Infra

Imahn Shekhzadeh
imahn.shekhzadeh@unige.ch

December 6, 2023

Contents

Note

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

1 Baobab/Yggdrasil

- To connect to Baobab from your local machine, just type into a terminal:

```
eval $(ssh-agent)
ssh-add /home/imahn/.ssh/id_ed25519_unige_hpc
ssh shekhza2@login2.baobab.hpc.unige.ch
# ssh shekhza2@login1.yggdrasil.hpc.unige.ch
```

Listing 1: ssh

- To mount your scratch directory to a local folder do

```
mkdir /home/imahn/baobab # if non-existent
sshfs shekhza2@login2.baobab.hpc.unige.ch:/home/users/s/shekhza2/ /home/
    imahn/baobab/
```

Listing 2: Local mounting

- Scp into yggdrasil:

```
scp file shekhza2@login1.yggdrasil.hpc.unige.ch:/home/users/s/shekhza2/
# scp -r folder_name shekhza2@login1.yggdrasil.hpc.unige.ch:/home/users/s/
    shekhza2/
```

Listing 3: Scp file (folder) onto cluster

- With

  htop

  you can see the login resource consumption for all users.

- If you did an 'sshfs' and the connection hung up, type

  fusermount -zu /home/imahn/folder_name

- To see all the machines that are occupied, just type

```
squeue
squeue -p cms-uhh # partition
squeue -u shekhza2 # user
```

Listing 4: Squeue commands

- To find out about your conda environment, just type (e.g. whether you use Anaconda2 or Anaconda3)

```
conda info
```

- To export an yml-file to share it with others, type

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

  At the end of the file, there will be a line starting with "Prefix:", you can safely delete it, for details see here

-

```
pip install ipykernel
python -m ipykernel install --user --name <environment_name> --display-name
    "customStuff"
```

2 Bash

- Downloading file from URL and allowing for redirects,

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

- For this directory structure,

```
infra_upd.tex
infra_upd.log
infra_upd.aux
infra_upd.out
infra_upd.pdf
```

Listing 5: Example directory structure

rename *all* of them via

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

What happens is called a substring replacement.

- Appending line to file,

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

- Checking whether provided string (e.g. via an argument) is empty or not (typically used within conditional statements):

```
test_sth() {
 local env_name="$1" # bash starts counting indices from 1

 if [ -z "$env_name" ]; then # spacing after `[` and before `]` needed
 echo "The string is empty."
 return 1 # return value of 1 indicates error
 fi
}
```

Listing 6: Check (e.g. in if-clause) whether string is empty or not

- For retrieving all but the first argument,

```
test_sth(){
 shift

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

- And of course there is nothing stopping us from doing this $N \geq 1$-times ... Pseudocode:

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

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

If N arguments are not provided, this is **not** a problem, the code will still run through.

- Example for an alias:

```
# forward output
ts(){
 test_sth "$@"
}
```

- Finding out the size of a file or 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}'
```

- When you want to create a new directory and you want all parent directories to be created as well (assuming they don't already exist), do

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

The *-p* option is safe, since if the directory is already existent, no error will be outputted

- Searching for all files with a specific extension, e.g. *.ext*:

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

Note that this can be nicely combined with *grep*.

- In Bash, using [[ ]] instead of [ ] is preferred, since [[ ]] is safer and more capable within Bash scripts. Also, within [ ] (where word splitting and filename expansion do occur), it's good practice to double-quote your variables. But it's safe to omit the double-quotes for e.g. $# within [[ ]].

- 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
```

3 Linux

- Under Ubuntu, listing *all* available kernels,

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

Listing 7: Find kernel versions in Ubuntu

Finding the currently *active* kernel version,

```
uname -a
```

Listing 8: Current kerel versions in Ubuntu

The *-a* option stands for *appending*, otherwise *tee* overwrites *output.out* (if existent).

- Better colors in CLI:

  1. Use 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 *-h* stands for human readability and *-s* for summarizing.

- Retrieving the number of available CPU resources,

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

- It is possible to use colored outputs in Bash. Check the bash function *str_diff* in App. A. (Note that the *-e* option is mandatory to enable interpretation of the backslash escapes).

- Print day and time from CLI,

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

3.1 Opening Programs from CLI

- 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 9: 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
```

4 Anaconda

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 Installation & Removal of Packages

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

```
pip install -e .
```

- 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
```

4.3 Usage in VSCode

- Selecting a conda environment in VSCode, do Ctrl + Shift + P and type *Python: select interpreter*.
- Stepping into external code with Python debugger: `https://stackoverflow.com/questions/53594900/visual-studio-code-python-debugging-step-into-the-code-of-external-function`
- Creating a JSON file, here some instructions: `https://code.visualstudio.com/docs/python/debugging`
- Listing all installed environments,

```
conda env list
```

4.4 PyTorch

- Checking whether gpu version of PyTorch is installed, from python shell (**for this, activate the right conda env first!**):

```
import os

import torch

if __name__ == "__main__":
 os.path.dirname(torch.__file__)
```

Afterwards, do

```
ls -larht <path_from_prev_alg> | grep -E "cuda"
```

- If you had installed PyTorch via conda instead of pip, then this is easier, where the *-E* means we are searching for extended regular expressions (**again activate the right conda env first!**):

```
conda list | grep -E "torch|pytorch"
# or `conda list | grep -E "torch|pytorch"`
```

5 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 `https://docs.nvidia.com/cuda/cuda-installation-index.html` again.
  6. For PyTorch to recognize the GPU, I had to reboot.

6 Docker

6.1 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
```

6.2 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
```

6.3 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 *Image_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).

6.4 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
```

6.5 Docker containers

6.5.1 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.

6.5.2 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
```

6.5.3 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
```

6.6 Pushing to DockerHub & HPC

1. First login to Docker via

```
d login -u user_name -p password
```

2. Then follow the instructions from this tutorial (from minute 17:05 on)
3. And then follow the HPC tutorial from UNIGE

6.7 Git

- This SO post provides an excellent way of using git in docker. And here an example of using my personal git ssh keys:

```
d build -t "ubuntu_octave:latest" -f Dockerfile_git --build-arg ssh_prv_key
   ="$(cat ~/.ssh/id_ed25519_github)" --build-arg ssh_pub_key="$(cat ~/.
   ssh/id_ed25519_github.pub)" --squash .
```

- Actual cloning of repo should be done inside docker container instead of docker image

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
```

If you have several notebooks in the same directory,

```
for nb in *.ipynb
do
 jupyter nbconvert --to pdf "$nb"
done
```

8 AWS S3

8.1 Installation & Configuration

1. Installation instructions: `https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html#getting-started-install-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.

8.2 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
```

```
8
9   [profile2]
10  aws_access_key_id = YET_ANOTHER_ACCESS_KEY_ID
11  aws_secret_access_key = YET_ANOTHER_SECRET_ACCESS_KEY
```

To use a specific profile when running *aws cli* commands, you can use the *–profile* option in the command:

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

8.3 Buckets

One can have several buckets.

8.3.1 Creation

- To create a new bucket:

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

8.3.2 Listings

- Directly showing the file contents of an s3 bucket,

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

8.3.3 File Copying

- Local machine ⟶ S3:

```
1   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:

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

8.3.4 Directory Copying

- Local machine ⟶ S3:

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

8.3.5 Folder/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
```

8.4 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()
```

9 Git

9.1 Renaming a Repository

1. Rename the repository remotely first (by going to your repository's URL),
2. then go to the locally cloned version of the repository and do

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

3. and finally

```
git remote -v
```

   which lists the remote names and their URLs.

9.2 Restore File

To reset a specific file to the state of a previous commit in Git, you can use the *git restore* command:

```
git restore --source=<commit-hash> <file_path>
# git restore --source=HEAD <file_path>
```

Replace *<commit-hash>* with the commit SHA and *<file-path>* with the path to the file. After doing this, the file will be in the state it was in at the specified commit. This change is local and you would need to commit it if you want it to be reflected in the repository history.

10 Remote Development

10.1 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.

10.2 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
```

The *custom-username* here refers to the username on the remote machine.

11 Jax

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

- To put a Jax array onto a specific device, use this:

```
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()}")
```

Listing 10: Device specification in Jax

- Specifying the dtype of an array:

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

Listing 11: Jax device retrieval

- To find out the device of a Jax array, use this:

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

Listing 12: Jax device retrieval

- To make a Jax array out of a Python list or a NumPy array (do not use for PyTorch tensors):

```
from jax import numpy as jnp

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

Listing 13: Jax array creation

- jit (just-in-time compilation): sets up a function with XLA (extended linear algebra): check out the NB *test___jit-compil.ipynb*. To use jit, do this:

```
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))
```

Listing 14: Jax array

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() {

 # check number of passed arguments via `$#`
 if [[ $# -ne 1 ]]; then
  echo "NOTE: Exactly one argument needs to be provided"
 else
  conda deactivate && conda remove -n "$1" --all -y
 fi

```

```
}

# alias for `conda__remove_packages`
# usage (e.g.): `crm myenv pkg1 pkg2`
crm() {
 conda__remove_packages "$@"
}

# remove conda packages from environment
# usage (e.g.): `conda__remove_packages myenv pkg1 pkg2`
conda__remove_packages() {

 # define local variables first
 local env_name="$1"
 local conda_out="$(conda env list | grep -E "$env_name" | head -n 1 | awk '{
    print $1}')"

 # forget first argument (which is saved in `env_name`)
 shift

 # check non-emptiness
 if [ -z "$env_name" ]; then
  echo "Usage: conda__remove_packages <env_name> [package1] [package2] ... [
     packageN]"
  return 1
 fi

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

}

# ------------------------------ GIT ------------------------------

# 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() {
 git stash "$@"
}

# Stash files, if arguments are provided, they are ignored
lstf() {
 git stash --include-untracked
}

# https://stackoverflow.com/questions/19595067/git-add-commit-and-push-commands-in-one
# https://stackoverflow.com/questions/14763608/use-conditional-in-bash-script-to-check-string-argument
# if-else statements in bash: https://linuxhandbook.com/if-else-bash/
# example usage: lgit "bit" "add ..."
lpush() {
 git add . && git commit -a -m "$1" && git push origin $(bname) && llog
}

# https://stackoverflow.com/questions/3236871/how-to-return-a-string-value-from-a-bash-function
bname() {
 branch=$(git branch --show-current)
 echo $branch
}

lupd() {
 git fetch origin $(bname) && git log HEAD..origin/$(bname) --oneline
}

lpull() {
 git pull origin $(bname)
}

ldiff() {
 git status "$@" && git diff --color "$@"
}

lforce() {
 git push origin $(bname) --force
}

```

```
147 llog() {
148  git log
149 }
150
151 lrm() {
152  git rm -r "$@"
153 }
154
155 lreb() {
156  # Set default value to 5:
157  num1=${1:-5}
158  git rebase -i HEAD~$num1
159 }
160
161 # Reset entire repo to state of `HEAD`, or reset specific file to a specific
      commit hash.
162 lres() {
163  if [ $# -eq 0 ]; then
164   git reset --hard HEAD
165  elif [ $# -eq 2 ]; then
166   local commit_hash="$1"
167   local file_path="$2"
168   git restore --source="$commit_hash" "$file_path"
169  else
170   echo "Usage: lres [commit_hash file_path]"
171  fi
172 }
173
174 lsh(){
175  git show "$@"
176 }
177
178 lm() {
179  git mv "$@"
180 }
181
182 # ---------------------------- PROTONVPN ----------------------
183
184 p() {
185  protonvpn-cli "$@"
186 }
187
188 # ---------------------------- MISCELLANEOUS ------------------
189
190 # convert input notebook to PDF
191 jconv() {
192  jupyter nbconvert --to pdf "$1"
193 }
194
195 # `less` with ANSI escape characters
196 less() {
197  /usr/bin/less -R "$@"
```

```
}

diff() {
 /usr/bin/diff --color "$@"
}

# overload `shred` func
# usage (e.g.): shred 10 <file_name>
# shred <file_name>
shred() {

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

 /usr/bin/shred -uz -n "$iterations" "$@"
}

# 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/0xacx/chatGPT-shell-cli
gpt(){
 chatgpt --model gpt-4
}

export OPENAI_KEY=[...]

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

add_bit
```

Listing 15: Contents of .bashrc file

B Amazing Programs, Extensions, Plugins & Packages

- **https://github.com/charmbracelet/glow**
- **https://github.com/0xacx/chatGPT-shell-cli**
  - Prerequisite: `https://jqlang.github.io/jq/` (for download cf. `https://jqlang.github.io/jq/download/`)
- **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`

C VSCode

C.1 Recommended Extensions

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

C.2 Open the *settings.json* 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.

C.3 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**, cf. App. C.2, and add the following setting:

```
1    "python.analysis.diagnosticSeverityOverrides": {
2        "reportMissingImports": "none"
3    }
```

  How to incorporate this into the **settings.json** file is shown in App. C.5.

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

C.4 Opening a Duplicate Workspace

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

C.5 *settings.json*

```
1    {
2     "workbench.colorTheme": "Default Dark Modern",
3     "telemetry.telemetryLevel": "off",
4     "editor.wordWrap": "wordWrapColumn",
5     "editor.wordWrapColumn": 79,
6     "workbench.editor.enablePreview": false,
7     "gitlens.telemetry.enabled": false,
8     "notebook.lineNumbers": "on",
9     "explorer.confirmDragAndDrop": false,
10    "window.zoomLevel": 1,
11    "python.analysis.diagnosticSeverityOverrides": {
12        "reportMissingImports": "none"
13     },
14     "todo-tree.general.tags": [
15        "BUG",
16        "HACK",
```

```
        "FIXME",
        "TODO",
        "NOTE",
        "XXX",
        "[ ]",
        "[x]"
    ],

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

Listing 16: Contents of settings.json file

D LibreOffice

D.1 Dark Theme

Go to *Tools* → *Options* → *LibreOffice* → *Application Colors* → *Custom Colors* → *General* → *Document Background* and choose a dark color.

E UNIGE-specific

- Setting up UNIGE e-mail in TB:
  `https://plone.unige.ch/distic/pub/messagerie/configuration/comment-configurer-compte-`