SIDEBAR Programming

Django Commands Cheat Sheet

Sidebar

Programming · Django Commands Cheat Sheet

The commands for an everyday Django project, starting with a copy-and-paste sequence that creates `my_project` from nothing. Every command in the sequence is preceded by `pwd` so you can see which directory you are in before it runs. Written against Django 6.1.

START HERE: A NEW PROJECT FROM NOTHING (LINUX)

```
# Run these in order, one at a time. After each "pwd", compare the path with the "expect" r

# 1. Make the project folder (works from any directory)
pwd
mkdir -p /home/philip/Projects/my_project

# 2. Go into it. Every later step runs from here unless it says otherwise.
pwd
cd /home/philip/Projects/my_project

# 3. Create the virtual environment, a folder named .venv inside the project
pwd                                  # expect: /home/philip/Projects/my_project
python3 -m venv .venv

# 4. Activate it. Your prompt should now start with (.venv)
pwd                                  # expect: /home/philip/Projects/my_project
source .venv/bin/activate

# 5. Upgrade pip inside the venv (a new venv often ships an old one)
pwd                                  # expect: /home/philip/Projects/my_project
python -m pip install --upgrade pip

# 6. Install Django into the venv
pwd                                  # expect: /home/philip/Projects/my_project
pip install django

# 7. Create the project. The dot at the end means "here", so manage.py lands in this folder
pwd                                  # expect: /home/philip/Projects/my_project
django-admin startproject my_project .

# 8. Create an app called blog. Run it where manage.py is
pwd                                  # expect: /home/philip/Projects/my_project
ls                                   # expect to see manage.py in the list
python manage.py startapp blog

# 9. Register the app: edit my_project/settings.py and add 'blog' to INSTALLED_APPS (not a

# 10. Create the database tables for Django's built-in apps
pwd                                  # expect: /home/philip/Projects/my_project
python manage.py migrate

# 11. Create an admin login (it asks for a username, email and password)
pwd                                  # expect: /home/philip/Projects/my_project
python manage.py createsuperuser

# 12. Save the package list so the environment can be rebuilt
pwd                                  # expect: /home/philip/Projects/my_project
pip freeze > requirements.txt
```

```
# 13. Start the development server, then open http://127.0.0.1:8000/ (Ctrl+C stops it)
pwd                                 # expect: /home/philip/Projects/my_project
python manage.py runserver
```

WHAT YOU END UP WITH

```
my_project/                 # the folder yo
├── .venv/                  # the virtual e
├── manage.py               # the command r
├── requirements.txt
├── db.sqlite3              # appears after
├── my_project/             # the project's
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   ├── asgi.py
│   └── wsgi.py
└── blog/                   # the app
    ├── migrations/
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── tests.py
    └── views.py
```

COMING BACK ANOTHER DAY

`cd /home/philip/Projects/my_project`	Go to the project root first
`source .venv/bin/activate`	Look for `(.venv)` at the start of the prompt
`python manage.py runserver`	Start the server
`deactivate`	Leave the venv when you are finished
`pip install -r requirements.txt`	On a fresh clone: after creating and activating a new venv

THE DEVELOPMENT SERVER

`python manage.py runserver`	Serves on 127.0.0.1:8000 and restarts when you save a file
`python manage.py runserver 8001`	Use a different port
`python manage.py runserver 0.0.0.0:8000`	Reachable from other devices; add the host to `ALLOWED_HOSTS` if you get a 400
`Ctrl+C`	Stop the server
`python manage.py check`	Look for configuration problems without starting anything
`python manage.py check --deploy`	Security checklist for production (a new project reports 8 warnings)

APPS

`python manage.py startapp shop`	Creates the folder `shop/` with models, views, admin, tests and migrations
`'blog'`	Add this string to `INSTALLED_APPS` in `settings.py` after creating an app
`django-admin startproject name .`	Project: only once, and with the dot to avoid an extra nested folder
`python manage.py startapp --help`	Options, including `--template`

MODELS AND MIGRATIONS

`python manage.py makemigrations`	After changing a model: writes the migration files
`python manage.py makemigrations blog -n add_body`	Only for one app, with your own name for the file
`python manage.py migrate`	Applies every unapplied migration to the database
`python manage.py showmigrations`	`[X]` applied, `[ ]` not yet
`python manage.py sqlmigrate blog 0001`	Prints the SQL a migration would run
`python manage.py migrate blog zero`	Undoes all of one app's migrations
`python manage.py makemigrations --check --dry-run`	Changes nothing; exit code 1 if a migration is missing (useful in CI)

USERS AND THE ADMIN

`python manage.py createsuperuser`	Prompts for username, email and password
`python manage.py changepassword admin`	Reset a user's password
`python manage.py createsuperuser --noinput --username admin --email a@b.com`	For scripts, with no prompts: set `DJANGO_SUPERUSER_PASSWORD` in the environment first (worked here)
`/admin/`	The admin site's URL: `http://127.0.0.1:8000/admin/`
`python manage.py clearsessions`	Deletes expired sessions

THE SHELL AND DATA

`python manage.py shell`	Python prompt with Django loaded; it imports your models for you
`python manage.py shell -c "..."`	Run one line of Python and exit
`python manage.py dumpdata blog --indent 2 > blog.json`	Export an app's rows as JSON
`python manage.py loaddata blog.json`	Import them again
`python manage.py inspectdb`	Print model code for an existing database's tables
`python manage.py flush`	Empties every table (it asks first). Keeps the tables
`python manage.py dbshell`	Opens the database's own client (needs `sqlite3` installed; not run here)

TESTS

`python manage.py test`	Runs every test in the project
`python manage.py test blog`	Only one app
`python manage.py test blog.tests.PostTests.test_create`	One test method
`python manage.py test -v 2`	Verbose: shows each test and the database setup
`python manage.py test --keepdb`	Reuse the test database between runs
`python manage.py test --parallel`	Spread tests across processes
`"NO TESTS RAN"`	A new app has an empty `tests.py`; test names must start with `test_`

STATIC FILES

`STATIC_ROOT = BASE_DIR / 'staticfiles'`	Add to `settings.py` first, or the next command fails
`python manage.py collectstatic`	Copies every app's static files into `STATIC_ROOT` (130 for a new project)
`python manage.py collectstatic --noinput`	Without the yes/no prompt
`python manage.py findstatic admin/css/base.css`	Shows which file Django would serve for a path

SETTINGS WORTH KNOWING

`python manage.py diffsettings`	Lists the settings that differ from Django's defaults
`DEBUG`	`True` in development only
`SECRET_KEY`	Generated for you and marked `django-insecure-`; replace and keep out of git for a real site
`ALLOWED_HOSTS`	Must list your domain once `DEBUG` is off
`DATABASES`	SQLite file `db.sqlite3` by default
`python manage.py help`	Lists every command available; `help <command>` for one

PACKAGES AND THE VENV

`pip install djangorestframework`	Install into the active venv only
`pip freeze > requirements.txt`	Run again whenever you add a package
`python -m django --version`	Check the installed Django version
`pip list --outdated`	See which packages have newer versions
`.gitignore`	Add `.venv/`, `db.sqlite3`, `staticfiles/` and `__pycache__/`

TROUBLESHOOTING

`django-admin: command not found`	The venv is not active. Check the prompt for `(.venv)` and run the activate line
`can't open file 'manage.py'`	Wrong directory. Run `pwd` and `ls`; you need the folder that contains `manage.py`
`No module named 'django'`	Not installed in this venv, or a different Python is running. Check `which python`
`You have N unapplied migration(s)`	Run `python manage.py migrate`
`That port is already in use.`	Another server is running (or use a different port: `runserver 8001`). The wording comes from Django's source; I did not trigger it here
`Model or template of a new app is ignored`	The app is not in `INSTALLED_APPS`
`ensurepip is not available`	Debian/Ubuntu: install the `python3-venv` package, then repeat step 3 (from general knowledge, not run here)

THE THREE QUESTIONS BEHIND MOST BEGINNER ERRORS

Which directory am I in (`pwd`, and does `ls` show `manage.py`)? Is the venv active (the `(.venv)` prefix)? Did I add the app to `INSTALLED_APPS`? Almost every "it does nothing" or "command not found" message in this sheet is one of those three.

WHAT WAS CHECKED ON THIS SHEET

I ran the commands from "Start here" onward in a scratch project with Python 3.14.0 and Django 6.1.2 on Windows 11 (`startproject`, `startapp`, `makemigrations`, `migrate`, `createsuperuser`, `shell`, `dumpdata`, `loaddata`, `test`, `collectstatic`, `findstatic`, `diffsettings`, `check`, `clearsessions`, `flush` (declined at its prompt), `runserver` with real requests, including a 400 for an unknown host on `0.0.0.0`). Not run: the Linux-specific steps (`mkdir -p`, `python3 -m venv`, `source .venv/bin/activate` and the `/home/philip/...` paths, which differ only in syntax from Windows), `dbshell` (no `sqlite3` program here), and the port-in-use error.