1 – Unix basics

Bálint Aradi

Course: Scientific Programming / Wissenchaftliches Programmieren (Python)

Universität Bremen

Bremen Center for Computational Materials Science BCCMS

https://www.bccms.uni-bremen.de/cms/people/b-aradi/wissen-progr/python/2022

Outline

- General information
- Basic commands under Linux

Scientific programming

Scientific programming = Implementation of numerical algorithms in a given programming language in order to solve scientific problems.

- Make a model
- Choose the right numerical algorithm
- Plan the program structure
- Define interfaces
- Implement the algorithms (coding)
- Test your implementation
- Document your code
- Extend, reuse your code

- Correctness
- Numerical stability
- Proper discretisation (error estimation!)
- Flexibility
- Efficiency (speed, memory, scaling, etc.)

Some famous numerical disasters:

http://www-users.math.umn.edu/~arnold/disasters/

Content of the course

We will cover following topics:

- Introduction into Unix/Linux
- Basic data types, arrays
- Control structures
- Input / Output handling
- Functions, modules, packages, data hiding
- Basics of object oriented programming
- Graphical output, plotting
- Version control (git), cooperative development
- Unit testing
- Source code documentation
- Code profiling and code optimisation
- Parallel programming (eventually)

Literature: Slides + whatever you find about Python

Unix in general

Unix history in a nutshell

- Created 1969 (AT&T Bell Labs), originally written in assembler
- 1972: Rewrite from scratch in C (portability!)
- 70s, 80s: Unix gets popular in academics
- Most high performance computing (HPC) centers use Unix
- 1991: Linux Torwald starts to develop a Unix for i386-PC (Linux)
- 90s: Linux gets more and more popular on PCs.

Unix has many flavours

- **Linux** (open source under GPL license)
- BSD (FreeBSD, NetBSD, OpenBSD, open source under BSD license)
- AIX (IBM, commercial)
- :
- Mac OS X (based on a BSD-derivative)
- Windows? (not yet, but Windows 10 has Linux subsystem)

Advantages of Unix (for users)

Modular

- Operating system assembled from independent parts
- Often several alternatives for the same functionality
  - Unix shell: sh, ksh, csh, tcsh, **bash**, zsh, ...
  - Graphical environment: KDE, Gnome, LXDE, etc.

Communication and network oriented

Multi-tasking and multi-user capable by design

Contains efficient tools for many different tasks

- Tools can easily be combined with each other

Communicating with the operating system

Graphical user interface (GUI)

- Low entry barrier
- Functionality somewhat limited (like under Windows…)
- Not always clear, what happens under the hood

Command line interface (Shell)

- Needs more knowledge (higher entry barrier)
- Very complex tasks possible
- Tasks are often easier formulated
  - Typing one line instead of clicking 20 times…
- Closer to the operating system
  - Easier to understand what is going on (esp. in case of errors)

Unix shell

User commands are processed by the so called Shell

- Received
- Interpreted
- Executed
- Confirmed (e.g. error messages)

Various different popular shells available:

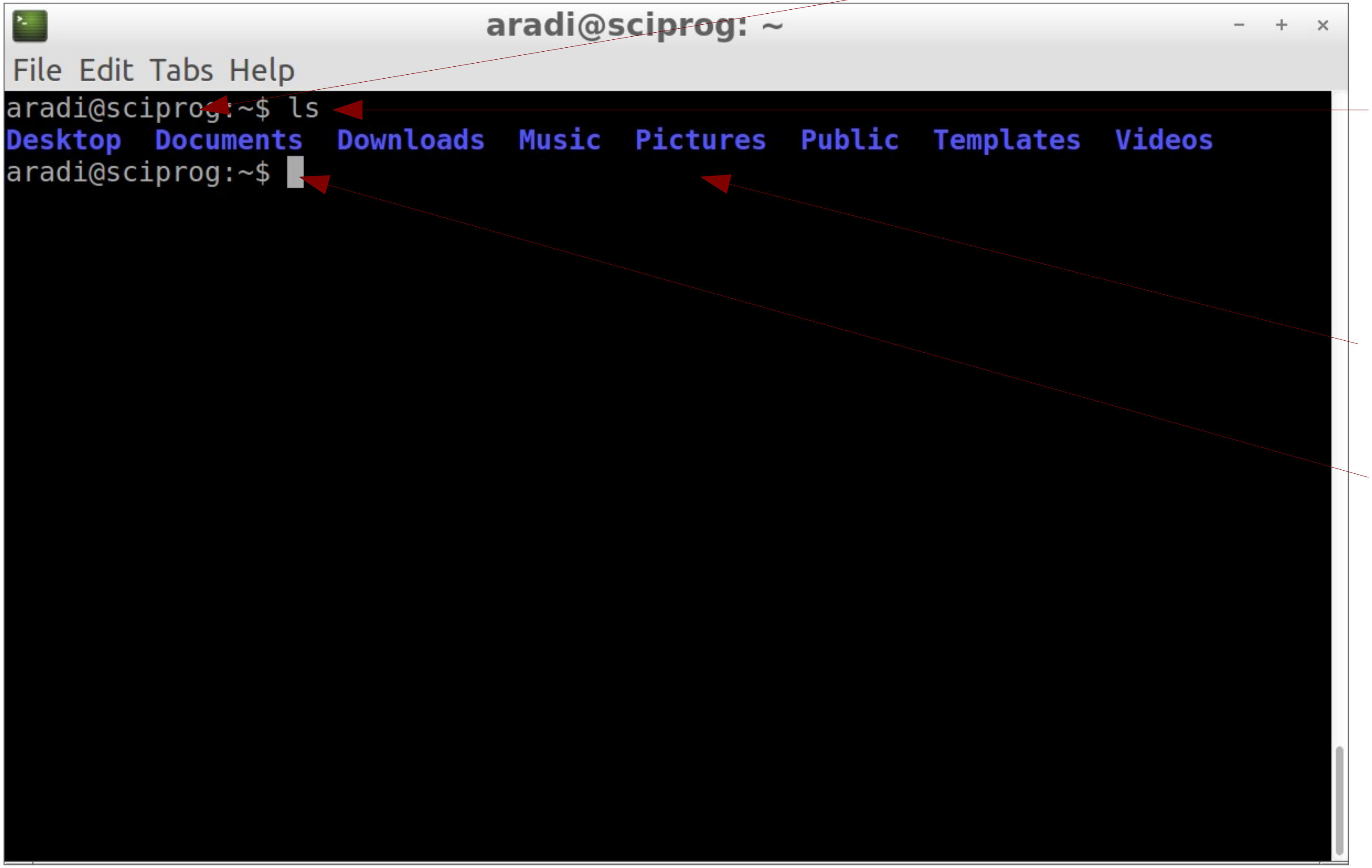
- SH and **BASH**, CSH and TCSH, ZSH
- User experience slightly different
- Shell command syntax (shell programming) slightly different
- However, most commands we will use are shell-independent programs

Let’s start!

Open a command line window (LXTerminal)

Type the command `ls`

Hit **Enter**



```
aradi@sciprog:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
aradi@sciprog:~$ 
```

Prompt

(shell waits for input)

Command

(submitted with Enter)

Response / Result

Prompt

(shell waits for input)

Typical shell commands

Working with files

- Manipulating files (copy, rename, remove)
- Edit file content
- Extract information from a file

Start other programs, applications

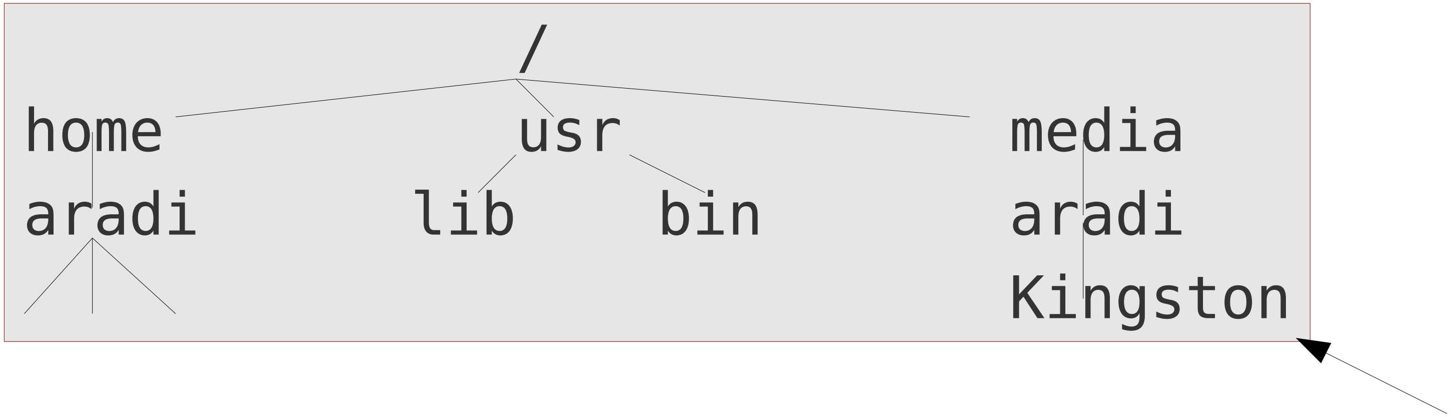
- Editor
- Python-interpreter
- Any kind of application programs

Interact with the operating system

- Change permissions for a file
- Stop, suspend, restart running programs

File system

- Hierarchical: All files are part of **one single tree structure**
- There is one single top node (root folder): **/** (NOT \!)



/

home

aradi

usr

lib

bin

media

aradi

Kingston

/media/aradi/Kingston

- Levels in the tree separated by /
- **Path** of a file: how can it be reached from root
- No drive letters (A:, C:, etc.)
- Mobile devices appear in special directories when inserted – **mounting** a device (e.g. /media/aradi/Kingston)
- When device is removed, special directory disappears (**unmounting**)

Important directories

HOME-directory

- Every user has an own special directory
- All user created files should be stored within that directory
- Permissions for access by other users can be changed
- Often (but not necessary) the directory *`/home/username`*

Directories with executable programs

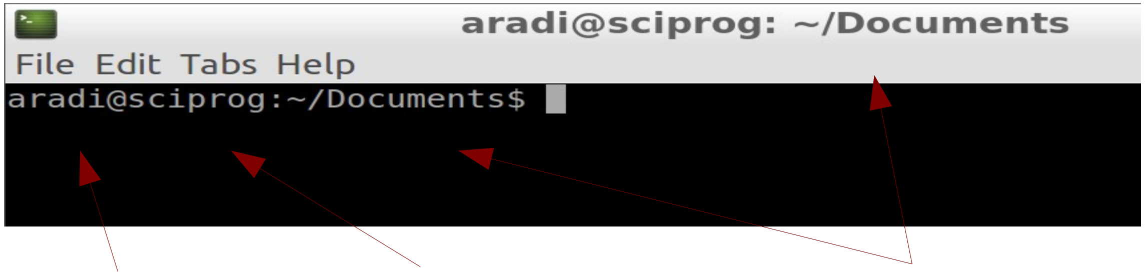
- Contain the programs which can be executed by the user
- Typically `/bin`, `/usr/bin`, `/usr/local/bin`, etc.

Temporary directory (/tmp)

- Running programs store temporary data here
- Usually gets cleaned up at start up
- Never store anything permanent here!

Current working directory

Current working directory is usually shown at the **prompt**



```
aradi@sciprog:~/Documents$
```

User name

Host name

Current working directory

The character tilde (**~**) is the abbreviation for the HOME-directory

`~/Documents` = `/home/aradi/Documents`

Command pwd (**p**rint **w**orking **d**irectory) shows current folder:

```
pwd
/home/aradi/Documents
```

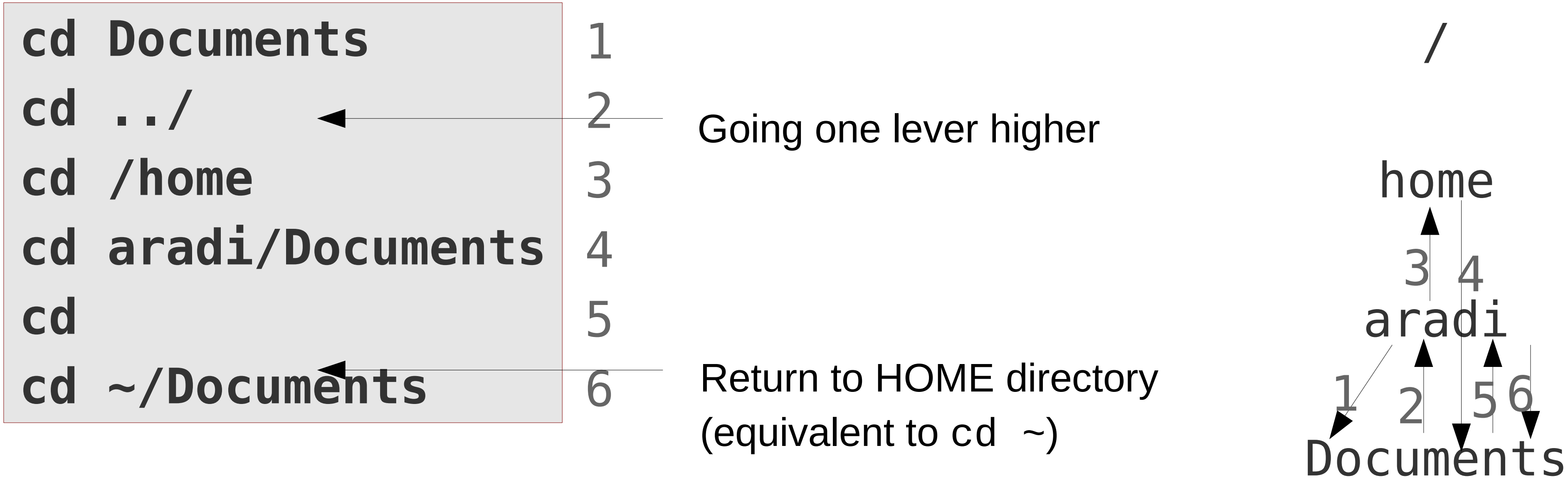
Command

Response (full path starting from /)

Navigating in the directory tree

- Command **cd** (**c**hange **d**irectory) changes between directories

  Usage: **`cd`** ***`DirectoryName`***



```
cd Documents          1
cd ../                2
cd /home              3
cd aradi/Documents    4
cd                    5
cd ~/Documents        6
```

Going one lever higher

Return to HOME directory (equivalent to `cd ~`)

- Absolute path: When relative to / (e.g. `cd /home`)
- Relative path: When relative to current working directory (e.g. `cd Documents`)

Create and remove directories

- Command **`mkdir`** (make directory) creates a directory

  Usage: **`mkdir`** ***`DirectoryName`***

- OS does not change into the newly created directory

- Command **`rmdir`** (remove directory) removes an empty directory

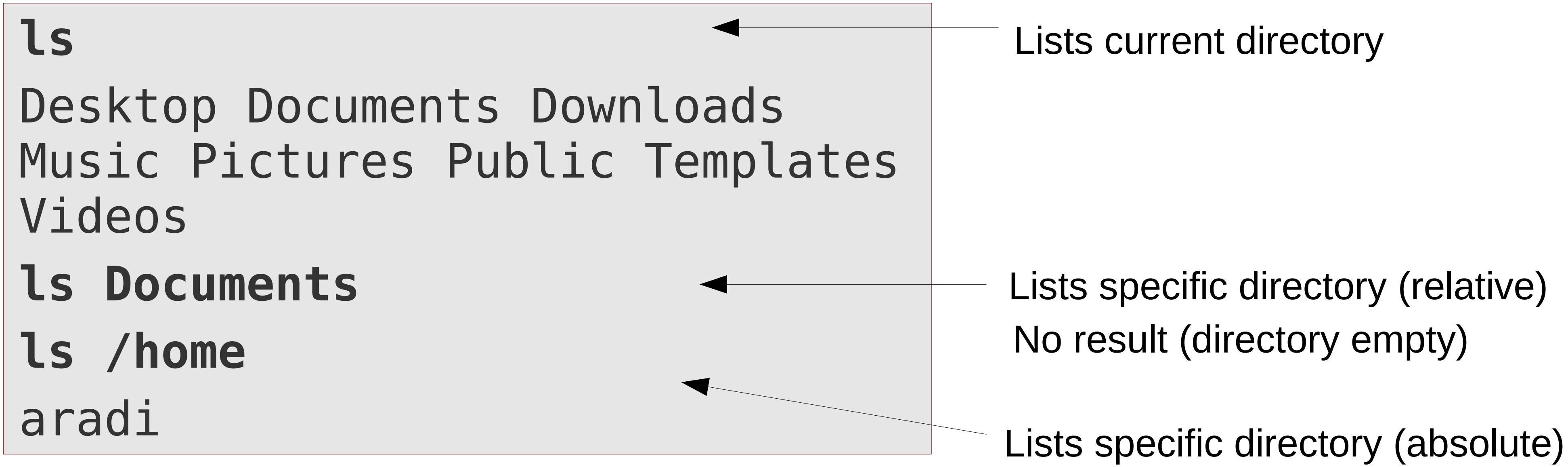
  Usage: **`rmdir`** ***`DirectoryName`***

```
cd
mkdir test
cd test
cd ../
rmdir test
```

- Directory name can be relative or absolute

Listing files and directories

- Command **ls** (**l**i**s**t) lists the content of a directory (or specific files)



```
ls
Desktop Documents Downloads
Music Pictures Public Templates
Videos
ls Documents
ls /home
aradi
```

Lists current directory

Lists specific directory (relative)

No result (directory empty)

Lists specific directory (absolute)

Command options and arguments

Unix commands accept two different kind of arguments

Optional arguments (options)

- Modify the behaviour of the command
- Always optional and can be left away, if standard behaviour is desired
- Start with dash (“-”) or double dash (“--”)

Positional arguments (arguments)

- Usually specify the targets of the command (typically file names)
- Are sometimes optional, but often compulsory

Command options and arguments

```
mkdir test
cd test
touch file1.dat file2.dat .hidden
mkdir subdir
```

```
ls
file1.dat  file2.dat  subdir
```

No options, no arguments

```
ls file1.dat
file1.dat
```

No options, one argument

```
ls -l -h
total 12K
-rw-rw-r-- 1 aradi aradi    7 Apr  2 18:01 file1.dat
-rw-rw-r-- 1 aradi aradi    6 Apr  2 18:01 file2.dat
drwxrwxr-x 2 aradi aradi 4,0K Apr  2 18:11 subdir
```

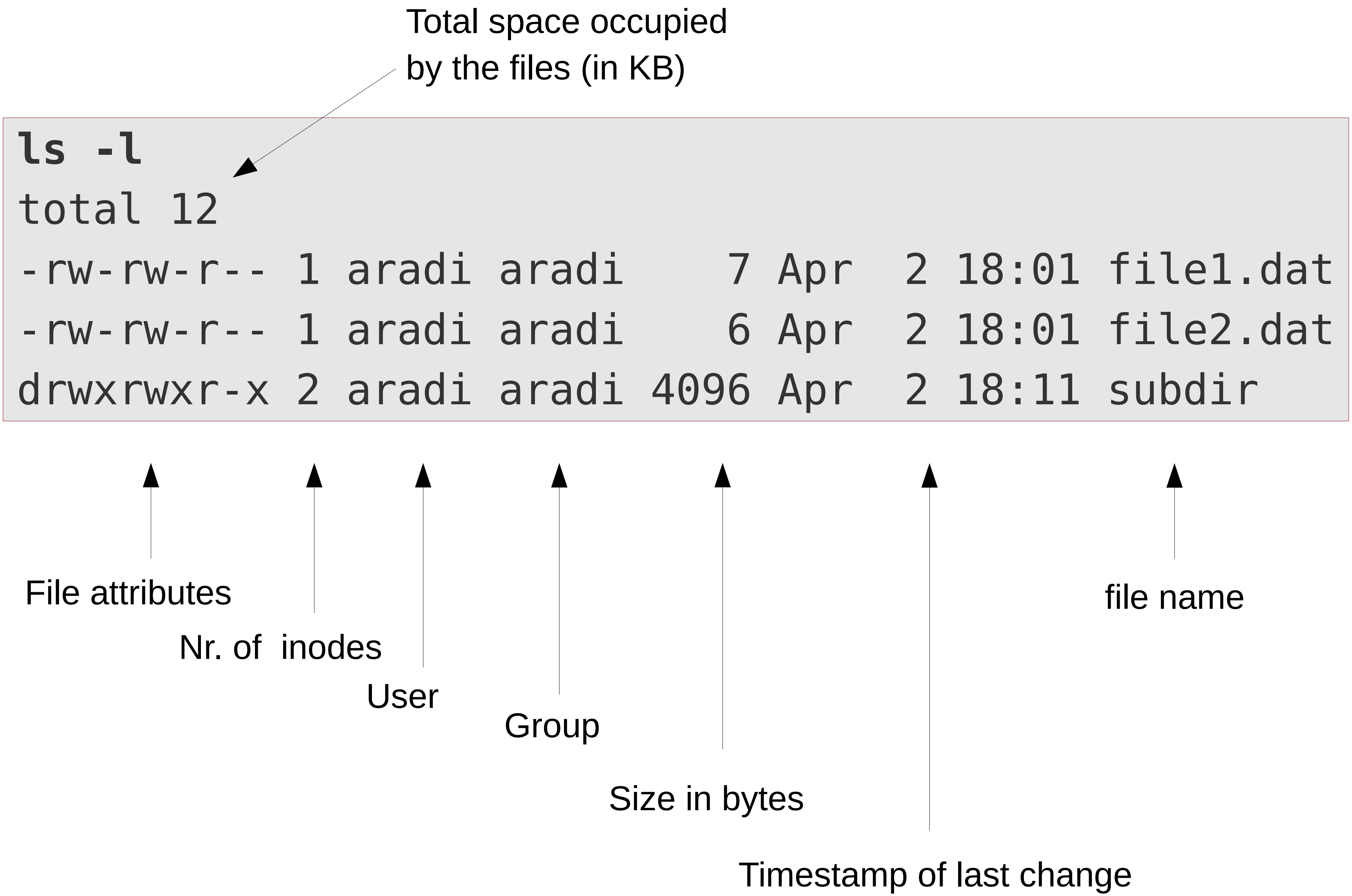
Multiple options, no arguments

```
ls -l -h file1.dat file2.dat
-rw-rw-r-- 1 aradi aradi 7 Apr  2 18:01 file1.dat
-rw-rw-r-- 1 aradi aradi 6 Apr  2 18:01 file2.dat
```

Multiple options, multiple arguments

Options for ls

- **-l** (long listing)



Total space occupied by the files (in KB)

```
ls -l
total 12
-rw-rw-r-- 1 aradi aradi    7 Apr  2 18:01 file1.dat
-rw-rw-r-- 1 aradi aradi    6 Apr  2 18:01 file2.dat
drwxrwxr-x 2 aradi aradi 4096 Apr  2 18:11 subdir
```

File attributes

Nr. of inodes

User

Group

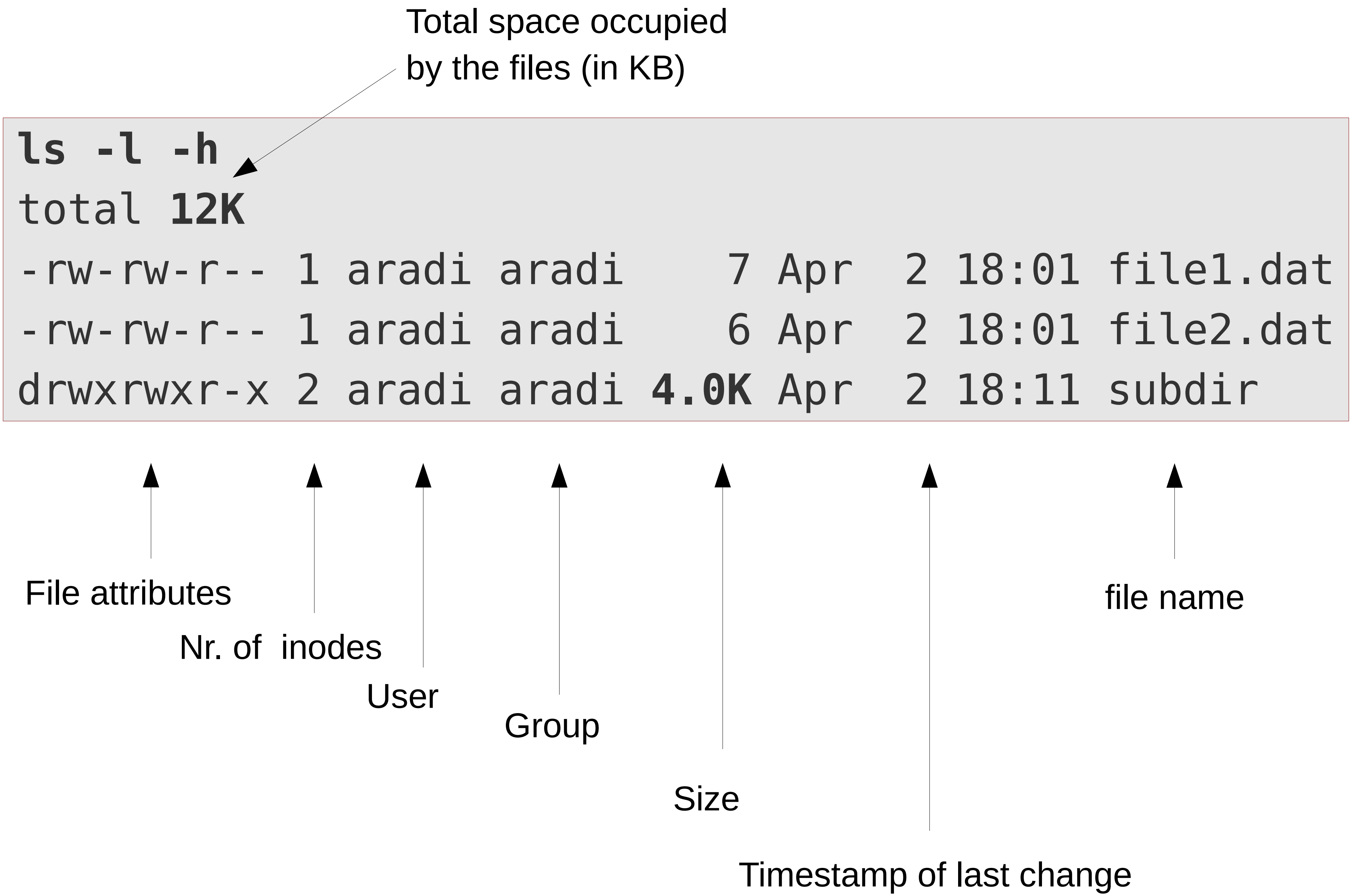
Size in bytes

Timestamp of last change

file name

Options for ls

- **-l -h** (long listing, human readable): like -l, but sizes with prefixes



Total space occupied by the files (in KB)

```
ls -l -h
total 12K
-rw-rw-r-- 1 aradi aradi    7 Apr  2 18:01 file1.dat
-rw-rw-r-- 1 aradi aradi    6 Apr  2 18:01 file2.dat
drwxrwxr-x 2 aradi aradi 4.0K Apr  2 18:11 subdir
```

File attributes

Nr. of inodes

User

Group

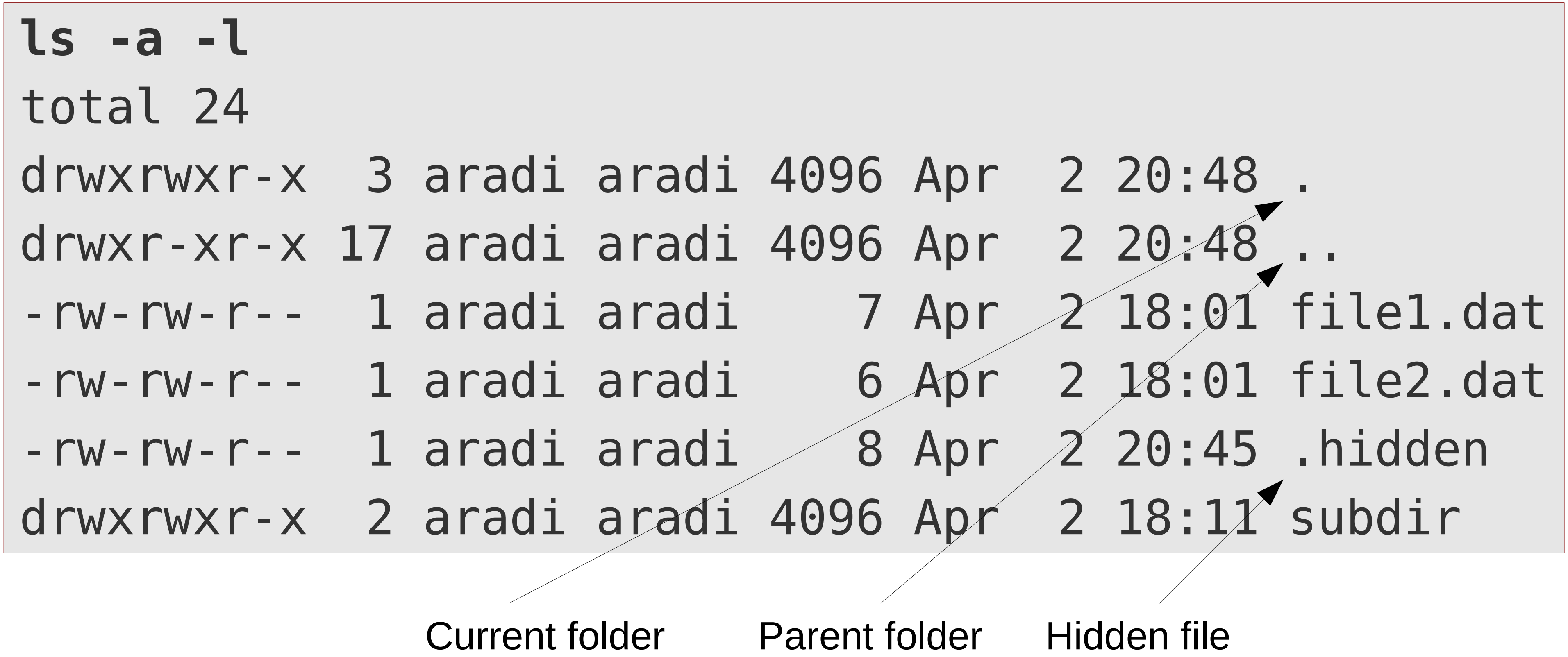
Size

Timestamp of last change

file name

Options for ls

- **-a** (all): Shows also hidden files and folders (name starts with "**.**")



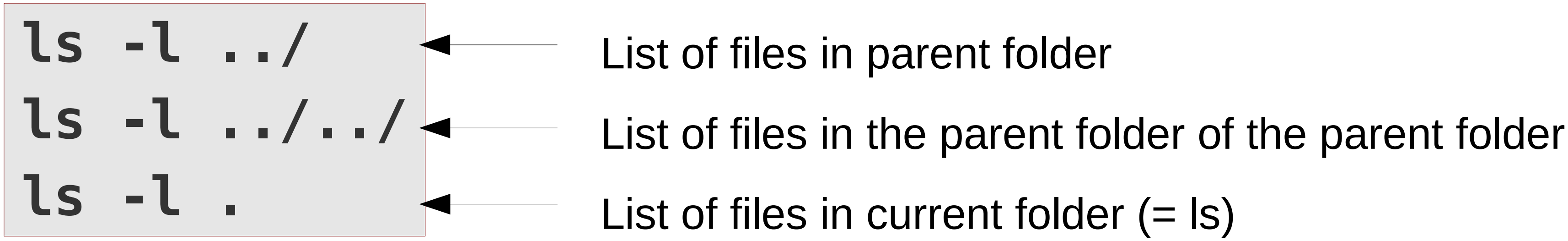
```
ls -a -l
total 24
drwxrwxr-x  3 aradi aradi 4096 Apr  2 20:48 .
drwxr-xr-x 17 aradi aradi 4096 Apr  2 20:48 ..
-rw-rw-r--  1 aradi aradi    7 Apr  2 18:01 file1.dat
-rw-rw-r--  1 aradi aradi    6 Apr  2 18:01 file2.dat
-rw-rw-r--  1 aradi aradi    8 Apr  2 20:45 .hidden
drwxrwxr-x  2 aradi aradi 4096 Apr  2 18:11 subdir
```

Current folder

Parent folder

Hidden file

Folder names **.** and **..** can also be used in various commands:



```
ls -l ../
ls -l ../../
ls -l .
```

List of files in parent folder

List of files in the parent folder of the parent folder

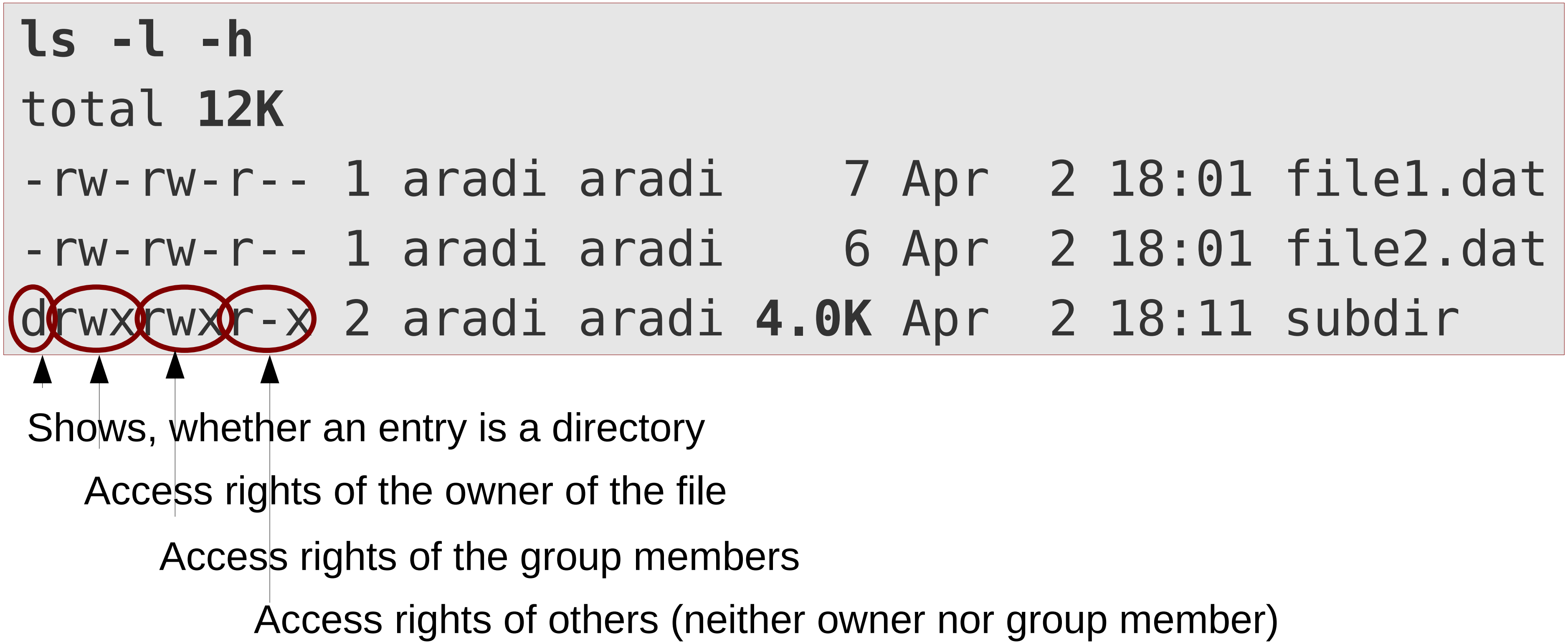
List of files in current folder (= ls)

Help! – man pages

- Options and arguments for a given command can be looked up in the manual
- Usage: **`man`** ***`Command`*** (e.g. `man ls`)
- Navigation on the man-page:
  - **Page Up / Page Down** (**Seite Auf / Seite Runter**) – going up and down
  - **q** – Exit the man page
  - ***/word***[ENTER] – Search forward for a given word and go to first match
  - **?word**[ENTER] – Seach backward for a given word and go to first match
  - **n** – go to the next match of the last search

File attributes

Attributes set access permissions for given entry



```
ls -l -h
total 12K
-rw-rw-r-- 1 aradi aradi    7 Apr  2 18:01 file1.dat
-rw-rw-r-- 1 aradi aradi    6 Apr  2 18:01 file2.dat
drwxrwxr-x 2 aradi aradi 4.0K Apr  2 18:11 subdir
```

Shows, whether an entry is a directory

Access rights of the owner of the file

Access rights of the group members

Access rights of others (neither owner nor group member)

Permission rights:

- **r** **read**
- **w** **write**
- **x** **execute** (if file),
  change into (if directory)

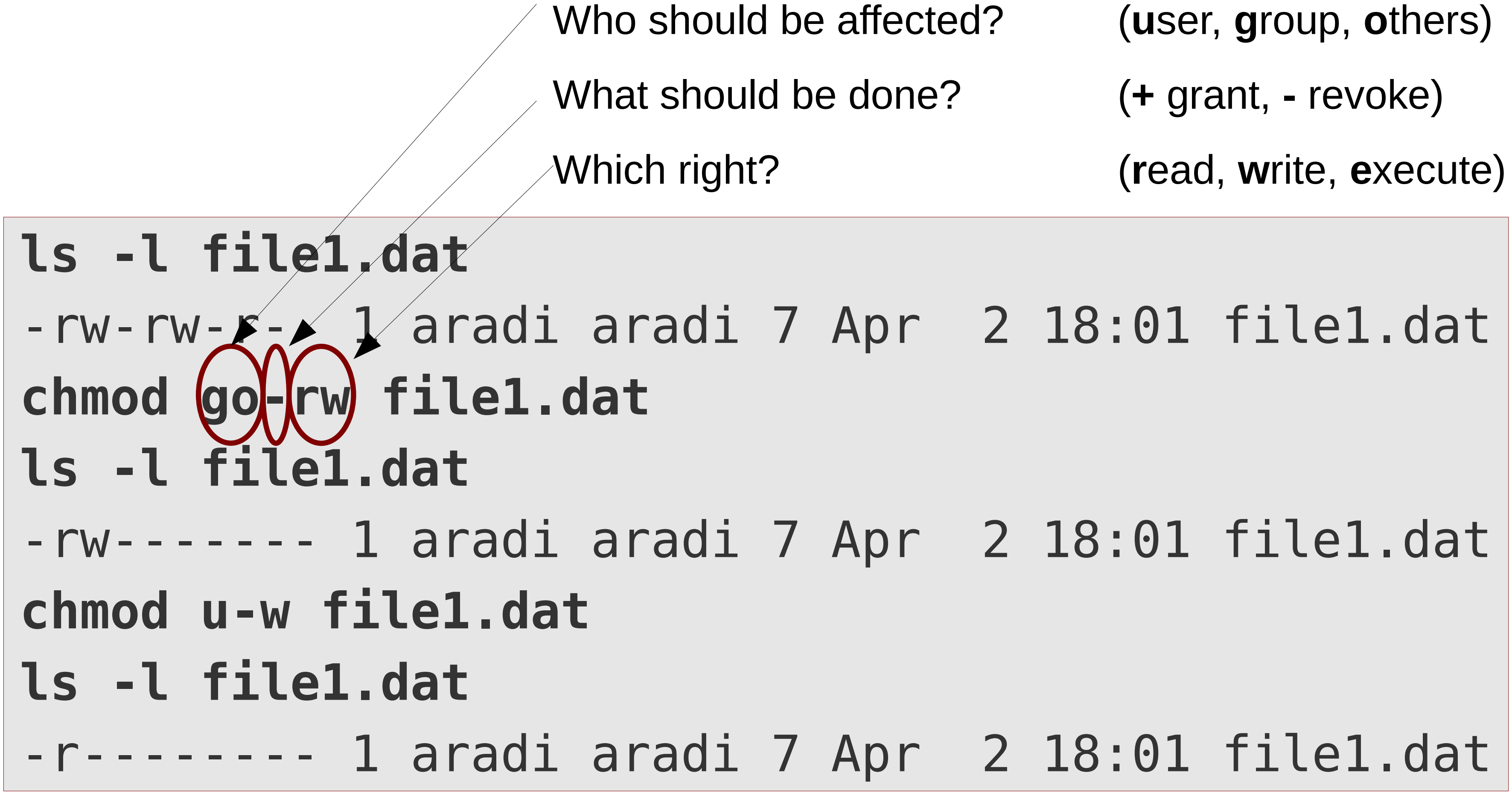
Each user belongs to several groups:

```
id -G -n
aradi adm cdrom sudo
dip plugdev lpadmin
sambashare vboxsf
```

Changing file attributes (chmod)

- User(s) having write access to a file can change their attributes
- Command: **chmod** (**ch**ange file **mod**e bits)

  Usage: **chmod** ***Change FileOrDir***



Who should be affected? (**u**ser, **g**roup, **o**thers)

What should be done? (**+** grant, **-** revoke)

Which right? (**r**ead, **w**rite, **e**xecute)

```
ls -l file1.dat
-rw-rw-r-- 1 aradi aradi 7 Apr  2 18:01 file1.dat
chmod go-rw file1.dat
ls -l file1.dat
-rw------- 1 aradi aradi 7 Apr  2 18:01 file1.dat
chmod u-w file1.dat
ls -l file1.dat
-r-------- 1 aradi aradi 7 Apr  2 18:01 file1.dat
```

Wildcards

- Instead of file and directory names, special placeholders can be used to indicate files/directries matching a given pattern

Wildcard	Matching pattern
`*`	arbitrary character or characters (including nothing)
`?`	arbitrary character (exactly one)
`[0-9,a,...]`	one character matching any of the listed characters or character intervals
`[!0-9,a,...]`	one character not matching any of the listed characters or character intervals

Wildcards

```
touch file{,1,2,3,4,A,B,C,D}.dat
```

```
ls
file1.dat  file3.dat  fileA.dat  fileC.dat  fileD.dat
file2.dat  file4.dat  fileB.dat  file.dat  subdir
ls file*.dat
file1.dat  file3.dat  fileA.dat  fileC.dat  fileD.dat
file2.dat  file4.dat  fileB.dat  file.dat
ls file?.dat
file1.dat  file3.dat  fileA.dat  fileC.dat
file2.dat  file4.dat  fileB.dat  fileD.dat
ls file[1-4,A].dat
file1.dat  file2.dat  file3.dat  file4.dat  fileA.dat
ls file[!1-4,A].dat
fileB.dat  fileC.dat  fileD.dat
ls *[A-C].dat
fileA.dat  fileB.dat  fileC.dat
```

Copy and move (rename) files

- **cp** (copy) and **mv** (move) commands can be used to copy and move files
- Usage:

cp *File Copy*	Make a copy
cp *Files TargetDir*	Make a copy in a different directory
mv *FileOrDirectory NewName*	Rename
mv *FilesOrDirectories TargetDir*	Move into a different directory

```
cp file1.dat newfile1.dat
cp file1.dat ../newfile1.dat
mkdir newdir
cp file*.dat newdir
cp -r newdir newdir2

mv file1.dat newfile1.dat
mkdir newdir3
mv fileA.dat newdir3
```

Recursive copy: copy dir1 and all its content (including subdirectories)

Delete files (rm)

- **`rm`** (remove) command can be used to delete files
- Usage:

rm *Files* Removes specified files

Remove does not ask for confirmation!!!

THINK TWICE BEFORE HITTING [ENTER]!

```
rm fileC.dat
rm *.dat
```

rm -r *FilesOrDirs* Removes specified files and directories, including all subdirectories (recursive delete)

```
rm -r newdir1
rm -r *
```

Be very-very careful with this!!!

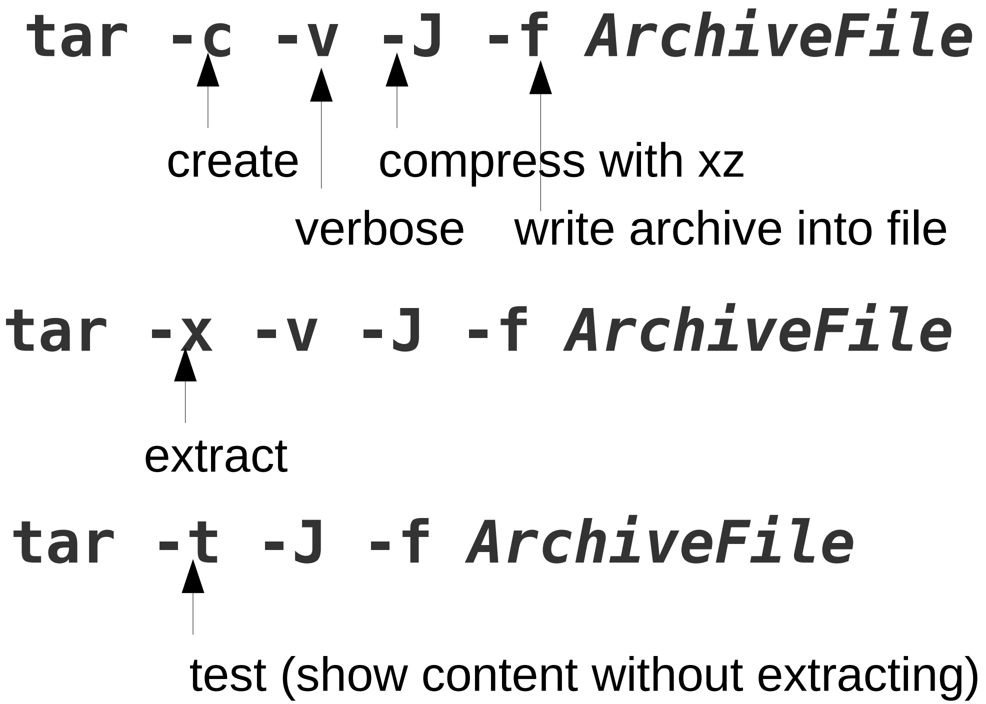
rm -i *FilesOrDirs* Interactive delete (asks for confirmation for every file)

```
rm -i file2.dat
rm -r -i newdir2
```

Creating archives (tar)

- Creates / Extracts an xz-compressed archive of files and directories

Usage:



```
tar -c -v -J -f ArchiveFile FilesDirsToArchive
```

create

verbose

compress with xz

write archive into file

```
tar -x -v -J -f ArchiveFile
```

extract

```
tar -t -J -f ArchiveFile
```

test (show content without extracting)

Note: Archive extraction overwrites files without confirmation!

```
tar -c -v -J -f test.tar.xz test

tar -t -J -f exercise1.tar.xz

tar -x -v -J -f exercise1.tar.xz
```

Creates a compressed archive (dir1.tar.xz) of the directory dir1

Shows archive content

Extracts the compressed archive (exercise1.tar.xz) in the current directory

Command line navigation

- The shell remembers the command lines entered
- Within the command line and between the command line can be navigated with following keys (similar to Emacs key-binding)

`Ctrl-A or Home`	Jump to the start of the line
`Ctrl-E or End`	Jump to the end of the line
`Up`	Go one line backwards in history
`Down`	Go one line forwards in history
`Ctrl-K`	Cut (kill) from position to end of line
`Ctlr-Y`	Insert (yank) last cut
`Ctrl-R`	Search backwords in history

Command line completion

- When you hit the **[TAB] key** during entering a command/file name, the shell tries to extend it automatically
- The command/argument will be extended, up to the point, where the extension is unique.
- If the extension is not unique, hitting **[TAB] twice** shows a list of possible extensions

```
ls
file1.dat  file3.dat  fileA.dat  fileC.dat  fileD.dat
file2.dat  file4.dat  fileB.dat  file.dat
rm f[TAB]
rm file[TAB][TAB]
file1.dat  file3.dat  fileA.dat  fileC.dat  fileD.dat
file2.dat  file4.dat  fileB.dat  file.dat
rm fileB[TAB]
rm fileB.dat
```

Editing files

- Linux offers many different editors to edit files
- The most popular (classic) ones: **vi** and **emacs**
  - Both are increadibly powerful, but it needs some exercising to get used to them (however, a must for geeks)
- Depending on the GUI, you may have additional different graphical editors (**gedit**, **kate**).
- Lubuntu offers a simple editor: **leafpad**

  Usage:

```
leafpad FileName
```

```
leafpad file1.dat &
```

Opens the file file1.dat

Advises the shell to execute the command in the background.

Practical when starting graphical applications from the command line, as they run in a separate window, and command window can then be used for entering further commands while they are running.

Initialisation files

- Commands, which should be always executed whenever a command terminal is opened, can be written in an shell-initialisation file
- The initialisation file is automatically executed whenever a shell is started.
- Bash-shell has two initialisation files:
  - **~/.bashrc**
    Executed, whenever a non-login shell is opened (e.g. opening a terminal in Lubuntu)
  - **~/.profile**
    Executed, whenever a login-shell is opened (e.g. logging in via SSH)

Setting aliases

- An alias replaces a complex shell command with a simple name
- It can also be used to apply options without specifying them each time
- Usage:

```
alias aliasname=”command to execute”
```

```
alias rm=”rm -i”
alias mv=”mv -i”
alias cp=”cp -i”
```

Invoke remove, move and copy with the interactive option. They will ask for confirmation before deleting anything.

```
rm file1.dat
rm: remove regular empty file 'file1.dat'? y
```

- Aliases are typically added to the shell initialisation file (e.g. `~/.bashrc`)
- You can still use the original command by prepending `\` to it

```
\rm file1.dat
```

It will not ask for confirmation, as it does not use the alias but the original command

Exercise

See the course web site for the exercises!