

# PEMBANGKIT TIMELAPSE PENGEMBANGAN PROYEK PERANGKAT LUNAK BERBASIS WEB

BILLY ADIWIJAYA—2015730053

## 1 Data Skripsi

Pembimbing utama/tunggal: **Pascal Alfadian Nugroho**

Pembimbing pendamping: -

Kode Topik : **PAN4401**

Topik ini sudah dikerjakan selama : **1 semester**

Pengambilan pertama kali topik ini pada : Semester **45 - Ganjil 18/19**

Pengambilan pertama kali topik ini di kuliah : **Skripsi 1**

Tipe Laporan : **B -** Dokumen untuk reviewer pada presentasi dan **review Skripsi 1**

## 2 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 *timelapse* 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.

## 3 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?

## 4 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.

## 5 Detail Perkembangan Pengerjaan Skripsi

Detail bagian pekerjaan skripsi sesuai dengan rencan kerja/laporan perkembangan terakhir :

1. **Melakukan studi literatur tentang Selenium WebDriver, Git, dan JGit.**

**Status :** Ada sejak rencana kerja skripsi.

**Hasil :**

- **Git**

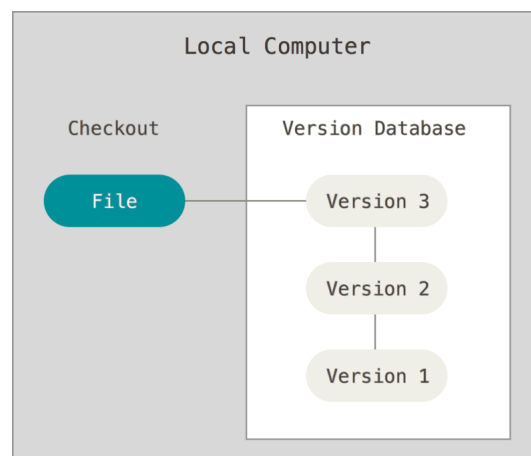
Git merupakan perangkat lunak *Version Control Systems*. Pada bagian ini, dijelaskan mengenai *Version Control Systems*, cara kerja Git, *Git checkout*, dan operasi-operasi dasar pada Git. Subbab ini mengacu pada [1].

### 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

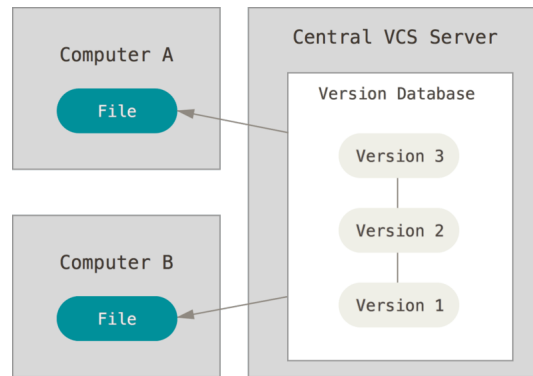
Metode *version-controlled* yang banyak digunakan orang adalah dengan cara menyalin sekumpulan *file* ke direktori lain. Namun cara tersebut rentan terhadap *error*. Misalnya, terdapat direktori A dan B, pengguna ingin mengubah *file* yang terdapat pada direktori B, tetapi pengguna lupa kalau dia sedang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



Gambar 1: Local version control[1].

Gambar 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



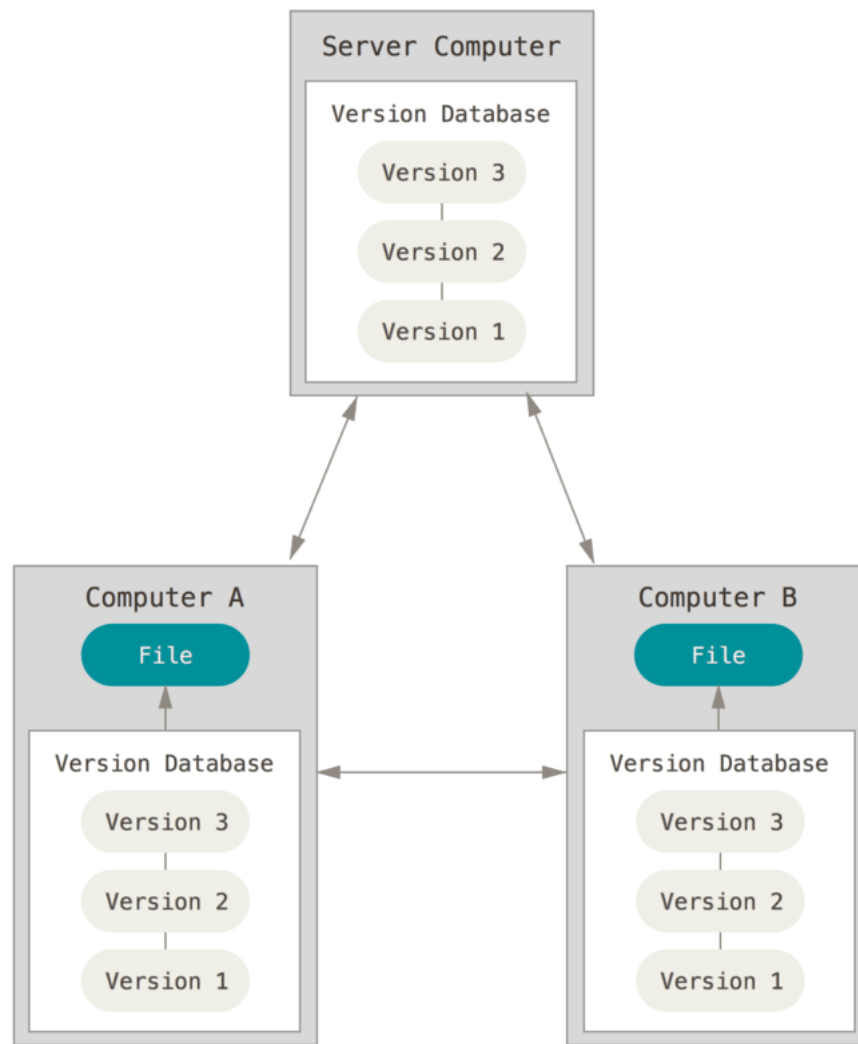
Gambar 2: Centralized version control[1].

*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 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

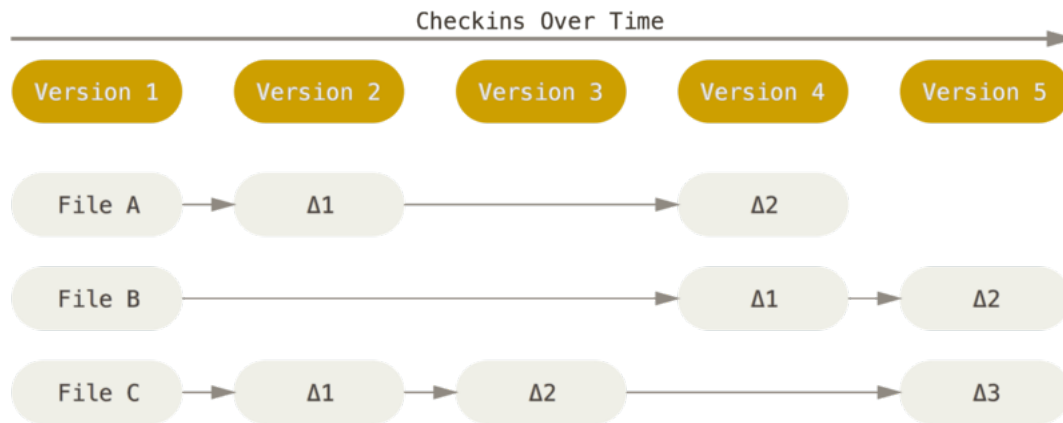


Gambar 3: Distributed version control[1].

Gambar 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*.

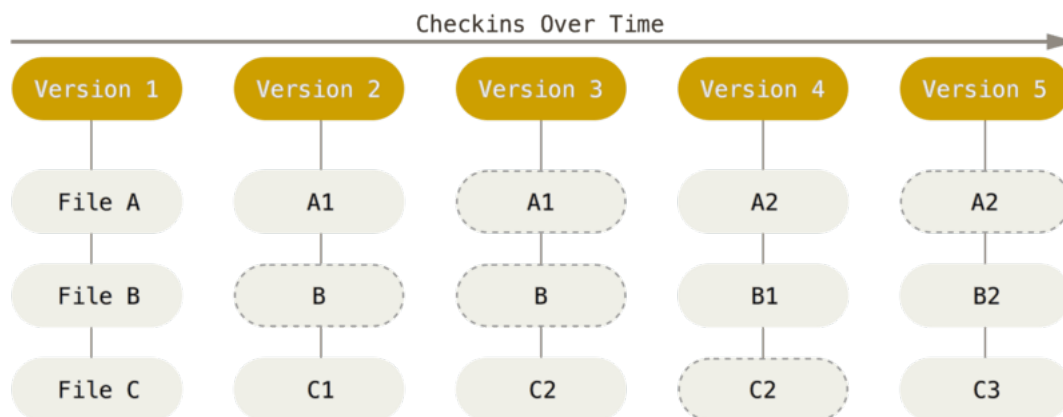
### Cara Kerja Git

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

### State pada Git

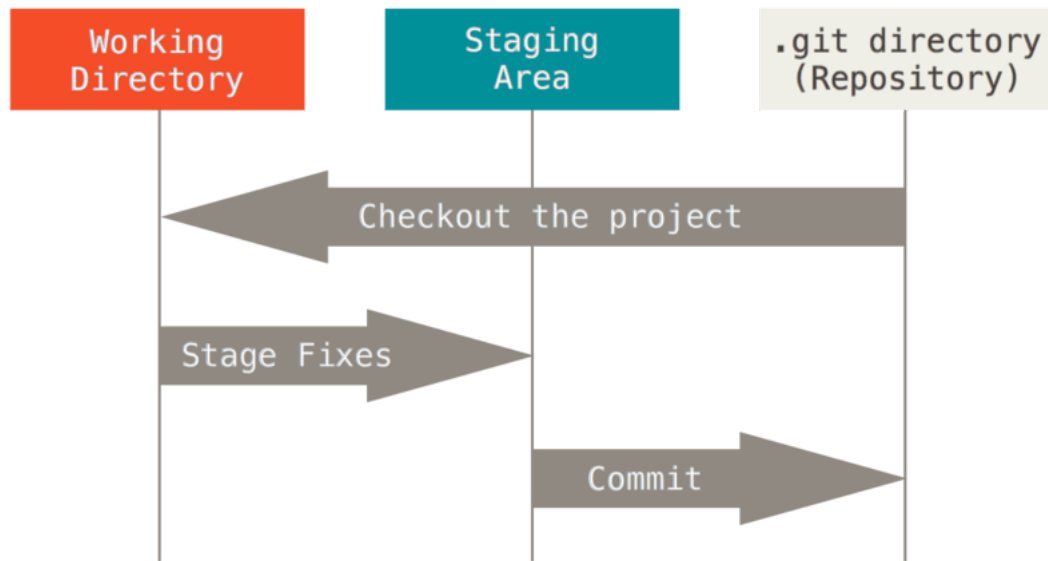
Terdapat tiga *state* pada Git yaitu *committed*, *modified*, and *staged*. *Committed* adalah *state* dimana data sudah disimpan di *local database*. *Modified state* dimana terdapat perubahan pada *file*, namun *file* tersebut belum di *commit* ke *database*. *Staged* adalah *state* dimana *file* telah 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 6.

Alur kerja dari Git adalah sebagai berikut:

- (a) Melakukan modifikasi pada *file*.
- (b) Menandai perubahan pada *file* dan memindahkannya ke *staging area*.

- (c) Mengambil *file* dari *staging area* dan menyimpan *snapshot* ke direktori Git. Proses ini disebut dengan *commit*.



Gambar 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 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 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 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.

### Operasi Dasar pada Git

Pada bagian 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:

#### (a) 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]
```

#### (b) 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]
```

#### (c) 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]"
```

#### (d) 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]

```

#### (e) 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]
```

(f) 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]
```

(g) Fetch

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

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

(h) 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]
```

(i) Pull

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

```
$ git pull
```

(j) 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]
```

(k) 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]
```

(l) Log

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

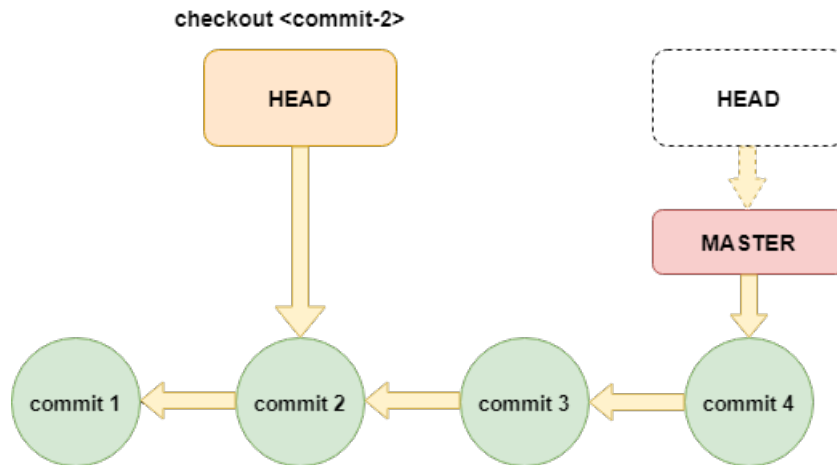
```
$ git log
```

## Git Checkout

Seperti yang sudah dijelaskan pada bagian Operasi Dasar Git, *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 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 7: Checkout pada commit

### • 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 bagian ini dijelaskan beberapa kelas dari *library* JGit. Subbab ini mengacu pada [4].

#### 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*.

**FileRepository** Kelas ini merupakan turunan dari kelas *Repository*. Berikut ini adalah *constructor* 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*.

#### 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*.

### 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.

### 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*.

### 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*.

### • Selenium WebDriver

Selenium adalah kumpulan dari kaskas perangkat lunak, dengan pendekatan yang berbeda pada setiap kaskas dalam mendukung *automation test*[5]. Selenium mendukung bahasa pemrograman C#, Java, Perl, PHP, Python, Ruby, dan JavaScript. Selenium terdiri dari beberapa kaskas, 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 kaskas yang digunakan untuk mengembangkan Selenium *test cases*.

WebDriver merupakan kaskas 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 bagian ini dijelaskan beberapa kelas dari *library* Selenium WebDriver. Subbab ini mengacu pada [6].

### 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*.

**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*.

### 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*.

### 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 bagian OutputType).

### 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 bagian ini dijelaskan beberapa kelas dari *library* Apache Commons CLI. Bagian ini mengacu pada [8].

### CommandLineParser

Kelas ini merupakan *interface*. Kelas yang mengimplementasikan *interface* ini dapat menguraikan *array of String* berdasarkan pada *options* 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*.

### 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.

### 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.

### 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*.

### 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*.

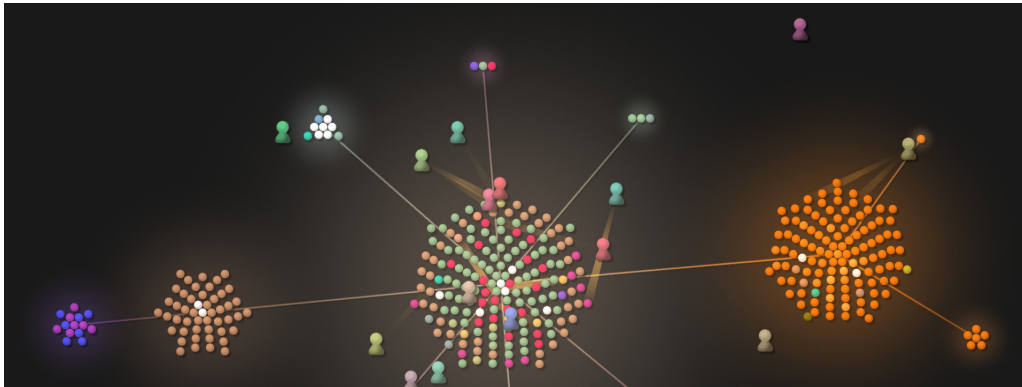


## 2. Melakukan analisis program sejenis.

**Status :** baru ditambahkan pada semester ini.

**Hasil :**

Saat skripsi ini dibuat, aplikasi sejenis yang digunakan untuk membangkitkan animasi adalah Gource. Proyek perangkat lunak ditampilkan oleh Gource sebagai animasi pohon, dimana pusatnya adalah *root directory* dari proyek perangkat lunak[9]. Direktori ditampilkan sebagai *branch*, sedangkan *file* ditampilkan sebagai *leaf*. Developer dapat terlihat di *working tree* pada saat mereka berkontribusi untuk proyek.



Gambar 8: Visualisasi proyek perangkat lunak menggunakan Gource.

Gambar 8 menunjukkan contoh visualisasi proyek perangkat lunak menggunakan Gource. Efek cahaya yang terdapat pada Gambar 8 disebut dengan *bloom*. Pada awalnya ukuran *working tree* tidak terlalu besar. Setiap kali ditambahkan *file* dan *folder* baru, akan dibuat *branch* dan *leaf* baru pada *working tree*.

Gource memiliki beberapa fitur. Fitur-fitur tersebut dapat diatur melalui *command line options*. Berikut ini adalah beberapa *command line options* yang terdapat pada Gource:

(a) `gource -[WIDTH]x[HEIGHT]`

Opsi ini berfungsi untuk mengatur resolusi layar dari animasi. Parameter dari opsi ini adalah lebar dan panjang layar dalam satuan piksel.

(b) `gource -camera-mode [MODE]`

Opsi ini berfungsi untuk mengatur mode kamera pada Gource. Parameter dari opsi ini adalah mode dari kamera. Terdapat dua mode yaitu *overview* dan *track*. Dalam mode *track*, kamera bergerak mengikuti *user* yang sedang aktif. Dalam mode *overview*, kamera menampilkan seluruh repositori.

(c) `gource -path [PATH]`

Opsi untuk berfungsi untuk mengatur *path* dari direktori yang akan dibuat animasinya. Opsi dari parameter ini adalah *path* dari direktori.

(d) `gource -start-date [YYYY-MM-DD hh:mm:ss +tz] -stop-date [YYYY-MM-DD hh:mm:ss +tz]`

Opsi untuk berfungsi untuk mengatur periode waktu dalam menampilkan animasi. Parameter dari opsi ini adalah waktu mulai dan waktu akhir dalam format "YYYY-MM-DD hh:mm:ss +tz". Dimana YYYY adalah tahun, MM adalah bulan, DD adalah tanggal, hh adalah jam, mm adalah menit, ss adalah detik, dan +tz adalah zona waktu. Parameter jam, menit, detik, dan zona waktu bersifat opsional.

- (e) `gource -bloom-multiplier [FLOAT]`  
 Opsi untuk berfungsi untuk mengatur radius dari efek *bloom*. Parameter dari opsi ini adalah radius dalam format bilangan riil.
- (f) `gource -bloom-intensity [FLOAT]`  
 Opsi untuk berfungsi untuk mengatur intensitas dari efek *bloom*. Parameter dari opsi ini adalah intensitas *bloom* dalam format bilangan riil.
- (g) `gource -disable-bloom`  
 Opsi ini berfungsi untuk menonaktifkan animasi *bloom*.
- (h) `gource -date-format [FORMAT]`  
 Opsi untuk mengatur format waktu yang ditampilkan pada bagian tengah atas. Opsi dari parameter ini adalah format waktu dalam bentuk *string*.
- (i) `gource -background [FFFFFF]`  
 Opsi ini berfungsi untuk mengatur warna *background*. Parameter dari opsi ini adalah warna dalam format heksadesimal.
- (j) `gource -background-image [IMAGE]`  
 Opsi ini berfungsi untuk mengatur gambar *background*. Parameter dari opsi ini adalah nama *file* dari gambar.
- (k) `gource -font-size [SIZE]`  
 Opsi ini digunakan untuk mengatur ukuran *font* pada tulisan *title* dan tanggal. Parameter dari opsi ini adalah ukuran *font*.
- (l) `gource -font-colour [FFFFFF]`  
 Opsi ini digunakan untuk mengatur warna *font* pada tulisan *title* dan tanggal. Parameter dari opsi ini adalah warna *font* dalam format heksadesimal.
- (m) `gource -logo [IMAGE]`  
 Opsi ini berfungsi untuk memasukkan logo. Parameter dari opsi ini adalah nama *file* dari gambar.
- (n) `gource -logo-offset [X]x[Y]`  
 Opsi ini berfungsi untuk mengatur posisi dari logo. Parameter dari opsi ini adalah posisi x dan posisi y dari logo.
- (o) `gource -title [TITLE]`  
 Opsi ini berfungsi untuk memberi judul. Dimana judul tersebut ditampilkan pada pojok kiri bawah layar.
- (p) `gource -output-framerate [FPS]`  
 Opsi ini berfungsi untuk mengatur jumlah *frame* per detik pada video animasi. Parameter dari opsi ini adalah jumlah *frame* per detik.
- (q) `gource -hide [DISPLAY-ELEMENT]`  
 Opsi ini berfungsi untuk menyembunyikan satu atau lebih *display element*. Parameter dari opsi ini adalah elemen yang akan disembunyikan. *Display element* yang dapat disembunyikan yaitu:
  - *bloom*: efek *bloom*.
  - *date*: waktu.
  - *dirname*: nama direktori.
  - *files*: ikon dari berkas.
  - *filenames*: nama berkas.
  - *root*: *root directory*.
  - *users*: ikon dari *user*.

- *usernames*: nama dari *user*.

### 3. Menganalisis penggunaan Selenium WebDriver dan JGit untuk membangkitkan animasi timelapse.

**Status :** Ada sejak rencana kerja skripsi.

**Hasil :**

### 4. Menganalisis Command Line Options.

**Status :** baru ditambahkan pada semester ini.

**Hasil :**

Program pada skripsi ini akan menerima masukan dari Command Line Options. Selain itu konfigurasi program juga didapatkan dari Command Line Options. Berikut ini adalah *command line options* yang akan diimplementasikan pada skripsi ini:

- **-project-path [PATH]**  
Opsi ini berfungsi untuk mengatur *path* dari direktori yang akan dibuat animasinya. Parameter dari opsi ini adalah *path* dari proyek perangkat lunak web. Opsi ini wajib ada.
- **-before-capture [PHP-SCRIPT]**  
Opsi ini berfungsi untuk menjalankan *script* PHP. *Script* ini dijalankan sebelum melakukan *screenshot*. Parameter dari opsi ini adalah *script* PHP. Opsi ini bersifat opsional.
- **-capture-url [url]**  
Opsi ini berfungsi untuk mengatur alamat *url* dari halaman *web*, dimana dilakukan pengambilan *screenshot* pada halaman ini. Parameter dari opsi ini adalah alamat *url* dari halaman *web*. Opsi ini wajib ada.
- **-fps [FPS]**  
Opsi ini berfungsi untuk mengatur jumlah *frame* per detik pada animasi. Parameter dari opsi ini adalah jumlah *frame* per detik. Opsi ini bersifat opsional.
- **-title [TITLE]**  
Opsi ini berfungsi untuk memberi judul. Opsi ini bersifat opsional.
- **-logo [IMAGE]**  
Opsi ini berfungsi untuk memasukkan logo. Parameter dari opsi ini adalah *file* gambar. Opsi ini bersifat opsional.
- **-start-commit [ID] -stop-commit [ID]**  
Opsi ini berfungsi untuk mengatur rentang *commit* yang akan dibuat animasinya. Parameter dari opsi ini adalah *commit ID* awal dan *commit ID* akhir. Opsi ini bersifat opsional.

### 5. Merancang perangkat lunak.

**Status :** Ada sejak rencana kerja skripsi.

**Hasil :** belum ada perkembangan.

### 6. Membangun perangkat lunak.

**Status :** Ada sejak rencana kerja skripsi.

**Hasil :**

### 7. Melakukan eksperimen dan pengujian pada perangkat lunak.

**Status :** Ada sejak rencana kerja skripsi.

**Hasil :** belum ada perkembangan.

### 8. Menulis dokumen skripsi.

**Status :** Ada sejak rencana kerja skripsi.

**Hasil :** Dokumen skripsi sudah ditulis hingga bab 3. Bab 1 berisi tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metode penelitian, dan sistematika penulisan. Bab 2 berisi tentang teori Git, JGit, Selenium WebDriver, dan Apache Commons CLI. Bab 3 berisi tentang analisis aplikasi sejenis, analisis penggunaan JGit dan Selenium WebDriver, prapengujian, dan analisis Command Line Options.

## 6 Pencapaian Rencana Kerja

Langkah-langkah kerja yang berhasil diselesaikan dalam Skripsi 1 ini adalah sebagai berikut:

1. Melakukan studi literatur tentang Selenium WebDriver, Git, dan JGit.
2. Menganalisis program sejenis.
3. Menganalisis penggunaan Selenium WebDriver dan JGit untuk membangkitkan animasi *timelapse*.
4. Menganalisis Command Line Options.

## Pustaka

- [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].
- [9] Gource - a software version control visualization tool. <https://gource.io/>. [Online; diakses 29-Oktober-2018].

Bandung, 15/11/2018

Billy Adiwijaya

Menyetujui,

Nama: Pascal Alfadian Nugroho  
Pembimbing Tunggal