Using command line to combine EPC data

We have downloaded data from the EPC website at https://epc.opendatacommunities.org/login - this consists of 340 folders each containing two CSV files.

The first attempt - reusing old code

We start by adapting some command line we'd used to combine crime data CSVs, by changing the name of the CSV file. The bash file created first had this code:

```
#!/bin/bash

echo "looking for street EPC CSV files..."
find $HOME -name "certificates.csv"
echo "making a new folder on the Desktop..."
mkdir -p $HOME/Desktop/allcombined
echo "moving all certificates.csv files to new folder..."
cp `find $HOME -name "certificates.csv"`
$HOME/Desktop/allcombined
echo "moving to new directory..."
cd $HOME/Desktop/allcombined
echo "combining all the files into one"
cat certificates.csv > certificatescombined.csv
```

However, because all the CSV files had the same name, this code resulted in a single certificates.csv file (essentially each time it looped it overwrote the previous file)

Finding some code to rename files

We need to find a way to rename the files as we move them.

Googling around we found this code:

```
num=0; for i in *; do mv "$i" "$(printf '%04d' $num).${i#*.}";
((num++)); done
```

This code is tested on a small subset of the data: when run it renames all files or folders in the current directory.

Adapting the command

So we need to combine this with the previous code that moved files. We settle on this:

```
num=0; for i in `find all-domestic-certificates -name
"certificates.csv"`; do cp "$i" "$(printf '%04d' $num).${i#*.}";
((num++)); done
```

Breaking down the command line

What have we changed? Let's break it down.

- We set a variable at 0: `num=0;`
- We begin a loop, calling each item in that loop 'i': `for i in`
- We loop through the results of a 'find' command. Specifically, that command finds all files in the directory 'all-domestic-certificates'* with the name certificates.csv: `` `find all-domestic-certificates -name "certificates.csv"`; ``
- Now, what to do with each of the results? We copy that file (at this point i is a variable, and the dollar sign indicates that it is referring to a variable name): `do cp "$i"`
- And we change the name to the current value of that variable 'num' (again, indicated as a variable by prefixing with a dollar sign). The 'printf' part formats it, and %04d means to four digits, which is why the resulting files have names like 0001 and 0344: `"$(printf '%04d' $num)`
- We also add the file extension of the original filename: `.${i#*.}";`
- Then we increase the value of the variable 'num' by one so it's different for the next loop: `((num++)); done`

*Note: in Terminal/command line you must navigate (using `cd`) to the folder containing the 'all-domestic-certificates' folder first before running this.

We can now put this into a bash file.

A bash file for combining EPC certificates

We can adapt this code, along with some of the code from the previous bash file. Some things that we change:

- The bash file will sit in the folder where the 'all-domestic-certificates' folder is going to be (and where it will be unzipped in future). So we need to change directory into that folder first.
- This is because in the last line it combines all the CSV files, so we don't want any other CSV files in the directory where we run the code.

- We therefore need to look not in ‘all-domestic-certificates’ for ‘certificates.csv’ but in any folder that begins ‘domestic’

```
#!/bin/bash

echo "changing into the all-domestic-certificates folder"
cd all-domestic-certificates
echo "looking for EPC CSV files in folders beginning ‘domestic’"
find domestic* -name "certificates.csv"
echo "copying, moving and renaming those files into the current
directory"
num=0; for i in `find domestic* -name "certificates.csv"`; do cp
"$i" "$(printf '%03d' $num).${i#*.}"; ((num++)); done
echo "combining all the files into one"
cat *.csv > certificatescombined.csv
```

The result has 24m rows and 92 columns.

We can adapt that for the recommendations easily:

```
num=0; for i in `find domestic* -name "recommendations.csv"`; do
cp "$i" "$(printf '%04d' $num).${i#*.}"; ((num++)); done
cat *.csv > recscombined.csv
```

Using sed to reduce the resulting file: rows

The resulting file is over 20GB. Can we use command line to reduce that? Yes.

First we make a copy of the file so we don’t change it. We call that ‘allcerts.csv’.

Now we try the following command, which aims to remove any lines with ‘2010’ in them. This would affect addresses with 2010 in, as well as reference codes with 2010 in, so we expand that to ‘, 2010-’ because the 2010 we watch to target always follows a comma and precedes a dash.

```
sed -n '/, 2010-/!p' allcerts.csv > no2010.csv
```

This works. We can also use it to check how many are in a particular year by removing the exclamation mark:

```
sed -n '/, 2010-/p' allcerts.csv > 2010only.csv
```

Can we try multiple matches?

```
sed -n '/, 2010-|, 2009-/!p' allcerts.csv > no2010.csv
```

No - that results in no rows being removed. It might be that we've used a pipe when a slash is needed.

```
sed -n '/, 2010-/!p;/, 2009-/!p' allcerts.csv > no2010.csv
```

That results in a file with twice as many rows, not less. That may be because we're using !p to keep everything *but* those. Let's try the delete command instead.

```
sed -n '/, 2010-/d;/, 2009-/d' allcerts.csv > after2010.csv
```

That results in an empty file.

Let's just chain them together:

```
sed -n '/, 2010-/!p' allcerts.csv > after2010.csv
sed -n '/, 2009-/!p' after2010.csv > after2010.csv
sed -n '/, 2008-/!p' after2010.csv > after2010.csv
sed -n '/, 2007-/!p' after2010.csv > after2010.csv
sed -n '/, 2006-/!p' after2010.csv > after2010.csv
```

Using cut to reduce the columns

According to this thread, we can use cut to remove columns as long as there's no other commas in the file.

```
cut --complement -f 3 -d, allcerts.csv > allbutcol3.csv
```

We try this to just grab column 2 and see how it looks:

```
cut -f 2 -d, allcerts.csv > col2.csv
```

Unfortunately, trying this on the address column results in addresses like 80, House End Terrace being reduced to '80'

Counting matches to check BigQuery import

We know we've lost some rows in the import to BigQuery, but from which years? We can count how many contain a particular year in the original CSV using grep:

```
grep 2022 allcerts.csv > 2022matches.csv
```

We can count by using -c

```
grep -c 2022 allcerts.csv
```

Or wc:

```
wc 2022matches.csv
```

If we want to be more specific and include spaces we'll need quotation marks:

```
grep ", 2022-" allcerts.csv > 2022matches.csv
```