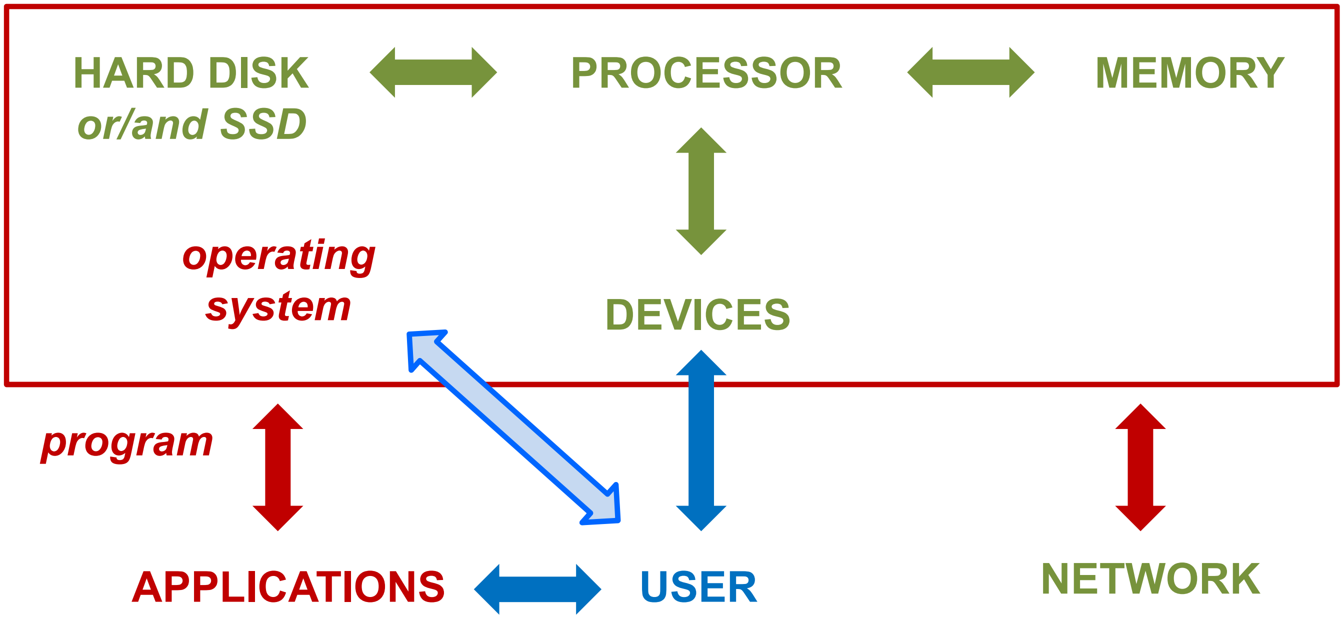
UNIX
(START)

Operating systems

- Simplified representation of a computer



HARD DISK
or/and SSD

PROCESSOR

MEMORY

operating system

DEVICES

program

APPLICATIONS

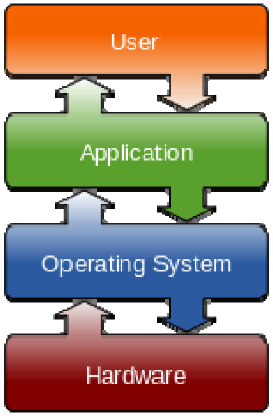
USER

NETWORK

🟩 *hardware*

🟥 *software*

🟦 *(brainware)*



- The operating system (OS)

→ controls the hardware / is accessed by applications

→ common variants: Windows (PC), OS X (Mac), UNIX, linux

are actually variants of UNIX

- UNIX (developed in Berkeley in 1974)

→ **text-based** (no clicking!)

→ most used by IT **professionals** and **scientists** (servers, [super]computers)

→ **linux**: non-commercial variant of UNIX, since 1991

The UNIX Operating system

- Drawback of UNIX

  → A bit **harder** to learn

- Advantages of UNIX

  → More **transparent**

  → Easier to **automatize** (scripts)

In this lecture, you will have a bit "the pain without the gain"... But if you explore further and start scripting, you will rapidly see how much efficiency you can gain over Windows or OS X front-ends when you have to do repetitive tasks !

- Want to have UNIX on your Windows PC ?

  → Install it with linux

  → Install it dual-boot (Windows + linux)

  → Install a UNIX emulator, *e.g.* cygwin (non-commercial): www.cygwin.com

- Want to have UNIX on your Mac PC ?

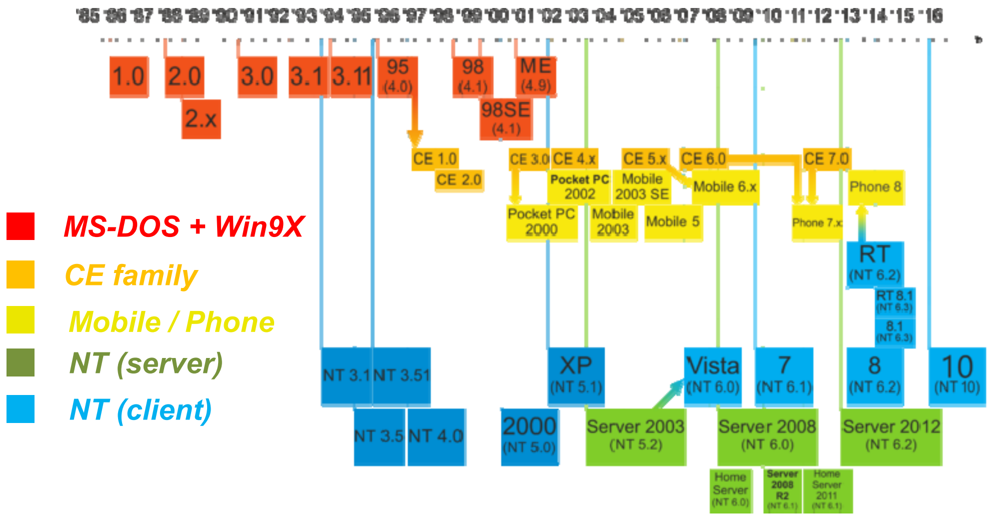
  → OS X is already UNIX based (you just need to find out how to access it directly)

HS14→22

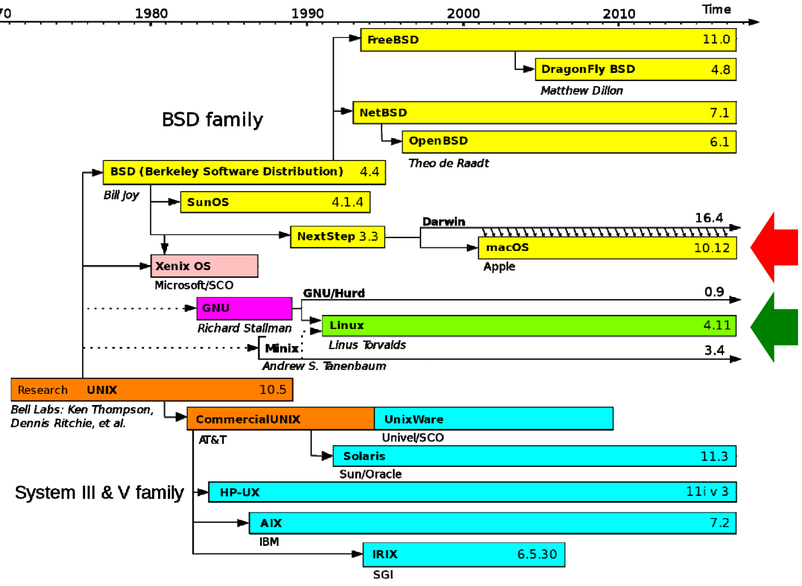
Operating systems

- To be precise

Windows is a specific operating system (anciently MS-DOS)



But Mac OS X is actually a form of UNIX



- To summarize, you can use modern PCs at ***two levels***

	Windows PC	***Mac PC***	***Linux PC****
Direct OS level	*MS-DOS-like command prompt***	*UNIX terminal (shell)*	*UNIX terminal (shell)*
Front-end level (GUI)***	⬅ *window-based (click & drag & drop & copy & paste)* ➡		

: there exist many linux variants nowadays (our PC's run "Fedora")

*****: or install cygwin (UNIX shell emulator)***

******: GUI = Graphical User Interface***

Note: modern operating systems are (largely) programmed in C++ !

⬅ ***most people use this level, and then all is more or less similar***

starting
next week:

+read first: Ex0
Starting document for
students

"The computer says I need to upgrade my brain
to be compatible with the new software."

... duration 2 weeks

EXERCISE SERIES 1

Working with UNIX

(mini-project: file manipulation/processing/visualization using UNIX)

Getting started

- **Logging in**

→ Type your nethz **username** and **password** (case sensitive!)

→ Open a **terminal window** / type `csh` — ***starts the "c-shell" unix variant***

→ **Prompt** (computer ready to recieve input) `[user@comp dir]$` — ***the form of the prompt varies from system to system (you can even customize it if you like...)***

- Typing/changing a **command line**

You don't need to learn all this from the start...
But at some point, this is what will give you <u>SPEED</u> !

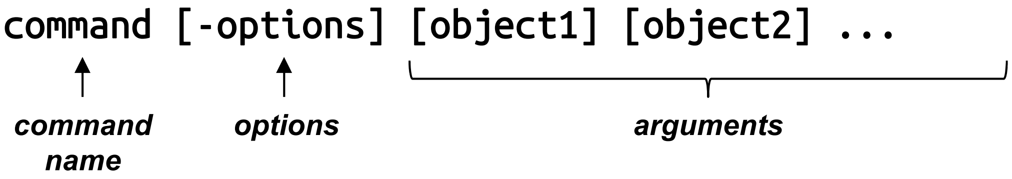
→ Type text…	Well, just type it…	
→ Move cursor along text	<←> and <→>	
→ Jump to line start	`<CTRL-a>`	
→ Jump to line end	`<CTRL-e>`	
→ Delete one character before cursor	`<BACKSPACE>`	
→ Delete one character after cursor	`<DEL>` or `<CTRL-d>`	***Note: <CTRL-d> on an empty line will log you out instead!***
→ Delete all after cursor	`<CTRL-k>`	
→ Delete entire line	`<CTRL-u>`	***For real die-hards, try <CTRL-z>***
→ Quit unfinished line / terminate execution of current command (type once or twice)	`<CTRL-c>`	***This is what you have to do to interrupt a command that never stops (silently or verbosely)***
→ Browse through command history	<↑> and <↓>	***Try it: it will save you <u>A LOT OF TYPING</u> !!!***
→ Execute command line	`<ENTER>`	

- **Logging out**

→ Logout of system (computer)	`<CTRL-d>` or `logout` or `exit`	***(twice: once to exit csh, once more to kill terminal window)***

Commands

- The UNIX **commands** are of the form



```
command [-options] [object1] [object2] ...
```

command name ***options*** ***arguments***

→ The *square brackets* mean “optional” (need, number and types depend on the command)

→ The *options* are preceded by a minus sign and further specify/modulate the action of the command

→ The *arguments* define objects (*e.g.* numbers, text strings, file names) relevant to the command (need, number and types depend and the command and its options); most commands use *defaults* when no arguments are specified

- Multiple UNIX commands on one line

→ Normally, each command is a **single line**

→ But multiple commands can also be given on the same line with a **semicolon separator**

```
command1 ... ; command2 ... ; command 3 ...
```

- Examples of commands: later...

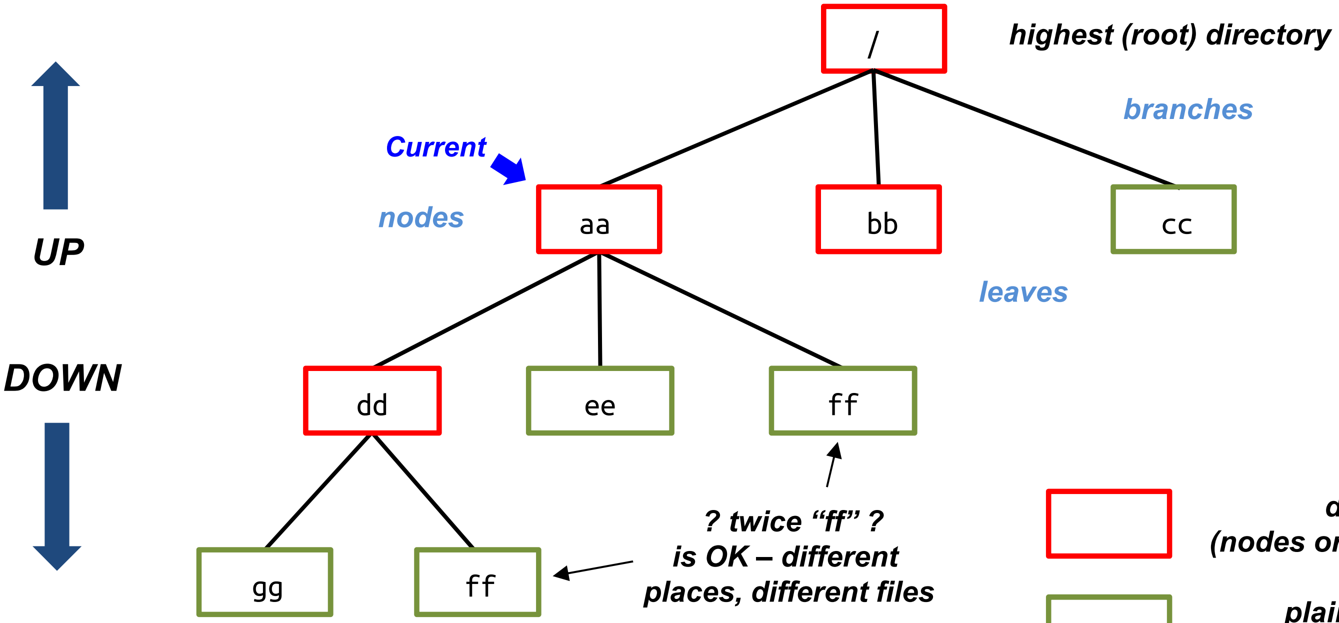
Directory structure

@003

- The UNIX **directory structure** is an upside-down tree

in fact very similar to windows, just windows "hides" the tree using clickable icons

[contains everything!]



highest (root) directory

branches

Current

nodes

leaves

? twice "ff" ? is OK – different places, different files

In this tree, __relative__ names are given in the boxes

🟥 *directories (nodes or leaves [if empty])* *[list of pointers to objects one level below]*

🟩 *plain (data) files (always leaves)*

- Referencing a file or directory

→ By **absolute** path from **root**, *e.g.* `/aa/dd/ff`

start with a slash → absolute

→ By **relative** path from the **current (working) directory**, *e.g.* if my current directory is `/aa` , I may just write `dd/ff`

start without slash → relative

- Reading or setting the current directory

→ **Get** its *absolute path* (*i.e.* print it to screen) with the command: `pwd` ➡ *writes to terminal window* `/aa`

→ **Set** it with the command: `cd dir_name` *e.g.* `cd /aa`

Directory structure

- The following shortcuts to specify **directories** are very handy

Root directory	The top directory of the system	`/`
Current directory	The one you are currently in	`.`
Home directory	The highest directory for you as a user	`~`
A child directory	One down from your current directory	`dir_name`
The parent directory	The one up from your current directory	`..`

for the home directory of user "batman" `~batman`

→ Can be combined, *e.g.* : `../../xx` `~/xx` `~/./../batman`

a silly way to say the same

- It is custom to append **extensions** to filenames using a **dot**

Text file	`.txt`
C++ code source file	`.cc`
Compiled object file	`.o`
Data file	`.dat`

Executable files (commands, i.e. scripts or programs) usually have no filename extension (on windows, typically «.exe»)

Note: windows also has these extensions but they are not shown in the filename (by default – you can actually change this if you find the proper settings menu) – they are used to select the type of icon you see on screen

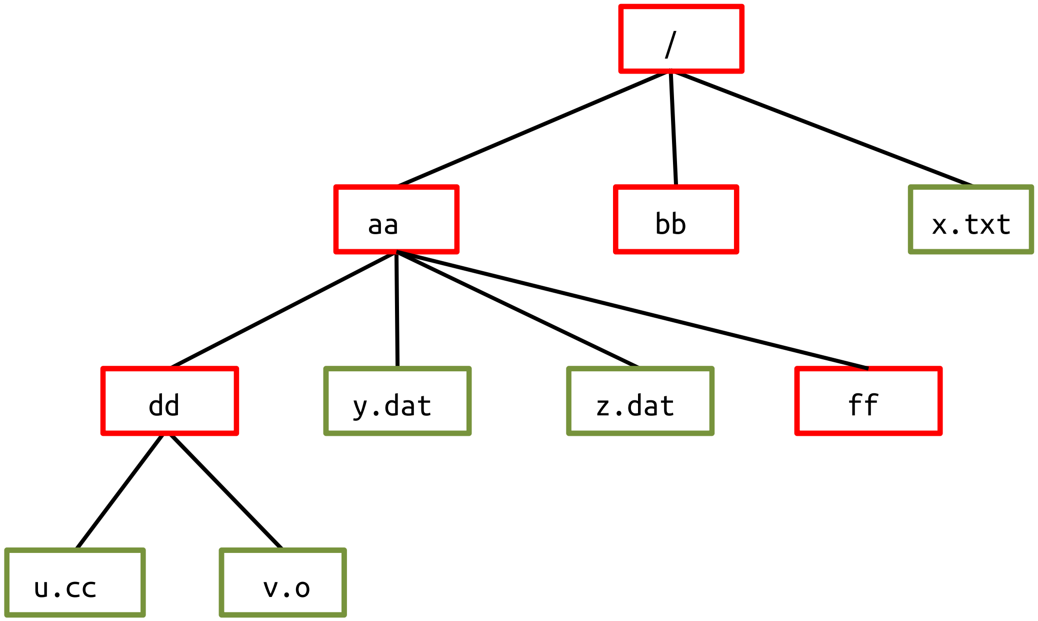
- When interpreting commands pertaining to files, certain **wildcards** can also be used; they are expanded to **lists** in the following way

		Example
Any single character	`?`	`results?.dat`
Any string of character	`*`	`results*.txt` `*.dat` `*`
Any character from list	`[ABC]`	`results[1234].dat`
Any character in range	`[A-Z]`	`results[a-zA-Z][0-9]?.*`

no character is not a match, e.g. «rm a?» will not erase a file «a»

here, no character is also a match, e.g. «rm a*» will also erase a file «a»

Directory structure



In this tree, <u>relative</u> names are given in the boxes

🟥 ***directories***

🟩 ***plain (data) files***

- ***Questions:***

→ Give absolute filenames for files in the above tree

→ Assume `/aa` to be your **current** directory, give filenames relative to it

→ Assume `/bb` to be your **home** directory, give filenames using “~”

→ What will `/aa/*` be expanded to?

→ What will `/aa/?.dat` be expanded to?

→ What will `/aa/??` be expanded to?

→ What will `/aa/[b-es-y]*` be expanded to?

Handling directories and files

- Printing or changing the **current directory**

Display name of current directory	`pwd`	`cd` *with no argument brings you home (i.e. like* `cd ~`*)*
Make specified directory the new current directory	`cd dir_name`	`cd ..` *brings you one up*

- **Creating** or **deleting** directories and files

Delete specified file	`rm file_name`
Create specified new (empty) directory	`mkdir dir_name`
Delete specified (empty) directory	`rmdir dir_name`
Delete specified directory and all its content	`rm -rf dir_name`

- Displaying **content** of directory or data file

Lists files in current directory	`ls`	
Lists files in specified directory	`ls dir_name`	
Lists all files (including “hidden” files)	`ls -a`	*hidden files are those having a name starting with a dot*
Lists files with extra information	`ls -l`	
Displays content of specified data file	`cat file_name`	
Displays content page by page	`more file_name`	*alternative (I prefer)* `less`
Displays type of object (file or dir) and type of contents	`file file_name`	

- ***Questions:***

→ What does the command `rm */*` do ? And the command `rm -rf *` ?

Directory structure

- For people who like it ***super precise***

→ Only a ‘/’ at the start means “absolute path” – other ‘/’ are just ignored

e.g. `aa//bb` is interpreted as `aa/bb`

→ Only a ‘~’ at the start is allowed

e.g. `aa/~/bb` will give an error `No such file or directory`

→ The use of ‘.’ as meaning “current directory” is normally unnecessary in a path

(but can be useful to copy or move files; see later)

e.g. `./aa` is interpreted as `aa`

`aa/./bb` is interpreted as `aa/bb`

interesting exception: when you run a command, UNIX will look for it in a standard set of directory

to see which, just type

```
echo $path
```

(UNIX searches along this list and stops at first match)

if the command, *e.g.* `my_command`, is in your current directory but this directory is not in the standard set, then

`my_command` will give an error `my_command: Command not found`

`./my_command` will work and execute the command

?

To see where a UNIX command, e.g. `cat`, ***is actually located, type:***

```
which cat
```

Gives usually:

```
/bin/cat
```

In practice, most users set up their $path to have ‘.’ at the start of the list

Permissions

- Unix distinguishes **file-access permissions**

  → for the *user* (*i.e.* the owner of the file)

  → for the *group* (*i.e.* the user-group including the owner of the file)

  → for the *others* (*i.e.* anyone who has an account on the computer)

- The permissions can always be changed by the **owner** of the file (irrespective of the current permissions)

  → to change permissions use the command `chmod UGO file_name` where `UGO` is the three-digit octal string determining the permissions (U: user; G: group; O: other), each digit in the range 0-7

  → octal digit

  0 = none
  1 = execute (x)
  2 = write (w)
  3 = write+execute (wx)
  4 = read (r)
  5 = read+execute (rx)
  6 = read+write (rw)
  7 = read+write+execute (rwx)

TRICK:
Start from 0
Add 4 for «read»
Add 2 for «write»
Add 1 for «execute»

- Example

```
chmod 700 file_name
```

  → gives rwx permissions to the user, and no permission for anyone else

- ***Questions:***

  → What does the command `chmod 754 *.*` do ?

Permissions

- Another way to change the **file-access permissions**

  → you can also use the command `chmod ugoa±rwx file_name`
  where `u`, `g`, `o` or/and `a` determine who is concerned by the change (`a`=all),
  `+` or `-` grants or retracts a permission, and `r`, `w` or/and `x` is the specific right

- Example

```
      chmod a-rwx file_name
then  chmod u+rwx file_name
```

Note: the first command removes rwx permission from the user, but not his right to further change the file permissions (since she/he remains owner of the file)

  → gives the same result as

```
chmod 700 file_name
```

  → gives rwx permissions to the user, and no permission to anyone else

- ***Questions:***

  → How would would one translate `chmod 754 *.*`
  into this second formalism ?

- The **directory-access permissions** are defined in a slightly different way

  → *read* : right to read the names of files in the directory (but if alone, no additional information)

  → *write* : right to modify entries in the directory (creating files, deleting files, renaming files)

  → *execute* : right to access file contents and metainfo (but alone, not to list the directory)

This is a bit paradoxical. With x-only permission on a directory “dir” containing a file “file”, you can do “ls dir/file” but not “ls dir” !

Student questions

- UNIX **filesystem permissions**: owner vs. user ?

→ Example

```
% ls -l ~phil
drwxr-xr-x 21 phil igc     4096 2013-08-28 07:46 adm
drwxr-xr-x  5 phil igc     4096 2013-02-26 07:45 arc
drwxr-xr-x  3 phil igc     4096 2013-08-28 09:22 bin
drwxr-xr-x 13 phil igc     4096 2013-07-29 16:11 crt
-rw-r--r--  1 phil igc   143074 2013-09-26 21:41 paper.pdf
-rw-r--r--  1 phil igc   134144 2013-09-25 12:09 questionaire.doc
...
```

prompt

type (-: file; d: directory)

permissions UGO

number of directories inside directory

owner = user

group

size of object in bytes

[for directories including all content, use: `du -s`*]*

last modification date & time

name

→ By default, when you create a file/directory at a place where you have permissions to do so, you are automatically the **owner**

→ The **owner** can **always** change the permissions with `chmod`

Including changing her/his own permissions as user... (the ability to use chmod as owner does not depend on the permissions you set for yourself as user)

→ If she/he belongs to more than one group, the **owner** can change the **group** concerned by the permissions with `chgrp`

→ Only a **superuser** (system administrator; usually a “Gandalf-The-White”) with user name **root** can change the **owner** of a file with `chown`

(the superuser also bypasses all permissions)

Creating, copying, renaming and deleting data files

- Two ways to **create a file**

See also "touch": create empty file

→ **Standard output** of a UNIX command

```
cat > file_name
...
...
<CTRL-d>
```

Painful! (once a line is written, you cannot go back to it)

Places characters from keyboard into the file with specified name until <CTRL-d> (EOF)

Recommended!

Much more powerful (but for later...)

→ Using an **editor**, *e.g.* `gedit` , `vi` , `emacs` , ...

Will overwrite if the target file already exists (and you have write permission)

- To **copy** a file (plain files)

```
cp file_name_1 file_name_2
```

Copies the first file to the second file; first file is unaffected

- To **rename** a file (plain files)

```
mv file_name_1 file_name_2
```

Renames the file from the first name to the second name; first file no longer exists

- To **delete** or **remove** a file (plain files)

```
rm file_name
```

Deletes the file with the specified name

- ***Questions:***

→ How can you delete all the files in your working directory ? And in your home directory ?

Copying, renaming and deleting files or directories

- More information on **copying**, **renaming/moving** and **deleting**

→ Copying

`cp file_name_1 file_name_2`	Copies file to second file	***if file 2 exists →overwrite***
`cp file_name dir_name`	Copies file into directory	***dir must already exist***
`cp file_name_1 file_name_2 dir_name`	Copies files into directory	
`cp file_name_1 file_name_2 file_name_3`	→ error message	***"cp: target is not a directory"***
`cp dir_name file_name`	→ error message	***"cp: omitting directory"***
`cp dir_name_1 dir_name_2`	→ error message	***"cp: omitting directory"***
`cp -r dir_name_1 dir_name_2`	Copies directory and content as or into second directory	***if dir 2 already exists***

→ Renaming

`mv file_name_1 file_name_2`	Renames file to new name	***if file 2 exists →overwrite***
`mv file_name dir_name`	Moves file into directory	***dir must already exist***
`mv file_name_1 file_name_2 dir_name`	Moves files into directory	
`mv file_name_1 file_name_2 file_name_3`	→ error message	***"mv: target is not a directory"***
`mv dir_name file_name`	→ error message	***"mv: cannot overwrite non-directory with directory"***
`mv dir_name_1 dir_name_2`	Renames directory to new name or move it into second directory	***if dir 2 already exists***

→ Deleting ***[already seen before]***

`rm file_name`	Deteles file	
`rm dir_name`	→ error message	***"rm: cannot remove directory"***
`rmdir dir_name`	Deletes empty directory	***(not empty →error message)***
`rm -rf file_or_dir_name`	Deletes file or directory (incl. content)	

Redirection of input and output/error data streams

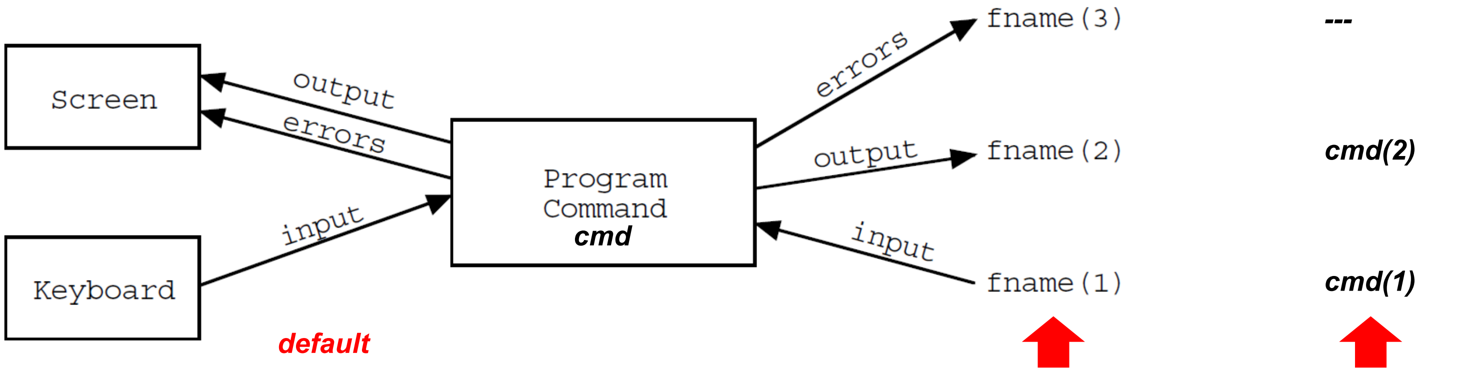
- All UNIX commands have one input and two output **standard channels** (which they may use or not) in addition to possibly reading or/and writing files

  → The **standard input** is where it reads data (default = keyboard)

  → The **standard output** is where it writes data (default = screen [*i.e.* terminal window])

  → The **standard error** is where it writes error messages (default = screen [*i.e.* terminal window])

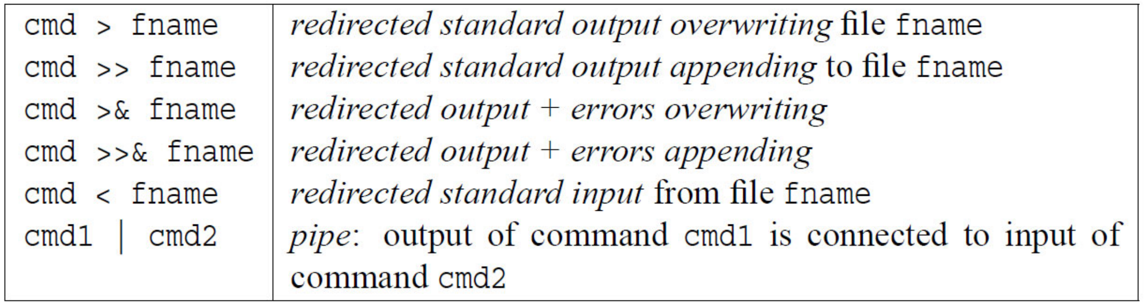
- It is possible to change the above defaults and **redirect** the channels either from/to a file or from/to another UNIX command



redirected from/to file

redirected from/to other command

- Redirection commands



`cmd > fname`	*redirected standard output overwriting* file fname
`cmd >> fname`	*redirected standard output appending* to file fname
`cmd >& fname`	*redirected output + errors overwriting*
`cmd >>& fname`	*redirected output + errors appending*
`cmd < fname`	*redirected standard input* from file fname
`cmd1 | cmd2`	*pipe*: output of command cmd1 is connected to input of command cmd2

“ | ” is called a “pipe”

Redirection of input and output/error data streams

- **Concatenating** command

`cat`	Copy standard input to standard output (default: input = keyboard till <CTRL-d>; output = screen)
`cat < file_name`	Copy file content to standard output (*via* input redirection)
`cat file_name`	Copy file content to standard output (name as argument)
`cat file_name_1 file_name_2`	Concatenate file contents to standard output
`cat > file_name`	Copy keyboard input (till <CTRL-d>) to file

Alone: fairly silly !

- ***Question:*** what do the following lines do ? Can we simplify them ?

```
cat | cmd                          cat file | cmd
cmd | cat                          cat < file1 > file2
cmd1 | cat | cmd2                  cat < file1 > file2; rm file1
```

- ***Question:*** how to concatenate files `a` and `b` into file `c` ?

On-line help, processes control, remote login

- One-line **help** (manual)

→ For information on a given command `cmd` (about its function, available options...), just type

```
man cmd
```

(use <SPACE> to go down and <q> to quit the page)

This should be a UNIX-reflex when you are not sure what a command does or want to modulate its behaviour

- **Process** control

→ Whenever a command is being executed by the UNIX system, this corresponds to a **UNIX process**

→ Each UNIX process has a **process identification number** (PID)

→ Each UNIX process has a **user identification number** (UID)

A UNIX system typically runs thousands of processes in parallel!

→ For **information** on processes, just type

```
ps -efj
```

(generates a list of all running processes; see "man ps" for the meaning of the options)

→ Also nice to see processes

```
top
```

(use <q> to quit)

→ To **interrupt** the execution of a **command-line** process (*i.e.* running in your window), type `<CTRL-c>`

Or to make it sleep: <CTRL-z>

→ To **interrupt** the execution of **any** process (incl. running in the background)

```
kill PID
```

(kills the process with specified PID – gently !)

```
kill -9 PID
```

(kills the process with specified PID – mercilessly !)

- Remote **login**

→ To login on another computer with

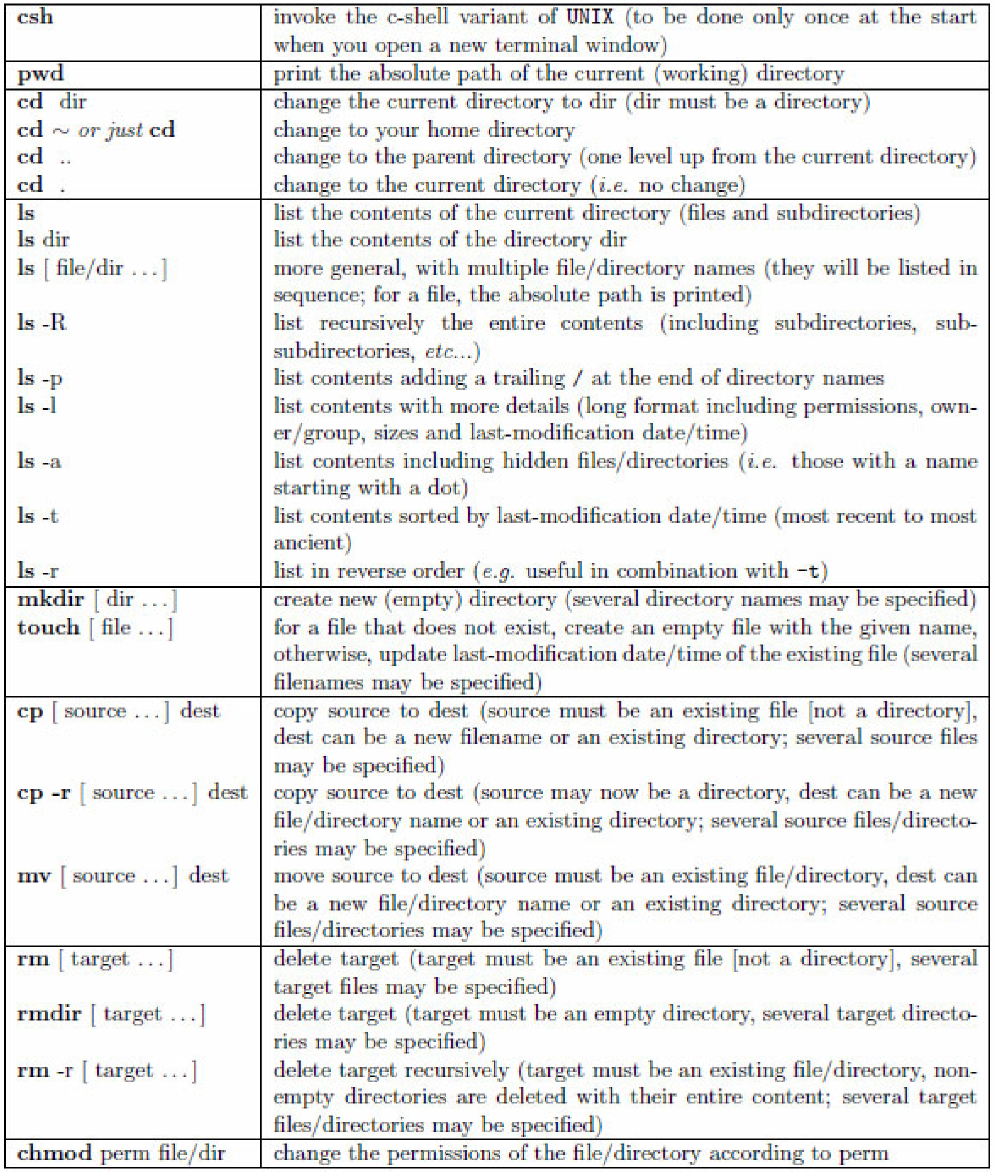
```
ssh hostname
```

(where hostname is the name of the remote computer)

Minimal set of commands

@016

- Ex1, Table 1.1



csh	invoke the c-shell variant of UNIX (to be done only once at the start when you open a new terminal window)
pwd	print the absolute path of the current (working) directory
cd dir	change the current directory to dir (dir must be a directory)
cd ~ *or just* **cd**	change to your home directory
cd ..	change to the parent directory (one level up from the current directory)
cd .	change to the current directory (*i.e.* no change)
ls	list the contents of the current directory (files and subdirectories)
ls dir	list the contents of the directory dir
ls [file/dir . . .]	more general, with multiple file/directory names (they will be listed in sequence; for a file, the absolute path is printed)
ls -R	list recursively the entire contents (including subdirectories, sub-subdirectories, *etc.*..)
ls -p	list contents adding a trailing / at the end of directory names
ls -l	list contents with more details (long format including permissions, owner/group, sizes and last-modification date/time)
ls -a	list contents including hidden files/directories (*i.e.* those with a name starting with a dot)
ls -t	list contents sorted by last-modification date/time (most recent to most ancient)
ls -r	list in reverse order (*e.g.* useful in combination with `-t`)
mkdir [dir . . .]	create new (empty) directory (several directory names may be specified)
touch [file . . .]	for a file that does not exist, create an empty file with the given name, otherwise, update last-modification date/time of the existing file (several filenames may be specified)
cp [source . . .] dest	copy source to dest (source must be an existing file [not a directory], dest can be a new filename or an existing directory; several source files may be specified)
cp -r [source . . .] dest	copy source to dest (source may now be a directory, dest can be a new file/directory name or an existing directory; several source files/directories may be specified)
mv [source . . .] dest	move source to dest (source must be an existing file/directory, dest can be a new file/directory name or an existing directory; several source files/directories may be specified)
rm [target . . .]	delete target (target must be an existing file [not a directory], several target files may be specified)
rmdir [target . . .]	delete target (target must be an empty directory, several target directories may be specified)
rm -r [target . . .]	delete target recursively (target must be an existing file/directory, non-empty directories are deleted with their entire content; several target files/directories may be specified)
chmod perm file/dir	change the permissions of the file/directory according to perm

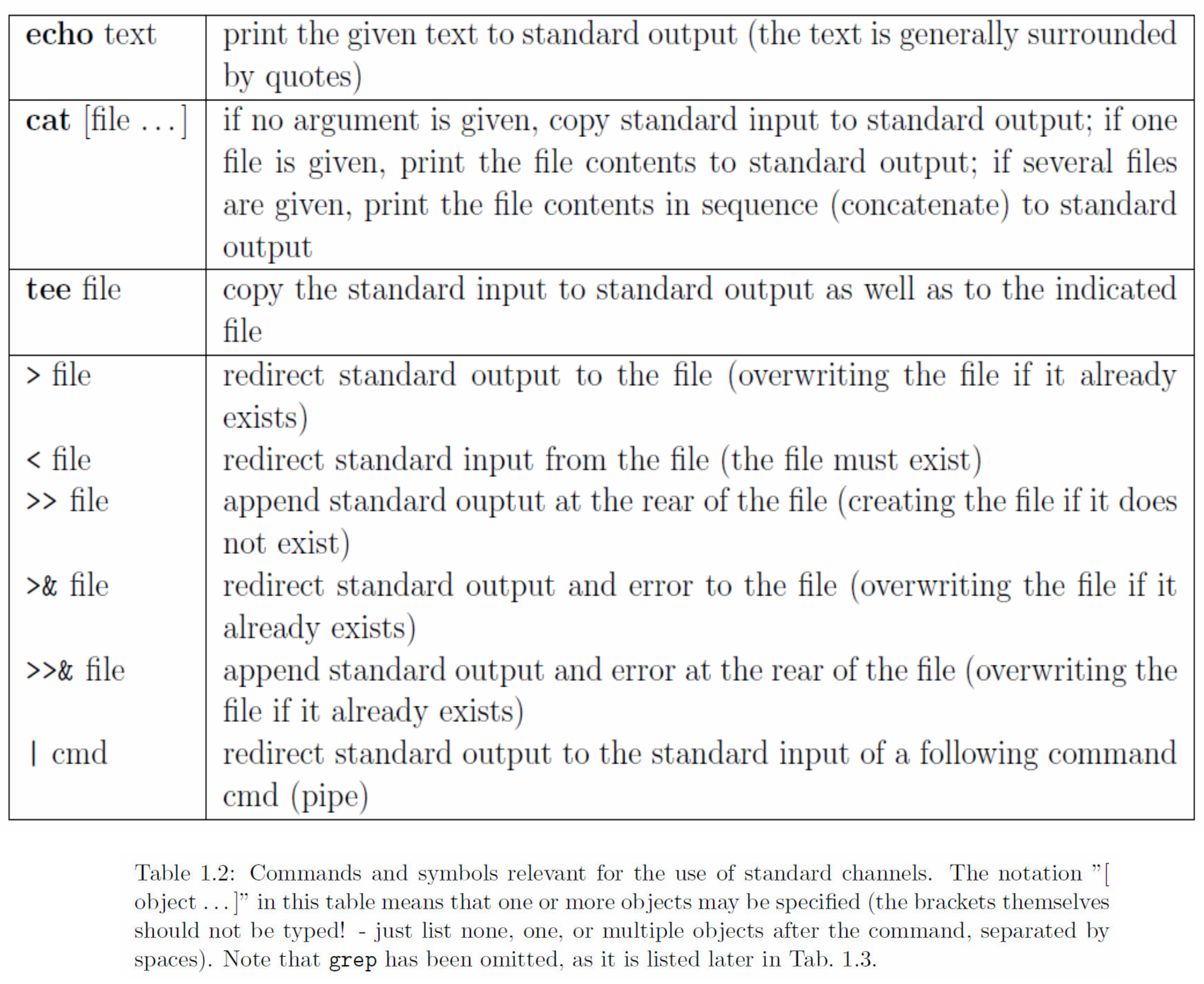
Table 1.1: Basic UNIX commands for filesystem navigation and file operations. The notation "[object . . .]" in this table means that one or more objects may be specified (the brackets themselves should not be typed! - just list none, one, or multiple objects after the command, separated by spaces). For all the commands, the files/directories can be always specified either by absolute or by relative paths.

This list is really the minimal survival kit – read a good UNIX book on your own, explore and learn more commands, and you will see the real power of this operating system !!!

Try them out at the exercise sessions (I will assume you know them at the exam)

Minimal set of commands

- Ex1, Table 1.2



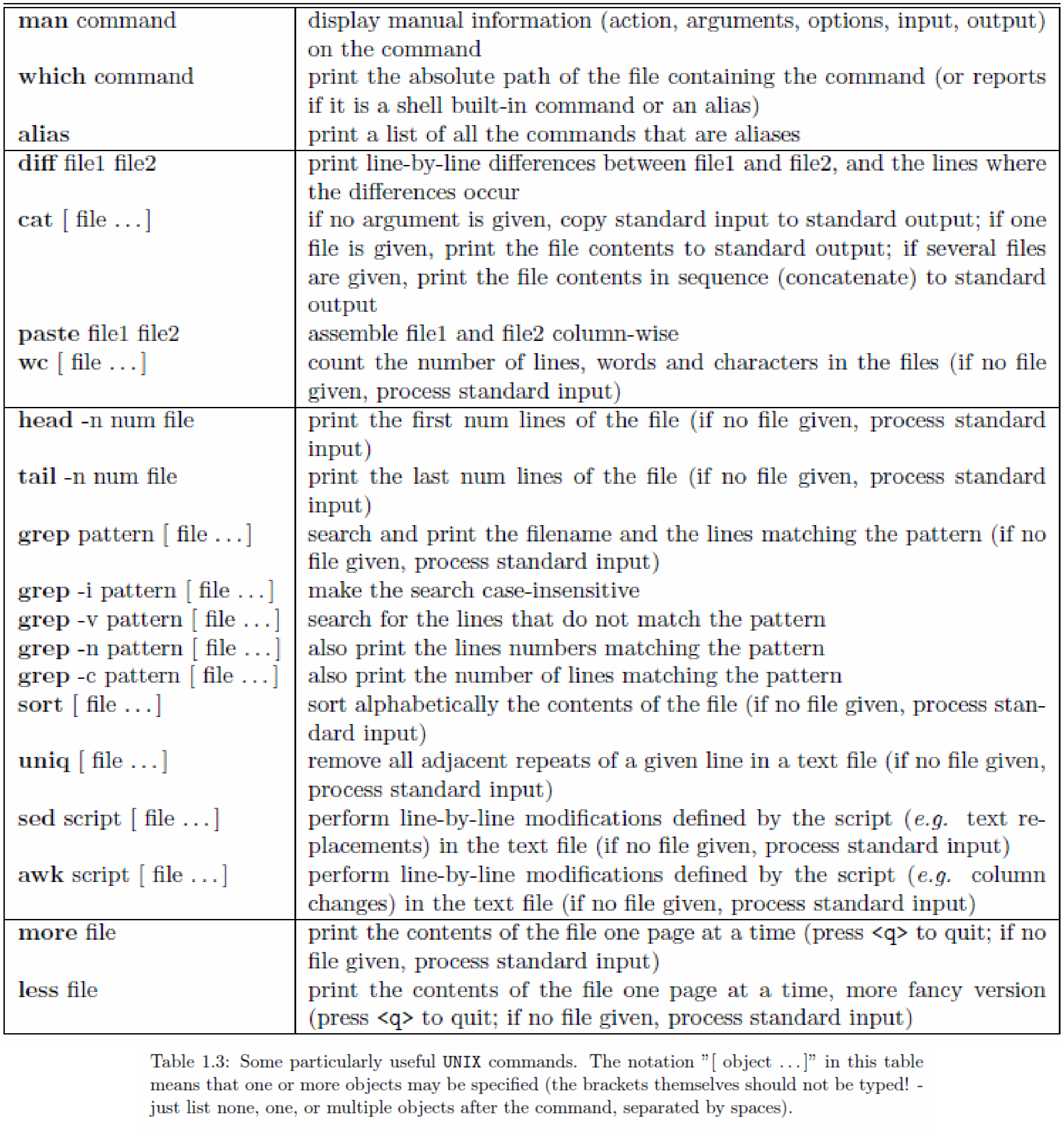
echo text	print the given text to standard output (the text is generally surrounded by quotes)
cat [file . . .]	if no argument is given, copy standard input to standard output; if one file is given, print the file contents to standard output; if several files are given, print the file contents in sequence (concatenate) to standard output
tee file	copy the standard input to standard output as well as to the indicated file
> file	redirect standard output to the file (overwriting the file if it already exists)
< file	redirect standard input from the file (the file must exist)
>> file	append standard ouptut at the rear of the file (creating the file if it does not exist)
>& file	redirect standard output and error to the file (overwriting the file if it already exists)
>>& file	append standard output and error at the rear of the file (overwriting the file if it already exists)
| cmd	redirect standard output to the standard input of a following command cmd (pipe)

Table 1.2: Commands and symbols relevant for the use of standard channels. The notation "[object . . .]" in this table means that one or more objects may be specified (the brackets themselves should not be typed! - just list none, one, or multiple objects after the command, separated by spaces). Note that `grep` has been omitted, as it is listed later in Tab. 1.3.

Try them out at the exercise sessions (I will assume you know them at the exam)

Minimal set of commands

- Ex1, Table 1.3



man command	display manual information (action, arguments, options, input, output) on the command
which command	print the absolute path of the file containing the command (or reports if it is a shell built-in command or an alias)
alias	print a list of all the commands that are aliases
diff file1 file2	print line-by-line differences between file1 and file2, and the lines where the differences occur
cat [file . . .]	if no argument is given, copy standard input to standard output; if one file is given, print the file contents to standard output; if several files are given, print the file contents in sequence (concatenate) to standard output
paste file1 file2	assemble file1 and file2 column-wise
wc [file . . .]	count the number of lines, words and characters in the files (if no file given, process standard input)
head -n num file	print the first num lines of the file (if no file given, process standard input)
tail -n num file	print the last num lines of the file (if no file given, process standard input)
grep pattern [file . . .]	search and print the filename and the lines matching the pattern (if no file given, process standard input)
grep -i pattern [file . . .]	make the search case-insensitive
grep -v pattern [file . . .]	search for the lines that do not match the pattern
grep -n pattern [file . . .]	also print the lines numbers matching the pattern
grep -c pattern [file . . .]	also print the number of lines matching the pattern
sort [file . . .]	sort alphabetically the contents of the file (if no file given, process standard input)
uniq [file . . .]	remove all adjacent repeats of a given line in a text file (if no file given, process standard input)
sed script [file . . .]	perform line-by-line modifications defined by the script (*e.g.* text replacements) in the text file (if no file given, process standard input)
awk script [file . . .]	perform line-by-line modifications defined by the script (*e.g.* column changes) in the text file (if no file given, process standard input)
more file	print the contents of the file one page at a time (press <q> to quit; if no file given, process standard input)
less file	print the contents of the file one page at a time, more fancy version (press <q> to quit; if no file given, process standard input)

Table 1.3: Some particularly useful UNIX commands. The notation "[object . . .]" in this table means that one or more objects may be specified (the brackets themselves should not be typed! - just list none, one, or multiple objects after the command, separated by spaces).

Try them out at the exercise sessions (I will assume you know them at the exam)

The power of UNIX scripting

- A **script** is a succession of UNIX commands in a file, that can be run as a single command

→ Example: script to rename all the .JPG files in a directory to .jpg

```
pic1.JPG           pic1.jpg
...          ➡     ...
pic999.JPG         pic999.jpg
```

→ Just type *(note: don’t type the ‘%’, it represents the prompt !)*

```
% cat > my_command                                  make the script file
#!/bin/csh                                          this is a C-shell script (name of interpreter)
foreach i (`ls *.JPG`)                              loop over .JPG files in directory
  mv $i `echo $i | sed s/[.]JPG$/.jpg/g`            change name (ending .JPG to .jpg)
end                                                 end loop
<CTRL-d>                                            end of making script file
% chmod u+x my_command                              give execute permission
% ./my_command                                      run script...
```

→ And it is all...

→ ... now try to do the same with a window-based (*i.e.* click & drag & drop & copy & paste) front-end...

The power of UNIX scripting

→ Example: script to know how many students there are in each exercise group

→ My file infoI_HS16.txt (205 entries)

```
Abegg           Manon           16-917-080  CHAB-ceng A
Akman           Erol            15-933-740  CHAB-chem A
Asgari          Farahani        16-916-157  CHAB-inbp D
Azizbaig        Mohajer         16-944-399  CHAB-chem A
Baeriswyl       Viviane         16-921-058  CHAB-chem A
Balbi           Petra           16-940-843  CHAB-inbp B
Bangerter       Jana            16-933-970  CHAB-inbp A
[...]
Wolf            Robin           16-950-008  CHAB-chem E
Zamboni         Lara            16-923-708  CHAB-inbp A
Zech            Patrick         16-916-900  CHAB-ceng E
Zenuni          Fatjona         16-936-734  CHAB-chem E
Ziegler         Lars            15-920-960  CHAB-ceng E
Zimmerli        Can             16-922-510  CHAB-chem E
Zimmermann      Sandra          16-935-603  CHAB-ceng E
Zuercher        Jerome          16-937-823  CHAB-inbp C
```

→ Script (or command line)

```
cat infoI_HS16.txt  |  awk ‘{print $NF}’  |  sort  |  uniq -c
```

Print last field of each line

Sort the list of A, B, C, D and E's

Print the occurrence of all repeating lines

→ Output

```
50 A
39 B
50 C
33 D
33 E
```

And it is even correct !
(groups D,E are for introverts;
groups A,C are for extraverts;
group B is for normal people...)

UNIX: arguments, input, output, ...

- I am getting ***kind of confused*** about different ways to ***copy a file***

→ What’s the difference between all this ???

```
cp file1 file2

cp file1 > file2

cp < file1 > file2

cp file1 | file2


cat file1 file2

cat file1 > file2

cat < file1 > file2

cat file1 | file2
```

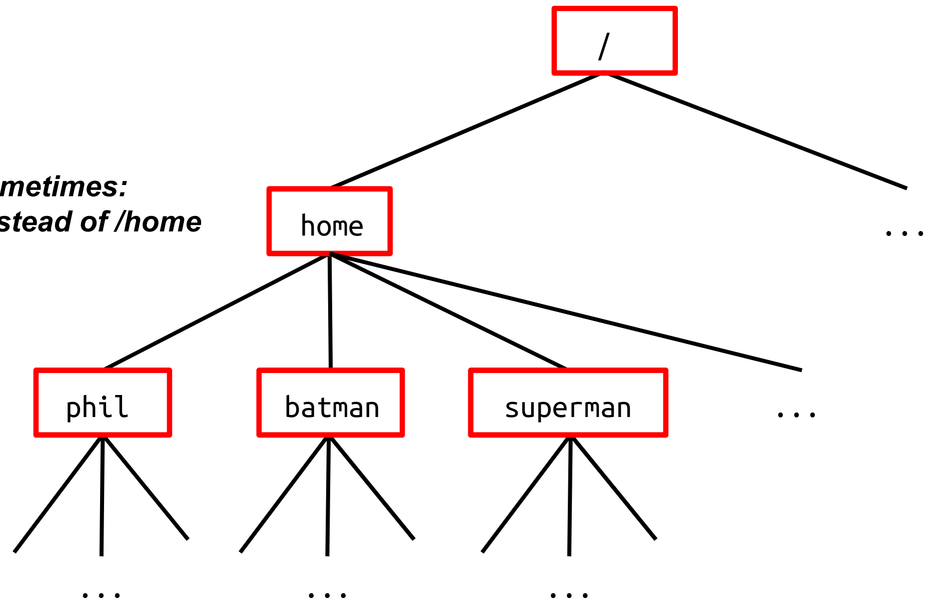
→ Other examples

```
cp file1 file2 > file3
cp file1 file2 >& file3
cat file1 file2 > file3
cat file2 >> file1
```

HS17→22

What is the difference between “~”, “~phil” and “~batman”

sometimes:
/user instead of /home



In this tree, <u>relative</u> names are given in the boxes

🟥	***directories***
🟩	***plain (data) files***

→ I am phil	~	***is the same as***	`~phil`	➡	`/home/phil`
		`~batman`		➡	`/home/batman`
→ I am batman	~	***is the same as***	`~batman`	➡	`/home/batman`
		`~phil`		➡	`/home/phil`

What does the "-f" mean in "rm -rf"

- Doubts about action/options/arguments of a UNIX command? --- look up the **manual entry**!

```
man rm
```

```
RM(1)                                User Commands                                RM(1)

NAME
       rm - remove files or directories

SYNOPSIS
       rm [OPTION]... FILE...

DESCRIPTION
       This manual page documents the GNU version of rm.  rm removes each specified file.  By default, it does not remove directories.

       If the -I or --interactive=once option is given, and there are more than three files or the -r, -R, or --recursive are given, then rm prompts the user for
       whether to proceed with the entire operation.  If the response is not affirmative, the entire command is aborted.

       Otherwise, if a file is unwritable, standard input is a terminal, and the -f or --force option is not given, or the -i or --interactive=always  option  is
       given, rm prompts the user for whether to remove the file.  If the response is not affirmative, the file is skipped.

OPTIONS
       Remove (unlink) the FILE(s).

  -f, --force
              ignore nonexistent files and arguments, never prompt

[...]

  -r, -R, --recursive
              remove directories and their contents recursively

[...]
```

Means: you don't want to be bothered by details & warnings – just go ahead and DELETE EVERYTHING!

HS17→22

Line feed, newline and carriage return

- If you edit your text files for the InfoI exercises on a Windows/Mac PC

  → there is a risk that you encounter **weird errors** when you run a text script or compile a text program on our UNIX computers

- This is because Windows/Mac vs UNIX encode **end-of-lines** differently

  → UNIX uses the special character

  `‘\n’` → *line feed (LF; ASCII code 10)*

  → Mac/Windows often use two special characters in sequence `‘\r’ then ‘\n’`

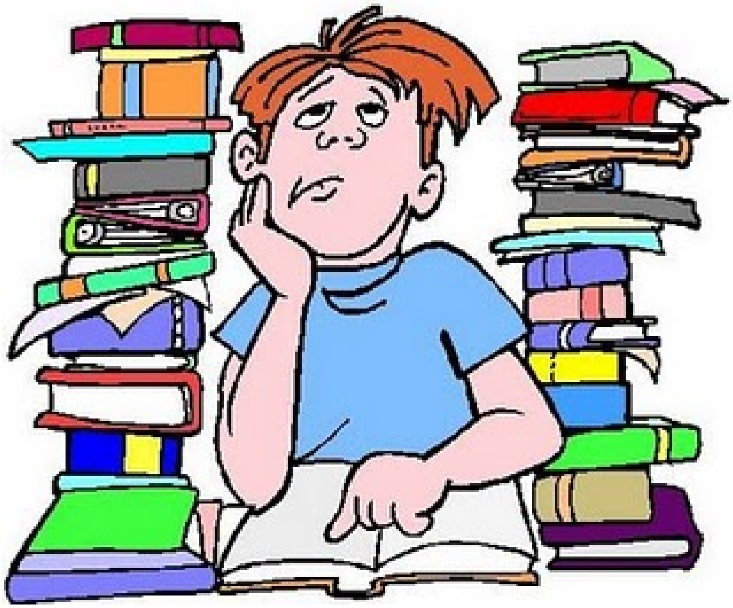
  `‘\r’` → *carriage return (CR; ASCII code 13)*

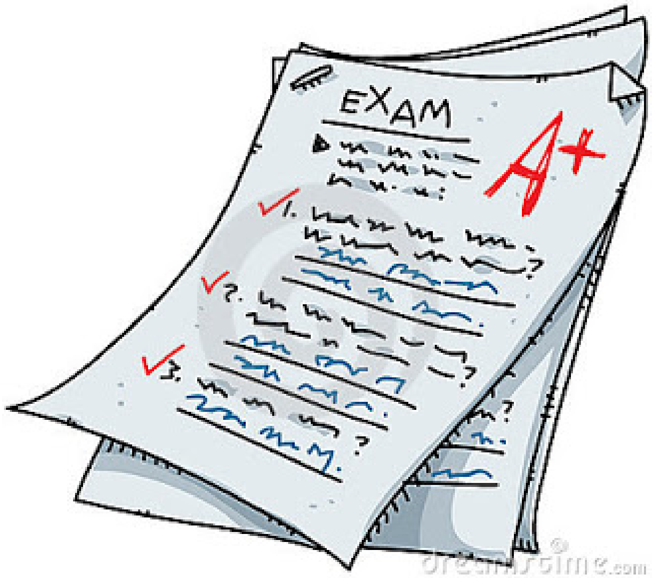
- If it happens to you, just use (on our UNIX computers)

```
cat messed_up.txt | sed ‘s/\r//g’ > works_with_unix.txt
```

  → This will remove the spurious `‘\r’`

Typical exam questions

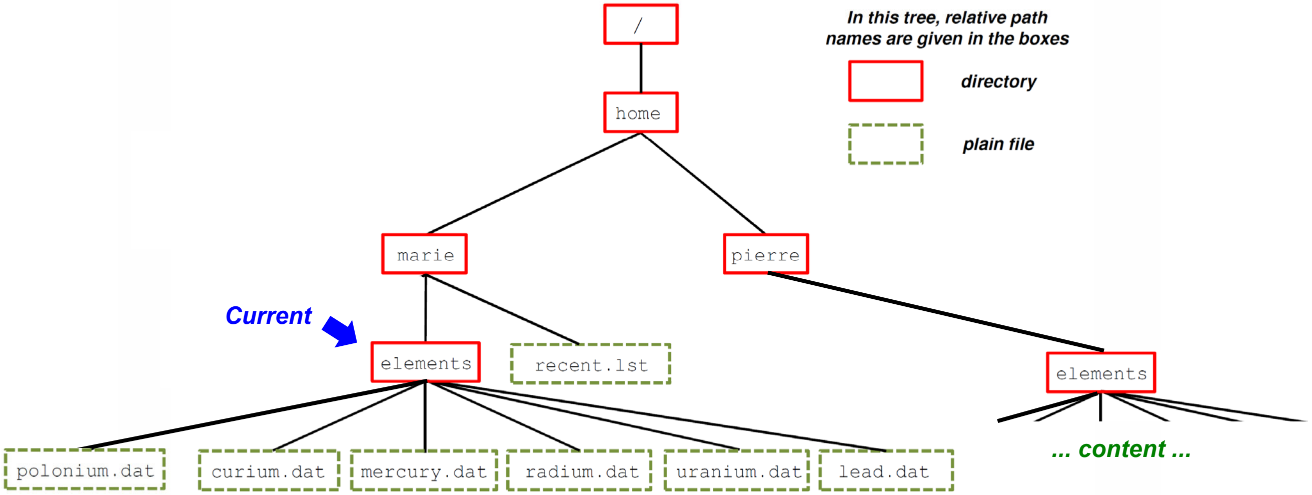




Typical exam questions

- ***S2017.1***

Answer the questions below concerning the UNIX operating system, considering the following directory tree



The root directory is ”/”. Your home directory is ”/home/marie”. Your current (working) directory is also ”/home/marie”. Explain the **effects** of the following UNIX commands **executed in sequence** (it is assumed that you have **all permissions**). When a command writes something to the screen or to a file, you have to **specify exactly** what is written.

a.
```
mv polonium.dat elements
```

b.
```
cp -r elements ~pierre
```

c.
```
cd elements
```

d.
```
pwd
```

e.
```
ls *m.* | sort >> ../recent.lst
```

f.
```
cat < radium.dat | more
```

g.
```
chmod 765 ~/*
```