Data Structures & Algorithms Lab Project

Version Control System

Submitted to: Mam. Raeena Tauqir

Introduction

- **Version control systems** are software tools that help software teams manage changes to source code over time
- Version Control Management System helps manage your code efficiently
- You can track the history of project
- Our Project is a minimal clone of a very popular version control system “Git”.

Background

- Git is a popular version control system that was initially created by Linus Torvalds for development of the Linux Kernel.
- managing the code like remote work issue, code review issues & tracking bugs was impossible.
- To overcome this issue Centralized Version Control Systems (CVS) were introduced.
- CVS had an issue of making changed in central version.
- To avoid that issue Distributed Version Control (DVS) System was introduced like Git.

Features

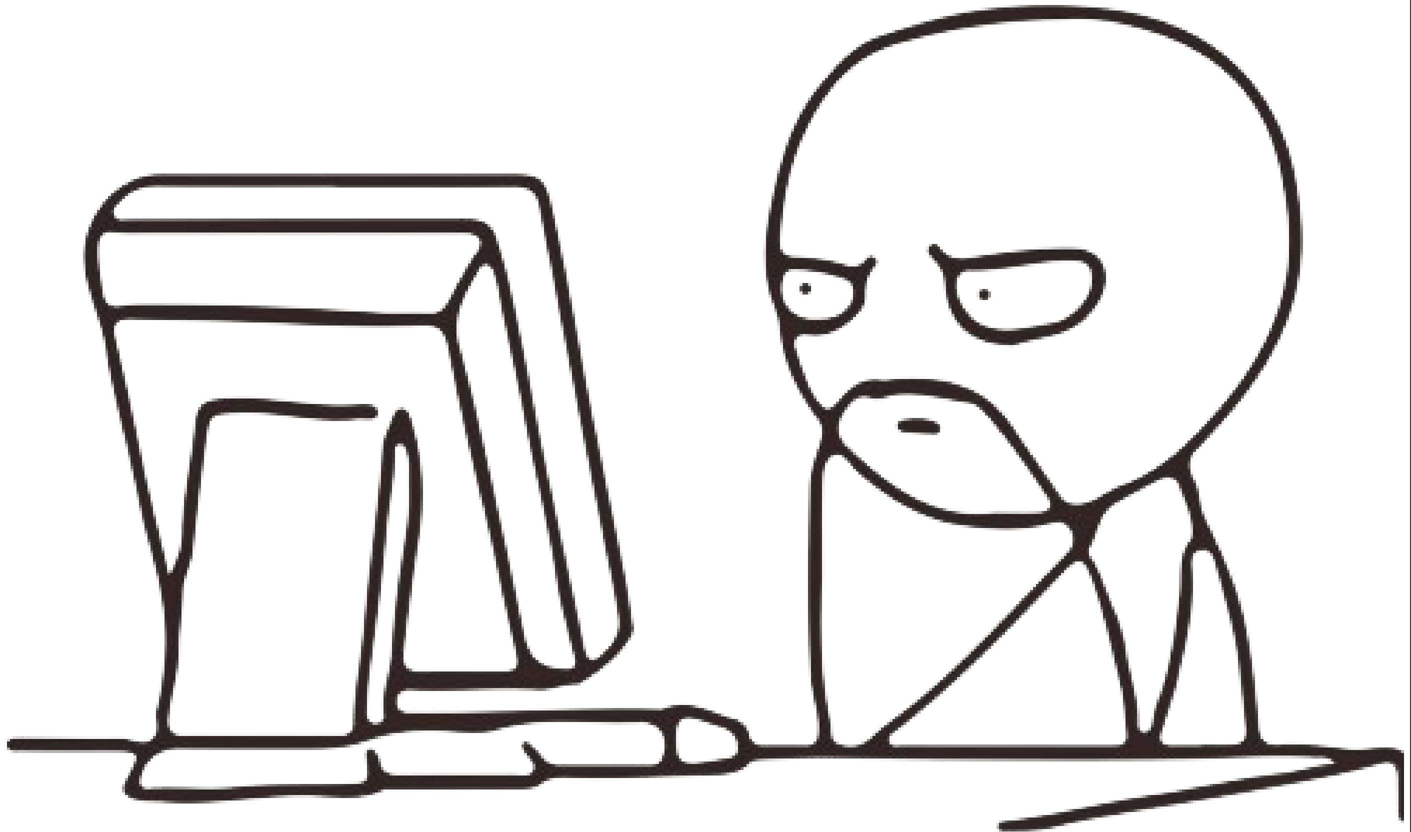
- You can manage your code efficiently by making different versions of project
- You can work on different modules of project without being dependent on other peoples’ work
- Track the history of project
- If applications stop working because of some issue it’s easy to find bugs.

Features

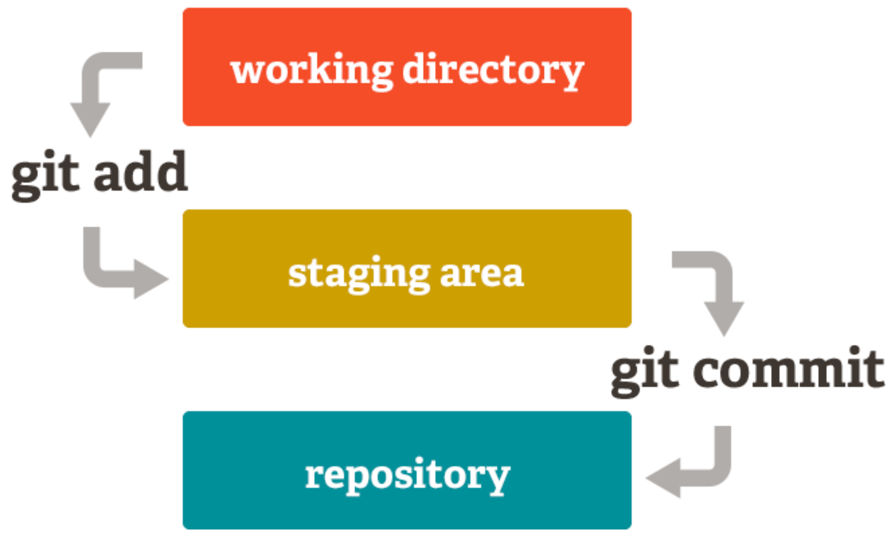
- As in our project which is a simple clone of Git Version Control System.
- You can add your code to staging area.
- Make commits to create different versions of project files.
- Display all the commits OR versions in console.
- Even Revert back to a previous version commit.
- Track changes.

but how does it work?

- Basically It’s divided into 3 Main Parts
- LET’S SEE



WORKING – BASIC IDEA

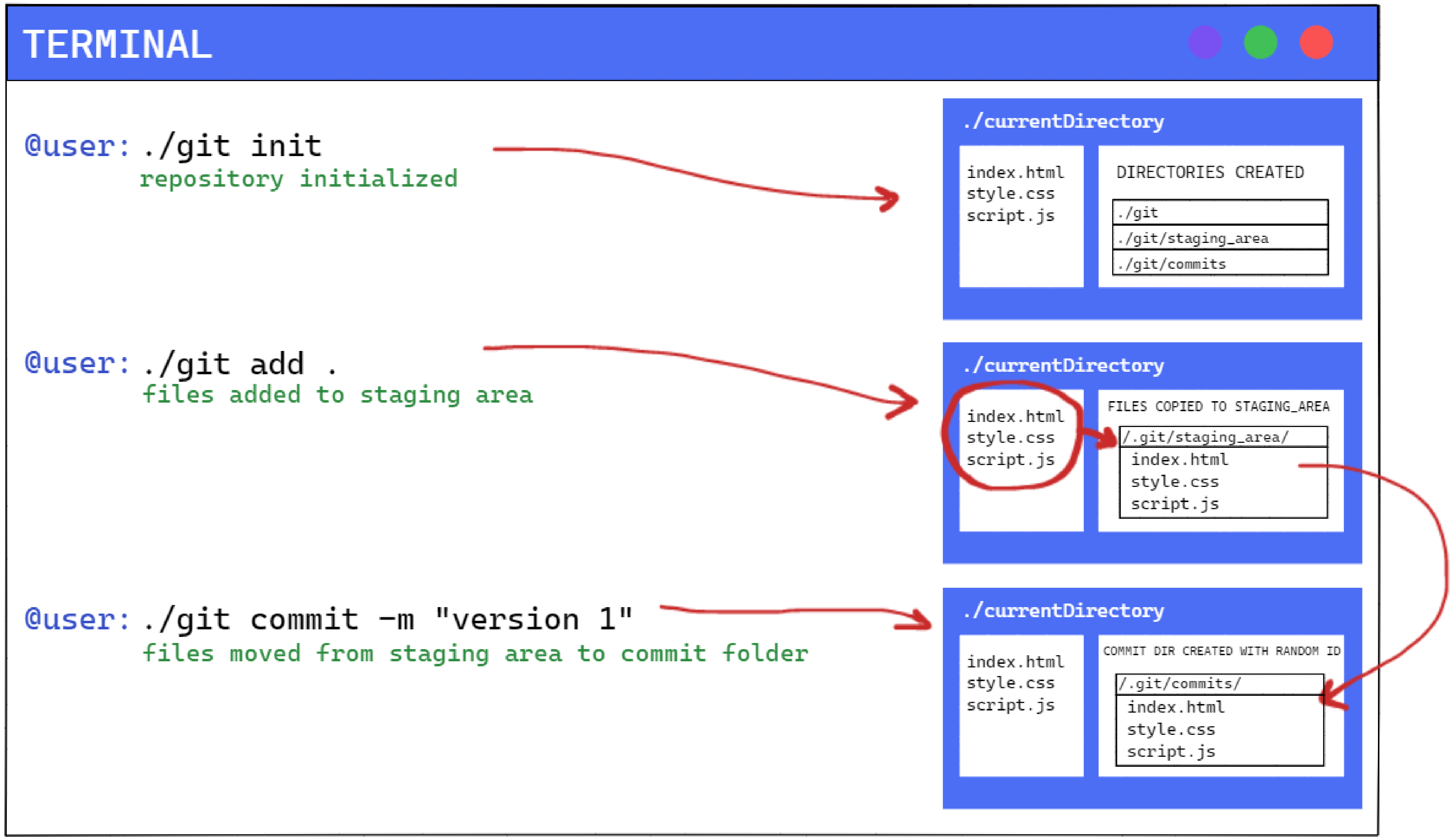


you enter the Mall →	<git init>	initialization for .git folder
you collect items in your trolley →	<git add .>	adding all source files in staging area to prepare for committing
you go to cashier, bill is made & you take your items →	<git commit>	all changes are committed and a version OR backup is made

WORKING – PROJECT REFERENCE

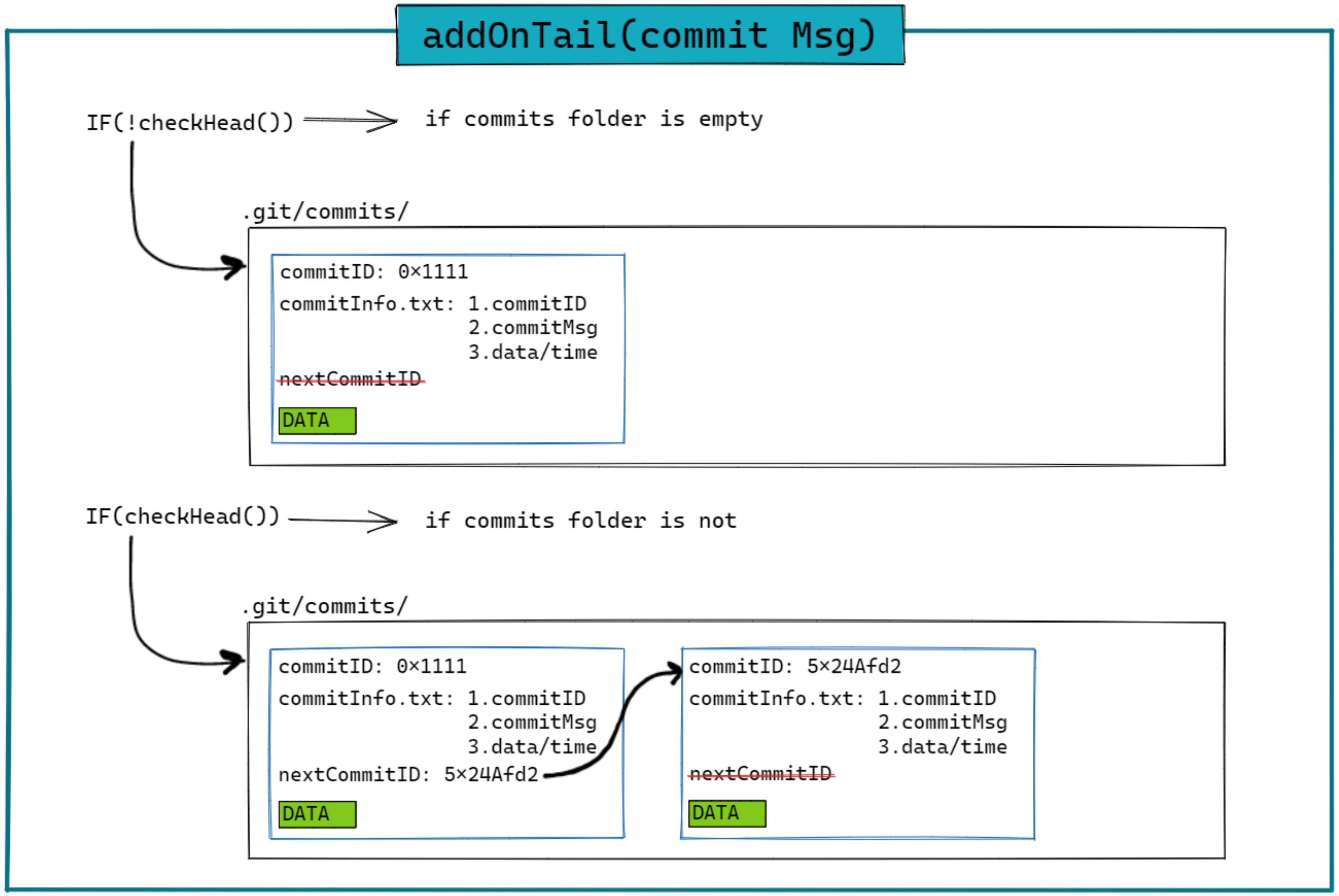
- All the functions work using the command arguments passed by user when using terminal environment.
- This works by using argc **& *argv[] in main function.**
- **argc & argv are how command line arguments are passed to main() in C and C++**
- **argc will be number of arguments of string type and *argv[] is an array that contains those arguments**
- **With each commit, a new directory is created**
- **and hence every time an instance of the program runs it iterates through linked list nodes created using directory paths.**

WORKING – PROJECT REFERENCE

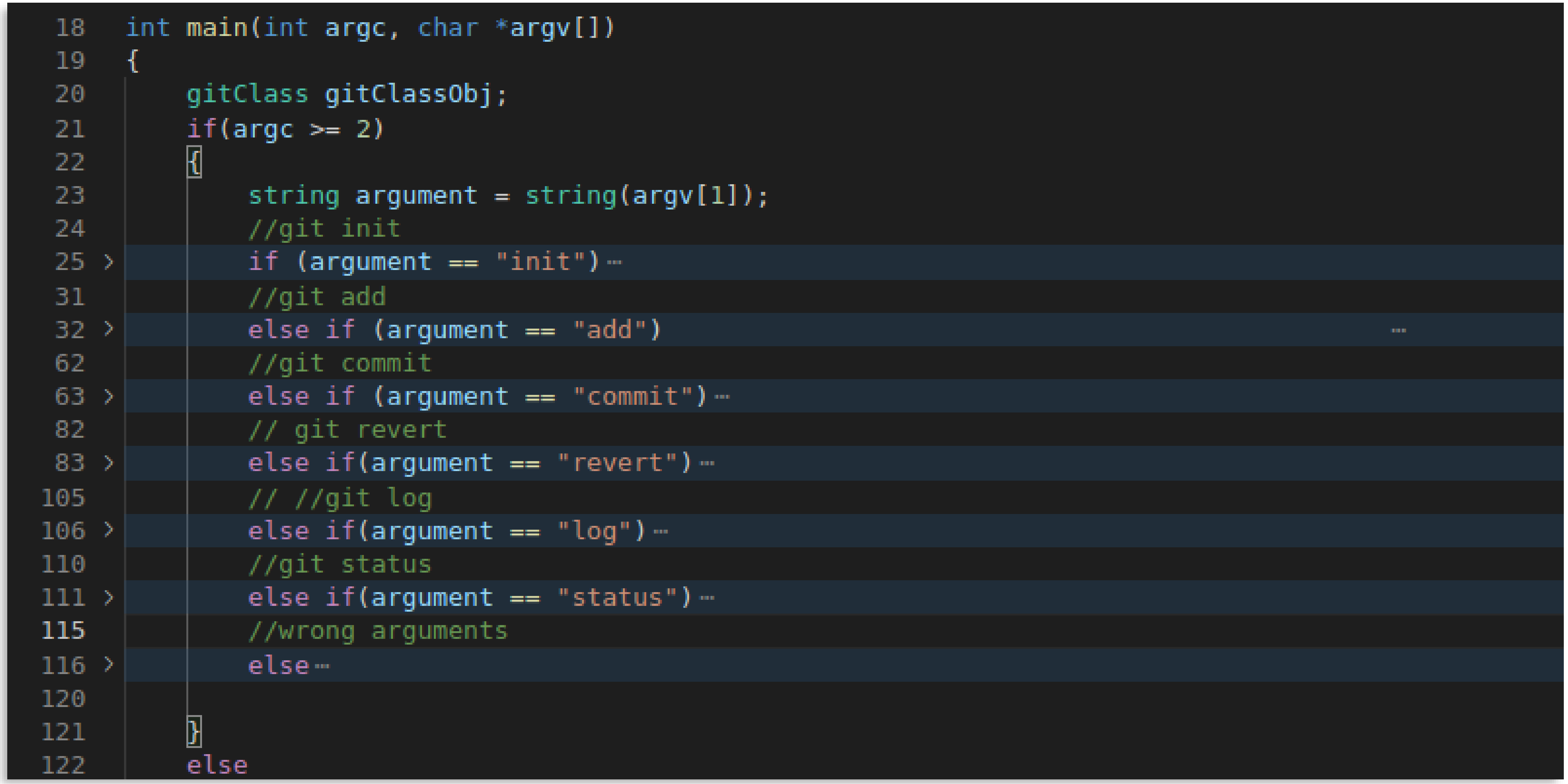


```
TERMINAL

@user: ./git init
       repository initialized

@user: ./git add .
       files added to staging area

@user: ./git commit -m "version 1"
       files moved from staging area to commit folder
```

WORKING – HOW COMMITS WORK AS NODES

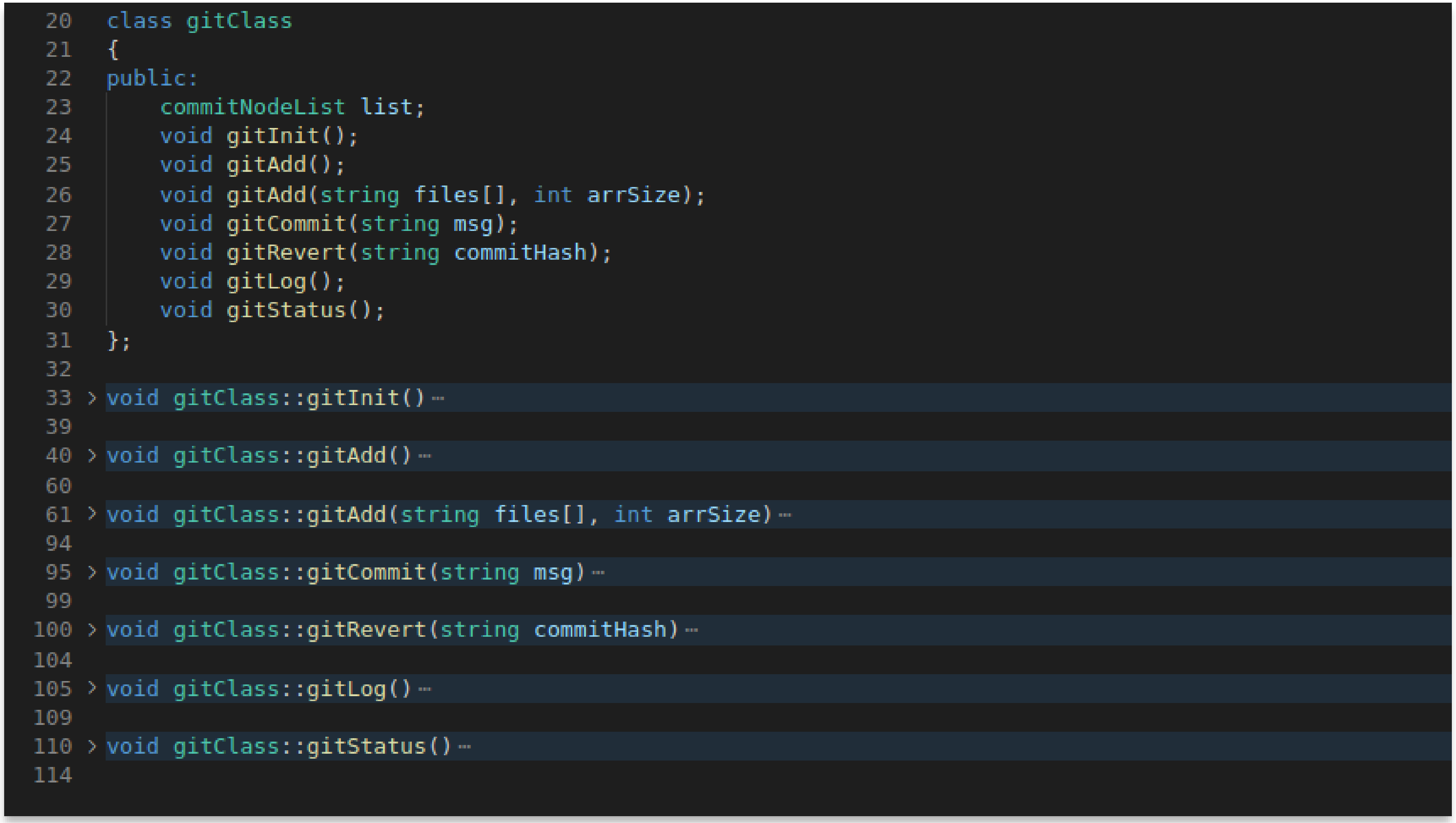


Code Structure – main.cpp

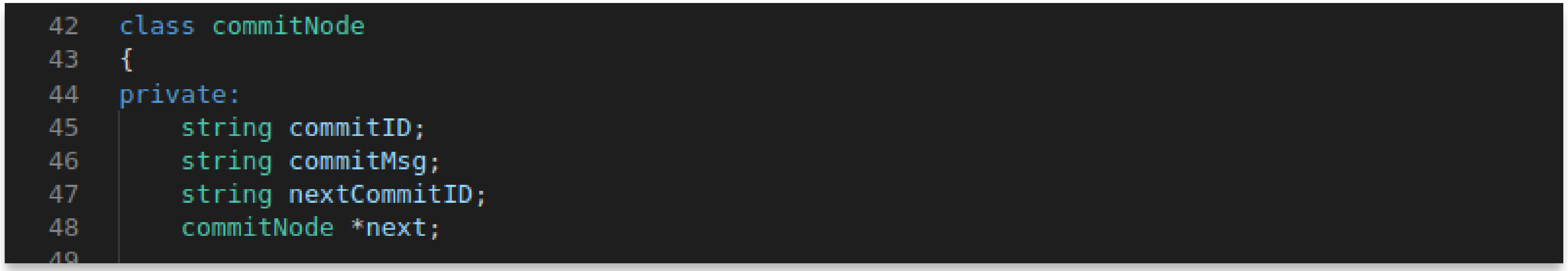


```
int main(int argc, char *argv[])
{
    gitClass gitClassObj;
    if(argc >= 2)
    {
        string argument = string(argv[1]);
        //git init
        if (argument == "init")⋯
        //git add
        else if (argument == "add")                                    ⋯
        //git commit
        else if (argument == "commit")⋯
        // git revert
        else if(argument == "revert")⋯
        // //git log
        else if(argument == "log")⋯
        //git status
        else if(argument == "status")⋯
        //wrong arguments
        else⋯

    }
    else
```

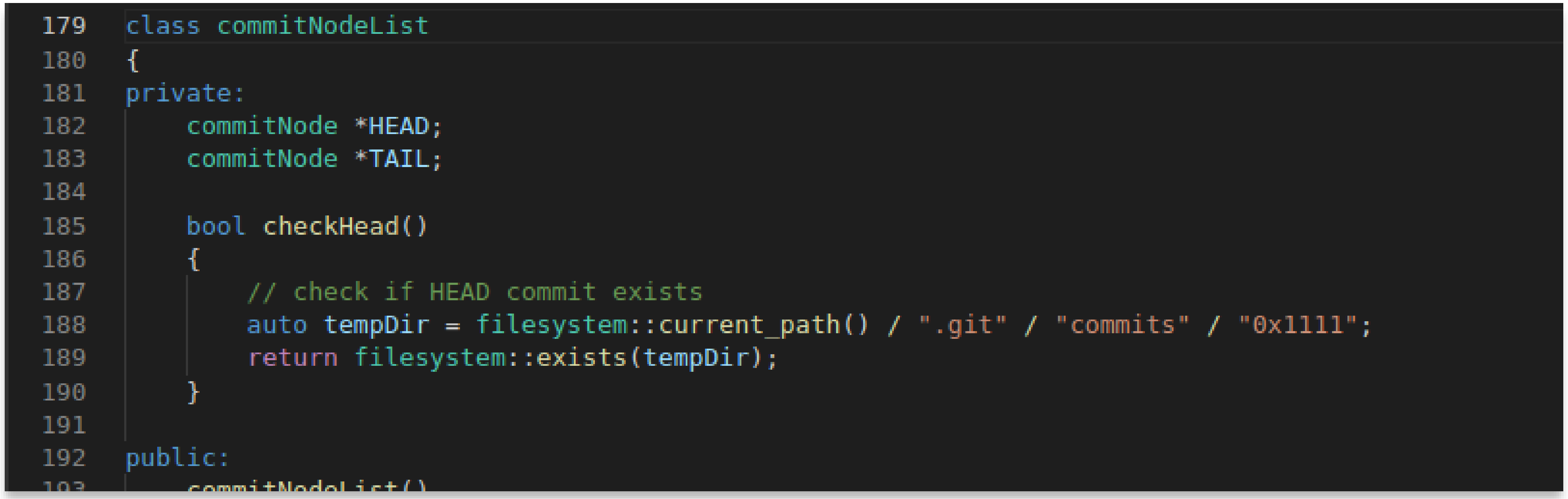
Code Structure – gitClass.cpp


```
class gitClass
{
public:
    commitNodeList list;
    void gitInit();
    void gitAdd();
    void gitAdd(string files[], int arrSize);
    void gitCommit(string msg);
    void gitRevert(string commitHash);
    void gitLog();
    void gitStatus();
};

void gitClass::gitInit() ⋯

void gitClass::gitAdd() ⋯

void gitClass::gitAdd(string files[], int arrSize) ⋯

void gitClass::gitCommit(string msg) ⋯

void gitClass::gitRevert(string commitHash) ⋯

void gitClass::gitLog() ⋯

void gitClass::gitStatus() ⋯
```

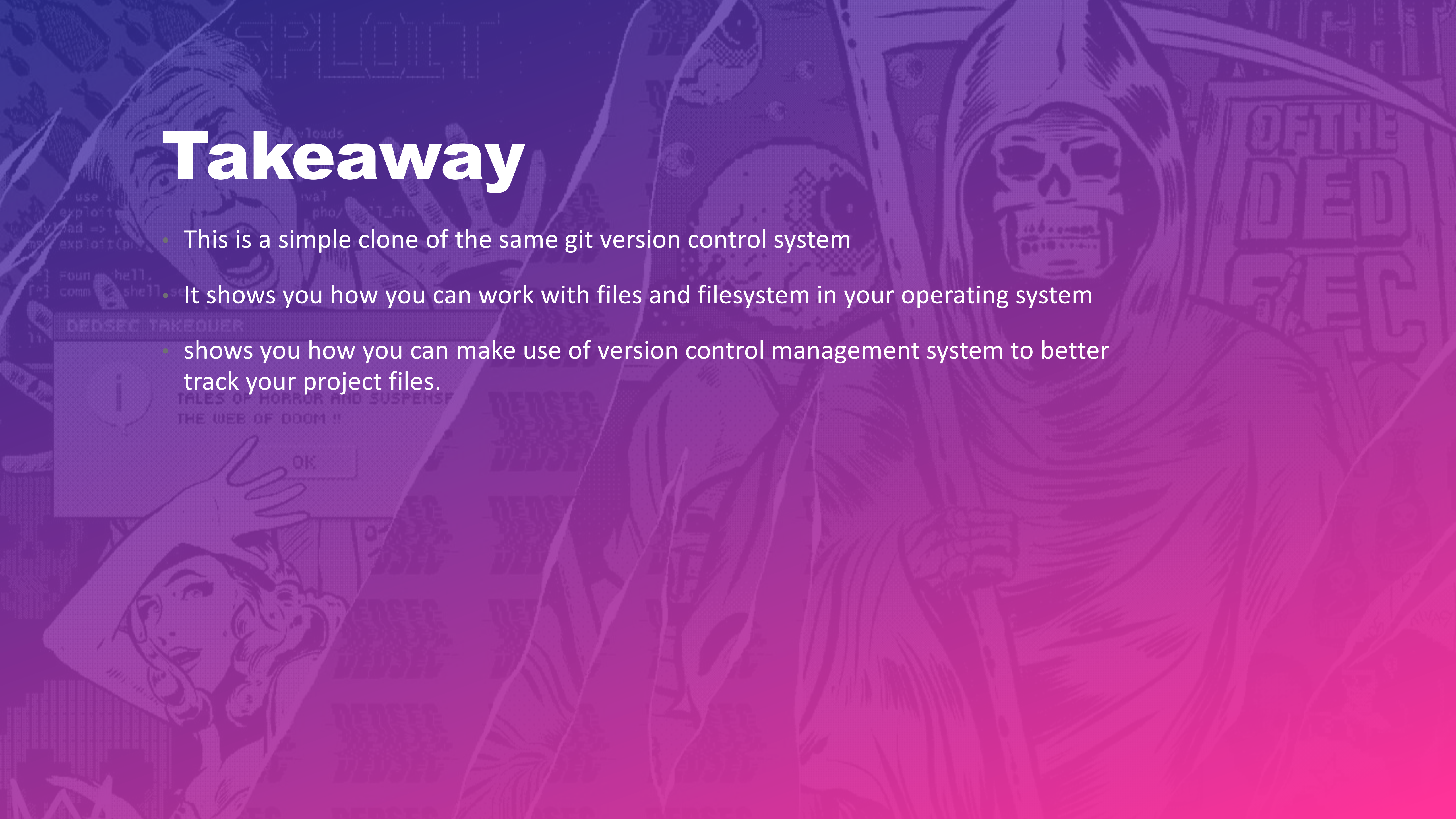
Code Structure – commitNodeList.cpp



```
class commitNode
{
private:
    string commitID;
    string commitMsg;
    string nextCommitID;
    commitNode *next;
```

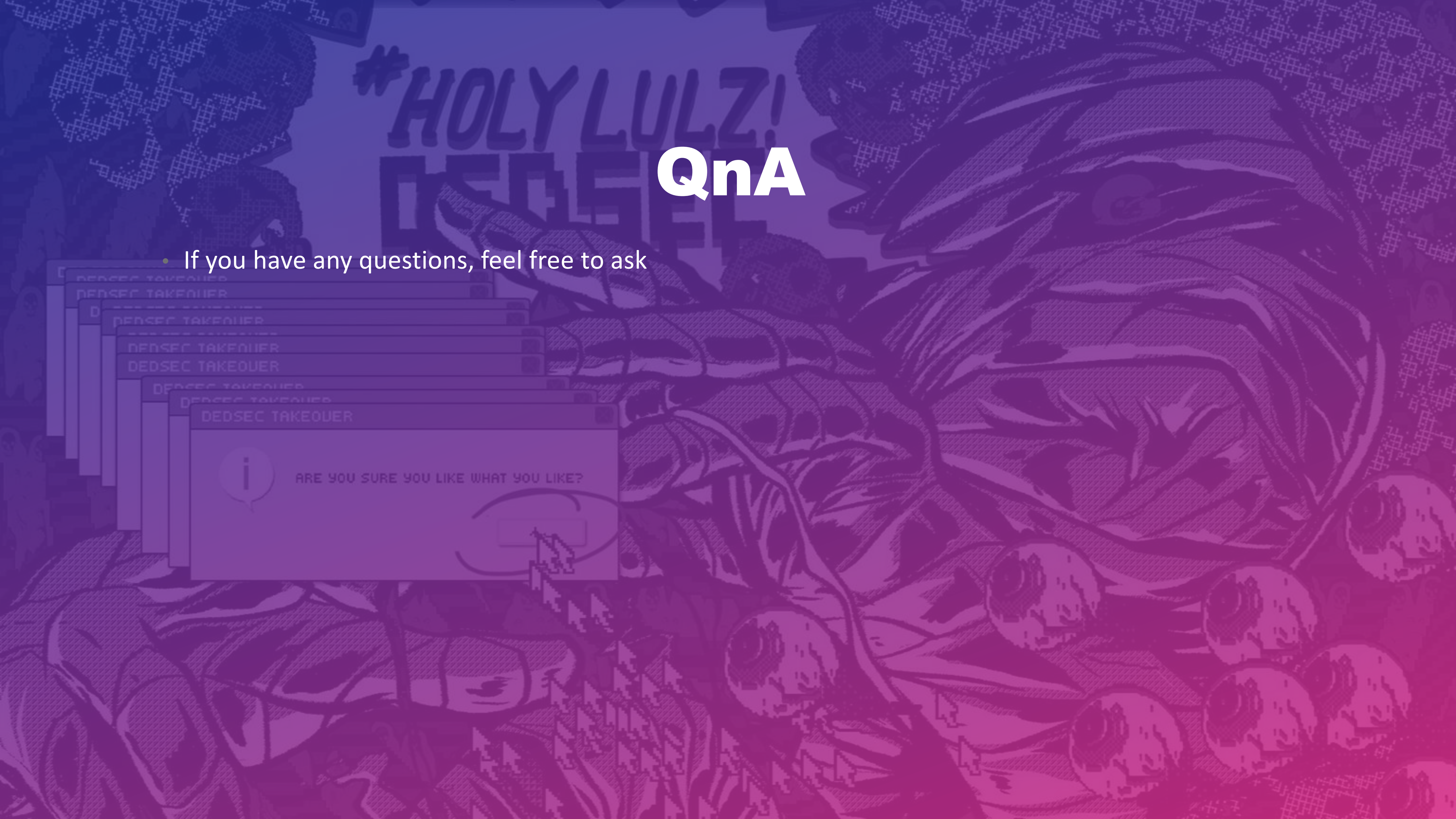


```
class commitNodeList
{
private:
    commitNode *HEAD;
    commitNode *TAIL;

    bool checkHead()
    {
        // check if HEAD commit exists
        auto tempDir = filesystem::current_path() / ".git" / "commits" / "0x1111";
        return filesystem::exists(tempDir);
    }

public:
```



Takeaway

- This is a simple clone of the same git version control system
- It shows you how you can work with files and filesystem in your operating system
- shows you how you can make use of version control management system to better track your project files.



QnA

- If you have any questions, feel free to ask