

Introduction to Linux

Command Line Interface – The basics

Samuel Braun, SCC, KIT



UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Hochschule
für Technik
Stuttgart

Hochschule Esslingen
University of Applied Sciences

Universität
Konstanz



UNIVERSITÄT
MANNHEIM



Universität Stuttgart

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN



KIT
Karlsruher Institut für Technologie

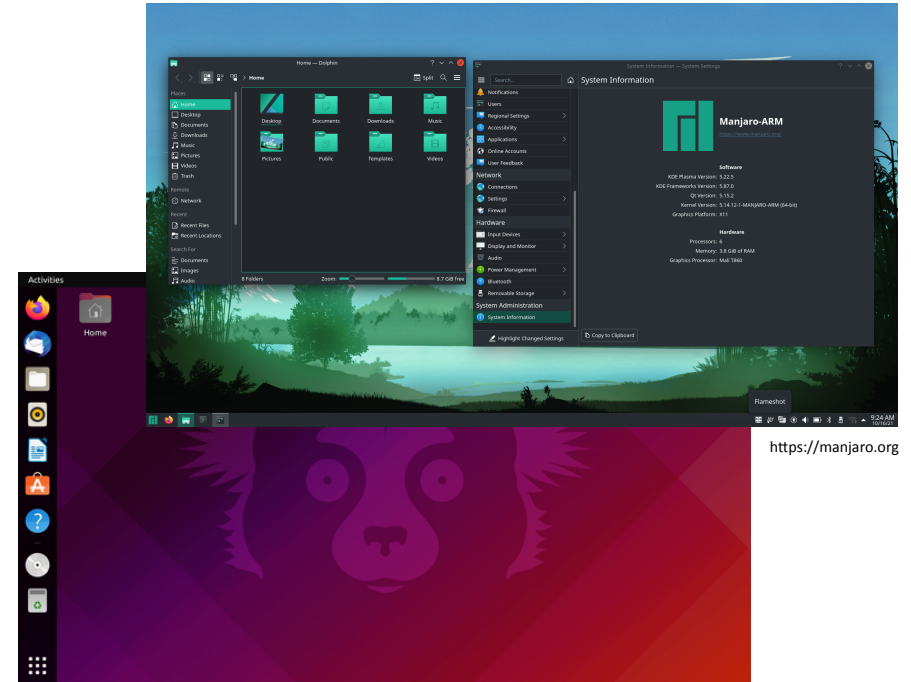


ulm university universität
uulm



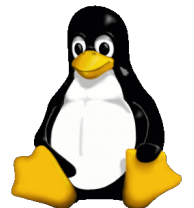
Linux

- Operating system
- **Kernel**: Layer between Hardware and (User-) software
- **Desktop Environment**
 - KDE
 - GNOME
 - ...
- **Distribution**: software collection + package management system
 - Ubuntu
 - Manjaro
 - Fedora
 - ...
 - Red Hat Enterprise Linux



<https://manjaro.org>

<https://ubuntu.com>



Command Line: Motivation

- Command-line interface (CLI) is often the most **powerful** and **flexible** way to interact with a computer
- Commands that tell the computer what to do
- These commands can be combined
- **HPC** context:
 - Often no GUI needed
 - Overhead of GUI
 - SSH access
- Shell: User interface for OS
 - Bash (other flavors exist)
 - Command line interpreter
 - **Scripting language** (!)
- **Not Linux-only**: Windows, MacOS, ...
- You can feel like a real **haxxor**

What is this course about and what not?

We will present the *basics* of the command line:

- simple commands
- navigating
- some usability features
- no special command
- no programming (what you can do inside of the command line)

The command line interface

```
[projects]$ ls -l foo.txt █
```

- prompt [projects]\$
- command line `ls -l foo.txt`
 - command `ls`
 - option `-l`
 - argument `foo.txt`
 - cursor █

The first command

```
$ echo hello bwHPC
hello bwHPC
$
```

Echo the STRING(s) to standard output.

```
$ echo hello          bwHPC
hello bwHPC
$
```

→ *Task*: Please do all examples by yourself.

Using *up-Arrow* for the last command. Use *double* or *single* quote to mark strings.

```
$ echo "hello          bwHPC"
hello          bwHPC
$ echo 'hello          bwHPC'
hello          bwHPC
$
```

Getting out of trouble

You can get in trouble by:

- unfinished typing of a command
- long or endless running command
- command expecting further input

Solution: holding Ctrl-Key (Strg) and pressing C. Short written as `Ctrl-C` or `^C` (remember as 'cancel')

If it is not working try `Ctrl-D` (remember as 'end of transmission', 'end of input'), ESC or just `q`.

→ *Task:* Try and exit the following commands:

- `$ echo "hello"`
- `$ yes`
- `$ cat`

Effect of single and double quotes

Each variable begins with `$`. There are many variable set that defines the environment.

```
$ echo My home is $HOME  
My home is /home/kit/scc/ab1337
```

→ *Task:* Try out the different effects of quoting by print out the variable `$HOME`.

Getting help

`$man echo` open the manual pages of the command `echo`. It uses `less` as a page viewer, where you can use the arrow keys to navigate.

`less` basics:

- `up` & `down` arrow Move up or down one line
- `spacebar` Move forward one page
- `q` quit
- `/<string>` search file for `<string>`.
- `n` Move to next search result.
- `N` Move to previous search result.
- `p` goto beginning of the file
- `h` help

→ *Task*: Find out how to print text without the newline at the end.

Summary

- `echo <string>` Prints string to screen.
- `man <command>` Displays manual page for command.
- `^C` Get out of trouble.
- *Up & down arrow* Scrolls through previous command history.

Manipulating files

Create files

```
$ touch foobar  
$ touch foobaz
```

redirecting standard output (stdout) to a file:

- redirect operator > (overwrites files)
- append operator >>

```
$ echo "This is the first line." > foobar.txt  
$ echo "This is the second line." >> foobar.txt  
$ cat foobar.txt  
This is the first line.  
This is the second line.
```

Listing

```
$ ls  
foobar  foobar.txt  foobaz
```

→ *Task*: What does `ls -lha` do? Try also `-t` and `-r`. Use different combinations.

Note: For short flags you can combine the flags instead of using `ls -l -h -a`. Long flags beginning with two dashes `ls --help`

Make life easier (Tab completion)

use the *tab* key →

```
$ cat f→
```

expands to

```
$ cat fooba
```

twice tab print possible matches

```
$ cat fooba→→
```

foobar foobar.txt foobaz

```
$ cat foobar.→
```

→ *Task*: Print out `foobar.txt` without typing to much. How many keystrokes are needed?

Make life easier (copy and paste by mouse)

```
$ ls
foobar  foobar.txt  foobaz
$ cat █
```

Mark `foobar.txt` with mouse. Click middle mouse button to insert at cursor.

→ *Tasks:*

- Print out `foobar.txt`.
- What will happen when you print out `foobaz` including the space after z?

Make life easier (reverse search)

Search the history.

```
$ ^R echo
(reverse-i-search)`echo': echo "This is the
second line." >> foobar.txt
```

- ^C cancel
- ^R previous search
- Enter run command
- right arrow select command for editing

→ *Task:* Insert a third line to foobar.txt.

Manipulating files: Rename, copy, delete

Moving

Moving a file will rename it.

```
mv foobaz test
$ ls
foobar  foobar.txt  test
```

Manipulating files: Rename, copy, delete

Copy from source to target

```
cp foobar.txt test_text.txt
```

```
$ ls
```

```
foobar  foobar.txt  test  test_text.txt
```

Manipulating files: Rename, copy, delete

Remove

WARNING: It is deleted, really, no trash, nothing.

```
rm foobar.txt
```

```
$ ls
```

```
foobar  foobar.txt  test  test_text.txt
```

Editors (nano)

A basic editor

```
$ nano test_text.txt
```

Help for some useful commands see bottom lines

Try ^K^K^U^U to cut and uncut lines

exit nano

^X exits and asks for save changes, type y, type filename or press enter for current filename.

Editors (vim)

A much more powerful and very fast editor

```
$ vim test_text.txt
```

Press `i` to go to insert mode. Now you can type text.

exit vim

Press `ESC` to go back to the normal (command) mode. Type `:q` to quit. If you change something you have to write and quit `:wq` or force quit `:q!`.

LEARN vim!

```
$ vimtutor (-g language)
```

It takes 25-30 min. You can start at the end of this course.

Summary

- `>` Redirect output to filename
- `>>` Append output to filename
- `touch <file>` Create an empty file
- `cat <file>` Print contents of file to screen
- `ls` List directory or file
- `mv <old> <new>` Rename (move) from old to new
- `cp <old> <new>` Copy from old to new
- `rm <file>` Delete (remove) file (no recovery!)
- `→` Auto completion
- `^R` Reverse search
- `vimtutor` Tutor for learning vim

Directories

```
$ pwd  
/home/kit/scc/ab1337
```

Prints the current working path, starting with the root directory / followed by the directories home, kit, scc and ab1337.

Create directory

```
$ mkdir text_files
```

```
$ mkdir example
```

This is relative. Paths are normally relative from the current working path.
Absolute paths beginning with the root /.

Moving directories

Move files to directories

```
$ mv *.txt text_files/  
$ ls text_files/
```

Use tabs! *.txt is a Wildcard matching all files ended on .txt.

Moving directories

Move (rename) directories

```
$ mv example/ data
```

```
$ ls
```

Changing directories (Navigation)

```
$ cd text_files/
```

```
$ cd ..
```

```
$ cd data
```

`cd ..` goes one directory up.

→ *Task*: Double check directories with `pwd` and `ls`.

Relative changing

A path beginning with `../` goes relative to the current working directory one directory layer for each `../` up. Try

```
$ cd ../text_files/
```

Tab completion adds automatically a `/` at end of directory name. (Don't matter at the moment.)

→ *Task*: create `foo/bar/` at once.

Special navigation

Moving to the last directory

```
$ cd -
```

Moving to the home directory

```
$ cd
```

```
$ cd ~
```

```
$ cd $HOME
```

`$HOME` is the already known variable for the home directory. See later courses for other path variables.

Copying directories

Add `-r` option for recursive

```
$ cd
$ mkdir foobar
$ cd foobar/
$ cp -r ../text_files .
$ ls
text_files
```

`.` is the current directory.

→ *Task:* What happens if you add a `/` to the source directory `../text_files/`?

```
$ cp -r ../text_files/ .
$ ls
text_files text.txt
```

Remember as `../text_files` is the directory and `../text_files/` is already inside the directory.

Remove directories

Add `-r` option for recursive

```
$ cd
```

```
$ rm -r foobar
```

Warning: Again, there is no warning, it will be deleted, not trash, nothing.

→ *Task:* Do not execute it! What are the options `-f` and `-r` are doing in `rm`? Why you should *NEVER* use `rm -rf /`?

Summary

- `mkdir <name>` Make directory with name
- `pwd` Print working directory
- `cd <dir>` Change to `<dir>`
- `cd ~/<dir>` cd relative to home
- `cd` Change to home directory
- `cd -` Change to previous directory
- `.` The current directory
- `..` One directory up
- `cp -r <old> <new>` Copy recursively
- `rm -r <dir>` Remove dir and content

Find files

→ *Task*: create some `.txt` files in different directories.

```
$ find . -name "*.txt"
```

Search recursively for files beginning in the current directory `.`, filter by name, only display files ending with `.txt`.

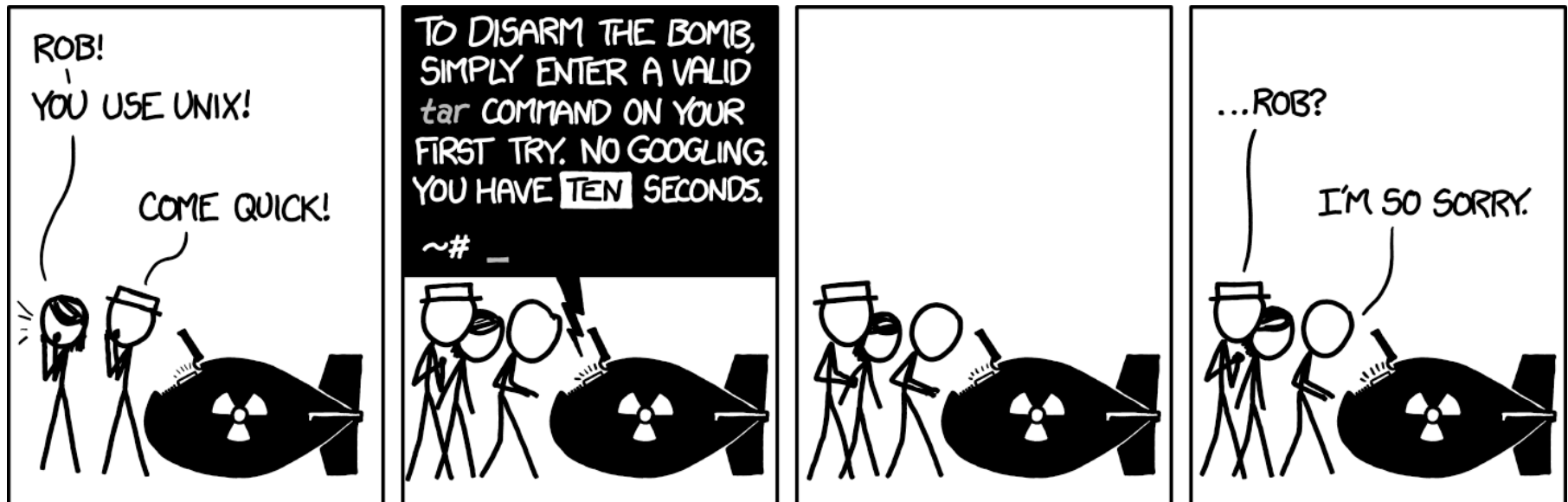
```
$ grep -r line
```

Search recursively (`-r`) for files and print lines containing `line`.

→ *Task*:

- Print line number of lines with 'first' using `grep`
- Print lines not containing 'first' using `grep`

Any Questions Left?



<https://xkcd.com/1168>