Document version 1.0, 2024-10-17, Colin Joseph

Project and Training 1 (BTI3001)

Abstract

Purpose: The goal of the git-chat project is to create a Git-based chat system. The project is designed to help become familiar with Git commands, Linux commands and set up and purpose of Secure Shell (SSH). Additionally, Visual Studio Code was used for documentation and editing of skripts, which should lead to learn to use the application.

Methods: The script is written in a Linux environment using Git commands and bash scripting with standard Linux commands. It consists of a frontend script for user interactions, while the backend stores the message history in a Git repository.

Results: The git-chat system allows multiple users to interact. Messages can be viewed by different users on separate terminals simultaneously.

Introduction

This project requires working in a Linux-based environment, provided by BFH. The first step is to connect the local system to remote servers, such as `castor.ti.bfh.ch` and `gitlab.ti.bfh.ch`. This connection is configured using SSH.

Furthermore, Visual Studio Code is used for documentation and remote connection management. Once the setup is complete, the Git project is pulled from gitlab to the user profile on `castor` to work on the project.

The key objective of the Git-based chat system is to initialize a bare Git repository to store messages. Additionally, it involves creating and connecting a user directory to the backend. Handling user message input and committing it to Git to log the chat history is also an important part of the project. The method section will describe the setup of the SSH key connection, Visual Studio Code and guide to the git-chat script.

Methods

Project 1: git-chat

SSH Configuration & Visual Studio Code Setup

SSH Setup [1]

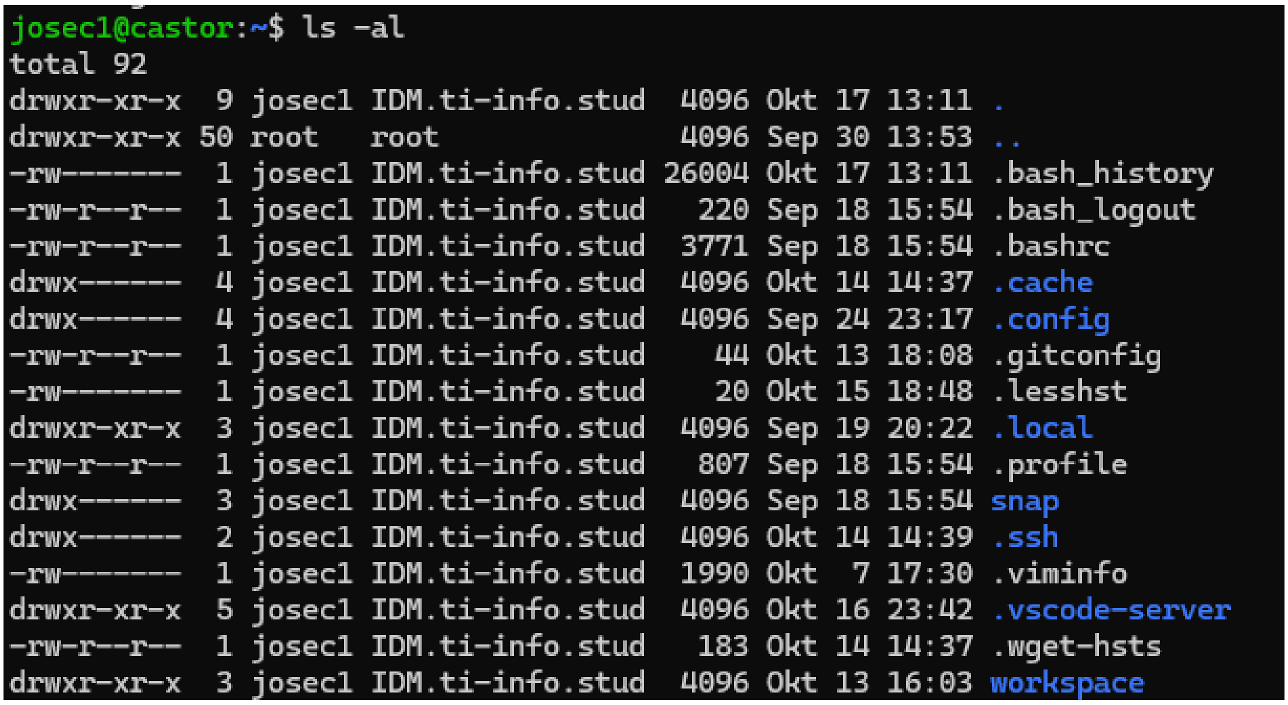
SSH is created for secure connections and enabling the transfer of data between machines or servers. One of the main advantage is to protect against man-in-the-middle attacks, which significantly reduces the chances of eavesdropping or unauthorized access. This is achieved by encryption techniques to safeguard the data during transmission.

A key aspect of SSH security is its use of public-key authentication. The client generates a pair of cryptographic keys, a public key and a private key. The public key is shared with the server or other machine, while the private key remains stored on the local machine. Even if an attacker obtains the public key, they would still be unable to access the system without the corresponding private key. [2]

Before logging into `castor` account, make sure to connected to BFH's VPN. The manual for setting up the VPN can be found at this link: VPN Setup Guide.

Steps to set up SSH:

1. Check if you have an `.ssh` directory using the `ls -al` command.
   - `ls` is a Linux command that lists the contents of a directory. Adding `-a` shows all files, including hidden files. The `-al` option also shows detailed information, like file permissions, owner, size, and modification date. [3]
2. If the directory does not exist, create it using the `mkdir` command [3]:
   ```
   mkdir .ssh
   ```
3. Ensure the `.ssh` directory is only accessible by the owner. Modify its access rights using the `chmod` command [3]:
   ```
   chmod 700 .ssh
   ```
   The access of `.ssh` directory should only be manimulated by the owner. It can be modified by the command `chmod` followed by three digit and the file/directory name. First digit stands for owner user, second digit group and the third digit for other users. To modify the access right enter the command on the terminal `chmod 700`. Now only the owner should have the modification right as we can see from the image.



```
josec1@castor:~$ ls -al
total 92
drwxr-xr-x  9 josec1 IDM.ti-info.stud  4096 Okt 17 13:11 .
drwxr-xr-x 50 root   root              4096 Sep 30 13:53 ..
-rw-------  1 josec1 IDM.ti-info.stud 26004 Okt 17 13:11 .bash_history
-rw-r--r--  1 josec1 IDM.ti-info.stud   220 Sep 18 15:54 .bash_logout
-rw-r--r--  1 josec1 IDM.ti-info.stud  3771 Sep 18 15:54 .bashrc
drwx------  4 josec1 IDM.ti-info.stud  4096 Okt 14 14:37 .cache
drwx------  4 josec1 IDM.ti-info.stud  4096 Sep 24 23:17 .config
-rw-r--r--  1 josec1 IDM.ti-info.stud    44 Okt 13 18:08 .gitconfig
-rw-------  1 josec1 IDM.ti-info.stud    20 Okt 15 18:48 .lesshst
drwxr-xr-x  3 josec1 IDM.ti-info.stud  4096 Sep 19 20:22 .local
-rw-r--r--  1 josec1 IDM.ti-info.stud   807 Sep 18 15:54 .profile
drwx------  3 josec1 IDM.ti-info.stud  4096 Sep 18 15:54 snap
drwx------  2 josec1 IDM.ti-info.stud  4096 Okt 14 14:39 .ssh
-rw-------  1 josec1 IDM.ti-info.stud  1990 Okt  7 17:30 .viminfo
drwxr-xr-x  5 josec1 IDM.ti-info.stud  4096 Okt 16 23:42 .vscode-server
-rw-r--r--  1 josec1 IDM.ti-info.stud   183 Okt 14 14:37 .wget-hsts
drwxr-xr-x  3 josec1 IDM.ti-info.stud  4096 Okt 13 16:03 workspace
```

Explantation of the symbols in the long listing: r = Read w = Write x = Execute d = Directory

4. Use the `ssh-keygen` command to generate a key pair. Name the key appropriately to avoid confusion when multiple keys exist [2]:

   ```
   ssh-keygen -f gitlab.ti.bfh.ch
   ```

   This generates two keys: the private key (`gitlab.ti.bfh.ch`) and the public key (`gitlab.ti.bfh.ch.pub`).

Take care where you creat the keys. If it is not in .ssh directory, move it with the command `mv` followd by name of key and where it should be moved [3]. Example: `mv gitlab.ti.bfh.ch .ssh`

5. Copy the public key. You can display it using the `cat` command [3]:

   ```
   cat gitlab.ti.bfh.ch.pub
   ```

   Paste the copied text into your gitlab account settings.

6. The `ssh -i` command allows you to specify which private key to use when connecting to a remote system [1]:

   ```
   ssh -i ~/.ssh/gitlab.ti.bfh.ch git@gitlab.ti.bfh.ch
   ```

   `.ssh -i` name of private key and username to connect, where public key is stored

7. Set up the SSH config file (optional but recommended) to simplify the connection process. Use `nano` or `vim` to create and edit the `.ssh/config` file [1]:

```
Host gitlab.ti.bfh.ch
    HostName gitlab.ti.bfh.ch
    User git
    IdentityFile ~/.ssh/gitlab.ti.bfh.ch
```

You can also set up key-based authentication between your local machine and the `castor` server for automated connections, but this is optional and will not be covered in detail here. The principle and executation is similiar of above listed manual.

Visual Studio Code Setup [1]

Visual Studio Code (VS Code) is an application developed by Microsoft, which is used for software developed tasks. We used here for creating a markdown file. Markdown PDF plugin was installed to convert the final version of markdown file into a pdf file. Additionally Remote-SSH Plugin is installed for managing SSH-connection to castor server to simplify the access and editing of scripts.

Explanation of the git-chat Bash Script

Git-chat script is divided into two files: `functions.sh` and `git-chat.sh`. Together, they implement a chat system that allows users to exchange messages. Below is a breakdown of how the code works, along with an explanation of each section.

functions.sh

This file contains several functions that the main script (`git-chat.sh`) calls during the execution. Here is a breakdown of each function:

- `welcome()`: Displays a welcome message when the user starts the chat.

```
welcome() {
    echo "Welcome to git-chat!"
}
```

- `fetch_and_merge()` : Git-Fetch only updates the meta information of a remote repository and doesn't merge it. Git-Merge helps to bring changes from diffrent branches together in a structured way [4] [5]

```
fetch_and_merge() {
    git fetch -q
    git merge --commit --no-edit --allow-unrelated-histories -q > /dev/null
}
```

- `format_log()` : Output of log [4] [6]

```
format_log() {
    git --no-pager log --pretty=format:"%ad %s" --date=format:'%Y-%m-%d %H:%M:%S'
--reverse
}
```

git-chat.sh

This is the main script that runs the git-chat system. It manages the overall workflow, including setting up the backend, handling user input, and calling the appropriate functions from `functions.sh`.

- **Backend Initialization**: The script checks if the backend Git repository exists. If not, it initializes a bare repository to store the messages. [5]

```
if [ ! -d "../backend" ]; then
    git init --bare backend
fi
```

- **import function skript**:

```
source /home/josec1/workspace/git-chat-josec1/frontend/functions.sh
```

- **Username Validation**: The script checks if the username is provided as an argument. If not, it exits with an error. [7]

```
if [ -z "$1" ]; then
    echo "Username is required."
    exit 1
fi
```

- **create/connect (clone) user message dir from backend**: Where message of users will pushed in the backend [4] [5]

```
if [ ! -d "$1" ]; then
    git clone /home/josec1/workspace/git-chat-josec1/backend $1
    git remote add origin /home/josec1/workspace/git-chat-josec1/backend
fi
```

- **Welcome Message**: Calls the `welcome` function to display a greeting to the user.

```
welcome "$1"
sleep 3
```

- **Main loop**: Enters an infinite loop, which firstly do the fetches and merges messages from the backend and afterwads displays the log for message entry part followed by commiting and pushing by Git-Commands [5]

```
cd $1 || exit

while [ true ]
do
    clear
    echo "it is working..."
    sleep 2

    # TODO: fetch/merge messages from backend
fetch_and_merge
    # TODO: format message log
format_log
    # TODO: message prompt ->
    # e.g., 'enter message: '

echo -e "\n$1 enter a message: "

    # TODO: wait 10 secs for user input (message)
if read -t 10 message; then
    # TODO: skip empty message
[ -z "$message" ] && continue
sleep 3
    # TODO: add message to queue
git commit -q -m "$1: $message" --allow-empty
fi
    # TODO: deliver message
git push -q
done
```

Results

The git-chat system allows multiple users to interact by synchronizing their messages through Git. The system takes advantage of Git's ability to track changes and handle conflicts.

Discussion & Conclusion

The biggest challenge was writing the script using Linux command and Git commands. The knowledge of these commands was new to me and I needed a basic structure for the script with the help of ChatGPT. I tried to adapt it using my knowledge of other programming languag (Java), but ChatGPT led me on the wrong path. The user messages were processed on the frontend instead of the backend.

This also resulted in unnecessary commits on gitlab. After receiving advice from tutor Peter A. von Niederhäusern I was able to achieve the correct result, where the chat is managed in the backend and the user interactions occur in the frontend.

Throughout this project I learned a lot about the basics of Linux and Git. Additionally I become more familiar using VS Code. Applying SSH in practice also helped deepen my understanding of the topic.

The most important lesson for me after this project is to ask for help earlier and read the hints and documentation more carefully. It could save me a lot of time and stress.

Citation

[1] https://moodle.bfh.ch/course/view.php?id=37649 - SSH set up was mainly taken from the Introduction video. Set up of VS Code was done during the classes of Peter A. Niederhäusern
[2] Slidedeck-1a-ssh from Mr. Niederhäusern & https://www.ssh.com/academy/ssh/keygen#what-is-ssh-keygen? - 17.10.2024 22:30
[3] Slidedeck-1b-basic-commands from Peter A. Niederhäusern
[4] Git Crash Course Version 2.1.12 of September 15, 2024 from Author: Prof. Andreas Habegger and Peter A. von Niederhäusern
[5] Advice and bash line provided from Peter A. Niederhäusern - 17.10.2024 20:31
[6] ChatGPT 15.10.2024 time unknown
[7] ChatGPT 14.10.2024 time unknown

Additionally resources used for understanding the content - 14.10.2024 time unknown
Explanation of Git-commands and logic of Git: https://www.youtube.com/watch?v=mJ-qvsxPHpY
Explanation of Linux Commands: https://www.youtube.com/watch?v=gd7BXuUQ91w&t=274s
Explanation of Linux: PDF-File Linux Crash course Version 3.0.7 of September 15, 2024 from Tutor: Peter A. von Niederhäusern, Author: Lukas Studer and Prof. Andreas Habegger
Explanation of basic operators in shell scripting https://www.geeksforgeeks.org/basic-operators-in-shell-scripting/

Representation of bash in markdown: ChatGPT 18.10.2024 00:15