VOID

SYSTEM ADMINISTRATION

Lab Configuration & Implementation

Sprint 2 Technical Report

Yahya El Omari

Encadré par: Mr. Hamza Bahlaouane

February 2, 2026

CONTENTS

INTRODUCTION

This report documents the configuration and administration of a Linux environment (VM) as part of Sprint 2. The objectives covered include:

- System Setup and SSH Configuration
- SSL Certificate Analysis
- Automation with Cronjobs
- Users and Permissions Management
- Webserver (Apache) Deployment

The following sections detail the methodology, observations, and evidence of successful implementation.

SSH CONFIGURATION

2.1 OVERVIEW

Secure Shell (SSH) was configured to allow secure, passwordless access to the Virtual Machine. This involved user creation, RSA key pair generation, and SSH agent configuration.

2.2 IMPLEMENTATION STEPS

2.2.1 VAGRANT SSH CONFIGURATION PARAMETERS DISPLAY

We started by running `vagrant ssh-config` to discover the SSH connection parameters for our VM. This gave us critical information like the connection port (2222), hostname (127.0.0.1), and the identity file path needed for authentication.

```
mac@MacBook-Pro-van-mac vm % vagrant ssh-config
Host default
  HostName 127.0.0.1
  User vagrant
  Port 2222
  UserKnownHostsFile /dev/null
  StrictHostKeyChecking no
  PasswordAuthentication no
  IdentityFile /Users/mac/dev/labs/vm/.vagrant/machines/default/vmware_desktop/private_key
  IdentitiesOnly yes
  LogLevel FATAL
  PubkeyAcceptedKeyTypes +ssh-rsa
  HostKeyAlgorithms +ssh-rsa
```

Figure 2.1: Vagrant SSH Configuration Parameters Display

2.2.2 USER ACCOUNT CREATION WITH USERADD AND PASSWORD CONFIGURATION

After connecting to the VM, we created a new user named 'alpha' using `sudo useradd -m -s /bin/bash alpha`. The `-m` flag created a home directory, and `-s /bin/bash` set bash as the default shell. We then set a simple password using `echo "alpha:123456" | sudo chpasswd` for initial testing.

```
mac@MacBook-Pro-van-mac vm % vagrant ssh
Linux debian-11 5.10.0-31-amd64 #1 SMP Debian 5.10.221-1 (2024-07-14) x86_64

This system is built by the Bento project by Chef Software
More information can be found at https://github.com/chef/bento

Use of this system is acceptance of the OS vendor EULA and License Agreements.

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
vagrant@debian-11:~$ pwd
/home/vagrant
vagrant@debian-11:~$ sudo useradd -m -s /bin/bash alpha
vagrant@debian-11:~$ echo "alpha:123456" | sudo chpasswd
vagrant@debian-11:~$ exit
logout
mac@MacBook-Pro-van-mac vm %
```

Figure 2.2: User Account Creation with useradd and Password Configuration

2.2.3 USER IDENTITY VERIFICATION VIA ID COMMAND

To verify our user creation was successful, we logged in as alpha and ran the `id` command. This confirmed that the alpha user was properly created with UID 1001 and the correct group membership.

```
mac@MacBook-Pro-van-mac vm % ssh alpha@127.0.0.1 -p 2222
alpha@127.0.0.1's password:
Linux debian-11 5.10.0-31-amd64 #1 SMP Debian 5.10.221-1 (2024-07-14) x86_64

This system is built by the Bento project by Chef Software
More information can be found at https://github.com/chef/bento

Use of this system is acceptance of the OS vendor EULA and License Agreements.

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
alpha@debian-11:~$ id
uid=1001(alpha) gid=1001(alpha) groups=1001(alpha)
```

Figure 2.3: User Identity Verification via id Command

2.2.4 RSA KEY PAIR GENERATION WITH 1024-BIT ENCRYPTION

We generated a new RSA key pair specifically for the alpha user using `ssh-keygen -t rsa -b 1024 -f ~/.ssh/alphaKeys -N ""`. We chose 1024-bit encryption and skipped the passphrase for convenience in this lab environment.

```
mac@MacBook-Pro-van-mac vm % ssh-keygen -t rsa -b 1024 -f ~/.ssh/alphaKeys -N ""
Generating public/private rsa key pair.
Your identification has been saved in /Users/mac/.ssh/alphaKeys
Your public key has been saved in /Users/mac/.ssh/alphaKeys.pub
The key fingerprint is:
SHA256:O/KOsak+R8a2gpsAY+Qw0TGHP/jjHoN4nMFLqPyNDyA mac@MacBook-Pro-van-mac.local
The key's randomart image is:
+---[RSA 1024]----+
|..oo.            |
| .oo             |
|o. o             |
|++. o            |
|E++. o  S        |
|*=.=o =  .       |
|+.*+o=o.o        |
| oo.BooB .       |
|  o*=B+.o        |
+----[SHA256]-----+
```

Figure 2.4: RSA Key Pair Generation with 1024-bit Encryption

2.2.5 SSH AGENT CONFIGURATION AND KEY REGISTRATION

We set up the SSH agent to manage our keys automatically. First, we started the agent with `eval "$(ssh-agent -s)"`, then added our new key using `ssh-add ~/.ssh/alphaKeys`. We verified the key was loaded by listing all keys with `ssh-add -l`, which showed our key's SHA256 fingerprint.

```
mac@MacBook-Pro-van-mac vm % eval "$(ssh-agent -s)"
Agent pid 9650
mac@MacBook-Pro-van-mac vm % ssh-add ~/.ssh/alphaKeys
Identity added: /Users/mac/.ssh/alphaKeys (mac@MacBook-Pro-van-mac.local)
mac@MacBook-Pro-van-mac vm % ssh-add -l
1024 SHA256:O/KOsak+R8a2gpsAY+Qw0TGHP/jjHoN4nMFLqPyNDyA mac@MacBook-Pro-van-mac.local (RSA)
```

Figure 2.5: SSH Agent Configuration and Key Registration

2.2.6 SUCCESSFUL PASSWORDLESS SSH AUTHENTICATION TEST

Finally, we tested our passwordless authentication by connecting with `ssh alpha@127.0.0.1 -p 2222`. The connection succeeded without asking for a password, confirming that our SSH key setup was working properly.

```
mac@MacBook-Pro-van-mac vm % ssh alpha@127.0.0.1 -p 2222
Linux debian-11 5.10.0-31-amd64 #1 SMP Debian 5.10.221-1 (2024-07-14) x86_64

This system is built by the Bento project by Chef Software
More information can be found at https://github.com/chef/bento

Use of this system is acceptance of the OS vendor EULA and License Agreements.

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Feb  2 14:08:43 2026 from 172.16.160.1
alpha@debian-11:~$ exit
logout
Connection to 127.0.0.1 closed.
```

Figure 2.6: Successful Passwordless SSH Authentication Test

SSL CERTIFICATE ANALYSIS

3.1 CERTIFICATE INSPECTION

We investigated SSL/TLS configurations using various OpenSSL commands and badssl.com test servers to understand certificate validation, expiration handling, and troubleshooting techniques.

3.1.1 CURL TLS HANDSHAKE FAILURE ON EXPIRED CERTIFICATE

We tested SSL certificate validation by attempting to connect to `expired.badssl.com` using `curl -v`. As expected, the TLS handshake failed because the server's certificate had expired. This demonstrated how curl properly validates certificates and rejects expired ones.

```
mac@MacBook-Pro-van-mac vm % curl -v https://expired.badssl.com/
*   Trying 104.154.89.105:443...
* Connected to expired.badssl.com (104.154.89.105) port 443 (#0)
* ALPN: offers h2,http/1.1
* (304) (OUT), TLS handshake, Client hello (1):
*  CAfile: /etc/ssl/cert.pem
*  CApath: none
* (304) (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS alert, certificate expired (557):
* SSL certificate problem: certificate has expired
* Closing connection 0
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS alert, certificate expired (557):
curl: (60) SSL certificate problem: certificate has expired
More details here: https://curl.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
mac@MacBook-Pro-van-mac vm % curl -v https://self-signed.badssl.com/
*   Trying 104.154.89.105:443...
* Connected to self-signed.badssl.com (104.154.89.105) port 443 (#0)
* ALPN: offers h2,http/1.1
* (304) (OUT), TLS handshake, Client hello (1):
*  CAfile: /etc/ssl/cert.pem
*  CApath: none
* (304) (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS alert, unknown CA (560):
* SSL certificate problem: self signed certificate
* Closing connection 0
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS alert, unknown CA (560):
curl: (60) SSL certificate problem: self signed certificate
More details here: https://curl.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
```

Figure 3.1: curl TLS Handshake Failure on Expired Certificate

3.1.2 OPENSSL CERTIFICATE CHAIN VERIFICATION AND ERROR ANALYSIS

We dove deeper into certificate validation using `openssl s_client -connect expired.badssl.com:4`
This showed us the complete certificate chain and the specific verification errors,
helping us understand exactly why the certificate was rejected.

```
mac@MacBook-Pro-van-mac vm % openssl s_client -connect expired.badssl.com:443
CONNECTED(00000005)
depth=0 C = US, ST = California, L = San Francisco, O = BadSSL Fallback. Unknown subdomain or no SNI., CN = badssl-fallback-unknown-subdomain-or-no-sni
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = US, ST = California, L = San Francisco, O = BadSSL Fallback. Unknown subdomain or no SNI., CN = badssl-fallback-unknown-subdomain-or-no-sni
verify error:num=21:unable to verify the first certificate
verify return:1
write W BLOCK
---
Certificate chain
 0 s:/C=US/ST=California/L=San Francisco/O=BadSSL Fallback. Unknown subdomain or no SNI./CN=badssl-fallback-unknown-subdomain-or-no-sni
   i:/C=US/ST=California/L=San Francisco/O=BadSSL/CN=BadSSL Intermediate Certificate Authority
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIE8DCCAtigAwIBAgIJAM28Wkrsl2exMA0GCSqGSIb3DQEBCwUAMH8xCzAJBgNV
BAYTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRYwFAYDVQQHDA1TYW4gRnJhbmNp
c2NvMQ8wDQYDVQQKDAZCYWRTU0wxMjAwBgNVBAMMKUJhZFNTTCBJbnRlcm1lZGlh
dGUgQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTE2MDgwODIxMTcwNVoXDTE4MDgw
ODIxMTcwNVowgagxCzAJBgNVBAYTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRYw
FAYDVQQHDA1TYW4gRnJhbmNpc2NvMTYwNAYDVQQKDC1CYWRTU0wgRmFsbGJhY2su
IFVua25vd24gc3ViZG9tYWluIG9yIG5vIFNOSS4xNDAyBgNVBAMMK2JhZHNzbC1m
YWxsYmFjay11bmtub3duLXN1YmRvbWFpbi1vci1uby1zbmkwggEiMA0GCSqGSIb3
DQEBAQUAA4IBDwAwggEKAoIBAQDCBOz4jO4EwrPYUNVwWMyTGOtcqGhJsCK1+ZWe
sSssdj5swEtgTEzqsrTAD4C2sPlyyYYC+VxBXRMrf3HES7zplC5QN6ZnHGGM9kFC
xUbTFocnn3TrCp0RUiYhc2yETHlV5NFr6AY9SBVSrbMo26r/bv9glUp3aznxJNEx
tt1NwMT8U7ltQq21fP6u9RXSM0jnInHHwhR6bCjqN0rf6my1crR+WqIW3GmxV0Tb
ChKr3sMPR3RcQSLhmvkbk+atIgYplrG6SRwMJ56j+4v3QHIArJII2YxXhFQBBcvm
/mtUmEAnhccQu3Nw72kYQQdFVXz5ZD89LMQpfOuTGkyG0cqFAgMBAAGjRTBDMAkG
A1UdEwQCMAAwNgYDVR0RBC8wLYIrYmFkc3NsLWZhbGxiYWNrLXVua25vd24tc3Vi
ZG9tYWluLW9yLW5vLXNuaTANBgkqhkiG9w0BAQsFAAOCAgEAsuFs0K86D2IB20nB
QNb+4vs2Z6kECmVUuD0vEUBR/dovFE4PfzTr6uUwRoRdjToewx9VCwvTL7toq3dd
oOwHakRjoxvq+lkvPq+0FMTlKYRjOL6Cq3wZNcsyiTYr7odyKbZs383rEBbcNu0N
c666/ozs4y4W7ufeMFrKak9UenrrPlUe0nrEHV3IMSF32iV85nXm95f7aLFvM6Lm
EzAGgWopuRqD+J0QEt3WNODWqBSZ9EYyx9121+KI1QcMalG20QXuxDNHmTEzMaCj
4Zl8k0szexR8rbcQEgJ9J+izxsecLRVp70siGEYDkhq0DgIDOjmmu8ath4yznX6A
pYEGtYTDUxIvsWxwkraBBJAfVxkp2OSg7DiZEVlMM8QxbSeLCz+63kE/d5iJfqde
cGqX7rKEsVW4VLfHPF8sfCyXVi5sWrXrDvJm3zx2b3XToU7EbNONO1C85NsUOWy4
JccoiguV8V6C723IgzkSgJMlpblJ6FVxC6ZX5XJ0ZsMI9TIjibM2L1Z9DkWRCT6D
QjuKbYUeURhScofQBiIx73V7VXnFoc1qHAUd/pGhfkCUnUcuBV1SzCEhjiwjnVKx
HJKvc9OYjJD0ZuvZw9gBrY7qKyBX8g+sglEGFNhruH8/OhqrV8pBXX/EWY0fUZTh
iywmc6GTT7X94Ze2F7iB45jh7WQ=
-----END CERTIFICATE-----
subject=/C=US/ST=California/L=San Francisco/O=BadSSL Fallback. Unknown subdomain or no SNI./CN=badssl-fallback-unknown-subdomain-or-no-sni
issuer=/C=US/ST=California/L=San Francisco/O=BadSSL/CN=BadSSL Intermediate Certificate Authority
---
```

Figure 3.2: OpenSSL Certificate Chain Verification and Error Analysis

3.1.3 SSL CONNECTION WITH SERVER NAME INDICATION (SNI)

We refined our test by adding the `-servername` flag to properly support Server Name Indication (SNI). This is important for servers hosting multiple SSL certificates on the same IP address.

```
mac@MacBook-Pro-van-mac vm % openssl s_client -connect expired.badssl.com:443 -servername expired.badssl.com
CONNECTED(00000005)
depth=2 C = GB, ST = Greater Manchester, L = Salford, O = COMODO CA Limited, CN = COMODO RSA Certification Authority
verify return:1
depth=1 C = GB, ST = Greater Manchester, L = Salford, O = COMODO CA Limited, CN = COMODO RSA Domain Validation Secure Server CA
verify return:1
depth=0 OU = Domain Control Validated, OU = PositiveSSL Wildcard, CN = *.badssl.com
verify error:num=10:certificate has expired
notAfter=Apr 12 23:59:59 2015 GMT
verify return:1
depth=0 OU = Domain Control Validated, OU = PositiveSSL Wildcard, CN = *.badssl.com
notAfter=Apr 12 23:59:59 2015 GMT
verify return:1
write W BLOCK
---
Certificate chain
 0 s:/OU=Domain Control Validated/OU=PositiveSSL Wildcard/CN=*.badssl.com
   i:/C=GB/ST=Greater Manchester/L=Salford/O=COMODO CA Limited/CN=COMODO RSA Domain Validation Secure Server CA
 1 s:/C=GB/ST=Greater Manchester/L=Salford/O=COMODO CA Limited/CN=COMODO RSA Domain Validation Secure Server CA
   i:/C=GB/ST=Greater Manchester/L=Salford/O=COMODO CA Limited/CN=COMODO RSA Certification Authority
 2 s:/C=GB/ST=Greater Manchester/L=Salford/O=COMODO CA Limited/CN=COMODO RSA Certification Authority
   i:/C=SE/O=AddTrust AB/OU=AddTrust External TTP Network/CN=AddTrust External CA Root
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIFSzCCBDOgAwIBAgIQSueVSfqavj8QDxekeOFpCTANBgkqhkiG9w0BAQsFADCB
kDELMAkGA1UEBhMCR0IxGzAZBgNVBAgTEkdyZWF0ZXIgTWFuY2hlc3RlcjEQMA4G
A1UEBxMHU2FsZm9yZDEaMBgGA1UEChMRQ09NT0RPIENBIExpbWl0ZWQxNjA0BgNV
BAMTLUNPTU9ETyBSU0EgRG9tYWluIFZhbGlkYXRpb24gU2VjdXJlIFNlcnZlciBD
QTAeFw0xNTA0MDkwMDAwMDBaFw0xNTA0MTIyMzU5NTlaMFkxITAfBgNVBAsTGERv
bWFpbiBDb250cm9sIFZhbGlkYXRlZDEdMBsGA1UECxMUUG9zaXRpdmVTU0wgV2ls
ZGNhcmQxFTATBgNVBAMUDCouYmFkc3NsLmNvbTCCASIwDQYJKoZIhvcNAQEBBQAD
ggEPADCCAQoCggEBAMIE7PiM7gTCs9hQ1XBYzJMY61yoaEmwIrX5lZ6xKyx2PmzA
S2BMTOqytMAPgLaw+XLJhgL5XEFdEyt/ccRLvOmULlA3pmccYYz2QULFRtMWhyef
dOsKnRFSJiFzbIRMeVXk0WvoBj1IFVKtsyjbqv9u/2CVSndrOfEk0TG23U3AxPxT
uW1CrbV8/q71FdIzSOciccfCFHpsKOo3St/qbLVytH5aohbcabFXRNsKEqveww9H
dFxBIuGa+RuT5q0iBikusbpJHAwnnqP7i/dAcgCskgjZjFeEU4EFy+b+a1SYQCeF
xxC7c3DvaRhBB0VVfPlkPz0sw6l865MaTIbRyoUCAwEAAaOCAdUwggHRMB8GA1Ud
IwQYMBaAFJCvajqUWgvYkOoSVnPfQ7Q6KNrnMB0GA1UdDgQWBBSd7sF7gQs6R2lx
GH0RN5O8pRs/+zAOBgNVHQ8BAf8EBAMCBaAwDAYDVR0TAQH/BAIwADAdBgNVHSUE
FjAUBggrBgEFBQcDAQYIKwYBBQUHAwIwTwYDVR0gBEgwRjA6BgsrBgEEAbIxAQIC
BzArMCkGCCsGAQUFBwIBFh1odHRwczovL3NlY3VyZS5jb21vZG8uY29tL0NQUzAI
BgZngQwBAgEwVAYDVR0fBE0wSzBJoEegRYZDaHR0cDovL2NybC5jb21vZG9jYS5j
b20vQ09NT0RPUlNBRG9tYWluVmFsaWRhdGlvblNlY3VyZVNlcnZlckNBLmNybDCB
hQYIKwYBBQUHAQEEeTB3ME8GCCsGAQUFBzAChkNodHRwOi8vY3J0LmNvbW9kb2Nh
LmNvbS9DT01PRE9SU0FEb21haW5WYWxpZGF0aW9uU2VjdXJlU2VydmVyQ0EuY3J0
MCQGCCsGAQUFBzABhhhodHRwOi8vb2NzcC5jb21vZG9jYS5jb20wIwYDVR0RBBww
GoIMKi5iYWRzc2wuY29tggpiYWRzc2wuY29tMA0GCSqGSIb3DQEBCwUAA4IBAQBq
evHa/wMHcnjFZqFPRkMOXxQhjHUa6zbgH6QQFezaMyV8O7UKxwE4PSf9WNnM6i1p
OXy+l+8L1gtY54x/v7NMHfO3kICmNnwUW+wHLQI+G1tjWxWrAPofOxkt3+IjEBEH
fnJ/4r+3ABuYLyw/zoWaJ4wQIghBK4o+gk783SHGVnRwpDTysUCeK1iiWQ8dSO/r
ET7BSp68ZVVtxqPv1dSWzfGuJ/ekVxQ8lEEFeouhN0fX9X3c+s5vMaKwjOrMEpsi
8TRwz311SotoKQwe6Zaoz7ASH1wq7mcvf71z81oBIgxw+s1F73hczg36TuHvzmWf
RwxPuzZEaFZcVlmtqoq8
-----END CERTIFICATE-----
subject=/OU=Domain Control Validated/OU=PositiveSSL Wildcard/CN=*.badssl.com
issuer=/C=GB/ST=Greater Manchester/L=Salford/O=COMODO CA Limited/CN=COMODO RSA Domain Validation Secure Server CA
---
```

Figure 3.3: SSL Connection with Server Name Indication (SNI)

3.2 UNDERSTANDING SSL CERTIFICATE ERRORS

During our testing, we encountered various SSL certificate errors which helped us understand the certificate validation process:

3.2.1 CERTIFICATE TRUST CHAIN

When analyzing the expired certificate, we observed:

- Subject: The website (badssl-fallback-unknown-subdomain-or-no-sni)
- Issuer: The one who signed it (BadSSL Intermediate Certificate Authority)

Our system (MacBook) considers the issuer to not be a legitimate authority like it would recognize established Certificate Authorities such as VeriSign or DigiCert. This is why the connection fails - the certificate chain cannot be verified back to a trusted root certificate authority.

3.2.2 SERVER NAME INDICATION (SNI)

We learned that specifying the `-servername` parameter resolves certain SSL handshake issues. SNI allows the client to indicate which hostname it's attempting to connect to at the start of the TLS handshake. This is particularly important for servers hosting multiple SSL certificates on the same IP address.

3.2.3 SOLUTIONS TO CERTIFICATE ERRORS

When encountering SSL certificate errors, we have several options:

For Testing/Development:

- Use `curl -k` or `curl --insecure` to bypass certificate verification (not recommended for production)
- Add `-k` flag to skip SSL verification temporarily

For Production:

- Install the certificate authority's root certificate in your system's trust store
- Obtain a valid certificate from a trusted CA (like Let's Encrypt, DigiCert, or VeriSign)
- Ensure certificates are renewed before expiration
- For expired certificates, contact the site administrator to renew them

For Self-Signed Certificates:

- Export the certificate and manually add it to your system's trusted certificates
- On macOS: Import into Keychain Access and mark as trusted
- On Linux: Copy to `/usr/local/share/ca-certificates/` and run `update-ca-certificates`

AUTOMATION WITH CRONJOBS

4.1 OBJECTIVE

A cronjob was created to automate the cleanup of temporary files. The requirement was to delete files older than 7 days in the ‘ /temp‘ directory.

4.2 IMPLEMENTATION & VERIFICATION

4.2.1 GEANY TEXT EDITOR INSTALLATION VIA HOMEBREW

Before writing our cleanup script, we needed a text editor. We installed Geany on macOS using `brew install --cask geany`, which would allow us to write and edit our shell scripts comfortably.

```
mac@MacBook-Pro-van-mac vm % mkdir ~/temp
mac@MacBook-Pro-van-mac vm % touch ~/temp/file{1..10}.txt
mac@MacBook-Pro-van-mac vm % pwd
/Users/mac/dev/labs/vm
mac@MacBook-Pro-van-mac vm % geany clean_temp.sh
zsh: command not found: geany
mac@MacBook-Pro-van-mac vm % brew install --cask geany
✔ JSON API cask.jws.json
✔ JSON API formula.jws.json
==> Downloading https://download.geany.org/geany-2.1_osx.dmg
######################################################################################
==> Installing Cask geany
==> Moving App 'Geany.app' to '/Applications/Geany.app'
🍺  geany was successfully installed!
Warning: You are using macOS 13.
We (and Apple) do not provide support for this old version.
You may have better luck with MacPorts which supports older versions of macOS:
  https://www.macports.org

This is a Tier 3 configuration:
  https://docs.brew.sh/Support-Tiers#tier-3
You can report Tier 3 unrelated issues to Homebrew/* repositories!
Read the above document before opening any issues or PRs.

mac@MacBook-Pro-van-mac vm % echo 'alias geany="open -a Geany"' >> ~/.zshrc
mac@MacBook-Pro-van-mac vm % source ~/.zshrc
```

Figure 4.1: Geany Text Editor Installation via Homebrew

4.2.2 TEMPORARY DIRECTORY AND TEST FILE CREATION

We created a test environment by making a `temp` directory and populating it with 10 dummy files using `touch ~/temp/file{1..10}.txt`. This gave us a safe space to test our cleanup script without risking real files.

```
alpha@debian-11:~$ pwd
/home/alpha
alpha@debian-11:~$ mkdir -p /home/alpha/temp
alpha@debian-11:~$ touch /home/alpha/temp/file{1..10}.txt
alpha@debian-11:~$ ls
temp
alpha@debian-11:~$ ls -l /home/alpha/temp
total 0
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file1.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file10.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file2.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file3.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file4.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file5.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file6.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file7.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file8.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file9.txt
```

Figure 4.2: Temporary Directory and Test File Creation

4.2.3 FILE TIMESTAMP MANIPULATION FOR CLEANUP SCRIPT TESTING

To test our cleanup logic, we needed files that appeared old. We used `touch -d "10 days ago" /home/alpha/temp/file1.txt` to artificially age file1.txt beyond our 7-day threshold, while leaving the other files recent.

```
alpha@debian-11:~$ touch -d "10 days ago" /home/alpha/temp/file1.txt
alpha@debian-11:~$ ls
temp
alpha@debian-11:~$ ls -l /home/alpha/temp
total 0
-rw-r--r-- 1 alpha alpha 0 Jan 23 15:54 file1.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file10.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file2.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file3.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file4.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file5.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file6.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file7.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file8.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file9.txt
```

Figure 4.3: File Timestamp Manipulation for Cleanup Script Testing

4.2.4 CLEANUP SCRIPT VERIFICATION AND OLD FILE DELETION TEST

We ran our cleanup script manually to verify it was working correctly. As expected, it identified and deleted the aged file while preserving the recent ones, proving

our logic was sound before scheduling it as a cron job.

```
alpha@debian-11:~$ find ~/temp -type f -mtime +7 -delete
alpha@debian-11:~$ ls -l /home/alpha/temp
total 0
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file10.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file2.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file3.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file4.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file5.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file6.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file7.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file8.txt
-rw-r--r-- 1 alpha alpha 0 Feb  2 15:52 file9.txt
```

Figure 4.4: Cleanup Script Verification and Old File Deletion Test

4.2.5 SCRIPT FILE CREATION AND EXECUTABLE PERMISSION SETUP

We created our cleanup script file and made it executable using `chmod +x`. This step was essential to allow the script to run automatically when triggered by cron.

```
alpha@debian-11:~$ nano clean_temp.sh
alpha@debian-11:~$ chmod +x clean_temp.sh
alpha@debian-11:~$ ll
-bash: ll: command not found
alpha@debian-11:~$ ls -l
total 8
-rwxr-xr-x 1 alpha alpha   51 Feb  2 16:01 clean_temp.sh
drwxr-xr-x 2 alpha alpha 4096 Feb  2 15:57 temp
```

Figure 4.5: Script File Creation and Executable Permission Setup

4.2.6 MOVING CLEANUP SCRIPT TO SYSTEM DIRECTORY

For better organization and to follow Linux conventions, we moved our script to an appropriate system directory where cron could easily access it. This also separated our scripts from temporary user files.

```
alpha@debian-11:~$ mv clean_temp.sh /home/
alpha/   vagrant/
alpha@debian-11:~$ mv clean_temp.sh /home/
alpha/   vagrant/
alpha@debian-11:~$ mv clean_temp.sh /home/alpha/temp/
alpha@debian-11:~$ ls
temp
alpha@debian-11:~$ cd temp/
alpha@debian-11:~/temp$ ls
clean_temp.sh  file2.txt  file4.txt  file6.txt  file8.txt
file10.txt     file3.txt  file5.txt  file7.txt  file9.txt
alpha@debian-11:~/temp$
```

Figure 4.6: Moving Cleanup Script to System Directory

4.2.7 CRONTAB CONFIGURATION FILE EDITING

We edited the crontab to schedule our cleanup task. This involved deciding when the script should run and ensuring the path and command were correct for automated execution.

```
  GNU nano 5.4                          /tmp/crontab.B388DL/crontab *
0 0 * * * /home/alpha/clean_temp.sh
```

Figure 4.7: Crontab Configuration File Editing

4.2.8 ADDING LOGGING FUNCTIONALITY TO CLEANUP SCRIPT

We enhanced our script by adding logging capabilities. This would help us monitor what files were being deleted and troubleshoot any issues that might occur during automated runs.

```
alpha@debian-11:~/temp$ nano clean_temp.sh
alpha@debian-11:~/temp$ cat clean_temp.sh
#!/bin/bash

LOG_FILE="/home/alpha/cron_log.txt"

echo "----------------------------------------" >> "$LOG_FILE"
echo "LANCEMENT : $(date)" >> "$LOG_FILE"

find /home/alpha/temp -type f -mtime +7 -print -delete >> "$LOG_FILE" 2>&1

echo "TERMINE : $(date)" >> "$LOG_FILE"
```

Figure 4.8: Adding Logging Functionality to Cleanup Script

4.2.9 FINAL CRONTAB CONFIGURATION UPDATE

We finalized our crontab configuration with the complete scheduled job, including the logging redirection. This ensured our cleanup would run automatically while keeping a record of its actions.

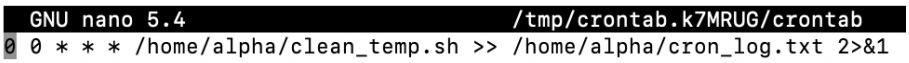

Figure 4.9: Final Crontab Configuration Update

PERMISSIONS AND FILE MANAGEMENT

5.1 SUDO & FILE OPERATIONS

This chapter covers network configuration, permission issues, file downloads, and archive extraction.

5.1.1 NETWORK INTERFACE CONFIGURATION CHECK

We tried running `ifconfig` to test whether the alpha user had sudoer privileges, knowing that accessing network interface information typically requires elevated permissions. This test helped us understand the alpha user's permission level in the system.

```
vagrant@debian-11:~$ su - alpha
Password:
alpha@debian-11:~$ /usr/sbin/ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.160.129  netmask 255.255.255.0  broadcast 172.16.160.255
        inet6 fe80::20c:29ff:feba:3420  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:ba:34:20  txqueuelen 1000  (Ethernet)
        RX packets 25018  bytes 28685925 (27.3 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4718  bytes 618454 (603.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 18  memory 0xfd4a0000-fd4c0000

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.33.10  netmask 255.255.255.0  broadcast 192.168.33.255
        inet6 fe80::20c:29ff:feba:342a  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:ba:34:2a  txqueuelen 1000  (Ethernet)
        RX packets 325  bytes 62488 (61.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 17  bytes 1286 (1.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 16  memory 0xfd2a0000-fd2c0000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 5.1: Network Interface Configuration Check

5.1.2 FILE CREATION PERMISSION FAILURE

We encountered a permission error when trying to create a file in a restricted directory. This taught us about Linux file permissions and the importance of proper ownership and access rights.

```
alpha@debian-11:~$ curl -o /opt/bat.zip https://github.com/sharkdp/bat/archive/refs/tags/v0.24.0.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
Warning: Failed to create the file /opt/bat.zip: Permission denied
```

Figure 5.2: File Creation Permission Failure

5.1.3 DIRECTORY OWNERSHIP CHANGE WITH CHOWN

To resolve our permission issues, we used `sudo chown alpha:alpha /opt/` to grant the alpha user ownership of the /opt directory. We also learned that `curl` doesn't follow HTTP redirects by default, requiring the `-L` flag to handle redirections properly.

```
alpha@debian-11:~$ su - vagrant
Password:
vagrant@debian-11:~$ sudo chown alpha:alpha /opt/
vagrant@debian-11:~$ su - alpha
Password:
su: Authentication failure
vagrant@debian-11:~$ su - alpha
Password:
alpha@debian-11:~$ curl -o /opt/bat.zip https://github.com/sharkdp/bat/archive/refs/tags/v0.24.0.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
```

Figure 5.3: Directory Ownership Change with chown

5.1.4 CURL DOWNLOAD WITH HTTP REDIRECT FOLLOWING

We attempted to download a file using `curl -o`, which failed because the URL redirected. We then tried again with `curl -L -o`, which successfully followed the redirect and downloaded the file. This taught us the importance of the `-L` flag when dealing with URLs that redirect.

```
alpha@debian-11:/opt$ curl -o /opt/bat.zip https://github.com/sharkdp/bat/archive/refs/tags/v0.24.0.
zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
alpha@debian-11:/opt$ file /opt/bat.zip
/opt/bat.zip: empty
alpha@debian-11:/opt$ curl -L -o /opt/bat.zip "https://github.com/sharkdp/bat/archive/refs/tags/v0.2
4.0.zip"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 2971k    0 2971k    0     0  2241k      0 --:--:--  0:00:01 --:--:-- 4300k
```

Figure 5.4: curl Download with HTTP Redirect Following

5.1.5 ARCHIVE EXTRACTION WITHOUT UNZIP UTILITY

We discovered that the standard `unzip` utility wasn't available on our system. Rather than installing it, we explored alternative methods for extracting archives, demonstrating resourcefulness when tools are limited.

Bash

```
python3 -m zipfile -e bat.zip .
```

- `-m zipfile` : Utilise le module ZIP intégré à Python.
- `-e` : Signifie "Extract".
- `bat.zip` : Ton fichier source.
- `.` : Signifie "ici" (le dossier courant `/opt`).

Figure 5.5: Archive Extraction Without unzip Utility

5.1.6 ARCHIVE EXTRACTION USING PYTHON ZIPFILE MODULE

We used Python's built-in zipfile module with `python3 -m zipfile -e bat.zip .` to extract our archive. This approach simulates scenarios where we lack sudo access to install packages, or when we choose to work with only available tools to avoid installing unnecessary binaries (like unzip) that might slow down the VM later, even though unzip is a small binary. This demonstrates resourcefulness in restricted environments.

```
alpha@debian-11:/opt$ file /opt/bat.zip
/opt/bat.zip: Zip archive data, at least v1.0 to extract
alpha@debian-11:/opt$ python3 -m zipfile -e bat.zip .
alpha@debian-11:/opt$ ls
bat-0.24.0  bat.zip
```

Figure 5.6: Archive Extraction using Python zipfile Module

WEBSERVER DEPLOYMENT

6.1 APACHE INSTALLATION & TROUBLESHOOTING

Apache2 web server was installed with attention to privilege management, port conflicts, and network configuration.

6.1.1 APACHE2 INSTALLATION ERROR AND RESOLUTION

We attempted to install Apache2 but encountered sudo privilege issues with the alpha user. We had to switch to the vagrant user, who had proper sudo rights, to successfully install Apache2 using `sudo apt-get install -y apache2`.

```
alpha@debian-11:~$ sudo su -
[sudo] password for alpha:
alpha is not in the sudoers file.  This incident will be reported.
alpha@debian-11:~$ su - vagrant
Password:
vagrant@debian-11:~$ sudo su -
root@debian-11:~# sudo apt-get install -y apache2
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
apache2 is already the newest version (2.4.66-1~deb11u1).
0 upgraded, 0 newly installed, 0 to remove and 94 not upgraded.
root@debian-11:~# sudo systemctl start apache2
Job for apache2.service failed because the control process exited with error code.
See "systemctl status apache2.service" and "journalctl -xe" for details.
root@debian-11:~#
```

Figure 6.1: Apache2 Installation Error and Resolution

6.1.2 RESOLVING PORT 80 CONFLICT

After installing Apache, we discovered that port 80 was already in use by another process. We had to identify and terminate the conflicting process before Apache could bind to the port and start serving pages.

```
root@debian-11:~# lsof -i :80
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
nc      11412 root    3u  IPv4  28754      0t0  TCP *:http (LISTEN)
root@debian-11:~# kill 11412
root@debian-11:~# sudo systemctl start apache2
```

Figure 6.2: Resolving Port 80 Conflict

6.1.3 ADDITIONAL PORT CONFLICT RESOLUTION

We continued troubleshooting port conflicts to ensure Apache could start cleanly. This involved checking for other services that might be competing for the same ports.

```
root@debian-11:~# sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2026-02-02 17:03:14 UTC; 3min 8s ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 13367 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 13371 (apache2)
      Tasks: 55 (limit: 1110)
     Memory: 14.9M
        CPU: 92ms
     CGroup: /system.slice/apache2.service
             ├─13371 /usr/sbin/apache2 -k start
             ├─13372 /usr/sbin/apache2 -k start
             └─13373 /usr/sbin/apache2 -k start

Feb 02 17:03:14 debian-11 systemd[1]: Starting The Apache HTTP Server...
Feb 02 17:03:14 debian-11 apachectl[13370]: [Mon Feb 02 17:03:14.073425 2026] [core:error] [pid 133>
Feb 02 17:03:14 debian-11 systemd[1]: Started The Apache HTTP Server.
```

Figure 6.3: Additional Port Conflict Resolution

6.1.4 NETWORK CONFIGURATION FOR WEB SERVER ACCESS

We opted for the localhost port forwarding approach to access our web server. By adding `config.vm.network "forwarded_port", guest: 80, host: 8080` in the Vagrantfile and running `vagrant reload`, we could access the Apache server via `localhost:8080`. This approach is universal and mirrors common development workflows (like `php artisan serve` or `npm run dev`), eliminating the need to search for VM IPs or modify configuration files like `APP_URL` in Laravel or Node projects.

```
# Create a forwarded port mapping which allows access to a specific port
# within the machine from a port on the host machine. In the example below,
# accessing "localhost:8080" will access port 80 on the guest machine.
# NOTE: This will enable public access to the opened port
# config.vm.network "forwarded_port", guest: 80, host: 8080
```

Figure 6.4: Network Configuration for Web Server Access

CONCLUSION

All lab objectives were successfully completed. The environment is now properly configured with secure SSH access, automated file maintenance via cronjobs, comprehensive SSL certificate analysis skills, proper permission management, and a functional Apache web server.