HPC

Digital Learning Hub_
Academy
Powered by HPC

CS ACADEMY

PREPARATORY COURSE FOR CYBERSECURITY ACADEMY - GUIDE 1 OF 4

1 Linux Fundamentals

Introductory Course to Linux Fundamentals

THE THREE GUIDES

1 Linux Fundamentals

2 Network Fundamentals

3 Scripting Fundamentals

4 GitHub

Cybersecurity Academy by Digital Learning Hub

November 2026

I - What Is Linux

Linux is strictly the kernel: the core piece of software that talks to hardware (CPU, memory, disks, network cards) and shares it between programs. A **distribution** (distro) bundles the kernel with tools, a package manager and (optionally) a desktop. Ubuntu, Debian, Fedora, Arch and Kali are all distributions.

Distro	Family	Why you will meet it
Ubuntu / Debian	Debian	Most common on servers and in tutorials.
Kali Linux	Debian	Ships with hundreds of security tools pre-installed.
Fedora / RHEL / Rocky	Red Hat	Enterprise servers.
Alpine	Independant	Tiny, popular inside containers.

The shell

The terminal is the window. The shell is the program inside it that reads your commands. The most common shell is bash (Bourne Again Shell).

Three ideas that explain Linux

1 - Everything is a file. Devices, processes and configuration are all exposed as files or directories.

2 - Small tools that do one thing well, combined with pipes (|) to solve big problems.

3 - Text is the universal interface. Configuration, logs and command output are plain text, so you can search, filter and automate them.

II - First steps in the terminal

Anatomy of a prompt



Your survival kit

Command	What it does
pwd	Print the current working directory
ls	List files (-l details, -a include hidden)
cd <dir>	Change directory
whoami	Who am I ?
man <cmd>	Open the manual page (quit with q)
<cmd> - -help	Short built-in help
clear	Clean the screen (or press Ctrl+L)
history	List previous commands

Mandatory Keyboard shortcuts to know

Keys	Effect
Tab	Auto-complete commands and file names (press twice to list options)
Up / Down keys	Browse command history
Ctrl+R	Reverse-search your history (type part of old command)
Ctrl+C	Stop the running command
Ctrl+A / Ctrl+E	Jump to start / end of line
Ctrl+L	Clear the screen

Practice makes perfect ! Try this:

1 - Run pwd, then ls -la. Which entries start with a dot and why ?

2 - Use man ls to find which option sorts files by modification time.

3 - Type ech and press Tab. Then type ls /us and press Tab Twice.

4 - Run five different commands, then use Ctrl+R to find the first one again

III - The Filesystem

Linux has one single tree starting at / (root). There are no drive letters. Everything, including other disks and USB sticks, is attached somewhere in this tree.

Path	Purpose	Security Relevance
/	Root of the whole tree	
/home	Personal files of each user (/home/xavier)	Where user data and SSH keys live
/root	Home directory of the root user	High-value target
/etc	System-wide configuration files	Misconfigurations are common vulnerabilities
/var/log	Log files	First place to look during an incident
/tmp	Temporary files, world-writable	Attackers drop files here
/bin, /usr/bin	Programs (commands)	Tampering means a compromised system
/sbin, /usr/sbin	Administration programs	
/dev	Devices as files (/dev/null, /dev/sda)	
/proc, /sys	Live view of kernel and processes	Great of investigation
/opt	Optional third-party software	

Absolute vs relative paths

cd /var/log	Absolute path, works from anywhere
cd ../etc	Relative path, from the current location
cd ~	/home/<you>
.	The current directory
..	The parent directory

Practice makes perfect ! Try this:

1 - Starting in your home, reach /etc/ssh using first an absolute, then a relative path.

2 - List the contents of /var/log sorted by size, largest first.

3 - What does ls /dev | head show ? What is /dev/null ?

IV - Working with files and directories

Create, copy, move, delete

Command	What it does
mkdir	Create a directory
cp	Copy a file
touch	Create an empty file
mv	Move a file
rm	Delete a file
rmdir	Delete a directory

> **CAREFUL**
> There is **no recycle bin**. rm is permanent. Never run rm -rf with a path you have not double checked, and especially never with a variable that could empty.

Looking inside files

Command	What it does
cat	Displays the contents of a file.
head	Displays the first 10 lines of a file by default.
tail	Displays the last 10 lines of a file by default.
wc	Counts the number of lines, words, and characters in a file.
nano	Opens a simple, beginner-friendly text editor in the terminal.
vim	Opens a powerful text editor used to create and edit files in the terminal.

Wildcards

Pattern	Matches	Example
*	Any number of characters	ls *.txt
?	Exactly one character	ls file?.txt
[abc]	One of the listed characters	ls file[123].txt
[a-z]	One character in a range	ls [a-z]*.txt
{a,b}	Each alternative	cp file.{txt,pdf} backup/

Finding things

Command	What it does	Example
find	Searches for files and directories based on criteria such as name, type, or size.	find . -name "*.txt"
which	Shows the location of the executable file for a command.	which python
grep	Searches for specific words or patterns within files or command output.	grep "hello" file.txt

> **GOOD TO KNOW**
>
> 2>/dev/null throws away error messages (Like *Permission denied*) so that results stay readable

Practice makes perfect ! Try this:

1 - Create lab/{docs,src,tests} with one command, then create three files inside docs.

2 - Copy /etc/hostname to your lab folder, rename it and read it with cat.

3 - Find every file in /etc that ends with .conf and count them using wc -l.

V - Pipes, redirection and text processing

This is the heart of Linux power: connect small programs so the output of one becomes the input of the next.

Standard streams

Stream	Number	Default	Purpose
stdin	0	keyboard	Input
stdout	1	screen	Normal output
stderr	2	screen	Error messages

Pipes

Pipes (|) in Linux are used to send the output of one command directly as the input of another command. For example cat file.txt | grep "hello" displays only the lines containing hello from file.txt.

The text-processing toolbox

Command	What it does
grep	Keep lines that match
grep -v	Keep lines that do NOT match
cut	Pick columns
sort	Sort lines (-n numeric, -r reverse)
uniq	Remove duplicates (needs sorted input; -c counts)
tr	Translate or delete characters
sed	Search and replace in a stream
awk	Column-aware mini language
head / tail	First / last lines
tee	Write to a file AND pass on
xargs	Turn input lines into arguments

Practice makes perfect ! Try this:

1 - Print only the usernames of /etc/passwd whose login shell is /bin/bash .

2 - Count how many lines in /etc/passwd contain the word nologin .

3 - Show all lines of /etc/ssh/sshd_config that are not comments and not empty.

4 - Using echo , sort and uniq -c , count word frequency in a short text you write.

VI - Users, groups and permissions

Permissions are the first line of defence on a Linux system and a very frequent source of vulnerabilities. Make sure you understand this chapter fully.

Users and groups

Command	What it does
id	Displays the current user's UID, GID, and group memberships.
cat /etc/passwd	Displays the list of user accounts and their basic information (username, UID, home directory, shell, etc.).
sudo cat /etc/shadow	Displays password-related information, including hashed passwords. Requires administrator privileges.
cat /etc/group	Displays the system's groups and their members.
sudo adduser alexia	Creates a new user account named alexia.
sudo usermod -aG sudo alexia	Adds alexia to the sudo group, granting administrator privileges.
sudo passwd alexia	Sets or changes the password for the user alexia.
sudo deluser alexia	Deletes the user account alexia.
su - alexia	Switches to the user alexia, starting a login shell with that user's environment.

UID	Meaning
0	root: the all-powerful administrator
1 to 999	System / service accounts (usually cannot log in)
1000+	Regular human users

Reading permissions (do ls -l)

Letter	Value	On a file	On a directory
r	4	Read the content	List the names inside
w	2	Modify the content	Create / Delete files inside
x	1	Run it as a program	Enter it (cd) and access items

Changing permissions

Command	What it does
chmod 750 report.sh	Sets permissions to rwx r-x ---: owner can read, write, and execute; group can read and execute; others have no permissions.
chmod u+x script.sh	Adds execute permission (x) for the file owner without changing other permissions.
chmod go-rwx secret.txt	Removes all permissions (read, write, execute) from the group (g) and others (o).
sudo chown xavier:staff report.sh	Changes the owner to xavier and the group to staff for report.sh.
umask	Displays the default permission mask used when creating new files and directories.

Number	Symbolic	Typical use
600	rw- --- ---	Private files, SSH Private keys
644	rw- r-- r--	Normal documents
700	rwx --- ---	Private scripts and directories
755	rwx r-x r-x	Public programs and web directories
777	rwx rwx rwx	Avoid. Anyone can change it

Special permission bits

1 - SUID (4755, shown as s in the owner field): the program runs with the rights of its *owner*. passwd uses it legitimately; unexpected SUID root programs are a privilege-escalation risk.

2 - SGID (2755): runs with the group's rights; on directories new files inherit the group.

3 - Sticky bit (1777, shown as t): in a shared directory like /tmp, only a file's owner can delete it.

Practice makes perfect ! Try this:

1 - Create secret.txt, make it readable only by you, and verify with ls -l.

2 - Create a user guest, and try to read your secret as guest. What happens and why?

3 - Convert rwx r-x -w- to numeric form, then 640 to symbolic form.

4 - Run find / -perm -4000 -type f 2>/dev/null. Which of the results do you recognise?

VII - Processes and services

A process is a running program with its own ID (PID). The first process, systemd, has PID 1 and starts everything else.

Useful Commands

Command	What it does
ps aux	Displays all running processes, including their PID, CPU usage, memory usage, and owner.
ps aux | grep ssh	Searches the running processes for entries containing ssh.
top	Displays a real-time, interactive overview of running processes and system resource usage (CPU and memory).
pgrep -a nginx	Finds processes matching nginx and displays their PIDs and full command lines.
kill 1234	Sends the SIGTERM signal to process 1234, requesting it to terminate gracefully.
kill -9 1234	Sends the SIGKILL signal to forcefully terminate process 1234.
pkill firefox	Sends SIGTERM to processes matching the name firefox, requesting them to terminate.

Foreground and background jobs

Command	What it does
sleep 300 &	Starts a process that sleeps for 300 seconds in the background, allowing you to continue using the terminal.
jobs	Lists the jobs running or stopped in the current terminal session.
fg %1	Brings job number 1 from the background into the foreground.
Ctrl+Z	Suspends the currently running foreground process and returns control to the terminal.
nohup ./long.sh &	Runs long.sh in the background and prevents it from being terminated when you log out. Its output is normally redirected to nohup.out.

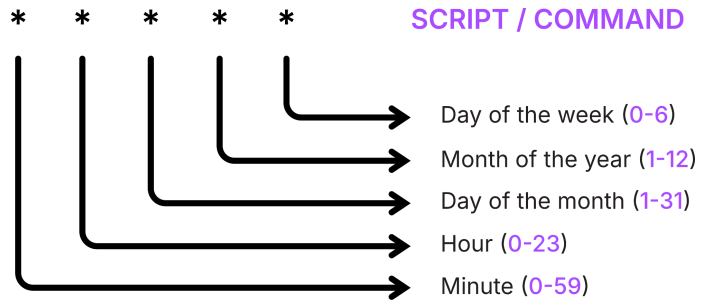
Services with systemd

Command	What it does
systemctl status ssh	Displays the current status of the SSH service (running, stopped, failed, etc.).
sudo systemctl start ssh	Starts the SSH service.
sudo systemctl stop ssh	Stops the SSH service.
sudo systemctl restart ssh	Restarts the SSH service (stops it and starts it again).
journalctl -f	Displays system logs in real time, continuously showing new log entries as they appear.

Scheduling with cron

Cron is a Linux utility that automatically executes commands or scripts at scheduled times (minutes, hours, days, etc.).

Command	What it does
crontab -e	Opens the current user's crontab file to create or edit scheduled tasks.
crontab -r	Removes all scheduled tasks for the current user.



GOOD TO KNOW

Attackers love cron and systemd services for **persistence** (re-starting their access after a reboot). Checking crontab -l and systemctl list-unit-files is part of every investigation

Practice makes perfect ! Try this:

1 - Start sleep 500 & , find its PID with pgrep , then stop it with kill.

2 - Which service listens for SSH on your machine? Is it enabled at boot?

3 - Write a cron line that runs a script every weekday at 18:00.

HPC

Digital Learning Hub_
Academy
Powered by HPC

CS
ACADEMY

PREPARATORY COURSE FOR CYBERSECURITY ACADEMY - GUIDE 2 OF 4

2 Network Fundamentals

Introductory Course to Network Fundamentals

THE THREE GUIDES

1 Linux Fundamentals

2 Network Fundamentals

3 Scripting Fundamentals

4 GitHub

Cybersecurity Academy by Digital Learning Hub

November 2026

I - What is a network ?

A network is two or more devices connected so that they can exchange data. The data travels in small chunks. Everything you do online (a web page, a video call, an e-mail) is cut into chunks, sent, and reassembled at the destination.

Types of network by size

Type	Meaning	Example
PAN	Personal Area Network	Your phone and Bluetooth headphones
LAN	Local Area Network: one building or site	Your home Wi-Fi, the academy lab
MAN	Metropolitan Area Network: a city	A city-wide fibre ring
WAN	Wide Area Network: links LANs across distances	A company's offices linked together
Internet	A global network of networks	All of the above, interconnected

Network devices

Device	Works with	What it does
Hub (obsolete)	Bits	Repeats every signal to every port (no intelligence)
Switch	MAC addresses	Forwards frames only to the right port, it connects devices inside a LAN
Router	IP addresses	Connects different networks and chooses the path for packets
Access point	Wi-Fi	Lets wireless devices join the wired LAN
Firewall	IPs, ports, apps	Allows or blocks traffic according to rules
Modem	Line signal	Converts between your provider's line and digital data
Load balancer / proxy	Applications	Distributes or relays requests to servers

Network devices

In client-server, a client asks and a server answers (your browser asks a web server). In peer-to-peer, every node can be both (file sharing, some chat apps). Most of what you attack and defend is client-server.

Common topologies

Star: all devices connect to a central switch. Most common today, one failing cable affects one device.

Bus / ring: historic. Know the names, not the details.

Mesh: many redundant links, used on the Internet's backbone and in Wi-Fi mesh systems.

Bandwidth, latency and throughput

Bandwidth: the maximum capacity of the link (for example 1 Gbit/s).

Latency: the delay for one packet to arrive (milliseconds).

Throughput: the real data rate you actually get.

II - The OSI and TCP/IP models

Networking is complicated, so it is split into layers. Each layer does one job and relies on the one below. Models give us a shared vocabulary: when someone says layer 2 attack or layer 7 firewall, they mean a specific layer.

#	OSI layer	Job	Examples	Data unit
7	Application	Services used by programs	HTTP, DNS, SMTP, SSH	Data
6	Presentation	Format, encryption, compression	TLS, JPEG, ASCII	Data
5	Session	Open, keep and close conversations	Sessions, RPC	Data
4	Transport	End-to-end delivery, ports	TCP, UDP	Segment / Datagram
3	Network	Addressing and routing	IP, ICMP, routers	Packet
2	Data link	Delivery inside one LAN	Ethernet, MAC, ARP, switches	Frame
1	Physical	Bits on the medium	Cables, Wi-Fi radio, fibre	Bits

The TCP/IP model (what the Internet really uses)

The TCP/IP model is a networking framework that explains how data is transmitted between devices over the Internet through four layers: Application, Transport, Internet, and Network Access.

TCP/IP layer	Covers OSI layers	Typical protocols
Application	5, 6, 7	HTTP, HTTPS, DNS, SSH, SMTP, DHCP
Transport	4	TCP, UDP
Internet	3	IP, ICMP
Network Access (link)	1, 2	Ethernet, Wi-Fi, ARP

Encapsulation: how data is wrapped

When you send a message, each layer adds its own header (like adding envelopes). The receiver peels them off in reverse order (decapsulation).

```
Application     [ HTTP request: GET /index.html ]                                  = data
Transport       [ TCP hdr: src port 51514, dst port 80 ][ HTTP request ]           = segment
Network         [ IP hdr: src 192.168.1.20, dst 93.184.216.34 ][ segment ]         = packet
Data link       [ Eth hdr: src MAC, dst MAC, type 0x0800 ][ packet ][ trailer (FCS) ]  = frame
Physical        0101101001110100...                                                = bits
```

Practice makes perfect ! Try this:

1 - At which OSI layer does a switch work? A router? A firewall that filters by port?

2 - A packet capture shows a source MAC and a destination MAC. At which layer are they?

3 - Name the data unit at layer 4, layer 3 and layer 2.

III - Layer 2: Ethernet, MAC addresses and switches

MAC addresses

A **MAC address** identifies a network card inside the local network. It has 48 bits, written as six hex pairs, for example 3c:52:82:a1:09:f4. The first three pairs identify the manufacturer (OUI). The special MAC ff:ff:ff:ff:ff:ff means broadcast (everyone on the LAN).

```
$ ip link show
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ... state UP
    link/ether 08:00:27:4b:1e:92 brd ff:ff:ff:ff:ff:ff
```

How a switch learns

1 - The switch keeps a **MAC address table**, which MAC is behind which port.

2 - When a frame arrives, it records the source MAC and port.

3 - If the destination MAC is known, the frame goes out of that single port, otherwise it is flooded to all ports.

ARP: from IP address to MAC address

Devices think in IP addresses, but on the wire they need MAC addresses. **ARP** (Address Resolution Protocol) asks: "Who has 192.168.1.1? Tell 192.168.1.20."
The owner answers with its MAC. The answers are cached.

VLANs

A **VLAN** splits one physical switch into several separate logical LANs. Staff, guests and servers can share the same hardware yet remain isolated at layer 2. Devices in different VLANs can only talk through a router or firewall, which is where you can enforce rules. VLANs are a core tool of **network segmentation**.

IV - Layer 3: IP addressing

IPv4 basics

An IPv4 address is 32 bits, written as four decimal numbers (octets) separated by dots: 192.168.10.77. Each octet is 8 bits, so it ranges from 0 to 255.

Binary in 30 seconds

192.168.10.77 = 11000000.10101000.00001010.01001101

128	64	32	16	8	4	2	1	Total
1	**1**	0	0	0	0	0	0	128+64 = **192**
1	0	**1**	0	**1**	0	0	0	128+32+8 = **168**
0	0	0	0	**1**	0	**1**	0	8+2 = **10**
0	**1**	0	0	**1**	**1**	0	**1**	64+8+4+1 = **77**

Network part and host part: the subnet mask

An IP address has two parts: the **network** (which LAN) and the **host** (which device on it). The **subnet mask** shows where the split is. Written as **CIDR prefix /24** it means the **first** 24 bits are the network. **Usable hosts = 2^(host bits) - 2**. One address is the network address (all host bits 0) and one is the broadcast address (all host bits 1).

CIDR	Subnet mask	Host bits	Addresses	Usable hosts
/8	255.0.0.0	24	16,777,216	16,777,214
/16	255.255.0.0	16	65,536	65,534
/24	255.255.255.0	8	256	254
/25	255.255.255.128	7	128	126
/26	255.255.255.192	6	64	62
/27	255.255.255.224	5	32	30
/28	255.255.255.240	4	16	14
/29	255.255.255.248	3	8	6
/30	255.255.255.252	2	4	2

Special and private ranges

Range	Meaning
10.0.0.0/8	Private (RFC 1918): not routed on the Internet
172.16.0.0/12	Private (172.16.x.x to 172.31.x.x)
192.168.0.0/16	Private (home and small office networks)
127.0.0.0/8	Loopback: this very machine (127.0.0.1, localhost)
169.254.0.0/16	Link-local: self-assigned when DHCP fails
224.0.0.0/4	Multicast

IPv6 in brief

- 128-bit addresses written in hex: 2001:db8:85a3::8a2e:370:7334 (:: replaces a run of zero groups).
- Designed to fix IPv4 exhaustion; NAT is not needed because of the enormous address space.
- Every LAN uses a /64. ::1 is loopback, fe80::/10 is link-local.
- Security teams must secure IPv6 too: many networks run it unnoticed alongside IPv4.

Practice makes perfect ! Try this:

1 - Convert 172.20.3.9 to binary, and 11000000.00000000.00000010.00000001 to decimal.

2 - You need a subnet for 50 devices. What is the smallest prefix that fits?

3 - Are 192.168.1.200/24 and 192.168.2.5/24 on the same network?

V - Layer 3: IP addressing

How a host decides where to send a packet

1 - Your computer compares the destination IP to its own network (using the mask):

2 - **Same network**: send directly. ARP for the destination's MAC.

3 - **Different network**: send to the default gateway (your router). ARP for the router's MAC, but keep the destination IP unchanged.

NAT: sharing one public address

Private addresses cannot be routed on the Internet. NAT (Network Address Translation) on your router rewrites the source address of outgoing packets to its own public IP and remembers the mapping (using ports, so it is technically PAT) so that replies return to the right device.

```
Laptop 192.168.1.20:51514  ->  Router (NAT)  ->  203.0.113.5:40001  ->  Web server
Laptop 192.168.1.20:51514  <-  Router (NAT)  <-  203.0.113.5:40001  <-  Web server
```

> **GOOD TO KNOW**
> NAT is not a firewall, but its side effect is useful: unsolicited incoming connections from the Internet have no mapping and are dropped. **Port forwarding** deliberately creates a mapping so that an inside server is reachable, which also creates exposure.

DHCP: automatic configuration

When a device joins a network it needs an IP, mask, gateway and DNS server. **DHCP** provides them in four steps, remembered as **DORA**:

Step	Message	Meaning
1	Discover	Client broadcasts: "Is there a DHCP server?"
2	Offer	Server offers an address and settings
3	Request	Client accepts that offer
4	Acknowledge	Server confirms; the client gets a lease for a period of time

Because the first message is a broadcast and nothing is authenticated, a **rogue DHCP server** can hand out a malicious gateway or DNS. Switches can defend with **DHCP snooping**.

VI - Layer 4: TCP, UDP and ports

IP gets a packet to the right machine. The transport layer gets the data to the right program on that machine, using **port numbers** (0 to 65535). An address + port is called a **socket**: 192.168.1.20:443.

Port Range	Name	Notes
0 - 1023	Well-known	Standard services (HTTP 80, SSH 22). Need root to listen on Linux
1024 - 49151	Registered	Applications, databases
49152 - 65535	Dynamic / ephemeral	Chosen randomly by clients for their end of a connection

TCP vs UDP

	TCP	UDP
Connection	Connection-oriented (handshake first)	Connectionless (just send)
Reliability	Acknowledgements, retransmission, ordering	None: packets may be lost or reordered
Speed / Overhead	Slower, more overhead	Faster, tiny header
Used for	Web, e-mail, SSH, file transfer	DNS queries, video calls, gaming, VoIP, DHCP

The TCP three-way handshake

```
Client                                  Server
  |  ---- SYN  (seq=100) ------------->    |    "I want to talk. Start at 100"
  |  <--- SYN-ACK (seq=300, ack=101) ---   |    "OK. I start at 300"
  |  ---- ACK  (ack=301) ------------->    |    "Great, connected"
  |  ============= data ================   |
  |  ---- FIN ------------------------->   |    closing (FIN / ACK in both directions)
```

TCP flag	Meaning
SYN	Start a connectiona
ACK	Acknowledge received data
FIN	Finish: graceful close

RST	Reset: abort immediately (also the answer from a closed port)
PSH / URG	Push data to the application now / urgent data

GOOD TO KNOW
Port scanners exploit the handshake. A **SYN scan** sends SYN and watches the answer: SYN-ACK = port **open**, RST = port **closed**, silence = probably **filtered** by a firewall.

Ports you must know

Port	Protocol	Service	Security Note
20/21	TCP	FTP	Plain text credentials. Prefer SFTP
22	TCP	SSH	Encrypted remote shell; a top brute-force target
23	TCP	Telnet	Plain text remote shell. Obsolete, never use
25	TCP	SMTP	Mail sending; spam relay risk
53	UDP / TCP	DNS	Name resolution; abused for tunnelling and spoofing
67/68	UDP	DHCP	Server / client
80	TCP	HTTP	Unencrypted web
110 / 143	TCP	POP3 / IMAP	Mail retrieval (use the TLS versions 995 / 993)
123	UDP	NTP	Time sync; abused for amplification DDoS
161	UDP	SNMP	Device management; default community strings
389 / 636	TCP	LDAP / LDAPS	Directory services (Active Directory)
443	TCP	HTTPS	Encrypted web
445	TCP	SMB	Windows file sharing; frequent worm target
3306	TCP	MySQL	Databases should never face the Internet
3389	TCP	RDP	Windows remote desktop; heavily attacked

VII - Layer 7: the protocols you meet every day

DNS: the phone book of the Internet

People use names (**example.com**), computers use IP addresses. **DNS** translates between them. Your computer asks a resolver, which asks the **root servers**, then the **TLD servers** (**.com**), then the domain's **authoritative server**, and caches the answer for the record's **TTL**.

Record	Purpose	Example
A	Name to IPv4 address	dlh.lu → 158.64.182.90
AAAA	Name to IPv6 address	example.com → 2606:2800:220:1:248:1893:25c8:1946
CNAME	Alias to another name	www.example.com → example.com
MX	Mail servers of the domain	example.com → mail.example.com
NS	Authoritative name servers	example.com → ns1.example.com
TXT	Free text (SPF, DKIM, domain verification)	v=spf1 include:_spf.google.com ~all
PTR	Reverse lookup: IP to name	158.64.182.90 → dlh.lu

```
$ dig example.com A +short          # just the answer
$ dig example.com MX                # mail servers
$ dig @8.8.8.8 example.com          # ask a specific resolver
$ dig -x 8.8.8.8 +short             # reverse lookup
$ nslookup example.com              # alternative tool
```

HTTP and HTTPS

HTTP is a text request/response protocol. The browser sends a request, the server replies with a status code and content.

REQUEST (left) / RESPONSE (right)

```
GET /login HTTP/1.1                 HTTP/1.1 200 OK
Host: example.com                   Content-Type: text/html
User-Agent: Mozilla/5.0             Set-Cookie: session=ab12cd...; HttpOnly
Cookie: session=ab12cd...           <html> ... </html>
```

Method	Use	Status	Meaning
GET	Read a resource	200	OK
POST	Send data (forms, logins)	301/302	Redirect

PUT / DELETE	Replace / delete	401/403	
HEAD	Headers only	404	Not Found
OPTIONS	Ask what is allowed	500	Server error

HTTPS is HTTP wrapped in **TLS**, which provides encryption (nobody on the path can read it), **integrity** (nobody can modify it) and **authentication** (the certificate proves you reached the real site). A **certificate** is signed by a trusted **certificate authority (CA)** and binds a domain name to a public key.

Other protocols in one table

SSH	What is it for	Secure alternative
SSH	Remote shell and file transfer (22)	Already secure
FTP	File transfer (21)	SFTP (over SSH) or FTPS
Telnet	Remote shell in clear text	SSH
SMTP / IMAP / POP3	Send and receive e-mail	STARTTLS or implicit TLS ports
SMB	Windows file and printer sharing (445)	Keep off the Internet; SMB 3 encryption
SNMP	Monitoring network devices	SNMPv3
ICMP	Errors and diagnostics (ping)	Filter selectively

What happens when you open https://dlh.lu

1 - Your browser asks DNS for the IP of dlh.lu (UDP 53).

2 - Your computer finds the route; ARP resolves the gateway's MAC.

3 - TCP three-way handshake with port 443.

4 - TLS handshake: certificate check, key exchange, encrypted session.

5 - HTTP request travels inside TLS, the server replies, the page renders.

VIII - Network security concepts

The CIA triad

Property	Goal	Network Example
Confidentiality	Only authorised people can read data	TLS, VPN encryption
Integrity	Data is not altered unnoticed	Hashes, digital signatures, TLS
Availability	Services stay usable	Redundancy, DDoS protection

Defensive building blocks

Control	What it does
Firewall	Allows or denies traffic by rules (source, destination, port, protocol). Stateful firewalls track connections: next-gen firewalls understand applications.
DMZ	A separate segment for Internet-facing servers (web, mail) so a compromise does not expose the internal LAN.
Segmentation / VLANs	Break the network into zones with controlled traffic between them; limits an attacker's movement (lateral movement).
IDS / IPS	Detect (and for IPS, block) suspicious traffic using signatures or behaviour. Examples: Snort, Suricata, Zeek.
VPN	An encrypted tunnel across an untrusted network (WireGuard, OpenVPN, IPsec).
Proxy / reverse proxy / WAF	Intermediaries that filter, cache or protect web traffic.
Zero trust	Never trust by location. Verify every user and device, every time.

Classic attacks, how they work, how to defend (concepts)

Attack	Idea of the attack	Defense
Sniffing	Capturing traffic that is not encrypted	TLS everywhere, VPN, avoid FTP/Telnet/HTTP
ARP spoofing / MITM	Attacker lies in ARP replies to sit between victim and gateway	Dynamic ARP inspection, static entries for critical hosts, TLS and HSTS
DNS spoofing / poisoning	Fake DNS answers lead users to a malicious IP	DNSSEC, trusted resolvers, DoH / DoT
Port scanning	Mapping open services before an attack	Close unused ports, firewall, alert on scans (IDS)
Brute force / credential stuffing	Trying many passwords	Strong passwords, MFA, lockout, fail2ban, keys for SSH
DoS / DDoS	Overwhelming a service with traffic (e.g. SYN flood)	Rate limiting, SYN cookies, CDN / scrubbing, redundancy
Rogue DHCP / rogue AP	Fake infrastructure hands victims a malicious config	DHCP snooping, 802.1X, wireless monitoring
VLAN hopping	Escaping a VLAN through trunk misconfiguration	Disable unused trunks, set native VLAN safely
Phishing + malicious domains	Social engineering via look-alike sites	User awareness, DNS filtering, mail authentication (SPF, DKIM, DMARC)

> **GOOD TO KNOW**
> Understanding an attack is how you learn to detect it. In the academy you will reproduce these in an isolated lab, never on a live network.

Wi-Fi security at a glance

- WEP is broken. WPA2 with a strong passphrase is acceptable. WPA3 is current best practice.
- WPA2/WPA3-Enterprise (802.1X, RADIUS) gives each user their own credentials.
- Disable WPS, change default router passwords, keep firmware updated.

HPC

Digital Learning Hub_
Academy
Powered by HPC

CS ACADEMY

PREPARATORY COURSE FOR CYBERSECURITY ACADEMY - GUIDE 3 OF 4

3 Scripting Fundamentals

Introductory Course to Scripting Fundamentals

THE THREE GUIDES

1 Linux Fundamentals

2 Network Fundamentals

3 Scripting Fundamentals

4 GitHub

Cybersecurity Academy by Digital Learning Hub

November 2026

I - Your first Bash script

A **shell** is a program that reads commands and runs them. **Bash** is the most common shell on Linux. A **Bash script** is a text file containing commands that Bash executes in order: instead of typing the same commands every time, you save them in a file and run that file.

Do not try to memorise every symbol: run the examples, change them, and observe what happens.

By the end, you will be able to write scripts that accept input, make decisions, repeat tasks, process files and report success or failure.

A safe way to practise

- Use a practice directory, such as ~/bash-practice.
- Never test destructive commands on important files. Use sample files you can recreate.
- Read a command before running it, especially anything involving rm, sudo or wildcards.

```
mkdir -p ~/bash-practice
cd ~/bash-practice
```

Create a script

Create a file named hello.sh with nano hello.sh and type the following. The first line is the **shebang**: it tells the system which interpreter must run the file when you execute it directly. In nano, save with Ctrl+O, Enter, then Ctrl+X.

```
#!/usr/bin/env bash
# hello.sh - my first Bash script
echo "Hello, $(whoami)! Today is $(date +%A)."
```

Run the script

Command	What it does
bash hello.sh	Asks Bash to read the file. No execute permission needed.
chmod +x hello.sh	Adds execute permission to the file.
./hello.sh	Runs the file directly, using its shebang. The prefix ./ means "look in the current directory".

> **GOOD TO KNOW**
> Why not just type hello.sh? Linux searches the directories listed in PATH when you type a command, and the current directory is normally not in PATH. This prevents you from accidentally running an unexpected file.

Comments and commands

A line starting with # is a comment (except the shebang on line 1). Bash ignores comments; they explain **why** a script does something.

```
# Display a short status message
echo "The script is running"
```

Where should personal scripts live?

A common place is ~/bin. Create it and add it to your PATH in ~/.bashrc. After that, any executable script placed in ~/bin can be run by name from any directory.

```
mkdir -p "$HOME/bin"
echo 'export PATH="$PATH:$HOME/bin"' >> ~/.bashrc
source ~/.bashrc
```

> **CAREFUL**
> Only add directories you trust to your PATH. Anyone who can write into a directory in your PATH can make you run their program.

II - Variables, quoting and input

What is a variable?

A variable is a named place where the shell keeps a value so you can reuse it. You assign with name=value, and there must be **no spaces** around the equals sign. The dollar sign ($name) asks Bash to substitute the value. Variables are not strongly typed: they usually hold text, and arithmetic contexts interpret suitable values as numbers.

```
name="Alice"
age=30
echo "Name: $name"
echo "Age: $age"
echo "Next year: $((age + 1))"
```

Quoting: a vital habit

Quoting controls whether Bash expands special characters and how it splits text into arguments.

Form	What it does	Example
Double quotes	Expands variables and command substitutions; keeps the result together as one argument.	echo "Hi $name"
Single quotes	Treats almost everything literally; no variable expansion.	echo 'Hi $name'
Backslash	Escapes the special meaning of the next character.	echo \$name

Consider a filename containing a space:

```
file="My Notes.txt"
ls $file     # wrong: expands into two words
ls "$file"   # right: one filename
```

> **GOOD TO KNOW**
> A strong beginner rule: put double quotes around every variable expansion, such as "$file". It prevents unwanted word splitting and pathname expansion.

Read input from the user

read asks the person running the script for information. -r stops backslashes being treated as escape characters and -p displays a prompt.

```
#!/usr/bin/env bash
read -r -p "What is your name? " name
echo "Welcome, $name!"
```

Command substitution

Command substitution captures the output of a command and places it in a variable or another command. The modern form is $(...). The command runs first and its standard output becomes the value (trailing newlines are removed).

```
today=$(date +%F)
hour=$(date +%H)
echo "Today is $today"
echo "The hour is $hour"
```

Arithmetic

Use $((...)) for integer arithmetic: addition, subtraction, multiplication, division and remainder.

```
a=17
b=5
echo "$((a + b))"   # 22
echo "$((a - b))"   # 12
echo "$((a * b))"   # 85
echo "$((a / b))"   # 3 (integer division)
echo "$((a % b))"   # 2 (remainder)
```

Useful parameter expansions

Expression	Meaning
${name:-guest}	Use guest if name is unset or empty; does not change name.
${#name}	Length of the value.
${name^^}	Convert letters to uppercase (Bash feature).
${name,,}	Convert letters to lowercase (Bash feature).
${file%.txt}	Remove a matching .txt suffix.
${file##*.}	Keep the text after the last dot.

III - Arguments and special variables

Pass information when starting a script

Sometimes a script should not ask questions: you give it information on the command line, such as ./greet.sh Alice. Inside the script, $1 is the first argument. If the script is run without an argument, $1 is empty, so you can provide a default.

```
#!/usr/bin/env bash
name=${1:-stranger}
echo "Hello, $name!"
```

The most useful special variables

Variable	Meaning
$0	The name or path used to invoke the script.
$1, $2...	First, second and later positional arguments.
$#	Number of positional arguments.
"$@"	All arguments, each kept as a separate item when quoted.
$?	Exit status of the most recently completed command.
$$	Process ID of the current shell.
$HOME, $USER, $PWD, $PATH	Common environment variables.

IV - Decisions and tests

Make a decision with if

An if statement runs a block of commands only when a test succeeds.

```
if condition; then
  commands
elif another_condition; then
  commands
else
  commands
fi
```

Example: ask for a number and classify it. The test [[...]] is Bash's built-in conditional syntax, and the spaces just inside the brackets are required. This example assumes a valid integer; validating input is a separate task.

```
#!/usr/bin/env bash
read -r -p "Enter a number: " n
if [[ $n -gt 100 ]]; then
  echo "big"
elif [[ $n -gt 10 ]]; then
  echo "medium"
else
  echo "small"
fi
```

Compare numbers and strings

Purpose	Operators	Example
Numeric equality / inequality	-eq / -ne	[[$n -eq 5]]
Numeric greater / less	-gt, -ge, -lt, -le	[[$n -ge 18]]
String equality / inequality	== / !=	[[$name == "Sam"]]
Empty / non-empty string	-z / -n	[[-z $name]]
Pattern match	== with a pattern	[[$file == *.txt]]
Regular expression	=~	[[$n =~ ^[0-9]+$]]

Inside [[...]], quote variables to make the intent clear. For a glob pattern or a regular expression on the right side of == or =~, leave the pattern unquoted when you want it interpreted as a pattern.

Combine conditions

Use && for AND and || for OR inside [[...]]. They can also connect whole commands: the command after && runs only if the previous one succeeds, and the command after || runs only if it fails.

```
if [[ -f "$file" && -r "$file" ]]; then
  echo "This is a readable regular file"
fi
```

A command can be a condition

A command succeeds or fails according to its exit status: 0 means success and any other value means failure. This is why you can place a command directly after if.

```
if ping -c 1 -W 1 192.0.2.1 >/dev/null 2>&1; then
  echo "Host responded"
else
  echo "No response"
fi
```

> **GOOD TO KNOW**
> 192.0.2.1 is reserved for documentation, so it is safe to use in examples. >/dev/null 2>&1 discards both normal output and error output.

V - Loops and reading files

A loop repeats commands. Use one when you want to perform the same operation for several values, files or lines of input.

for: repeat over a list

```
for name in alice bob carol; do
  echo "Hello, $name"
done

for number in {1..5}; do
  echo "Round $number"
done
```

The list after in supplies the values. Bash expands {1..5} into five values before the loop runs.

while and until

A while loop repeats while its condition succeeds. An until loop repeats until its condition succeeds.

```
count=1
while [[ $count -le 3 ]]; do
  echo "count=$count"
  count=$((count + 1))
done

attempt=1
until [[ $attempt -gt 3 ]]; do
  echo "Attempt $attempt"
  attempt=$((attempt + 1))
done
```

Read a file one line at a time

Use while IFS= read -r line to preserve leading and trailing whitespace and backslashes in each line, and redirect the file into the loop.

```
while IFS= read -r line; do
  printf 'Line: %s\n' "$line"
done < users.txt
```

> **CAREFUL**
> Avoid for line in $(cat file) to read lines. Command substitution and word splitting break lines apart at every space and tab.

break and continue

```
for n in {1..10}; do
  [[ $n -eq 3 ]] && continue
  [[ $n -eq 6 ]] && break
  echo "$n"
done
```

continue skips the rest of the current iteration. break exits the loop entirely.

VI - Functions and reusable code

A function is a named group of commands. It avoids repeating code and makes a script easier to read. A function receives arguments just like a script: $1 is its first argument. Use local for variables that must stay inside the function.

```
say_hello() {
  local name=$1
  echo "Hello, $name!"
}

say_hello "Alice"
say_hello "Morgan"
```

Return a status, not printed text

A function can print information and it can return a status. return sets the exit status of the function; it does **not** return a general-purpose string. The final test in the function supplies its success or failure.

```
is_readable_file() {
  [[ -f "$1" && -r "$1" ]]
}

if is_readable_file "/etc/passwd"; then
  echo "File is readable"
else
  echo "File is not readable"
fi
```

> **GOOD TO KNOW**
> A useful pattern: write small functions that answer one clear yes/no question, then combine them with if.

VII - Input, output, redirection and pipes

The three standard streams

Stream	Name	Default destination
0	stdin (standard input)	Keyboard or redirected input
1	stdout (standard output)	Terminal
2	stderr (standard error)	Terminal

Redirect output

Syntax	What it does
command > file	Send stdout to a file (creates or overwrites it).
command >> file	Append stdout to the end of a file.
command < file	Read stdin from a file.
command 2> file	Send stderr to a file.
command > file 2>&1	Send stdout and stderr to the same file.

```
echo "Report" > report.txt          # create / overwrite
echo "Another line" >> report.txt   # append
sort < names.txt                    # read input from a file
ls missing-file 2> errors.txt       # send errors to a file
```

> **CAREFUL**
> > overwrites the destination file without asking. Use >> to append.

Pipes

A pipe sends the standard output of one command to the standard input of another, so you can build a workflow from small tools. Read a pipeline from left to right: the first command produces data and the next receives it. A pipe is not just a visual separator: it connects processes.

```
cut -d: -f1 /etc/passwd | sort
grep -i "error" app.log | wc -l
ps aux | grep ssh
```

Discard output

/dev/null is a special destination that discards everything written to it. The first line below discards normal output; the second discards both normal output and errors.

```
command >/dev/null
command >/dev/null 2>&1
```

HPC

Digital Learning Hub_
Academy
Powered by HPC

CS ACADEMY

PREPARATORY COURSE FOR CYBERSECURITY ACADEMY - GUIDE 4 OF 4

4 GitHub

Introductory Course to GitHub

THE THREE GUIDES

1 Linux Fundamentals

2 Network Fundamentals

3 Scripting Fundamentals

4 GitHub

I - What Is GitHub

Git is a version control system: a program that records every change made to your files so you can go back in time, compare versions and work in parallel. **GitHub** is an online platform that hosts Git repositories and adds collaboration tools on top of them. GitLab and Bitbucket are similar platforms.

Tool	Where it runs	What it does
Git	Your computer	Tracks changes in your files and stores the history locally.
GitHub	The cloud (github.com)	Hosts your repositories online, shares them and lets teams collaborate.

Three ideas that explain GitHub

1 - Store your code safely in the cloud. If your computer breaks, your work is still on GitHub.

2 - Work together through branches and pull requests, without overwriting each other.

3 - Track every change. Each modification is saved with an author, a date and a message.

Vocabulary you will meet

Term	Meaning
repository (repo)	A project folder whose history is tracked by Git.
commit	A saved snapshot of your changes, with a message describing them.
branch	An independent line of work inside a repository (often main or master).
pull request	A request to merge your branch into another one, reviewed by others.
clone	Download a copy of a remote repository to your machine.
remote / origin	The online copy of your repository. origin is its default nickname.
push	Send your local commits to the remote repository.

What this project teaches you

This project is a foundation for future assessments. By the end you will be able to set up a new repository on GitHub, then create and organize a simple project with folders and files.

II - Getting Started

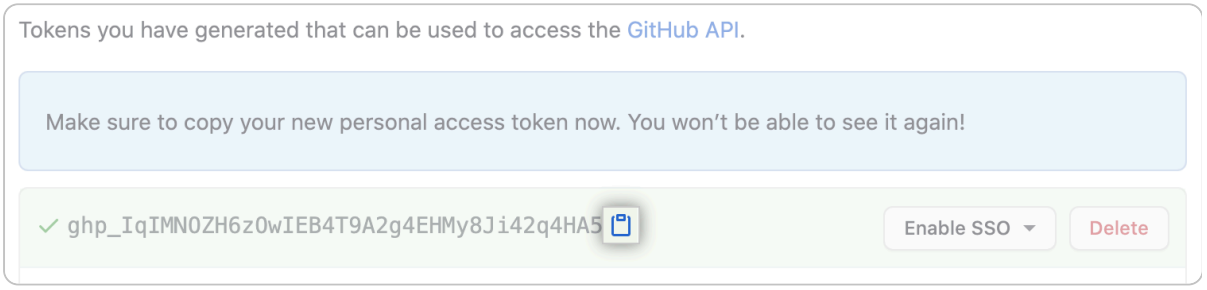
Step 0 - Create an account on GitHub

You need a GitHub account for all your projects. Creating one is free.

Step 1 - Create a Personal Access Token

To access your repositories and authenticate yourself from the terminal, you need a Personal Access Token (PAT). GitHub no longer accepts your account password for Git operations over HTTPS, so the token takes its place. Create it from your GitHub settings under Developer settings, Personal access tokens.

Once created, the token is displayed only one time. It starts with ghp_ followed by a long random string, as in the example below.



GitHub shows the new token once: copy it immediately

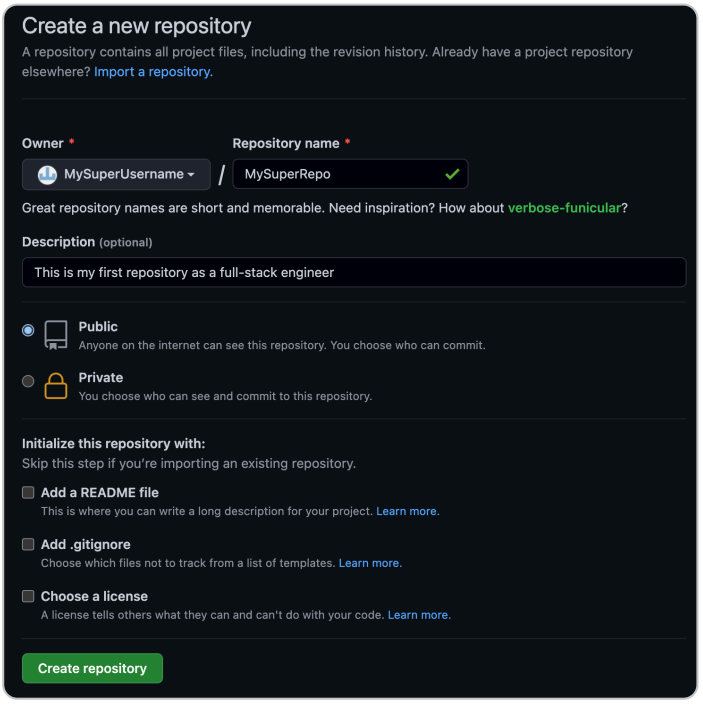
> **CAREFUL**
> A token is a **password**. Never share it, never write it in a file you commit and never post a screenshot of a real one. If it leaks, delete it in GitHub settings and create a new one. Also note that a token typed in a clone URL is saved in the .git/config file of the repository and displayed by git remote -v.

III - Your First Repository

Create it on the GitHub website

Use the graphic interface on github.com (the **+** menu, then **New repository**) and fill in the form exactly as follows.

Field	Value
Owner	Your own username
Repository name	The name given at the bottom of the task. Pay attention to it: it must match exactly.
Visibility	Public
Add a README file	Leave unchecked
Add .gitignore	Leave unchecked
Choose a license	Leave unchecked



The "Create a new repository" form. Click Create repository when done

> **CAREFUL**
> Do not tick README, .gitignore or license. The repository must start completely empty, otherwise the first push may be rejected.

IV - Clone and First Commit

Using the sandbox (terminal or VSCode) and your personal access token, you can now clone the repository and work on it. To validate your work, you will create a folder my-first-project containing a file README.md.

> **GOOD TO KNOW**
> Cloning an empty repository prints the warning *You appear to have cloned an empty repository*. This is normal: that is exactly what section III asked for.

Replace the placeholders

Placeholder	Replace with
{YOUR_PERSONAL_TOKEN}	The token from Step 1
{YOUR_USERNAME}	Your GitHub username from Step 0
{YOUR_REPO}	The repository name given at the bottom of the task

The workflow, step by step

#	Goal	Command
1	Clone your repository	git clone https://{YOUR_PERSONAL_TOKEN}@github.com/{YOUR_USERNAME}/{YOUR_REPO}.git
2	Move into the new directory	cd {YOUR_REPO}
3	Create the folder	mkdir my-first-project
4	Create README.md with the content My first readme	echo 'My first readme' > my-first-project/README.md
5	Check the content	cat my-first-project/README.md
6	Stage the new file	git add .
7	Commit with the required message	git commit -m 'My first commit'
8	Push to the remote server (origin)	git push

V - Pushing Your Work

The commit and the push

```
root@896cf839cf9a:/{YOUR_REPO}# git commit -m 'My first commit'
[master (root-commit) 98eef93] My first commit
 1 file changed, 1 insertion(+)
 create mode 100644 my-first-project/README.md
root@896cf839cf9a:/{YOUR_REPO}# git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 212 bytes | 212.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/{YOUR_USERNAME}/{YOUR_REPO}.git
 * [new branch]      main -> main
```

Good job! You pushed your first file in your first repository. Open your repository on GitHub and check that the folder my-first-project and its README.md are there.

The journey of a change

Area	Moved there with	What it means
Working directory	edit / echo / mkdir	The files you see and modify on your machine.
Staging area	git add	The changes you selected for the next snapshot.
Local repository	git commit	The snapshot is saved in your local history.
Remote (GitHub)	git push	The snapshot is uploaded and visible online.

> **GOOD TO KNOW**
> Your branch may be called master or main depending on your Git configuration. If Git asks who you are when you commit, set your identity first with git config --global user.name and git config --global user.email.

VI - Git Survival Kit

Everyday commands

Command	What it does
git status	Shows modified, staged and untracked files.
git add <file>	Stages a file for the next commit (git add . stages everything).
git commit -m "message"	Saves the staged changes with a message.
git log --oneline	Lists previous commits, one per line.
git diff	Shows the changes not yet staged.
git push	Sends local commits to the remote repository.
git pull	Downloads and merges new commits from the remote.
git remote -v	Displays the remote repositories linked to this one.