

# CS 421 – Computer Networks Programming Assignment 1 *PseudoGit*

**Due: November 06, 2024 at 11:59 PM**

In this assignment, you are asked to write a Python or Java program to clone a git repository, make changes to the downloaded files, and send a pull merge request to the repository. Commonly, git commands are used to execute these operations. However, for this programming assignment, you cannot use any git related libraries or commands. So, you will implement the git commands from scratch. The assignment consists of 2 main parts, where the details are given in the following sections. All the communication between the program and GitHub has to be done over HTTP without any third-party git related libraries. Furthermore, any third-party HTTP client libraries, or the HTTP specific core or non-core APIs are not allowed. The goal of this assignment is to familiarize you with the internals of the HTTP protocol, and using any (third party, core or non-core) API that provides any level of abstraction specific to the HTTP protocol is not allowed. You must implement your program using either the Java Socket API of the JDK or the Socket package in the Python distribution. For Python, using any class/function from HTTP or the requests package is prohibited. You can use the SSL package (ssl for Python and javax.net.ssl for Java) to wrap your HTTP request. Furthermore, you can use the JSON package to parse HTTP responses. If you have any doubts about what to use or not, please contact [bora.tuncer@bilkent.edu.tr](mailto:bora.tuncer@bilkent.edu.tr) or [furkan.guzelant@bilkent.edu.tr](mailto:furkan.guzelant@bilkent.edu.tr).

## **Task 1: Cloning a repository**

Typically, a GitHub repository can be cloned to the local computer with the command *git clone <repository url>*. Next to downloading the contents of the repository, it configures a working directory which is a checkout of the latest commit on the default branch. In this assignment, you are going to implement a watered-down version of this operation,

which, in reality, is quite complex and hard to implement with bare HTTP. For this task, the steps are as follows:

- 1) If you don't have a GitHub account, create one at [github.com/signup?source=login](https://github.com/signup?source=login)
- 2) Go to [https://github.com/CankutBoraTuncer/CS421-PA1\\_Fall2024](https://github.com/CankutBoraTuncer/CS421-PA1_Fall2024), and from the right part of the page, click **Fork**. This will create a public copy of the given repository which belongs to you. The folder contains several different files, such as .txt, .png, .py, .etc.
- 3) When you are sending HTTP messages to retrieve the files, git would require an **authentication token**. To generate this token, click your profile picture located in the top right corner and go to **Settings -> Developer Settings -> Personal Access Tokens -> Tokens (classic)**. Then click **Generate new token -> Generate new token (classic)**. Give an appropriate name and **enable all** the boxes under the **Select scopes**. After clicking **Generate Token**, your token will be shown. Be sure that you save that token to some .txt file since you cannot see it again, and if you lose your token, you will need to generate it from scratch.
- 4) Implement the HTTP communication between the program and the GitHub. While downloading the files, you would need to first retrieve the repository content and then ask for the individual files. **During grading, we will check your codes with another repository, and the contents of the repository may not be the same as the given one.** So, write your codes in a general manner.
- 5) In the repository, some files could be larger than **1MB**. In that case, you would need to **download those files in parallel** by dividing them into chunks. For instance, given a test.txt file, which is 1.5MB, and a parallel download count of 5, you would need to concurrently download 5 equal parts of the file using 5 different threads and HTTP range requests. This parallel download count parameter will be given as a console command argument.
- 6) The downloaded files should be functional, if a .py file is in the repository after the download, that python file should work on your computers.

## Task 2: Git Pull Merge Request

In the next task, you will make some changes in the cloned repository and send a pull merge request to the repository. The pull merge request is a collaboration feature in GitHub that is used to propose and discuss changes to a repository before merging them into the main branch. The main steps are as follows:

- 1) In the .py file, you will find a line as **print("Hello World")**. Change the string inside the print function with your student ID like **print("12345678")**.
- 2) Create a branch with your ID number using HTTP requests. A branch is the working directory in GitHub. The default base branch is the **main**. To create a branch, you will need the latest commit SHA from the base branch.
- 3) Push your changes to the newly created branch in GitHub by sending an HTTP request. If the file with the same name exists in the remote repository, you should get the SHA of the file and use it to update it.
- 4) Create a pull merge request with the title "New Pull Request" and body "Cool new feature!" from your branch with your ID number to the main branch via HTTP.
- 5) Get the pull request number by listing pull requests with open status by sending an HTTP request. Finally, merge the pull request from your branch with your ID number and send it to the main branch via HTTP.

Your program must be a console application (no graphical user interface(GUI) is required) and should be named PseudoGit (i.e., the name of the class that includes the main method for Java should be PseudoGit). Your program should run with the

```
java PseudoGit <command> <repository> <branch> <file_name> <pr_number>  
                <parallel count>
```

or

```
python PseudoGit.py <command> <repository> <branch> <file_name> <pr_number>  
                  <parallel count>
```

command where *<command>*, *<repository>*, *<branch>*, *<file\_name>*, *<pr\_number>*, and *<parallel count>* are the command-line arguments.

The details of the command-line arguments are as follows:

**<command>**: [Required] The command to perform operations in Git.

**clone**: Download files that are missing or have been changed remotely from the remote repository. If a file exists locally and its size is the same as the remote version, it should not be downloaded again.

**branch**: Create a new branch.

**upload**: Upload a file to the remote repository.

**create-pr**: Create a pull request.

**list-pr**: List the pull requests with open status.

**merge-pr**: Merge the pull request.

**<repository>** [Required] The name of the repository where the operation will be performed (e.g. *<username>/<repository\_name>*)

**<branch>** [Optional] For the branch operation, it specifies the name of the branch that will be created. For the upload operation, it specifies the branch to which the file will be uploaded. The create-pr operation specifies the head branch that will be merged into the base branch, with the main being the default base branch in this assignment.

**<file\_name>** [Optional]: The name of the file that will be uploaded. The local files are assumed to have the same names as the corresponding files in the remote repository.

**<pr\_number>** [Optional]: The number of the pull request to be merged. The pull request number can be found in the Pull Request tab in the GitHub repository, in the title of the pull request, after the # symbol. Also, you should be able to see the open pull requests with the list-pr operation that you will implement.

**<parallel\_count>** [Optional]: When cloning the repository, if a file is over 1 MB, the file should be downloaded using parallel TCP connections. If the parallel\_count is given as 5, the program should open 5 threads and download 5 equal-sized (if the file size is not

divisible by 5, the last part can be the remaining leftover) parts of the file in parallel using HTTP range requests. If the *<command>* is clone, this parameter is **required**.

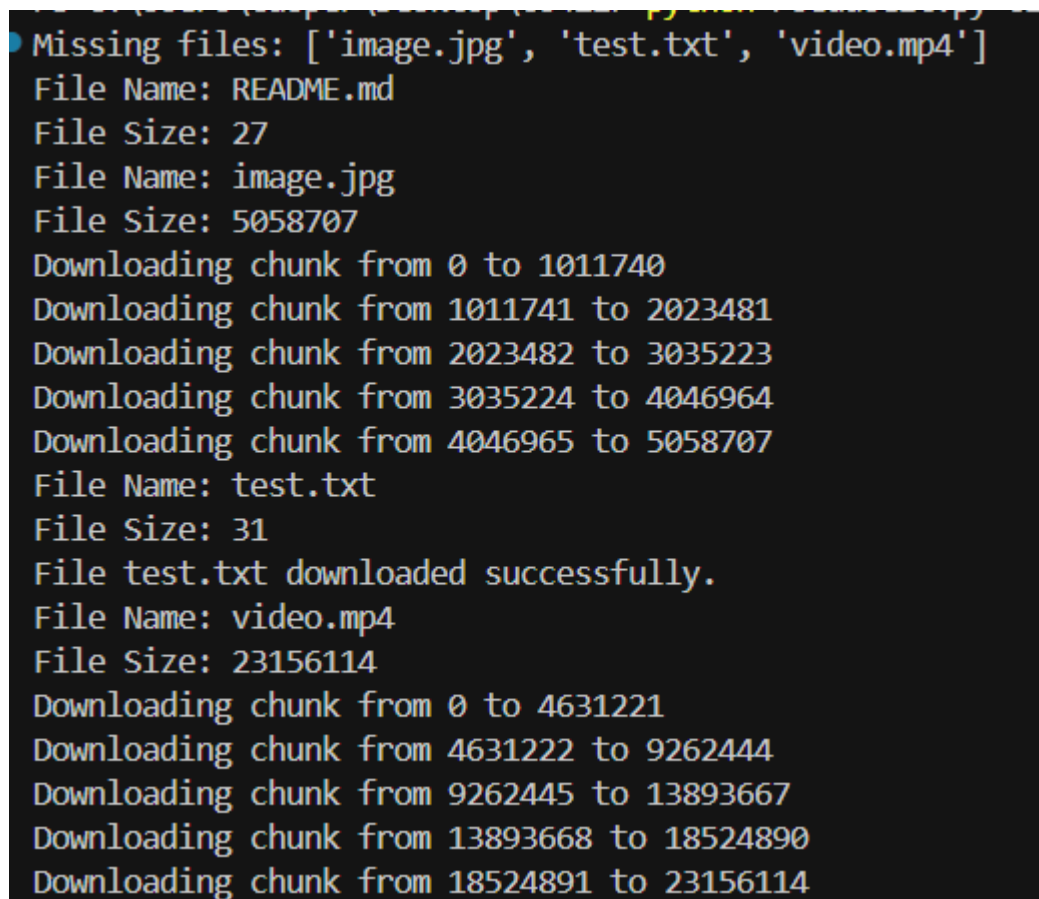
You can use the java.util.Base64 library in Java, and base64 module in Python when you need Base64 encoding.

You can use the SSL package (ssl for Python and javax.net.ssl for Java) to wrap your HTTP request. Furthermore, you can use json package to parse the HTTP responses.

## Example Commands and Outputs

### 1. Clone

```
python PseudoGit.py clone <username>/<repository_name> <parallel_count>
```



```
Missing files: ['image.jpg', 'test.txt', 'video.mp4']
File Name: README.md
File Size: 27
File Name: image.jpg
File Size: 5058707
Downloading chunk from 0 to 1011740
Downloading chunk from 1011741 to 2023481
Downloading chunk from 2023482 to 3035223
Downloading chunk from 3035224 to 4046964
Downloading chunk from 4046965 to 5058707
File Name: test.txt
File Size: 31
File test.txt downloaded successfully.
File Name: video.mp4
File Size: 23156114
Downloading chunk from 0 to 4631221
Downloading chunk from 4631222 to 9262444
Downloading chunk from 9262445 to 13893667
Downloading chunk from 13893668 to 18524890
Downloading chunk from 18524891 to 23156114
```

### 2. Branch

```
python PseudoGit.py branch <username>/<repository_name> test-branch
```

The newly created branch should appear on the Branches page in GitHub.

### 3. Upload

```
python PseudoGit.py upload <username>/<repository_name> test-branch test.txt
```

The given file should be uploaded to a branch named “test-branch”.

#### 4. Create Pull Request

```
python PseudoGit.py create-pr <username>/<repository_name> test-branch
```

You should be able to see the created pull request in the Pull Request tab in GitHub.

#### 5. List Pull Request

```
python PseudoGit.py list-pr <username>/<repository_name>
```

The created pull request should be listed as follows:

PR #1: New Pull Request (state: open)

#### 6. Merge Pull Request

```
python PseudoGit.py merge-pr <username>/<repository_name> 1
```

The pull request should be merged with the main branch and closed. The file in the main branch should be updated.

As a **bonus assignment**, we expect you to add additional features and mechanisms to PseudoGit. We will evaluate these features based on creativity and implementational complexity. The bonus part is not mandatory, and the grading of this part is separate from the original grading scheme. The bonus points that you will get will be added to the original grade.

Lastly, write a **report** (PDF file) in which you explain the important parts of your code. We should be able to navigate the source code just from the report. The number of pages should not exceed 5.

## Assumptions and Hints

- You should be able to handle all the described Git operations using HTTP requests. Using Git commands or using the GitHub interface for this assignment is not allowed.
- Please refer to [GitHub RestAPI documentation](#) to find the corresponding endpoints for downloading and uploading files, as well as pull request operations.

- For threading, you can check [Python Thread Tutorial](#) or [Java Thread Tutorial](#).

For Threading in Python:

1. Official Python Documentation: Start with the threading module in Python's official documentation. It provides a comprehensive overview and examples.
  - Visit: [Python Threading Module](#)
2. TutorialsPoint: Offers clear, beginner-friendly tutorials on threading in Python.
  - Visit: [TutorialsPoint - Python Threading](#)
3. Real Python: Find in-depth tutorials and examples here that cover threading basics and advanced topics.
  - Visit: [Real Python - An Intro to Threading in Python](#)

For Threading in Java:

1. Official Java Tutorials: The Concurrency lessons in the Java tutorials provide a solid foundation.
    - Visit: [Oracle Java Docs - Concurrency](#)
  2. GeeksforGeeks: A good resource for Java threading tutorials that range from beginner to advanced levels.
    - Visit: [GeeksforGeeks - Multithreading in Java](#)
  3. JavaPoint: Offers tutorials on threading with simple examples to understand the basics.
    - Visit: [JavaPoint - Multithreading in Java](#)
- For SHA, check [GitHub SHA](#).
  - You can assume that no directories are involved; all operations are performed on individual files with a flat file structure.
  - You can assume the base branch for the merge operation to be the main branch.
  - To download a file in several chunks, you need to specify the 'Range' header, indicating the start and end bytes of the chunk. If the chunk is downloaded successfully, the response status should be 206 (Partial Content).
  - Please contact your assistant if you have any doubts about the assignment.

## Submission rules

You need to apply all the following rules in your submission. **You will lose points if you do not obey the submission rules below or your program does not run as described in the assignment above.**

- The assignment should be uploaded to the Moodle in a zip file. Any other methods of submission will not be accepted.
- The name of the submission should start with [CS421\_PA1], and include your name and student ID. For example, the name must be  
`[CS421_PA1]AliVelioglu20111222`  
if your name and ID are Ali Velioglu and 20111222. You are not allowed to work in groups.
- All the files must be submitted in a zip file whose name is described above. The file must be a .zip file, not a .rar file or any other compressed file.
- All of the files must be in the root of the zip file; directory structures are not allowed. Please note that this also disallows organizing your code into Java packages. The archive should not contain:
  - Any class files or other executables,
  - Any third-party library archives (i.e. jar files),
  - Any text files,
  - Project files used by IDEs (e.g., JCreator, JBuilder, SunOne, Eclipse, Idea or NetBeans etc.). You may, and are encouraged to, use these programs while developing, but the end result must be a clean, IDE-independent program.
- The standard rules for plagiarism and academic honesty apply; if in doubt refer to [Academic Integrity Guidelines for Students](#) and [Academic Integrity, Plagiarism & Cheating](#).



# Important Notes

## General Submission Rules:

- 1- ) Your submission should include source code(s) (.java/.py).
- 2- ) The format of the report should be **PDF**. (Do not upload .doc, .docx or any other types). The number of pages should not exceed 5.
- 4- ) Your submission should not contain any other files other than the source code(s) (.java/.py), report (.pdf) and optional README. No .txt files, no folders, no IDE related files should be included.
- 5-) Compress these files with **.zip** format. (.rar, .7z or any other compressing types will not be accepted.)
- 6-) Make sure to follow rules in the “Submission Rules” section like name of the zip file, method of the submission etc.

## For Python Submissions:

- 1-) The code should run with the “python PseudoGit.py <command> <repository> <branch><file\_name><pr\_number>” command.
- 2-) Python version should be **3.6 or higher**. Other versions (like Python 2) are **not accepted**.

## For Java Submissions:

- 1-) The code should run with the following commands:  
  
Compile: “javac \*.java”  
  
Run: “java PseudoGit <command> <repository> <branch><file\_name><pr\_number>” command.
- 2-) Java version should be **8 or higher**.
- 3-) The JDK should be Oracle JDK (**not** OpenJDK).