsl3.csv
#==================================================
# diagnosis for the CSV file
# file name : dat1.csv
#--------------------------------------------------
# meaning of the first charactor on each line
# # : commnet or resutl with no error
#   : that is, mcmd can handle the CSV file if all lines begin with '#'
# ? : error that mcmd cannot handle
#   : refer the 'explanation' section for the meaning of the alphabet next of '?'
#==================================================
############################ CSV header for field names(first line) ###
# the number of fields : 4
# fieldNo.  name
# 1   product
# 2   date
# 3   quantity
# 4   amount
#
############## about EOL(End Of Line) (including a CSV header) ##
# the number of lines with LF : 6 (LineNo: 0 1 2 ... )
#
################# about data lines (no including a CSV header) ###
# the number of lines : 5
# data volume in byte : 66
# the average number of lines : 13.2
# the maximum number of lines : 16 (LineNo:2)
# the maximum number of lines : 6 (LineNo:4)
# note: EOL charactor is counted in the numbers
#
################################# aoubt the number of fields ###
?g lines with different number of fields are detected
?g  # of fields:3 (LineNo:4)
?g  # of fields:3 (LineNo:6)
#
####################################### about field ###
#  fieldNo[1] name[product]
#    # of lines in NULL : 0
#    # of lines not enclosed with double quotation : 5 (LineNo: 1 2 3 ... )
#    # of lines enclosed with double quotation : 0
#
#  fieldNo[2] name[date]
#    # of lines in NULL : 0
#    # of lines not enclosed with double quotation : 5 (LineNo: 1 2 3 ... )
#    # of lines enclosed with double quotation : 0
#
#  fieldNo[3] name[quantity]
#    # of lines in NULL : 0
#    # of lines not enclosed with double quotation : 5 (LineNo: 1 2 3 ... )
#    # of lines enclosed with double quotation : 0
#
#  fieldNo[4] name[amount]
#    # of lines in NULL : 1 (LineNo: 2 )
#    # of lines not enclosed with double quotation : 3 (LineNo: 1 2 4 )
#    # of lines enclosed with double quotation : 0
#
################################## explanation ###
# ?a : It cannot assign the field if there are duplicat name of fields.
#  [how to] Specify a complete set of field names like 'mchkcsv a=x,y,z'
# ?b : MCMD has a valid set of charactors for a field name.
#  [how to]Specify a complete set of field names like 'mchkcsv a=x,y,z'
# ?c : MCMD can handle only lines with LF as a EOL.
#      This is MCMD original restriction, conforming to RFC4180.
#  [how to] Run mchkcsv command that convert CR,CRLF to LF.
#
# ?d : The last line does not have a EOL charactor.
#      It does not conform to RFC4180.
#  [how to] Run mchkcsv command add LF at the end.
#
# ?e : Data file has '\0' charactor.
#      The data may not be a text.
#      It does not conform to RFC4180.
#  [how to] Run mchkcsv that convert the charactor to "&#x00
#           Run mchkcsv command with '-r' option that delete '\0' characotr.
#
# ?f : The number of charactors per line exceed the limit MCMD can handle.
#      MCMD can handle a line less than or equal to 1024000 bytes of charactors.
#  [how to] Change a enviroment variable KG_MaxRecLen.
#        ex) export KG_MaxRecLen=2048000
#      You cannot specify the number greater than 10240000 bytes.
#      This is MCMD original restriction, conforming to RFC4180.
#
# ?g : MCMD can handle only a CSV data that all lines have same number of fields.
#      This is MCMD original restriction, conforming to RFC4180.
#  [how to]
#       1) Use mchkcsv command that aligns all lines with same number of the field name header.
#            Exceeded field value will be cut off, and it will add a NULL value for missing field.
#       2) Use mchkcsv command with a= parameter.
#            It uses the field names on a= parameter just as a header line (the header line will be skipp
#
# ?h : Field value include a control charactors (0x01~0x1F,0x7F).
#      The data may not be a text.
#      It does not conform to RFC4180.
#  [how to] Run mchkcsv that convert the charactor to text like "&#x01
#           Run mchkcsv command with '-r' option that delete the control characotrs.
#
# ?i : TAB cannot be used.
#      It does not conform to RFC4180.
#  [how to] Run mchkcsv that convert the TAB to "&#x09
#           Run mchkcsv command with '-r' option that delete the TAB.
#
# ?j : Double quotation charactor is found in a value not enclosed by double quotation.
#        ex) NG: xxx,oo"oo,xxx  -> OK: xxx,"oo""oo",xxx
#      It does not conform to RFC4180.
#  [how to] Run mchkcsv that makes convertion in the above example.
#
# ?k : Double quotation charactor is found in a value enclosed by double quotation.
#      ex) NG: xxx,"oo"oo",xxx  -> OK: xxx,"oo""oo",xxx
#      It does not conform to RFC4180.
#  [how to] Run mchkcsv that makes convertion in the above example.
# ?l : It has BOM (Bite Order Mark) at the beginning of data.
#  [how to] Run mchkcsv command that remove the BOM.
#------------------------------------------------------------
```

Related Command

4.10 mcombi - Compute Combination

`f=` parameter specifys the set of fields, `n=` parameter defines the combination, and `a=`parameter specifies the output field name. Permutation output is possible by specifying `-p` option.

Format

`mcombi a= f= n= [s=] [k=] [-p] [-dup]` [i=] [o=] [-assert_diffSize] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`a=`	Name of the field to be added.
`f=`	Compute the combinations for the set of field name(s) (multiple fields can be specified) specified . Enumerate all combinations of the array of values in the fields specified.
`n=`	Number of combinations. When you increase the number of combinations, note that the number of output records will increase exponentially.
`s=`	After sorted by specified field (multiple fields can be specified), combinations of items specified in field `f=` are enumerated.
`k=`	List of key field name(s) (multiple fields can be specified) Compute combinations based on the list of key field name(s).
`-p`	Compute the permutations.
`-dup`	Output combinations with the same value.

Examples

Example 1: Basic Example

Enumerate all combinations of two items in the `item` field for each `customer`, and save the output in `item1,item2`. Fields not specified at `k=,f=` (`item` field in this case) remains after the key field column.

```
$ more dat1.csv
customer,item
A,a1
A,a2
A,a3
B,a4
B,a5
$ mcombi k=customer f=item n=2 a=item1,item2 i=dat1.csv o=rsl1.csv
#END# kgcombi a=item1,item2 f=item i=dat1.csv k=customer n=2 o=rsl1.csv
$ more rsl1.csv
customer%0,item,item1,item2
A,a3,a1,a2
A,a3,a1,a3
A,a3,a2,a3
B,a5,a4,a5
```

Example 2: Basic Example 2

When you specify the `-dup` option, the output includes combination of the same field.

```
$ mcombi k=customer f=item n=2 a=item1,item2 i=dat1.csv o=rsl2.csv -dup
#END# kgcombi -dup a=item1,item2 f=item i=dat1.csv k=customer n=2 o=rsl2.csv
$ more rsl2.csv
customer%0,item,item1,item2
A,a3,a1,a1
A,a3,a1,a2
A,a3,a1,a3
A,a3,a2,a2
A,a3,a2,a3
A,a3,a3,a3
B,a5,a4,a4
```

```
B,a5,a4,a5
B,a5,a5,a5
```

Example 3: Compute permutation

Enumerate permutation of two items in the `item` field for each `customer`, and save the output in column `item1,item2`.

```
$ mcombi k=customer f=item n=2 a=item1,item2 -p i=dat1.csv o=rsl3.csv
#END# kgcombi -p a=item1,item2 f=item i=dat1.csv k=customer n=2 o=rsl3.csv
$ more rsl3.csv
customer%0,item,item1,item2
A,a3,a1,a2
A,a3,a2,a1
A,a3,a1,a3
A,a3,a3,a1
A,a3,a2,a3
A,a3,a3,a2
B,a5,a4,a5
B,a5,a5,a4
```

Related Command

4.11 mcommon - Select Common Records in Reference File

Compare records in the input file with the ones in the reference file, at which reference file is specified in `m=` parameter. Set the key parameter at `k=` for selecting records common in both files.

Format

`mcommon k= [K=] [u=] [-r] m=|` i= [o=] [-assert_diffSize] [-assert_nullkey] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`k=`	List of key field(s) to match the input data (Multiple keys can be specified) Records in the input file matching the key field(s) in the reference file specified at `K=` is selected. Records with same key values will appear in consecutive rows.
`m=`	Specify reference file name Standard input is used when this parameter is omitted (when `i=` option is specified).
`K=`	Key field(s) to match with from the reference data (Multiple keys can be specified). Records in the reference file that matches the key field specified in the `k=` parameter is selected. The parameter is not required if the key field is the same on both input data and reference file. Records with same key values will therefore appear in consecutive rows.
`u=`	File name of output with unmatched records.
`-r`	Reverse selection Compare the value specified at `k=` parameter from the input file with the value from the reference file specified at `m=` parameter, and select unmatched record(s) from the input file.

Examples

Example 1: Basic Example

Select records with common customers in both input file and reference file. Save unmatched records to a separate file `oth.csv`.

```
$ more dat1.csv
Customer,Quantity
A,1
B,2
C,1
D,3
E,1
$ more ref1.csv
Customer,Gender
A,Female
B,Male
E,Female
$ mcommon k=Customer m=ref1.csv u=oth.csv i=dat1.csv o=rsl1.csv
#END# kgcommon i=dat1.csv k=Customer m=ref1.csv o=rsl1.csv u=oth.csv
$ more rsl1.csv
Customer%0,Quantity
A,1
B,2
E,1
$ more oth.csv
Customer%0,Quantity
C,1
D,3
```

Example 2: Select unmatched records from the input file

Reverse selection criteria by using the `-r` option, the "Customer" not in the reference file are selected.

```
$ mcommon k=Customer m=ref1.csv -r i=dat1.csv o=rsl2.csv
#END# kgcommon -r i=dat1.csv k=Customer m=ref1.csv o=rsl2.csv
$ more rsl2.csv
Customer%0,Quantity
C,1
D,3
```

Example 3: Different names of join key

If the join key field name in the reference file is different, specify the field name at ￥verb—K=—.

```
$ more ref2.csv
CustomerID,Gender
A,Female
B,Male
E,Female
$ mcommon k=Customer K=CustomerID i=dat1.csv m=ref2.csv o=rsl3.csv
#END# kgcommon K=CustomerID i=dat1.csv k=Customer m=ref2.csv o=rsl3.csv
$ more rsl3.csv
Customer%0,Quantity
A,1
B,2
E,1
```

Example 4: Example with duplicate key fields

Record selection with duplicate records in both input file and reference file.

```
$ more dat3.csv
Customer,Quantity
A,1
A,2
A,3
B,1
D,1
D,2
$ more ref3.csv
Customer
A
A
D
$ mcommon k=Customer m=ref3.csv -r i=dat3.csv o=rsl4.csv
#END# kgcommon -r i=dat3.csv k=Customer m=ref3.csv o=rsl4.csv
$ more rsl4.csv
Customer%0,Quantity
B,1
```

Related commands

mselstr : This command can be used when the types of join key in reference file is not a lot.

mnrcommon : This command can be used when the join key in the reference file is not unique.

mjoin : This command is not only used for selecting data, but also for combining fields.

4.12 mcount - Count the Number of Rows

Count the number of rows and store the results in a new column defined in `a=` parameter. Counting is carried out by each aggregate key when `k=` parameter is specified. Otherwise, if `k=` is not specified, all rows are counted.

Format

`mcount a= [k=]` [i=] [o=] [-nfn] [-nfno] [-assert_diffSize] [-assert_nullkey] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`a=`	Specify the new field name to be added. This parameter is not required when `nfn` option is specified.
`k=`	Key item(s) (Multiple keys can be specified) [aggregate key break processing] Count the number of instances for incremental rows based on the key field(s) defined.

Examples

Example 1: Basic Example

Count the number of rows by date, and save the results in a new column `count`.

```
$ more dat1.csv
date
20090109
20090109
20090109
20090110
20090110
$ mcount k=date a=count i=dat1.csv o=rsl1.csv
#END# kgcount a=count i=dat1.csv k=date o=rsl1.csv
$ more rsl1.csv
date%0,count
20090109,3
20090110,2
```

Example 2: Count without aggregate key

Count the number of rows without specifying the aggregate key.

```
$ mcount a=count i=dat1.csv o=rsl2.csv
#END# kgcount a=count i=dat1.csv o=rsl2.csv
$ more rsl2.csv
date,count
20090110,5
```

Related command

mstats : Specify `c=count` to count non-null values in the data.

4.13 mcross - Crosstab

Build a crosstab. Transpose the fields specified at `s=` parameter horizontally, itemize the array of data specified at the `f=` parameter to the corresponding key specified at k=.

Format

`mcross f= s= [a=] [k=] [v=]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Field name specified will become cross tab fact cell. When multiple fields are specified, the variables will be transposed to multiple rows. The `fld` field is created to identify multiple rows of the field. Specify the name of field at f= as the column variable. Use `a=` parameter to rename the field `fld`.
`s=`	Field names of transposed field. The data series in the field becomes column name of transposed rows.
`a=`	Specify the field name to transpose the row of field names. The field name is set to `fld` by default when this parameter is not defined.
`k=`	Key field name [aggregate key break processing] Key to transpose data into horizontal rows.
`v=`	Null value replacement string Replace null character with a character defined at `v=` parameter.

Examples

Example 1: Basic Example

Expand the array of `date` horizontally and itemize `quantity` to the corresponding `item`.

```
$ more dat1.csv
item,date,quantity,price
A,20081201,1,10
A,20081202,2,20
A,20081203,3,30
B,20081201,4,40
B,20081203,5,50
$ mcross k=item f=quantity s=date i=dat1.csv o=rsl1.csv
#END# kgcross f=quantity i=dat1.csv k=item o=rsl1.csv s=date
$ more rsl1.csv
item%0,fld,20081201,20081202,20081203
A,quantity,1,2,3
B,quantity,4,,5
```

Example 2: Restore the original input data

Restore the output from Example 1 to the original input data with `mcross`.

```
$ more rsl1.csv
item%0,fld,20081201,20081202,20081203
A,quantity,1,2,3
B,quantity,4,,5
$ mcross k=item f=2008* s=fld a=date i=rsl1.csv o=rsl2.csv
#END# kgcross a=date f=2008* i=rsl1.csv k=item o=rsl2.csv s=fld
$ more rsl2.csv
item%0,date,quantity
A,20081201,1
A,20081202,2
```

```
A,20081203,3
B,20081201,4
B,20081202,
B,20081203,5
```

Example 3: Crosstab with multiple fields

Display crosstab results on two fields `quantity,price`.

```
$ mcross k=item f=quantity,price s=date i=dat1.csv o=rsl3.csv
#END# kgcross f=quantity,price i=dat1.csv k=item o=rsl3.csv s=date
$ more rsl3.csv
item%0,fld,20081201,20081202,20081203
A,quantity,1,2,3
A,price,10,20,30
B,quantity,4,,5
B,price,40,,50
```

Example 4: Reverse data sequence

Restore the sequence of the items that was expanded horizontally.

```
$ mcross k=item f=quantity,price s=date%r i=dat1.csv o=rsl4.csv
#END# kgcross f=quantity,price i=dat1.csv k=item o=rsl4.csv s=date%r
$ more rsl4.csv
item%0,fld,20081203,20081202,20081201
A,quantity,3,2,1
A,price,30,20,10
B,quantity,5,,4
B,price,50,,40
```

Related Command

mtra : Creates the same data image of horizontal transposition, but `mtra` output the vector as a single field in the output.

4.14 m2cross Perform 1-to-N Crosstab

This command performs 1-to-N crosstab. Use the `s=` parameter to specify the field whose value will be horizontally transposed as a fieldname. Use the `f=` parameter to specify the field to be output as a cell. Use the `a=` parameter (specify two fields) to specify the values to be fieldnames; the fieldname specified for the `f=` parameter will be vertically transposed to the first field, and the value specified for the `f=` parameter will be vertically transposed to the second field. Use the `k=` parameter to specify the value to be the row id used for the transposition.

Format

`m2cross f= s=|a= [k=] [v=]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Specify the field whose value is to be output as the cell value. Multiple fields can be specified only when a= is used.
`s=`	Specify the field to be transposed as the column fieldname. The value of the field specified here will be output as the fieldname.
`a=`	Specify two fields. In the first field, specify the fieldname to which the fieldname specified for `f=` is transposed. In the second field, specify the fieldname of the field value specified for `f=`.
`k=`	Specify the key fieldname list. The field specified here will be used as the key for transposition.
`v=`	Specify the character string to replace a null value. A null value will be replaced by the character string specified here.

Examples

Example 1: Example 1: Basic Example

The `item` field is used as the key, and the `date` field is horizontally transposed and the `quantity` field is output.

```
$ more dat1.csv
item,date,quantity
A,20081201,1
A,20081202,2
A,20081203,3
B,20081201,4
B,20081203,5
$ m2cross k=item f=quantity s=date i=dat1.csv o=rsl1.csv
#END# kg2cross f=quantity i=dat1.csv k=item o=rsl1.csv s=date
$ more rsl1.csv
item%0,20081201,20081202,20081203
A,1,2,3
B,4,,5
```

Example 2: Example 2: Restoring the original input data

The `m2cross` command is used to restore the original input data for the output results of Example 1.

```
$ more rsl1.csv
item%0,20081201,20081202,20081203
A,1,2,3
B,4,,5
$ m2cross f=2008* a=date,quantity i=rsl1.csv o=rsl2.csv
#END# kg2cross a=date,quantity f=2008* i=rsl1.csv o=rsl2.csv
```

```
$ more rsl2.csv
item%0,date,quantity
A,20081201,1
A,20081202,2
A,20081203,3
B,20081201,4
B,20081202,
B,20081203,5
```

Example 3: Example 3: Reversing data sequence

The sequence of the horizontally transposed fieldnames is reversed.

```
$ m2cross k=item f=quantity s=date%r i=dat1.csv o=rsl4.csv
#END# kg2cross f=quantity i=dat1.csv k=item o=rsl4.csv s=date%r
$ more rsl4.csv
item%0,20081203,20081202,20081201
A,3,2,1
B,5,,4
```

Related Command

mcross : A very similar command. Performs N-to-N crosstab.

4.15 mcsv2arff - Convert csv to arff Format

Convert csv formatted data into arff file (data format for WEKA). User must specify the type of attribute for arff, for instance, `d=` defines category format field, `n=` defines numeric format field, `s=` defines string format field, and finally, `D=` defines date format field. The date format includes time information when attached `%t` to date format field name.

Format

`mcsv2arff n=|d=|D=|s= [T=]` i= [o=] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`n=`	Numeric field name(s) (multiple items can be specified).
`d=`	Category field name(s) (multiple items can be specified).
`D=`	List of date (time) field name(s) (multiple items can be specified). [%t] When `%t` is not specified : yyyyMMdd When `%t` is specified : yyyyMMddHHmmss
`s=`	Character string field names (multiple items can be specified).
`T=`	Title in character string.

Examples

Example 1: Convert csv format data to arff format

Convert data to arff format and define "customer" field as string type, "product" field as category type, "date" field as date type (exclude time), " quantity " and " amount " fields as numeric attributes.

```
$ more dat1.csv
customer,product,date,quantity,amount
No.1,A,20081201,1,10
No.2,A,20081202,2,20
No.3,A,20081203,3,30
No.4,B,20081201,4,40
No.5,B,20081203,5,50
$ mcsv2arff s=customer d=product D=date n=quantity,amount T=Purchase_Data i=dat1.csv  o=rsl1.csv
#END# kgcsv2arff D=date T=Purchase_Data d=product i=dat1.csv n=quantity,amount o=rsl1.csv s=customer
$ more rsl1.csv
@RELATION        Purchase_Data

@ATTRIBUTE        customer        string
@ATTRIBUTE        date        date yyyyMMdd
@ATTRIBUTE        quantity        numeric
@ATTRIBUTE        amount        numeric
@ATTRIBUTE        product        {A,B}

@DATA
No.1,20081201,1,10,A
No.2,20081202,2,20,A
No.3,20081203,3,30,A
No.4,20081201,4,40,B
No.5,20081203,5,50,B
```

Example 2: Convert csv format data to arff format (include time in the date attribute)

Specify the date with the time information by adding `%t` such that `D=date%t`.

```
$ more dat2.csv
customer,product,date,quantity,amount
No.1,A,20081201102030,1,10
No.2,A,20081202123010,2,20
No.3,A,20081203153010,3,30
No.4,B,20081201174010,4,40
No.5,B,20081203133010,5,50
$ mcsv2arff s=customer d=product D=date%t n=quantity,amount T=Purchase_Data i=dat2.csv  o=rsl2.csv
#END# kgcsv2arff D=date%t T=Purchase_Data d=product i=dat2.csv n=quantity,amount o=rsl2.csv s=customer
$ more rsl2.csv
@RELATION        Purchase_Data

@ATTRIBUTE        customer        string
@ATTRIBUTE        date        date yyyyMMddHHmmss
@ATTRIBUTE        quantity        numeric
@ATTRIBUTE        amount        numeric
@ATTRIBUTE        product        {A,B}

@DATA
No.1,20081201102030,1,10,A
No.2,20081202123010,2,20,A
No.3,20081203153010,3,30,A
No.4,20081201174010,4,40,B
No.5,20081203133010,5,50,B
```

Related Command

marff2csv : Reverse conversion.

Reference

http://weka.wikispaces.com/ARFF

4.16 mcsv2json Convert CSV to JSON format

Note: This command is a beta. Its specifications may be changed.

This command converts CSV data into JSON format, which is the object notation syntax for JavaScript. This command has the following characteristics. For details, see the examples.

- Only character strings are supported on JSON.
- Outputs can be arrays and objects (key-value pairs).
- With a key field specified, outputs can be in nesting structures of arrays.

Format

`mcsv2json [k=] [s=] f=|h=|p= -flat` i= [o=] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`k=`	Specify a list of fieldnames for which arrays are nested on JSON Specifying three fields makes three-fold JSON arrays.
`s=`	Specify the order of values. Specify `%n` for numerical order or `%r` for reverse order.
`f=`	Specify the field whose value is to be output as a JSON array.
`h=`	Specify the fieldname used as the key for outputting a JSON object (hash structure).
`p=`	Specify pairs of key and value fieldnames of JSON objects. Use colons as delimiters between the two fieldnames as follows: `p=key fieldname 1: value fieldname 1,key fieldname 2:valu`

Examples

Example 1: Example 1: Outputting arrays

```
$ more dat1.csv
key1,key2,v1,v2
A,X,1,a
A,Y,2,b
A,Y,3,c
B,X,4,d
B,Y,5,e
$ mcsv2json f=v1,v2 i=dat1.csv
[["1","a"],["2","b"],["3","c"],["4","d"],["5","e"]]
#END# kgcsv2json f=v1,v2 i=dat1.csv
```

Example 2: Example 2: Outputting objects (key-value pairs)

```
$ mcsv2json h=v1,v2 i=dat1.csv
[{"v1":"1","v2":"a"},{"v1":"2","v2":"b"},{"v1":"3","v2":"c"},{"v1":"4","v2":"d"},{"v1":"5","v2":"e"}]
#END# kgcsv2json h=v1,v2 i=dat1.csv
```

Example 3: Example 3: Outputting objects according to field specification

```
$ mcsv2json p=v2:v1 i=dat1.csv
[{"a":"1"},{"b":"2"},{"c":"3"},{"d":"4"},{"e":"5"}]
#END# kgcsv2json i=dat1.csv p=v2:v1
```

Example 4: Example 4: Specifying a key field

The three rows whose `key1` field is `A` are output as a single array. Then, the two rows whose `key1` field is `B` are output as a single array.

```
$ mcsv2json k=key1 f=v1 i=dat1.csv
[[["1"],["2"],["3"]],
[["4"],["5"]]]
#END# kgcsv2json f=v1 i=dat1.csv k=key1
```

Example 5: Example 5: Nesting key fields

One row where `key1=A` and `key2=X` is output as a single array, two rows where `key1=A` and `key2=Y` are output as a single array, and the two arrays (that is, rows where `key1=A`) are bundled as a single array.

```
$ mcsv2json k=key1,key2 f=v1 i=dat1.csv
[[[["1"]],
[["2"],["3"]]],
[[["4"]],
[["5"]]]]
#END# kgcsv2json f=v1 i=dat1.csv k=key1,key2
```

Example 6: Example 6: Outputting rows flatly without bundling them as an array

```
$ mcsv2json f=v1,v2 -flat i=dat1.csv
["1","a","2","b","3","c","4","d","5","e"]
#END# kgcsv2json -flat f=v1,v2 i=dat1.csv
```

Related Commands

Reference

http://www.rfc-editor.org/rfc/rfc4627.txt

4.17 mcsvconv Convert CSV into various other formats

Note: This command is a beta. Its specifications may be changed.

This command converts CSV data into various other formats. Prepare a text file describing the rules of the conversion target data format, and the command outputs the specified CSV data fields according to the rule keywords. The following describes the fieldnames and output timings to be specified in the rule file.

Fieldname keyword Enclose a fieldname between two `%` symbols, and it will be replaced with the corresponding field value in the CSV file. Ex. `%fieldname%`

Output timing keyword There are three types of keywords that determine the CSV data output timing:

LINEDATA Use `%LINEDATA` and `%LINEEND` to enclose a fieldname keyword or any other character string, and it will be output every time a row of the CSV file is read.

KEYHEAD Use `%KEYHEAD` and `%KEYEND` to enclose a fieldname keyword or any other character string, and it will be output only when a keybreak has occurred for the key field specified for `k=`. It will be output before LINEDATA.

KEYFOOT Use `%KEYFOOT` and `%KEYEND` to enclose a fieldname keyword or any other character string, and it will be output only when a keybreak has occurred for the key field specified for `k=`. It will be output after LINEDATA.

Before and after a keyword Character strings will be output only once before the CSV file is read if they are placed before a block enclosed between any of the above timing keywords. Character strings will be output only once after the CSV file is read if they are placed after the last such block.

When multiple fields are specified for the `k=` parameter, you can control timing more precisely with `KEYHEAD` and `KEYFOOT`. By specifying a number at the end, e.g., `%KEYHEAD1`, you can specify which field to monitor for a keybreak. For instance, if you specify k=A,B,C and `%KEYHEAD1`, data will be output when a keybreak has occurred for key field `A`. If you specify `%KEYHEAD2`, data will be output when a keybreak has occurred for key fields `A` and `B`. If you specify `%KEYHEAD3` or just `%KEYHEAD`, data will be output when a keybreak has occurred for all of key fields `A`, `B`, and `C`. When an output timing control keyword is described in a row, no other character can be described in it.

Format

`mcsvconv [k=] [s=] m=` i= [o=] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`k=`	Key fieldname list
`s=`	A fieldname list that determines the sorting sequence in a row where the values in the fields specified for `k=` are the same.
`m=`	Rule filename

Examples

Example 1: Example 1: Basic example 1

Two fields in the CSV file, `key1` and `v1`, are output with spaces as delimiters.

```
$ more dat1.csv
key1,key2,v1,v2
A,X,1,a
A,Y,2,b
A,Y,3,c
B,X,4,d
B,Y,5,e
$ more form1.txt
%LINEDATA
```

```
%key1% %v1%
%LINEEND
$ mcsvconv m=form1.txt i=dat1.csv o=rsl1.txt
#END# kgcsvconv i=dat1.csv m=form1.txt o=rsl1.txt
$ more rsl1.txt
A 1
A 2
A 3
B 4
B 5
```

Example 2: Example 2: Basic example 2

Data is output in two rows, with the second rows indented.

```
$ more dat1.csv
key1,key2,v1,v2
A,X,1,a
A,Y,2,b
A,Y,3,c
B,X,4,d
B,Y,5,e
$ more form2.txt
%LINEDATA
%key1% %v1%
       %key2% %v2%
%LINEEND
$ mcsvconv m=form2.txt i=dat1.csv o=rsl2.txt
#END# kgcsvconv i=dat1.csv m=form2.txt o=rsl2.txt
$ more rsl2.txt
A 1
       X a
A 2
       Y b
A 3
       Y c
B 4
       X d
B 5
       Y e
```

Example 3: Example 3: Specifying keys

```
$ more dat1.csv
key1,key2,v1,v2
A,X,1,a
A,Y,2,b
A,Y,3,c
B,X,4,d
B,Y,5,e
$ more form3.txt
Head Area
%KEYHEAD
KeyHead=%key1% %key2% %v1% %v2%
%KEYEND
%LINEDATA
%v1%-%v2%
%LINEEND
%KEYFOOT
KeyFoot=%key1% %key2% %v1% %v2%
%KEYEND
Foot Area
$ mcsvconv k=key1 m=form3.txt i=dat1.csv o=rsl3.txt
#END# kgcsvconv i=dat1.csv k=key1 m=form3.txt o=rsl3.txt
$ more rsl3.txt
Head Area
KeyHead=A X 1 a
1-a
2-b
3-c
KeyFoot=A Y 3 c
KeyHead=B X 4 d
4-d
```

```
5-e
KeyFoot=B Y 5 e
Foot Area
```

Example 4: Example 4: Outputting results as tex data

```
$ more dat1.csv
key1,key2,v1,v2
A,X,1,a
A,Y,2,b
A,Y,3,c
B,X,4,d
B,Y,5,e
$ more form3.txt
Head Area
%KEYHEAD
KeyHead=%key1% %key2% %v1% %v2%
%KEYEND
%LINEDATA
%v1%-%v2%
%LINEEND
%KEYFOOT
KeyFoot=%key1% %key2% %v1% %v2%
%KEYEND
Foot Area
$ mcsvconv k=key1 m=form4.txt i=dat1.csv o=rsl4.tex
#END# kgcsvconv i=dat1.csv k=key1 m=form4.txt o=rsl4.tex
$ more rsl4.tex
\documentclass{article}
\begin{document}
\begin{table}
\begin{tabular}{l|l|r|r}
\hline
key1 & key2 & v1 & v2 \\
\hline
A & X & 1 & a \\
  & X & 1 & a \\
  & Y & 2 & b \\
  & Y & 3 & c \\
\hline
B & X & 4 & d \\
  & X & 4 & d \\
  & Y & 5 & e \\
\hline
\end{tabular}
\end{table}
\end{document}
```

Related Commands

mcsv2arff : Converts CSV data into arff data (data format for WEKA).
mcsv2json : Converts CSV data into JSON format, the object notation syntax of Java Script.

4.18 mcut - Select Column

Extract the specified column(s). The specified column is removed with `-r` option.

Format

`mcut f= [-r] [-nfni]` [i=] [o=] [-assert_diffSize] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help]` `[--helpl] [--version]`

Parameters

`f=`	Define name of column to be extracted New column name for can be specified by defining the field name, followed by colon and the new field name. ex. `f=a:A,b:B`
`-r`	Field removal switch Remove all columns defined in the `f=` parameter.
`-nfni`	When header is not present in first row of the input data, position number of column is used to identify corresponding field(s). New column name(s) for each column can be specified in the output file as follows. Example f=0:date,2:store,3:quantity

Examples

Example 1: Basic Example

Extract customer and amount information from the data file `dat1.csv` Rename the column "amount " to "sales" in the output.

```
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ mcut f=customer,amount:sales i=dat1.csv o=rsl1.csv
#END# kgcut f=customer,amount:sales i=dat1.csv o=rsl1.csv
$ more rsl1.csv
customer,sales
A,10
A,20
B,15
B,10
B,20
```

Example 2: Remove columns

Remove columns customer and amount specified at `-r`.

```
$ mcut f=customer,amount -r i=dat1.csv o=rsl2.csv
#END# kgcut -r f=customer,amount i=dat1.csv o=rsl2.csv
$ more rsl2.csv
quantity
1
2
1
3
1
```

Example 3: Data without field names

Select columns 0, 2 from an input file without field header, add `customer` and `amount` as field names in the output file.

```
$ mcut f=0:customer,2:amount -nfni i=dat1.csv o=rsl3.csv
#END# kgcut -nfni f=0:customer,2:amount i=dat1.csv o=rsl3.csv
$ more rsl3.csv
customer,amount
customer,amount
A,10
A,20
B,15
B,10
B,20
```

related command

mfldname : Use `mfldname` to change the field names.

4.19 mdata Generate datasets

This command generates various datasets. For details of the datasets, see the descriptions below.

Format

```
mdata [O=] -iris|-man0|-man1|-tutorial_en|-tutorial_jp|-yakiniku_en|-yakiniku_jp [--help] [--helpl]
[--version]
```

Parameters

`O=`	Output filename. By default, data is output to the standard output. When `-tutorial_jp` or `-tutorial_en` is specified, it is a directory name. By default, `O=tutorial_jp` or `O=tutorial_en` is assumed.
`-iris`	A data set designed for building a classification model for the species of irises based on the sepal and petal sizes Fieldnames: SepalLength, SepalWidth, PetalLength, PetalWidth, Species http://archive.ics.uci.edu/ml/datasets/Iris?ref=datanews.io
`man0`	5-row data used in Figure 2.4 of this manual Fieldnames: customer, amount
`man1`	8-row data used in Figure 2.6 of this manual Fieldnames: customer, date, product

`yakiniku_jp`	Sales data from a rotisserie (sales data from Rotisserie Fukugyu provided by Okayama Food Service)
	`Fieldnames: 日付 (date), 時間 (time), レシート (receipt), 商品 (product), 単価 (unit price), 数量 (quantity), 金`
	`Note 1: The 日付 (date) field holds the order time. Even on the same receipt, a different time is recor`
	`Note 2: 金額 (totalAmount)=単価 (unit price) ×数量 (quantity)`

`yakiniku_en`	English version of the rotisserie order data Fieldnames: date,time,receipt,item,price,quantity,totalAmount

`tutorial_jp`	Pseudo purchase data of a supermarket used in the tutorial
	`dat.csv`: Purchase history data
	`Fieldnames:店 (store), 日付 (date), 時間 (time), レシート (receipt), 顧客 (customer), 商品 (product), 大分類 (Ca`
	`仕入単価 (purchase price), 単価 (unit price), 数量 (quantity), 金額 (totalAmount), 仕入金額 (purchase amount)`
	`syo.csv`: product master
	`Fieldnames: 商品 (product), 商品名 (product name), 大分類 (CategoryCode1), 中分類 (CategoryCode2), 小分類 (Ca`
	`cust.csv`: Customer master
	`Fieldnames: 顧客 (customer), 生年月日 (birthday), 性別 (sex)`
	`jicfs1,2,4,6.csv`: Product category master
	Fieldnames: 大分類 (CategoryCode1), 大分類名 (Category1), 中分類 (CategoryCode2), 中分類名 (Category2), 小分類 (Cat

`tutorial_en`	English version of the tutorial_jp dataset `dat.csv`: Purchase history data Fieldnames: shop,date,time,receipt,customer,product,CategoryCode1,CategoryCode2,CategoryCode4, CategoryCode6,manufacturer,brand,unitCost,unitPrice,quantity,amount,costAmount,profit `syo.csv`: product master Fieldnames: product,productName,CategoryCode1,CategoryCode2,CategoryCode4,CategoryCode6, manufacturer,brand,unitCost `cust.csv`: customer master Fieldnames: customer,dob,gender `jicfs1,2,4,6.csv`: product category master Fieldnames: CategoryCode1,Category1(CategoryCode2,Category2)(CategoryCode4,Category4) (CategoryCode6,Category6)

Examples

Example 1 Generating iris dataset

The iris dataset is generated and sent to the standard output.

```
$ mdata -iris
SepalLength,SepalWidth,PetalLength,PetalWidth,Species
```

```
5.1,3.5,1.4,0.2,setosa
4.9,3,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
4.6,3.1,1.5,0.2,setosa
         :
```

Example 2 Creating tutorial datasets

All tutorial datasets are output to a file.

```
$ mdata -tutorial_en
#END# mdata -tutorial_en

$ ls -l tutorial_en
total 4704
-rw-r--r--  1 nysol  staff    20673  8 22 08:14 cust.csv
-rw-r--r--  1 nysol  staff  2281312  8 22 08:14 dat.csv
-rw-r--r--  1 nysol  staff      128  8 22 08:14 jicfs1.csv
-rw-r--r--  1 nysol  staff      529  8 22 08:14 jicfs2.csv
-rw-r--r--  1 nysol  staff     6630  8 22 08:14 jicfs4.csv
-rw-r--r--  1 nysol  staff    36400  8 22 08:14 jicfs6.csv
-rw-r--r--  1 nysol  staff    46466  8 22 08:14 syo.csv

$ more tutorial_en/dat.csv
customer,dob,gender
00000A,19711107,female
00000B,19461025,female
00000C,19660307,female
         :
```

Example 3 Generating rotisserie data

```
$ mdata -yakiniku_jp
日付,時間,レシート,商品,単価,数量,金額
20070701,1123,10000,焼肉ヘルシーセット,1410,1,1410
20070701,1152,10001,和牛焼肉弁当,1240,1,1240
20070701,1202,10002,ランチコーヒー,130,2,260
             :
$ mdata -yakiniku_en
date,time,receipt,item,price,quantity,totalAmount
20070701,1123,10000,Low-fat BBQ set,1410,1,1410
20070701,1152,10001,Japanese grilled beef lunch box,1240,1,1240
20070701,1202,10002,Lunchtime coffee,130,2,260
         :
```

4.20 mdelnull - Remove rows with null values

Remove the row if values defined at `f=` parameter contain null value(s).

Format

`mdelnull f= [k=] [u=] [-F] [-r] [-R]` [i=] [o=] [bufcount=] [-assert_diffSize] [-assert_nullkey] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Search NULL values in specified field name(s) (Multiple fields can be specified).
`k=`	Remove records with null values based on the same key field(s) defined (multiple fields can be specified).
`u=`	Print unmatched records to the output file specified.
`-F`	AND condition for multiple fields Records consisting of null values in multiple fields defined at `f=` parameter will be selected and removed.
`-r`	Reverse selection Selected records are not removed.
`-R`	AND condition for multiple records Remove all records if the field with the same key field specified at the `k=` parameter contains NULL values.

Examples

Example 1: Basic Example

Remove records where `Quantity` and `Amount` contain null values. Save records with null values to a separate file `oth.csv`.

```
$ more dat1.csv
Customer,Quantity,Amount
A,1,10
A,,20
B,1,15
B,3,
C,1,20
$ mdelnull f=Quantity,Amount u=oth.csv i=dat1.csv o=rsl1.csv
#END# kgdelnull f=Quantity,Amount i=dat1.csv o=rsl1.csv u=oth.csv
$ more rsl1.csv
Customer,Quantity,Amount
A,1,10
B,1,15
C,1,20
$ more oth.csv
Customer,Quantity,Amount
A,,20
B,3,
```

Example 2: Select rows with NULL values

Select records with NULL values by specifying `-r`.

```
$ mdelnull f=Quantity,Amount -r i=dat1.csv o=rsl2.csv
#END# kgdelnull -r f=Quantity,Amount i=dat1.csv o=rsl2.csv
$ more rsl2.csv
Customer,Quantity,Amount
A,,20
B,3,
```

Example 3: Remove records with the same key if any record contains NULL values

Remove based on the aggregate key specified at `k=`. Remove records where `Quantity` and `Amount` contain null values for each customer.

```
$ mdelnull k=Customer f=Quantity,Amount i=dat1.csv o=rsl3.csv
#END# kgdelnull f=Quantity,Amount i=dat1.csv k=Customer o=rsl3.csv
$ more rsl3.csv
Customer%0,Quantity,Amount
C,1,20
```

Example 4: AND condition between fields

Remove the record where both `Quantity` and `Amount` fields contain null values.

```
$ more dat2.csv
Customer,Quantity,Amount
A,1,10
A,,
B,1,15
B,3,
C,1,20
$ mdelnull f=Quantity,Amount -F i=dat2.csv o=rsl4.csv
#END# kgdelnull -F f=Quantity,Amount i=dat2.csv o=rsl4.csv
$ more rsl4.csv
Customer,Quantity,Amount
A,1,10
B,1,15
B,3,
C,1,20
```

Example 5: AND condition between records

Remove the `Customer` record if all values in `Quantity` is NULL.

```
$ mdelnull k=Customer f=Quantity -R i=dat1.csv o=rsl5.csv
#END# kgdelnull -R f=Quantity i=dat1.csv k=Customer o=rsl5.csv
$ more rsl5.csv
Customer%0,Quantity,Amount
A,1,10
A,,20
B,1,15
B,3,
C,1,20
```

Related Command

mnullto : NULL value(s) in records are convert to specified character strings.

Does not delete the row that contains a NULL value, it is converted to a string of digits of a NULL value.

4.21 mdformat Extract Date Time

CSV data that was exported from other systems often contains forward slash and colon symbol in date columns, in addition, date and time are stored in single digit (Example: `2014/7 / 18 1:57`). Sorting and range specification processing on these kind of items in MCMD is not possible.

For ease of processing data and time formatted data Date Time Format, the `mdformat` command extracts the date, hour, minute, and second on fields specified in `f=` parameter according to the format specified in the `c=` parameter.

Format

`mdformat c= f= [-A]` [i=] [o=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Specify the field name for extraction (multiple items can be specified).
`c=`	Specify string format according to the specified format.
`-A`	Specify the new column name and save output results in new column.

Conversion Specification of Character Format

Table 4.1 shows the possible character conversion formats that can be defined in `c=` parameter.

Table 4.1: Conversion characters

Conversion Characters	Description
%Y	Number representing year (4 digits)
%y	Number representing year (2 digits)
%m	Number representing month (2 digits)
%d	Number representing day (2 digits)
%H	Time (2 digits)
%M	Minute (2 digits)
%S	Second (2 digits)

Examples

Example 1: Basic Example

Extract and convert time and date information from `fld` field. Save the converted format as "a:yearmonthday:b:timeminutesecond by specifying "`a:%Y%m%d:b:%H%M%S`" in the `c=` parameter.

```
$ more dat1.csv
fld
a:20120304:b:121212
a:20101204:b:011309
$ mdformat f=fld c=a:%Y%m%d:b:%H%M%S i=dat1.csv o=rsl1.csv
#END# kgdformat c=a:%Y%m%d:b:%H%M%S f=fld i=dat1.csv o=rsl1.csv
$ more rsl1.csv
fld
```

```
20120304121212
20101204011309
```

Example 2: Add Results to New Column

Store results in `fld2` from format conversion in `fld1` field, specify the format by "`%Y/%m/%d`" in `c=` parameter. Use `-A` option to save results in `f2` field.

```
$ more dat2.csv
fld,fld2
2010/11/20,2010/11/21
2010/1/1,2010/1/2
2011/01/01,2010/01/02
2010/1/01,2010/1/02
$ mdformat f=fld:f1,fld2:f2 c=%Y/%m/%d i=dat2.csv -A o=rsl2.csv
#END# kgdformat -A c=%Y/%m/%d f=fld:f1,fld2:f2 i=dat2.csv o=rsl2.csv
$ more rsl2.csv
fld,fld2,f1,f2
2010/11/20,2010/11/21,20101120,20101121
2010/1/1,2010/1/2,20100101,20100102
2011/01/01,2010/01/02,20110101,20100102
2010/1/01,2010/1/02,20100101,20100102
```

Example 3: Case of failed extraction

The date format in `fld` field is saved as "Year Month Day Time:Minute:Second", "`%Y %m %d %H:%M:%S`" is specified in `c=` parameter. However, it failed since the format is different in different rows.

```
$ more dat3.csv
fld
2010 11 20 12:34:56

2011 01 01 23:34:56
2010  1 01 123455
$ mdformat f=fld:f1 c='%Y %m %d %H:%M:%S' i=dat3.csv -A o=rsl3.csv
#END# kgdformat -A c=%Y %m %d %H:%M:%S f=fld:f1 i=dat3.csv o=rsl3.csv
$ more rsl3.csv
fld,f1
2010 11 20 12:34:56,20101120123456
,
2011 01 01 23:34:56,20110101233456
2010  1 01 123455,
```

Related Command

4.22 mdsp Display character strings on screen

Note: This command is a beta. Its specifications may be changed.

This command displays the character string specified by the `str=` parameter at the position on the terminal specified by coordinate `x=`,`y=`. Use the `i=` parameter to specify a filename, and its contents will be displayed. When both `str=` and `i=` are specified, `i=` will prevail. If multiple rows are specified for the `i=` parameter, all rows will be displayed from the coordinate specified by `x=`. Use the `fc=` parameter to specify the color of the character string. Use the `bg=` parameter to specify the background color of the character string. You can choose from eight colors: black, red, green, yellow, blue, magenta, cyan, and white. By default, `fc=black` and `bg=white` are assumed. Specify the `-bold` option to display the character string in bold letters.

Format

```
mdsp x= y= str=|i= [fc=] [bg=] [-bold]  [--help] [--helpl] [--version]
```

Parameters

`x=`	Specify the display start position (1 or greater) on the x axis (horizontal, left to right).
`y=`	Specify the display start position (1 or greater) on the y axis (vertical, top to bottom).
`str=`	Character string to be displayed
`i=`	Filename of the contents to be displayed
`fc=`	Character color
`bg=`	Background color
`-bold`	Bold letter

Examples

Example 1: Basic example

Character string `abcd` is displayed at `x=10,y=5` on the terminal.

```
$ mdsp x=10 y=5 str=abcd

The following will be displayed:
+---------------------------------------
|
|
|
|
|          abcd
|
|
```

Example 2: Using a filename to specify what to display

The contents of `dat.txt` are displayed at `x=10,y=5` on the terminal.

```
$ more dat.txt
abcd
efg
$ mdsp x=10 y=5 i=dat.txt

The following will be displayed:
+---------------------------------------
|
|
|
|
|          abcd
|          efg
|
```

Example 3: Using colors

Character string `abcd` is displayed at `x=10,y=5` on the terminal, with the characters inbeing red and the background inbeing blue.

```
$ mdsp x=10 y=5 str=abcd fc=red bc=blue

The following will be displayed:
+--------------------------------------
|
|
|
|
|          \textColor{blue}{red}{abcd}
|
|
```

Related Commands

minput : Displays the input screen. mminput : Displays the input screen consisting of multiple input frames. mseldsp : Displays a single-choice input window on the screen. mmseldsp : Displays a multiple-choice input window on the screen.

4.23 mduprec - Duplicate Record

Duplicate each record. Specify the number of rows to duplicate at `n=`, and duplicate records based on the field specified at `f=` parameter.

Format

`mduprec f=|n=` [i=] [o=] [-assert_diffSize] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Duplicate records based on the values in the field name. Duplicate the number of rows for each record based on the numeric values in the column.
`n=`	Number of records to duplicate Specify the number of rows to duplicate for each record.

Examples

Example 1: Basic Example

Generate multiple records of the data based on the numeric value in the " Quantity " field. Records containing NULL values will not be duplicated.

```
$ more dat1.csv
store,val
A,2
B,
C,5
$ mduprec f=val i=dat1.csv o=rsl1.csv
#END# kgduprec f=val i=dat1.csv o=rsl1.csv
$ more rsl1.csv
store,val
A,2
A,2
C,5
C,5
C,5
C,5
C,5
```

Example 2: Define the number of rows to duplicate

Duplicate two rows (`n=2`) for each record in the dataset.

```
$ mduprec n=2 i=dat1.csv o=rsl2.csv
#END# kgduprec i=dat1.csv n=2 o=rsl2.csv
$ more rsl2.csv
store,val
A,2
A,2
B,
B,
C,5
C,5
```

Related Commands

mcount : Reverse the operation of `mduprec`.

mwindow : Copy and shift a specified number of records.

4.24 mfldname - Rename Field

Specify the field to rename at `f=`, and the new column name at `n=`. Add the -q option to revert to the field header names as appeared in Ver. 1, and removes the sort order symbols appended to field names in output for 2.0 commands.

Format

`mfldname f=|n= [-nfni]` [i=] [o=] [-assert_diffSize] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Specify the field name to change (current field name:new field name). The original field name will not change if this parameter is not set.
`n=`	Specify the new target column name. The number of item names must be the same as the number of columns in the data.
`-nfni`	Field name not present in input data. This option cannot be used with `f=`.

Examples

Example 1: Basic Example

Change column name from " customer " to " cust ", and " october " to " oct ".

```
$ more dat1.csv
customer,itemID,october
a,xx,11
b,yy,122
c,zz,
$ mfldname f=customer:cust,october:oct. i=dat1.csv o=rsl1.csv
#END# kgfldname f=customer:cust,october:oct. i=dat1.csv o=rsl1.csv
$ more rsl1.csv
cust,itemID,oct.
a,xx,11
b,yy,122
c,zz,
```

Example 2: Rename column

Change field names to `x,y,z`.

```
$ mfldname n=x,y,z i=dat1.csv o=rsl2.csv
#END# kgfldname i=dat1.csv n=x,y,z o=rsl2.csv
$ more rsl2.csv
x,y,z
a,xx,11
b,yy,122
c,zz,
```

Example 3: Data without field names

```
$ more dat2.csv
a,xx,11
b,yy,122
c,zz,
$ mfldname -nfni n=x,y,z i=dat2.csv o=rsl3.csv
#END# kgfldname -nfni i=dat2.csv n=x,y,z o=rsl3.csv
$ more rsl3.csv
```

```
x,y,z
a,xx,11
b,yy,122
c,zz,
```

Example 4: Remove sort order symbols in field names

```
$ more dat3.csv
customer%r,itemID,october
c,zz,
b,yy,122
a,xx,11
$ mfldname -q  i=dat3.csv o=rsl4.csv
#END# kgfldname -q i=dat3.csv o=rsl4.csv
$ more rsl4.csv
customer,itemID,october
c,zz,
b,yy,122
a,xx,11
```

Related Command

mcut : Performs similar function to `mfldname`, however, the operation is more complicated to rename certain items. Nevertheless, the operation is slightly faster using `mfldname`.

4.25 mfsort - Sort Field

Sort according to the values of the specified fields at `f=` within each record (in default ascending order by character string). Note that this does not change the sequence of field names.

Format

`mfsort f= [-r] [-n]` [i=] [o=] [-assert_diffSize] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Specify multiple fields where data items are sorted. The result remains the same when one field is defined.
`-n`	Arrange in numerical order.
`-r`	Arrange in reverse order.

Examples

Example 1: Basic Example

Arrange the values in `v1,v2,v3` in ascending order for each record, and output the data items in sequential order corresponding to fields `v1,v2,v3`.

```
$ more dat1.csv
id,v1,v2,v3
1,b,a,c
2,a,b,a
3,b,,e
$ mfsort f=v* i=dat1.csv o=rsl1.csv
#END# kgfsort f=v* i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3
1,a,b,c
2,a,a,b
3,,b,e
```

Example 2: Descending Order

Add `-r` to arrange in descending order.

```
$ mfsort f=v* -r i=dat1.csv o=rsl2.csv
#END# kgfsort -r f=v* i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3
1,c,b,a
2,b,a,a
3,e,b,
```

Related Command

4.26 mhashavg - Compute Average with Hash Function

Calculate the average of data series specified at `f=` parameter based on the key at `k=` with hash function.

The processing speed of this command is faster than mavg since the key fields do not have require prior sorting. However, variation in key lengths (different length of strings in field) will slow down the processing speed.

Format

`mhashavg f= [hs=] [k=] [-n]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Calculate the average of the field name (Multiple fields can be specified) . Specify the new field name after colon ":". Example: f=Quantity:AverageQuantity.
`k=`	Calculate the average on the data series based on the key field(s) (Multiple keys can be specified). This command do not use aggregate key break processing, prior sorting is not required.
`hs=`	Hash size (Default value: 199999) Refer to mhashsum for related information.
`-n`	Return NULL in output if there are null values in `f=`.

Examples

Example 1: Basic Example

Calculate the average `Quantity` and average `Amount` for each `Customer`.

```
$ more dat1.csv
Customer,Quantity,Amount
A,1,
B,,15
A,2,20
B,3,10
B,1,20
$ mhashavg k=Customer f=Quantity,Amount i=dat1.csv o=rsl1.csv
#END# kghashavg f=Quantity,Amount i=dat1.csv k=Customer o=rsl1.csv
$ more rsl1.csv
Customer,Quantity,Amount
A,1.5,20
B,2,15
```

Example 2: NULL value in output

The output returns NULL if there NULL value is present in `Quantity` and `Amount`. Use `-n` option to print the null value.

```
$ mhashavg k=Customer f=Quantity,Amount -n i=dat1.csv o=rsl2.csv
#END# kghashavg -n f=Quantity,Amount i=dat1.csv k=Customer o=rsl2.csv
$ more rsl2.csv
Customer,Quantity,Amount
A,1.5,
B,,15
```

Remarks

Refer to the benchmark at mhashsum to find out more on processing speed.

Related commands

mavg : Compute average

mhashsum : Compute hash total value

4.27 mhashsum - Compute Total Sum Using Hash

Calculate the hash total value of the column specified at the `f=` parameter for each line item based on the key specified at the `k=` parameter.
The processing speed of this command is faster than `msum` since key fields do not require prior sorting. However, variation in length of keys (different length of strings in field) will slow down the processing speed.
User shall assess the usage of mhashsum and msum based on the contents of the data (Refer to "Benchmark" in the second half of this manual).

Format

`mhashsum f= [hs=] [k=] [-n]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Compute the sum of of values in the column specified (Multiple fields can be specified) Specify the new field name after colon ":". Example f=Quantity:TotalQuantity.
`k=`	Calculate the sum on the data series based on the key field(s) (Multiple keys can be specified.) This command do not use aggregate key break processing, prior sorting is not required.
`hs=`	Hash size [Default value: 199999] User shall specify the key size for data processing based on speed and memory consumption optimisation requirements. Prime number should be used as hash table size. The processing speed will slow down if the hash table is not big enough for data with large key size. A larger hash table will speed up processing but will also require more memory (Refer to "Benchmark" in the second half of this manual). Estimating memory requirements: K*(24+F*16) byte, K: key size, F: number of fields specified `f=` parameter.
`-n`	Return NULL in output if there are null values in `f=`.

Examples

Example 1: Basic Example

Calculate the total `Quantity` and total `Amount` for each `Customer` using the hash function.

```
$ more dat1.csv
Customer,Quantity,Amount
A,1,
B,,15
A,2,20
B,3,10
B,1,20
$ mhashsum k=Customer f=Quantity,Amount i=dat1.csv o=rsl1.csv
#END# kghashsum f=Quantity,Amount i=dat1.csv k=Customer o=rsl1.csv
$ more rsl1.csv
Customer,Quantity,Amount
A,3,20
B,4,45
```

Example 2: NULL value in output

The output returns NULL if NULL value is present in `Quantity` and `Amount`. Use `-n` option to print the null value.

```
$ mhashsum k=Customer f=Quantity,Amount -n i=dat1.csv o=rsl2.csv
#END# kghashsum -n f=Quantity,Amount i=dat1.csv k=Customer o=rsl2.csv
$ more rsl2.csv
Customer,Quantity,Amount
A,3,
B,,45
```

Overview of algorithm

`mhashsum` command uses a hash method known as separate chaining.
In this method, a M sequence or a list is created containing all the keys that hash to the same value. The hash function converts and stores the character string containing the keys into integer (hash values) from 0 to M. Two or more keys with the same hash value (conflict keys) will be stored in a linked list from the conflicted slot of hash table. The address of keys are stored in sequential order, but the list is searched in linear order. Thus, the lookup procedure will scan all its entries, and the processing speed decreases with more key conflicts. The default hash size for `mhashsum` is 199999, if the key size is up to 200,000, the average list size is 1 less of the key size. Multiple key conflicts may occur even if the key size is small depending on the data content and structure. The key size can be changed at the `hs=` parameter (maximum value: 1999999).

Benchmark test

Method of Benchmark test

Compare the computation speed of mhashsum command (hash size: 199,999) and msum command (sort data using msort command beforehand) on 13 different types of data with 10 to 1,000,000 key sizes.

Sample data with two columns (key and fld) and 500 million rows of random values is generated as shown in the following table. The key is a 6 digit fixed-length numerical value and the fld is a 3 digit number.

Table 4.2: Table 1: Sample data for Benchmark test

key	fld
100020	120
100007	107
100029	129
100065	165
100030	130
100088	188
100055	155
100093	193
100072	172

Commands for benchmark

Using the mhashsum method
`$ time mhashsum k=key f=fld i=dat.csv o=/dev/null`

Using the msort+msum method
`$ time msort i=dat.csv` msum k=key f=fld o=/dev/null —

Experiment Results

The proceeding speed of `mhashsum` is five times faster than sorting when the key size is small (10,000). As the key size increases, the difference between the two methods is reduced, the processing speed of the two methods are the same when the key size is more than 800,000.

The following is a guideline on the usage of `msum` or `mhashsum`, actual results varies depending on the distribution of the key values.

Table 4.3: Table 2: Comparison of processing speed for mhashsum and msum(msort+msum)

key size	mhashsum(a)(second)	msort+msum(b)(second)	ratio(a/b)
100	0.672	2.955	0.227
1,000	0.731	3.981	0.184
10,000	0.814	4.201	0.194
100,000	1.793	4.291	0.418
200,000	2.241	4.336	0.517
300,000	2.604	4.394	0.593
400,000	2.993	4.448	0.673
500,000	3.380	4.497	0.752
600,000	3.793	4.579	0.828
700,000	4.128	4.618	0.894
800,000	4.514	4.667	0.967
900,000	4.901	4.707	1.041
1,000,000	5.352	4.771	1.122

Benchmark environment

iMac, Mac OS X 10.5 Leopard, 2.8GHz Intel Core 2 Duo, 4GB memory

Related commands

msum : Same computation function but requires key break process.

mhashavg : Compute average using the same hash function

4.28 minput Enable on-screen input

Note: This command is a beta. Its specifications may be changed.

This command displays an input frame of the length specified by the `len=` parameter at a position on the terminal specified by coordinate `x=,y=`, and outputs the input contents to the file specified by the `o=` parameter. When the Enter key is pressed on the input screen, the command returns end status 0 and ends. When the ESC key is pressed on the screen, the command returns end status 1 and ends. In either case, the contents that have been entered that far are output to the file. The output CSV file contains one row and one field. When the command ends with nothing entered in the input frame, null is output (that is, only a line feed is output). To output a fieldname, use the `f=` parameter. With `f=` omitted, no fieldname header is output. In the coordinate system, the top left is `x=1,y=1` (escape sequence specifications). If the value specified for `x=` or `y=` is smaller than 1, the command assumes 1. The operation is indefinite if the specified coordinate is outside the scope of the terminal.

Format

```
minput x= y= len= [f=]  o= [-nfn] [-nfno] [-x] [--help] [--helpl] [--version]
```

Parameters

`x=`	Specify the display start position (1 or greater) on the x axis (horizontal, left to right).
`y=`	Specify the display start position (1 or greater) on the y axis (vertical, top to bottom).
`len=`	Specify the length by the number of single-byte characters in the input area.
`f=`	Specify the output fieldname.

Examples

Example 1: Basic example

A 12-character input area is displayed at x=10,y=5 on the terminal. The user types abcd in the input frame and presses Enter. The input results are saved as rsl1.csv.

```
$ minput x=10 y=5 len=12 o=rsl1.csv
$ more rsl1.csv
abcd

The following will be displayed:
+---------------------------------------
|
|
|
|
|          [abcd        ]
|
|
```

Example 2: Basic example, with a fieldname specified

Operation is the same as in Example 1, with an addition of the f= parameter to specify a fieldname.

```
$ minput x=10 y=5 len=12 f=name o=rsl1.csv
$ more rsl1.csv
name
abcd
```

Example 3: Judging end status

The parameters are the same as in Example 1. This script judges the end status and performs different operations accordingly.

```
$ more scp.sh
rm -f rsl2.csv
clear
minput x=10 y=5 len=12 o=rsl2.csv
if [ $? = 0 ] ; then
  clear ; echo "end by enter key"
else
  clear ; echo "end by escape key"
fi

# Result of typing abcd and pressing Enter
$ bash scp.sh
end by enter key
$ more rsl2.csv
abcd

# Result of typing abcd and pressing ESC key
$ bash scp.sh
end by escape key
$ more rsl2.csv
abcd
```

Related Commands

mminput : Displays the input screen consisting of multiple input frames. mdsp : Displays a character string at the specified position on the screen. mseldsp : Displays a single-choice input window on the screen. mmseldsp : Displays a multiple-choice input window on the screen.

4.29 mjoin - Join field(s) from Reference File

Compare the key field(s) from the input file specified at the `k=` parameter with the ones from the reference file, field from the reference file specified at `f=` parameter are joined for records with common key values in both files. The key fields from the reference file must be unique. Use the mnjoin command when there are more than one record with the same key values in the reference file. If `f=` is not set, all columns are joined except the key field in reference file.

Format

`mjoin k= [f=] [K=] [-n] [-N] m=|` i= [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`k=`	Specify the key field name from input data field and at the `K=` parameter. Join records with the same field(s) from reference data. Set to NULL value when key fields do not match with the fields from reference file specified at `K=`.
`f=`	Field name of the reference file to join. When this parameter is not set, all fields except the key field will be joined.
`m=`	Specify list of reference file(s). Read from standard input when this parameter is not set (used when input data is specified at `i=`).
`K=`	List of field names from reference data for matching. Specify key field(s) from reference data to join with records with the key field from the input data defined at the `k=` parameter. Set to NULL value when key fields do not match with the fields from input file specified at `k=`. The key field(s) name from the reference file does not need to be the same as the `k=` parameter.
`-n`	Output NULL values when reference data does not consist of input data.
`-N`	Output NULL values when input data does not consist of reference data.

Examples

Example 1: Basic Example

Join the field `cost` from the reference file for records where the values of the `item` column from the input file is the same as the values in `item` column in the reference file.

```
$ more dat1.csv
item,date,price
A,20081201,100
A,20081213,98
B,20081002,400
B,20081209,450
C,20081201,100
$ more ref1.csv
item,cost
A,50
B,300
E,200
$ mjoin k=item f=cost m=ref1.csv i=dat1.csv o=rsl1.csv
#END# kgjoin f=cost i=dat1.csv k=item m=ref1.csv o=rsl1.csv
$ more rsl1.csv
item%0,date,price,cost
A,20081201,100,50
A,20081213,98,50
B,20081002,400,300
B,20081209,450,300
```

Example 2: Output unmatched data

Join the `cost` field for records with common key values in the `item` field from the input file and reference file, join `cost` item. At the same time, join all keys from the reference file if the value in the reference file is not in input data range, and set as NULL values.

```
$ mjoin k=item f=cost m=ref1.csv -n -N i=dat1.csv o=rsl2.csv
#END# kgjoin -N -n f=cost i=dat1.csv k=item m=ref1.csv o=rsl2.csv
$ more rsl2.csv
item%0,date,price,cost
A,20081201,100,50
A,20081213,98,50
B,20081002,400,300
B,20081209,450,300
C,20081201,100,
E,,,200
```

Related Command

mnjoin : Use `mnjoin` to duplicate key from reference file. mpaste : Join according to row number.

mcommon : Use `mcommon` to select records instead of joining.

4.30 mkeybreak Keybreak Point

Add an indicator to first rows and last rows of each key specified at `k=` parameter. The first row is indicated as `1` in the `top` field, and the same indicator is added to last row in the `bot` column. Records that are not in the first row nor last row will appear as NULL values.

Format

`mkeybreak k= [s=] [a=]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameter

`k=`	Specify field names (multiple fields can be specified) of the aggregate key.
`s=`	After sorted by specified fields (multiple fields can be specified), the first and last rows are marked.
`a=`	Specify the field name for top and bottom indicators in the output. [default value: top,bot]

Examples

Example 1: Basic Example

Sort the records by ￥verb—k1— field, add an indicator (`1`) to the first record (`top` field) and last record (`bottom` field) where `k1` is key field.

```
$ more dat1.csv
id,k1,k2,val
1,A,a,1
2,A,b,2
3,A,b,3
4,B,a,4
5,B,a,5
$ mkeybreak k=k1 i=dat1.csv o=rsl1.csv
#END# kgkeybreak i=dat1.csv k=k1 o=rsl1.csv
$ more rsl1.csv
id,k1%0,k2,val,top,bot
1,A,a,1,1,
2,A,b,2,,
3,A,b,3,,1
4,B,a,4,1,
5,B,a,5,,1
```

Example 2: 2 key fields

After fields `k1` and `k2` are sorted, the beginning of key field `k1` (`top`field) and end (`bottom`field) is marked (`1`).

```
$ mkeybreak s=k1,k2 k=k1 i=dat1.csv o=rsl2.csv
#END# kgkeybreak i=dat1.csv k=k1 o=rsl2.csv s=k1,k2
$ more rsl2.csv
id,k1,k2,val,top,bot
1,A,a,1,1,
2,A,b,2,,
3,A,b,3,,1
4,B,a,4,1,
5,B,a,5,,1
```

Related Command

4.31 mmbucket - Multi-dimensional Uniform Bucket Partition

Segment numerical values in multiple fields specified at `f=` into equal sized buckets. For example, set `f=a,b,c` and `n=5`, mbucket command segments fields `a,b,c` into 5 equal buckets. Whereas mmbucket allocate items `a,b,c` into 3 dimensional space for each bucket (bucket number becomes $5^3 = 125$) such that each interval is distributed as evenly as possible.

Format

`mmbucket f= n= [F=] [k=] [O=] [-ms] [-r]` [i=] [o=] [bufcount=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Values in this field(s) (multiple fields can be specified) is(are) partitioned. When there are multiple fields, the number of dimensions is determined based on equal number of buckets. When 1 field is specified, the result is the same as using `mbucket`. `-x,-nfn` options can be used to specify field number (0∼).
`n=`	Each bucket size corresponds to a field specified at `f=`. The number of items defined here is the same as the number of fields specified at `f=`. However, if there is only 1 number specified, the same partition number is applied to other field.
`F=`	Ouput format [default value: 1] Output format of bucket name. 0: Display bucket number. 1: Display value range of buckets. 2: Display both bucket number and value range.
`k=`	Unique key field(s) (multiple fields can be specified) to retrieve rows of data for partitioning into buckets.
`O=`	Specifies the output file with numeric range of each bucket from the items specified in the `f=` parameter.
`-ms`	When partitioning buckets for each field sequentially, the first item is changed multiple times during the bucket partition trial. For further information, refer to the "Introduction to Algorithms" to find out the best possible solution.
`-r`	Display bucket number in reverse order.

Examples

Example 1: Basic Example

Partition the number of records in column `x,y` into two multi-dimentional equal subsets. At the same time, save the numeric range of each bucket in the file named `rng.csv`.

```
$ more dat1.csv
id,x,y
A,2,7
B,6,7
C,5,6
D,7,5
E,6,4
F,1,3
G,3,3
H,4,2
I,7,2
J,2,1
$ mmbucket f=x:xb,y:yb n=2,2 O=rng.csv i=dat1.csv o=rsl1.csv
calculating on dimension ... #0 #1 done. VAR=30 updated!
calculating on dimension ... #0 #1 done. VAR=28 updated!
calculating on dimension ... #0 #1 done. VAR=28
#END# kgmbucket O=rng.csv f=x:xb,y:yb i=dat1.csv n=2,2 o=rsl1.csv
$ more rsl1.csv
id,x,y,xb,yb
A,2,7,1,2
B,6,7,2,2
C,5,6,2,2
```

```
D,7,5,2,2
E,6,4,2,1
F,1,3,1,1
G,3,3,1,1
H,4,2,2,1
I,7,2,2,1
J,2,1,1,1
$ more rng.csv
fieldName,bucketNo,rangeFrom,rangeTo
x,1,,3.5
x,2,3.5,
y,1,,4.5
y,2,4.5,
```

Example 2: Outupt Format

Partition the number of records in column `x,y` into two multi-dimentional equal subsets based on the `id` field. The output format shall display the both the bucket number and numeric range.

```
$ more dat2.csv
id,x,y
A,2,7
A,6,7
A,5,6
B,7,5
B,6,4
B,1,3
C,3,3
C,4,2
C,7,2
C,2,1
$ mmbucket k=id f=x:xb,y:yb n=2,2 F=2 i=dat2.csv o=rsl2.csv
calculating on dimension ... #0 #1 done. VAR=3 updated!
calculating on dimension ... #0 #1 done. VAR=3
calculating on dimension ... #0 #1 done. VAR=3 updated!
calculating on dimension ... #0 #1 done. VAR=3
calculating on dimension ... #0 #1 done. VAR=6 updated!
calculating on dimension ... #0 #1 done. VAR=6
#END# kgmbucket F=2 f=x:xb,y:yb i=dat2.csv k=id n=2,2 o=rsl2.csv
$ more rsl2.csv
id%0,x,y,xb,yb
A,2,7,1:_3.5,2:6.5_
A,6,7,2:3.5_,2:6.5_
A,5,6,2:3.5_,1:_6.5
B,7,5,2:3.5_,2:4.5_
B,6,4,2:3.5_,1:_4.5
B,1,3,1:_3.5,1:_4.5
C,3,3,1:_3.5,2:1.5_
C,4,2,2:3.5_,2:1.5_
C,7,2,2:3.5_,2:1.5_
C,2,1,1:_3.5,1:_1.5
```

Overview of algorithm[1]

Assuming data set D consists of two columns A, B with numerical values. The numeric range in column A and B is divided into K bits and L bits respectively. `mmbucket` determines how to divide the data equally into segments (buckets). Dispersion is used as a basis to measure the uniformity of distribution. The variance value minimizes the number of buckets partitions based on the heuristics described below, which is not guaranteed to be optimal. The algorithm is described as follows.

1. Partition data in field A by one dimensional uniform partition.
2. Determine the partition of A from step 1, partition the range of numerical values in B into buckets. The evaluation criteria for two-dimensional bucket partition is based on the partitions. Dynamic programming and one-dimensional bucket partition is used to divide buckets in to equal sized partitions.

[1]This algorithm is developed by Professor Naoki Kato (Kyoto University, Graduate School of Engineering).

3. Next, using the partitions of B obtained from step 2, partition the range of numerical values in A. As in step 2, the values of two dimensional bucket partition is used as an evaluation criteria.

4. Repeat step 2 and 3 recursively until improvement of variance is minute.

5. Return partition output.

During implementation, when A and B 's parts are switched in step 1, the solution returns the one with more optimal results. The same algorithm applies to uniform buckets partition for three or more dimensions.

Comparison of mbucket and mmbucket

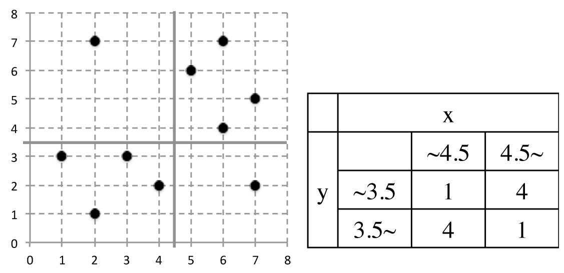
The following explains how multi-dimensional bucket partition operates compared to one-dimensional bucket partition. Table 4.4 shows two columns x and y, with 10 rows data of id from A to J.

Using this data, each of x and y is divided into two partitions, thus create a total of four buckets as shown in Figure . The results of one-dimensional bucket partition on x and y is shown in column x1, y1. In addition, the results of multi-dimensional bucket partition is shown in column x2 and y2 as displayed in Figure .

Variance (sum of squares of each bucket : for more information, refer to formulation of mbucket) of one dimensional bucket partition is computed as $Var'_a = 1^2 + 4^2 + 4^2 + 1^2 = 34$, and $Var'_b = 1^2 + 3^2 + 3^2 + 3^2 = 28$ for two-dimensional bucket partition. During one-dimensional bucket partition x and y is treated independently to optimize solution with minimal variance (i.e. each is divided into 5). The solution of one dimensional bucket partition is inferior to two-dimensional bucket partition. One dimensional bucket partition can be improved by using the multi-dimensional bucket partition. However, multi-dimensional partition requires a lot of CPU time dependent on the contents of the data.

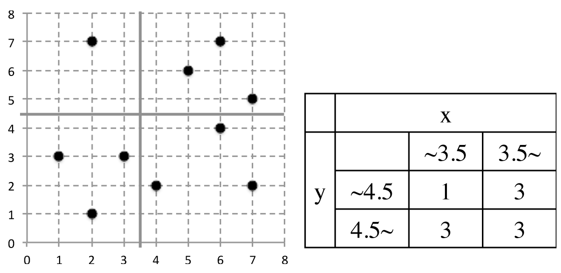
Table 4.4: Sample result of output from mbucket,mmbucket

Data			mbucket		mmbucket	
id	x	y	x1	y1	x2	y2
A	2	7	1	2	1	2
B	6	7	2	2	2	2
C	5	6	2	2	2	2
D	7	5	2	2	2	2
E	6	4	2	2	2	1
F	1	3	1	1	1	1
G	3	3	1	1	1	1
H	4	2	1	1	2	1
I	7	2	2	1	2	1
J	2	1	1	1	1	1



		x	
		~4.5	4.5~
y	~3.5	1	4
	3.5~	4	1

Figure 4.1: One-dimensional partition (mbucket) $\times$ 2



		x	
		~3.5	3.5~
y	~4.5	1	3
	4.5~	3	3

Figure 4.2: Two-dimensional partition (mmbuket)

Precision experiment using on variety of fields

The following compares the precision of mmbucket and mbucket using real data collected by Tropical Atmosphere Ocean (TAO) Project. The project is designed for the study of year-to-year climate variations related to El Nino the Southern Oscillation (ENSO), and provides provides in-situ data collection of high quality oceanographic and surface meteorological data for monitoring, forecasting, and understanding of climate swings associated with El Nino and La Nina. Snapshot of the data is shown in the following table.

Table 4.5: Data set used for precision experiment

id	year	month	day	date	latitude	longitude	zonwinds	merwinds	humidity	air_temp.	sstemp
4060	93	5	9	930509	-0.02	-109.96	-2.1	2.1	81.2	26.8	27.02
4061	93	5	10	930510	-0.02	-109.96	-3.4	1.4	84.2	26.95	26.91
4062	93	5	11	930511	-0.02	-109.96	-3.8	2.2	84.9	26.98	26.78
4063	93	5	12	930512	-0.02	-109.96	-3	1.5	86.9	26.93	26.74
4064	93	5	13	930513	-0.02	-109.96	-4.5	1.9	87.6	27.01	26.82
4065	93	5	14	930514	-0.02	-109.96	-5	1.3	85.6	26.96	26.68

The precision experiment uses attributes with numeric values (7 attributes from latitude to sea surface temperature) for bucket partition. The number of record rows is 93,935. Statistics for each attribute is shown in Table 3. "Types of data values" significantly affects computation time of bucket partition.

Table 4.6: Various statistics of numerical data

Measurement	Latitude	Longitude	Zonal wind	North-south wind	Humidity	Temp	Sea surface temp
Type	482	924	228	206	385	1104	1201
Arithmetic average	0.305	-70.8	-3.35	-0.046	81.3	27.1	27.9
Standard deviation	4.77	128.7	3.42	3.021	5.28	1.674	1.87
Minimum	-8.33	-180	-10.7	-10.6	52.1	17.5	18.2
Mean	0.01	-125	-4.1	-0.1	81.3	27.5	28.4
Maximum	9.05	170.0	14.3	13	99.9	31.5	31.0

The comparison of variance for one-dimensional bucket partition (mbucket) and two-dimensional bucket partition (mmbucket) based on a total of 21 combinations from the 7 attributes are distributed into two columns as shown in Table 4.7.

For example, the first row shows the comparison results of bucket partition on " temperature $\times$ humidity ", the variance of mbucket is 408274409 and the variance of mmbucket is 406438211, resulting in the ratio of (mmbucket / mbucket) 0.996. The ratio is shown for partitions of 10 $\times$ 10,15 $\times$ 15 and 20 $\times$ 20. The results differs according to the combination of attributes, there are minimal improvements in some combinations such as "humidity $\times$ zonal wind speed", improvements can be seen at " latitude x longtitude (15 x 15) with an improvement in accuracy of 19% (1-0.81).

Table 4.7: Comparison of two-dimensional partition accuracy of mbucket and mmbucket

Item 1	Item 2	Variance (5 × 5 partition)		Comparison of variance (mmbucket/mbucket)			
		mbucket	mmbucket	5 × 5	10 × 10	15 × 15	20 × 20
Temperature	Humidity	408274409	406438211	0.996	0.987	0.988	0.981
	Latitude	374125463	371258775	0.992	0.988	0.982	0.964
	Longitude	490727955	454663065	0.927	0.949	0.946	0.929
	North-south wind speed	396436813	394724219	0.996	0.991	0.993	0.989
	Sea surface temperature	816199131	747215787	0.915	0.897	0.883	0.862
	Zonal wind speed	382069455	381225143	0.998	0.998	0.997	0.996
Temperature	Humidity	368492959	367941709	0.999	0.994	0.994	0.990
	Latitude	372116309	370591351	0.996	0.991	0.993	0.986
	North-south wind speed	355658757	355658757	1.000	1.000	1.000	1.000
	Sea surface temperature	380382203	379546293	0.998	0.991	0.991	0.983
	Zonal wind speed	357729819	357697283	1.000	1.000	1.000	1.000
Latitude	Longtitude	365459223	361754113	0.990	0.923	0.812	0.816
	North-south wind speed	371478669	370970251	0.999	0.988	0.985	0.982
	Sea surface temperature	392810425	389589403	0.992	0.987	0.979	0.952
	Zonal wind speed	364521077	364406663	1.000	0.999	0.991	0.992
Longtitude	Latitude	414154185	408431667	0.986	0.976	0.976	0.962
	Sea surface temperature	510979465	463576537	0.907	0.945	0.939	0.934
	Zonal wind speed	400021527	392710641	0.982	0.983	0.982	0.973
North-south wind	Sea surface temperature	393233841	392432943	0.998	0.995	0.994	0.993
	Zonal wind speed	359290669	359074015	0.999	1.000	1.000	0.997
Sea surface temp	Zonal wind speed	442877539	438326669	0.990	0.988	0.984	0.986

Speed Comparison

Next, the experiments are carried out to compare the differences in speed by against different number of partitions. The execution time is computed using the same data “ temperature × sea surface temperature ” and “ latitude × longitude ” with 5 to 40 partitions at increments of 5. The results are shown in Table 4.8. The execution time of `mbucket` is consistent regardless of the number of partitions. However, more time is required to compute the number of partitions for two dimensional partitions. This is due to the fact that the algorithm is not efficient in selecting multidimensional orthogonal numeric range required by computation. The partition speed will be noted as possible improvements in the next version.

Table 4.8: Sample results of mbucket,mmbucket

	Temperature × Sea Surface Temperature		Latitude × Longtitude	
Number of buckets	mbucket	mmbucket	mbucket	mmbucket
5	0.221	2.21	0.216	0.67
10	0.227	3.90	0.216	1.67
15	0.233	10.4	0.230	3.28
20	0.231	26.1	0.228	7.13
25	0.232	32.9	0.237	13.3
30	0.236	46.7	0.236	11.4
35	0.237	62.3	0.240	15.2
40	0.237	80.1	0.237	25.8

Related Command

mbucket : This command processes one-dimensional bucket partition for each field even when more than one field is specified.

4.32 mminput Display form input screen

Note: This command is a beta. Its specifications may be changed.

This command displays a data input screen using the text file specified by the `i=` parameter as the screen form. The character strings on the screen form is displayed as is, and the area between square brackets (`[]`) is shown as a freehand input field. There can be two or more input fields. The data entered by the user is output to the CSV file specified by the `o=` parameter. The output data consists of one row. When there are two or more input fields, a multiple-field CSV file is output. When the command ends with nothing entered in the input frame, null is output. To output a fieldname, use the `f=` parameter. With `f=` omitted, no fieldname header is output. The operation is indefinite if the specified coordinate is outside the scope of the terminal.

Format

`mminput i= [f=]` o= [-nfn] [-nfno] [-x] `[--help] [--helpl] [--version]`

Parameters

`i=`	Specify the text file containing the screen form.
`f=`	Specify the output fieldname.

Examples

Example 1: Basic example

A `name` and `address` input screen is displayed. The entered name and address are output to `rsl1.csv` under fieldnames name,address.

```
$ more screen.txt

     name   :[               ]
     address:[               ]

$ mminput i=screen.txt f=name,address o=rsl1.csv
$ more rsl1.csv
name,address
Taro,Japan


The following will be displayed:
+---------------------------------------
|
|     name   :[Taro           ]
|     address:[Japan          ]
|
```

Example 2: Judging end status

The parameters are the same as in Example 1. This script judges the end status and performs different operations accordingly.

```
$ more scp.sh
rm -f rsl3.csv
clear
mminput i=screen.txt f=name,address o=rsl3.csv
if [ $? = 0 ] ; then
  clear ; echo "end by enter key"
else
  clear ; echo "end by escape key"
fi
```

```
# Result of typing Taro and Japan and pressing Enter
$ bash scp.sh
end by enter key
$ more rsl3.csv
name,address
Taro,Japan

# Result of typing Taro and Japan and pressing ESC
$ bash scp.sh
end by escape key
$ more rsl3.csv
name,address
Taro,Japan
```

Related Commands

minput : Displays the input screen. mdsp : Displays a character string at the specified position on the screen. mseldsp : Displays a single-choice input window on the screen. mmseldsp : Displays a multiple-choice input window on the screen.

4.33 mmseldsp Multiple display option selection screen

Note: This command is a beta. Its specifications may be changed. This command displays a list of character strings to choose from at the position on the terminal specified by coordinate `x=,y=`. The list is specified by the `i=` or `seldata=` parameter. Once the user has selected a character string, the command outputs it to the file specified by the `o=` parameter. While the mseldsp command allows the user to choose only one option from the list, the `mmseldsp` command allows multiple choice. When multiple character strings are selected, they are output as multiple rows in a CSV file. When the command ends with nothing entered in the input frame, null is output (that is, only a line feed is output). To output a fieldname, use the `f=` parameter. With `f=` omitted, no fieldname header is output. If there are too many choices to fit into the screen, use the `height=` parameter to specify the number of rows shown in a scroll window. When the Enter key is pressed on the selection screen, the command returns end status 0 and ends. When the ESC key is pressed on the screen, the command returns end status 1 and ends. In either case, the choice made on the selection screen is output to a file. In the coordinate system, the top left is `x=1,y=1` (escape sequence specifications). If the value specified for `x=` or `y=` is smaller than 1, the command assumes 1 and operates. The operation is indefinite if the specified coordinate is outside the scope of the terminal.

Format

```
mmseldsp x= y= [height=] i=|seldata= o= [-nfn] [-nfno] [-x] [--help] [--helpl] [--version]
```

Parameters

`x=`	Specify the display start position (1 or greater) on the x axis (horizontal, left to right)
`y=`	Specify the display start position (1 or greater) on the y axis (vertical, top to bottom)
`height=`	Specify the number of rows in which the choices are shown.
`i=`	Specify the name of the CSV file containing the choice character strings as fields.
`f=`	Specify the name of the CSV file containing the choice character strings as fields.
`seldata=`	Specify the list of character strings to choose from, using commas ad delimiters.

Examples

Example 1: Basic example

The contents of `sel.txt` are displayed at x=10,y=2 on the terminal, and the character string selected by the user is output to `rsl1.txt`.

```
$ more sel.txt
apple
pineapple
grape
orange
$ mmseldsp x=10 y=2 i=sel.txt o=rsl1.txt
# Assume that the user has selected the first row.
$ mose rsl1.txt
apple
orange

The following will be displayed:
+----------------------------------------
|
|           apple
|           pineapple
|           grape
|           orange
|
```

Example 2: Using arguments to specify the character strings

This script works in the same way as Example 1, but the option character strings are specified by `seldata=`.

```
$ mmseldsp x=10 y=2 seldata=apple,pineapple,grape,orange o=rsl2.txt
# Assume that the user has selected the second row.
$ mose rsl2.txt
apple
grape

The following will be displayed:
+---------------------------------------
|
|
|          apple
|          pineapple
|          grape
|          orange
|
```

Example 3: Judging end status

The parameters are the same as in Example 2. This script judges the end status and performs different operations accordingly.

```
$ more scp.sh
rm -f rsl3.csv
clear
mmseldsp x=10 y=2 seldata=apple,pineapple,grape,orange o=rsl3.csv
if [ $? = 0 ] ; then
  clear ; echo "end by enter key"
else
  clear ; echo "end by escape key"
fi

# Result of selecting apple and pressing Enter
$ bash scp.sh
end by enter key
$ more rsl3.csv
apple

# Result of selecting apple and pressing ESC
$ bash scp.sh
end by escape key
$ more rsl3.csv
apple
```

Related Commands

minput :Displays the input screen. mminput : Displays the input screen with multiple input frames. mdsp : Displays a character string at a specified position on the screen. mseldsp : Displays a single-choice input window on the screen.

4.34 mmvavg - Calculate Moving Average

Calculate the moving average. The three different ways to calculate moving average include simple moving average (SMA), weighted moving average (WMA), and exponential moving average (EMA).

The value of t time is expressed by x_t, and period is represented by m as defined in several formulas of moving average (4.1,4.2,4.3).

$$SMA_t = \frac{1}{m}\sum_{i=0}^{m-1} x_{t-i} \tag{4.1}$$

$$WMA_t = \sum_{i=0}^{m-1} \frac{m-i}{S} x_{t-i}, \quad S = \sum_{i=1}^{m} i \tag{4.2}$$

$$EMA_t = \alpha x_t + (1-\alpha)EMA_{t-1} \tag{4.3}$$

Format

`mmvavg [s=] [k=] [n=] f= [t=] [-exp|-w] [alpha=] [skip=]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

`s=`	After the specified field is sorted (multiple fields can be specified), moving average is calculated. `s=` parameter is required when `-q` option is not specified.
`k=`	Aggregate records using the specified field name(s) (multiple fields can as unit of calculation.
`f=`	Compute the moving averages of the field(s) (multiple fields can be specified).
`t=`	Interval numbers of integers greater than 1. When `-exp` is used with `alpha=`, the `t=` parameter do not need to be defined.
`-w`	Linear weighted moving average.
`-exp`	Exponential smoothing moving average.
`alpha=`	Use a real number as smoothing coefficient when `-exp` is specified. The default value of alpha is `alpha=2/(value of = t+1)`。
`skip=`	Specify the number of rows to hide from the top in the output. Default value: `skip=(value of t= -1)`, `skip=0` when `-exp` is specified.

Examples

Example 1: Basic Example

The first row is not printed as there is less than the number of required intervals for computation.

```
$ more dat1.csv
id,value
1,5
2,1
3,3
4,4
5,4
6,6
7,1
8,4
9,7
$ mmvavg s=id f=value t=2 i=dat1.csv o=rsl1.csv
```

```
#END# kgmvavg f=value i=dat1.csv o=rsl1.csv s=id t=2
$ more rsl1.csv
id%0,value
2,3
3,2
4,3.5
5,4
6,5
7,3.5
8,2.5
9,5.5
```

Example 2: Basic Example 2

The first row is not printed as there is less than the number of required intervals for computation.

```
$ mmvavg s=id f=value t=2 -w i=dat1.csv o=rsl2.csv
#END# kgmvavg -w f=value i=dat1.csv o=rsl2.csv s=id t=2
$ more rsl2.csv
id%0,value
2,2.333333333
3,2.333333333
4,3.666666667
5,4
6,5.333333333
7,2.666666667
8,3
9,6
```

Example 3: Basic Example 3

Exponential smoothing moving average (`-exp`) includes the first row in the output.

```
$ mmvavg s=id f=value t=2 -exp i=dat1.csv o=rsl3.csv
#END# kgmvavg -exp f=value i=dat1.csv o=rsl3.csv s=id t=2
$ more rsl3.csv
id%0,value
1,5
2,2.333333333
3,2.777777778
4,3.592592593
5,3.864197531
6,5.288065844
7,2.429355281
8,3.47645176
9,5.82548392
```

Example 4: An example of assigning key

```
$ more dat2.csv
id,key,value
1,a,5
2,a,1
3,a,3
4,a,4
5,a,4
6,b,6
7,b,1
8,b,4
9,b,7
$ mmvavg s=key,id k=key f=value t=2 i=dat2.csv o=rsl4.csv
#END# kgmvavg f=value i=dat2.csv k=key o=rsl4.csv s=key,id t=2
$ more rsl4.csv
id,key,value
2,a,3
3,a,2
4,a,3.5
5,a,4
7,b,3.5
```

```
8,b,2.5
9,b,5.5
```

Example 5: Display all records including those that are less than the defined intervals

```
$ more dat3.csv
key,value
a,1
a,2
a,3
a,4
a,5
b,6
b,1
b,4
b,7
$ mmvavg -q k=key f=value t=2 skip=0 i=dat3.csv o=rsl5.csv
#END# kgmvavg -q f=value i=dat3.csv k=key o=rsl5.csv skip=0 t=2
$ more rsl5.csv
key,value
a,1
a,1.5
a,2.5
a,3.5
a,4.5
b,6
b,3.5
b,2.5
b,5.5
```

Related Commands

mmvstats : Specify the average as well as various types of statistics.

mmvsim : Compute bivariate statistics.

mwindow : Computes statistics on sliding window data which cannot be computed using `mmvstats`.

4.35 mmvsim - Sliding Windows Similarity Meausre

Compute the similarity measure (bivariate statistics) using settings of sliding window. This is the sliding windows version of the msim command. The main difference between the two is that `msim` can only carry out similarity calculation for 1 target object, whereas `mmvsim` quantifies similarity between two objects.

Format

`mmvsim [s=] [k=] f= c= a= [t=] [skip=] -n` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

`s=`	After the specified field is sorted (multiple fields can be specified), a variety of similarity computation is carried out. `s=` parameter is required if `-q` is not specified.
`k=`	Aggregate records using the specified field name(s) (multiple fields can be specified).
`f=`	Field name(s) (multiple fields can be specified) for computation .
`t=`	Integer intervals that is greater than 1.
`c=`	Define measures of similarity (one from the following). `covar|ucovar|pearson|spearman|kendall|euclid|` `cosine|cityblock|hamming|chi|phi|jaccard|support|lift` Refer to msim command for detailed definition.
`skip=`	Specify the number of rows to hide from the top in the output [default value:`skip=(value of t= -1)`]

Examples

Example 1: Basic Example

Calculate the Pearson product-moment correlation coefficient for 3 window intervals for fields `x,y`.

```
$ more dat1.csv
t,x,y
1,14,0.17
2,11,0.2
3,32,0.15
4,13,0.33
5,8,0.1
6,19,0.56
$ mmvsim s=t t=3 c=pearson f=x,y a=sim i=dat1.csv o=rsl1.csv
#END# kgmvsim a=sim c=pearson f=x,y i=dat1.csv o=rsl1.csv s=t t=3
$ more rsl1.csv
t%0,x,y,sim
3,32,0.15,-0.8746392857
4,13,0.33,-0.6515529194
5,8,0.1,-0.1164257338
6,19,0.56,0.9986254289
```

Related Commands

msim : Find out the degree of similarity without setting sliding window.

mwindow : Create sliding window data for use with `mmvstats` for the computation of statistics.

mmvavg : This command only computes moving average.

4.36 mmvstats - Compute Statistics of Sliding Window

Calculate various statistics (1 variable) for sliding windows. This a variant of the mstats command with functionality to compute sliding windows.

Format

`mmvstats [s=] [k=] f= [t=] c= [skip=] -n` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

`s=`	After the specified field is sorted (multiple fields can be specified), various statistics is computed. `s=` parameter is required when `-q` is not specified.
`k=`	Aggregate records using the specified field name(s) (multiple fields can be specified).
`f=`	Field name(s) (multiple fields can be specified) for computation .
`t=`	Integer interval greater than 1.

`c=`	Type of statistics (specify one of the following list)
	`sum|mean|devsq|var|uvar|sd|usd|cv|min|`
	`|max|range|skew|uskew|kurt|ukurt`
	Refer to mstats for detailed definitions.

`skip=`	Specify the number of rows to hide from the top in the output.

Examples

Example 1: Basic Example

Calculate sum of sliding window. The first row is not printed as there is less than the required nubmer of intervals for computation.

```
$ more dat1.csv
id,value
1,5
2,1
3,3
4,4
5,4
6,6
7,1
8,4
9,7
$ mmvstats s=id f=value t=2 c=sum i=dat1.csv o=rsl1.csv
#END# kgmvstats c=sum f=value i=dat1.csv o=rsl1.csv s=id t=2
$ more rsl1.csv
id%0,value
2,6
3,4
4,7
5,8
6,10
7,7
8,5
9,11
```

Related Commands

mmvavg : Calculate moving average.

mwindow : Create sliding window data for use with `mmvstats` for the computation of statistics.

mmvsim : Compute the similarity measure (bivariate statistics) .

4.37 mnewnumber - Generate List of Sequential Numbers

Define the start value of the alphabetic sequence at the `S=` parameter, set the the interval of alphabet sequence at `I=` parameter, and define the column name of the sequence at `a=` parameter. The alphabet sequence uses 26 alphabetic characters in base-26 from A to Z (A,B,···,Z,AA,AB,···,AZ,BA,BB,···,ZZ,AAA,AAB,···).

Format

`mnewnumber a= [I=] [S=] [l=]` [o=] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`a=`	Specify the field name of the list of new serials. This parameter is not required when `-nfn,-nfno` option is specified.
`I=`	Interval between the sequence of numbers [default value: 1]
`S=`	Starting value/alphabet(upper case letters) [default value:1] Assign either alphabet or numbers as the starting value of the sequence. A list of serial numbers is generated when the starting numeric value is specified. An alphabet sequence is generated when the starting alphabet is specified (cannot be specified in lowercase).
`l=`	Number of rows to generate [default value:10]

Examples

Example 1: Basic Example

Generate a dataset with 5 sequential numbers starting from 1 incremented by 1. Name the sequence as `No.`.

```
$ mnewnumber a=No. I=1 S=1 l=5 o=rsl1.csv
#END# kgNewnumber I=1 S=1 a=No. l=5 o=rsl1.csv
$ more rsl1.csv
No.
1
2
3
4
5
```

Example 2: Change the starting number and interval

Generate a dataset consisting of 5 sequential numbers starting from 10 with an incremental interval of 5. Name the sequence as `No.`.

```
$ mnewnumber a=No. I=5 S=10 l=5 o=rsl2.csv
#END# kgNewnumber I=5 S=10 a=No. l=5 o=rsl2.csv
$ more rsl2.csv
No.
10
15
20
25
30
```

Example 3: Generate series of alphabet

Generate a dataset consisting of 5 alphabet sequence starting from A with 1 alphabet in between. Name the sequence as `No.`.

```
$ mnewnumber a=No. I=1 S=A l=5 o=rsl3.csv
#END# kgNewnumber I=1 S=A a=No. l=5 o=rsl3.csv
$ more rsl3.csv
No.
A
B
C
D
E
```

Example 4: Generate data without header

Generate a dataset consisting of 11 alphabet sequence starting from B with 3 alphabets in between. Exclude the header from the output.

```
$ mnewnumber  -nfn  I=3 l=11 S=B o=rsl4.csv
#END# kgNewnumber -nfn I=3 S=B l=11 o=rsl4.csv
$ more rsl4.csv
B
E
H
K
N
Q
T
W
Z
AC
AF
```

Related Commands

mnewrand : Generate a dataset with random numbers.

mnewstr : Generate fixed character strings.

4.38 mnewrand - Generate Dataset with Random Numbers

Generate real random numbers from the range of 0.0 to 1.0. Use the **-int** option to generate randomized sequences of integers.

This command uses Mersenne Twister to generate random numbers. (Webpage of author , boost library)。

Format

`mnewrand a= [max=] [min=] [S=] [l=] [-int]` [o=] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`a=`	Name of column in the new dataset created. This parameter is not required when **-nfn,-nfno** options are specified.
`max=`	Maximum value of random number [default value: INT_MAX] **-int** must be specified with this parameter.
`min=`	Minimum value of random number [default value: 0] **-int** must be specified with this parameter.
`S=`	Random seed [default value: current time]
`l=`	Number of lines [default value: 10] Generate a new dataset with random numbers according to the number of rows specified.
`-int`	Generate random integers.

Examples

Example 1: Basic Example

Generate 10 rows of random integers. Use a fixed random seed so that it will always return the same sequence of random numbers.

```
$ mnewrand a=rand S=1 o=rsl1.csv
#END# kgnewrand S=1 a=rand o=rsl1.csv
$ more rsl1.csv
rand
0.4170219984
0.9971848081
0.7203244893
0.9325573612
0.0001143810805
0.1281244478
0.3023325677
0.9990405154
0.1467558926
0.2360889763
```

Example 2: Random Integers

Use random seed 1 to generate 5 rows of random integers with minimum value of 10 and maximum value of 100.

```
$ mnewrand a=rand -int max=1000 min=0 l=5 S=1 o=rsl2.csv
#END# kgnewrand -int S=1 a=rand l=5 max=1000 min=0 o=rsl2.csv
$ more rsl2.csv
rand
417
998
721
933
0
```

Example 3: Generate Output without Header

Specify `-nfn` option to generate random number data without header.

```
$ mnewrand -nfn l=5 S=1 o=rsl3.csv
#END# kgnewrand -nfn S=1 l=5 o=rsl3.csv
$ more rsl3.csv
0.4170219984
0.9971848081
0.7203244893
0.9325573612
0.0001143810805
```

Related Commands

mnewnumber : Generate list of sequential numbers.

mnewstr : Generate a list of character strings.

4.39 mnewstr - Generate Fixed String Data

Define the character string to generate at `v=` parameter, and pass the new column name at `a=` parameter. Multiple columns can be generated at a time.

Format

`mnewstr a= [v=] [l=]` [o=] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`a=`	Field name(s) of the new data. Define multiple field names separated with a comma in between the field names. This argument is not required when `-nfn,-nfno` options are specified.
`v=`	Specify the new character string to generate. Define multiple field separated with a comma in between the values. The number of fields must be the same as the number of field names defined at `a=`.
`l=`	Number of rows of random data to generate [default value:10].

Examples

Example 1: Basic Example

Generate a new dataset with characters strings `custNo` and `A0001` printed in 5 rows, and name the fields as `attribute` and `code` respectively.

```
$ mnewstr a=attribute,code v=custNo,A0001 l=5 o=rsl1.csv
#END# kgnewstr a=attribute,code l=5 o=rsl1.csv v=custNo,A0001
$ more rsl1.csv
attribute,code
custNo,A0001
custNo,A0001
custNo,A0001
custNo,A0001
custNo,A0001
```

Related Commands

mnewnumber : Generate list of sequential numbers in a new dataset.

mnewrand : Generate random numbers in a new dataset.

4.40 mnjoin - Natural Join with Reference File

A natural join selects rows from input data and reference file that have equal values in columns defined at `k=` parameter, the reference file is specified at the `m=` parameter, fields in the reference file specified at `f=` parameter is added through natural join. The difference with `mjoin` command is that key field(s) in the reference field is not unique.

Given that the input data has n records with a certain key value, and the reference file has m records with the same key value, $n \times m$ records will be generated. In addition, if `f=` is not defined, all fields except the key field will be added.

Format

`mnjoin k= [f=] [K=] [-n] [-N] m=|` i= [o=] [bufcount=] [-nfn] [-nfno] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`k=`	key field name(s) from the input data for matching This key field is specified in the input data and at the `K=` parameter. Join rows when this fields matches with the fields from the reference data.
`f=`	Specify the field name(s) to join from the reference file. When this parameter is not defined, all fields except the key field will be joined.
`m=`	Reference file name. Read from standard input if this parameter is not set. (when `i=` is specified)
`K=`	Key field name(s) from the reference data for matching This key field(s) from reference data is compared with the key field(s) from input data specified at `k=` parameter, fields where records with common key are joined. This parameter is not required if the field name in reference file is the same as the one defined at the `k=`parameter.
`-n`	Output NULL values when reference data does not consist of input data.
`-N`	Output NULL values when input data does not consist of reference data.

Examples

Example 1: Basic Example

The `item` field in the input file is compared with the `item` field from the reference file, add `cost` field for records with the same value. There are two records where `item=A` in both input file and reference file, therefore, 2×2=4 rows of `item=A` is written to the output file.

```
$ more dat1.csv
item,date,price
A,20081201,100
A,20081213,98
B,20081002,400
B,20081209,450
C,20081201,100
$ more ref1.csv
item,cost
A,50
A,70
B,300
E,200
$ mnjoin k=item f=cost m=ref1.csv i=dat1.csv o=rsl1.csv
#END# kgnjoin f=cost i=dat1.csv k=item m=ref1.csv o=rsl1.csv
$ more rsl1.csv
item%0,date,price,cost
A,20081201,100,50
A,20081201,100,70
A,20081213,98,50
A,20081213,98,70
```

```
B,20081002,400,300
B,20081209,450,300
```

Example 2: Ouput unmatched data

Use `-n` to print records in the input data that do not match with those in the reference file (row where `item="C"`), and use -N to print records in the reference file that do not match with those in the input file (row where `item="E"`).

```
$ more ref2.csv
item,cost
A,50
B,300
E,200
$ mnjoin k=item f=cost m=ref2.csv -n -N i=dat1.csv o=rsl2.csv
#END# kgnjoin -N -n f=cost i=dat1.csv k=item m=ref2.csv o=rsl2.csv
$ more rsl2.csv
item%0,date,price,cost
A,20081201,100,50
A,20081213,98,50
B,20081002,400,300
B,20081209,450,300
C,20081201,100,
E,,,200
```

Related Commands

mjoin : It is faster to use `mjoin` if the key in the reference file is unique.

mproduct : Join combination of all records without using key. Each row in the reference file is joined with all records in the input data.

4.41 mnormalize - Normalization

Specify the field at the `f=` parameter, and specify the normalization method at `c=` parameter.

Format

`mnormalize c= f= [k=]` [i=] [o=] [bufcount=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

`c=`	Specify the normalisation method listed as follows. `z` : z score : $z_i = (x_i - m)/u$ (x_i: $i$number of data, m :arithmetic mean, u :standard deviation) `Z` : deviation value : $Z_i = 50 + 10 \times z_i$ `range` : use linear conversion to transform minimum value 0 to maximum value 1 $r_i = (x_i - \min_x)/(\max_x - \min_x)$
`f=`	Specify the field to normalize here. Specify the new field name after :(colon). Example: `f=`quantity:quantityNorm
`k=`	Key field name(s) [aggregate key break processing] The key field specified is used as the unit for normalization.

Examples

Example 1: Basic Example

Normalize (z score) `quantity` and `amount` field based on each `customer`, label the column names of the output as `qtyNominal` and `amtNorminal` respectively.

```
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ mnormalize c=z k=customer f=quantity:qtyNominal,amount:amtNorminal i=dat1.csv o=rsl1.csv
#END# kgnormalize c=z f=quantity:qtyNominal,amount:amtNorminal i=dat1.csv k=customer o=rsl1.csv
$ more rsl1.csv
customer%0,quantity,amount,qtyNominal,amtNorminal
A,1,10,-0.7071067812,-0.7071067812
A,2,20,0.7071067812,0.7071067812
B,1,15,-0.5773502692,0
B,3,10,1.154700538,-1
B,1,20,-0.5773502692,1
```

Example 2: Deviation value

```
$ mnormalize c=Z k=customer f=quantity:qtyNominal,amount:amtNorminal i=dat1.csv o=rsl2.csv
#END# kgnormalize c=Z f=quantity:qtyNominal,amount:amtNorminal i=dat1.csv k=customer o=rsl2.csv
$ more rsl2.csv
customer%0,quantity,amount,qtyNominal,amtNorminal
A,1,10,42.92893219,42.92893219
A,2,20,57.07106781,57.07106781
B,1,15,44.22649731,50
B,3,10,61.54700538,40
B,1,20,44.22649731,60
```

Example 3: Linear transformation from 0 to 1

```
$ mnormalize c=range k=customer f=quantity:qtyNominal,amount:amtNorminal i=dat1.csv o=rsl3.csv
#END# kgnormalize c=range f=quantity:qtyNominal,amount:amtNorminal i=dat1.csv k=customer o=rsl3.csv
$ more rsl3.csv
customer%0,quantity,amount,qtyNominal,amtNorminal
A,1,10,0,0
A,2,20,1,1
B,1,15,0,0.5
B,3,10,1,0
B,1,20,0,1
```

Related Command

4.42 mnrcommon - Select Records within Specified Range(s) from Reference File

Select the record in the input file that matches the records within the defined range(s) defined from the reference file. `k=` parameter specifies the key field name from the input file to match with the key defined in `K=` from the reference file. The selection criteria is based on the data series from the input file defined in `r=` parameter for records that falls within the data range in the reference file defined in the `R=` parameter. Add `%n` after the item name if the field defined at `r=` parameter is a numerical value.

Format

`mnrcommon [k=] R= r= [K=] [u=] [-r] m=|` i= [o=] [-assert_diffSize] [-assert_nullkey] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`k=`	Key item(s) to match in the input file (Multiple keys can be specified). The key(s) specified will be matched with the key field(s) at `K=` parameter from the reference file.
`m=`	Specify name of reference file. Data is read from standard input if this parameter is not defined. (when i= is specified)
`R=`	Define the value range (start,end) in the reference file. If the first argument is NULL means less than, if the second argument is NULL means more than.
`r=`	Field name of input file for range comparison. [%n] Records in the input file that matches the key field specified in the `k=` parameter in the reference data is selected. when processing as numeric value, %n will be added to field name defined at `r=` parameter.
`K=`	Key field(s) in the reference data for matching (Multiple keys can be specified) The key specified will be matched with the key field defined in `k=` parameter from the input file. Records in the input file that matches the key field specified in the `k=` parameter in the reference data is selected.
`u=`	Write unmatched records to this output file.
`-r`	Reverse selection Select records that is not within the data range defined at `R=` parameter.

Sort Criteria

Fields specified at `r=,R=` must be sorted beforehand. However, the numerical values defined in `r=,R=` should be sorted in ascending order to join with the numerical range. Where `k=,K=` is specified, the strings defined at the parameter must be sorted in ascending order.

For example, when the parameter `k=key K=Key r=val%n R=range i=dat.csv m=ref.csv` is specified, `dat.csv` data, should be sorted with `msortf f=key,val%n` as the criteria, and `ref.csv` data, should be sorted with `msortf f=Key,range%n` as the criteria.

Examples

Example 1: Basic Example

Select records where the transaction date is `20080203` with transaction "Amount" greater than `5` and less than `15` or greater than `40` and less than `50`.

```
$ more dat1.csv
Date,Amount
20080123,10
20080203,10
20080203,20
20080203,45
20080410,50
$ more ref1.csv
```

```
Date,AmountF,AmountT
20080203,5,15
20080203,40,50
$ mnrcommon k=Date m=ref1.csv R=AmountF,AmountT r=Amount%n i=dat1.csv o=rsl1.csv
#END# kgnrcommon R=AmountF,AmountT i=dat1.csv k=Date m=ref1.csv o=rsl1.csv r=Amount%n
$ more rsl1.csv
Date%0,Amount
20080203,10
20080203,45
```

Example 2: Reverse selection

Add `-r` option to reverse selection criteria.

```
$ mnrcommon k=Date m=ref1.csv R=AmountF,AmountT r=Amount%n -r i=dat1.csv o=rsl2.csv
#END# kgnrcommon -r R=AmountF,AmountT i=dat1.csv k=Date m=ref1.csv o=rsl2.csv r=Amount%n
$ more rsl2.csv
Date%0,Amount
20080123,10
20080203,20
20080410,50
```

Related commands

mcommon : Select common records in reference file

mnrjoin : Natural join data from the reference file with multiple ranges.

4.43 mnrjoin - Natural Join within Multiple Ranges with Reference File

Join columns according to the range of values in the column from reference file. The field specified at `r=` parameter is matched with the the range of values defined as two arguments at the `R=` parameter in the reference file defined at the `m=` parameter. the field(s) specified at`f=` parameter are joined for records with the same value.

If there are more than one match for each record, natural join returns output for all rows. The range of values is compared as character strings by default. Attach %n after the field name at the `r=` parameter to process as numerical values.

Format

`mnrjoin R= r= [k=] [K=] [f=] [-n] [-N] m=|` i= [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	The field name(s) (multiple fields can be specified) to join from the reference file. When this is not defined, the all fields except the key specified at `K=` will be joined.
`m=`	Reference file name. Read from standard input if this parameter is not set (when `i=` is specified).
`R=`	Field names of the range (limit to 2 fields). Field names (start,end) of the range in reference file. If the first field is NULL, the range is any number less than the ending value of the range. If the second field is NULL, the range is any number greater than the starting value of the range.
`r=`	Compare the values in this field [%n] against the range. Field name in the input file. Add %n after the field name in the `r=` parameter to process as numerical values.
`k=`	Key field name(s) (multiple fields can be specified) from the input data for comparison Join records with same key fields in the input data `k=` and reference data `K=`.
`K=`	Key field name(s) (multiple fields can be specified) from the reference data for comparison This key field(s) from reference data is compared with the key field(s) from the input data specified at `k=` parameter, fields where records with common key are joined. This parameter is not required if the field name name is the same as the one defined at the `k=` parameter.
`-n`	Output NULL values when reference data does not consist of input data.
`-N`	Output NULL values when input data does not consist of reference data.

For example, given the parameters `k=key K=Key r=val%n R=range i=dat.csv m=ref.csv`, if the sort criteria for the input data `dat.csv` is carried out by `msortf f=key,val%n`, the sort criteria for `ref.csv` should follow accordingly as `msortf f=Key,range%n`.

Examples

Example 1: Basic Example

For records where the value of date field is 20080203, select those records in the input data where `amount` field is more than 5 but less than 15 and join field where `avg=150`. For records where `amount` field is more than 40 but less than 50, join field `avg=200`.

```
$ more dat1.csv
date,price
20080123,10
20080123,20
20080203,10
20080203,35
20080410,50
```

```
$ more ref1.csv
date,priceF,priceT,avg
20080203,5,15,150
20080203,40,50,200
$ mnrjoin k=date f=avg m=ref1.csv R=priceF,priceT r=price%n i=dat1.csv o=rsl1.csv
#END# kgnrjoin R=priceF,priceT f=avg i=dat1.csv k=date m=ref1.csv o=rsl1.csv r=price%n
$ more rsl1.csv
date%0,price,avg
20080203,10,150
```

Example 2: Output unmatched data

Use `-n` to return all records in the input data even if they do not match with those in the reference file (row where `avg=` Null), and use `-N` to return records in the reference file even if they do not match with those in the input file (rows where `price=` null). This is known as outer-join.

```
$ mnrjoin k=date f=avg m=ref1.csv R=priceF,priceT r=price%n -n -N i=dat1.csv o=rsl2.csv
#END# kgnrjoin -N -n R=priceF,priceT f=avg i=dat1.csv k=date m=ref1.csv o=rsl2.csv r=price%n
$ more rsl2.csv
date%0,price,avg
20080123,10,
20080123,20,
20080203,10,150
20080203,35,
20080203,,200
20080410,50,
```

Related Command

mrjoin : Use `mrjoin` if there are repeated values in join key (`K=` field) from the reference data.

4.44 mnullto - Replace NULL Values

Replace NULL values in the field(s) specified at `f=` parameter with a character string defined at `v=` parameter.

Format

`mnullto f= [v=|-p] [O=] [-A]` [i=] [o=] [-assert_diffSize] [-nfn] [-nfno] [-x] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Replace null values in the field(s) (multiple fields can be specified).
`v=`	Replace null values with this string.
`-p`	Replace null values in the previous row. This option cannot be specified with `v=` parameter.
`O=`	String to replace non-null values. When this parameter is not specified, non-null values will not be replaced.
`-A`	Add replacement string as new column. When `-A` option is specified, define the new field name using a colon (:) after the field name. Example: f=quantity:ReplacementFieldName.

Examples

Example 1: Basic Example

Replace NULL values in the ￥verb—birthday— field with the string “ `no value` ”.

```
$ more dat1.csv
customer,birthday
A,19690103
B,
C,19500501
D,
E,
$ mnullto f=birthday v="no value" i=dat1.csv o=rsl1.csv
#END# kgnullto f=birthday i=dat1.csv o=rsl1.csv v=no value
$ more rsl1.csv
customer,birthday
A,19690103
B,no value
C,19500501
D,no value
E,no value
```

Example 2: Replace non-NULL values

Replace Null values in the `birthday` field with the string `"no value"` and change non-null values to the string ￥verb—”value”—, and rename the output column as `entry`.

```
$ mnullto f=birthday:entry v="no value" O="value" i=dat1.csv o=rsl2.csv
#END# kgnullto O=value f=birthday:entry i=dat1.csv o=rsl2.csv v=no value
$ more rsl2.csv
customer,entry
A,value
B,no value
C,value
D,no value
E,no value
```

Example 3: Add new column

Replace Null values in the `birthday` field with the string `"no value"` and change non-null values to the string `"value"`. Output the replacement strings in a new column named `entry`.

```
$ mnullto f=birthday:entry v="no value" O="value" -A i=dat1.csv o=rsl3.csv
#END# kgnullto -A O=value f=birthday:entry i=dat1.csv o=rsl3.csv v=no value
$ more rsl3.csv
customer,birthday,entry
A,19690103,value
B,,no value
C,19500501,value
D,,no value
E,,no value
```

Example 4: Replace values in previous row

```
$ more dat2.csv
id,date
A,19690103
B,
C,19500501
D,
E,
$ mnullto f=date -p i=dat2.csv o=rsl4.csv
#END# kgnullto -p f=date i=dat2.csv o=rsl4.csv
$ more rsl4.csv
id,date
A,19690103
B,19690103
C,19500501
D,19500501
E,19500501
```

Related Commands

mdelnull : Remove rows containing NULL values.

mchgstr : Replace NULL value with character strings.

4.45 mnumber - Serials

Show the alphabetical sequence (A,B,...,Z,AA,AB,...,AZ,BA,BB,...,ZZ,AAA,AAB,...) and save the output in a new column defined at `a=` parameter.

Format

`mnumber a= [e=] [I=] [k=] [s=] [S=] [-B]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`a=`	Specify the field name and the list of new serials. [However, this parameter is not required when `-nfn or -nfno` options are specified]
`e=`	Process records with same Rank Specify how to handle fields with same key same sort values. Default setting is used(`e=seq`) with " No " attached as the field name of the sequence, if the parameter is not specified. `seq`: Create sequential serial numbers or alphabets for records with same rank. `same`: Records with equal numerical or alphabetical values receive the same rank order. `skip`: Records with equal numerical or alphabetical values receive the same rank order, number of subsequent rank is skipped for the following record. Note: `e={same/skip}` must be specified with the `s=` parameter.
`I=`	Interval between the sequence. However, a negative interval value cannot be specified for alphabet sequence.
`k=`	Generate sequential characters for the key field(s) (multiple fields can be specified)
`s=`	Specified field(s) (multiple fields can be specified) containing same rank values . Note: This parameter must be declared with `e={same/skip}`.
`S=`	Starting No Specify the starting value of the sequence. Uppercase alphabet letters for the alphabet sequence.
`-B`	Assign same sequential number or alphabet to each key. Records with the same key will each be assigned the same number (No) or alphabet.

Examples

Example 1: Sequential numbers

Generate sequential numbers for each value in ascending order in the `Customer` column. Name the sequence as `No` in a new column.

```
$ more dat1.csv
Customer,Val,Sum
A,29,300
B,35,250
C,15,200
D,23,150
E,10,100
$ mnumber s=Customer a=No i=dat1.csv o=rsl1.csv
#END# kgnumber a=No i=dat1.csv o=rsl1.csv s=Customer
$ more rsl1.csv
Customer%0,Val,Sum,No
A,29,300,0
B,35,250,1
C,15,200,2
D,23,150,3
E,10,100,4
```

Example 2: Serialize the Date column

Sequentially number items in the `Date` column according to earliest date to latest date. Use same sequence number (`No`) for same `Date`. Save the sequence in a new column named `"No"`.

```
$ more dat2.csv
Date
20090101
20090101
20090102
20090103
20090103
$ mnumber k=Date a=No -B i=dat2.csv o=rsl2.csv
#END# kgnumber -B a=No i=dat2.csv k=Date o=rsl2.csv
$ more rsl2.csv
Date%0,No
20090101,0
20090101,0
20090102,1
20090103,2
20090103,2
```

Example 3: Serialize the Sum column (use same alphabet for same Rank order)

Create a alphabetical sequence according to the `Sum` column which is arranged in descending order. Save the sequence in a new column named " `Rank` ". Assign the same alphabet character to items with the same values.

```
$ more dat3.csv
Customer,Val,Sum
A,3,300
B,1,250
C,2,250
D,1,150
E,1,100
$ mnumber a=Rank e=same s=Sum%nr S=A  i=dat3.csv o=rsl3.csv
#END# kgnumber S=A a=Rank e=same i=dat3.csv o=rsl3.csv s=Sum%nr
$ more rsl3.csv
Customer,Val,Sum%0nr,Rank
A,3,300,A
B,1,250,B
C,2,250,B
D,1,150,C
E,1,100,D
```

Example 4: Serialize the Sum column (sequential numbers for same Rank order)

Number records sequentially according to `Sum` column (sum arranged in descending order), and save serials in the `"Rank"` column. For items with same rank order, assign sequential numbers according to sort order.

```
$ mnumber a=Rank e=seq s=Sum%nr i=dat3.csv o=rsl4.csv
#END# kgnumber a=Rank e=seq i=dat3.csv o=rsl4.csv s=Sum%nr
$ more rsl4.csv
Customer,Val,Sum%0nr,Rank
A,3,300,0
B,1,250,1
C,2,250,2
D,1,150,3
E,1,100,4
```

Example 5: Serialize the Sum column (Same No for same Rank)

Number records sequentially according to `Sum` column (sum arranged in descending order), and save the numbers in the " `Rank` " column. Assign the same No to records with the same Rank order.

```
$ mnumber a=Rank e=same s=Sum%nr i=dat3.csv o=rsl5.csv
#END# kgnumber a=Rank e=same i=dat3.csv o=rsl5.csv s=Sum%nr
$ more rsl5.csv
```

```
Customer,Val,Sum%0nr,Rank
A,3,300,0
B,1,250,1
C,2,250,1
D,1,150,2
E,1,100,3
```

Example 6: Serialize the Sum column (duplicate numbers for same Rank and skip number for next record)

Number records sequentially according to `Sum` column (sum arranged in descending order), and save the numbers is the“ Rank ”column. Assign same `RankNo` number to records with same rank order, subsequent No is skipped for the following record.

```
$ mnumber a=Rank e=skip s=Sum%nr i=dat3.csv o=rsl6.csv
#END# kgnumber a=Rank e=skip i=dat3.csv o=rsl6.csv s=Sum%nr
$ more rsl6.csv
Customer,Val,Sum%0nr,Rank
A,3,300,0
B,1,250,1
C,2,250,1
D,1,150,3
E,1,100,4
```

Example 7: Number sequence starting from 10

Serialize the `Sum` column sequentially from 10 with items, where values of sum is arranged in ascending order. Save the serials in the `"Score"` column. Assign same RankNo to records with same Rank order , subsequent No is skipped for the following record.

```
$ more dat4.csv
Customer,Val,Sum
A,1,100
B,1,150
C,1,250
D,2,250
E,3,300
$ mnumber a=Score e=same s=Sum%n S=10 i=dat4.csv o=rsl7.csv
#END# kgnumber S=10 a=Score e=same i=dat4.csv o=rsl7.csv s=Sum%n
$ more rsl7.csv
Customer,Val,Sum%0n,Score
A,1,100,10
B,1,150,11
C,1,250,12
D,2,250,12
E,3,300,13
```

Example 8: Start sequence from 10 with an interval of 5

Number the `Sum` column sequentially from 10 at an interval of 5, where values of sum is arranged in ascending order. Save the serials in the “ `Score` ” column. Assign the same number to records with the same Rank order.

```
$ mnumber a=Score e=same s=Sum%n S=10 I=5 i=dat4.csv o=rsl8.csv
#END# kgnumber I=5 S=10 a=Score e=same i=dat4.csv o=rsl8.csv s=Sum%n
$ more rsl8.csv
Customer,Val,Sum%0n,Score
A,1,100,10
B,1,150,15
C,1,250,20
D,2,250,20
E,3,300,25
```

Related Commands

mnewnumber : Generate list of sequential numbers in a new dataset.

mbest : Use `mnumber` if the query requires selection of records according to line numbers.

4.46 mpadding - Row Padding

Fill in values in between the records specified at `f=` parameter based on the key field specified at `k=` parameter. When `v=` parameter is specified, create padding records in between records with the specified string other than the fields specified at `k=`,`f=`. Create padding with null values when `-n` option is specified. (Note: previous item value will be used as padding if both `v=` and `-n` parameters are not specified)

Format

`mpadding [k=] f= [v=] [S=] [E=] [-n]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameter

`k=`	Specify the key field.
`f=`	Target field name for continuous padding. Specify the field name to create padding as continuous values between records. When creating padding in numerical values, attach %n as no%n. Attach %d when creating padding for dates, or attach %t for padding for time. Atttach %r when creating padding value in descending order as no%d%r.
`v=`	Specify padding value as character string. Create padding with with the specified string other than the fields specified at k=,f=.
`S=`	Starting value Specify the starting value of the series in f=.
`E=`	Ending value Specify the ending value of the series in f=.
`-n`	Use null value as padding. Return null values in fields other than those specified in k=, f=.

Examples

Example 1: Basic Example

Create padding with integer values (`type=int`) between records in `no` column. Insert `4,5` between `3` and `6`, and `7` between `6` and `8`.

```
$ more dat1.csv
no
3
6
8
$ mpadding f=no%n i=dat1.csv o=rsl1.csv
#END# kgpadding f=no%n i=dat1.csv o=rsl1.csv
$ more rsl1.csv
no%0n
3
4
5
6
7
8
```

Example 2: Specify the starting and ending value

Insert padding between records as well as before and after the first and last records from the input data. Specify the starting and ending range at `S=,E=`.

```
$ mpadding f=no%n S=1 E=10 i=dat1.csv o=rsl2.csv
#END# kgpadding E=10 S=1 f=no%n i=dat1.csv o=rsl2.csv
$ more rsl2.csv
no%0n
1
2
3
4
5
6
7
8
9
10
```

Example 3: Padding with date

Create padding to fill in values between dates (`type=date`) in the `date` column. Create padding values in columns other than those specified at `k=,f=`.

```
$ more dat2.csv
date,dummy
20130929,a
20131002,b
20131004,c
$ mpadding f=date%d i=dat2.csv o=rsl3.csv
#END# kgpadding f=date%d i=dat2.csv o=rsl3.csv
$ more rsl3.csv
date%0,dummy
20130929,a
20130930,a
20131001,a
20131002,b
20131003,b
20131004,c
```

Example 4: Specify character string for padding

Specify the character string padding value at `v=`.

```
$ mpadding f=date%d v=padding i=dat2.csv o=rsl4.csv
#END# kgpadding f=date%d i=dat2.csv o=rsl4.csv v=padding
$ more rsl4.csv
date%0,dummy
20130929,a
20130930,padding
20131001,padding
20131002,b
20131003,padding
20131004,c
```

Example 5: Specify NULL value as padding character

NULL value can be used as padding when the `-n` option is specified.

```
$ mpadding f=date%d -n i=dat2.csv o=rsl5.csv
#END# kgpadding -n f=date%d i=dat2.csv o=rsl5.csv
$ more rsl5.csv
date%0,dummy
20130929,a
20130930,
20131001,
20131002,b
20131003,
20131004,c
```

Related Command

4.47 mpaste - Match and Merge Fields from Reference File

Merge input file with the reference file for matching rows. If data is different in size, merge with data with smaller size. It is also possible to match all data with different sizes with the inclusion of null values by specifying `-n` and `-N`.

Format

`mpaste [f=] -n -N m=|` i= [o=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	field name(s) (multiple fields can be specified) to merge with reference file. All fields except the key field are merged.
`m=`	Reference file name. Read from standard input when this parameter is noted defined (when `i=` is specified).
`-n`	Output NULL values when input data does not consist of reference data.
`-N`	Output NULL values when reference data does not consist of input data.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id1
1
2
3
4
$ more ref1.csv
id2
a
b
c
d
$ mpaste m=ref1.csv i=dat1.csv o=rsl1.csv
#END# kgpaste i=dat1.csv m=ref1.csv o=rsl1.csv
$ more rsl1.csv
id1,id2
1,a
2,b
3,c
4,d
```

Example 2: Example of merging data of different sizes

If the number of rows in the input file is different from the reference file , merge records according to the smaller file.

```
$ more ref2.csv
id2
a
b
$ mpaste m=ref2.csv i=dat1.csv o=rsl2.csv
#END# kgpaste i=dat1.csv m=ref2.csv o=rsl2.csv
$ more rsl2.csv
id1,id2
1,a
2,b
```

Example 3: Outer join

If there are less number of rows in the reference file, NULL values will be assigned to records that did not match with the input file when `-n` option is specified.

```
$ mpaste m=ref2.csv -n i=dat1.csv o=rsl3.csv
#END# kgpaste -n i=dat1.csv m=ref2.csv o=rsl3.csv
$ more rsl3.csv
id1,id2
1,a
2,b
3,
4,
```

Example 4: Define fields to join

```
$ more ref3.csv
id2,val
a,R0
b,R1
c,R2
d,R3
$ mpaste f=val m=ref3.csv i=dat1.csv o=rsl4.csv
#END# kgpaste f=val i=dat1.csv m=ref3.csv o=rsl4.csv
$ more rsl4.csv
id1,val
1,R0
2,R1
3,R2
4,R3
```

Related Command

mjoin : Join using key field(s) if row numbers are not present.

4.48 mproduct - Cartesian Join with Reference File

Combine every row of column specified at `f=` parameter from the reference file at the `m=` parameter with every record from the input file.

Format

`mproduct [f=] m=|` i= [o=] [bufcount=] [-assert_diffSize] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Combine field name(s) (multiple fields can be specified) from reference file. All field(s) are combine if this parameter is not specified.
`m=`	Specify reference file name. Read from standard input if this parameter is not defined (when i= is specified).

Examples

Example 1: Basic Example

Combine the `date` column from reference file to the `customer` column from the input file.

```
$ more dat1.csv
customer
A
B
$ more ref1.csv
date
20090101
20090201
20090301
$ mproduct f=date m=ref1.csv i=dat1.csv o=rsl1.csv
#END# kgproduct f=date i=dat1.csv m=ref1.csv o=rsl1.csv
$ more rsl1.csv
customer,date
A,20090101
A,20090201
A,20090301
B,20090101
B,20090201
B,20090301
```

Related Command

mnjoin : Similar operation with `mproduct` but join key is specified.

4.49 mrand - Generate Random Numbers

Generate random number from the range 0.0 to 0.1, or generate random integers from a defined range. Define output column name at `a=` parameter.

This command uses Mersenne twister (developed in 1937) as pseudo random number generator. (Webpage of author , boost library).

Format

`mrand [k=] a= [max=] [min=] [S=] [-int]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`k=`	Same random number is generated for same key value at the specified key field.
`a=`	New column name. [However, this parameter is not required when -nfn,-nfno option is specified]
`max=`	Maximum value of random number [default value: INT_MAX] Integer up to `0\UTF{FF5E}2^32` [21 billion]). `-int` must be specified with this parameter.
`min=`	Minimum value of random number [default=0]. Integer up to `0\UTF{FF5E}2^32` [21 billion]). `-int` must be specified with this parameter.
`S=`	Random seed [default value: current time] The same random seed generates the same random number When `S=` is not specified, the default setting of random seed is set to the current time. Random seed value can be specified between -2147483648～2147483647.
`-int`	Generate random integers.

Examples

Example 1: Basic example

Generate random real numbers between 0.0 to 1.0.

```
$ more dat1.csv
Customer
A
B
C
D
E
$ mrand a=rand i=dat1.csv o=rsl1.csv
#END# kgrand a=rand i=dat1.csv o=rsl1.csv
$ more rsl1.csv
Customer,rand
A,0.644309161
B,0.6673309144
C,0.2995640722
D,0.6937571361
E,0.1363564723
```

Example 2: Basic Example 2

Generate random integers with -int.

```
$ mrand a=rand -int i=dat1.csv o=rsl2.csv
#END# kgrand -int a=rand i=dat1.csv o=rsl2.csv
$ more rsl2.csv
Customer,rand
A,1321664177
```

```
B,1896318462
C,1776188712
D,1396773284
E,886219333
```

Example 3: Specify the minimum and maximum value of the random number

Generate a random number with a minimum value of 10 and maximum value of 100. Add the random numbers to a new column named `rand`.

```
$ mrand a=rand -int min=10 max=100 S=1 i=dat1.csv o=rsl3.csv
#END# kgrand -int S=1 a=rand i=dat1.csv max=100 min=10 o=rsl3.csv
$ more rsl3.csv
Customer,rand
A,47
B,100
C,75
D,94
E,10
```

Example 4: Generate random number by key

Given 4 customers `A,B,C,D`, same random number is generated for same customer.

```
$ more dat2.csv
Customer
A
A
A
B
B
C
D
D
D
$ mrand k=Customer -int min=0 max=1 a=rand i=dat2.csv o=rsl4.csv
#END# kgrand -int a=rand i=dat2.csv k=Customer max=1 min=0 o=rsl4.csv
$ more rsl4.csv
Customer%0,rand
A,0
A,0
A,0
B,1
B,1
C,0
D,1
D,1
D,1
```

Related Command

mselrand : Select a random record.

mnewrand : Generate new random dataset without using input file.

4.50 mrjoin - Join Reference File According to Specified Range

Join fields from reference file according to specified range.

The value specified at `r=` parameter from the input data is matched with the range (value that falls above the first row and less than the next row) from the reference file, and subsequently joined with the value from the field specified at `f=` parameter. Use `mnrjoin` to join with complex conditions using range values. Consider using `chgnum` if there are not a lot of range.

Format

`mrjoin r= [k=] [K=] [R=] [f=] [-n] [-lo] [m=]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	The field name(s) (multiple fields can be specified) to join from the reference file. When this is not defined, the all fields except the key specified at `K=` will be joined.
`m=`	Reference file name. Read from standard input if this parameter is not set (when `i=` is specified).
`r=`	Field name of the range for comparison [%n] Specify the field name from the input file. After the specified field is sorted (multiple fields can be specified), fields are joined The value is interpreted as numeric range when %n is specified, otherwise, it is treated as character range. The specified field should not contain NULL values or the data may not be processed properly.
`R=`	Field name containing range values in the reference file. When this parameter is not defined, the range is processed at `r=` parameter by default.
`k=`	Key field name(s) (multiple fields can be specified) from the input data for comparison Join records with same key fields in the input data `k=` and reference data `K=`.
`K=`	Key field name(s) (multiple fields can be specified) from the reference data for comparison This key field(s) from reference data is compared with the key field(s) from the input data specified at `k=` parameter, fields where records with common key are joined. This parameter is not required if the field name name is the same as the one defined at the `k=` parameter.
`-n`	Output NULL values when reference data does not consist of input data.
`-lo`	left-open interval The range with left open interval specified at `R=` parameter (greater than - below).

Examples

Example 1: Basic Example

Join category field `low, middle,high` to corresponding `price` range.

```
$ more dat1.csv
price
8
15
35
50
90
200
$ more ref1.csv
range,category
10,low
35,middle
80,high
100,
$ mrjoin r=price%n m=ref1.csv R=range f=category i=dat1.csv o=rsl1.csv
#END# kgrjoin R=range f=category i=dat1.csv m=ref1.csv o=rsl1.csv r=price%n
$ more rsl1.csv
```

```
price%0n,category
15,low
35,middle
50,middle
90,high
```

Example 2: Basic Example 2

```
$ mrjoin -lo r=price%n m=ref1.csv R=range f=category i=dat1.csv o=rsl2.csv
#END# kgrjoin -lo R=range f=category i=dat1.csv m=ref1.csv o=rsl2.csv r=price%n
$ more rsl2.csv
price%0n,category
15,low
35,low
50,middle
90,high
```

Example 3: Basic Example 3

```
$ mrjoin -n r=price%n m=ref1.csv R=range f=category i=dat1.csv o=rsl3.csv
#END# kgrjoin -n R=range f=category i=dat1.csv m=ref1.csv o=rsl3.csv r=price%n
$ more rsl3.csv
price%0n,category
8,
15,low
35,middle
50,middle
90,high
200,
```

Example 4: Join with different ranges for corresponding products

```
$ more dat2.csv
item,price
A,10
A,20
B,10
B,20
$ more ref2.csv
item,price,category
A,10,low
A,15,high
A,100,
B,10,low
B,35,high
B,100,
$ mrjoin k=item r=price%n m=ref2.csv f=category i=dat2.csv o=rsl4.csv
#END# kgrjoin f=category i=dat2.csv k=item m=ref2.csv o=rsl4.csv r=price%n
$ more rsl4.csv
item%0,price%1n,category
A,10,low
A,20,high
B,10,low
B,20,low
```

Related Commands

mchgnum : Specify a number range to replace / add value.

mjoin : Use this command to join matching strings instead of using numeric range.

mnrcommon : Use this command to select records rather than to join fields.

4.51 msed - Replace String Matching Regular Expression

Replace string in the fields specified in the `f=` parameter with a string specified in the `v=` parameter for content that matches the regular expression specified in the `c=` parameter .

Format

msed c= f= v= [-A] [-g] [-W] [i=] [o=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [--help] [--helpl] [--version]

Parameters

`f=`	specify the target list of field name(s) (multiple fields can be specified) for parsing.
`c=`	Define the regular expression for string substitution. Refer to usage of regular expressions.
`v=`	Specify the string to replace the substring that matches with the regular expression specified in the `c=` parameter. It is possible to substitute match result with the following methods: `$&` : Matched string `` $` `` : Search for the string from the beginning of the target replacement character string, until a string is matched. `$'` : After a matched string, substitute target replacement string with matched string till the end. `$N` : partial string match for the N-th occurrance (`N>=1`).
`-A`	Instead of replacing the specified field, add field as a new column.
`-g`	Replace all matches of the regular expression.
`-W`	Replace wide character matches of the regular expression.

Using regular expressions

List of regular expression specified in the `c=` parameter is shown from Table 4.9 to Table 4.12.

Table 4.9: Regular expression match with 1 character

Regular expression	Description	Example of pattern	Example of c=,v=	Result
.	Any character	abbbcc	c=. v=X -g	XXXXXX
[abc]	either a,b, or c character	abbbcc	c=[ac] v=X -g	XbbbXX
[^abc]	Any character other than a,b,c	abbbcc	c=[^ac] v=X -g	aXXXcc
[a-z]	Any character from a to z	abbbcc	c=[a-b] v=X -g	XXXXcc
[^a-z]	Any character outside the range of a to z	abbbcc	c=[^a-b] v=X -g	abbbXX
\t	Tab character			
\w	Word string ([0-9a-zA-Z_])	ab#cd&ef	c=\w v=X -g	XX#XX&XX
\W	Characters other than Word string	ab#cd&ef	c=\w v=X -g	abXcdXef
\s	Space character ([\t])	ab cd ef	c=\s v=X -g	abXcdXef
\S	Non-whitespace character	ab cd ef	c=\s v=X -g	XX XX XX
\d	Numeric constituent characters ([0-9])	ab12c0	c=\d v=X -g	abXXcX
\D	Non-numeric constituent characters	ab12c0	c=\d v=X -g	XX12X0

Table 4.10: Repetition of regular expressions

Regular expression	Description	Example of pattern	Example of c=,v=	Result
a*	Zero or more repetition of a	abbbcc	c=ab* v=X	Xcc
a+	Repetition of one or more a	abbbcc	c=ab+ v=X	Xcc
a?	Single occurrence of a	abbbcc	c=ab? v=X	Xbbcc
a{M,N}	Repetition of a more than M and less than N	abbbbbcc	c=ab{3,4} v=X	Xbcc
a{M}	Repetition of a more than M times	abbbbbcc	c=ab{3} v=X	Xbbcc
a | b	a or b	abbbc	c=(ab) | (bc) v=X	XbX
?	Shortest match after the repeat sign	abbbc	c=ab*? v=X	Xbbbc

Table 4.11: Position of regular expression

Regular expression	Description	Example of pattern	Example of `c=,v=`	Result
`^`	Match from the beginning	`abac`	`c=^a v=X -g`	`Xbac`
`$`	Match till the end	`acac`	`c=c$ v=X -g`	`acaX`
`\b`	Match starting characters of string	`aac ba ac bac`	`c=\ba v=X -g`	`Xac bX Xc bac`
`\B`	Match within the string	`aac ba ac bac`	`c=\Ba v=X -g`	`aXc ba ac bXc`

Table 4.12: Others

Regular expression	Description	Example of pattern	Example of `c=,v=`	Result
(expr)	Grouping			
`\1,..,\9`	Back reference	`abbcababc`	`c=(ab)(bc)\1 v=x`	`Xabc`
`(?=expr)`	Position before matched string at `expr`			
`(?!expr)`	Position before unmatched string at `expr`			

Examples

Example 1: Basic Example

Replace the 4-digit substring in the `zipCode` field starting 00 with `####`.

```
$ more dat1.csv
customer,zipCode
A,6230041
B,6240053
C,6330032
D,6230087
E,6530095
$ msed f=zipCode c=00.. v=#### i=dat1.csv o=rsl1.csv
#END# kgsed c=00.. f=zipCode i=dat1.csv o=rsl1.csv v=####
$ more rsl1.csv
customer,zipCode
A,623####
B,624####
C,633####
D,623####
E,653####
```

Example 2: Specify field name

Replace the 4-digit substring in the `zipCode` field starting 00 with `####`. Save output in column `zipCode4`.

```
$ msed f=zipCode:zipCode4 c='00\d\d' v=#### i=dat1.csv o=rsl2.csv
#END# kgsed c=00\d\d f=zipCode:zipCode4 i=dat1.csv o=rsl2.csv v=####
$ more rsl2.csv
customer,zipCode4
A,623####
B,624####
C,633####
D,623####
E,653####
```

Example 3: Global replacement

Global search using the regular expression `-` to replace value of `0` in `zipCode`.

```
$ msed f=zipCode c=0 v=- -g i=dat1.csv o=rsl3.csv
#END# kgsed -g c=0 f=zipCode i=dat1.csv o=rsl3.csv v=-
$ more rsl3.csv
customer,zipCode
A,623--41
B,624--53
C,633--32
```

```
D,623--87
E,653--95
```

Example 4: Replace substring

Delete `fruit` from the beginning of the string in `item`. Note that when first match (^) is specified, the substring within the word `grapefruit` in the last row is retained.

```
$ more dat2.csv
item,price
fruit:apple,100
fruit:peach,250
fruit:pineapple,300
fruit:orange,450
fruit:grapefruit,500
$ msed f=item c='^fruit' v= -g i=dat2.csv o=rsl4.csv
#END# kgsed -g c=^fruit f=item i=dat2.csv o=rsl4.csv v=
$ more rsl4.csv
item,price
:apple,100
:peach,250
:pineapple,300
:orange,450
:grapefruit,500
```

Example 5: Substitution using match results

Replaced 1 or more consecutive character strings of `b` using `$&` is defined in the `v=`.

```
$ more dat3.csv
str1
abc
abbc
ac
$ msed f=str1 c='b+' v='#$&#' i=dat3.csv o=rsl5.csv
#END# kgsed c=b+ f=str1 i=dat3.csv o=rsl5.csv v=#$&#
$ more rsl5.csv
str1
a#b#c
a#bb#c
ac
```

Example 6: Combination of the global match

When performing a global match, each match is evaluated against the contents defined at `v=`.

```
$ msed f=str1 c=b v='#$&#' -g i=dat3.csv o=rsl6.csv
#END# kgsed -g c=b f=str1 i=dat3.csv o=rsl6.csv v=#$&#
$ more rsl6.csv
str1
a#b#c
a#b##b#c
ac
```

Example 7: Prefix substitution

Replace the matching first character of b in the character string (prefix) using `` $` ``.

```
$ msed f=str1 c=b v='#$`#' i=dat3.csv o=rsl7.csv
#END# kgsed c=b f=str1 i=dat3.csv o=rsl7.csv v=#$`#
$ more rsl7.csv
str1
a#a#c
a#a#bc
ac
```

Example 8: Suffix substitution

Replace the matching last character of b in the character string (suffix) using `$'`.

```
$ msed f=str1 c=b v="#$'#" i=dat3.csv o=rsl8.csv
#END# kgsed c=b f=str1 i=dat3.csv o=rsl8.csv v=#$'#
$ more rsl8.csv
str1
a#c#c
a#bc#bc
ac
```

Related Commands

mchgstr : Use this command to replace with a simple string match.

mcal : Include several functions to handle the regular expression.

4.52 msel - Select Records with Conditions

Define the computation criteria at `c=` parameter, the record is selected if condition returns true. All operators and functions available in mcal command can be used in the conditional function. For more details, please refer to mcal.

Format

`msel c= [u=] [-r]` [i=] [o=] [-assert_diffSize] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`c=`	Define the expression using combinations of operators and functions. Refer to mcal for more details.
`o=`	Records matching the condition will be printed to this output file.
`u=`	Records that do not match the condition will be printed to this output file.
`-r`	Reverse selection Select records excluded from the selection condition defined in `c=`

Examples

Example 1: Basic example

Select records where "Amount" is greater than 40. Write the unmatched records to a different output file file `unmatch1.csv`.

```
$ more dat1.csv
Customer,Quantity,Amount
A,1,10
A,2,20
B,1,30
B,3,40
B,1,50
$ msel c='${Amount}>40' u=unmatch1.csv i=dat1.csv o=match1.csv
#END# kgsel c=${Amount}>40 i=dat1.csv o=match1.csv u=unmatch1.csv
$ more match1.csv
Customer,Quantity,Amount
B,1,50
$ more unmatch1.csv
Customer,Quantity,Amount
A,1,10
A,2,20
B,1,30
B,3,40
```

Example 2: Selecting records with null value(s)

No records will be selected when the condition defined `c=` returned a null value. Records that do not match the condition will be written to a separate file defined in `u=`.

In the following example, the first three rows of data from column `b` are `-1`, null, and `1`. When selecting records where `b` is greater than 0, the query record with a null value will be treated as an exception saved in the unmatched records file.

```
$ more dat2.csv
a,b
A,-1
B,
C,1
$ msel c='${b}>0' i=dat2.csv o=match2.csv u=unmatch2.csv
#END# kgsel c=${b}>0 i=dat2.csv o=match2.csv u=unmatch2.csv
```

```
$ more match2.csv
a,b
C,1
$ more unmatch2.csv
a,b
A,-1
B,
```

Example 3: Specify -r option

Null value is always evaluated as a unknown value regardless of the condition. Thus, records with null value is not selected.

In the following example, the reverse selection parameter `-r` is used with the same condition in the previous example. Even though the selection criteria is inverted, the query record with a null value will be treated as an exception saved in the unmatched records file as in the previous example.

```
$ msel -r c='${b}>0' i=dat2.csv o=match3.csv u=unmatch3.csv
#END# kgsel -r c=${b}>0 i=dat2.csv o=match3.csv u=unmatch3.csv
$ more match3.csv
a,b
A,-1
$ more unmatch3.csv
a,b
B,
C,1
```

Related Commands

mselnum : Select records with simple numeric range.

mselstr : Select records matching query string

mcal : Return the calculated results instead of selecting records.

4.53 mseldsp Display option selection screen

Note: This command is a beta. Its specifications may be changed.

This command displays a list of character strings to choose from at the position on the terminal specified by coordinate `x=,y=`. The list is specified by the `i=` or `seldata=` parameter. The user can choose only one from the list. To allow multiple choice, use the mmseldsp command. The output CSV file contains one row and one field. When the command ends with nothing entered in the input frame, null is output (that is, only a line feed is output). To output a fieldname, use the `f=` parameter. With `f=` omitted, no fieldname header is output. If there are too many choices to fit into the screen, use the `height=` parameter to specify the number of rows shown in a scroll window. When the Enter key is pressed on the selection screen, the command returns end status 0 and ends. When the ESC key is pressed on the screen, the command returns end status 1 and ends. In either case, the choice made on the selection screen is output to a file. In the coordinate system, the top left is `x=1,y=1` (escape sequence specifications). If the value specified for `x=` or `y=` is smaller than 1, the command assumes 1 and operates. The operation is indefinite if the specified coordinate is outside the scope of the terminal.

Format

`mseldsp x= y= [height=] i=|seldata=` o= [-nfn] [-nfno] [-x] `[--help] [--helpl] [--version]`

Parameters

`x=`	Specify the display start position (1 or greater) on the x axis (horizontal, left to right)
`y=`	Specify the display start position (1 or greater) on the y axis (vertical, top to bottom)
`height=`	Specify the number of rows in which the choices are shown.
`i=`	Specify the name of the CSV file containing the choice character strings as fields.
`f=`	Specify the name of the CSV file containing the choice character strings as fields.
`seldata=`	Specify the list of character strings to choose from, using commas ad delimiters.

Examples

Example 1: Basic example

The contents of `sel.txt` are displayed at x=10,y=3 on the terminal, and the character string selected by the user is output to `rsl1.txt`.

```
$ more sel.txt
1:I agree.
2:I do not know.
3:I disagree.
$ mseldsp x=10 y=3 i=sel.txt o=rsl1.txt
# Assume that the user has selected the first row.
$ mose rsl1.txt
1:I agree.

The following will be displayed:
+---------------------------------------
|
|
|          \textColor{red}{black}{1:I agree.}
|          \textColor{white}{black}{2:I do not know.}
|          \textColor{white}{black}{3:I disagree.}
|
```

Example 2: Using arguments to specify the character strings

This script works in the same way as Example 1, but the option character strings are specified by `seldata=`.

```
$ mseldsp x=10 y=3 seldata=1:I agree.,2:I do not know.,3:I disagree. o=rsl2.txt
# Assume that the user has selected the second row.
$ mose rsl2.txt
2:I do not know.

The following will be displayed:
+---------------------------------------
|
|
|          \textColor{white}{black}{1:I agree.}
|          \textColor{red}{black}{2:I do not know.}
|          \textColor{white}{black}{3:I disagree.}
|
```

Example 3: Judging end status

The parameters are the same as in Example 1. This script judges the end status and performs different operations accordingly.

```
$ more scp.sh
rm -f rsl3.csv
clear
mseldsp x=10 y=3 seldata=aaaa,bbbb,cccc o=rsl3.csv
if [ $? = 0 ] ; then
  clear ; echo "end by enter key"
else
  clear ; echo "end by escape key"
fi

# Result of selecting aaaa and pressing Enter
$ bash scp.sh
end by enter key
$ more rsl3.csv
aaaa

# Result of selecting bbbb and pressing ESC
$ bash scp.sh
end by escape key
$ more rsl3.csv
bbbb
```

Related Commands

minput : Displays the input screen. mminput :Displays the input screen with multiple input frames. mdsp : Displays a character string at a specified position on the screen. mmseldsp : Displays a multiple-choice input window on the screen.

4.54 mselnum - Select Records Matching Range

This command selects records that matches the range specified at `c=` with the values in the column specified at the `f=` parameter. Various selection criteria can be specified as arguments in the parameters as follows. Use msel command to query complex selection criteria such as matching against a combination of string characters. For more information about OR, AND conditions for key field, refer to `mselstr` command.

- Range types include open range, closed range, inclusive and exclusive bounds and infinite range.
- `c=`specify multiple range and select records that matches either range (OR condition).
- `f=`specify multiple fields at which the values matches either range (OR condition).
- `f=`change the logical operator from OR to AND with the `-F` option.
- `k=`specify key value as the selection unit.
- Use AND logical operator to select records based on the key value with the `-R` option.

Typical examples are shown in 4.13 to 4.16.

Table 4.13: Input Data

key	val
a	1
a	-3
b	3
b	6

Table 4.14: Select rows if the values in the val column falls on the range of 1 to 3

`mselnum f=val c='[1,3]'`

key	val
a	1
b	3

Table 4.15: Select rows in val field with values greater than or equal to 1 but less than 3

`mselnum f=val c='[1,3)'`

key	val
a	1

Table 4.16: Select rows with values greater than or equal to 5

`mselnum f=val c='[5,)'`

key	val
b	6

Table 4.17: Select rows less than or equal to 1 or greater than or equal to 5

`mselnum f=val c='(,1],[5,)'`

key	val
a	1
a	-3
b	6

Format

`mselnum f= c= [k=] [u=] [-F] [-r] [-R]` [i=] [o=] [bufcount=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

`f=`	Field name of target query (allow multiple fields).
`c=`	Select row(s) where the data array specified at `f=` parameter matches with the specified range at this parameter (allow multiple ranges).
`k=`	Key field(s) as unit of selection (Multiple range can be specified).
`o=`	Records matching the condition will be printed to this output file.
`u=`	Records that do not match the condition will be printed to this output file.
`-F`	Select all records that matches the defined value if multiple items are defined at `f=` parameter.
`-r`	Reverse selection Select records excluded from the selection condition.
`-R`	Select if all rows with the same key specified at `k=` parameter matches the criteria.

Parameters

Examples

Example 1: Basic Example

Select rows where `val` is greater than 2 and below 5, i.e. records matching `id=2,5` are selected.

```
$ more dat1.csv
id,val
1,5.1
2,5
3,-2.0
4,
5,2.0
$ mselnum f=val c='[2,5]' i=dat1.csv o=rsl1.csv
#END# kgselnum c=[2,5] f=val i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val
2,5
5,2.0
```

Example 2: Greater than range

Select rows where `val` is greater than 2, i.e. records where `id=1,2,5`.

```
$ mselnum f=val c='[2,]' i=dat1.csv o=rsl2.csv
#END# kgselnum c=[2,] f=val i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val
1,5.1
2,5
5,2.0
```

Related Commands

msel : Selection of records by conditional expressions.

mselstr : Selection by string matching.

4.55 mselrand - Random Sampling

Random selection of records based on the number of rows set at `c=` and `p=` parameters (random sampling without replacement). When `k=` is specified, a defined number of records with same key are randomly selected, when option `-B` specified at the same time, records are selected based in the key.

This command used Mersenne twister (developed in 1937) as pseudo random number generator (Webpage of author , boost library).

Format

`mselrand c=|p= [k=] [S=] [u=] [-B]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`c=`	Select row(s) based on the number of keys and field specified. This parameter must be specified when `p=` is not specified.
`p=`	Define the percentage of records for selection based on each key value. This parameter must be specified if `c=` parameter is not specified.
`k=`	Select certain number of rows randomly from records with same key (Allow multiple fields).
`S=`	The same random seed generates the same row selection sequence. The default setting of random seed is set to the current time if the random seed is not specified. Range of random seed value is between -2147483648 - 2147483647.
`u=`	Print unmatched records to this output file.
`-B`	Selection based on key unit.

Examples

Example 1: Basic Example

Randomly select 1 transaction for each customer.

```
$ more dat1.csv
Customer,Date,Amount
A,20081201,10
A,20081207,20
A,20081213,30
B,20081002,40
B,20081209,50
$ mselrand k=Customer c=1 S=1 i=dat1.csv o=rsl1.csv
#END# kgselrand S=1 c=1 i=dat1.csv k=Customer o=rsl1.csv
$ more rsl1.csv
Customer%0,Date,Amount
A,20081201,10
B,20081002,40
```

Example 2: Randomly select a percentage of records

Select 50% of each customers' records at random. Save other records to a separate file `oth.csv`.

```
$ mselrand k=Customer p=50 S=1 u=oth2.csv i=dat1.csv o=rsl2.csv
#END# kgselrand S=1 i=dat1.csv k=Customer o=rsl2.csv p=50 u=oth2.csv
$ more rsl2.csv
Customer%0,Date,Amount
A,20081201,10
B,20081002,40
$ more oth2.csv
Customer%0,Date,Amount
A,20081207,20
```

```
A,20081213,30
B,20081209,50
```

Example 3: Select records by same key

In the following example, select two out of the four customers `A,B,C,D` at random. Customer `C,D` is selected, and all records of customer `C,D` is printed to the output.

```
$ more dat2.csv
Customer,Date,Amount
A,20081201,10
A,20081207,20
A,20081213,30
B,20081002,40
B,20081209,50
C,20081210,60
D,20081201,70
D,20081205,80
D,20081209,90
$ mselrand k=Customer c=2 S=1 -B i=dat2.csv o=rsl3.csv
#END# kgselrand -B S=1 c=2 i=dat2.csv k=Customer o=rsl3.csv
$ more rsl3.csv
Customer%0,Date,Amount
C,20081210,60
D,20081201,70
D,20081205,80
D,20081209,90
```

Related Commands

msel : Use normally distributed random numbers.

mrand : Add random numbers as a new column.

4.56 mselstr - Select Records Matching Query String

For records where the values in fields `f=` match with the string specified at `v=` are selected.

Commonly used examples are shown in 4.18 - 4.20. In table 4.19, select records where `val` is `"y"` regardless of the value of `key`. In table 4.20, if any of the record contains the value `"x"` in `val`, records with the same key value will be selected. i.e. all records with value `"a"` in `key` column are selected. Since none of the records with key value `"b"` contains value `"x"`, none records are selected.

Table 4.18: Input data

key	val
a	x
a	y
b	y
b	z

Table 4.19: f=val v=y

key	val
a	y
b	y

Table 4.20: k=key f=val v=x

key	val
a	x
a	y

Various selection criteria can be carried out with the parameters below. Use msel command to build complex conditions using regular expressions and operators which cannot be specified in this command.

- `v=` Match any character string from the list of string(s) specified.
- `f=` Match character string from the column(s) specified.
- AND operator can be used to match values multiple fields (`-F` option).
- Matching for exact, start, middle or partial string can be specified (`-head,-tail,-sub` option).
- `k=` Select records related to the defined key.
- Select records that matches all conditions by the key field (`-R` option).

Sample data with same key, containing two records and two columns is shown in (Table4.21).

```
mselstr k=key f=fld1,fld2 v=s1,s2
```

Matching criteria without `-R,-F` options are shown in 4.22.

Table 4.21: Input data

key	fld1	fld2
k	v_{a1}	v_{a2}
k	v_{b1}	v_{b2}

Table 4.22: Using the input data shown in Table 4.21, the query results of the command mselstr k=key f=fld1,fld2 v=v1,v2 with and without -R and F options differs accordingly. If query matches all conditions, the output will print all rows (2 rows), when there is no matched records, no records will returned.

-F	-R	Matching conditions
		((v_{a1} == s1 or v_{a1} == s2) or (v_{a2} == s1 or v_{a2} == s2)) or ((v_{b1} == s1 or v_{b1} == s2) or (v_{b2} == s1 or v_{b2} == s2))
-F		((v_{a1} == s1 or v_{a1} == s2) and (v_{a2} == s1 or v_{a2} == s2)) or ((v_{b1} == s1 or v_{b1} == s2) and (v_{b2} == s1 or v_{b2} == s2))
	-R	((v_{a1} == s1 or v_{a1} == s2) or (v_{a2} == s1 or v_{a2} == s2)) and ((v_{b1} == s1 or v_{b1} == s2) or (v_{b2} == s1 or v_{b2} == s2))
-F	-R	((v_{a1} == s1 or v_{a1} == s2) and (v_{a2} == s1 or v_{a2} == s2)) and ((v_{b1} == s1 or v_{b1} == s2) and (v_{b2} == s1 or v_{b2} == s2))

Format

`mselstr f= v= [k=] [u=] [-F] [-r] [-R] [-sub] [-head] [-tail] [-W]` [i=] [o=] [bufcount=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

`f=`	Target field name for query (allow multiple fields).
`v=`	Select rows(s) where the string specified at `f=` parameter matches any of the specified string(s) (allow multiple fields).
`k=`	Select records based on the defined key field (allow multiple fields).
`o=`	Print record(s) matching query to specified output file.
`u=`	Print unmatched record(s) to this output file.
`-F`	Match all character strings specified at the `f=` parameter.
`-r`	Reverse selection Remove record matching records.
`-R`	Returns rows that match all character strings specified at the `k=` parameter.
`-sub`	Search for substring that matches the part of the string pattern. Select records based on substring match specified in `v=` parameter, against the string specified column(s) at the `f=` parameter.
`-head`	Match beginning of string
`-tail`	Match end of string
`-W`	Match a sequence of wide-character substring when `-sub`,`-head`,`-tail` option is specified.

Parameters

Examples

Example 1: Basic example

Select records matching `apple` and `orange` in the Product field , print matching results to `rsl1.csv` file. Unmatched records such as `pineapplejuice` will be saved to other.csv file using the parameter `u=oth1.csv`.

```
$ more dat1.csv
Product,Amount
apple,100
milk,350
orange,100
pineapplejuice,500
wine,1000
$ mselstr f=Product v=apple,orange u=oth1.csv i=dat1.csv o=rsl1.csv
#END# kgselstr f=Product i=dat1.csv o=rsl1.csv u=oth1.csv v=apple,orange
$ more rsl1.csv
Product,Amount
apple,100
orange,100
$ more oth1.csv
Product,Amount
milk,350
pineapplejuice,500
wine,1000
```

Example 2: Remove records

Contrary to example 1, remove records matching keywords `apple` and `orange` using the `-r` option, the output is saved to `rsl2.csv` file.

```
$ mselstr f=Product  v=apple,orange -r i=dat1.csv o=rsl2.csv
#END# kgselstr -r f=Product i=dat1.csv o=rsl2.csv v=apple,orange
$ more rsl2.csv
Product,Amount
milk,350
pineapplejuice,500
wine,1000
```

Example 3: Select based on the key unit

Select all records of customer who have purchased oranges by specifying `Customer` at the `k=` parameter. Save unmatched records to `oth2.csv`.

```
$ more dat2.csv
Customer,Product,Amount
A,apple,100
```

```
A,milk,350
B,orange,100
B,orange,100
B,pineapple,500
B,wine,1000
C,apple,100
C,orange,100
$ mselstr k=Customer f=Product v=orange u=oth2.csv i=dat2.csv o=rsl3.csv
#END# kgselstr f=Product i=dat2.csv k=Customer o=rsl3.csv u=oth2.csv v=orange
$ more rsl3.csv
Customer%0,Product,Amount
B,orange,100
B,orange,100
B,pineapple,500
B,wine,1000
C,apple,100
C,orange,100
$ more oth2.csv
Customer%0,Product,Amount
A,apple,100
A,milk,350
```

Example 4: Partial match

Select records where the `Product` field contain the keyword `apple`, and save the output to a file named `rsl4.csv`. Records with partial match such as `pine(apple)juice` will also be saved in the output file `rsl4.csv`.

```
$ mselstr f=Product v=apple -sub i=dat1.csv o=rsl4.csv
#END# kgselstr -sub f=Product i=dat1.csv o=rsl4.csv v=apple
$ more rsl4.csv
Product,Amount
apple,100
pineapplejuice,500
```

Example 5: Wide character substring match

Select records where the `Product` field contains wide characters "柿", "桃", and "葡萄".

Matching maybe based on single byte character if the query string includes wide character, the query string maybe interpreted as multibyte character for matching. Therefore, it is necessary indicate wide character in the query string with `-W` option.

```
$ more dat3.csv
Product,Amount
fruit:柿,100
fruit:桃,250
fruit:葡萄,300
fruit:梨,450
fruit:苺,500
$ mselstr f=Product v=柿,桃,葡萄 -sub -W i=dat3.csv o=rsl5.csv
#END# kgselstr -W -sub f=Product i=dat3.csv o=rsl5.csv v=柿,桃,葡萄
$ more rsl5.csv
Product,Amount
fruit:柿,100
fruit:桃,250
fruit:葡萄,300
```

Example 6: Select product(s) with consecutive purchases in 2013.

Use the `-F` option to select transactions where the date of purchase and the previous date of purchase for the product both took place in 2013. Save the query results to an output file `rsl6.csv`. Save unmatched records to `oth3.csv`.

```
$ more dat4.csv
Customer,Product,Amount,Gender,Date,PreviousDate
A,apple,100,1,2013/01/04,2013/01/01
A,milk,350,1,2013/04/04,2011/05/06
```

```
B,orange,100,2,2012/11/11,2011/12/12
B,orange,100,2,2013/05/30,2012/11/11
B,pineapple,500,2,2013/04/15,2013/04/01
B,wine,1000,2,2012/12/24,2011/12/24
C,apple,100,2,2013/02/14,NULL
C,orange,100,2,2013/02/14,2013/01/31
D,orange,100,2,2011/10/28,NULL
$ mselstr f=Date,PreviousDate -F -sub v=2013 u=oth3.csv i=dat4.csv o=rsl6.csv
#END# kgselstr -F -sub f=Date,PreviousDate i=dat4.csv o=rsl6.csv u=oth3.csv v=2013
$ more rsl6.csv
Customer,Product,Amount,Gender,Date,PreviousDate
A,apple,100,1,2013/01/04,2013/01/01
B,pineapple,500,2,2013/04/15,2013/04/01
C,orange,100,2,2013/02/14,2013/01/31
$ more oth3.csv
Customer,Product,Amount,Gender,Date,PreviousDate
A,milk,350,1,2013/04/04,2011/05/06
B,orange,100,2,2012/11/11,2011/12/12
B,orange,100,2,2013/05/30,2012/11/11
B,wine,1000,2,2012/12/24,2011/12/24
C,apple,100,2,2013/02/14,NULL
D,orange,100,2,2011/10/28,NULL
```

Example 7: Extract all transactions of customers who have consecutive purchases in 2013

Use `k=` parameter to select all transactions of customers who have purchased a product with date of purchase and date of previous purchase both took place in 2013. Save unmatched records to a file `oth4.csv`.

```
$ mselstr k=Customer f=Date,PreviousDate -F -sub v=2013 u=oth4.csv i=dat4.csv o=rsl7.csv
#END# kgselstr -F -sub f=Date,PreviousDate i=dat4.csv k=Customer o=rsl7.csv u=oth4.csv v=2013
$ more rsl7.csv
Customer%0,Product,Amount,Gender,Date,PreviousDate
A,apple,100,1,2013/01/04,2013/01/01
A,milk,350,1,2013/04/04,2011/05/06
B,orange,100,2,2012/11/11,2011/12/12
B,orange,100,2,2013/05/30,2012/11/11
B,pineapple,500,2,2013/04/15,2013/04/01
B,wine,1000,2,2012/12/24,2011/12/24
C,apple,100,2,2013/02/14,NULL
C,orange,100,2,2013/02/14,2013/01/31
$ more oth4.csv
Customer%0,Product,Amount,Gender,Date,PreviousDate
D,orange,100,2,2011/10/28,NULL
```

Example 8: Select new customer(s) who purchased in 2013

Use the `-R` option to select all transactions of new customer(s) who made their first purchase in 2013, where date of previous purchase is NULL. Write the query results to an output file ¥verb—rsl8.csv—, and save unmatched records to `oth5.csv`.

```
$ mselstr k=Customer f=Date,PreviousDate -F -R -sub v=2013,NULL u=oth5.csv i=dat4.csv o=rsl8.csv
#END# kgselstr -F -R -sub f=Date,PreviousDate i=dat4.csv k=Customer o=rsl8.csv u=oth5.csv v=2013,NULL
$ more rsl8.csv
Customer%0,Product,Amount,Gender,Date,PreviousDate
C,apple,100,2,2013/02/14,NULL
C,orange,100,2,2013/02/14,2013/01/31
$ more oth5.csv
Customer%0,Product,Amount,Gender,Date,PreviousDate
A,apple,100,1,2013/01/04,2013/01/01
A,milk,350,1,2013/04/04,2011/05/06
B,orange,100,2,2012/11/11,2011/12/12
B,orange,100,2,2013/05/30,2012/11/11
B,pineapple,500,2,2013/04/15,2013/04/01
B,wine,1000,2,2012/12/24,2011/12/24
D,orange,100,2,2011/10/28,NULL
```

Related Commands

msel : Select records with more complex criteria.

mcommon : When selecting a large number of target strings use `mcommon` command.

4.57 msep - Partition Records

Define the file name and location the save the separate records at the `d=` parameter. Since it is possible to embed the `${fieldname}` in the file name specified, as a result, the data can be split into separate files for corresponding fields. For example, the argument `d=./out/${date}.csv` specifies the `out` directory in the current directory, and the file is created from corresponding values in the `date` field.

Format

`msep d= [-p] [f=]` [i=] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`d=`	Specify the field name used for splitting to different data files. String specify here will be added as file name for each record. Embed field name as `${field name}`.
`-p`	Create the new directory name specify at the `d=` parameter which does not currently exist.

Examples

Example 1: Basic Example

Create a directory named `dat`, and separate the data according to the `date` value in the directory.

```
$ more dat1.csv
item,date,quantity,price
A,20081201,1,10
B,20081201,4,40
A,20081202,2,20
A,20081203,3,30
B,20081203,5,50
$ msep d='./dat/${date}.csv' -p i=dat1.csv
#END# kgsep -p d=./dat/${date}.csv i=dat1.csv
$ ls ./dat
20081201.csv
20081202.csv
20081203.csv
$ more ./dat/20081201.csv
item,date%0,quantity,price
A,20081201,1,10
B,20081201,4,40
$ more ./dat/20081202.csv
item,date%0,quantity,price
A,20081202,2,20
$ more ./dat/20081203.csv
item,date%0,quantity,price
A,20081203,3,30
B,20081203,5,50
$ more ./dat/B.csv
./dat/B.csv: No such file or directory
```

Related Commands

msep2 : While the functionality is similar with `msep`, serial number is used to name the output file, and prints a list of corresponding key and file name to a separate file.

mcat : This command restore and merge all the partitioned files by `sep` to the original file.

4.58 msep2 - Separate Records And Return Fields Table

Separate the data according to the value of the field(s) specified in `k=`. Partitioned data is automatically stored in numbered file name sequence. A table is created with the list of keys specified at `k=` and the corresponding file name for each key.

Format

`msep2 k= O= a= [-p]` [i=] [o=] [-assert_nullkey] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`k=`	List of field name(s) as unit of division.
`O=`	Create list of sequentially numbered file (serial number starting from 0) in the specified directory.
`o=`	Correspondence table with sequentially numbered file names and values specified as key at `k=` is output as CSV file. Output is printed to standard output if this parameter is not specified.
`a=`	Field name of the path of output specified at `o=`.
`-p`	Force create directory specified by pathname at `O=`.

Examples

Example 1: Basic Example

Split the data by corresponding values in `item` field. Output file names are sequential numbers starting from 0. The key and corresponding number is printed to `table.csv`.

```
$ more dat1.csv
item,no
A,1
A,1
A,2
B,1
B,2
$ msep2 k=item O=./output a=fileName o=table.csv i=dat1.csv
#END# kgsep2 O=./output a=fileName i=dat1.csv k=item o=table.csv
$ ls ./output
0
1
$ more table.csv
item%0,fileName
A,./output/0
B,./output/1
$ more output/0
item%0,no
A,1
A,1
A,2
$ more output/1
item%0,no
B,1
B,2
```

Example 2: Multiple key fields

Each file name is created according to the sequential number using `item,no` as the composite key field. The key field and its corresponding sequential file names are printed to `table.csv`.

```
$ more dat1.csv
item,no
A,1
A,1
```

```
A,2
B,1
B,2
$ msep2 k=item,no O=./output2 a=fileName o=table.csv i=dat1.csv
#END# kgsep2 O=./output2 a=fileName i=dat1.csv k=item,no o=table.csv
$ ls ./output2
0
1
2
3
$ more table.csv
item%0,no%1,fileName
A,1,./output2/0
A,2,./output2/1
B,1,./output2/2
B,2,./output2/3
$ more output/0
item%0,no
A,1
A,1
A,2
```

Related Command

msep : Use this command to include the field header name in file name.

4.59 msetstr - Add String Column

Add specified string as new field(s) in all rows. More than one fields can be added.

Format

`msetstr v= a=` [i=] [o=] [-assert_diffSize] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`v=`	List of character strings to add. NULL value is added if the value is not specified.
`a=`	Add field name. The number of string and field name of specified at `v=` must be the same number. `v=`.

Examples

Example 1: Basic Example

Calculate the date by setting a reference date (defined as January 01, 2007) and add the string " `20070101` " in all lines and save the output as a new column named " ReferenceDate ".

```
$ more dat1.csv
customer,date
A,20081202
A,20081204
B,20081203
$ msetstr v=20070101 a=ReferenceDate i=dat1.csv o=rsl1.csv
#END# kgsetstr a=ReferenceDate i=dat1.csv o=rsl1.csv v=20070101
$ more rsl1.csv
customer,date,ReferenceDate
A,20081202,20070101
A,20081204,20070101
B,20081203,20070101
```

Example 2: Add multiple fields

```
$ msetstr v=20070101,20070201 a=RefDate1,RefDate2 i=dat1.csv o=rsl2.csv
#END# kgsetstr a=RefDate1,RefDate2 i=dat1.csv o=rsl2.csv v=20070101,20070201
$ more rsl2.csv
customer,date,RefDate1,RefDate2
A,20081202,20070101,20070201
A,20081204,20070101,20070201
B,20081203,20070101,20070201
```

Example 3: Add column with null values

```
$ msetstr v= a=NewColumn i=dat1.csv o=rsl3.csv
#END# kgsetstr a=NewColumn i=dat1.csv o=rsl3.csv v=
$ more rsl3.csv
customer,date,NewColumn
A,20081202,
A,20081204,
B,20081203,
```

Related Command

mcal : Use the `if` function to add a fixed string according to differently conditions for each row.

4.60 mshare - Calculate Composition Ratio

Calculate the composition ratio of the fields specified in `f=`, and add results as a new field.

Format

`mshare f= [k=]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Calculate share value of field(s) (multiple fields can be specified) specified here. Specify the new field name using : (colon). Example: f=Quantity:volume share.
`k=`	Specify the list of field name(s) (multiple items can be specified) as the unit to calculate share. If the key is omitted, all lines are assumed to have the same key value.

Examples

Example 1: Basic Example

Calculate the share of "quantity" and "amount" fields for each "customer". Save the output in columns "volume share" and "share amount ".

```
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ mshare k=customer f=quantity:qtyShare,amount:amountShare i=dat1.csv o=rsl1.csv
#END# kgshare f=quantity:qtyShare,amount:amountShare i=dat1.csv k=customer o=rsl1.csv
$ more rsl1.csv
customer%0,quantity,amount,qtyShare,amountShare
A,1,10,0.3333333333,0.3333333333
A,2,20,0.6666666667,0.6666666667
B,1,15,0.2,0.3333333333
B,3,10,0.6,0.2222222222
B,1,20,0.2,0.4444444444
```

Related Command

4.61 mshuffle Partition records

This command partitions the input file into multiple files. The number of files is determined by the hash value of the field specified by the `f=` parameter. With the number of divisions (hash size) being n, the hash value of the field value v specified by `f=` is calculated as the remainder of n ($v \bmod n$). The field value is calculated as the total value of the character codes in bytes, with the data considered as character strings. When `f=` is not specified, the row number is used as the field value. Then, each row is output to the file whose name contains the obtained hash value. This scheme ensures that the rows containing the same field data are output to the same file. You can assign multiple hash values to the partitioned files by using the `v=` parameter to specify a weight. For instance, specify `n=3,v=2,1,3`. The hash size will be 6, which is the total of the weights ($2 + 1 + 3 = 6$). Two hash values (0 and 1) will be output to partitioned file 0, one hash value (2) to partitioned file 1, and three hash values (3, 4, and 5) to partitioned file 2. Note that the weight refers to the number of hash value allocations, not to the number of output rows.

Format

```
mshuffle n=|v= d= [f=] [i=] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [--help] [--helpl] [--version]
```

Parameters

`d=`	Specify the prefix to the output filename. The actual output filename will be the value specified here + a serial number (hash value).
`f=`	Specify the key used as the unit of partitioning. Those that have the same values in the field specified here are output to the same file.
`n=`	Specify the number of partitioned files.
`v=`	Specify the weight of the data amount for each of the partitioned files.

Examples

Example 1: Example 1: Basic example

The input file is partitioned into two, so that the rows are output to the same file as long as the value of the specified field (customer) is the same.

```
$ more dat2.csv
customer,date,amount
A,20081201,10
A,20081207,20
A,20081213,30
B,20081002,40
B,20081209,50
C,20081003,60
C,20081219,20
$ mshuffle f=customer d=./dat/d n=2 i=dat2.csv
#END# kgshuffle d=./dat/d f=customer i=dat2.csv n=2
$ ls ./dat
d_0
d_1
$ more ./dat/d_0
customer,date,amount
B,20081002,40
B,20081209,50
$ more ./dat/d_1
customer,date,amount
A,20081201,10
A,20081207,20
A,20081213,30
C,20081003,60
C,20081219,20
```

Example 2: Example 2: Not specifying f=

The f= parameter is not specified and the input file is partitioned into two. As row number hash values are used, the two files will have almost the same numbers of rows.

```
$ more dat2.csv
customer,date,amount
A,20081201,10
A,20081207,20
A,20081213,30
B,20081002,40
B,20081209,50
C,20081003,60
C,20081219,20
$ mshuffle d=./dat/d n=2 i=dat2.csv
#END# kgshuffle d=./dat/d i=dat2.csv n=2
$ ls ./dat
d_0
d_1
$ more ./dat/d_0
customer,date,amount
A,20081207,20
B,20081002,40
C,20081003,60
$ more ./dat/d_1
customer,date,amount
A,20081201,10
A,20081213,30
B,20081209,50
C,20081219,20
```

Example 3: Example 3: Specifying v=,f=

With v=2,1 specified, the input file is partitioned into two. Two hash values are allocated to file 0 (d_0) and one hash value is allocated to file 1 (d_1).

```
$ more dat2.csv
customer,date,amount
A,20081201,10
A,20081207,20
A,20081213,30
B,20081002,40
B,20081209,50
C,20081003,60
C,20081219,20
$ mshuffle f=customer d=./dat/d v=2,1 i=dat2.csv
#END# kgshuffle d=./dat/d f=customer i=dat2.csv v=2,1
$ ls ./dat
d_0
d_1
$ more ./dat/d_0
customer,date,amount
B,20081002,40
B,20081209,50
C,20081003,60
C,20081219,20
$ more ./dat/d_1
customer,date,amount
A,20081201,10
A,20081207,20
A,20081213,30
```

Example 4: Example 4: Specifying v=

The script of Example 3 is run without the f= parameter specified. As row number hash values are used, the ratio of the numbers of output rows will be almost the same as the ratio of the weights.

```
$ more dat2.csv
customer,date,amount
A,20081201,10
A,20081207,20
A,20081213,30
```

```
B,20081002,40
B,20081209,50
C,20081003,60
C,20081219,20
$ mshuffle d=./dat/d v=2,1 i=dat2.csv
#END# kgshuffle d=./dat/d i=dat2.csv v=2,1
$ ls ./dat
d_0
d_1
$ more ./dat/d_0
customer,date,amount
A,20081201,10
A,20081213,30
B,20081002,40
C,20081003,60
C,20081219,20
$ more ./dat/d_1
customer,date,amount
A,20081207,20
B,20081209,50
```

Related Command

msep: Partitions records according to field values.

4.62 msim - Calculate Similarity Between Two Variables

Find out the degree of similarity between two variable fields (distance) at `f=` parameter, specify the degree of similarity (distance) function at `c=` parameter to derive the similarity matrix.

Format

`msim c= f= [a=] [k=] [n=] [-d]` [i=] [o=] [bufcount=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

`k=`	Field(s) (multiple items can be specified) specified here is used as the unit of calculation.
`f=`	Field names for the calculation of degree of similarities between two fields.
`c=`	Specify the similarity measure(s) (distance) (multiple fields can be specified). As shown in the example below, the field name of the similarity measure results can be defined by using a : (colon). If the name of field is not defined with colon, the type of degree of similarity (distance) is used as the field name. Example: `msim f=x,y,z c=pearson:Pearson product-moment correlation coefficient, euclid:Euclidean distance,cosine:Cosine` Similarity measure=`covar|ucovar|pearson|spearman|kendall|euclid|cosine|cityblock|hamming|chi|phi|jaccard|supportr|lift|confMax|confMin|yuleQ|yuleY|kappa|oddsRatio|convMax|convMin`
`a=`	Specify the field name that indicates the name of the two variables. Specify the two arguments with a comma. Field names `fld1,fld2` are used if `a=` is not defined.
`-d`	Output as diagonal matrix and upper triangular matrix. Only the lower triangular matrix of similarity matrix is shown if `-d` option is not specified, but both upper triangular matrix and diagonal matrix are shown by when `-d` option is specified.

Definition of similarity (distance)

Real vector

Definition of size for the degree of similarity (or distance) in relation to two real number vectors $\mathbf{x} = (x_1, x_2, \cdots, x_n), \mathbf{y} = (x_1, x_2, \cdots, x_n)$ is shown in Table 4.23.

0-1Vector

Take the value as 0 or 1, the definition of degree of similarity of two 0-1 vectors $\mathbf{a} = (a_1, a_2, \cdots, a_n), \mathbf{b} = (b_1, b_2, \cdots, b_n)$ is shown in Table 4.25. The f_{jk} symbols used in the table, the value of a_i, b_i is enumerated in different combinations of (0,1), and shown in Table 4.24.

Further, meaning of $P(\cdot)$ is shown below.

Examples

Example 1: Basic Example

Calculate the cosine and Pearson's product-moment correlation coefficient for the combination of two items among `x, y, z` fields.

```
$ more dat1.csv
x,y,z
14,0.17,-14
11,0.2,-1
32,0.15,-2
13,0.33,-2
```

Table 4.23: Summary of degree of similarity for real number vectors

Parameter value	Detail	Distance/similarity	Equation definition
covar	Covariance	Degree of similarity	$\frac{1}{n}\sum_{i=1}^{n} (x_i-\bar{x})(y_i-\bar{y})$
ucovar	Unbiased covariance	Degree of similarity	$\frac{1}{n-1}\sum_{i=1}^{n} (x_i-\bar{x})(y_i-\bar{y})$
pearson	Pearson's product-moment correlation coeff	Degree of similarity	$\frac{\frac{1}{n}\sum_{i=1}^{n} (x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} (x_i-\bar{x})^2}\sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i-\bar{y})^2}}$
spearman	Spearman's rank correlation coefficient	Degree of similarity	$\mathbf{x},\mathbf{y}$ Product-moment correlation coefficient is c
kendall	Kendall's rank correlation coefficient	Degree of similarity	$\frac{c-d}{\frac{1}{2}n(n-1)}^{Note:1,2}$
euclid	Euclidean distance (number)	Distance	$\sqrt{\sum_{i=1}^{n} (x_i-y_i)^2}$
cosine	Cosine	Degree of similarity	$\frac{\mathbf{x}\cdot\mathbf{y}}{\lvert\mathbf{x}\rvert\lvert\mathbf{y}\rvert} = \frac{\sum_{i=1}^{n} x_i\ y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$
cityblock	City block distance	Distance	$\sum_{i=1}^{n} \mid x_i-y_i \mid$
hamming	Hamming distance	Distance	$\mid \{i \mid x_i \neq y_i, i=1,2,\cdots,n\} \mid$

Note 1: $c = |\{(i,j)|(x_i > x_j \text{ and } y_i > y_j) \text{ or } (x_i < x_j \text{ and } y_i < y_j), i > j, i = 1,2,\cdots,n, j = 1,2,\cdots,n\}|$

Note 2: $d = |\{(i,j)|(x_i > x_j \text{ and } y_i < y_j) \text{ or } (x_i < x_j \text{ and } y_i > y_j), i > j, i = 1,2,\cdots,n, j = 1,2,\cdots,n\}|$

Table 4.24: Combinations of the values of the 2 variables in 2 × 2 contingency table

	$b_i = 1$	$b_i = 0$	Total
$a_i = 1$	f_{11}	f_{10}	$f_{1.}$
$a_i = 0$	f_{01}	f_{00}	$f_{0.}$
Total	$f_{.1}$	$f_{.0}$	$f_{..}$

$P(a) = f_{1.}/f_{..}$

$P(b) = f_{.1}/f_{..}$

$P(\bar{a}) = f_{0.}/f_{..}$

$P(a,b) = f_{11}/f_{..}$

$P(a|b) = f_{11}/f_{.1}$

Table 4.25: Summary of degree of similarity for vector 0-1

Parameter values	Content	Distance/similarity	Equation	Range
chi	Chi-square value	Degree of similarity	$\sum_{i=0}^{1} \sum_{j=0}^{1} \frac{f_{ij}-e_{ij}}{e_{ij}}^{Note:1}$	$0 \sim \infty$
phi	Phi coefficient	Degree of similarity	$\frac{f_{11}f_{00}-f_{10}f_{01}}{\sqrt{f_{1.}f_{0.}f_{.1}f_{.0}}}$	$-1.0 \sim 1.0$
jaccard	Jack card factor	Degree of similarity	$\frac{P(a,b)}{P(a)+P(b)-P(a,b)}$	$0.0 \sim 1.0$
support	Support	Degree of similarity	$P(a,b)$	$0.0 \sim 1.0$
lift	Value of lift	Degree of similarity	$\frac{P(a,b)}{P(a)P(b)}$	$0 \sim \infty$
confMax	Maximum confidence	Degree of similarity	$\max(P(a\|b), P(b\|a))$	$0.0 \sim 1.0$
confMin	Minimum confidence	Degree of similarity	$\min((P(a\|b), P(b\|a))$	$0.0 \sim 1.0$
yuleQ	Ren correlation coefficient of yule (Q)	Degree of similarity	$\frac{\alpha-1}{\alpha+1}^{Note:2}$	$-1.0 \sim 1.0$
yuleY	Ren correlation coefficient of yule (Y)	Degree of similarity	$\frac{\sqrt{\alpha}-1}{\sqrt{\alpha}+1}^{Note:2}$	$-1.0 \sim 1.0$
kappa	kappa	Degree of similarity	$\frac{P(a,b)+P(\bar{a},\bar{b})-P(a)P(b)-P(\bar{a})P(\bar{b})}{1-P(a)P(b)-P(\bar{a})P(\bar{b})}$	$-1.0 \sim 1.0$
oddsRatio	oddsRatio	Degree of similarity	$\frac{P(a,b)P(\bar{a},\bar{b})}{P(a,\bar{b})P(\bar{a},b)}$	$0 \sim \infty$
convMax	Maximum conviction	Degree of similarity	$\max(\frac{P(a)P(\bar{b})}{P(a,\bar{b})}, \frac{P(\bar{a})P(b)}{P(\bar{a},b)})$	$0.5 \sim \infty$
convMin	Minimum conviction	Degree of similarity	$\min(\frac{P(a)P(\bar{b})}{P(a,\bar{b})}, \frac{P(\bar{a})P(b)}{P(\bar{a},b)})$	$0.5 \sim \infty$

Note 1: $e_{ij} = \frac{f_{i.}f_{.j}}{f_{..}}$ Note 2: $\alpha = \frac{f_{11}f_{00}}{f_{10}f_{01}}$

```
$ msim c=pearson,cosine f=x,y,z i=dat1.csv o=rsl1.csv
#END# kgsim c=pearson,cosine f=x,y,z i=dat1.csv o=rsl1.csv
$ more rsl1.csv
fld1,fld2,pearson,cosine
x,y,-0.5088704666,0.7860308044
x,z,0.1963041929,-0.5338153343
y,z,0.3311001423,-0.5524409416
```

Example 2: Output diagonal matrix, the upper triangular matrix

Calculate the cosine and Pearson's product-moment correlation coefficient for the combination of two items between `x, y, z` fields (with d option).

```
$ msim c=pearson,cosine f=x,y,z -d i=dat1.csv o=rsl2.csv
#END# kgsim -d c=pearson,cosine f=x,y,z i=dat1.csv o=rsl2.csv
$ more rsl2.csv
fld1,fld2,pearson,cosine
x,x,1,1
x,y,-0.5088704666,0.7860308044
x,z,0.1963041929,-0.5338153343
y,x,-0.5088704666,0.7860308044
y,y,1,1
y,z,0.3311001423,-0.5524409416
z,x,0.1963041929,-0.5338153343
z,y,0.3311001423,-0.5524409416
z,z,1,1
```

Example 3: Calculation based on key unit

Calculate using `key` field as unit.

```
$ more dat2.csv
key,x,y,z
A,14,0.17,-14
A,11,0.2,-1
A,32,0.15,-2
B,13,0.33,-2
B,10,0.8,-5
B,15,0.45,-9
$ msim k=key c=pearson,cosine f=x,y,z i=dat2.csv o=rsl3.csv
#END# kgsim c=pearson,cosine f=x,y,z i=dat2.csv k=key o=rsl3.csv
$ more rsl3.csv
key%0,fld1,fld2,pearson,cosine
A,x,y,-0.8746392857,0.8472573627
A,x,z,0.3164384831,-0.521983618
A,y,z,0.1830936883,-0.6719258683
B,x,y,-0.7919009884,0.8782575583
B,x,z,-0.471446429,-0.9051543403
B,y,z,-0.1651896746,-0.8514129252
```

Example 4: Specify the type of degree of similarity

Using the data with 01 values, compute the phi coefficient and Hamming distance.

```
$ more dat3.csv
x,y,z
1,1,0
1,0,1
1,0,1
0,1,1
$ msim c=hamming,phi f=x,y,z i=dat3.csv o=rsl4.csv
#END# kgsim c=hamming,phi f=x,y,z i=dat3.csv o=rsl4.csv
$ more rsl4.csv
fld1,fld2,hamming,phi
x,y,0.75,-0.5773502692
x,z,0.5,-0.3333333333
y,z,0.75,-0.5773502692
```

Example 5: Rename the column containing degree of similarity

Using the data with 01 values, compute the phi coefficient and Hamming distance and change the output field name.

```
$ msim c=hamming:HammingDist,phi:PhiCoeff a=variable1,variable2 f=x,y,z i=dat3.csv o=rsl5.csv
#END# kgsim a=variable1,variable2 c=hamming:HammingDist,phi:PhiCoeff f=x,y,z i=dat3.csv o=rsl5.csv
$ more rsl5.csv
variable1,variable2,HammingDist,PhiCoeff
x,y,0.75,-0.5773502692
x,z,0.5,-0.3333333333
y,z,0.75,-0.5773502692
```

Related Commands

mstats : Calculate the statistics of one variable.

mmvsim : Calculate sliding window similarity measure.

4.63 mslide - Slide Data Series

Shift data series in the specified column to a new column by specified number of times. For example, the function can be used to calculate difference between data items in the same field such as deriving the rate of return using daily stock price data (today's stock price/previous day's stock price)

Table 4.26 - 4.29 below highlights a commonly used example.

Table 4.26: input data

date	val
4/6	1
4/7	2
4/8	3
4/9	4

Table 4.27: f=val:nextVal

date	val	nextVal
4/6	1	2
4/7	2	3
4/8	3	4

Table 4.28: f=val:nextVal -r

date	val	nextVal
4/7	2	1
4/8	3	2
4/9	4	3

Table 4.29: f=val t=2

date	val	val1	val2
4/6	1	2	3
4/7	2	3	4

Table 4.26 shows the input data containing daily total values for four consecutive days. The figures could represent the changes in supermarket sales or stock price trends. Calculate the rate of increase between two days (for simplicity, assume "rate of increase = rate of day2/rate of day1").

The input data contains data series from 4/6 to 4/9. In Table 4.27, the data series is shifted up by one line and stored in a new column (newVal). The rate of increase is calculated from nextVal/val using mcal command. After the records have shifted, the last record in the input data dated 4/9 is not longer available. In this case, specify the `-n` option to print NULL values.

Table 4.27 shows the data output when sliding up the data series by one row, it is also possible to slide in reverse order by sliding the first row to a new column as shown in (Table 4.28).

Furthermore, `t=` allows user to specify the number of times to slide up or down. Table4.29 shows the result when t=2. The following function is the same as using mslide consecutively:

"`mslide f=val:val1` mslide f=val1:val2—".

When `t=` parameter is used, field names are created for each new column based on the field name specified at `f=` followed by an incremental value. When `t=` parameter is used with `-l` option, data from the initial column and the last column is displayed.

The functions of mwindow and mslide are similar. The difference between the two is that mslide is used for calculation between data items, and the command is often followed by mcal or msel. mwindow is used for calculation by row, which is usually followed by msum or mavg.

Format

`mslide f= [s=] [k=key] [t=] [-r] [-n] [-l] [i=] [o=]` [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

`f=`	Field name for sliding records. Multiple fields can be specified. The If you do not specify `t=`, the field name can be specified by f=fieldname:newfieldname.
`s=`	After the specified field is sorted (multiple fields can be specified), records are shifted. `s=` parameter is required when `-q` option is not specified.
`k=`	Specify the field for shifting of records.
`t=`	Number of times (rows) to shift. Default value is `t=1` if this parameter is not defined.
`-r`	Shift records in the opposite direction (shift the first record below).
`-n`	Print a null value if next (or previous) line is not available.
`-l`	Print results from the final shift.

Examples

Example 1: Basic Example

```
$ more dat1.csv
date,val
20130406,1
20130407,2
20130408,3
20130409,4
$ mslide s=date f=val:newVal i=dat1.csv o=rsl1.csv
#END# kgslide f=val:newVal i=dat1.csv o=rsl1.csv s=date
$ more rsl1.csv
date%0,val,newVal
20130406,1,2
20130407,2,3
20130408,3,4
```

Example 2: Slide rows in reverse direction

```
$ mslide s=date f=val:newVal -r i=dat1.csv o=rsl2.csv
#END# kgslide -r f=val:newVal i=dat1.csv o=rsl2.csv s=date
$ more rsl2.csv
date%0,val,newVal
20130407,2,1
20130408,3,2
20130409,4,3
```

Example 3: Slide records more than once

```
$ mslide s=date f=val t=2 i=dat1.csv o=rsl3.csv
#END# kgslide f=val i=dat1.csv o=rsl3.csv s=date t=2
$ more rsl3.csv
date%0,val,val1,val2
20130406,1,2,3
20130407,2,3,4
```

Example 4: Display output from the last column shifted

```
$ mslide s=date f=val t=2 -l i=dat1.csv o=rsl4.csv
#END# kgslide -l f=val i=dat1.csv o=rsl4.csv s=date t=2
$ more rsl4.csv
date%0,val,val2
20130406,1,3
20130407,2,4
```

Example 5: Change multiple field names

```
$ mslide s=date f=date:d_,val:v_ t=2 i=dat1.csv o=rsl5.csv
#END# kgslide f=date:d_,val:v_ i=dat1.csv o=rsl5.csv s=date t=2
$ more rsl5.csv
date%0,val,d_1,d_2,v_1,v_2
20130406,1,20130407,20130408,2,3
20130407,2,20130408,20130409,3,4
```

Related Command

4.64 msortf - Sort Records

Sort records according to the field defined at `f=` parameter.
This commands uses quicksort algorithm and it is not a stable sort (original order is retained for rows with same key value).

Format

`msortf f= [pways=] [maxlines=] [blocks=] [threadCnt=] [-noflg]` [i=] [o=] [-assert_diffSize] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Specify the column name where record values will be sorted accordingly. Four types of sequence order can be specified namely numeric, string, ascending, descending. Specify `%n` after the field name, followed by `n` or `r`. Character string ascending order:`field name` (`%` is not specified), character string descending order:`f=field%r`, numeric ascending order:`f=field%n`, numeric descending order:`f=field%nr`.
`-noflg`	Do not attach sorting marks (`|%0,%0n|`etc.) to the output CSV header.

Remarks

1. Character string fields specified at `k=` may not be sorted correct when `%n` is specified.
2. When `k=` is not specified, specify the files in merging order at `i=` (same as mcat).
3. When key field include NULL values, NULL value is treated as a value less than any value.
4. Field names of all input data specified at `i=` is assumed to have the same field names, whereas `mcat` has more flexibility in field names.

Examples

Example 1: Basic example

Sort by `item` and `date`.

```
$ more dat1.csv
item,date,quantity,price
B,20081201,4,40
A,20081201,10,200
A,20081201,10,100
B,20081203,5,50
B,20081201,2,500
A,20081201,3,300
$ msortf f=item,date i=dat1.csv o=rsl1.csv
#END# kgsortf f=item,date i=dat1.csv o=rsl1.csv
$ more rsl1.csv
item%0,date%1,quantity,price
A,20081201,10,200
A,20081201,10,100
A,20081201,3,300
B,20081201,4,40
B,20081201,2,500
B,20081203,5,50
```

Example 2: Sort by quantity in descending order and price in ascending order.

```
$ msortf f=quantity%nr,price%n i=dat1.csv o=rsl2.csv
#END# kgsortf f=quantity%nr,price%n i=dat1.csv o=rsl2.csv
$ more rsl2.csv
item,date,quantity%0nr,price%1n
A,20081201,10,100
A,20081201,10,200
B,20081203,5,50
B,20081201,4,40
A,20081201,3,300
B,20081201,2,500
```

Advanced parameters

`pways=`	Merge multiple files simultaneously ([2-100]:default 32) [Optional] Specify number of files to merge at a time while sorting multiple files.
`blocks=`	Number of buffer block ([1-1000]: default 100 1blk=400KB) [Optional] Specify memory size limit in the block size when sorting in memory. Maximum size for 1 block is × 4. Default = 400KB.
`maxlines=`	Row fetch limit of memory sort ([100-10,000,000]: 500,000 defaults) [Optional] Specify the maximum number of records sorted at once in memory. Set -block limit and -maxlines limit depending on the average size of record in the data.
`threadCnt=`	Number of threads to use when sorting in memory ([1-50] Default: 8) [Optional] Specify the number of threads for sorting through multi-threading function.

Notes on sorting order of CSV special characters

msortf interprets and sorts CSV special characters (e.g. comma and double quotes) differently than the sort command in UNIX. The data fields/columns are separated by comma character. For example, the values in the first column (f1) from the first row onwards are represented by the following ASCII characters: a -¿ (0x61) null -¿ (0x00) space -¿ (0x20) + -¿ (0x2b) - -¿ (0x2d) , -¿ (0x2c) " -¿ (0x22) Comma and double quotes is treated as special characters in CSV is enclosed in double quotes. For ease of illustration, "x" is populated in the second column f2 for all records as follows.

```
[baselinestretch=0.7,frame=single,fontsize=\small]
-----------------------------------------------
f1,f2
a,x
,x
 ,x
+,x
-,x
",",x
"""",x
-----------------------------------------------
```

The statement `"msortf f=f1"` sorts the data as follows. The sort order for CSV format special characters (null, space, double quotation, +, comma,-, a) is explained.

```
-----------------------------------------------
f1,f2
,x
 ,x
"""",x
+,x
",",x
-,x
a,x
-----------------------------------------------
```

Benchmark Test

The benchmark test described here shows the performance of msort and msortf. The input data consist of 6 fields and all data values are uniform random numbers.

```
-----------------------------------------------
key,fld1,fld2,fld3,fld4,fldn
95547922,162,159,192,118,74
81438069,138,157,155,122,58
26885062,129,199,133,198,75
32651684,180,107,123,170,-14
10245631,164,103,159,154,-63
15145156,182,191,175,107,-60
29254245,188,185,129,124,5
85423170,116,164,175,113,57
55155879,105,163,195,167,25
66997216,195,139,195,113,39
.
.
-----------------------------------------------
```

Compare number of key types and values

The sample data size is 1 million, the following table shows the results according to variation in types of key values at 2,10,100,1000,10000. Data in the "random number" column is generated using the maximum limit of the random number as key. Data is sorted according to the values in "random number ascending / descending order" column before the benchmark test. The comparison table shows the processing results of msort, MUSASHI xtsort command, and UNIX sort command against the msortf command. The sort command sort one or more sort keys extracted from each line of input, whereas "sort -k1" sorts data on the first column. The last 3 rows of the table show the result of msortf, xtsort and sort sorted on numeric value stored in the first key field. ※ Input data size: about 28MB.
※ Unit: seconds. Measurement in real time from beginning to end of program using the time command.
※ Environment: iMac, Mac OS X 10.5 Leopard, 2.8GHz Intel Core 2 Duo, 4GB memory

Table 4.30: Comparison of types of key values and its condition among various sort commands

No.	Command	2 Types	10 Types	100 Types	1000 Types	10000 Types	Rand	Rand Asc	Rand Desc
(1)	msortf f=key	0.29	0.33	0.37	0.40	0.43	0.50	0.29	0.28
(2)	xtsort -k key	1.25	1.24	1.22	1.20	1.19	1.12	0.85	1.00
(3)	sort -k1	16.96	16.63	16.05	15.56	15.08	13.68	6.85	7.13
(4)	msortf f=key%n	0.46	0.56	0.65	0.72	0.79	1.02	0.59	0.59
(5)	xtsort -k key%n	2.52	2.72	2.96	3.16	3.21	3.22	2.31	2.32
(6)	sort -k1 -n	16.65	14.52	11.54	8.56	5.71	0.95	0.33	0.36

msortf is 2 to 5 times faster than xtsort. In relation to sort, it can be more than ten times faster depending on the conditions. This command uses the exactly the same quick sort algorithm as in MUSASHI, however, in MCMD multi-threading is used for the parallel processing of sort in separate threads. The impact of the difference is shown.

Next, the experiment shows the change in speed of character string sorting from 1 million records to 10 million records given the number of key types is set as 100 and the maximum value. The comparison of the two commands msortf and xtsort is shown in Figure 4.5, 4.6.

Related Command

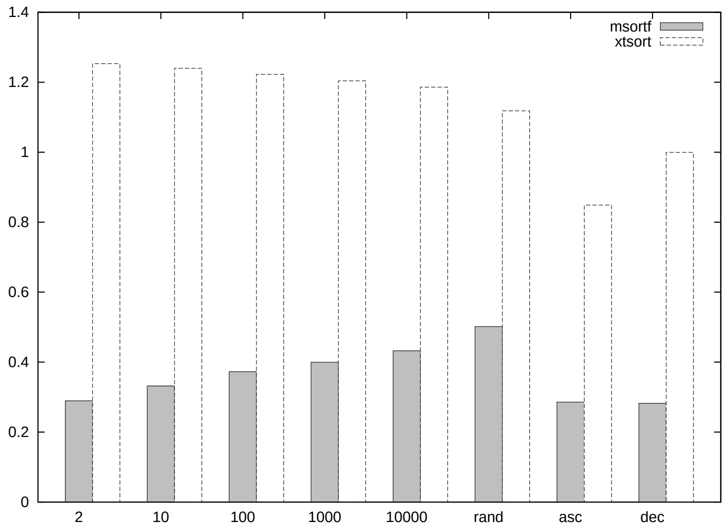


Figure 4.3: Compare sort results on character strings with msort, msortf, xtsort on various key types. (x-axis: number of the key types, y-axis: seconds)

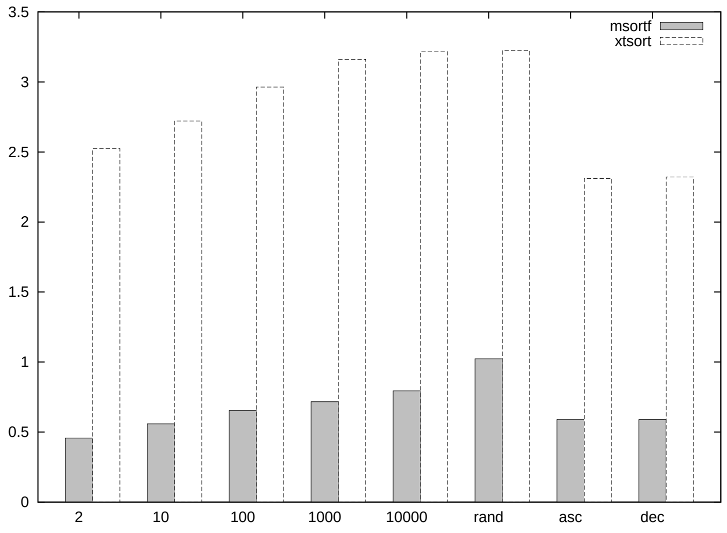


Figure 4.4: Compare sort results on numerical values with msortf,xtsort on various key types (x-axis: number of key types, y-axis: seconds)

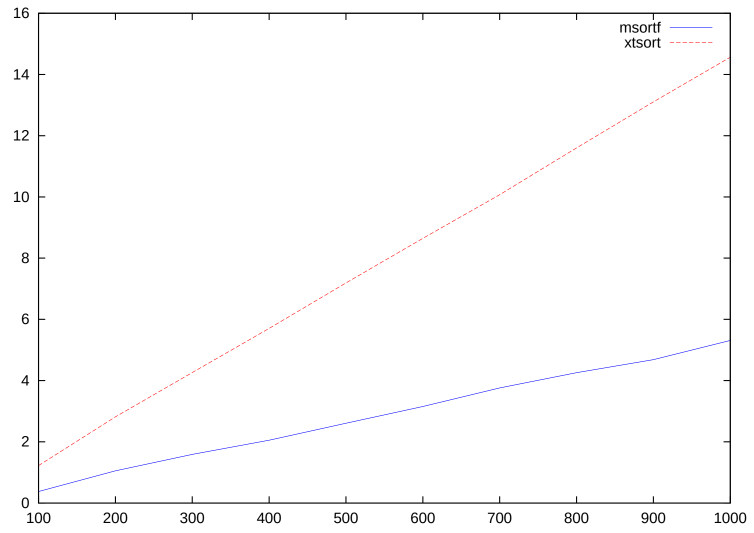


Figure 4.5: Sorting results with 100 key types (x-axis: number of records, y-axis: seconds)

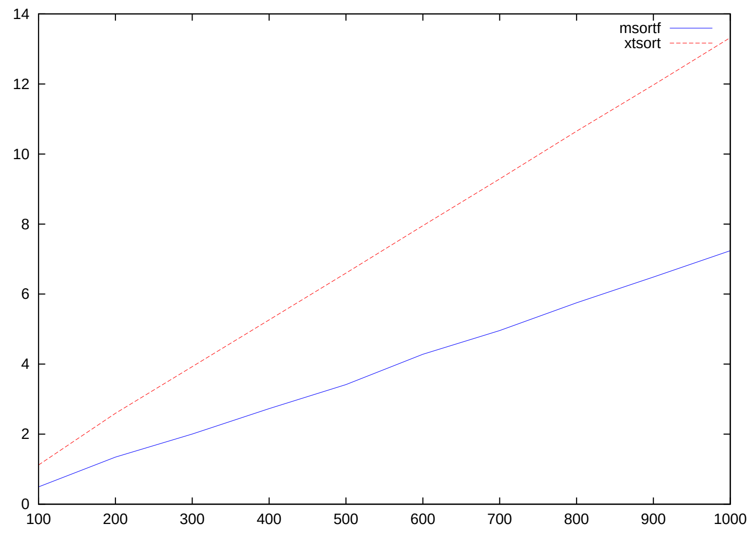


Figure 4.6: Sorting results with different key types using random number (maximum) (x-axis: number of records, y-axis: seconds)

4.65 msplit Partitioning fields by delimiters

This command partitions fields by delimiters.

Format

`msplit f= a= [-r] [delim=]` [i=] [o=] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Specify the fieldname of the data to be partitioned by delimiters.
`a=`	Specify new fields. The data will be partitioned into the number of fieldnames specified here. If the data falls short, the values will be NULL.
`delim=`	Specify a new delimiter. The default is the en space.
`-r`	Use this option to remove the field specified by f=.

Examples

Example 1: Example 1: Basic example

Fields are partitioned by en spaces.

```
$ more dat1.csv
id,data
A,1 10 2
A,2 20 3
B,1 15 5
B,3 10 4
B,1 20 6
$ msplit f=data a=d1,d2,d3 i=dat1.csv o=rsl1.csv
#END# kgsplit a=d1,d2,d3 f=data i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,data,d1,d2,d3
A,1 10 2,1,10,2
A,2 20 3,2,20,3
B,1 15 5,1,15,5
B,3 10 4,3,10,4
B,1 20 6,1,20,6
```

Example 2: Example 2: Using -r

The `-r` option is specified to delete the fields specified by the `f=` parameter.

```
$ msplit f=data a=d1,d2,d3 -r i=dat1.csv o=rsl2.csv
#END# kgsplit -r a=d1,d2,d3 f=data i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,d1,d2,d3
A,1,10,2
A,2,20,3
B,1,15,5
B,3,10,4
B,1,20,6
```

Example 3: Example 3: Unmatched partition counts

If the partitionable fields are fewer than the number of fields specified by the `a=` parameter, NULL will be added. If they are more than the number of fields specified by the `a=` parameter, they are output from the beginning until the specified number of partitions is reached.

```
$ more dat2.csv
id,data
A,1 10 2
A,2 20 3
B,1 15 5
B,3 4
B,1
$ msplit f=data a=d1,d2 i=dat2.csv o=rsl3.csv
#END# kgsplit a=d1,d2 f=data i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,data,d1,d2
A,1 10 2,1,10
A,2 20 3,2,20
B,1 15 5,1,15
B,3 4,3,4
B,1,1,
```

Example 4: Example 4: Specifying delim

The `delim=` parameter is used to partition fields by a character other than the en space.

```
$ more dat3.csv
id,data
A,1_10_3
A,2_20_5
B,1_15_6
B,3_10_7
B,1_20_8
$ msplit f=data a=d1,d2,d3 delim=_ i=dat3.csv o=rsl4.csv
#END# kgsplit a=d1,d2,d3 delim=_ f=data i=dat3.csv o=rsl4.csv
$ more rsl4.csv
id,data,d1,d2,d3
A,1_10_3,1,10,3
A,2_20_5,2,20,5
B,1_15_6,1,15,6
B,3_10_7,3,10,7
B,1_20_8,1,20,8
```

Related Commands

4.66 mstats - Calculate Statistics of 1 Variable

Specify the numeric fields in the parameter `f=`, and calculate the statistics specified in the parameter `c=`. Specify the aggregate key unit at `k=`. NULL value in the specified field(s) at `f=` are ignored. However, if all records include NULL values, NULL values will be included in the output.

Format

`mstats c= f= [k=] [-n]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

`k=`	Compute aggregate statistics on the key field(s) specified (multiple fields can be specified).
`f=`	Fields for which statistics are computed (multiple fields can be specified).

`c=`	Statistics (select one from the list below)
	`sum|mean|count|ucount|devsq|var|uvar|sd|usd|USD|cv|min|qtile1|`
	`median|qtile3|max|range|qrange|mode|skew|uskew|kurt|ukurt`

`-n`	Flag to be NULL if null data.

List of statistics

Examples

Example 1: Basic

Calculate the statistical sum of "quantity" and "amount" field for each "customer".

```
$ more dat1.csv
customer,quantity,amount
A,1,10
B,5,20
B,2,10
C,1,15
C,3,10
C,1,21
$ mstats k=customer f=quantity,amount c=sum i=dat1.csv o=rsl1.csv
#END# kgstats c=sum f=quantity,amount i=dat1.csv k=customer o=rsl1.csv
$ more rsl1.csv
customer%0,quantity,amount
A,1,10
B,7,30
C,5,46
```

Example 2: Basic Example 2

Calculate the statistical maximum value.

```
$ mstats k=customer f=quantity,amount c=max i=dat1.csv o=rsl2.csv
#END# kgstats c=max f=quantity,amount i=dat1.csv k=customer o=rsl2.csv
$ more rsl2.csv
customer%0,quantity,amount
A,1,10
B,5,20
C,3,21
```

Related Commands

msim : Find out the bivariate statistics.

mavg : Commands specific to`c=avg`.

msum : Commands specific to `c=sum`.

mcount : Unlike `c=count`, this count the number of rows for each aggregate key.

Value of `c=`	Description	Equation	Remarks
count	Count (Except NULL value)	n: Number of non-NULL records	It can not be applied to character string field.
ucount	Unique count	un: Number of duplicate values removed	It can not be applied to character string field.
sum	Total	$sum = \sum_{i=1}^{n} x_i$	
mean	Arithmetic mean	$m = \frac{1}{n}\sum_{i=1}^{n} x_i$	
devsq	Sum of squared deviation	$S = \sum_{i=1}^{n}(x_i - m)^2$	
var	Variance	$s^2 = \frac{1}{n}S$	
uvar	Variance (unbiased estimate)	$u^2 = \frac{1}{n-1}S$	
sd	Standard deviation	$s = \sqrt{s^2}$	
usd	Standard deviation (unbiased variance)	$u = \sqrt{u^2}$	commonly used standard deviation
USD	Unbiased standard deviation	Omission	Accurate unbiased estimation
cv	Coefficient of variation	$cv = s/mx100\%$	
mode	Mode	$mode$: Most frequent value	Print the value of the smaller value if the frequency is same Print NULL if values are different.
min	Minimum value	$min = \min_i x_i$	
max	Maximum value	$max = \max_i x_i$	
range	Range	$r = max - min$	
median	Median	$Q2 = Secondquartilewhensortedinascendingorder$	
qtile1	First quartile	$Q1 = Firstquartilewhensortedinascendingorder$	
qtile3	Third quartile	$Q3 = Thirdquartilewhensortedinascendingorder$	
qrange	Interquartile range	$rq = Q3 - Q1$	
skew	Skewness	$\frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-m)^3}{s^3}$	
uskew	Skewness (unbiased estimate)	omitted	
kurt	Kurtosis	$\frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-m)^4}{s^4} - 3.0$	
ukurt	Kurtosis (unbiased estimated)	omitted	

4.67 msum - Sum of Column

Aggregate the sum of values in the records at the specified field defined at `f=` parameter for records with the same key value defined at `k=`.

Format

`msum f= [k=] [-n]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

`k=`	Specify list of field name(s) (multiple fields can be specified) as aggregate unit.
`f=`	Aggregate the values specified at the field(s) (multiple items can be specified). Records with NULL values are ignored.
`-n`	If NULL exist in the field defined at `f=`, the result will return NULL.

Examples

Example 1: Basic Example

Calculate the total value of "quantity" and "amount" for each "customer". Save the output with field names "total quantity" and "total amount".

```
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ msum k=customer f=quantity:quantitySum,amount:amountSum i=dat1.csv o=rsl1.csv
#END# kgsum f=quantity:quantitySum,amount:amountSum i=dat1.csv k=customer o=rsl1.csv
$ more rsl1.csv
customer%0,quantitySum,amountSum
A,3,30
B,5,45
```

Related Commands

mhashsum : Compute sum without sorting by aggregate key in advance.

mavg : Compute average.

mstats : Compute a variety of statistics.

4.68 msummary Calculate Statistics for 1 Variable

Calculate the type of statistics specified at `c=` parameter for fields specified at `f=` parameter.

Format

`msummary c= f= [a=] [k=] [-n]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

`k=`	Compute statistics based on the key field(s) specified (multiple fields can be specified).
`f=`	Field lists for computation of statistical summary (multiple fields can be specified). When `-x`,`-nfn` option is specified, specify the field number (0).
`c=`	Statistics (multiple fields can be specified) Specify list of statistics delimited by comma. Statistics list: `sum/mean/count/ucount/devsq/var/uvar/sd/usd/cv/min/qtile1/median/qtile3/max/range/qrange/mode/skew/uskew/kurt/ukurt`
`-a`	New column name. Results from calculation on column(s) specified at `f=` parameter (default is fld).

List of Statistics

The list of statistics specified at `c=` parameter is shown in Table 4.31.

Examples

Example 1: Basic Example

Find out the median and average "quantity" and "amount" by each customer. Save the output in a new column named "type".

```
$ more dat1.csv
customer,quantity,amount
A,1,10
A,2,20
B,1,15
B,3,10
B,1,20
$ msummary k=customer f=quantity,amount c=median:medianval,mean:meanval a=type i=dat1.csv o=rsl1.csv
#END# kgsummary a=type c=median:medianval,mean:meanval f=quantity,amount i=dat1.csv k=customer o=rsl1.csv
$ more rsl1.csv
customer%0,type,medianval,meanval
A,quantity,1.5,1.5
A,amount,15,15
B,quantity,1,1.666666667
B,amount,15,15
```

Related Commands

mstats : Compute one type of statistics.

Table 4.31: List of Statistics

Value of `c=`	Description	Equation	Remarks
count	Count (Except NULL value)	n: Number of non-NULL records	It can not be applied to character string field.
ucount	Unique count	un: Number of duplicate values removed	It can not be applied to character string field.
sum	Total	$sum = \sum_{i=1}^{n} x_i$	
mean	Arithmetic mean	$m = \frac{1}{n}\sum_{i=1}^{n} x_i$	
devsq	Sum of squared deviation	$S = \sum_{i=1}^{n}(x_i - m)^2$	
var	Variance	$s^2 = \frac{1}{n}S$	
uvar	Variance (unbiased estimate)	$u^2 = \frac{1}{n-1}S$	
sd	Standard deviation	$s = \sqrt{s^2}$	
usd	Standard deviation (sort of unbiased variance)	$u = \sqrt{u^2}$	commonly used standard deviation
cv	Coefficient of variation	$cv = s/m100\%$	
mode	Mode	$mode$: Most frequent value	Print the value of the smaller value if the frequency is same.
min	Minimum value	$min = \min_i x_i$	
max	Maximum value	$max = \max_i x_i$	
range	Range	$r = max - min$	
median	Median	$Q2 = Secondquartilewhensortedinascendingorder$	
qtile1	First quartile	$Q1 = Firstquartilewhensortedinascendingorder$	
qtile3	Third quartile	$Q3 = Thirdquartilewhensortedinascendingorder$	
qrange	Interquartile range	$rq = Q3 - Q1$	
skew	Skewness	$\frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-m)^3}{s^3}$	
uskew	Skewness (unbiased estimate)	omitted	
kurt	Kurtosis	$\frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-m)^4}{s^4} - 3.0$	
ukurt	Kurtosis (unbiased estimate)	omitted	

4.69 mtab2csv Convert TSV data into CSV data

This command converts tab-separated values into comma-separated values. The data in the source text file does not have to be tab-separated; use the `d=` parameter to specify a delimiter other than the tab. If the numbers of fields differ before and after the conversion, the data is output until the immediately preceding row and the command abends.

Format

`mtab2csv [d=] [-r]` [i=] [o=] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`d=`	Specify a delimiter. (Only a single-byte character can be specified.)
`-r`	Use this option to remove carriage return codes (`CR:\r`). When handling CSV data, MCMD recognizes only LF(`\n`) as the line feed code. If the CSV data contains Windows text return codes CR+LF(`\r\n`) or Mac text return codes CR(`\r`), conversion can result in an error because MCMD handles them as mere characters. This option is for avoiding this issue.

Examples

Example 1: Example 1: Basic example

TSV data is converted into CSV data.

```
$ more dat1.tab
id        data        data2
A        1102        a
A        2203        aaa
B        1155        bbbb
B        3104        c
B        1206        de
$ mtab2csv i=dat1.tab o=rsl1.csv
#END# kgtab2csv i=dat1.tab o=rsl1.csv
$ more rsl1.csv
id,data,data2
A,1102,a
A,2203,aaa
B,1155,bbbb
B,3104,c
B,1206,de
```

Example 2: Example 2: Specifying d=

The `d=` parameter is specified to use a delimiter other than the tab.

```
$ more dat2.bar
id-data-data2
A-1102-a
A-2203-aaa
B-1155-bbbb
B-3104-c
B-1206-de
$ mtab2csv d=- i=dat2.bar o=rsl2.csv
#END# kgtab2csv d=- i=dat2.bar o=rsl2.csv
$ more rsl2.csv
id,data,data2
A,1102,a
A,2203,aaa
B,1155,bbbb
```

```
B,3104,c
B,1206,de
```

Related Commands

mxml2csv: Converts XML data into CSV data. msplit: Partitions fields by delimiters.

4.70 mtee - Copy to Multiple Output Files

Contents of input data are copied directly to multiple files and standard output.

Format

`mtee [o=] [-nostdout]` [i=] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`o=`	Output file names. The contents in the input file is duplicated into multiple files. When this parameter is not defined, `mtee` copies input to standard output.
`-nostdout`	Copy to output file but not to standard output.

Examples

Example 1: Basic Examples

Copy `dat1.csv` file to two files `rsl1.csv` and `rsl2.csv` In addition, display output on screen through standard output.

```
$ more dat1.csv
customer,quantity,price
A,1,10
A,2,20
B,1,15
$ mtee i=dat1.csv o=rsl1.csv,rsl2.csv
customer,quantity,price
A,1,10
A,2,20
B,1,15
#END# kgtee i=dat1.csv o=rsl1.csv,rsl2.csv
$ more rsl1.csv
customer,quantity,price
A,1,10
A,2,20
B,1,15
$ more rsl2.csv
customer,quantity,price
A,1,10
A,2,20
B,1,15
```

Example 2: Do not print to standard output

When `-nostdout` is specified, the command only copy the two files `rsl1.csv` and `rsl2.csv` but not to standard output.

```
$ mtee i=dat1.csv o=rsl1.csv,rsl2,csv -nostdout
#END# kgtee -nostdout i=dat1.csv o=rsl1.csv,rsl2,csv
```

Related Command

4.71 mtonull - Substitute Value for NULL

Specify the target field parameter at `f=` , substitute the value at the `v=` parameter with NULL in the data. Select the methods for finding patterns using exact string matching (the default) or substring matching (`-sub` option).

Format

`mtonull f= v= [-sub] [-W]` [i=] [o=] [-assert_diffSize] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	specify list of field name(s) (multiple fields can be specified) where the values are replaced .
`v=`	Specify the character string value (multiple items can be specified) for matching the field defined at `f=` parameter. Replace the matched values with NULL.

Options

`-sub`	Rather than matching the exact string, compare character substring with the values in the field specified in the `f=` parameter and replace the value defined in the `v=` parameter containing the substring with NULL value.
`-W`	Substring match for wide character strings when the `-sub` option is specified.

Examples

Example 1: Basic Example

Replace 0 with NULL value in columns `quantity` and `price`.

```
$ more dat1.csv
item,quantity,price
A,0,1
B,1,0
C,2,200
D,3,0
E,0,298
$ mtonull f=quantity,price v=0 i=dat1.csv o=rsl1.csv
#END# kgtonull f=quantity,price i=dat1.csv o=rsl1.csv v=0
$ more rsl1.csv
item,quantity,price
A,,1
B,1,
C,2,200
D,3,
E,,298
```

Example 2: Replace a specified number with NULL value

Replace 0 or 1 with NULL value in columns `quantity` and `price`.

```
$ mtonull f=quantity,price v=0,1 i=dat1.csv o=rsl2.csv
#END# kgtonull f=quantity,price i=dat1.csv o=rsl2.csv v=0,1
$ more rsl2.csv
item,quantity,price
A,,
B,,
C,2,200
```

```
D,3,
E,,298
```

Example 3: Substitute substring match

Replace with a NULL value where `quantity` and `price` columns contain 0.

```
$ mtonull -sub f=quantity,price v=0 i=dat1.csv o=rsl3.csv
#END# kgtonull -sub f=quantity,price i=dat1.csv o=rsl3.csv v=0
$ more rsl3.csv
item,quantity,price
A,,1
B,1,
C,2,
D,3,
E,,298
```

Example 4: Substitute character string

Replace the string in the `item` field that matches character string `apple, orange, pineapple` with NULL value.

```
$ more dat2.csv
item,price
fruit:apple,100
fruit:peach,250
fruit:grape,300
fruit:pineapple,450
fruit:orange,500
$ mtonull f=item v=apple,orange,pineapple -sub i=dat2.csv o=rsl4.csv
#END# kgtonull -sub f=item i=dat2.csv o=rsl4.csv v=apple,orange,pineapple
$ more rsl4.csv
item,price
,100
fruit:peach,250
fruit:grape,300
,450
,500
```

Related Command

mnullto : Reversely, replace NULL value with a character string.

4.72 mtra - Convert Vertical Data to Vector

Connect the items from the specified fields in `f=`, and save the string of items as a vector in a new column (referred to as transaction field). Specify the delimiter character of the items at `delim=`.

Format

`mtra f= [s=] [k=] [delim=] [-r]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`f=`	Specify field(s) (multiple fields can be specified) where the transaction fields are connected as an item. NULL values will not be included in the vector.
`k=`	Key field name(s) (multiple items can be specified) of the character string pattern. `-r` option cannot be specified with this parameter.
`delim=`	Specify the delimiting character (Space is used as the default delimiter).
`-r`	Reverse conversion Convert transaction items column to vertically structured data.

Examples

Example 1: Basic Example

Combine the corresponding `item` for each `customer` in a string using a space as the delimiter, and save output in the column named `transaction`.

```
$ more dat1.csv
customer,item
A,a
A,b
B,c
B,d
B,e
$ mtra k=customer f=item:transaction i=dat1.csv o=rsl1.csv
#END# kgtra f=item:transaction i=dat1.csv k=customer o=rsl1.csv
$ more rsl1.csv
customer%0,transaction
A,a b
B,c d e
```

Example 2: Use hyphen (-) as item delimiter

```
$ mtra k=customer f=item:transaction delim=- i=dat1.csv o=rsl2.csv
#END# kgtra delim=- f=item:transaction i=dat1.csv k=customer o=rsl2.csv
$ more rsl2.csv
customer%0,transaction
A,a-b
B,c-d-e
```

Example 3: Convert items in descending sort order

```
$ mtra k=customer s=item%r f=item:transaction i=dat1.csv o=rsl3.csv
#END# kgtra f=item:transaction i=dat1.csv k=customer o=rsl3.csv s=item%r
$ more rsl3.csv
customer%0,transaction
A,b a
B,e d c
```

Related Commands

mvsort : Vector based transaction data can be processed by a set of commands (with `mv` as prefix) which handles vector data.

mcross : Rather than converting as transaction data, every item is saved separately as individual field in the output.

mtrafld : Create transaction data using " field name=value ".

mtraflg : Create transaction data as item with field names.

4.73 mtrafld - Convert Transaction Field to Cross (pivot) Table

Create item pairs from the fields specified at `f=`, concatenate the item pairs and save as a new vector field (also referred to as transaction field).

Format

`mtrafld a= [f=] [delim=] [delim2=] [-r] [-valOnly]` [i=] [o=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`a=`	Specify the transaction field name.
`f=`	List of field name(s) (Multiple fields can be specified) [required when `-r` is specified, otherwise, this parameter is optional] The field names specified here will be created as connected items and saved in the transaction field. When -r option is specified, specify the field name to extract from the transaction data. This parameter is optional when `-r` option is specified. If the parameter is not specified, all field names are processed as value pairs. However, when`f=` is not defined, this command cannot read standard input (using pipe).
`delim=`	Specify the character to separate each transaction field item (default delimiter: space if this parameter is not defined).
`delim2=`	Specify the character to separate value pairs and field name (default character: =).
`-r`	Reverse conversion Convert transaction field to cross table.
`-valOnly`	When this option is specified, the item do not return the prefix "field name=" in the output.

Examples

Example 1: Basic Example

Join the fields `price` and `quantity` to a string, rename output field as `transaction`.

```
$ more dat1.csv
customer,price,quantity
A,198,1
B,325,2
C,200,3
D,450,2
E,100,1
$ mtrafld a=transaction f=price,quantity i=dat1.csv o=rsl1.csv
#END# kgtrafld a=transaction f=price,quantity i=dat1.csv o=rsl1.csv
$ more rsl1.csv
customer,transaction
A,price=198 quantity=1
B,price=325 quantity=2
C,price=200 quantity=3
D,price=450 quantity=2
E,price=100 quantity=1
```

Example 2: Basic Example 2

Use `-r` option to revert the output results back to the original data.

```
$ mtrafld -r a=transaction f=price,quantity i=rsl1.csv o=rsl2.csv
#END# kgtrafld -r a=transaction f=price,quantity i=rsl1.csv o=rsl2.csv
$ more rsl2.csv
customer,price,quantity
A,198,1
B,325,2
```

```
C,200,3
D,450,2
E,100,1
```

Example 3: Specify the delimiter

`Price` and `quantity` field is separated by " _ " (underscore) character and connected by 1 character string. Use colon and name the output field as `transaction`.

```
$ mtrafld a=transaction f=price,quantity delim=_ delim2=':' i=dat1.csv o=rsl3.csv
#END# kgtrafld a=transaction delim2=: delim=_ f=price,quantity i=dat1.csv o=rsl3.csv
$ more rsl3.csv
customer,transaction
A,price:198_quantity:1
B,price:325_quantity:2
C,price:200_quantity:3
D,price:450_quantity:2
E,price:100_quantity:1
```

Example 4: When data contains NULL value

```
$ more dat2.csv
customer,price,quantity
A,198,1
B,,2
C,200,
D,450,2
E,,
$ mtrafld a=transaction f=price,quantity i=dat2.csv o=rsl4.csv
#END# kgtrafld a=transaction f=price,quantity i=dat2.csv o=rsl4.csv
$ more rsl4.csv
customer,transaction
A,price=198 quantity=1
B,quantity=2
C,price=200
D,price=450 quantity=2
E,
```

Example 5: When data contains NULL value 2

Use `-r` option to revert the output results back to the original data.

```
$ mtrafld -r a=transaction f=price,quantity i=rsl4.csv o=rsl5.csv
#END# kgtrafld -r a=transaction f=price,quantity i=rsl4.csv o=rsl5.csv
$ more rsl5.csv
customer,price,quantity
A,198,1
B,,2
C,200,
D,450,2
E,,
```

Example 6: Specify -valOnly option

```
$ mtrafld -valOnly f=price,quantity a=transaction i=dat2.csv o=rsl6.csv
#END# kgtrafld -valOnly a=transaction f=price,quantity i=dat2.csv o=rsl6.csv
$ more rsl6.csv
customer,transaction
A,198 1
B,2
C,200
D,450 2
E,
```

Related Commands

mvsort : Vector based transaction data can be processed by a set of commands (with mv as prefix) which handles vector data.

mcross : Rather than converting as transaction data, every item is saved separately as individual field in the output.

mtra : Create transaction data using values in the field.

mtraflg : Create transaction data with field names.

4.74 mtraflg - Convert Cross (pivot) Table to Transaction Fields

Check whether the field(s) specified in the f= parameter contains NULL value. Fields with non-NULL values are connected as one item and saved as a new vector.

Format

`mtraflg a= f= [delim=] [-r]` [i=] [o=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`a=`	Specify the transaction field name.
`f=`	Check the values in the specified field name(s) (multiple fields can be specified) to create transaction data. (`-r` option is specified, extract list of values as the field name of the transaction data)
`delim=`	Specify the character to separate each transaction field item (Default character is space if this parameter is omitted). Character string should not be used. 1 byte character can be specified.
`-r`	Reverse conversion Convert transaction based data to vertically structured data.

Examples

Example 1: Basic Example

Create a string of vector from the list of non-null values in column `egg` and `milk`.

```
$ more dat1.csv
customer,egg,milk
A,1,1
B,,1
C,1,
D,1,1
$ mtraflg f=egg,milk a=transaction i=dat1.csv o=rsl1.csv
#END# kgtraflg a=transaction f=egg,milk i=dat1.csv o=rsl1.csv
$ more rsl1.csv
customer,transaction
A,egg milk
B,milk
C,egg
D,egg milk
```

Example 2: Basic Example 2

Use `-r` option to revert the output results back to the original data.

```
$ mtraflg -r f=egg,milk a=transaction i=rsl1.csv o=rsl2.csv
#END# kgtraflg -r a=transaction f=egg,milk i=rsl1.csv o=rsl2.csv
$ more rsl2.csv
customer,egg,milk
A,1,1
B,,1
C,1,
D,1,1
```

Example 3: Specify the delimiter

Combine items using the “ `-` ” (hyphen) as delimiter. Save output in column named `transaction`.

```
$ mtraflg f=egg,milk a=transaction delim=- i=dat1.csv o=rsl3.csv
#END# kgtraflg a=transaction delim=- f=egg,milk i=dat1.csv o=rsl3.csv
$ more rsl3.csv
customer,transaction
A,egg-milk
B,milk
C,egg
D,egg-milk
```

Related Commands

mvsort : Vector based transaction data can be processed by a set of commands (with `mv` as prefix) which handles vector data.

mcross : Rather than converting as transaction data, every item is saved separately as individual field in the output.

mtra : Create transaction data using values in the field.

mtrafld : Create transaction data with the format " field name=value ".

4.75 muniq - Unique Records

Remove duplicate values and create unique records.

Format

`muniq [k=]` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameter

`k=` Specify the field name (s) as the unique identifier of the records.

Examples

Example 1: Basic Example

Remove duplicate records in the `date` field.

```
$ more dat1.csv
date,customer
20081201,A
20081202,A
20081202,B
20081202,B
20081203,C
$ muniq k=date i=dat1.csv o=rsl1.csv
#END# kguniq i=dat1.csv k=date o=rsl1.csv
$ more rsl1.csv
date%0,customer
20081201,A
20081202,B
20081203,C
```

Example 2: Delete duplicate rows in multiple columns

Remove duplicate records based on unique values in `date` and `customer` field.

```
$ muniq k=date,customer i=dat1.csv o=rsl2.csv
#END# kguniq i=dat1.csv k=date,customer o=rsl2.csv
$ more rsl2.csv
date%0,customer%1
20081201,A
20081202,A
20081202,B
20081203,C
```

Related Command

mbest : Use `mbest` command to select the line number for records with the same key.

4.76 mvcat - Combine Vectors

Merge multiple vectors into one vector.

Examples are shown in Table 4.32, 4.33 and 4.34.

Table 4.32: Input data
`in.csv`

no	items1,items2
1	a c,b
2	a d,a e
3	b f,
4	e,e

Table 4.33: Basic example
`mvcat vf=item1,items2 a=catItems i=in.csv`

no	catItems
1	a c b
2	a d a e
3	b f
4	e e

Table 4.34: Retain original vectors before merging
`mvcat vf=item1,items2 -A i=in.csv`

no	items1,items2,new
1	a c,b,a c b
2	a d,a e,a d a e
3	b f,,b f
4	e,e,e e

Format

`mvcat vf= a= [-A]` [i=] [o=] [delim=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`vf=`	Merge specified field names of vectors (from the input file `i=`). Wildcard can be substituted for a character within the field name.
`a=`	Field name of the merged vector.
`-A`	Add results as a new field. If this option is not specified, the original field (`vf=`) will be removed.

Examples

Example 1: Merge vectors using wild character

```
$ more dat1.csv
items1,items2,items3,items4
b a c,b,x,y
c c,,x,y
e a a,a a a,x,y
$ mvcat vf=items* a=items i=dat1.csv o=rsl1.csv
#END# kgvcat a=items i=dat1.csv o=rsl1.csv vf=items*
$ more rsl1.csv
items
b a c b x y
c c x y
e a a a a a x y
```

Related command

4.77 mvcommon - Select Common Elements of Vector

Within the vector, select common elements specified in reference file.

See examples in Table 4.35∼4.38.

Table 4.35: Input data

in.csv

no	items
1	a b c
2	a d
3	b f e f
4	f c d

Table 4.36: Reference file

ref.csv

item
a
c
e

Table 4.37: Basic example

vf=items m=ref.csv K=item

no	items
1	a c
2	a
3	e
4	c

Table 4.38: Selection of unmatched items.

vf=items m=ref.csv K=item -r

no	items
1	b
2	d
3	b f f
4	f d

Since the `mvcommon` command reads the whole reference file at once into the memory, note that huge reference file may consume massive amount of memory,

Format

`mvcommon vf= [A=] K= [-r] m= |` i= [o=] [delim=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`vf=`	Specify the field name(s) of vector (from `i=` input file) for matching. Multiple fields can be specified. Sorting on vectors is not required. Output of vector series can be defined with colon followed after the vector name.
`m=`	Reference file.
`K=`	Join key of reference file (`m=`) where corresponding taxonomy elements are joined to the vector.
`-r`	Select records where key elements that do no match in `vf=` and `K=` .

Examples

Example 1: Match common elements in multiple vectors

```
$ more dat1.csv
items1,items2
b a c,b b
c c,a d
e a a,a a
$ more ref1.csv
item
a
c
e
$ mvcommon vf=items1,items2 K=item m=ref1.csv i=dat1.csv o=rsl1.csv
#END# kgvcommon K=item i=dat1.csv m=ref1.csv o=rsl1.csv vf=items1,items2
$ more rsl1.csv
items1,items2
a c,
c c,a
e a a,a a
```

Example 2: Print output to a new column

Define new column name for `item2` as `new2`.

```
$ mvcommon vf=items1,items2:new2 K=item m=ref1.csv i=dat1.csv o=rsl2.csv
#END# kgvcommon K=item i=dat1.csv m=ref1.csv o=rsl2.csv vf=items1,items2:new2
$ more rsl2.csv
items1,new2
a c,
c c,a
e a a,a a
```

Related command

mvjoin : Join reference element to vector instead of select.

4.78 mvcount - Calculate Vector Size

Calculate the size of vector (number of elements in a vector) .

An example is shown in Table 4.39 - 4.40.

Table 4.39: Input data

`in.csv`

no	items
1	a b c
2	a d
3	b f e f
4	

Table 4.40: Basic example

`mvcount vf=items:size i=in.csv`

no	items	size
1	a b c	3
2	a d	2
3	b f e f	4
4		0

Format

`mvcount vf=` [i=] [o=] [delim=] [-assert_diffSize] [-assert_nullin] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help]` `[--helpl]` `[--version]`

Parameters

`vf=` Specify the field names (from input file `i=`) of vectors to count the number of elements.
Field name(s) of result(s) can be defined with : followed after the vector name.
Multiple vectors can be specified.

Examples

Example 1: Count multiple vectors

```
$ more dat1.csv
items1,items2
b a c,b
c c,
e a a,a a a
$ mvcount vf=items1:size1,items2:size2 i=dat1.csv o=rsl1.csv
#END# kgvcount i=dat1.csv o=rsl1.csv vf=items1:size1,items2:size2
$ more rsl1.csv
items1,items2,size1,size2
b a c,b,3,1
c c,,2,0
e a a,a a a,3,3
```

Related command

4.79 mvdelim - Change Vector Delimiter

Change delimiter used to separate between string of characters in a vector. However, the delimiter will be removed if an empty string is specified as the delimiter.

Some examples are shown in Table 4.41 - 4.45. When comma is used as delimiter, a pair of double quotation marks is added to the vector (Table4.43). If a null character is specified as delimiter at `v=`, the delimiter between characters will be removed (Table 4.44).

Alphabet and chinese characters can be used as delimiter as shown in Table 4.45. Since the character type delimiter is read as character string as part of the vector by other commands such as mvsort, character type delimiter can be specified in `delim=`.

Table 4.41: input data

`in.csv`

no	items
1	b a a
2	a a b b
3	a b b a
4	a b c

Table 4.42: Basic example : Replace space delimiter with minus character.

`vf=items v=- i=in.csv`

no	items
1	b-a-a
2	a-a-b-b
3	a-b-b-a
4	a-b-c

Table 4.43: Use comma as a delimiter.

`vf=items v=, i=in.csv`

no	items
1	”b,a,a”
2	”a,a,b,b”
3	”a,b,b,a”
4	”a,b,c”

Table 4.44: Remove delimiter between characters in a vector

`vf=items v= i=in.csv`

no	items
1	baa
2	aabb
3	abba
4	abc

Table 4.45: Specify more than one character

`vf=items v=xx i=in.csv`

no	items
1	bxxaxxa
2	axxaxxbxxb
3	axxbxxbxxa
4	axxbxxc

Format

`mvdelim vf= v= [-A]` [i=] [o=] [delim=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`vf=`	Field name of vector to replace the delimiter. Multiple fields can be specified.
`v=`	Define new delimiter. If the parameter is not defined, the delimiter is treated as an empty character.

Examples

Example 1: Basic Example

Replace the default space delimiter to _(underscore).

```
$ more dat1.csv
item1
b a c
c c
e a a
$ mvdelim vf=item1 v=_ i=dat1.csv o=rsl1.csv
#END# kgvdelim i=dat1.csv o=rsl1.csv v=_ vf=item1
$ more rsl1.csv
item1
b_a_c
```

```
c_c
e_a_a
```

Example 2: Comma

In CSV data with comma delimited characters, when the delimiter of vector is replaced as comma, the entire vector is enclosed in double quotes to differentiate from the delimiter of CSV.

```
$ mvdelim vf=item1 v=, i=dat1.csv o=rsl2.csv
#END# kgvdelim i=dat1.csv o=rsl2.csv v=, vf=item1
$ more rsl2.csv
item1
"b,a,c"
"c,c"
"e,a,a"
```

Related command

4.80 mvdelnull - Remove a NULL Element in Vector

Remove all NULL elements in the vector. If NULL element exist in vector, there will be consecutive delimiters of the elements. All vectors shown below contains NULL elements. However, for ease of reading, '\n' is added at the end of each vector. Reading from the top row, the 3rd element, 1st element, 4th element are NULL.

```
a b  c\n
 a b\n
a b c \n
```

Format

`mvdelnull vf= [-A]` i= [o=] [delim=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`vf=`	Specify the field name (from input file `i=`) which contains NULL element for removal. Multiple files can be specified.
`-A`	Instead of replacing the specified item, this option adds output as a new field. When -A open is specified, the new field name must be specified after :(colon). Example: f=quantity:substitution field name

Examples

Example 1: Example 1: Basic example of removing null characters

```
$ more dat1.csv
items
b a  c
 c c
e a   b
$ mvdelnull vf=items i=dat1.csv o=rsl1.csv
#END# kgvdelnull i=dat1.csv o=rsl1.csv vf=items
$ more rsl1.csv
items
b a c
c c
e a b
```

Example 2: Example 2: Example of using .(dot) as delimiting character

```
$ more dat2.csv
items
b.a..c
.c.c
e.a...b.
$ mvdelnull vf=items delim=. i=dat2.csv o=rsl2.csv
#END# kgvdelnull delim=. i=dat2.csv o=rsl2.csv vf=items
$ more rsl2.csv
items
b.a.c
c.c
e.a.b
```

Example 3: Example 3: Change field name and output

```
$ mvdelnull vf=items:new i=dat1.csv o=rsl3.csv
#END# kgvdelnull i=dat1.csv o=rsl3.csv vf=items:new
$ more rsl3.csv
new
b a c
c c
e a b
```

Example 4: Example 3: Add output as an new field by specifying -A

```
$ mvdelnull vf=items:new -A i=dat1.csv o=rsl4.csv
#END# kgvdelnull -A i=dat1.csv o=rsl4.csv vf=items:new
$ more rsl4.csv
items,new
b a  c,b a c
 c c,c c
e a   b ,e a b
```

Related Command

mvnullto : Replace NULL element to any value.

4.81 mvjoin - Join Reference Vector Elements

Join vector elements with corresponding taxonomy elements from reference file with the same key. A vector field is shown in Table 4.46 where the column item includes multiple elements separated by a space delimiter.

Table 4.46 - 4.49 highlights some examples.

Table 4.46: Input data

in.csv

no	items
1	a b c
2	a d
3	b f e f
4	f c d

Table 4.47: Reference file

ref.csv

item	taxo
a	X
b	Y
c	Z
e	X
f	Z

Table 4.48: Basic example

vf=items m=ref.csv K=item f=taxo

no	items
1	a b c X Y Z
2	a d X
3	b f e f Y Z X Z
4	f c d Z Z

Table 4.49: An example defining unmatched taxonomy elements

vf=items m=ref.csv K=item f=taxo n=*

no	items
1	a b c X Y Z
2	a d X *
3	b f e f Y Z X Z
4	f c d Z Z *

Take note that the `mvjoin` common read the whole reference file at once into memory, thus huge reference file may consume massive amounts of memory.

Format

`mvjoin vf= [-A] K= f= [n=] m=|` i= [o=] [delim=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`vf=`	Field name of vector (from `i=` input file) for joining. Multiple fields can be specified. Sorting of the vectors is not required.
`m=`	Reference file.
`K=`	Specify key field in reference file (`m=`) where corresponding taxonomy elements are joined to the vector. The sequence of vector should be unique, sorting is not required. The output may differ if the string sequence is not unique.
`f=`	Field name of vector (element) for joining.
`n=`	Specify the replacement character when the key elements do not match in `vf=` and `K=` . The vector (element) will not be joined with the reference file when this option not specified.

Examples

Example 1: Combine vector with elements from reference file

```
$ more dat1.csv
items
b a c
c c
e a a
$ more ref1.csv
item,taxo
```

```
a,X Y
b,X
c,Z Z
$ mvjoin vf=items K=item m=ref1.csv f=taxo i=dat1.csv o=rsl1.csv
#END# kgVjoin K=item f=taxo i=dat1.csv m=ref1.csv o=rsl1.csv vf=items
$ more rsl1.csv
items
b a c X X Y Z Z
c c Z Z Z Z
e a a X Y X Y
```

Example 2: Join elements to multiple fields

```
$ more dat2.csv
items1,items2
b a c,b b
c c,a d
e a a,a a
$ more ref2.csv
item,taxo
a,X
b,X
c,Y
d,Y
$ mvjoin vf=items1,items2 K=item m=ref2.csv f=taxo i=dat2.csv o=rsl2.csv
#END# kgVjoin K=item f=taxo i=dat2.csv m=ref2.csv o=rsl2.csv vf=items1,items2
$ more rsl2.csv
items1,items2
b a c X X Y,b b X X
c c Y Y,a d X Y
e a a X X,a a X X
```

related command

mvcommon : Use this command to select common elements of vector.

4.82 mvnullto - Replace NULL in vector elements

Replace NULL elements in the vector with an ad hoc value. If NULL element exist in the vector, there will be consecutive delimiters of the elements. All the vectors described below contain NULL element. However, for ease of reading, '\n' is added at the end of each vector. Reading from the top row, the 3rd element, 1st element, and 4th element are NULL.

```
a b  c\n
 a b\n
a b c \n
```

Format

mvnullto vf= [v=|-p] [O=] [-A] i= [o=] [delim=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [--help] [--helpl] [--version]

Parameters

vf=	Specify the field name (i=on file) to replace NULL. Multiple items can be specified.
-A	Instead of replacing the specified item, this option adds output as a new field.
v=	Specify the replacement string.
-p	Replace NULL with the previous element. It can not be specified with v=.
O=	Replace all non-NULL elements with the string specified here. Non-NULL value will not be replaced unless specified. When -A open is specified, the new field name must be specified after :(colon). Example: f=quantity:substitution field name

Examples

Example 1: Example1: Replace null characters to the character string 'null '

```
$ more dat1.csv
items
b a  c
 c c
e a   b
$ mvnullto vf=items v=null i=dat1.csv o=rsl1.csv
#END# kgvnullto i=dat1.csv o=rsl1.csv v=null vf=items
$ more rsl1.csv
items
b a null c
null c c
e a null null b null
```

Example 2: Example 2: Use .(dot) as delimiting character

```
$ more dat2.csv
items
b.a..c
.c.c
e.a...b.
$ mvnullto vf=items v=null delim=. i=dat2.csv o=rsl2.csv
#END# kgvnullto delim=. i=dat2.csv o=rsl2.csv v=null vf=items
$ more rsl2.csv
items
b.a.null.c
null.c.c
e.a.null.null.b.null
```

Example 3: Example 3: Replace null with the previous value

```
$ mvnullto vf=items -p i=dat1.csv o=rsl3.csv
#END# kgvnullto -p i=dat1.csv o=rsl3.csv vf=items
$ more rsl3.csv
items
b a a c
 c c
e a a a b b
```

Example 4: Example 4: Replace all values except null by specifying O=

```
$ mvnullto vf=items v=null O=X i=dat1.csv o=rsl4.csv
#END# kgvnullto O=X i=dat1.csv o=rsl4.csv v=null vf=items
$ more rsl4.csv
items
X X null X
null X X
X X null null X null
```

Related command

mvdelnull : Delete a NULL element.

4.83 mvreplace - Replace an Element in Vector

Replace vector data with corresponding taxonomy character in the reference file joined by key. Table 4.50 shows items in a vector with sequential elements separated by a space.

The examples are highlighted in Table 4.50 - 4.53.

Table 4.50: Input data

in.csv

no	items
1	a b c
2	a d
3	b f e f
4	f c d

Table 4.51: Reference file

ref.csv

item	taxo
a	X
b	Y
c	Z
e	X
f	Z

Table 4.52: Basic example

`vf=items m=ref.csv K=item f=taxo`

no	items
1	X Y Z
2	X d
3	Y Z X Z
4	Z Z d

Table 4.53: Define unmatched elements

`vf=items m=ref.csv K=item f=taxo n=*`

no	items
1	X Y Z
2	X *
3	Y Z X Z
4	Z Z *

Since the `mvreplace` command reads the whole reference file at once in memory, note that huge reference file may consume massive amount of memory, .

Format

`mvreplace vf= K= f= [n=] m= [-A]` i= [o=] [delim=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`vf=`	Specify the field name of vector (from input file `i=`). Multiple fields can be specified. Sorting on vectors is not required.
`m=`	Reference file
`K=`	Key field in reference file (`m=`) where corresponding taxonomy elements are joined with vector items. The sequence of the vector is unique, sorting is not required. The output may differ if the string sequence is not unique.
`f=`	Field name of vector for joining.
`n=`	Specify the replacement character when the elements that do not match in `vf=` and `K=` . The element will not be joined with the reference file when this option not specified.

Examples

Example 1: Replace elements in a vector

```
$ more dat1.csv
items
b a c
c c
e a a
$ more ref1.csv
item,taxo
a,X Y
b,X
```

```
c,Z Z
$ mvreplace vf=items K=item m=ref1.csv f=taxo i=dat1.csv o=rsl1.csv
#END# kgvreplace K=item f=taxo i=dat1.csv m=ref1.csv o=rsl1.csv vf=items
$ more rsl1.csv
items
X X Y Z Z
Z Z Z Z
e X Y X Y
```

Example 2: Replace character in multiple elements

```
$ more dat2.csv
items1,items2
b a c,b b
c c,a d
e a a,a a
$ more ref2.csv
item,taxo
a,X
b,X
c,Y
d,Y
$ mvreplace vf=items1,items2 K=item m=ref2.csv f=taxo i=dat2.csv o=rsl2.csv
#END# kgvreplace K=item f=taxo i=dat2.csv m=ref2.csv o=rsl2.csv vf=items1,items2
$ more rsl2.csv
items1,items2
X X Y,X X
Y Y,X Y
e X X,X X
```

Related Command

mvjoin : Use `mvjoin` to combine the elements instead of replacing elements.

4.84 mvsort - Sort Vectors

This command sort series of vectors in column. The vector in the fields shown in Table 4.54 shows multiple character strings separated by space delimiter. Table 4.54 - 4.57 highlight examples sorting vectors.

In Table 4.55, character strings are arranged in ascending order by default. The character strings can be sorted in numerical ascending order by attaching `%` after the item name followed by `n` (see Table 4.56), and in reverse order by specifying `r` after an item name (Table 4.57).

Table 4.54: Input data `in.csv`

no	items
1	2 1 13
2	4 5 2 5
3	112 14
4	5 31

Table 4.55: Basic usage: Sort vector elements in character string ascending order

`vf=items`

no	items
1	1 13 2
2	2 4 5 5
3	112 14
4	31 5

Table 4.56: Sort numerical in ascending order

`vf=items%n`

no	items
1	1 2 13
2	2 4 5 5
3	14 112
4	5 31

Table 4.57: Sort in numerical descending order

`vf=items%nr`

no	items
1	13 2
2	5 5 4 2
3	112 14
4	31 5

Format

`mvsort vf= [-A]` [i=] [o=] [delim=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`vf=`	Specify the field name(s) of vectors for sorting. Multiple fields can be specified. Add `n` after `%` after field name to sort in ascending numerical order. Add `r` after `%` after field name to sort in reverse order. Add both `n` and `r` to sort in descending numerical order.

Examples

Example 1: Sort multiple vectors

Sort `item1` data series in ascending order and `item2` in numerical ascending order.

```
$ more dat1.csv
items1,items2
b a c,10 2
c c,2 5 3
e a a,1
$ mvsort vf=items1%r,items2%n i=dat1.csv o=rsl1.csv
#END# kgvsort i=dat1.csv o=rsl1.csv vf=items1%r,items2%n
$ more rsl1.csv
items1,items2
c b a,2 10
c c,2 3 5
e a a,1
```

Related Command

4.85 mvuniq - Unique Vector Elements

This command merges duplicate elements in a vector. However, since the merging process uses a tree structure, the sequence of elements in the output maybe not be in order.

When `-n` option is specified, the command reads the vector as a sequential series. The vector series is scanned from the beginning of the string, and output unique character strings in the vector.

The examples are highlighted in Table 4.59, 4.60. Table 4.59 shows all merged elements in the data series. When `-n` option is specified, same elements next to each other are merged in sequential order. In Table 4.60, consecutive elements of `b` are merged.

Table 4.58: Input data

`in.csv`

no	items
1	b a a
2	a a b b b
3	a b b a
4	a b c

Table 4.59: Basic example

`vf=items i=in.csv`

no	items
1	a b
2	a b
3	a b
4	a b c

Table 4.60: Merge same elements adjacent to each other in a vector series

`vf=items -n i=in.csv`

no	items
1	b a
2	a b
3	a b a
4	a b c

Format

`mvuniq vf= [-A] [-n]` [i=] [o=] [delim=] [-assert_diffSize] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

`vf=`	Specify the target field name(s) of vectors. Multiple field name(s) of vectors can be specified.
`-n`	Merges same elements adjacent to each other in the vector.

Examples

Example 1: Merges vector elements in multiple fields

```
$ more dat1.csv
items1,items2
b a c,1 1
c c,2 2 3
e a a,3 1
$ mvuniq vf=items1,items2 i=dat1.csv o=rsl1.csv
#END# kgvuniq i=dat1.csv o=rsl1.csv vf=items1,items2
$ more rsl1.csv
items1,items2
a b c,1
c,2 3
a e,1 3
```

Related Command

4.86 mwindow - Generate Sliding Window

Replicate original records and shift specified fields. A fixed window with constant width is set when calculating moving averages for time series data. The first element of moving average is obtained by taking the average of the initial fixed subset of the number series. The subset is shifted forward and included the next number following the original subset in the series. This method is known as sliding window calculation.

An example is shown from Table 4.63.

Table 4.61: input data

date	val
4/6	1
4/7	2
4/8	3
4/9	4

Table 4.62: wk=date:win t=2

win	date	val
4/7	4/6	1
4/7	4/7	2
4/8	4/7	2
4/8	4/8	3
4/9	4/8	3
4/9	4/9	4

Table 4.63: wk=date:win t=2 -r

win	date	val
4/6	4/6	1
4/6	4/7	2
4/7	4/7	2
4/7	4/8	3
4/8	4/8	3
4/8	4/9	4

Table 4.61 shows the input data which contains total daily values of four consecutive days. The figures could represent the changes in sales in supermarket and stock price trends.

This example calculates the moving average from 4/6 to 4/9 with a subset size of 2 for each window. Three window intervals [(4/6,1),(4/7,2)], [(4/7,2),(4/8,3)], [(4/8,3),(4/9,4)] are generated, where [ ] indicates a window, and ( ) indicates a line.

Based on the unique key of each windows (referred as "window key"), the output prints the maximum value of the window (the last row of item can be specified by `wk=` parameter) and the field name based (Table 4.62). `-r` option is used as the minimum value (first row of data) of each window (Table 4.63). Afterwards, the output results (Table 4.62) is followed by using `mavg` to calculate the averages of the data series.

The `mmvavg` command is equivalent to processing the data with `mwindow`+`mavg` as described above. However, `mmvavg` is 3.5 times faster when experimented with a data set of 200MB for 10 million records with a subset size of 10 for each window.

Format

`mwindow wk= t= [k=key] [-r] [-n] [i=] [o=]` [-assert_diffSize] [-assert_nullkey] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameters

`wk=`	Specify an unique value from the field name in the input data that identifies the window. After the specified field is sorted, the sliding window is created, %r is added for descending sort order, %n is added for numeric sorting. When sorting in numeric descending order, %nr is added. It is necessary to define the field name of window key after a colon. Multiple fields can be specified.
`t=`	Specify the window size (number of rows).
`k=`	Specify the unit for the generation of windows.
`-r`	Use the first row of data as baseline of sliding window. By default, the last row of data is used as baseline.
`-n`	Print all window intervals even though the window size less than the defined parameter at `t=`.
`i=`	Input file name
`-nfn`	Input data without field header in the first row.

Examples

Example 1: Basic Example

```
$ more dat1.csv
date,val
20130406,1
20130407,2
20130408,3
20130409,4
$ mwindow wk=date:win t=2 i=dat1.csv o=rsl1.csv
#END# kgwindow i=dat1.csv o=rsl1.csv t=2 wk=date:win
$ more rsl1.csv
win%0,date,val
20130407,20130406,1
20130407,20130407,2
20130408,20130407,2
20130408,20130408,3
20130409,20130408,3
20130409,20130409,4
```

Example 2: Use first row as baseline data

```
$ mwindow wk=date:win t=3 -r i=dat1.csv o=rsl2.csv
#END# kgwindow -r i=dat1.csv o=rsl2.csv t=3 wk=date:win
$ more rsl2.csv
win%0,date,val
20130406,20130406,1
20130406,20130407,2
20130406,20130408,3
20130407,20130407,2
20130407,20130408,3
20130407,20130409,4
```

Example 3: Print all window intervals even if the window size is less than the defined parameter

```
$ mwindow wk=date:win t=3 -r -n i=dat1.csv o=rsl3.csv
#END# kgwindow -n -r i=dat1.csv o=rsl3.csv t=3 wk=date:win
$ more rsl3.csv
win%0,date,val
20130406,20130406,1
20130406,20130407,2
20130406,20130408,3
20130407,20130407,2
20130407,20130408,3
20130407,20130409,4
20130408,20130408,3
20130408,20130409,4
20130409,20130409,4
```

Example 4: Example of specifying key field

```
$ more dat2.csv
store,date,val
a,20130406,1
a,20130407,2
a,20130408,3
a,20130409,4
b,20130406,11
b,20130407,12
b,20130408,13
b,20130409,14
$ mwindow k=store wk=date:win t=2 i=dat2.csv o=rsl4.csv
#END# kgwindow i=dat2.csv k=store o=rsl4.csv t=2 wk=date:win
$ more rsl4.csv
win%1,store%0,date,val
20130407,a,20130406,1
20130407,a,20130407,2
```

```
20130408,a,20130407,2
20130408,a,20130408,3
20130409,a,20130408,3
20130409,a,20130409,4
20130407,b,20130406,11
20130407,b,20130407,12
20130408,b,20130407,12
20130408,b,20130408,13
20130409,b,20130408,13
20130409,b,20130409,14
```

Example 5: Find out the moving averages between current day and previous day

In the above example, moving average is calculated based on the last day of the window. `mslide` can be used for instances to calculate the moving averages of current day and previous day. The example is as follows:

```
$ mslide f=date:date2 -q i=dat1.csv o=rsl5.csv
#END# kgslide -q f=date:date2 i=dat1.csv o=rsl5.csv
$ more rsl5.csv
date,val,date2
20130406,1,20130407
20130407,2,20130408
20130408,3,20130409
```

Example 6: Find out the moving averages from the previous day

```
$ mwindow wk=date2:win t=2 i=rsl5.csv o=rsl6.csv
#END# kgwindow i=rsl5.csv o=rsl6.csv t=2 wk=date2:win
$ more rsl6.csv
win%0,date,val,date2
20130408,20130406,1,20130407
20130408,20130407,2,20130408
20130409,20130407,2,20130408
20130409,20130408,3,20130409
```

Related command

mmvavg : Command that specializes in computing average of sliding windows.

mmvstats : Compute various statistics of sliding windows.

4.87 mxml2csv - Convert XML to CSV

Convert XML formatted data to CSV. The basic rule of conversion is by specifying the element as unit of each record (XML tag) and the element corresponding to the column (or attribute) at the parameters `k=`, `f=`. The value of the column can be specified in four ways : text bounded by elements, presence of elements, the value of the attribute, presence of attributes.

When SAX is used as the parser of XML, there is no size constraints of XML. If other encoding besides UTF-8 is used in the XML file, the XML file is converted to UTF-8 and output as CSV. XML data should be structured in a complete, well-formed XML document. Otherwise, the program may return unexpected processing results.

Table 4.64 shows a typical format of XML data. More details are explained in the next section, however, a brief outline is illustrated as follows.

Table 4.65 shows the returned output. Element `<b>` is used as the key unit of each record (the element is referred to as "key element"). The column is defined by the attribute of element `b` for the value of `att` (CSV column name `b_att`). The attributes of element `c` includes the value of `p` (`b_p`) and flag (`b_p_f`), as well as the text inside element `d` and `a` (`d`, `a`).

Here, the flag indicates the presence of specified elements or attributes by the value 0-1 in the output. The text output of the element includes the concatenation of all strings in the range within specified element. However, note that the spaces and the control characters are not included in the output.

Table 4.64: input XML data

```
<a att="aa">
  <b att="bb1">
    <c p="pp1" q="qq1"/>
    <d>text1</d>
  </b>
  <b att="bb2">
    <c q="qq2"></c>
    <d>text2</d>
  </b>
  <b>
    <c p="pp3"/>
    <d>text3</d>
  </b>
</a>
```

Table 4.65: key:/a/b, item:b@att,c@p,d,/a

b_att	c_p	c_p_f	d	a
bb1	pp1	1	text1	text1
bb2			text2	text1text2
	pp3	1	text3	text1text2text3

Specifying the key element

Specify the key element as the key unit of each record (specified in the parameter `k=`) with an absolute path. The absolute path is defined from the root directory starting from the symbol (`'/'`), the hierarchy of elements is separated by the sign `'/'`. The role of the key elements in this command is to perform the following two functions corresponding to the end tag of the key elements.

- Output one row of column data. In the example above, the end tag of the key elements `</b>` has appeared three times, one row of CSV data is inserted as a new line for every instance in the output.
- Initialize the column data. However, it does not initialize the column elements outside the key elements in the output data. In Table 4.65, the text in the output of element `a` will be consolidated, even when the end tag of the key elements has emerged, the element `/a` is outside the key elements `/a/b`. As a result, the output data is not initialized.

Specifying the elements in output column

If the element defined at `f=` is returned as a CSV field in the output, follow the format shown below.

```
Element path[%flag]: CSV field name
```

"Field name" is the column name in the CSV output which must be specified.

There are two methods to display elements as columns in the output. The first method is to return the text enclosed by the opening and closing tags of the specified element. The other method is to return 0-1 value to indicate whether the specified element exists. The target element path is defined in the former method, and the the flag `%f` is added when using the latter method.

The two methods of specifying the element path include absolute and relative paths. A relative path can be specified by defining the path from elements at k=. Table 4.64 shows examples on how to specify the element paths of the XML data.

- Given `k=/a/b`, when `f=:B` is specified, the key elements is the same when relative path is nil. `B` is the column name of CSV.
- Given `k=/a/b`, `f=c:C` and `f=/a/b/c:C` performs the same function. The former is specified by relative path, while the latter by an absolute path. The CSV field name is defined as `C` for both methods.
- `f=d:D` returns the text within the element `d`. `f=d%f:D returns output when element \verbd`— exists. The field name of CSV is `D`.
- When `k=/a/b`, it is assumed that `f=/a:A`, the column element is outside the key element, the text contained in the element `a` are combined in sequential order. The end tag of the key element exists, however, the end tag of the field element does not exists, this is because data is not cleared at the time.

Specifying attributes in output column

If the attribute defined at `f=` is returned as a CSV column, use the format shown below.

`Element path@Element name[%flag]:CSV field name`

"Field name" is the column name in the CSV output which must be specified.

The method of specifying the element path is the same as specifying the elements in the output field. The attribute name is specified after the element path connected with `@`. By adding `%f` after the element name, the presence of the element can be indicated by 0-1 value in the output.

Format

`mxml2csv k= f= [i=]` [o=] [-nfn] [-nfno] [-x] [-q] [tmpPath=] `[--help] [--helpl] [--version]`

Parameter

`k=`	Specify the pathname from the root based on the element as the unit of records. The path starts from the root symbol '/', and the specified elements are connected with '/'. Example: /article/sentence/chunk

`f=`	Specify the element or attribute as fields in the output by delimiting the field names with comma.
	Format is as follows.
	`Element path[%flag]:CSV field name`
	`Element path@Element name[%flag]:CSV field name`

`i=`	Specify the file name of XML data. The input is read from standard input by default when the input file is not specified.

Examples

Example 1: Example 1: Basic example

The example below is illustrated in the summary above. Output the 5 CSV fields with /a/b set as the key elements.

```
$ more dat1.xml
<a att="aa">
  <b att="bb1">
    <c p="pp1" q="qq1"/>
    <d>text1</d>
  </b>
  <b att="bb2">
    <c q="qq2"></c>
    <d>text2</d>
  </b>
  <b>
    <c p="pp3"/>
    <d>text3</d>
  </b>
</a>
$ mxml2csv k=/a/b f=@att:b_att,c@p:c_p,c@p%f:c_p_f,d:d,/a:a i=dat1.xml  o=rsl1.csv
#END# kgxml2csv f=@att:b_att,c@p:c_p,c@p%f:c_p_f,d:d,/a:a i=dat1.xml k=/a/b o=rsl1.csv
$ more rsl1.csv
b_att,c_p,c_p_f,d,a
bb1,pp1,1,text1,text1
bb2,,,text2,text1text2
,pp3,1,text3,text1text2text3
```

Example 2: Example 2: Absolute path

Specification of same element as in the basic example with an absolute path. Output the 5 CSV fields with /a/b as the key elements.

```
$ mxml2csv k=/a/b f=/a/b@att:b_att,/a/b/c@p:c_p,/a/b/c@p%f:c_p_f,/a/b/d:d,/a:a i=dat1.xml  o=rsl2.csv
#END# kgxml2csv f=/a/b@att:b_att,/a/b/c@p:c_p,/a/b/c@p%f:c_p_f,/a/b/d:d,/a:a i=dat1.xml k=/a/b o=rsl2.csv
$ more rsl2.csv
b_att,c_p,c_p_f,d,a
bb1,pp1,1,text1,text1
bb2,,,text2,text1text2
,pp3,1,text3,text1text2text3
```

Example 3: Example 3: Changing key elements

Example of changing a key element to a using an absolute path. Since there is only one end tag a, one row of record will be returned as output. /a/b@att specified at f= appeared twice, the last value is returned as output.

```
$ mxml2csv k=/a f=/a/b@att:b_att,/a/b/c@p:c_p,/a/b/c@p%f:c_p_f,/a/b/d:d,/a:a i=dat1.xml o=rsl3.csv
#END# kgxml2csv f=/a/b@att:b_att,/a/b/c@p:c_p,/a/b/c@p%f:c_p_f,/a/b/d:d,/a:a i=dat1.xml k=/a o=rsl3.csv
$ more rsl3.csv
b_att,c_p,c_p_f,d,a
bb2,pp3,1,text3,text1text2text3
```

Related command

Chapter 5

mcal

`mcal` command is developed for computation between columns.

Commands other than `mcal` carry out record based processing, however, mcal command specializes in item-based calculations. There are more than 100 functions/operators in mcal, user has the flexibility to define various processing functions that combine multiple functions and operators.

5.1 mcal - Computation Between Columns

Define the computation formula at the `c=` parameter, and name the new data attribute at the `a=` parameter. The output of `mcal` is limited to 1 result without exception to simplify the program. For details of the calculation formula, please refer to the section on " Components in the expression".

Format

`mcal a= c=` [i=] [o=] [-assert_diffSize] [-assert_nullkey] [-assert_nullin] [-assert_nullout] [-nfn] [-nfno] [-x] [-q] [tmpPath=] [precision=] `[--help] [--helpl] [--version]`

Parameters

a=	Specify the new column name to store the calculated field.
c=	Define an expression with a combination of calculation functions available.

Examples

Basic usage of `mcal` is illustrated in the following example. For more information on the explanation and usage of individual functions and operators, please refer to the corresponding reference.

```
# Input data (dat1.csv)
customer,quantity,unitprice
A,3,10
B,1,15
C,2,20

$ mcal c='${quantity}*${unitprice}' a=amount i=dat1.csv
customer,unit price,amount
A,3,10,30
B,1,15,15
C,2,20,40

$ mcal c='${quantity}*${unitprice}<=30' a=amountbelow30 i=dat1.csv
customer,unit price,amountbelow30
A,3,10,1
B,1,15,1
C,2,20,0

$ mcal c='if(top(),${unit price},#{}+${unitprice})' a=AccumUnitprice i=dat1.csv
customer,unitprice,AccumUnitprice
A,3,10,10
B,1,15,25
C,2,20,45
```

Considerations when using shell

When using BASH shell on UNIX operating systems, the symbol of operators often have special meaning to the shell. For example, a shell variable is represented by the `$` symbol followed by a character string. On the other hand, mcmd uses the `$` symbol to refers to the value of the data field. Thus, mcmd variables is enclosed in squiggly brackets preceded by the `$` symbol in order not to be misinterpreted as shell variables.

```
$ mcal c='${date}-10'
```

Error message

```
#ERROR# unknown function or operator
```

This error message appears when there is an error in the specified operator or function. For instance, refer to the error message from the concatenation of strings function `cat`.

```
$ mcal c='cat("-",1,2)'
ERROR : unknown function or operator: cat_SNN(cat_SN) (kgcal)
```

The string before underscore character in "cat_SNN" indicates the function name `cat`, subsequently, SNN refers to the type of the argument. S refers to string type, N refers to number type, D refers to date type, and T is the time type, B refers to boolean type. The 3 characters (SNN) specifies 3 arguments. Thus, this error message means " arguments SNN of cat function " is not registered. The second and third argument is converted to string as follows.

```
$ mcal c='cat("-","1","2")'
```

The above returns an error message with 2 characters in parenthesis (SN), this refers to specification of variable or number at the first two parameters, however, only 1 variable is expressed correctly in the function.

Related Commands

msel : Use this command to select the row from the computation result.

5.2 Components in the expression

The four key components in `mcal` includes constants, item value, operator, and function. In every component, it is important for user to understand the application of data type. `mcal` handles CSV text data, all values are expressed as character strings, user can define the data type to be handled in mcal. There are five string types in mcal, they are string type (*str*), numeric type (*num*), date type (*date*), time type (*time*), boolean type (*bool*). The following sessions illustrates how each component constitutes to the expression and how to treat the data types.

5.3 Constant

Table 5.1: Summary of constant attributes

Data type	Format	Description	Example
Numerical Value(*num*)	integer, real number string	Double precision floating point numbers is used internally	`20, 0.55, 1.5*e10`
Character string(*str*)	"Character string"	Character string enclosed in double quotes	`"abc" "日本語"`
Date (*date*)	0dyyyymmdd	Add "0d" before fixed length year month day	`0d20080923`
Time (*time*)	0tyyyymmddHHMMSS	Add "0t" before fixed length year, month, date, time, minute and second	`0t20080923121115`
	0tHHMMSS	add "0t" before fixed length time, minute and second (current date stored internally)	`0t121115`
Boolean (*bool*)	0b1, 0b0	Add "0b" before "1"(true) and "0"(false)	`0b1, 0b0`

5.4 Field value

Table 5.2 shows the different data formats in the data field, the type of CSV data varies depends on how it is used and defined by the user.

5.5 Wildcard

Wildcard can be used in field names. For example, when using the sum function to compute the total across multiple fields with numeric labels, wildcard can be use to simplify the listing of all fields as one label. For example, if there are three columns are named `A1,A2,A3` in the input data, the total sum of `A1,A2,A3` can be calculated as `sum(${A*})`. It is also possible to specify multiple wildcard such as the expression `sum(${A*},${B*})`.

5.6 Value from Previous Row

Use the `#` symbol instead of `$` to refer to values from a field in the previous row. However, the function will return null when used on the first record since there is no record preceding the first record. The specification of each data type is shown in 5.3 below.

5.7 Values from Next Row

Use the expression to obtain value from the previous record without specifying the field name to obtain the value in the next record. The specification of the data types are shown in 5.4.

It is possible to calculate total by combining `if` function with `top()` function. The cumulative calculation on the amount field is shown below.

```
$ mcal c='if(top(),${amount},${amount}+#{})' a=cumulativeAmount
```

Table 5.2: Format of field

Data	Format	Content of CSV Data	Example
Numerical value(*num*)	${fieldname}	Integer, real number (including floating Numerical string	`${amount}, ${stockprice}`
Character string(*str*)	$s{fieldname}	Character string	`$s{gender}, $s{gender}`
Date(*date*)	$d{fieldname}	Fixed length year month day (yyyymmdd)	`$d{date}, $d{orderdate}`
Time(*time*)	$t{fieldname}	Fixed length year month day minute second (yyyymmddHH-MMSS) Fixed length hour minute second (HHMMSS) (current date stored internally)	`$d{time}, $d{departuretime}`
Boolean(*bool*)	$b{fieldname}	The value is expressed as "1" if true and "0" if false、 Other cases treated as NULL	`$b{condition}, $b{condition}`

Table 5.3: Specification of retrieving values from previous row

Data type	Format	Example
Numeric(*num*)	#{fieldname}	`#{amount}, #{stockprice}`
Character string(*str*)	#s{fieldname}	`#s{gender}, #s{gender}`
Date(*date*)	#d{fieldname}	`#d{date}, #d{releasedate}`
Time(*time*)	#t{fieldname}	`#d{time}, #d{departuretime}`
Boolean(*bool*)	#b{fieldname}	`#b{condition}, #b{condition}`

Table 5.4: Specification to retrieve values from previous row

Data Type	Format	Example
Numeric(*num*)	#{}	`#{}`
Character String(*str*)	#s{}	`#s{}`
Date(*date*)	#d{}	`#d{}`
Time(*time*)	#t{}	`#d{}`
Boolean(*bool*)	#b{}	`#b{}`

5.8 Arithmetic Operators

The `+` and `-` arithmetic operators can be used on numeric format strings as well as date and character format strings. The data format is shown in Table 5.5.

Table 5.5: Summary of arithmetic operators

Operator	Format	Description	Example
Addition(+)	$num_1 + num_2$	Addition of numeric values	`1.5+2.3 (3.8)`
	$str_1 + str_2$	Join character strings	`"150"+"円" ("150円")`
	$date + num$	*num* days after date	`0d20121130+2 (0d20121202)`
	$time + num$	*num* seconds after time	`0t095959+2 (0t100001)`
Substraction(-)	$num_1 - num_2$	Subtraction of numeric values	`1.5-2.3 (-1.8)`
	$str_1 - str_2$	Remove substring	`"aababa"-"a" ("bb")`
		(by greedy match algorithm)	`"aababa"-"aba" ("aba")`
	$date - num$	*num* days before date	`0d20121202-2 (0d20121130)`
	$time - num$	*num* seconds before time	`0t100001-2 (0t095959)`
	$date_1 - date_2$	Date difference	`0d20121202-0d20121130 (2)`
	$time_1 - time_2$	Time difference	`0t095959-0t100001 (-2)`
Multiplication (*)	$num_1 * num_2$	multiply	`10*2 (20)`
Division (/)	num_1/num_2	divide	`10/2 (5)`
Remainder (%)	$num_1\%num_2$	remainder	`10%3 (1)`
Power (ˆ)	$num_1\hat{}num_2$	power	`10^3 (1000)`

The results of the examples are shown in parentheses (the content is shown using constant numbers).

5.9 Comparison Operators

The comparison operators can only be used on data of the same type. Table 5.6 shows the list of operators for numeric format data. Similarly, the operators shown in the table below can be applied to character format, date format, time format data.

Table 5.6: Summary of comparison operators

Details of comparison	Format	Example
Equal	$num_1 == num_2$	`1.5==1.5(0b1), "abc"=="abcd" (0b0)`
Not equal	$num_1 != num_2$	`1.5!=1.5(0b0), "abc"=="abcd" (0b1)`
Greater than	$num_1 > num_2$	`10>5(0b1), "abc">"abcd" (0b0)`
Less than	$num_1 < num_2$	`10<5(0b0), "abc"<"abcd" (0b1)`
Above	$num_1 >= num_2$	`10>=10(0b1), "a">"" (0b1)`
Below	$num_1 <= num_2$	`8<=9(0b1), "a"<="a" (0b1)`

The results of the examples are shown in parentheses (the content is shown using constant numbers).

5.10 Logical Operator

The usage of the three logical operators (conjunction, disjunction, exclusive or) is shown in Table 5.7. In addition the results of the combination of boolean values (1:true, 0:false) are shown in Table 5.8,Table 5.9 and Table 5.10.

Table 5.7: Summary of logical operators

Description	Format	Example
Conjunction	$bool_1\&\&bool_2$	`"abc"=="abc" && "xyz"=="abc" (0b0)`
Disjunction	$bool_1 \| bool_2$	`"abc"=="abc" || "xyz"=="abc" (0b1)`
Exclusive or	$bool_1$ ^^ $bool_2$	`"abc"=="abc" ^^ "xyz"=="abc" (0b1)`

The results of the examples are shown in parentheses (the content is shown using constant numbers).

5.11 Operator Precedence

The operators in Table 5.11 are listed according to precedence order.

The order of precedence starts from the top, operators on the same line with equal precedence are evaluated from left to right. When operators of equal precedence appear in the same expression, use parentheses to change to operator precedence and the expression to evaluated first.

5.12 Function

The following highlights the 9 types of functions in relation to numeric strings (5.12), trigonometric function (), character strings (5.14), regular expression (5.15), date / time (5.16), logical (5.17), row/column information (5.18), Null value (5.19), data type conversion (5.20).

5.13 Date and Time Format

There are two data types in `mcal`, namely date and time format. One is the date type and the other are date type. Time formatted data is represented with date formatted data as a set. The command uses date_time library of boost C + + library based on the Gregorian calendar, date type uses class boost::gregorian::date, and time type uses class boost::posix_time::ptime. For more details, refer to documentation in boost.org.

Date class is managed as a 32-bit integer internally, and supports dates ranging from January 1,1400 to December 31, 9999. Operations on date is based on the Gregorian calendar. NULL value will be returned on invalid date (for example, 2013/2/29 or 1399/12/31).

On the other hand, class ptime is managed as 64 bit. It is a time system with nano-second/micro-second resolution. The mcal command do not have an interface for time point manipulation. Class ptime is dependent on gregorian::date for the interface to the date portion of a time point, thereby enable time calculations across different dates. NULL value is returned on invalid time (e.g.18:62:11).

MCMD deals with CSV text, date/time must be assigned as character string in the data. The command then converts character string to date and time type for processing various operations.The final result converted back to character string in the output. The string format is expressed as 8-digit fixed-length string for date (e.g. "20130911"), 14-digit fixed-length string for time (e.g. "20130911110528") or the standard 6-digit fixed-length string (for example, "110528").

Figure 5.1 below shows the relationship of date type, time type and various functions.

Table 5.8: Conjunction

$bool_1$	$bool_2$	Result
1	1	1
1	0	0
0	1	0
0	0	0
null	1	null
null	0	0
null	null	null

Table 5.9: Disjunction

$bool_1$	$bool_2$	Result
1	1	1
1	0	1
0	1	1
0	0	0
null	1	1
null	0	null
null	null	null

Table 5.10: Exclusive Or

$bool_1$	$bool_2$	Result
1	1	0
1	0	1
0	1	1
0	0	0
null	1	null
null	0	null
null	null	null

Table 5.11: Precedence of operators

Order	Operator
1	`*,/,%,^`
2	`+,-`
3	`>,<,>=,<=`
4	`== ,!=`
5	`&&`
6	`||,^^`

Other than using fixed length character string as a standard for date/time, user can use Julian day (e.g. continuous count of days since Julian period such as January 1, 4713 BC) or UNIX time (e.g. number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970) as a signed integer to represent date and time. The command supports Julian day and UNIX time, as well as conversion functions of date and time data type.

The mcal command uses an internal date/time format which is based on the Gregorian calendar, thus the date range is limited from January 1,1400 to December 31,9999. Since UNIX time is a signed integer data type of 32 bits, the furthest time that can be represented this way is 03:14:07 UTC on Tuesday, 19 January 2038, date beyond this point will be interpreted incorrectly due to integer overflow. The drawback of using UNIX time

Table 5.12: Summary of numerical functions

Section	Function name	Function	Output type
5.98	sum($num_1, num_2, \cdots$)	Sum	*num*
5.22	avg($num_1, num_2, \cdots$)	Average	*num*
5.97	sqsum($num_1, num_2, \cdots$)	Sum of squares	*num*
5.68	min($num_1, num_2, \cdots$)	Minimum value	*num*
5.66	max($num_1, num_2, \cdots$)	Maximum value	*num*
5.78	product($num_1, num_2, \cdots$)	Product	*num*
5.42	factorial(*num*)	Factorial	*num*
5.48	gcd(num_1,num_2)	Greatest common divisor	*num*
5.57	lcm(num_1,num_2)	Least common multiple	*num*
5.96	sqrt(*num*)	Square root	*num*
5.14	abs(*num*)	Absolute value	*num*
5.93	sign(*num*)	Sign	*num*
5.53	int(*num*)	Integer part	*num*
5.47	fract(*num*)	Fraction part	*num*
5.91	round(*num*,nominal value)	Rounding up	*num*
5.45	floor(*num*,nominal value)	Rounding down	*num*
5.28	ceil(*num*,nominal value)	Ceiling	*num*
5.77	power(*num*,exponent)	Power	*num*
5.41	exp(*num*)	Exponential function	*num*
5.63	log(*num*,base)	logarithm	*num*
5.62	ln(*num*)	Natural logarithm	*num*
5.65	log2(*num*)	Binary logarithm	*num*
5.64	log10(*num*)	Common logarithm	*num*
5.37	dist(type,$num_1, num_2, \cdots$)	Distance	*num*
5.38	distgps(latitude1,longtitude1,latitude2,longtitude2)	GPS distance	*num*
5.50	heron($num_1, num_2, \cdots$)	Heron 's formula	*num*
5.80	rand([random seed])	Uniform random number	*num*
5.81	randi(minimum value, maximum value[, random seed])	Uniform random number	*num*
5.74	nrand(minimum value, maximum value[, random seed])	Normal random number	*num*
5.76	pi()	Pi	*num*
5.40	e()	Napier's constant	*num*
5.46	format()	Format output	*str*

Table 5.13: List of trigonometric functions

Section	Function Name	Function	Output range
5.15	acos(*num*)	Inverse cosine	$0 \sim \pi$
5.19	asin(*num*)	Inverse sine	$-\pi \sim \pi$
5.20	atan(*num*)	Inverse tangent	$-\pi \sim \pi$
5.21	atan2(num_1,num_2)	Angle of coordinates (num_1, num_2)	$-\pi \sim \pi$
5.29	cos(r)	Cosine	$-1.0 \sim 1.0$
5.94	sin(r)	Sine	$-1.0 \sim 1.0$
5.100	tan(r)	Tangent	$-\infty \sim \infty$
5.35	degree(r)	Degree	$-\pi \sim \pi$
5.79	radian(angle)	Enter angle as input, return radian as output	$-\pi \sim \pi$
5.30	cosh(r)	Hyperbolic cosine	$0 \sim \infty$
5.95	sinh(r)	Hyperbolic sine	$-\infty \sim \infty$
5.101	tanh(r)	Hyperbolic tangent	$-1.0 \sim 1.0$

Radian is represented by the variable r.

and Julian day is that one will not be able to tell the date and time by looking at the number.

Table 5.14: Character string related functions

Section	Function name	Function	Output format
5.27	cat($token, str_1, str_2, \cdots$)	Merge character string	*str*
5.60	length(*str*)	Length of character string	*num*
5.43	fixlen(*str*,length,position,padding character)	Fixed length conversion	*str*
5.90	right(*str*,length)	Extract substring from the end	*str*
5.59	left(*str*,length)	Extract substring from the beginning	*str*
5.67	mid(*str*,starting position,length)	Extract substring	*str*
5.106	toupper(*str*)	Convert characters from lowercase to uppercase	*str*
5.104	tolower(*str*)	Converts characters from uppercase to lowercase	*str*
5.26	capitalize(*str*)	Capitalize the first character	*str*
5.54	match(search string,$str_1, str_2, \cdots$)	Search for matched strings	*bool*
5.49	hasspace(*str*)	Search for white-space characters	*bool*

Table 5.15: Regular expression related functions

Section	Function name	Function	Output format
5.83	regexm(*str*,regular expression)	Match whole string	*bool*
5.87	regexs(*str*,regular expression)	Match	*bool*
5.86	regexrep(*str*,regular expression,replacement string)	Replace matching character string	*str*
5.82	regexlen(*str*,regular expression)	Match number of characters	*num*
5.85	regexpos(*str*,regular expression)	Start position of character	*num*
5.89	regexstr(*str*,regular expression)	Match character string	*str*
5.84	regexpfx(*str*,regular expression)	Match prefix of character string	*str*
5.88	regexsfx(*str*,regular expression)	Match suffix of character string	*str*

Table 5.16: Date and Time Related Functions

Section	Function Name	Function	Output
5.103	today()	Today 's date	*date*
5.72	now()	Current time	*time*
5.107	tseconds(*time*)	Seconds elapsed	*num*
5.58	leapyear(*dt*)	Decide leap year	*bool*
5.111	year(*dt*)	Gregorian calendar	*num*
5.70	month(*dt*)	Month	*num*
5.33	day(*dt*)	Day	*num*
5.109	week(*dt*)	Week number	*num*
5.39	dow(*dt*)	Day of week	*num*
5.102	time(*time*)	Hour minute second	*str*
5.34	date(*time*)	Year month day	*str*
5.51	hour(*time*)	Hour	*num*
5.69	minute(*time*)	Minute	*num*
5.92	second(*time*)	Second	*num*
5.16	age(dt_1, dt_2)	Age	*num*
5.36	diff(dt_1, dt_2)	Period	*num*
5.108	uxt(*dt*)	Convert to UNIX time	*num*(UNIX time)
5.56	julian(*dt*)	Convert to Julian day	*num*(Julian day)

dt represents either *date* or *time*.

Table 5.17: Logical Functions

Section	Function Name	Function	Output
5.17	and($bool_1, bool_2, \cdots$)	Conjunction	*bool*
5.75	or($bool_1, bool_2, \cdots$)	Disjunction	*bool*
5.71	not(*bool*)	NOT	*bool*
5.52	if($bool, num_1, num_2$)	Check logical condition	*num*
5.52	if($bool, str_1, str_2$)		*str*
5.52	if($bool, date_1, date_2$)		*date*
5.52	if($bool, time_1, time_2$)		*time*

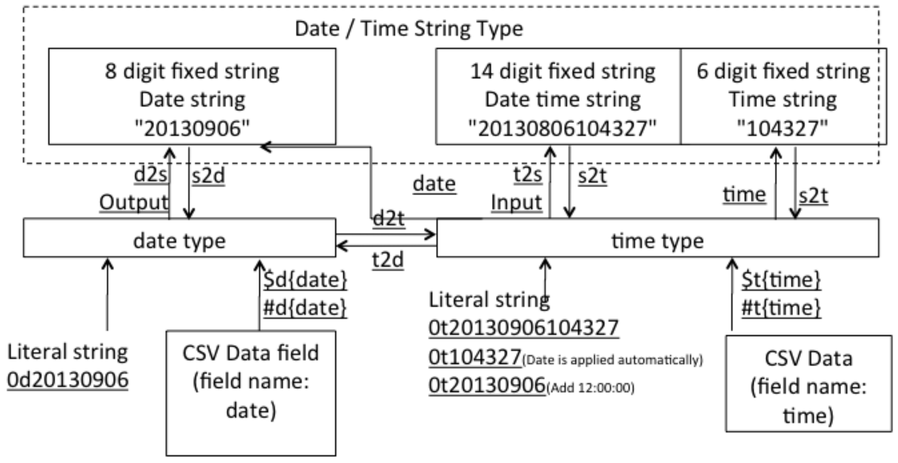


Figure 5.1: The relationship among time type, date type and various functions using September 6, 2013 at 43 minutes, 27 seconds. The solid line box indicates actual data, the dotted line indicate the functions.

Table 5.18: Row/column related functions

Section	Function Name	Function	Output Format
5.61	line()	Return the processing line number	*num*
5.105	top()	Top row	*bool*
5.25	bottom()	Last row	*bool*
5.44	fldsize()	Number of fields	*num*
5.18	argsize($str_1, str_2, \cdots$)	Number of arguments	*num*

Table 5.19: NULL value related functions

Section	Function Name	Function	Output Format
5.73	nulln()	NULL value	*num*
5.73	nulls()		*str*
5.73	nulld()		*date*
5.73	nullt()		*time*
5.73	nullb()		*bool*
5.55	isnull(*num*)	NULL value check	*bool*
5.55	isnull(*str*)		*bool*
5.55	isnull(*date*)		*bool*
5.55	isnull(*time*)		*bool*
5.55	isnull(*bool*)		*bool*
5.31	countnull($num_1, num_2, \cdots$)	Number of NULL values	*num*
5.31	countnull($str_1, str_2, \cdots$)		*num*
5.31	countnull($date_1, date_2, \cdots$)		*num*
5.31	countnull($time_1, time_2, \cdots$)		*num*
5.31	countnull($bool_1, bool_2, \cdots$)		*num*

Table 5.20: Type conversion related functions

5.112	*num*	*str*	*date*	*time*	*bool*
num		n2s(*num*)			n2b(*num*)
str	s2n(*str*)		s2d(*str*)	s2t(*str*)	s2b(*str*)
date		d2s(*date*)		d2t(*date*)	
time		t2s(*time*)	t2d(*time*)		
bool	b2n(*bool*)	b2s(*bool*)			

Each cell corresponds to the conversion function from the labels in the top row to labels in the left column. Empty cells means that conversion function is not available.

5.14 abs - Absolute value

Format: abs(*num*)

Compute absolute value of *num*.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,1.0
2,-2.5
3,
4,0
$ mcal c='abs(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=abs(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,1.0,1
2,-2.5,2.5
3,,
4,0,0
```

5.15 acos - Inverse Cosine (arccosine)

Format: acos(*num*)

Compute arccosine (inverse cosine). The function evaluates the principal value ranges of $-1.0 \sim 1.0$, and returns values within the parameter of $0.0 \sim \pi$.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,-1.0
2,0
3,1.0
4,
5,2
$ mcal c='acos(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=acos(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,-1.0,3.141592654
2,0,1.570796327
3,1.0,0
4,,
5,2,
```

5.16 age - Age

Format 1: age(Birth dateYYYYMMDD,*date*)

Format 2: age(Birth dateYYYYMMDD,*time*)

Computes age with *date* (format 1) and *time* (format 2). Conversion of date or time values to birth date (in year, month, day format).

Example

Example 1: Basic Example

Compute the age from date of birth to a fixed point on calendar on September 1, 2013.

```
$ more dat1.csv
id,dob
1,19641010
2,20000101
3,
4,19770812
$ mcal c='age($d{dob},0d20130901)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=age($d{dob},0d20130901) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,dob,rsl
1,19641010,48
2,20000101,13
3,,
4,19770812,36
```

5.17 and - Logical Conjunction

Format: and($bool_1, bool_2, \cdots$)

Calculate the logical conjunction of boolean value $bool_i$. Please refer to Table 5.8 on boolean value table with NULL values.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,b1,b2,b3
1,1,0,1
2,1,1,1
3,1,,1
4,1,1,1
$ mcal c='and($b{b1},$b{b2},$b{b3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=and($b{b1},$b{b2},$b{b3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,b1,b2,b3,rsl
1,1,0,1,0
2,1,1,1,1
3,1,,1,
4,1,1,1,1
```

Example 2: Example of using wildcard

Specify columns names that start with `b` as in (`b1,b2,b3`) using a wildcard in column name such as "`b*`".

```
$ mcal c='and($b{b*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=and($b{b*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,b1,b2,b3,rsl
1,1,0,1,0
2,1,1,1,1
3,1,,1,
4,1,1,1,1
```

5.18 argsize - Number of Arguments

Format 1: argsize($str_1, str_2, \cdots$)

Return the number of character strings specified by str_i. The column names can be specified using wildcard. It is possible to count the number of items using wild card. An important point to note is that this function is only compatible with character string format.

Example

Example 1: Basic Example

Count the number of column names that start with "v".

```
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,,
$ mcal c='argsize($s{v*})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=argsize($s{v*}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,3
2,-5,2,1,3
3,1,,3,3
4,,,,3
```

5.19 asin - Inverse Sine

Format: asin(*num*)

Compute arcsine (inverse sine). The function evaluates the principal value ranges from $-1.0 \sim 1.0$, and returns values within the parameter of $-\pi/2 \sim \pi/2$.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,-1.0
2,0
3,1.0
4,
5,2
$ mcal c='asin(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=asin(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,-1.0,-1.570796327
2,0,0
3,1.0,1.570796327
4,,
5,2,
```

5.20 atan - Inverse Tangent

Format: atan(*num*)

Compute arctangent (inverse tangent). The function evaluates the principal value ranges from $-\infty \sim \infty$, and returns values within the parameter of $-\pi/2 \sim \pi/2$.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,-1.0
2,0
3,1.0
4,
5,1.0e+10
$ mcal c='atan(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=atan(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,-1.0,-0.7853981634
2,0,0
3,1.0,0.7853981634
4,,
5,1.0e+10,1.570796327
```

5.21 atan2 - Angle of coordinates

Format: atan2(num_1, num_2)

Return the angle in radians formed by the axis and line segment of x, y coordinates (num_1, num_2) and origin.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,x,y
1,5,10
2,10,20
3,-1,0
4,0,0
5,,
$ mcal c='atan2(${x},${y})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=atan2(${x},${y}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,x,y,rsl
1,5,10,1.107148718
2,10,20,1.107148718
3,-1,0,3.141592654
4,0,0,0
5,,,
```

5.22 avg - Average

Format: avg($num_1, num_2, \cdots$)

Compute the average of numbers given in num_i. The function ignore NULL values, and return NULL result if all values in input is NULL.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,,
$ mcal c='avg(${v1},${v2},${v3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=avg(${v1},${v2},${v3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,2
2,-5,2,1,-0.6666666667
3,1,,3,2
4,,,,
```

Example 2: Example of using wildcard

Specify columns names that start with `v` (`v1,v2,v3`) using a wildcard in column name such as `v*`.

```
$ mcal c='avg(${v*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=avg(${v*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3,rsl
1,1,2,3,2
2,-5,2,1,-0.6666666667
3,1,,3,2
4,,,,
```

5.23 binomdist Cumulative probability of binomial distribution

Format: binomdist(number of successes k, number of trials n, probability of success p) This command calculates cumulative probability $P[Xk]$ regarding probability variable X that is in accordance with the binomial distribution $B(n, p)$ of the number of trials n and the probability of success p. Parameters k, n, and p can be specified as a constant for a field. The calculation results will be null where $k > n$, $p < 0$, or $p > 1$.

Examples

Example 1: Example 1: Basic example

```
$ mcal c='binomdist(${k},${n},${p})' a=prob i=dat1.csv o=rsl1.csv
#END# kgcal a=prob c=binomdist(${k},${n},${p}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
k,n,p,prob
0,2,0.5,0.25
1,2,0.5,0.75
2,2,0.5,1
4,10,0.2,0.9672065024
40,100,0.2,0.9999987078
3,2,0.2,
1,2,1.2,
```

5.24 berrand Bernoulli random numbers

Format: berrand(probability[, random number seed])

This command generates random numbers for x at the specified probability p regarding Bernoulli distribution $f(x) = px(1p)1xx = 0, 1$. This function returns values of a logical type where $x = 0$ is false and $x = 1$ is true. The probability p can be specified only as a constant, and no fieldname can be specified. From the same random number seed, random numbers of the same series are generated. The range of the specifiable random number seeds is -2147483648 to +2147483647. When the random number seed is omitted, a different random number seed is used according to the time (in units of 1/1000 sec). Random number generation is based on the Mersenne-Twister scheme (original author 's page,boost library).

Examples

Example 1: Example 1: Basic example

Bernoulli random numbers are generated at probability $p = 0 : 2$. Since a random number seed is specified, the same random number series will be generated no matter how many times the script is run.

```
$ mnewnumber l=10 a=num |
$ mcal c='berrand(0.2,111)' a=rand o=rsl1.csv
#END# kgNewnumber a=num l=10
#END# kgcal a=rand c=berrand(0.2,111) o=rsl1.csv
$ more rsl1.csv
num,rand
0,0
1,1
2,1
3,0
4,0
5,0
6,0
7,0
8,0
9,0
```

5.25 bottom - Last row

Format: bottom()

Return true if record is positioned in the last row, otherwise return false.

Example

Example 1: Basic Example

```
$ more dat1.csv
val
1
2
3
4
$ mcal c='bottom()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=bottom() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
val,rsl
1,0
2,0
3,0
4,1
```

5.26 capitalize - Capitalize first character

Format: capitalize(*str*)

Change the first character of the string to uppercase. This function does not affect non-alphabet strings.

Example

Example 1: Basic Example

Convert the first character of *str* field to uppercase.

```
$ more dat1.csv
id,str
1,abc
2,aBd
3,
4,#abc
$ mcal c='capitalize($s{str})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=capitalize($s{str}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,abc,Abc
2,aBd,ABd
3,,
4,#abc,#abc
```

5.27 cat - Concatenate Character String

Format: cat($token, str_1, str_2, \cdots$)

Concatenate the specified str_i in order using *token* as the delimiter to create a character string. If an empty character `""` is specified, the list of strings are simply concatenated.

Examples

Example 1: Basic Example

Concatenate 3 columns `str1,str2,str3` with the inclusion of `"#"` as delimiter token in between characters.

```
$ more dat1.csv
id,str1,str2,str3
1,abc,def,ghi
2,A,,CDE
3,,,
4,,,XY
$ mcal c='cat("#",$s{str1},$s{str2},$s{str3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=cat("#",$s{str1},$s{str2},$s{str3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str1,str2,str3,rsl
1,abc,def,ghi,abc#def#ghi
2,A,,CDE,A##CDE
3,,,,##
4,,,XY,##XY
```

Example 2: Empty token

```
$ mcal c='cat("",$s{str1},$s{str2},$s{str3})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=cat("",$s{str1},$s{str2},$s{str3}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,str1,str2,str3,rsl
1,abc,def,ghi,abcdefghi
2,A,,CDE,ACDE
3,,,,
4,,,XY,XY
```

Example 3: Example using wildcard

Use wildcard to specify columns names that start with `str` (`str1,str2,str3`) such as `str*`.

```
$ mcal c='cat("",$s{str*})' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=cat("",$s{str*}) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,str1,str2,str3,rsl
1,abc,def,ghi,abcdefghi
2,A,,CDE,ACDE
3,,,,
4,,,XY,XY
```

5.28 ceil - Ceiling

Format: ceil(*num*,base)

Round up *num* based on multiples of the base number, and returns the smallest integer that is greater than or equal to *num*.

For example, given ceil(3.42,05), the numerical value will be rounded up by scales of 0.5 that is greater than 3.42. Thus, the rounded number at base 0.5 becomes 3.5. When base is not specified as an argument, the default base is set at 1. This is equivalent to rounding up to the nearest integer after the decimal.

Examples

Example 1: Basic Example

Truncate all digits after decimal point.

```
$ more dat1.csv
id,val
1,3.28
2,3.82
3,
4,-0.6
$ mcal c='floor(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=floor(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.28,3
2,3.82,3
3,,
4,-0.6,-1
```

Example 2: Basic Example

Truncate the second digit after decimal point.

```
$ mcal c='floor(${val},0.1)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=floor(${val},0.1) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val,rsl
1,3.28,3.2
2,3.82,3.8
3,,
4,-0.6,-0.6
```

Example 3: Round to base 0.5

Rounding to the nearest 0.5.

```
$ mcal c='floor(${val},0.5)' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=floor(${val},0.5) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,val,rsl
1,3.28,3
2,3.82,3.5
3,,
4,-0.6,-1
```

Example 4: Round to base 10

Rounding to the nearest 10th digit.

```
$ more dat2.csv
id,val
1,1341.28
2,188
3,1.235E+3
4,-1.235E+3
$ mcal c='floor(${val},10)' a=rsl i=dat2.csv o=rsl4.csv
#END# kgcal a=rsl c=floor(${val},10) i=dat2.csv o=rsl4.csv
$ more rsl4.csv
id,val,rsl
1,1341.28,1340
2,188,180
3,1.235E+3,1230
4,-1.235E+3,-1240
```

5.29 cos - Cosine

Format: cos(r)

Compute radians of r using cosine function.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.141592
2,1.047197
3,
4,6.283185
$ mcal c='cos(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=cos(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.141592,-1
2,1.047197,0.5000004774
3,,
4,6.283185,1
```

5.30 cosh - Hyperbolic Cosine

Format: cosh(r)

Compute hyperbolic cosine.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.141592
2,-1.047197
3,
4,6.283185
$ mcal c='cosh(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=cosh(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.141592,11.59194573
2,-1.047197,1.600286169
3,,
4,6.283185,267.7466792
```

5.31 countnull - Sum total

Format 1: countnull($num_1, num_2, \cdots$)

Format 2: countnull($str_1, str_2, \cdots$)

Format 3: countnull($date_1, date_2, \cdots$)

Format 4: countnull($time_1, time_2, \cdots$)

Format 5: countnull($bool_1, bool_2, \cdots$)

Return the number of NULL values in num_i (same for other types).

Examples

Example 1: Basic Example

```
$ more dat1.csv
a,b,c,d
1,,3,4
1,,,
,,,
$ mcal c='countnull(${a},${b},${c},${d})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=countnull(${a},${b},${c},${d}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
a,b,c,d,rsl
1,,3,4,1
1,,,,3
,,,,4
```

Example 2: Specify other types of data format in field with wildcard character

```
$ mcal c='countnull($s{*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=countnull($s{*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
a,b,c,d,rsl
1,,3,4,1
1,,,,3
,,,,4
```

5.32 d2julian: date-Change date to a Julian day

Format: d2julian(*date*) Use this function to obtain a Julian day from the date.

```
A Julian day is the number of days from noon of January 1, 4713 B.C. (universal time).
The effective range of the Gregorian calendar, however, is 1400-1-1 to 9999-12-31, so the corresponding ra
NULL is output if the result is outside the range.
```

Usage

```
c=d2julian(0d20080822)
```

Example

Table 5.21: Input data

Date	Time	Number
20020824	20020824145408	10660
20020622	20020622173449	22740
20020824	20020824145408	14800
20021009	20021009095743	54510
20020121	20020121173449	18750

Examples based on the above data are shown below.

Execution example1)

The value in the " Date " field is input and converted into a Julian day. A " Julian day " field is newly created, and the result is output.

```
------------------------------------------------
mcal c='d2julian($d{Date})' a="Julian day" i=date.csv o=od2julian.csv
------------------------------------------------
```

Table 5.22: Output file(od2julian.csv)

Date	Time	Number	Julian day
20020824	20020824145408	10660	2452511
20020622	20020622173449	22740	2452448
20020824	20020824145408	14800	2452511
20021009	20021009095743	54510	2452557
20020121	20020121173449	18750	2452296

Return to mcal - date related

5.33 day - Day

Format 1: day(*date*)

Format 2: day(*time*)

Format 3: days(*date*)

Format 4: days(*time*)

Extract day from *time* or *date*. The function shown in format 1 and 2 returns the numeric value, the function in format 3 and 4 returns 2-digit fixed-length string.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,date
1,20000101
2,20121021
3,
4,19770812
$ mcal c='day($d{date})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=day($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,20000101,1
2,20121021,21
3,,
4,19770812,12
```

Example 2: Time format

```
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19770812122212
$ mcal c='day($t{time})' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=day($t{time}) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,1
2,20121021111213,21
3,,
4,19770812122212,12
```

5.34 date - Year Month Day

Format: date(*time*)

Extract 8-digit fixed length string of Year Month Day (YYYMMDD) from *time*.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19770812122212
$ mcal c='date($t{time})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=date($t{time}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,rsl
1,20000101000000,20000101
2,20121021111213,20121021
3,,
4,19770812122212,19770812
```

5.35 degree - Degree

Format: degree(r)

Compute the degree corresponding to radian r.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.141592
2,1.047197
3,
4,6.283185
$ mcal c='degree(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=degree(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.141592,179.9999626
2,1.047197,59.99996842
3,,
4,6.283185,359.9999824
```

5.36 diff - Calculate Interval

Format 1: diffyear($date_1, date_2$)

Format 2: diffyear($time_1, time_2$)

Format 3: diffmonth($date_1, date_2$)

Format 4: diffmonth($time_1, time_2$)

Format 5: diffday($date_1, date_2$)

Format 6: diffday($time_1, time_2$)

Format 7: diffhour($time_1, time_2$)

Format 8: diffminute($time_1, time_2$)

Format 9: diffsecond($time_1, time_2$)

Format 10: diffusecond($time_1, time_2$)

If the two arguments are date types, compute the interval between $date_1$ and $date_2$ ($date_1$ - $date_2$), where the interval is measured in terms of years (diffyear), months (diffmonth), days (diffday). If the two arguments are time types, compute the interval between $time_1$ and $time_2$ ($time_1$ - $time_2$), where the interval is measured in terms of years (diffyear), months (diffmonth), days (diffday), hours (diffhour), minutes (diffminute) , seconds (diffsecond), and microseconds (diffusecond). Fractions are rounded up.

Examples

Example 1: Compute the interval in terms of months

Compute the number of months during the period between the value in the `date` column and September 1, 2013.

```
$ more dat1.csv
id,date
1,19641010
2,20000101
3,20130831
4,20130901
5,20130902
$ mcal c='diffday(0d20130901,$d{date})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=diffday(0d20130901,$d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,19641010,17858
2,20000101,4992
3,20130831,1
4,20130901,0
5,20130902,-1
```

Example 2: Compute the interval in terms of minutes

Compute the minutes between the value in the `time` column and January 1, 2012 00 hours 00 minutes 00 seconds.

```
$ more dat2.csv
id,time
1,20120101000000
2,20120101011112
3,
4,20111231235000
5,20111231235000.123456
$ mcal c='diffmonth(0t20120101000000,$t{time})' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=diffmonth(0t20120101000000,$t{time}) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20120101000000,0
2,20120101011112,-1
```

```
3,,
4,20111231235000,0
5,20111231235000.123456,0
```

Example 3: Compute the interval in terms of microseconds

Compute the seconds and microseconds between the value in the `time` column and January 1, 2012 00 hours 00 minutes 00 seconds.

```
$ mcal c='diffsecond(0t20120101000000,$t{time})' a=rsl i=dat2.csv o=rsl3.csv
#END# kgcal a=rsl c=diffsecond(0t20120101000000,$t{time}) i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,time,rsl
1,20120101000000,0
2,20120101011112,-4272
3,,
4,20111231235000,600
5,20111231235000.123456,599
$ mcal c='diffusecond(0t20120101000000,$t{time})' a=rsl i=dat2.csv o=rsl4.csv
#END# kgcal a=rsl c=diffusecond(0t20120101000000,$t{time}) i=dat2.csv o=rsl4.csv
$ more rsl4.csv
id,time,rsl
1,20120101000000,0
2,20120101011112,-4272
3,,
4,20111231235000,600
5,20111231235000.123456,599.876544
```

5.37 dist - Distance

Format: dist(type,$num_1, num_2, \cdots, n_k, num_{k+1}, num_{k+2}, \cdots, num_{2k}$)

Compute the distance between two dimension vectors $(num_1, num_2, \cdots, n_k), (num_{k+1}, num_{k+2}, \cdots, num_{2k})$. Refer to `msim` for detailed definitions.

- `euclid`: Euclidean distance
- `cityblock`: City Block distance
- `hamming`: Hamming distance

Hamming distance must be specified in character string format(refer to the example below).

Examples

Example 1: Euclidean distance

```
$ more dat1.csv
id,x1,y1,x2,y2
1,0,0,1,1
2,0,1,2,0
3,,,,
$ mcal c='dist("euclid",${x1},${y1},${x2},${y2})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=dist("euclid",${x1},${y1},${x2},${y2}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,x1,y1,x2,y2,rsl
1,0,0,1,1,1.414213562
2,0,1,2,0,2.236067977
3,,,,,
```

Example 2: City Block distance

```
$ mcal c='dist("cityblock",${x1},${y1},${x2},${y2})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=dist("cityblock",${x1},${y1},${x2},${y2}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,x1,y1,x2,y2,rsl
1,0,0,1,1,2
2,0,1,2,0,3
3,,,,,
```

Example 3: Hamming distance

Hamming distance must be specified in character string format for the calculation of Hamming distance.

```
$ more dat2.csv
id,x1,y1,x2,y2
1,a,b,a,c
2,0,1,0,1
3,,,,
$ mcal c='dist("hamming",$s{x1},$s{y1},$s{x2},$s{y2})' a=rsl i=dat2.csv o=rsl3.csv
#END# kgcal a=rsl c=dist("hamming",$s{x1},$s{y1},$s{x2},$s{y2}) i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,x1,y1,x2,y2,rsl
1,a,b,a,c,1
2,0,1,0,1,2
3,,,,,
```

5.38 distgps - GPS Distance

Format: distgps(latitude1,longitude1,latitude2,longitude2[,orientation])

Find out the straight-line / direct distance (km unit) between two points based on the latitude and longtitude coordinates. The latitude and longitude can be expressed in signed decimal degrees without compass direction, where positive indicates north/east, negative indicates west/south, on the basis of a spherical earth. The latitude and longtitude is represent in heading of 60 degrees, and the format is expressed as degree, minute, second (d, m, s). The values must be converted to base of 10 for use with this function. Base of 10 coordinates can be calculated by $d + m/60 + s/60/60$.

For example, the distance between Osaka (north latitude 34.702398,east longitude 135.495188) and Tokyo (north latitude 35.681391,east longitude 139.766103) is calculated as follows.

```
distgps(34.702398,135.495188,35.681391,139.766103)
```

In addition, the distance from Everest (north latitude 32.655556,east longitude 79.015833) to Aconcagua (southern latitude 27.987778,west longitude 86.944444) is specified as follows.

```
distgps(32.655556,79.015833,-27.987778,-86.944444)
```

Example

Example 1: Basic Example

```
$ more dat1.csv
point1,point2,lat1,lon1,lat2,lon2
osaka,tenma,34.702398,135.495188,34.704923,135.512233
osaka,tokyo,34.702398,135.495188,35.681391,139.766103
osaka,kobe,34.702398,135.495188,34.679453,135.178221
osaka,Fuji,34.702398,135.495188,35.360556,138.727500
Evelest,Aconcagua,32.655556,79.015833,-27.987778,-86.944444
Denali,Kilimanjaro,63.069444,-151.007222,-3.075833,37.353333
$ mcal c='distgps(${lat1},${lon1},${lat2},${lon2})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=distgps(${lat1},${lon1},${lat2},${lon2}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
point1,point2,lat1,lon1,lat2,lon2,rsl
osaka,tenma,34.702398,135.495188,34.704923,135.512233,1.585046048
osaka,tokyo,34.702398,135.495188,35.681391,139.766103,405.774306
osaka,kobe,34.702398,135.495188,34.679453,135.178221,29.12042213
osaka,Fuji,34.702398,135.495188,35.360556,138.727500,304.7527532
Evelest,Aconcagua,32.655556,79.015833,-27.987778,-86.944444,16956.12242
Denali,Kilimanjaro,63.069444,-151.007222,-3.075833,37.353333,11362.37758
```

5.39 dow - Day of week

Format 1: dow(*dt*) Number of day of week number (1-7)

Format 2: dowj(*dt*) Day of week in Japanese

Format 3: dowe(*dt*) Day of week in English

Format 4: dowes(*dt*) Day of week abbreviation in English

Return the day of week from *date* and *time*. The notions of day of week is indicated in format 1 to 4 above. Weekday number is follows the ISO8601 date and time standard, number 1 to 7 corresponds to Monday to Sunday.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,date
1,20000101
2,20121021
3,
4,19770812
$ mcal c='dow($d{date})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=dow($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,20000101,6
2,20121021,7
3,,
4,19770812,5
```

Example 2: Weekday in Japanese

```
$ mcal c='dowj($d{date})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=dowj($d{date}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,date,rsl
1,20000101,土
2,20121021,日
3,,
4,19770812,金
```

Example 3: Weekday in English

```
$ mcal c='dowe($d{date})' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=dowe($d{date}) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,date,rsl
1,20000101,Saturday
2,20121021,Sunday
3,,
4,19770812,Friday
```

Example 4: Weekday Abbreviated in English

```
$ mcal c='dowes($d{date})' a=rsl i=dat1.csv o=rsl4.csv
#END# kgcal a=rsl c=dowes($d{date}) i=dat1.csv o=rsl4.csv
$ more rsl4.csv
id,date,rsl
1,20000101,Sat
2,20121021,Sun
```

```
3,,
4,19770812,Fri
```

Example 5: Time format

```
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19770812122212
$ mcal c='dow($t{time})' a=rsl i=dat2.csv o=rsl5.csv
#END# kgcal a=rsl c=dow($t{time}) i=dat2.csv o=rsl5.csv
$ more rsl5.csv
id,time,rsl
1,20000101000000,6
2,20121021111213,7
3,,
4,19770812122212,5
```

5.40 e - Napier 's Constant

Format: e()

Return Napier 's Constant (*e*).

Example

Example 1: Basic Example

```
$ more dat1.csv
id
1
2
$ mcal c='e()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=e() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,2.718281828
2,2.718281828
```

5.41 exp - Exponential Function

Format: exp(*num*)

A type of power function using *e* (Napier 's constant) as base to compute *num*.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,exponent
1,1
2,-1
3,
4,0
5,0.5
$ mcal c='exp(${exponent})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=exp(${exponent}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,exponent,rsl
1,1,2.718281828
2,-1,0.3678794412
3,,
4,0,1
5,0.5,1.648721271
```

5.42 factorial - Factorial

Format: factorial(*num*)

Returns the factorial of *num*. NULL value is returned if the result exceeds the maximum value of real number.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,1
2,5
3,
4,10000
$ mcal c='factorial(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=factorial(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,1,1
2,5,120
3,,
4,10000,
```

Example 2: Example of using constants

Calculate the factorial of 5. When constants is used as an argument, all rows will return the same result.

```
$ mcal c='factorial(5)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=factorial(5) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val,rsl
1,1,120
2,5,120
3,,120
4,10000,120
```

5.43 fixlen - Convert of Fixed Length String

Format 1: length(*str*,length,position,padding character)

Format 2: lengthw(*str*,length,position,padding character)

Convert *str* to fixed length string. Specify padding characters for left justified or right justified *str* if it is less than specified length. Specify the justification using "L" or "R" in the “ position ” parameter. Define the character to be embedded in the “ padding character ” parameter. Note that if the length of *str* exceeds the defined length, the ending characters for right justified character strings or beginning characters for left justified character strings will be cut off.

Use `fixlenw` function for fixed-length conversion on multi-byte characters.

Examples

Example 1: Basic Example

Convert values in the `str` column to 5 character fixed-length string. Right justified ("R") the strings and fill the empty positions with "#" for strings with less than 5 characters.

```
$ more dat1.csv
id,str
1,abc
2,123
3,
4,1234567
$ mcal c='fixlen($s{str},5,"R","#")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=fixlen($s{str},5,"R","#") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,abc,##abc
2,123,##123
3,,#####
4,1234567,34567
```

Example 2: Left justified

Fill empty positions for left justified ("L") text with "#".

```
$ mcal c='fixlen($s{str},5,"L","#")' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=fixlen($s{str},5,"L","#") i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,abc,abc##
2,123,123##
3,,#####
4,1234567,12345
```

Example 3: Multibyte string

```
$ more dat2.csv
id,str
1,こんにちは
2,大阪
$ mcal c='fixlenw($s{str},4,"R","埋")' a=rsl i=dat2.csv o=rsl3.csv
#END# kgcal a=rsl c=fixlenw($s{str},4,"R","埋") i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,str,rsl
1,こんにちは,んにちは
2,大阪,埋埋大阪
```

5.44 fldsize - Number of Fields

Format: fldsize()

Return the number of fields in the input file. MCMD assumes all records in the input data has same number of fields, thus the resulting value from the function applies to all records.

Examples

Example 1: Basic Example

```
$ more dat1.csv
a,b,c,d
1,2,3,4
2,3,4,5
3,,,
4,x,y,z
$ mcal c='fldsize()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=fldsize() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
a,b,c,d,rsl
1,2,3,4,4
2,3,4,5,4
3,,,,4
4,x,y,z,4
```

5.45 floor - Rounding Down

Format: floor(*num*,base)

Round *num* down to the nearest integer. This function round down to the maximum integer that is not greater than *num*. For example, in the argument floor(3.82,0.5), the decimal point of 3.82 is above the scale point of 0.5, thus, rounding down to the nearest 0.5 base returns 3.5. The default value of base is 1 if the argument is not defined. This is equivalent to rounding to an integer value by truncating all decimal digits.

Examples

Example 1: Basic Example

Truncate all digits after decimal point.

```
$ more dat1.csv
id,val
1,3.28
2,3.82
3,
4,-0.6
$ mcal c='floor(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=floor(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.28,3
2,3.82,3
3,,
4,-0.6,-1
```

Example 2: Basic Example

Truncate the second digit after decimal point.

```
$ mcal c='floor(${val},0.1)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=floor(${val},0.1) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val,rsl
1,3.28,3.2
2,3.82,3.8
3,,
4,-0.6,-0.6
```

Example 3: Round to base 0.5

Rounding to the nearest 0.5.

```
$ mcal c='floor(${val},0.5)' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=floor(${val},0.5) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,val,rsl
1,3.28,3
2,3.82,3.5
3,,
4,-0.6,-1
```

Example 4: Round to base 10

Rounding to the nearest 10th digit.

```
$ more dat2.csv
id,val
1,1341.28
2,188
3,1.235E+3
```

```
4,-1.235E+3
$ mcal c='floor(${val},10)' a=rsl i=dat2.csv o=rsl4.csv
#END# kgcal a=rsl c=floor(${val},10) i=dat2.csv o=rsl4.csv
$ more rsl4.csv
id,val,rsl
1,1341.28,1340
2,188,180
3,1.235E+3,1230
4,-1.235E+3,-1240
```

5.46 format - Formatted Output

Format: format(*num*, output format)

mcal internally manages floating point numbers. This function changes the output format of the number by specifying the output format available in the C language printf function. The 3 available formats are as follows.

- %f: Decimal format
- %e,%E: Exponent format (Specify %E to use uppercase E as exponent notation)
- %g,%G: Automatically select output format of f or e (Specify %G for uppercase notation)

For example, the decimal format is expressed as `0.00726` can be expressed as `7.260000e-03` in exponent format, similarly, the exponent format of `1265` is represented as `1.265000e+03`.

By specifying the `%` or plus sign after the number, the function formats the decimal digits and characters. The number format can be specified in the format of " `whole number.decimal` ". For example, the number `123.456789` with the format specifier of `%5.2f` becomes `123.46`, and the format specifier of `%8.3f` print number as `123.458` with a floating point of 8 characters and 3 characters after decimal.

Add a plus sign before the number to display the plus sign before the number. For example, `123.456789` with the format specifier of `%+5.2f` format the number as `+123.46`.

Besides the format highlighted above, it is possible to include any character string in the format. For example, the number `250` can be formatted as " `Total 250 Yen` " using the format specifier of " Total

Examples

Example 1: Basic Example

Format `val` as real number with 3 decimal places.

```
$ more dat1.csv
id,val
1,0.00726
2,123.456789
3,
4,-0.335
$ mcal c='format(${val},"%8.3f")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=format(${val},"%8.3f") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,0.00726,   0.007
2,123.456789, 123.457
3,,
4,-0.335,  -0.335
```

Example 2: Display the exponent

Format `val` in exponential notation.

```
$ mcal c='format(${val},"%e")' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=format(${val},"%e") i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val,rsl
1,0.00726,7.260000e-03
2,123.456789,1.234568e+02
3,,
4,-0.335,-3.350000e-01
```

5.47 fract - Fractional Part

Format: fract(*num*)

Returns the fractional part of *num*. Fractions of values in scientific notation will be output as is.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.14
2,3
3,
4,-12.56789
5,1.2345e+2
6,1.2345e-10
$ mcal c='fract(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=fract(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.14,0.14
2,3,0
3,,
4,-12.56789,-0.56789
5,1.2345e+2,0.45
6,1.2345e-10,1.2345e-10
```

5.48 gcd - Greatest Common Divisor

Format: gcd(num_1, num_2)

Find out the greatest common divisor of num_1 and num_2. Real number will be rounded down to whole number.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val1,val2
1,12,36
2,6,5
3,,
4,12.1,36.2
$ mcal c='gcd(${val1},${val2})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=gcd(${val1},${val2}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val1,val2,rsl
1,12,36,12
2,6,5,1
3,,,
4,12.1,36.2,12
```

Example 2: Example of constants

Find out the greatest common divisor of `val1` column and 36.

```
$ mcal c='gcd(${val1},36)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=gcd(${val1},36) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val1,val2,rsl
1,12,36,12
2,6,5,6
3,,,
4,12.1,36.2,12
```

5.49 hasspace - Search Whitespace Characters

Format 1: hasspace(*str*, length)

Format 2: hasspacew(*str*, length)

This function returns true if there is whitespace characters in character string *str* and false if not. Whitespace characters is represented as ASCII code 0X20 to 0x09-0x0d. The respective ASCII code are represented as: single byte space (0x20), horizontal tab (0x09), new line (0x0a), vertical tabulation (0x0b), new page (0x0c), carriage return (0x0d). Use `hasspacew` to search for multibyte white space character.

Examples

Example 1: Basic Example

Returns true if `str` column contains white space characters. The first row where `id=1`contains single-byte space character, in addition, the second row where `id=2` contains tab character, thus, these two rows return true. However, records where the function returns false are `id=4` which contains carriage return, and `id=3` which contains double-byte space character.

```
$ more dat1.csv
id,str
1,a b
2,ab        c
3,ab　c
4,
5,"aa
bb"
$ mcal c='hasspace($s{str})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=hasspace($s{str}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,a b,1
2,ab        c,1
3,ab　c,0
4,,0
5,"aa
bb",1
```

Example 2: Multibyte character

Use `hasspacew` function to detect double character space.

```
$ mcal c='hasspacew($s{str})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=hasspacew($s{str}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,a b,1
2,ab        c,1
3,ab　c,1
4,,0
5,"aa
bb",1
```

5.50 heron - Area of Triangle

Format: sim(type,$num_1, num_2, \cdots, n_k, num_{k+1}, num_{k+2}, \cdots, num_{2k}, num_{2k+1}, num_{2k+2}, \cdots, num_{3k}$)

Computes the area of a triangle given the 3 points of coordinates at k-dimensional space $(num_1, num_2, \cdots, n_k), (num_{k+1}, num_{k+2}$
at k-dimensional space.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,x1,y1,x2,y2,x3,y3
1,0,0,1,0,0,1
2,0,0,0,2,2,0
4,,,,,,
3,0,0,1,1,2,2
$ mcal c='heron(${x1},${y1},${x2},${y2},${x3},${y3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=heron(${x1},${y1},${x2},${y2},${x3},${y3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,x1,y1,x2,y2,x3,y3,rsl
1,0,0,1,0,0,1,0.5
2,0,0,0,2,2,0,2
4,,,,,,,
3,0,0,1,1,2,2,0
```

5.51 hour - Hour

Format 1: hour(*time*) Numerical value

Format 2: hours(*time*) 2 digit fixed length string

Extract hour from *time*. The function can be used in different purposes as shown in format 1,2.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19770812122212
$ mcal c='hour($t{time})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=hour($t{time}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,rsl
1,20000101000000,0
2,20121021111213,11
3,,
4,19770812122212,12
```

Example 2: Output character string

```
$ mcal c='hours($t{time})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=hours($t{time}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,00
2,20121021111213,11
3,,
4,19770812122212,12
```

5.52 if - Conditional Statements

Format: if($bool, num_1, num_2$), if($bool, str_1, str_2$), if($bool, date_1, date_2$),if($bool, time_1, time_2$),if($bool, bool_1, bool_2$)

If the first parameter is true, return the second parameter, otherwise, return the third parameter. If the first parameter includes NULL value, return NULL value. Note that second and third argument should use the same parameter type.

Examples

Example 1: Basic Example

If the value in time column is less than 120000, return "AM", otherwise, return "PM".

```
$ more dat1.csv
id,time
1,101215
2,210110
3,
4,120001
$ mcal c='if(${time}<=120000,"AM","PM")' a=ampm i=dat1.csv o=rsl1.csv
#END# kgcal a=ampm c=if(${time}<=120000,"AM","PM") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,ampm
1,101215,AM
2,210110,PM
3,,
4,120001,PM
```

Example 2: Different parameter types

The function returns an error if the character format is different in the second and third parameter.

```
$ mcal c='if(${time}<=120000,"am",1)' a=ampm i=dat1.csv o=rsl2.csv
#ERROR# unknown function or operator: if_BSN (kgcal)
$ more rsl2.csv
```

Example 3: Return boolean value on conditional statements

Read the value in the column using `$b{fieldname}`, if the value is " 1 " return true, if it is " 0 " return false, other values will be treated as NULL.

```
$ more dat2.csv
id,val
1,1
2,0
3,
4,-2
$ mcal c='if($b{val},"ture","false")' a=bool i=dat2.csv o=rsl3.csv
#END# kgcal a=bool c=if($b{val},"ture","false") i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,val,bool
1,1,ture
2,0,false
3,,
4,-2,
```

Example 4: Time format comparison

```
$ mcal c='if($t{time}<=0t120000,"am","pm")' a=ampm i=dat1.csv o=rsl4.csv
#END# kgcal a=ampm c=if($t{time}<=0t120000,"am","pm") i=dat1.csv o=rsl4.csv
$ more rsl4.csv
id,time,ampm
1,101215,am
2,210110,pm
```

```
3,,
4,120001,pm
```

Example 5: Nested if function

```
$ more dat3.csv
id,val
1,10
2,0
3,-5
4,0
$ mcal c='if(${val}>0,"plus",if(${val}<0,"minus","zero"))' a=sign i=dat3.csv o=rsl5.csv
#END# kgcal a=sign c=if(${val}>0,"plus",if(${val}<0,"minus","zero")) i=dat3.csv o=rsl5.csv
$ more rsl5.csv
id,val,sign
1,10,plus
2,0,zero
3,-5,minus
4,0,zero
```

5.53 int - Integer

Format: int(*num*)

Return the integer value of *num*. Plus and minus signs are returned as is.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.14
2,3
3,
4,-12.56789
5,1.2345e+2
6,1.2345e-10
$ mcal c='int(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=int(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.14,3
2,3,3
3,,
4,-12.56789,-12
5,1.2345e+2,123
6,1.2345e-10,0
```

5.54 match - Search

Format 1: match(search string ,$str_1, str_2, \cdots$)

Format 2: matcha(search string ,$str_1, str_2, \cdots$)

Format 3: matchs(search string ,$str_1, str_2, \cdots$)

Format 4: matchas(search string ,$str_1, str_2, \cdots$)

Search for the search string in $str_1, str_2, \cdots$, and returns true if there is a match and false otherwise.

OR search and AND search returns partial match and exact match of string. The corresponding functions are shown in Table 5.23.

Table 5.23: Input Data

	OR search	AND search
Exact match	match	matcha
Partial match	matchs	matchas

The `match` function returns true if there is an exact match of the specified character string with any of the string in $str_1, str_2, \cdots$. The `matcha` function returns true if there is an exact match with all strings in $str_1, str_2, \cdots$. `matchs` function returns true if there is partial match with any of the string in $str_1, str_2, \cdots$. `matchas` function returns true if there is partial match with all strings in $str_1, str_2, \cdots$. Refer to Table 5.8 on the boolean table of OR/AND logical operation for NULL values.

Examples

Example 1: OR exact match

Returns true if either column `f1,f2,f3` contains `1`.

```
$ more dat1.csv
id,f1,f2,f3
1,1,1,1
2,1,0,1
3,,,
4,1,,1
$ mcal c='match("1",$s{f1},$s{f2},$s{f3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=match("1",$s{f1},$s{f2},$s{f3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,f1,f2,f3,rsl
1,1,1,1,1
2,1,0,1,1
3,,,,0
4,1,,1,1
```

Example 2: AND exact match

Returns true if columns `f1,f2,f3` contains the character `"1"`.

```
$ mcal c='matcha("1",$s{f1},$s{f2},$s{f3})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=matcha("1",$s{f1},$s{f2},$s{f3}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,f1,f2,f3,rsl
1,1,1,1,1
2,1,0,1,0
3,,,,0
4,1,,1,0
```

Example 3: OR partial match

Returns true if the character string `ab` exists in either column `s1,s2,s3`.

```
$ more dat2.csv
id,s1,s2,s3
1,ab,abx,x
2,abc,xaby,xxab
3,,,
4,#ac,x,x
$ mcal c='matchs("ab",$s{s1},$s{s2},$s{s3})' a=rsl i=dat2.csv o=rsl3.csv
#END# kgcal a=rsl c=matchs("ab",$s{s1},$s{s2},$s{s3}) i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,s1,s2,s3,rsl
1,ab,abx,x,1
2,abc,xaby,xxab,1
3,,,,0
4,#ac,x,x,0
```

Example 4: AND partial match

Returns true if the character string `ab` exists in columns `s1,s2,s3`.

```
$ mcal c='matchas("ab",$s{s1},$s{s2},$s{s3})' a=rsl i=dat2.csv o=rsl4.csv
#END# kgcal a=rsl c=matchas("ab",$s{s1},$s{s2},$s{s3}) i=dat2.csv o=rsl4.csv
$ more rsl4.csv
id,s1,s2,s3,rsl
1,ab,abx,x,0
2,abc,xaby,xxab,1
3,,,,0
4,#ac,x,x,0
```

Example 5: Search for NULL value

Return true if `str` column contains NULL value.

```
$ mcal c='match(nulls(),$s{s1},$s{s2},$s{s3})' a=rsl i=dat2.csv o=rsl5.csv
#END# kgcal a=rsl c=match(nulls(),$s{s1},$s{s2},$s{s3}) i=dat2.csv o=rsl5.csv
$ more rsl5.csv
id,s1,s2,s3,rsl
1,ab,abx,x,0
2,abc,xaby,xxab,0
3,,,,1
4,#ac,x,x,0
```

5.55 isnull - Evaluate NULL value

Format: isnull(*num*), isnull(*str*), isnull(*date*), isnull(*time*), isnull(*bool*)

Determine whether NULL value is included in the value *num* (the same applies to other data type). Returns `0b1` (true) if NULL value exists, and `0b0` (false) otherwise.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,a
2,
3,b
$ mcal c='isnull(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=isnull(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,a,0
2,,1
3,b,0
```

Example 2: Specify other field types

```
$ mcal c='isnull($s{val})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=isnull($s{val}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val,rsl
1,a,0
2,,1
3,b,0
```

Example 3: Specify null character

```
$ mcal c='isnull("")' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=isnull("") i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,val,rsl
1,a,1
2,,1
3,b,1
```

5.56 julian - Julian Calendar Conversion

Format 1: julian(*date*)

Format 2: julian(*time*)

Format 3: julian2d(*num*)

Format 4: julian2t(*num*)

Format 1 and 2 converts *date* or *time* to Julian day of year. Format 3 and 4 reverse the conversion from Julian day of year to date or time. Given the date type, the beginning of the day is counted as `00:00:00`.

Examples

Example 1: Basic Example

Convert the data in the `date` column to Julian day of year with `d2julian` formula, and convert back using `julian2d` function.

```
$ more dat1.csv
id,date
1,20000101
2,20121021
3,
4,19700101
$ mcal c='julian($d{date})' a=julian i=dat1.csv o=rsl1.csv
#END# kgcal a=julian c=julian($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,julian
1,20000101,2451545
2,20121021,2456222
3,,
4,19700101,2440588
$ mcal c='julian2d(${julian})' a=date2 i=rsl1.csv o=rsl2.csv
#END# kgcal a=date2 c=julian2d(${julian}) i=rsl1.csv o=rsl2.csv
$ more rsl2.csv
id,date,julian,date2
1,20000101,2451545,20000101
2,20121021,2456222,20121021
3,,,
4,19700101,2440588,19700101
```

Example 2: Apply the function to time formatted data

```
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19700101000100
$ mcal c='julian($t{time})' a=julian i=dat2.csv o=rsl3.csv
#END# kgcal a=julian c=julian($t{time}) i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,time,julian
1,20000101000000,2451544.5
2,20121021111213,2456221.967
3,,
4,19700101000100,2440587.501
$ mcal c='julian2t(${julian})' a=time2 i=rsl3.csv o=rsl4.csv
#END# kgcal a=time2 c=julian2t(${julian}) i=rsl3.csv o=rsl4.csv
$ more rsl4.csv
id,time,julian,time2
1,20000101000000,2451544.5,20000101000000
2,20121021111213,2456221.967,20121021111228.800015
3,,,
4,19700101000100,2440587.501,19700101000126.400014
```

5.57 lcm - Least Common Denominator

Format: lcm(num_1, num_2)

Find out the least common denominator between num_1 and num_2 . Real number will be rounded down to whole number.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,val1,val2
1,5,3
2,12,4
3,,
4,5.1,3.1
$ mcal c='lcm(${val1},${val2})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=lcm(${val1},${val2}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val1,val2,rsl
1,5,3,15
2,12,4,12
3,,,
4,5.1,3.1,15
```

Example 2: Example using constant

Find out the least common denominator between column `val1` and 15.

```
$ mcal c='lcm(${val1},15)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=lcm(${val1},15) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val1,val2,rsl
1,5,3,15
2,12,4,60
3,,,
4,5.1,3.1,15
```

5.58 leapyear - Determine Leap Year

Format 1: leapyear(*date*)

Format 2: leapyear(*time*)

Determine leap year from *date* or *time*.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,date
1,20000101
2,20121021
3,
4,19770812
$ mcal c='leapyear($d{date})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=leapyear($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,20000101,1
2,20121021,1
3,,
4,19770812,0
```

Example 2: Determine leap year from time formatted data

```
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19770812122212
$ mcal c='leapyear($t{time})' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=leapyear($t{time}) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,1
2,20121021111213,1
3,,
4,19770812122212,0
```

5.59 left - Extract String from Left

Format 1: left(*str*, length)

Format 2: leftw(*str*, length)

Extract specified number of characters from the left as defined in the length parameter from the character string *str*. Use `leftw` if the string contains multibyte characters.

Examples

Example 1: Basic Example

Extract the first 3 characters in the `str` column.

```
$ more dat1.csv
id,str
1,abcdefg
2,12345678
3,
4,12
$ mcal c='left($s{str},3)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=left($s{str},3) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,abcdefg,abc
2,12345678,123
3,,
4,12,12
```

Example 2: Include multibyte character

Use `leftw` function if the string contains multibyte characters.

```
$ more dat2.csv
id,str
1,あいうえお
2,1あ2345678
3,1あ
4,ああ
$ mcal c='leftw($s{str},3)' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=leftw($s{str},3) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,あいうえお,あいう
2,1あ2345678,1あ2
3,1あ,1あ
4,ああ,ああ
```

5.60 length - Length of Character String

Format 1: length(*str*)

Format 2: lengthw(*str*)

Compute the length of character string. Use `lengthw` function to initialize wide character in *str*. Note that length of NULL value is 0.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,str
1,abc
2,3.1415
3,
4,hello world!
$ mcal c='length($s{str})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=length($s{str}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,abc,3
2,3.1415,6
3,,0
4,hello world!,12
```

Example 2: Include multibyte character

The following example uses Japanese in UTF-8 encoding. Each UTF-8 Japanese character is encoded in 3 bytes, thus, the length function returns the number of bytes rather than the number of Japanese characters.

```
$ more dat2.csv
id,str
1,こんにちは
2,大阪
$ mcal c='length($s{str})' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=length($s{str}) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,こんにちは,15
2,大阪,6
```

Example 3: Initialize wide character

The `lengthw` function converts each wide characters internally into multibyte character for computation.

```
$ mcal c='lengthw($s{str})' a=rsl i=dat2.csv o=rsl3.csv
#END# kgcal a=rsl c=lengthw($s{str}) i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,str,rsl
1,こんにちは,5
2,大阪,2
```

5.61 line - Line Number

Format: line()

Return the line number processed by `mcal` command. mcmd standarize all line numbers starting from 0, similarly, `line` function initializes the first row of data from 0.

Examples

Example 1: Basic Example

Print number starting from 0 in output.

```
$ more dat1.csv
id
1
2
3
4
$ mcal c='line()' a=no i=dat1.csv o=rsl1.csv
#END# kgcal a=no c=line() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,no
1,0
2,1
3,2
4,3
```

Example 2: Start from 1

Print number starting from 1 in output.

```
$ mcal c='line()+1' a=no i=dat1.csv o=rsl2.csv
#END# kgcal a=no c=line()+1 i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,no
1,1
2,2
3,3
4,4
```

5.62 ln - Natural Logarithm

Format: ln(*num*,base)

Compute the natural logarithm of *num*.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,10
2,2.718281828
3,
4,0
5,1
6,-8
$ mcal c='ln(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=ln(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,10,2.302585093
2,2.718281828,0.9999999998
3,,
4,0,
5,1,0
6,-8,
```

5.63 log - Logarithm

Format: log(*num*,base)

Compute logarithm of *num* with specified base.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val,base
1,100,10
2,256,2
3,,
4,2,0
5,0,2
6,1,10
7,-8,2
$ mcal c='log(${val},${base})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=log(${val},${base}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,base,rsl
1,100,10,2
2,256,2,8
3,,,
4,2,0,-0
5,0,2,
6,1,10,0
7,-8,2,
```

5.64 log10 - Common logarithm

Format: log10(*num*,base)

Compute common logarithm of *num*.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,10
2,0.1
3,
4,0
5,1
6,-8
$ mcal c='log10(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=log10(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,10,1
2,0.1,-1
3,,
4,0,
5,1,0
6,-8,
```

5.65 log2 - Log base 2

Format: log2(*num*)

Compute logarithm of *num* with base 2.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,10
2,256
3,
4,0
5,1
6,-8
$ mcal c='log2(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=log2(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,10,3.321928095
2,256,8
3,,
4,0,
5,1,0
6,-8,
```

5.66 max - Maximum Value

Format: max($num_1, num_2, \cdots$)

Compute the maximum number in num_i. NULL values are ignored and returned as NULL value.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,,
$ mcal c='max(${v1},${v2},${v3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=max(${v1},${v2},${v3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,3
2,-5,2,1,2
3,1,,3,3
4,,,,
```

Example 2: Example using wildcard

Specify columns starting from `v` (`v1,v2,v3`) using wildcard " `v*` ".

```
$ mcal c='max(${v*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=max(${v*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3,rsl
1,1,2,3,3
2,-5,2,1,2
3,1,,3,3
4,,,,
```

5.67 mid - Extract Substring

Format 1: mid(*str*, starting position, length)

Format 2: midw(*str*, starting position, length)

Extract the character string *str* from the specified starting position for a specified length. Note that the starting position starts from 0. Use `leftw` function if the string contains multibyte characters.

Example

Example 1: Basic Example

Extract the first 3 characters from the beginning in the str column.

```
$ more dat1.csv
id,str
1,abcdefg
2,12345678
3,
4,12
$ mcal c='mid($s{str},2,3)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=mid($s{str},2,3) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,abcdefg,cde
2,12345678,345
3,,
4,12,
```

Example 2: Cases containing multibyte character

Use midw function if the string contains multibyte characters.

```
$ more dat2.csv
id,str
1,あいうえお
2,1234567あ8
3,1あ
4,ああ
$ mcal c='midw($s{str},2,3)' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=midw($s{str},2,3) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,あいうえお,うえお
2,1234567あ8,345
3,1あ,
4,ああ,
```

5.68 min - Minimum Value

Format: $\min(num_1, num_2, \cdots)$

Compute the minimum number in num_i. NULL values are ignored and returned as NULL value.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,,
$ mcal c='min(${v1},${v2},${v3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=min(${v1},${v2},${v3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,1
2,-5,2,1,-5
3,1,,3,1
4,,,,
```

Example 2: Example using wildcard

Use wildcard “ `v*` ” to specify columns starting from `v` (`v1,v2,v3`).

```
$ mcal c='min(${v*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=min(${v*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3,rsl
1,1,2,3,1
2,-5,2,1,-5
3,1,,3,1
4,,,,
```

5.69 minute - Minute

Format 1: minute(*time*) Numerical value

Format 2: minutes(*time*) 2 digit fixed length string

Extract minute from *time*. The output is returned in a format listed in format 1 and 2.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19770812122212
$ mcal c='minute($t{time})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=minute($t{time}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,rsl
1,20000101000000,0
2,20121021111213,12
3,,
4,19770812122212,22
```

Example 2: Print in character string

```
$ mcal c='minutes($t{time})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=minutes($t{time}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,00
2,20121021111213,12
3,,
4,19770812122212,22
```

5.70 month - Month

Format 1: month(*dt*) Numeric month

Format 2: months(*dt*) Two digit fixed length numeric month in character string

Format 3: monthe(*dt*) Month in English

Format 4: monthes(*dt*) Abbreviation of month in English

Return month from *date* and *time* data. Month can be expressed in different formats as shown in format 1 to 4.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,date
1,20000101
2,20121021
3,
4,19770812
$ mcal c='month($d{date})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=month($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,20000101,1
2,20121021,10
3,,
4,19770812,8
```

Example 2: Fixed length character string

```
$ mcal c='months($d{date})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=months($d{date}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,date,rsl
1,20000101,01
2,20121021,10
3,,
4,19770812,08
```

Example 3: Month in English

```
$ mcal c='monthe($d{date})' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=monthe($d{date}) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,date,rsl
1,20000101,January
2,20121021,October
3,,
4,19770812,August
```

Example 4: Abbreviation of month in English

```
$ mcal c='monthes($d{date})' a=rsl i=dat1.csv o=rsl4.csv
#END# kgcal a=rsl c=monthes($d{date}) i=dat1.csv o=rsl4.csv
$ more rsl4.csv
id,date,rsl
1,20000101,Jan
2,20121021,Oct
3,,
4,19770812,Aug
```

Example 5: Extract month from time formatted data

```
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19770812122212
$ mcal c='month($t{time})' a=rsl i=dat2.csv o=rsl5.csv
#END# kgcal a=rsl c=month($t{time}) i=dat2.csv o=rsl5.csv
$ more rsl5.csv
id,time,rsl
1,20000101000000,1
2,20121021111213,10
3,,
4,19770812122212,8
```

5.71 not - Negation

Format: not(*bool*)

Returns the reversed logical value of *bool*.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,b
1,1
2,
3,0
$ mcal c='not($b{b})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=not($b{b}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,b,rsl
1,1,0
2,,
3,0,1
```

5.72 now - Current Time

Format: now()

The `now` function returns current time in time format.

Example

Example 1: Basic Example

```
$ more dat1.csv
id
1
2
$ mcal c='now()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=now() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,20171013133634
2,20171013133634
```

Example 2: Extract time from time formatted data

```
$ mcal c='time(now())' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=time(now()) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,rsl
1,133634
2,133634
```

Example 3: Extract date from time formatted data

```
$ mcal c='date(now())' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=date(now()) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,rsl
1,20171013
2,20171013
```

5.73 null - NULL Value

Format: nulln(),nulls(),nulld(),nullt(),nullb()

Return NULL value for corresponding types. It can be used to return NULL value in conjunction with the `if` function.

Example

Example 1: Basic Example

Print NULL values to column `rsl`.

```
$ more dat1.csv
id
1
2
3
$ mcal c='nulls()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=nulls() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,
2,
3,
```

Example 2: Use of if statement

Use nulln() function to match the value specified in the second parameter.

```
$ mcal c='if(${id}==1,1,nulln())' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=if(${id}==1,1,nulln()) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,rsl
1,1
2,
3,
```

Example 3: Equivalent to isnull function

```
$ mcal c='if(${val}==nulln(),"null","notNull")' a=rsl i=dat2.csv o=rsl3.csv
#END# kgcal a=rsl c=if(${val}==nulln(),"null","notNull") i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,val,rsl
1,a,
2,,
3,b,
```

5.74 nrand - Random Numbers in Normal Distribution

Generate random numbers with specified mean and standard deviation.

The same random seed will generate the same sequence of random numbers. The random seed can be specified from -2147483648 to 2147483647. If the random seed is not defined, the random numbers will be seeded according to current time (1/1000 second).

This function uses Mersenne Twister as random number generator (Author 's original website , boost library)

Example

Example 1: Basic Example

Create a set of random numbers with a standard deviation between 0 to 1 (standard normal distribution). Set the random seed as 10.

```
$ more dat1.csv
id
1
2
3
4
$ mcal c='nrand(0,1,10)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=nrand(0,1,10) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,1.161957445
2,-0.7429729875
3,-1.10165004
4,1.03149223
```

5.75 or - Logical Disjunction

Format: or($bool_1, bool_2, \cdots$)

Compute the boolean result using logical disjunction on the array of boolean values in $bool_i$. Refer to Table 5.9 on the boolean truth table with NULL values.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,b1,b2,b3
1,1,0,0
2,1,,1
3,0,,0
4,0,0,0
$ mcal c='or($b{b1},$b{b2},$b{b3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=or($b{b1},$b{b2},$b{b3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,b1,b2,b3,rsl
1,1,0,0,1
2,1,,1,1
3,0,,0,
4,0,0,0,0
```

Example 2: Example using wildcard

Use the wildcard character “ `b*` ” to specify columns starting from `b` (`b1,b2,b3`).

```
$ mcal c='or($b{b*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=or($b{b*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,b1,b2,b3,rsl
1,1,0,0,1
2,1,,1,1
3,0,,0,
4,0,0,0,0
```

5.76 pi - Pi

Format: pi()

This function returns the value of Pi (π).

Example

Example 1: Basic Example

```
$ more dat1.csv
id
1
2
$ mcal c='pi()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=pi() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,3.141592654
2,3.141592654
```

5.77 power - Raised to a Power

Format: power(*num*,power)

Calculate the power of *num*. This function is equivalent to the ^ operator. The function returns NULL value if the result exceeds the largest real number.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,base,exponent
1,5,2
2,2,8
3,,
4,0,10
5,10,0
6,2,0.5
7,2,-1
$ mcal c='power(${base},${exponent})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=power(${base},${exponent}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,base,exponent,rsl
1,5,2,25
2,2,8,256
3,,,
4,0,10,0
5,10,0,1
6,2,0.5,1.414213562
7,2,-1,0.5
```

5.78 product - Product

Format: product($num_1, num_2, \cdots$)

Calculate the product of the array of values in num_i. NULL values are ignored, if NULL values exist in all items, the result will return a NULL value.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,,
$ mcal c='product(${v1},${v2},${v3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=product(${v1},${v2},${v3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,6
2,-5,2,1,-10
3,1,,3,3
4,,,,
```

Example 2: Example using wildcard

Use the wildcard character " `v*` " to specify columns starting from `v` (`v1,v2,v3`).

```
$ mcal c='product(${v*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=product(${v*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3,rsl
1,1,2,3,6
2,-5,2,1,-10
3,1,,3,3
4,,,,
```

5.79 radian - Radian

Convert degrees to radians.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,60
2,180
3,
4,0
5,-90
$ mcal c='radian(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=radian(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,60,1.047197551
2,180,3.141592654
3,,
4,0,0
5,-90,-1.570796327
```

5.80 rand - Uniformly distributed random numbers

Format: ([Random seed])

Generate uniformly distributed random number from 0.0 to 1.0. This function returns the same result as using mrand command without `-int` option. The same random seed will generate the same sequence of random numbers. The random seed can be specified from -2147483648 to 2147483647. If the random seed is not defined, the random numbers will be seeded according to current time (1/1000 second). This function uses Mersenne Twister as random number generator (Author 's original website , boost library).

Examples

Example 1: Basic Example

Generate uniformly distributed random number from 0.0 to 1.0. Since a random seed is specified, the random number sequence generated is always the same.

```
$ more dat1.csv
id
1
2
3
4
$ mcal c='rand(1)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=rand(1) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,0.4170219984
2,0.9971848081
3,0.7203244893
4,0.9325573612
```

Example 2: Random seed is not set

The random sequence will be different upon each run since random seed is not specified.

```
$ mcal c='rand()' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=rand() i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,rsl
1,0.9931602769
2,0.3491961618
3,0.6793976133
4,0.08248335845
```

5.81 randi - Uniformly Generated Random Integers

Format: (minimum, maximum[, random seed])

Generate random integers between the specified minimum value to maximum value. The mrand command using `-int` option returns the same results. The same random seed will generate the same sequence of random numbers. The random seed can be specified from -2147483648 to 2147483647. If the random seed is not defined, the random numbers will be seeded according to current time (1/1000 second). This function uses Mersenne Twister as random number generator (Author 's original website , boost library).

Examples

Example 1: Basic Example

Generate random integers from 100 to 999 (3 digit integers 900 types). Since a random seed is specified, the random number sequence generated is always the same.

```
$ more dat1.csv
id
1
2
3
4
$ mcal c='randi(100,999,1)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=randi(100,999,1) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,475
2,997
3,748
4,939
```

Example 2: 0,1 random integers

Generate two sets of random integers 0 and 1. The random sequence will be different upon each run since the random seed is not specified.

```
$ mcal c='randi(0,1)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=randi(0,1) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,rsl
1,1
2,0
3,1
4,0
```

5.82 regexlen - Match Length of Character String

Format 1: regexlen(*str*,regular expression)

Format 2: regexlenw(*str*,regular expression)

Returns the length of substring where the defined regular expression matches the string *str*. The function returns 0 if no match is found, in other words, 0 character match.

Use regexlenw function if *str* or regular expression contain multibyte characters.

Examples

Example 1: Basic Example

Find out the length of the longest substring that matches with the regular expression `c.*a`. Since the same input data is used for matching substring in the `regexstr` function, it is easier to compare the results.

```
$ more dat1.csv
id,str
1,xcbbbayy
2,xxcbaay
3,
4,bacabbca
$ mcal c='regexlen($s{str},"c.*a")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexlen($s{str},"c.*a") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,xcbbbayy,5
2,xxcbaay,4
3,,0
4,bacabbca,6
```

Example 2: Multibyte characters

Find out the length of the longest substring that matches `"い.*あ"`. However, since regexlen function do not support multibyte character, it returns the number of bytes instead of number of characters.

```
$ more dat2.csv
id,str
1,漢漢あbbbいyy
2,漢あbいいy
3,
4,bあいあbbいあ
$ mcal c='regexlen($s{str},"あ.*い")' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=regexlen($s{str},"あ.*い") i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,漢漢あbbbいyy,9
2,漢あbいいy,10
3,,0
4,bあいあbbいあ,14
```

Example 3: Multibyte characters 2

Find out the length of the longest substring that matches `"い.*あ"`. The `regexlenw` function is able to process multibyte characters to count the number of characters.

```
$ mcal c='regexlenw($s{str},"あ.*い")' a=rsl i=dat2.csv o=rsl3.csv
#END# kgcal a=rsl c=regexlenw($s{str},"あ.*い") i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,str,rsl
1,漢漢あbbbいyy,5
2,漢あbいいy,4
```

```
3,,0
4,bあいあbbいあ,6
```

5.83 regexm - Complete Match

Format 1: regexm(*str*,regular expression)

Format 2: regexmw(*str*,regular expression)

Return true if the regular expressions matches the whole character string *str*. Use regexmw function if multibyte characters exists in *str* or regular expression, or characters in Shift_JIS encoding does not correspond to search results.

Examples

Example 1: Basic Example

Both records where `id=1,id=2` contains regular expression beginning with `c` and ends with `aa`, in record `id=2`, there is only partial match (matches from `c` at the second position to the end) with the regular expression.

```
$ more dat1.csv
id,str
1,caabaa
2,acabaaa
3,
4,bbcbcc
$ mcal c='regexm($s{str},"c.*aa")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexm($s{str},"c.*aa") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,caabaa,1
2,acabaaa,0
3,,0
4,bbcbcc,0
```

Example 2: String ends with same substring

The full string in column *str* that matches the regular expression `.*c` is only found in the record where `id=4`.

```
$ mcal c='regexm($s{str},".*c")' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=regexm($s{str},".*c") i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,caabaa,0
2,acabaaa,0
3,,0
4,bbcbcc,1
```

Example 3: Match blank characters

Match the blank characters at `id=3` using the regular expression `^$`.

```
$ mcal c='regexm($s{str},"^$")' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=regexm($s{str},"^$") i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,str,rsl
1,caabaa,0
2,acabaaa,0
3,,1
4,bbcbcc,0
```

5.84 regexpfx - Prefix of Match String

Format 1: regexpfx(*str*,regular expression)

Format 2: regexpfx(*str*,regular expression)

Return the prefix (substring before character string) of the longest substring where the defined regular expression matches with the character string *str*. When the same character string and regular expression is used with the three functions to extract different parts of the string in sequential order, namely regexpfx function,regexstr function, and regexsfx function, the original string can be restored by merging the resulting character strings from the functions. Use the regexpfxw function if *str* or regular expression contain multibyte characters, or when characters in Shift_JIS encoding does not correspond to search results.

Example

Example 1: Basic Example

Find out the prefix of the longest substring that matches with the regular expression `c.*a`. For example, in record `id=4`, the function matches `cabbca` in the column and returns the prefix `ba`. Since the same input data and substring are used in the `regexstr,regexpfx` function, it is easy to compare the results.

```
$ more dat1.csv
id,str
1,xcbbbayy
2,xxcbaay
3,
4,bacabbca
$ mcal c='regexpfx($s{str},"c.*a")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexpfx($s{str},"c.*a") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,xcbbbayy,x
2,xxcbaay,xx
3,,
4,bacabbca,ba
```

5.85 regexpos - Match Position

Format 1: regexpos(*str*,regular expression)

Format 2: regexposw(*str*,regular expression)

Return the starting position of the longest substring where the defined regular expression matches with the character string *str*. The first character of the string starts at position 0. The function return NULL if no match is found. Use the regexposw function if *str* or regular expression contain multibyte characters.

Examples

Example 1: Basic Example

Find out the position of the longest substring that matches the regular expression `c.*a`. Note that the first character starts at position 0. Since the same input data and substring are used in the `regexstr` function, it is easy to compare the results and understand the differences.

```
$ more dat1.csv
id,str
1,xcbbbayy
2,xxcbaay
3,
4,bacabbca
$ mcal c='regexpos($s{str},"c.*a")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexpos($s{str},"c.*a") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,xcbbbayy,1
2,xxcbaay,2
3,,
4,bacabbca,2
```

Example 2: Multibyte character

Find out the longest substring that matches the regular expression `"い.*あ"`. However, since regexpos does not support multibyte characters, the function returns the position of bytes instead of characters.

```
$ more dat2.csv
id,str
1,漢漢あbbbいyy
2,漢あbいいy
3,
4,bあいあbbいあ
$ mcal c='regexpos($s{str},"あ.*い")' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=regexpos($s{str},"あ.*い") i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,漢漢あbbbいyy,6
2,漢あbいいy,3
3,,
4,bあいあbbいあ,1
```

Example 3: Multibyte character 2

Find out the longest substring that matches the regular expression `"い.*あ"`. The regexposw function is able to process multibyte characters accurately.

```
$ mcal c='regexposw($s{str},"あ.*い")' a=rsl i=dat2.csv o=rsl3.csv
#END# kgcal a=rsl c=regexposw($s{str},"あ.*い") i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,str,rsl
1,漢漢あbbbいyy,2
2,漢あbいいy,1
```

```
3,,
4,bあいあbbいあ,1
```

5.86 regexrep - Replace Match String

Format 1: regexrep(*str*,regular expression,replacement string)

Format 2: regexrepw(*str*,regular expression,replacement string)

Return the starting position of the longest substring where the defined regular expression matches with the character string *str*. Use the regexrepw function if *str* or regular expression contain multibyte characters, or when characters in Shift_JIS encoding does not correspond to search results.

Example

Example 1: Basic Example

In row where `id=1,id=2`, the matched substrings in the column is replaced with `MMM`.

```
$ more dat1.csv
id,str
1,caabaa
2,acabaaa
3,
4,cbcbcc
$ mcal c='regexrep($s{str},"c.*aa","MMM")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexrep($s{str},"c.*aa","MMM") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,caabaa,MMM
2,acabaaa,aMMM
3,,
4,cbcbcc,cbcbcc
```

5.87 regexs - Substring Match

Format 1: regexs(*str*,regular expression)

Format 2: regexsw(*str*,regular expression)

Return *true* if the specified regular expression matches a part of the string *str*. Use the regexsw function if *str* or regular expression contain multibyte characters, or when characters in Shift_JIS encoding does not correspond to search results.

Example

Example 1: Basic Example

Records that contain the regular expression starting with `c` until `aa` includes `id=1,id=2`. The function returns true for matching records.

```
$ more dat1.csv
id,str
1,caabaa
2,acabaaa
3,
4,cbcbcc
$ mcal c='regexs($s{str},"c.*aa")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexs($s{str},"c.*aa") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,caabaa,1
2,acabaaa,1
3,,0
4,cbcbcc,0
```

Example 2: Match start of string

The regular expression `.*c` which matches the value in the *str* column for all records except `id=3`.

```
$ mcal c='regexs($s{str},".*c")' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=regexs($s{str},".*c") i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,caabaa,1
2,acabaaa,1
3,,0
4,cbcbcc,1
```

5.88 regexsfx - Suffix of Matched String

Format 1: regexsfx(*str*,regular expression)

Format 2: regexsfx(*str*,regular expression)

Return the suffix (substring at the end of character string) of the longest substring where the defined regular expression matches part of the character string *str*. When the same character string and regular expression is used with the three functions to extract different parts of the string in sequential order, namely regexpfx function,regexstr function, and regexsfx function, the original string can be restored by merging the resulting character strings from the functions. Use the regexsfxw function if *str* or regular expression contain multibyte characters, or when characters in Shift_JIS encoding does not correspond to search results.

Example

Example 1: Basic Example

Extract the suffix of the longest substring that matches the regular expression `c.*a`. For example, in row where `id=4`, the regular expression matches `cabbca` till the ending position in the column, there is no suffix to the matching string, thus NULL character is returned. Since the same input data and substring are used as in the `regexstr,regexpfx` function, it is easy to compare the results and understand the differences.

```
$ more dat1.csv
id,str
1,xcbbbayy
2,xxcbaay
3,
4,bacabbca
$ mcal c='regexsfx($s{str},"c.*a")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexsfx($s{str},"c.*a") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,xcbbbayy,yy
2,xxcbaay,y
3,,
4,bacabbca,
```

5.89 regexstr - Match Character String

Format 1: regexstr(*str*,regular expression)

Format 2: regexstrw(*str*,regular expression)

Return the longest substring where the defined regular expression matches part of the character string *str*. Use the regexstrw function if *str* or regular expression contain multibyte characters, or when characters in Shift_JIS encoding does not correspond to search results.

Example

Example 1: Basic Example

Extract the longest substring that matches the regular expression `c.*a`. At row where `id=2`, the regular expression matches the string `cba` and `cbaa`, however, the longer substring is returned.

```
$ more dat1.csv
id,str
1,xcbbbayy
2,xxcbaay
3,
4,bacabbca
$ mcal c='regexstr($s{str},"c.*a")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=regexstr($s{str},"c.*a") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,xcbbbayy,cbbba
2,xxcbaay,cbaa
3,,
4,bacabbca,cabbca
```

5.90 right - Extract Substring from Right

Format 1: right(*str*, length)

Format 2: rightw(*str*, length)

Return the number of characters as defined in the parameter from the right side of the string in *str*. Use rightw function if the string contains multibyte characters.

Example

Example 1: Basic Example

Extract the last 3 characters from the end in the `str` column.

```
$ more dat1.csv
id,str
1,abcdefg
2,12345678
3,
4,12
$ mcal c='right($s{str},3)' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=right($s{str},3) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,abcdefg,efg
2,12345678,678
3,,
4,12,12
```

Example 2: Example of data containing multibyte characters

Use the function `right` if the data contains multibyte characters.

```
$ more dat2.csv
id,str
1,あいうえお
2,1234567あ8
3,1あ
4,ああ
$ mcal c='right($s{str},6)' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=right($s{str},6) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,str,rsl
1,あいうえお,えお
2,1234567あ8,67あ8
3,1あ,1あ
4,ああ,ああ
```

5.91 round - Rounding

Format: round(*num*,base)

Rounding *num* to the specified base. The number will be rounded to the nearest integer close to the multiple of the base. For example, in the expression (3.82,0.5), the decimal digits of 3.82 is above 0.5, thus rounding to the nearest 0.5 returns 4.0. The default value of the base is 1 if the argument is not specified. This is equivalent to rounding an integer value to the first digit after decimal.

Example

Example 1: Basic Example

Round to the nearest digit after decimal.

```
$ more dat1.csv
id,val
1,3.28
2,3.82
3,
4,-0.6
$ mcal c='round(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=round(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.28,3
2,3.82,4
3,,
4,-0.6,-1
```

Example 2: Basic Example

Round to the first decimal.

```
$ mcal c='round(${val},0.1)' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=round(${val},0.1) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,val,rsl
1,3.28,3.3
2,3.82,3.8
3,,
4,-0.6,-0.6
```

Example 3: Example using base of 0.5

Round to the nearest 0.5.

```
$ mcal c='round(${val},0.5)' a=rsl i=dat1.csv o=rsl3.csv
#END# kgcal a=rsl c=round(${val},0.5) i=dat1.csv o=rsl3.csv
$ more rsl3.csv
id,val,rsl
1,3.28,3.5
2,3.82,4
3,,
4,-0.6,-0.5
```

Example 4: Example using base 10

Round to the nearest 10th digit.

```
$ more dat2.csv
id,val
1,1341.28
2,188
3,1.235E+3
```

```
4,-1.235E+3
$ mcal c='round(${val},10)' a=rsl i=dat2.csv o=rsl4.csv
#END# kgcal a=rsl c=round(${val},10) i=dat2.csv o=rsl4.csv
$ more rsl4.csv
id,val,rsl
1,1341.28,1340
2,188,190
3,1.235E+3,1240
4,-1.235E+3,-1230
```

5.92 second - Second

Format 1: second(*time*) numeric value

Format 2: seconds(*time*) 2 digit fixed length string

Extract second from *time*. The result of the function is noted in format 1 and 2.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19770812122212
$ mcal c='second($t{time})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=second($t{time}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,rsl
1,20000101000000,0
2,20121021111213,13
3,,
4,19770812122212,12
```

Example 2: Return character string

```
$ mcal c='seconds($t{time})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=seconds($t{time}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,00
2,20121021111213,13
3,,
4,19770812122212,12
```

5.93 sign - Sign

Format : sign(*num*)

Determine the sign of *num*. Returns 0 if zero, 1 if positive, -1 if negative.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,5
2,-5
3,
4,0
$ mcal c='sign(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=sign(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,5,1
2,-5,-1
3,,
4,0,0
```

5.94 sin - Sine

Format : sin(r)

Calculate sine of radian r.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.141592
2,0.523599
3,
4,6.283185
$ mcal c='sin(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=sin(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.141592,6.535897931e-07
2,0.523599,0.5000001943
3,,
4,6.283185,-3.071795869e-07
```

5.95 sinh - Hyperbolic Sine

Format : sinh(r)

Calculate sine of radian r.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.141592
2,-1.047197
3,
4,6.283185
$ mcal c='sinh(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=sinh(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.141592,11.54873178
2,-1.047197,-1.249366168
3,,
4,6.283185,267.7448118
```

5.96 sqrt - Square Root

Format : sqrt(*num*)

Find out the square root of *num*.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,9
2,2
3,
4,-1
$ mcal c='sqrt(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=sqrt(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,9,3
2,2,1.414213562
3,,
4,-1,
```

5.97 sqsum - Sum of Squares

Format : sqsum($num_1, num_2, \cdots$)

Calculate the sum of squares of the array of numbers in num_i. NULL values are ignored, if NULL values exist in all numbers in the array, the result will return a NULL value.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,,
$ mcal c='sqsum(${v1},${v2},${v3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=sqsum(${v1},${v2},${v3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,14
2,-5,2,1,30
3,1,,3,10
4,,,,
```

Example 2: Example using wildcard

Specify fields starting with `v` (`v1,v2,v3`) by using wildcard ”`v*`”.

```
$ mcal c='sqsum(${v*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=sqsum(${v*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3,rsl
1,1,2,3,14
2,-5,2,1,30
3,1,,3,10
4,,,,
```

5.98 sum - Sum

Format : sum($num_1, num_2, \cdots$)

Calculate the sum of the array of numbers in num_i. NULL values are ignored, if NULL values exist in all numbers in the array, the result will return a NULL value.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,v1,v2,v3
1,1,2,3
2,-5,2,1
3,1,,3
4,,,
$ mcal c='sum(${v1},${v2},${v3})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=sum(${v1},${v2},${v3}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,v1,v2,v3,rsl
1,1,2,3,6
2,-5,2,1,-2
3,1,,3,4
4,,,,
```

Example 2: Example using wildcard

Specify fields starting with `v` (`v1,v2,v3`) by using wildcard "`v*`".

```
$ mcal c='sum(${v*})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=sum(${v*}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3,rsl
1,1,2,3,6
2,-5,2,1,-2
3,1,,3,4
4,,,,
```

5.99 t2julian(date and time)

Use this function to obtain a Julian day from the date and time.

```
A Julian day is the number of days from noon of January 1, 4713 B.C. (universal time).
The effective range of the Gregorian calendar, however, is 1400-1-1 to 9999-12-31,
so the corresponding range of the Julius days is 2232300 to 5373484. NULL is output if the result is outsi
During the conversion, the time is discarded from the specified date and time. Only the date is converted.
```

Usage

```
c=t2julian(0t20080822101010)
```

Example

Table 5.24: Input data

Date	Time	Number
20020824	20020824145408	10660
20020622	20020622173449	22740
20020824	20020824145408	14800
20021009	20021009095743	54510
20020121	20020121173449	18750

Examples based on the above data are shown below.

Execution example 1)

The value in the “ Time ” field is input and converted into a Julian day. A “ Julian day ” field is newly created, and the result is output.

```
------------------------------------------------
mcal c='t2julian($t{Time})' a="Julian day" i=date.csv o=ot2julian.csv
------------------------------------------------
```

Table 5.25: Output file(ot2julian.csv)

Date	Time	Number	Julian day
20020824	20020824145408	10660	2452511.621
20020622	20020622173449	22740	2452448.733
20020824	20020824145408	14800	2452511.621
20021009	20021009095743	54510	2452557.415
20020121	20020121173449	18750	2452296.733

Return to mcal - date related

5.100 tan - Tangent

Format : tan(r)

Compute tangent of radian r.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,0.785398
2,1.047197
3,
4,3.141593
$ mcal c='tan(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=tan(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,0.785398,0.9999996732
2,1.047197,1.732048603
3,,
4,3.141593,3.464102066e-07
```

5.101 tanh - Hyperbolic Tangent

Format : tanh(r)

Calculate the hyperbolic tangent.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,val
1,3.141592
2,-1.047197
3,
4,6.283185
$ mcal c='tanh(${val})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=tanh(${val}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,val,rsl
1,3.141592,0.9962720714
2,-1.047197,-0.7807142201
3,,
4,6.283185,0.9999930253
```

5.102 time - Hour Minute Second

Format : time(*time*)

Extract the 6-digit fixed length string of hour minute and second from *time*.

Example

Example 1: Basic Example

```
$ more dat1.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19770812122212
$ mcal c='time($t{time})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=time($t{time}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,rsl
1,20000101000000,000000
2,20121021111213,111213
3,,
4,19770812122212,122212
```

5.103 today - Today ’s date

Format : today()

Return today ’s date in date format.

Example

Example 1: Basic Example

```
$ more dat1.csv
id
1
2
$ mcal c='today()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=today() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,20171013
2,20171013
```

5.104 tolower - Lowercase Conversion

Convert character string to lowercase characters. This does not affect non-alpha characters outside the 26 alphabets.

Example

Example 1: Basic Example

Convert the values in column *str* to lowercase characters.

```
$ more dat1.csv
id,str
1,ABC
2,aB$12!Cd
3,
4,cBA
$ mcal c='tolower($s{str})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=tolower($s{str}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,ABC,abc
2,aB$12!Cd,ab$12!cd
3,,
4,cBA,cba
```

5.105 top - Top Rows

Format: top()

Return true if first row, otherwise false.

Example

Example 1: Basic Example

```
$ more dat1.csv
val
1
2
3
4
$ mcal c='top()' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=top() i=dat1.csv o=rsl1.csv
$ more rsl1.csv
val,rsl
1,1
2,0
3,0
4,0
```

Example 2: Compute the cumulative value

```
$ mcal c='if(top(),${val},${val}+#{})' a=rsl i=dat1.csv o=rsl2.csv
#END# kgcal a=rsl c=if(top(),${val},${val}+#{}) i=dat1.csv o=rsl2.csv
$ more rsl2.csv
val,rsl
1,1
2,3
3,6
4,10
```

5.106 toupper - Uppercase Conversion

Convert character string to uppercase characters. This does not affect non-alpha characters outside the 26 alphabets.

Usage Examples

Example 1: Basic Example

Convert the values in *str* column from lowercase to uppercase.

```
$ more dat1.csv
id,str
1,abc
2,Ab$12!cD
3,
4,Cba
$ mcal c='toupper($s{str})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=toupper($s{str}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,str,rsl
1,abc,ABC
2,Ab$12!cD,AB$12!CD
3,,
4,Cba,CBA
```

5.107 tseconds - Elapsed Time in Seconds

Format: tseconds(*time*)

Compute the number of seconds elapsed from 00:00:00 to *time*

Usage Examples

Example 1: Basic Example

```
$ more dat1.csv
id,time
1,000103
2,235959
3,
4,000000
$ mcal c='tseconds($t{time})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=tseconds($t{time}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,time,rsl
1,000103,63
2,235959,86399
3,,
4,000000,0
```

Example 2: The result is the same using on date values

```
$ more dat2.csv
id,date
1,20130901000103
2,20130902000103
$ mcal c='tseconds($t{date})' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=tseconds($t{date}) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,date,rsl
1,20130901000103,63
2,20130902000103,63
```

5.108 uxt - UNIX Time Conversion

Format 1: uxt(*date*)

Format 2: uxt(*time*)

Format 3: uxt2d(*num*)

Format 4: uxt2t(*num*)

In format 1 and 2, convert *date* or *time* to UNIX time. In 3 and 4 format, by contraries, convert UNIX time to date type or time type. Giving date type, calculate `00:00:00` as the first time of the day.

Usage Examples

Example 1: Basic Example

Convert the `date` formatted strings in the date column to UNIX time using `d2uxt` function, and convert back to original date string using `uxt2d` function.

```
$ more dat1.csv
id,date
1,20000101
2,20121021
3,
4,19700101
$ mcal c='uxt($d{date})' a=uxt i=dat1.csv o=rsl1.csv
#END# kgcal a=uxt c=uxt($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,uxt
1,20000101,946684800
2,20121021,1350777600
3,,
4,19700101,0
$ mcal c='uxt2d(${uxt})' a=date2 i=rsl1.csv o=rsl2.csv
#END# kgcal a=date2 c=uxt2d(${uxt}) i=rsl1.csv o=rsl2.csv
$ more rsl2.csv
id,date,uxt,date2
1,20000101,946684800,20000101
2,20121021,1350777600,20121021
3,,,
4,19700101,0,19700101
```

Example 2: Example of using time formatted data

```
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19700101000100
$ mcal c='uxt($t{time})' a=uxt i=dat2.csv o=rsl3.csv
#END# kgcal a=uxt c=uxt($t{time}) i=dat2.csv o=rsl3.csv
$ more rsl3.csv
id,time,uxt
1,20000101000000,946684800
2,20121021111213,1350817933
3,,
4,19700101000100,60
$ mcal c='uxt2t(${uxt})' a=time2 i=rsl3.csv o=rsl4.csv
#END# kgcal a=time2 c=uxt2t(${uxt}) i=rsl3.csv o=rsl4.csv
$ more rsl4.csv
id,time,uxt,time2
1,20000101000000,946684800,20000101000000
2,20121021111213,1350817933,20121021111213
3,,,
4,19700101000100,60,19700101000100
```

5.109 Week

Format 1: week(*date*)

Format 2: week(*time*)

Format 3: week111(*date*)

Format 4: week111(*time*)

Return the week number following the ISO8601 standard from *date* or *time*. ISO8601 prescribed week number starts at the first week containing Thursday in the new ISO year. The function week111 returns the week number that starts at the first day of week 01 on 1/1 regardless of the day of week.

Usage Example

Example 1: Basic Example

```
$ more dat1.csv
id,date
1,20000101
1,20000102
1,20000103
1,20000104
1,20000105
1,20000106
1,20000107
1,20000108
1,20000109
2,20121021
3,
4,19770812
$ mcal c='week($d{date})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=week($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,20000101,52
1,20000102,52
1,20000103,1
1,20000104,1
1,20000105,1
1,20000106,1
1,20000107,1
1,20000108,1
1,20000109,1
2,20121021,42
3,,
4,19770812,32
```

Example 2: Example of using time formatted data

```
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19770812122212
$ mcal c='week($t{time})' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=week($t{time}) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,52
2,20121021111213,42
3,,
4,19770812122212,32
```

5.110 xor Exclusive logical OR

Format: '*bFieldnamedbFieldname*'

Use this function to calculate the exclusive logical OR of all the true/false values given by $bool_i$. For the true/false value table including the NULL value, see Table 5.10.

Example

Example 1: Basic example

```
$ more dat1.csv
id,b1,b2,b3
1,1,0,0
2,1,,1
3,0,,0
4,0,0,0
$ mcal c='$b{b1}^^$b{b2}^^$b{b3}' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=$b{b1}^^$b{b2}^^$b{b3} i=dat1.csv o=rsl1.csv; IN=4 OUT=4
$ more rsl1.csv
id,b1,b2,b3,rsl
1,1,0,0,1
2,1,,1,
3,0,,0,
4,0,0,0,0
```

5.111 year - Liturgical Year

Format 1: year(*date*)

Format 2: year(*time*)

Format 3: years(*date*)

Format 4: years(*time*)

Extract *date* and *time* from year. Format 1 and 2 return numerical value, format 3 and 4 return character string.

Examples

Example 1: Basic Example

```
$ more dat1.csv
id,date
1,20000101
2,20121021
3,
4,19770812
$ mcal c='year($d{date})' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=year($d{date}) i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,date,rsl
1,20000101,2000
2,20121021,2012
3,,
4,19770812,1977
```

Example 2: Time formatted values

```
$ more dat2.csv
id,time
1,20000101000000
2,20121021111213
3,
4,19770812122212
$ mcal c='year($t{time})' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=year($t{time}) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,time,rsl
1,20000101000000,2000
2,20121021111213,2012
3,,
4,19770812122212,1977
```

5.112 Cast

Format: s2n(*str*), n2s(*num*), n2b(*num*), s2n(*str*), s2d(*str*), s2t(*str*) s2b(*str*), d2s(*date*), d2t(*date*), t2s(*time*), t2d(*time*), b2n(*bool*), b2s(*bool*)

Set of functions to convert data type. `mcal` do not automatically converts data type, user must specify the required data type conversion.

`n2b(s2b)` function converts true to 1("1") and false to 0("0"), other values are treated as NULL values. `b2n(b2s)` function converts true to 1("1") and false to 0("0").

`d2t` function converts date and time type data, and automatically completes the time as 12:00:00. `d2s` function converts data to 8 digit fixed length character string ("yyyymmdd"), `t2s` function converts data to 14 digit fixed length character string ("yyyymmddHHMMSS").

Refer "5.13 Date and Time Type" for more information on date and time.

Examples

Example 1: Fixed length random number

Generate random numbers from 1 to 9999 as 4 digit fixed length string. Integer data (results of randi) is not supported by `fixlen` function, thus the data must be converted to character string with n2s function.

```
$ more dat1.csv
id
1
2
3
4
$ mcal c='fixlen(n2s(randi(1,9999,11)),4,"R","0")' a=rsl i=dat1.csv o=rsl1.csv
#END# kgcal a=rsl c=fixlen(n2s(randi(1,9999,11)),4,"R","0") i=dat1.csv o=rsl1.csv
$ more rsl1.csv
id,rsl
1,1803
2,0684
3,0195
4,6647
```

Example 2: True false pattern

Detect unusual pattern in columns v1,v2,v3 and print as 01 in output.

```
$ more dat2.csv
id,v1,v2,v3
1,10,5,7
2,5,12,11
3,3,6,2
4,14,16,11
$ mcal c='cat("",b2s(${v1}>=10),b2s(${v2}>=10),b2s(${v3}>=10))' a=rsl i=dat2.csv o=rsl2.csv
#END# kgcal a=rsl c=cat("",b2s(${v1}>=10),b2s(${v2}>=10),b2s(${v3}>=10)) i=dat2.csv o=rsl2.csv
$ more rsl2.csv
id,v1,v2,v3,rsl
1,10,5,7,100
2,5,12,11,011
3,3,6,2,000
4,14,16,11,111
```

Index