

SKRIPSI

**PEMBANGKIT TIMELAPSE PENGEMBANGAN PROYEK
PERANGKAT LUNAK BERBASIS WEB**



Billy Adiwijaya

NPM: 2015730053

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2019**

UNDERGRADUATE THESIS

**TIMELAPSE GENERATOR FOR WEB-BASED SOFTWARE
PROJECT DEVELOPMENT**



Billy Adiwijaya

NPM: 2015730053

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2019**

LEMBAR PENGESAHAN

PEMBANGKIT TIMELAPSE PENGEMBANGAN PROYEK PERANGKAT LUNAK BERBASIS WEB

Billy Adiwijaya

NPM: 2015730053

Bandung, «tanggal» «bulan» 2019

Menyetujui,

Pembimbing

Pascal Alfadian, M.Comp.

Ketua Tim Penguji

Anggota Tim Penguji

«penguji 1»

«penguji 2»

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PEMBANGKIT TIMELAPSE PENGEMBANGAN PROYEK PERANGKAT LUNAK BERBASIS WEB

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal «tanggal» «bulan» 2019

Meterai Rp. 6000

Billy Adiwijaya
NPM: 2015730053

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia»

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

«kepada siapa anda mempersembahkan skripsi ini...?»

KATA PENGANTAR

«Tuliskan kata pengantar dari anda di sini ...»

Bandung, «bulan» 2019

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xix
DAFTAR TABEL	xxi
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	1
1.3 Tujuan	1
1.4 Batasan Masalah	2
1.5 Metodologi	2
1.6 Sistematika Pembahasan	2
2 LANDASAN TEORI	3
2.1 Git	3
2.1.1 Version Control Systems	3
2.1.2 Cara Kerja Git	5
2.1.3 Operasi Dasar pada Git	8
2.1.4 Git Checkout	9
2.2 JGit	10
2.2.1 Repository	10
2.2.2 FileRepository	10
2.2.3 Git	11
2.2.4 RevWalk	12
2.2.5 RevCommit	13
2.2.6 PersonIdent	13
2.3 Selenium WebDriver	13
2.3.1 WebDriver	14
2.3.2 WebElement	14
2.3.3 OutputType	15
2.3.4 TakesScreenshot	15
2.4 Apache Commons CLI	16
2.4.1 CommandLineParser	16
2.4.2 CommandLine	16
2.4.3 Options	17
2.4.4 Option	17
2.4.5 Option.Builder	18
DAFTAR REFERENSI	19

A KODE PROGRAM	21
B HASIL EKSPERIMEN	23

DAFTAR GAMBAR

2.1	Local version control[1].	3
2.2	Centralized version control[1].	4
2.3	Distributed version control[1].	5
2.4	Menyimpan data sebagai <i>snapshots</i> dari <i>project</i> [1].	6
2.5	Menyimpan data sebagai perubahan terhadap versi dasar dari setiap <i>file</i> [1].	6
2.6	<i>Working tree</i> , <i>Staging area</i> , dan Git direktori[1].	7
2.7	<i>Checkout</i> pada <i>commit</i>	10
B.1	Hasil 1	23
B.2	Hasil 2	23
B.3	Hasil 3	23
B.4	Hasil 4	23

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Git merupakan perangkat lunak *Version Control Systems*[1]. *Version control* adalah sistem yang merekam perubahan pada *file* atau sekumpulan *file* dari waktu ke waktu. Perubahan yang terjadi pada *repository* dicatat oleh Git dalam bentuk histori *commit*. Setiap *commit* mengandung informasi mengenai perubahan yang terjadi pada *repository*, waktu perubahan, dan orang yang melakukan perubahan. *Database* pada *git* tidak bersifat terpusat, melainkan terdistribusi. Setiap orang yang terlibat mempunyai *database* lokal pada masing-masing komputer, sehingga pengelolaan perangkat lunak dapat dilakukan secara *online* dan *offline*.

JGit adalah *library* Java murni yang mengimplementasikan Git *version control systems*[2]. JGit dikembangkan oleh Eclipse Foundation. JGit bersifat *open source*. Dengan menggunakan JGit, fitur-fitur dalam Git dapat diakses melalui program Java.

Selenium adalah seperangkat alat yang secara khusus digunakan untuk mengotomatisasi *web browsers*[3]. Dengan menggunakan Selenium WebDriver, pengguna dapat memasukkan *script* bahasa pemrograman tertentu untuk melakukan pengujian. Bahasa pemrograman yang didukung yaitu C#, Java, Perl, PHP, Python, Ruby, dan JavaScript. Selenium WebDriver dapat melakukan pengujian pada *browser* Google Chrome, Mozilla Firefox, Opera, Safari, dan Internet Explorer.

Pada skripsi ini, akan dibuat sebuah perangkat lunak yang dapat menampilkan animasi *time-lapse* dari pengembangan proyek perangkat lunak berbasis web. Perangkat lunak ini dibangun menggunakan bahasa Java. Perangkat lunak ini menggunakan tampilan terminal/konsol. Dalam pembuatan animasi *timelapse*, dibutuhkan perangkat lunak Selenium WebDriver dan JGit.

1.2 Rumusan Masalah

Rumusan masalah dari skripsi ini adalah sebagai berikut:

1. Bagaimana cara membangkitkan animasi *timelapse* pada pengembangan proyek perangkat lunak berbasis web?
2. Bagaimana cara mengimplementasikan aplikasi untuk membangkitkan *timelapse* pada pengembangan proyek perangkat lunak berbasis web?

1.3 Tujuan

Tujuan dari skripsi ini adalah sebagai berikut:

1. Mengetahui cara untuk membangkitkan animasi *timelapse* pada pengembangan proyek perangkat lunak berbasis web.
2. Mengetahui cara untuk mengimplementasikan aplikasi untuk membangkitkan *timelapse* pada pengembangan proyek perangkat lunak berbasis web.

1.4 Batasan Masalah

Batasan masalah pada skripsi ini adalah sebagai berikut:

1. Perangkat lunak ini hanya membangkitkan animasi *timelapse* untuk perangkat lunak berbasis web.
2. Masukan perangkat lunak berupa alamat direktori proyek perangkat lunak yang terekam oleh Git.

1.5 Metodologi

Metodologi penelitian yang digunakan dalam skripsi ini adalah sebagai berikut:

1. Melakukan studi literatur tentang Git, Selenium WebDriver, Git, dan JGit.
2. Melakukan analisis penggunaan Selenium WebDriver dan JGit untuk membangkitkan animasi *timelapse*.
3. Merancang perangkat lunak.
4. Membangun perangkat lunak.
5. Melakukan eksperimen dan pengujian pada perangkat lunak.

1.6 Sistematika Pembahasan

Setiap bab dalam penelitian ini memiliki sistematika penulisan yang dijelaskan ke dalam poin-poin sebagai berikut:

1. Bab 1: Pendahuluan, yaitu membahas mengenai gambaran umum penelitian ini. Berisi tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metode penelitian, dan sistematika penulisan.
2. Bab 2: Dasar Teori, yaitu membahas mengenai teori-teori yang mendukung berjalannya penelitian ini. Berisi tentang teori Git, JGit, Selenium WebDriver, dan Apache Commons CLI.
3. Bab 3: Analisis, yaitu membahas mengenai analisa masalah. Berisi tentang analisis penggunaan Jgit dan Selenium WebDriver untuk membangkitkan animasi *timelapse*.
4. Bab 4: Perancangan, yaitu membahas mengenai perancangan yang dilakukan sebelum melakukan tahapan implementasi. Berisi tentang perancangan perangkat lunak pembangkit *timelapse* proyek pengembangan perangkat lunak.
5. Bab 5: Implementasi dan Pengujian, yaitu membahas mengenai implementasi dan pengujian aplikasi yang telah dilakukan. Berisi tentang implementasi dan hasil pengujian aplikasi.
6. Bab 6: Kesimpulan dan Saran, yaitu membahas hasil kesimpulan dari keseluruhan penelitian ini dan saran-saran yang dapat diberikan untuk penelitian berikutnya.

BAB 2

LANDASAN TEORI

Pada bab ini dibahas dasar teori yang mendukung berjalannya skripsi ini. Dasar teori yang dibahas yaitu Git, JGit, Selenium WebDriver, dan Apache Commons CLI.

2.1 Git

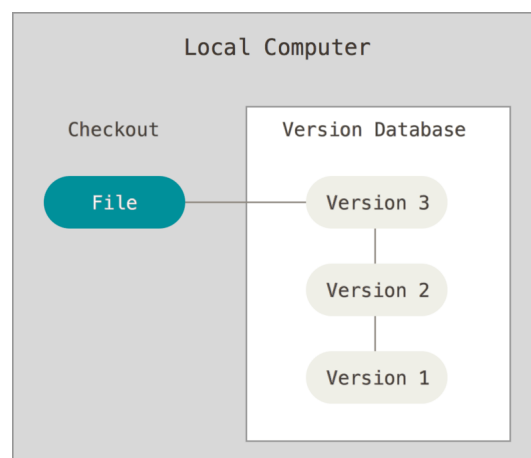
Git merupakan perangkat lunak *Version Control Systems*. Pada subbab ini, dijelaskan mengenai *Version Control Systems*, cara kerja Git, *Git checkout*, dan operasi-operasi dasar pada Git. Subbab ini mengacu pada [1].

2.1.1 Version Control Systems

Version Control Systems adalah sistem yang merekam perubahan pada *file* atau sekumpulan *file* dari waktu ke waktu. *Version Control Systems* biasanya digunakan untuk merekam *file* yang berisi *source code program*, tetapi pada kenyataannya *Version Control Systems* dapat merekam hampir semua jenis *file* dalam komputer. Terdapat tiga jenis *Version Control Systems*, yaitu: *Local Version Control Systems*, *Centralized Version Control Systems*, dan *Distributed Version Control Systems*.

Local Version Control Systems

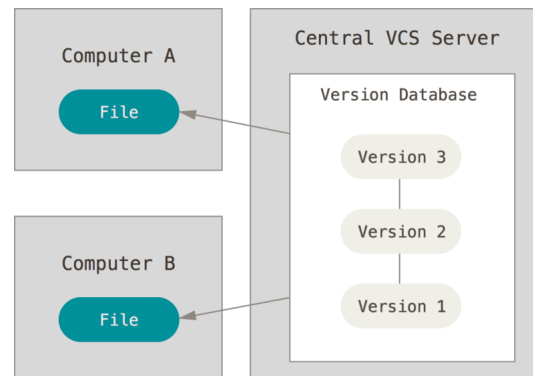
Metode *version-controlled* yang banyak digunakan orang adalah dengan cara menyalin sekumpulan *file* ke direktori lain. Namun cara tersebut rentan terhadap *error*. Misalnya, terdapat direktori A dan B, pengguna ingin mengubah *file* yang terdapat pada direktori B, tetapi pengguna lupa kalau dia sedang berada di direktori A, maka pengguna mengubah *file* pada direktori yang salah. Untuk mengatasi masalah tersebut, *programmer* mengembangkan *Local Version Control Systems*.



Gambar 2.1: Local version control[1].

Gambar 2.1 merupakan struktur dari *Local Version Control Systems*. *Database local Version Control Systems* ini tersimpan pada *local directory* di komputer. *Database* ini menyimpan perubahan *file* ke dalam beberapa versi atau *state*. *Local Version Control*, dapat melakukan *checkout file* ke versi atau *state* tertentu.

Centralized Version Control Systems



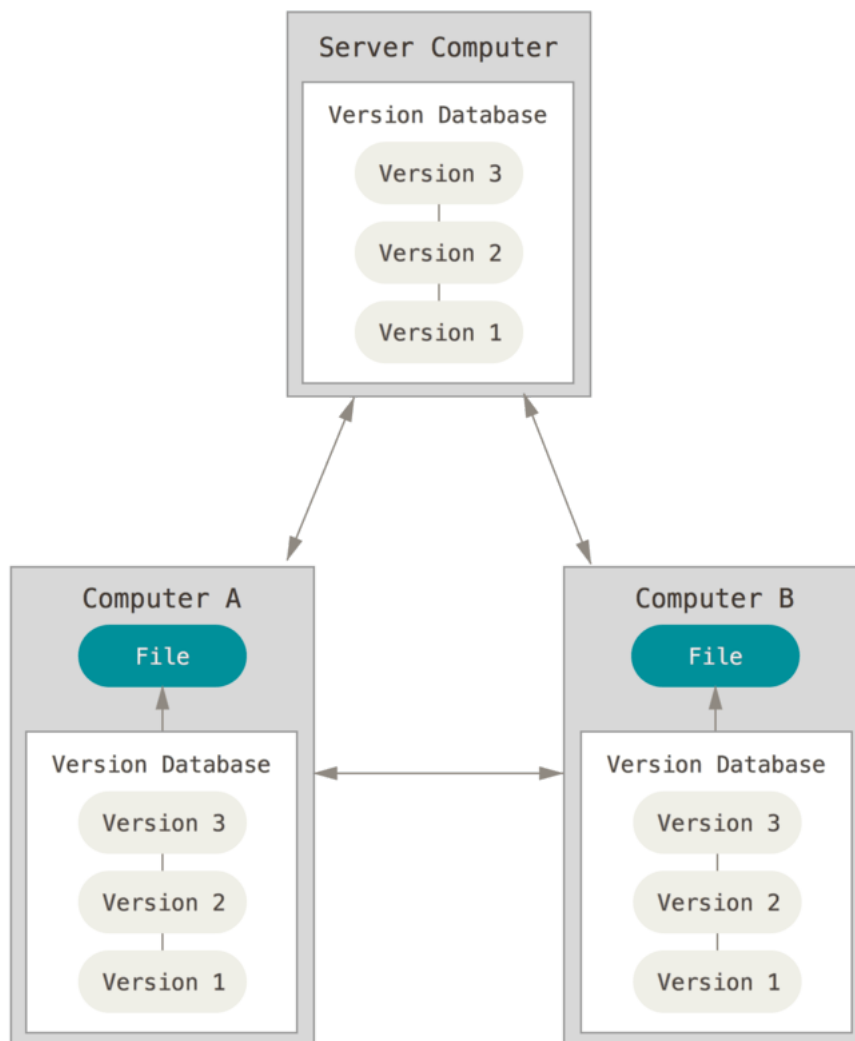
Gambar 2.2: Centralized version control[1].

Local Version Control hanya menyimpan *file* pada satu komputer saja. Muncul masalah baru ketika *user* ingin berkolaborasi dengan *user* lain. Untuk mengatasi masalah ini dikembangkan *Centralized version control*. Gambar 2.2 merupakan struktur dari *Centralized Version Control Systems*. Dalam *Centralized Control Version Systems* terdapat sebuah *server* yang menyimpan setiap versi *file*, dan klien yang dapat melakukan *checkout file*.

Sistem *Centralized Version Control Systems* memiliki beberapa kelebihan. Setiap *user* dapat mengetahui pekerjaan yang dilakukan oleh *user* lain. Administrator dapat lebih mudah mengontrol *database Centralized Version Control Systems* dibandingkan dengan *database Local Version Control Systems* dari setiap klien.

Sistem *Centralized Version Control Systems* memiliki kelemahan. Jika *server* pusat *Centralized Version Control Systems* mati, maka perubahan pada *file* tidak bisa disimpan. Klien juga tidak dapat melakukan kolaborasi dengan klien lain. Jika *harddisk* pada server rusak, maka semua versi *file* akan hilang.

Distributed Version Control Systems

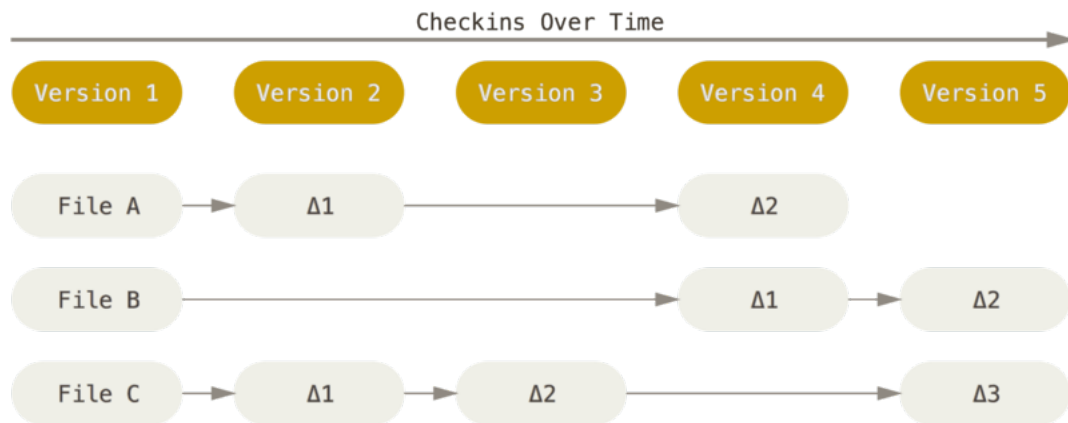


Gambar 2.3: Distributed version control[1].

Gambar 2.3 merupakan struktur dari *Distributed Version Control Systems*. Dalam sebuah DVCS (seperti Git, Mercurial, Bazaar atau Darcs), klien tidak hanya melakukan *checkout* untuk *snapshot* terakhir setiap *file*, namun klien juga memiliki salinan dari repositori tersebut. Dengan kata lain setiap klien memiliki *version database local* pada komputernya. Jika server pusat mati, klien masih bisa melakukan kolaborasi dan klien manapun dapat mengirimkan kembali salinan repositori ke *server*.

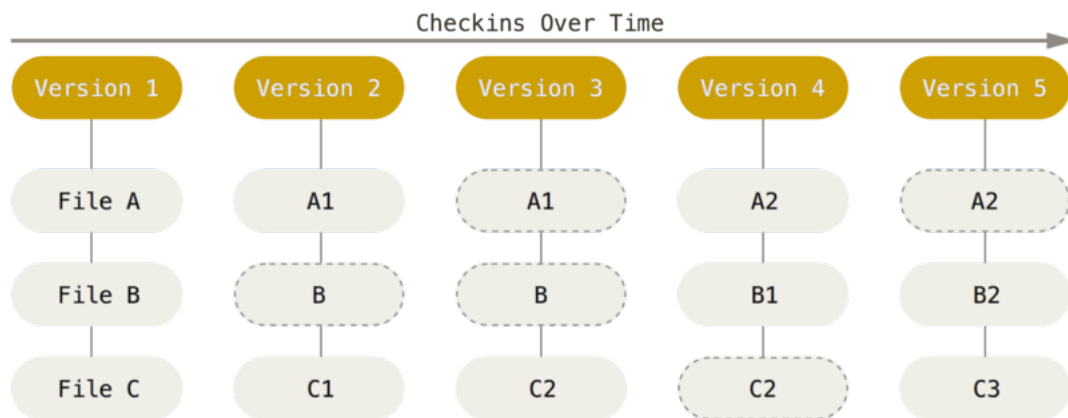
2.1.2 Cara Kerja Git

Salah satu perbedaan antara Git dengan VCS lainnya adalah dalam cara Git memperlakukan datanya. Kebanyakan sistem *Version Control Systems* lain menyimpan informasi sebagai daftar perubahan *file*. Pada Gambar 2.4, terdapat tiga *file*. *Version Control Systems* menyimpan *file* A, B, dan C pada versi pertama saja. Untuk versi kedua dan seterusnya yang disimpan adalah perubahan pada setiap *file*. Sistem ini disebut juga sebagai *delta-based Version Control Systems*.



Gambar 2.4: Menyimpan data sebagai *snapshots* dari *project*[1].

Berbeda dengan *Version Control Systems* lainnya, Git memperlakukan datanya sebagai sebuah kumpulan *snapshot* dari sebuah miniatur *file system*. Setiap kali dilakukan *commit*, git merekam *state* dari sekumpulan *file* dan menyimpanannya sebagai *reference snapshot* tersebut. Gambar 2.5, menunjukkan *snapshots* dari *file* A, B, dan C. Pada versi kedua, *file* B tidak mengalami perubahan, sehingga *file* yang disimpan adalah referensi *file* B pada versi sebelumnya.



Gambar 2.5: Menyimpan data sebagai perubahan terhadap versi dasar dari setiap *file*[1].

State pada Git

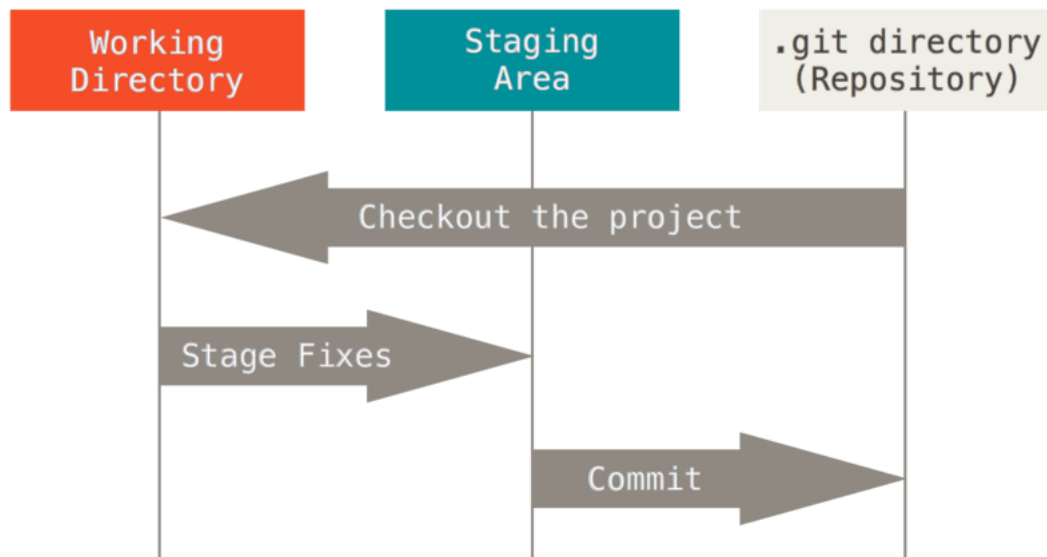
Terdapat tiga *state* pada Git yaitu *committed*, *modified*, and *staged*. *Committed* adalah *state* dimana data sudah disimpan di *local database*. *Modified state* dimana terdapat perubahan pada *file*, namun *file* tersebut belum di *commit* ke *database*. *Staged* adalah *state* dimana *file* telah ditandai untuk kemudian dilakukan *commit*.

Terdapat tiga bagian utama dari sebuah *project* Git yaitu direktori Git, *working directory*, dan *staging area*. Direktori Git merupakan tempat dimana Git menyimpan *metadata* dan *object database* dari *project*. *Working tree* adalah suatu *snapshot* dari *project*. Sekumpulan *file* ini diambil dari *database* di direktori Git dan ditempatkan pada *disk* untuk digunakan dan dimodifikasi. *Staging area* adalah *file* yang menyimpan informasi mengenai apa yang menjadi *commit* selanjutnya. *File staging area* terdapat pada direktori Git. Untuk lebih jelasnya, lihat Gambar 2.6.

Alur kerja dari Git adalah sebagai berikut:

1. Melakukan modifikasi pada *file*.
2. Menandai perubahan pada *file* dan memindahkannya ke *staging area*.

3. Mengambil *file* dari *staging area* dan menyimpan *snapshot* ke direktori Git. Proses ini disebut dengan *commit*.



Gambar 2.6: *Working tree*, *Staging area*, dan Git direktori[1].

Commit

Commit merupakan sebuah *snapshot* dari suatu *file* atau direktori. *Commit* menggambarkan *state* dari *working directory*. Gambar 2.5 menunjukkan terdapat tiga *file* pada versi/*commit* keempat. Dimana terdapat *file* A1, B1, dan C1 pada *working directory*. *File* A1, B1, dan C2 merupakan *state file* A, B, dan C pada *commit* keempat.

Git melakukan *check-summed* pada *commit* sebelum menyimpannya ke Git repositori. Mekanisme yang digunakan untuk melakukan *check-summed* disebut dengan *SHA-1 hash*. *SHA-1 hash* terdiri dari empat puluh karakter heksadesimal(0-9 a-f). Nilai dari *SHA-1 hash* dihitung berdasarkan isi dari *working directory* atau struktur direktori Git.

Listing 2.1: Contoh histori commit dalam pengembangan perangkat lunak

```

1 C:\Users\user\Documents\GitHub\train-tracker-ellena-angelica>git log
2
3 commit b8aeacbd4743619b7b2d790d45bde26b899641e0 (HEAD -> master, origin/master,
   origin/HEAD)
4 Author: adamadamadamadamadam <adamnurmishwari@gmail.com>
5 Date: Thu May 3 01:15:31 2018 +0700
6
7     commitan terakhir. mastiin g buang memory sm batre
8
9 commit f836cc65bf6d50e274df54aa06c6fb529667aa06
10 Author: Evelyn Wijaya <evelynwijaya777@gmail.com>
11 Date: Wed May 2 22:03:25 2018 +0700
12
13     Update README.md
14
15 commit 2e1ce9a03a1f417326c3c6586503303cf6daf6b8
16 Author: Evelyn Wijaya <evelynwijaya777@gmail.com>
17 Date: Wed May 2 22:01:10 2018 +0700
18
19     Create README.md
20
```

```

21 | commit 2f04488f9008745e8e6f67da33ffb2f6c2c9e747
22 | Author: Evelyn Wijaya <evelynwijaya777@gmail.com>
23 | Date:   Wed May 2 14:08:02 2018 +0700
24 |
25 |     fix stasiun double
26 |
27 | commit 7d8b66a9c6500de2753cdeac1084dc049c0c9f20
28 | Author: Evelyn Wijaya <evelynwijaya777@gmail.com>
29 | Date:   Wed May 2 13:32:21 2018 +0700
30 |
31 |     fix stasiun double

```

Seperti yang diperlihatkan pada Listing 2.1, setiap *commit* memiliki beberapa informasi. Baris pertama menunjukkan *commit* ID yang berupa *SHA-1 hash*. Pada baris ini, *Master* menunjukkan *branch* yang sedang aktif, *master* juga merupakan *pointer* ke *commit* terakhir. *Head* merupakan *reference* ke *branch master*. *Origin/master* dan *origin/HEAD* merupakan *master* dan *HEAD* pada *remote repository*. Baris kedua menunjukkan orang yang melakukan *commit* dan alamat emailnya. Baris ketiga menunjukkan waktu terjadinya *commit*. Baris terakhir berisi deskripsi dari *commit* tersebut.

2.1.3 Operasi Dasar pada Git

Pada subbab ini dijelaskan mengenai operasi dasar dalam Git dan sintaks-sintaksnya. Sintaks-sintaksnya ini dimasukkan pada Git *command line*. Berikut ini adalah operasi-operasi dasar dalam Git:

1. Init

Operasi ini digunakan untuk membuat repositori lokal baru dengan nama tertentu. Bisa juga digunakan untuk merekam direktori yang sudah ada. Berikut adalah sintaks untuk melakukan operasi *init*:

```
$ git init [project-name]
```

2. Add

Operasi ini digunakan untuk menandai perubahan pada *file* dan memindahkan *file* tersebut ke *staging area*. Operasi ini juga digunakan untuk menambahkan *file* yang dipantau perubahannya. Berikut adalah sintaks untuk melakukan operasi *add*:

```
$ git add [file]
```

3. Commit

Operasi ini digunakan untuk merekam *snapshot* atau *state file* atau sekumpulan *file*. Operasi ini juga digunakan untuk memindahkan *file* yang berada di *staging area* ke repositori Git. Berikut adalah sintaks untuk melakukan operasi *commit*:

```
$ git commit -m "[descriptive message]"
```

4. Branch

Operasi ini digunakan untuk menampilkan semua *branch* yang ada pada repositori Git, membuat *branch* baru, dan menghapus *branch*. Berikut adalah sintaks-sintaks untuk melakukan operasi *branch*:

```

$ git branch
$ git branch [branch-name]
$ git branch -d [branch-name]
$ git branch -D [branch-name]

```

5. Diff

Operasi ini digunakan untuk menampilkan perbedaan pada *file* yang belum masuk *staging area*, menampilkan perbedaan pada *file* yang berada di *staging area* dengan *file* di *commit*

sebelumnya, dan perbedaan *file* antara dua *branch*. Berikut adalah sintaks-sintaks untuk melakukan operasi *diff*:

```
$ git diff
$ git diff -staged
$ git diff [first-branch]...[second-branch]
```

6. Clone

Operasi ini digunakan untuk menyalin repositori Git yang berada di komputer lain atau suatu *server*. Berikut adalah sintaks untuk melakukan operasi *clone*:

```
$ git clone [url]
```

7. Fetch

Operasi ini digunakan untuk mengambil data dari *remote* repositori ke repositori lokal. Berikut adalah sintaks untuk melakukan operasi *fetch*:

```
$ git fetch [bookmark]
```

8. Merge

Operasi ini digunakan untuk menggabungkan *branch* tertentu dengan *branch* yang sedang aktif. Operasi ini juga digunakan untuk menggabungkan data yang diambil dari *remote* repositori dengan data pada *working directory*. Berikut adalah sintaks untuk melakukan operasi *merge*:

```
$ git merge [branch]/[bookmark]
```

9. Pull

Operasi ini adalah gabungan dari operasi *fetch* dan *merge*. Berikut adalah sintaks untuk melakukan operasi *pull*:

```
$ git pull
```

10. Push

Operasi ini digunakan untuk mengirim data pada repositori Git lokal ke *remote repository*. Berikut adalah sintaks untuk melakukan operasi *push*:

```
$ git push [alias] [branch]
```

11. Checkout

Operasi ini digunakan untuk berpindah ke *branch* atau *commit* tertentu, setelah itu memperbarui *file* pada *working directory* berdasarkan *branch* atau *commit* tersebut. Berikut ini adalah sintaks-sintaks untuk operasi *checkout*:

```
$ git checkout [SHA-1 commit]
$ git checkout [branch-name]
```

12. Log

Operasi ini digunakan untuk menampilkan semua histori *commit* pada *branch* yang sedang aktif. Berikut ini adalah sintaks untuk melakukan operasi *log*:

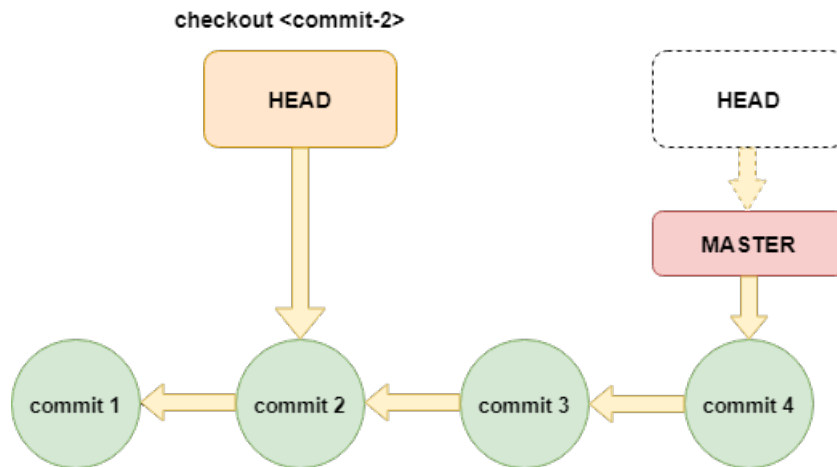
```
$ git log
```

2.1.4 Git Checkout

Seperti yang sudah dijelaskan pada subbab 2.1.3, *checkout* dapat digunakan untuk berpindah ke *branch* atau *commit* tertentu. Operasi *checkout* dapat dilakukan menggunakan sintaks `$ git checkout` diikuti dengan nama *branch* atau *SHA-1 hash*. Gambar 2.7 menunjukkan contoh *checkout* pada *commit*. Posisi awal *HEAD* menunjuk pada *branch master*, setelah dilakukan *checkout* ke *commit kedua*, posisi *HEAD* menunjuk pada *commit kedua*. *Working directory* diperbarui berdasarkan *state* pada *commit* kedua.

HEAD yang menunjuk langsung ke suatu *commit* disebut dengan *detached HEAD*. Perubahan yang terjadi pada *detached HEAD* tidak akan terekam oleh Git. Jika terdapat perubahan, kemudian

dilakukan *checkout commit* atau *branch*, perubahan tersebut akan hilang. Tetapi, perubahan tersebut bisa disimpan dengan cara membuat *branch* baru. Posisi *HEAD* akan menunjuk pada *branch* baru dan *HEAD* sudah tidak lagi dalam keadaan *detached HEAD*.



Gambar 2.7: Checkout pada commit

2.2 JGit

JGit adalah *library* Java murni yang mengimplementasikan Git *version control systems*[2]. Dengan menggunakan JGit, operasi-operasi dalam Git bisa dilakukan melalui program Java. Pada subbab berikut dijelaskan beberapa kelas dari *library* JGit. Subbab ini mengacu pada [4].

2.2.1 Repository

Kelas ini merepresentasikan repositori Git. Berikut ini adalah beberapa *method* dalam kelas ini:

- `public void create() throws IOException`
Berfungsi untuk membuat repositori Git baru.
- `public void create(boolean bare) throws IOException`
Berfungsi untuk membuat repositori Git baru.
Parameter: jika bernilai *true* maka dibuat *bare repository* (repositori tanpa *working directory*).
- `public String getBranch() throws IOException`
Berfungsi untuk mendapatkan nama *branch* yang ditunjuk oleh *HEAD*, *method* ini melempar *IOException*.
Kembalian: nama dari *branch* yang sedang aktif, contohnya *master*.
- `public ObjectId resolve(String revstr) throws AmbiguousObjectException, IncorrectObjectTypeException, RevisionSyntaxException, IOException`
Parameter: *expression* dari *git object references*. *Method* ini melempar *AmbiguousObjectException*, *IncorrectObjectTypeException*, *RevisionSyntaxException*, dan *IOException*.
Kembalian: sebuah objek *ObjectId*.

2.2.2 FileRepository

Kelas ini merupakan turunan dari kelas *Repository*. Berikut ini adalah *construtor* dari kelas ini:

- `public FileRepository(String gitDir)` throws `IOException`
Constructor ini membuat repositori berdasarkan parameter, *constructor* ini melempar `IOException`.
Parameter: lokasi dari *repository metadata*, lokasi ini berupa *path*.

2.2.3 Git

Kelas ini menyediakan API yang mirip *Git Command Line* untuk berinteraksi dengan repositori git. Berikut ini adalah *constructor* dan beberapa *method* dalam kelas ini:

- `public Git(Repository repo)`
Constructor ini membuat objek *Git* yang digunakan untuk berinteraksi dengan repositori *Git*. Parameter: objek *Repository* yang digunakan untuk berinteraksi. Parameter tidak boleh bernilai *null*.
- `public static InitCommand init()`
Method ini mengembalikan objek *command* untuk mengeksekusi operasi *init*.
Kembalian: objek *InitCommand* yang berfungsi untuk mengumpulkan parameter opsional dan akhirnya mengeksekusi operasi *init*.
- `public AddCommand add()`
Method ini mengembalikan objek *command* untuk mengeksekusi operasi *add*.
Kembalian: objek *AddCommand* yang berfungsi untuk mengumpulkan parameter opsional dan akhirnya mengeksekusi operasi *add*.
- `public LogCommand log()`
Method ini mengembalikan objek *command* untuk mengeksekusi operasi *log*.
Kembalian: objek *LogCommand* yang berfungsi untuk mengumpulkan parameter opsional dan akhirnya mengeksekusi operasi *log*.
- `public CheckoutCommand checkout()`
Method ini mengembalikan objek *command* untuk mengeksekusi operasi *checkout*.
Kembalian: objek *CheckoutCommand* yang berfungsi untuk mengumpulkan parameter opsional dan akhirnya mengeksekusi operasi *checkout*.
- `public CommitCommand commit()`
Method ini mengembalikan objek *command* untuk mengeksekusi operasi *commit*.
Kembalian: objek *CommitCommand* yang berfungsi untuk mengumpulkan parameter opsional dan akhirnya mengeksekusi operasi *commit*.
- `public FetchCommand fetch()`
Method ini mengembalikan objek *command* untuk mengeksekusi operasi *fetch*.
Kembalian: objek *FetchCommand* yang berfungsi untuk mengumpulkan parameter opsional dan akhirnya mengeksekusi operasi *fetch*.
- `public PushCommand push()`
Method ini mengembalikan objek *command* untuk mengeksekusi operasi *push*.
Kembalian: objek *PushCommand* yang berfungsi untuk mengumpulkan parameter opsional dan akhirnya mengeksekusi operasi *push*.
- `public DiffCommand diff()`
Method ini mengembalikan objek *command* untuk mengeksekusi operasi *diff*.
Kembalian: objek *DiffCommand* yang berfungsi untuk mengumpulkan parameter opsional dan akhirnya mengeksekusi operasi *diff*.

- `public static CloneCommand cloneRepository()`
Method ini mengembalikan objek *command* untuk mengeksekusi operasi *clone*.
 Kembalian: objek *DiffCommand* yang berfungsi untuk mengumpulkan parameter opsional dan akhirnya mengeksekusi operasi *clone*.
- `public MergeCommand merge()`
Method ini mengembalikan objek *command* untuk mengeksekusi operasi *merge*.
 Kembalian: objek *MergeCommand* yang berfungsi untuk mengumpulkan parameter opsional dan akhirnya mengeksekusi operasi *merge*.
- `public PullCommand pull()`
Method ini mengembalikan objek *command* untuk mengeksekusi operasi *pull*.
 Kembalian: objek *PullCommand*.
- `public CreateBranchCommand branchCreate()`
Method ini mengembalikan objek *command* untuk membuat *branch* baru.
 Kembalian: objek *CreateBranchCommand*.
- `public ListBranchCommand branchList()`
Method ini mengembalikan objek *command* untuk menampilkan daftar *branch*.
 Kembalian: objek *ListBranchCommand*.
- `public DeleteBranchCommand branchDelete()`
Method ini mengembalikan objek *command* untuk menghapus *branch*.
 Kembalian: objek *DeleteBranchCommand*.

2.2.4 RevWalk

Kelas ini digunakan untuk menelusuri *commit graph*. *Instance* dari kelas ini hanya bisa melakukan *graph traversal* satu kali, untuk melakukan *traversal* kedua dibutuhkan *instance* baru atau memanggil *method* `reset()`. Berikut ini adalah *constructor* dan beberapa *method* dalam kelas ini:

- `public RevWalk(Repository repo)`
Constructor ini membuat objek *revision walker* untuk suatu *repository*.
 Parameter: repositori yang digunakan untuk *traversal*.
- `public RevCommit parseCommit(AnyObjectId id)`
 Menempatkan *reference* ke suatu *commit* kemudian melakukan *parsing* pada isi *commit*.
 Parameter: nama dari objek *commit*.
 Kembalian: *reference* ke objek *commit*.
- `public void sort(RevSort s)`
 Berfungsi untuk mengurutkan *commit* berdasarkan metode dari parameter.
 Parameter: metode untuk mengurutkan *commit*.
- `public Iterator<RevCommit> iterator()`
 Berfungsi untuk mengembalikan *iterator* yang bertipe *RevCommit*.
 Kembalian: *iterator* dari *RevCommit*.
- `public void markStart(RevCommit c) throws MissingObjectException, IncorrectObjectTypeException, IOException`
 Berfungsi untuk menandai *commit* pertama untuk memulai *traversal*. *Method* ini melempar *MissingObjectException*, *IncorrectObjectTypeException*, dan *IOException*.
 Parameter: *commit* awal yang digunakan untuk melakukan *traversal*.

- `public final void reset()`
Berfungsi untuk mengembalikan *state* dari kelas ini ke *state* semula, sehingga *instance RevWalk* bisa digunakan lagi.

2.2.5 RevCommit

Kelas ini merupakan *reference* ke *commit* yang ada di *Directed Acyclic Graph*. Berikut ini adalah *constructor* dan beberapa *method* dari kelas ini:

- `protected RevCommit(AnyObjectId id)`
Constructor ini membuat objek yang merupakan *reference* ke suatu *commit*.
Parameter: nama dari objek *commit*.
- `public final String getFullMessage()`
Berfungsi untuk melakukan *parsing* pada *full commit message* dan mengubahnya ke *string*.
Kembalian: *string* hasil *decode* dari *commit message*.
- `public final String getShortMessage()`
Berfungsi untuk melakukan *parsing* pada *commit message* dan mengubahnya ke *string*, hanya baris pertama yang dikembalikan.
Kembalian: baris pertama *string* hasil *decode* dari *commit message*.
- `public final String getName()`
Method ini mengembalikan *SHA-1* dalam bentuk *string*. Kembalian: *string SHA-1* dalam bentuk heksadesimal.
- `public final PersonIdent getAuthorIdent()`
Berfungsi untuk mendapatkan informasi mengenai *author* yang melakukan *commit*.
Kembalian: objek *PersonIdent* yang memuat informasi tentang *author* (nama dan *email*) dan waktu dilakukannya *commit*.

2.2.6 PersonIdent

Kelas ini memberikan informasi mengenai *author* dari suatu *commit*. Berikut ini adalah beberapa *method* dari kelas ini:

- `public String getName()`
Berfungsi untuk mengembalikan nama dari *author* yang melakukan *commit*.
Kembalian: nama dari *author*.
- `public String getEmailAddress()`
Berfungsi untuk mengembalikan alamat *email* dari *author* yang melakukan *commit*.
Kembalian: alamat *email* dari *author*.
- `public Date getWhen()`
Berfungsi mengembalikan waktu dilakukannya suatu *commit* oleh *author*.
Kembalian: sebuah *timestamp*.

2.3 Selenium WebDriver

Selenium adalah kumpulan dari kakas perangkat lunak, dengan pendekatan yang berbeda pada setiap kakas dalam mendukung *automation test*[5]. Selenium mendukung bahasa pemrograman C#, Java, Perl, PHP, Python, Ruby, dan JavaScript. Selenium terdiri dari beberapa kakas, yaitu Selenium 1(Selenium RC), Selenium 2(Selenium WebDriver), Selenium-Grid, dan Selenium IDE. Selenium RC merupakan proyek utama *Selenium* untuk waktu yang lama, sebelum akhirnya

bergabung dengan *WebDriver* menjadi Selenium 2. Selenium RC bekerja dengan cara menginjeksi kode JavaScript ke *browser* ketika *browser* dimuat dan menggunakan JavaScript tersebut untuk menjalankan *Application Under Test* dalam *browser*. Selenium RC sekarang sudah *deprecated* dan tidak digunakan lagi. Selenium WebDriver merupakan gabungan dari Selenium RC dan WebDriver. Selenium IDE merupakan kakas yang digunakan untuk mengembangkan Selenium *test cases*.

WebDriver merupakan kakas untuk mengotomatisasi pengujian pada perangkat lunak web[5]. WebDriver dapat berkomunikasi dengan *browser* menggunakan *native support* pada *browser* untuk automasi. Setiap *browser* memiliki WebDriver masing-masing. WebDriver yang terdapat pada SeleniumDriver antara lain ChromeDriver, FirefoxDriver/GeckoDriver, OperaDriver, InternetExplorerDriver, dan HtmlUnitDriver.

Pada skripsi ini *tools* Selenium yang digunakan hanya Selenium WebDriver. WebDriver yang digunakan adalah ChromeDriver. Bahasa yang digunakan adalah Java. Pada subbab berikut dijelaskan beberapa kelas dari *library* Selenium WebDriver. Subbab ini mengacu pada [6].

2.3.1 WebDriver

Kelas ini merupakan *interface* utama yang digunakan untuk pengujian, kelas ini merepresentasikan *web browser* yang ideal . Berikut ini adalah beberapa *method* dalam kelas ini:

- void close()
Berfungsi untuk menutup *window* pada *browser*, jika *window* yang sekarang merupakan satu-satunya *window* yang terbuka maka *browser* akan ditutup.
- void quit()
Berfungsi untuk menutup driver dan semua *window* yang sedang terbuka.
- void get(String url)
Berfungsi untuk memuat halaman *web* pada *window* saat ini. *Method* ini mengirim *HTTP GET Request* untuk memuat halaman, dan *method* ini akan melakukan *blocking* sampai halaman *web* selesai dimuat.
Parameter: alamat *url* untuk memuat halaman *web*.
- String getTitle()
Berfungsi untuk mengembalikan judul dari halaman *web* yang sedang aktif.
Kembalian: judul dari halaman *web*.
- String getCurrentUrl()
Berfungsi untuk mendapatkan URL yang sedang aktif di *browser*.
Kembalian: URL dari halaman *web* yang sedang dimuat di *browser*.

2.3.2 WebElement

Kelas ini adalah *Interface* yang merupakan representasi dari elemen HTML. Berikut ini adalah beberapa *method* yang dimiliki kelas ini:

- void click()
Berfungsi untuk mengklik suatu elemen HTML.
- void submit()
Berfungsi untuk mengirimkan elemen *form* ke *remote server*. Fungsi ini akan melempar *NoSuchElementException* jika elemen yang dikirim tidak berada di dalam *form*.
- String getText()
Berfungsi untuk mendapatkan teks pada suatu elemen.
Kembalian: Teks yang *visible* pada elemen.

- `void clear()`
Berfungsi untuk menghapus teks pada elemen yang digunakan untuk memasukkan teks.
- `WebElement findElement(By by)`
Berfungsi untuk mendapatkan *WebElement* pertama menggunakan metode yang diberikan parameter. *Method* ini akan melempar *NoSuchElementException* jika *WebElement* tidak ditemukan.
Kembalian: *WebElement* pertama yang sesuai dengan mekanisme pencarian.
Parameter: mekanisme pencarian, bisa berupa pencarian dengan *ID*, *class*, dll.
- `List<WebElement> findElements(By by)`
Berfungsi untuk mendapatkan semua *WebElement* sesuai dengan mekanisme yang diberikan parameter.
Kembalian: *list* dari *WebElement*, atau *list* kosong jika pencarian tidak ditemukan.
Parameter: mekanisme pencarian, bisa berupa pencarian dengan *ID*, *class*, dll.
- `void sendKeys(java.lang.CharSequence... keysToSend)`
Berfungsi untuk mengirimkan kumpulan karakter/teks ke elemen *input*. *Method* ini akan melempar *java.lang.IllegalArgumentException* jika parameter *keysToSend* bernilai *null*.
Parameter: kumpulan karakter/teks yang dikirim ke elemen.
- `String getAttribute(String name)`
Berfungsi untuk mendapatkan nilai dari *attribute* suatu *web element*.
Kembalian: nilai dari *attribute* dari *web element*.

2.3.3 OutputType

Kelas ini merupakan *interface* yang menentukan tipe *output* pada *screenshot*. Terdapat tiga konstanta untuk menentukan tipe *output* pada *screenshot*. Konstanta tersebut adalah sebagai berikut:

- `static final OutputType<String> BASE64`
Berfungsi untuk mendapatkan *screenshot* dalam bentuk *base64 data*.
- `static final OutputType<byte[]> BYTES`
Berfungsi untuk mendapatkan *screenshot* dalam bentuk *raw bytes*.
- `static final OutputType<java.io.File> FILE`
Berfungsi untuk mendapatkan *screenshot* dalam bentuk *temporary file* yang akan dihapus setelah program keluar dari *Java Virtual Machine*.

2.3.4 TakesScreenshot

Kelas ini merupakan *interface* yang digunakan untuk mengambil *screenshot*. Kelas ini hanya mempunyai satu method yaitu:

- `<X> X getScreenshotAs(OutputType<X> target)` throws *WebDriverException*
Method ini berfungsi untuk mengambil *screenshot* dan menyimpannya ke lokasi yang sudah ditentukan.
Kembalian: objek yang menyimpan informasi terkait *screenshot*
Parameter: tipe *output* yang diinginkan (lihat 2.3.3).

2.4 Apache Commons CLI

Library Apache Commons CLI menyediakan API untuk menguraikan *command-line options* yang dikirimkan ke program[7]. Apache Commons CLI termasuk ke dalam salah satu *project* Apache Commons. Tujuan utama dari *project* Apache Commons adalah membuat dan melakukan *maintain* pada komponen Java yang *reusable*. Pada subbab berikut dijelaskan beberapa kelas dari *library* Apache Commons CLI. Subbab ini mengacu pada [8].

2.4.1 CommandLineParser

Kelas ini merupakan *interface*. Kelas yang mengimplementasikan *interface* ini dapat menguraikan *array of String* berdasarkan pada parameter/argumen yang diberikan. Berikut ini adalah beberapa *method* yang dimiliki *interface* ini:

- `CommandLine parse(Options options, String[] arguments) throws ParseException`
 Berfungsi untuk menguraikan argumen berdasarkan pada *option* yang ditentukan. *Method* ini melempar *ParseException*.
 Parameter: *option* yang ditentukan, argumen *command line*.
 Kembalian: objek *CommandLine*.
- `CommandLine parse(Options options, String[] arguments, boolean stopAtNonOption) throws ParseException`
 Berfungsi untuk menguraikan argumen berdasarkan pada *option* yang ditentukan.
 Parameter: *option* yang ditentukan, argumen *command line*, dan suatu *boolean* yang menentukan apakah *parsing* dihentikan jika terdapat argumen yang tidak valid. Jika bernilai *true*, *parsing* akan dihentikan dan semua argumen yang sudah diuraikan akan ditambahkan ke objek *CommandLine*. Jika bernilai *false*, akan dilempar *ParseException* bila terdapat argumen yang tidak valid.
 Kembalian: objek *CommandLine*.

2.4.2 CommandLine

Kelas ini merepresentasikan kumpulan argumen yang diuraikan terhadap *options descriptor*. Berikut ini adalah beberapa *method* yang dimiliki kelas ini:

- `public String getOptionValue(String opt)`
 Berfungsi untuk mendapatkan nilai dari suatu *option* berdasarkan parameter.
 Parameter: nama dari *option*.
 Kembalian: nilai dari *option*. Jika *option* belum diatur, akan dikembalikan *null*.
- `protected void addOption(Option opt)`
 Berfungsi untuk menambahkan *option* ke *command line*.
 Parameter: objek *option* yang ingin ditambahkan.
- `public boolean hasOption(String opt)`
 Berfungsi untuk menentukan apakah suatu *option* sudah diatur.
 Parameter: nama dari *option*.
 Kembalian: *true* jika *option* sudah diatur, *false* jika *option* belum diatur.
- `public Option[] getOptions()`
 Berfungsi untuk mengembalikan *array* dari *option* yang sudah diproses.
 Kembalian: *iterator* dari *option* yang sudah diproses.
- `public Iterator<Option> iterator()`
 Berfungsi untuk mengembalikan *iterator* dari *option* yang sudah diproses.
 Kembalian: *array* dari *option* yang sudah diproses.

2.4.3 Options

Kelas ini merepresentasikan kumpulan dari objek *Option*, yang mendeskripsikan kemungkinan *option* pada *command line*. Berikut ini adalah beberapa *method* yang dimiliki kelas ini:

- `public Options addOption(Option opt)`
Berfungsi untuk menambahkan objek *Option* ke kelas ini. Parameter: *option* yang akan ditambahkan.
Kembalian: hasil dari *option* yang ditambahkan.
- `public Option getOption(String opt)`
Berfungsi untuk mengembalikan objek *Option* sesuai dengan nama yang diberikan parameter. Parameter: nama dari *option* yang ingin dikembalikan.
Kembalian: objek *option* berdasarkan parameter.

2.4.4 Option

Kelas ini mendeskripsikan sebuah *command-line option*. Berikut ini adalah *constructor* dan beberapa *method* yang dimiliki kelas ini:

- `public Option(String opt, String description) throws IllegalArgumentException`
Constructor ini membuat objek *option* sesuai dengan parameter yang diberikan. *Constructor* ini melempar *IllegalArgumentException*.
Parameter: nama pendek *option*, dan deskripsi dari *option*.
- `public Option(String opt, boolean hasArg, String description) throws IllegalArgumentException`
Constructor ini membuat objek *option* sesuai dengan parameter yang diberikan. *Constructor* ini melempar *IllegalArgumentException*.
Parameter: nama pendek *option*, suatu *boolean* yang menentukan apakah *option* membutuhkan argumen, dan deskripsi dari *option*.
- `public Option(String opt, String longOpt, boolean hasArg, String description) throws IllegalArgumentException`
Constructor ini membuat objek *option* sesuai dengan parameter yang diberikan. *Constructor* ini melempar *IllegalArgumentException*.
Parameter: nama pendek *option*, nama panjang *option*, suatu *boolean* yang menentukan apakah *option* membutuhkan argumen, dan deskripsi dari *option*.
- `public boolean hasArg()`
Berfungsi untuk mengetahui apakah suatu *option* membutuhkan argumen.
Kembalian: *true* jika *option* ini membutuhkan argumen, *false* jika *option* ini tidak membutuhkan argumen.
- `public String getDescription()`
Berfungsi untuk mendapatkan deskripsi dari suatu *option*.
Kembalian: deskripsi dari *option* ini.
- `public String getArgName()`
Berfungsi untuk mendapatkan nama dari suatu *option*.
Kembalian: nama dari argumen suatu *option*.
- `public String getLongOpt()`
Berfungsi untuk mendapatkan nama panjang dari suatu *option*.
Kembalian: nama panjang dari suatu *option*.

2.4.5 Option.Builder

Kelas ini merupakan *nested class* dari kelas *Option*. Kelas ini digunakan untuk membuat objek *Option* dengan *descriptive methods*. Berikut ini adalah beberapa *method* yang dimiliki kelas ini:

- `public Option.Builder desc(String description)`
Berfungsi untuk memberikan deskripsi pada *option*.
Parameter: deskripsi dari *option*.
Kembalian: objek *Option.Builder* yang bisa digunakan untuk *method chaining*.
- `public Option.Builder longOpt(String longOpt)`
Berfungsi untuk memberikan nama panjang pada *option*.
Parameter: nama panjang *option*.
Kembalian: objek *Option.Builder* yang bisa digunakan untuk *method chaining*.
- `public Option.Builder hasArg()`
Berfungsi untuk menyatakan bahwa *option* ini membutuhkan argumen.
Kembalian: objek *Option.Builder* yang bisa digunakan untuk *method chaining*.
- `public Option.Builder argName(String argName)`
Berfungsi untuk memberi nama pada argumen.
Parameter: nama argumen.
Kembalian: objek *Option.Builder* yang bisa digunakan untuk *method chaining*.
- `public Option build()`
Berfungsi untuk membuat objek *Option* berdasarkan nilai pada *Option.Builder*.
Kembalian: objek *Option*.

DAFTAR REFERENSI

- [1] Chacon, S. dan Straub, B. (2014) *Pro Git* The expert's voice. Apress.
- [2] Jgit | the eclipse foundation. <https://www.eclipse.org/jgit/>. [Online; diakses 2-September-2018].
- [3] Selenium webdriver. <https://www.seleniumhq.org/about/>. [Online; diakses 2-September-2018].
- [4] Jgit - parent 5.0.3.201809091024-r api. <http://download.eclipse.org/jgit/site/5.0.3.201809091024-r/apidocs/index.html>. [Online; diakses 17-September-2018].
- [5] Selenium documentation. <https://www.seleniumhq.org/docs/>. [Online; diakses 17-September-2018].
- [6] Generated documentation. <https://seleniumhq.github.io/selenium/docs/api/java/>. [Online; diakses 17-September-2018].
- [7] Commons - home. <https://commons.apache.org/proper/commons-cli/index.html>. [Online; diakses 11-Oktober-2018].
- [8] Apache commons cli 1.3.1 api. <https://commons.apache.org/proper/commons-cli/javadocs/api-release/index.html>. [Online; diakses 11-Oktober-2018].

LAMPIRAN A

KODE PROGRAM

Listing A.1: MyCode.c

```

1 // This does not make algorithmic sense,
2 // but it shows off significant programming characters.
3
4 #include<stdio.h>
5
6 void myFunction( int input, float* output ) {
7     switch ( array[i] ) {
8         case 1: // This is silly code
9             if ( a >= 0 || b <= 3 && c != x )
10                 *output += 0.005 + 20050;
11             char = 'g';
12             b = 2^n + ~right_size - leftSize * MAX_SIZE;
13             c = (--aaa + &daa) / (bbb++ - ccc % 2 );
14             strcpy(a,"hello_$@?");
15         }
16         count = ~mask | 0x00FF00AA;
17     }
18 }
19
20 // Fonts for Displaying Program Code in LATEX
21 // Adrian P. Robson, nepsweb.co.uk
22 // 8 October 2012
23 // http://nepsweb.co.uk/docs/progfonts.pdf

```

Listing A.2: MyCode.java

```

1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id; //id of the set
8     protected MyEdge FurthestEdge; //the furthest edge
9     protected HashSet<MyVertex> set; //set of vertices close to furthest edge
10    protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each trajectory
11    protected ArrayList<Integer> closeID; //store the ID of all vertices
12    protected ArrayList<Double> closeDist; //store the distance of all vertices
13    protected int totaltrj; //total trajectories in the set
14
15    /*
16     * Constructor
17     * @param id : id of the set
18     * @param totaltrj : total number of trajectories in the set
19     * @param FurthestEdge : the furthest edge
20     */
21    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
22        this.id = id;
23        this.totaltrj = totaltrj;
24        this.FurthestEdge = FurthestEdge;
25        set = new HashSet<MyVertex>();
26        ordered = new ArrayList<ArrayList<Integer>>();
27        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
28        closeID = new ArrayList<Integer>(totaltrj);
29        closeDist = new ArrayList<Double>(totaltrj);
30        for (int i = 0;i <totaltrj;i++) {
31            closeID.add(-1);
32            closeDist.add(Double.MAX_VALUE);
33        }
34    }
35
36 }

```


LAMPIRAN B

HASIL EKSPERIMEN

Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4