Synopsis:

This project is about make a smaller version of git/github (version control). We are making a system where the user/client can store their code at the server and perform various tasks like pushing, creating, updating, upgrading, adding and removing of the projects from the server.

User Detailing

The client will first configure to connect to the server by providing the host name and the port number. Once, they get the access to the server, they can maintain the project/ their versions on the server based on different command that are provided. If at any point, the server cannot be connected, the program will halt. The program also uses a protocol to pass message to the server and the client (back and forth).

WTF Client

```
int getVersion(char * manifestData)

given a manifest data, it reads till \n and finds the current version
converts it to char and then returns it in int form.
```

char * manifestData: manifestData store the information/data/contents of .manifest file.

int doActualUpdate(int ufd, int cfd)

This method traverse through the linked list server and client manifests, where they compare if the files are same or not, if not what are the changes, if there files which are not in server and in the client then it is going to delete the file from the client. It is going to Append 'D <file/path> <server's hash> to .Update. It there are files on server which are not present on the client then it is going to take action: Append 'A <file/path> <server's hash> to .Update. If the .manifest version of client and server are different with different hash and live hash too, then it is a confluent and it going to take an action: Append 'C <file/path> <server's hash> to .Conflict. If the .manifest version of client and server are different with different hash but

the same live hash, then it is a modify and it is going to take an action: Append ‘M <file/path> <server’s hash> to .Update.

int ufd: it is update file descriptor.

int cfd: it is conflict file descriptor.

void addToManifestLL(char *t,char *v, char *p, char*h,int isTagPresent)

It takes the tag, version, pathname, hashcode and add/store all of them in the linked list.

char *t: the tag

char *v: the version

char *p: the pathname

char*h: the hashcode

int isTagPresent: indicates whether the tag is present or not.

void clientManifestLL(char *manifest)

It tokenize the .Manifest file from the client to store the tag, version, pathname and the hashcode in the variable and send to to addToManifest to store in the linked list.

char* manifest: the content of the .Manifest file is stored here.

char * getUpdatedManifest(int *socketfd)

I reads the message from the socket based on the protocol reads till ":" whatever is read till then say 56 then 56 bytes are read and returned.

int *socketfd: address of socket descriptor.

char *getContent(int fd)

Returns the content of the files based on the file descriptor passed in.

int fd: it is going to read the content of fd, the file descriptor.

int numberOfEndLines(char* str)

Returns the number of end line in the string.

char* str: the string to be read to return the number of lines.

char endLineSeparate(char* str)**

It is going to take a string, and then tokenize it based on the \n and stores the tokens in the char** array.

char* str: the string to be tokenize

char* pathSeparate(char* str)

It takes the string and return the path name out of the string, usually the path name is after the second \t. It returns the path.

char* str: the string in which we got have to find the path name.

int makeCommit(char *proj)

Makes Commit file in the project and returns the file descriptor proj is where the commit to be made so proj/.Commit

char* proj: It is the directory, where it has to make a. commit file.

void addToCommit(char mark, char* v, char *path, char* hashcode, char* proj, int* fdp)

Adds the to Commit file in case commit passes.

char mark: It is a tag (A, M or R)

char* v: v is the version to write it in the .commit file.

char *path: it is the path to write in the .commit file.

char* hashcode: it is the hashcode to write in the .commit file.

char* proj: it is the name of the directory where the .commit file is located.

int* fdp: it is file descriptor of the commit file.

int deleteCommit(char *prog)

Finds the .Commit file in a given project and deletes it. Project in which .Commit is to be located and deleted.

char* prog: the project in which .commit is to be located and deleted.

void compare(int *fdp,int *socketfd,char *v,char *p, char *h,char* pro,char Mark)

It is traversing the server linked list and adding the appropriate tag to .commit file and remove .commit file if it fails.

int*fdp: commit file descriptor

int* socketfd: Socket descriptor's address

char* v: the version

char* p: the path name

char* h: the hashcode

char Mark: the tag (R or A)

void tokenizeClient(int *fdp,int *socketfd,char *manifest,char *projec

It is going to read a .Manifest and tokenize each line to store the data (tag, pathname, version, hashcode) in the linked list.

int* fdp: .Commit file descriptor

int* socketfd: socket descriptor's address

char* manifest: the content inside the .Manifest file

char* project: it is the directory where the .Manifest file is present.

void addToList(char *v,char *p, char* h)

It is going to add the node that contains the data of the server's manifest at the end of the server's Linked list.

char* v:the version

char* p: the path code

char*h: the hash code

void serverManifestLL(char *manifest)

It tokenizes the server's manifest to extract file version, path and associated hashcode.

char* manifest: the content inside the .Manifest file

char * loadClientManifest(char *dir, char* fileToRead)

It gets the file based on the path, it forms the path, opens the file, loads the contents and returns. It is used to get the content of manifest.

char* dir : The Directory

char* fileToRead: the file to read

char* readFile(char *filepath)

reads from the file if the path has already been formed and passed here by traversing directories and subdirectories where the paths are made.

char* filepath : The File path

void makeFile (char *fname, char * name)

Makes a Manifest file.

char* fname : Path of .Manifest

char* name:

void crtDir (char *name , char c)

Create a directory and makes .Manifest in it and in and initializes the file and write 0/n in the file

char* name : Name of Directory to be created.

char c:

void addManifest (char *progName , char *filePath , char * file, char *hashcode)

It adds the content to the .Manifest during the add function. A is appended in front i.e A <version> <path> <hashcode>

char* progName: The Program Name

char* filePath : The File path

char* file: The File

char* hashcode: The Hash Code

int locateTheFile (char *directory , char *file , char *d)

It locates the file by traversing the directories, if it finds the file then only it does it

char* directory: The directory

char* file : The File to be searched

char* d: The name of Directory

int removeLine (int fd, char* buffer , char* file)

It removes the content to the .Manifest. R is appended in front of it i.e R <version> <path> <hashcode>

int fd: The file descriptor

char* buffer : Content of .Manifest

char* file: The File

void opnDir (char * progName, char *file, char flag)

Opens a given directory and searches for the file if found along with a function based on the flag. Major purpose is to add or remove (depends on flag)

char* progName: The Program Name

char* file : The File

char flag: Determines whether to Add (A) or Remove (R)

int IsPathValid (char *directory, char* file)

It returns 1 for success and 0 for failure. checks if. Commit and .Update files are there or not. If there, then empty or not empty.

char* directory: The directory

char* file : The File

WTF SERVER ----------

int getFileStartIdx (char *path)

Gets the path of the Directory and chooses the File form it..

char*path: The Path of the directory

void OpenDir (char *s)

Returns 0 if it is needed to make a directory. Returns 1 if it already exists.

char* s: Path to directory

void createDir (char *path)

Creates a directory if it does not exists

char*path: The Path of the directory

int IsPathValid (char* directory, char* file)

Returns -1 if .Commit does not exist. Returns 0 if .Commit is empty, Returns 1 if .Commit has data

char* directory: The directory

char* file : The File

char* updateManifest (int fd,int len)

It is going to read a .Manifest and tokenize each line to store the data (tag, pathname, version, hashcode) in the linked list.

int fdp: .Commit file descriptor

int len: *******

void deleteNode(char *path)

It will find this path in the linklist and it will delete wherever it is found

char *path: The file Path

void addNode(char *pathName,char * version,char *hashcode)

Creates a new Node with the given parameters and adds it to the end of the linked list.

char *pathName : The File path

,char * version : The given version

char *hashcode: The hash Code

void findAndUpdate(char* version,char *path)

It will find the node in the linked list and update it.

char * version : The version

char * path : The path

void updateManifestLL()

Traverse through the Manifest Linked List and perform the actions

char *getCode(char *path)

Return the tag at the path given in parameter .

char *path : The path to the file

void addToList(char *t,char *p, char *h,char *v)

It will create a new node with given parameters and add it to the Commit link list.

char *t : The tag

,char *p : The path

char *h: The hash Code

char *v: The Version

void addToMList(char *v,char *p, char *h)

It will create a new node with given parameters and add it to the Manifest link list.

,char *p : The path

char *h: The hash Code

char *v: The Version

void serverManifestLL(char *manifest)

Tokenize the given manifest and send it to the add to manifest method.

char *manifest: The manifest

void CommitLL(char *manifest)

Tokenize the given Commit and send it to the add to commit method.

char *manifest: The manifest

char *getContent(int fd)

Opens the file descriptor, reads it and returns the content .

int fd : the file descriptor

void makeFile(char *fname,char *name)

Makes a Manifest file.

char* fname : Path of .Manifest

int duplicateDir(char *directory,char *verDir)

Creates a duplicate Directory and returns 1 if success and 0 if failure.

char *directory : The directory

char *verDir : The version of dir

char* getProjectVersion(char *path)

Return the version at a given path.

char *path : The Path

char *getPath(char *dir, char *fileName)

It returns the path to the file name.

char *dir : The directory

char *fileName : The Name of the file

char * getUpdatedManifest(int *socketfd)

It reads the message from the socket based on the protocol reads till ":" whatever is read till then say 56 then 56 bytes are read and returned.

int *socketfd: address of socket descriptor.

int removeDir(char *directory)

Removes the directory at the given path and return 1 if success and 0 if failure

char *directory : The directory

int sendManifest(int *sockefd, char *dir)

It finds and sends the maniset of directory to the given socket.

int *sockefd : The target Socket

char *dir : The directory

int getBytes(int *start, char delimeter)

Start and return the bytes till the delimeter is reached.

int *start : The start pointer

char delimeter : The end point (delimeter)

void create(char *fname,int *sockfd)

creates a dir and returns 1 if success to the client .

char *fname : The file name

int *sockfd : Socket descriptor

char* readProtocol(int *sockid)

read the protocol from client from the given socket id

int *sockid : Socket descriptor

char *currVersion(char * Dname, int *sockfd)

Reads manifest from Dname and return to the socket.
char * Dname: Directory Name
int *sockfd: Socket descriptor

void *client_handler(void *arg)

Create a thread to handle the clients.

void *arg : Socket descriptor