Command Line Cheat Sheet for Data Wrangling

21 August, 2025

This document lists a series of examples of managing data on the command line. All commands are run as shown on macOS 26 (Tahoe), but should work just as well on Linux.

These notes are from a presentation at the Computational Humanities (CMP 25) programme on the 20th of August 2025. They are deliberately terse and intented as an inspiration of how to get started with using the command line for exploring and validating files.

Get started and create some synthetic test data

create a folder to use as sandbox. Folders in a UNIX system are typically called directories.

```
exploring-files $ mkdir exploring-files
```

change focus to the new folder

```
exploring-files $ cd exploring-files
```

Maybe open the folder in the Finder. This is a macOS specific command, but Windows have a corresponding command that can be found on the WWW.

```
exploring-files $ open .
```

List the files in this folder. This `ls` command has a `-l` after its name. We call this an option. All the commands shown in this cheat sheet has very many options, which can all be explored by using the manual with e.g., this command: `man ls` for `ls`.

Let's list all files in this folder

```
exploring-files $ ls -l

total 0
```

off course, we have no files, as we just created this folder `;-)`.

Now, adjust the prompt. The prompt is the letters and numbers appearing in front of your cursor, waiting for your input. I don't know, how the prompt looks at your specific system, but you can adjust is like this

```
exploring-files $ export PS1=" %1d :> "
```

Go up one level in the folder hierarchy. `cd` means change directory and `..` means up.

```
exploring-files :> cd ..
```

Go back to where we came from. `-` means: the previous folder. The command line is built for effective use, and has a lot of shortcuts.

```
exploring-files :> cd -
```

I've created a prompt to make a Large Language Model (LLM) produce some synthetic data. Which LLM you use, does not matter. In this cheat sheet, I'll use a local model managed by the LM Studio application. I use the very small model called `qwen3-4b-2507`, that seems sufficient for the purpose of this cheat sheet.

```
exploring-files :> cat ../create-data.prompt

create a pseudo data set in CSV format 100 rows with the following five columns: <id> <name> <age> <gender> <occupation> <voting party>, where

<id>: a running unique id
<name>: a made up danish sounding <given name> <surname>
<age>: a integer taken from a distribution normal distribution
<gender>: M (for male), F (for female) or "-" for undefined
```

```
<birth country>: This is simulation a Danish survey, and this variable represents people born in countries other than Denmark. Use counties from al
<education>: type of education, e.g., elementary school, apprenticeship, college, university
<occupation>: an occupation related to education with outliers. E.g., have some college people working as unskilled work
<religion>: which religion does the person adhere to. Include all world religions and also a few "ateist" and "agnostic"
<voting party>: <far right>, <right>, <center>, <left>, <far left> with a normal distribution

output ONLY the data and no preliminary or ending notes!
```

LM Studio provides a command line integration. The `|` character is called a pipe and sends the output of the left command into the right command. I.e., the output of `cat` is send into the `lms`. The `>` character sends the output of the left command into a file named using the right word. So, this command line outputs the file `create-data.prompt` from the upper folder into the LLM and takes the output of the LLM and stores it in a file called `survey.csv`.

```
exploring-files :> cat ../create-data.prompt | lms chat > survey.csv
```

Explore the file

Let's see what we got

```
exploring-files :> ls -lh

total 16
-rw-r--r--  1 au15929  staff   7.0K Aug 21 11:39 survey.csv
```

How much data did we get?

```
exploring-files :> wc survey.csv

101     277    7128 survey.csv
```

Just count the number of lines (`-l`)

```
exploring-files :> wc -l survey.csv

101 survey.csv
```

List the first five lines

```
exploring-files :> head -5 survey.csv

id,name,age,gender,birth_country,education,occupation,religion,voting_party
1,Line Hansen,34,M,Denmark,university,teacher,Christianity,center
2,Anette Møller,29,F,Norway,apprenticeship,barista,Protestant,right
3,Anders Jørgensen,41,M,Sweden,college,librarian,agnostic,center
4,Ingrid Larsen,38,F,Denmark,elementary school,caretaker,Christianity,far left
```

List the last five lines

```
exploring-files :> tail -5 survey.csv

96,Thomas Søndergaard,41,M,Denmark,university,academic,Protestant,center
97,Line Nygaard,39,F,India,high school,barista,Protestant,center
98,Jonas Højgaard,53,M,Denmark,college,teacher,Protestant,center
99,Amalie Fenger,23,F,China,elementary school,weaver,Protestant,right
100,Preben Fjord,46,M,Denmark,university,professor,Protestant,center
```

What kind of file did we get? The `file` command is used to identify file types and can also detect the character encoding used for a given file.

```
exploring-files :> file --mime-type --mime-encoding survey.csv

survey.csv: text/csv; charset=utf-8
```

The file extension is of no significance to these commands:

```
exploring-files :> cp survey.csv unknown-file
exploring-files :> file --mime-type --mime-encoding unknown-file

unknown-file: text/csv; charset=utf-8
```

Let's extract values from the second column

```
exploring-files :> cat survey.csv | cut -f 2 | head -5

id,name,age,gender,birth_country,education,occupation,religion,voting_party
1,Line Hansen,34,M,Denmark,university,teacher,Christianity,center
2,Anette Møller,29,F,Norway,apprenticeship,barista,Protestant,right
3,Anders Jørgensen,41,M,Sweden,college,librarian,agnostic,center
4,Ingrid Larsen,38,F,Denmark,elementary school,caretaker,Christianity,far left
```

That did not work! It gave us all fields. That's because `cut` by default operates in data, where columns are seperated by the tabulator character. This behaviour can be changed by the `-d` option like:

```
exploring-files :> cat survey.csv | cut -d, -f 2 | head -5

name
Line Hansen
Anette Møller
Anders Jørgensen
Ingrid Larsen
```

We can now easily sort the data. Alas, we have a lot of name reuse, which is an artefact of using a very small (but fast) LLM.

```
exploring-files :> cat survey.csv | cut -d, -f 2|sort | head -10

Amalie Andersen
Amalie Fenger
Amalie Fenger
Amalie Fenger
Amalie Fenger
Amalie Højgaard
Amalie Højgaard
Amalie Krog
Amalie Søndergaard
Amalie Søndergaard
```

Let's look at unique names

```
exploring-files :> cat survey.csv | cut -d, -f 2|sort|uniq | tail -5

Thomas Kjær
Thomas Madsen
Thomas Nygaard
Thomas Søndergaard
Thomas Sørensen
```

Let's count the number of different names

```
exploring-files :> cat survey.csv | cut -d, -f 2|sort | uniq | wc -l

75
```

How many do we have with the name 'Madsen'?

```
exploring-files :> grep Madsen survey.csv

11,Stig Madsen,49,M,Canada,apprenticeship,plumber,Christianity,right
19,Thomas Madsen,42,M,Denmark,elementary school,unskilled laborer,agnostic,right
35,Thomas Madsen,56,M,Denmark,apprenticeship,plumber,agnostic,right
62,Elisabeth Madsen,40,F,Indonesia,high school,weaver,Christianity,center
79,Anders Madsen,36,M,Denmark,high school,unskilled laborer,Protestant,center
93,Elisabeth Madsen,37,F,Chile,high school,weaver,Protestant,center
```

Count them

```
exploring-files :> grep Madsen survey.csv

6
```

Now, let's look at the distribution of occupations based on the F and M words.

```
exploring-files :> grep ',M,' survey.csv | head -5
```

First make two new files, one for the unique occupations for the M category and one for the F category.

```
exploring-files :> grep ',M,' survey.csv | cut -d, -f7|sort|uniq>m
```

```
exploring-files :> grep ',F,' survey.csv | cut -d, -f7|sort|uniq>f
```

Then compare these two lists. `<` means the value is only in the `f` file and `>` in the `m` file, and yes, the small Qwen LLM do have an occupation bias based only on the letters F and M!

```
exploring-files :> diff m f

1,6c1,7
< barista
< barmaid
< caretaker
< doctor
< domestic worker
< nurse
---
> academic
> agriculturist
> carpenter
> construction worker
> engineer
> librarian
> plumber
8,9d8
< street vendor
< student
11c10,11
< weaver
---
> university lecturer
> unskilled laborer
```

Other data formats

As an experiment, let us try to use the LLM for transforming the data from CSV.

```
exploring-files :> head -5 survey.csv|lms chat -p "convert to json" > survey.json
exploring-files :> head -5 survey.csv|lms chat -p "convert to xml" > survey.xml
```

Did it produce any files?

```
exploring-files :> ls -hl

total 64
-rw-r--r--  1 au15929  staff   102B Aug 21 11:56 f
-rw-r--r--  1 au15929  staff   136B Aug 21 11:56 m
-rw-r--r--  1 au15929  staff   7.0K Aug 21 11:39 survey.csv
-rw-r--r--  1 au15929  staff   1.0K Aug 21 12:06 survey.json
-rw-r--r--  1 au15929  staff   1.3K Aug 21 12:06 survey.xml
-rw-r--r--  1 au15929  staff   7.0K Aug 21 11:45 unknown-file
```

Yes, we got a new JSON and a new XML file, and they are identified correctly

```
exploring-files :> file --mime-type survey.*

survey.csv:  text/csv; charset=utf-8
survey.json: application/json; charset=utf-8
survey.xml:  text/xml; charset=utf-8
```

What about an Excel sheet?

~~~
exploring-files :> head -5 survey.csv|lms chat -p "convert to xlsx"

I can't directly create or send files like `.xlsx` in this chat interface. However, I can provide you with **Python code** that will convert your d

### ✅ Here's the Python script to convert your data to an XLSX file:

```python
import pandas as pd
...
~~~

No, it did not work. Instead, we got a Python program for this transformation. I'll leave it up to the reader to try if it works...

Encodings

I've created a version of the `survey.csv` file in the Latin 1 encoding. The `sed` command is a so-called stream editor, i.e., it takes some input, edits this using a very terse edit language and outputs the results. we will see more to this command later, but here, I just use it to only print line 4 to 7

```
exploring-files :> sed -n '4,7p' survey-latin1

3,Sofia Nielsen,28,F,India,University,Doctor,Muslim,right
4,Oscar Larsen,52,M,Sweden,Apprenticeship,Construction Worker,Christian,far right
5,Astrid M¯ller,31,F,Italy,University,Engineer,Christian,left
6,Mikkel Hansen,41,M,Denmark,Elementary School,Unskilled Work,Atheist,far left
```

Line five has this name Astrid M¯ller, where it should have been Astrid Møller, but as the file has an encoding different from that of the terminal, it prints wrong.

We can get `file` to detect the actual encoding of the file

```
exploring-files :> file --mime-encoding survey-latin1

survey-latin1: iso-8859-1
```

If we want to use this file, the best thing would be to transform it into the more modern encoding called UTF-8. To that end, we have the `iconv` command. As of today, it can handle 873 different encodings!

```
exploring-files :> iconv -l | wc -w
873
```

Now, transform from Latin 1 to UTF-8

```
exploring-files :> iconv -f latin1 -t utf8 survey-latin1 > survey-utf8
exploring-files :> file --mime-encoding survey-utf8

survey-utf8: utf-8
```

Now, what about a file with a mix of encodings?

```
exploring-files :> file --mime-encoding survey-bad-encoding-latin1

survey-bad-encoding-latin1: unknown-8bit
```

In such cases, only manual inspection can help you. When you have identified the problems, it is very much possible to make commands and put the into scripts to automate the corrections of many files.

Structural errors

What if we have a CSV file with structural errors? Look closely at line numbered 2 in this file

```
exploring-files :> head -5 survey-field-error

id,name,age,gender,birth country,education,occupation,religion,voting party
1,Emma Hansen,34,F,Germany,University,Software Engineer,Christian,left
2,Lars Jensen,45,Denmark,College,Teacher,Buddhist
3,Sofia Nielsen,28,F,India,University,Doctor,Muslim,right
4,Oscar Larsen,52,M,Sweden,Apprenticeship,Construction Worker,Christian,far right
```

Lars Jensen have no F or M letter and therefore one less field than the rest. We can still extract the 9th field of all rows with `cut` without running into system errors. We will just get an empty value, but we cannot deduce which field is missing.

```
exploring-files :> cut -d, -f9 survey-field-error|head -5

voting party
left

right
far right
```

If we then try to ingest the data file into a system, that transforms the CSV data into an internal representation, it will hopefully fail. Such systems could be Python Pandas, R Tidyverse, SPSS, or, like in this example, the Wolfram Language

```
wl 'Import["survey-field-error","CSV"]//ToTabular'

...cannot be converted to a Tabular object since is it not a list of rows that are all associations or lists of the same length.
```

Broken data

Okay, so here is another issue, that I have encountered in the wild. All characters in a file has a number representation. Simply said, a could be 1, b could b 2, and so on. This number is then represented as a binary number and can therefore be stored in computer memory or on a storage system. Besides the characters that we are able to see, we also need codes for e.g., space, newline, tabulator and so on. One of these unseeable codes are 0. So, what happens, if we have the 0 code within a string of characters?

For illustration, I have created such a file, where I inserted the 0 (or null) character after the E in Emma. If we then look at the file using `head`, it looks fine.

```
exploring-files :> head -5 survey-null-char.csv

id,name,age,gender,birth country,education,occupation,religion,voting party
1,Emma Hansen,34,F,Germany,University,Software Engineer,Christian,left
2,Lars Jensen,45,M,Denmark,College,Teacher,Buddhist,center
3,Sofia Nielsen,28,F,India,University,Doctor,Muslim,right
4,Oscar Larsen,52,M,Sweden,Apprenticeship,Construction Worker,Christian,far right
```

One could also open it in a spreadsheet program, and might not see anything wrong.

But then, if we try to count the length of the names in the second column, we see something different

```
exploring-files :> awk -F, '{print $2, length($2)}' survey-null-char.csv | head -5

name 4
E 1
Lars Jensen 11
Sofia Nielsen 13
Oscar Larsen 12
```

Emma's name is now just an E! That is because in a lot of systems, the null character is the signal for end-of-string. To `awk` the string of characters representing Emma's name ends after the initial E. So, invisible stuff in your data, can have serious consequences.

If we really want to see how the file looks, we can use the `hexdump` command, which displays the codes for all characters in a file. In the right column, We can see a dot after the E in Emma. If we then find the same position in the middle column, we read `00` and have identified a problem.

```
exploring-files :> hexdump -C survey-null-char.csv|sed -n '1,8p'

00000000  69 64 2c 6e 61 6d 65 2c  61 67 65 2c 67 65 6e 64  |id,name,age,gend|
00000010  65 72 2c 62 69 72 74 68  20 63 6f 75 6e 74 72 79  |er,birth country|
00000020  2c 65 64 75 63 61 74 69  6f 6e 2c 6f 63 63 75 70  |,education,occup|
00000030  61 74 69 6f 6e 2c 72 65  6c 69 67 69 6f 6e 2c 76  |ation,religion,v|
00000040  6f 74 69 6e 67 20 70 61  72 74 79 0a 31 2c 45 00  |oting party.1,E.|
00000050  6d 6d 61 20 48 61 6e 73  65 6e 2c 33 34 2c 46 2c  |mma Hansen,34,F,|
00000060  47 65 72 6d 61 6e 79 2c  55 6e 69 76 65 72 73 69  |Germany,Universi|
00000070  74 79 2c 53 6f 66 74 77  61 72 65 20 45 6e 67 69  |ty,Software Engi|
```

Converting from English numbers to Danish numbers

I have used `lms` to create a small synthetic data set of some animals and their weights

```
exploring-files :> cat animals

id,animal,weight
1,"Lion",190.50
2,"Elephant",5400.25
3,"Giraffe",1350.75
4,"Monkey",6.25
5,"Whale",12000.00
6,"Bear",320.10
7,"Zebra",180.45
8,"Wolf",38.90
9,"Rabbit",2.30
10,"Hippo",1400.00
```

Now, the numbers are using the English decimal character, the dot, but let us assume, that we need to use a program made for the Danish decimal character, the comma. OK, we cannot just convert all dots to commas, as the comma is the field separator in the data file. Alas, we need first to convert all the commas to semicolons, and then all the dots to commas. Let us do that using the `sed` command mentioned above, but this time use it for actual editing

```
exploring-files :> sed 's/,/;/g;s/\./,/g' animals

id;animal;weight
1;"Lion";190,50
2;"Elephant";5400,25
3;"Giraffe";1350,75
4;"Monkey";6,25
5;"Whale";12000,00
6;"Bear";320,10
7;"Zebra";180,45
```

```
  8;"Wolf";38,90
  9;"Rabbit";2,30
  10;"Hippo";1400,00
```

Other relevant commands

`man, history, ssh, python, Rscript, alias`