Directories and files

Concepts
- Files are organised in a directory tree/hierarchy
- Everything is a file (e.g. keyboard, printers, ...)
- Each process has access to the files *stdin* (input), *stdout* (buffered output), *stderr* (unbuffered output)
- Each process operates in a *working directory*
- Each user has a *home directory*

Paths
Path = Identifier for the location of file/directory
- Paths consists of a parent directory list + file/directory
- Files and directories are separated by a '/'
- Directory paths may contain a trailing '/'

Absolute path = Full location (first character = '/')
Relative path = Relative location (first character $\neq$ '/')

`.`	path to the directory itself
`..`	path to the parent directory
`/usr/bin/ls`	example for an absolute file path
`/home/foo/`	example for an absolute directory path
`./a.out`	example for a relative file path

File system hierarchy

`/`	Root directory
`/bin`	Essential command executables
`/dev`	Device files
`/etc`	System-wide configuration files
`/opt`	Manually added software
`/sbin`	Essential administrative executables
`/tmp`	Temporary files
`/usr`	System resources for users
`/usr/bin`	Command executables
`/usr/local`	Site-local data
`/usr/sbin`	Administrative executables
`/var`	Variable files

man `hier` (or **man** `file-hierarchy` on recent Linux distributions) to get a more detailed overview

Terminal (emulator)

Text terminal = Computer interface for text entry/display
Terminal emulator = Application that emulates a *text terminal* in a graphical environment
Examples for terminal emulators: `xterm`, `urxvt`, `guake`

Opening a terminal

Unity/GNOME	`Ctrl`+`Alt`+`T`
Mac OS	`Cmd`+`⎵` → "terminal" → `↵`
Bash on Windows	`Win`+`R` → "bash" → `↵`

Shell

Unix shell = User interface that accepts commands to operate a computer
man `intro` to get an introduction into basic shell usage

Examples for shell programs: `sh`, `bash`, `zsh`, `fish`, `ksh`

Prompt
Prompt = Text sequence that precedes each line that prompts the user to enter a command

`[foo@bar /var/www]$` example prompt in bash
$\Rightarrow$ *user* `foo` is operating in the *working directory* `/var/www` at the computer with the *host name* `bar`

Line editing

`Ctrl`+`A`/`E`	Go to beginning/end of the line
`Ctrl`+`U`/`K`	Erase to beginning/end of the line
`Alt`+`B`/`F`	Go one word forward/backward in line
`Ctrl`+`W`	Erase to beginning of current word
`Ctrl`+`C`	Cancel the current command line

Metacharacters
The following characters have special meaning and sometimes they can't be used directly as arguments/words:
`| & ; < > ( ) $ ` \" ' * ? [# ~ = % ! { }`
`⎵` `⇆` `↵`
Their special meaning can be disabled:

`\`	preserves the literal value of the following character
`' '`	preserves the literal values of enquoted characters
`" "`	like `' '` but characters `` ` `` `$` `\` retain their meaning

Expansions

`~`	*home directory* of the current user
`*`	matches any character sequence
`?`	matches a single character
`[...]`	matches a character enclosed by the braces
`${var}`	value of the environment variable *var*
`$(cmd)`	output of *cmd*
`$((expr))`	result of the mathematical expression *expr*

Shell utilities

apropos text	searches the manual pages for *text*
cat file	prints the contents of *file*
cd dir	changes the *working directory* to *dir*
chmod prm file	changes permissions of *file* to *prm*
cp src dst	copies the file/directory *src* to *dst*
echo text	prints *text*
file file	determines the file type of *file*
find dir expr	finds files in *dir* that match *expr*
grep expr file	searches for pattern *expr* in *file*
ls dir	list the entries in the directory *dir*
man cmd	displays the manual for *cmd*
mkdir dir	creates the directory *dir*
mv src dst	moves/renames *src* to *dst*
pwd	prints the current *working directory*
rm file	removes the file *file*
sort	sorts lines of text from input
touch file	creates the empty file *file*

Input output redirection

cmd1 `|` *cmd2*	runs *cmd1* and *cmd2* and redirects the output of *cmd1* to the input of *cmd2*
cmd `>` file	runs *cmd* and redirects output to *file*, content of *file* is overwritten
cmd `>>` file	runs *cmd* and redirects output to *file*, the output is appended to *file*
cmd `<` file	runs *cmd* and redirects *file* to its input

Job control
Job = Shell command and its associated process(es)
- Each job has a job id and corresponding process ids
- Jobs can run in the foreground or in the background
- The execution of a job can be temporarily suspended

cmd `&`	starts *cmd* as background job (id is printed)
fg %job	puts the job *job* in foreground
bg %job	continues suspended job *job* in background
ps	prints the process ids of all active jobs
kill pid	terminates a process with the process id *pid*
`Ctrl`+`S`	suspends active job
`Ctrl`+`Q`	continues active job
`Ctrl`+`Z`	puts active job to background and suspends it
`Ctrl`+`C`	aborts the active job (most of the times)
`Ctrl`+`D`	sends an EOF character