# Aplikasi Teori Graf Dalam Memodelkan Version Control System Git

Kevin Rizki Mohammad 13518100
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*13518100@std.stei.itb.ac.id*

***Abstrak*—Dalam proyek pembuatan perangkat lunak, file yang dikelola selalu berganti seiring berjalannya waktu. Perlu ada sistem untuk merekam semua perubahan tersebut. Version Control System (VCS) dibuat untuk melakukan fungsi tersebut. Salah satu VCS yang banyak digunakan adalah Git. Makalah ini menjelaskan aplikasi teori graf dalam memodelkan riwayat perubahan oleh Git. Juga penjelasan mengenai fitur-fitur dari Git seperti commit, branch, dan merge.**

***Kata Kunci*—Branch, Commit, Git, Graf Berarah.**

## I. Pendahuluan

*Version Control System* (VCS) adalah sistem untuk merekam perubahan pada suatu file sehingga penggunanya dapat berpindah ke suatu versi tertentu untuk file tersebut. VCS sangat berguna untuk berbagai pekerjaan, contohnya desainer grafis atau developer perangkat lunak. VCS memungkinkan beberapa file untuk kembali ke keadaan sebelumnya, membandingkan perubahan yang terjadi, melihat perubahan mana yang mengakibatkan masalah dan masih banyak lagi.

Salah satu VCS yang paling sering digunakan oleh developer adalah Git. Git mengelola penyimpanan versi dengan menggunakan yang disebut commit. Banyak sekali fitur dari Git yang dapat digunakan untuk memudahkan pekerjaan. Namun untuk menggunakan fitur-fitur tersebut, perlu adanya pemahaman mendasar mengenai cara kerja dari Git dalam menyimpan dan mengelola perubahan versi file. Fitur branch dari Git adalah salah satu fitur yang membuat Git sangat unggul sebagai VCS. Dibandingkan dengan VCS lainnya, Git melakukan branch dengan sangat ringan dan perpindahan antara branch secara umum sangatlah cepat. Memahami cara kerja dari Git tidak hanya memudahkan dalam pekerjaan menyimpan versi, melainkan juga membantu dalam berkolaborasi dengan orang lain. Tidak hanya sudah digunakan oleh sebagian besar developer, perusahaan-perusahaan besar Google, Facebook, Microsoft dan lain-lain telah menggunakan Git sebagai pembantu proyek-proyek mereka.

Graf adalah struktur diskrit yang memodelkan sesuatu dalam simpul dan sisi. Graf digunakan untuk memodelkan sebuah sistem menjadi representasi yang sederhana dengan himpunan simpul dan sisi saja. Tidak hanya dapat menyederhanakan pemodelan sistem, dengan graf cara kerja dan alur dari suatu sistem tersebut juga dapat dengan mudah dipahami. Pemodelan alur kerja dari Git dan beberapa konsep lainnya seperti commit, branch, dan merge dapat dengan mudah dimodelkan dengan menggunakan graf. Graf yang cocok digunakan untuk memodelkan VCS adalah graf berarah, dengan tambahan beberapa pointer yang menunjukkan branch tertentu.

## II. Dasar Teori

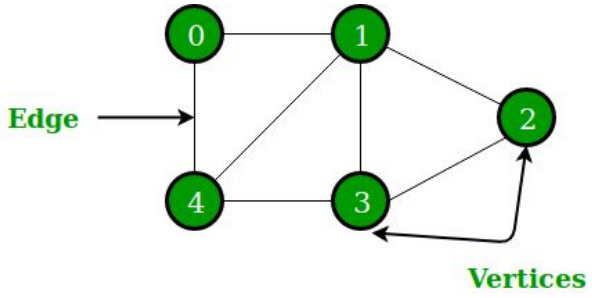
### A. Graf

**Graf** adalah struktur diskrit dalam matematika yang berisikan himpunan **simpul** (atau vertex) dan **sisi** yang menghubungkan antar simpul. Secara formal, sebuah graf $G$ didefinisikan sebagai

$$G = (V, E)$$

dengan $V$ adalah himpunan tidak kosong simpul dan $E$ adalah himpunan sisi. Setiap elemen pada $E$ adalah sebuah pasangan $(u, v)$ yaitu $u$ dan $v$ masing-masing adalah sebuah simpul yang ada pada $V$.

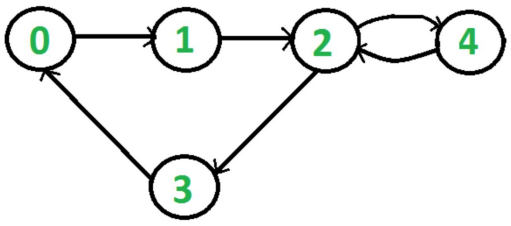
Ada banyak cara untuk merepresentasikan graf. Representasi graf yang paling sering digunakan adalah seperti yang ada pada gambar 1. Sekumpulan titik merepresentasikan simpul dan sekumpulan garis merepresentasikan sisi.




Gambar 1: Representasi graf
Sumber: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

*B. Graf Berarah*

Graf yang memiliki arah disebut dengan **graf berarah**. Pada graf berarah, sisi bisa juga disebut dengan busur. Setiap busur (*u*, v) mendefinisikan bagaimana sisi pada dua buah simpul *u* dan *v*. Simpul *u* disebut dengan **simpul asal** dan simpul *v* disebut dengan **simpul terminal**. Graf berarah membolehkan adanya sisi gelang namun tidak dengan sisi ganda. Representasi gambar graf berarah dapat terlihat di gambar 2.




Gambar 2: Graf berarah
Sumber: https://www.geeksforgeeks.org/detect-cycle-in-a-directed-graph-using-bfs/

*C. Terminologi Pada Graf Berarah*

Terminologi pada graf berarah adalah sebagai berikut:

1. Ketetanggaan (*Adjacent*)
   Pada sisi (*u*, *v*) dari sebuah graf berarah, *u* bertetangga dengan *v* dan *v* bertetangga dengan *u*.
2. Derajat (*Degree*)
   Pada graf berarah, derajat sebuah simpul *v* dibagi menjadi dua jenis, yaitu derajat masuk (*in-degree*) dan derajat keluar (*out-degree*). Derajat masuk dari simpul *v* adalah jumlah sisi dengan *v* sebagai simpul terminalnya. Derajat keluar dari simpul *v* adalah jumlah sisi dengan *v* sebagai simpul asalnya.
3. Lintasan, Sirkuit, dan Keterhubungan
   Pada graf berarah *G,* lintasan dari *u* ke *v* pada *G* adalah barisan sisi-sisi *e1*, *e2*, . . ., *en* dari *G* dengan *e1* adalah (*x0*, *x1*), *e2* adalah (*x1, x2*) dan seterusnya, dengan *en* adalah (*xn-1*, *xn*), dimana *x0* adalah *u* dan *xn* adalah *v*. Lintasan yang bermula dan berakhir pada simpul yang sama disebut dengan sirkuit. Lintasan atau sirkuit sederhana adalah yang tidak mengandung sisi yang sama lebih dari satu kali. Jika ada lintasan berarah dari *u* ke *v* dan ada lintasan berarah dari *v* ke *u*, maka dua simpul *u* dan *v* disebut terhubung kuat.
4. Terhubung dan *Cut-Set*
   Dua buah simpul *u* dan *v* disebut terhubung jika memiliki satu atau lebih lintasan. Graf *G* disebut graf terhubung jika untuk setiap pasangan simpul pada graf saling terhubung. *Cut-set* pada graf terhubung *G* adalah himpunan sisi yang jika salah satu sisi tersebut dibuang mengakibatkan *G* menjadi tidak terhubung
5. Upagraf
   Sebuah graf $H = (W, F)$ adalah upagraf dari graf $G = (V, E)$ jika $W \subseteq V$ dan $F \subseteq E$.

*E. Pohon*

Pohon adalah graf terhubung yang tidak mengandung sirkuit. Sebuah pohon dengan *n* simpul akan memiliki *n* - 1 sisi. Penambahan sebuah sisi pada pohon akan membentuk tepat satu buah sirkuit. Pohon yang pada salah satu simpulnya dijadikan seperti akar dan sisi-sisinya diberikan arah disebut dengan pohon berakar.

*F. Definisi Version Control System*

*Version Control System* (VCS) adalah sebuah sistem yang menyimpan rekaman perubahan dari sebuah dokumen, program komputer, atau informasi lain. Perangkat lunak penyedia VCS diantaranya adalah CVS, Subversion, Peforce, Mercurial, dan Git. VCS sangat cocok untuk mengelola file yang berupa program komputer.

*G. Jenis-Jenis Version Control System*

Ada beberapa jenis VCS yang ada, diantaranya *Local Version Control Systems, Centralized Version Control Systems,* dan *Distributed Version Control Systems.*

1. *Local Version Control Systems*
   Pada jenis VCS lokal, versi dari file disimpan pada sebuah basis data sederhana dalam komputer lokal yang menyimpan semua perubahan. Umumnya VCS jenis ini menyimpan bagian-bagian yang berbeda dari sebuah file.
2. *Centralized Version Control Systems*
   Pada jenis ini, versi dari file disimpan pada sebuah server. Dengan sistem demikian memungkinkan untuk beberapa orang untuk saling berkolaborasi. Setiap pihak yang tergabung dengan sistem akan menyimpan duplikat dari file pada server.
3. *Distributed Version Control Systems*
   Berbeda dengan *Centralized* VCS yang menyimpan riwayat perubahannya pada sebuah server, pada *Distributed VCS*, semua pihak yang tergabung dengan sistem tidak hanya menyimpan duplikat file, melainkan riwayat perubahan juga. Salah satu VCS jenis ini adalah Git.

## III. *Aplikasi Teori Graf Dalam Memodelkan Version Control System Git*

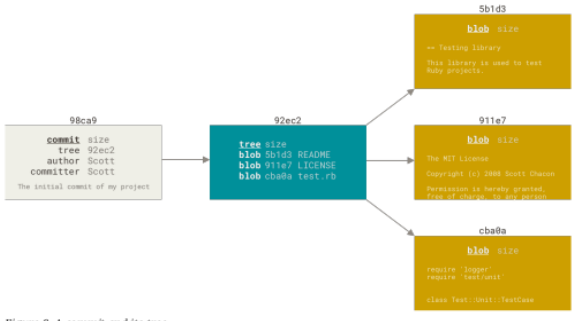
*A. Penjelasan Mengenai Git*


Gambar 3: Logo Git

**Git** adalah salah satu dari *distributed* VCS yang memungkinan untuk mengelola perubahan file di dalam folder. Folder yang digunakan untuk menyimpan disebut dengan **repository** / **repo**. Git menyimpan rangkaian perubahan pada

file dengan menggunakan yang disebut dengan **commit**. Setelah file dilakukan commit maka Git akan menyimpan rekaman dari file tersebut.

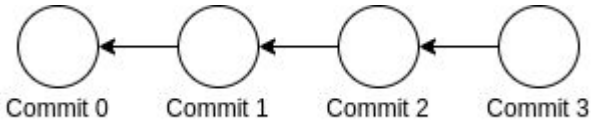
Ketika sebuah commit dibuat, Git akan mencatat semua subfolder dalam repository dan menyimpannya dalam sebuah objek pohon dan sebuah pointer yang menunjuk ke akar pohon tersebut. Jika sebuah commit dibuat lagi, commit tersebut akan menunjuk ke commit tepat sebelumnya.




Gambar 4: Pohon yang terbentuk dari setiap commit yang dibuat
Sumber: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

## *B. Graf pada Riwayat Commit*

Rangkaian commit yang dibuat disimpan dalam bentuk yang bisa direpresentasikan sebagai graf berarah.




Gambar 5: Model graf berarah untuk sebuah rangkaian commit
Sumber: draw.io

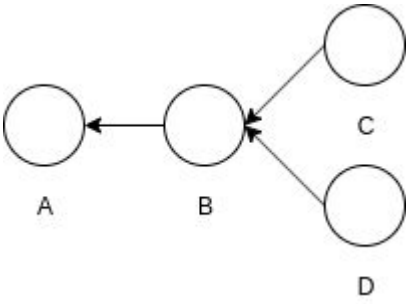
Pada graf riwayat commit tersebut, simpul yang tidak memiliki busur keluar atau simpul dengan derajat keluar sama dengan 0 adalah commit pertama yang dibuat pada repository tersebut. Busur $e = (u, v)$ menunjukkan commit dari simpul $u$ dilakukan tepat setelah commit $v$. Setiap simpul commit terdiri dari beberapa jenis informasi yaitu kode hash dari commit tersebut, identitas pembuat commit, waktu commit, dan pesan commit.

## *C. Branch Pada Git*

Umumnya graf yang terbentuk hanya terdiri dari satu buah alur saja. Atau pada sebuah simpul $v$ hanya akan ada satu lintasan yang menghubungkan v dengan simpul awal. Setiap simpul pada graf tersebut memiliki maksimal derajat masuk dan derajat keluar sebanyak 1.

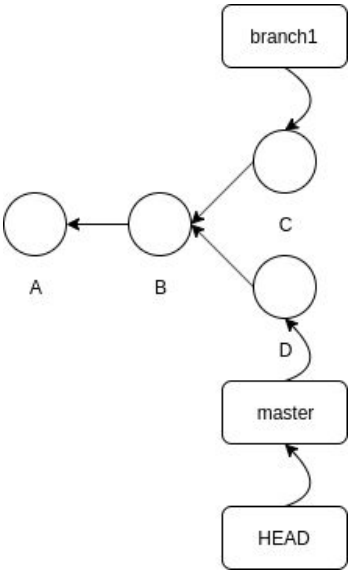
Dalam Git, setiap commit akan membuat simpul baru yang bertetangga dengan commit sebelumnya. Namun ketika Git melakukan yang disebut dengan **branch**, commit berikutnya dapat dilakukan pada satu atau beberapa simpul baru yang berbeda dari alur utama. Ilustrasi dari branch dapat dilihat pada gambar 6

.




Gambar 6: Ilustrasi branch
Sumber: draw.io

Untuk menandai branch tersebut, terdapat **pointer** dengan nama sesuai nama dari suatu branch misalnya pointer master sebagai penunjuk untuk branch master. Setiap pointer ini menunjuk kepada sebuah commit tempat branch ini bekerja. Ada juga pointer khusus yang menunjuk pada branch yang sedang aktif bernama HEAD.

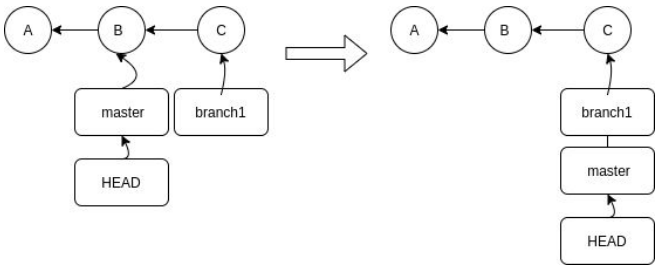



Gambar 7: Pointer pada graf riwayat commit
Sumber: draw.io

## *D. Merge Pada Git*

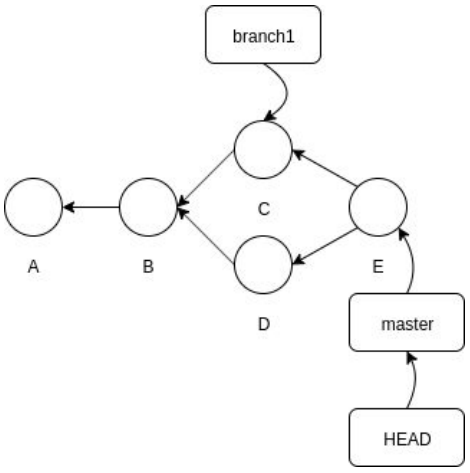
Ketika sebuah branch selesai dikerjakan, alur dari commit pada branch tersebut dapat digabungkan dengan branch utama lagi. Proses penggabungan ini disebut dengan **merge**. Git melakukan merge dengan dua macam jenis tergantung pada posisi dari commit yang ditunjuk antar branch yang ingin di-merge.

Jenis pertama disebut dengan "*fast-forward*". Merge jenis ini terjadi ketika dua buah commit yang ditunjuk oleh branch yang ingin di-merge saling bertetangga. Merge ini dilakukan dengan memindahkan pointer pada branch yang menunjuk pada simpul terminal menuju simpul asal. Hasilnya kedua branch akan menunjuk pada sebuah simpul yang sama.




Gambar 8: Ilustrasi graf ketika dilakukan “fast-forward” merge
Sumber: draw.io

Jenis merge satunya lagi disebut dengan “*three-way*” merge. Merge jenis dilakukan dengan membuat commit baru hasil penggabungan dua buah commit dari branch yang ingin di merge dengan branch satunya. Ilustrasi graf nya adalah dengan membuat simpul baru yang memiliki dua buah lintasan menuju simpul tempat kedua branch pertama kali terpisah..




Gambar 9: Ilustrasi graf ketika dilakukan “three-way” merge
Sumber: draw.io

Perhatikan pada gambar 9. Simpul E akan memiliki dua buah lintasan menuju simpul B yaitu melalui sisi-sisi (E, C), (C, B) dan sisi-sisi (E, D), (D, B). Simpul commit yang dibuat hasil dari merge akan ditunjuk oleh branch yang dilakukan merge.

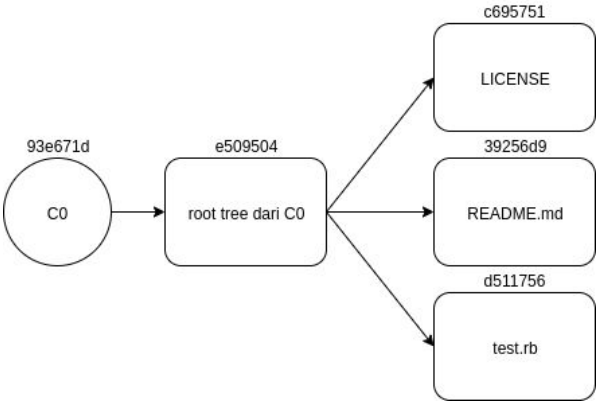
## IV. Studi Kasus Penggunaan Git dan Aplikasi Graf Di Dalamnya

### *A. Struktur Direktori*

Buat sebuah folder yang diinginkan untuk menjadi repository dengan cara sebagai berikut.

```
$ git init
Initialized    empty    Git    repository    in
/home/roger/website-matdis/.git/
$ git add README.md test.rb LICENSE
$ git commit -m “C0”
```

ketika melakukan sebuah commit, Git akan memeriksa setiap file yang di-commit pada direktori lalu menyimpannya pada sebuah objek yang berupa pohon. Setelah itu Git akan membuat sebuah objek commit yang mempunya pointer ke pohon. Struktur dari pohon yang dibuat dapat dilihat pada gambar 10.

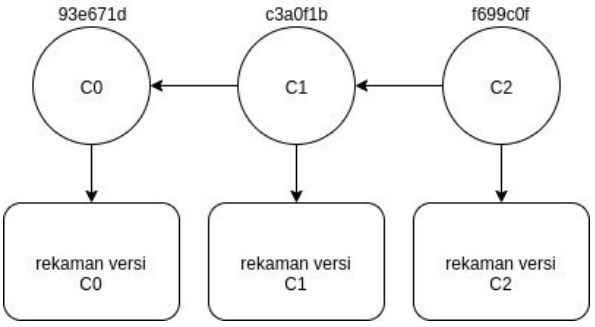



Gambar 10: Pohon dari sebuah commit
Sumber: draw.io

Objek pohon pada gambar 10 terdiri dari sebuah root dan mempunyai anak yang berupa daun pada pohon. Setiap daun adalah sebuah rekaman versi dari file yang di-commit pada C0.

### *B. Graf Commit*

Pada commit berikutnya, git akan membuat commit tersebut seperti pada commit pertama dengan tambahan membuat sisi berarah (busur) dari commit baru ke commit sebelumnya.




Gambar 11: Setelah dilakukan beberapa kali commit
Sumber: draw.io

Berikutnya sebuah object commit dan pointer dari commit ke rekaman versinya dianggap sebagai satu kesatuan simpul saja.

### *C. Branch dan Merge*

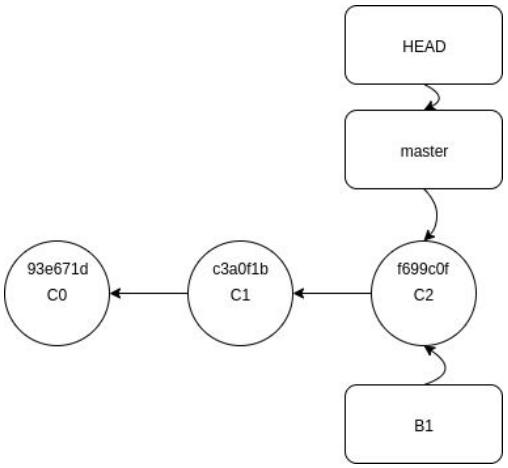
Untuk melihat pointer branch dan HEAD adalah sebagai berikut.

```
$ git log --all --decorate --oneline --graph
* f699c0f (HEAD -> master) C2
* c3a0f1b C1
* 93e671d C0
```

Pointer HEAD sedang menunjuk pada branch master. Branch master sendiri menunjuk pada commit f699c0f atau C2. Sekarang akan dibuat branch baru bernama B1.

```
$ git branch B1
```

Maka ilustrasi yang terjadi pada graf adalah sebagai berikut.

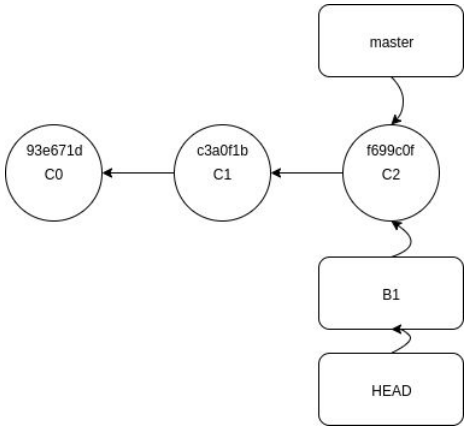



Gambar 12: Ilustrasi pointer branch
Sumber: draw.io

Pada gambar 12 terlihat branch yang sedang ditunjuk adalah branch master. Untuk berpindah ke branch lain jalankan perintah berikut:

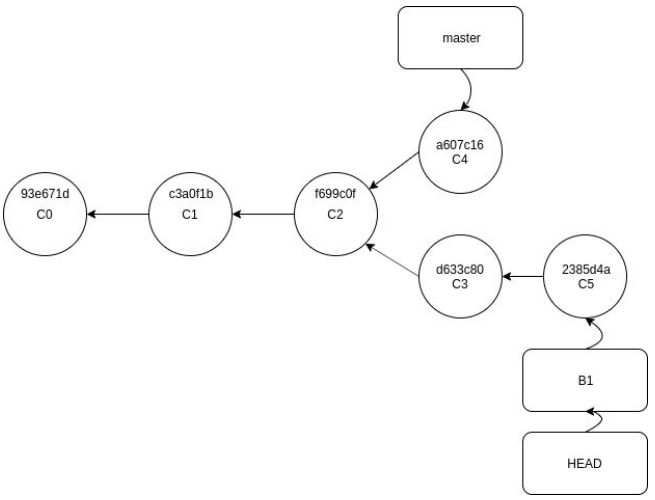
```
$ git checkout B1
```

HEAD akan berpindah menunjuk B1.



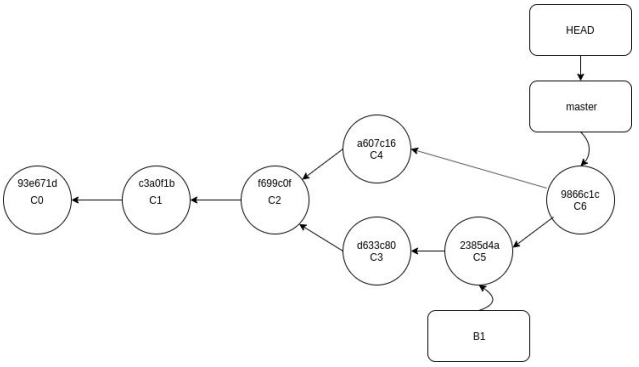

Gambar 13 Ilustrasi pointer branch 2
Sumber: draw.io

Setelah HEAD berpindah ke branch B1, sekarang lakukan beberapa sebuah commit C3 pada branch tersebut. Kemudian kembali lagi ke branch master dengan menggunakan checkout lalu lakukan commit C4 pada branch master. Lalu checkout kembali ke branch B1 dan lakukan commit C5. Hasil dari proses tersebut akan menjadikan ilustrasi graf nya menjadi demikian:




Gambar 14 Ilustrasi graf setelah dilakukan beberapa kali commit
Sumber: draw.io

Apabila dilakukan merge dari branch master dengan branch B1, maka proses yang terjadi adalah "*three-way*" merge. Git akan membuat simpul baru yaitu sebuah commit hasil merge antara kedua branch tersebut. Simpul ini akan memiliki dua buah busur keluar. Satu busur akan menunjuk pada C4, yaitu commit terakhir yang ditunjuk oleh branch master, dan satu busur lagi akan menunjuk pada C5, yaitu commit terakhir yang ditunjuk oleh branch B1. Commit yang terbentuk akan memiliki dua buah lintasan menuju ke C2, yaitu commit yang menjadi terhubung dengan C4 dan C5.




Gambar 15 Ilustrasi graf setelah dilakukan merge dengan "*three-way*" merge
Sumber: draw.io

Jika kita melihat kembali bagaimana keadaan graf dapat melakukan cara:

```
$ git log --all --decorate --oneline --graph
*   9866c1c (HEAD -> master) C6 Merge branch 'B1'
|\
| * 2385d4a (B1) C5
| * d633c80 C3
* | a607c16 C4
|/
* f699c0f C2
* c3a0f1b C1
* 93e671d C0
```

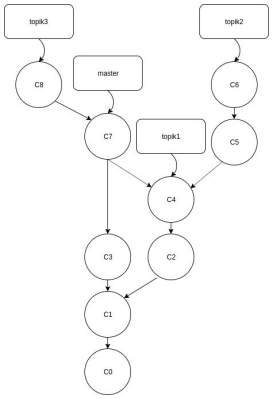
dapat dilihat bahwa Git mencetak graf yang sesuai pada gambar 15.

### *D. Analisis Graf Riwayat Commit*

Pada saat bekerja dengan Git, pembuatan branch seperti pada bagian A sampai C dapat diimplementasikan untuk melakukan eksperimen tanpa takut mengganggu alur kerja utama. Apabila eksperimen dirasa berhasil, maka bisa langsung dilakukan merge dengan branch utama. Alur kerja juga dapat dimodifikasi dengan membuat branch yang tepat. Graf riwayat commit yang dibuat dengan Git memiliki beberapa karakteristik, antara lain:

1. Memiliki satu simpul yang tidak memiliki busur keluar. Simpul ini disebut simpul pertama, yaitu commit pertama yang dibuat pada suatu repository. Semua simpul lain pada graf riwayat commit selalu terhubung dengan simpul pertama ini.
2. Ketika sebuah commit dibuat pada suatu branch, akan dibentuk simpul baru dengan busur yang menghubungkan simpul baru tersebut dengan simpul yang sebelumnya ditunjuk branch tersebut. Busur awal terletak pada simpul baru dan simpul terminal pada simpul sebelumnya. Setelah terbentuk simpul tersebut, branch akan berpindah menunjuk pada simpul baru tersebut.
3. Melakukan commit di branch yang berbeda akan membuat sebuah simpul tempat branch tersebut dibuat memiliki lebih dari satu busur masuk.
4. Melakukan merge antar dua branch akan membentuk lebih dari satu lintasan antara simpul baru hasil merge dengan simpul tempat branch dibuat.
5. Versi tertentu dari suatu file dapat diakses dengan memindahkan pointer HEAD ke branch atau simpul yang diinginkan.

Cara kerja merge Git yang menggunakan "*three-way*" merge ini membuat alur kerja dengan menggunakan Git ini menjadi sangat fleksibel. Pemanfaatan dari sistem merge Gir ini yaitu dalam alur kerja yang biasa digunakan dengan branch, yaitu dengan membuat beberapa branch untuk topik-topik tertentu. Salah satu contoh ilustrasinya adalah seperti pada gambar 16.




Gambar 16 Ilustrasi alur kerja Git dengan banyak branch
Sumber: draw.io

## V. Kesimpulan

Teori Graf memiliki aplikasi yang sangat luas di berbagai bidang. Penerapan teori graf dalam memodelkan *Version Control System* Git menunjukkan bagaimana konsep alur kerja dari Git dapat dengan mudah direpresentasikan dalam graf dengan berarah. Dengan pemodelan alur kerja pada Git dengan menggunakan graf, akan lebih mudah bagi seseorang untuk memahami apa yang sedang terjadi pada alur kerja dan versi file itu sendiri.

## VI. Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada dosen mata kuliah IF2120 Matematika Diskrit, Bapak Dr. Ir. Rinaldi Munir, MT., Ibu Harlili M.Sc, dan Ibu Fariska Zakhralativa, M.T. atas ilmu dan bimbingannya selama 1 semester ini. Penulis juga mengucapkan terima kasih kepada orang tua dan seluruh keluarga dekat penulis atas dukungannya selama ini. Dan terakhir, penulis juga mengucapkan terima kasih kepada teman-teman satu jurusan di Teknik Informatika baik yang satu angkatan maupun kakak tingkat yang telah membantu memberikan ide dan referensi bagi saya untuk menulis makalah ini.

## Referensi

[1] Rosen, Kenneth H. , *Discrete Mathematics and Its Application*. New York : McGraw-Hill, 2012, pp. 641 – 727

[2] Munir, Rinaldi. *Matematika Diskrit,* ed, 3. Penerbit Informatika, 2010, pp. 353-437.

[3] Chacon, Scott and Ben Straub. *Pro Git*. The Export's Voice, 2019, pp. 9-101.

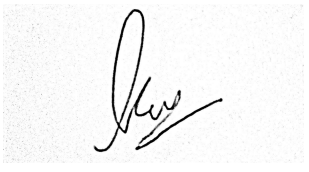
[4] Zvonimir Spajic "*Understanding Git - Data Model"*, https://hackernoon.com/https-medium-com-zspajich-understanding-git-data-model-95eb16cc99f5, diakses pada 5 Desember 2019, 19.00 WIB.

[5] Zvonimir Spajic "*Understanding Git - Branching*", https://hackernoon.com/understanding-git-branching-2662f5882f9, diakses pada 5 Desember 2019, 20.00 WIB.

## Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 6 Desember 2019



Kevin Rizki Mohammad 13518100