Useful commands

This pdf is meant as a quick reference to revisit things we discussed in the tutorial. If you missed the session and want to learn about using the command line more interactively, you can read this guide, or watch either this video if you are in a hurry, and/or this video, and this video if you have a bit more time.

Basics

Who and where am I

- `whoami`: Show user name
- `pwd`: Show current working directory ("**p**rint **w**orking **d**irectory")

Interacting with files & directories

- `ls`: List directory contents. Called with no arguments, it lists contents of the current working directory, but you can also call it on a directory path, like `ls ~/Documents` in which case it will list the contents of that directory. *Useful flags:*
  - `-l`: **l**ong **l**ist format (also shows some metadata like permissions, owner, creation date etc.)
  - `-a`: **a**ll (also show hidden files)
- `cd`: **C**hange **d**irectory. Call on a path to "move there", e.g. `cd ~/Documents` will put you into the Documents-directory. Called without any arguments, it will always put you into your home directory (~; on macOS: `/Users/username`, on Linux `/home/username`).
- `touch`: create files and modify metadata. "Touching" a file will change its last modified date, and/or create the file if it does not exist. *Example:* `touch somefile.txt`.
- `cp`: **C**o**p**y files (or directories). *Usage:* `cp [origin] [destination]`. *Useful flags:*

  - `-r`: copy **r**ecursively (use to copy a directory; copy recursively = copy directory & everything in it)

- `mv`: **M**o**v**e (or rename) files. *Usage:* `mv [origin] [destination]`. To move: `mv somefile.txt ~/Documents`; to rename: `mv oldname.txt newname.txt`; to move and rename: `mv oldname.txt ~/Documents/newname.txt`.

- `rm`: remove files (or directories). The command takes no prisoners, it does not move anything to the trash bin, it *removes*, so use carefully. *Useful flags:*

  - `-r`: **r**ecursively (used to remove directories; remove directory and everything in it)
  - `-f`: **f**orce; don't prompt user, ignore missing arguments/files etc. ("Don't ask me or complain, just remove!")

- `echo`: Prints to output. You can use it to talk to yourself, or to write text into a file by using `>`. *Usage:* `echo "Something very important" > very_important.txt`

- `mkdir`: create directory or directories. When called with multiple arguments, it will create separate directories, e.g. `mkdir code/ data/` will create two directories. *Useful flags:*

  - `-p`: Create **p**arents as well (if you want to create nested directories)

- `rmdir`: remove empty directories; will fail if directory contains files (in that case use `rm -r`, cf. above).

- `man`: Show **man**ual page for a command. On the manpages, you can read up on what commands & their options/flags do. *Usage* Just call it on a command, like `man [command]`

- `cat`: Con**cat**enate & print file contents. Can be used to just print the contents of a file to the output or to concatenate & print multiple files if called with multiple arguments.

- `wc`: **W**ord **c**ount, count the number of words in a file. *Useful flags:*

  - `-l`: number of lines instead of words.

- `head` & `tail`: print first/last few lines of file. *Useful flags:*

  - `-n`: **n**umber of lines to print

- `less`: browse file (not just print it).

- `grep`: Search for text strings in files. **G**lobal **R**egular **E**xpression **P**rint. *Usage:* `grep [string] [file]`, e.g. `grep "God" bible.txt`. *Useful flags:*

  - `-i`: **i**gnore case
  - `-v`: in**v**ert match (show lines *not* containing string)

"Globbing"

"Wildcards" for string matching. Here is a more comprehensive overview.

- `^`: Starts with
- `$`: Ends with
- `*`: Anything
- `[...]`: Any of the enclosed characters

Compression

- `zip`: create zip-archive
- `unzip`: extract zip-archive
- `tar`: create tar-archive (**T**ape **Ar**chive). *Useful flags:*
  - `-x`: e**x**tract
  - `-c`: **c**reate
  - `-v`: **v**erbose (display progress/output during extraction or creation)
  - `-f`: **f**ilename
  - `-z`: Use gzip compression
  - `-j`: Use bzip compression
  - `-J`: Use xz compression.

Let's talk tar for a moment as it's notoriously confusing. For example, to extract a tar-archive you downloaded from somewhere, you can extract it like this (`x` = extract, `v` = verbose, `f` = from file name):

```
tar -xvf file.tar
```

to create an archive of your own, you have to use `c` (create) instead of `x`. `z` tells tar to use gzip compression:

```
tar -cvzf my_archive.tar.gz my_folder
```

Operators (Piping and redirection)

- `|`: "Pipe"; chain commands together. Read in your head as *and then*. *Example:* Counting the number of pdfs in my documents folder: `ls ~/Documents | grep pdf$ | wc -l`; i.e. list contents of the "Documents"-folder, *and then* `grep` for names ending in "pdf", *and then* count the number of lines.

- `>`: "Redirect", e.g. to file to write to that file (`>` overwrites current contents, if any). *Example*: `echo "hi mom" > hello.txt`. A more useful example would be something like making a list of all pdf documents in my documents folder, like `ls ~/Documents | grep pdf$ > pdfs.txt`.
- `>>`: Redirect (appending); does not overwrite existing file but appends to the end of the file.

Editing text

- `nano`/`pico`: Small & basic editors.
- `vi`/`vim`/`neovim`: Configurable modal text editor; may seem esoteric at first but when mastered allows editing at the speed of thought. Major rabbit hole.
- `emacs`: Programmable & extremely extensible text editor (although calling it a "text editor" is an understatement, it's basically a swiss army knife). *Major* rabbit hole, even has its own programming language, Elisp. Can be used in the terminal & as a GUI editor.

Downloading stuff

- `wget`: Download files using HTTP, HTTPS or FTP. *Usage:* `wget [flags] [url]`. For example: `wget https://some_website.com/some_file.txt`. *Useful flags:*
  - `-O`: Rename downloaded file (**O**utput document), like `wget -O new_name https://my_url.com/my_file`
- `curl`: transfer data using protocols such as HTTP, HTTPS, FTP, SCP, SFTP, and more. In the most simplest case, you can use it to download a website or a file at a specific url: `curl https://example.com`. However, `curl` also allows you to send custom HTTP requests using GET, POST, PUT etc. For more on `curl`, click here. *Useful flags:*
  - `-X`: change method to use when starting transfer. You can use this to send custom requests. For example: `curl -X POST -d "key1=value1&key2=value2" https://api.example.com/resource` (where `-d` specifies the data to send).

Managing software

To manage software on the command line, you need a *package manager*. On macOS, you can use Homebrew (`brew`). All Linux-distributions have a package manager preinstalled. The ones you are most likely to encounter are `apt` (on Debian-derivatives like Ubuntu & Mint), `yum`/`dnf` on RedHat-distros (RHEL or Fedora), and maybe `pacman` (Arch & derivatives). Commands that make changes need to be run with superuser privileges (`sudo`). The following is about `apt`:

- `apt install`: Install a package or multiple packages
- `apt remove`: Uninstall a package.
- `apt update`: refresh package index (ask remote repository if there are more recent versions of the packages you have installed).
- `apt upgrade`: Download & install the updates. You can update & upgrade in one go by chaining them: `sudo apt update && sudo apt upgrade`.
- `apt list --installed`: List installed packages
- `apt search`: Search for a package by name

Networking

- `ip`: Show and configure network addresses. *Useful subcommands:*
  - `ip addr show`: Show your current IP adresse(s)
  - `ip link show`: Show link status
  - `ip route show`: Show routing table
- `ping`: Send echo requests (use to test connectivity), *example:* `ping google.com`. *Useful flags:*
  - `-a`: audible ping
  - `-4`: Use IPv4 only
  - `-6`: Use IPv6 only
- `ssh`: **S**ecure **sh**ell. Securely log into and operate remote computers via network. *Usage:* `ssh [username]@[hostname/IP]`. If you connect to some machine repeatedly, you should set up a config file for easy access.
- `scp`: **S**ecure **c**o**p**y. Securely copy files or directories between two devices via network. *Usage:* `scp [user]@[source_host]:/path/to/file [user]@[target_host]:/path/to/file`.

Processes

- `ps`: **P**rocess **S**napshot; get a list of currently running processes with other information like process ID (PID) and user. *Example:* You'll probably want something like `ps -aux` which shows basically everything. Protip: Use `grep` to search for the process you are looking for. *Useful flags:*
  - `-a`: **a**ll
  - `-u`: **u**ser
  - `-x`: I assume it means something like e**x**tend...? Lifts restrictions on what processes are displayed (actually display *all* processes when used together with `-a`)

- `kill`: Kill a process via PID (you can use `ps` to find the PID of the process you are looking to kill). *Example usage:* `kill 6006`.
- `pkill`/`killall`: Kill process via name; `killall` takes the exact name, while `pkill` also works with partial names (e.g. if you don't know exactly what the process is called). *Example usage:* `killall dropbox`.
- `top`/`htop`: System monitoring utilities. `htop` is just a more "modern" version of `top`.
- `systemctl`: Tool to access background processes running via systemd. *Example:* Depends a bit, for example if you changed your network configuration and want the changes to apply, you don't need to fully reboot, but can just restart the corresponding background process: `systemctl restart NetworkManager.service`. *Useful subcommands:*
  - `systemctl status`: Show current status of a background process
  - `systemctl stop`: Stop process
  - `systemctl start`: Start process
  - `systemctl restart`: Restart process

Multitasking

- `tmux`: **T**erminal **mu**ltiple**x**er. Lets you switch easily between several programs in one terminal, detach them (they keep running in the background) and reattach them to a different terminal. A powerful usecase for you will likely be to keep programs running on a remote server even when your connection to the server is severed. Below is a simple example, you can read up on how it works in a more detailed manner here. *A simple example:*
  - Run `tmux` to start a session (notice the green bar at the bottom).
  - Run some command (e.g. `htop`).
  - Use Ctrl+B D (hit control and b, then release and hit d) to **d**etach the session. This will put you back into your "normal" shell.
  - Run `tmux attach` to reattach the session (go back into tmux); you will see that your program kept running in the background.
  - To fully quit `tmux`, use `exit`.

Version control with Git

- `git status`: Show the status of the current working tree (what *branch* you are on, changes made to files etc.)
- `git init`: Create empty repository or reinitialize existing one
- `git clone`: Clone a repository into a new local directory (= get a local copy of a remote repository)
- `git fetch`: Download objects and refs from a rmeote repository.

- `git merge`: Join two or more "development histories" together (use e.g. to integrate changes you downloaded using fetch)
- `git pull`: Fetch & merge (download changes from a local repository and integrate them with your local version right away)
- `git add`: Add files to the staging area (prepare them for commit)
- `git commit`: Commit changes you made to local repository (they are now recorded & tracked by git)
  - `-m`: Add commit **m**essage (you should *always* do this)
- `git push`: Push changes made to your local repository to the remote repository
  - `-u`: Set **u**pstream (select which branch to push to)
- `git checkout`: Switch branches (or restore working tree files)
  - `-b`: New **b**ranch; creates the specified branch if it does not exist yet
- `git reset`: Reset to a previous state. You can specify a certain commit you want to revert to; if no commit is specified, it will reset to the last commit. *Useful flags:*
  - `--hard`: Reset to last commit discarding all changes (changes your files!)
  - `--soft`: Reverts a commit without discarding changes (rolls back commit, but does not change your files!)
- `git stash`: Momentarily store changes in a "dirty" working directory. Use this if you have made changes, but cannot push because your branch has changes you do not have locally. It "puts your changes aside" for a moment for you to work out what to do, so you can go back to a clean state without losing your changes.
  - `git stash pop`: Reapply changes (take from the top of the stash). Let's say you had remote changes not reflected, so you `stash` your changes away, `reset` to a clean working state, `pull` in the remote changes, and then `stash pop` your previously made changes.

You can create a `.gitignore`-file in your repository if there are some files or folders you would not like git to track (e.g. notes you keep there or random temporary clutter created e.g. by notebooks or your OS).

General Notes

- Hit Tab to autocomplete
- Copy and paste works with Ctrl+Shift+C and Ctrl+Shift+V
- Ctrl+C to **c**ancel a running command
- Use `clear` (or on most systems Ctrl+L) to clear your screen
- Use `exit` to exit your shell

Paths

Paths can be *absolute* or *relative*. The absolute path of a file is the path from your filesystem root to a file, e.g. `/home/user/Documents/Uni/ICSS/notes.txt`, while the relative path is the path to a file from your current working directory: let's assume you are already in `~/Documents`, the path to the same file would be `ICSS/notes.txt`.

Exercises

- Download the bible as text file (https://openbible.com/textfiles/kjv.txt)
- How many times is "God" mentioned in the bible? How many times is "Jesus" mentioned? (Hint: to be exact you can use `grep` with the `-o` flag)
- Make a backup copy of the bible text file that has another name (e.g. bible.txt.bak)
- Change the original text file
- Remove the original text file
- Rename the backup copy to something else
- Zip the text file
- Unzip it again
- Delete it