Leibniz Supercomputing Centre

of the Bavarian Academy of Sciences and Humanities

Introduction to GNU/Linux – Part 1

October 7th, 2024 | M. Ohlerich *)

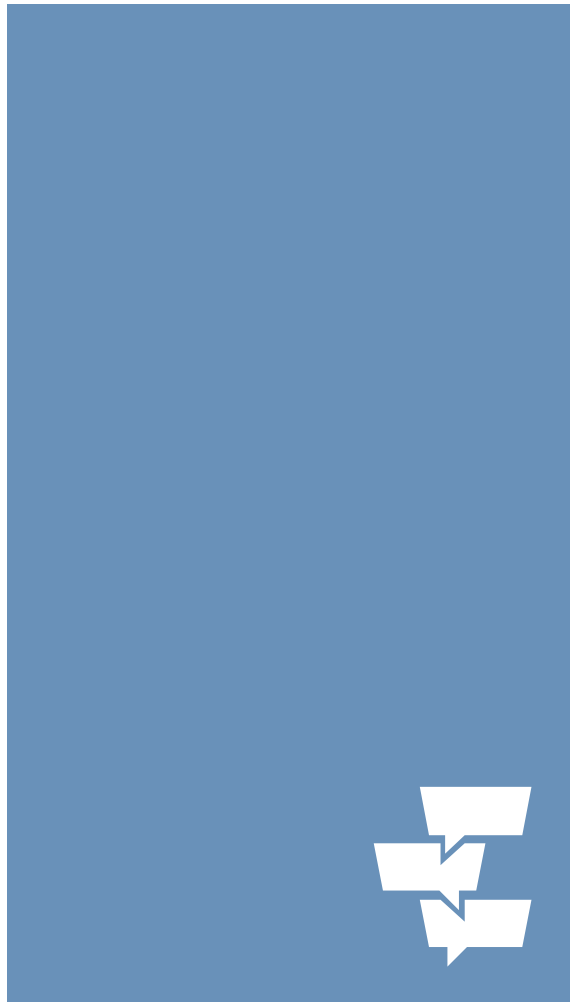
*) Slides vastly prepared by J. Albert-von der Gönna

Session Information

- Aim: *provide an intro to GNU/Linux*

- You will probably benefit the most if you're not yet familiar with GNU/Linux, but if you plan to work on the AI, HPC and/or Compute Cloud infrastructure provided by LRZ
  -> by the end of this introduction, you should *have a basic understanding of GNU/Linux-based systems*

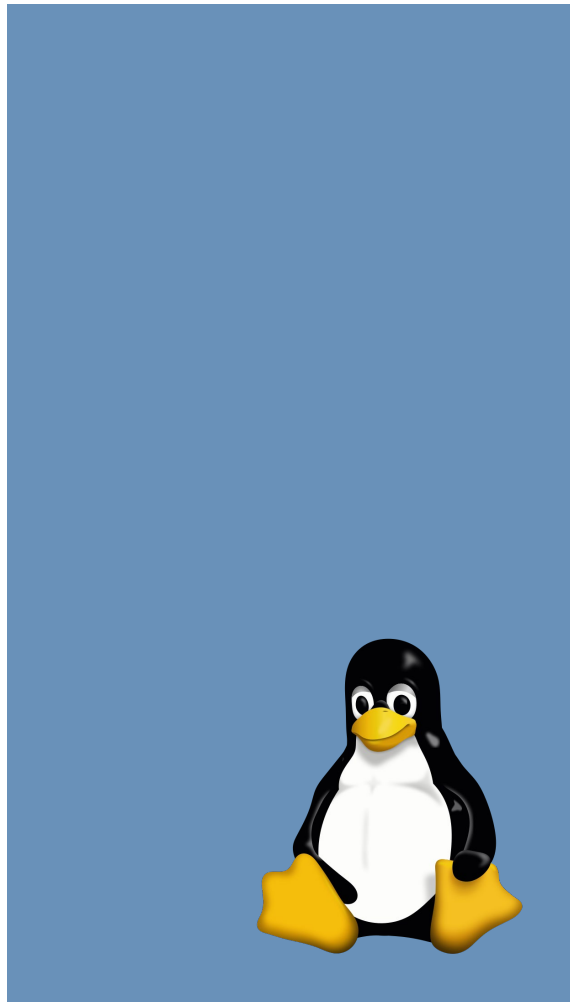
- *If you have questions, please ask them at any time*



What is GNU/Linux?

- *Free*, *open-source*, *secure/stable*, *flexible* operating system
  (available on all kinds of hardware)
- *Alternative* to
  Microsoft Windows, Apple macOS, Google Android …
- Generally consists of the *Linux kernel, libraries and tools*,
  (possibly) a *desktop environment* and various *applications*
  (e.g. web browser, office suite, …)
- Different *distributions*:
  Arch Linux, Debian/Ubuntu, Fedora/RHEL, openSUSE/SLES, …
  (differences in kernel version, package manager, application and system package versions and availability)

https://en.wikipedia.org/wiki/List_of_Linux_distributions



https://www.gnu.org/licenses/quick-guide-gplv3

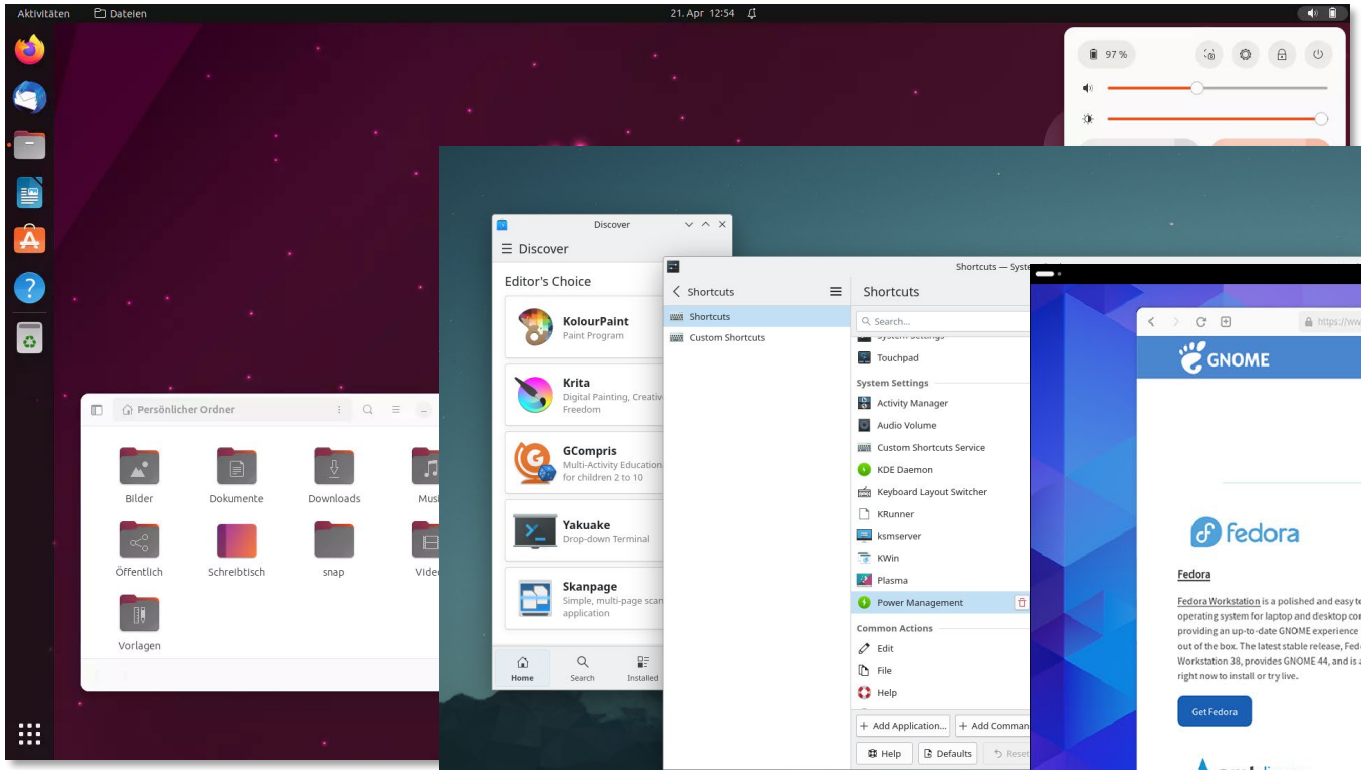
GNU General Public License (GPL)

The powerful versatility of free and open-source software is rooted in their licenses, such as the GNU General Public License (GPL). This license grants four essential freedoms or rights to the users of the software:

- The freedom to “**use** a program as they wish, for any purpose",
- the right to "**study** how the program works, and change it so it does the computing as they wish",
- the freedom to "**share** and redistribute copies so they can help their neighbor" and finally
- the right to "**improve** the software and to distribute copies of their modified versions to others".

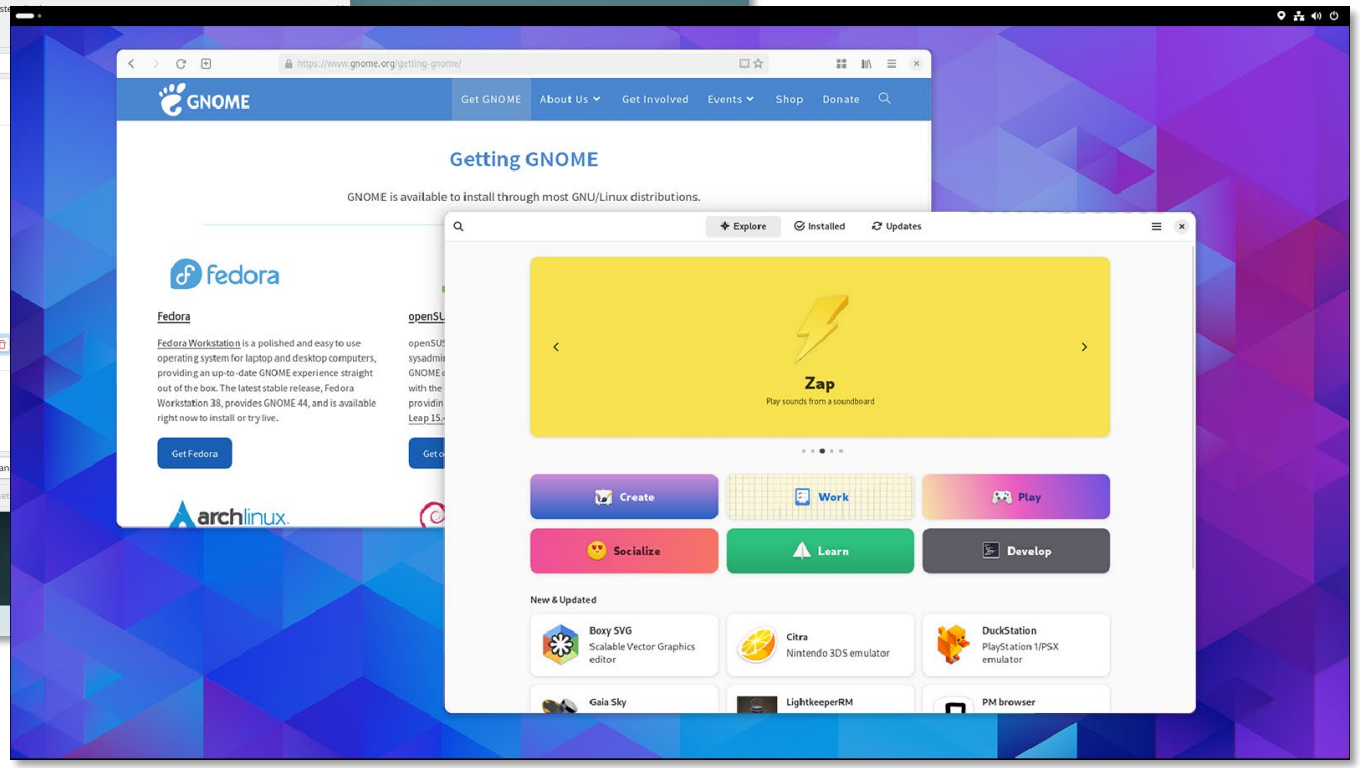
These rights are key for e.g. tuning software on a one-of-a-kind supercomputer, but also, more generally, in an environment where the goal is to create reproducible research and open science.

Popular Desktop Environments



Ubuntu Desktop

KDE Plasma

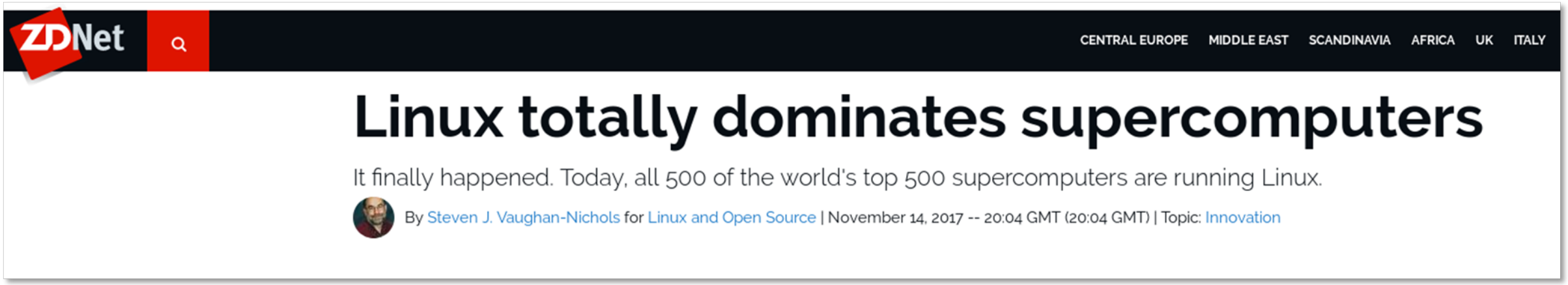


GNOME Shell

… and several more.

https://www.zdnet.com/article/linux-totally-dominates-supercomputers/

Linux and High Performance Computing (HPC)



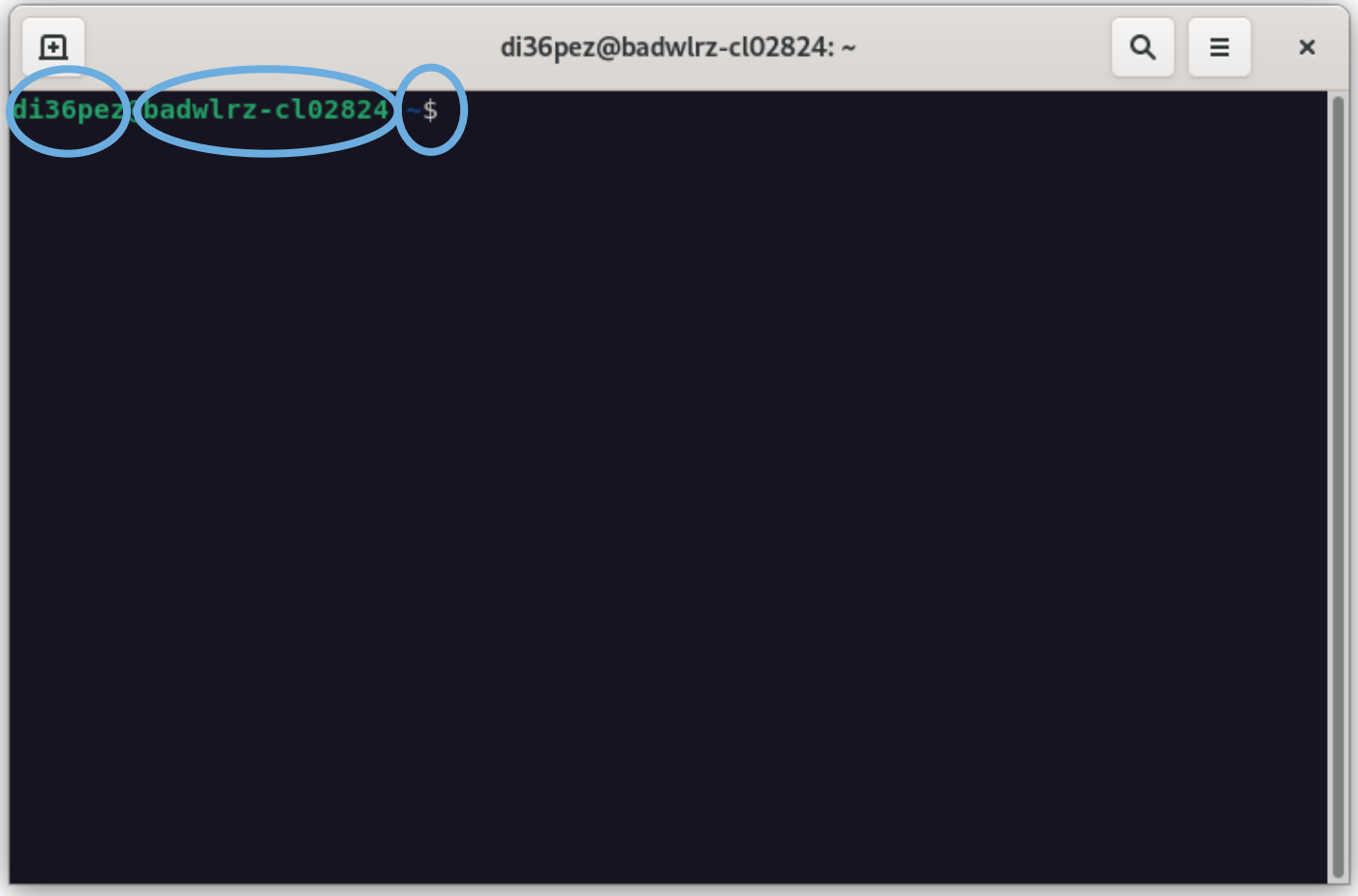
It’s time to get started!

- GNU/Linux: choose your favorite terminal application
- macOS: launch Terminal
- Windows:
  - Windows 10/11: SSH to remote Linux System
    ‘Command Prompt’ (cmd) and ‘Powershell’ if SSH installed
  - Windows 10/11: Windows Subsystem for Linux (WSL)

  - Alternatively (specifically older Windows):
    - Git BASH (part of Git for Windows) or MSYS2
    - MobaXterm (Home Edition)
    - ...

Explore a shell environment!

A Unix-like Shell in a Terminal Application



prompt

File System Hierarchy Standard (FHS)

- On Unix-like systems – everything == *file* or *directory*
- All files and directories appear (somewhere) under *root directory* "/", even if stored on different – possibly remote – devices.
  There are no *drive letters* like on Windows
- Use `pwd` to get name of *p*resent (current) *w*orking *d*irectory
- Use `ls` to *l*i*s*t all files and directories in the current directory
- Use `ls /` to list all files and directories in the *root directory*
- Use `ls /any/other/dir` to list all files and directories in the specified directory
- Note: separate directory hierarchy with slash "/"!!

On Unix-like systems:
try the commands introduced on the left.

Exploring the File System

```
[root@localhost ~]# pwd
/root
[root@localhost ~]# ls
dos       hello.c
[root@localhost ~]# ls /
bin       etc       lib       linuxrc   mnt       proc      run       sys       usr
dev       home      lib32     media     opt       root      sbin      tmp       var
[root@localhost ~]#
```

/bin*: command binaries (e.g. ls)
/etc: configuration files
/home: (regular) users' home directories
/lib*: libraries (for binaries in /bin et al.)
/media: mount points for removable media
/mnt: mounted filesystems
/root: home directory of the root user
/sbin*: system binaries
/usr: secondary hierarchy for read-only user data
/var: variable, i.e. changing files

* On modern systems, these (and /libXX) are only symlinks/shortcuts. Their former contents have been merged into their respective /usr/… counterparts, which they then point to.

Detailed Listing of All Files

```
[root@localhost ~]# ls /usr
bin                             libexec
i486-buildroot-linux-uclibc     local
include                         sbin
lib                             share
lib32                           var
lib64
[root@localhost ~]# ls /home
```

Use the `l` and `a` options with `ls` (i.e. `ls -la`) to get a detailed listing of all files in your current (home) directory (we will cover most of this information later).

Can you spot the differences to the previous listing (using just `ls`)?

```
[root@localhost ~]# ls -la
total 20
drwx------    3 root     root          135 Oct 24 19:34 .
drwxrwxrwx   19 root     root          457 Feb 27 13:44 ..
-rw-------    1 root     root            0 Jul  8  2017 .Xauthority
-rwxr-xr-x    1 root     root           28 Jun 24  2017 .xsession
drwxr-xr-x    3 root     root          163 Aug 20  2011 dos
-rw-r--r--    1 root     root          242 Jul 15  2017 hello.c
[root@localhost ~]#
```

General Command Syntax

This command syntax can serve as general example to distinguish different components:

```
$ ls -la /home
```

`ls` is the **command**
with the (short) **options** (also **switches** or **flags**) `-la` and
the **argument** `/home`

Options generally start with either a *single dash* **-** (short, as above),
or *two dashes* **--** (long).

Getting Help

- There are at least two common *local* ways you can try to find out how a command works and which options it accepts…
  (many additional resources online, e.g https://www.mankier.com/ or https://tldr.sh/
  or Linux Command Handbook or ...)

1. Pass `--help` option to command:
   ```
   $ ls --help
   ```
2. Read a command's manual (man pages), using the `man` command:
   ```
   $ man ls
   ```
   (use the *arrow keys* to *move* up and down, press *q* to *quit* the man page)

- What effect may the `-h` option have on the `ls` command?
- Can you spot other interesting options?
- Did you try `man man`?

At the outset…

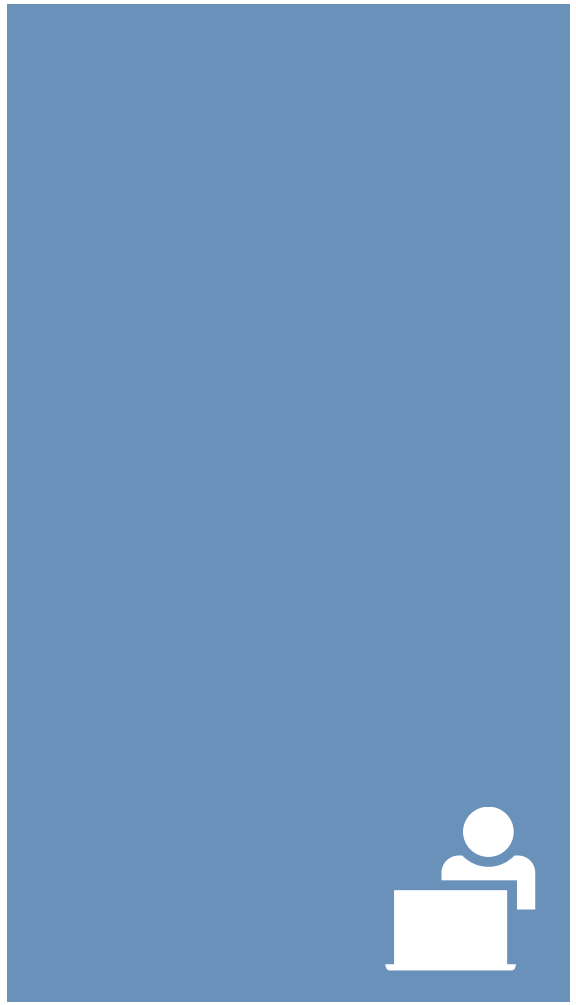
- At this point, you should have an *initial understanding* of what a GNU/Linux operating system is, you should have access to a (Unix-like) shell environment on your *local* machine.
- First steps were taken to explore a GNU/Linux system. You have encountered the very first commands to interact with the shell environment and you know how to get additional help for such commands.
- These are already the very basic skills that allow you to start working on remote systems using the Secure Shell (SSH).
- You will continue – and gain more experience – working with GNU/Linux systems in one of our later sessions (navigate the file system, file manipulation and ownership, characteristics of the shell environment, useful commands & concepts, …).

Introduction to GNU/Linux – Part 2

October 7th, 2024 | M. Ohlerich

Directories

- Create a new directory in your current (home) directory called „my_dir“:
  `$ mkdir my_dir`
- Change your current working directory to this folder:
  `$ cd my_dir`



Unix-like Commands (Addendum)

- There are *real programs* and *bash commands,* e.g.

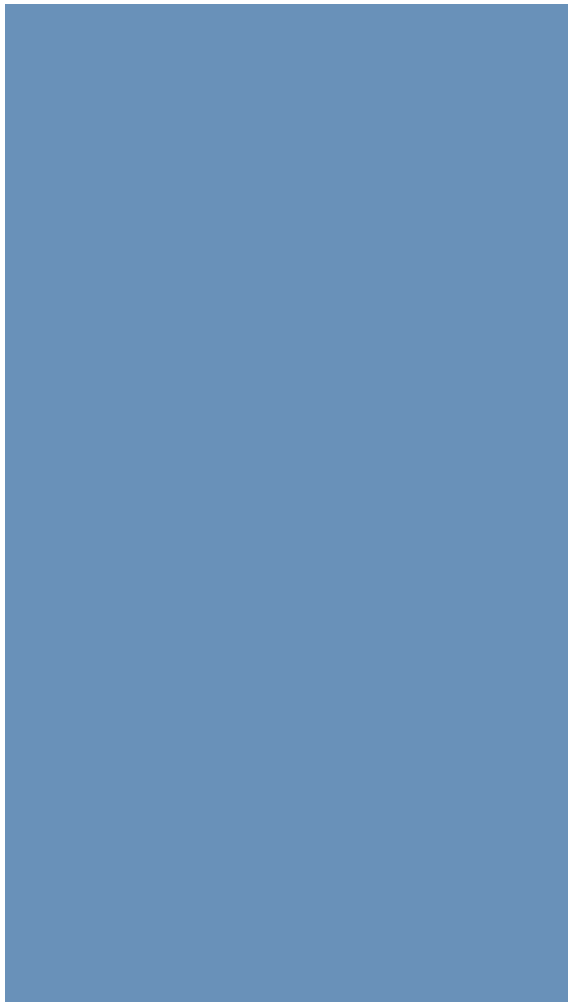
```
$ which ls
/usr/bin/ls

$ which cd          # Ooops!

$ help cd           # bash‘s “man page“  (help help)
```

- What about clashes? … (Keep that in mind. Things are vastly more complicated.)

Bash Reference Manual



Navigating Directories

```
[root@localhost ~]# mkdir my_dir
[root@localhost ~]# cd my_dir/
[root@localhost my_dir]# ls
[root@localhost my_dir]#
```

Notice the changing prompt…

What does the “~” symbol represent?

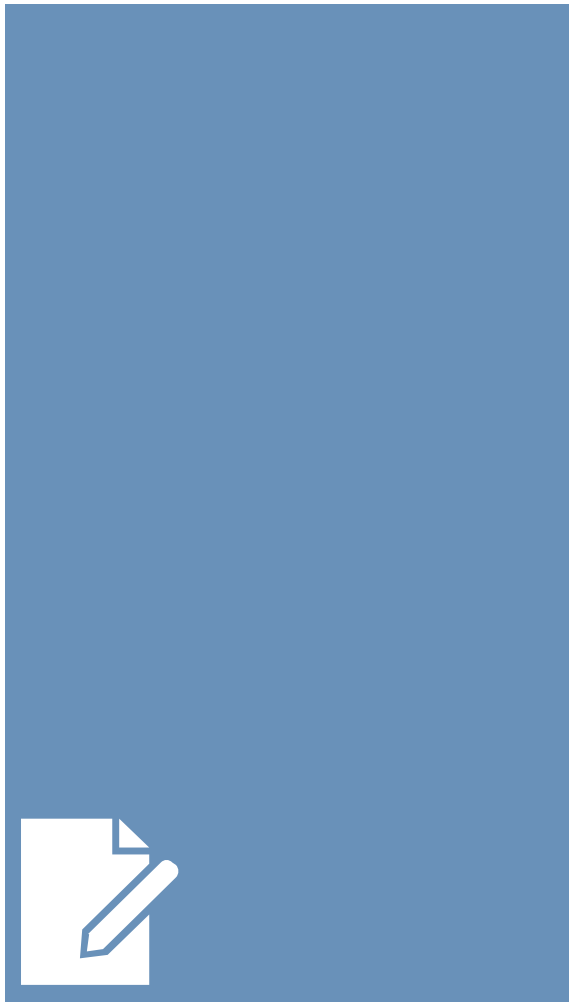
`cd ..` move back to parent directory

single dot . represents the current, two dots .. the parent directory

absolute vs. relative paths:
specifying a location with leading slash / indicates start at root the file system (*absolute*), omitting it → *relative to current directory*

Tip: *use* the **tab key** for **auto-completion**!

File Manipulation



- Make sure you‘re located in the my_dir directory created earlier
- Create a new (text) file by “touching” it:
  ```
  $ touch my_file
  ```
- Can you spot the newly created file in a file listing?
- What‘s the content of this new file? How can you tell?

File Manipulation (Editors)

```
[root@localhost my_dir]# touch my_file
[root@localhost my_dir]# ls
my_file
[root@localhost my_dir]# ls -la
total 8
drwxr-xr-x    2 root     root              61 Feb 27 14:27 .
drwx------    4 root     root             158 Oct 24 19:34 ..
-rw-r--r--    1 root     root               0 Feb 27 14:27 my_file
[root@localhost my_dir]#
```

Can you spot the size of this (empty) file?

On most systems, you can use editors like e.g. nano, vi(m) or emacs to edit text files directly in the console.

Use nano to edit the existing file (write something to it):

```
$ nano my_file
```

Note the shortcuts along the bottom of the *nano* screen; "^" represents the Control (CTRL) key

Use nano to create another file and write a couple of lines:

```
$ nano another_file.txt
```

Be aware of (missing) file extensions: In contrast to other operating systems, GNU/Linux does not rely on file extensions to specify the type of a file. For interoperability and clarity, file extensions can still be used, of course.

File Manipulation and Redirection

Refer to the *cat* manpage for additional information.

- There is a tool called cat. What does it do?
- “Con*cat*enate FILE(s) to standard output.
  With no FILE, or when FILE is -, read standard input.”
- Use cat to display the contents of my_file
  ```
  $ cat my_file
  ```
- The shell allows for input/output redirection using > (and <)
- Use cat to write something to nice_file.txt and display it afterwards

```
$ cat > nice_file.txt
write something nice here
and add another line
<CTRL+D>
$ cat nice_file.txt
```

And what does *tac*??

File Manipulation and Redirection

- Files can be appended using `>>`

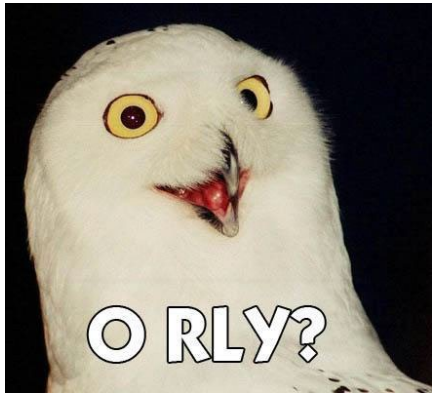
```
$ echo "yet another line of text" >> nice_file.txt
$ cat nice_file.txt
```

- Using `<<` allows for the creation of here documents (input stream literals), the general format is:

```
command << delimiter  # (commonly EOF, or "Ctrl+D")
input stream
delimiter
```

- Try the following. Can you explain how the here document is used?

```
$ tr a-z A-Z << EOF
> all lower case
> o rly?
> EOF
```



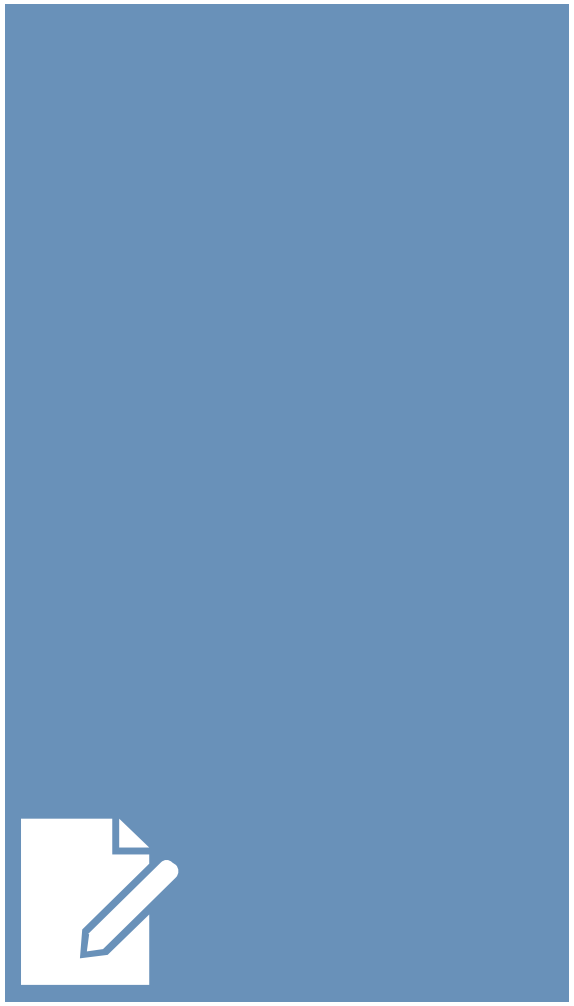
Pipes

- Commands can be chained using | (the pipe). It will instruct the shell to use the output of one command directly as input for another command. Pipes can be used consecutively.

```
$ echo "some fancy words" | wc -l
$ echo "some fancy words" | tr " " "\n" | wc -l
```

(More useful example appear with availability of more advanced tools.
See find, grep, sed, awk, …)

File Manipulation

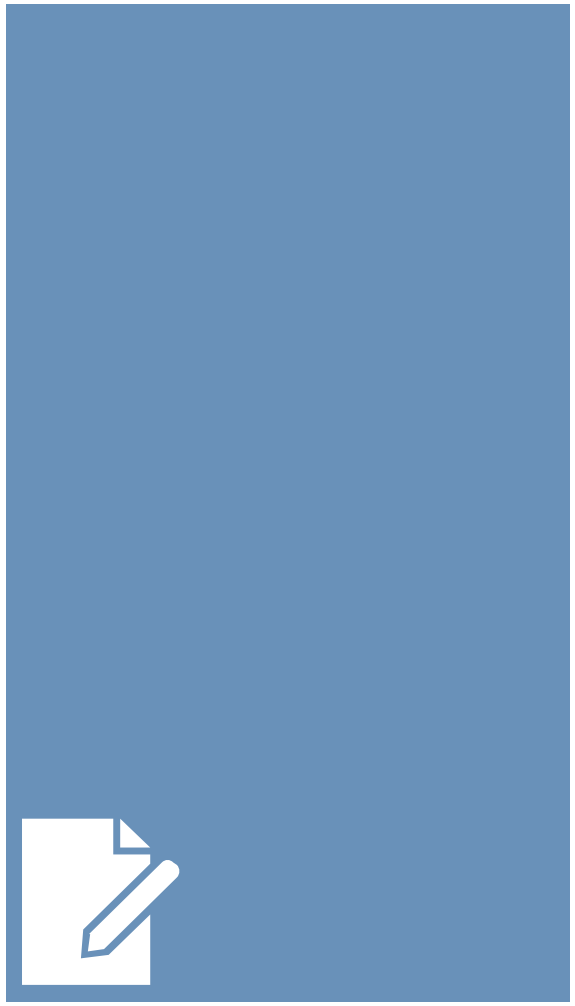


- Create a copy of “my_file” called “my_file1”:
  ```
  $ cp my_file my_file1
  ```
- Rename/move the copy “my_file1” to “new_file”:
  ```
  $ mv my_file1 new_file
  ```
- Delete the original file “my_file” :
  ```
  $ rm my_file
  ```
  Caution: there is *neither trash bin, nor undo*!
- Take a look at the file listing. What is the expected output? Does it match?

File Manipulation

```
[root@localhost my_dir]# ls
my_file
[root@localhost my_dir]# cp my_file my_file1
[root@localhost my_dir]# ls
my_file   my_file1
[root@localhost my_dir]# mv my_file1 new_file
[root@localhost my_dir]# ls
my_file   new_file
[root@localhost my_dir]# rm my_file
[root@localhost my_dir]# ls
new_file
[root@localhost my_dir]#
```

File Manipulation



- Create two more copies of “new_file”, “01.bak” and “02.bak”
  ```
  $ cp new_file 01.bak
  $ cp new_file 02.bak
  ```
- Move to your home directory.
  ```
  $ cd ..
  ```
  Alternatively (there are many ways to get home):
  `$ cd` or `$ cd /path/to/home/dir` or `$ cd ~` or `$ cd $HOME`
- Copy “new_file” to your home directory.
  ```
  $ cp my_dir/new_file .
  ```
- Make a (full) copy of “my_dir” called “another_dir”.
  ```
  $ cp –r my_dir another_dir
  ```

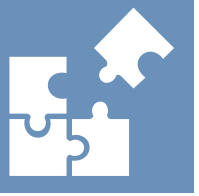
Shell Wildcards

- Wildcards can be used flexibly for character matching in the shell:
  - Zero or more characters → `*`
    ```
    $ ls -la mydir/n*
    ```
  - Exactly one character → `?`
    ```
    $ ls -la mydir/0?.bak
    ```

- They can be combined in any way and are useful for operating on files and directories that contain certain patterns.

- Count the combined number of words in all (created text) files with a file extension:
  ```
  $ cat */*.??? | wc -w
  ```

Can you think of other patterns to match certain files or directories?



Searching: grep

- ***grep*** … search for things **in** text **files** or **streams** (pipes)
- uses *regular expressions* (regex). (formal language to describe text patterns)
- As basic building blocks (besides characters), they can include:
  - a boolean “or”, represented by the vertical bar or pipe |.
  - Parentheses () are used for grouping.
  - Placeholders (similar to shell wildcards) can be used for quantification. The ? represents zero or one occurrence of the preceding element, * represents zero or more occurrences of the preceding element and + represents one or more occurrence(s) of the preceding element.
  - The regular expression wildcard dot . matches any character and can also be combined with the quantifiers mentioned above.
- Consider options like
  -w: Select only those lines containing matches that form whole words
  -n: Prefix (file name and) line number to each match
  -i: Make search case-insensitive
  -v: Invert search, i.e. output non-matching lines
  and many more…
- Basic examples (without regex quantifiers, wildcards, etc.) are

```
$ grep something my_dir/nice_file.txt
$ grep -R 'another line' ./*
```

Can you think of other regular expressions to match specific strings (but not others)?

Searching: find

- The ***find*** command can be used to search for files and directories, e.g.

```
$ find .
$ find . -type d
$ find . -name "*.txt"
$ find . -type f -name 'a*’
```

- Some practical examples (update all file time stamps)

```
$ find . -type f -exec touch {} +
```

- (remove all object files from a compilation, in a tree)

```
$ find . -type f –name '*.o' -exec rm {} +
```

Refer to the find manpage for additional information.

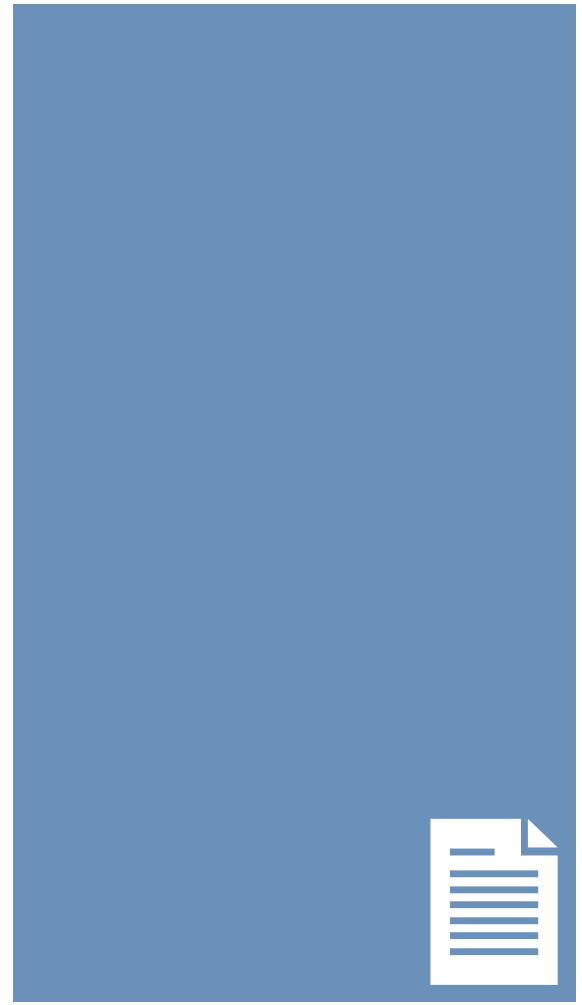
Shell Scripting

- Use shell scripts to save and re-use commands
- Create a new file myscript.sh containing the line

```
echo "This script is simple."
```

- Once saved, you can run it explicitly (using the Bash shell)

```
$ bash myscript.sh
```



Shell Scripting

- Modify the script to allow for argument use. Add the line:
  ```
  echo "This $1 is $2."
  ```
- Alternative methods:
  ```
  $ echo 'echo "This $1 is $2."' > myscript.sh
  ```
  or
  ```
  $ cat > myscript.sh
  echo "This $1 is $2."
  <Ctrl+D>
  ```
- Provide the needed arguments when calling the script:
  ```
  $ bash myscript.sh "scripting" "getting somewhere"
  ```
  or
  ```
  $ source ./my_script.sh "scripting" "getting
  somewhere"
  ```

Single vs double quotes …

Shell Scripting

- Add a shebang interpreter directive as the first line for direct execution:

```
#!/bin/bash
echo "This script is simple."
```

- Afterwards, call the script directly
  `$ ./myscript.sh`

- What is going on? What about `./`…?

Can you explain the unexpected outcome?

Ownership and Permissions

```
[root@localhost ~]# ls -la
total 20
drwx------    3 root     root          135 Oct 24 19:34 .
drwxrwxrwx   19 root     root          457 Feb 27 13:44 ..
-rw-------    1 root     root            0 Jul  8  2017 .Xauthority
-rwxr-xr-x    1 root     root           28 Jun 24  2017 .xsession
drwxr-xr-x    3 root     root          163 Aug 20  2011 dos
-rw-r--r--    1 root     root          242 Jul 15  2017 hello.c
[root@localhost ~]#
```

- Every file/directory is owned by a specific user
  (usually the original creator, but this can be changed)
- Every user is member of a (primary) group (and potentially additional ones)
- Notice the two “root” columns above:
  the first one is the owner of the respective file/directory (here, a user called root)
  the second one is the group assigned to the file/directory (here, a group called root)

Ownership and Permissions

```
[root@localhost ~]# ls -la
total 20
drwx------    3 root     root          135 Oct 24 19:34 .
drwxrwxrwx   19 root     root          457 Feb 27 13:44 ..
-rw-------    1 root     root            0 Jul  8  2017 .Xauthority
-rwxr-xr-x    1 root     root           28 Jun 24  2017 .xsession
drwxr-xr-x    3 root     root          163 Aug 20  2011 dos
-rw-r--r--    1 root     root          242 Jul 15  2017 hello.c
[root@localhost ~]#
```

- Permissions (access rights) for files and directories are managed in three different classes: user, group and others
- Three specific permissions apply to each class:
  - read (a file or the names of files in a directory)
  - write (modify a file or the entries of a directory)
  - execute (a file or access file contents of a directory)

Ownership and Permissions

```
[root@localhost ~]# ls -la
total 20
drwx------    3 root     root          135 Oct 24 19:34 .
drwxrwxrwx   19 root     root          457 Feb 27 13:44 ..
-rw-------    1 root     root            0 Jul  8  2017 .Xauthority
-rwxr-xr-x    1 root     root           28 Jun 24  2017 .xsession
drwxr-xr-x    3 root     root          163 Aug 20  2011 dos
-rw-r--r--    1 root     root          242 Jul 15  2017 hello.c
[root@localhost ~]#
```

- The leftmost column represents these permissions as they apply to files and directories for each of these three classes

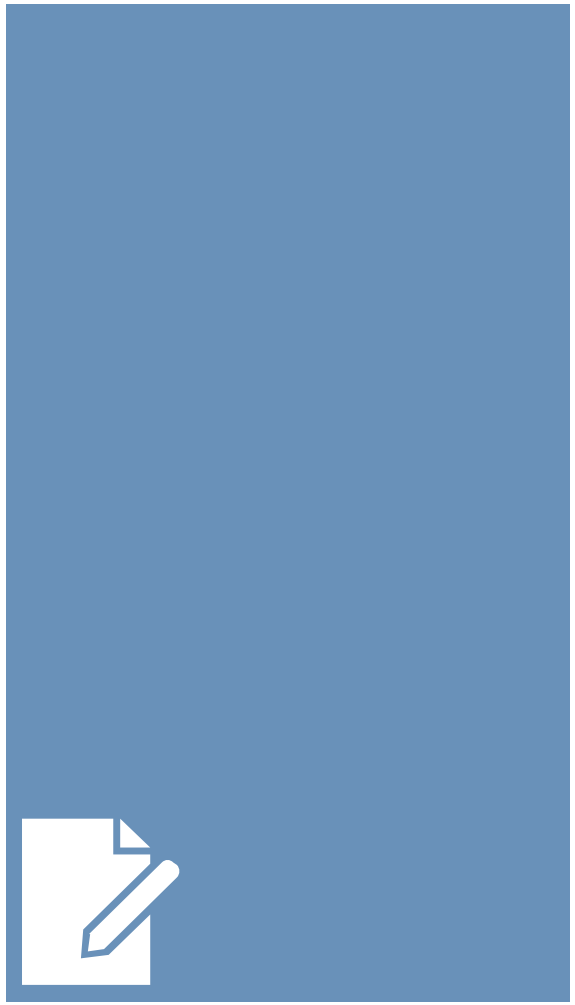
- Two examples from the output above:
  `dos:` `drwxr-xr-x` This is a directory. User (root) has rwx, (members of) group (root) rx and (all) other (users) rx permissions.
  `hello.c:` `-rw-r--r--` This is a file. User has rw, group r and other r permissions.

Shell Scripting

- In order to execute the previously created script file…
- …use `chmod` to change file permissions/mode bits
  `$ chmod +x myscript.sh` or `$ chmod u+x myscript.sh`
- Afterwards, call the script directly again
  `$ ./myscript.sh`
- I want to call *myscript.sh* also from an other directory!!
  `$ export PATH=$PATH:$PWD` # or path of script location

Refer to the chmod manpage for additional information.

File Manipulation



- Finally, let‘s clean up: completely delete “another_dir”.

  ```
  $ rm –r another_dir
  ```

  Again, be cautious: there is no trash bin or undo!

- There is another command called `rmdir`? Does this also work?

Additional material

Visit https://linuxjourney.com/ for many more interactive tutorials!

One more thing: Environment Variables

- *Environment variables* == named values that can influence how programs are run in the shell environment (e.g. by providing context information)
- Use the command `env` to print these variables in the current environment
- To print a specific environment variable, use the `echo $VARNAME` command
  e.g. `echo $HOME`
- To set (or change) a specific environment variable,
  use the `export VARNAME=<value>` command
- On many LRZ systems, we provide advanced mechanisms to adjust these environment variables for user-specific modifications, e.g. on the high performance computing clusters a “module system” is available that (amongst other functionalities) allows for providing/running different versions of the same application (making changes to environment variables to do so).

What else?

- Bash is a fully-fledged programming language (variables, conditionals, loops, functions, ...)
  Bash Scripting , Advanced Bash Scripting
- Bash can more: process/job control, history, parallelism ...
- Advanced tools: coreutils, sed, awk, ...
- Advanced Editors: VI(M), Emacs, ...
- Regular Expressions
- GNU Tools .... GNU Manuals
  (Don‘t read that all at once!! That‘s mostly for reference when you really need it!!)

- Some Guidelines:
  simplicity, efficiency, complexity by modularity, flexibility, continuity by defaults, know your tools

Course Evaluation

Please visit
https://survey.lrz.de/index.php/944988?lang=en
and rate this course.

Your feedback is highly appreciated!
Thank you!