Airflow Cheatsheet

[A][B][C] — HEIG-IST 2025 GROUP X

What It Does

Apache Airflow	automates workflows such as **data pipelines** using Python-based DAGs. Handles **scheduling**, **logging**, and **retries**.

Installation (via Docker Compose)

`git clone https://github.com/Dansnts/workshop-IST`	clone the workshop repository
`cd airflow-docker`	enter the project folder
`docker compose build`	build the Docker images
`docker compose run airflow-webserver airflow db init`	initialize the Airflow database
`docker compose run airflow-webserver airflow users create ...`	create an **admin user** for Airflow
`docker compose up`	start the Airflow services

Basic CLI Commands

`airflow dags trigger <dag_id>`	manually trigger a **DAG**
`airflow dags list`	list all **available DAGs**
`airflow tasks logs <dag_id> <task_id> <execution_date>`	view logs for a specific **task** execution

Key Files

`dags/`	folder containing **DAG definitions**
`docker-compose.yml`	defines services like **Airflow** and **PostgreSQL**
`requirements.txt`	list of **Python packages** used in DAGs

Airflow CLI - DAG Management

`airflow dags list`	list all available **DAGs**
`airflow dags trigger <dag_id>`	manually trigger a DAG run
`airflow dags backfill <dag_id> -s <start_date> -e <end_date>`	run past DAG runs over a date range
`airflow dags pause <dag_id>`	pause a DAG (disable scheduling)
`airflow dags unpause <dag_id>`	unpause a DAG (enable scheduling)
`airflow dags delete <dag_id>`	delete a DAG from the metadata database

Airflow CLI - Task Management

`airflow tasks list <dag_id>`	list all tasks in the DAG
`airflow tasks test <dag_id> <task_id> <execution_date>`	run a single task **without dependencies**
`airflow tasks run <dag_id> <task_id> <execution_date>`	run a single task **with dependencies**
`airflow tasks clear <dag_id> --start-date <s> --end-date <e>`	clear task instances in a date range
`airflow tasks logs <dag_id> <task_id> <execution_date>`	view logs for a task execution

Airflow CLI - Database & Maintenance

`airflow db init`	initialize the Airflow metadata database
`airflow db upgrade`	apply database **schema migrations**
`airflow db reset`	reset the Airflow database (danger: deletes data)
`airflow db check`	check DB connectivity and migrations
`airflow db shell`	open a **SQL shell** connected to the metadata DB

Airflow CLI - User Management

`airflow users create --username ...`	create a user with specified credentials and role
`airflow users list`	list all users in the system

Airflow CLI - Monitoring & Scheduling

`airflow scheduler`	start the Airflow **scheduler** process
`airflow webserver`	start the Airflow **web UI**
`airflow celery worker`	start a **Celery worker** (if using CeleryExecutor)

`airflow celery flower`	start the **Flower UI** for Celery monitoring

Airflow Environment & Config

`airflow info`	show information about Airflow environment
`airflow config get-value <section> <key>`	read value from `airflow.cfg`
`airflow variables get <key>`	get a variable from Airflow metadata DB
`airflow variables set <key> <value>`	set a variable in Airflow metadata DB
`airflow connections list`	list all Airflow **connections**
`airflow connections add`	create a new connection (use flags to configure)
`airflow plugins list`	list installed plugins

Advanced DAG Commands

`airflow dags show <dag_id>`	print the **graph structure** of a DAG in text format
`airflow dags report`	report on DAG status, duration, success rate, etc.
`airflow dags next-execution <dag_id>`	print the next scheduled **execution date** of the DAG

Debugging & Development

`airflow tasks test <dag_id> <task_id> <execution_date>`	test a task in **isolation** without a scheduler
`AIRFLOW__CORE__LOAD_EXAMPLES=False`	disable example DAGs in `airflow.cfg`
`airflow webserver --reload`	reload webserver automatically when code changes
`python -m pdb dags/<file>.py`	debug a DAG using the Python debugger

Task Instance Management

`airflow tasks state <dag_id> <task_id> <execution_date>`	print the current **state** of a task
`airflow tasks failed-deps <dag_id> <task_id> <execution_date>`	show **why** a task did not run
`airflow tasks render <dag_id> <task_id> <execution_date>`	render the task template with context

Scheduler & Triggering Logic

`airflow dags next-execution <dag_id>`	get the next time the DAG will run
`airflow dags list-runs --dag-id <dag_id>`	list all **DAG runs** for a DAG
`airflow dags state <dag_id> <execution_date>`	get the DAG run state