Day 00 – Piscine Python for Data Science

Linux Command Line Tools

Summary: The first day will help you to acquire the skills of using Linux command-line tools for basic data science tasks. You will learn how to use curl, sort, uniq, jq, sed, cat for data collection and preprocessing.

Contents

Chapter I

Preamble

We as humanity have known that data helps us to make better decisions for a long time. In Ancient Egypt, the government would conduct censuses to know better how many taxes they could get from the population. Even earlier shepherds would count livestock to know better how many animals they could sell and how many they needed for goods production.

Since then we have been developing more and more sophisticated algorithms for data processing. Now we are able to replace something that we do not know by something predicted by machine learning algorithms. It helps us to be prepared for something in the future: to predict demand for our goods and to adjust our facilities accordingly. We can predict whether a person will return their credit or not to save our money for others and get more profits.

We have been developing not only algorithms but the technologies and tools that made data analysis cheaper and more convenient. They democratized the whole data field. Nowadays it is much easier for a company to start using data for its own good. That is why we have that hype around big data, artificial intelligence, and other buzzwords.

Everybody can use data. Everybody can get value from it. Not only those who have a lot of money and resources as before.

As it was said in Mr. Robot TV series, “it's an exciting time in the world right now”.

Chapter II

Instructions

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Here and further we use `Python 3` as the only correct version of Python.
- The python files for python exercises (d01, d02, d03) must have a block in the end: `if __name__ == '__main__'`.
- Pay attention to the permissions of your files and directories.
- To be assessed your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google.
- Remember to discuss on the Intra Piscine forum.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- And may the Force be with you!

Chapter III

Exercise 00

Exercise 00 : First shell script
Directory to store your solution : `ex00/`
Files to be in the directory : `hh.sh`, `hh.json`
Authorized functions : `curl`
Comments : n/a

In this exercise, you will need to interact with HeadHunter API to parse some information about vacancies. In order to do this, you will need to understand how both `curl` and HeadHunter API work.

Write a shell script that:

1. gets as an argument name of a vacancy (e.g. 'data scientist'),
2. downloads information about the first 20 vacancies corresponding to the search parameters,
3. stores it in a file with the name `hh.json`.

The result in the file must be formatted in the way that each field is placed on a different line. An example is below:


```
{
   "page" : 0,
   "found" : 344,
   "clusters" : null,
   "arguments" : null,
   "per_page" : 20,
   "pages" : 18,
   "items" : [
      {
         "apply_alternate_url" : "https://hh.ru/applicant/vacancy_response?vacancyId=35895583",
         "address" : {
            "id" : "118668",
            "lat" : 55.762556,
            "metro" : {
               "station_id" : "7.67",
               "line_name" : "Таганско-Краснопресненская",
               "lng" : 37.624423,
               "line_id" : "7",
               "station_name" : "Кузнецкий мост",
               "lat" : 55.761498
```

```
    }
    "street" : "Кузнецкий мост",
    "lng" : 37.627175,
    "metro_stations" : [
       {
          "line_name" : "Таганско-Краснопресненская",
          "station_id" : "7.67",
          "lng" : 37.624423,
          "line_id" : "7",
          "lat" : 55.761498,
          "station_name" : "Кузнецкий мост"
       }
    ],
    "building" : "21/5",
    "city" : "Москва",
    "description" : null,
    "raw" : "Москва, Кузнецкий мост, 21/5"
}
```


Your script must be executable. The interpreter to use is `/bin/sh`.

Put your script in the `ex00` folder in the root of your repository as long as your result of parsing.

Chapter IV

Exercise 01

Exercise 01 : Transforming JSON to CSV
Directory to store your solution : `ex01/`
Files to be in the directory : `filter.jq`, `hh.csv`
Authorized functions : `jq`
Comments : n/a

What you got in the previous exercise was a JSON file. It is a popular file format for API but can be hard for data analysis itself. So here you will need to convert it into a more convenient CSV file.

Write a shell script `json_to_csv.sh` that:

1. executes `jq` with a filter written in a separate file `filter.jq`,
2. filters the following 5 columns corresponding to the vacancies: “`id`”, “`created_at`”, “`name`”, “`has_test`”, “`alternate_url`”,
3. saves the result to the CSV file `hh.csv`.

You can see the example below:

```
“id”,”created_at”,”name”,”has_test”,”alternate_url”
"35895583","2020-04-12T12:06:33+0300","Специалист / data scientist (big data, прогностическая аналитика,
data mining)",false,"https://hh.ru/vacancy/35895583"
"36359628","2020-04-11T19:25:48+0300","Senior Data Scientist",false,"https://hh.ru/vacancy/36359628"
"35218725","2020-04-11T18:03:53+0300","Junior Data scientist",false,"https://hh.ru/vacancy/35218725"
```

The CSV file must have headers in the first row.

Your script must be executable. The interpreter to use is `/bin/sh`.

Put your filter file in the `ex01` folder in the root of your repository as long as your result of the conversion.

Chapter V

Exercise 02

Exercise 02 : Sorting a file
Directory to store your solution : `ex02/`
Files to be in the directory : `sorter.sh`, `hh_sorted.csv`
Authorized functions : `cat`, `sort`
Comments : n/a

Sometimes having your data not in a random order but sorted in some way can be efficient for later stages of data analysis. So in this exercise, you will need to sort your CSV file by several columns.

Write a shell script `sorter.sh` that:

1. sorts `hh.csv` file from the previous exercise by the date of vacancy creation and then by the id in the ascending order,
2. saves the result in the CSV file `hh_sorted.csv`.

The CSV file still must have headers in the first row.

Your script must be executable. The interpreter to use is `/bin/sh`.

Put your shell script in the `ex02` folder in the root of your repository as long as your result of the sorting.

Chapter VI

Exercise 03

Exercise 03 : Replacing strings in a file
Directory to store your solution : `ex03/`
Files to be in the directory : `cleaner.sh`, `hh_positions.csv`
Authorized functions : `sed`
Comments : n/a

Raw data is a mess. Before you can start analyzing it, you need to do a lot of preprocessing. In this exercise, you continue doing it. If you look at your file from the previous exercise, you will see that every name of position contains “Data Scientist”. It is not a surprise since we used that string as the keyword for the search in HeadHunter API. But for us and for algorithms it does not give any useful information. To be honest it is a noise that worsens data analysis.

Write a shell script `cleaner.sh` that:

1. removes “Data Scientist” and other extra characters from the names of position, if the name contains any other information,
2. keeps “Data Scientist” if it is the only thing that the name contains,
3. saves the result in the CSV file `hh_positions.csv`.

You can see the example below:

```
“id”,”created_at”,”name”,”has_test”,”alternate_url”
"35218725","2020-04-11T18:03:53+0300","Junior",false,"https://hh.ru/vacancy/35218725"
"36359628","2020-04-11T19:25:48+0300","Senior",false,"https://hh.ru/vacancy/36359628"
"35895583","2020-04-12T12:06:33+0300","Специалист (big data, прогностическая аналитика, data
mining)",false,"https://hh.ru/vacancy/35895583"
```

The CSV file still must have headers in the first row and to be sorted accordingly to the previous exercise.

Your script must be executable. The interpreter to use is `/bin/sh`.

Put your shell script in the `ex03` folder in the root of your repository as long as your result of cleaning.

Chapter VII

Exercise 04

Exercise 04 : Descriptive statistics
Directory to store your solution : `ex04/`
Files to be in the directory : `counter.sh`, `hh_uniq_positions.csv`
Authorized functions : `uniq`
Comments : n/a

Before doing something more sophisticated, it is better to get the basic knowledge about your data. In this exercise, you will need to count the unique positions from your file. As a result, you can understand that there is some kind of skew in your data: for instance, there are more seniors than juniors. Such kinds of facts might be useful for further analysis.

Write a shell script `counter.sh` that:

1. counts unique values of the name column,
2. sorts the table by the count in the descending order,
3. stores the result in the CSV file `hh_uniq_positions.csv`.

You can see the example below:

```
”name”,”count”
"Junior",10
"Data Scientist",5
"Senior",3
```

The CSV file must have headers in the first row as in the example.

Your script must be executable. The interpreter to use is `/bin/sh`.

Put your shell script in the `ex04` folder in the root of your repository as long as your result of counting.

Chapter VIII

Exercise 05

Exercise 05 : Partitioning and concatenation
Directory to store your solution : `ex05/`
Files to be in the directory : `partitioner.sh, concatenator.sh`
Authorized functions : `cat`
Comments : n/a

When you have a big dataset, sometimes it might be useful to slice it into partitions. Each partition has a specific range of keys. One of the popular ways for partitioning is to do it by date. Each partition contains data for the specific date. In this exercise, you will need to perform that task.

Write one shell script `partitioner.sh` that:

1. takes as input the result of Exercise 03,
2. stores slices of data with different “created_at” dates in separate CSV files with the name of that date.

You can see the example of such a file below:

```
“id”,”created_at”,”name”,”has_test”,”alternate_url”
"35218725","2020-04-11T18:03:53+0300","Junior",false,"https://hh.ru/vacancy/35218725"
"36359628","2020-04-11T19:25:48+0300","Senior",false,"https://hh.ru/vacancy/36359628"
```

Write another shell script `concatenator.sh` that:

1. takes as input separate files from the result of `partitioner.sh`,
2. concatenates all separate files into one CSV file.

The CSV files must have headers in the first row as in the example. The CSV from the result of `concatenator.sh` must be equal to the result of Exercise 03.

Your scripts must be executable. The interpreter to use is `/bin/sh`.

Put your shell scripts in the `ex05` folder in the root of your repository.