

www.howtogeek.com /826549/parse-csv-data-in-bash/

## How to Parse CSV Data in Bash

Dave McKay : 9/12/2022

Jane Kelly/Shutterstock.com

Comma Separated Values (CSV) files are one of the most common formats for exported data. On Linux, we can read CSV files using Bash commands. But it can get very complicated, very quickly. We'll lend a hand.

### What Is a CSV File?

A Comma Separated Values file is a [text file that holds tabulated data](#). CSV is a type of delimited data. As the name suggests, a comma "," is used to separate each field of data—or *value*—from its neighbors.



#### RELATED [What Is a CSV File, and How Do I Open It?](#)

CSV is everywhere. If an application has import and export functions, it'll almost always support CSV. CSV files are human-readable. You can look inside them with less, open them in any text editor, and move them from program to program. For example, you can export the data from an [SQLite](#) database and open it in [LibreOffice Calc](#).

However, even CSV can become complicated. Want to have a comma in a data field? That field needs to have quotation marks "" wrapped around it. To include quotation marks in a field each quotation mark needs to be entered twice.

Of course, if you are working with CSV generated by a program or [script that you have written](#), the CSV format is likely to be simple and straightforward. If you're forced to work with more complex CSV formats, with Linux being Linux, there are solutions we can use for that too.

### Some Sample Data

You can easily generate some sample CSV data, using sites like [Online Data Generator](#). You can define the fields you want and choose how many rows of data you want. Your data is generated using realistic dummy values and downloaded to your computer.

We created a file containing 50 rows of dummy employee information:

- **id**: A simple unique integer value.
- **firstname**: The first name of the person.
- **lastname**: The last name of the person.
- **job-title**: The person's job title.
- **email-address**: The person's email address.
- **branch**: The company branch they work in.
- **state**: The state the branch is located in.

Some CSV files have a header line that lists the field names. Our sample file has one. Here's the top of our file:

```
Open  [icon] sample.csv
~/.
1 id,firstname,lastname,job-title,email-address,branch,state
2 1,Rosalyn,Brennan,Steward,Rosalyn_Brennan4351@mafthy.com,Minneapolis,
3 2,Danny,Redden,Budget Analyst,Danny_Redden1443@brety.org,Venice,North
4 3,Lexi,Roscoe,Pharmacist,Lexi_Roscoe5211@supunk.biz,Otawa,Vermont
5 4,Priscilla,Flanders,Web Developer,Priscilla_Flanders7233@quentu.biz,
6 5,Alexander,Morris,Bookkeeper,Alexander_Morris2330@irrepsy.com,Murfre
7 6,Ryan,Pickard,Business Broker,Ryan_Pickard773@gmail.com,Escondido,Al
8 7,Colleen,Swan,Steward,Colleen_Swan5286@cispeto.com,Memphis,Arkansas
9 8,Wendy,Edler,Pharmacist,Wendy_Edler6067@hourpy.biz,Lincoln,Iowa
10 9,Belinda,Avery,Systems Administrator,Belinda_Avery8993@naiker.biz,Ja
11 10,Maya,Woodley,Front Desk Coordinator,Maya_Woodley8067@nickia.com,Lo
12 11,Caleb,Wood,Business Broker,Caleb_Wood8718@brety.org,Madrid,North D
13 12,Abdul,Furnell,Global Logistics Supervisor,Abdul_Furnell4889@jiman.
14 13,Caleb,Blackburn,Healthcare Specialist,Caleb_Blackburn5403@bulaffy.
```

The first line holds the field names as comma-separated values.

### Parsing Data Form the CSV file

Let's write a script that will read the CSV file and extract the fields from each record. Copy this script into an editor, and save it to a file called "field.sh."

```
#!/bin/bash
```

```
while IFS="," read -r id firstname lastname jobtitle email branch state
do
    echo "Record ID: $id"
    echo "Firstname: $firstname"
    echo " Lastname: $lastname"
    echo "Job Title: $jobtitle"
    echo "Email add: $email"
    echo " Branch: $branch"
    echo " State: $state"
    echo ""
```

```
done < <(tail -n +2 sample.csv)
```

There's quite a bit packed into our little script. Let's break it down.



**RELATED** [How to Process a File Line by Line in a Linux Bash Script](#)

We're using a while loop. As long as the while loop *condition* resolves to true, the body of the while loop will be executed. The body of the loop is quite simple. A collection of echo statements are used to print the values of some variables to the terminal window.

The while loop condition is more interesting than the body of the loop. We specify that a comma should be used as the internal field separator, with the IFS="," statement. The IFS is an environment variable. The read command refers to its value when parsing sequences of text.

We're using the read command's -r (retain backslashes) option to ignore any backslashes that may be in the data. They'll be treated as regular characters.

The text that the read command parses is stored in a set of variables named after the CSV fields. They could just as easily have been named field1, field2, ... field7, but meaningful names make life easier.

The data is obtained as the output from [the tail command](#). We're using tail because it gives us a simple way to skip over the header line of the CSV file. The -n +2 (line number) option tells tail to start reading at line number two.

The <(...) construct is called [process substitution](#). It causes Bash to accept the output of a process as though it were coming from a file descriptor. This is then redirected into the while loop, providing the text that the read command will parse.

Make the script executable using [the chmod command](#). You'll need to do this each time you copy a script from this article. Substitute the name of the appropriate script in each case.

```
chmod +x field.sh
```

```
dave@Ubuntu-22-04:~$ chmod +x field.sh
```

When we run the script, the records are correctly split into their constituent fields, with each field stored in a different variable.

```
./field.sh
```

```
dave@Ubuntu-22-04:~$ ./field.sh
Record ID: 1
Firstname: Rosalyn
Lastname: Brennan
Job Title: Steward
Email add: Rosalyn_Brennan4351@mafthy.com
Branch: Minneapolis
State: Maryland

Record ID: 2
Firstname: Danny
Lastname: Redden
Job Title: Budget Analyst
Email add: Danny_Redden1443@brety.org
Branch: Venice
State: North Carolina

Record ID: 3
Firstname: Lexi
Lastname: Roscoe
Job Title: Pharmacist
```

Each record is printed as a set of fields.

## Selecting Fields

Perhaps we don't want or need to retrieve every field. We can obtain a selection of fields by incorporating [the cut command](#).

This script is called "select.sh."

```
#!/bin/bash
```

```
while IFS="," read -r id jobtitle branch state
do
    echo "Record ID: $id"
    echo "Job Title: $jobtitle"
    echo "Branch: $branch"
    echo "State: $state"
    echo ""
done < <(cut -d "," -f1,4,6,7 sample.csv | tail -n +2)
```

We've added the cut command into the process substitution clause. We're using the -d (delimiter) option to tell cut to use commas "," as the delimiter. The -f (field) option tells cut we want fields one, four, six, and seven. Those four fields are read into four variables, which get printed in the body of the while loop.

This is what we get when we run the script.

```
./select.sh
```

```
dave@Ubuntu-22-04:~$ ./select.sh
Record ID: 1
Job Title: Steward
  Branch: Minneapolis
    State: Maryland

Record ID: 2
Job Title: Budget Analyst
  Branch: Venice
    State: North Carolina

Record ID: 3
Job Title: Pharmacist
  Branch: Irlington
    State: Vermont

Record ID: 4
Job Title: Web Developer
  Branch: Tokyo
    State: Nevada
```

By adding the cut command, we're able to select the fields we want and ignore the ones we don't.

## So Far, So Good. But...

If the CSV you deal with is uncomplicated without commas or quotation marks in field data, what we've covered will probably meet your CSV parsing needs. To show the problems we can encounter, we modified a small sample of the data to look like this.

```
id,firstname,lastname,job-title,email-address,branch,state
1,Rosalyn,Brennan,"Steward, Senior",Rosalyn_Brennan4351@mafthy.com,Minneapolis,Maryland
2,Danny,Redden,"Analyst ""Budget""",Danny_Redden1443@brety.org,Venice,North Carolina
3,Lexi,Roscoe,Pharmacist,,Irlington,Vermont
```

- Record one has a comma in the job-title field, so the field needs to be wrapped in quotation marks.
- Record two has a word wrapped in two sets of quotation marks in the jobs-title field.
- Record three has no data in the email-address field.

This data was saved as "sample2.csv." Modify your "field.sh" script to call the "sample2.csv", and save it as "field2.sh."

```
#!/bin/bash
```

```
while IFS="," read -r id firstname lastname jobtitle email branch state
do
    echo "Record ID: $id"
    echo "Firstname: $firstname"
    echo "  Lastname: $lastname"
    echo "Job Title: $jobtitle"
    echo "Email add: $email"
    echo "  Branch: $branch"
    echo "  State: $state"
    echo ""
done < <(tail -n +2 sample2.csv)
```

When we run this script, we can see cracks appearing in our simple CSV parsers.

```
./field2.sh
```

```
dave@Ubuntu-22-04:~$ ./field2.sh
```

The first record splits the job-title field into two fields, treating the second part as the email address. Every field after this is shifted one place to the right. The last field contains both the branch and the state values.

```
dave@Ubuntu-22-04:~$ ./field2.sh
Record ID: 1
Firstname: Rosalyn
Lastname: Brennan
Job Title: "Steward
Email add: Senior"
  Branch: Rosalyn_Brennan4351@mafthy.com
    State: Minneapolis,Maryland
```

The second record retains all quotation marks. It should only have a single pair of quotation marks around the word "Budget."

```
Record ID: 2
Firstname: Danny
Lastname: Redden
Job Title: "Analyst ""Budget"""
Email add: Danny_Redden1443@brety.org
  Branch: Venice
    State: North Carolina
```

The third record actually handles the missing field as it should. The email address is missing, but everything else is as it should be.

```
Record ID: 3
Firstname: Lexi
  Lastname: Roscoe
Job Title: Pharmacist
Email add:
  Branch: Irlington
  State: Vermont
```

Counterintuitively, for a simple data format, it is very difficult to write a robust general-case CSV parser. Tools like `awk` will let you get close, but there are always edge cases and exceptions that slip through.



**RELATED** [How to Use the `awk` Command on Linux](#)

Trying to write an infallible CSV parser is probably not the best way forward. An alternative approach—especially if you're working to a deadline of some sort—employs two different strategies.

One is to use a purpose-designed tool to manipulate and extract your data. The second is to sanitize your data and replace problem scenarios such as embedded commas and quotation marks. Your simple Bash parsers can then cope with the Bash-friendly CSV.

## The csvkit Toolkit

The CSV toolkit `csvkit` is a collection of utilities expressly created to help work with CSV files. You'll need to install it on your computer.

To install it on Ubuntu, use this command:

```
sudo apt install csvkit
```

```
dave@Ubuntu-22-04:~$ sudo apt install csvkit
```

To install it on Fedora, you need to type:

```
sudo dnf install python3-csvkit
```

```
[dave@fedora-36 ~]$ sudo dnf install python3-csvkit
```

On Manjaro the command is:

```
sudo pacman -S csvkit
```

```
sudo pacman -S csvkit
```

If we pass the name of a CSV file to it, the `csvlook` utility displays a table showing the contents of each field. The field content is displayed to show what the field contents represent, not as they're stored in the CSV file.

Let's try `csvlook` with our problematic "sample2.csv" file.

```
csvlook sample2.csv
```

```
dave@Ubuntu-22-04:~$ csvlook sample2.csv
| id | firstname | lastname | job-title | email-address |
| -- | -|-----| -|-----| -|-----|
| 1 | Rosalyn | Brennan | Steward, Senior | Rosalyn_Brennan4351@ma |
| 2 | Danny | Redden | Analyst "Budget" | Danny_Redden1443@brety |
| 3 | Lexi | Roscoe | Pharmacist | |
dave@Ubuntu-22-04:~$
```

All of the fields are correctly displayed. This proves the problem isn't the CSV. The problem is our scripts are too simplistic to interpret the CSV correctly.

To select specific columns, use the `csvcut` command. The `-c` (column) option can be used with field names or column numbers, or a mix of both.

Suppose we need to extract the first and last names, job titles, and email addresses from each record, but we want to have the name order as "last name, first name." All we need to do is put the field names or numbers in the order we want them.

These three commands are all equivalent.

```
csvcut -c lastname,firstname,job-title,email-address sample2.csv
```

```
csvcut -c lastname,firstname,4,5 sample2.csv
```

```
csvcut -c 3,2,4,5 sample2.csv
```

```
dave@Ubuntu-22-04:~$ csvcut -c 3,2,4,5 sample2.csv
lastname,firstname,job-title,email-address
Brennan,Rosalyn,"Steward, Senior",Rosalyn_Brennan4351@mafthy.com
Redden,Danny,"Analyst ""Budget""",Danny_Redden1443@breyt.org
Roscoe,Lexi,Pharmacist,
dave@Ubuntu-22-04:~$
```

We can add the `csvsort` command to sort the output by a field. We're using the `-c` (column) option to specify the column to sort by, and the `-r` (reverse) option to sort in descending order.

```
csvcut -c 3,2,4,5 sample2.csv | csvsort -c 1 -r
```

```
dave@Ubuntu-22-04:~$ csvcut -c 3,2,4,5 sample2.csv | csvsort -c 1 -r
lastname,firstname,job-title,email-address
Roscoe,Lexi,Pharmacist,
Redden,Danny,"Analyst ""Budget""",Danny_Redden1443@breyt.org
Brennan,Rosalyn,"Steward, Senior",Rosalyn_Brennan4351@mafthy.com
dave@Ubuntu-22-04:~$
```

To make the output prettier we can feed it through `csvlook`.

```
csvcut -c 3,2,4,5 sample2.csv | csvsort -c 1 -r | csvlook
```

```
dave@Ubuntu-22-04:~$ csvcut -c 3,2,4,5 sample2.csv | csvsort -c 1 -r | csvlook
| lastname | firstname | job-title | email-address |
|-----|-----|-----|-----|
| Roscoe | Lexi | Pharmacist | |
| Redden | Danny | Analyst "Budget" | Danny_Redden1443@breyt.org |
| Brennan | Rosalyn | Steward, Senior | Rosalyn_Brennan4351@mafthy.com |
dave@Ubuntu-22-04:~$
```

A neat touch is that, even though the records are sorted, the header line with the field names is kept as the first line. Once we're happy we have the data the way we want it we can remove the `csvlook` from the command chain, and create a new CSV file by redirecting the output into a file.

We added more data to the "sample2.file", removed the `csvsort` command, and created a new file called "sample3.csv."

```
csvcut -c 3,2,4,5 sample2.csv > sample3.csv
```

```
dave@Ubuntu-22-04:~$ csvcut -c 3,2,4,5 sample2.csv > sample3.csv
```

## A Safe Way to Sanitize CSV Data

If you open a CSV file in LibreOffice Calc, each field will be placed in a cell. You can use the find and replace function to search for commas. You could replace them with "nothing" so that they vanish, or with a character that won't affect the CSV parsing, like a semi-colon ";", for example.

You won't see the quotation marks around quoted fields. The only quotation marks you'll see are the embedded quotation marks *inside* field data. These are shown as single quotation marks. Finding and replacing these with a single apostrophe "'" will replace the double quotation marks in the CSV file.

The screenshot shows the 'Find and Replace' dialog box. The 'Find' field contains a double quote character ("). The 'Replace' field contains a single quote character ('). The 'Replace All' button is highlighted with a red box. Below the dialog box, there are checkboxes for 'Other options': 'Current selection only', 'Wildcards', 'Replace backwards', and 'Cell Styles'.

Doing the find and replace in an application like LibreOffice Calc means you can't accidentally delete any of the field separator commas, nor delete the quotation marks around quoted fields. You'll only change the *data values* of fields.

We changed all commas in fields with semicolons and all embedded quotation marks with apostrophes and saved our changes.

	A	B	C	D
1	lastname	firstname	job-title	email-address
2	Brennan	Rosalyn	Steward; Senior	Rosalyn_Brennan4351@mafthy.com
3	Redden	Danny	Analyst 'Budget'	Danny_Redden1443@breyt.org
4	Roscoe	Lexi	Pharmacist	
5	Flanders	Priscilla	Web Developer	Priscilla_Flanders7233@guentu.biz
6	Morris	Alexander	Bookkeeper	Alexander_Morris2330@irrepsy.com
7	Pickard	Ryan	Business Broker	Ryan_Pickard773@gmail.com
8	Swan	Colleen	Steward	Colleen_Swan5286@cispeto.com
9	Edler	Wendy	Pharmacist	Wendy_Edler6067@hourpy.biz

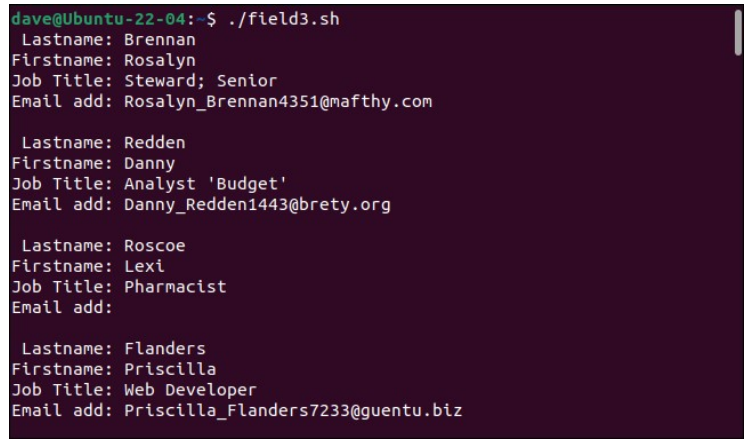
We then created a script called "field3.sh" to parse "sample3.csv."

```
#!/bin/bash
```

```
while IFS="," read -r lastname firstname jobtitle email
do
    echo " Lastname: $lastname"
    echo "Firstname: $firstname"
    echo "Job Title: $jobtitle"
    echo "Email add: $email"
    echo ""
done < <(tail -n +2 sample3.csv)
```

Let's see what we get when we run it.

```
./field3.sh
```



```
dave@Ubuntu-22-04:~$ ./field3.sh
  Lastname: Brennan
  Firstname: Rosalyn
  Job Title: Steward; Senior
  Email add: Rosalyn_Brennan4351@mafthy.com

  Lastname: Redden
  Firstname: Danny
  Job Title: Analyst 'Budget'
  Email add: Danny_Redden1443@brety.org

  Lastname: Roscoe
  Firstname: Lexi
  Job Title: Pharmacist
  Email add:

  Lastname: Flanders
  Firstname: Priscilla
  Job Title: Web Developer
  Email add: Priscilla_Flanders7233@guentu.biz
```

Our simple parser can now handle our previously problematic records.

## You'll See a Lot of CSV

CSV is arguably the closest thing to a common tongue for application data. Most applications that handle some form of data support importing and exporting CSV. Knowing how to handle CSV—in a realistic and practical way—will stand you in good stead.

**RELATED:** [9 Bash Script Examples to Get You Started on Linux](#)