SGit

Project Report

By Soufiane Benchraa

Instructions

Dependencies

Running the application requires the installation of the following dependencies:

- sbt version ..
- scala version..

Download

Clone the git repository by executing the following command:

```
git clone https://github.com/sbenchra/SGit.git
```

Build

I created a bash script allowing the user to build the project and create the jar file, and also use the sgit command where he wants,by executing the following command :

```
source build.sh
```

⚠ *Do the same operation, if you want to use the application in another shell window*

Build

To run the application you need to execute the following commands :

Example:

```
sgit command
```

To use sgit you should first start by initializing you repository :

```
sgit init
```

To stage a file or directory :

```
sgit add filenames or directories names
```

To commit your staged files:

```
sgit commit -m “message”
```

To display the status of your working directory :

```
sgit status
```

To display the changes made on the staged files:

```
sgit diff
```

To checkout to a different branch or commit or tag:

```
sgit checkout tagname or commit sha1 or branchname
```

To display the working directory the details of changes above time:

```
sgit log --p
```

To display all the commits:

```
sgit log
```

To display all created branches statistics above time:

```
sgit branch --av
```

To display all commits statistics above time:

```
sgit log --stat
```

To create a new tag :

```
sgit tag tagname
```

To create a new tag :

```
sgit tag tagname
```

Architecture

In the purpose of carrying out the project, I tried to purely implement the application, by respecting the functional programming best practices and design patterns that I have learnt during courses, such as purity, immutability and match pattern.

After starting to learn the git anatomy, and taking time for a brainstorming I designed the sgit following a layered architecture. I separated the program into 3 separated layers. Each layer provides information to the layer above.

The first layer is the presentation layer or the user interface layer. This layer is responsible for managing user input and calling the adequate business classes.

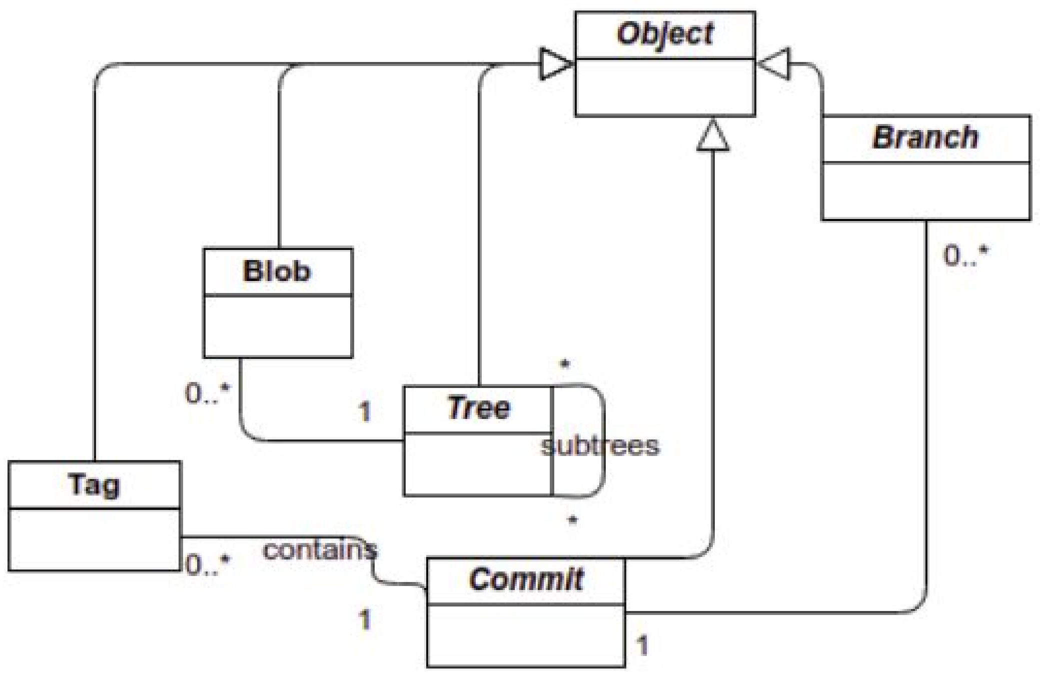
The second layer is the business layer which has two types of classes : commands and objects(Object-Tree-Commit-Tag-Branch-ObjectBL). For each command that can be called by the user, there is a business class that has all the functions and logic to execute the command. These classes use Objects to fulfill their job.

The last layer is the data layer, which consists of a utility class responsible for exchanging with the filesystem to make file input and output.

The advantage of this architecture is that it helps organize and simplify the code . Each layer manages a certain part of the application separately. It allows a separated testability, and makes the update of any legacy system easier, so the changes that need to be done, would be simpler and less complex.

In the other hand, the disadvantages of the architecture that use is ,the more we add layers, the lower the performance will decrease, and also the interdependence among layers, which means any modifications in one of the layers means, changing another one.

The figure below explains, the design that I used to manage git immutable objects.



Objects class diagram

Tests

Sbt plugin, allowed me to automate my running tests in IntelliJ, so I implemented many unit tests to test my most important functions, to be sure of their features.

For my tests, I used a hosted continuous integration service which travis CI. After every commit Travis CI sends me an email notification containing the results of my tests.

Post mortem

First of all, the pure implementation of a git-like was a great challenge, especially since my knowledge of functional programming was limited, but I took the time to deepen my understanding of what I learned in class, which allowed me to improve my skills in scala language and functional programming.

In the beginning, I tried to understand the anatomies of git in order to create an architecture that would allow me to feel comfortable when coding. My understanding of git anatomy and programming was continuous throughout the project. This helped me to understand the internal workings of git.

My objective was to purely implement sgit by using design patterns and best practices in functional programming, by using immutability, recursion and match pattern. I would liked to use options to manage exceptions but I didn't, and it is one of my mistakes that I consider as lessons.

I didn't have time to finish developing all the features, despite the fact that I put a lot of investment into the project. There was many cases to handle for each feature, which made the workload significant. Despite that, I tried to implement as many features as possible, while carrying out a flexible code that I could easily change, if necessary.

To conclude, this project was an opportunity to put into practice my skills acquired in class.
In addition, I have learned a lot of good practices, not only in terms of coding but also in terms of organization, which saves a lot of time.
The project is not finished yet, but I am proud of the effort I have done, and the result that I have reached, and also proud to be able to add Sgit implementation in my resume.