# Infra

Imahn Shekhzadeh
imahn.shekhzadeh@unige.ch

December 18, 2023

# Contents

## Note

The commands in this document might only run through if you use the `.bashrc` file provided in App. A

# 1. Bash & Linux

In Bash, `[[ ]]` over `[ ]` is preferred, since `[[ ]]` is safer and more capable within Bash scripts. Within `[ ]`, where word splitting and filename expansion do occur, it is good practice to double-quote variables. But it is safe to omit the double-quotes for e.g. $# within `[[ ]]`.

## 1.1. File Download

Downloading file from URL and allowing for redirects,

```
curl -Lo output.out https://url.com
```

When taking a `GitHub` link, note that you need to take the URL of the raw file.

## 1.2. `for`-loops

For this directory structure,

```
infra_upd.tex
infra_upd.pdf
```

rename via

```
for file in infra_upd.*; do mv "$file" "${file/infra_upd/infra}"; done
```

What happens is a substring replacement.

## 1.3. Argument Retrieval

Retrieving all but the first argument,

```
bash_func(){
 shift

 echo "all provided args (except the first): $@"
}
```

Doing this $N \geq 1$-times,

```
test_sth(){
 shift
 ...
 shift

 echo "all provided args (except the first N): $@"
}
```

## 1.4. Colored Outputs

Using colored outputs in Bash, cf. `str_diff` in App. A.

Personally, I use the following color scheme for the CLI,

1. monokai color scheme, i.e. dark gray background (#272822) with light peach color for the text (#F8F8F2).
2. File paths are still displayed in blue, which is suboptimal, to change the color to the better readable cyan-blue color, click on the three horizontal lines in the CLI, then on **Preferences**, then choose the currently active color, switch to the **Colors** tab, then go to **Palette**, click on the blue color & instead use the color #66D9EF

where `-e` stands for human readability and `-s` for summarizing.

## 1.5. String/File/Directory Operations

Appending line to file (`-a`: `appending`, otherwise `tee` overwrites `output.out` if existent),

```
echo "this is a line" | tee -a output.out # -a: appending, important
```

Checking whether string is empty,

```
[[ -z "$env_name" ]] && echo "The string is empty."
```

Finding out size of file/directory,

```
du -hs <path_to_file_or_dir> # du -hs file.ext

# for shorter summary (single quotation strings required)
du -hs <path_to_file_or_dir> | awk '{print $1}'
```

Unzipping a file via the CLI,

```
unzip /path/to/file.zip -d /path/to/destination
```

Opening a file and automatically scrolling to the bottom,

```
less +G /path/to/file.ext
```

Searching for files with specific extension, e.g. `.ext`:

```
find . -name "*.ext"
# find . -name "*.png"
```

Creating new directory including all parent directories (`-p` option is safe, since if directory is already existent, no error will be outputted),

```
mkdir -p <dir>
```

Comparing the contents of two directories,

```
diff -r --color directory1 directory2 # `-r` for recursive comparison
diff -rq --color directory1 directory2 # `-q` suppresses the output of
    differences and only shows which files differ
```

Ignoring files only existent in one of the directories (which treats absent files as empty),

```
diff -rq --color --unidirectional-new-file directory1 directory2
```

## 1.6. Monitoring

```
htop
```

## 1.7. Systems Information

Retrieving the number of available CPU resources,

```
echo "$(nproc)"
```

Print day and time from CLI,

```
echo "$(date +%d_%m_%y-%H_%M_%S)"
# echo "$(date +%dp%mp%y-%Hp%Mp%S)"
```

Listing all available kernels in Debian-based Linux systems,

```
dpkg --list | grep linux-image
```

Currently active kernel version,

```
uname -a
```

## 1.8. CUDA

- When you need to find out the CUDA version installed, install `nvidia-cuda-toolkit`, but do NOT reboot. After its use, immediately remove this package and any package installed alongside with it!
- In case NVIDIA drivers do not allow for boot into Ubuntu, e.g. because you did not uninstall the `nvidia-cuda-toolkit` package,
  1. Boot into an older kernel version of Linux (in order to get there, do a ”hard” reboot, and then go into “Advanced options for Ubuntu”, and choose an older kernel version).
  2. Once booted into the older kernel version, I removed ‘nvidia-cuda-toolkit‘ and rebooted.
  3. After a few more hard reboots and booting into the older kenel version, at some point, the newer kernel version was picked up and worker again.
  4. Now to fix the monitors (because dual-monitor setup didn’t work), I had to open the program “Additional Drivers” and change the driver from the open-source version to an NVIDIA proprietary one.
  5. Then I had to install CUDA according to these instructions.
  6. For PyTorch to recognize the GPU, I had to reboot.

# 2. Docker

## 2.1. Installation

- Follow this great tutorial by DigitalOcean.
- To use NVIDIA GPUs (both in PyTorch & Jax), install the NVIDIA Container Toolkit
- Once done with the installation of the NVIDIA Container Toolkit, proceed with the configuration. During the configuration, it will be necessary to restart the docker daemon, which you can achieve as follows:

```
sudo systemctl restart docker
```

## 2.2. Basics

- Interactive start of containers:

```
d ps -a # find out ID (also docker container name)
d start -i ID
```

- Copying files from local system to docker container and vice versa; **run both commands from local CLI**

```
d cp file_name container_ID:/target_dir # local -> docker
d cp container_ID:/file_name dir_name # docker -> local
```

## 2.3. Dockerfile

- When you find the command for pulling a docker image on `https://hub.docker.com`, e.g.

```
d pull ubuntu:jammy-20231004
```

  then in the Dockerfile, just write

```
FROM ubuntu:jammy-20231004
```

  When no tag is specified, by default the *latest* one will be taken. However, using the *latest* tag can potentially cause issues with reproducibility and consistency, because you might pull a different version of the image at different times without knowing it if the latest tag gets updated. **For more predictable builds, it is advised to use a specific version tag.**

- Note that the structure of the *docker pull* command is

```
d pull [OPTIONS] NAME[:TAG|@DIGEST]
```

In general, the *NAME* is in the format *repository/image*. If *repository* is not specified, Docker assumes the image is located in the default DockerHub library repository. However, many images (like PyTorch) are hosted under a specific user or organization's namespace on DockerHub, rather than the top-level library. That's why the command for the docker pull (for the latest tag) reads

```
d pull pytorch/pytorch
```

- If using a Docker image like *pytorch/pytorch:latest*, conda is already installed. In this case, the default environment is named *base*, which is a common practice in Docker images with conda – unless otherwise stated.
- Copying local scripts into docker container,

```
COPY relative/path/to/script.py .
```

From the documentation:

> Multiple ⟨src⟩ resources may be specified but the paths of files and directories will be interpreted as relative to the source of the context of the build.

It is also important to put the . at the end, since it represents the destination in the Docker image where the file should be copied. The dot . refers to the current working directory inside the Docker image, which is determined by the WORKDIR command in the Dockerfile. If WORKDIR is not set, it defaults to the root directory (/) of the image.

Also, each time the script `relative/path/to/script.py` changes, the Dockerfile needs to be rebuilt – **however, a cached version will be used, which speeds things up**.

- Copying local dirs into docker container,

```
COPY relative/path/to/dir/ .
```

- Running a Dockerfile,

```
d build -f file_name -t img_name .
d build -f file_name -t img_name:tag_name . # tag name optional, but recommended, e.g. 1.0 (no quotes required)
# d build -f file_name --no-cache -t [...] # forcing to rebuild from scratch, no cached version is used (only do if really required)
```

where `img_name` will be the name of the newly created image, `tag_name` the tag name and `file_name` the name of the docker file.

- Via

```
EXPOSE custom-port-number
# EXPOSE 80
```

it is possible to expose a port. Note that port exposure is related to network access. Note that even though network access might not be needed, there is still no harm in exposing a port (since an exposure of the port does not make the docker container more vulnerable).

## 2.4. Docker images

- A Dockerfile does not necessarily need to have the name *Dockerfile*. To pass another name when building the img, do

```
d build -f custom_docker_file .
```

The . specifies the context of the build, which is the current directory in this case. **I would recommend running this command from the same dir in which `custom_docker_file` is located**.

- Check all available Docker images via

```
d images
```

- Cleaning up dangling docker images (these are the entries with ⟨*none*⟩ in the repository or tag name in the output of the previous algo):

```
d image prune -f
```

- Removing a Docker image – **only do this when finished with using the image**

```
d image rm Image_name:Tag
# d container rm <container_id> # in case some containers are using
    the image
```

## 2.5. Docker containers

### 2.5.1. Basics

- Running Docker images – without being able to utilize NVIDIA GPUs:

```
d run -it img_name # if `tag_name` was not provided
d run -it img_name:tag_name # if `tag_name` was provided during
    build (recommended)
```

- Running Docker images & utilizing GPUs:

```
d run --gpus all -it img_name
d run --gpus all -it img_name:tag_name # recommended
```

- To mount a local file to the container at runtime, do

```
d run -v /absolute/path/to/script.py:/path/to/workdir/script.py --
    gpus all -it img_name
d run -v /absolute/path/to/script.py:/path/to/workdir/script.py --
    gpus all -it img_name:tag_name # recommended, provide `img_name`
     & `tag_name`
```

The mounting expects **absolute** file paths on the side of the host machine.

- Note that you can include the bash command **pwd** to avoid having to manually pass absolute paths for the mounting

```
d run -v $(pwd)/script.py:/path/to/workdir/script.py --gpus all -it
    img_name:tag_name # recommended, provide `img_name` & `tag_name`
```

If you need the container to reflect changes made to the scripts on the host without rebuilding the image every time, you would use the *-v* flag to mount the directory. If the scripts won't change, or you don't need to reflect changes in real-time, you don't need to mount the directory, as the necessary scripts have already been copied into the image during the build process.

- It is also possible to directly mount directories:

```
d run -v $(pwd)/dir_path:/path/to/workdir --gpus all -it img_name:
    tag_name
```

Note that the specified directory from the host is mounted into the container at the specified mount point. If there are any existing files or directories in the container at the mount point, they become obscured by the mount.

- In several cases it can be useful to remove the docker container right after execution: When you...
  - ...are running many short-lived containers, like during development or testing,
  - ...want to avoid manual cleanup of stopped containers later on,
  - ...are running containers for one-off tasks that do not need to persist any state after they are finished.

  In this case,

```
d run --rm -v $(pwd)/dir_path:/path/to/workdir --gpus all -it
    img_name:tag_name
```

- It is also possible to mount two separate host directories to two separate directories within the container,

```
d run --rm -v $(pwd)/dir_path1:/path/to/workdir1 -v $(pwd)/dir_path2
    :/path/to/workdir2 --gpus all -it img_name:tag_name
```

This will not cause any overwriting as each *-v* flag creates a unique mount point inside the container.

- Finding out the python version of the Docker image

```
d run -it --rm img_name:tag_name python3 --version
```

This command will immediately remove the container after execution.

### 2.5.2. Passing Arguments

It is possible to pass arguments when running a docker container.

1. Assuming you have a bash script *run_scripts.sh*, in which a Python script, e.g.

```
#!/bin/sh
isort /app/scripts/*.py
black /app/scripts/*.py

python3 -B /app/scripts/test_script.py
python3 -B /app/scripts/test_anil.py
```

   Modify this bash script s.t. any arguments passed to the CLI when running the docker container are picked up,

```
python3 -B /app/scripts/test_anil.py "$@"
# python3 -B /app/scripts/test_script.py "$@" # alternative
```

2. Rebuild (!) the docker image.

3. Now run the docker container as follows:

```
d run --rm -v $(pwd)/dir_path:/path/to/workdir --gpus all -it
    img_name:tag_name arg1 arg2
# d run --rm -v $(pwd)/dir_path:/path/to/workdir --gpus all -it
    img_name:tag_name --n_ways 1 --k_shots 1 # example
```

### 2.5.3. Listing & Stopping

- Listing all running containers,

```
d ps
```

  Listing only the container ID (of all running containers),

```
d ps -q
```

- Stopping a running container,

```
d stop container-ID
```

- Stopping a running container and removing it,

```
d stop container-ID && d rm container-ID
```

# 3. AWS S3

## 3.1. Installation & Configuration

1. Installation instructions

2. The CLI will display the path under which the *aws* package was installed, but it might be sufficient to simply run

```
aws
```

   Double check by running

```
which aws
```

3. After installation, configuration is necessary. For this run

```
aws configure
```

   You can leave these fields empty:

```
Default region name [None]:
Default output format [None]:
```

   A configuration file will be saved under

```
~/.aws/credentials
```

4. In the case you are a member of UNIGE, you can obtain the AWS access key ID and the secret access key as follows:

```
echo -n "$user_name" | base64 # the `-n` is important in this
    context
echo -n "$passwd" | md5sum
```

   where $user_name and $passwd need to be provided

   Otherwise, you need login to the AWS Management Console.

5. **To test the configuration was successful, do this:**

```
aws s3 ls --endpoint-url https://your-custom-s3-endpoint.com
```

   where you replace the endpoint-url `https://your-custom-s3-endpoint.com` with yours.

## 3.2. AWS Credentials (Profiles)

- It is possible to use several profiles in the file ~/.aws/credentials.
- For example,

```
[default]
aws_access_key_id = YOUR_DEFAULT_ACCESS_KEY
aws_secret_access_key = YOUR_DEFAULT_SECRET_KEY

[profile1]
aws_access_key_id = ANOTHER_ACCESS_KEY_ID
aws_secret_access_key = ANOTHER_SECRET_ACCESS_KEY

[profile2]
aws_access_key_id = YET_ANOTHER_ACCESS_KEY_ID
aws_secret_access_key = YET_ANOTHER_SECRET_ACCESS_KEY
```

Using specific profile when running `aws cli` commands via `--profile` option in the command:

```
aws s3 --profile profile1 [...]
# aws s3 --profile default [...]
```

## 3.3. Buckets

One can have several buckets.

### 3.3.1. Creation

- Creating a new bucket,

```
aws s3api create-bucket --bucket custom-bucket-name --endpoint-url
    https://custom-s3-endpoint.com --profile default
```

### 3.3.2. Listings

- Directly "folder" contents of an s3 bucket,

```
aws s3 ls s3://custom-bucket-name --recursive --endpoint-url https
    ://custom-s3-endpoint.com --profile default # `--recursive`
    optional
```

- Showing file contents,

```
aws s3 ls s3://custom-bucket-name/prefix/ --recursive --endpoint-url
     https://custom-s3-endpoint.com --profile default # `--recursive
    ` optional
```

Note that the / at the end of the prefix ("folder") is necessary.

### 3.3.3. File Copying

- Local machine ⟶ S3:

```
aws s3 cp path/to/custom_file.ext s3://custom-bucket-name/path/to/
    custom_file.ext --endpoint-url https://custom-s3-endpoint.com --
    profile default
```

- S3 ⟶ local machine:

```
aws s3 cp s3://custom-bucket-name/path/to/s3_file.ext custom/
    destination --endpoint-url https://custom-s3-endpoint.com --
    profile default
```

### 3.3.4. Directory Copying

- Local machine ⟶ S3:

```
aws s3 sync path/to/dir s3://custom-bucket-name/path/to --endpoint-
    url
https://custom-s3-endpoint.com --profile default
```

### 3.3.5. Directory/File Deletion

- Deleting a folder (which is essentially a prefix in S3) and its contents in an S3 bucket,

```
aws s3 rm s3://your-bucket-name/path-to-your-folder --recursive --
    endpoint-url https://custom-s3-endpoint.com --profile default
```

- Deleting a file,

```
aws s3 rm s3://your-bucket-name/path-to-your-file.out --recursive --
    endpoint-url https://custom-s3-endpoint.com --profile default
```

## 3.4. Cloudpathlib

- When you use the cloudpathlib module, and you want to specify a profile, do this:

```
from cloudpathlib import S3Path, S3Client

# Create an S3 client with a specific AWS profile
s3_client = S3Client(
aws_access_key_id=aws_access_key_id,
aws_secret_access_key=aws_secret_access_key,
endpoint_url=endpoint_url,
profile_name="profile1", # specify profile here
)

# Make `client` default:
client.set_as_default_client()
```

# 4. Conda

- Retrieving information about currently activated conda environment,

```
conda info
```

- Listing all installed environments,

```
conda env list
```

## 4.1. Installation of Environments

- Installing conda with specific python version,

```
# only `myenv` needs to be specified (quotation marks necessary)
env_name="myenv" && conda create -n "$env_name" python=3.11.3 -y &&
    conda activate "$env_name"
```

As of Oct 16, I wouldn't recommend installing python 3.12.0 yet — I got a lot of unmet dependency problems when trying to install torch 2.1 with NVIDIA Cuda version 11.8 afterwards.

- Installation of conda environment from bash file:

```
conda deactivate # go into base environment
source conda/filename.sh
touch .env
```

- Completely remove conda environment,

```
conda deactivate && conda remove -n custom-env-name --all -y
```

## 4.2. Export

- Exporting an `.yml`-file to share with others for reproducibility,

```
conda env export > environment.yml
```

- Line "Prefix:" at end of `.yml` file can be safely deleted, for details cf. here

## 4.3. Installation & Removal of Packages

- Installation of packages from `pyproject.toml` file,

```
pip install -e .
```

If there is not enough free space, do

```
TMPDIR=[...] pip install -e .
```

where `TMPDIR` needs to exist.

- Installing specific conda package version,

```
conda install -c conda-forge custom-pkg-name -y
# conda install -c conda-forge cloudpathlib=0.15.1 -y
```

- Removing list of packages from conda environment,

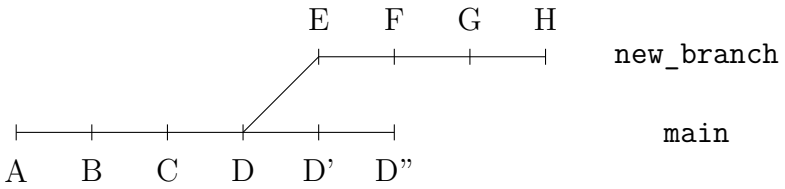
```
conda remove -n custom-env-name pkg1 pkg2 ... pkgN -y
# conda remove -n google_jax matplotlib -y
```

## 4.4. Usage in VSCode

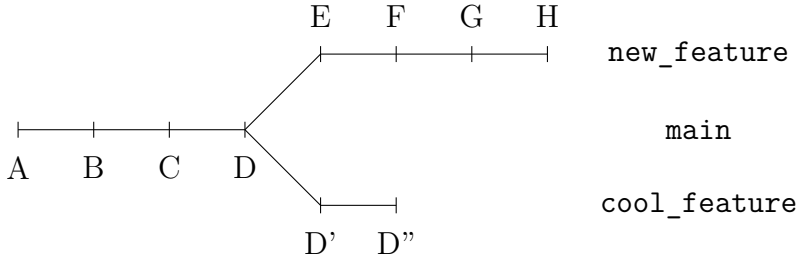
- Selecting a conda environment in VSCode, do `Ctrl + Shift + P` and type `Python: select interpreter`.

# 5. Git

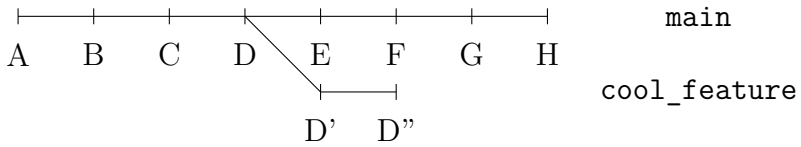
## 5.1. Merging




When merging `main` and `new_branch`, the commits `D` and `D''` will either be auto-merged or there will be a merge conflict. For this commit history,




merging `main` with `new_feature` would result in a fast-forward,




Merging `new_feature` into `main` can be done via

```
git switch main && git merge new_feature
```

Running dry merges to proactively check for conflicts *if* a merge was performed,

```
git merge --no-commit --no-ff branch-name && git merge --abort
```

For a fast-forward only,

```
git merge branch_to_be_merged --ff-only
```

## 5.2. Merge Conflicts

1. Resolving a merge conflict,

```
git mergetool
```

2. Confirm with Enter that you want to use vimdiff as default editing tool. vimdiff display will resemble the following structure:

```
| LOCAL | BASE | REMOTE |
          MERGED
```

If file did not already exist in BASE, then we need this view:

```
| LOCAL | MERGED | REMOTE |
```

LOCAL – Current branch
BASE – Common ancestor (how did the file look like before both changes?)
REMOTE – File that I am merging into the current branch
MERGED – Merge result

3. It is probably easiest to take the merged view and edit it directly. In the vim editor, an entire line can be deleted by pressing D (no control before!). If I instead wanted the changes from either LOCAL, BASE or REMOTE, you have to do one of these,

```
:diffg LO
:diffg BA
:diffg LO
```

Of course, the merged view can also be edited directly.

4. Type

```
:wqa
```

into vim. Afterwards, do not forget to commit and push. And if you want, do

```
git clean -f
```

## 5.3. Checking History

- Viewing the history of commits,

```
git log
```

- Viewing a specific file,

```
git show <commit-hash>:<file-name>
# git show 123abc:example.txt
```

## 5.4. Removing a File/Folder

- To remove a file/folder that is already tracked, adding it to .gitignore won't remove it (though this also needs to happen). For this, do:

```
git rm --cached <file>
git rm -r --cached <folder>
```

- Adding the file/folder to *.gitignore* is still a good idea, though, since the file/dir won't be removed locally with the commands.

## 5.5. Renaming a Repository

After having renamed the repository remotely, go to the locally cloned version of the repository and do

```
git remote set-url origin https://github.com/username/new-repo-name.git
```

double-check via

```
git remote -v
```

which lists the remote names and their URLs. No force push or the alike is necessary for the changes to take place.

## 5.6. Restore File

Resetting specific file to state of previous commit,

```
git restore --source=<commit-hash> <file-path> && git push
# git restore --source=HEAD README.md && git push
```

## 5.7. Remote URL

Obtaining the remote URL,

```
git remote get-url origin
git remote get-url origin | sed 's/\.git$//' # optional: trim output
```

## 5.8. Repo Change

Moving all files from `branch-to-move` to `branch-to-merge-into` and preserving the commit history — do all of this while in the old repo,

```
git remote -v # check existing remotes
git remote add <target> https://target-repo-url.git # add new remote
# git remote add new-remote url
git push target branch-to-move:branch-to-merge-into
```

Do all of this in the old repo. If issue emerges during the last step, reclone the new repo and check whether this solves the issue.

## 5.9. Remote Repo Creation

If all files to be added to the repository are in `$folder_name`,

```
folder_name="[...]" && mkdir $folder_name && cd $folder_name && git
    init && lpush "this is the commit msg."
```

where the command `lpush` is defined in App. A. Make sure there is a `README.md` file in the folder `$folder_name`.

Then install GitHub CLI and do

```
gh repo create <repository-name> --public # --private
```

Then commit and push.

## 5.10. Pull Requests

```
gh pr create --base main --head "$bname" --title "Pin isort & black
versions" --body "This pull request fixes the issue that worklfows fail
because of different isort/black versions used in the workflows &
specified in the \`pyproject.toml\` file."
```

Note that `bname` is a bash function defined in App. A. Escaping the `\` is necessary, since in shell commands, backticks (`` ` ``) are used to execute commands and substitute their output into the command line.

## 5.11. Updating Files from other Branch

When working on `branch_my`, it is possible to incorporate changes from another branch `branch_x`,

```
git switch <branch_x>
git pull origin <branch_x>
git switch <branch_my>
git rebase -i <branch_x>
git push origin <branch_my> --force
```

Save the interactive view via `:wq` and make a force push to `branch_my`.

# 6. Remote Development

## 6.1. Connection

1. When connecting two machines remotely, install this extension on local machine (also directly in VSCode possible),
2. open VSCode on local machine,
3. press F1-button, choose "Remote-SSH: Connect to Host..." and type for the SSH host (optionally save it in the SSH config file) the same as in Algo. (B),
4. enter the passwd for the remote SSH host.

## 6.2. Troubleshooting

If you find you are getting a permission error for saving a file on the remote machine (in VSCode when doing the local coding), try

```
sudo chown custom-username path/to/custom/script.ext
```

`custom-username` here refers to the username on the remote machine. If the remote connection hung up,

```
fusermount -zu /path/to/dir
```

# 7. Python

## 7.1. Config File & JSON Files

- When using `argparse` in combination with a JSON configuration file, the JSON keys need to match the long option names specified in `parser.add_argument()` method calls. The `argparse` module itself does not automatically recognize abbreviated forms from a JSON file.

## 7.2. Jupyter Notebooks

- Converting jupyter notebooks into PDFs,

```
for nb in /path/one/Notebook1.ipynb /path/two/Notebook2.ipynb [...]
do
jupyter nbconvert --to pdf "$nb"
done
```

Wildcarding notation would also work,

```
# optionally: `output_dir="[...]"`

for nb in *.ipynb; do
nb_name="${nb%.ipynb}"
jupyter nbconvert --to pdf "$nb" # `--output "$output_dir/$nb_name.
    pdf"`
done
```

## 7.3. Map (Built-In Function)

- Function signature:

```
map(function, iterable, *iterables)
```

Description provided in the documentation:

> Return an iterator that applies function to every item of iterable, yielding the results. If additional iterables arguments are passed, function must take that many arguments and is applied to the items from all iterables in parallel. With multiple iterables, the iterator stops when the shortest iterable is exhausted. For cases where the function inputs are already arranged into argument tuples, see *itertools.starmap()*.

- Example usage: Natively multiplying Python lists elementwise,

```
from typing import List

def multiply(x: List, y: List):
return x * y

list_one = [i for i in range(1000)]
list_two = [j for j in range(1000, 2000)]
result = list(map(multiply, list_one, list_two)) # `map` is a built-
    in function, do not use `(list_one, list_two)` in this case
```

- Example usage: Converting NumPy arrays into PyTorch tensors,

```
a = np.array([1, 2, 3, 4])
tensor_list = list(map(torch.from_numpy, (a,))) # list containing
    tensor, use of additional brackets necessary
```

- Example usage: Converting NumPy arrays into PyTorch tensors,

```
a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])
a, b = map(torch.from_numpy, (a, b)) # tuple unpacking
```

## 7.4. PyTorch

### Leaf Tensors

- If `requires_grad=False`, then the tensor will be leaf by convention. If `requires_grad=True`, then the tensor will be leaf if it was created directly by the user and is **not** the result of an operation, e.g. `.to(device)` when the tensor is on `cpu` and `device="cuda:0"`.

- However, by definition, leaf tensors themselves do not have a **gradient function** `.grad_fn` because they are not the result of a differentiable operation applied to other tensors, i.e. `grad_fn` on such tensors will return `None`. The gradient function in neural network libraries like PyTorch or TensorFlow is associated with tensors that are outputs of differentiable operations.

- The `.grad` attribute on leaf tensors that require gradients, i.e. those for which `requires_grad=True`, stores the gradient computed during backpropagation. (For leaf tensors that have `requires_grad=False`, calling the `.grad` attribute outputs `None`.) Note that for non-leaf tensors, calling `.grad` results in a `UserWarning`, since non-leaf tensors are generally intermediate results in the computation graph, and their gradients are usually not needed once the gradients of the leaf tensors have been obtained. However, there are cases where those gradients are needed, which can be enforced by setting `retain_grad=True` on those tensors,

```
x = torch.tensor([1., 2., 3.], requires_grad=True, device=torch.
    device("cuda:0"))

# fwd pass
```

```
y = x**2

# retain gradients
y.retain_grad()

# backward pas
y.sum().backward()

# access gradients
y.grad # `torch.tensor([1., 1., 1.], device="cuda:0")`
```

Note that in the example of this code snippet, doing `y.grad` means that we access the gradient of the scalar loss function `y.sum()` — on which we performed `.backward()`. Correspondingly, doing `x.grad` implies the gradient of the scalar loss function `y.sum()` with respect to `x`.

- In general, it is **not** possible to perform **in-place** operations on leaf tensors for which `requires_grad=True`, since PyTorch dynamically builds a computational graph during the forward pass, which is used during backpropagation to calculate the gradients. If leaf tensors that have `requires_grad=True` are changed in-place, then the values used during the forward pass are changed, which will affect the gradient calculations in the backward pass. However, note that when no gradients are required for the operations, e.g. when performing parameter updates manually, one can use the context manager `with torch.no_grad()`, in which case in-place operations on leaf tensors **can** be performed, since inside the context manager, `requires_grad=False`.

### Autograd & Backward

- The function `torch.autograd.grad()` computes the gradient. If the gradient of a scalar (loss function) wrt a (weight) matrix is taken, then the output will also be a matrix, where each element corresponds to the partial derivative of the scalar (loss function) wrt to the (weight) matrix element.
- `torch.autograd.grad()` is particularly useful if more direct control over the gradient computation is desired, in particular compared to `.backward()`.
- Note that the default behavior of `.backward()` accumulates gradients in the `.grad` attribute of tensors,

```
x = torch.tensor([1., 2., 3.], requires_grad=True, device=torch.
    device("cuda:0"))

# fwd pass
y = 2 * x

# first backward pass
y.sum().backward(retain_graph=True)
print(f"Gradients of `x` after first backward pass: {x.grad}") # `
    torch.tensor([2., 2., 2.])`

# second backward pass
```

```
    y.sum().backward()
    print(f"Gradients of `x` after second backward pass: {x.grad}") # `
        torch.tensor([4., 4., 4.])`, notice how gradients have
        accumulated
```

However, this behavior can be suppressed by simply zeroing the gradients, i.e. `x.grad.zero_()` — note that `x.grad` returns a tensor, and `<tensor>.zero_()` is a general PyTorch function that sets all elements in-place to 0.

# 8. Jax

Try to install via pip first. Only if this doesn't work use conda!

- Putting a Jax array onto a specific device,

```
import jax
from jax import devices, device_put, numpy as jnp

x = device_put(jnp.arange(10), device=devices("cpu")[0]) # NOTE: put `[0]`
# x = device_put(jnp.arange(10), device=devices("gpu")[0]) # NOTE: put `[0]`
print(f"Device: {x.device_buffer.device()}")
```

- Dtype specification,

```
x = jnp.array([1, 2, 3], dtype=jnp.float32)
print(f"Dtype: {x.dtype}")
```

- Device inference,

```
x.device_buffer.device() # x: Jay array
```

- Making a Jax array out of a Python list or a NumPy array (do not use for tensors),

```
from jax import numpy as jnp

a = jnp.array([1., 2., 3.])
b = jnp.array(np.array([1., 2., 3.]))
```

- jit (just-in-time compilation): sets up a function with XLA (extended linear algebra): check out the NB `test__jit-compil.ipynb`. Using jit,

```
import jax
from jax import numpy as jnp

@jax.jit
def selu(x: jnp.array, lamb: float = 1., alpha: float = 0.):
 return lamb * jnp.where(x > 0, x, alpha * (jnp.exp(x) - 1.0))
```

# A. .bashrc

```
ca() {
 local conda_out="$(conda env list | grep -E "$env_name" | head -n 1 |
    awk '{print $1}')"

 # check non-emptiness
 if [ -z "$1" ]; then
  echo "Usage: ca <env_name>"
  return 1
 fi

 # check env existence
 if [ ! -z "$conda_out" ]; then
  conda activate "$1"
 else
  echo "Conda environment '$env_name' does not exist." # single quotes
     (') only for display
  return 1
 fi

}

# ---------------------------- CONDA ----------------------------

# activate conda environment
# usage: `ca custom-env-name`
ca() {
 conda activate "$@"
}

# deactivate currently activated conda environment
cod() {
 conda deactivate
}

# List all available conda envs:
cel() {
 conda env list
}

# remove conda environment
# usage: `crme ant-migrate-dev`
crme() {
```

```
41
42   # check number of passed arguments via `$#`
43   if [[ $# -ne 1 ]]; then
44    echo "NOTE: Exactly one argument needs to be provided"
45   else
46    conda deactivate && conda remove -n "$1" --all -y
47   fi
48
49  }
50
51  # alias for `conda__remove_packages`
52  # usage (e.g.): `crm myenv pkg1 pkg2`
53  crm() {
54   conda__remove_packages "$@"
55  }
56
57  # remove conda packages from environment
58  # usage (e.g.): `conda__remove_packages myenv pkg1 pkg2`
59  conda__remove_packages() {
60
61   # define local variables first
62   local env_name="$1"
63   local conda_out="$(conda env list | grep -E "$env_name" | head -n 1 |
        awk '{print $1}')"
64
65   # forget first argument (which is saved in `env_name`)
66   shift
67
68   # check non-emptiness
69   if [ -z "$env_name" ]; then
70    echo "Usage: conda__remove_packages <env_name> [package1] [package2]
         ... [packageN]"
71    return 1
72   fi
73
74   # check env existence
75   if [ ! -z "$conda_out" ]; then
76    conda remove -n "$env_name" "$@" -y
77    echo "Package(s) '$@' removed from environment '$env_name'"
78   else
79    echo "Conda environment '$env_name' does not exist." # single quotes
        (') only for display
80    return 1
81   fi
82
83  }
84
85  # ---------------------------- AWS -------------------------
86
```

```
# helper function
get__profile_endpoint_url() {

 # check if the first argument contains "https://"
 if [[ "$1" == https://* ]]; then
  local endpoint_url="$1"

  # if there's a second argument, it's the profile
  if [ -n "$2" ]; then
   local profile="$2"
  fi

 elif [ -n "$1" ]; then

  # if the first argument doesn't contain "https://", it's the profile
  local profile="$1"
 fi

  echo "$1 $2"

}

# define default vals and update based on provided args
update__profile_url() {
 local endpoint_url="https://kalousis.s3.unige.ch"
 local profile="default"

 # update `endpoint_url` and `profile` if provided
 if [[ "$2" == https://* ]]; then
  read endpoint_url profile <<< $(get__profile_endpoint_url "$2" "$3")
 else
  read profile <<< $(get__profile_endpoint_url "$2" "$3") # for `
     endpoint_url`, default val will be taken
 fi

 echo "$endpoint_url $profile"
}

# listing
# example usages (only bucket name provided):
# `lal path`
# `lal path default`
# `lal path https://kalousis.s3.unige.ch`
# `lal path https://kalousis.s3.unige.ch default`
lal() {
 local path="$1"

 read endpoint_url profile <<< $(update__profile_url "$2" "$3")

```

```
135     $(which aws) s3 ls s3://"$path" --recursive --endpoint-url "
            $endpoint_url" --profile "$profile"
136   }
137
138   # removing prefixes/files
139   # example usages (only bucket name provided):
140   # `larm path`
141   # `larm path default`
142   # `larm path https://kalousis.s3.unige.ch`
143   # `larm path https://kalousis.s3.unige.ch default`
144   larm() {
145     local path="$1"
146
147     read endpoint_url profile <<< $(update__profile_url "$2" "$3")
148
149     command_output=$($(which aws) s3 rm s3://"$path" --recursive --endpoint
            -url "$endpoint_url" --profile "$profile")
150
151     if [[ -z "$command_output" ]]; then
152     # no use of `--recursive`, which shouldn't be used for single file
            deletion
153       $(which aws) s3 rm s3://"$path" --endpoint-url "$endpoint_url" --
            profile "$profile"
154     fi
155   }
156
157   # ------------------------------ GIT ------------------------------
158
159   # list all local and remote branches
160   lb() {
161     git branch -a
162   }
163
164   # create remote branch
165   # usage:
166   # `lbc new-branch`
167   lbc() {
168     local branch_name="$1"
169
170     git branch $(branch_name) && git push origin $(branch_name)
171   }
172
173   # delete remote branch
174   lbd() {
175     local branch="$1"
176     local exists__in_local=$(git branch --list "$branch")
177
178     if [[ -z ${exists__in_local} ]]; then
179       git push origin --delete "$branch" && git clean -f
```

```
 else
  git branch -D "$branch"
 fi

}

# switch branches and create if non-existent
lsw() {
 if git rev-parse --verify "$1" >/dev/null 2>&1; then
   git switch "$1"
 else
   git switch -c "$1" && git push origin $(bname)
 fi
}

# cloning
# example usage:
# `lcl git@github.com:ImahnShekhzadeh/infra.git `
# `lcl git@github.com:ImahnShekhzadeh/infra.git infra`
# `lcl git@github.com:ImahnShekhzadeh/infra.git infra main`
lcl() {
 local dir_name="${2:-$(pwd)}"
 local branch_name="${3:-main}"

 git clone "$1" "$dir_name" && cd "$dir_name" && lsw "$branch_name"
}

# example usage: `lsta 2` or `lsta`
lsta() {
 local stash_index=${1:-0} # Default to 0 if no argument provided

 # Check if the provided argument is an integer
 if ! [[ $stash_index =~ ^[0-9]+$ ]]; then
  echo "The provided index is not a valid integer."
  return 1
 fi

 # Check if the stash index exists
 if ! git rev-parse --verify stash@{$stash_index} >/dev/null 2>&1; then
  echo "No stash found at index $stash_index"
  return 1
 fi

 # If all checks pass, apply the stash
 git stash apply "stash@{$stash_index}" --index
}

# Forward commands to `git stash`
lst() {
```

```
229  git stash "$@"
230 }
231
232 # Stash files, if arguments are provided, they are ignored
233 lstf() {
234  git stash --include-untracked
235 }
236
237 # https://stackoverflow.com/questions/19595067/git-add-commit-and-push-
       commands-in-one
238 # https://stackoverflow.com/questions/14763608/use-conditional-in-bash-
       script-to-check-string-argument
239 # if-else statements in bash: https://linuxhandbook.com/if-else-bash/
240 # example usage: lgit "bit" "add ..."
241 lpush() {
242
243  (
244  # use subshell to change directory to Git root and perform actions
245  cd "$(git rev-parse --show-toplevel)" || exit
246  git add . && git commit -a -m "$1" && git push origin $(bname) && llog
247  )
248
249 }
250
251
252 # https://stackoverflow.com/questions/3236871/how-to-return-a-string-
       value-from-a-bash-function
253 bname() {
254  branch=$(git branch --show-current)
255  echo $branch
256 }
257
258 lupd() {
259  git fetch origin $(bname) && git log HEAD..origin/$(bname) --oneline
260 }
261
262 lpull() {
263  git pull origin $(bname)
264 }
265
266 ldiff() {
267  git status "$@" && git diff --color "$@"
268 }
269
270 lforce() {
271  git push origin $(bname) --force
272 }
273
274 llog() {
```

```
 git log
}

lrm() {
 git rm -r "$@"
}

lreb() {
 # Set default value to 5:
 num1=${1:-5}
 git rebase -i HEAD~$num1
}

# Reset entire repo to state of `HEAD`, or reset specific file to a
    specific commit hash.
lres() {
 if [[ $# -eq 0 ]] || [[ $# -eq 1 ]]; then
  local commit_hash=${1:-HEAD}
  git reset --hard "$commit_hash"
 elif [ $# -eq 2 ]; then
  local commit_hash="$1"
  local file_path="$2"
  git restore --source="$commit_hash" "$file_path"
 else
  echo "Usage: lres [commit_hash file_path]"
 fi
}

lsh(){
 git show "$@"
}

# usage:
# `lmv file1 file2 file3 [...] /path/to/target_dir`
# `lmv source_dir target_dir`
lmv() {
  # convert the arguments to an array
  local args=("$@")

  # get the last element of the array
  local target_dir="${args[-1]}"

  if [[ ! -d "$target_dir" ]]; then
   mkdir -p "$target_dir" || {
     echo "Target directory does not exist: $target_dir" >&2
     return 1
   }
  fi

```

```
323     # shift the arguments so that last argument (`target_dir`) is dropped
324     unset args[-1]
325 
326     # now, loop through all the remaining arguments
327     for path in "${args[*]}"; do
328       if [[ -e "$path" ]]; then
329         if [[ -d "$path" ]]; then
330           # move each item in the directory individually
331           for item in "$path"/*; do
332            git mv "$item" "$target_dir"
333           done
334 
335           # remove the now-empty source dir
336           rmdir "$path"
337         else
338           # if it's a file, just move it
339           git mv "$path" "$target_dir"
340         fi
341       else
342         echo "File does not exist: $path" >&2
343       fi
344     done
345   }
346 
347   # ---------------------------- PROTONVPN ----------------------
348 
349   p() {
350    protonvpn-cli "$@"
351   }
352 
353   # ---------------------------- MISCELLANEOUS ------------------
354 
355   # pdflatex
356   pd() {
357    /usr/bin/pdflatex "$@"
358   }
359 
360   # convert input notebook to PDF
361   jconv() {
362    jupyter nbconvert --to pdf "$1"
363   }
364 
365   # `less` with ANSI escape characters
366   less() {
367    /usr/bin/less -R "$@"
368   }
369 
370   diff() {
371    /usr/bin/diff --color "$@"
```

```
}

# overload `shred` func, allow (recursive) shredding of dirs/files
# multiple files/dirs can be provided, mixing allowed
# usage (e.g.): `shred 10 <file_name>`
# shred <file_name>
# shred <dir_path>
# shred <file_name> <dir_path>
shred() {

 # check whether first argument is a number
 if [[ "$1" =~ ^[0-9]+$ ]]; then
  local iterations="$1"
  shift
 else
  iterations=5 # default
 fi

 # check file/dir existences
 for path in "$@"; do
  if check_existence "$path"; then
  # check whether passed input is directory or not
   if [[ -d "$path" ]]; then
    echo "Files to be shredded in $path:"
    find "$path" -type f -print0 | xargs -0 ls -ld
   fi
  else
   echo "Error occurred in check_existence for file/dir: $path"
   return 1
  fi
 done

 # prompt user for confirmation
 read -rp "Do you wish to proceed with shredding all files in $@ for
     $iterations iterations? (yes/no): " confirmation

 if [[ $confirmation = [yY] || $confirmation = [yY][eE][sS] ]]; then
  for path in "$@"; do
   if [[ -d "$path" ]]; then
    # shred all files within the directory
    find "$path" -type f -exec /usr/bin/shred -uz -n "$iterations" {} \;
    rm -rf "$path"
    echo "All files in '$path' have been shredded for $iterations
        iterations."
   elif [[ -f "$path" ]]; then
    # shred the individual file
    /usr/bin/shred -uz -n "$iterations" "$path"
    echo "File '$path' has been shredded for $iterations iterations."
   fi
```

```
  done
 else
  echo "Shredding aborted."
 fi
}

# shortcut for clearing terminal output
c() {
 clear
}

# shortcuts for exiting terminal
q() {
 exit
}

e() {
 q
}

# tailscale
ts() {
 tailscale status "$@"
}

# xournalpp (https://github.com/xournalpp/xournalpp)
xopp() {
 xournalpp "$@"
}

# strings comparison
# usage (e.g.): `str_diff "blub1" "blub1"` or `str_diff blub1 blub1`
# or `str_diff $(echo "hey") $(echo "hey")`
# NOTE: exactly two arguments need to be provided
str_diff() {

 # check number of passed arguments via `$#`
 if [[ $# -ne 2 ]]; then
  echo "NOTE: Exactly two arguments need to be provided"
  return 1 # return non-zero exit code to indicate error
 else

  # compare strings
  if [[ $1 == $2 ]]; then
   echo -e "Strings '$1' and '$2' \033[92mmatch\033[0m"
  else
   echo -e "Strings '$1' and '$2' do \033[91mNOT\033[0m match"
  fi
 fi
```

```
}


# ------------------------------ DOCKER -----------------------
d() {
 docker "$@"
}

# ------------------------------ CHATGPT -----------------------

# https://github.com/kardolus/chatgpt-cli/tree/main
gpt(){
 chatgpt -i
}

export OPENAI_KEY=[...]

# ------------------------------ ALWAYS EXECUTE ----------------

add_bit
```

# B. Amazing Programs, Extensions, Plugins & Packages

- **https://etherpad.org/**
- **https://github.com/charmbracelet/glow**
- **https://github.com/0xacx/chatGPT-shell-cli**
- **https://github.com/kardolus/chatgpt-cli/tree/main**
  - For setting the right model (cf. here for all available models),

```
chatgpt --set-model gpt-4-1106-preview --set-max-tokens 128000
```

  - Usage:

```
chatgpt -i
```

- **https://tailscale.com/download/**
  - Once installation is complete, the command

```
sudo tailscale up
```

    should be run to login, though this command will also display after installation in the CLI. The signing in should happen via GitHub. To be able to use Tailscale from a new device, it must be added as a device under `https://login.tailscale.com/admin/machines`. Once this is done, open a CLI and type

```
ssh name@ip_address # find out <name> and <ip_address> via tailscale console
# ssh ellie@100.xx.xxx.xx
```

    NOTE that if the file already exists locally, it will be overwritten.
  - For file copying (e.g. from the host machine to the currently used machine), do this

```
scp name@ip_address:/path/to/remote_file.ext /local/path # find out <name> and <ip_address> via tailscale console
# ssh ellie@100.xx.xxx.xx
```

    For directory copying,

```
scp -r name@ip_address:/path/to/remote_dir /local/path # find out <name> and <ip_address> via tailscale console
# ssh ellie@100.xx.xxx.xx
```

- `https://tailscale.com/kb/1080/cli/` (no separate installation necessary, only tailscale needs to be installed)
  - Finding out the IPv4 address of the currently active machine,

```
tailscale ip -4
```

  - Finding out the IPv4 address of another machine connected via the Tailscale network,

```
tailscale ip -4 custom-name
# tailscale ip -4 ellie
```

- `https://github.com/aws/aws-cli`
- `https://github.com/termcolor/termcolor`
- **LibreOffice dark theme,**

  `Tools` → `Options` → `LibreOffice` → `Application Colors` → `Custom Colors` → `General` → `Document Background`, choose a dark color.

# C. Opening Programs from the CLI in Linux

- Opening the settings from CLI,

```
gnome-control-center
```

- Opening VSCode from CLI:

```
code path_to_file/file_name.ext
```

If a VSCode editor is already open, use the *-n* flag to open the file in a new editor:

```
code -n path_to_file/file_name.ext
```

A folder can also be opened directly:

```
code path_to_dir
```

Listing C.1: Opening VSCode dir from CLI

- Opening LibreOffice from CLI:

```
libreoffice --writer path_to_dir/filename.odt
```

- Opening an image via the CLI:

```
eog /path/to/your/image.jpg
```

# D. VSCode

## D.1. Recommended Extensions

- `https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remot`
- `https://marketplace.visualstudio.com/items?itemName=Gruntfuggly.todo-tree`

## D.2. Debugging

- Stepping into external code with Python debugger, tutorial here
- Creating a JSON file, here some instructions

## D.3. `settings.json`

Opening the file,

1. press `Ctrl + Shift + P` to the Command Palette,
2. type `Open User Settings (JSON)` and select it to open the `settings.json` file.

## D.4. Fix Unresolved Python Imports

- If you run a docker container where a conda environment is installed (with packages that you do not have locally), then VSCode will show those imports as unresolved. To fix this, open the `settings.json` file, cf. App. D.3, and add the following setting:

```
"python.analysis.diagnosticSeverityOverrides": {
    "reportMissingImports": "none"
}
```

  Incorporating this into the `settings.json` file is shown in App. D.6.

- Note that if you have an SSH connection to another machine going on, e.g. in the `Remote Development` extension, putting the above lines into the `settings.json` file will not have an immediate effect, for this the SSH connection needs to be restarted.

## D.5. Opening a Duplicate Workspace

1. press `Ctrl + Shift + P` to open the Command Palette,
2. then type `Workspaces: Duplicate As Workspace is New Window`

## D.6. `settings.json`

Contents of `settings.json`,

```
{
 "workbench.colorTheme": "Default Dark Modern",
 "telemetry.telemetryLevel": "off",
 "editor.wordWrap": "wordWrapColumn",
 "editor.wordWrapColumn": 79,
 "workbench.editor.enablePreview": false,
 "gitlens.telemetry.enabled": false,
 "notebook.lineNumbers": "on",
 "explorer.confirmDragAndDrop": false,
 "window.zoomLevel": 1,
 "python.analysis.diagnosticSeverityOverrides": {
    "reportMissingImports": "none"
 },
 "todo-tree.general.tags": [
    "BUG",
    "HACK",
    "FIXME",
    "TODO",
    "NOTE",
    "XXX",
    "[ ]",
    "[x]"
 ],

 "files.associations": {"*.log": "plaintext"},
 "[plaintext]": {"editor.wordWrap": "off"},
 "[shellscript]": {"editor.wordWrap": "off"},
 "workbench.editor.tabSizing": "shrink"
}
```