`shell` & LaTeX
A crash course

Outline

Outline

Why knowing shell matters

- Command line interface, command line, command shell, or just “shell”

Why knowing shell matters

- Command line interface, command line, command shell, or just “shell”
- *Ubiquitous* in programming

Why knowing shell matters

- Command line interface, command line, command shell, or just “shell”
- *Ubiquitous* in programming
- It’s a flexible *set of basic tools* for commonplace operations

Why knowing shell matters

- Command line interface, command line, command shell, or just “shell”
- *Ubiquitous* in programming
- It’s a flexible *set of basic tools* for commonplace operations
- Knowing how to use them will speed up most menial tasks

Why knowing TEX/LATEXmatters

- TEX is a typesetting system from which LATEX is derived

Why knowing TEX/LATEXmatters

- TEX is a typesetting system from which LATEX is derived
- LATEX is extremely widespread, especially in “hard-science” academia

Why knowing TEX/LATEXmatters

- TEX is a typesetting system from which LATEX is derived
- LATEX is extremely widespread, especially in “hard-science” academia
- It’s extremely versatile

Outline

`shell`

Command line instructions

- What is command-line ?

`shell`

Command line instructions

- What is command-line ? Everything that can be typed and run on terminal.

`shell`

Command line instructions

- What is command-line ? Everything that can be typed and run on terminal.
- Historically, terminal was the only available user interface for computers.

`shell`

Command line instructions

- What is command-line ? Everything that can be typed and run on terminal.
- Historically, terminal was the only available user interface for computers.
- We'll be focusing on POSIX OS shell, which are more widespread in academia and engineering contexts.

shell

Command line instructions

- What is command-line ? Everything that can be typed and run on terminal.
- Historically, terminal was the only available user interface for computers.
- We'll be focusing on POSIX OS shell, which are more widespread in academia and engineering contexts.
- There exist many shells : `bash`, `zsh`, `sh`...

`shell`

Command line instructions

- What is command-line ? Everything that can be typed and run on terminal.
- Historically, terminal was the only available user interface for computers.
- We'll be focusing on POSIX OS shell, which are more widespread in academia and engineering contexts.
- There exist many shells : **`bash`**, **`zsh`**, **`sh`**...
- Open up terminal :
  - Windows 10 : Install and run Windows Subsystem for Linux (WSL). Windows-native command line ("Powershell") also exist, but won't be covered here.
  - Mac : **`Cmd+N`** ; you can also find through the Finder (**`Cmd+Space`**, then type in "terminal")
  - Most Linux-based desktop OS have a dedicated shortcut, e.g. Ubuntu : **`Ctrl+Alt+T`**

`shell`

Command line instructions

- What is command-line ? Everything that can be typed and run on terminal.
- Historically, terminal was the only available user interface for computers.
- We'll be focusing on POSIX OS shell, which are more widespread in academia and engineering contexts.
- There exist many shells : **`bash`**, **`zsh`**, **`sh`**...
- Open up terminal :
  - Windows 10 : Install and run Windows Subsystem for Linux (WSL). Windows-native command line ("Powershell") also exist, but won't be covered here.
  - Mac : **`Cmd+N`** ; you can also find through the Finder (**`Cmd+Space`**, then type in "terminal")
  - Most Linux-based desktop OS have a dedicated shortcut, e.g. Ubuntu : **`Ctrl+Alt+T`**
- Using the terminal effectively is something you learn through *practice*, but Google and `stackoverflow.com` can help you solve most problems.

`shell`: cheat sheet

Don't Panic

- Open a terminal with `Ctrl+Alt+T`, close it with `Ctrl+D`
- `echo ARGS`: print the arguments `ARGS` on screen
- `man COMMAND`: open the **man**ual entry for a given `command`
- `history`: manage or consult recent input commands. `Ctrl+Shift+R`: lookup in previous input commands history

Paths

- current working directory: the "folder" where you currently are in the filesystem. Represented in terminal as `./`
- subdirectory: a directory `SUBDIR` contained in another directory `DIR` is a subdirectory of `DIR`. Represented as `DIR/SUBDIR` in terminal.
- parent directory: the directory "one above" a given directory, i.e., the directory that contains a given directory. Represented in terminal as `../`; thus the parent directory of the directory `DIR/` is `DIR/../`
- absolute and relative path:
  - An absolute path starts from the "root" (represented as `/` in terminal), the "topmost" directory that is the direct or indirect parent of all other directories. All paths that start with `/` are absolute.
  - A relative path starts from the current working directory.
- home: the "default place" where the terminal is started, corresponding to the absolute path `/home/USER`, where USER is the username as returned by the command `whoami`. This absolute path is represented as `~` in the terminal.

Directory Navigation

- `pwd`: **p**rint the absolute path of the current **w**orking **d**irectory
- `cd DIR`: **c**hange current working **d**irectory to `DIR`
- `ls DIR`: **l**i**s**t the content of directory `DIR`; list subdirectories recursively using `ls -r`, more details using `ls -l`.
- `tree DIR`: display **tree** structure of directory `DIR`
- `wc FILE_1 ... FILE_N`: lines, **w**ords and character **c**ounts for each file `FILE_1`, ..., `FILE_N`; restrict to line counts with `wc -l FILE`, word counts with `wc -w FILE`, character counts with `wc -c FILE`.
- `du DIR` and `df`: show **d**isk **u**sage of directory `DIR` and **d**isk **f**ree memory; human-friendly format with option `-h`
- `find DIR ...`: **find** files/directories under `DIR`, matching criteria, optionally execute actions. Common criteria: `-type f` for files, `-type d` for directories, `-name PATTERN` for names matching `PATTERN`. Common actions: `-delete` to delete, `-quit` to stop on first hit, `-exec COMMAND` to execute `COMMAND`
- `mkdir DIR`: **m**a**k**e **dir**ectory, yield error if directory or file of the same name exists/ parent directory do not exist. Create parents as needed and do not error if `DIR` exists with `mkdir -p DIR`.
- `mv SOURCE DEST`: **m**o**v**e file `SOURCE` to new location `DEST`; move directory contents recursively with `mv -r SOURCE DEST`
- `cp SOURCE DEST`: **c**o**p**y file `SOURCE` to new location `DEST`; copy directory contents recursively with `cp -r SOURCE DEST`

File Display

- `cat FILE_1 ... FILE_N`: con**cat**enate and print contents of files
- `head FILE` and `tail FILE`: show beginning or end of file `FILE` respectively. Show the first/last 42 lines (or any other number) of a file using option `-42`.
- `less FILE` and `more FILE`: interactively display file contents
- `diff FILE_1 FILE_2`: show **diff**erences between files `FILE_1` and `FILE_2`
- `basename FILE_PATH`: remove directories from path `FILE_PATH` and keep only the file name; to also strip suffix `EXT` use `basename -s EXT FILE_PATH`

File Manipulation

- `grep PATTERN`: **g**lobal **r**egular **e**xpression **p**rint: print matches of regexp `PATTERN` found in `FILE`; many relevant options exist: `-i` to ignore case, `-P` for PERL regexps, `-n` to print line number, `-o` to print only match, `-l` to print only the file name when a match is found...
- `awk PROGRAM FILE`: programming language for CSV-like files
- `sed INSTR FILE`: **s**tream **ed**itor for regexp-based substitutions and deletions
- `split FILE`: **split** a large file into smaller files; specify their size with `-l`
- `cut FILE`: trim files column-wise, specify column delimiters with `-d`, restrict to the 3rd to the 5th columns with `-f 3-5`
- `paste FILE`: merge files column-wise, specify column delimiters with `-d`
- `sort FILE`: **sort** file `FILE`; random sort using `-R`, keep only distinct (unique) lines with `-u`, specify output file with `-o`, reverse order with `-r`, merge (*but do not sort*) sorted files with `-m`
- `uniq FILE`: keep only **uniq**ue (distinct) adjacent lines in `FILE`, add a count number of unique lines with `-c`.
- `tar`, `zip`, `unzip`, `gzip` and `gunzip`: produce and extract file archives; for tar: extract using `tar -xvf ARCHIVE_NAME`, compress with `tar -cvf ARCHIVE_NAME ITEMS_TO_ARCHIVE`, apply gzip on top of tar with option `-z`.
- `rm FILE`: **re**m**o**ve and permanently destroy file `FILE`; to remove and destroy a directory with its contents, use `rm -r DIR`

Process Control

- `chmod FILE`: **ch**ange the **mod**e of access to a file `FILE`; in particular make it executable with `chmod +x FILE`
- `su IDENTITY -c COMMAND` and `sudo COMMAND`: identify yourself and execute `command` as **s**uper **u**ser (or as `IDENTITY` if provided); `su` without arguments opens a session as super user
- `source INSTRUCTIONS`: execute instructions listed in file `INSTRUCTIONS`; equivalent to `. INSTRUCTIONS`
- `ps`: show a **s**napshot of the current running **p**rocessess
- `kill PID`: **kill** or terminate a process identified with `PID`
- `top` and `htop`: display all linux processes
- `watch COMMAND`: repeat the same `command` indefinitely; specify interval between repetitions using `watch -n TIME COMMAND`

Remote File Access

- `wget URL`: **get** a document from the **w**eb, i.e., download from link `URL`
- `ssh HOST`: **s**ecure **sh**ell access to a remote server: generate an access key using `ssh-keygen`, have it accepted on the remote server, and then connect to the remote server using `ssh login@remote.server:port`
- `scp SOURCE DEST`: ssh **copy**, i.e., copy files from/to server to which you have ssh access
- `rsync SOURCE DEST`: **r**emote server **sync**hronization; make the contents of a remote and local directories equivalent

Basic syntax and operators

- Variables: declare a variable named `VAR` with a value of `FOO` using the syntax `VAR=FOO` (*without spaces*); refer to this variable elsewhere in the code using `${VAR}` or `$VAR`. Variable names are conventionally capitalized in `shell`.
- Loop control flow: loop over a list, and refer to each element with the variable `${ELEM}` using the syntax:

```
for ELEM in 1 2 3 4; do
    echo ${ELEM};
done;
```

- Conditional control flow: execute a command based on whether a test is true with the following syntax:

```
VAR=42;
if [ $VAR -gt 41 ]; then
    echo "the test went ok!";
else
    echo "alack! 'tis failed!";
fi;
```

- Piping and `xargs`: pass the output of one command as input to the next command using the syntax `command_1 | command_2`
  - the command `find . -type f -name '*.md' | grep '/data/'` will find all markdown files under the current directory and then prune the search results to files under a subdirectory called `data`
  - to use each line the first command's input as distinct e**x**ternal **arg**ument**s** of the second command, use `xargs`: the command `find data/ -type f -name '*.md' | xargs grep -li package` will list only markdown files under the directory `./data/` that contain the word "package" (ignoring case).
- Inputs and outputs:
  - To redirect the output of a `command` to a file `FILE_1.txt`, use the syntax `command > FILE_1.txt` ; if the file exists it will be overwritten *before the command is executed.* Therefore `cat FILE_1.txt > FILE_2.txt` is equivalent to `cp FILE_1.txt FILE_2.txt`
  - To append the output of a `command` to a file `FILE_1.txt`, use `command >> FILE_1.txt` ; if the file doesn't exist it will be created.
  - To use the contents of a file `FILE_1.txt` as input for a `command`, use the syntax `command < FILE_1.txt`
- Conditional execution: execute `command_2` only if the previous command `command_1` did not error with the syntax `command_1 && command_2`
- Background execution, suspension, termination, `fg` and `bg`:
  - Execute a `command` in the background (i.e., without blocking the terminal until its completion) with the syntax `command &`
  - Call background processes to the **f**ore**g**round with `fg`, e.g. to manually terminate them (using `Ctrl+C`) or suspend them (using `Ctrl+Z`)
  - Let a suspended process run in the **b**ack**g**round using `bg`
- Substitutions: variable names are replaced with corresponding values in double-quoted strings: `"here's an ${EXAMPLE}"`. A `command` can be replaced by its output using the syntax `$(command)`, e.g.:

```
for FILE in $(ls); do
    echo $(basename $FILE);
done;
```

- Arithmetics and `bc`: perform computations using the syntax `$((2 + 3))` or for floating point arithmetics `echo "2.1 / 3.2" | bc -l`

Outline

LaTeX

What it is

- TeX is a typesetting system & LaTeX implements a markup language on top of TeX. Relevant files have a `.tex` extension.
  - Typesetting systems provide (computer) instructions on how to display characters on screen or paper (i.e., how to set types).
  - Markup languages provide means of structuring documents.

LaTeX

What it is

- TeX is a typesetting system & LaTeX implements a markup language on top of TeX. Relevant files have a `.tex` extension.
    - Typesetting systems provide (computer) instructions on how to display characters on screen or paper (i.e., how to set types).
    - Markup languages provide means of structuring documents.

- TeX is especially useful for mathematical formulas

LaTeX

What it is

- TeX is a typesetting system & LaTeX implements a markup language on top of TeX. Relevant files have a `.tex` extension.
  - Typesetting systems provide (computer) instructions on how to display characters on screen or paper (i.e., how to set types).
  - Markup languages provide means of structuring documents.
- TeX is especially useful for mathematical formulas
- LaTeX provides many extensions, and can be used for very different projects : articles, posters, slides, theses, dissertations ...

LaTeX

What it is

- TeX is a typesetting system & LaTeX implements a markup language on top of TeX. Relevant files have a `.tex` extension.
  - Typesetting systems provide (computer) instructions on how to display characters on screen or paper (i.e., how to set types).
  - Markup languages provide means of structuring documents.
- TeX is especially useful for mathematical formulas
- LaTeX provides many extensions, and can be used for very different projects : articles, posters, slides, theses, dissertations ...
- LaTeX requires *practice* to properly master. The internet can and will help you when you encounter an issue.

LaTeX

What it is

- TeX is a typesetting system & LaTeX implements a markup language on top of TeX. Relevant files have a `.tex` extension.
    - Typesetting systems provide (computer) instructions on how to display characters on screen or paper (i.e., how to set types).
    - Markup languages provide means of structuring documents.
- TeX is especially useful for mathematical formulas
- LaTeX provides many extensions, and can be used for very different projects : articles, posters, slides, theses, dissertations ...
- LaTeX requires *practice* to properly master. The internet can and will help you when you encounter an issue.
- There exist many editors for LaTeX, try the online editor OverLeaf at `https://www.overleaf.com`, which also provides some tutorials.

LaTeX
Basics

- TeX commands start with a backslash `\`, e.g., **`\newline`** produces a new line

LaTeX

Basics

- TeX commands start with a backslash `\`, e.g., **`\newline`** produces a new line
  - arguments are between curly braces **`{}`** : **`\sqrt{2}`** yields $\sqrt{2}$

LaTeX

Basics

- TeX commands start with a backslash `\`, e.g., `\newline` produces a new line
  - arguments are between curly braces `{}` : `\sqrt{2}` yields $\sqrt{2}$
  - optional parameters are between square brackets : `\sqrt[p]{2}` yields $\sqrt[p]{2}$.

LaTeX

Basics

- TeX commands start with a backslash `\`, e.g., `\newline` produces a new line
  - arguments are between curly braces `{}` : `\sqrt{2}` yields $\sqrt{2}$
  - optional parameters are between square brackets : `\sqrt[p]{2}` yields $\sqrt[p]{2}$.

- Scopes can be defined using the curly braces `{}` : `{\large word}` yields $\large\text{word}$, limiting the effects of `\large` to the end of the scope.

LaTeX
Basics

- TeX commands start with a backslash \, e.g., **\\newline** produces a new line
  - arguments are between curly braces **{}** : **\\sqrt{2}** yields $\sqrt{2}$
  - optional parameters are between square brackets : **\\sqrt[p]{2}** yields $\sqrt[p]{2}$.
- Scopes can be defined using the curly braces **{}** : **{\\large word}** yields word, limiting the effects of **\\large** to the end of the scope.
- Inline comments can be written with the **%** symbol : characters following a **%** symbol until the end of the line will be ignored in the rendering process.

LaTeX

Basics

- TeX commands start with a backslash `\`, e.g., `\newline` produces a new line
  - arguments are between curly braces `{}` : `\sqrt{2}` yields $\sqrt{2}$
  - optional parameters are between square brackets : `\sqrt[p]{2}` yields $\sqrt[p]{2}$.
- Scopes can be defined using the curly braces `{}` : `{\large word}` yields $\large\text{word}$, limiting the effects of `\large` to the end of the scope.
- Inline comments can be written with the `%` symbol : characters following a `%` symbol until the end of the line will be ignored in the rendering process.
- LaTeX structure elements, or ‘environments’, start with the command `\begin{env}` and end with the command `\end{env}`, with `env` the name for that environment

LaTeX

Basics

- TeX commands start with a backslash `\`, e.g., `\newline` produces a new line
  - arguments are between curly braces `{}` : `\sqrt{2}` yields $\sqrt{2}$
  - optional parameters are between square brackets : `\sqrt[p]{2}` yields $\sqrt[p]{2}$.

- Scopes can be defined using the curly braces `{}` : `{\large word}` yields word, limiting the effects of `\large` to the end of the scope.

- Inline comments can be written with the `%` symbol : characters following a `%` symbol until the end of the line will be ignored in the rendering process.

- LaTeX structure elements, or 'environments', start with the command `\begin{env}` and end with the command `\end{env}`, with `env` the name for that environment

- LaTeX files start by a document class declaration that states what type of document the current file is, e.g. `\documentclass{article}` for anything that looks like an article

- TeX commands start with a backslash `\`, e.g., `\newline` produces a new line
  - arguments are between curly braces `{}` : `\sqrt{2}` yields $\sqrt{2}$
  - optional parameters are between square brackets : `\sqrt[p]{2}` yields $\sqrt[p]{2}$.
- Scopes can be defined using the curly braces `{}` : `{\large word}` yields word, limiting the effects of `\large` to the end of the scope.
- Inline comments can be written with the `%` symbol : characters following a `%` symbol until the end of the line will be ignored in the rendering process.
- LaTeX structure elements, or ‘environments’, start with the command `\begin{env}` and end with the command `\end{env}`, with `env` the name for that environment
- LaTeX files start by a document class declaration that states what type of document the current file is, e.g. `\documentclass{article}` for anything that looks like an article
- The actual content of a LaTeX file is within a `document` environment ; i.e. written between a `\begin{document}` command and a `\end{document}` command.

LaTeX

Basics

- TeX commands start with a backslash \, e.g., `\newline` produces a new line
  - arguments are between curly braces `{}` : `\sqrt{2}` yields $\sqrt{2}$
  - optional parameters are between square brackets : `\sqrt[p]{2}` yields $\sqrt[p]{2}$.

- Scopes can be defined using the curly braces `{}` : `{\large word}` yields word, limiting the effects of `\large` to the end of the scope.

- Inline comments can be written with the `%` symbol : characters following a `%` symbol until the end of the line will be ignored in the rendering process.

- LaTeX structure elements, or 'environments', start with the command `\begin{env}` and end with the command `\end{env}`, with `env` the name for that environment

- LaTeX files start by a document class declaration that states what type of document the current file is, e.g. `\documentclass{article}` for anything that looks like an article

- The actual content of a LaTeX file is within a `document` environment ; i.e. written between a `\begin{document}` command and a `\end{document}` command.

- Instructions before the `document` environment are referred to as the "preamble" .
  In particular the `\usepackage{packagename}` loads existing extensions or "packages" that contain specific commands.

Commands for font series :

`\textit{words in italics}`	*words in italics*
`\textsl{words slanted}`	*words slanted*
`\textsc{words in smallcaps}`	WORDS IN SMALLCAPS
`\textbf{words in bold}`	**words in bold**
`\texttt{words in teletype}`	`words in teletype`
`\textsf{sans serif words}`	sans serif words
`\textrm{roman words}`	roman words
`\underline{underlined words}`	$\underline{\text{underlined words}}$
`words\textsuperscript{in superscript}`	$\text{words}^{\text{in superscript}}$
`words\textsubscript{in subscript}`	$\text{words}_{\text{in subscript}}$

Some are much more frequently used than others, e.g., `\textit` and `\textbf` are very common, `\textsf` and `\textrm` are rare.

Commands for changing the current font size :

`{\Huge text}`	text
`{\huge text}`	text
`{\LARGE text}`	text
`{\Large text}`	text
`{\large text}`	text
`{\normalsize text}`	text
`{\small text}`	text
`{\footnotesize text}`	text
`{\scriptsize text}`	text
`{\tiny text}`	text

Rarely directly used in practice, the font size is more often defined per environment, or tweaked using the `relsize` package, which provides the `\smaller` and `\larger` commands.

LaTeX

Basics of typesetting : math mode

TeX also provides a *math mode* for writing formulas :
anything between two dollar signs `$ ... $` is interpreted using the math mode syntax.

Basics of typesetting : math mode

TeX also provides a *math mode* for writing formulas :
anything between two dollar signs `$ ... $` is interpreted using the math mode syntax.

- carets `^` produce superscripts, underscores `_` produce subscripts
- scopes can be defined using the curly braces `{}`
- spacing is automatic ; to add some more spaces, use a tilde `~`, or the commands `\quad` or `\qquad`
- many functions are only available in math mode, like `\frac{a}{b}` (yields $\frac{a}{b}$) or `\sqrt{a}` (yields $\sqrt{a}$)
- individual letters are typeset as variables, which *may* differ from normal typesetting ; to display them as words, use a function such as `\text`, `\textit`, `\mathbf`...

Basics of typesetting : math mode

TeX also provides a *math mode* for writing formulas :
anything between two dollar signs `$ ... $` is interpreted using the math mode syntax.

- carets `^` produce superscripts, underscores `_` produce subscripts
- scopes can be defined using the curly braces `{}`
- spacing is automatic ; to add some more spaces, use a tilde `~`, or the commands `\quad` or `\qquad`
- many functions are only available in math mode, like `\frac{a}{b}` (yields $\frac{a}{b}$) or `\sqrt{a}` (yields $\sqrt{a}$)
- individual letters are typeset as variables, which *may* differ from normal typesetting ; to display them as words, use a function such as `\text`, `\textit`, `\mathbf`...
- In all :

```
$\mathbf{pdf}_{\chi^{-1}\chi}(z)=\frac{z^k}{(z^2+1)^{k+\frac{1}{2}}}
\cdot\frac{2^{3/2}~\Gamma(k+\frac{1}{2})}{\big(\Gamma(k/2)\big)^2}$
```

produces the following equation : $\mathbf{pdf}_{\chi^{-1}\chi}(z)=\frac{z^k}{(z^2+1)^{k+\frac{1}{2}}}\cdot\frac{2^{3/2}~\Gamma(k+\frac{1}{2})}{\big(\Gamma(k/2)\big)^2}$

LaTeX

Basics of typesetting : math mode

TeX also provides a *math mode* for writing formulas :
anything between two dollar signs `$ ... $` is interpreted using the math mode syntax.

- carets `^` produce superscripts, underscores `_` produce subscripts
- scopes can be defined using the curly braces `{}`
- spacing is automatic ; to add some more spaces, use a tilde `~`, or the commands `\quad` or `\qquad`
- many functions are only available in math mode, like `\frac{a}{b}` (yields $\frac{a}{b}$) or `\sqrt{a}` (yields $\sqrt{a}$)
- individual letters are typeset as variables, which *may* differ from normal typesetting ; to display them as words, use a function such as `\text`, `\textit`, `\mathbf`...
- In all :

  ```
  $\mathbf{pdf}_{\chi^{-1}\chi}(z)=\frac{z^k}{(z^2+1)^{k+\frac{1}{2}}}
  \cdot\frac{2^{3/2}~\Gamma(k+\frac{1}{2})}{\big(\Gamma(k/2)\big)^2}$
  ```

  produces the following equation : $\mathbf{pdf}_{\chi^{-1}\chi}(z)=\frac{z^k}{(z^2+1)^{k+\frac{1}{2}}}\cdot\frac{2^{3/2}~\Gamma(k+\frac{1}{2})}{\big(\Gamma(k/2)\big)^2}$

Online TeX math mode editor at `https://www.codecogs.com/latex/eqneditor.php`

LaTeX

Basics of document layout and structure

- LaTeX is meant to do away with most layout problems, therefore the exact position of elements is generally computed by the compiler (the PDF-rendering program) itself rather than decided by the user.
  - This is particularly the case of floating elements, such as figures and tables.
  - Relatedly, spacing between lines and paragraphs is also automated, i.e., you cannot easily skip multiple lines.
  - Some commands exist to override this behavior, e.g. `\vspace{1cm}` requires a one centimeter blank space before moving on to the next line
  - In general, the best practice is to let the compiler handle layout as much as possible.

Basics of document layout and structure

- LaTeX is meant to do away with most layout problems, therefore the exact position of elements is generally computed by the compiler (the PDF-rendering program) itself rather than decided by the user.
  - This is particularly the case of floating elements, such as figures and tables.
  - Relatedly, spacing between lines and paragraphs is also automated, i.e., you cannot easily skip multiple lines.
  - Some commands exist to override this behavior, e.g. `\vspace{1cm}` requires a one centimeter blank space before moving on to the next line
  - In general, the best practice is to let the compiler handle layout as much as possible.

- The initial `\documentclass` command determines the overall layout of the document.
  - Use `\documentclass{beamer}` for slides and posters.
  - Use `\documentclass{article}` for most other projects, including dissertations and reports.

LaTeX

Basics of document layout and structure

- LaTeX is meant to do away with most layout problems, therefore the exact position of elements is generally computed by the compiler (the PDF-rendering program) itself rather than decided by the user.
  - This is particularly the case of floating elements, such as figures and tables.
  - Relatedly, spacing between lines and paragraphs is also automated, i.e., you cannot easily skip multiple lines.
  - Some commands exist to override this behavior, e.g. `\vspace{1cm}` requires a one centimeter blank space before moving on to the next line
  - In general, the best practice is to let the compiler handle layout as much as possible.
- The initial `\documentclass` command determines the overall layout of the document.
  - Use `\documentclass{beamer}` for slides and posters.
  - Use `\documentclass{article}` for most other projects, including dissertations and reports.
- In `beamer` projects, individual slides can be created with the `frame` environment.
- In `article` projects, you can use the `\section{Section Title}` command to create a new section titled "Section Title".

LaTeX

Miscellaneous useful environments

- In `article` documents, the `figure` environment creates a floating figure element.
  - `figure` environments often contain a call to the `\caption{...}` command to set a caption
  - They also often contain a call such as `\includegraphics{path/to/some/image.png}` to insert an image
  - LaTeX also contains many flexible packages for producing your own graphics from scratch, such as `tikz`

LaTeX

Miscellaneous useful environments

- In **`article`** documents, the **`figure`** environment creates a floating figure element.
  - **`figure`** environments often contain a call to the **`\caption{...}`** command to set a caption
  - They also often contain a call such as **`\includegraphics{path/to/some/image.png}`** to insert an image
  - LaTeX also contains many flexible packages for producing your own graphics from scratch, such as **`tikz`**
- The **`itemize`** environment produces lists of items, each specified using the command **`\item`**, e.g. :

  ```
  \begin{itemize}
      \item one
      \item two
      \item three
  \end{itemize}
  ```

  A related environment is **`enumerate`**, which produces ordered lists.

Miscellaneous useful environments

- In `article` documents, the `figure` environment creates a floating figure element.
  - `figure` environments often contain a call to the `\caption{...}` command to set a caption
  - They also often contain a call such as `\includegraphics{path/to/some/image.png}` to insert an image
  - LaTeX also contains many flexible packages for producing your own graphics from scratch, such as `tikz`
- The `itemize` environment produces lists of items, each specified using the command `\item`, e.g. :

  ```
  \begin{itemize}
      \item one
      \item two
      \item three
  \end{itemize}
  ```

  A related environment is `enumerate`, which produces ordered lists.
- The `equation` environment allows to insert numbered equations within the text ; `equation` environments are by default in math mode.

LaTeX

Miscellaneous useful environments

- In `article` documents, the `figure` environment creates a floating figure element.
  - `figure` environments often contain a call to the `\caption{...}` command to set a caption
  - They also often contain a call such as `\includegraphics{path/to/some/image.png}` to insert an image
  - LaTeX also contains many flexible packages for producing your own graphics from scratch, such as `tikz`
- The `itemize` environment produces lists of items, each specified using the command `\item`, e.g. :

```
\begin{itemize}
    \item one
    \item two
    \item three
\end{itemize}
```

  A related environment is `enumerate`, which produces ordered lists.
- The `equation` environment allows to insert numbered equations within the text ; `equation` environments are by default in math mode.
- The `verbatim` environment allows its contents to not be processed by the compiler, and simply copied instead ; which can be useful for code listings, etc.

Miscellaneous useful environments

- In `article` documents, the `figure` environment creates a floating figure element.
  - `figure` environments often contain a call to the `\caption{...}` command to set a caption
  - They also often contain a call such as `\includegraphics{path/to/some/image.png}` to insert an image
  - LaTeX also contains many flexible packages for producing your own graphics from scratch, such as `tikz`
- The `itemize` environment produces lists of items, each specified using the command `\item`, e.g. :

  ```
  \begin{itemize}
      \item one
      \item two
      \item three
  \end{itemize}
  ```

  A related environment is `enumerate`, which produces ordered lists.
- The `equation` environment allows to insert numbered equations within the text ; `equation` environments are by default in math mode.
- The `verbatim` environment allows its contents to not be processed by the compiler, and simply copied instead ; which can be useful for code listings, etc.
- The `center` environment center-aligns its content.

Tables and `tabular` environments

- Tables are defined with `tabular` environments
  NB : in `article` documents, floating elements for tables are defined using the `table` environment.

LaTeX

Tables and `tabular` environments

- Tables are defined with `tabular` environments
  NB : in `article` documents, floating elements for tables are defined using the `table` environment.

- To illustrate :

```
\begin{tabular}{|r|c|l} \hline \hline
    1 & zwei & $\sqrt[\sqrt{9}]{3^3}$ \\ quatre & V & six \\ \hline
\end{tabular}
```

  produces this rather ugly table :

1	zwei	$\sqrt[\sqrt{9}]{3^3}$
quatre	V	six

Tables and `tabular` environments

- Tables are defined with `tabular` environments
  NB : in `article` documents, floating elements for tables are defined using the `table` environment.

- To illustrate :

```
\begin{tabular}{|r|c|l} \hline \hline
    1 & zwei & $\sqrt[\sqrt{9}]{3^3}$ \\ quatre & V & six \\ \hline
\end{tabular}
```

  produces this rather ugly table :

1	zwei	$\sqrt[\sqrt{9}]{3^3}$
quatre	V	six

- Calls to `\begin{tabular}` require a second argument that defines the number and styles of columns and column separators.

LaTeX

Tables and `tabular` environments

- Tables are defined with **`tabular`** environments
  NB : in **`article`** documents, floating elements for tables are defined using the **`table`** environment.

- To illustrate :

```
\begin{tabular}{|r|c|l} \hline \hline
    1 & zwei & $\sqrt[\sqrt{9}]{3^3}$ \\ quatre & V & six \\ \hline
\end{tabular}
```

  produces this rather ugly table :

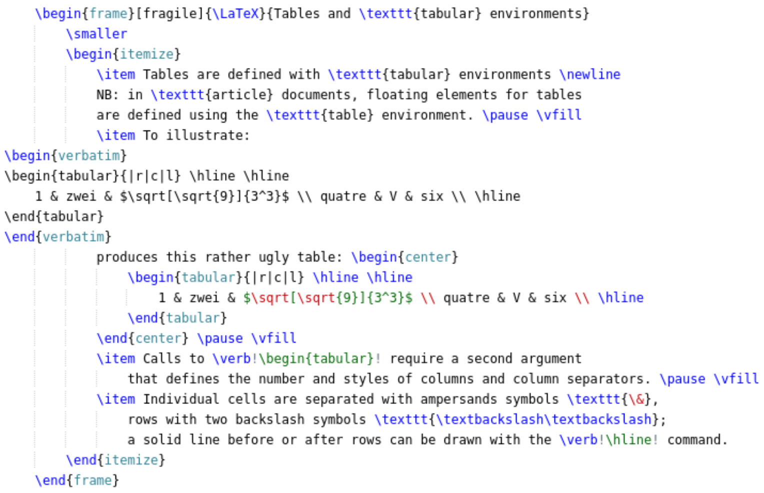
1	zwei	$\sqrt[\sqrt{9}]{3^3}$
quatre	V	six

- Calls to **`\begin{tabular}`** require a second argument that defines the number and styles of columns and column separators.

- Individual cells are separated with ampersands symbols **`&`**, rows with two backslash symbols **`\\`** ; a solid line before or after rows can be drawn with the **`\hline`** command.

LaTeX
Recap

- Here's a screen capture of the LaTeX code for the previous slide :



```
    \begin{frame}[fragile]{\LaTeX}{Tables and \texttt{tabular} environments}
        \smaller
        \begin{itemize}
            \item Tables are defined with \texttt{tabular} environments \newline
            NB: in \texttt{article} documents, floating elements for tables
            are defined using the \texttt{table} environment. \pause \vfill
            \item To illustrate:
\begin{verbatim}
\begin{tabular}{|r|c|l} \hline \hline
    1 & zwei & $\sqrt[\sqrt{9}]{3^3}$ \\ quatre & V & six \\ \hline
\end{tabular}
\end{verbatim}
            produces this rather ugly table: \begin{center}
                \begin{tabular}{|r|c|l} \hline \hline
                    1 & zwei & $\sqrt[\sqrt{9}]{3^3}$ \\ quatre & V & six \\ \hline
                \end{tabular}
            \end{center} \pause \vfill
            \item Calls to \verb!\begin{tabular}! require a second argument
                that defines the number and styles of columns and column separators. \pause \vfill
            \item Individual cells are separated with ampersands symbols \texttt{\&},
                rows with two backslash symbols \texttt{\textbackslash\textbackslash};
                a solid line before or after rows can be drawn with the \verb!\hline! command.
        \end{itemize}
    \end{frame}
```

LaTeX

Handling the bibliography

- LaTeX comes with many packages to handle bibliography.

Handling the bibliography

- LaTeX comes with many packages to handle bibliography.

  1. Create a file called `refs.bib`, in which you will store bilbiography entries. Here's an example entry :

     ```
     @article{Taylor53Cloze,
       author={Wilson Taylor},
       title={Cloze Procedure: A New Tool for Measuring Readability},
       journal={Journalism Quarterly},
       volume={30},
       year={1953}
     }
     ```

     What comes immediately after the opening curly brace (in this case, `Taylor53Cloze`) is the *key* by which you will refer to this entry in your LaTeX document.

Handling the bibliography

- LaTeX comes with many packages to handle bibliography.

1. Create a file called **`refs.bib`**, in which you will store bilbiography entries. Here's an example entry :

```
@article{Taylor53Cloze,
  author={Wilson Taylor},
  title={Cloze Procedure: A New Tool for Measuring Readability},
  journal={Journalism Quarterly},
  volume={30},
  year={1953}
}
```

   What comes immediately after the opening curly brace (in this case, `Taylor53Cloze`) is the *key* by which you will refer to this entry in your LaTeX document.

2. In the preamble, require a package to handle your bibliography, e.g. `\usepackage{biblatex}`, and then add your `.bib` resource : `\addbibresource{./path/to/refs.bib}`

Handling the bibliography

- LATEX comes with many packages to handle bibliography.

1. Create a file called **`refs.bib`**, in which you will store bilbiography entries. Here's an example entry :

```
@article{Taylor53Cloze,
  author={Wilson Taylor},
  title={Cloze Procedure: A New Tool for Measuring Readability},
  journal={Journalism Quarterly},
  volume={30},
  year={1953}
}
```

   What comes immediately after the opening curly brace (in this case, **`Taylor53Cloze`**) is the *key* by which you will refer to this entry in your LATEX document.
2. In the preamble, require a package to handle your bibliography, e.g. **`\usepackage{biblatex}`**, and then add your **`.bib`** resource : **`\addbibresource{./path/to/refs.bib}`**
3. Within the document, you can cite a reference using **`\cite{key}`**, where **`key`** is the associated key.

Handling the bibliography

- LaTeX comes with many packages to handle bibliography.

1. Create a file called `refs.bib`, in which you will store bilbiography entries. Here's an example entry :

   ```
   @article{Taylor53Cloze,
     author={Wilson Taylor},
     title={Cloze Procedure: A New Tool for Measuring Readability},
     journal={Journalism Quarterly},
     volume={30},
     year={1953}
   }
   ```

   What comes immediately after the opening curly brace (in this case, `Taylor53Cloze`) is the *key* by which you will refer to this entry in your LaTeX document.
2. In the preamble, require a package to handle your bibliography, e.g. `\usepackage{biblatex}`, and then add your `.bib` resource : `\addbibresource{./path/to/refs.bib}`
3. Within the document, you can cite a reference using `\cite{key}`, where `key` is the associated key.
4. Anywhere in the document, you can print the bibliography with `\printbibliography`.

Handling the bibliography

- LaTeX comes with many packages to handle bibliography.

  1. Create a file called **`refs.bib`**, in which you will store bilbiography entries. Here's an example entry :

     ```
     @article{Taylor53Cloze,
       author={Wilson Taylor},
       title={Cloze Procedure: A New Tool for Measuring Readability},
       journal={Journalism Quarterly},
       volume={30},
       year={1953}
     }
     ```

     What comes immediately after the opening curly brace (in this case, **`Taylor53Cloze`**) is the *key* by which you will refer to this entry in your LaTeX document.
  2. In the preamble, require a package to handle your bibliography, e.g. **`\usepackage{biblatex}`**, and then add your **`.bib`** resource : **`\addbibresource{./path/to/refs.bib}`**
  3. Within the document, you can cite a reference using **`\cite{key}`**, where **`key`** is the associated key.
  4. Anywhere in the document, you can print the bibliography with **`\printbibliography`**.

- Some editors, such as Overleaf, will deal with linking the **`.bib`** resource to the compiled file for you. Others (espacially command line TeX-compilers) require you to do some extra steps, which should be mentionned at the end of their output.

What next ?

This was only an overview ! We skipped over many parts of the syntax, this was merely an introduction.

What next ?

This was only an overview ! We skipped over many parts of the syntax, this was merely an introduction.

Here are some topics to look into :

- We haven't talked about defining your own commands

This was only an overview ! We skipped over many parts of the syntax, this was merely an introduction.

Here are some topics to look into :

- We haven't talked about defining your own commands
- We haven't discussed the `tikz` package to produce graphics

This was only an overview ! We skipped over many parts of the syntax, this was merely an introduction.

Here are some topics to look into :

- We haven't talked about defining your own commands
- We haven't discussed the `tikz` package to produce graphics
- We haven't seen how to automatically generate a table of contents based on sections

This was only an overview ! We skipped over many parts of the syntax, this was merely an introduction.

Here are some topics to look into :

- We haven't talked about defining your own commands
- We haven't discussed the `tikz` package to produce graphics
- We haven't seen how to automatically generate a table of contents based on sections
- We haven't covered how to use *labels* to dynamically refer to specific elements (sections, figures, tables, equations...)

This was only an overview ! We skipped over many parts of the syntax, this was merely an introduction.

Here are some topics to look into :

- We haven't talked about defining your own commands
- We haven't discussed the `tikz` package to produce graphics
- We haven't seen how to automatically generate a table of contents based on sections
- We haven't covered how to use *labels* to dynamically refer to specific elements (sections, figures, tables, equations...)
- And many more...