Penggunaan Struktur Graf dalam Pengontrol Versi Git

Devin Alvaro
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515062@std.stei.itb.ac.id

***Abstract*—Pada pengembangan perangkat lunak, terutama yang dilakukan oleh sekelompok pengembang (developer), agar dapat berjalan dengan baik diperlukan pengontrol versi untuk mengelola file-file program, agar menghindari konflik dan *bug* pada program. Salah satu pengontrol versi yang paling banyak digunakan adalah Git. Pada implementasinya, Git menggunakan struktur graf untuk mengindeks file-file yang dikelolanya dan mencatat perubahan yang dilakukan padanya. Pemahaman akan stuktur graf pada Git akan membantu seseorang menggunakannya secara optimal. Makalah ini bertujuan membahas struktur graf pada Git serta bagaimana memanipulasinya melalui perintah-perintah pada Git.**

***Keywords*—Struktur Graf, Properti Graf, Pengontrol Versi, Git.**

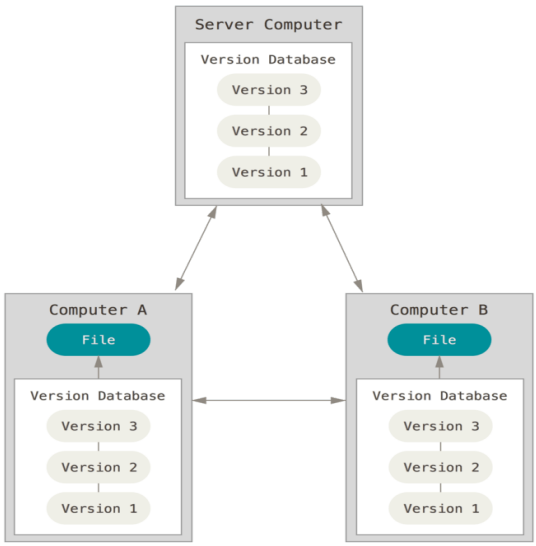
I. Pendahuluan

Majunya perkembangan teknologi di era digital ini tidak terlepas dari pesatnya pengembangan perangkat lunak yang menjalankannya. Agar pengembangan perangkat lunak berkembang pesat, pengembangan seringkali tidak dilakukan oleh seorang pengembang (*developer*) saja, melainkan oleh kelompok pengembang yang bekerja sama dalam mengembangkan suatu perangkat lunak.

Akan tetapi, bila tidak dikelola dengan baik, pengembangan perangkat lunak oleh banyak pengembang dapat dengan mudah menimbulkan hal-hal seperti kode yang konflik dan banyak *bug*. Untuk itulah diciptakan pengontrol versi (*version control)* seperti CVS (Concurrent Versions System), Bazaar, dan Git yang bertujuan untuk memudahkan kelompok pengembang dalam mengelola kode-kode yang ditulis oleh masing-masing pengembang agar tidak bertindihan, menghindari konflik, serta memudahkan penanganan *bug*.

Pengontrol versi memudahkan kelompok pengembang perangkat lunak karena dapat merekam perubahan-perubahan yang dilakukan terhadap suatu program, membaginya menjadi versi-versi yang berurutan. Apabila seorang pengembang merubah suatu bagian kode pada program, perubahan itu akan dicatat dan akan dibuat versi baru dari program tersebut. Sehingga bila suatu saat ditemukan eror yang fatal dari perubahan tersebut, program dapat dikembalikan lagi ke versi sebelumnya.

Selain itu, pengontrol versi juga memastikan bahwa perubahan pada program yang dilakukan oleh seorang pengembang akan sampai ke pengembang yang lain, sehingga versi program di komputer setiap pengembang selalu sama. Hal ini penting untuk menghindari kode yang saling bertindihan. Dengan demikian, pengembangan perangkat lunak dapat dilakukan secara berkesinambungan.



Gambar 1: Gambaran pengontrol versi pada beberapa komputer dengan server utamanya
(sumber: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control, diakses 7 Desember 2016, pukul 22:13)

Salah satu pengontrol versi yang paling banyak digunakan saat ini adalah Git. Dalam implementasinya, Git menggunakan konsep graf untuk mengindeks file-file pada program serta mencatat perubahan-perubahan yang dilakukan padanya. Melakukan perintah-perintah pada Git berarti memanipulasi struktur grafnya. Oleh karena itu, penting bagi seorang pengembang untuk memahami struktur graf pada Git agar dapat menggunakannya secara optimal.

II. DASAR TEORI

Dasar teori yang digunakan adalah teori graf yang mencakup definisi graf, arah graf, dan representasi graf. Teori graf termasuk dalam cakupan matematika diskrit[1].

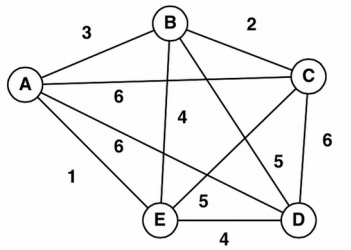
2.1 Graf

Dalam matematika diskrit, graf adalah struktur yang merepresentasikan relasi objek-objek suatu himpunan. Objek tersebut disebut simpul (*vertice*), sedangkan penghubung simpul-simpul tersebut disebut sisi (*edge)*. Sisi dinotasikan dengan (*p*, *q*), di mana *p* dan *q* adalah simpul.

Secara umum, graf dinotasikan dengan:

$$G = (V, E)$$

di mana *V* adalah himpunan tidak kosong dari simpul pada graf, dan *E* adalah himpunan dari sisi pada graf.



Gambar 2: Graf dengan 4 simpul dan 10 sisi
(sumber: http://i.stack.imgur.com/tMaUp.png, diakses pada 7 Desember 2016, pukul 22:48)

2.2 Jenis Graf

Berdasarkan ada tidaknya sisi ganda atau gelang (*loop*) pada graf, graf dibagi menjadi 2 jenis, yaitu graf sederhana dan tidak sederhana.

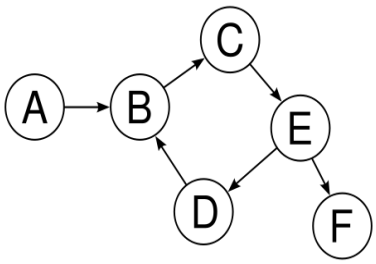
Graf sederhana adalah graf yang tidak memiliki sisi ganda, yaitu 2 atau lebih sisi yang menghubungkan 2 buah simpul yang sama, atau gelang, yaitu sisi yang berasal dan menuju simpul yang sama.

Graf tidak sederhana adalah graf yang memiliki sisi ganda atau gelang. Graf yang memiliki sisi ganda disebut graf ganda, sedangkan graf yang memiliki gelang disebut graf semu.

Menurut orientasi arah sisi-sisinya, graf dibagi menjadi 2 jenis, yaitu graf tidak berarah dan berarah.

Graf tidak berarah adalah graf yang setiap sisinya dapat berasal dari dan menuju kedua pasangan simpul, sehingga (*p*, *q*) sama dengan (*q*, *p)*. Graf pada Gambar 2 adalah contoh graf tidak berarah.

Graf berarah adalah graf yang setiap sisinya memiliki arah tertentu. Apabila suatu sisi menghubungkan dua buah simpul, misalnya *p* dan *q*, sisi tersebut harus berasal dari salah satu dari kedua simpul dan arahnya menuju simpul yang satunya. Pada graf berarah, (*p*, *q*) tidak sama dengan *(q*, *p*).



Gambar 3: Graf berarah
(sumber: http://www.mrgeek.me/wp-content/uploads/2014/04/directed-graph.png, diakses pada 7 Desember 2016, pukul 23:16)

2.3 Representasi Graf

Struktur graf sering dipakai dalam program komputer, misalnya struktur data *map* dan *set* pada bahasa pemrograman yang biasanya menggunakan struktur graf. Maka dari itu, terdapat beberapa representasi fisik graf pada komputer, yaitu matriks ketetanggaan (*adjacency matrix*), matriks bersisian (*incidency matrix*), dan senarai ketetanggan (*adjacency list*).

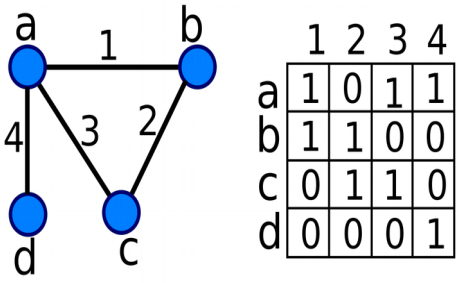
Matriks ketetanggaan merepresentasikan graf sebagai matriks dua dimensi yang menunjukkan hubungan setiap pasangan simpul pada graf, bernilai 1 apabila dihubungkan oleh suatu sisi, bernilai 0 bila sebaliknya. Berikut adalah contoh matriks ketetanggan untuk graf berarah pada Gambar 3:

	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	1	0	0	0
C	0	0	0	0	1	0
D	0	1	0	0	0	0
E	0	0	0	1	0	1
F	0	0	0	0	0	0

Matriks ketetanggaan untuk graf berarah pada Gambar 3

Kekurangan dari matriks ketetanggaan adalah banyak memori yang terbuang untuk merepresentasikan dua simpul yang tidak bersisian (bernilai 0).

Matriks bersisian juga merepresentasikan graf sebagai matriks, namun hanya hubungan setiap simpul dengan setiap sisi pada graf. Representasi ini lebih menghemat memori komputer dibanding matriks ketetanggaan.



Gambar 4: Representasi Matriks Bersisian
(sumber: https://upload.wikimedia.org/wikipedia/commons/thumb/5/5a/Incidence_matrix_-_undirected_graph.svg/1280px-Incidence_matrix_-_undirected_graph.svg.png, diakses 7 Desember 2016, pukul 23.52)

Senarai ketetanggaan merepresentasikan graf sebagai list linear, di mana setiap simpul diikuti oleh simpul-

simpul yang berhubungan dengannya sebagai list linear. Representasi ini memiliki keunggulan dalam penggunaan memori, karena tidak ada memori yang terbuang untuk representasi simpul yang tidak bersisian. Namun, karena menggunakan list linear, iterasi menjadi lebih lambat dibanding bila menggunakan matriks. Berikut adalah contoh senarai ketetanggaan untuk graf berarah pada Gambar 3:

A: B
B: C
C: E
D: B
E: D, F
F: -

Senarai ketetanggaan untuk graf berarah pada Gambar 3

2.4 Terminologi Graf

Berikut ini adalah istilah-istilah yang biasa digunakan dalam teori graf:

1. Bertetangga (*Adjacent*)

Dua buah simpul *p* dan *q* dikatakan bertetangga bila ada sisi *e* di mana *e = (p, q)*.

2. Bersisian

Sebuah sisi *e* dikatakan bersisian dengan simpul *p* dan *q* apabila terdapat *e = (p, q)*.

3. Simpul terpencil

Simpul terpencil adalah simpul pada graf yang tidak bersisian dengan sisi manapun.

4. Graf kosong

Graf kosong adalah graf yang himpunan sisinya merupakan himpunan kosong, G = (V, E) di mana E = {}.

5. Derajat

Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut.

6. Lintasan

Dua buah simpul dikatakan memiliki lintasan apabila terdapat rangkaian sisi yang menghubungkan kedua simpul tersebut.

7. Terhubung

Dua buah simpul dikatakan terhubung apabila terdapat lintasan yang menghubungkan dua buah simpul tersebut.

8. Upagraf

Upagraf dari graf G adalah graf yang himpunan simpul dan sisinya merupakan subhimpunan himpunan sisi dan simpul graf G.

III. Git

Pada bagian ini akan dijelaskan latar belakang Git dan dasar-dasar Git sebagai pendahuluan sebelum membahas struktur graf pada Git.

Git diciptakan oleh Linus Torvalds pada tahun 2005 karena ketidakpuasannya akan pengontrol versi yang ada untuk mengelola proyek *open-source* Linux-nya. Tidak seperti pengontrol versi lain yang tersentralisasi, Git merupakan pengontrol versi terdistribusi, yang berarti setiap pengembang dapat meng-*clone* seluruh direktori proyek yang dikelola Git ke sistem komputernya. Tidak seperti pengontrol versi tersentralisasi di mana hanya server utama yang memiliki akses penuh atas seluruh direktori proyek[2].

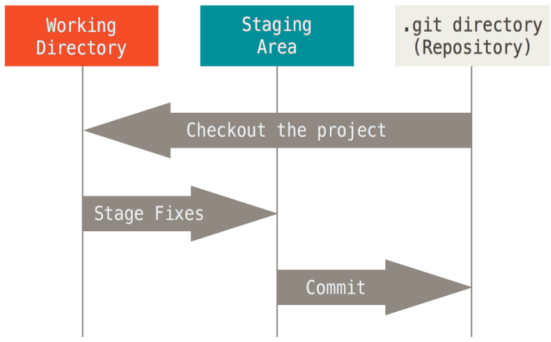
Terdapat tiga bagian utama pada proyek yang menggunakan Git. Yang pertama adalah *working directory* atau direktori kerja, yaitu folder biasa tempat file-file program berada. Di sini pengembang dapat melakukan perubahan pada file-file program seperti biasa, tanpa dicatat oleh Git. Setelahnya adalah *staging area*. Pengguna dapat memasukkan file ke *staging area* dengan perintah ‘git add <nama file>’. File-file yang telah dimasukkan ke *staging area* mulai dicatat perubahannya oleh Git.

Ketika file dimasukkan ke *staging area*, sebenarnya Git hanya menyimpan ‘rekaman’ dari file tersebut yang berupa hash sepanjang 40 karakter yang unik untuk setiap isi file. Misalnya, dua file teks berbeda yang sama-sama berisi tulisan ‘halo’ akan dicatat sebagai *hash* yang sama, sedangkan file teks lain yang berisi tulisan ‘helo’ akan dicatat sebagai hash yang berbeda. Berikut adalah contoh *hash* yang dicatat oleh Git: 24b9da6552252987aa493b52f8696cd6d3b00373.

Bagian terakhir adalah *.git directory*, di mana versi-versi program disimpan oleh Git. Di sinilah Git membuat versi baru program, ketika file di *staging area* dimasukkan ke *.git directory*. Pengguna dapat memasukkan file ke *.git directory* dengan perintah ‘git commit -m “<pesan versi>”’.

Secara umum, alir kerja dalam menggunakan Git adalah sebagai berikut:

1. Melakukan perubahan pada file program di direktori kerja
2. Menambahkan file ke *staging area* agar dicatat perubahannya ke Git.
3. Melakukan *commit* ke *.git directory* agar Git membuat versi baru program yang mencakup perubahan file tersebut.



Gambar 5: Gambaran bagian-bagian utama Git (sumber: https://git-scm.com/book/en/v2/book/01-introduction/images/areas.png, diakses pada 9 Desember 2016, pukul 07.03)

IV. Struktur Graf pada Git

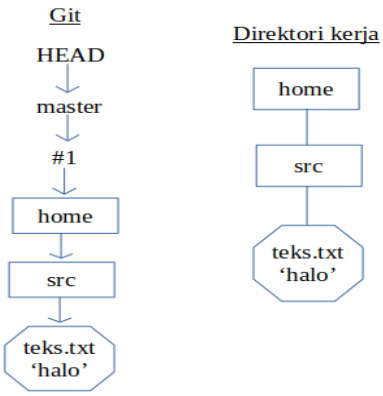
Pada bagian sebelumnya, telah dibahas tiga bagian utama pada pada proyek yang menggunakan Git. Pada bagian ini, akan dibahas peran struktur graf pada Git. Struktur graf yang ditunjukkan adalah penyederhanaan dari struktur sebenarnya, mengingat tujuan utama dari pembahasan ini adalah memahami garis besar struktur graf pada Git.

Working directory adalah tempat file-file program berada. Sebagai contoh, pada pembahasan ini direktori kerja berada di folder bernama *home*. Folder *home* memiliki folder *src* dan file teks *teks.txt* yang berisi kata ‘halo’.

Untuk mulai menggunakan git untuk mengelola file, pengguna dapat memberi perintah ‘git init’ melalui *terminal* pada direktori kerja tersebut. Selanjutnya, pengguna memasukkan teks.txt ke *staging area* dengan perintah ‘git add teks.txt’.

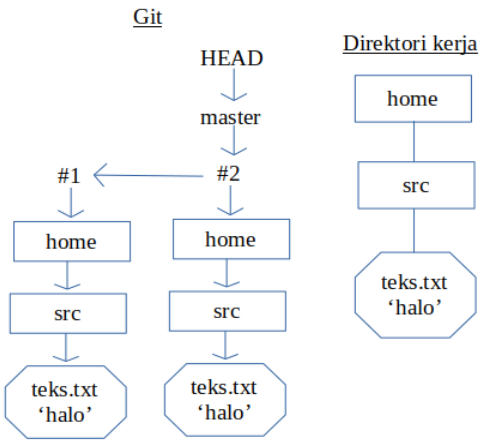
Seperti telah dijelaskan sebelumnya, git mencatat file-file yang dimasukkan ke *staging area* dalam rupa *hash* 40 karakter. Rekaman-rekaman ini dihubungkan oleh *pointer* dari HEAD, seperti pada pada list linear. Master di sini berarti cabang (*branch*) utama pada Git. Percabangan akan dijelaskan lebih lanjut.

Setelah file-file dicatat ke *staging area*, akan dilakukan *commit* pada file agar Git membuat versi pertama dari direktori Git ini. *Commit* dilakukan dengan perintah ‘git commit -m “#1”’. “#1” di sini adalah pesan *commit*, yang berguna untuk menandakan versi-versi yang berbeda, serta mengkomunikasikan kepada pengembang lain apa saja perubahan-perubahan yang dilakukan pada versi tertentu.



Gambar 7: Direktori setelah dilakukan commit “#1”

Struktur graf berarah pada Git akan mulai terlihat setelah pengguna melakukan *commit* lagi untuk membuat versi baru. Misalnya, sekarang pengguna mengubah isi teks.txt menjadi ‘mate’. Kemudian, pengguna menambahkan teks.txt yang telah diubah ke *staging area* lagi, selanjutnya melakukan commit dengan pesan “#2”. Berikut ini adalah gambaran direktori Git setelah dibuat versi baru “#2”.



Gambar 8: Direktori setelah dilakukan commit “#2”

Dapat diperhatikan bahwa ketika membuat versi baru, Git hanya memindahkan pointer yang mengarahkan HEAD beserta master ke versi yang baru, dalam hal ini “#2”. Seperti pada “#1”, Git juga hanya merekam isi teks.txt yang baru dalam bentuk hash 40 karakter.

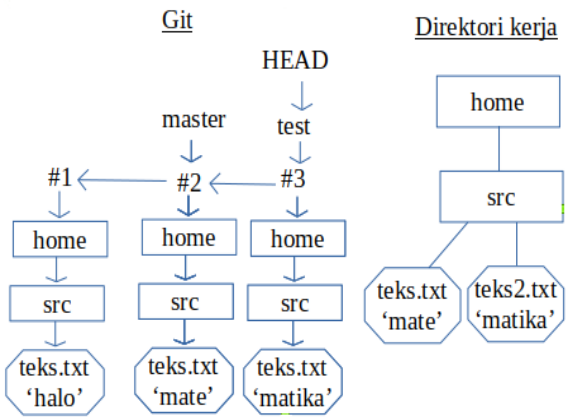
Dengan mengobservasi struktur graf yang terbentuk setelah melakukan operasi-operasi dasar pada Git, dapat ditarik properti-properti graf sebagai berikut. Setiap folder dan file pada direktori Git merupakan simpul yang isinya dicatat sebagai *hash* 40 karakter. Folder dan file tersebut dihubungkan dengan sisi yang berupa pointer pada setiap simpul yang mengarahkan ke alamat memori simpul yang bersisian dengannya. Dengan kata lain, representasi graf pada Git seperti representasi dengan senarai ketetanggaan.

Selain itu, dapat diobservasi bahwa setiap versi proyek merupakan upagraf tersendiri, sehingga setiap versi dapat terus tersimpan dengan aman. Apabila pengguna ingin mengembalikan direktori ke versi sebelumnya, Git tinggal mengarahkan pointer HEAD ke simpul *commit* versi sebelumnya.

Selanjutnya akan dibahas percabangan pada Git. Dengan membuat cabang baru, perubahan-perubahan yang dibuat pada cabang tidak akan mempengaruhi cabang utama atau *master*. Contoh penggunaan percabangan misalnya untuk menambah fitur baru pada program, namun masih belum yakin bahwa fitur ini akan berjalan lancar dan tidak merusak program utama. Nantinya, cabang dapat digabung (*merge*) kembali dengan cabang utama atau *master*.

Pengguna dapat membuat cabang baru pada Git dengan perintah ‘git branch <nama cabang>’. Selanjutnya, pengguna dapat berpindah ke cabang tersebut dengan perintah ‘git checkout < nama cabang>’. Sebenarnya, ketika membuat cabang baru, cabang tersebut hanyalah simpul baru yang pointernya menunjuk ke cabang leluhurnya (*ancestor*). Akan tetapi, perubahan pada file di direktori kerja akan dicatat pada branch baru tersebut saja.

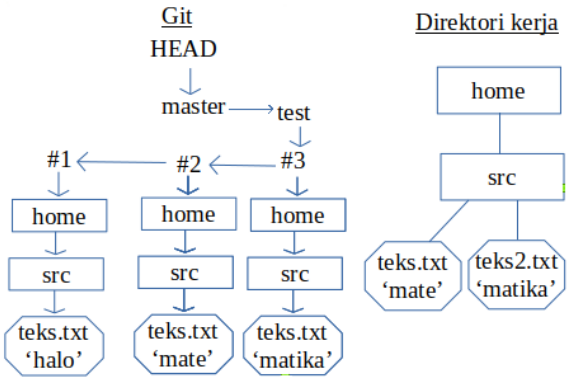
Berikut akan diberikan contoh percabangan beserta struktur grafnya. Cabang yang baru dibuat bernama *test*, yang berasal dari cabang *master*. Setelah berada di cabang ini, pengguna membuat file baru yang bernama *teks2.txt* yang berisi kata ‘matika’. Kemudian, pengguna melakukan *commit* “#3”.



Gambar 9: Direktori setelah membuat cabang baru *test*

Dapat dilihat bahwa ketika membuat cabang baru, simpul cabang masih berhubungan dengan cabang leluhurnya. Dengan demikian, file-file pada cabang baru yang beririsan dengan cabang leluhurnya tidak diperbanyak, tetapi masih tetap dapat diakses olehnya dengan sisi yang berupa pointer.

Bila pengguna melakukan penggabungan (*merge*) dengan master, maka pointer master akan menunjuk ke pointer cabang, sehingga *master* menjadi sama dengan cabangnya. Akan tetapi bila ditemukan konflik dengan cabang, misalnya bila ada file yang bernama sama namun isinya berbeda pada *master* dan cabangnya (karena isinya telah diubah di cabang), pengguna harus menyelesaikan konflik tersebut dahulu dengan melakukan penyuntingan secara manual.



Gambar 10: Direktori setelah dilakukan merge antara master dan test

Setelah melakukan operasi-operasi dasar seperti *add*, *commit*, *checkout*, *branch*, dan *merge* tersebut serta melihat representasinya struktur grafnya, pengguna dapat memahami bahwa pada dasarnya, Git adalah graf[3] dan operasi-operasi pada Git adalah untuk memanipulasi struktur graf tersebut, demi memudahkan mengelola proyek perangkat lunak.

V. Kesimpulan

Git adalah salah satu pengontrol versi yang dapat memudahkan pengelolaan proyek perangkat lunak. Sebagai pengontrol versi yang paling banyak digunakan, penguasaan Git merupakan salah satu kemampuan yang dibutuhkan dari seorang pengembang perangkat lunak. Pada dasarnya, Git adalah sebuah graf, dan operasi-operasi pada Git adalah untuk memanipulasi graf tersebut sesuai kebutuhan pengembangan proyek perangkat lunak yang menggunakannya. Oleh karenanya, pemahaman akan struktur graf pada Git yang mendalam akan membantu untuk menguasai penggunaan Git.

VI. Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada Dr. Ir. Rinaldi Munir dan Dra. Harlili, sebagai dosen mata kuliah Matematika Diskrit. Penulis juga mengucapkan terima kasih kepada seluruh penulis yang tulisannya dijadikan referensi untuk makalah ini. Terakhir, penulis berterima kasih kepada orang tua dan teman-teman atas dukungannya dalam menyelesaikan makalah ini.

Referensi

[1] Munir, Rinaldi. Matematika Diskrit. Bandung: Penerbit Informatika, 2006.
[2] https://git-scm.com/doc, diakses pada tanggal 9 Desember 2016, pukul 00.36 GMT+7.
[3] https://maryrosecook.com/blog/post/git-from-the-inside-out, diakses pada tanggal 7 Desember 2016, pukul 12.18 GMT+7.
[4] https://eagain.net/articles/git-for-computer-scientists, diakses pada tanggal 7 Desember 2016, pukul 12.38 GMT+7.

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, 9 Desember 2016

Devin Alvaro