## What happens under the hood?

We can run `dvc add` in verbose mode to get a little more insight into how DVC does its data versioning:

```
dvc add data/data.xml -v
```

Output:

```
...
Computed stage: 'data/data.xml.dvc' md5: 'None'...
Saving 'data/data.xml' to
'.dvc/cache/a3/04afb96060aad9017668345e10355'...
Removing '.../dvc/course/dvc-2-data-versioning/data/data.xml'...
Created 'reflink':.dvc/cache/a3/04afb96060aad90176268345e10355 ->
data/data.xml
Saving information to 'data/data.xml.dvc'
...
To track the changes with git, run:

    git add data/data.xml/dvc
```

DVC first checks whether it already created a metafile for the file we added. Here that is not the case, as shown by `md5: 'None'`. DVC then saves the data to the DVC cache, before removing the original file. A reflink is then created to the newly cached file so that your filesystem can find the cached file. Lastly, all relevant information is saved to the DVC metafile.

```
outs:
- md5: a304afb96060aad9017668345e10355
  path: data.xml
```

The metafile contains the MD5 hash for the file and the original file location. In this case, whenever DVC is instructed to retrieve `data.xml` in the `data` directory, it will retrieve the file with the hash `a304afb96060aad9017668345e10355`.

Because the metafile is versioned by our Git history, DVC has tied data versions to specific code versions. A change to our dataset would constitute an updated DVC metafile and thus a new commit in our Git history.

This process works the same when adding directories rather than singular files (`dvc add datadir`).