University of Regensburg
Institute of Theoretical Physics
Summer term 2024
Discussion on: 6 June 2024
Dr. Jan Wilhelm, Dr. Štěpán Marek, Maximilian Graml
https://www.ur.de/physics/wilhelm/teaching/sose24-computational-nanoscience

Computational Nanoscience: Exercise Sheet No. Linux

Exercise Linux.1: Using command line on Linux systems

We will work extensively with the Linux terminal emulator to start our programs. If you are completely new to working in terminal, it might be a good idea to watch the following video tutorial, up to about 46:05

https://www.youtube.com/watch?v=oxuRxtrO2Ag

Typically, we submit commands in the terminal, which start with the program name and continue with arguments. Most programs have help implemented, which hints at what arguments can be supplied. This help can be accessed via

```
ls --help
```

More extensive manual is often registered with the manual program `man`, so to get documentation for `cat`, you could type and submit

```
man cat
```

A Linux system has a set of so-called `coreutils`, typically ones implemented by the GNU project. The documentation for those is accessible at

https://www.gnu.org/software/coreutils/manual/coreutils.html

We provide you with some simple exercises to practice your usage of coreutils in the terminal

(a) Use `echo` to write into a file via redirection of standard output, i.e.

```
echo "Hello, world!" > out.txt
```

(b) Use `cat` to put the contents of a file to the standard output, i.e.

```
cat out.txt
```

(c) Create a new directory via `mkdir`

```
mkdir my_dir
```

(d) Copy a file to the new directory via `cp`

```
cp out.txt my_dir
```

(e) List the contents of the new directory via `ls`

```
ls my_dir
```

(f) Generate a sequence of numbers via `seq`

```
seq 0 2 10
```

(g) Iterate over sequence of numbers via `for`, and append the results to the file. Then check the file contents by `cat`

```
for i in $(seq 0 2 10); do
 echo "Reached $i" >> my_dir/out.txt
done
cat my_dir/out.txt
```

Exercise Linux.2: Text editor

A lot of the time, you will need to edit or inspect text files which store input/output to your calculations. This is done via a text editor.
A text editor can be either terminal based or GUI based. For GUI editors, you can use `gedit`, which is installed on the university computers and has familiar layout.
On a supercomputer or when connecting to a a computer via ssh without X forwarding, GUI is not available, and you must rely on the terminal based editors.
A more forgiving editor is the `nano` editor. Here, we focus on a more involved editor called `vim`.
Unlike common text editors, when `vim` opens a file, it starts in the so called ”normal” mode, where you cannot directly insert new text. The normal mode is suited for fast editing and searching the text document.
If you want to actually write, you need to enter the ”insert” mode by pressing `i`.

- `i` enter insert mode

Then, you can insert your text as normal. If you have copied some text from a GUI application, you can enter it by pressing `Shift+Insert`, or by highlighting the text with mouse and clicking the middle mouse button at the required position in the document.

- `Shift+Insert` insert text from clipboard

After you are done with writing the text, exit back into the normal mode by pressing `Esc`. Save your work by entering a command - start by pressing `:`, continued by name of the command, in this case `w`, as in ”write”.

- `Esc` exit back to normal mode

- `:w` write changes
- `:wq` write changes and exit vim
- `:q` exit vim, no changes were made
- `:q!` exit vim, discarding all changes made

In normal mode, you can navigate the cursor around by arrows or by `hjkl` keys. You can also navigate by skipping between words

- `w` skips to the next beginning of a word
- `e` skips to the next end of word
- `b` skips to the previous beginning of a word

Navigating to a specific line is done by `gg` and `G` (`Shift+g`)

- `120gg` go to the line 120 from the beginning of the file
- `20G` go to the line 20 from the end of the file
- `10` go to the line 10 lines after the current line

You can also search for words in the normal mode

- `?needle` search for ”needle” in the file, starting from the current cursor position and going upwards, i.e. to the beginning of the file
- `/needle` search for ”needle” in the file, starting from the current cursor position and going downwards, i.e. to the end of the file
- `n` when searching, go to the next find in the order of search
- `N` (`Shift+n`) when searching, go to the previous match in the order of search

Search and substitute is done by the `s` command

- `:s/needle/nail/` substitutes the first match of ”needle” on the current line with ”nail”
- `:s/needle/nail/g` substitutes all matches of ”needle” on the current line with ”nail”
- `:%s/needle/nail/g` substitutes all matches of ”needle” on all lines with ”nail”

You can edit at several lines by using the block selection `Ctrl+v`, followed by either `s` to substitute the highlighted text by text you will enter, or by `Shift+i` to insert new text at the position before the highlighted column, or by `x` to cut the highlighted contents.
Contents can be pasted by `p`, and copied without cutting by `y` (”yank”). Changes can be undone by `u`.

Exercise Linux.3: Standard input and output redirection

In general, every program you run in the terminal can take data from arguments and standard input, and output data to standard output and standard error streams.
Sometimes, there is a lot of data, so we would like to store the standard output of a program in a file. This is done by redirection, which has the following syntax in bash

```
my_program argument > saved_output.txt
```

This overwrites the contents of file `saved_output.txt` - if you want to just append to the end of the file, use >> instead.
Arguably one of the biggest strengths of using a shell (programming language of the terminal) is the ability to chain programs together by getting a standard output of one program and setting it as a standard input of another program.
For example, if you are looking for a file which you do not exactly know the name of in a large directory, you can use

```
ls | grep "part"
```

where ”part” is a part of name of the file you remember. Here, the `grep` program searches each line of the standard input, which is provided by the `ls` program, which lists the contents of the current directory.
Acquiring some skill with the use of so called ”pipes” (the | character which connects the stdin - standard input - of one program to stdout - standard output - of another program) can help you save a lot of time.
For example, suppose we have a file of atomic coordinates `coord.xyz` with the following contents

```
10

C     -6.06103    1.44039    0.00000
C     -4.57785    1.39760    0.00001
H     -6.57411    2.20751   -0.54144
H     -6.61749    0.70411    0.54143
C     -2.73816    1.36908    0.00000
H     -4.60158    0.32786    0.00112
C     -1.32630    1.31203    0.00000
H     -2.70826    2.43866   -0.00022
H     -0.82914    0.36454    0.00020
H     -0.75432    2.21633   -0.00019
```

We can substitute all the C atoms by Si atoms by running the following command

```
cat coord.xyz | sed 's/C/Si/' > coord_si.xyz
```

Here, we used the stream editor program `sed`. Other similar program you might be interested in is `awk`, which could be for example used to offset all the carbon atoms by 1 Angström in y direction

```
cat coord.xyz | awk '/C/ {print $1, $2, $3+1.0, $4; next};{print $0}'
```

Finally, sometimes it is useful to see the output of a program in the terminal while also saving it in a file for later reference. This can be done by duplicating the output using the `tee` program. For example, to capture the output of a cp2k calculation into a file while also displaying it on a screen, so that we can visually check the progress, use

```
cp2k.psmp cp2k.inp | tee cp2k.out
```