Tools SS 2022

Introduction to Linux

Florian Beenen

Schaltungstechnik und Simulation

01.06.2022

Contents I

Contents II

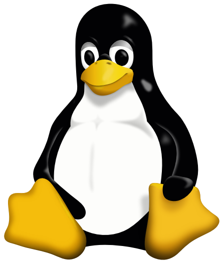
Introduction

What is Linux

- A derivative of a Unix operating system.
- Other types of Unix systems:
  - Solaris
  - MacOS X
  - BSD
  - IBM AIX
- Consists of ...
  - A kernel: Implements specific APIs, provides system calls, a file system, a networking stack and much more.
  - A set of optional programs:
    - A shell: Execute commands.
    - A graphical window subsystem: Displays windows.
    - Compilers and runtime environments.
    - ...

Linux Distributions

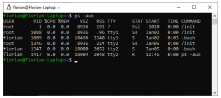
- A Linux distribution contains the Linux kernel and a lot of different applications.
- Some distributions are meant for headless server operation.
- Some are meant for desktop application and include a graphical user interface.
- Some have a focus on stability of the applications, other focus "bleeding-edge" software versions.
- Selection of common distributions:
  - Debian
  - Ubuntu (based on Debian)
  - Mint (based on Ubuntu)
  - Fedora
  - CentOS
  - Arch Linux
  - ...



https://de.wikipedia.org/wiki/Datei:Tux.svg

How to Work with Linux

- Linux is a multi-user system: ➥ Inherently supports multiple concurrent users.
- Most important tool to do something: The **terminal** / the **shell**.
- Terminal: Old expression to describe a terminal device that is used to send and receive command. The terminal itself has no computing power – it is hooked up to e. g. a mainframe in the basement.
- Terminal today: A program that provides a command prompt. May even be in a graphical window.
- You can run commands or scripts with the terminal.
- Different shell versions exist with slight variations (➥ see later).
- The Linux shell is easier to use and much more powerful than Windows `cmd.exe`.

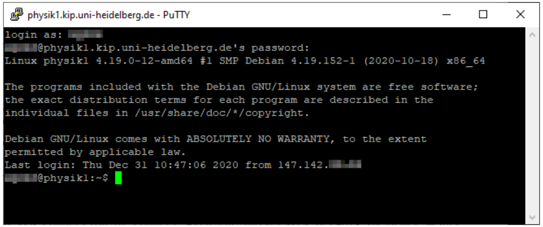


```
florian@Florian-Laptop:~$ ps -aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   8936   192 ?        Ssl   2020   0:00 /init
root      1088  0.0  0.0   8936    96 tty1     Ss   Jan02   0:00 /init
florian   1089  0.0  0.0  18496  3340 tty1     S    Jan02   0:03 -bash
root      1346  0.0  0.0   8936   224 tty2     Ss   Jan05   0:00 /init
florian   1347  0.0  0.0  18080  3652 tty2     S    Jan05   0:00 -bash
florian   1417  0.0  0.0  18904  2048 tty2     R    12:46   0:00 ps -aux
florian@Florian-Laptop:~$ _
```

https://de.wikipedia.org/wiki/Datei:Televideo925Terminal.jpg.

Working Environment

- If you do not have Linux on your system, you can log into a server at university.
- Open a VPN connection or connect with the university network by other means.
- Get an SSH client like PuTTY: `https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe`.
- Use PuTTY to connect to `physik1.kip.uni-heidelberg.de`.
- Log in with your Uni-ID as username (e. g. `jb007`) and the corresponding password.
- You get a shell on the remote host system.



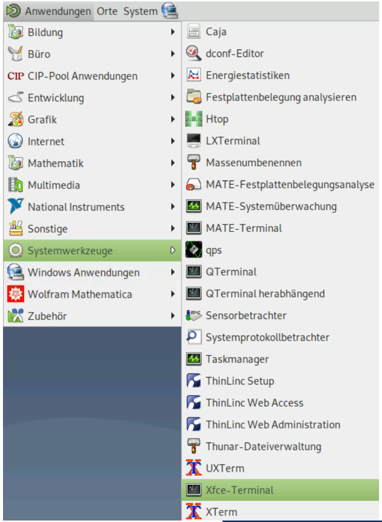
```
login as: [redacted]
[redacted]@physik1.kip.uni-heidelberg.de's password:
Linux physik1 4.19.0-12-amd64 #1 SMP Debian 4.19.152-1 (2020-10-18) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Dec 31 10:47:06 2020 from 147.142.[redacted]
[redacted]@physik1:~$
```

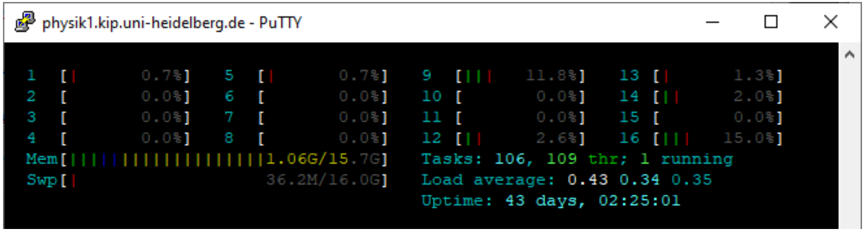
Working Environment (2)

- If you are in the CIP pool, you can open a terminal on the desktop.
- Click *Anwendungen* $\rightarrow$ *Systemwerkzeuge* $\rightarrow$ *Xfce-Terminal*.



Working Environment (2)

- KIP-Server has Debian installed.
- The default shell is `bash`.
- You can use the shell to issue commands. We will get to know some of them in the next sections.
- You can access your files linked with your Uni-ID.
- KIP machines are quite old – please do not run compute-intensive workload ☺.



```
  1  [|        0.7%]   5  [|        0.7%]   9  [|||    11.8%]   13 [|        1.3%]
  2  [         0.0%]   6  [         0.0%]   10 [        0.0%]   14 [||       2.0%]
  3  [         0.0%]   7  [         0.0%]   11 [        0.0%]   15 [         0.0%]
  4  [         0.0%]   8  [         0.0%]   12 [||      2.6%]   16 [|||     15.0%]
  Mem[|||||||||||||||||||1.06G/15.7G]   Tasks: 106, 109 thr; 1 running
  Swp[|               36.2M/16.0G]   Load average: 0.43 0.34 0.35
                                        Uptime: 43 days, 02:25:01
```

File System

File System in Linux

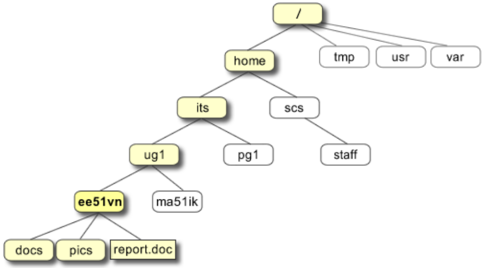
- Preface: Microsoft Windows
  - In Windows you have different drives (visible as `C:\`, `D:\`, ...).
  - Every file needs to be specified with respect to a drive. There is no *global* file root.
- In Linux everything is organized in a hierarchical way!
- All paths can either be specified relative to the current directory or absolute by referencing the **file root**.
- The root is indicated by the "slash character" `/`.
- Paths can also be specified relative to the user's home directory, which is indicated by the tilde `~`.
- To establish a true hierarchical namespace, every object must implement the file API. In other words: Everything is a file.

File Types

- Regular files (`-`):
  - Text files, ASCII style or equivalent.
  - Binary files like pictures, programs, videos, …
- Directories (`d`)
- Block files that represent block devices, e. g. HDDs or SSDs (`b`).
- Character files: Special input devices like the computer mouse or the terminal (`c`).
- Pipes (`p`) and Sockets (`s`): Means for inter-process communication.
- Links (`l`): Ways to point to another file object.

Browsing the Directory Tree

- With the hierarchical structure, a tree is established.
- Directories contain other directories and may also contain files.

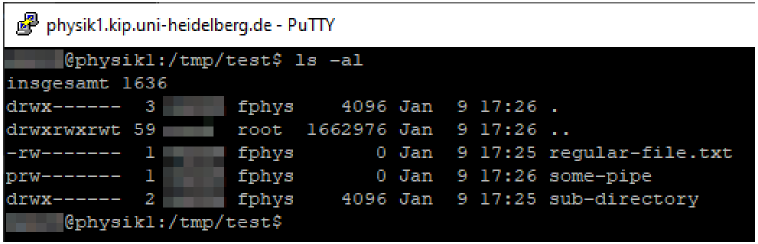


➥ Absolute path of the file `report.doc` is: `/home/its/ug1/ee51vn/report.doc`

http://www.ee.surrey.ac.uk/Teaching/Unix/unixintro.html

Browsing the Directory Tree (2)

- In each directory there are two special files that are available by default and cannot be removed.
- Special file `.` : Represents *this* directory.
- Special file `..` : Represents the parent directory of this directory.
- Special files are required to navigate the directory tree and to know where we are.



physik1.kip.uni-heidelberg.de - PuTTY

```
[redacted]@physik1:/tmp/test$ ls -al
insgesamt 1636
drwx------  3 [redacted] fphys    4096 Jan  9 17:26 .
drwxrwxrwt 59 [redacted] root  1662976 Jan  9 17:26 ..
-rw-------  1 [redacted] fphys       0 Jan  9 17:25 regular-file.txt
prw-------  1 [redacted] fphys       0 Jan  9 17:26 some-pipe
drwx------  2 [redacted] fphys    4096 Jan  9 17:25 sub-directory
[redacted]@physik1:/tmp/test$
```

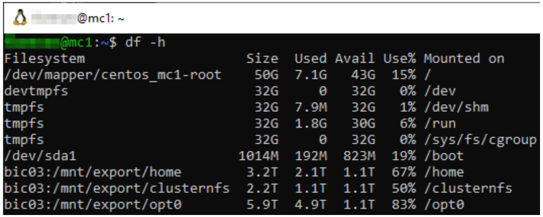
Browsing the Directory Tree (3)

Two commands to browse through the files with the shell:

- `cd`: Change directory.
  - The command requires one argument: The target **directory**, either as absolute or relative path.
  - Changes the current working directory to the given directory.
  - Special arguments:
    - `cd` (without any parameters): Switches to the user's home directory.
    - `cd -`: Switches to the previous working directory.
- `ls`: List directory contents.
  - Can be executed without any arguments to show list of files in current directory.
  - Can be given a list of directories to inspect instead: `ls /tmp /home/` lists the contents of `/home` and `/tmp`.
  - Parameters control output behavior:
    - `ls -a`: Show all files. Files that start with a dot (e. g. `.textfile.txt`) are considered hidden and normally not shown.
    - `ls -l`: Show files in a list with more info.
    - `ls -h`: Show file sizes in a human-readable syntax instead of byte count.
    - `ls -lah`: All of the above.

File System Usage

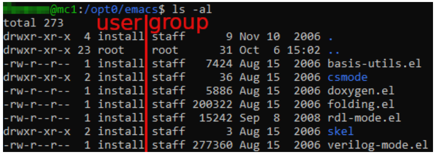
- If you need to find out how large a directory is, use `du` (“Directory Usage”?).
  - Useful parameter `du -h`: Print sizes in human-readable syntax instead of byte count.
- To check how full your storage medium is, use `df` (“Disk free”).
  - `df -h` extremely useful to read the output properly.
  - `df` also gives you an overview of all externally mounted files (additional hard drives, network shares, ...).



```
[redacted]@mc1: ~
[redacted]@mc1:~$ df -h
Filesystem                     Size  Used Avail Use% Mounted on
/dev/mapper/centos_mc1-root     50G  7.1G   43G  15% /
devtmpfs                        32G     0   32G   0% /dev
tmpfs                           32G  7.9M   32G   1% /dev/shm
tmpfs                           32G  1.8G   30G   6% /run
tmpfs                           32G     0   32G   0% /sys/fs/cgroup
/dev/sda1                     1014M  192M  823M  19% /boot
bic03:/mnt/export/home         3.2T  2.1T  1.1T  67% /home
bic03:/mnt/export/clusternfs   2.2T  1.1T  1.1T  50% /clusternfs
bic03:/mnt/export/opt0         5.9T  4.9T  1.1T  83% /opt0
```

File Permissions

- Every file or directory has an owning **user** and a **group**.
- Ownerships can be checked with `ls -l`: First name is the user, second name is the group.
- Permissions are split into three segments:
  - User permissions.
  - Group permissions.
  - Other permissions (people where user or group do not match).
- Every segment has three permission attributes:
  - Read permission (`r`, Code 4).
  - Write Permission (`w`, Code 2).
  - Execute Permission (`x`, Code 1).



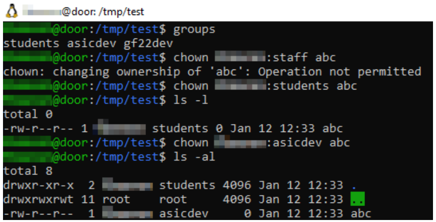
```
[redacted]@mc1:/opt0/emacs$ ls -al
total 273
drwxr-xr-x  4 install staff      9 Nov 10  2006 .
drwxr-xr-x 23 root    root      31 Oct  6 15:02 ..
-rw-r--r--  1 install staff   7424 Aug 15  2006 basis-utils.el
drwxr-xr-x  2 install staff     36 Aug 15  2006 csmode
-rw-r--r--  1 install staff   5886 Aug 15  2006 doxygen.el
-rw-r--r--  1 install staff 200322 Aug 15  2006 folding.el
-rw-r--r--  1 install staff  15242 Sep  8  2008 rdl-mode.el
drwxr-xr-x  2 install staff      3 Aug 15  2006 skel
-rw-r--r--  1 install staff 277360 Aug 15  2006 verilog-mode.el
```

Permission Codes

- Reminder: Read (4), Write (2), Execute (1).
- Codes are used as bit set:
  - Permission of 0: Nothing allowed (except maybe deletion, depending on directory permission).
  - Permission of 3: Write + Execute.
  - Permission of 5: Read + Execute.
  - Permission of 6: Read + Write.
  - Permission of 7: Read + Write + Execute.
- Permission string is first column in `ls -l`: $\underbrace{\texttt{-}}_{\text{type}}\underbrace{\texttt{rwx}}_{\text{user}}\underbrace{\texttt{r--}}_{\text{group}}\underbrace{\texttt{r--}}_{\text{other}}$. Corresponding code: 744.
- Permissions can be changed via the `chmod` command (change file mode bits).
  - `chmod 755 myfile.txt` sets permissions to 755, `-rwxr-xr-x`.
  - `chmod o-x myfile.txt` removes `x` permission from "other".
  - `chmod u+w myfile.txt` adds `w` permissions to "user".
  - `chmod g+r,o-r myfile.txt` adds `r` permissions to "group" and removes `r` permissions from "other".

Changing Ownership

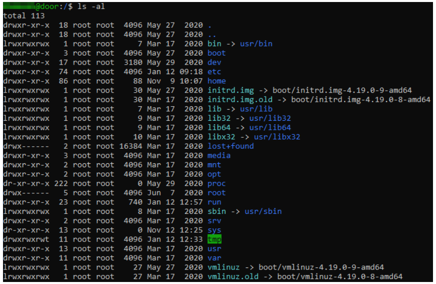
- To change ownership of a file or directory, use the `chown` command.
  - `chown install:staff myfile.txt` changes the ownership to the user `install` and the group `staff`.
  - `chown root. myfile.txt` changes the ownership to the user `root` and the group `root`.
- To give a file to another user, you need to be `root` (the admin user which is allowed to do everything).
- To give a file to another group, you need to be in that group (or be `root`).
- You can find out in which groups you are with the command `groups`.



```
[redacted]@door: /tmp/test
[redacted]@door:/tmp/test$ groups
students asicdev gf22dev
[redacted]@door:/tmp/test$ chown [redacted]:staff abc
chown: changing ownership of 'abc': Operation not permitted
[redacted]@door:/tmp/test$ chown [redacted]:students abc
[redacted]@door:/tmp/test$ ls -l
total 0
-rw-r--r-- 1 [redacted] students 0 Jan 12 12:33 abc
[redacted]@door:/tmp/test$ chown [redacted]:asicdev abc
[redacted]@door:/tmp/test$ ls -al
total 8
drwxr-xr-x  2 [redacted] students 4096 Jan 12 12:33 .
drwxrwxrwt 11 root     root     4096 Jan 12 12:33 ..
-rw-r--r--  1 [redacted] asicdev     0 Jan 12 12:33 abc
```

Linking

- You can create a “pseudo-file” that points to another file (➥ a link).
- Useful when the same file is needed at multiple locations: No need to copy the file.
- A linked file does not require any additional storage space (except for some book-keeping meta data).
- Two types of links:
  - Hard Link:
    - Create with `ln sourcefile.txt /other/dir/targetfile.txt`.
    - Hard-linked files are not distinguishable.
    - Extremely hard to keep track of linked files ☹.
    - Can only be made inside a single file system (e. g. not over two partitions).
  - Soft Link:
    - Create with `ln -s /dir/sourcefile.txt /other/dir/targetfile.txt`.
    - When source file is deleted, all links point to no file.
    - Can span multiple file systems.
    - Linked files have a special file type (Code `l`).



```
[redacted]@door:/$ ls -al
total 113
drwxr-xr-x  18 root root  4096 May 27  2020 .
drwxr-xr-x  18 root root  4096 May 27  2020 ..
lrwxrwxrwx   1 root root     7 Mar 17  2020 bin -> usr/bin
drwxr-xr-x   3 root root  4096 May 27  2020 boot
drwxr-xr-x  17 root root  3180 May 29  2020 dev
drwxr-xr-x  74 root root  4096 Jan 12 09:18 etc
drwxr-xr-x  86 root root    88 Nov  9 10:07 home
lrwxrwxrwx   1 root root    30 May 27  2020 initrd.img -> boot/initrd.img-4.19.0-9-amd64
lrwxrwxrwx   1 root root    30 Mar 17  2020 initrd.img.old -> boot/initrd.img-4.19.0-8-amd64
lrwxrwxrwx   1 root root     7 Mar 17  2020 lib -> usr/lib
lrwxrwxrwx   1 root root     9 Mar 17  2020 lib32 -> usr/lib32
lrwxrwxrwx   1 root root     9 Mar 17  2020 lib64 -> usr/lib64
lrwxrwxrwx   1 root root    10 Mar 17  2020 libx32 -> usr/libx32
drwx------   2 root root 16384 Mar 17  2020 lost+found
drwxr-xr-x   3 root root  4096 Mar 17  2020 media
drwxr-xr-x   2 root root  4096 Mar 17  2020 mnt
drwxr-xr-x   2 root root  4096 Mar 17  2020 opt
dr-xr-xr-x 222 root root     0 May 29  2020 proc
drwx------   5 root root  4096 Jun  7  2020 root
drwxr-xr-x  23 root root   740 Jan 12 12:57 run
lrwxrwxrwx   1 root root     8 Mar 17  2020 sbin -> usr/sbin
drwxr-xr-x   2 root root  4096 Mar 17  2020 srv
dr-xr-xr-x  13 root root     0 Nov 12 12:25 sys
drwxrwxrwt  11 root root  4096 Jan 12 12:33 tmp
drwxr-xr-x  13 root root  4096 Mar 17  2020 usr
drwxr-xr-x  11 root root  4096 Mar 17  2020 var
lrwxrwxrwx   1 root root    27 May 27  2020 vmlinuz -> boot/vmlinuz-4.19.0-9-amd64
lrwxrwxrwx   1 root root    27 Mar 17  2020 vmlinuz.old -> boot/vmlinuz-4.19.0-8-amd64
```

Basic Commands

Shell Overview

- Shell shows the prompt: `user@myhost:~$`
  - Your current user name.
  - The host name of the system that you are using.
  - The current working directory (user home: ~).
  - The $ indicates a user shell (# would indicate a `root` shell).
  - All of this is configurable; the default depends on your Linux installation (Debian, CentOS, ...)
- When you have executed several commands, you can navigate through prior commands with the arrow keys: ↑ , ↓ .
- You can auto-complete commands and directory names with the ⇆ key (press twice). Use this whenever possible!
- You can reverse-search through already executed commands (e. g. look for patterns) by pressing Ctrl + R and then type your search-phrase. Also use this whenever possible.

Getting Help

- Sometimes you don't know how to use a command.
- You can either google "... how to do XY in Linux?"...
- ... or you can use the build-in help!
- Most programs have a parameter `-h` or `--help` (e. g. `ls --help`) that gives a short overview.
- If you need more documentation, use `man COMMANDNAME` to find out about a certain command (e. g. `man ls` to learn about `ls`).
  - `man` can do much more than you think!
  - First of all, there is a man-page about `man` (execute `man man`, obviously).
  - You can use `man` to learn about C-functions (e. g. `man 3 sprintf`).

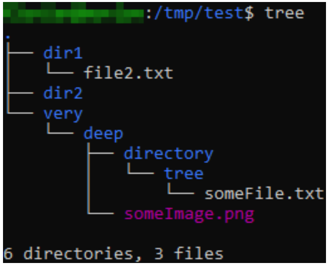
```
       The table below shows the section numbers of the manual followed by the types of pages they contain.
       1   Executable programs or shell commands
       2   System calls (functions provided by the kernel)
       3   Library calls (functions within program libraries)
       4   Special files (usually found in /dev)
       5   File formats and conventions eg /etc/passwd
       6   Games
       7   Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
       8   System administration commands (usually only for root)
       9   Kernel routines [Non standard]
```

File Commands

- We already know `cd` and `ls`.
  - Remember `cd` (without parameters) to switch to user home.
  - Remember `cd -` to switch to previous directory.
  - Remember tilde-notation (˜) as short-hand notation for home-directory (e. g. `ls ~/Downloads`).
- Print working directory: `pwd`
  - Get absolute path of your current working directory.
  - Particularly useful in scripts to find out where the user currently is.
- Create directory (“make directory”): `mkdir`
  - Needs an argument to specify the directory to create.
  - Can use relative or absolute path specification.
  - Useful parameter: `-p`. Also creates all non-existing sub-directories (e. g. `mkdir -p /tmp/some/long/dir/tree/that/is/deep`).

File Commands (2)

- Remove files / directories: `rm`
  - `rm file.txt` deletes the file `file.txt`.
  - `rm *.txt` deletes all files that end in `.txt`.
  - `rm -f file.txt` deletes the file `file.txt` and does not ask for confirmation if the file is protected (but deletable).
  - `rm -r myDir` deletes the directory `myDir` and everything that is in it.
  - `rm -rf myDir` deletes the directory `myDir` without asking back (can be more dangerous when combined with `sudo`).
- Get a directory tree: `tree`
  - Requires additional program (`sudo apt install tree`).
  - Print a graphical representation of the directory tree on the command line.



```
[redacted]:/tmp/test$ tree
.
├── dir1
│   └── file2.txt
├── dir2
└── very
    └── deep
        ├── directory
        │   └── tree
        │       └── someFile.txt
        └── someImage.png

6 directories, 3 files
```

File Commands (3)

- Create an empty file: `touch`
  - Use like `touch filename.txt`.
  - If file does not exist, `touch` will create it; it will be empty.
  - If file exists, `touch` will update the *last modified* timestamp (check with `ls`).
- Copy a file: `cp`
  - Use like `cp source.txt dest.txt`.
  - Will overwrite destination file if it exists already.
  - Copy entire directory: `cp -r srcDir /some/where/destDir`.
- Rename a file (“move”): `mv`
  - Use like `mv old.txt new.txt`.
  - Will overwrite destination file if it exists already.
  - You can also rename directories: `mv oldDir /some/where/newDir`.

Editing Files

- Write file to command line / “concatenate”: `cat`
  - Intention: Merge/concatenate multiple files and print result on command line.
  - Usage: `cat file1.txt file2.txt` or even `cat *.txt`.
  - Often used to have a quick glance at a small file.
- Read a file: `less` / `more`
  - `more` does the same as `less`; `less` can do more ☺.
  - Use like `less file.txt`.
  - You can navigate with arrow keys and space bar.
  - Quit by pressing `q`.
- Print first / last lines of file: `head` or `tail`
  - `head -20 file.txt` prints first 20 lines of the given file.
  - `tail -100 file.txt` prints last 100 lines of the given file.
  - `head` (without parameters) reads from `stdin` and then prints out the first 10 lines.

Editing Files (2)

- Count words / lines: `wc`
  - `wc file.txt` displays number of **lines**, **words** and **bytes** respectively.
  - `wc -l file.txt` only displays number of lines.
  - `wc -l` (no file name) will read from `stdin` and then count the number of lines.
- Edit a file: `nano`
  - Easy-to-use text editor.
  - Open or create a file with `nano file.txt`.
  - Do whatever you want.
  - Press Ctrl + X to quit. Confirm with Y + Enter.

Editing Files (3)

- More complicated editor: `vim`
  - Do not use before you read a tutorial!
  - Can dramatically increase your productivity **if** you know how to use it properly.
  - **Will** dramatically decrease your productivity if you don't know how to use it and just want to show off to your colleagues.
  - You can exit `vim` by pressing Esc numerous times, then write `:q!` + Enter
  - There are probably some people among us who will claim "`vim` is best, no one can live without it". I can live without it (most of the time) ☺.

How do you generate a random string? Put a web designer in front of VIM and tell him to save and exit.

https://comic.browserling.com/extra/36

Searching

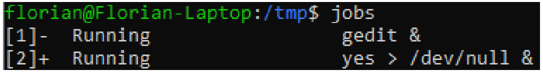
- Look for patterns in files ("Global Regular Expression Print"): `grep`
  - `grep "foobar" *.txt` will print all lines from all `.txt` files in the current directly that contain the term "foobar".
  - `grep -i "foobar" *.txt` will ignore the case – therefore "fOObAr" will also be found.
  - `grep -r "foobar" .` will look for "foobar" in **all** files and subdirectories of the current directory (remember special directory `.`).
  - `grep -E "key=[0-9]+" file.txt` will read `file.txt` and look for lines that contain `key=` followed by a sequence of characters in the range `0` to `9` that is at least one character long. You will get all lines that match as a result.
  - `grep -E -o "key=[0-9]+" file.txt` will do the same, except that you **only** get the matching section. If the file contains the line "Hello 123 key=456 other" you will get `key=456`.
  - `egrep` is the same as `grep -E`.

Searching (2)

- Looking for files: `find`
  - `find . -name "important.txt"` searches for the file `important.txt` in this directory and all subdirectories.
  - Prints all files that match.
  - Does not look *into* the files.
  - `find . -name "important.txt" -exec cat {} \;` executes `cat` on all files that were found. You can combine this with any other command (e. g. `rm` to delete all files found. Useful to clear `Thumbs.db` files ☺).

Multitasking

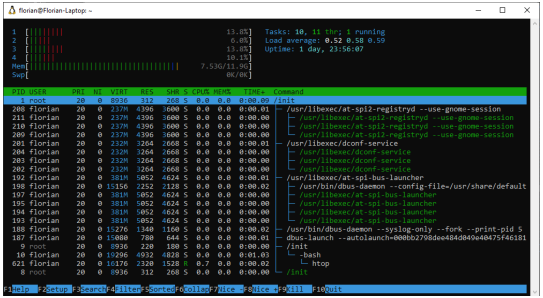
- You can start a command to run in the background – your shell can immediately be reused.
- Issue background command with an ampersand (`&`) after your command.
- Example with graphical editor: `gedit myfile.txt &`
- Micro-Tool to stress one CPU core: `yes > /dev/null &` (do **not** execute `yes` without redirecting output to `/dev/null`!)
- List jobs in the background: `jobs`
- Re-gain control of the jobs listed: `fg 1` (for the first job).
- Drop control again: Ctrl + Z , followed by `bg`.



```
florian@Florian-Laptop:/tmp$ jobs
[1]-  Running                 gedit &
[2]+  Running                 yes > /dev/null &
```

Task Hierarchy

- In Linux all tasks/processes are launched from a parent process.
- The `init` process has the PID 1 and launches other processes.
- If the parent process terminates, all child processes will terminate as well.
- Application:
  - You establish a SSH connection to a machine and launch a ton of processes.
  - If you close the SSH session, all launched programs will terminate (under normal circumstances).



florian@Florian-Laptop: ~

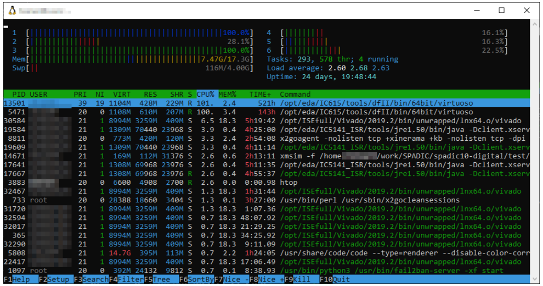
```
  1  [||||||||                                 13.8%]   Tasks: 10, 11 thr; 1 running
  2  [|||||                                     6.0%]   Load average: 0.52 0.58 0.59
  3  [||||||||                                 13.8%]   Uptime: 1 day, 23:56:07
  4  [||||||                                   10.1%]
  Mem[||||||||||||||||||||||||||||||||||   7.53G/11.9G]
  Swp[                                         0K/0K]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
    1 root       20   0  8936   312   268 S  0.0  0.0  0:00.09 /init
  208 florian    20   0  237M  4396  3600 S  0.0  0.0  0:00.01 ├─ /usr/libexec/at-spi2-registryd --use-gnome-session
  211 florian    20   0  237M  4396  3600 S  0.0  0.0  0:00.00 │  ├─ /usr/libexec/at-spi2-registryd --use-gnome-session
  210 florian    20   0  237M  4396  3600 S  0.0  0.0  0:00.00 │  ├─ /usr/libexec/at-spi2-registryd --use-gnome-session
  209 florian    20   0  237M  4396  3600 S  0.0  0.0  0:00.00 │  └─ /usr/libexec/at-spi2-registryd --use-gnome-session
  201 florian    20   0  232M  3264  2668 S  0.0  0.0  0:00.01 ├─ /usr/libexec/dconf-service
  204 florian    20   0  232M  3264  2668 S  0.0  0.0  0:00.00 │  ├─ /usr/libexec/dconf-service
  203 florian    20   0  232M  3264  2668 S  0.0  0.0  0:00.00 │  ├─ /usr/libexec/dconf-service
  202 florian    20   0  232M  3264  2668 S  0.0  0.0  0:00.00 │  └─ /usr/libexec/dconf-service
  192 florian    20   0  381M  5052  4624 S  0.0  0.0  0:00.01 ├─ /usr/libexec/at-spi-bus-launcher
  198 florian    20   0 15156  2252  2128 S  0.0  0.0  0:00.02 │  ├─ /usr/bin/dbus-daemon --config-file=/usr/share/default
  197 florian    20   0  381M  5052  4624 S  0.0  0.0  0:00.00 │  ├─ /usr/libexec/at-spi-bus-launcher
  195 florian    20   0  381M  5052  4624 S  0.0  0.0  0:00.00 │  ├─ /usr/libexec/at-spi-bus-launcher
  194 florian    20   0  381M  5052  4624 S  0.0  0.0  0:00.00 │  ├─ /usr/libexec/at-spi-bus-launcher
  193 florian    20   0  381M  5052  4624 S  0.0  0.0  0:00.00 │  └─ /usr/libexec/at-spi-bus-launcher
  188 florian    20   0 15276  1340  1160 S  0.0  0.0  0:00.02 ├─ /usr/bin/dbus-daemon --syslog-only --fork --print-pid 5
  187 florian    20   0 15080   780   644 S  0.0  0.0  0:00.01 ├─ dbus-launch --autolaunch=000bb2798dee484d049e40475f46181
    9 root       20   0  8936   220   180 S  0.0  0.0  0:00.00 ├─ /init
   10 florian    20   0 19296  4932  4828 S  0.0  0.0  0:01.03 │  └─ -bash
  621 florian    20   0 16176  2320  1528 R  0.7  0.0  0:00.02 │     └─ htop
    8 root       20   0  8936   312   268 S  0.0  0.0  0:00.00 └─ /init

F1Help  F2Setup F3SearchF4FilterF5SortedF6CollapF7Nice -F8Nice +F9Kill  F10Quit
```

Identifying Tasks

- With `htop` you can get an overview of your system's resources.
- On some systems `htop` is not installed.
- Very useful features:
  - Sort tasks by CPU utilization or RAM usage (click on `CPU%` or `MEM%`).
  - Check out what other users are doing on your system.
  - Kill / Terminate tasks that are unresponsive with `F9`.
  - Switch between tree and list view with the hotkey `t`.



```
[redacted]
  1  [|||||||||||||||||||||||||||||||||||||||||||||100.0%]   4  [|||||||||                                16.1%]
  2  [|||||||||||||||                             28.1%]   5  [|||||||||||                              16.3%]
  3  [|||||||||||||||||||||||||||||||||||||||||||||100.0%]   6  [|||||||||||||                            22.5%]
  Mem[||||||||||||||||||||||||||||||||||||||||7.47G/17.3G]   Tasks: 293, 578 thr; 4 running
  Swp[||                                      116M/4.00G]   Load average: 2.60 2.68 2.63
                                                             Uptime: 24 days, 19:48:44

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
13501 [redacted] 39  19 1104M  428M  229M R 101.  2.4     521h /opt/eda/IC615/tools/dfII/bin/64bit/virtuoso
 5471 [redacted] 20   0 1108M  610M  207M R 100.  3.4     143h /opt/eda/IC615/tools/dfII/bin/64bit/virtuoso
30584 [redacted] 21   1 8994M 3259M  409M S  6.5 18.3  5h19:42 /opt/ISEfull/Vivado/2019.2/bin/unwrapped/lnx64.o/vivado
19584 [redacted] 21   1 1309M 70440 23968 S  3.9  0.4  4h25:00 /opt/eda/IC5141_ISR/tools/jre1.50/bin/java -Dclient.xserv
 8811 [redacted] 20   0  773M  420M  120M S  3.3  2.4  2h54:08 x2goagent -nolisten tcp +xinerama +kb -nolisten tcp -dpi
19609 [redacted] 21   1 1309M 70440 23968 S  3.3  0.4  4h11:14 /opt/eda/IC5141_ISR/tools/jre1.50/bin/java -Dclient.xserv
14671 [redacted] 21   1  169M  112M 31376 S  2.6  0.6  2h13:11 xmsim -f /home[redacted]/work/SPADIC/spadic10-digital/test/
17641 [redacted] 21   1 1308M 69968 23976 S  2.6  0.4  5h11:35 /opt/eda/IC5141_ISR/tools/jre1.50/bin/java -Dclient.xserv
17667 [redacted] 21   1 1308M 69968 23976 R  2.6  0.4  4h55:37 /opt/eda/IC5141_ISR/tools/jre1.50/bin/java -Dclient.xserv
 3883 [redacted] 20   0  6600  4908  2700 R  2.6  0.0  0:00.98 htop
32467 [redacted] 21   1 8994M 3259M  409M S  1.3 18.3  1h31:44 /opt/ISEfull/Vivado/2019.2/bin/unwrapped/lnx64.o/vivado
  733 root       20   0 28388 18660  3404 S  1.3  0.1  3h27:00 /usr/bin/perl /usr/sbin/x2gocleansessions
31720 [redacted] 21   1 8994M 3259M  409M S  1.3 18.3  1:07.36 /opt/ISEfull/Vivado/2019.2/bin/unwrapped/lnx64.o/vivado
32594 [redacted] 21   1 8994M 3259M  409M S  0.7 18.3 48:07.92 /opt/ISEfull/Vivado/2019.2/bin/unwrapped/lnx64.o/vivado
32017 [redacted] 21   1 8994M 3259M  409M S  0.7 18.3 21:29.25 /opt/ISEfull/Vivado/2019.2/bin/unwrapped/lnx64.o/vivado
  365 [redacted] 21   1 8994M 3259M  409M S  0.7 18.3 34:25.92 /opt/ISEfull/Vivado/2019.2/bin/unwrapped/lnx64.o/vivado
32290 [redacted] 21   1 8994M 3259M  409M S  0.7 18.3  9:11.09 /opt/ISEfull/Vivado/2019.2/bin/unwrapped/lnx64.o/vivado
 5808 [redacted] 21   1 14.7G  395M  113M S  0.7  2.2  1h24:05 /usr/share/code/code --type=renderer --disable-color-corr
22417 [redacted] 21   1 8994M 3259M  409M S  0.7 18.3 17:06.49 /opt/ISEfull/Vivado/2019.2/bin/unwrapped/lnx64.o/vivado
 1097 root       20   0  392M 24132  9812 S  0.7  0.1  8:38.93 /usr/bin/python3 /usr/bin/fail2ban-server -xf start
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Terminating Tasks

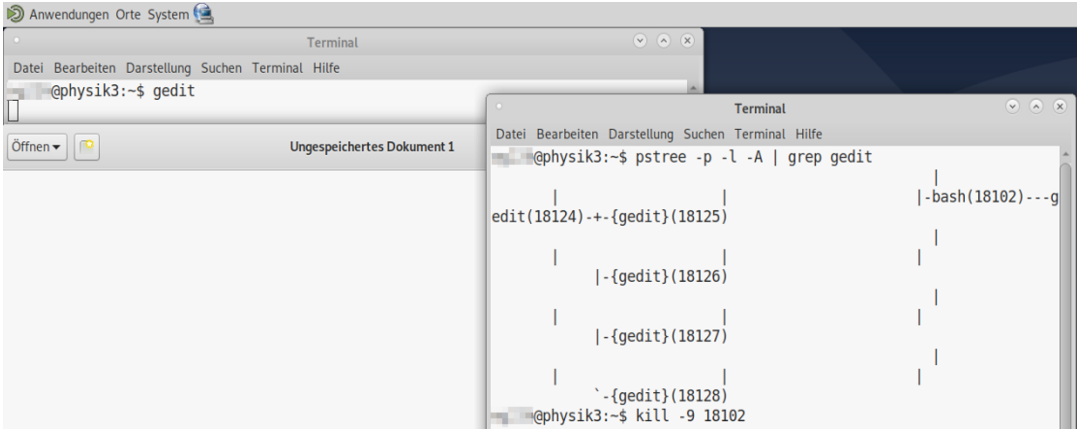
- There are several ways to terminate / kill a program.
- When it runs, press Ctrl + C in the command line. This will send an interrupt to the program.
- If the program is unresponsive and refuses to kill itself, you can try sending SIGTERM.
  - Every process has a process ID (PID). You can get a list with `ps -aux`.
  - Use `kill PID` (e. g. `kill 21311`) to send the SIGTERM signal to the process.
- If the process still does not want to terminate, you can send the unmaskable interrupt SIGKILL.
  - Find the PID.
  - Execute `kill -9 PID`, e. g. `kill -9 21311`.
- Also useful: Kill all tasks with a certain name: `killall firefox`.

Exercise with Tasks

Process Termination in the Hierarchy

- Log onto `physik1.kip.uni-heidelberg.de` via SSH or via Remote Desktop.
- Launch two terminal sessions (either two SSH connections or two Terminal windows).
- Open `gedit` on the first terminal.
- Use the second terminal to find out the PID of the first terminal session.
- Kill the first terminal session via the second session.
- Observe, how `gedit` closes as well.

Exercise with Tasks (2)


Anwendungen Orte System
Terminal
Datei Bearbeiten Darstellung Suchen Terminal Hilfe
Öffnen
Ungespeichertes Dokument 1

```
[redacted]@physik3:~$ gedit
```

```
[redacted]@physik3:~$ pstree -p -l -A | grep gedit
                                                        |
      |                |                              |-bash(18102)---g
edit(18124)-+-{gedit}(18125)
                                                        |
      |                |                              |
           |-{gedit}(18126)
                                                        |
      |                |                              |
           |-{gedit}(18127)
                                                        |
      |                |                              |
           `-{gedit}(18128)
[redacted]@physik3:~$ kill -9 18102
```

Scripting

Redirecting Outputs

- There are three different default streams:
  - `stdout` – “Standard Out”: Default channel for command output/results.
  - `stderr` – “Standard Error”: Default channel for reporting errors/failures.
  - `stdin` – “Standard In”: Default channel for the command to receive input (e. g. over the keyboard of the user).
- Streams `stdout` and `stderr` are often merged automatically and shown together on your terminal!
- You can redirect the streams to files or other commands!
  - Operator `>` Write result of `stdout` to a file.
    - `echo "Abc123" > file.txt` : Overwrites the file and writes the result of the command into the file.
    - `echo "Abc123" > /dev/null` : Redirects the output to the “black hole” to discard results.
    - `echo "Abc123" 2> /dev/null 1> file.txt` : Redirects `stderr` to `/dev/null` and `stdout` to `file.txt`.

Redirecting Outputs (2)

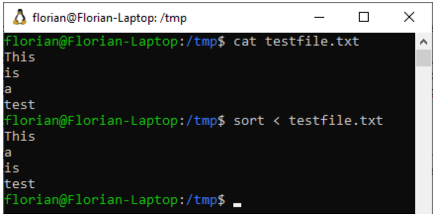
- Operator `>>` Appends the result of `stdout` to a file.
  - `echo "Abc123" >> file.txt` : Adds a new line to the file with the result.
  - `echo "Abc123" 1>> file.txt 2>err.txt` : Appends result of `stdout` to `file.txt` and overwrites `err.txt` with the output of `stderr`.
- Operator `<` Takes a file and redirects it to `stdin`.
  - `cat < file.txt` : `cat` will read from `stdin` if launched without arguments. `stdin` is connected to the contents of `file.txt`, therefore `cat` will print all its lines.
- Operator `<<` is used for *in-command text* (so called *here-doc*). Needs a **termination string**. Will read the following lines until termination string and connects this to `stdin`.

```
cat << "the termination string"
hello
another line
the termination string
```

Redirecting Outputs (3)

Try Input Redirection

- Create a file with some words in it. Put one word per line. You can do this with e. g. `nano`.
- Print the file to the command line with `cat`.
- Sort the file line-wise with the `sort` command, like: `sort < myfile.txt`.



```
florian@Florian-Laptop:/tmp$ cat testfile.txt
This
is
a
test
florian@Florian-Laptop:/tmp$ sort < testfile.txt
This
a
is
test
florian@Florian-Laptop:/tmp$ _
```

Pipes

- Multiple commands can easily be concatenated with **pipes**. ➥ Inter-process communication.
- The pipe operator is `|`.
- A pipe will connect `stdout` of the first command to `stdin` of the next command.
- Example: `cat file.txt | grep "e" | sort | tail` gives you the last 10 lines that contain the small letter “e” in a sorted way.
- Note: `cat | grep` combo generally useless, since `grep` can read files.

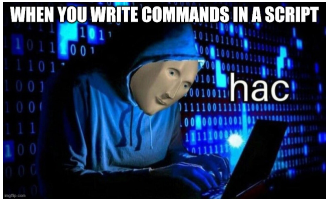
Pipes Example

Try Pipes

- There is a dictionary file located at `/usr/share/dict/words`. It contains a lot of words.
- We want only want to have the lines that match this:
  - Line starts with a capital *N* (help: use `egrep` with RegEx: `^N`)
  - Line does not end with *'s* (help: inverse match with RegEx: `\'s$`.
- We want to reverse-sort the result (help: study parameters of `sort`).
- We want to count the characters of the resulting output (help: study parameters of `wc`).
- The output should be a number!

Batch Execution

- You don't always need to enter your command interactively.
- You can write a *script* which contains all your commands.
- A script is just a series of commands which are interpreted by the shell.
- You can use control flow commands (if-then-else) and iteration commands (loops) to write real programs!
- You can also define functions to make your script modular.



Content of a script:

- First line: Some cryptic thing called *Shebang*.
  - Specifies the shell that should be used to interpret the following script.
  - Necessary to cope with the differences of the various shells.
  - To use `bash`, just write `#!/bin/bash`.
- Following lines: Contain the commands to be executed.

Different Shells

- Bourne shell: `sh`
- C shell: `csh`. Uses C-like syntax.
- Korn shell: `ksh`. Combines features of `sh` and `csh`.
- Bourne again shell: `bash`. Default shell for GNU/Linux. Extended version of `sh`.
- Restricted Bourne again shell: `rbash`. Shell where stream redirections and changing the directory are prohibited. Useful for restricted jumphosts. Not useful for scripts.

➥ Most of the time you will end up with `bash`.

Variables

- The shell supports variables and knows exactly one type: *String*.
- Variables do not have to be declared – they can just be defined.
- Define a variable with `myvar="some text"`.
  - No whitespace is allowed between `myvar` and `=`.
  - It is best to always use quotation marks.
- Define environment variables that are also visible in child processes: `export variable="value"`.
- Get a list of all set variables: `set`.
- Get a list of all available environment variables: `env`.

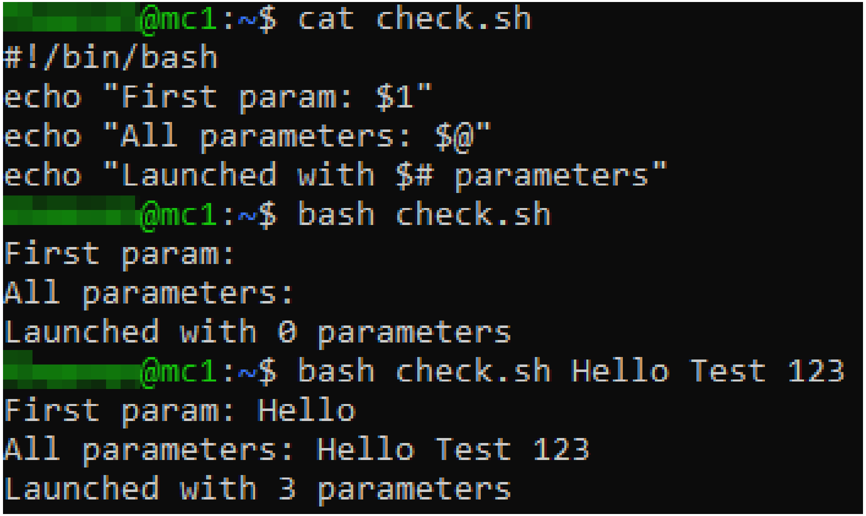
```
#!/bin/bash
set | grep testvar1
testvar1=1
set | grep testvar1
```

Variables (2)

- Read a variable with `$myvar`.
- Alternatively if whitespace is not possible after variable name: `${myvar}`.
- Special variables:
  - `$PATH` includes a list of directories that are searched for valid commands.
  - `$PS1` is used to process strings to be shown on the shell before they are printed.
  - `$SHELL` gives you the path of the shell that is currently used.
  - `$LANG` contains the system language settings.
  - `$0` contains the program name, if executed from a script.
  - `$1`, `$2` ... contain the command-line arguments of the script.
  - `$@` is an array of all command-line parameters.
  - `$?` contains the return code of the last command.
  - `$$` contains the PID of this process.

```
#!/bin/bash
echo "First param: $1"
echo "All parameters: $@"
echo "Launched with $# parameters"
```

Variables (3)



```
[redacted]@mc1:~$ cat check.sh
#!/bin/bash
echo "First param: $1"
echo "All parameters: $@"
echo "Launched with $# parameters"
[redacted]@mc1:~$ bash check.sh
First param:
All parameters:
Launched with 0 parameters
[redacted]@mc1:~$ bash check.sh Hello Test 123
First param: Hello
All parameters: Hello Test 123
Launched with 3 parameters
```

Execute Commands

- You often need the result of a command as a variable.
- Way 1: Use backticks:
  - `` myvar=`cat /usr/share/dict/words | wc -l` ``
  - Looks simple (☺), but you cannot "cascade" this.
- Way 2: Use `$(cmd)`:
  - `myvar=$(cat /usr/share/dict/words | wc -l)`.
  - Looks more complicated but is cascadeable.
  - `abc=$(cat $(ls *.sh))`.

If Statements

- Syntax: `if CONDITION ; then COMMANDS else COMMANDS fi`
- Conditions are often formed with the `test` command.
- The `test` command has a useful shortcut: `[ some stuff ]`
- Examples:
  - Check if file exists: `if [ -f myfile.txt ] ; then` ...
  - Check if directory exists: `if [ -d myDir ] ; then` ...
  - Check if file does not exist: `if ! [ -f myfile.txt ] ; then` ...
  - Check if variable contains stuff: `if ! [ -z $var ] ; then` ...
  - Check if variable equals 5: `if [ $var -eq 5 ] ; then` ...

```bash
#!/bin/bash
if [ $1 -eq 1 ] ; then
  echo "Success"
else
  echo "Fail"
fi
```

Loops

- Often useful: Loop over set of files. Hint: Avoid iterating over the result of `ls`.
- Use a *Glob expression* instead.
- Syntax: `for f in *.txt ; do ; COMMANDS ; done`.
- Iterating over array: `for e in $@ ; do ; COMMANDS ; done`.
- C-Style loops also possible: `for ((i=0; i <= 10; ++i)) ; do ; COMMANDS ; done`.

```
#!/bin/bash
for e in $@ ; do
  echo $e
done

for ((i=0; i < 10; ++i)) ; do
  echo $i
done
```

Calculating

- You can also calculate in `bash`. Often useful to compute numbers, increment counters, etc...
- Way 1: Use `expr` to evaluate expressions.
  - `expr 10 + 12` yields 22.
  - `expr 10 * 12` yields a syntax error in `bash`.
  - `expr 10 \* 12` yields 120.
- Way 2 (preferred by me): Use `bash`-mechanics with `$((...))`.
  - `echo $((10+12))` yields 22.
  - `echo $((10*12))` yields 120.
  - `echo $((10**12))` yields 10^{12}.

```
#!/bin/bash
for ((i=0; i <= 10; ++i)) ; do
  echo $((2**$i))
done
```

Functions

- Define functions with `function myFunc { ... }`.
- Functions cannot have parameters ☹. Only implicit through `$1, $2, ...` which are exclusive for a function.
- Functions also have no return value. They return whatever you print to `stdout` during the function ☹.

```bash
#!/bin/bash
function replace {
  user="$1"
  pass="$2"
  sed -e "s|++USERNAME++|$user|g;s|++PASSWORD++|$pass|g" templ.txt
}

cat > templ.txt << EOF
Hello,
your username is ++USERNAME++ and your password is ++PASSWORD++.
EOF

replace Hannes abc123
replace Julia 213abc
```

Exercise

Backup-Tool

- We want to write a tool that backups all files in a directory to a ZIP-archive.
- The program receives one parameter: The directory to backup.
- If the directory does not exist, the program should say “Directory does not exist” and then terminate.
- Otherwise, an archive name is generated in a function called `getArchiveName`.
  - The function does not take any parameters.
  - It will return a string that looks like this: `Backup-2022-05-22.zip`.
  - The number should be replaced by the actual current date.
  - If a file already exists that has this name, an index should be appended and counted correctly: `Backup-2022-05-22_1.zip`. If that file also exist, create `Backup-2022-05-22_2.zip` and so on...
- The given directory will be zipped with the `zip` command which creates an archive with the calculated name.
- Test your solution extensively.

Hints

- You can use the `date` command to get a formatted string of the current date: Example: `date +"%d:%m"`. Check out the `man` page!
- You may want to use `while` loops. Syntax: `while CONDITION ; do ; COMMANDS ; done`. Conditions may be used identical to the if-statement.
- To archive the files, use `zip -9 -r filename.zip dirToPack`.
  - `-9` gives maximal compression.
  - `-r` makes the program process entire directories.
- Try to use variables for everything. You don't need fancy tricks if you manage your variables correctly.
- You can concatenate strings by just writing them after one another:

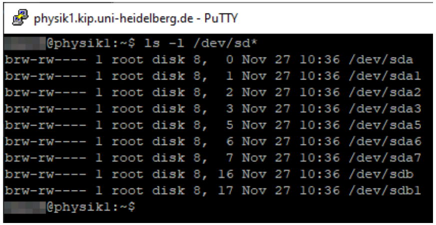
```
#!/bin/bash
str1="Hello"
str2="World"
combo="${str1} ${str2}"
echo $combo
```

➥ The entire script requires round about 20 lines and no "hacks".

Extra Information

Drives and Partitions

- Storage media are **block devices**.
- Block devices are special files in the directory `/dev/`.
- Some examples:
  - IDE hard disks are found as `/dev/hdXY`.
  - SATA/SCSI hard disks are `/dev/sdXY`.
  - CD-ROM drives are `/dev/cdromY`.
  - Floppy drives are `/dev/fdY`.
- Drives are labeled with letters:
  - First IDE drive is `/dev/hdaY`.
  - Second SCSI drive is `/dev/sdbY`.
- Partitions are labeled with numbers:
  - Second partition on first IDE drive is `/dev/hda2`.
  - Sixth partition on third SCSI drive is `/dev/sdc6`.


physik1.kip.uni-heidelberg.de - PuTTY

```
[redacted]@physik1:~$ ls -l /dev/sd*
brw-rw---- 1 root disk 8,  0 Nov 27 10:36 /dev/sda
brw-rw---- 1 root disk 8,  1 Nov 27 10:36 /dev/sda1
brw-rw---- 1 root disk 8,  2 Nov 27 10:36 /dev/sda2
brw-rw---- 1 root disk 8,  3 Nov 27 10:36 /dev/sda3
brw-rw---- 1 root disk 8,  5 Nov 27 10:36 /dev/sda5
brw-rw---- 1 root disk 8,  6 Nov 27 10:36 /dev/sda6
brw-rw---- 1 root disk 8,  7 Nov 27 10:36 /dev/sda7
brw-rw---- 1 root disk 8, 16 Nov 27 10:36 /dev/sdb
brw-rw---- 1 root disk 8, 17 Nov 27 10:36 /dev/sdb1
[redacted]@physik1:~$
```

Mounting Devices

- When accessing files on other locations than the system disk, these other locations need to be *mounted*.
- To mount a file system, a *mount point* is needed: A directory under which the mounted file system should be made available.
- For hard disks, a file system driver is required: If the hard disk is formatted with NTFS, you need an NTFS driver.
- Network file systems are e. g. NFS or SMB/CIFS – you need additional drivers for this as well.
- Mounting files usually requires admin privileges (➥ You cannot do this on university systems).
- To mount a hard drive:
  - Find the correct hard drive and partition with `fdisk -l`.
  - Create a mountpoint, e. g. with `mkdir -p /media/data-hdd`.
  - Mount the device (in this case `/dev/sdb2`): `mount /dev/sdb2 /media/data-hdd`.
  - Check out the data: `ls -al /media/data-hdd`.
  - Unmount with `umount /media/data-hdd`.

Mounting in User Space

For experts:

- There are libraries to mount file systems in user space without admin privileges.
- You can mount a remote directory over SSH via `sshfs`.
- Target system needs to have `sshfs` installed, remote system only needs SSH server.
- Mount directory with `sshfs user@server.example.org:/remote/directory /local/mountpoint`.
- Unmount with `fusermount -u /local/mountpoint`.
- My opinion: Only use this as quick hack if other means are unavailable. You have a lot of problems, if the SSH server is unresponsive or has high latency.