Technische Hochschule Rosenheim

Modul - IT Systems (IT)

Bachelor Programme AAI

05 - Lecture: File Systems

Prof. Dr. Marcel Tilly

Faculty of Computer Science, Cloud Computing

Agenda

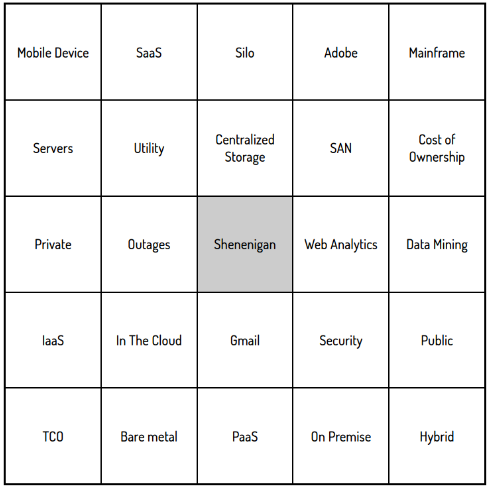
- File Systems
- Shell commands for file manipulation
- First set of shell commands

Fun part

Bullshit Bingo!

Buzzword bingo, also called bullshit bingo in later usage, is a humorous variation of the bingo game that satirizes the often content-less use of numerous buzzwords in lectures, presentations, or meetings.

https://gitlabio.z6.web.core.windows.net/tools/bsb/index.html?shell



Mobile Device	SaaS	Silo	Adobe	Mainframe
Servers	Utility	Centralized Storage	SAN	Cost of Ownership
Private	Outages	Shenenigan	Web Analytics	Data Mining
IaaS	In The Cloud	Gmail	Security	Public
TCO	Bare metal	PaaS	On Premise	Hybrid

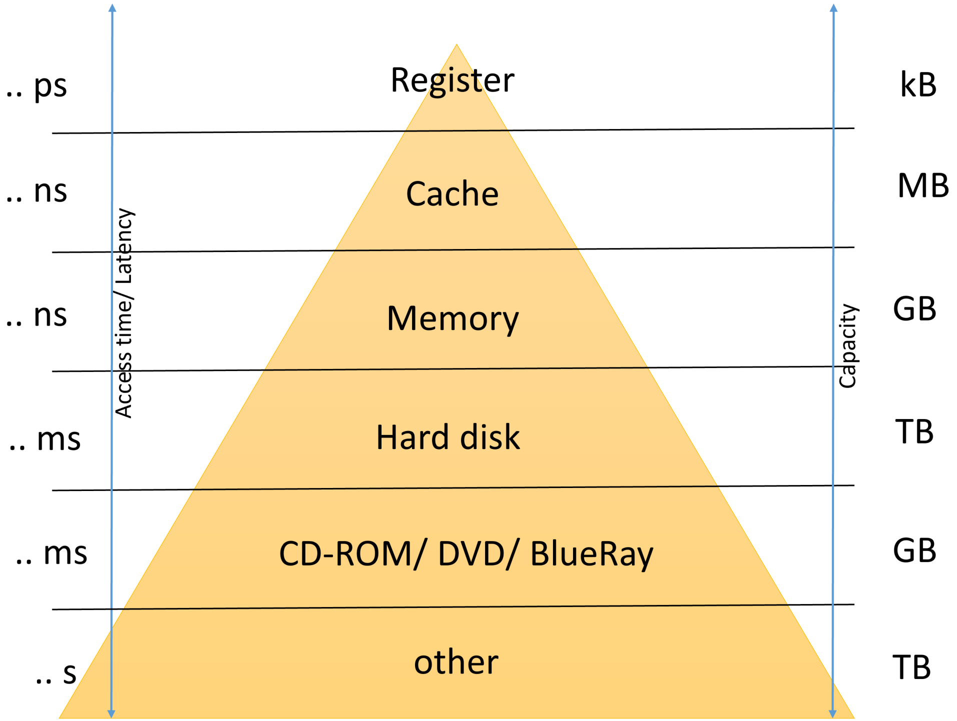
Learning Objectives

Students will be able to …

- … create, rename, copy and delete files and directories using the shell
- … find data within files
- … change data within files

Storage hierarchy

Scale of capacities and access times



File System

Tasks of file systems:

- Permanent storage of data in the form of named objects (*files*).
- Areas on the storage medium are provided with a so-called file system
- File system provides the management structure for objects

Challenges:

- Slowness of media, as mostly mechanical operations are required
- Extent of the information
- Fault tolerance

heterogeneity:

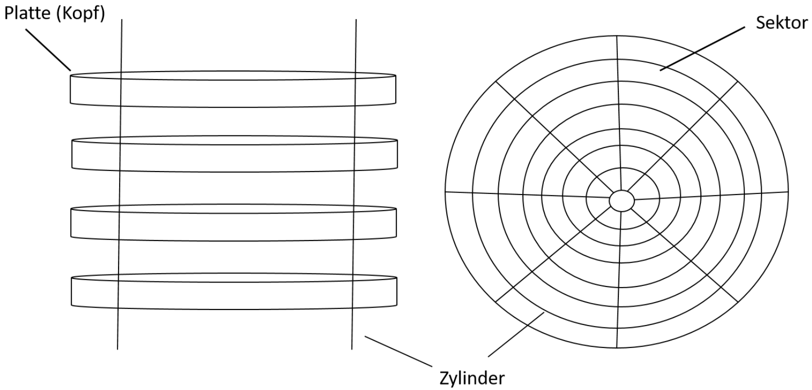
- magnetic mass storage media
- optical
- electrical

Structure of a hard disk

- Stack of rotating magnetic disks, constant rotation speed (CAV - Constant Angular Velocity)
- Rotation speed approx. 5400 - 15000 min
- 2-16 disks
- concentric tracks, approx. 10.000 per surface
- tracks on top of each other form a so called cylinder
- smallest addressable unit: physical block ("sector"; 512 bytes), e.g. 150-300 sectors per track
- 1 read/write head per disk surface, radial movement of all heads together

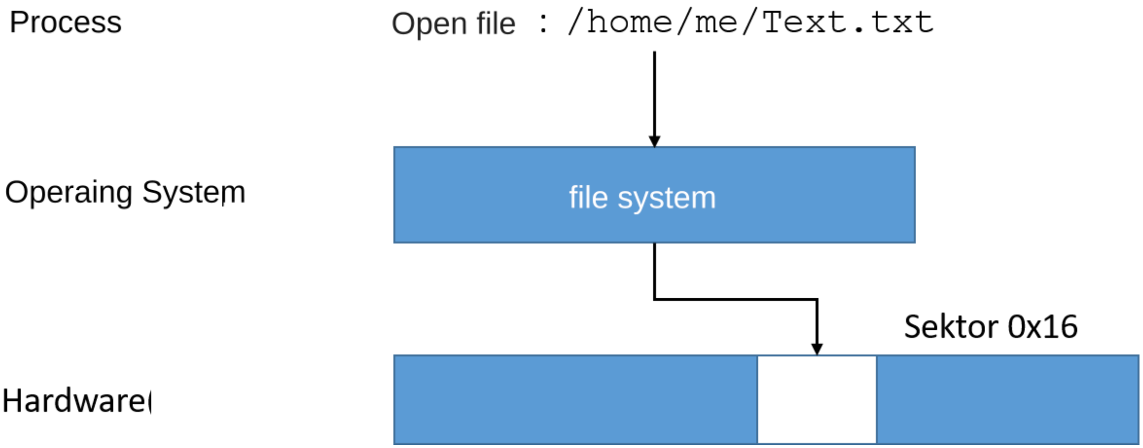
Structure of a hard disk

- Hard Disc (=HD)
- **historical**: Addressing of a sector via *{cylinder, head, sector}* triples (*cylinder, head, sector - CHS*).
- **today**: *Logical Block Addressing (LBA)*, simple numbering of all blocks
- physical layout hidden from user: mapping of logical block numbers to physical block numbers (LBN -> PBN) by hard disk electronics



Request for file

User's request for contents of a file must be answered by the operating system



Access procedure

1. send access command of driver via memory bus system (e.g. PCI)
2. forwarding to peripheral bus (e.g. SCSI, FireWire, USB)
3. conversion *LBN -> PBN*
4. positioning of the read/write head over the correct track: *seek time*.
5. possibly switching to corresponding head
6. wait until beginning of desired sector under head
7. read the sector into the buffer memory of the hard disk: transfer time
8. transfer of data via bus system (SCSI, PCI) into the main memory of the computer access time dominated by mechanical part!

File systems

A **file system** is an abstraction of the operating system for device-independent management of files

- uniform view of different types of secondary storage, e.g. hard disk, floppy, CD-ROM, DVD or tape drives
- user does not have to worry about physical data formats on different types of secondary storage devices
- each file is represented as a set of fixed size blocks (where a block can correspond to one or more sectors)

- File systems consist of
  - files for storing executable programs and data (source code, documents, images, ...) and
  - directories for the hierarchical structuring of files
  - links (*Links*) and special files (*Special Files*)

File systems

Partition

- a partition is a set of directories and their files on a part of a data carrier
  - is used to separate parts of a hard disk or CD (the cylinders of a hard disk can be divided into several independent partitions, e.g. for operating system and user files)
  - each partition gets its own file system
  - some file systems allow mounting of additional partitions (e.g. from other devices)
- Requirements for a file system:
  - persistent storage of data in files
  - fast access to files
  - simultaneous access by several processes
  - implementation of protection rights

File systems

Files

- Typical attributes of files:
  - Symbolic name, readable and interpretable by the user (usually consisting of name + extension, e.g. uebung.pdf)
  - Type, e.g. character-oriented file (for sequential access), record-oriented file (for random access), executable binary file, special file (for I/O device), …
  - Size (in bytes or file blocks)
  - Identification of the owner (user ID)
  - Access rights
  - Timestamp, e.g. of creation and last modification (important for incremental backup and development tools)
  - Location information, i.e. place of physical storage

File systems

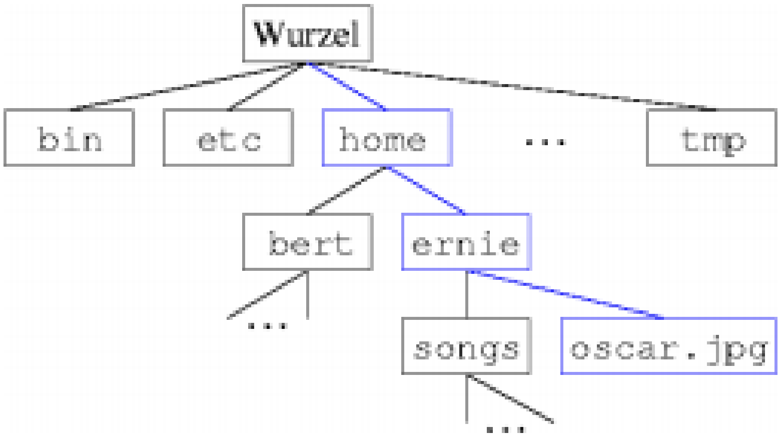
Operations

- typical operations on files:
  - **Create** (create): create an empty file in directory and set attributes.
  - **Delete** (delete): delete a file in directory and free space on disk
  - **Open**: access by process and load attributes
  - **Close**: release by process
  - **Read** (read): read at current position of file pointer
  - **Write** (write): ditto, for writing
  - **Append** (append): write new data at the end of a file
  - **Position** (seek): set file pointer to specific location

File systems

Directory (folder)

- A directory contains the names of files and, if applicable, the names of subdirectories.
  - also referred to as folder
  - user can arrange files in logical groups
  - Build a hierarchical directory structure taking into account multiple users:



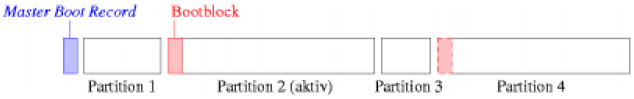
Path names

- Access to directory i.a. via path names
  - Names of all directories on the path from root to file, separated by special separator character
  - Unix/Linux: `/home/ernie/oscar.jpg`
  - Windows: `C:\home/ernie/oscar.jpg`
  - Parent directory: `..`
  - Self reference: `.`
- typical operations on directories:
  - **create** (mkdir)
  - **delete** (rmdir), only possible on empty directories
  - **open** (opendir) and **close** (closedir)
  - **read** a directory entry (readdir)

Implementation of file systems

layout of a file system

- one or more partitions
- sector 1 (in cylinder 0, head 0) is called MBR ("Master Boot Record") and contains
  - a code which is executed by the BIOS when starting the computer
  - a partition table with start and end address of each partition
  - marking of an active system partition from which booting is performed
- first block in each partition is a boot block which is executed when booting from this partition
- structure of a partition depends on the respective file system



Implementation of file systems

Contiguous allocation:

- each file is stored contiguously in a continuous sequence of blocks
- to access a file, the directory must store the address of the starting block and the number of occupied blocks
- easy to implement
- high reading speed
- subsequent change of file size problematic
- leads to external fragmentation

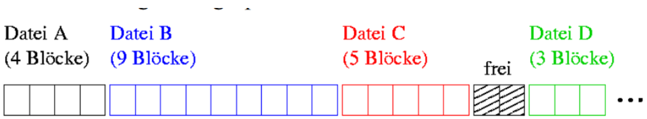
Why?

Implementation of file systems

Contiguous allocation:

- each file is stored contiguously in a continuous sequence of blocks
- to access a file, the directory must store the address of the starting block and the number of occupied blocks
- easy to implement
- high reading speed
- subsequent change of file size problematic
- leads to external fragmentation

Why?

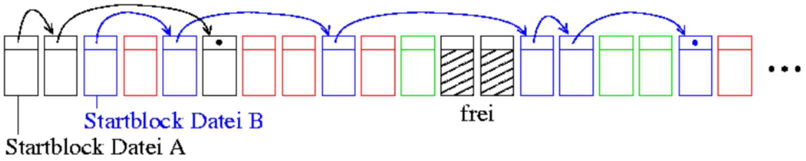


Implementation of file systems

Technische
Hochschule
Rosenheim

Chained block list:

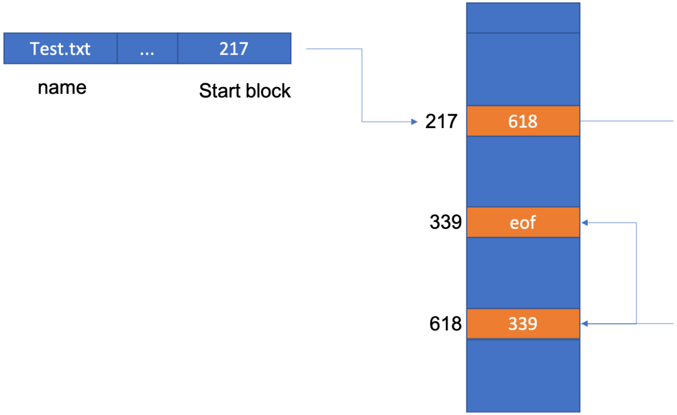
- in the beginning of each block a pointer to the next block (or an end symbol) is stored
- in the rest of each block the data is stored
- to access a file, only the address of the start block must be stored in the directory
- no external fragmentation
- very slow random access: to access the kth block all k- 1 preceding blocks must be read first



Implementation of file systems

Central index structure:

- a central table (*File Allocation Table*, FAT) manages all blocks of the file system, where the block number serves as index



- for each block of a file the index of the following block (or an end symbol) is stored in the FAT
- additionally free blocks can be marked in the FAT by special symbols
- to access a file only the index of the start block must be stored in the directory

FAT Limits

- FAT is stored on disks and is loaded from there (completely or partially) into main memory
- FAT becomes very large for today's hard disks
- if FAT is lost or destroyed, the associated file system is no longer usable
- with FAT16 is addressed over a 16-Bit address

Question:

- How many files can be managed with FAT16?

FAT Limits

- FAT is stored on disks and is loaded from there (completely or partially) into main memory
- FAT becomes very large for today's hard disks
- if FAT is lost or destroyed, the associated file system is no longer usable
- with FAT16 is addressed over a 16-Bit address

Question:

- How many files can be managed with FAT16?
  - 4.096 (FAT12)
  - 65,536 (FAT16)
  - 4,294,967,296 (FAT32)

Implementation of file systems

Distributed index structure:

- for each file there is a separate index list with the numbers of all used blocks in the corresponding order
- index list of a file is stored in separate index block in the file system; for long files several index blocks are needed
- attributes of the file can also be stored in the index block
- to access a file, only the number of the corresponding index block must be stored in the directory
- index block must be in main memory only when file is open

Realization of directories

Each **directory** occupies like a file one or more blocks of the file system; possibilities of realization:

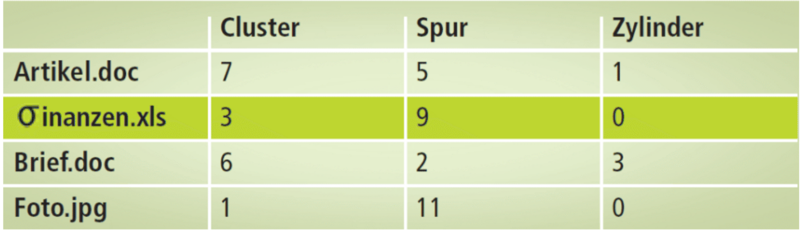
1. **list with entries of fixed length**: for each file/subdirectory there is one entry of fixed length with name, attributes and location information.
2. **concatenated list with entries of variable length**: for each file/subdirectory there is one entry of variable length with name, attributes, location information and a pointer to the next entry
3. **concatenated list, but using index blocks**.
    - for each file/subdirectory there is one entry with name and number of the corresponding index block
    - index block contains attributes and location information
    - when accessing a file whose name is given, the list is usually searched sequentially

Free blocks

Possibilities of management of free blocks:

1. **Free list**: In a concatenated list of blocks the addresses of the free blocks of the file system are stored.
2. **Bitmap**: In a bitmap, it is recorded for each block whether it is free (bit = 1) or occupied (bit = 0)
3. **Table**: With central index structure free blocks can be marked in FAT

„σ“ means spot is available



	Cluster	Spur	Zylinder
Artikel.doc	7	5	1
σinanzen.xls	3	9	0
Brief.doc	6	2	3
Foto.jpg	1	11	0

Task

Which file systems do you know?

Go to https://pingo.coactum.de/693750 ... **693750**

2min. time!

What file systems are there?

BS	file system
MS-DOS	FAT12, FAT16
Windows 9x	VFAT
Windows NT. . . 10	NTFS
MacOS	HFS, HFS+, APFS
Linux	ext2, ext3, ext4fs, btrfs, XFS, . . .

> Good OSes also read the file systems of the "competition", as long as they are open. In addition, there are cross-BS file systems, e.g. IS09660 (file system of the CD-ROM) or CIFS.

EXT2 file system

- EXT2 is since 1992 the standard file system for Linux (today i.a. replaced by successor EXT3 with journaling)
- works on blocks of size 1 KByte, 2 KByte or 4 KByte (selectable by **mke2fs** as parameter when creating the file system)

File in EXT2:

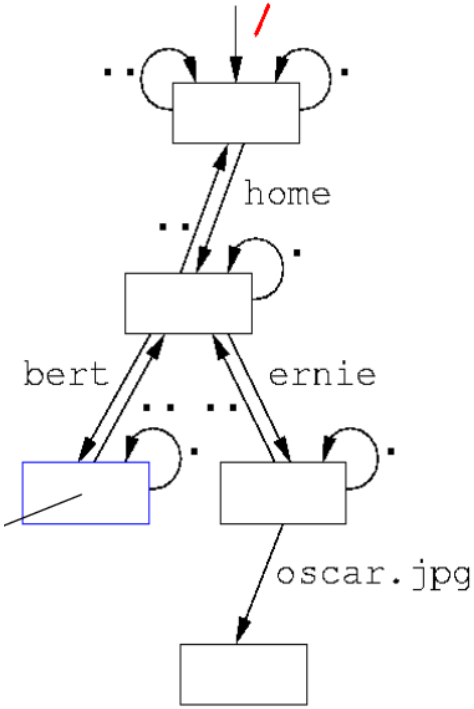
- unstructured byte sequence with arbitrary content
- access rights: readable (r), writable (w), executable (x), can be set separately for owner, group and all other users

Directory in EXT2:

- each process is assigned a current directory (cwd = current working directory)
- access rights: readable (r), writable (w), searchable (x), can be set separately for owner, group and all other users

File tree in EXT2

- root directory is "/"
- not files and directories, but the links between them are named
- each directory has a reference to itself (".")
- and to the parent directory (".")
- path names with separator "/" (slash)
- absolute path from root, e.g. */home/ernie/oscar.jpg*
- relative path from current directory, e.g. *../ernie/oscar.jpg*
- Files and directories are thus addressable by multiple pathnames



FAT32 file system

some differences to the Linux file system EXT2:

- no user identification for files and directories!
- Partitions are represented by drives, which are represented by letters, e.g.:
    - A: (floppy)
    - C: (disk)
    - D: (DVD)
- each Windows program is assigned a current drive and a current directory from file tree
- there are no inodes: all attributes of a file are stored in the directory
- there are no hard links
- the smallest addressable unit is called cluster and is a block with a power of two from 1 to 128 sectors (selectable at formatting)
- central index structure with a FAT (*File Allocation Table*), in which the concatenation of the clusters of all files is stored

NTFS file system

some differences to the FAT32 file system:

- support of multiple users and groups with extensive access rights
- a logical mass storage unit is called a *volume*.
- each *volume* consists of a linear numbered sequence of clusters of fixed size (512, 1024, 2048, ... , or 65536 bytes)
- Addressing of a cluster is done by 64-bit cluster numbers (very large partitions possible)
- File management in a volume via a central table (MFT=Master File Table), which contains an entry for each file
- Files can be stored automatically compressed
- Consistency check via journal file

Basic abstractions: File

file = *collection* of user data + attributes

Examples of typical attributes:

- Protection: who may perform what operation with file?
- Owner of the file
- Restrictions on allowed operations (read-only)
- Restrictions on visibility of file (hidden flag, .filename)
- file name
- Timestamp (last access, last modification, creation)
- Size of the file
- Position of the file position pointer

Information about file

```
$ stat index.html


  File: 'index.html
  Size: 431683 Blocks: 896 IO Block: 4096 regular file
Device: 2h/2d Inode: 6473924465011793 Links: 1
Access: (0666/-rw-rw-) Uid: ( 1000/ marcel) Gid: ( 1000/ marcel)
Access: 2019-11-09 09:55:34.000000000 +0100
Modify: 2019-11-09 09:55:10.000000000 +0100
Change: 2019-11-09 09:55:34.150298800 +0100
Birth: -
```

Types of files

Differentiation of file types

- by attributes (file names, ASCII/binary flag),
- by filenames
- by *Magic Word*.

A *Magic Word* is a characteristic byte sequence at the beginning of the file, by which its type can be identified.

Sequence **Meaning**	
JFIF	JPEG File Interchange Format
GIF89a	Graphics Interchange Format (V.89a)
#!/bin/bash	Shell script
ELF	Executable and Linkable Format

Examples: PDF, JPG

```
$hexdump -C 00-introduction.pdf
00000000 25 50 44 46 2d 31 2e 35 0a 25 e4 f0 ed f8 0a 31 |%PDF-1.5.%.....1|
00000010 31 20 30 20 6f 62 6a 0a 3c 3c 2f 46 69 6c 74 65 |1 0 obj.<</Filte|
00000020 72 2f 46 6c 61 74 65 44 65 63 6f 64 65 2f 4c 65 |r/FlateDecode/Le|
00000030 6e 67 74 68 20 35 36 35 3e 3e 0a 73 74 72 65 61 |ngth 565>>.strea|
00000040 6d 0a 78 da c5 55 cb 6e 53 41 0c dd f3 15 fe 81 |m.x..U.nSA......|
```

```
$hexdump -C brainstorming.jpg
00000000 ff d8 ff e0 00 10 4a 46 49 46 00 01 01 00 48 |......JFIF.....H|
00000010 00 48 00 00 ff db 00 43 00 0a 07 07 09 07 06 0a |.H.....C........|
00000020 09 08 09 0b 0b 0a 0c 0f 19 10 0f 0e 0e 0f 1e 16 |................|
00000030 17 12 19 24 20 26 25 23 20 23 22 28 2d 39 30 28 |...$ &%# #"(-90(|
00000040 2a 36 2b 22 23 32 44 32 36 3b 3d 40 40 26 30 |*6+"#2D26;=@@&0|
```

File naming conventions

Each file system has rules for the structure of a file name:

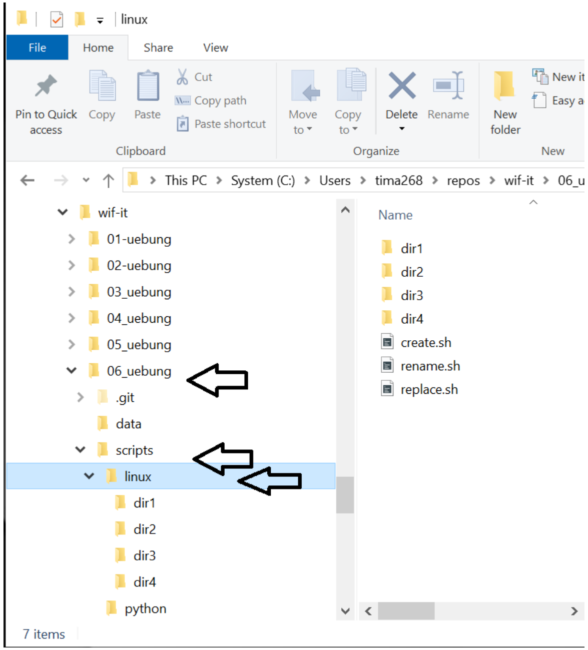
- FAT (File Allocation Table) - MS-DOS
  - "infamous" 8.3 convention
  - .COM, .EXE - executable files -.BAT - batch files (analogous to shell scripts)
- VFAT - as of Windows 95
  - up to 255 characters long
  - or 32000 characters (>=Windows 8)
  - Unicode encoded
  - **no** distinction between upper and lower case letters
  - several points possible, last one defines file type: e.g. *my.prog.c*, *help.now.pdf*
- Unix
  - **distinct** case sensitivity
  - name.ext actually uncommon, but still used (e.g. *.sh*)

File commands

activity	Linux	Windows
create file	touch	
list directory contents	ls	dir
Copy (copy)	cp	copy
Move	mv	move
rename	mv	ren
Delete (delete)	rm	del
File output	cat	type, echo
Change directory	cd	
create directory	mkdir	mkdir
print working directory	pwd	echo %cd%

pwd (print working directory)

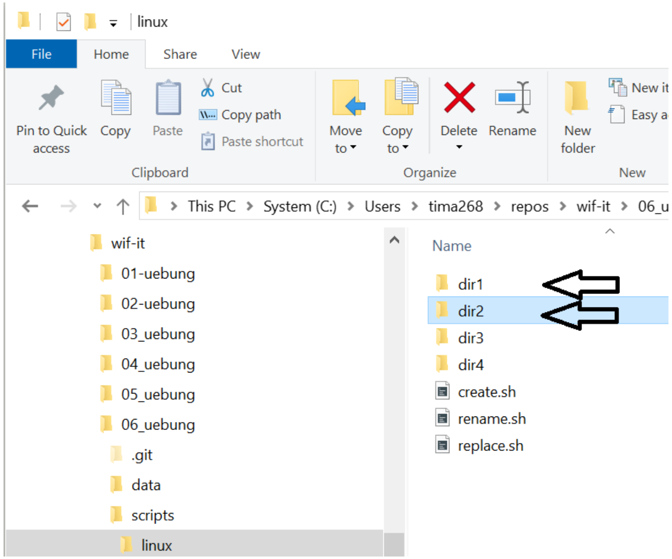
- shows current position in the file system
- more precisely: the path to the directory you are currently in.



```
$ pwd
/c/Users/aai-it/06_exercise/scripts/linux
$ cd ..
$ pwd
/c/Users/aai-it/06_exercise/scripts
```

cd (change directory)

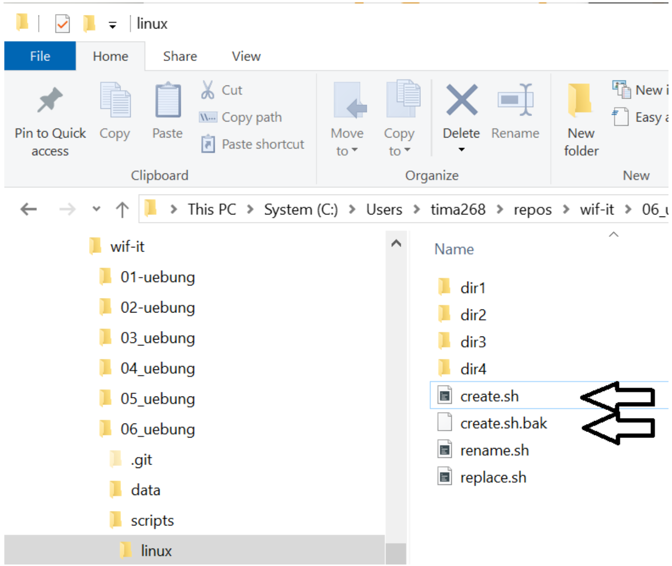
- change to another directory



```
$ pwd
/c/Users/aai-it/06_exercise/scripts/linux/dir2
$ cd ../dir1
$ pwd
/c/Users/aai-it/06_exercise/scripts/linux/dir1
```

cp (copy)

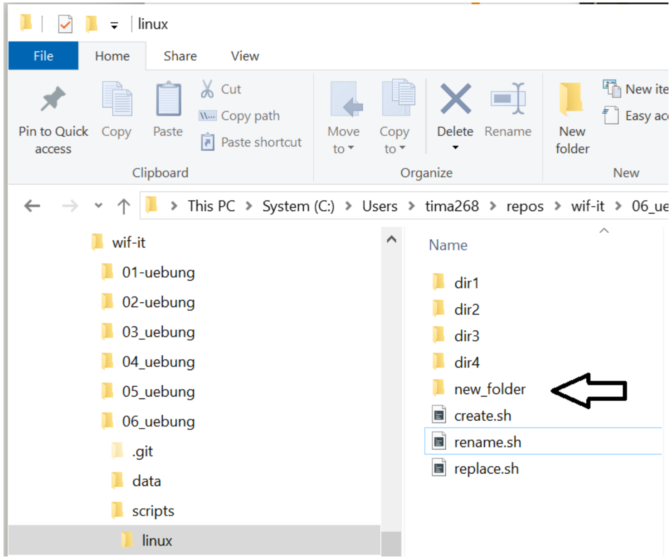
- copies a file or directory
- The copy can also be in another directory
    - with the same name
    - with a different name



```
$ cp create.sh create.sh.bak

$ cp create.sh ../dir1/create.sh.bak

$ cp ../linux/dir1/file1.txt ./file1.txt
```

mkdir (make directory)

- creates a subdirectory

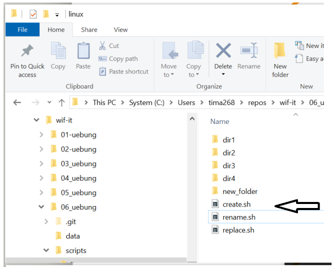


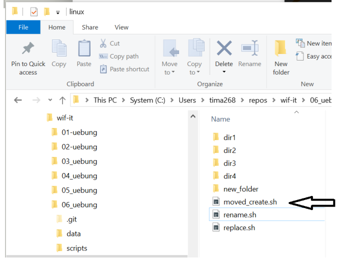
```
$ mkdir new_folder

$ mkdir ~/AppData/Local/testFolder
```

*(~ = User Home)

mv (move)

- Rename file / directory
- Files and directories can also be moved to other directories:
    - and keep their name
    - or get a new name





```
$ mv create.sh moved_create.sh

$ mv create.sh ../linux/moved_create.sh

$ mv dir1 dir5
```

Attention: Delete

- rm (remove)
  - delete file: `$ rm file`

  Caution: way is gone! There is no un-rm / undelete!
- rmdir (remove directory)
  - deletes an empty directory : `$ rmdir directory`
- `rm -rf` (remove recursively)
  - delete a directory with all its contents

> **BE CAREFUL!**

Display files

Under Windows:

- Windows Explorer
- On the command line (*cmd*) with the command **dir**.

Under Unix/Linux:

- Also there may be a graphical File Explorer.
- Otherwise with the shell command **ls**
    - **ls -l**: Displays more file information (l=long).
    - **ls -a**: Also shows "hidden" files (a=all).

On Mac:

- Finder
- Terminal and then Linux commands (see above)

Output of the command ls -l

```
$ ls -l
total 8
drwxrwxrwx 1 marcel th 4096 Sep 29 20:06 assets
-rwxrwxrwx 1 marcel th 5 Oct 29 16:12 exclude.txt
drwxrwxrwx 1 marcel th 4096 Sep 29 20:06 img
-rwxrwx 1 marcel th 6207 Oct 6 11:21 README.md
```

Meaning of the fields:

- Entry type: file, directory
- permissions: owner, group owner, all other users
- Number of hardlinks (file), number of subdirectories (directory)
- owner
- group owner
- Disk space usage
- Date and time of last modification
- name

Tip of the day!

Tab completion

Tab completion: Enter only the beginning of a command, then press the *Tab* key:

- If there is only one possible continuation, the word will be completed e.g.

```
$ mkd [Tab]
$ mkdir
```

- If not, then: twice in a row Tab returns a list of possible completions: e.g.

```
$ mk [Tab] [Tab]
    mk4ht.exe mkpasswd.exe
    mkfifo.exe mktemp.exe
```

Works also with filenames!

Access rights in Unix

- each file has 3 permissions: read, write, execute
- permissions are given for 3 categories of users: the owner (u), the group (g), all other users of the system (o) -> 3x3 bits which can be set or cleared
- execute right for directory: you are allowed to change into it
- Change by `chmod` command

Example:

```
$ chmod u+rwx g+r-wx o-rwx foo.sh
$ ls -l foo.sh
-rwxr----- 1 marcel TH 4 2018-10-28 10:26 foo.sh
```

What are the parameters to get `rw-r-x--x`?

For loop

- You can also use the for loop to display file lists
- The advantage is that the value can be processed

Use `./*` for all files (and directories in the current directory)

```
for FILE in ./*; do echo $FILE; done
```

Use `./**/*` for all files in subdirectories

```
for FILE in ./*; do echo $FILE; done
```

or the combination

```
for FILE in ./* ./**/*; do echo $FILE; done
```

Display the value without the `.sh` extension:

```
for FILE in ./* ; do echo ${FILE%.sh}; done
```

While loop

- The commands are executed as long as the condition is met.
- The condition is checked before the commands are executed.
- The condition is usually formulated with the `test` command, just like the if statement.

```
$ I=0
$ while [ $I -le 3 ]; do echo $I; I=$(( $I+1 )); done
```

```
while [ -n "$1" ]; do
      echo $1
      shift # with shift the parameters are shifted to
                # left ($2 becomes $1)
done
```

```
while [ ! -d "$FILE" ]; do

    ...
done
```

Pipe - Command

- One of the great strengths of Linux (Unix) is the possibility to combine commands arbitrarily to new commands.
- The execution of commands one after the other is called **pipe**.
- By sequential execution is meant that the output of the first (in time) program is used as input for the next program.
- The vertical bar (|) connects the commands to a pipe.

```
$ command1 | command2 | command3
```

Example:

```
ls -l /usr | less
```

The vertical bar (|) connects the commands "ls -l" and "less" to a pipe.

grep - command

- To search for certain patterns in files grep is often used. grep stands for **Global search for a Regular Expression and Print out matched lines**.
- The *grep* command searches for a pattern of characters in one or more strings or file(s).
- If the pattern contains whitespace, it must be quoted accordingly.
- Thus, the pattern is either a quoted string or a simple word.
- All other words after the pattern are then used by grep as input in which to search for the pattern.
- The output grep sends to standard output (usually the screen) and does not make any changes to the input file.

```
grep word file1 [file2] ... [files]
```

Example:

```
$ grep "while" moved_create.sh
```

sed - command

- **sed** (from stream editor) is a non-interactive text editor for use on the command line or in scripts.
- **sed** is one of the "bedrock" programs in the Unix / Linux world and is included in virtually every Linux installation (even minimal installations).

Examples:

Every occurrence of "Anton" is replaced by "Berta" (but also "Antonius" becomes "Bertaius"). If g (global) is omitted, only the first occurrence in a line is replaced. The "-i" flag indicates that the replacement is done inline. If omitted, the result will be printed to standard output.

```
$ sed -i s/Anton/Berta/g text file
```

sed - command

All words "Anton" will be replaced by "Berta" (not "Antonius"), but only in lines containing "name".

```
$ sed s/Anton/Berta/g text file
```

Replaces all "Anton" with "Berta" and prints only the affected lines. ("-n" Prevents automatic output of the result. Outputs only via the p command)

```
$ sed -n s/Anton/Berta/gp text file
```

Summary

Lessons Learned ...

- ... File systems, e.g. FAT, EXT, NTFS, ...
- ... File Operations
- ... File processing
    - grep
    - sed