ENH1301 - Application Laboratory

October 8, 2025

Contents

1 LaTeX Typesetting

1.1 What is LaTeX?

LaTeX (pronounced "Lay-tech" or "Lah-tech") is a high-quality typesetting system[24]. Unlike a traditional word processor, LaTeX is driven by plain-text source code that describes the structure and content of a document rather than its appearance. This separation of content and formatting makes LaTeX ideal for scientific, technical and academic writing because it automatically handles layout and typography[24].

1.2 Why Choose LaTeX?

- **Professional formatting:** automatic numbering of sections, figures and tables as well as consistent headers and bibliographies[25].
- **Consistency:** headings, captions and references always match because they are generated programmatically[25].
- **Excellent for mathematics:** LaTeX excels at typesetting formulas, equations and special symbols[25].
- **Scalability:** large documents such as theses or books maintain their layout without corruption[25].
- **Cross-references:** figures, tables and equations can be referenced by label and are automatically updated[25].

1.3 Typical Use Cases

LaTeX is widely used for academic papers, research journals, theses, dissertations and technical reports. It is also suitable for CVs and resumes, slide presentations using the *beamer* class and for creating books or technical manuals[26].

1.4 Tools for Working with LaTeX

Several distributions and editors make LaTeX accessible:[27]

- **Overleaf:** an online collaborative editor that requires no local installation and is ideal for group projects.

- **TeX Live:** a comprehensive LaTeX distribution for Linux, macOS and Windows.
- **MiKTeX:** a lightweight distribution targeted at Windows users.
- **Editors:** TeXstudio, TeXmaker and VS Code with the LaTeX extension provide syntax highlighting and build tools.

1.5 Document Structure

A minimal LaTeX document begins by declaring its class (article, report, book or beamer) using `\documentclass{...}` and encloses all content between `\begin{document}` and `\end{document}`[28]. Titles, authors and dates are inserted with the commands `\title{...}`, `\author{...}` and `\date{...}` and displayed using `\maketitle`. Sections and subsections are created with `\section{...}` and `\subsection{...}` commands respectively, and LaTeX automatically numbers and formats them.

1.6 Formatting Text

LaTeX provides numerous commands for emphasising text. Use `\textbf{...}` for bold, `\textit{...}` for italics and `\underline{...}` for underlining. Superscripts and subscripts in ordinary text are produced with `\textsuperscript{...}` and `\textsubscript{...}`. Lists are created using the `itemize` (bulleted) and `enumerate` (numbered) environments, each item introduced by `\item`.

1.7 Mathematics

Because of its precision and beauty, LaTeX is the de facto standard for mathematical typesetting. It produces professional, clean equations and gives precise control over symbols and spacing[29]. There are two main math modes:[29]

- **Inline math** uses `$...$` to embed equations within a paragraph, for example $E = mc^2$.
- **Display math** uses `\[ ... \]` to centre an equation on its own line.

Superscripts are written with `^` and subscripts with `_`, for instance x_i^2. Fractions use `\frac{numerator}{denominator}`, square roots use `\sqrt{...}` and higher-order roots with `\sqrt[n]{...}`. Greek letters and common symbols are available through simple

commands (e.g., `\alpha`, `\beta`, `\gamma`, `\Omega` for Greek letters and `\infty`, `\le`, `\ge`, `\rightarrow` for common symbols).

1.8 Tables and Figures

LaTeX offers powerful tools for including data and graphics. The `tabular` environment creates tables by specifying columns (e.g., `{lcr}` for left, centre and right aligned columns). Horizontal lines are inserted with `\hline` and entries are separated by `&` and terminated with `\\`. Figures are included with `\includegraphics{...}` provided by the `graphicx` package. Captions and cross-references are added using `\caption{...}` and `\label{...}` within a `figure` environment. A reference to a labelled figure automatically updates if the figure number changes[30]. Table 1 shows a simple table.

Table 1: Example of a simple table in LaTeX.

Name	Age	Grade
Alice	20	A
Bob	22	B
Charlie	21	A

1.9 Labels, Cross-References and Citations

Labels allow you to refer to sections, figures, tables or equations without hard-coding numbers. Assign a label to an object with `\label{key}` and reference it with `\ref{key}` or `\eqref{key}` in the case of equations. LaTeX automatically updates numbers when objects move or are added, ensuring that references remain accurate[30]. Citations are created with the `\cite{...}` command and a bibliography file (`.bib`) processed with `\bibliography{...}`. For example, `\cite{einstein1905}` would produce a citation and the bibliography would contain entries in BibTeX format. Managing references through a BibTeX file centralises bibliographic data and keeps the main document clean.

1.10 Useful Packages

Several packages extend LaTeX's capabilities:[31]

- **graphicx** – for including images.
- **amsmath** – enhanced mathematical notation and environments.

- **geometry** – flexible page layout control.
- **hyperref** – adds clickable hyperlinks and internal document navigation.
- **xcolor** – defines and uses colours in text and graphics.

2 Guidelines for Combining LaTeX, Code Listings and Bibliography Files

Exam questions sometimes provide multiple files—typically a main `.tex` document, a script (e.g., `sample.sh`) and a bibliography database (e.g., `ex.bib`). Your task is to understand how these components are integrated and to produce the final typeset output.

Understanding the Components

- **Main LaTeX file (e.g., `ex.tex`):** Contains the document structure, text, tables and commands that reference external resources. In the example shown in the exam, it uses the `natbib` package for citations and the `listings` package to include code.
- **Code file (e.g., `sample.sh`):** A Bash script or other source code that is included in the document via `\lstinputlisting`. The `language` option controls syntax highlighting.
- **Bibliography file (e.g., `ex.bib`):** A BibTeX database containing entries referenced with commands like `\citep{...}`. The bibliography is formatted according to the chosen style (e.g., `apalike`).

Key Commands in the Example

`\citep{kostova2020privacy}` Inserts an in-text citation referring to the entry with key `kostova2020privacy` in `ex.bib`.

`\begin{tabular}{||l|c|r|}` Creates a table with three columns aligned left, centred and right. Double vertical bars draw thicker lines at the left and right edges; `\hline` inserts horizontal rules.

`\lstinputlisting[language=bash]{sample.sh}` Reads and displays the contents of `sample.sh` with Bash syntax highlighting.

`\bibliography{ex}` Loads bibliography entries from `ex.bib`. The preceding `\bibliographystyle{apalike}` sets the citation and reference style.

Compiling the Document

To see the final output that combines all files, run LaTeX and BibTeX in sequence. From a terminal in the directory containing the files, execute:

```
pdflatex ex.tex    # first pass: generates a .aux file
bibtex ex          # processes citations and builds bibliography
pdflatex ex.tex    # second pass: resolves references
pdflatex ex.tex    # final pass: ensures all links are updated
```

The resulting PDF will include the mathematical expression ("The text $10_{\log}2$ found in the book"), the formatted table with bold and aligned columns, a syntax-highlighted listing of the Bash script that prints "Count: 1" through "Count: 5," and a properly formatted bibliography entry drawn from `ex.bib`.

What to Explain in Your Answer

When asked to "write the output" of such a script, describe or reproduce the visible contents of the compiled document:

1. The typeset text and mathematical expressions exactly as rendered.
2. The table, with its column headings and cell contents aligned as specified.
3. The code listing, including line breaks and indentation.
4. The citation marker in the text (e.g., "[1]") and the corresponding full reference in the bibliography.

Clearly indicating these elements shows that you understand how the LaTeX source, the script and the bibliography file work together to produce the final document.

3 Version Control with Git

3.1 Git Commands Used in the Video

The following table summarises commonly used Git commands demonstrated in the course video. Each command is shown with a brief description of its purpose.

Table 2: Common Git commands and their purposes.

Command	Purpose
`git init`	Initializes a new Git repository in the current directory.
`git config --global user.name`	Sets the global user name for commits across all repositories.
`git config --global user.email`	Sets the global email address for commits across all repositories.
`git config --list`	Displays the current Git configuration settings.
`git status`	Shows the state of the working directory and staging area.
`git add <file>`	Stages changes in a specific file for the next commit.
`git add .`	Stages all new and modified files in the current directory.
`git add *`	Stages files using a wildcard (equivalent to `git add .`).
`git commit -m ‘‘message’’`	Creates a new commit with a short message.
`git commit`	Opens the default editor to create a detailed commit message.
`git log`	Displays the commit history.
`git log -p`	Shows each commit with the patch (diff) it introduces.
`git log -- <file>`	Shows commits that affected a specific file.
`git diff`	Displays unstaged changes (working directory vs. staging area).
`git diff --staged`	Displays staged changes (staging area vs. last commit).
`git rm <file>`	Removes a file and stages its deletion.
`git checkout -- <file>`	Discards changes in a file, restoring it from the staging area.
`git checkout <hash> -- <file>`	Restores a file from a specific commit.
`git reset HEAD <file>`	Unstages a file, moving it back to the working directory.

3.2 Other Mentioned Commands and Options

- `sudo apt-get install git` – installs Git on Debian/Ubuntu systems.
- `--global` – applies a configuration change to all repositories on the system.
- `--local` – applies a configuration change only to the current repository.
- `HEAD` – represents the current commit that the branch pointer references.

4 Essential Linux Commands

4.1 Navigation and Directory Management

`pwd` Prints the current working directory.

```
$ pwd
/home/user
```

`ls` Lists files and directories. Options such as `-l` produce a long listing, and `-a` includes hidden files.

```
$ ls -l
drwxr-xr-x 2 user user 4096 Oct  1 10:00 Documents
-rw-r--r-- 1 user user  123 Oct  1 09:58 notes.txt
```

`cd` *`dir`* Changes the working directory to *dir*. Running `cd` without arguments returns to the home directory.

```
$ cd Documents
$ pwd
/home/user/Documents
```

`mkdir` *`dir`* Creates a new directory named *dir*.

```
$ mkdir project
$ ls
project  notes.txt
```

`rmdir` *`dir`* Removes an empty directory.

```
$ rmdir project
$ ls
notes.txt
```

4.2 File Operations

`touch` *`file`* Creates an empty file or updates the modification time if the file exists.

```
$ touch report.txt
$ ls
notes.txt  report.txt
```

`cp` *`source dest`* Copies a file or directory. Use `-r` to copy directories recursively.

```
$ cp notes.txt notes.bak
$ ls
notes.bak  notes.txt  report.txt
```

`mv` *`source dest`* Moves or renames files and directories.

```
$ mv report.txt project/report.txt
```

`rm` *`file`* Removes files. Use `-r` to delete directories and their contents.

```
$ rm notes.bak
$ rm -r project
```

`tar` Archives or extracts collections of files and directories; options like `-c` (create), `-x` (extract) and `-f` (specify archive file) are commonly used.

```
$ tar -cf backup.tar Documents
$ tar -xf backup.tar
```

`zip/unzip` Compresses or extracts files in the ZIP format.

```
$ zip archive.zip *.txt
adding: notes.txt (stored 0%)
adding: report.txt (stored 0%)
$ unzip archive.zip
```

```
Archive:  archive.zip
  inflating: notes.txt
  inflating: report.txt
```

4.3 Viewing and Editing Files

`cat` *file* Concatenates files and prints them to the standard output.

```
$ cat notes.txt
Meeting agenda:
1. Review progress
2. Plan next steps
```

`less` / `more` Displays the contents of a file one page at a time, allowing forward and backward navigation.

```
$ less notes.txt
Meeting agenda:
1. Review progress
...
(press q to quit)
```

`head` *file* Shows the first 10 lines of a file (use `-n` to specify a different number).

```
$ head -n 2 notes.txt
Meeting agenda:
1. Review progress
```

`tail` *file* Shows the last 10 lines of a file; `-f` follows a file as it grows, useful for logs.

```
$ tail -n 2 notes.txt
2. Plan next steps
(End of file)
```

`nano`, `vim` Text editors for creating and modifying files directly from the terminal.

```
$ nano notes.txt   # opens the file in the Nano editor
$ vim notes.txt    # opens the file in the Vim editor
```

4.4 Searching and Text Processing

`grep` *`pattern file`* Searches for lines matching a pattern within files. Options like `-i` ignore case and `-r` search directories recursively.

```
$ grep -i "meeting" notes.txt
Meeting agenda:
```

`find` *`path expression`* Searches for files and directories in a hierarchy based on name, size, type or modification time.

```
$ find . -name "*.txt"
./notes.txt
./project/report.txt
```

`wc` *`file`* Counts lines, words and bytes in files; options `-l`, `-w` or `-c` report specific counts.

```
$ wc -l notes.txt
3 notes.txt
```

`cut`, `sort`, `uniq` Utilities for column extraction, ordering and duplicate removal in text streams.

```
$ cut -d',' -f2 data.csv | sort | uniq -c
  3 Sales
  5 Support
```

4.5 Permissions and Ownership

`chmod` *`mode file`* Changes the permissions of a file or directory. Modes can be specified numerically (e.g., `chmod 755 file`) or symbolically (e.g., `chmod u+rwx,g+rx,o+rx file`).

```
$ ls -l script.sh
-rw-r--r-- 1 user user 0 Oct  1 12:00 script.sh
$ chmod +x script.sh
$ ls -l script.sh
-rwxr-xr-x 1 user user 0 Oct  1 12:00 script.sh
```

`chown` *`owner`*`[:`*`group`*`]` *`file`* Changes the owner and optionally the group of a file or directory.

```
$ sudo chown root:root script.sh
$ ls -l script.sh
-rwxr-xr-x 1 root root 0 Oct  1 12:00 script.sh
```

`umask` Sets default permission bits for newly created files and directories.

```
$ umask
0022
```

4.6 System Information and Process Management

`uname -a` Prints system information including kernel name, version and architecture.

```
$ uname -a
Linux mymachine 5.15.0-41-generic #44-Ubuntu SMP x86_64 GNU/Linux
```

`df -h` Reports disk space usage of mounted filesystems in human-readable form.

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        50G   20G   28G  42% /
```

`du -h` *`dir`* Shows disk usage of a directory and its contents.

```
$ du -h project
4.0K    project/report.txt
8.0K    project
```

`top` Displays running processes and resource usage in real time.

```
$ top -b -n1 | head -n 5
top - 12:00:00 up 1 day,  3:42,  1 user,  load average: 0.00, 0.01, 0.05
Tasks: 193 total,   1 running, 192 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.0 us,  0.5 sy,  0.0 ni, 98.0 id,  0.5 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   7973.2 total,   6000.0 free,   1000.0 used,    973.2 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   6973.2 avail Mem
```

`ps aux` Lists currently running processes with detailed information.

```
$ ps aux | head -n 3
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1 168816  1152 ?        Ss   Sep28   0:05 /sbin/init
user      1234  0.1  0.2  98764  3456 pts/0    Ss   11:59   0:00 bash
```

`kill` *`PID`* Sends a signal (default is `SIGTERM`) to terminate the process with the given PID.

```
$ kill 1234   # terminates process with PID 1234
```

`man` *`command`* Opens the manual page for a command, describing its usage and options.

```
$ man ls
LS(1)                        User Commands                        LS(1)
NAME
       ls - list directory contents
...
```

`history` Shows previously executed commands in the current shell session.

```
$ history | tail -n 2
101  ls -l
102  history
```

4.7 Networking and Remote Access

`ping` ***host*** Sends ICMP echo requests to test network connectivity.

```
$ ping -c 2 example.com
PING example.com (93.184.216.34) 56(84) bytes of data.
64 bytes from example.com: icmp_seq=1 ttl=56 time=20.3 ms
64 bytes from example.com: icmp_seq=2 ttl=56 time=20.5 ms

--- example.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
```

`ifconfig` **or** `ip addr` Displays network interface configuration.

```
$ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFA
    inet 192.168.1.10/24 brd 192.168.1.255 scope global dynamic eth0
```

`ssh` ***user@host*** Securely logs in to a remote machine. Options such as `-p` specify an alternative port, and `-i` selects a private key.

```
$ ssh alice@remote.server
alice@remote.server's password:
Welcome to Ubuntu 22.04 LTS
alice@remote:~$
```

`scp` ***source dest*** Securely copies files between hosts over SSH.

```
$ scp notes.txt alice@remote.server:~/notes.txt
notes.txt                                 100%   50     1.5KB/s   00:00
```

`wget` *URL* Downloads files from the web via HTTP, HTTPS or FTP.

```
$ wget https://example.com/index.html
Saving to: ‘index.html’
index.html          100%[===================>]   138  --.-KB/s    in 0s
```

`curl` *URL* Transfers data to or from a server using various protocols; options allow sending requests, headers and payloads.

```
$ curl -I https://example.com
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
...
```

5 Advanced Linux Commands and Bash Programming

Advanced Command-Line Tools

`find` Searches for files and directories in a directory hierarchy. Common options include `-name` to match by name, `-type` to filter by file type (`f` for files, `d` for directories), `-size` to match by size, and `-exec` to run a command on each match. For example:

```
find /var/log -type f -name "*.log" -mtime +7 -exec rm
```

finds log files older than seven days and deletes them.

`grep` Searches text for patterns. Use `-i` for case-insensitive matching, `-r` for recursive search, `-n` to show line numbers, `-v` to invert the match and `-E` for extended regular expressions. Example: `grep -rn "error" /etc` searches all files under `/etc` for lines containing “error”.

`awk` A powerful text-processing language for pattern scanning and reporting. Fields in a line are accessed with `$1`, `$2`, etc. Example: `awk 'print $1, $3' file.txt` prints the first and third fields of each line.

`sed` A stream editor used to transform text. The substitute command `s/old/new/g` replaces all occurrences of `old` with `new`. For instance, `sed 's/foo/bar/g' file.txt` outputs the contents of `file.txt` with every "foo" replaced by "bar".

`xargs` Constructs command lines from standard input. It pairs nicely with `find` and `grep`. For example: `ls *.zip | xargs -n1 unzip` unzips every `.zip` file in the current directory.

`sort` and `uniq` `sort` orders lines alphabetically or numerically (`-n`), while `uniq` removes adjacent duplicates (`-c` counts occurrences). Chaining them together, `sort file.txt | uniq -c | sort -nr` displays the most frequent lines first.

`tar` and `gzip` `tar` archives multiple files into a single file and `gzip` compresses data. The combined command `tar czf archive.tar.gz /path/to/dir` creates a compressed archive of a directory, while `tar xzf archive.tar.gz` extracts it.

Introduction to Bash Programming

Bash is the default command interpreter on most Linux systems and can execute scripts written in plain text.

Creating a Script Start a new file with a shebang line to specify the interpreter, then add commands. Make the script executable with `chmod +x script.sh` and run it using `./script.sh`.

```
#!/bin/bash
# A simple greeting script
echo "Hello, $USER!"
```

Variables and Arithmetic Assign variables without spaces (`name=value`) and reference them with a dollar sign (`$name`). Use (()) for arithmetic:

```
count=5
echo "Count is $count"
echo "Next is $((count + 1))"
```

Conditionals Use `if` statements to branch execution. Numeric comparisons use `-eq`, `-ne`, `-gt`, `-lt` while string comparisons use `=` and `!=`.

```
if [ "$count" -gt 10 ]; then
echo "Large count"
else
echo "Small count"
fi
```

Loops `for` loops iterate over a list of values or the output of a command; `while` loops run as long as a condition is true.

```
for file in *.txt; do
echo "Processing $file"
done

i=0
while [ $i -lt 3 ]; do
echo "i is $i"
i=$((i + 1))
done
```

Functions Encapsulate reusable code blocks in functions. Arguments are accessed as `$1`, `$2`, etc.

```
greet() {
 echo "Hello, $1!"
}
greet "Alice"
```

Input/Output Redirection and Pipes Use `>` to redirect output to a file, `>>` to append, and `|` to pipe the output of one command into another.

```
echo "Log entry" >> logfile.txt
cat logfile.txt | grep "entry"
```

This section provides a foundation for using more sophisticated command-line tools and writing simple Bash scripts to automate tasks. You can expand upon these examples with additional commands, tests and error handling as your proficiency grows.

6 Vim Editor Commands

Vim is a modal editor: commands entered in *command mode* manipulate text without inserting characters. Exam questions often ask for the key sequences to perform specific actions. The table below lists the commands used in the example question and other frequently tested operations.

Essential Navigation and Editing

`0` **or** `^` Move the cursor to the beginning of the current line.

`yy` "Yank" (copy) the current line into the unnamed register.

`120j` Move the cursor down 120 lines; replace `120` with any count to move multiple lines.

`:vsp Banda` Split the view vertically and open a new file named `Banda` in the right pane.

`:r !ls -l` Read the output of the shell command `ls -l` into the current buffer at the cursor position.

`]s` Jump to the next spelling mistake when spell-checking is enabled (`:set spell`).

`gg` Go to the top of the document; use `G` to go to the bottom.

`u` Undo the last change; `Ctrl+r` redoes an undone change.

Other Useful Commands

`:w` Save ("write") the current file to disk.

`:q` Quit Vim. Combine as `:wq` to save and quit, or use `:q!` to discard changes.

`dd` Delete (cut) the current line; `p` pastes the most recently yanked or deleted text after the cursor.

`/pattern` Search forward for `pattern`; press `n` to repeat the search forward or `N` to search backward.

`:%s/old/new/g` Replace all occurrences of `old` with `new` in the entire file. Omit `g` to replace only the first occurrence on each line.

`v`, `V`, `Ctrl+v` Enter visual mode to select characters, entire lines, or a rectangular block for subsequent commands.

`>>`, `<<` Indent or unindent the current line; prefix with a count (e.g., `3>>`) to adjust multiple lines.

When answering exam questions, specify the exact key presses in the order shown. Remember that most commands can be prefixed with a numeric count to repeat the action multiple times.

7 LibreOffice Calc Essentials

LibreOffice Calc offers a rich set of spreadsheet tools analogous to other spreadsheet applications. Below are some key commands and features to know:

Basic Shortcuts

Ctrl+N Create a new spreadsheet.

Ctrl+O Open an existing spreadsheet file.

Ctrl+S Save the current spreadsheet.

Ctrl+Z / Ctrl+Y Undo and redo the last action.

Ctrl+C, Ctrl+V, Ctrl+X Copy, paste and cut selected cells.

Ctrl+F1 Opens the *Function Wizard* to help insert formulas.

F2 Edit the active cell without overwriting its contents.

Formula Basics

`=SUM(A1:A10)` Adds all numbers in the range `A1` through `A10`.

`=AVERAGE(B1:B5)` Calculates the average of a range of numbers.

`=MIN()`, `=MAX()` Return the smallest or largest value in a range.

`=IF(condition; value_if_true; value_if_false)` Performs conditional calculations.

`=VLOOKUP(key; table; col; sorted)` Searches the first column of a table for a key and returns a value from another column.

`$A$1` Absolute cell reference; the dollar signs prevent the row or column from changing when copying formulas.

Data Tools

Sorting Use *Data* → *Sort* to order rows by the values in one or more columns.

Filtering *Data* → *AutoFilter* inserts drop-down menus into header cells for quick filtering.

Cell Formatting Press **Ctrl+1** or use *Format* → *Cells* to change number formats, alignment, fonts and borders.

Insert/Delete Right-click a row or column header and select *Insert Rows/Columns* or *Delete* to modify the sheet structure.

Charts Select your data, then choose *Insert* → *Chart* to create bar, line or pie charts.

8 Touch Typing Practice

Proper Finger Placement

Figure 1 illustrates the recommended finger placement for a QWERTY keyboard. Each colour corresponds to the keys controlled by a particular finger. The home row (ASDF for the left hand and JKL; for the right hand) is where your fingers should rest when not typing other keys.

Questions for Practice

1. According to Figure 1, which finger is responsible for typing the keys `E`, `D` and `C`? List all the keys that finger must cover.
2. When your left index finger is resting on the `F` key (home row), which other keys should it reach? Explain why returning to the home row after each keystroke is important.
3. Identify the keys assigned to the right pinky finger. Besides letters and numbers, which punctuation marks or modifier keys does it control?

Figure 1: Colour–coded finger assignments on a QWERTY keyboard for touch typing. Fingers return to the home row (middle row) after pressing other keys.

9 Secure Shell (SSH)

9.1 Overview and Key Features

Secure Shell (SSH) is a *cryptographic network protocol* that enables secure remote login and command execution on remote machines[1]. Unlike older protocols such as Telnet, SSH encrypts all data—including passwords—before transmission so that sensitive information is not sent in clear text over the network[1]. In addition to remote login, SSH supports secure file transfer using the `scp` and `sftp` protocols and provides a robust platform for remote administration[1]. Figure 2 illustrates a conceptual view of an SSH connection.

9.2 Installing and Configuring OpenSSH

To accept incoming SSH connections on a Linux system the OpenSSH server package must be installed. The server can be installed with the standard package manager (e.g., `sudo apt update` followed by `sudo apt install openssh-server`)[2]. Many distributions already provide the SSH client; the installed version can be checked with `ssh -V`. If the client is missing, it can be installed via `sudo apt install openssh-client`[2].

Figure 2: Conceptual illustration of a secure SSH connection. The network traffic between the client and server is encrypted, protecting credentials and data in transit.

Once installed, the SSH service is managed with `systemd`. Use `sudo systemctl start ssh` to start the service, `sudo systemctl enable ssh` to enable automatic startup, and `sudo systemctl status ssh` to verify that it is running[3]. These commands should be executed on the server (the machine accepting connections).

9.3 Connecting to a Remote Host

The basic syntax for connecting to a remote machine is

```
ssh userhostname_or_ip
```

where `user` is a valid account on the remote system and `hostname_or_ip` is the domain name or IP address[4]. On the first connection SSH prompts the user to verify the host's fingerprint; after trusting the host, it requests the remote user's password[4]. Instead of starting an interactive shell it is often convenient to run a single command remotely:

```
ssh userhost "command"
```
[5]

which executes `command` on the remote server and returns its output.

9.4 Key-based Authentication

SSH supports passwordless login through public–private key pairs. Keys are generated locally with

```
ssh-keygen -t rsa -b 4096
```

which, by default, creates a private key (`~/.ssh/id_rsa`) and a corresponding public key (`~/.ssh/id_rsa.pub`)[6]. The public key is copied to the remote account using

```
ssh-copy-id userhost
```

After the key has been installed on the remote system, subsequent logins no longer prompt for a password[6]. Keys eliminate the need to transmit passwords and allow further hardening by disabling password authentication entirely.

9.5 Transferring Files Securely

Two built-in utilities handle encrypted file transfer over SSH. The `scp` command copies files or directories. For example, to send a local file to a remote host use

```
scp file.txt userhost:/home/user/
```
[7]

and to retrieve a file back to the local machine use

```
scp userhost:/home/user/file.txt .
```
[7]

For interactive transfers the Secure File Transfer Protocol (SFTP) is available. Running `sftp userhost` opens an FTP-like session where commands such as `ls`, `cd`, `put`, `get` and `bye` are available[8].

9.6 Configuration and Useful Options

Client-side configuration can be stored in `~/.ssh/config`. A host alias is defined with entries such as

```
Host myserver
  HostName 192.168.1.100
  User alice
  Port 22
```

Once configured you may simply run `ssh myserver`[9]. Common command-line options include `-p` to select a non-standard port, `-v` for verbose debugging output, `-X` to enable X11 forwarding for graphical applications and `-N` to establish a tunnel without executing a remote command[10].

9.7 Hardening Tips

Security can be improved by changing the default port in `/etc/ssh/sshd_config`, disabling direct root login via `PermitRootLogin no`, and turning off password authentication with `PasswordAuthentication no`[11]. Any changes require restarting the SSH service with `sudo systemctl restart ssh`[11]. Enforcing key-only authentication and using non-standard ports help reduce automated attacks.

10 Using PuTTY on Windows

PuTTY is a free and open-source SSH client for Windows that provides terminal emulation for connecting to Linux systems securely[13]. To install PuTTY, visit the official web site, download the Windows installer (e.g., `putty-64bit<version>.msi`), run the installer using default settings and launch PuTTY from the Start menu[14].

10.1 Connecting with PuTTY

When connecting to a server, enter the remote host's IP address or domain name into the *Host Name* field and ensure the port is set to 22 and the connection type is *SSH*[15]. Optionally, save the session under a meaningful name so you can quickly reconnect later. Click *Open*, accept the security prompt on the first connection, then log in with your username and password[15].

10.2 Key Management with PuTTYgen

PuTTY uses its own private-key format (`.ppk`). Launch *PuTTYgen* from the Start menu, select RSA, click *Generate* and move the mouse to create entropy. Save the generated private key as a `.ppk` file and copy the displayed public key into the remote account's `/.ssh/authorized_keys` file[16]. To use the key, load the saved session in PuTTY, navigate to *Connection extgreater SSH extgreater Auth*, browse for the `.ppk` file and connect. After successful setup, logins no longer require a password[16].

10.3 File Transfer via PSCP and SFTP

PuTTY does not include a graphical file transfer client. Instead, the companion utility *pscp.exe* can be downloaded from the PuTTY web site. Running

`pscp C:\Users\user\Documents\hello.txt userhost:/home/user/` [17]

copies a file from Windows to a Linux host. To copy a file from Linux back to Windows simply reverse the order of the source and destination paths[17]. A GUI alternative is FileZilla: select the SFTP protocol, enter the host address, username, password and port 22, and drag-and-drop files between the local and remote directories[17].

10.4 Connecting to Virtual Machines and Troubleshooting

When connecting to a virtual machine in the cloud or via virtualization software you will need the VM's public IP (or domain), the SSH port (usually 22), a username and either the user's password or an SSH key[18]. In PuTTY, enter the IP, verify the port is correct and select *SSH*; optionally save the session[19]. Accept any host-key warning, then enter the username and password when prompted[19]. If the connection fails, verify that the SSH service is running on the VM using `sudo systemctl status ssh` and ensure that port 22 is open in the firewall (e.g., `sudo ufw status`)[20]. Long-running sessions can be kept alive by setting a keepalive interval (e.g., 30 seconds) under *Connection > Seconds between keepalives*[19]. Common troubleshooting commands include `whoami`, `uname -a`, `df -h`, `uptime` and `sudo apt update` to verify system state[21]. Should you receive *permission denied* errors, double-check the username and that the correct private key is used[20]. When using UFW, allow port 22 with `sudo ufw allow 22` and reload the firewall to apply changes[22].

Activities for Hands-On Practice

In addition to studying the theory, practical exercises reinforce understanding. The following activities are adapted from the course materials:[12]

1. **Enable and test SSH locally:** install the OpenSSH server, start and enable the service, determine the local IP with `ip a` and connect to your own machine using `ssh yourusername@127.0.0.1`; exit the session with `exit`[12].
2. **Set up key-based authentication:** generate a key pair with `ssh-keygen` and copy the public key to the remote machine using `ssh-copy-id`; verify that subsequent logins no longer require a password[12].
3. **Practice secure file transfer:** create a test file, copy it to the remote host with `scp`, then retrieve it back; explore SFTP commands like `ls`, `put` and `get`[12].
4. **Explore remote commands:** run commands such as `df -h` or `cat /etc/os-release` via SSH without opening an interactive shell[12].
5. **Manage directories and permissions:** create nested directories (e.g., `mkdir -p /project/reports/2025`), navigate using relative paths, and experiment with `pushd`/`popd`. Create files, view and change their permissions with `chmod` and examine the effects as a different user[23].

References

[1] "SSH - Secure Shell," course notes, lines 2–15.

[2] "SSH - Secure Shell," course notes, lines 18–29.

[3] "SSH - Secure Shell," course notes, lines 30–39.

[4] "SSH - Secure Shell," course notes, lines 41–51.

[5] "SSH - Secure Shell," course notes, lines 53–57.

[6] "SSH - Secure Shell," course notes, lines 59–70.

[7] "SSH - Secure Shell," course notes, lines 72–79.

[8] "SSH - Secure Shell," course notes, lines 80–85.

[9] “SSH - Secure Shell,” course notes, lines 87–97.

[10] “SSH - Secure Shell,” course notes, lines 98–103.

[11] “SSH - Secure Shell,” course notes, lines 105–118.

[12] “SSH - Secure Shell,” course notes, lines 121–145.

[13] “SSH - Secure Shell,” course notes, lines 149–152.

[14] “SSH - Secure Shell,” course notes, lines 154–160.

[15] “SSH - Secure Shell,” course notes, lines 161–174.

[16] “SSH - Secure Shell,” course notes, lines 176–203.

[17] “SSH - Secure Shell,” course notes, lines 204–225.

[18] “SSH - Secure Shell,” course notes, lines 262–272.

[19] “SSH - Secure Shell,” course notes, lines 274–295.

[20] “SSH - Secure Shell,” course notes, lines 317–355.

[21] “SSH - Secure Shell,” course notes, lines 309–315.

[22] “SSH - Secure Shell,” course notes, lines 346–353.

[23] “SSH - Secure Shell,” course notes, lines 360–377.

[24] “LaTeX – What you get is what you want,” slides, lines 12–22.

[25] “LaTeX – What you get is what you want,” slides, lines 25–40.

[26] “LaTeX – What you get is what you want,” slides, lines 43–49.

[27] “LaTeX – What you get is what you want,” slides, lines 51–63.

[28] “LaTeX – What you get is what you want,” slides, lines 65–69.

[29] “LaTeX – What you get is what you want,” slides, lines 95–104.

[30] “LaTeX – What you get is what you want,” slides, lines 117–122.

[31] “LaTeX – What you get is what you want,” slides, lines 148–154.