Inhalt

1. Bash

1.1. Commands

- . is pwd, .. is parent folder, ~ is home directory
- code -r <file>: to open vscode editor instead of something like nano

ls -la	Lists all files & directories in this path - l: long form - a: show hidden files
rm <file> or rm -r <directory>	Removes file. -r means recursive
touch <file>	Creates a file
mkdir <name>	Creates a directory
mv <file> <destination, file name included>	Move file to another location or change name
cp <file> <destination, file name included> cp -R <dir> <copy dir>	Copies file <copy dir> need to exist first before doing cp
pwd	Prints working directory
cd <directory>	Changes directory
wc <file>	counts words in file
man	Opens manual
grep “<word>” <file> • grep -win -C 2 “test” test.txt • grep -winr “test” . • grep -wirl “tets” . • grep -win “test” ./*.txt • history | grep “works with pipe” | <…> • grep “^mülle*r\> ” names.txt • grep “…-…-…“ numbers.txt or: grep -P “\d{3}-\d{3}-\d{4}“ numbers.txt => -P for Perl Compatible Regular Expression which is the standard for programming languages	Searches for word in file - w: matches only whole words - i: not case sensitive - n: get line number - C: get context 2 lines before & after match - r: recursive checks subfolders without -r error - l: only shows in which files the matches are - c: counts number of matches - ./* searches for txt files in this folder - ^: how to start - *: how to end - \>: what should follow - /S: no space character, includes multiple signs - dot(.): any character
find <dir> <parameters> • find -name “*.pyc” -exec chmod 775 {} + • find . -type f -name “*.pyc” -exec rm {} +	Searches for files/dirs, default for dir is dot(.). -exec: to use commands for the results - {}: placeholder for results - +: to end command -type f: type should be file
cat	Shows content of file
less <file>	Opens text files
dig <dns>	Figure out ip address
dig MX <dns>	Figure out mail server
nano <file>	Edit file
sudo apt-get update	Updates list of available upgrades
sudo apt-get upgrade	Upgrades list of available upgrades for packages
source <file>	Reads and executes file
chmod <mode> <directory/file>	Change access privileges
sudo <rest>	Execute a command as admin
ssh <ip address>	Build connection with ssh
ssh-copy-id <user>@<ip address>	Build connection & will put id_rsa.pub & file authorized keys automatically

scp <file> <remote server>@<ip address>:<location>	Copy a file from local machine to ssh remote server. If location is empty default is ~
redirect: ls -l *.pdf > <file>	> means redirect output if ls to file
ls -l *.pdf | wc	| (pipe) means output of ls is input of wc
rsync -av original/ backup/ remote: rsync -zaP original/ <username>@<ip address>:<location>	Synchs folders, this also works with remote server with ssh. The slash after dir is important otherwise the original dir is included, except that's what you want - a: restores permissions & stuff and includes -r for recursive - v: verbose just prints synced files & dirs - z: compresses data - P: shows progress over time

1.2. Cron Jobs – schedule commands

- Ubuntu on windows 10 these line are necessary in home dir(~):
  - sudo usermod -a -G crontab "username"
  - sudo cron

crontab -l	lists all current users cron job
crontab -e	create a cron job

Format: interpreted as time, not duration. E.g.: minute=30 => 00:30, 01:30, 02:30 and so on…
<minute> <hour> <day of month> <month> <day of week> <commands to execute>

Ranges: or use * for every value
<0-59>   <0-23>   <1-31>   <1-12>   <0-6 from Sunday to Saturday> <command>

e.g. simplest job: Always give absolute path
* * * * * echo 'hello' >> <absolute path>/test.txt

Set multiple arguments with comma: e.g.: 15,45 * * * * <command>

Set intervals with */: e.g.: */10 * * * * <command> => runs every 10min

Explicit ranges with dash: e.g.: 0-5 * * * * <command> => runs minutes from 0 to 5

2. Git

- Note: always commit and then push
- Note: always pull before push except creating a new branch and push

2.1. Setup git

Set Config Values
git config --global user.name “<name>”
git config --global user.email “<email>”
git config --list

Initialize Project – Local Repository	
git init	Initialize git repo in current folder
git status	check status of repo
touch .gitignore	creates a git ignore file All files in the git ignore file will not be tracked from git

File Control	
git add -A git add <file>	Add files to staging area => -A means all files in repo sub & upper dirs. -A <dirname> only does in this dir & subdirs => -u only stages for tracked files, no new files in entire tree. Specify dir with -u <dirname> => a Point(.) is like git add -A . and would skip upper dirs Specifying a file also possible
git reset git reset <file>	In both cases from the last change Remove all files from staging area Remove one file from staging area
git commit -m “<message>”	Commit files, -m for the message
git log	Returns information about commits

2.2. Remote Repository - clone

Clone Note: git init not necessary	
git clone <url> <where to clone>	Clone repo from GitHub
git remote -v git branch -a	get information about repo, -a means list all
git diff	Shows changes

Create Branch	
git branch <name of branch>	Clone repo first and then create a branch, <name of branch> should be the feature which will be added.
git branch	Lists all branches, the asterisk(*) shows current branch
git checkout <name of branch>	Switch to branch <name of branch>

Push Branch to Remote Repo

git push -u origin <name of branch>	Push branch to remote repository, pull not necessary. -u coordinates the local and server repo according to the branch
<u>Merge a Branch</u>	
git checkout <master or main>	Always go to master/main
git pull origin <master or main>	
git branch --merged	Lists all merged branches
git merge <name of branch>	Merge branch to master/main
git push origin <master/main>	Push to remote repo
<u>Remove Branch after pushed to remote repo</u>	
git branch -d <name of branch> git push origin --delete <name of branch>	Delete branch locally & Delete on remote repo

2.3. Github

<u>Clone repo from Github from</u> git init not necessary will be done automatically
git clone <url>
then do add & commit or a branch
git pull origin main
git push origin main

<u>Pull GitHub repo & push your files to Github from scratch with remote</u> remote: does not copy repo to your local machine just set the link
git init
git branch -m <new name for branch> => change name to main due to GitHub Note: sometimes this does not work until something was commited
touch .gitignore => add files or directories
git add -A
git commit -m <message>
git remote add origin <GitHub URL>.git
if repo on Github is empty this step not necessary and not intended with remote. Cloning repo right at the beginning better but this way works too git pull origin <name of branch> --allow-unrelated-histories
git push origin <name of branch>

<u>Push to GitHub repo from already pulled GitHub repo</u>
then do git add & git commit
git pull origin main
git push origin main

2.4. Advanced edits to undo mistakes

- Delete changes: if code has been changed but not added, this will delete it
  - git checkout <file>

Edit commit due to a mistake	
	--amend will change the git history, thus never use it if this commit was pushed to others. Only use it if it only affects you

git commit --amend -m "<new message>" git commit --amend -m	When message of last commit was wrong, this will update this This command open text file which can be manipulated. It will also automatically add files which might was forgotten to commit
Get hash of commit git checkout <correct branch> git cherry-pick <hash from commit>	This will commit the commit to this branch, but does not remove it from the other branch like master
git reset --hard <hash of commit> git reset <hash of commit> git reset => will delete everything in stage area	Will delete the commit Default soft-mixed will remove it from final state but keep it in staging area
git clean –df	Removes untracked directories(d) and files & forced(f). This will happen when doing a hard reset

Restore branch state	
1. git reflog	Is like a garbage collector and lists everything but only for like a month
2. get hash before reset	
3. git checkout <hash>	Go to the state of the branch before reset
4. git branch <name e.g. backup>	The branch will be deleted therefore we need to save it with a new branch
Revert Commits which were already pulled => history will not be corrupted	
git revert <hash of commit>	Will revert the commit
git diff <hash of original commit> <hash of revert commit>	Shows differences between those commits

2.5. git stash

git stash => save temporary a state of branch They come in handy when switching branches and want code to be committed to another branch it will automatically update the file(s) and merges original with stash	
git stash save "message"	Save state of branch (not sure if it must not be committed before) and go back to original state
git stash list	Lists all stashes
git stash apply <stash-code>	Go back to stash state with the stash code which can be found in: git stash list
git stash pop	Applies the same command as the git stash apply but it will take the first item in the stash list and will remove it too
git stash drop <stash-code>	Will remove the stash & return to original state if current state was the stash which was deleted
Git stash clear	Removes all stashes

3. Create Virtual Environment

3.1. venv

- Name virtual environment venv, in good practice <folder_name> means venv
- Don’t put project files into venv

Linux	
python3 -m venv <folder_name>	creates a virtual environment
source <folder_name>/bin/activate	activates venv in terminal
deactivate	deactivates venv
rm -rf <folder_name>/	delete venv

Windows	
python -m venv <folder_name>	creates a virtual environment
<folder_name>\Scripts\activate.bat	activates venv in terminal
deactivate	deactivates venv
rmdir <folder_name> /s	delete venv & /s for deleting sub directories etc.

4. Pip commands

pip help	lists commands
pip help <keyword e.g. install>	lists parameters of this command
pip list	lists all installed packages
pip install <package>	installs package
pip uninstall <package>	uninstalls package
pip install -U <package>	update package
pip freeze > requirements.txt	list all packages in a text file with the version
pip install -r requirements.txt	installs all packages & version inside the text file
pip show <package>	shows info about package & which python

5. Pipenv

pipenv install <package>	automatically creates venv in this folder if no venv exists and installs package
pipenv shell	activate venv
exit	deactivate venv
pipenv run <command e.g. python>	run commands in venv terminal
pipenv install -r <requirement.txt>	installs all packages in the file
pipenv lock -r	list all packages
pipenv install <package> --dev	install only in YOUR venv not when “shipped” to other systems with pipenv lock -r
pipenv uninstall <package>	uninstall package
1. go to file and set python version to want you want 2. pipenv --python <version> or 1 go to file and set python version to want you want 2 pipenv –rm 3 pipenv install	change python version of venv
pipenv --rm	remove venv
pipenv check	check for security issues in the packages
1. change version in venv file 2. pipenv install	change version of a package
pipenv graph	lists dependencies of packages
1. pipenv lock 2. pipenv install --ignore-pipfile	to update pipfile.lock and push finished project into “production”
1. create .env file 2. add env variables 3. pipenv run python 4. to check it worked: import os os.environ[“<name of variable>”]	add secret env variables in venv

6. Environment Variables

6.1. Linux

python script	
Import os db_user = os.environ.get(<DB_USER>) # environ is a dictionary of the environment variables	
terminal	
nano ~/.bashrc	to open text editor for this file
export <DB_USER>=”<username>” export <DB_PASS>=”<password>”	create environment variables

6.2. Windows

- Create in user system variables

7. Commands

7.1. Linux subsystem on Windows

Linux	
ls /mnt/	windows system accessible through mnt
cd /mnt/c	go to the hard drive (c)
nano ~/.bashrc • alias winhome='cd <directory>' • source ~/.bashrc	edit script • typing winhome will go to this directory • reads and executes file

Windows	
bash	opens linux terminal
exit	closes linux terminal