CLI Commands

Installing Airflow

```
docker run -it -p 8080 python:3.6-slim /bin/bash
```

- Create and start a docker container from the Docker image python:3.6-slim and execute the command /bin/bash in order to have a shell session

```
python -V
```

- Print the Python version

```
export AIRFLOW_HOME=/usr/local/airflow
```

- Export the environment variable AIRFLOW_HOME used by Airflow to store the dags folder, logs folder and configuration file

```
env | grep airflow
```

- To check that the environment variable has been well exported

```
apt-get update -y && apt-get install -y wget libczmq-dev curl
libssl-dev git inetutils-telnet bind9utils freetds-dev libkrb5-dev
libsasl2-dev libffi-dev libpq-dev freetds-bin build-essential
default-libmysqlclient-dev apt-utils rsync zip unzip gcc && apt-get
clean
```

- Install all tools and dependencies that can be required by Airflow

```
useradd -ms /bin/bash -d ${AIRFLOW_HOME} airflow
```

- Create the user airflow, set its home directory to the value of AIRFLOW_HOME and log into it

```
cat /etc/passwd | grep airflow
```

- Show the file /etc/passwd to check that the airflow user has been created

```
pip install --upgrade pip
```

- Upgrade pip (already installed since we use the Docker image python 3.6)

```
pip install virtualenv
```

- Install virtualenv to create a Python Virtual Environment (avoid conflicts between python libraries when they are installed on the system)

```
su - airflow
```

- Log into airflow

```
virtualenv sandbox
```
- Create the virtual env named sandbox

```
source sandbox/bin/activate
```
- Activate the virtual environment sandbox

```
pip install
apache-airflow[crypto,celery,postgres,hive,jdbc,mysql,ssh,docker,hdfs
,kerberos,kubernetes,redis,rabbitmq,ldap,slack,webhdfs]==1.10.3
```
- Install the version 1.10.3 of apache-airflow with all subpackages defined between square brackets. (Notice that you can still add subpackages after all, you will use the same command with different subpackages even if Airflow is already installed)

```
airflow initdb
```
- Initialise the metadatabase

```
airflow scheduler &
```
- Start Airflow's scheduler in background

```
airflow webserver &
```
- Start Airflow's webserver in background

```
cd
/where_you_downloaded_the_materials/airflow-materials/airflow-basic
```
- Move to the folder airflow-materials/airflow-basic (Replace where_you_downloaded_the_materials by where you actually downloaded the course materials)

```
docker build -t airflow-basic .
```
- Build a docker image from the Dockerfile in the current directory (airflow-materials/airflow-basic) and name it airflow-basic

Quick Tour of Airflow CLI

```
docker ps
```
- Show running docker containers

```
docker exec -it container_id /bin/bash
```
- Execute the command /bin/bash in the container_id to get a shell session

```
pwd
```

- Print the current path where you are

```
airflow initdb
```
- Initialise the metadatabase

```
airflow resetdb
```
- Reinitialize the metadatabase (Drop everything)

```
airflow upgradedb
```
- Upgrade the metadatabase (Latest schemas, values, ...)

```
airflow webserver
```
- Start Airflow's webserver

```
airflow scheduler
```
- Start Airflow's scheduler

```
airflow worker
```
- Start a Celery worker (Useful in distributed mode to spread tasks among nodes - machines)

```
airflow list_dags
```
- Give the list of known dags (either those in the examples folder or in dags folder)

```
ls
```
- Display the files/folders of the current directory

```
airflow trigger_dag example_python_operator
```
- Trigger the dag example_python_operator with the current date as execution date

```
airflow trigger_dag example_python_operator -e 2015-03-02
```
- Trigger the dag example_python_operator with a date in the past as execution date (This won't trigger the tasks of that dag unless you set the option catchup=True in the DAG definition)

```
airflow trigger_dag example_python_operator -e '2019-07-08
19:04:00+00:00'
```
- Trigger the dag example_python_operator with a date in the future (change the date here with one having +2 minutes later than the current date displayed in the Airflow UI). The dag will be scheduled at that date.

```
airflow list_dag_runs example_python_operator
```
- Display the history of example_python_operator's dag runs

```
airflow list_tasks example_python_operator
```

- List the tasks contained into the example_python_operator dag

```
airflow test example_python_operator print_the_context 2018-05-07
```

- Allow to test a task (print_the_context) from a given dag (example_python_operator here) without taking care of dependencies and past runs. Useful for debugging.