A dockerized Python Development Environment

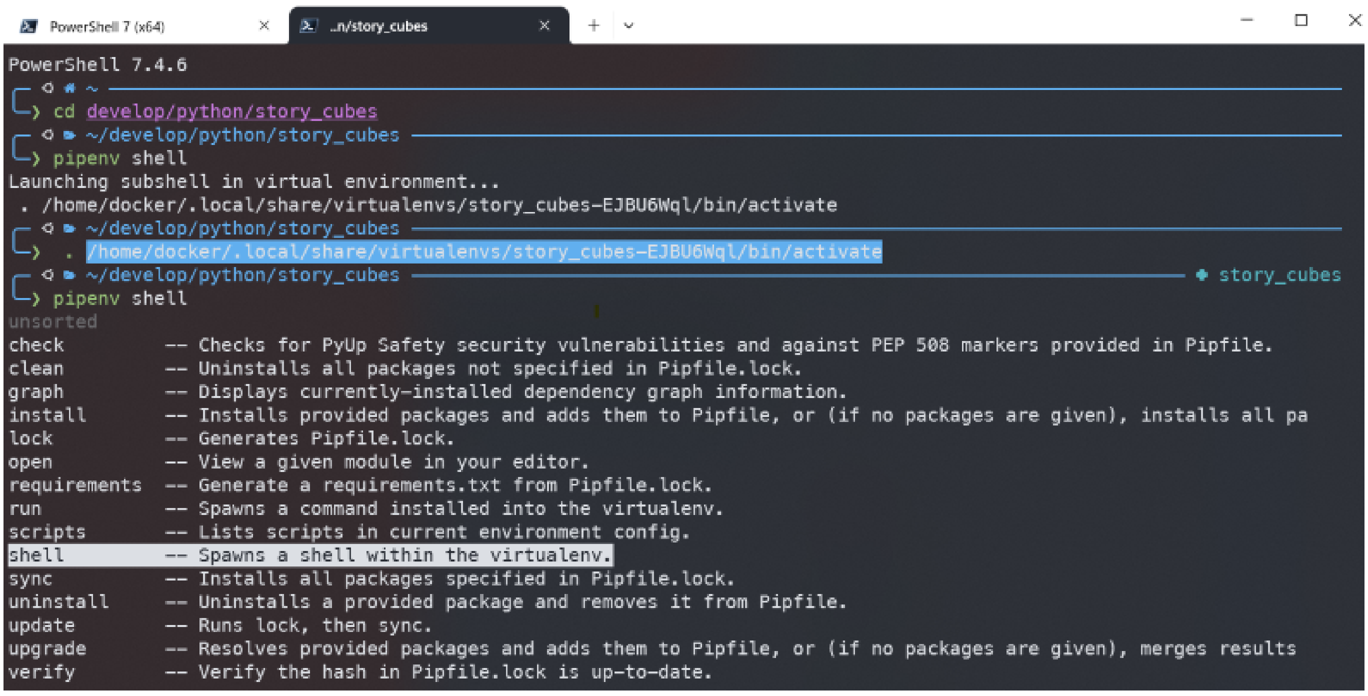
Motivation

On our Window PC, we like to have an Linux environment, for

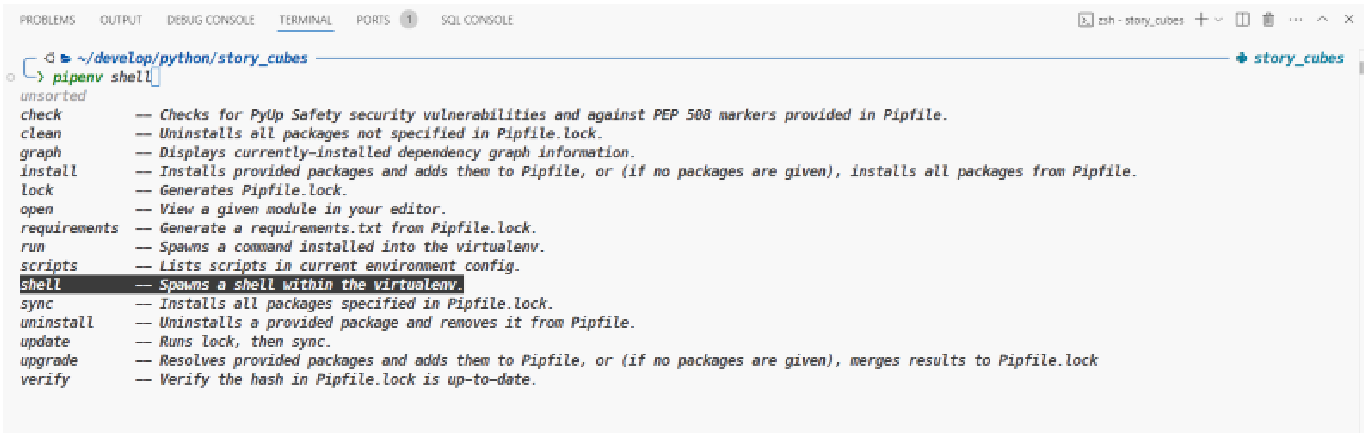
- Unix file management workflow,
- scripting and
- Python development.

We will not use WSL neither WSL2, but Docker - this, we can transfer to another host system - to provide an image and a container that offer a Linux environment, with

- TTY only
- work as a user, e.g. *docker*, including *sudo* rights
- with its home directory */home/docker*
- that */home/docker* shall be directly accessible from the host system's native tools, e.g. via *File Explorer*:
  - at a defined directory, by docker bind or volume (*works_*)
  - at a Windows drive letter, by Unix Samba (*t.b.d.*)
- the environment shall be usable via *VScode* and *Windows PowerShell*
- use *zsh* a shell, including *auto_complete* & *auto_suggestion*
- provide fancy prompt with *p10k* and font *MesloLGS Nerd Font Mono*



```
PowerShell 7.4.6
╭─ ~
╰─❯ cd develop/python/story_cubes
╭─ ~/develop/python/story_cubes
╰─❯ pipenv shell
Launching subshell in virtual environment...
 . /home/docker/.local/share/virtualenvs/story_cubes-EJBU6Wql/bin/activate
╭─ ~/develop/python/story_cubes
╰─❯  . /home/docker/.local/share/virtualenvs/story_cubes-EJBU6Wql/bin/activate
╭─ ~/develop/python/story_cubes                                                       story_cubes
╰─❯ pipenv shell
unsorted
check          -- Checks for PyUp Safety security vulnerabilities and against PEP 508 markers provided in Pipfile.
clean          -- Uninstalls all packages not specified in Pipfile.lock.
graph          -- Displays currently-installed dependency graph information.
install        -- Installs provided packages and adds them to Pipfile, or (if no packages are given), installs all pa
lock           -- Generates Pipfile.lock.
open           -- View a given module in your editor.
requirements   -- Generate a requirements.txt from Pipfile.lock.
run            -- Spawns a command installed into the virtualenv.
scripts        -- Lists scripts in current environment config.
shell          -- Spawns a shell within the virtualenv.
sync           -- Installs all packages specified in Pipfile.lock.
uninstall      -- Uninstalls a provided package and removes it from Pipfile.
update         -- Runs lock, then sync.
upgrade        -- Resolves provided packages and adds them to Pipfile, or (if no packages are given), merges results
verify         -- Verify the hash in Pipfile.lock is up-to-date.
```



```
~/develop/python/story_cubes                                    story_cubes
> pipenv shell
unsorted
check        — Checks for PyUp Safety security vulnerabilities and against PEP 508 markers provided in Pipfile.
clean        — Uninstalls all packages not specified in Pipfile.lock.
graph        — Displays currently-installed dependency graph information.
install      — Installs provided packages and adds them to Pipfile, or (if no packages are given), installs all packages from Pipfile.
lock         — Generates Pipfile.lock.
open         — View a given module in your editor.
requirements — Generate a requirements.txt from Pipfile.lock.
run          — Spawns a command installed into the virtualenv.
scripts      — Lists scripts in current environment config.
shell        — Spawns a shell within the virtualenv.
sync         — Installs all packages specified in Pipfile.lock.
uninstall    — Uninstalls a provided package and removes it from Pipfile.
update       — Runs lock, then sync.
upgrade      — Resolves provided packages and adds them to Pipfile, or (if no packages are given), merges results to Pipfile.lock
verify       — Verify the hash in Pipfile.lock is up-to-date.
```

Installation

Let us create a docker image from ***ubuntu***, so that docker will provide a docker container that we can use as our Python Development Environment and some Unix Scripting.

Prerequisites - installation on local host OS (e.g. Windows):

1. Docker Desktop
    - for Windows see: Install Docker Desktop on Windows
    - for Mac, see: Install Docker Desktop on Mac
    - for Linux, see: Install Docker Desktop on Linux
2. VScode, see: Download Visual Studio Code

Define some folders for our working environment by chosing any folder at your host machine OS (e.g. Windows) and create a structure like:

```
docker/
|-- ubuntu/
|   |-- .dockerignore
|   |-- Dockerfile
|   \-- README.md
|
|-- home_docker/
|   |
|   |   **************************************
|   |   * This is the                        *
|   |   *    my_ubuntu_cont: /home/docker/   *
|   |   *    --> host: home_docker/          *
|   |   **************************************
```

The *my_ubuntu_cont* with its user *docker* will use a dedicated volume, i.e. the users directory `/home/docker/` that is linked to the host's OS (e.g. Windows) directory `home_docker/`.

Build a new image with a *Dockerfile*

1. Go to directory with *Dockerfile*, e.g. *docker/ubuntu/*

```
docker/
    |-- ubuntu/            <-- go here
    |   |-- .dockerignore
    |   |-- Dockerfile
    |   \-- README.md
    |
```

2. check the content of the *Dockerfile*:

```
FROM ubuntu:latest

# see: https://stackoverflow.com/questions/36611052/install-pip-in-docker
RUN DEBIAN_FRONTEND=noninteractive \
apt-get update && apt-get -y install \
    zsh \
    git curl wget lynx \
    iputils-ping lshw net-tools \
    nano bc gawk htop eza fzf bat neovim stow \
    sudo \
    python3.12 python3-pip pipenv \
    tzdata \
    unminimize

# correct timezone
ENV TZ=Europe/Berlin
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ >
/etc/timezone

# see: https://github.com/deluan/zsh-in-docker
# Default powerline10k theme, no plugins installed
RUN sh -c "$(wget -O- https://github.com/deluan/zsh-in-
docker/releases/download/v1.2.1/zsh-in-docker.sh)"

# -------------------------- user:passwd
RUN useradd docker && echo "docker:docker" | chpasswd
RUN usermod --shell /usr/bin/zsh -aG sudo docker
# ------------------------------------- user:group
RUN mkdir -p /home/docker && chown -R docker:docker /home/docker

USER root
```

3. Build a new image *ubuntu_img* and tag it (-t) with new image name:

```
docker build -t ubuntu_img .   # <-- notice the dot at the end
```

With `docker image ls`, you see all images, including the new *pandoc_img*:

```
REPOSITORY    TAG    IMAGE ID     CREATED        SIZE
mubuntu_img   latest daedc675771f 30 minutes ago 947MB
```

Create a new container from the *ubuntu_img*

... to create a container *ubuntu_cont* for the user *docker*:

Go to **host**'s directory ***docker/home_docker/*** !!

```
docker/
|-- ubuntu/
|   |-- .dockerignore
|   |-- Dockerfile
|   \-- README.md
|
|-- home_docker/  <-- go here
```

If you are here: `PS ...\docker\home_docker>`, proceed with **create** the container with :

for Windows PS or cmd prompt, a one-liner

```
docker container create -it --name ubuntu_cont --user docker -v $PWD/:/home/docker
ubuntu_img zsh
docker container create -it --name ubuntu_cont --user docker -v $PWD/:/home/docker
ubuntu_img zsh
```

for Unix/Mac cmd prompt, multiple lines

```
docker container create -it \
    --name ubuntu_cont \
    --user docker \
    -v $PWD/:/home/docker \
    ubuntu_img \
    zsh
```

If you initially did above container creation you may **unminimize** the *ubuntu* installation by

```
sudo unminimize
```

Sometimes you like to reconfigure the prompt, do so by

```
p10k configure
```

Ensure that the ***MesloLGS Nerd Font Mono*** ist configured at your **PS** and **VScode**

- **PS**: Go to Einstellungen > Standardwerte > Darstellung > Schriftart > set to: `MesloLGS Nerd Font Mono`
- **VScode**: Goto File > Preferences > Settings > at "Search settings" input: Terminal:Integrated:Font > set to: `MesloLGS Nerd Font Mono`

Start a container

If you already have a container ***ubuntu_cont***:

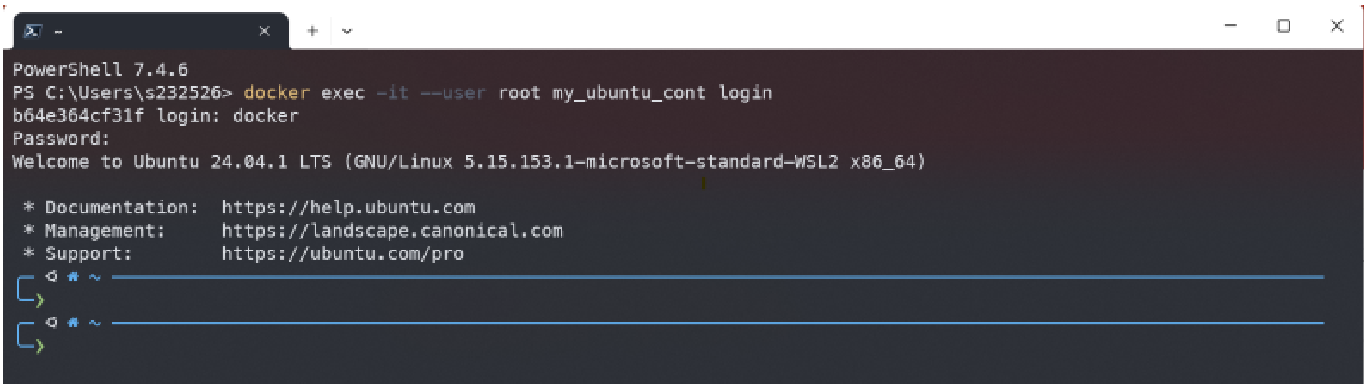
> **Note:**
>
> the container ***ubuntu_cont*** knows: it will use the volume, defined with
>
> `-v $PWD/:/home/docker`
>
> so you do NOT need to start at a dedicated host's directory (as with step 1.)

Start *ubuntu_cont* with

```
docker start ubuntu_cont
```

If container is running: **exec** with **login**:

```
docker exec -it ubuntu_cont login
```



```
PowerShell 7.4.6
PS C:\Users\s232526> docker exec -it --user root my_ubuntu_cont login
b64e364cf31f login: docker
Password:
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 5.15.153.1-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro
```

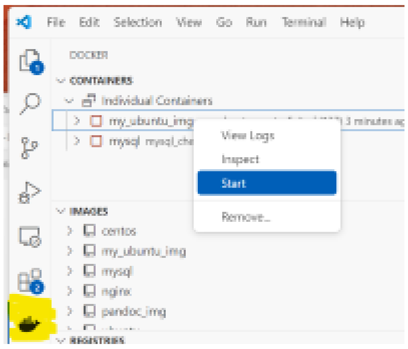
Alternatives, but NOT prefered - use existing container *ubuntu_cont*

After ***ubuntu_cont*** is running, you can **attach** to, with: `docker attach ubuntu_cont` and your prompt will show:

```
PS ... \docker\home_docker> docker start ubuntu_cont
ubuntu_cont
PS ... \docker\home_docker> docker attach ubuntu_cont
```

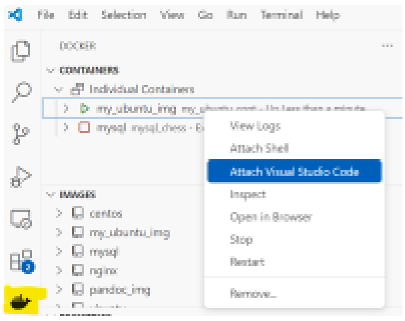
```
╭─ ∅ / ──────────────────────────────────────────────────────────
╰─❯ cd
╭─ ~ ───────────────────────────────────────────────────────────
╰─❯
```

1. Start the container from
   - PS or VScode Terminal prompt (aka command line) with: `docker start ubunut_cont`
   - inside VScode:



```
PS ... \docker\home_docker> docker start ubuntu_cont
ubuntu_cont
```

2. Attach - better **exec** - to a running container via
   - PS or VScode Terminal prompt (aka command line) with: `docker attach ubunut_cont`
   - inside VScode:



```
PowerShell 7.4.6
PS ... > docker start ubuntu_cont
ubuntu_cont
PS ... > docker exec -it ubuntu_cont login
e4215510551d login: docker
Password: docker
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 5.15.153.1-microsoft-standard-WSL2
x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro
╭─ □ □ ~ ──────────────────────────────────────────────────────────────
╰─❯ <ENTER>
╭─ □ □ ~ ──────────────────────────────────────────────────────────────
╰─❯
```

Create image from a container

Sometimes you like to make your own image from the container that you are working with - Google `docker commit`

Open Items:

```
# TODO:

# TODO:
#
```