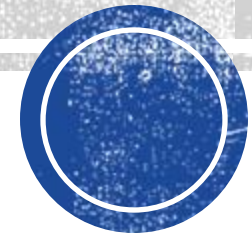


LINUX TERMINAL

Beginner Tutorial



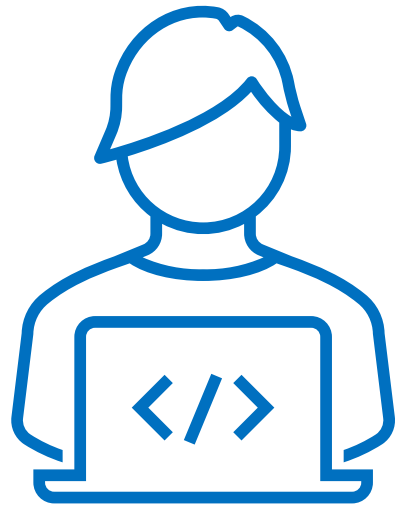


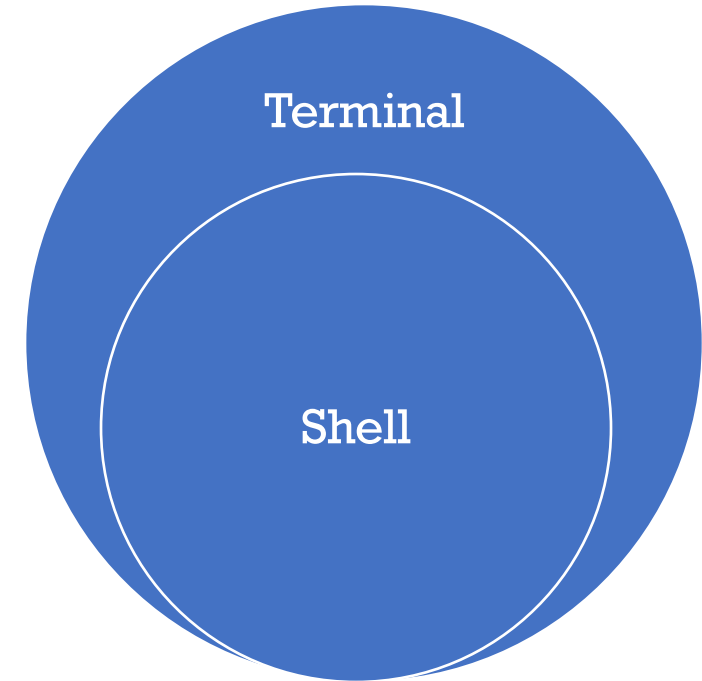
TABLE OF CONTENTS

- 1) Introduction
- 2) Navigation
- 3) Files and Directories
- 4) Editing Files Using Vim
- 5) Streams and Redirection
- 6) Subshells
- 7) Process Management
- 8) Installing New Programs with apt
- 9) Automation Using Scripts



1. INTRO: WHAT IS THE COMMAND LINE?

- Text-based tool to control the computer
- Terminal: interface to enter text commands
- Shell: program that evaluates the commands
 - **bash** or **sh** on Linux
 - **zsh** or **bash** on Mac
 - **Powershell** or **Command Prompt** on Windows
 - Other shells with various functions available



1. INTRO: WHY COMMAND LINE?

Advantages over GUI (Graphical User Interface):

- Available on every computer
- Easy development of software (no design experts necessary)
- Requires less resources (CPU, mem)
- Automation using scripts
- No mouse and searching for buttons needed

Disadvantages:

- Unintuitive for beginners



1. INTRO: WHY LINUX?

- Free usage
 - Open source: Flexible and independent
 - Higher performance than e.g. Windows
 - Full control over updates and system state
 - Many open source software are targeted on Linux
- > Basic knowledge important for computer scientists



1. INTRO: LAB SETUP

- Course material at **www.github.com/Jonas-Wessner/linux-beginner-lab**
- Install Docker from **www.docker.com/products/docker-desktop/**
- Docker allows running a "mini-linux" on your PC.
- We have created a preconfigured environment that contains exercises





QUESTIONS?



2. NAVIGATION: APPEARANCE

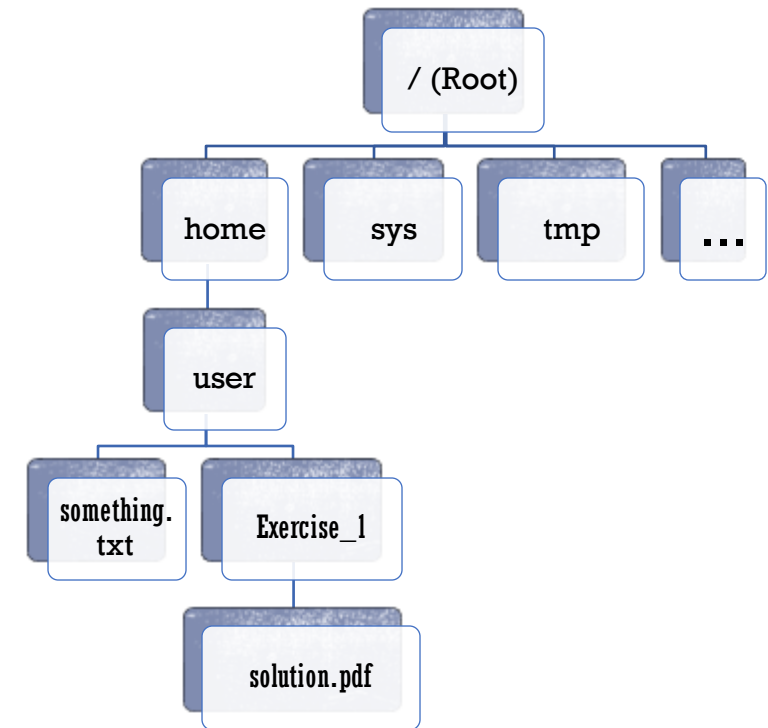
```
myuser@8a1bc344a6ba:~$ pwd  
/home/myuser
```

- "**myuser**": user name, privileged user is called "root"
- "**8a1bc344a6ba**": computer name
- Start position is user's home directory a.k.a. ~
- Show current position: **pwd** (**P**rint **W**orking **D**irectory)



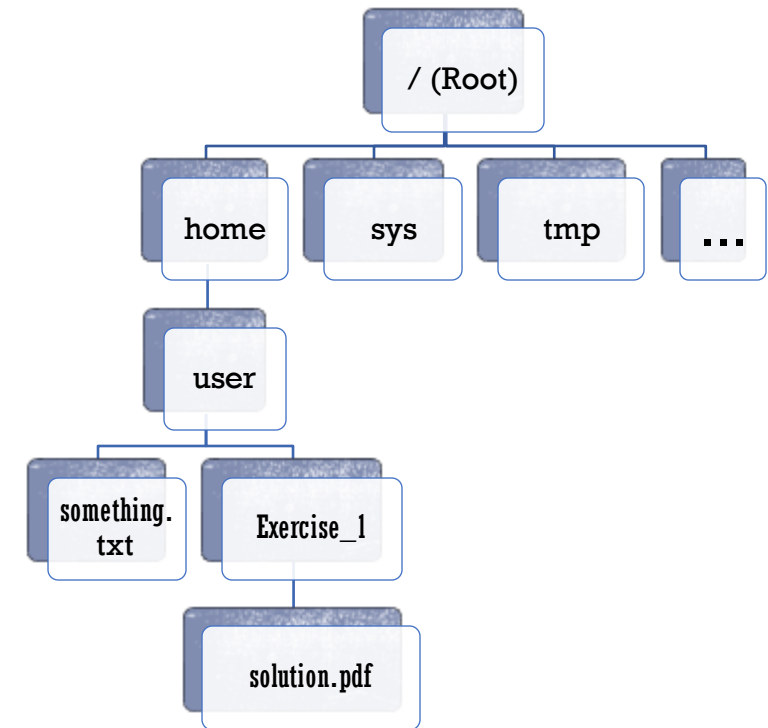
2. NAVIGATION: DIRECTORY TREE

- `/`: root directory
- `..`: upwards one directory
- `~`: shortcut for `/home/user`
- Specify location as path:
 - Absolute (starting from root)
e.g. `/home/user/Aufgabe1/loesung.txt`
 - Relative (starting from working directory)
e.g. `../Aufgabe1/loesung.txt` oder `Aufgabe1/loesung.txt`
- **C**hange **d**irectory: `cd` `<destination_path>`



2. NAVIGATION: LINUX-DATEISYSTEM

- Print contents of directory:
 - **ls <options>** (details in chapter 3)
- Tips:
 - Tab key for auto completion
 - Arrow up and down to use previous commands
 - **history** shows all previously used commands
 - **clear** to clear all previous terminal outputs





QUESTIONS?





2. NAVIGATION: EXERCISE

- Exercise at [home/myuser/2_navigation/task.txt](#)
- Solution at [home/myuser/2_navigation/solution.txt](#)

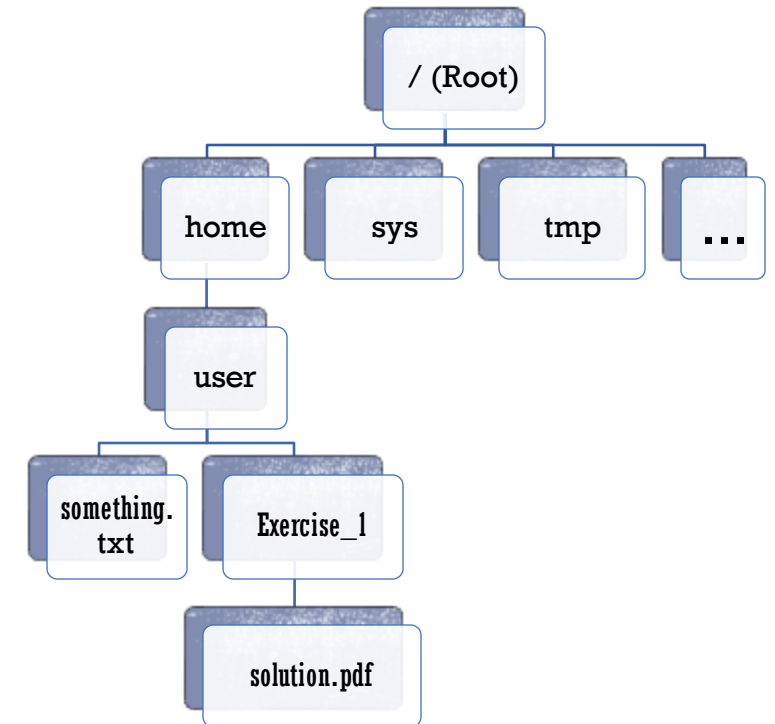
```
myuser@8a1bc344a6ba:~/2_navigation$ cat task.txt
1. Show all contents of this directory
2. Find the file called 'passwd.secret' and look at its content. It
   is located somewhere in this directory and its subdirectories.
3. Navigate to the root of the directory tree.
4. Show all home directories on the machine
5. Show the contents of the directory /bin. What do you see?
6. Change to your home directory using the '~' syntax
7. Navigate to locations of your own choice using absolute and relative
   paths.
8. Use the command history (arrow up and down) to execute some of the
   commands, which you have used previously once again.
9. Print your current location in the directory tree

NOTE: Try to use Tab-completion for all of the exercises.
```



3. FILES AND DIRECTORIES

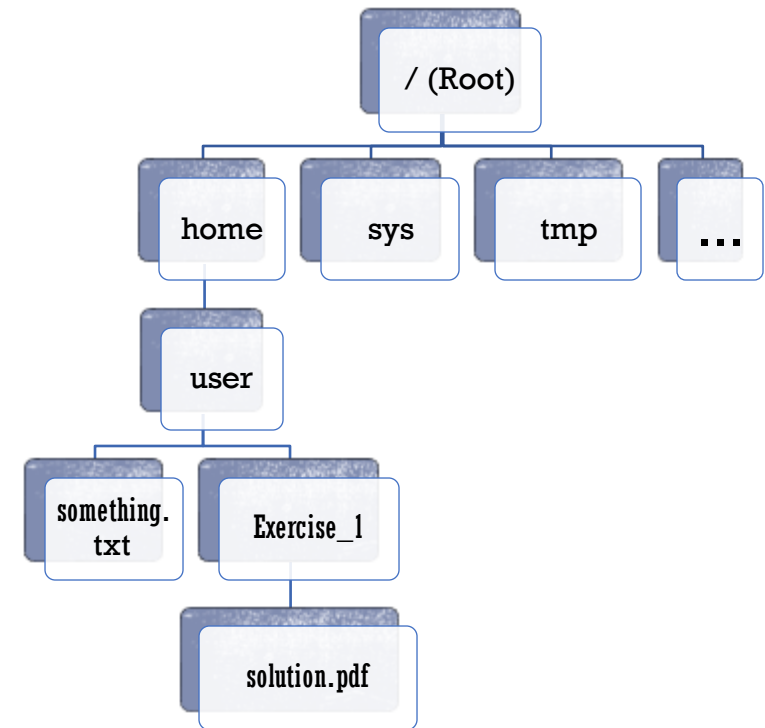
- File names are globally unique
- Files starting with a dot (.) are hidden
- File name extension does not matter
(e.g. with extension .pptx could contain PDF contents)
- `cat <file>` prints contents of a file
- `touch <file>` creates a new empty file
- `mkdir <directory>` creates a new directory
- `rm <file>` deletes a file, `-r` (recursive) option for directories
- `cp <source> <dest>` copies a file, `-r` (recursive) option for directories
- `mv <source> <dest>` copies a file or directory, also used for renaming



3. FILES AND DIRECTORIES

- **ls -l** shows every entry in new line (one column)
- **ls -l** shows a "long listing" with more details, e.g.:

-rw-rw-r-- 1 user user 0 Oct 10 10:22 Test.txt
drwxr-xr-x 2 user user 4096 Oct 10 10:22 Aufgabel
- Organized as:
 - directory, read, write, execute
 - [type] [access permissions: user (owner), group (owner), others] [hard link] [user] [group] [size] [date] [name]
- **ls -A** (All) shows hidden files as well



3. CHANGING ACCESS PERMISSIONS

- `chmod <opt> <Filename>`
- `<opt>` specifies the permissions to be changed:
 - `+x` grants permission to execution for everyone
 - `-x` removes permission to execute for everyone
 - `u+x` grants permission to execute for user `u`
 - `u-x` removes permission to execute for user `u`
- Analogously for groups and other users (`g`roup), andere Nutzer (`o`thers)
- And for other types of permissions (`r`ead, `w`rite)



3. WILDCARDS

- Select more than one file or directory e.g.:

```
rm -r *.pdf
```

-> removes all files or directories ending with .pdf

- Wildcards do not match hidden files (starting with dot)





3. FILES AND DIRECTORIES: EXERCISE

```
myuser@8a1bc344a6ba:~/3_files_and_dirs$ cat task.txt
1. Show all contents of this directory
2. Show all contents of this directory in separate lines including hidden files
3. Show all contents of this directory with their permissions
4. Create a new file called `my_file.txt`
5. Delete the file called `useless.json`
6. Create a directory with the name `new_files`
7. Move all files with the suffix `.new` to the directory `new_files`
8. Remove all files with the suffix `.old`
9. Rename the file `weird_name.txt` to `good_name.txt`
10. Clear all contents of the directory `trash`
11. Use the command `./check.sh` in this directory to verify your results
```

NOTE 1: Wildcards (*) can be helpful

NOTE 2: Watch out for hidden files





QUESTIONS?



4. EDITING FILES WITH VIM

- Vim is a command line editor that is usually available on Linux machines
- `vim <filename>` opens existing file in vim
- Different modes:
 - Command mode (`ESC` key) : initial mode
 - Insert mode (`i` key): edit text
 - `2x d` delete whole line
 - `:q` (`quit`) exit document (if no changes made)
 - `:q!` exit document and discard changes
 - `:wq` (`write, quit`) save changes and exit document



The screenshot shows the Vim editor interface. The top part displays four lines of text: "Test", "Test2", "Test3", and "Test4". The cursor is positioned at the end of the fourth line, "Test4". Below the text, there are several lines of tilde (~) characters, indicating that the text has been wrapped. At the bottom of the screen, the status line shows "-- INSERT --" in the center, "4,5" on the right, and "All" on the far right.





4. VIM: EXERCISE

```
myuser@8a1bc344a6ba:~/4_vim$ cat task.txt
```

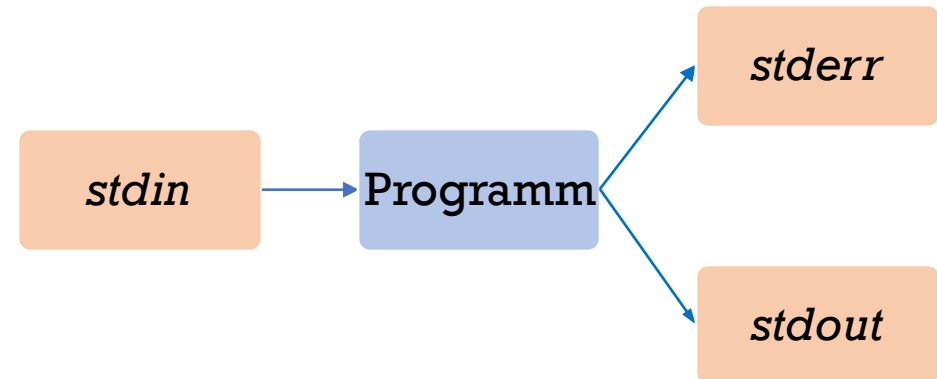
1. Create a file called 'lets_go_vim.txt'
2. Use vim to edit the file 'lets_go_vim.txt' and change its contents to 'Hello Vim!'
3. Look at the modified file using the cat command
4. Use Vim to replace all numbers with dots in the file 'shopping_list.txt' by hyphens (-). E.g. the first line should look like this '- Water'.
5. Delete the entry with cream in the file 'shopping_list.txt'
6. Use the command './check.sh' in this directory to verify your results

NOTE: Do not add any extra white spaces in the files for the check command to work properly.



5. STREAMS AND REDIRECTION

- Programs have 3 data streams
 - Datandard input: ***stdin***
 - Standard output: ***stdout***
 - Standard error: ***stderr***



5. STREAMS AND REDIRECTION

> redirects standard output to a file and overwrites contents

E.g.: `pwd > verzeichnis.txt`

>> same but appends to file

< reads from a file and sends to standard in of process

e.g.: `egrep "subway" < input_file.txt`

| connects stdout of process 1 to stdin of process 2

e.g.: `ls | egrep ".txt"`



5. STREAMS AND REDIRECTION

Useful commands for the exercise:

egrep <pattern> <file> searches a substring in a file and writes all matching lines to *stdout*

wc <file> counts the # of lines, words and characters in a file; options -l, -w, -c for counting only lines, words, characters respectively

- Note:

wc and **grep** read from a file if specified, otherwise from stdin until stream closes





QUESTIONS?





5. STREAMS: EXERCISE

```
myuser@8a1bc344a6ba:~/5_pipes_redirection_search$ cat task.txt
1. Copy the files 'best_recipe.txt.new' and 'morning_routine.txt.new'
from exercise '3_files_and_dirs' to this directory
2. Append the contents of 'best_recipe.txt.new' to the file 'morning_r
outine.txt.new' using output stream redirection
3. Create a file called 'search_results.txt'
4. Search for all lines containing the substring ',Q' in the file 'tit
anic_train.csv' using egrep and append the search result to the file '
search_results.txt'
5. Create a file called 'count.txt'
6. Use the 'egrep' command and the command 'wc -l' to count the number
of lines in the 'titanic_train.csv' containing the substring ',S'. Wr
ite the result to the file 'count.txt'.
7. Use the command './check.sh' in this directory to verify your resul
ts
```

NOTE: The 'wc' command reads from its standard input stream



6. SUBSHELLS

- Execute a command inside of another command using `$(subcommand)`

- E.g.:

```
[myuser@99ac344ce626:~/3_cde$ echo "Your current directory is $(pwd)"]  
Your current directory is /home/myuser/3_cde
```

- Execution from inside out





QUESTIONS?





6. SUBSHELLS: EXERCISE

```
myuser@8a1bc344a6ba:~/6_subshells$ cat task.txt
```

1. Use the echo command, a subshell and the date command to print the text 'It is now <date>', where 'date' is the current date and time. Redirect the output to a file called 'date.txt'
2. Use a subshell and the 'date +%y_%m_%d_%H' command to automatically create a file with the name '<year>_<month>_<day>_<hour>.log', where the in brackets encapsulated variables equal the attributes of the current point in time.
3. Check whether the entries with the ID 121 are equal for the file 'titanic_train_1.csv' and the file 'titanic_train_2.csv' using the egrep command and a subshell. Before you start, use the cat command to get an impression of the data in the files in order to decide how to search for the entry with this ID.
4. Repeat the third task with id 122
5. Use the command './check.sh' in this directory to verify your results for tasks 1 and 2
6. Compare your results for exercises 3 and 4 with those of the person next to you or check the file 'solution.txt'



7. PROCESS MANAGEMENT

- Process = running program instance
 - Processes are independent
 - Multiple processes of on program possible e.g. multiple firefox windows open
- In the terminal, processes can run in the foreground or background
 - Foreground (default): blocks shell, shell is busy until program exits
 - Background: are disconnected from shell (asynchronous)



7. BACKGROUND PROCESSES

- **&** after a command executes it as background process

e.g.: **sleep 60 &**

```
myuser@99ac344ce626:~/3_cde$ sleep 60 &  
[1] 325  
myuser@99ac344ce626:~/3_cde$ jobs  
[1]+  Running                  sleep 60 &  
myuser@99ac344ce626:~/3_cde$
```

- **jobs** shows background processes started from this shell
- **fg <jobID>** pulls a process to the foreground
- **bg <jobID>** resumes a process in the background
- **Ctrl + Z** pauses a foreground process
- **Ctrl + C** stops a foreground process





QUESTIONS?





AUFGABE 7:

PROZESSMANAGEMENT

NOTE: In Docker for Windows, there is a bug with CTR+Z and CTRL+C, so you might not be able to complete the entire exercise

```
myuser@8a1bc344a6ba:~/7_process_management$ cat task.txt
1. execute the command '$(sleep 180 ; echo "finished at $(date)" > finished.txt)' as a background process
2. Save the current output of the command 'jobs' to a file called 'running.txt'
3. Pull the process started earlier into the foreground
4. Stop the process (Do not cancel it but stop it temporarily!
5. Save the current output of the command 'jobs' to a file called 'stopped.txt'
6. Resume the process in the background
7. Wait until the background process has finished and try to understand what happened.
8. Use the command './check.sh' in this directory to verify your results for tasks 1-6
9. Discuss your results of task 7 with the person next to you or verify your hypothesis using the file 'solutions.txt'
```

NOTE: Unfortunately the Ctrl+Z does not work correctly with Docker on Windows. Therefore, if you are running Windows, you cannot fully complete this exercise.



8. INSTALL PROGRAMS WITH APT

- There are many preinstalled programs: `ls`, `pwd`, `cat` ...
- `apt` is a possibility to install new programs from the CL
 - Packet manager
 - Maintains a list of all installed and installable programs on the computer
 - `apt update` updates program lists (recommended before each install!)
 - `apt upgrade` updates all installed programs
 - `apt install <package>` installes a program
 - `apt remove <package>` deinstalls a program
- Allows automatic installation of programs
- apt requires `sudo` command (SuperUserDo)
- Not all programs are available to install with apt





QUESTIONS?





8. INSTALL PROGRAMS WITH APT:

EXERCISE

NOTE: The password for the user 'myuser' is 'secret'

```
myuser@8a1bc344a6ba:~/8_apt$ cat task.txt
```

1. Install the program 'unzip' using apt
2. Unzip the zip-archive 'big_data.zip' using the program unzip
3. Uninstall the program 'unzip' using apt
4. Use the command './check.sh' in this directory to verify your results

NOTE 1: Installing and uninstalling requires administrator rights (sudo)

NOTE 2: The password for the super user (sudo) is 'secret'



9. AUTOMATION WITH SCRIPTS

- A script is a non-compiled program (i.e. just-in-time interpreted code)
 - Bash defines its own programming language
 - Simple programs only contain commands, but loops, conditions ... are possible
 - Allows automating repetitive tasks
 - Scripts must have the execute permission set (remember chapter 3)





QUESTIONS?





9. AUTOMATION WITH SCRIPTS

EXERCISE

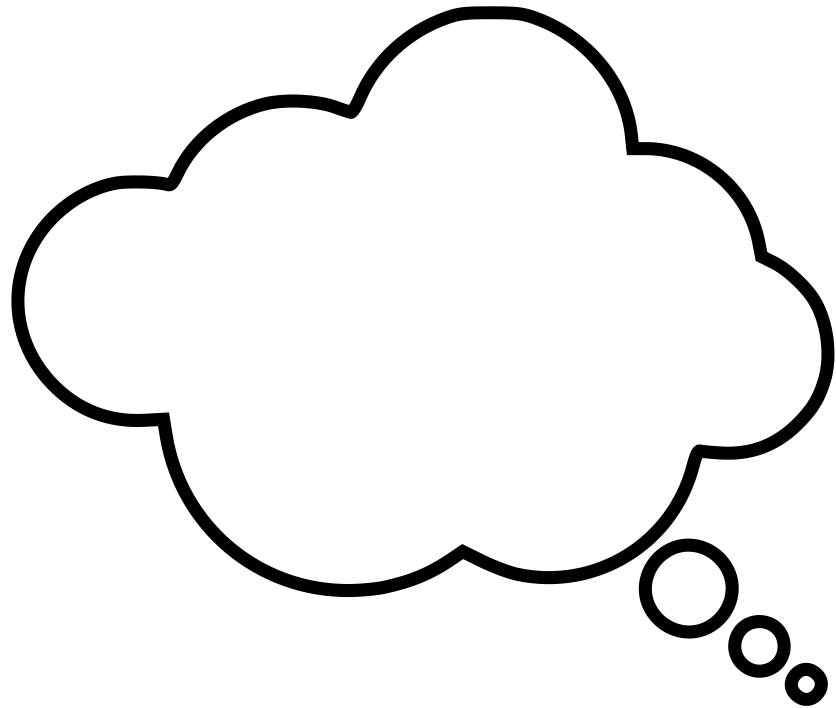
```
myuser@8a1bc344a6ba:~/9_scripting$ cat task.txt
This directory is an exact copy of the directory '3_files_and_dirs'.
This time it is your task to create a script, which completes the exercise for you.
As a reminder, the tasks of exercise 3, which should be automated in this exercise, are:
```

- Show all contents of this directory
- Show all contents of this directory in separate lines including hidden files
- Show all contents of this directory with their permissions
- Create a new file called `my_file.txt`
- Delete the file called `useless.json`
- Create a directory with the name `new_files`
- Move all files with the suffix `.new` to the directory `new_files`
- Remove all files with the suffix `.old`
- Rename the file `weird_name.txt` to `good_name.txt`
- Clear all contents of the directory `trash`

Bonus: If that is fun, implement another script for one of the other exercises.

NOTE: If you need to reset the directory to test your script multiple times, then use the `./reset 9_scripting` command in your home directory. If you do that you might want to backup your script somewhere outside of this directory, so it will not get deleted during the reset process.





FEEDBACK

Help us improving this seminar
by sharing your opinion.

