CSVfix 1.7 Manual

Table of contents

Introduction

Welcome to **CSVfix 1.7**.

If you have any dealings at all with data and databases, then you almost certainly will have to deal with comma-separated values (CSV) data. Unfortunately, the CSV files you are given, or are required to produce, never seem to be in quite the right format for your particular business application. And because of the structure of CSV records, using standard text processing tools like sed, awk and perl is not as simple as it might be.

CSVfix aims to provide a solution to these problems. It is a command-line stream editor specifically designed to deal with CSV data. With it you can, among many other things:

- Convert fixed format, multi-line, and DSV files to CSV.
- Reorder, remove, split and merge fields.
- Sort CSV data on different CSV fields.
- Convert case, trim leading & trailing spaces.
- Search for specific content using regular expressions.
- Filter out duplicate data or data on exclusion lists.
- Perform sed/perl style editing.
- Enrich with data from other sources.
- Add sequence numbers, fixed text and file source information.
- Split large CSV files into smaller files based on field contents.
- Perform arithmetic calculations on individual fields.
- Validate CSV data against a collection of validation rules.
- Convert between CSV and XML, fixed format, SQL, DSV and plain text.
- Check for differences between CSV files.

Please also see the Problems & Solutions section for some common problems and their CSVfix solution.. See the Change Log for information on what is new in this release, and the Commands section for a list of all CSVfix commands.

CSVfix is Free Open Source Software, licensed under the MIT License. If you would like to encourage its development, please consider making a donation.

Usage

CSVfix is a command-line program that must be run via a command-line prompt. To use CSVfix you will need to open a Windows command prompt or a Linux/UNIX shell window.

The general form of the CSVfix command line is as follows:

```
csvfix command flags files
```

where:

- `csvfix` is the name of the CSVfix executable.
- `command` is one of the CSVfix commands listed in this manual

- `flags` is zero or more flags, each preceded by a hyphen, which are specific to the command. A few flags are not command-specific, and are described here.
- `files` is a list of zero or more files which provide input for the command

Here are some examples of CSVfix usage. There are many more examples in the alphabetic commands section of this manual.

```
csvfix help

csvfix help order

csvfix order -f 1,3,7 file.csv file2.csv

csvfix fileinfo -t -bc afile.dat

csvfix upper mydata.csv | csvfix order -f 3,1,2
```

Running CSVfix with no command-line parameters displays version, configuration and copyright information.

Commands can be abbreviated to their minimal non-ambiguous form, so the last command above could also have been written as:

```
csvfix up mydata.csv | csvfix or -f 3,1,2
```

Some command parameters may need to be quoted (for example, those containing spaces). If you are using the Windows CMD shell, you should normally use double-quotes for this, if you are using the Bash or similar shells, you should use single-quotes. The examples in this manual use single quotes because I use the Bash shell for all my work.

If no files are specified CSVfix reads its standard input. You can also force CSVfix to read standard input by using a hyphen as a file name. For example:

```
csvfix echo file1.dat - file2.dat
```

reads file1.dat, then standard input, then file2.dat.

CSVfix normally writes its output to standard output, unless the -o flag is used . This means that CSVfix can be used in pipelines and indeed this is one of the important means of using it; if it seems that one CSVfix command invocation cannot do the job, two (or possibly more), connected via pipes, almost certainly can.

Solutions to Common Problems

Here are some of the common problems that you may experience when dealing with CSV data, and brief suggestions as to how CSVfix can help you with them:

"The fields in the data are in the wrong order"

Reorder them using the order command.

"The records in the data are in the wrong order"

Sort them using the sort command. Remove duplicates with the unique command.

"Some fields are missing from my data"

If you simply need to add empty fields to keep your application happy, use the pad command. If you need to enrich your data with values from another file, use the join command. If you need to add a fixed string to the data, use the put command.

"My CSV data isn't comma-separated!"

Use the -sep and -rsep flags to tell CSVfix what separator to use.

"I need to read tab-separated data"

Use the read_dsv command with the -s '\t' flag to convert tab-separated data to CSV.

"I need to convert XML to CSV"

The from_xml command probably does what you need.

"I need to perform calculations on some of the fields in my CSV data"

The eval command provides arithmetic and other functions you can apply to your CSV data.

"I need to extract data from the middle of a field"

Use the split_fixed and split_char commands to break fields up into sub fields. If you need to build a field from values in other fields, use the merge command.

"I need to remove some lines from a CSV file before processing it further"

The find and remove commands allow you to filter your CSV files using regular expressions, value ranges and field lengths. You can also create exclusion lists and use the join command with the -inv flag to exclude selected rows.

"I need to merge the data from two CSV files"

Use the order command to get the fields in the same order, and then the unique command to merge them, discarding duplicates.

"I want to add record numbers to my CSV data"

The sequence command provides flexible record numbering.

"I want to split my data into different files"

The file_split command can split CSV data streams depending on field values.

"My data is full of duplicate values"

Remove duplicates using the unique command.

"The data I get given is always full of invalid values"

Validate your data using the validate command. Remove bad values with the remove command.

"I need to convert CSV data into XML"

The to_xml command does exactly this.

"I need to convert CSV data to this weird format"

Use write_fixed, write_dsv and the printf and template commands to format your output.

"I want to extract data from a SQL database as CSV"

In the Windows version of CSVfix, the odbc_get command allows you to extract data from SQL databases.

"I need to import data into a database, but the data is in a weird format"

Use CSVfix to convert the data to CSV using the read_fixed, read_dsv, and/ot read_multi commands, and then convert the CSV to SQL INSERT statements using sql_insert.

"I need special date formatting"

Use the date_format and date_iso comnands to read and re-format dates.

Config Files

From version 1.5, CSVfix provides two features implemented via configuration files. These are default options which will be applied to all commands, and the ability to create command aliases. These features are specified by configuration files. The files are called **csvfix.cfg** if running on Windows, and **.csvfix** if running on one of the UNIX-like operating systems. CSVfix looks for a configuration file first in the current working directory, and then in the user's home directory, which the directory specified by the **USERPROFILE** environment variable on Windows, and **HOME** on the UNIX-like systems. The path to the actual config file used can be seen by running CSVfix with no command-line parameters.

The configuration file is a text file which consists of a number of configuration commands and/or comments. Comments are any lines where the first non-blank character is a '#'. Commands are lines beginning with either **defaults** or **alias**.

To specify default options, use the defaults command. For example, this command:

```
defaults -smq -sep ';'
```

says that you always want smart quoting to be on, and that you always want the CSV separator to be a semicolon. You can specify only one defaults command per configuration file.

To specify aliases, use the alias command. This example sets lsempty to be an alias for a find command that lists any records that have empty fields in them:

```
alias lsempty find -e '^$'
```

Once you have this alias defined, you can use lsempty like other csvfix commands:

```
csvfix lsempty mydata.csv
```

You can specify multiple aliases in a configuration file.

Skip and Pass

A common requirement when using CSVfix is to apply a command to only some records in the CSV input data. To a certain extent you can manage this using commands like find to filter the input, and operating system pipes to connect the filtered input to the relevant command. However, this isn't always what is wanted, and so from version 1.5 of CSVfix, the ability to filter input for most commands has been added using the **-skip** and **-pass** options.

The -skip option is probably the easier of the two options to understand. Suppose you want to re-order the names.csv data, so that surnames come first, but only for female authors. You can use the -skip option to do this, together with the order command (please see here for information on the quoting required by the expression language):

```
csvfix order -skip "$3 != 'F'" -f 2,1 names.csv
```

The parameter "$3 != 'F'" of the -skip option is an expression in the CSVfix expression language, which will be evaluated for every CSV input record. If the expression evaluates to true, then the input record is discarded, is not fed into the actual command (order, in this case), and so produces no output. The result of this command is:

```
"Austen","Jane"
"Elliot","George"
"Woolf","Virginia"
```

The -pass option works somewhat similarly. Suppose that instead of discarding the male authors, you want them in the output, but you do not want them re-ordered:

```
csvfix order -pass "$3 != 'F'" -f 2,1 names.csv
```

In this case, if the expression specified by -pass evaluates to true, the CSV input record is passed to the output unchanged, so the result of this command is:

```
"Charles","Dickens","M"
"Austen","Jane"
"Herman","Melville","M"
"Flann","O'Brien","M"
"Elliot","George"
"Woolf","Virginia"
"Oscar","Wilde","M"
```

The -skip and -pass options are currently implemented for the following commands (asterisk indicates -skip only):

ascii_table *	call	date_format
echo *	edit	escape
exclude	exec	file_info
flatten *	merge	money
pad	printf	rmnew
split_char	split_fixed	sql_delete *
sql_update	template *	timestamp
trim	truncate	unflatten *
write_dsv *	write_fixed *	head *

Quoting

Some parameters of CSVfix commands will require quoting. These are typically those that specify strings containing spaces, or those that use the expression language. For example, this find command requires quotes around the name being searched for in the books.csv file:

```
csvfix find -f 3 -s 'Great Expectations' books.csv
```

The type of quotes you need to use will vary with the shell you are using, but in general you should use double-quotes if you are using the Windows cmd.exe shell, and single quotes if you are using a shell such as bash. This also affects the quoting for the expression language. Using cmd.exe:

```
csvfix eval -e "if( $1 == 'Emma', 'Based in Surrey', 'Somewhere else')"
books.csv
```

while when using bash:

```
csvfix eval -e 'if( $1 == "Emma", "Based in Surrey", "Somewhere else")'
books.csv
```

Failure to use the correct quotes will result in strange error messages, such as:

```
ERROR: Cannot open Expectations' for input
```


Support

The CSVfix website is at http://neilb.bitbucket.org/csvfix, where you will find downloads of the latest version of the software and manuals. Please note that the old Google Code site is no longer updated, due to Google forbidding posting dowloadable executables.

Please report any bugs, comments or suggestions for enhancements via the CSVfix support forum at http://groups.google.com/group/csvfix. You will probably get a more immediate response by posting to the support forum than you will from submitting issues via the bitbucket issue tracker.

The CSVfix source code is available from the Google Code website - link above. To access it you must use the Mercurial version control system. You can get a zip of a selected revision from the clone of the source at https://bitbucket.org/neilb/csvfix/changesets - use the **get source** menu to download the source without the need for Mercurial.

Change Log

Changes from Version 1.6 to 1.7

- Added split_regex command to split a field using regular expressions.
- Added pivot command to perform simple table pivoting.
- Added rowsort command to perform sorting fields within CSV records.
- Added erase command to remove fields from records using regex.
- Added squash command to reduce rows with same key fields to single row.
- Added -ix option to exec command to ignore exit codes from program being executed.
- The stat command now supports -fs and -fn options for more detailed reports.
- Added errorif() function to expression language.

- The printf command now understands backspace escaped characters like \t and \n.
- Added -csv option to printf command to force output to be CSV.
- The -skip and -pass options now can use $line and $file variables.
- The split_fixed command can now split on variable field length.
- Added -a and -s options to trim command to remove embedded spaces.

Changes from Version 1.5 to 1.6

- Added number command to convert formatted numbers to arithmetic values.
- Added write_multi command to write multi-line master/detail records.
- Added -hdr option to allow CSV header record to be specified.
- Added isint, find and round functions to expression language.
- Added -q option to printf command to support quoting of internal double-quotes.
- Added -ec option to validate command to allow for error return value on validation failure.
- Added -me option to flatten command to support master/detail flattening.
- Added -if option to eval command to mitigate problems with the expression language if() function.
- Added -en option to the SQL generation commands to convert empty fields to NULLs.
- Added -k option to join command to retain all join fields on output.

Changes from Version 1.4 to 1.5

- Added -skip and -pass options to filter command input and output
- Added ability to specify default options and aliases via configuration files.
- Actual config file in use is displayed if CSVfix is run with no command line parameters.
- Added head and tail commands to display first/last CSV records in file.
- Added -ic option to join command to ignore case of joined fields.
- Added -f option to printf command to specify order.
- Added -s option to asci_table command to add separator after each record.
- The template command now allows expressions in templates.
- Added -fn option to template command to allow for templated output file names.
- Add -f option to shuffle command to allow shuffling of fields.
- Added several new functions for use by expression language.
- The random number generator used by the expression language can be seeded with the -seed option.
- Can now use != as well as <> for not-equal operator in expression language.
- Output of help now goes to standard output, not standard error.
- Removed the -rin option as the idea behind it was basically wrong.

Changes from Version 1.3 to 1.4

- Added timestamp command.
- Added call command to call function in external DLL.
- Added money command to do currency formatting.
- Added -rf option to exclude command to allow specifying of fields from end of record.
- Added -d option to eval command to discard input data
- Added -h option to odbc_get command to get column names as CSV header record
- Fixed -s option bug for find and remove commands
- The validate command now supports multiple error reports for the same record.
- Added -rin flag to make use of expressions mixing numbers and strings a bit easier.
- Added -rh flag to sort command to retain header record on output

- Various updates to the manual.

Changes from Version 1.2 to 1.3

- Added block command to mark blocks of records.
- Added stat command to produce CSV file record/field counts.
- Improved error reporting for commands using the expression language.
- Many bug fixes.

Changes from Version 1.1 to 1.2

- Command abbreviations are now supported.
- Added check command to do CSV syntactic validation.
- The odbc_get command now has a -dir option to simplify using the ODBC text driver.
- The map command now supports dynamically mapping to other fields in the input record.
- Added match() function to match regular expressions to eval command.
- Added -r flag to eval command to allow field replacement as well as appending.
- The find and remove commands now support field testing via eval-style expressions.
- The exclude command now supports excluding fields based on an eval-style expression.
- The -si flag in the find and remove commands now works correctly.

Changes from Version 1.0 to 1.1

- The find and remove commands now support non-regular expression strings.
- Added diff command for comparing CSV files.
- Added file_merge command for merging sorted files.
- The split_char command now supports splitting on character type transitions.
- Added rmnew command to remove newlines inside CSV fields.
- Help now implemented using HelpNDoc - looks nicer & spelling mistakes all (?) fixed.

Changes from Version 0.97 to 1.0

- Added the from_xml command to convert XML data to CSV. This is a much improved facility over the original read_xml command, which was removed a while back.
- Added ability to search for ranges and lengths with the find and remove commands.
- Commands like order now support field indexes specified as ranges.
- Fixed problem with trailing empty fields being ignored.
- Added -nc flag to order command to allow skipping of non-existent input fields
- Added new functions to eval command.
- Added -d flag to allow decremented numbers for sequence command.
- Added -fc flag to find and remove commands to allow search on CSV field counts.
- The put command can now output field counts.

Older change log entries removed for clarity.

Licence

Commands

This section is an alphabetic list of the currently available CSVfix commands. Each command has one or more examples of how it should be used - see the Usage section for general help on command syntax.

You can also get a list of commands available in your CSVfix executable by entering the following on the command line:

```
csvfix help
```

More detailed built-in help is also available. You can get a help summary by entering:

```
csvfix help command
```

where command is the actual name of the command you are interested in. This facility is intended as a reminder of command syntax only - the definitive documentation of CSVfix remains this help file.

Note that all CSVfix commands and command line flags are case-sensitive, and should be entered in lower-case. CSVfix supports several command line flags not described in the detailed command pages, as they are available for almost all commands:

Flag	Req'd?	Description
`-o filename`	No	Write output to named file rather than standard output.
`-ibl`	No	If this flag is used, CSVfix ignores any blank lines in input streams. By defaul not ignored and will result in CSV records consisting of a single empty field be
`-ifn`	No	This flag tells CSVfix to remove a field-name record, which must be the first li file, from output. CSVfix does not check that such a record exists, it merely fi line. This flag has no effect for the commands that do not read CSV input.
`-sep separator`	No	Specify an alternative CSV record separator. This must be a single character whitespace, alphanumeric or the double-quote. The separator will be used for from files or standard input. It will not be used for formatting CSV for output - f rsep flag.
`-rsep separator`	No	As for -sep (with which it is mutually exclusive), but the same separator is als writing CSV output. Note that this may make it possible to produce invalid CS
`-osep separator`	No	Specifies separator to use on CSV output - this setting overrides the -sep and

`-smq`	No	By default, CSVfix wraps every CSV output field in double quotes. If you don't behaviour, the -smq flag turns on smart quoting, which only double-quotes fiel special characters such as commas and quotes. This flag has no effect for th that do not produce CSV output.
`-sqf fields`	No	Specify a list indexes of fields that must be quoted using CSV quoting rules. `-sqf 2,4` specifies that fields 2 and 4 must be quoted. Fields not in the list are not quoted, which means that you can prevent any fie by providing a single, large field number, greater than the number of existing fi using this flag makes it possible to produce invalid CSV output (you can spec that contain embedded quotes must not be quoted) and the flag is mutually ex -smq flag.
`-seed n`	No	Seeds the random number generator used by the expression language randor the integer value n. If this option is not used, the random number generator is current time.
`-skip test`	No	Perform a test using the expression language. If the test is true for the current record, the record is discarded. See here for more information.
`-pass test`	No	Perform a test using the expression language. If the test is true for the current record, the record is passed-through the command with no processing being p
`-hdr "header text"`	No	Inserts the specified text as the first record in the CSV output.

ascii_table

The **ascii_table** command is used to format CSV into "ascii art" tables. These are useful for presentation and documentation. If you want to create XHTML tables, see the to_xml command.

See also: to_xml

Flag	Req'd?	Description
`-h header`	No	A comma-separated list of strings which will form the headers of the table. If the specia "@" is used, the headers are taken from the first row of the CSV input. If no headers ar specified, the table is produced without headers.
`-ra fields`	No	A list of fields which will be right-aligned rather than the default left alignment. This is u numeric fields.
`-s`	No	Insert separator after each record.

The following example outputs the names.csv file, with suitable headers:

```
csvfix ascii_table -h "Forename,Surname,Sex" data/names.csv
```

which produces:

```
+----------+----------+-----+
| Forename | Surname  | Sex |
+----------+----------+-----+
| Charles  | Dickens  | M   |
| Jane     | Austen   | F   |
| Herman   | Melville | M   |
| Flann    | O'Brien  | M   |
```

```
| George    | Elliot    | F    |
| Virginia | Woolf     | F    |
| Oscar     | Wilde     | M    |
+-----------+-----------+------+
```

You can insert a separator after each record using the -s option:

```
csvfix ascii_table -s -h "Forename,Surname,Sex" data/names.csv
```

produces:

```
+-----------+-----------+------+
| Forename | Surname   | Sex |
+-----------+-----------+------+
| Charles   | Dickens   | M    |
+-----------+-----------+------+
| Jane      | Austen    | F    |
+-----------+-----------+------+
| Herman    | Melville | M    |
+-----------+-----------+------+
| Flann     | O'Brien   | M    |
+-----------+-----------+------+
| George    | Elliot    | F    |
+-----------+-----------+------+
| Virginia | Woolf     | F    |
+-----------+-----------+------+
| Oscar     | Wilde     | M    |
+-----------+-----------+------+
```


block

The **block** command allows you to mark blocks of CSV records depending on values in the first and last records of the blocks, and to remove them, or to remove records not in a block.

See also: eval, find, remove

Flag	Req'd?	Description
-be expr	Yes	Specifies how a block begins using the eval expression language. If the expression eva to true, the record is considered to be the start of a block.
-ee expr	Yes	Specifies how a block ends - if the expression evaluates to true, then the record marks end of the block. If no end record is encountered before end-of-file, the end-of-file is tre the end record.
-r	No	If this flag is used, then blocks are removed from the output stream.
-k	No	If this flag is used, then blocks are retained in the output stream, and records not in a are removed.
-m marks	No	Mark blocks. The marks parameter is a two-entry comma-separated list. Blocks are m with the first entry in the list, and non-blocks with the second. The marker is inserted first field in the output stream.
-x	No	Specifies that the begin/end records are not part of a block.

This example marks all the countries in the countries.csv file that belong to the Euro with "EUR"

```
csvfix block -be '$1 == "FR"' -ee '$1 == "IT"' -m 'EUR,---' data/countries.csv
```

producing:

```
"---","GB","United Kingdom"
"EUR","FR","France"
"EUR","DE","Germany"
"EUR","NL","Netherlands"
"EUR","IT","Italy"
"---","US","United States"
```


call

The **call** command allows you to call a function in a DLL. It is somewhat similar to the exec command, but as it does not require spawning extra processes, it is potentially much faster. **This command is currently Windows-only and is experimental** - the interface may well change in the future. An example of a DLL implementing a function that the command can call can be found in the project source code in the **call-dll** directory.

Flag	Req'd?	Description
`-fnc name`	Yes	Name of function to call.
`-dll name`	Yes	Path/name of DLL containing the function to be called.
`-f fields`	No	Indexes of fields to be passed to function - default is all fields.
`-bs size`	No	Size of buffer to use to return values from the DLL function. The size is specified in kilo so `-bs 8` would be an 8Kb buffer. The default size is 4Kb.

check

The **check** command checks that inputs conform to the CSV standard specified by IETF RFC 4180.

See also: validate

Flag	Req'd?	Description
`-nl`	No	Specifies that newlines may be embedded in quoted fields per RFC 4180. This is off by because most CSV files don't contain embedded newlines, and turning it off improves t quality of the diagnostics.
`-q`	No	Don't produce any output, simply return 0 as the application's exit code if the inputs we validated, 1 if there was a failure.
`-s sep`	No	Use sep as the field separator. Using this means that you are no longer testing agains 4180.
-v	No	Verbose output. Print "OK" for each input that passed validation

date_format

The **date_format** command is used to format dates in various ways. The command requires that dates in the CSV input are in ISO YYYY-MM-DD format. You can convert dates to this format using the date_iso command.

See also: date_iso

Flag	Req'd?	Description
-f fields	Yes	A comma-separated list of fields to attempt to format. If a specified field does not conta valid date in ISO YYYY-MM-DD format, it is not converted.
-fmt format	Yes	A format specification. The formatters currently available are as follows; d - day as 1 or 2 digits dd - day with leading zero m - month as 1 or digits mm - month with leading zero mmm - month as 3-char code e.g. Jan, Aug M - full month name w - weekday as 3-char code e.g. Mon, Tue W - full weekday name y - 4-digit year yyyy - 4 digit year Note 2-digit years are not currently supported. All other characters are treated literally.

The following example takes the file birthdays.csv, converts it to ISO format and then pipes the resulting output into another CSVfix command which outputs the birthday field in the form "Mon 3 Jun 2009" (please note the pipe symbol in the command line):

```
csvfix date_iso -f 2 -m 'd/m/y' data/birthdays.csv |
csvfix date_format -f 2 -fmt 'w d mmm yyyy'
```

which produces:

```
"Peter","Sun 20 Jan 2000
"Jane","Tue 12 Jan 1970"
"Bill","Sat 14 Jan 1971"
"Anna","Sat 27 Jan 1976"
```


date_iso

The **date_iso** command converts dates in user formats to the ISO YYYY-MM-DD format. You can specify the format of the input date using a mask. Once dates are in ISO format, you can reformat them with the date_format command. If a field cannot be converted to ISO format, it is left unchanged.

See also: date_format

Flag	Req'd?	Description
-f fields	Yes	A list of fields to attempt to convert.
-m mask	Yes	The mask that specifies how the parts of the field are to be treated. The mask must be five characters long. Three of these characters must be one each of 'd', 'm' and 'y", spe positions of day, month and year. The remaining two characters must be non-alphanur and specify the separators between the dmy characters. Examples of masks: d/m/y - reads dates like '1/12/2000', '19/Aug/2005' m,d/y - reads dates like '1,30/200', 'January, 12/2010' d m y - reads dates like '01 01 2000' , '4 Jul 2002'

-cy year	No	Specifies a base year to use for dates that have two-digit year values. The default base 1930, so the date '1/1/01' means '1/1/2000' and '1/1/36' means '1/1/1936'
-mn names	No	Specifies a comma-separated list of month names. There must be exactly twelve name they must appear in the order of the months.
-bdl	No	Outputs only records that fail the date conversion.
-bdx	No	Excludes records that fail the date conversion from output.

The following example reads the file birthdays.csv and transforms the dates in the second column into ISO format:

```
csvfix date_iso -f 2 -m 'd/m/y' data/birthdays.csv
```

which produces:

```
"Peter","2000-08-20"
"Jane","1970-02-12"
"Bill","1971-06-14"
"Anna","1976-12-27"
```


diff

The **diff** command compares two (and only two) CSV files and reports on differences between them. The diffing is done on the CSV field contents, not on the raw text, so the CSVfix diff command and other pure text diff utilities will almost certainly produce different results - this is intentional. The output of this command is not CSV - it indicates what changes need to be made to the left-hand file to turn it into the right-hand one. The output format is currently specific to CSVfix, but I intend to change this to make it more compatible with tools like patch. Currently, the format looks like this:

```
"-","3","3","three"
"+","5","6","six"
"-","3","3","three"
"+","3","7","seven"
```

which says that two lines need to be deleted from the left-hand file (indicated by the "-" values in the first column) and two lines need to be added (indicated by the "+" signs).

This command is currently "experimental" - it does work, but has not been heavily tested.

Flag	Req'd?	Description
-f fields	No	A comma-separated list indicating which fields to compare when performing ths diff - th default is to compare all fields.
-q	No	Don't report anything, but return zero if the files match, 1 if they do not and 2 on error.
-ic	No	Ignore case differences when performing a diff.
-is	No	Ignore leading and trailing spaces when performing a diff.

echo

The **echo** command echoes its CSV input to its output. It is intended mainly for debugging, sanity-checking CSVfix itself and as a way of listing CSV files in a standard format. Used with the -sep and -osep flags, it also provides a simple way to change the CSV inter-field separator.

See also: order

The following example displays the cities.csv file:

```
csvfix echo data/cities.csv
```

which produces:

```
"London","GB"
"Paris","FR"
"Edinburgh","GB"
"Amsterdam","NL"
"Rome","IT"
"Athens","GR"
"Berlin","DE"
```

To change the inter-field separator to a semicolon, and remove quoting, use:

```
csvfix echo -smq -osep ';' data/cities.csv
```

which produces:

```
London;GB
Paris;FR
Edinurgh;GB
Amsterdam;NL
Rome;IT
Athens;GR
Berlin;DE
```


edit

The **edit** command performs editing of input fields in much the same way as the UNIX utility *sed*. In order to fully use this feature, you will need to understand regular expressions. For simple replacements, you may be better of using the map command.

See also: escape, trim. eval, map

Flag	Req'd?	Description
`-e cmd`	Yes	Specifies the edit command to apply. The commands are modeled after those used by stream editor. Currently only one command is implemented, the s(earch/replace) comm `s/find/replace/opts` where: **find** is a regular expression to search for **replace** is the string to replace find with if found **opts** are options, which can be either i (ignore case) and/or g (replace all) Any number of commands can be specified - they will be applied sequentially.

		Matched expressions can be remembered using the \(*pattern*\) syntax. They can be re the replacement text using the values \1, \2..\9. As with sed, you can use any character you like to separate the parts of the edit comm and it may be more convenient to so so. For example, this command changes all occu of two slashes ("//") to two minus signs ("--"): `s://:--:g`
`-f fields`	No	Specifies a comma-separated list of fields to apply commands to. If none are specified commands are applied to all fields.

The following example removes the lower-case characters from the first field of the names.csv file:

```
csvfix edit -e 's/[a-z]//g' -f 1 data/names.csv
```

which produces:

```
"C","Dickens","M"
"J","Austen","F"
"H","Melville","M"
"F","O'Brien","M"
"G","Elliot","F"
"V","Woolf","F"
"O","Wilde","M"
```

The next example changes the M/F sex indicators into "Male" or "Female"

```
csvfix edit -e 's/M/Male/' -e 's/F/Female/' -f 3 data/names.csv
```

producing:

```
"Charles","Dickens","Male"
"Jane","Austen","Female"
"Herman","Melville","Male"
"Flann","O'Brien","Male"
"George","Elliot","Female"
"Virginia","Woolf","Female"
"Oscar","Wilde","Male"
```

This example shows recall of a patterns value, enclosing the sex indicator in a colon pair:

```
csvfix edit -e 's/\([MF]\)/:\1:/' -f 3 data/names.csv
```

producing:

```
"Charles","Dickens",":M:"
"Jane","Austen",":F:"
"Herman","Melville",":M:"
"Flann","O'Brien",":M:"
"George","Elliot",":F:"
"Virginia","Woolf",":F:"
"Oscar","Wilde",":M:"
```


erase

The **erase** command is used to remove fields from CSV records using regular expressions to specify the fields to remove.

See also: exclude

Flag	Req'd?	Description

`-f fields`	No	Specifies a comma-separated list of field indices identifying the fields in the input which considered for removal. You can use ranges to specify excluded fields, so;
		
		`-f 3:6`
		
		and:
		
		`-f 3,4,5,6`
		
		do the same thing. If no fields are specified, all fields are considered for removal.

`-r regex`	No	Test the fields specified by -f against the regular expression. If the expression matches fields are removed. Multiple -r options can be used; if any match, the fields are remove
`-n regex`	No	As for -r, but remove fields that do not match the expression.
`-k`	No	Keep records where all fields have been removed as a blank line. By default such lines removed from the output.

For example, this command:

```
csvfix erase -f 1:3 -r '^[A-Za-z]' foo.csv
```

removes any of the first three fields in the input where the field begins with an alphabetic character.

escape

The **escape** command is useful if the system that is going to consume your CSV data requires that certain characters in the input be escaped. For example, some systems require that single-quotes be escaped. Note this is an application level issue - CSVfix handles CSV and SQL escaping for you automatically, so you should only need to use this command if the application consuming your CSV has special escaping requirements.

See also: edit

Flag	Req'd?	Description
`-s chars`	Yes	Specifies a list of characters which need to be escaped.
`-e esc`	No	Specifies the string that will be placed before each occurrence of each character speci the -s flag. By default, escaping is performed with a single backslash. Any occurrence escape string in the input will themselves be escaped.
`-sql`	No	Specifies that SQL-style escaping of single quotes, replacing them with two single quo should be performed.
`-f fields`	No	Specifies the CSV fields to escape. If not provided, all fields are escaped.

The following example escapes all the lower-case vowels in all fields in the names.csv file:

```
csvfix escape -s "aeiou" data/names.csv
```

which produces:

```
"Ch\arl\es","D\ick\ens","M"
"J\an\e","A\ust\en","F"
"H\erm\an","M\elv\ill\e","M"
"Fl\ann","O'Br\i\en","M"
"G\e\org\e","Ell\i\ot","F"
"V\irg\in\i\a","W\o\olf","F"
"Osc\ar","W\ild\e","M"
```


eval

The **eval** command provides the ability to perform arithmetic and string manipulation on CSV input fields using a simple expression-based language. Each CSV field in a CSV record is assigned to a positional parameter, with $1 being the first, $2 the second, and so on. You can then use simple arithmetic expressions such as ($1 + $2)/2, which calculates the average of the first two fields. The results of these expressions are appended to the output, or replace existing fields if the -r flag is used. If you want to remove the base fields that **eval** used from the output, use the -d option.

Note that (depending on your command shell), the field place holders $1, $2 etc. may have a special meaning which will interfere with correct expression evaluation. To be safe, always enclose the expression being evaluated in quotes - single quotes on Linux, double quotes on Windows.

A problem with the eval command is that the if() function in the expression language will always evaluate all of its parameters, which means you cannot write code like this:

```
csvfix eval -e 'if( $2 == 0, "divide by zero", $1 / $2 )
```

as the divide expression will result in an exception being thrown if $2 contains zero. You can use the -if option to get round this:

```
csvfix eval -if '$2 == 0' -e '"divide by zero"' -e '$1 / $2'
```

Here, if field $2 contains zero, the divide expression will never be evaluated, and instead the message "divide by zero" is output to the CSV stream.

See also: edit, map, summary

Flag	Req'd?	Description
`-e expr`	No	Specifies an expression to evaluate. The result of the evaluation will be added as a new the end of the output row. You may have more than one -e flag - each one adds a new the output. At least one of -e or -r must be specified.
`-r field,expr`	No	As for -e, but replace specified field with result of expression evaluation.
`-d`	No	Discard input data, resulting in output which *only* contains the results of evaluating -r o flags. This saves having to pipe the output through order, if you only want evaluation re
`-if expr`	No	Evaluate the expression. If this results in a true value, then evaluate the next -e option, result is false, skip the next -e option, and evaluate the one after that.

The following example calculates the average of two temperatures in the minmax.csv file:

```
csvfix eval -e '($2 + $3)/2' data/minmax.csv
```

producing:

```
"2009-01-01","-5","2","-1.5"
"2009-01-02","-6","0","-3"
"2009-01-03","-5","2","-1.5"
"2009-01-02","-5","4","-0.5"
"2009-01-02","-3","6","1.5"
```


exclude

The **exclude** command is used to remove fields from CSV data. This can also be done with the order command, but exclude is more convenient when the data being processed contains records with variable numbers of fields, or when there are a large number of fields and you only want to exclude a few of them.

See also: order, erase

Flag	Req'd?	Description

`-f fields`	No	Specifies a comma-separated list of field indices identifying the fields in the input which removed. You can use ranges to specify excluded fields, so;
		
		`-f 3:6`
		
		and:
		
		`-f 3,4,5,6`
		
		do the same thing.

`-rf fields`	No	As for -f, but fields are specified from the end of the CSV record, so: `-f 1,2` excludes the last two fields in the records. **Note one of -f or -rf must be specified, but not both.**
`-if expr`	No	Evaluate expr using the eval expression language. If it evaluates to true, remove fields specified by -f.

The following example removes the surname field from the names.csv data:

```
csvfix exclude -f 2 data/names.csv
```

which produces:

```
"Charles","M"
"Jane","F"
"Herman","M"
"Flann","M"
"George","F"
"Virginia","F"
"Oscar","M"
```


exec

The **exec** command is used to execute an external command, using the CSV input data as command parameters. This command is intended to be used when the built-in features of CSVfix do not suffice to solve a problem.

See also: eval, call

Flag	Req'd?	Description
`-c cmd`	Yes	Specifies the command to be executed together with the parameters to use. Paramete specified by place-holders in the format %1, %2 etc. which indicate the first, second (a on) CSV fields. The command is executed once for each CSV input record. For examp `-c "echo %1 %4"` specifies that the echo command should be called for each CSV input record, with the in CSV input fields 1 and 4 as parameters. If you need a literal % character in the command, use a %% pair.
`-r`	No	By default, the command's output is parsed as CSV and the resulting column(s) are ap to the input row. Using the -r flag causes the command's output to replace all the input data and to be treated as plain text
`-ix ecode`	No	Tells the command to ignore exit codes from the command being executed which have less than or equal to ecode. This is useful for commands like grep which return non-ze values even though the command has worked. For example: `csvfix exec -ix 1 -c 'grep %1 data/books.csv' data/names.c` If the -ix option were not used you would get a "Command execution error" message if grep for a name failed.

The following example transliterates all lower-case vowels in the surname & forename fields of names.csv into an upper-case X character by using the UN*X tr command:

```
csvfix exec -c 'echo %1 %2 | tr aeiou X' data/names.csv
```

which produces:

```
"Charles","Dickens","M","ChXrlXs DXckXns"
"Jane","Austen","F","JXnX AXstXn"
"Herman","Melville","M","HXrmXn MXlvXllX"
"Flann","O'Brien","M","FlXnn O'BrXXn"
"George","Elliot","F","GXXrgX EllXXt"
"Virginia","Woolf","F","VXrgXnXX WXXlf"
"Oscar","Wilde","M","OscXr WXldX"
```


file_info

The **file_info** command prepends the filename and line number for the CSV input file in the output stream. It is useful for identifying where particular data came from when you are processing large numbers of files. If the input is actually from standard input, the special token `<stdin>` is used as the file name.

See also: sequence, put

Flag	Req'd?	Description
-b	No	Strips any path information from the file name.
-tc	No	Specifies that filename and line number should appear in two separate CSV fields.

The following example lists all rows from the cities.csv and names.csv data files. It prepends the base file name (no path information) and line number:

```
csvfix file_info -b data/cities.csv  data/names.csv
```

which produces:

```
"cities.csv (1)","London","GB"
"cities.csv (2)","Paris","FR"
"cities.csv (3)","Edinburgh","GB"
"cities.csv (4)","Amsterdam","NL"
"cities.csv (5)","Rome","IT"
"cities.csv (6)","Athens","GR"
"cities.csv (7)","Berlin","DE"
"names.csv (1)","Charles","Dickens","M"
"names.csv (2)","Jane","Austen","F"
"names.csv (3)","Herman","Melville","M"
"names.csv (4)","Flann","O'Brien","M"
"names.csv (5)","George","Elliot","F"
"names.csv (6)","Virginia","Woolf","F"
"names.csv (7)","Oscar","Wilde","M"
```


file_merge

The **file_merge** command is used to merge two or more sorted CSV files into a single sorted CSV stream. The input files should be sorted in ascending order.

See also: sort, unique

Flag	Req'd?	Description
-c fields	No	Comma-separated list of fields to compare while merging - default is to compare all fiel

file_split

The **file_split** command splits a CSV input stream into a number of files based on the values of specified fields in the CSV input stream. All the CSV records with the same values for those fields will be placed in the same file. By default, the created files are numbered, but you can also generate files based on the contents of the fields used to perform the split. Unlike most other CSVfix commands, this command does not write anything to standard output, or to any file specified by the -o flag.

Note that any existing files will be overwritten by this command, without warning. Use the -fd flag to locate the output files, and the -fp and -fx flags to name them.

See also: find, remove

Flag	Req'd?	Description
-f fields	Yes	Comma-separated list of filed indexes on which to base the split.

-fd dir	No	Specifies the directory in which to place the results of the split. Defaults to the current directory.
-fp prefix	No	Specifies the prefix to use when constructing file names. Default is file_
-fx ext	No	Specifies the extension to use when constructing file names. The default is csv
-ufn	No	Use the contents of the field(s) specified by the -f flag to generate file names. No check made that the fields contain valid file name components, and the command will fail if th not.

The following example splits the cities.csv file based on the second field, which contains the country code.

```
csvfix file_split -f 2 data/cities.csv
```

This produces the following files, each of which contains the cities for a particular country:

```
file_0001.csv
file_0002.csv
file_0003.csv
file_0004.csv
file_0005.csv
file_0006.csv
```

With the same data, the following example:

```
csvfix file_split -f 2 -ufn data/cities.csv
```

uses the country code values to generate the file names, producing:

```
file_DE.csv
file_FR.csv
file_GB.csv
file_GR.csv
file_IT.csv
file_NL.csv
```

Here, file_DE.csv will contain German cities, file_FR.csv French cities, and so on.

find

The **find** command is used to filter rows in the input depending on CSV field values. In this way, it is very similar to the UNIX *grep* command but, unlike grep, understands the CSV format, and can apply the search to specific CSV fields. To simulate grep -v, see the remove command.

See also: edit, remove

Flag	Req'd?	Description
-f fields	No	Specifies a comma-separated list of field indices identifying the fields in the input which used by the filter. If the -f flag is not specified, all fields are used.
-e expr	No	Specifies a regular expression which will be compared against the fields specified by t flag. If the expression matches, the row will appear in the command output. You can sp more than one expression, by using multiple -e flags, in which case the row will be out any of them match. Note that at least one of -e -ei -s -si -r -fc or -l flags must be speci
-s str	No	As for -e, but do not treat str as regular expression.
-ei expr	No	As for -e, but ignore case when performing comparison.

`-si str`	No	As for -ei, but do not treat str as regular expression.
`-n`	No	Instead of outputting matched records outputs the number of matches . This is useful w using CSVfix in scripts.

`-r range`	No	Specifies a range to search for. For example:
		
		`-r 10:50`
		
		would search for numbers in the range 10 - 50 inclusive.
		
		`-r A:C`
		
		would search for all strings that begin with A, B or C. If both elements of a range are nu then numeric searching is performed, and non-numeric values in the input will not be considered part of the range.
		
		Multiple ranges may be used, and as with the regex flags an input row will be output if range matches.

`-l length`	No	Search for fields having specific lengths. Lengths may be specified as a single number range:
		
		`-l 20`
		`-l 1:10`

`-fc count`	No	Specify that only rows with a certain number of fields will be found. The field count can specified as a single number or a range for example:
		
		`-fc 3`
		
		specifies rows with only three fields. whereas:
		
		`-fc 2:6`
		
		specifies rows containing between two and size fields inclusive.

`-if expr`	No	Evaluates the expression expr using the same expression language used by the eval command for each row. If the expression evaluates to true, the remaining options if any applied, otherwise the row is treated as not found.

The following example lists British cities in the cities.csv file:

```
csvfix find -e 'GB' -f 2 data/cities.csv
```

which produces:

```
"London","GB"
"Edinburgh","GB"
```

This further example lists only the entry for Paris (length 5) from the same data;

```
csvfix find -l 5 -f 1 data/cities.csv
```

This example lists all people in names.csv who's first name is longer than 5 characters:

```
csvfix find -if 'len($1)>5' data/names.csv
```


flatten

The **flatten** command is used to flatten multiple input rows into a single input row, depending on an identical key field or fields. This is useful if you have data like average temperature data for each month in a year (not all values shown):

```
1990, 0
1990, 0
1990, 4
...
1990, 2
1990, 1
1991, 1
...
```

and you would like them to look like:

```
1990, 0, 0, 4, ... 2, 1
1991, 1, ...
```

Values are pulled together so long as the key (in this case the year) for successive input rows is the same, and the flattened row is output when the key changes.

You can also use **flatten** to flatten master/detail records. For example, suppose you have this data file:

```
Joe,Blow
2012-01-13,Phone bill,105.10
Jane,Doe
2012-01-10,Groceries,5.10
2012-01-12,Drinks,7.15
```

and you would like to convert it to flat CSV file that looks like this:

```
Joe,Blow,2012-01-13,Phone bill,105.10
Jane,Doe,2012-01-10,Groceries,5.10
Jane,Doe,2012-01-12,Drinks,7.15
```

You can do that with this flatten command:

```
csvfix flatten -me '$fields == 2'  expend.csv
```

which says that a master record is one that consists of two fields. The **-me** option specifies an expression which tests each input record; if it evaluates to true, the record is treated as the master for the following detail records. The details are simply appended to the master.

See also: unflatten

Flag	Req'd?	Description
-k key	No	Specifies one or more key fields. Default is to use the first field as the key
-f data	No	Specifies the data fields. By default, all fields except the key are considered to be data
-r	No	Remove the key from the output. By default the key is retained and becomes the first f

		in the output.
-me expr	No	Specify an expression that can identify a master record in a stream of master/detail da above for more details. Mutually exclusive with the other options.

The following example flattens the data in the flat.csv file:

```
csvfix flatten data/flat.csv
```

producing:

```
"A","a1","a2","a3","a4"
"B","b1","b2"
"A","a5","a6"
"C","c1","c2","c3","c4"
```


from_xml

The **from_xml** command converts XML data into CSV records. Converting any arbitrary XML into arbitrary CSV requires a Turing-complete programming language, and CSVfix does not attempt to do this. However, it can convert most XML data into useful CSV records, which can then be further tweaked using other CSVfix commands.

To illustrate the use of the from_xml command, we will use the XML input file books.xml - this is in fact the output file produced by the to_xml command. Converting this XML data is pretty simple; this command line:

```
csvfix from_xml -re 'character' books.xml
```

produces the following CSV:

```
"Charles","Dickens","Bleak House","Esther Sumerson","Drippy heroine"
"Charles","Dickens","Bleak House","Inspector Bucket","Prototype detective"
"Charles","Dickens","Great Expectations","Pip","Deluded ex-blacksmith"
"Charles","Dickens","Bleak House","Mr Vholes","Vampiric lawyer"
"Jane","Austen","Emma","Emma Woodhouse","Smug Surrey goddess"
"Jane","Austen","Pride & Prejudice","Elizabeth Bennet","Non-drippy heroine"
"Jane","Austen","Pride & Prejudice","Mr Darcy","Proud, wet-shirted landowner"
```

How does it work? Well, the -re flag is used to specify the XML tag that marks the start of a new record, in this case 'character'. For each character tag, CSVfix outputs a record, using by default all the fields from the tag's parents, and the fields of the tag itself, plus any child tags. There is no way of specifying the field order - to do that you should pipe the output through the order command. If your XML contains multiple tags with the same name, you can narrow the tag matching by using a tag path fragment such as 'author@book@character'.

See also: to_xml

Flag	Req'd?	Description
-re tags	Yes	Specifies a comma-separated list of tags which will be used to mark the start of new re The tags may be simple, for example 'name', or be path fragments using the '@' charac a separator, for example 'character@name'.
-np	No	Do not output data from parent tags.
-nc	No	Do not output data from child tags.
-na	No	Do not output data from attributes.
-ip	No	Insert the path of the tag that produced the output as the first CSV field in the output.

-ml sep	No	Specify the separator string used within a CSV output field for multi-line text input data Default is a single space.

head

The **head** command displays the first N CSV records in a file. Note that this is not necessarily the same as displaying the first N lines, as CSV records may include the newline character. By default, the number of lines displayed is 10, but this can be changed using the -n option.

See also: tail

Flag	Req'd?	Description
-n records	No	Specifies how many records to display - default is 10.

inter

The **inter** command is used to interleave fields from two (and only two) CSV sources. The command reads a row from each sources, and then interleaves as specified by the -f flag. Reading stops when the first (left-hand) source is exhausted. If no -f flag is specified, the command appends the data from the second (right-hand) source to that from the first.

See also: order

Flag	Req'd?	Description
-f fields	No	Comma-separated list which specifies the order of the fields. This works in a way simil order command, but the field specifications are prefixed by an indicator of which sourc read from. So L3 means the third field from the first, left-hand source, while R1 means field of the second, right-hand source. If no -f flag is specified, all fields from the second source are appended to all fields from first.

The following example interleaves field 2 from the second source between fields 1 and 3 of the first source, using names.csv and dates.csv as input:

```
csvfix  inter  -f L1,R2,L3 data/names.csv data/dates.csv
```

producing:

```
"Charles","1/12/1980","M"
"Jane","23/4/1964","F"
"Herman","3/3/1878","M"
"Flann","Not A Date","M"
"George","","F"
"Virginia","","F"
"Oscar","","M"
```

join

The **join** command performs a relational join between two sets of CSV data. This command is intended to be used for enriching one CSV data set with data from another. There are two important points to note when using **join.** Firstly**,** This command treats files somewhat differently from other CSVfix commands. Each file from the command line except the last one will be joined, in turn against the last one. So in the following command line (from which the flags have been elided):

```
csvfix join ... file1.csv file2.csv file3.csv
```

file1.csv will be joined with file3.csv and output produced, then file2.csv will be joined with file3.csv and output produced. It follows from this that the **join** command requires at least two input streams. Note that either one of these may be the standard input stream, if required. Secondly, in the output from the command, fields that partook in the join are removed from the fields in the last file.

Flag	Req'd?	Description
`-f fields`	Yes	Specifies a comma-separated list of fields to join. Each entry in the list consists of two separated field indices: -f 1:2,3:3 This would specify that you wish to join field 1 from the first file with field 2 from the sec and field 3 from the first file with field 3 from the second. Note that the join command does not currently support joins between more than two fil
`-oj`	No	Specifies that an outer join should be performed. This will retain any rows in the first file cannot be joined to rows in the second - by default, such rows are removed.
`-inv`	No	Inverts the sense of the join so that only rows from the left hand side of the join with fie do not match rows on the right are output. This can be used to create exclusion lists. T is mutually exclusive with the -oj flag.
`-ic`	No	Ignore case for fields being joined, so a join on fields containing 'foo' and 'FOO' would s Default is to respect case.
`-k`	No	Retain both sets of fields partaking in the join.

The following example joins the cities.csv and countries.csv files to produce a list of cities with long country names:

```
csvfix join -f 2:1 data/cities.csv  data/countries.csv
```

which produces:

```
"London","GB","United Kingdom"
"Paris","FR","France"
"Edinburgh","GB","United Kingdom"
"Amsterdam","NL","Netherlands"
"Rome","IT","Italy"
"Berlin","DE","Germany"
```

lower

The **lower** command converts fields to lowercase.

See also: mixed, upper

Flag	Req'd?	Description
-f fieldlist	No	Specifies a comma-separated list of field indices identifying the fields in the input whic converted to lower-case on output. If the -f flag is not used, all fields are converted.

The following example converts the all fields in the names.csv file to lowercase:

```
csvfix lower data/names.csv
```

which produces:

```
"charles","dickens","m"
"jane","austen","f"
"herman","melville","m"
"flann","o'brien","m"
"george","elliot","f"
"virginia","woolf","f"
"oscar","wilde","m"
```


map

The **map** command allows you to map CSV field values from one value to another. You can do the same thing with the edit command, but **map** is generally easier to use for simple cases, as it does not require expertise in regular expressions.

See also: edit

Flag	Req'd?	Description
-f fields	No	Specified a comma-separated list of fields to perform mapping on. If not specified, all fi used. You can specify fields using the range syntax.
-fv values	Yes	Specifies a comma-separated list of values to map **from**. The items in the list are treat literal values, not regular expressions.
-tv values	Yes	Specifies a list of values to map **to**. This list may be empty but cannot contain more va than the from list. When mapping is performed, values in the **from** list are mapped to v the same position in the **to** list. If there is no matching value, the last value in in the **to** used. You can thus map multiple values in a **from** list to a single value in a **to** list. The special values $1, $2 .. $N are treated as the values of fields 1,2 .. N in the curren input record. If you want to have a literal $ sign as the first value in a field, use $$.
-ic	No	Specifies if case should be ignored when testing if a mapping should be performed. Th is to respect case.

The following example changes the country code of those countries in the countries.csv file which are in the Euro zone to 'EUR':

```
csvfix map -f 1 -fv 'FR,NL,IT,DE' -tv 'EUR' data/countries.csv
```

which produces:

```
"GB","United Kingdom"
"EUR","France"
"EUR","Germany"
"EUR","Netherlands"
"EUR","Italy"
"US","United States"
```


merge

The **merge** command merges together two or more input fields into a single field on output. It is useful for creating composite names from components, for example creating a person's full name from their forename and surname.

See also: split_char, printf, template

Flag	Req'd?	Description
-f fields	No	Specifies a comma-separated list of fields to merge together. If this flag is used, at lea fields must be specified. If the flag is omitted, all fields are merged into a single output
-s sep	No	Specifies the character(s) that will be used to separate the merged fields. Default is a space. The following special characters can be used: \t replaced by tab \n replaced by new line \r replaced by carriage return \\ replaced by single backslash All other characters are treated literally.
-p position	No	Specifies the position at which the merged fields will be inserted in the output. By defa position is that of the first field specified by the -f flag.
-k	No	If supplied, indicates that the fields that were merged should be retained in the output default is to remove them.

The following example merges the first two fields (forename and surname) of the names.csv file to produce a single name (using the -f flag) but retaining the original fields (using the -k flag). It then positions the single name at the end of the output rows, the fourth output field (using the -p flag):

```
csvfix merge -f 1,2 -k -p 4 data/names.csv
```

which produces:

```
"Charles","Dickens","M","Charles Dickens"
"Jane","Austen","F","Jane Austen"
"Herman","Melville","M","Herman Melville"
"Flann","O'Brien","M","Flann O'Brien"
"George","Elliot","F","George Elliot"
"Virginia","Woolf","F","Virginia Woolf"
"Oscar","Wilde","M","Oscar Wilde"
```

mixed

The **mixed** command converts fields to mixed case, capitalising the initial letter of each word.

See also: lower, upper

Flag	Req'd?	Description
-f fieldlist	No	Specifies a comma-separated list of field indices identifying the fields in the input whic converted to mixed case on output. If the -f flag is not used, all fields are converted.

The following example converts the operators.dsv file to CSV and then converts the CSV fields to mixed case. Note the pipe symbol in the command line:

```
csvfix  read_dsv data/operators.dsv | csvfix mixed
```

producing:

```
"Asterisk","*","Multiplication"
"Equals","=","Assignment"
"Pipe","|","Bitwise Or"
"Backslash","\","Not A C++ Operator"
```


money

The **money** command allows you to format CSV numeric fields as if they were currency amounts. For example, if you had a field containing the value 12400.6, and applied money formatting to it, the resulting field would be a string containing the value "12,400.60". The money command will only format decimal money amounts - i.e. the currency must follow the dollar and cents model.

See also: number

Flag	Req'd?	Description
-f fieldlist	No	Specifies a comma-separated list of field indices identifying the fields in the input whic formatted. If this is omitted, all fields will be formatted.
-r	No	Replace input fields with output - default is to append formatted fields.
-db chr	No	Specify decimal point character - default is "."
-ts chr	No	Specify thousands separator - default is ","
-cs sym	No	Specify currency symbol which will prefix the amount - default is none.
-ms minus	No	Specify string to use to indicate negative amounts - default is "-"
-ps plus	No	Specify string to use to indicate positive amounts - default is none.
-w width	No	Specify width of output field in which amount will be right-aligned - default is not to do alignment.
-cn	No	Treat the amount being formatted as in integer number of cents by dividing it by 100 be formatting.

number

The **number** commands converts formatted numeric values such as 1,234.56 to unformatted values on which arithmetic operations can be correctly performed, such as 1234.56. Numbers formatted using US/UK

English format (comma thousands separator, full-stop decimal separator) or European format (full-stop thousands separator, comma decimal separator).

See also: money

Flag	Req'd?	Description
-f fieldlist	No	Specifies a comma-separated list of field indices identifying the fields in the input whic converted. If none are specified, conversion will be attempted on all fields.
-fmt format	No	Format to convert from. Must be one of EN (US/UK English, the default), or EU (Europ
-ec	No	Treat conversion failure as an error. Normally, if conversion cannot be performed, the o value is used on output.
-es str	No	If conversion fails, use the string str as the output value.

odbc_get

The **odbc_get** command is used to extract data in CSV format from a database. The database must have a suitable ODBC driver installed on your system in order for this command to work.

This command is only available in the Windows version of CSVfix.

Flag	Req'd?	Description
-cs constr	No	Specifies an ODBC connection string to use to connect to the database. The exact for the connection string is driver specific, but the following are common settings: DRIVER = ODBC driver name SERVER = name (or IP address) of machine hosting SQL database DATABASE = SQL database name UID = database user id PWD = database password Fields are separated by semicolons - see below for an example connection string. Note that one of -cs or -dir must be specified, but not both.
-dir dirname	No	Uses the ODBC text file driver and specifies dirname as the default directory to look fo Using the text driver means you can perform SQL queries on CSV files in that directory this, the files must have the extension .CSV and must contain an initial filed-name rec
-sql stmt	No	A SQL statement that will be used to extract data from the database. If the stmt string with the '@' character it is treated as the name of a text file that contains the SQL stat to execute. **Note that no check is made that the statement is a SELECT - it is therefore poss use this command to execute any SQL, with possibly damaging effects.** You must specify one of -sql or -tbl, but not both.
-tbl table	No	Extract the named table or view as CSV. This is a shorthand for doing a SELECT * FR on the named table or view.
-ns nullstr	No	Specifies the string used to represent NULL data. By default, this is the empty string.
-h	No	Use SQL column names as CSV field header record on output.

The following example extracts data from the **jobs** table of the uses Microsoft SQL Server **pubs** database: running on the **dbs** server:

```
csvfix odbc_get -cs "server=dbs;driver=sql server;database=pubs;uid=sa;pwd=" -
tbl jobs
```

which produces:

```
"1","New Hire - Job not specified","10","10"
"2","Chief Executive Officer","200","250"
"3","Business Operations Manager","175","225"
"4","Chief Financial Officer","175","250"
"5","Publisher","150","250"
...
```

This example uses the **-dir** option to allow SQL queries to be performed on the files in the tests/data directory, using the Windows ODBC text driver. In this case, the army.csv file is being queried:

```
csvfix odbc_get -dir tests/data -sql 'select rank, name from army.csv order by
rank'
```

which produces the output:

```
"maj","black"
"maj","smith"
"pvt","pink"
"pvt","white"
"sgt","jones"
```


order

The **order** command is used to change the order of the fields in a CSV stream. You can also use this command to duplicate or remove fields from a stream. CSVfix now supports named fields as well as numbered ones, provided the input data contains a header record specifying the field names.

See also: pad, truncate, exclude. inter

Flag	Req'd?	Description
-f fieldlist	No	Specifies a comma-separated list of field indices identifying the order in which input fie appear in the command output. Each field may appear zero or more times. If a field do exist in the input, an empty string representing it is inserted in the output. Fields can be specified as individual numbers or as ranges, so these are equivalent: -f 1,2,3,4,5,10,17,16,15 -f 1:5,10,17:15 Ranges also work with other commands that expect a list of numeric field indexes.

-rf fieldlist	No	As for the -f flag, but indexes from the end of the CSV record, rather than the beginning one of -f or -rf can be specified.
-fn names	No	Specifies a list of named fields. This works in the same way as the numbered ffields, b CSV file(s) being ordered must contain a header specifying the field names. See the ex below for more details. Note that no CSVfix commands except **order** support the -fn flag directly. If you wish t named fields, you must always use order as the initial command in a pipeline, to get th into a known numerical order. **One (and only one) of -f, -rf or -fn must be specified.**
-nc	No	If a field does not exist in the input, do not create it in the output.

The following example takes the forename (first field) and sex (third field) fields from the names.csv file and swaps their order:

```
csvfix order -f 3,1 data/names.csv
```

which produces:

```
"M","Charles"
"F","Jane"
"M","Herman"
"M","Flann"
"F","George"
"F","Virginia"
"M","Oscar"
```

This further example uses the file army.csv, which contains a header specifying the names of the fields, and then uses the names to order the fields. It also uses the -ifn flag to remove the field name header row from the output.

```
csvfix order -ifn -fn rank,name data/army.csv
```

which produces:

```
"sgt","jones"
"maj","smith"
"maj","black"
"pvt","white"
"pvt","pink"
```

To illustrate the use of the -nc flag, the following command run on sales_quarter.csv:

```
csvfix order -f 4:1 data/sales_quarter.csv
```

produces:

```
"","550","200","2000"
"233","200","178","2001"
"119","104","55","2002"
"","","77","2003"
```

but:

```
csvfix order -f 4:1 -nc data/sales_quarter.csv
```

produces:

```
"550","200","2000"
"233","200","178","2001"
"119","104","55","2002"
"77","2003"
```


pad

The **pad** command is used to pad CSV data records out to some fixed size. It is useful when your CSV source consists of rows containing variable numbers of fields but the application that consumes the CSV requires a fixed number of fields.

See also: order, truncate, put

Flag	Req'd?	Description
`-n num`	Yes	Specifies the number of fields to pad to. If a row contains more than this number of fiel will be untouched - the **pad** command never removes rows.

`-p values`	No	By default, padding is performed using the empty string. You can also specify values t with using the -p flag. For example:
		
		`-p NULL`
		
		will pad with the string "NULL" while
		
		`-p NULL,0`
		
		will pad the first missing field with "NULL" and the remainder with "0".

The following example pads the sales_quarter.csv file with NULLS:

```
csvfix pad -n 5 -p NULL data/sales_quarter.csv
```

which produces:

```
"2000","200","550","NULL","NULL"
"2001","178","200","233","140"
"2002","55","104","119","NULL"
"2003","77","NULL","NULL","NULL"
```


pivot

The **pivot** command provides simple pivot table functionality.

Flag	Req'd?	Description
`-c field`	Yes	Field to use for column headers.

-r field	Yes	Field to use for row headers.
-f field	Yes	Fact field.
-a action	Yes	Action to perform on fact field - must be one of sum, avg or count.

For example, you could use the command to process the pivot.csv data (which represents sales of foods for different sales areas) to summarise the data like this:

```
csvfix pivot -c 2 -r 1 -f 4  -a sum data/pivot.csv
```

which says that you want the column headers in the output to be the food names, the row headers to be the sales areas, the fact field to be the numbers of sales, and the operation to perform on the fact field to be addition. This produces the output:

```
"","beans","bread","rice"
"north","37","7","13"
"south","31","21","10"
```


printf

The **printf** command formats each row of CSV input using a format string similar to that used in the C printf function. For example, the following format string:

```
"%d --  %12s -- %3.8f"
```

formats the first CSV field as an integer, the second as a string left padded to 12 character positions and the third as a real number with 3 places before the decimal point and 5 after. Each field is separated by a pair of literal minus signs. Note that if the fields in the CSV data are not in the required order for the printf command, you can easily swap them around using the order command in a pipeline prior to printf.

If there are fields for which there is no corresponding formatter, they are not output. If there are formatters with no matching field values, the empty string is used. If a numeric conversion is attempted for a non-numeric field, the field is treated as if it contained numeric zero.

You can tell the command to ignore a particular field by using the non-standard %@ formatter. This behaves as if the field had been formatted with %s, but outputs nothing.

Note that the output of the printf command is not CSV, unless you explicitly format it to be so yourself, or use the -csv option.

See also: template, order

Flag	Req'd?	Description
-fmt format	Yes	Specifies the printf()-style format string to use to format the CSV data
-f order	No	Works exactly like the -f option of the order command, specifying an ordering of fields will be used before the format specified by -fmt is applied. This is provided as a conven as it turns out that most uses of the printf command otherwise require input to be pipe through the order command.
-q	No	Applies CSV-style quoting to any double-quotes in strings produced by printf formatter not to literal double-quotes. For example, a field containing **Al "Scarface" Capone** wo converted to **Al ""Scarface"" Capone**.
-csv	No	Make output be a single CSV record, which is appended to the current input row.

The following example prints the initial and surname of the authors in the names.csv file:

```
csvfix  printf -fmt "%1.1s. %s" data/names.csv
```

producing:

```
C. Dickens
J. Austen
H. Melville
F. O'Brien
G. Elliot
V. Woolf
O. Wilde
```


put

The **put** command allows you to put a literal string or the contents of an environment variable into the CSV output. This can be useful for adding missing values and for tagging output.

See also: pad, eval, timestamp, sequence

Flag	Req'd?	Description
`-p pos`	No	Specifies the field position for the value being put, so that a position of "1" puts the valu the first field. If this flag is omitted, the value is placed at the end of the input fields.
`-v value`	Yes	Specifies the value to put. You must specify this or the -e flag.
`-e envvar`	Yes	Specifies the name of an environment variable who's value will be put into the output. Y specify this or the -v flag. The special names @DATETIME and @DATE can be used to put a date stamp (with c without the time part) into the output. The format of the stamp is: yyyy-mm-dd hh:mm:sss For greater control over the stamp format, use the timestamp command. The special name @COUNT can be used to add the field count of each CSV record to output.

The following example inserts the string "Name" into the first position in the names.csv file:

```
csvfix put -p 1 -v "Name" data/names.csv
```

which produces:

```
"Name","Charles","Dickens","M"
"Name","Jane","Austen","F"
"Name","Herman","Melville","M"
"Name","Flann","O'Brien","M"
"Name","George","Elliot","F"
"Name","Virginia","Woolf","F"
"Name","Oscar","Wilde","M"
```


read_dsv

The **read_dsv** command is used to convert delimiter separated values (DSV) data into CSV. DSV data is

most commonly found in the UN*X world - for example, the /etc/passwd file is a DSV file. DSV data consists of values separated by a delimiter character - the vertical bar/pipe character is often used for this. For example, here's the data from the names.csv file in DSV format:

```
Charles|Dickens|M
Jane|Austen|F
Herman|Melville|M
Flann|O'Brien|M
George|Elliot|F
Virginia|Woolf|F
Oscar|Wilde|M
```

If a value needs to contain the delimiter, then it must be escaped using a backslash, as must any occurrences of the backslash itself. Here's a file that explains some C++ operators (this is operators.dsv):

```
asterisk|*|multiplication
equals|=|assignment
pipe|\||bitwise OR
backslash|\\|not a C++ operator
```

See also: write_dsv

Flag	Req'd?	Description
`-f fields`	No	Specifies comma-separated list of fields to extract from the DSV data and the order th appear in the CSV. If not specified, all fields in the DSV are extracted. If a non-existent specified, it is treated as though it does exist but is empty.

`-s sep`	No	Specifies single character separator used in DSV file - the default is the pipe character can use this flag to allow reading of the popular tab-separated format by specifying a s tab character as the separator:
		
		`csvfix read_dsv -s '\t' tabsep.dat`
		
		Note that the tab character '\t' is the only currently supported "special" character sequ Spaces are not considered special, so to to read space-separated data, you would use
		
		`csvfix read_dsv -s ' ' spacesep.dat`

`-cm`	No	Collapse multiple occurrences of a separator in the DSV input into a single instance. T useful when reading space-separated files where the fields are separated by a variable of spaces.
`-csv`	No	Treat the contents of fields as if they were actually CSV data. If this is not specified, a fields are double-quoted CSV data, the double-quotes will be escaped on output, result extraneous quoting.

The following example converts the operators.dsv file to CSV:

```
csvfix read_dsv data/operators.dsv
```

which produces:

```
"asterisk","*","multiplication"
"equals","=","assignment"
"pipe","|","bitwise OR"
"backslash","\","not a C++ operator"
```

read_fixed

The **read_fixed** command is used to convert fixed-format data into CSV. You specify the fields in the fixed-format input that you want to be converted.

See also: write_fixed

Flag	Req'd?	Description
`-f fields`	Yes	Specifies a comma-separated list of fields that you want to convert. Each field consists start position in the input, a colon separator and the width of the field. For example: -f 4:10,20:2 says to read two fields from input, the first starting at position 4 and of length 10 and th second starting at position 20 and of length 2. Fields may overlap.
`-k`	No	Indicates that you want to keep any trailing spaces in the input values. making the out field values all be of the same length. The default is to remove them.

The following example reads the fixed_names.dat file:

```
csvfix read_fixed -f 1:10,11:1 data/fixed_names.dat
```

which produces:

```
"Geraldine","F"
"Fred","M"
"Emmylou","F"
```


read_multi

The **read_multi** command is used to convert multi-line data into CSV. From CSVfix's point of view, multi-line data is data that consists either of a fixed number of lines, or a variable number of lines ended with a special terminator string. For example, the following has two line per record:

```
Superman
Fortress of Solitude
Batman
The Batcave
```

while this uses a dash as a separator, allowing different numbers of fields per record:

```
Superman
Fortress of Solitude
-
Batman
The Batcave
Gotham City
-
```

In both cases CSVfix reads each line into a CSV field, so the second example above would produce:

```
"Superman","Fortress of Solitude"
```

```
"Batman","The Batcave","Gotham City"
```

If you need all the records to have the same number of fields, consider piping the output through the pad command.

Flag	Req'd?	Description
-n number	No	Specifies the number of lines in a multi-line record. This is currently arbitrarily limited t One of -n or -s must be specified, but not both.
-s sep	No	Specifies the separator to use for multi-line records. This must be one or more charact length.

remove

The **remove** command behaves exactly like the find command except that only input records that do not match a regular expression are written to output. It is this similar to the UN*X grep -v command, except that it understands CSV records and fields.

See also: find

Flag	Req'd?	Description
-f fields	No	Specifies a comma-separated list of field indices identifying the fields in th input which used by the filter. If the -f flag is not specified, all fields are used.
-e expr	No	Specifies a regular expression which will be compared against the fields specified by t flag. If the expression matches, the record will not appear in the command output. You specify more than one expression, in which case the record will be filtered out if any of match.
-ei expr	No	As for -e, but ignore case when performing comparison.
-s str	No	As for -e, but do not treat str as regular expression.
-si str	No	As for -ei, but do not treat str as regular expression.
-n	No	Instead of outputting unmatched records outputs the number of non-matches . This is when using CSVfix in scripts.
-r range	No	Specifies a range to search for. For example, -r 10:50 would search for numbers in the 10 - 50 inclusive, while -r A:C would search for all strings that begin with A, B or C. If b elements of a range are numeric, then numeric searching is performed, and non-numer values in the input will not be considered part of the range.
-l length	No	Search for fields having specific lengths. Lengths may be specified as a single number range.
-fc count	No	Specify that only rows with a certain number of fields will be removed. The field count c specified as a single number or a range. For example, -fc 3 specifies rows with only th fields. whereas -fc 2:6 specifies rows containing between two and size fields inclusive.
-if expr	No	Evaluates the expression expr using the same expression language used by the eval command for each row. If the expression evaluates to true, the row is removed.

The following example removes British cities from cities.csv:

```
csvfix remove -f 2 -e 'GB' data/cities.csv
```

producing:

```
"Paris","FR"
"Amsterdam","NL"
"Rome","IT"
"Athens","GR"
"Berlin","DE"
```


rmnew

The **rmnew** command allows you to remove newline characters from within CSV fields. Although CSV can cope with such embedded newlines, some applications cannot. A typical use of rmnew is if you have a file that looks like this

```
"Joe Public", "101 Somwhere St
Anytown
USA"
```

and you want:

```
"Joe Public", "101 Somwhere St, Anytown, USA"
```

The rmnew command to do this conversion is:

```
csvfix rmnew -s ',' addresses.csv
```

Flag	Req'd?	Description
`-s sep`	No	Specifies separator which will replace newlines in input. By default, this is a single spa character.
`-x`	No	Remove all data from a field following the first newline character.
`-f fields`	No	List of fields to apply the command to. Default is all fields.

rowsort

The **rowsort** command performs sorting of fields within a CSV record.

Flag	Req'd?	Description
`-f fields`	No	Fields to sort - default is all..
`-a`	No	Sort ascending - default
`-d`	No	Sort descending
`-l`	No	Perform lexicographical comparisons - default
`-n`	No	Perform numeric comparisons.

shuffle

The **shuffle** command randomly shuffles its input CSV records or fields and writes them to output.

This command can be useful when the CSV data is sorted in some way and you wants to insert into a data structure which performs better if inputs are randomised, or when you want to pick a few records at random from a file of CSV data.

To perform a record shuffle, all file contents are read into memory. This can make the command slow or unusable for very large input files.

See also: sort

Flag	Req'd?	Description
-n count	No	Specifies number of CSV records from input to be output. This has the effect of picking records at random (with no duplicates) from the command's input. Default is to output records.
-rs seed	No	Specifies random seed to use for randomising data. By default, the generator is seede the current date and time, which normally produces acceptable pseudo-random sequer The seed value should be an integer.
-f fields	No	Instead of shuffling records, shuffle the specified fields in each CSV record.

The following example picks three records at random from the names.csv file:

```
csvfix shuffle -n 3 data/names.csv
```

which produces:

```
"George","Elliot","F"
"Jane","Austen","F"
"Oscar","Wilde","M"
```


sequence

The **sequence** command adds sequence numbers to your CSV data. You can specify the starting number, the increment between numbers and whether the number should be padded with leading zeros.

See also: file_info

Flag	Req'd?	Description
-n start	No	Specifies the starting value for the sequence - default is 1.
-i inc	No	Specifies the increment between sequence numbers - default is 1. Note that negative i work - use the -d flag instead.
-d dec	No	Specifies decrement between sequence numbers, providing sequences that count dow Only one of -d and -i can be specified.
-p pad	No	Specifies the width of the field, which will be padded with leading zeros. If not specified performed.
-f pos	No	Specifies the field index of the sequence number in the output - default is first field.
-m mask	No	Provides a mask into which the sequence number will be inserted. The insertion point i the '@' character. So -m 'A@Z' will create sequence numbers like A1Z, A2Z etc. All the applied to the sequence number before it is inserted into the mask.

The following example adds sequence numbers beginning at 100 to the names.csv file, padding them to five digits:

```
csvfix sequence -n 100 -p 5 data/names.csv
```

which produces:

```
"00100","Charles","Dickens","M"
"00101","Jane","Austen","F"
"00102","Herman","Melville","M"
"00103","Flann","O'Brien","M"
"00104","George","Elliot","F"
"00105","Virginia","Woolf","F"
"00106","Oscar","Wilde","M"
```


sort

The **sort** command sorts CSV input data on one or more fields. You can specify ascending and descending order, and use alphabetic or numeric comparisons. In order to perform a sort, CSVfix currently reads all data into memory prior to sorting - you should therefore be cautious about using this command on very large data files.

See also: shuffle

Flag	Req'd?	Description
`-f fields`	No	Specifies a comma-separated list of fields to sort on. If not specified, the data is sorted **first field only**. For example, -f 4,1 specifies that the data should be sorted using field within that field on field 1:. Each field number in the list may be flowed by a number of flags, separated from the fi number by a colon. The following flags are currently supported: A or D - specifies Ascending (the default) or Descending order S, I or N - specifies whether the field is sorted as a String (alphabetically, the default), string Ignoring case differences, or as a Number. If sorting as a number, all fields must valid numeric data. As an example, 1:AN means sort the first input field numerically in ascending order. Th difference between the numeric and string sorting can be seen by considering the follo data 2 1 10 If sorted (ascending) numerically, we get: 1 2 10 but if using string sorting, we get: 1 10 2
`-rh`	No	Treat the first input record as a CSV header containing the column names and do not s but place it as the first record in the sorted output.

The following example sorts the names.csv file into descending order of sex and ascending order of surname:

```
csvfix.exe  sort -f 3:D,2 data/names.csv
```

which produces:

```
"Charles","Dickens","M"
"Herman","Melville","M"
"Flann","O'Brien","M"
"Oscar","Wilde","M"
"Jane","Austen","F"
"George","Elliot","F"
"Virginia","Woolf","F"
```


split_char

The **split_char** command splits a field within a CSV record into a number of sub-fields at boundaries marked by a specific character or sequence of characters. You can also split on transitions between alpha and numeric characters.

See also: split_fixed

Flag	Req'd?	Description
-f field	Yes	Index of the field that you want to split. The first field in a row has index 1.
-c chars	No	Specifies one or more characters which acts as the field separator on which to split th By default, this is a single space. Splitting on character is mutually exclusive with the transition flags.
-tan	No	Split on first transition from alpha character to numeric character.
-tna	No	Split on first transition from numeric to alpha character.
-k	No	Indicates if the field being split is retained in the output. By default it is removed.

The following example splits the second field of the emp.cvs file into sub fields delimited by a space :

```
csvfix split_char -f 2 data/emp.csv
```

which produces:

```
"1090M","Jeff","Smith"
"1099F","Annette","King"
"1170M","Bill","Thompson"
"1101M","Jeremy","Fisher"
"1088F","Lynn","Morrice"
```

This example illustrates splitting on a character type transition. In this case, we split the first field of idname.csv at the transition from number to character:

```
csvfix split_char -f 1 -tna data/idname.csv
```

which produces:

```
"1234","fred","m"
"22","bill","m"
```

```
"171171","lynn","f"
```


split_fixed

The **split_fixed** command splits a field into sub fields based on fixed positions.

See also: split_char, read_fixed

Flag	Req'd?	Description
`-f field`	Yes	Index of the field that you want to split. The first field in a row has index 1.
`-p poslist`	No	Specifies a list of positions at which the field should be split. Each position consists o and a length separated by a colon. For example: `-p 1:10,14:2` says that the field should be split into two sub fields, the first starting at position 1 (the beginning of the field) and having length 10, and the second starting at position 14 and length 2. Fields positions may overlap. A field position may specify a length which extends beyo bounds of an actual input field, in which case the non-existent characters are ignored. This option is mutually exclusive with the -l option.
`-l lengths`	No	Specifies a list of sub-field lengths that will be used to split the field. The lengths are a starting at the left-hand side of the input field. A single * character may be used to indi that a field has variable length. For example: `-l 2,*,3` says split into the first 2 characters, the last 3, and any number of characters between
`-k`	No	Indicates if the field being split is retained in the output. By default it is removed.

The following example splits the first field of the emp.csv file into numeric employee number and sex indicator:

```
csvfix split_fixed -f 1 -p 1:4,5:1 data/emp.csv
```

producing:

```
"1090","M","Jeff Smith"
"1099","F","Annette King"
"1170","M","Bill Thompson"
"1101","M","Jeremy Fisher"
"1088","F","Lynn Morrice"
```


split_regex

The **split_regex** command splits a field into sub fields based on regular expressions. You specify a number of remembered expressions using the \(.... \) regular expression syntax. These are matched against the specified input field and the matches are then recalled to form the split output.

Flag	Req'd?	Description
-f field	Yes	Index of the field that you want to split. The first field in a row has index 1.
-r regex	Yes	Specifies the regular expression that will be used to perform the split.
-ic	No	Specifies that the regular expression should ignore alphabetic case differences.
-k	No	Indicates if the field being split is retained in the output. By default it is removed.

The following example splits the first field of the idname.csv file into a numeric id, followed by the remainder of the field (the name, in this case):

```
csvfix.exe split_r -f 1 -r '\([0-9]*\)\(.*\)' idname.csv
```

producing:

```
"1234","fred","m"
"22","bill","m"
"171171","lynn","f"
```


sql_delete

The **sql_delete** command is used to generate SQL DELETE statements from CSV data. The CSV fields are used to specify the WHERE clause of the DELETE statement.

See also: sql_insert, sql_update

Flag	Req'd?	Description
-t table	Yes	Specifies the name of the SQL table to use in the UPDATE statement.
-w fields	Yes	Specifies the fields that will be used to generate the WHERE clause of the UPDATE statement. The list is comma-separated, with each pair being colon-separated. For exa -f 1:name,2:rank,5:serialno
-s separator	No	Specifies the separator that will be appended to the end of each statement. By default new line followed by a semicolon. If your database requires COMMITs after each insert could use something like this: -s '\n;\nCOMMIT\n\;\n'
-nq fields	No	Turns off SQL quoting. See the sql_insert command for full description.
-qn	No	Specifies that the special value NULL should be quoted. By default CSVfix does not qu NULL string (in whatever case).
-en	No	Convert empty CSV fields to NULL

The following example generates DELETE statements from the names.csv file. The SQL table is the same as that used in the example for sql_update:

```
csvfix sql_delete -t mailing -w 1:fname,2:sname data/names.csv
```

which produces:

```
DELETE FROM mailing WHERE fname = 'Charles' AND sname = 'Dickens'
;
DELETE FROM mailing WHERE fname = 'Jane' AND sname = 'Austen'
```

```
;
DELETE FROM mailing WHERE fname = 'Herman' AND sname = 'Melville'
;
DELETE FROM mailing WHERE fname = 'Flann' AND sname = 'O''Brien'
;
DELETE FROM mailing WHERE fname = 'George' AND sname = 'Elliot'
;
DELETE FROM mailing WHERE fname = 'Virginia' AND sname = 'Woolf'
;
DELETE FROM mailing WHERE fname = 'Oscar' AND sname = 'Wilde'
;
```


sql_insert

The **sql_insert** command generates SQL INSERT statements from CSV data. Once generated, you can use your favourite SQL tool (or isql) to run the statements against your database. The **sql_insert** command handles quoting of the apostrophe in names like O'Brien automatically.

See also: sql_delete, sql_update

Flag	Req'd?	Description
`-t table`	Yes	Specifies the name of the SQL table to use in the INSERT statement.
`-f fields`	Yes	Specifies a list of field index/field name pairs to use to generate the SQL statement. Th comma-separated, with each pair being colon-separated. For example: `-f 1:name,2:rank,5:serialno` specifies that field 1 will be called 'name', field 2 will be called 'rank' and field 5 will be c 'serialno'. Fields 3 and 4, which are not mentioned in the list, will be excluded from the generated INSERT statements.
`-s separator`	No	Specifies the separator that will be appended to the end of each statement. By default new line followed by a semicolon. If your database requires COMMITs after each insert could use something like this: `-s '\n;\nCOMMIT\n\;\n'`
`-nq fields`	No	By default, CSVfix wraps all SQL values in single quotes. This works well in most circumstances as SQL can implicitly convert the quoted strings to the actual data type However, some types (particularly dates and times) are not (depending on your SQL implementation) convertible and so must not be quoted. The -nq flag specifies a list of the CSV input which will not be quoted in the SQL output. Note that the special NULL not normally quoted.
`-qn`	No	Specifies that the special value NULL should be quoted. By default CSVfix does not qu NULL string (in whatever case).
`-en`	No	Convert empty CSV fields to NULL

The following example generates INSERT statements from the names.csv file:

```
csvfix sql_insert -t people -f 1:fname,2:sname data/names.csv
```

which produces:

```
INSERT INTO people ( fname, sname ) VALUES( 'Charles', 'Dickens')
;
INSERT INTO people ( fname, sname ) VALUES( 'Jane', 'Austen')
```

```
;
INSERT INTO people ( fname, sname ) VALUES( 'Herman', 'Melville')
;
INSERT INTO people ( fname, sname ) VALUES( 'Flann', 'O''Brien')
;
INSERT INTO people ( fname, sname ) VALUES( 'George', 'Elliot')
;
INSERT INTO people ( fname, sname ) VALUES( 'Virginia', 'Woolf')
;
INSERT INTO people ( fname, sname ) VALUES( 'Oscar', 'Wilde')
;
```


sql_update

The **sql_update** command generates SQL UPDATE statements from CSV data. To be useful, such statements require a WHERE clause, so the sql_update command provides means of specifying the table to update, the columns to change and the WHERE clause to use to locate the row(s) to be updated.

See also: sql_delete, sql_insert

Flag	Req'd?	Description
`-t table`	Yes	Specifies the name of the SQL table to use in the UPDATE statement.
`-f fields`	Yes	Specifies a list of field index/field name pairs to use to generate the SET clause of the statement. The list is comma-separated, with each pair being colon-separated. For exa `-f 1:name,2:rank,5:serialno` specifies that field 1 will be called 'name', field 2 will be called 'rank' and field 5 will be c 'serialno'. Fields 3 and 4, which are not mentioned in the list, will be excluded from the generated SET clause.
`-w fields`	Yes	Specifies the fields that will be used to generate the WHERE clause of the UPDATE statement. The format is the same as that used in the -f flag, described above.
`-s separator`	No	Specifies the separator that will be appended to the end of each statement. By default new line followed by a semicolon. If your database requires COMMITs after each insert could use something like this: `-s '\n;\nCOMMIT\n\;\n'`
`-nq fields`	No	Turns off SQL quoting. See the sql_insert command for full description.
`-qn`	No	Specifies that the special value NULL should be quoted. By default CSVfix does not qu NULL string (in whatever case).
`-en`	No	Convert empty CSV fields to NULL

The following example generates UPDATE statements. We assume that the file names.dat contains a list of people we want to send a new mail shot out to, and that we are updating a SQL table that looks like this:

```
CREATE TABLE mailing (
    fname VARCHAR(32),
    sname VARCHAR(32),
    need_mail CHAR
)
```

the CSVfix command line to do this is:

```
csvfix pad -n 4 -p 'Y'  data/names.csv |    \
csvfix sql_update -t mailing -f 4:need_mail -w 1:fname,2:sname
```

This works by using the pad command to append a 'Y' field to all rows and then feeds the modified data into the **sql_update** command. The resulting output is:

```
UPDATE mailing SET need_mail = 'Y' WHERE fname = 'Charles' AND sname = 'Dicken
s'
;
UPDATE mailing SET need_mail = 'Y' WHERE fname = 'Jane' AND sname = 'Austen'
;
UPDATE mailing SET need_mail = 'Y' WHERE fname = 'Herman' AND sname = 'Melvill
e'
;
UPDATE mailing SET need_mail = 'Y' WHERE fname = 'Flann' AND sname = 'O''Brien
'
;
UPDATE mailing SET need_mail = 'Y' WHERE fname = 'George' AND sname = 'Elliot'
;
UPDATE mailing SET need_mail = 'Y' WHERE fname = 'Virginia' AND sname = 'Woolf
'
;
UPDATE mailing SET need_mail = 'Y' WHERE fname = 'Oscar' AND sname = 'Wilde'
;
```


squash

The **squash** command allows you to reduce multiple rows with the same key field values to a single row, accumulating values of numeric fields as it does so.

See also: unique

Flag	Req'd?	Description
-f fields	Yes	Comma-separated list specifying the key fields. All CSV records with the same key wi squashed into a single record on output.
-n fields	Yes	Comma-separated list specifying numeric fields which will be accumulated by addition output together with the matching key fields. Numeric fields are assumed to be integer by default.
-rn	No	Specifies that the the fields described by the -n option should be treated as real numbe
-nn value	No	Specifies a numeric value that should be used when any of the fields specified with th option does not contain a numeric value. If this option is not used, encountering such a numeric value is a fatal error.

The following example squashes the rows in sales_region.csv using the first field as the key:

```
csvfix.exe squash -f 1 -n 2 data/sales_region.csv
```

producing:

```
"East","200"
"North","2793"
"South","1118"
"West","77"
```

stat

The **stat** command provides basic statistics on the number of records and fields in CSV data files. For example, running the command:

```
csvfix stat data/names.csv
```

produces this output:

```
"data/names.csv","7","3","3"
```

which says that names.csv contains 7 records, the shortest of which contains 3 fields, and the longest of which contains 3 fields. Note that the number of records is not necessarily the same as the number of lines in the file, as a CSV record can span multiple lines.

Using the -fs option produces a more detailed report, with one record per field in the input file. For example:

```
csvfix stat -fs data/army.csv
```

produces:

```
"data/army.csv","1","string","4","5"
"data/army.csv","2","string","3","4"
"data/army.csv","3","string","5","9"
```

where the second field is the index of the field, the third is the field's type, and the remaining two are the minimum and maximum field lengths. Using the -fn option with -fs produces output with named fields instead of indexes, assuming the input contains a field name record at its start. So:

```
csvfix stat -fs -fn data/army.csv
```

produces:

```
"data/army.csv","name","string","4","5"
"data/army.csv","rank","string","3","3"
"data/army.csv","serial_no","number","5","5"
```


summary

The **summary** command applies summarisation functions on CSV input data. The currently available functions are:

average - compute the average (mean) of individual numeric fields
frequency - provide frequency information on aggregated fields
median - compute median values for individual numeric fields
mode - identify modal records based on aggregated fields
min/max - identify minimum and maximum values for individual fields
sum - perform numeric summation on individual numeric fields

All of the above use the same format:

```
csvfix summary -flag fields
```

where -flag specifies the operation and fields is a list of field indexes to apply the operation to. Only a single flag can be specified per command.

See also: eval

Flag	Req'd?	Description
-avg fields	No	Calculates the numeric average of the each of the fields specified. The input fields mus numeric.The output is single row of CSV data containing the averages.
-frq fields	No	Calculates the frequency with which the aggregated fields values, considered as a key in the input. The output is identical to the input, but with the frequency prepended.
-max fields	No	Identifies the maximum values for the specified fields. The output is a single row of CS
-med fields	No	Calculates the median values of the specified fields. The input fields must all be numer output is a single row of CSV data.
-min fields	No	Identifies the minimum values for the specified fields. The output is a single row of CSV
-mod fields	No	Identifies the modal values for the specified fields. As with the -frq flag, the field conten considered to be a single key. The output is one or more rows of CSV data identifying modal values.
-sum fields	No	Performs arithmetic summation on the individual specified fields. The output is a single CSV data.

The following example calculates the averages of the two fields in numbers.csv:

```
csvfix summary -avg 1,2 data/numbers.csv
```

which produces:

"3","39.5"

This example adds frequency information regarding country (the second field in the input) to the data in cities.csv:

```
csvfix summary -frq 2 data/cities.csv
```

producing:

```
"2","London","GB"
"1","Paris","FR"
"2","Edinburgh","GB"
"1","Amsterdam","NL"
"1","Rome","IT"
"1","Athens","GR"
"1","Berlin","DE"
```


tail

The **tail** command displays the last N CSV records in a file. Note that this is not necessarily the same as displaying the first N lines, as CSV records may include the newline character. By default, the number of lines displayed is 10, but this can be changed using the -n option.

See also: head

Flag	Req'd?	Description
-n records	No	Specifies how many records to display - default is 10.

template

The **template** command passes its output through a textual template specified in a template file. Place holders in the template are replaced by values from the CSV data.

Template files look like this:

```
This is the first value {1}
And this {2} is the second
```

where {1} and {2} are place holders that will be replaced by the values of fields 1 and 2 in the command's input. If you need to output literal braces, escape them with backslashes.

Place holders can also specify expressions. To do this, place an @ character after the opening brace:

```
This is the first value {1}
And this {2} is the second
And this is a random number {@random()}
```

When using expressions, the field numbers must be preceded by the $ sign:

```
This is the first and second values concatenated: {@$1 . $2}
```

Normally, all of the output of the template command will be directed to standard output (or to a single file, if you use the -o option). However, it can be desirable to direct the output of each individual input CSV record to a separate file. To do this, you use the -fn option, which takes a string as a parameter. This string is itself treated as a template. So, if you have the the CSV data in a file called `heroes.csv`:

```
Batman,Batcave
Superman,Fortress Of Solitude
```

and a template file called `lairs.txt` that looks like this:

```
{1} hangs out in the {2}
```

then the command:

```
csvfix template -tf lairs.txt -fn output/{1}_lair.txt heroes.csv
```

will produce two output files in the output directory:

```
Batman_lair.txt
Superman_lair.txt
```

The first will contain:

```
Batman hangs out in the Batcave
```

and the second:

```
Superman hangs out in the Fortress Of Solitude
```

Note that the output of a template command is not CSV, unless you explicitly make it so in the template specification.

See also: printf

Flag	Req'd?	Description

-tf tfile	Yes	Name of template file.
-fn ftpl	No	Specifies a string containing a template which will be used to generate file names unde template output will be saved. If this option is used, each CSV input record generates a output file with the name specified by the template. **Important: No check is made for existing files, which will be overwritten!**

The following example lists the names.csv file using a template. The template file (names.tpl) looks like this:

```
Name: {1} {2}
Sex: {3}
```

The command line:

```
csvfix.exe template -tf data/names.tpl data/names.csv
```

produces:

```
Name: Charles Dickens
Sex: M
Name: Jane Austen
Sex: F
Name: Herman Melville
Sex: M
Name: Flann O'Brien
Sex: M
Name: George Elliot
Sex: F
Name: Virginia Woolf
Sex: F
Name: Oscar Wilde
Sex: M
```


timestamp

The **timestamp** command adds a timestamp field in YYYY-MM-DD HH:MM:SS format to the CSV input.

See also: date_format, date_iso

Flag	Req'd?	Description
-d	No	Output only the date portion of the timestamp
-t	No	Output only the time portion of the timestamp
-n	No	Output the timestamp as a numeric value, with no date/time field separators.
-rt	No	Update the stamp as new rows are read from input. By default, the same stamp is use rows input.

The following adds a timestamp to the names.csv data:

```
csvfix.exe timestamp data/names.csv
```

producing:

```
"2012-08-05 10:06:30","Charles","Dickens","M"
```

```
"2012-08-05 10:06:30","Jane","Austen","F"
"2012-08-05 10:06:30","Herman","Melville","M"
"2012-08-05 10:06:30","Flann","O'Brien","M"
"2012-08-05 10:06:30","George","Elliot","F"
"2012-08-05 10:06:30","Virginia","Woolf","F"
"2012-08-05 10:06:30","Oscar","Wilde","M"
```


to_xml

The **to_xml** command is used to generate XML data from CSV inputs. This command can be used in two ways; the simplest is to use it to generate an XHTML table using the <table>, <tr> and <td> tags. To do this, run the command without specifying a configuration file:

```
csvfix to_xml somefile.csv
```

The more interesting mode uses a configuration file to produce customised, tree-structured XML data. For example, suppose we have the following CSVdata (this is actually a shortened books.csv) which describes some books, their author and some characters:

```
Dickens,Charles,Bleak House,Esther Sumerson,Drippy heroine
Dickens,Charles,Bleak House,Inspector Bucket,Prototype detective
Dickens,Charles,Great Expectations,Pip,Deluded ex-blacksmith
Dickens,Charles,Bleak House,Mr Vholes,Vampiric lawyer
Austen,Jane,Emma,Emma Woodhouse,Smug Surrey goddess
Austen,Jane,Pride & Prejudice,Elizabeth Bennet,Non-drippy heroine
Austen,Jane,Pride & Prejudice,Mr Darcy,"Proud, wet-shirted landowner"
```

We can transform this data to XML by writing a configuration file (books.xsp):

```
# create XML describing some fictional characters
tag characters
   tag author group 1,2 attrib forename 2 attrib surname 1
      tag book group 3 attrib title 3
         tag character group 4
            tag name
               text 4
            tag description
               text 5
```

and running the CSVfix command:

```
csvfix to_xml -xf books.xsp books.csv
```

producing:

```
<characters>
    <author forename="Charles" surname="Dickens">
        <book title="Bleak House">
            <character>
                <name>
                    Esther Sumerson
                </name>
                <description>
                    Drippy heroine
                </description>
            </character>
```

```
        <character>
            <name>
                Inspector Bucket
            </name>
            <description>
                Prototype detective
            </description>
        </character>
    </book>
    <book title="Great Expectations">
        <character>
            <name>
                Pip
            </name>
            <description>
                Deluded ex-blacksmith
            </description>
        </character>
    </book>
    <book title="Bleak House">
        <character>
            <name>
                Mr Vholes
            </name>
            <description>
                Vampiric lawyer
            </description>
        </character>
    </book>
</author>
<author forename="Jane" surname="Austen">
    <book title="Emma">
        <character>
            <name>
                Emma Woodhouse
            </name>
            <description>
                Smug Surrey goddess
            </description>
        </character>
    </book>
    <book title="Pride &amp;amp; Prejudice">
        <character>
            <name>
                Elizabeth Bennet
            </name>
            <description>
                Non-drippy heroine
            </description>
        </character>
        <character>
            <name>
                Mr Darcy
            </name>
            <description>
                Proud, wet-shirted landowner
            </description>
        </character>
    </book>
</author>
```

```
/characters>
```

We'll now look at how the input data must be structured and how the configuration file is written. The CSV input data must be grouped in a way that reflects the final XML output. In this case, we have grouped the CSV by author names and book title. Note that the data does not have to be sorted (alphabetically or otherwise) but if all the same values are not grouped together in the input, they will be separated in the output - for example, "Mr Vholes" is separated from the other "Bleak House" characters because he is not grouped with them in the CSV input.

Now let's look at the individual lines of the configuration file The first line:

```
# create XML describing some fictional characters
```

is a comment. Any lines where the first non-whitespace character is a '#', or which consist entirely of whitespace, are ignored by CSVfix.

```
tag characters
```

This line specifies the root tag of the XML output, using the tag keyword and giving it the name "characters". All configuration files must specify a single root (i.e. they must specify well-formed XML). The next line:

```
   tag author group 1,2 attrib forename 2 attrib surname 1
```

is indented using a single tab character. The difference in indentation means that it is a child of the root tag. It has the name "author". It also uses the group keyword to specify that this tag is used to group together CSV input data which share common values for the first two fields (the 1,2 values). It then specifies that the tag will have two attributes (using the attrib keyword) that will have the names "forename" and "surname" and take their values from the second and first fields respectively. Note their is no requirement that any attribute values are the same as the group values, though this will often be the case. The next line:

```
      tag book group 3 attrib title 3
```

specifies a tag which is the child of the author tag (because it is more deeply indented, using two tabs) and is grouped on the third field within the author tag's grouping - the title field. It also specifies a single attribute called "title".

```
         tag character group 4
```

This line specifies a tag called "character" grouped on the fourth field that has no attributes. As the fourth field is unique (within its parent) , there will only be one input row that matches this grouping. That means that the next two lines:

```
            tag name
               text 4
```

do not require a group. If the group keyword is omitted, the tag is produced using the first rowof the grouping provided by the parent tag. In this case, we ouput a single "name" tag with no attributes which contains an XML text element, specified by the text keyword. Text elements are always enclosed by their parent tag and cannot themaseleves have child tags. The next two rows therefore specify tags at the same level as the name:

```
            tag description
               text 5
```

Text fields have XML quoting applied to them - for example "Pride & Prejudice" becomes "Pride & Prejudice". If you want to avoid this, you can use the cdata keyword instead, which wraps the output text in an XML CDATA section.

Some final things to note about this command:

- It cannot (and is not intended to) produce any arbitrary XML from any arbitrary CSV input. This would need a Turing complete language. It is intended to produce simple tree structures that closely mirror the CSV input.
- The command does not currently check that tag and attribute names adhere to XML standards.
- If your input records are not grouped in the way you need them, use the CSVfix sort command to group them appropriately.
- There is no way of combining fields using to_xml - instead use the merge, edit and other similar CSVfix commands to get your data in the right format.
- The algorithm this command uses make it necessary to read all input data into memory - this may make it slow or even unusable for very large CSV input files.

The to_xml command understands the following flags:

Flag	Req'd?	Description
`-xf file`	No	Specifies a configuration file defining how to produce XML from CSV. If omitted, a gene
`-in indent`	No	Specify the number of spaces to use for each level of indent in the XML output. If the s single tab character will be used for each level of indent.
`-et`	No	Specifies that a separate XML end tag will be generated even if a tag has no content. H configuration file is not used.

trim

The **trim** command removes leading and/or trailing white space (spaces and tabs) from input fields, or truncates fields to a specific width. By default, both leading and trailing spaces are trimmed, but you can change this with the -l and -t flags described below.

See also: edit

Flag	Req'd?	Description
`-f fields`	No	Species list of fields to trim. If no fields are specified, all fields are trimmed.
`-l`	No	Trims leading white space.
`-t`	No	Trims trailing white space.
`-w widths`	No	Trims fields to widths by removing rightmost characters. The widths parameter is a comma-separated list of width values. A negative value mea the field should not be trimmed. If the -f flag is also used, the width values refer to the f specified by that flag, otherwise they are 1-based. White space trimming (both leading trailing) is always performed before width trimming.
`-a`	No	Remove all whitespace.
`-s`	No	Reduce multiple consecutive whitespaces to single space.

The following example trims both leading and trailing whitespace from the spaces.csv file:

```
csvfix trim data/spaces.csv
```

which produces:

```
"1","leading"
"2","trailing"
"3","both"
```

This example truncates all fields in names.csv to a single character:

```
csvfix trim -w 1,1,1 data/names.csv
```

producing:

```
"C","D","M"
"J","A","F"
"H","M","M"
"F","O","M"
"G","E","F"
"V","W","F"
"O","W","M"
```


truncate

The **truncate** command truncates CSV data by removing rightmost fields. The command will never add any fields, so truncating data which has rows containing fewer than the truncation value will not affect those rows in any way. In this way it behaves somewhat differently from the **order** command, which would append empty fields if none existed in the input.

See also: order, pad

Flag	Req'd?	Description
-n count	Yes	Specifies how many fields to truncate to.

The following example truncates the names.csv file to two fields per row:

```
csvfix truncate -n 2 data/names.csv
```

which produces:

```
"Charles","Dickens"
"Jane","Austen"
"Herman","Melville"
"Flann","O'Brien"
"George","Elliot"
"Virginia","Woolf"
"Oscar","Wilde"
```


unflatten

The **unflatten** command is used to convert multiple values on the same row into multiple rows. It is the inverse of the flatten command.

Flag	Req'd?	Description
-k key	No	Specifies one or more key fields. Default is to use the first field as the key
-n ndata	No	Specifies how many data items to output on each row - default is 1.

The following example converts the file unflat.csv into multiple rows:

```
csvfix unflatten data/unflat.csv
```

producing:

```
"1","a"
"1","b"
"1","c"
"2","d"
"2","e"
"3","f"
"3","g"
"3","h"
"3","i"
```


unique

The **unique** command is used to reduce rows that contain duplicate field values to a single row. The single row chosen to represent the duplicates will be the first on encountered in the input file. You can also specify that you want to output only the duplicates. Note that this command does not require that its input is sorted, but does require that all data be read into memory, which may make it slow or unusable for very large datasets.

See also: sort

Flag	Req'd?	Description
`-f fields`	No	A comma-separated list of fields to test for uniqueness. If not specified, each complete record is tested.
`-d`	No	Specifies if only duplicate fields should be output. This is the converse of the default be which is to only output unique fields.

The following example lists rows from the post.csv file where the first field value occurs more than once:

```
csvfix unique -d -f 1 data/post.csv
```

which produces:

```
"London","NW"
"London","W"
"London","E"
"London","SE"
"London","SW"
```

You can use the **unique** command to merge two or more CSV files into one, discarding any duplicate rows:

```
csvfix unique -o merged.csv file1.csv file2.csv
```

This assumes that the two input files have the possibly duplicate fields in the same order in the CSV records.

upper

The **upper** command converts fields to upper-case.

See also: lower, mixed

Flag	Req'd?	Description
-f fieldlist	No	Specifies a comma-separated list of field indices identifying the fields in the input whic converted to upper-case on output. If the -f flag is not used, all fields are converted.

The following example converts the surname (second field) in the names.csv file to upper-case:

```
csvfix upper -f 2 data/names.csv
```

which produces:

```
"Charles","DICKENS","M"
"Jane","AUSTEN","F"
"Herman","MELVILLE","M"
"Flann","O'BRIEN","M"
"George","ELLIOT","F"
"Virginia","WOOLF","F"
"Oscar","WILDE","M"
```


validate

The **validate** command is used to validate CSV data against a number of validation rules. This command does not validate the basic CSV syntax (the input to it must be syntactically correct CSV) - it's intended to be used to validate business rules. If you need to test that a file contains valid CSV data, use the check command.

See also: check

Flag	Req'd?	Description
-vf file	Yes	Specifies the file containing the validation rules.
-om mode	No	Specifies the output mode. Possible values are: **report**, which displays the filename, line number and validation error message for each This is the default **pass**, which displays all rows that pass validation **fail**, which displays all rows that fail the validation
-ec	No	If this option is specified, the validate command returns a value of 2 to the host operati system on validation failure. Without this option, validation failure does not return an er to the OS.

The validation rules are contained in a text file. Here's an example:

```
# val_names.txt
#
# check that:
```

```
#   each row in the file has at least three fields
#   all fields contain some non-whitespace data
#   the third field contains only the values "M" and "F"

required    1,2,3
notempty    1,2,3
values      3       "M" "F"
```

The format of the validation file is fairly straightforward. Lines starting with '#' are comments, other lines consists of two or more fields, separated by spaces. The fields are as follows:

- The first field contains the name of the rule - available rules are listed in the table below.
- The second field contains a comma-separated list of CSV field numbers to apply the rule to. There must be at least one field number.
- The remaining fields consist of parameters for specific rules - different rules use different parameters.

Once you have a validation file you, you can use it to validate CSV data:

```
csvfix validate -vf val_names.txt data/names.csv
```

This will produce no output, because all the data in names.csv pass all the validation rules. However, if you try it with another file, like bad_names.csv:

```
csvfix validate -vf val_names.txt data/bad_names.csv
```

you get a list of rows and fields that fail the validation rules:

```
data/bad_names.csv (2): Jane,,F
    field: 2 - field is empty
data/bad_names.csv (4): Flann,O'Brien,X
    field: 3 - "X" is invalid value
data/bad_names.csv (5): George,Elliot
    field: 3 - required field missing
data/bad_names.csv (6): Virginia,
    field: 3 - required field missing
```

The following rules are currently available:

Rule	Description
`required`	The field(s) must exist in the CSV, though they may be empty. For all other rules, it is not required that the field exists, so if a field must exist, it must be tested with this rule.
`notempty`	The field(s) must not contain only white space.
`fields`	Specifies a minimum and maximum number of fields in each row. This doers not require a field list: fields * 2:4 says we need a minimum of 2 and a maximum of 4 fields. If you don't need a range, make the minimum and maximum values the same.
`length`	The length of the field(s) must be between specified minimum and maximum values: length 1,2,3 10:20
`numeric`	The field must contain a numeric value. Additionally, you can specify a number of ranges as parameters. For example numeric 1 1:1000 -1,-1

	specifies that field 1 must be numeric and in the range 1 to 1000 (inclusive) or in the range -1 to -1 (i.e. it may also have the value -1)
values	The field must contain one of a number of values. For example: values 2 'EUR' 'USD' 'GPP' says that field 2 must contain one of EUR, USD or GBP. Values may be contained in single or double quotes, or simply be space-separated.
notvalues	As above, but the field must not contain the listed values.
lookup	Lookup one or more fields in a second CSV file (actually, you can use the same CSV file as the one you are validating, which is useful in some recondite circumstances). For example: lookup * 1:4,2:7 data/lookupfile.csv Here, the field list is not needed, so an asterisk is used as a place holder. The first parameter is a comma-separated list of field number pairs. The first value of each pair indicates the field in the current file and the second the field in the lookup file
date	Check that field is a valid date. The format of the date is specified by a mask value that must be supplied. See the date_iso command for details of mask format. An optional date range, consisting of two dates in ISO format separated by a colon can also be provided. For example: date 1,2 'd/m/y' 2000-1-1:2010-12-31 validates dates in dd/mm/yyyy format and checks that they are in the first decade of the 21st century.

The following example validates the cities.csv file against countries.csv. The validation file looks like this:

```
# val_country.txt
# lookup second field in cities.csv against
# the first field in countries.csv
required    1,2
lookup      *      2:1  data/countries.csv
```

The command line to use it is:

```
csvfix validate -vf rules/val_country.txt data/cities
```

which produces the following output (because Greece is not in the countries.csv file):

```
data/cities.csv (6): Athens,GR
    lookup of 'GR' in data/countries.csv failed
```


write_dsv

The **write_dsv** command is used to convert CSV data to DSV format. DSV format is explained in the page describing the read_dsv command.

See also: read_dsv

Flag	Req'd?	Description
-f fields	No	Specifies comma-separated list of fields to write to DSV. If not specified, all fields are w
-s sep	No	Specifies single character separator used in DSV file - default is the pipe character.

The following example outputs the forename and surname of names.csv as DSV:

```
csvfix.exe write_dsv -f 1,2 data/names.csv
```

which produces:

```
Charles|Dickens
Jane|Austen
Herman|Melville
Flann|O'Brien
George|Elliot
Virginia|Woolf
Oscar|Wilde
```


write_fixed

The **write_fixed** command produces fixed-format output. This can be useful if you need to convert CSV files into something acceptable to other (probably legacy) systems. To produce the output, you specify the CSV fields you want it to contain and their widths.

See also: read_fixed, split_fixed

Flag	Req'd?	Description
-f fields	Yes	Specifies a comma-separated list of fields that you want to output. Each field consists field index in the output, a colon separator and a width to pad or truncate the field to.
-ru	No	Outputs an 80-column ruler before outputting any data. This can be useful for checking you have your data formatted correctly.

The following example outputs the names.csv file in fixed format with 16 characters for the surname, 16 for the forename and one for the sex:

```
csvfix write_fixed -f 1:16,2:16,3:1 data/names.csv
```

which produces:

```
Charles         Dickens         M
Jane            Austen          F
Herman          Melville        M
Flann           O'Brien         M
George          Elliot          F
Virginia        Woolf           F
Oscar           Wilde           M
```


write_multi

The **write_multi** command produces multi-line master/detail records from CSV input.

For example, given this CSV input:

```
bob,dylan,bringing it all back home
bob,dylan,blonde on blonde
bob,dylan,john wesley harding
nick,drake,bryter later
nick,drake,pink moon
```

the write_multi command:

```
csvfix write_multi -m 1,2 -rs '-----' -smq
```

would produce:

```
bob,dylan
bringing it all back home
blonde on blonde
john wesley harding
-----
nick,drake
bryter later
pink moon
-----
```

See also: read_multi

Flag	Req'd?	Description
-m fields	Yes	Specifies a comma-separated list of fields that form the master record.
-d fields	No	Specifies a comma-separated list of fields that form the detail record. If not specified, the fields not specified by the -m option are assumed.
-rs sep	No	Specifies separator to be output at the end of the detail records.

Data Files

The examples in this manual use the sample data files that ship with **CSVfix**. The contents of the files are listed here for easy reference.

army.csv

```
name,rank,serial_no
jones,sgt,12345
smith,maj,34342
black,maj,56431
white,pvt,17139
pink,pvt,67543
```


bad_names.csv

```
Charles,Dickens,M
```

```
Jane,,F
Herman,Melville,M
Flann,O'Brien,X
George,Elliot
Virginia,
Oscar,Wilde,M
```


birthdays.csv

```
Peter,"20/8/2000"
Jane,"12/2/1970"
Bill,"14/Jun/1971"
Anna,"27/12/1976"
```


books.csv

```
Dickens,Charles,Bleak House,Esther Sumerson,Drippy Heroine
Dickens,Charles,Bleak House,Inspector Bucket,Prototype detective
Dickens,Charles,Great Expectations,Pip,Deluded ex-blacksmith
Dickens,Charles,Bleak House,Mr Vholes,Vampiric lawyer
Austen,Jane,Emma,Emma Woodhouse,Smug Surrey goddess
Austen,Jane,Pride & Prejudice,Elizabeth Bennet,Non-drippy heroine
Austen,Jane,Pride & Prejudice,Mr Darcy,Proud, wet-shirted landowner
Melville,Herman,Moby Dick,Queeqeg,Tattooed harpooneer
Melville,Herman,Moby Dick,Moby Dick,Great white whale
```


books.xml

```
<characters>
    <author forename="Charles" surname="Dickens">
        <book title="Bleak House">
            <character>
                <name>
                    Esther Sumerson
                </name>
                <description>
                    Drippy heroine
                </description>
            </character>
            <character>
                <name>
                    Inspector Bucket
                </name>
                <description>
                    Prototype detective
                </description>
            </character>
        </book>
        <book title="Great Expectations">
            <character>
                <name>
                    Pip
                </name>
```

```
                <description>
                    Deluded ex-blacksmith
                </description>
            </character>
        </book>
        <book title="Bleak House">
            <character>
                <name>
                    Mr Vholes
                </name>
                <description>
                    Vampiric lawyer
                </description>
            </character>
        </book>
    </author>
    <author forename="Jane" surname="Austen">
        <book title="Emma">
            <character>
                <name>
                    Emma Woodhouse
                </name>
                <description>
                    Smug Surrey goddess
                </description>
            </character>
        </book>
        <book title="Pride &amp;amp; Prejudice">
            <character>
                <name>
                    Elizabeth Bennet
                </name>
                <description>
                    Non-drippy heroine
                </description>
            </character>
            <character>
                <name>
                    Mr Darcy
                </name>
                <description>
                    Proud, wet-shirted landowner
                </description>
            </character>
        </book>
    </author>
</characters>
```


cities.csv

```
London,GB
Paris,FR
Edinburgh,GB
Amsterdam,NL
Rome,IT
Athens,GR
```

```
Berlin,DE
```


countries.csv

```
GB,United Kingdom
FR,France
DE,Germany
NL,Netherlands
IT,Italy
US,United States
```


dates.csv

```
Jim,1/12/1980
Pete,23/4/1964
Ann,3/3/1878
Bad,Not A Date
```


emp.csv

```
"1090M","Jeff Smith"
"1099F","Annette King"
"1170M","Bill Thompson"
"1101M","Jeremy Fisher"
"1088F","Lynn Morrice"
```


fixednames.dat

```
Geraldine F
Fred      M
Emmylou   F
```


flat.csv

```
A,a1,a2
A,a3,a4
B,b1,b2
A,a5,a6
C,c1,c2
C,c3,c4
```


idname.csv

```
1234fred,m
22bill,m
171171lynn,f
```

minmax.csv

```
2009-01-01,-5,2
2009-01-02,-6,0
2009-01-03,-5,2
2009-01-02,-5,4
2009-01-02,-3,6
```


names.csv

```
Charles,Dickens,M
Jane,Austen,F
Herman,Melville,M
Flann,O'Brien,M
George,Elliot,F
Virginia,Woolf,F
Oscar,Wilde,M
```


numbers.csv

```
1,17.0
6,100
3,42
2,-1
```


operators.dsv

```
asterisk|*|multiplication
equals|=|assignment
pipe|\||bitwise OR
backslash|\\|not a C++ operator
```


pivot.csv

```
north,beans,2014-01-01,12
north,beans,2014-01-01,10
north,rice,2014-01-02,5
north,bread,2014-01-02,7
north,beans,2014-01-03,15
north,rice,2014-01-03,8
south,bread,2014-01-01,13
south,beans,2014-01-01,19
south,rice,2014-01-02,1
south,rice,2014-01-02,9
south,beans,2014-01-03,12
south,bread,2014-01-03,8
```

post.csv

```
London,NW
London,W
London,E
Edinburgh,EH
London,SE
Lincoln,LN
Manchester,M
London,SW
```


sales_region.csv

```
South,1040
East,200
South,78
North,2023
West,77
North,770
```


sales_quarter.csv

```
2000,200,550
2001,178,200,233,140
2002,55,104,119
2003,77
```


simple.xml

```
<table>
  <tr>
    <td>Lucky</td><td>7</td>
  </tr>
  <tr>
    <td>Beast</td><td>666</td>
  </tr>
  <tr>
     <td>Meaning</td><td>42</td>
  </tr>
  <tr>
     <td>Gross</td><td>144</td>
  </tr>
</table>
```


spaces.csv

```
1,"     leading"
2,"trailing    "
3,"    both    "
```

unflat.csv

```
"1","a","b","c"
"2","d","e"
"3","f","g","h","i"
```


Terminology

This section contains explanations of some of the terminology used in this manual.

Comma-Separated List

CSVfix makes heavy use of comma-separated lists as the parameters for command-line flags. Such a list consists (surprisingly) of a number of values separated by commas. There should be no spaces after or before the comma. Some examples:

```
1,2,3,4

1:4,42:1,7:6

1:foo,2:bar
```


Expression Language

This page describes the language used for expressions by eval and other CSVfix commands.

The following data types are supported:

Data Type	Description
`string`	All expression values are stored as strings and can be treated as strings. Strings are denoted by single or double quotes.
`number`	A value can be treated as a number if it contains a valid real or integer number.
`boolean`	All values can be treated as booleans. An empty string or a numeric value of zero are taken to be all other values are true.

The following operators are supported:

Operators	Description
`* / %` `+ -`	Standard arithmetic binary operators. The operands must be numbers.
`&& ||`	Boolean AND and OR operators - the operands may be of any type but will be treated as booleans Note there is no NOT operator - see the not() function. also, CSVfix does not currently support sho circuited evaluation using these operators. This means that you cannot write expressions like this `$1 <> 0 && $2 / $1`

	as you will get a divide-by-zero error if $1 contains zero.
.	String concatenation - operands may be of any type but will be treated as strings.
== <> != < > <= >=	Comparison operators - the operands may be of any type. If both operands can be interpreted as numbers then numeric comparison is performed, otherwise a lexical comparison is used.

The following functions are available.

Function	Description
abs(num)	Returns the absolute (positive) value of num, which must be numeric.
bool(val)	Converts val to the boolean values 1 (true) or 0 (false)
day(d)	If d is a date in ISO YYYY-MM-DD format, returns the day part, otherwise returns empty string.
env(name)	Returns the value of the named environment variable, or an empty string if no such variable exists.
errorif(cond,rc,msg)	If the first parameter evaluates to true, write msg to standard error and exit CSVfix immed with return code rc.
field(i)	Returns the i'th field in the current CSV input record. If the index specified is less than 1, greater than the number of fields in the current row, returns the empty string.
find(regex)	Try to match each field in the current CSV record with regex, returning the numeric index the first match, or 0 if no match is found.
if(test,v1,v2)	Evaluates the expression test (which itself may consist of expressions and functions) as a boolean value - if the evaluation is true returns the result of evaluating the expression v1, e returns the result of evaluating v2. CSVfix does not currently support short-circuited evalua so both v1 and v2 must be valid expressions - the problems this causes can partially be mitigated by using the -if option of the eval command.
index(s,list)	Returns 1-based index of s in a comma-separated list. If s is not in the list, returns zero.
int(num)	Returns the integer part of num, which must be numeric. For example, int(12.34) would re 12.
isdate(d)	Returns true if d is a date in ISO YYYY-MM-DD format.
isempty(str)	Returns true if the string str contains only whitespace or is the empty string, returns false otherwise.
isint(str)	Returns true if the string str contains an integer, false otherwise.
isnum(str)	Tests if str is a valid number (either integer or real).
len(str)	Return length of string.
lower(str)	Returns the string str converted to lower-case.
match(str,regex)	Returns true if str matches regular expression.
max(a,b)	Returns the larger of the pair a,b. If both can be converted to numbers, numeric comparis performed, otherwise string comparison is used. For example, max(42,666) would return 6 and max('foo','bar') would return 'foo'.
min(a,b)	As for max(), but returns minimum value.
month(d)	If d is a date in ISO YYYY-MM-DD format, returns the month part, otherwise returns emp string.
not(bool)	Inverts the boolean sense of bool, which may be of any type.
pick(i,list)	Picks the 1-based entry in the comma-separated list using i as index. If i is less than one greater than the number of entries in the list, returns the empty string.
pos(s1,s2)	Returns the 1-based position of the string s2 in s1, or zero if s1 does not contain s2.
random()	Returns random real number in the range 0.0 <= N < 1.0. The generator can be seeded w the -seed command line option.
round(num,places)	Returns the number num, rounded to places decimal places.
sign(num)	Returns the sign of num. Negative numbers return -1, positive numbers 1, zero returns 0.

streq(s1,s2)	Returns true if string s1 is identical to s2, ignoring case differences.
substr(s,start,n)	Returns a substring of s starting at start and consisting of a maximum of n characters.
trim(str)	Returns the string str trimmed of leading and trailing whitespace.
upper(str)	Returns the string str converted to upper-case.
year(d)	If d is a date in ISO YYYY-MM-DD format, returns the year part, otherwise returns empty string.

The following read-only variables are set before an expression is evaluated. Variable names are case-insensitive.

Variables	Description
$1 $2 ... $N	Values of the fields in the current CSV input row. If a field variable is used that represents a field th does not exist in the input row, it evaluates as the empty string. Note you will probably need to quote eval command options that use field variables to prevent them being interpreted as shell script variables.
$fields	The number of fields in the current input row.
$file	The name of the current input file.
$line	Line number of the current input line in the current file.

Fixed-format Data

Fixed-format places data fields in fixed positions on an input line. For example, in the following fixed-format data the name occupies 10 character positions starting at position 1 and the sex indicator one position, starting at position 11.

```
12345678901234567890
Geraldine F
Fred       M
Emmylou    F
```

CSVfix supports reading and writing of fixed-format via the read_fixed and write_fixed commands.

Regular Expressions

Regular expressions are used for pattern matching in a number of CSVfix commands. The regular expression special characters understood by CSVfix are as follows:

Character	Meaning
.	Match any single character
*	Match zero or more occurrences of the preceding character or range.
[]	Specify character range
^	Match start of CSV field - if first character in [] brackets, negates a range
$	Match end of CSV field.
\(pat\)	Remember the matched pattern for later use
\n	Recall matched pattern (n is 1 to 9)

\	Remove special meaning for character

Examples:

Match a field consisting of a negative number:

```
^-[0-9][0-9]*$
```

Match a field containing M or F (only) in either upper or lower case:

```
^[MFmf]$
```

Match a field containing an asterisk (together with possibly other characters):

```
\*
```