shell: cheat sheet

Don't Panic

- Open a terminal with Ctrl+Alt+T, close it with Ctrl+D
- echo ARGS: print the arguments ARGS on screen
- man COMMAND: open the **man**ual entry for a given command
- history: manage or consult recent input commands. Ctrl+Shift+R: lookup in previous input commands history

Paths

- current working directory: the "folder" where you currently are in the file-system. Represented in terminal as ./
- subdirectory: a directory SUBDIR contained in another directory DIR is a subdirectory of DIR. Represented as DIR/SUBDIR in terminal.
- parent directory: the directory "one above" a given directory, i.e., the directory that contains a given directory. Represented in terminal as ../; thus the parent directory of the directory DIR/ is DIR/../
- absolute and relative path:
  - An absolute path starts from the "root" (represented as / in terminal), the "topmost" directory that is the direct or indirect parent of all other directories. All paths that start with / are absolute.
  - A relative path starts from the current working directory.
- home: the "default place" where the terminal is started, corresponding to the absolute path /home/USER, where USER is the username as returned by the command whoami. This absolute path is represented as ~ in the terminal.

Directory Navigation

- pwd: **p**rint the absolute path of the current **w**orking **d**irectory
- cd DIR: **c**hange current working **d**irectory to DIR
- ls DIR: **l**i**s**t the content of directory DIR; list subdirectories recursively using ls -r, more details using ls -l.
- tree DIR: display **tree** structure of directory DIR
- wc FILE_1 ... FILE_N: lines, **w**ords and character **c**ounts for each file FILE_1, ..., FILE_N; restrict to line counts with wc -l FILE, word counts with wc -w FILE, character counts with wc -c FILE.
- du DIR and df: show **d**isk **u**sage of directory DIR and **d**isk **f**ree memory; human-friendly format with option -h
- find DIR ...: **find** files/directories under DIR, matching criteria, optionally execute actions. Common criteria: -type f for files, -type d for directories, -name PATTERN for names matching PATTERN. Common actions: -delete to delete, -quit to stop on first hit, -exec COMMAND to execute COMMAND
- mkdir DIR: **m**a**k**e **dir**ectory, yield error if directory or file of the same name exists/ parent directory do not exist. Create parents as needed and do not error if DIR exists with mkdir -p DIR.
- mv SOURCE DEST: **m**o**v**e file SOURCE to new location DEST; move directory contents recursively with mv -r SOURCE DEST
- cp SOURCE DEST: **c**o**p**y file SOURCE to new location DEST; copy directory contents recursively with cp -r SOURCE DEST

File Display

- cat FILE_1 ... FILE_N: con**cat**enate and print contents of files
- head FILE and tail FILE: show beginning or end of file FILE respectively. Show the first/last 42 lines (or any other number) of a file using option -42.
- less FILE and more FILE: interactively display file contents
- diff FILE_1 FILE_2: show **diff**erences between files FILE_1 and FILE_2
- basename FILE_PATH: remove directories from path FILE_PATH and keep only the file name; to also strip suffix EXT use basename -s EXT FILE_PATH

File Manipulation

- grep PATTERN: **g**lobal **r**egular **e**xpression **p**rint: print matches of regexp PATTERN found in FILE; many relevant options exist: -i to ignore case, -P for PERL regexps, -n to print line number, -o to print only match, -l to print only the file name when a match is found...
- awk PROGRAM FILE: programming language for CSV-like files
- sed INSTR FILE: **s**tream **ed**itor for regexp-based substitutions and deletions
- split FILE: **split** a large file into smaller files; specify their size with -l
- cut FILE: trim files column-wise, specify column delimiters with -d, restrict to the 3rd to the 5th columns with -f 3-5
- paste FILE: merge files column-wise, specify column delimiters with -d
- sort FILE: **sort** file FILE; random sort using -R, keep only distinct (unique) lines with -u, specify output file with -o, reverse order with -r, merge (*but do not sort*) sorted files with -m
- uniq FILE: keep only **uniq**ue (distinct) adjacent lines in FILE, add a count number of unique lines with -c.
- tar, zip, unzip, gzip and gunzip: produce and extract file archives; for tar: extract using tar -xvf ARCHIVE_NAME, compress with tar -cvf ARCHIVE_NAME ITEMS_TO_ARCHIVE, apply gzip on top of tar with option -z.
- rm FILE: **rm**ove and permanently destroy file FILE; to remove and destroy a directory with its contents, use rm -r DIR

Process Control

- `chmod FILE`: **ch**ange the **mod**e of access to a file `FILE`; in particular make it executable with `chmod +x FILE`
- `su IDENTITY -c COMMAND` and `sudo COMMAND`: identify yourself and execute `command` as **s**uper **u**ser (or as `IDENTITY` if provided); `su` without arguments opens a session as super user
- `source INSTRUCTIONS`: execute instructions listed in file `INSTRUCTIONS`; equivalent to `. INSTRUCTIONS`
- `ps`: show a **s**napshot of the current running **p**rocessess
- `kill PID`: **kill** or terminate a process identified with `PID`
- `top` and `htop`: display all linux processes
- `watch COMMAND`: repeat the same `command` indefinitely; specify interval between repetitions using `watch -n TIME COMMAND`

Remote File Access

- `wget URL`: **get** a document from the **w**eb, i.e., download from link `URL`
- `ssh HOST`: **s**ecure **sh**ell access to a remote server: generate an access key using `ssh-keygen`, have it accepted on the remote server, and then connect to the remote server using `ssh login@remote.server:port`
- `scp SOURCE DEST`: **s**sh **cp**y, i.e., copy files from/to server to which you have ssh access
- `rsync SOURCE DEST`: **r**emote server **sync**hronization; make the contents of a remote and local directories equivalent

Basic syntax and operators

- Variables: declare a variable named `VAR` with a value of `FOO` using the syntax `VAR=FOO` (*without spaces*); refer to this variable elsewhere in the code using `${VAR}` or `$VAR`. Variable names are conventionally capitalized in `shell`.
- Loop control flow: loop over a list, and refer to each element with the variable `${ELEM}` using the syntax:

```
for ELEM in 1 2 3 4; do
    echo ${ELEM};
done;
```

- Conditional control flow: execute a command based on whether a test is true with the following syntax:

```
VAR=42;
if [ $VAR -gt 41 ]; then
    echo "the test went ok!";
else
    echo "alack! 'tis failed!";
fi;
```

- Piping and `xargs`: pass the output of one command as input to the next command using the syntax `command_1 | command_2`
  - the command `find . -type f -name '*.md' | grep '/data/'` will find all markdown files under the current directory and then prune the search results to files under a subdirectory called `data`
  - to use each line the first command's input as distinct **ex**ternal **arg**ument**s** of the second command, use `xargs`: the command `find data/ -type f -name '*.md' | xargs grep -li package` will list only markdown files under the directory `./data/` that contain the word "package" (ignoring case).
- Inputs and outputs:
  - To redirect the output of a `command` to a file `FILE_1.txt`, use the syntax `command > FILE_1.txt` ; if the file exists it will be overwritten *before the command is executed*. Therefore `cat FILE_1.txt > FILE_2.txt` is equivalent to `cp FILE_1.txt FILE_2.txt`
  - To append the output of a `command` to a file `FILE_1.txt`, use `command >> FILE_1.txt` ; if the file doesn't exist it will be created.
  - To use the contents of a file `FILE_1.txt` as input for a `command`, use the syntax `command < FILE_1.txt`
- Conditional execution: execute `command_2` only if the previous command `command_1` did not error with the syntax `command_1 && command_2`
- Background execution, suspension, termination, `fg` and `bg`:
  - Execute a `command` in the background (i.e., without blocking the terminal until its completion) with the syntax `command &`
  - Call background processes to the **fg** foreground with `fg`, e.g. to manually terminate them (using `Ctrl+C`) or suspend them (using `Ctrl+Z`)
  - Let a suspended process run in the **bg** background using `bg`
- Substitutions: variable names are replaced with corresponding values in double-quoted strings: `"here's an ${EXAMPLE}"`. A `command` can be replaced by its output using the syntax `$(command)`, e.g.:

```
for FILE in $(ls); do
    echo $(basename $FILE);
done;
```

- Arithmetics and `bc`: perform computations using the syntax `$((2 + 3))` or for floating point arithmetics `echo "2.1 / 3.2" | bc -l`