UNIVERSITAS BRAWIJAYA

MODUL PELATIHAN

SEPTEMBER 2022

PELATIHAN ONLINE GIT & GITHUB

//TIM PENGABDIAN FILKOM 2022





Tim Pengabdian FILKOM 2022

Online Training Terkait Pemanfaatan GIT & GITHUB sebagai Version Control System (VCS) sebagai Alat Bantu dalam Pengembangan Perangkat Lunak



PROGRAM STUDI PENDIDIKAN TEKNOLOGI INFORMASI

JURUSAN SISTEM INFORMASI

FAKULTAS ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MODUL GIT & GITHUB

Tim pengabdian FILKOM 2022

Modul Pelatihan GIT & GITHUB

© Tim Pengabdian FILKOM 2022

Program Studi Pendidikan Teknologi Informasi

Fakultas Ilmu Komputer

Universitas Brawijaya Malang

Table of Contents

[Membuat VPC dengan private dan public subnet **Error! Bookmark not defined.**](#_Toc83039228)

[Membuat Subnet Tambahan **Error! Bookmark not defined.**](#_Toc83039229)

[Membuat VPC Security Group untuk Web Server Public **Error! Bookmark not defined.**](#_Toc83039230)

[Membuat VPC Security Group untuk Private Amazon RDS DB Instance **Error! Bookmark not defined.**](#_Toc83039231)

[Membuat DB Subnet Group **Error! Bookmark not defined.**](#_Toc83039232)

[Membuat DB Subnet Group **Error! Bookmark not defined.**](#_Toc83039233)

[Membuat EC2 INSTANCES 31](#_Toc83039234)

[Install Apache Web Server Dan Php 38](#_Toc83039235)

[Menghubungkan Apache Web Server Ke Rds Db Instance 41](#_Toc83039236)

Table of Figure

[Gambar 1 Arsitektur VPC pada AWS **Error! Bookmark not defined.**](#_Toc83039331)

[Gambar 2 Amazon RDS dan penggunaannya pada skema Database yang beragam **Error! Bookmark not defined.**](#_Toc83039332)

[Gambar 3 Pemilihan Engine pada RDS 27](#_Toc83039333)

[Gambar 4 Setting Default pada Menu Configure Advanced Settings 29](#_Toc83039334)

[Gambar 5 Menu Connect Database 29](#_Toc83039335)

[Gambar 6 Halaman DB Details 30](#_Toc83039336)

[Gambar 7 Halaman Resources untuk setting EC2 Instance 31](#_Toc83039337)

[Gambar 8 Amazon Machine Image (AMI) 32](#_Toc83039338)

[Gambar 9 Halaman Tipe Instance 32](#_Toc83039339)

[Gambar 10 Halaman Konfigurasi Detil Instance 33](#_Toc83039340)

[Gambar 11 Halaman Add Tags 34](#_Toc83039341)

[Gambar 12 Halaman Konfigurasi Keamanan 34](#_Toc83039342)

[Gambar 13 Halaman Instance Launch 35](#_Toc83039343)

[Gambar 14 Halaman Key Pairing 36](#_Toc83039344)

[Gambar 15 Halaman Launch RDS Instances 37](#_Toc83039345)

Chapter

1

Version Control System (VCS)

Version Control System (VCS) memungkinkan seorang coder/programmer/pengguna untuk dapat memantau dan mengatur perubahan yang terjadi pada artefak dalam proses pembangunan perangkat lunak/project

V

*ersion control* (juga dikenal sebagai *revisionv control*, *source control*, atau *source code management*) adalah sebuah mekanisme yang memiliki tujuan untuk mengelola perubahan pada komponen/artefak dalam proyek seperti program komputer, dokumen, situs web, atau informasi lainnya. *Version Control* merupakan unsur penting dari Manajemen Konfigurasi (*Configuration Management*) jika disitir dari panduan manajemen proyek dalam *Project Management Body of Knowledge* (PMBOK).

Implementasi *Version Control* secara manual dapat dilakukan dengan cara membubuhkan penanda khusus berupa angka atau kode huruf pada setiap perubahan yang terjadi dalam artefak/komponen proyek. Angka atau kode huruf yang merepresentasikan perubahan ini disebut juga dengan “nomer revisi”, “tingkat revisi”, atau hanya disebut “revisi”. Contoh: Ketika dokumen kebutuhan pengguna telah disusun dan disepakati oleh tiap pemilik kepentingan maka penamaan dari dokumen tersebut dapat dibubuhi dengan nomor versi yang dicantumkan setelah pendefinisian nama, sehingga nama dokumen menjadi: “Dokumen Kebutuhan Pengguna v1.0”. Namun jika terjadi perubahan kebutuhan yang dirasa dapat diakomodasi oleh tim pengembang, maka dokumen yang telah direvisi serta divalidasi oleh pihak terkait akan diberikan nama yaitu “Dokumen Kebutuhan Pengguna SRS v2.0”. Perbedaan versi dari v1.0 menjadi v2.0 di akhir nama dokumen menandakan terdapat perubahan pada dokumen “Dokumen Kebutuhan Pengguna”.

Selain digunakan dalam manajemen proyek sebagai penanda komponen yang telah berubah, *Version Control* juga digunakan untuk membantu pihak pengembang dalam melakukan pengambilan keputusan seperti mengkomparasi komponen yang sama pada versi aplikasi yang berbeda, memperbaiki komponen dalam aplikasi, serta menggabungkan komponen-komponen yang dirasa perlu untuk diintegrasi. Hal ini dapat dilakukan karena dua sifat integral dari suatu revisi/*revision* yaitu: 1) Revisi selalu memiliki *timestamp,* yaitu catatan waktu perubahan komponen, serta 2) Revisi memiliki penanda untuk mendeteksi pihak yang bertanggungjawab dalam melakukan perubahan pada suatu komponen.

*Version Control* *Systems* merupakan sistem yang berjenis *stand alone* yang berfungsi untuk mencatat perubahan komponen pada suatu kegiatan pengelolaan proyek, namun sistem seperti ini juga tertanam pada berbagai perangkat lunak seperti aplikasi pengolah kata, *spreadsheet*, serta *content management systems* pada halaman riwayat didalam Wikipedia. Keunggulan dari penggunaan *Version Control Systems* antara lain: 1) Memungkinkan seorang *editor* dokumen untuk dapat mengembalikan dokumen ke revisi sebelumnya, 2) Editor dapat melacak perubahan suntingan pada suatu dokumen sehingga dapat meminimalisir kesalahan sebelum dokumen disetujui, 3) Memperbaiki kesalahan yang terjadi, serta 4) Menghindari perubahan yang dilakukan oleh pihak yang tidak bertanggung jawab.

Terdapat 3 jenis VCS yaitu *Local Version Control Systems*, *Centralized Version Control Systems*, dan *Distributed Version Control Systems* (VCS terdistribusi). Berikut adalah karakteristik dari tiap jenis VCS: **1)** ***Local VCS (LVCS)*** yaitu versi dari VCS yang tersusun dari sebuah database yang secara sengaja dikembangkan pada komputer lokal dengan tujuan untuk dapat menyimpan, memonitor, dan mengontrol perubahan pada *file*. **2)** ***Centralized VCS (CVCS)***merupakan jenis VCS yang menggunakan arsitektur *client-server* untuk dapat melakukan perannya sebagai *version control*. *Centralized VCS* terbentuk dari satu server terpusat yang didalamnya berisi file serta informasi lain yang mengalami perubahan, serta *client* yang terhubung dengan *server* untuk dapat mengakses, memonitor, serta berkolaborasi untuk melakukan perubahan. **3)** **VCS Terdistribusi (DVCS)**, adalah versi VCS yang dibangun dengan arsitektur *client-server*yang mana *client* dapat melakukan mirroring/duplikasi pada file sebagaimana adanya sesuai dengan kondisi teraktual file pada sebuah server. DVCS memungkinkan terjadinya kolaborasi yang tidak akan terkendala ketika server mengalami gangguan.

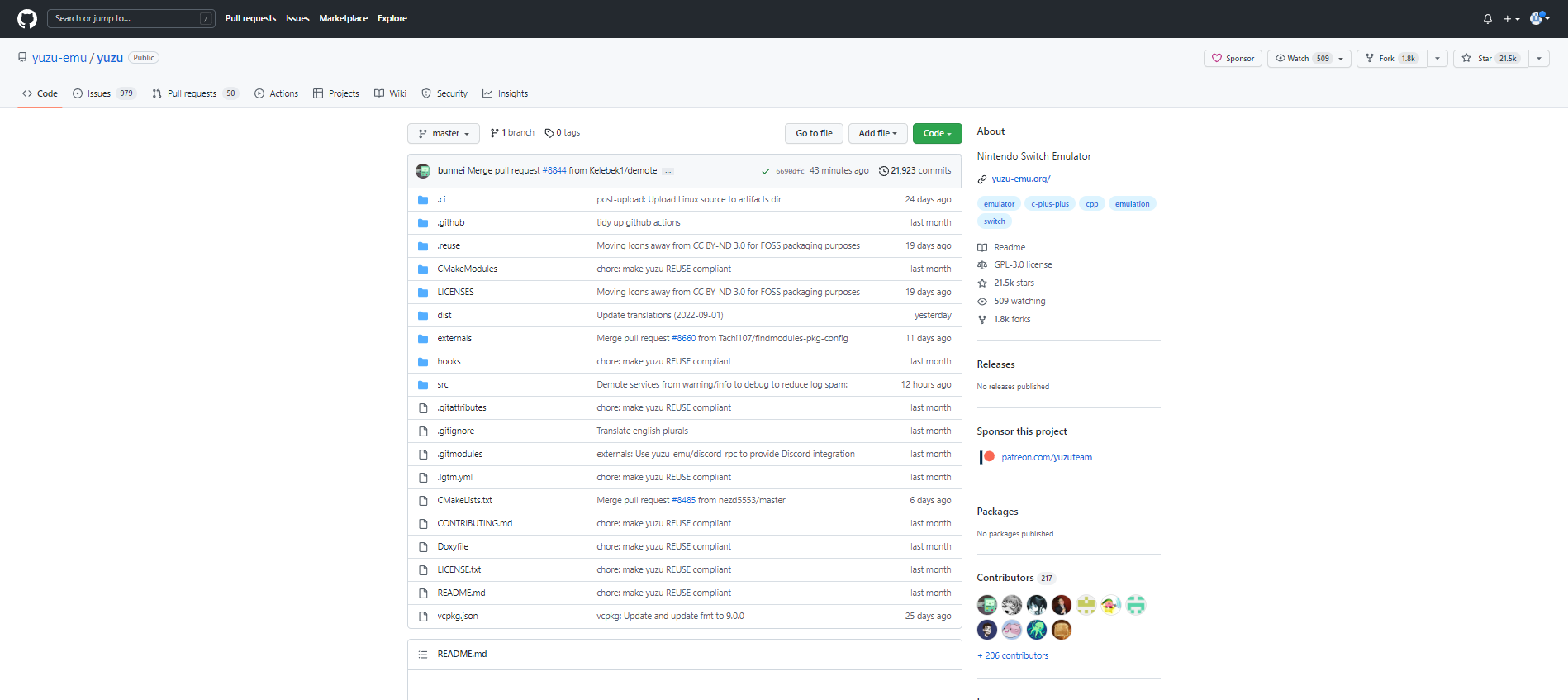
|  |  |  |
| --- | --- | --- |
|  |  |  |
| Gambar 1 LVCS | Gambar 2 CVCS | Gambar 3 DVCS |

Kompleksitas pembangunan perangkat lunak dapat terurai dengan pemanfaatan alat bantu yang tepat yang memungkinkan pihak pengembang perangkat lunak untuk dapat berkoordinasi dengan baik, dan secara efisien dapat mengontrol serta melacak perubahan komponen perangkat lunak.. Kapabilitas ini dapat dicapai dengan pemanfaatan alat bantu yang memiliki fitur untuk mengendalikan perubahan dari dokumen, *source code* atau informasi lain didalam lingkungan pembangunan perangkat lunak. Konsep *revision control* dalam Rekayasa Perangkat Lunak (RPL) didefinisikan sebagai segala jenis usaha untuk dapat melacak dan mengontrol perubahan pada suatu *source code*. Pengelolaan dokumentasi, *source code* serta file konfigurasi pada aktifitas pembangunan perangkat lunak dapat dijalankan dengan memanfaatkan sebuah perangkat lunak yang disebut dengan *revision control software*. *Revision Control Software* memungkinkan pihak pengembang perangkat lunak untuk dapat mengelola versi dokumen, program komputer, website serta kumpulan informasi lain, selain itu juga dapat menjadi *repository* dari luaran yang dihasilkan pada fase-fase dari proses pembangunan perangkat lunak. Tujuan penggunaan *Revision Control Software* adalah untuk mengefisiensikan aktifitas revisi yang membutuhkan usaha, waktu, dan biaya yang tidak sedikit. Terdapat beberapa pilihan *software revision control* yang sekarang ada dipasaran. Berikut adalah 10 alat bantu yang dapat digunakan untuk mengelola versi dari *source code* Perangkat Lunak:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a)** |  | GitHub | **f)** |  | Mercurial |
| **b)** |  | Git | **g)** |  | Monotone |
| **c)** |  | GitLab | **h)** |  | CVS |
| **d)** |  | Team Foundation Server (TFS) | **i)** |  | Apache Subversion (SVN) |
| **e)** |  | Bitbucket | **j)** | GNU Bazaar - Wikipedia | Bazaar |

**GITHUB**

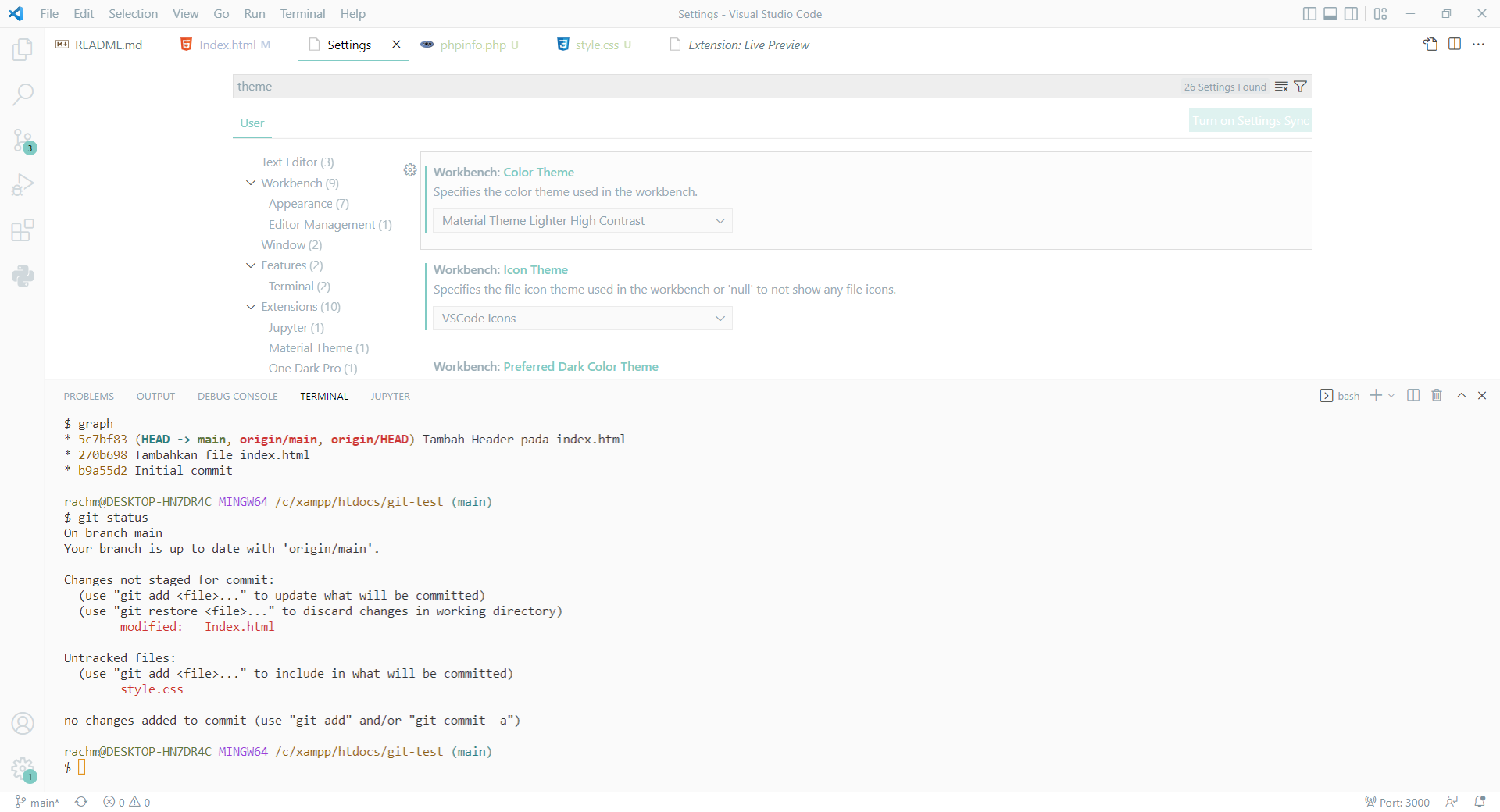
GITHUB merupakan sebuah platform berbasis komputer awan dengan layanan utama yaitu layanan hostingdan *version control* berbasis Git, dengan memanfaatkan Github akan memungkinkan pihak pengembang perangkat lunak dapat menjalankan aktifitas *version control* dan juga *issues tracking* secara *online*. Melalui Github aktifitas-aktifitas pengembangan seperti pelacakan bug, permintaan fitur, manajemen tugas, serta wiki atau user manual dari perangkat lunak dapat dieksekusi secara terstruktur dan terkolaborasi. Dinyatakan dalam laporan tahunan per April 2017 bahwa GITHUB telah mengelola lebih dari 20 juta pengguna dan lebih dari 57 juta repositori.



Gambar 4 Tampilan antarmuka GITHUB

**GIT**

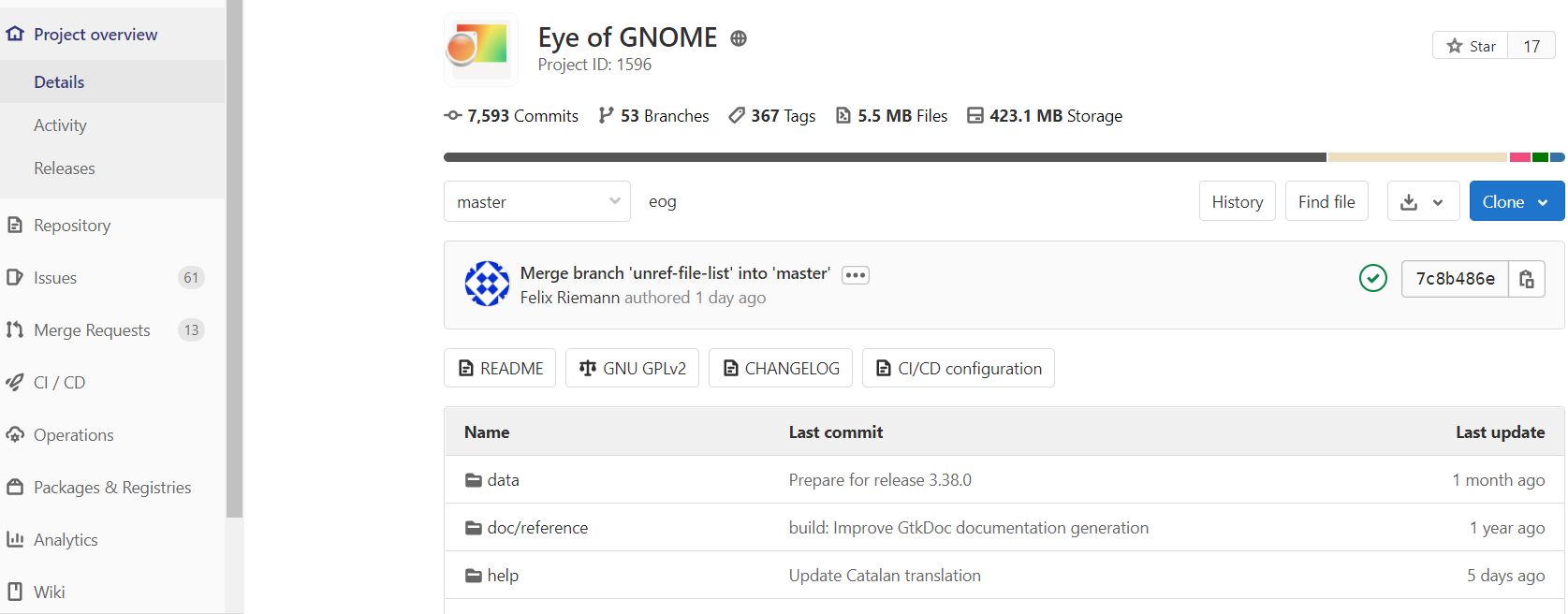
GIT adalah perangkat lunak yang tertanam dalam workstation atau komputer dari pengembang perangkat lunak dengan fungsi untuk memudahkan pengembang dalam menjalankan aktifitas *version control* dari kode perangkat lunak. Melalui GIT pihak pengembang perangkat lunak dapat mensinkronisasi kode yang telah disusun dalam komputer/workstation pribadi kedalam repositori GITHUB sehingga memungkinkan aktifitas kolaborasi untuk dilaksanakan. Analogi dari sinkronisasi antara GIT dan GITHUB menyerupai penggunaan perangkat lunak pengolah kata yang tertanam pada komputer pribadi seperti Microsoft Word 2016 (MS Word 2016) dengan Google Docs. Konten di dalam MS Word 2016 dapat diduplikasi pada Google Docs sehingga pengguna dokumen dapat saling berkolaborasi untuk menambah, mengubah, dan menghapus konten secara online.



Gambar 5 Tampilan GitBash pada VS Code

**GITLAB**

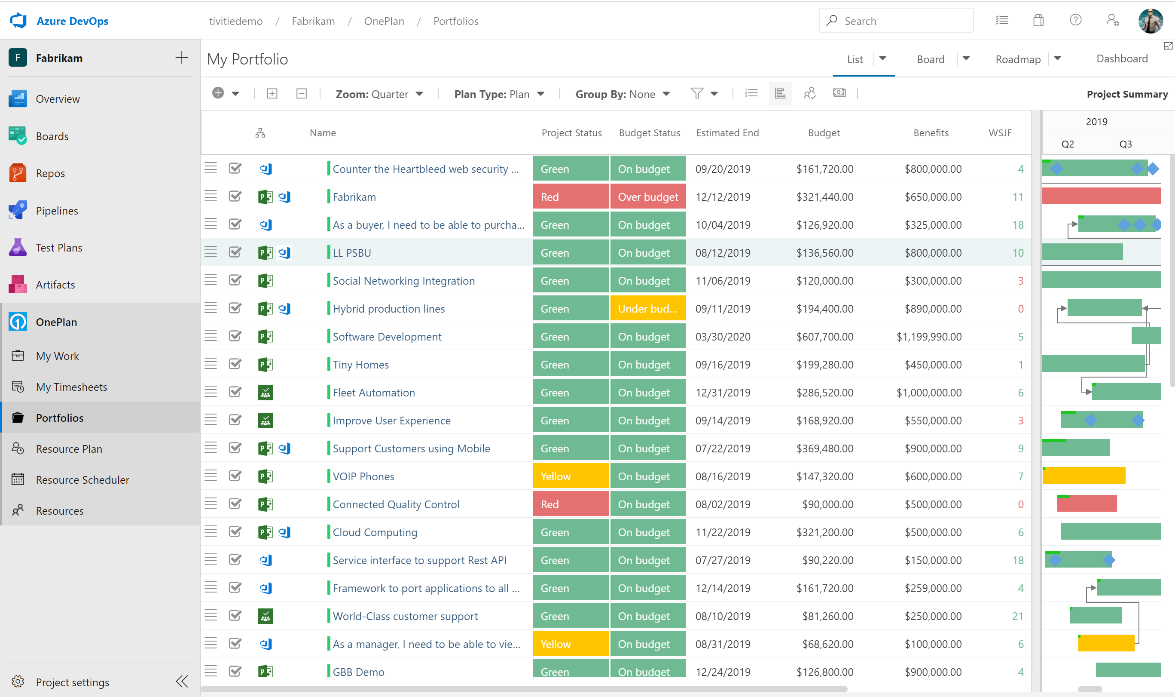
GITLAB menawarkan kemampuan bagi pengembang perangkat lunak untuk dapat secara kolaboratif merencanakan, membangun, mengamankan, dan *deploy*ing perangkat lunak sesuai dengan prinsip pengembangan DevOps. Selain *revision control* layanan hosting pada GitLab dilakukan pada tempat penyimpanan yang tertanam pada *cloud*. Komponen yang dapat di-hosting mencakup wiki, *issue tracking*, serta proses pada metode pengembangan perangkat lunak dengan prinsip CI/CD. GitLab, seperti GitHub, juga menawarkan produk Halaman GitLab gratis untuk menghosting halaman web statis dengan dukungan protokol HTTPS sejak versi 12.1.



Gambar 6 Tampilan Project pada Gitlab

**TEAM FOUNDATION SERVER (TSF)**

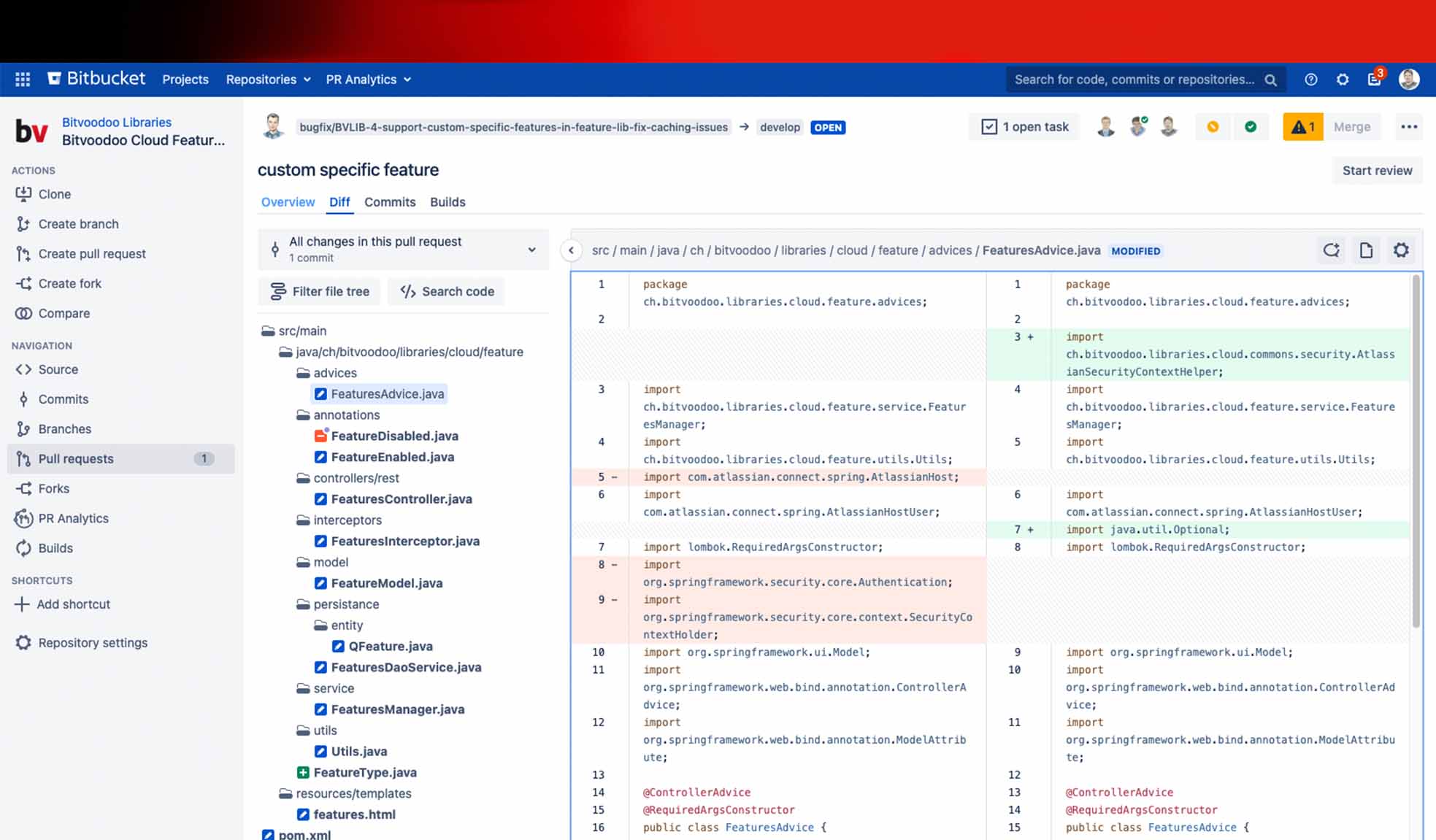
Sekarang disebut dengan Azure DevOps Server, yaitu produk yang dikembangkan oleh Microsoft sebagai strategi untuk bersaing pada pasar *software* pendukung kegiatan pengembangan perangkat lunak yang menyediakan layanan utama yaitu *version control*. Fungsi yang tertanam pada Azure DevOps Server antara lain *version control*, fitur untuk mempermudah pelaporan, manajemen kebutuhan, manajemen proyek, manajemen rilis aplikasi, serta memungkinkan perangkat lunak untuk dikembangkan selaras dengan siklus hidup aplikasi serta praktek-praktek terbaik DevOps.



Gambar 7 Tampilan Project pada Azure DevOps

**BITBUCKET**

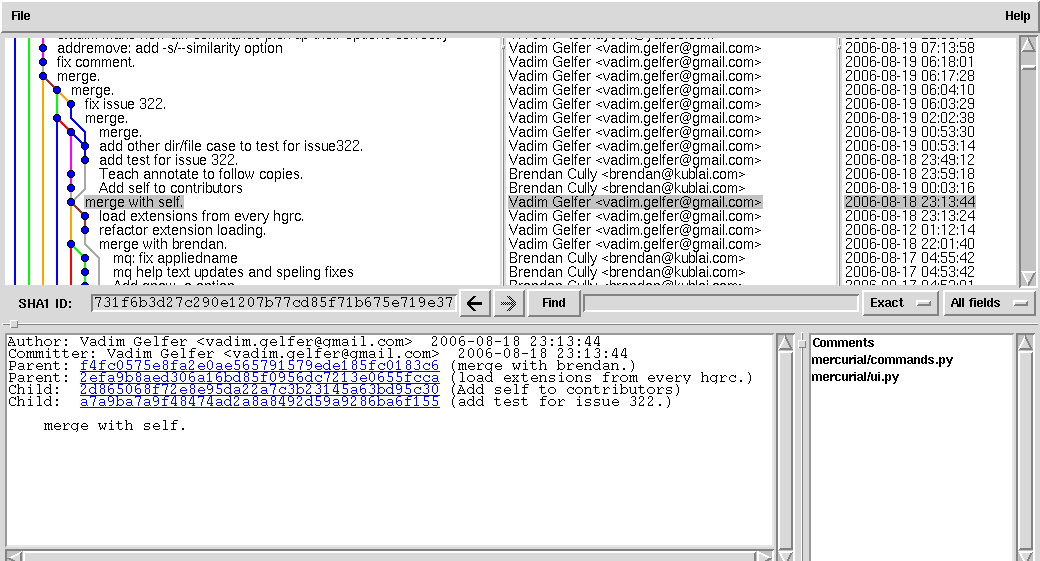
Bitbucket adalah sebuah platform layanan hosting repositori untuk *source code* berbasis GIT yang dimiliki oleh Atlassian. Bitbucket memberikan dua opsi paket repositori yaitu paket berbayar dan akun repositori gratis.



Gambar 8 Tampilan Antarmuka BitBucket

**MERCURIAL**

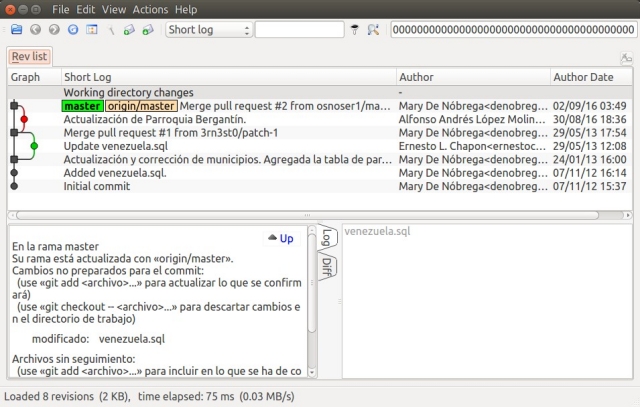
Mercurial adalah *tools* pengembangan perangkat lunak untuk membantu proses *revision control* secara terdistribusi sehingga memungkinkan kolaborasi oleh tim pengembang melalui perangkat lunak Mercurial. Perangkat lunak ini dapat dijalankan pada Microsoft Windows serta sistem turunan dari Unix, seperti FreeBSD, macOS, dan Linux. Tujuan utama dari Mercurial adalah untuk mengelola aktifitas pengembangan sehingga menghasilkan kinerja dan skalabilitas tingkat tinggi, memungkinkan kolaborasi, dan menyediakan fungsi percabangan dan penggabungan yang kompleks.



Gambar 9 Tampilan Antarmuka Mercurial

**MONOTONE**

Monotone adalah *tools* pengembangan perangkat lunak untuk *revision control* yang dijalankan dalam lingkungan sistem yang terdistribusi. Fitur Monotone meliputi pelacakan revisi kedalam sebuah file, mengelompokkan kumpulan revisi ke dalam perubahan-perubahan, dan melacak riwayat perubahan di seluruh proses pengembangan perangkat lunak. Fokus pada penggunaan Monotone pada sebuah proyek perangkat lunak dititikberatkan pada integritas antar komponen dibanding kecepatan serta kinerja. Monotone dirancang untuk mendukung operasi pengembangan perangkat lunak secara terdistribusi, dan menggunakan kriptografi untuk melacak revisi dan autentikasi dari aktifitas yang dilakukan pengguna. Pada bulan Januari 2008 ditemukan beberapa kekurangan dari Monotone yaitu: a) Pengguna tidak dapat commit melalui proxy karena protokol yang didukung adalah protokol non-http; b) Isu terkait performa perangkat lunak untuk operasi tertentu terutama pada operasi pull.



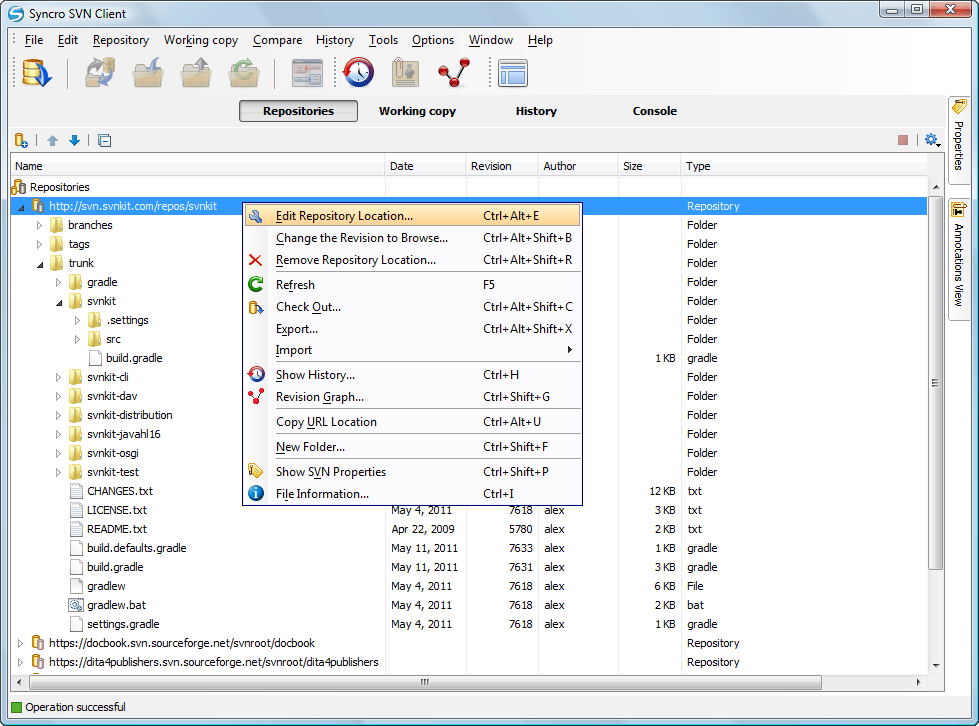
Gambar 10 Tampilan Antarmuka Monotone

**CVS**

CVS dikembangkan sebagai *front-end* dari sistem *revision control* yang mendahuluinya yang disebut dengan *Revision Control Systems* (RCS), sistem ini dikembangkan tidak untuk mengelola keseluruhan proyek namun lebih berfokus pada pengelolaan file individu yang ada didalamnya. Dalam pengembangan selanjutnya CVS memperluas fungsional dari RCS dengan menambahkan fitur seperti pelacakan perubahan, dan model aplikasi berbasis client-server.

**APACHE SUBVERSION**

Apache Subversion (sering disingkat SVN, serta memiliki inisial *command line* *svn*) adalah perangkat lunak dan sistem untuk mendukung *revision control* yang didistribusikan di bawah Lisensi Apache (open source). Pengembang perangkat lunak dapat menggunakan Subversion untuk mengatur pemeliharaan versi file yang sedang dalam pengerjaan dan versi historis dari *source code*, halaman web, dan dokumentasi.



Gambar 11 Tampilan Apache Subversion Client

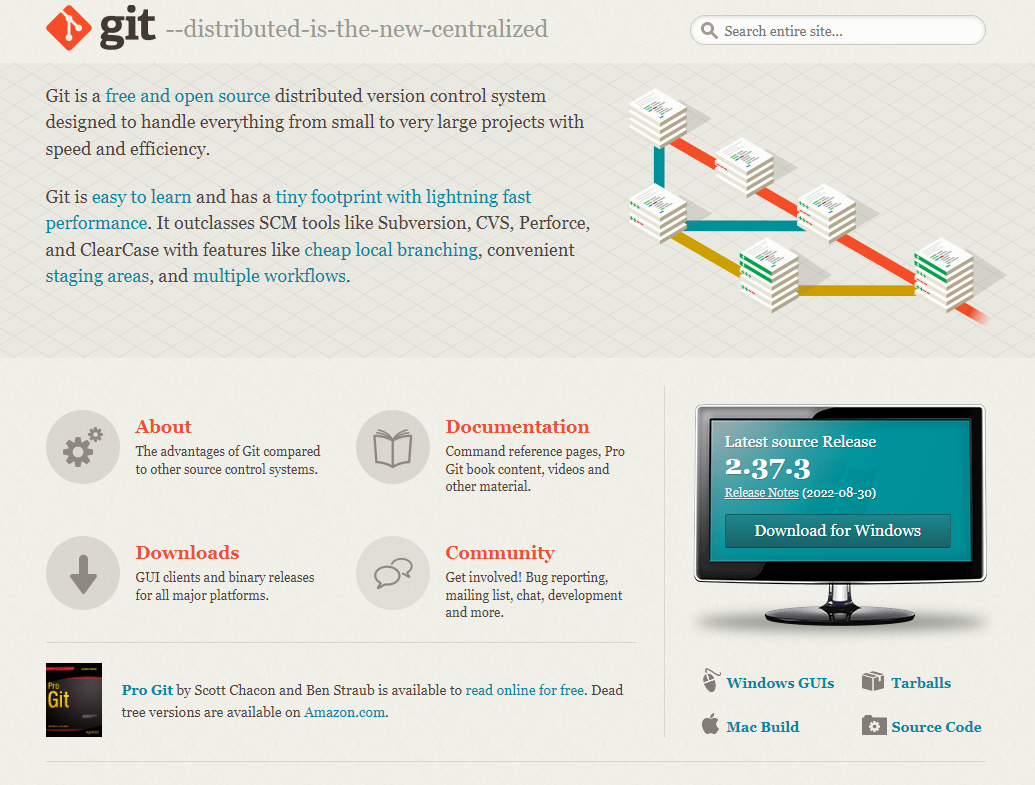
**BAZAAR**

GNU Bazaar (sebelumnya disebut dengan Bazaar-NG, dengan *command line tools* berinisial *bzr*) adalah sistem untuk mengelola *revision control* secara terdistribusi melalui struktur aplikasi berbasis klien-server yang disponsori oleh Canonical.

# INSTALASI GIT

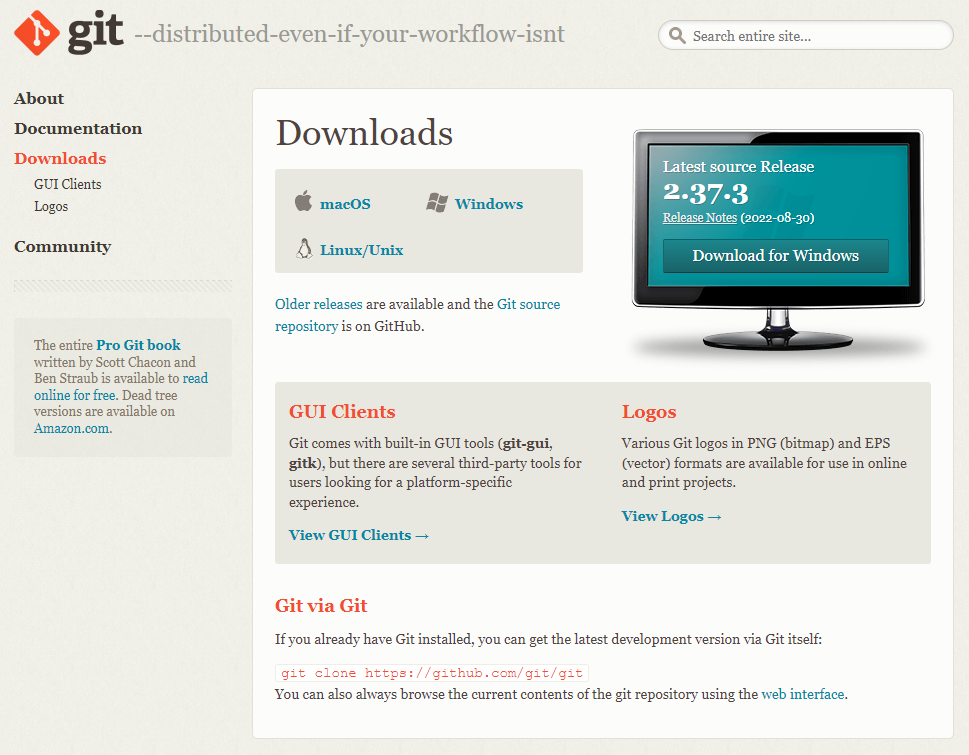
Instalasi GIT dilaksanakan dengan melakukan proses unduh file GIT dari situs <https://git-scm.com/>, kemudian dijalankan dengan melakukan langkah-langkah sebagai berikut:

1. Buka situs GIT pada <https://git-scm.com/>, kemudian pilih Downloads untuk membuka laman unduh GIT



Gambar 12 Website GIT

1. Download file instalasi GIT sesuai dengan OS yang digunakan, kemudian pilih file installer pada **Standalone Installer** dengan konfigurasi Windows sesuai dengan arsitektur OS (32-bit atau 64-bit). Tips: perintah dxdiag dapat digunakan untuk melihat arsitektur OS.

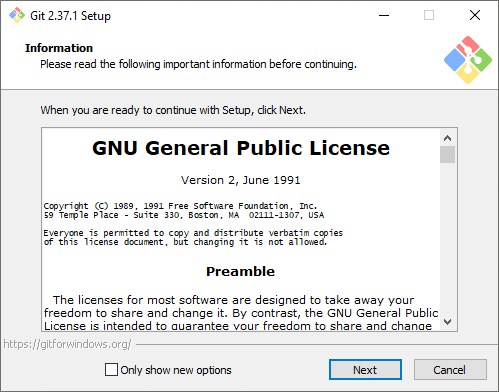


Gambar 13 Laman unduh file instalasi GIT

1. Setelah file installer GIT terunduh, maka buka file installer tersebut untuk memulai instalasi GIT.

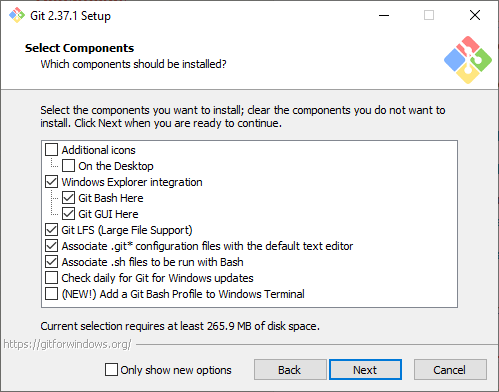


1. Pada tampilan setup klik Next untuk melanjutkan ke menu instalasi berikutnya.



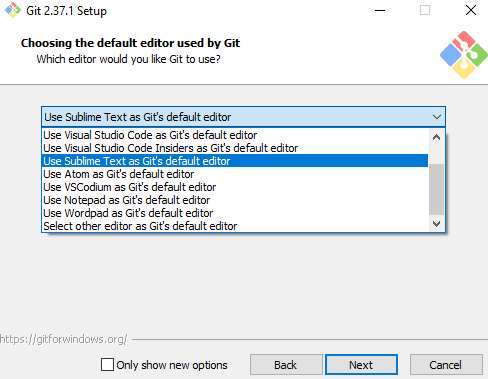
Gambar 14 Laman unduh file instalasi GIT

1. Pada tampilan pilihan komponen yang akan *diinstall*, kita dapat membubuhkan tanda centang untuk menambahkan komponen GIT seperti menambahkan icon GIT di Desktop, atau menambahkan profil Git Bash pada terminal di Windows. Klik Next untuk melanjutkan proses instalasi.



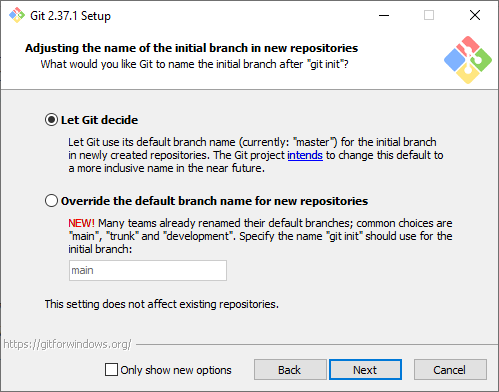
Gambar 15 Konfigurasi pada Instalasi GIT

1. Selanjutnya akan muncul tampilan untuk memilih *default text editor* yang akan kita gunakan dalam GIT. Klik Next untuk melanjutkan instalasi



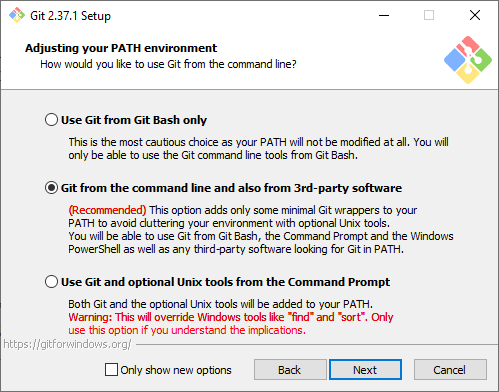
Gambar 16 Pilihan Text Editor pada Instalasi GIT

1. Tampilan selanjutnya adalah pilihan untuk menentaukan nama default yang akan diberikan oleh GIT pada *branch* didalam *repository*, pilihan default adalah **Let Git decide** yang artinya kita akan memberikan nama “master” untuk *branch* dalam repo. Klik Next untuk melanjutkan instalasi.



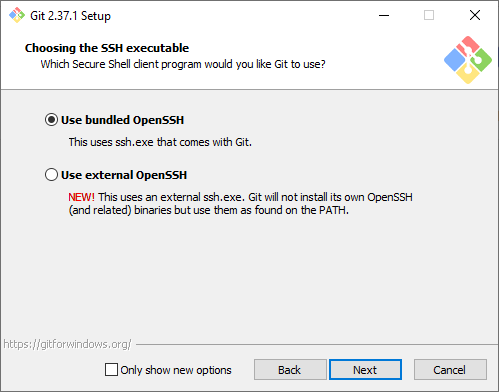
Gambar 17 Pilihan Nama *default branch* pada Instalasi GIT

1. Selanjutnya akan muncul tampilan untuk memilih *interface* dalam menjalankan GIT, pilih **Git from the command line also from 3rd-party software** untuk menjalankan GIT melalui *command line* (Git Bash) dan juga pada perangkat lunak tambahan (jika telah diinstalasi). Klik Next untuk melanjutkan instalasi



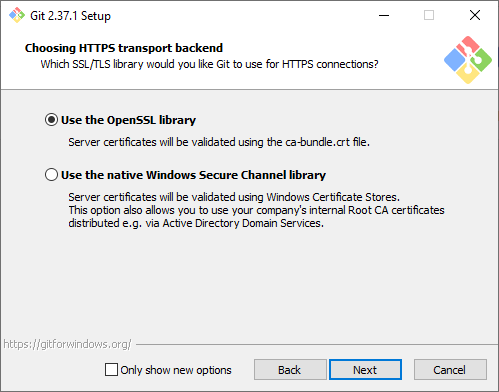
Gambar 18 Pilihan Nama *default branch* spada Instalasi GIT

1. Berikutnya pilih **Use bundled OpenSSH** untuk memilih *protocol* administrasi OpenSSH . Klik Next untuk melanjutkan instalasi



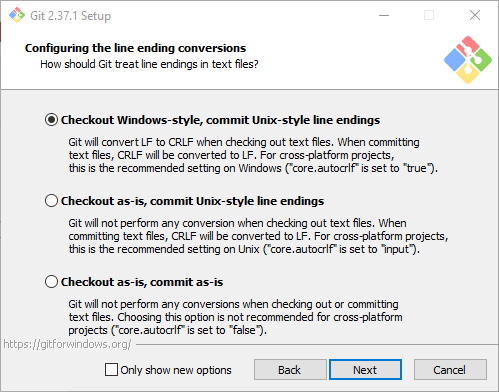
Gambar 19 Pilihan protocol komunikasi

1. Selanjutnya akan diberikan pilihan pada protocol keamanan komunikasi yang akan digunakan. Pilih **Use the OpenSSL library** untuk memilih *protocol* komunikasi OpenSSL. Klik Next untuk melanjutkan instalasi



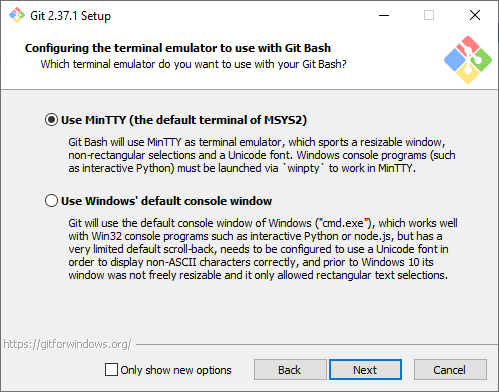
Gambar 20 Pilihan protocol keamanan komunikasi

1. Pada pilihan berikutnya pilih **Checkout Windows-style, commit Unix-style line endings**. Klik Next untuk melanjutkan instalasi



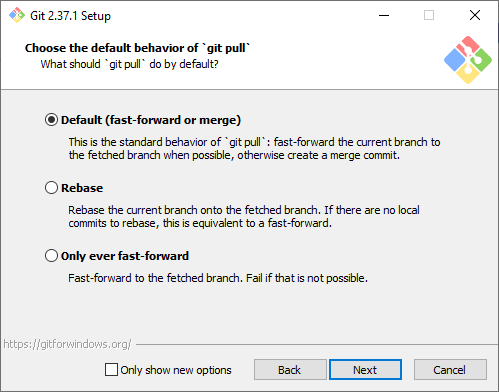
Gambar 21 Pilihan format CLRF-LF

1. Pilih **Use MinTTY (default terminal of MSYS2)**. Klik Next untuk melanjutkan instalasi



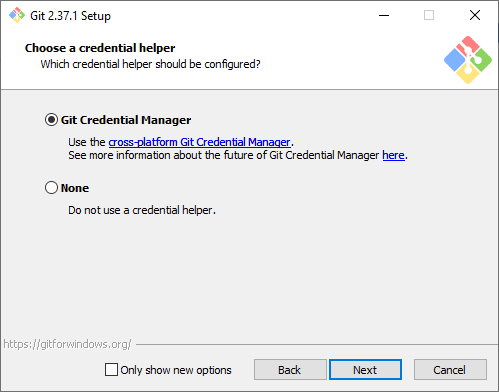
Gambar 22 Pilihan default terminal

1. Pilih **Default (fast-forward or merge)** untuk mengatur konfigurasi git pull. Klik Next untuk melanjutkan instalasi



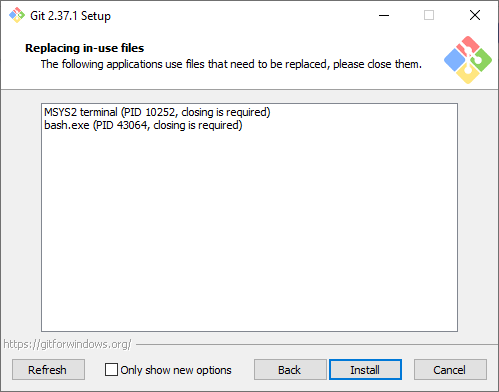
Gambar 23 Pilihan protocol keamanan komunikasi

1. Pilih **Git Credential Manager** untuk mengatur konfigurasi FAQ dan dokumentasi GIT. Klik Next untuk melanjutkan instalasi



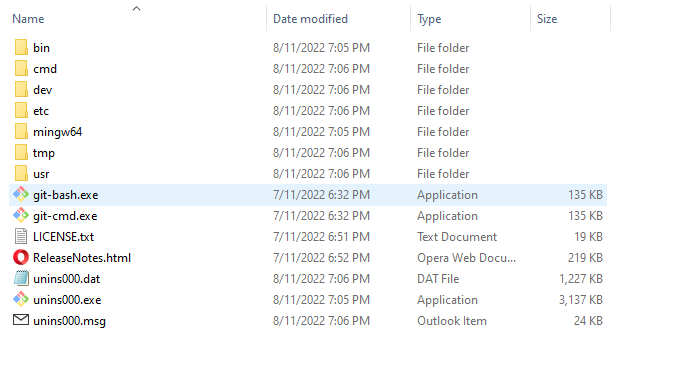
Gambar 24 Pilihan protocol keamanan komunikasi

1. Pilih **Enable file system caching** klik next, dilanjutkan dengan tidak memilih dua opsi konfigurasi eksperimental GIT. Klik Next untuk melanjutkan instalasi
2. Klik Install



Gambar 25 Pilihan protocol keamanan komunikasi

1. GIT akan terinstall pada C:\Program Files\Git. Klik pada Git-cmd.exe atau Git-bash.exe untuk menjalankan GIT.

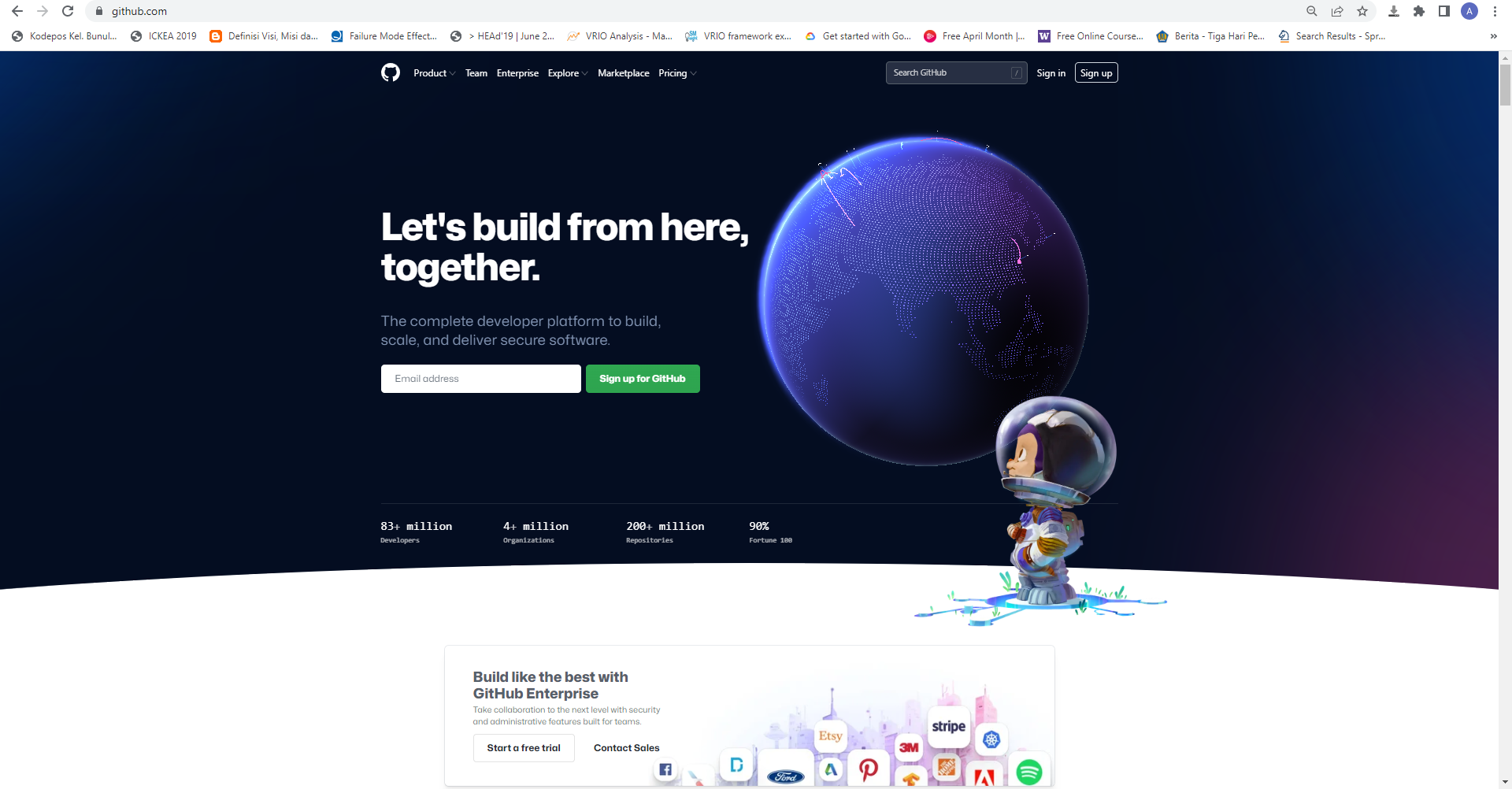


# MEMBUAT AKUN GITHUB

Layanan *version control* pada GITHUB dapat digunakan jika pengguna telah tergabung/telah regirstrasi pada website GITHUB. Website GITHUB dapat diakses pada tautan: github.com.

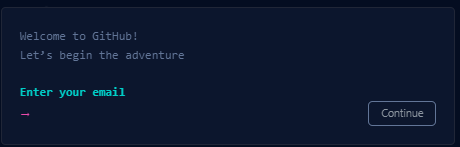
Proses registrasi di GITHUB dijalankan dengan melakukan langkah sebagai berikut:

1. Buka halaman website GITHUB pada: [github.com](https://console.aws.amazon.com/vpc/)



Gambar 26 Halaman website GITHUB

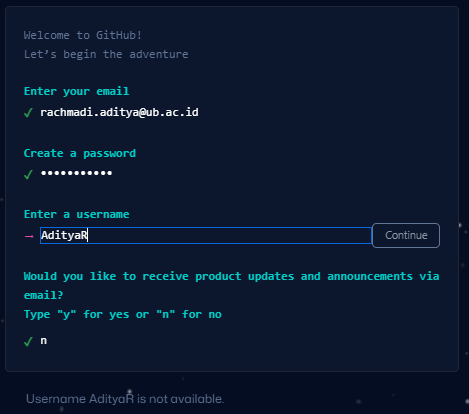
1. Pilih Sign up jika melakukan registrasi pada GITHUB dan Sign in jika telah memiliki akun pada GITHUB.
2. Masukkan email pada textbox yang tersedia kemudian tekan tombol Continue



1. Gambar 27 Halaman Sign up GITHUB
2. Masukkan password dan tekan tombol Continue 🡪 masukkan username yang akan digunakan pada GITHUB 🡪 Create Account

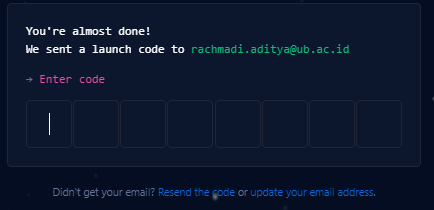


Gambar 28 Halaman Username & Password GITHUB



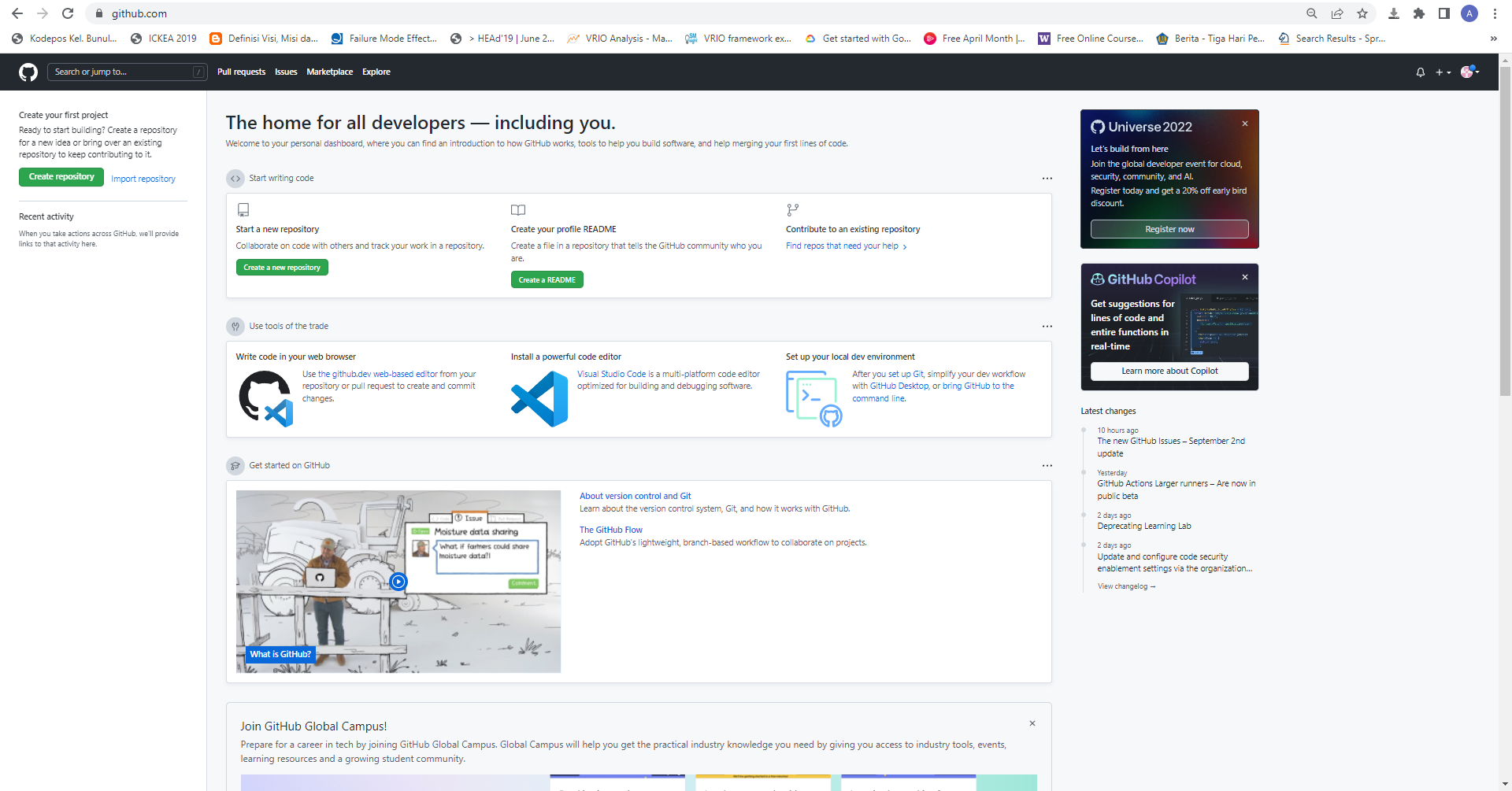
Gambar 29 Username pada GITHUB

1. Masukkan kode yang telah terkirim ke email



Gambar 30 Username pada GITHUB

1. Masuk ke halaman awal dari GITHUB dan siap untuk mengakses layanan pada GITHUB:



Gambar 31 Halaman utama GITHUB

Chapter

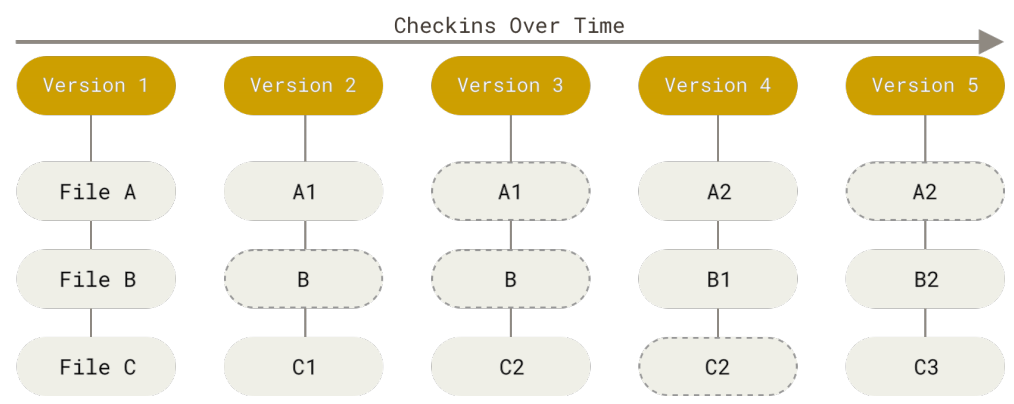
2

GIT

GIT merupakan sebuah Version Control System terdistribusi untuk mengelola perubahan file dalam folder (repository/repo)

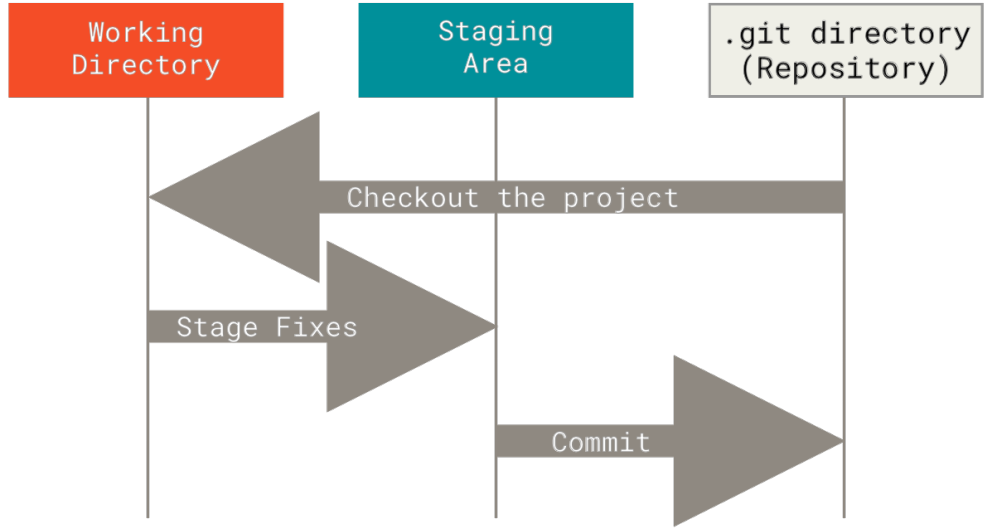
A

pa itu GIT? GIT merupakan alat bantu yang tertanam dalam sebuah komputer lokal untuk menangani aktifitas *version control.* Melalui konsep *Snapshots* GIT melakukan proses pengelolaan data dengan cara yang berbeda dengan VCS lain seperti CVS, Subversion, dan Bazaar. S*napshots* mengatur informasi didalam *repository* sebagai sebuah potret yang menggambarkan miniatur dari sebuah file. Pada saat GIT melakukan peyimpanan file yang berubah pada sebuah proyek, GIT akan berperan untuk memotret seluruh file yang ada dan selanjutnya membuat referensi dari file tersebut serta menyimpannya kedalam folder yang disebut *repository*/repo. Jika tidak ada perubahan pada sebuah file maka GIT tidak akan menyimpan file tersebut, namun hanya menyimpan tautan yang menunjuk ke salinan dari file. Berikut merupakan ilustrasi dari konsep *snapshots* pada GIT:



Gambar 32 Snapshots pada GIT

Implementasi konsep *snapshots* pada GIT direpresentasikan melalui 3 kondisi dari sebuah file yaitu *modified*, *staged*, dan *committed*. Status *Modified* memiliki arti bahwa perubahan dalam sebuah file belum dimasukkan dalam database. Status *Staged* memiliki arti bahwa file yang berubah telah diberikan tanda dan siap untuk dibuat duplikasinya kedalam *snapshots*. Status *Commit* menandakan bahwa *snapshots* dari data telah dibuat dan tautan serta data asli telah tersimpan kedalam database lokal. Berdasarkan 3 kondisi file yang berubah maka dapat ditarik kesimpulan bahwa struktur dan cara kerja didalam GIT juga terbagi menjadi 3 (tiga) area yaitu *Working Tree*/*Directory*, *Staging Area*, dan *Git directory*/*Repository*. Area **working tree** menandakan tempat (folder) yang telah diinisialisasi oleh GIT sebagai wadah untuk menyimpan sebuah file beserta perubahannya, **staging area** adalah tempat untuk menyimpan informasi dan data yang telah ditandai oleh GIT untuk diduplikasi dan dibuat suatu tautan yang disebut juga sebagai *snapshots*. **Git directory** merupakan tempat penyimpanan metadata dan database object yang mencatat perubahan pada file/konten. Bagian ini merupakan bagian terpenting dari GIT karena proses cloning antar repository dilakukan pada file yang ada didalam area ini.



Gambar 33 Area pada GIT

Berdasarkan penjelasan tentang kondisi dari perubahan pada sebuah file serta identifikasi area dalam cara kerja GIT maka berikut ini dijabarkan workflow dari GIT dalam langkah sebagai berikut:

1. Menginisiasi folder yang akan menjadi working tree dengan perintah Git init
2. File dimodifikasi pada area working tree. File yang telah dirubah dan berada pada area ini memiliki status *modified*
3. Memilih untuk menambah file kedalam *staging area,* file ini merupakan file yang kemudian akan dilakukan *commit*. Menambahkan file ke staging area dilakukan dengan perintah Git add *nama\_file..* File yang berada pada area ini berstatus *staged.*
4. *Commit* pada file yang telah ditambahkan kedalam *staging area* sehingga *snapshot* dapat tersimpan secara permanen di *repository*/repo. File pada langkah ini berstatus *committed*

Sebelum bekerja dengan GIT maka langkah awal yang harus dijalankan adalah menginisiasi username dan alamat email, langkah ini harus dijalankan karena informasi ini digunakan pada setiap langkah *commit*. Perintah yang dapat digunakan untuk menginisiasi username dan alamat email adalah:

$ Git Config –global user.name “*nama\_user*”

$ Git Config –global user.email *alamat\_email*

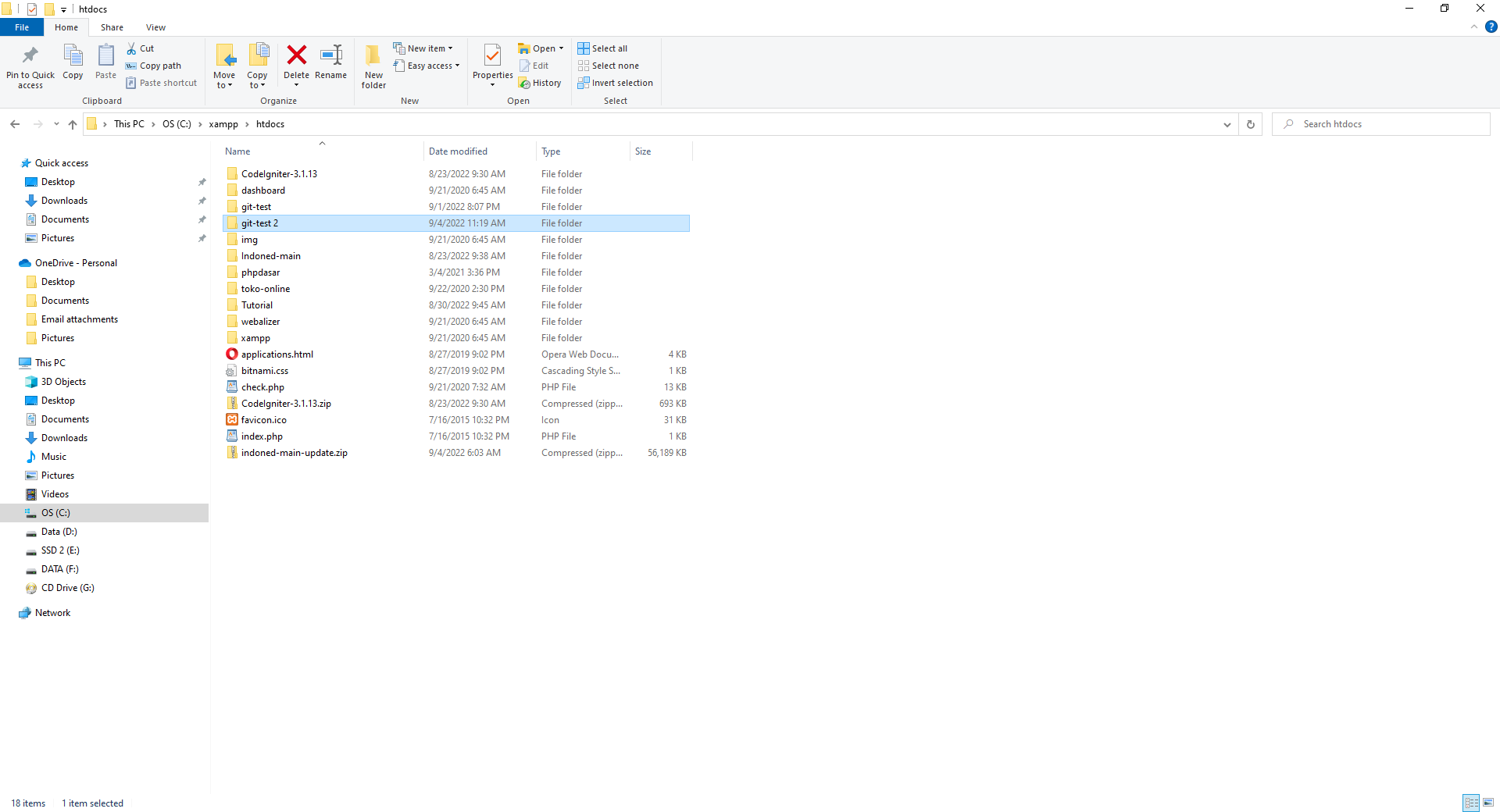
$ Git Config –global user.name “Aditya R.”

$ Git Config –global user.email rachmadi.aditya@ub.ac.id

# Bekerja dengan GIT

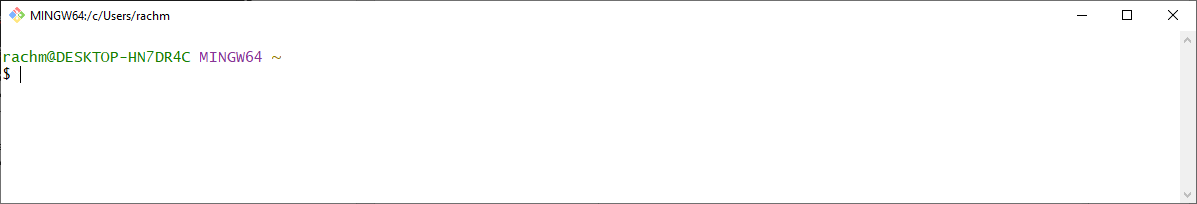
Langkah yang dijalankan untuk dapat bekerja menggunakan GITHUB dijabarkan dengan langkah sebagai berikut:

1. Sebelum memulai, buatlah sebuah folder yang akan menjadi area kerja atau disebut dengan *repository* pada GIT. Pada contoh folder diletakkan di C://xampp\htdocs\’git-test 2’



Gambar 34 Folder git-test 2 sebagai area kerja

1. Buka Git Bash sebagai terminal untuk memasukkan perintah GIT



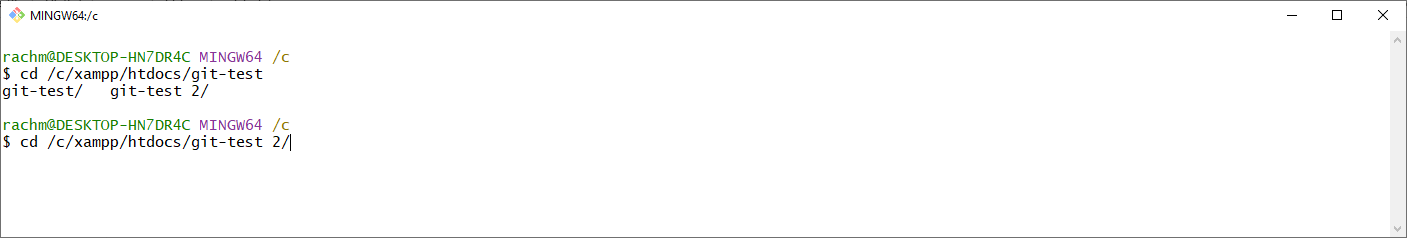
Gambar 35 Tampilan Git Bash

1. Inisialisasi username dan email pada Git bash. Apabila username dan email telah berhasil diinisialisasi gunakan perintah **git config –list** untuk melihat nama dan email yang telah diinputkan.



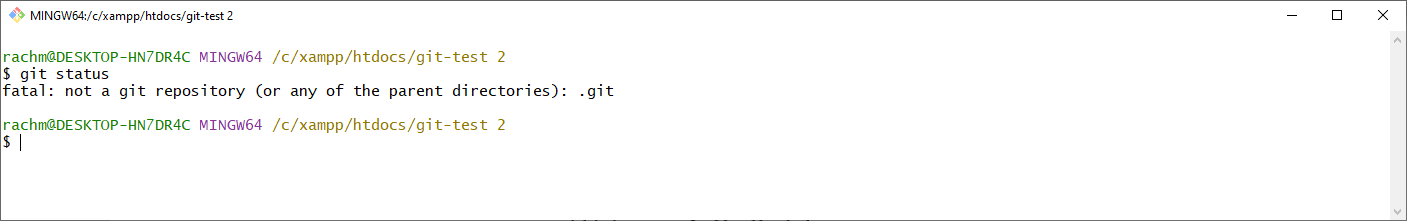
Gambar 36 git config --list

1. Berikan perintah pada Git hub untuk dapat mengakses folder yang menjadi area kerja, hal ini dapat dijalankan dengan menggunakan 1) perintah seperti gambar dibawah ini atau 2) buka folder yang diset menjadi area kerja 🡪 klik kanan 🡪 Git Bash here.



Gambar 37 Mengarahkan Git bash pada area kerja

1. Folder yang menjadi area kerja belum dilakukan inisialisasi sehingga tidak terdeteksi oleh GIT sebagai sebuah *repository* (dicek dengan perintah **git status**).



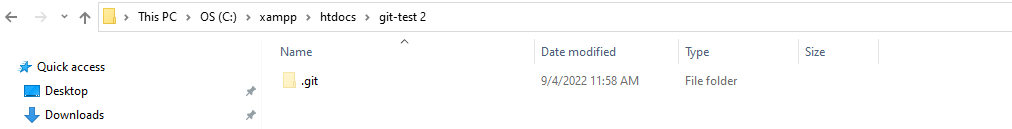
Gambar 38 Folder belum menjadi *working tree*

1. Inisialisasi folder yang menjadi area kerja sehingga menjadi *working tree* atau repository dengan perintah **git init**



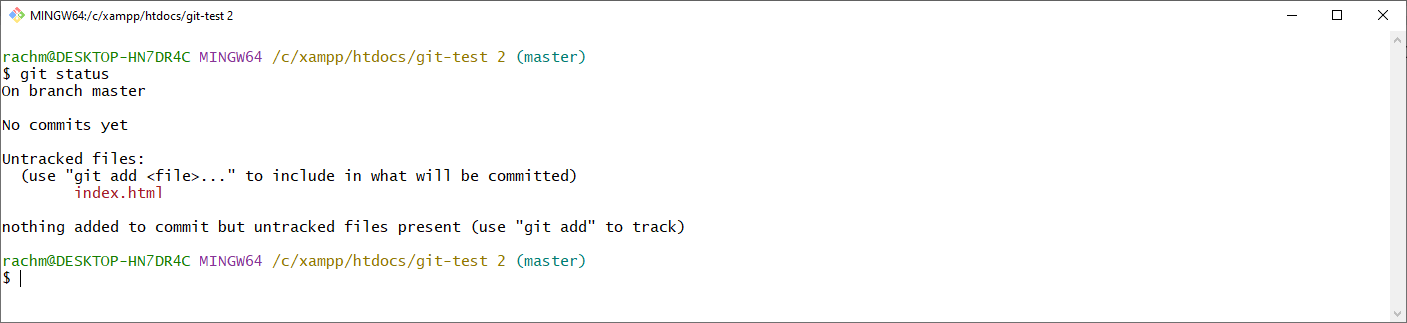
Gambar 39 Folder telah menjadi *working tree*

1. Apabila dipantau pada *file explorer* maka *folder* yang telah dinisialisasi sebagai repo akan memiliki *hidden folder* dengan nama .git. Hal ini menandakan bahwa folder ini telah menjadi *repository* dan GIT akan memantau tiap file yang ditanam dalam folder tersebut. Untuk mengembalikan menjadi folder biasa cukup hapus *hidden file* .git.



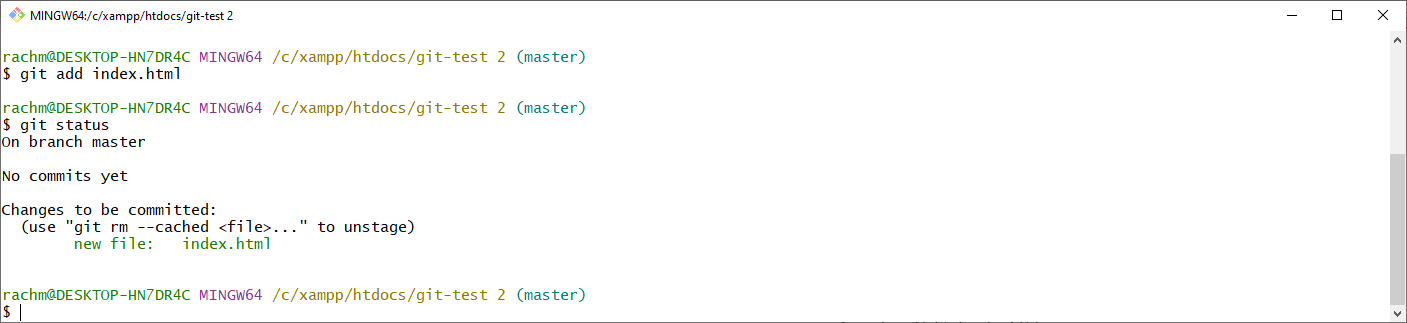
Gambar 40 Folder pada file explorer

1. Tambahkan file pada *folder*, kemudian masukkan perintah git status pada Git bash maka muncul gambar seperti dibawah ini (pada contoh ditambahkan file html dengan nama index.html), hal ini menandakan bahwa file belum masuk ke *staging area*



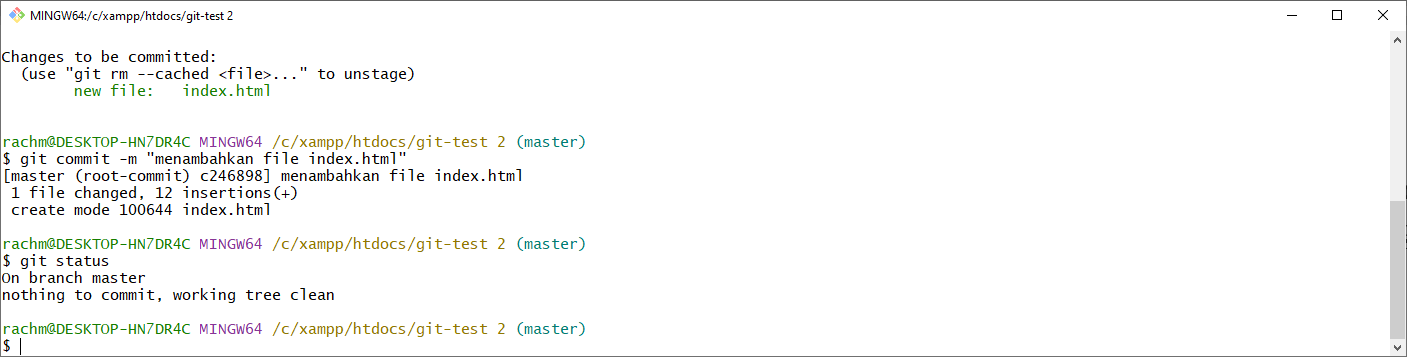
Gambar 41 Untracked File

1. Tambahkan file kedalam *staging area* dengan perintah **git add *­nama\_file*** , karena nama file pada contoh adalah index.html maka perintah yang dituliskan adalah **git add index.html**. **Tips**: apabila terdapat banyak file yang akan ditambah maka gunakan perintah **git add .** untuk menambahkan seluruh file ke *staging area*



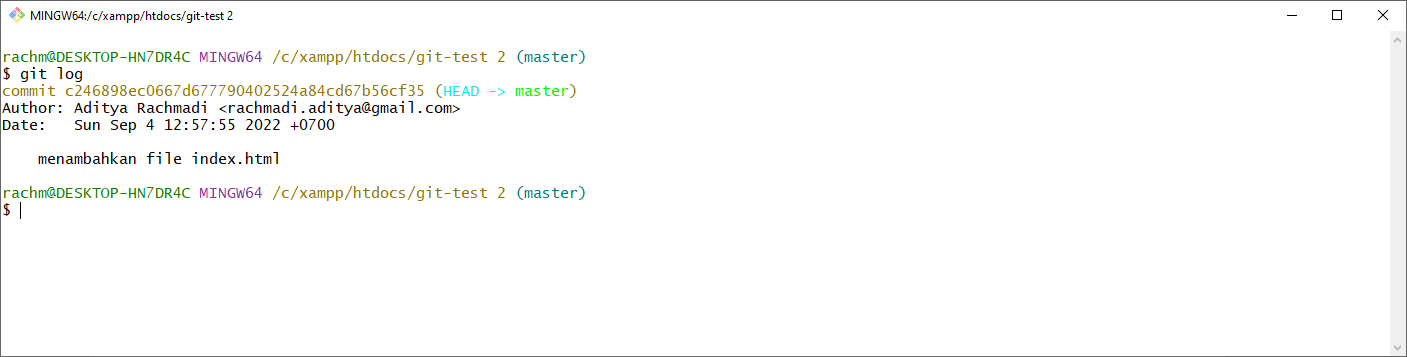
Gambar 42 File ditambah ke *staging area*

1. File yang telah ditambah kedalam *staging area* kemudian dilakukan commit untuk memasukkan file kedalam *.git directory*. Gunakan perintah **git commit -m “*keterangan*”**



Gambar 43 File masuk dalam *git directory*

1. Masukkan perintah **git log** untuk memantau riwayat *commit* yang telah dieksekusi. Hash pada file yang telah di-commit adalah **c246898ec0667d677790402524a84cd67b56cf35** atau **c245898** jika dilihat pada 7 angka pertama. Namun perlu diperhatikan bahwa disebelah *hash number* terdapat status (HEAD🡪MASTER) hal ini menegaskan bahwa *branch/*jalurutama didalam repository dinamakan “master”, sedangkan HEAD merepresentasikan *branch*/jalur yang sedang aktif.



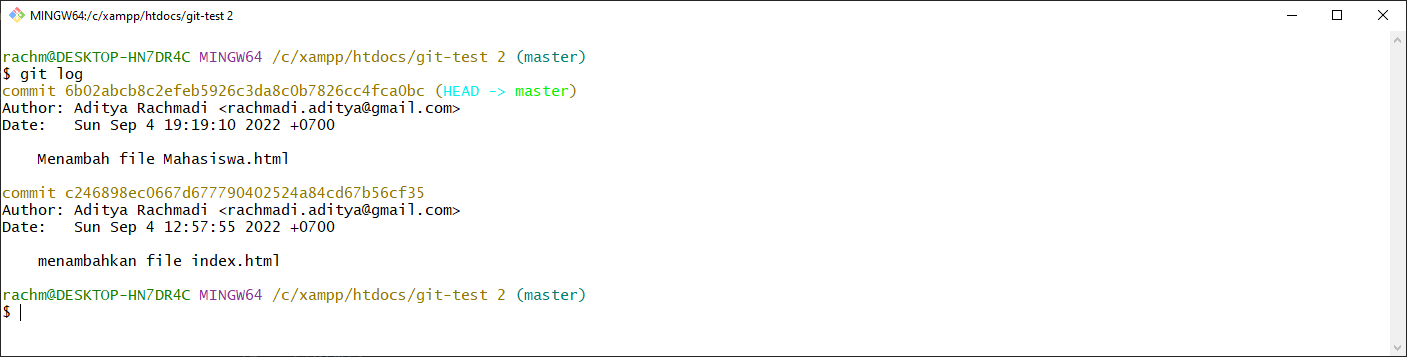
Gambar 44 Git Log

 **Task**: Buatlah 3-4 file tambahan pada *repository*. Jalankan perintah **git add** sampai dengan **git log** untuk tiap file yang ditambahkan, kemudian masukkan perintah **git log –all –decorate –oneline –graph**, dan kemudian catat luaran yang muncul pada GitBash.

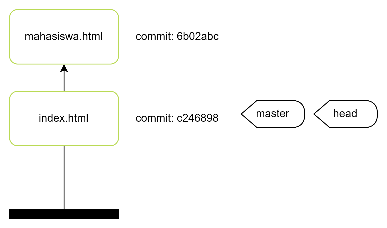
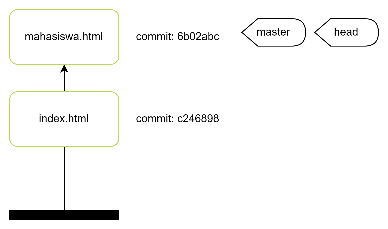
# GIT BRANCH & MERGE

Pada bagian ini akan dijelaskan langkah dalam melakukan penambahan *branch* dan juga menggabungkan *branch*.

1. Masuk dalam folder *repository* pada GIT dengan GIT Bash
2. Berdasarkan Gambar 45 telah ditambahkan file mahasiswa.html sehingga pada folder repo terdapat dua file yaitu index.html dan mahasiswa.html. Jika dijabarkan kedalam bagan maka bagan yang merepresentasikan commit yang telah dijalankan dalam repo digambarkan pada gambar 46.

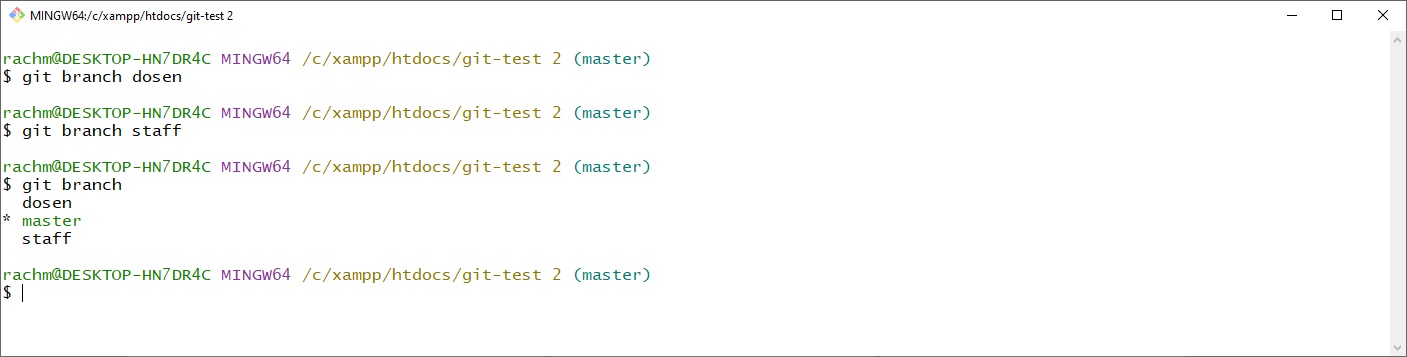


Gambar 45 File dalam repo



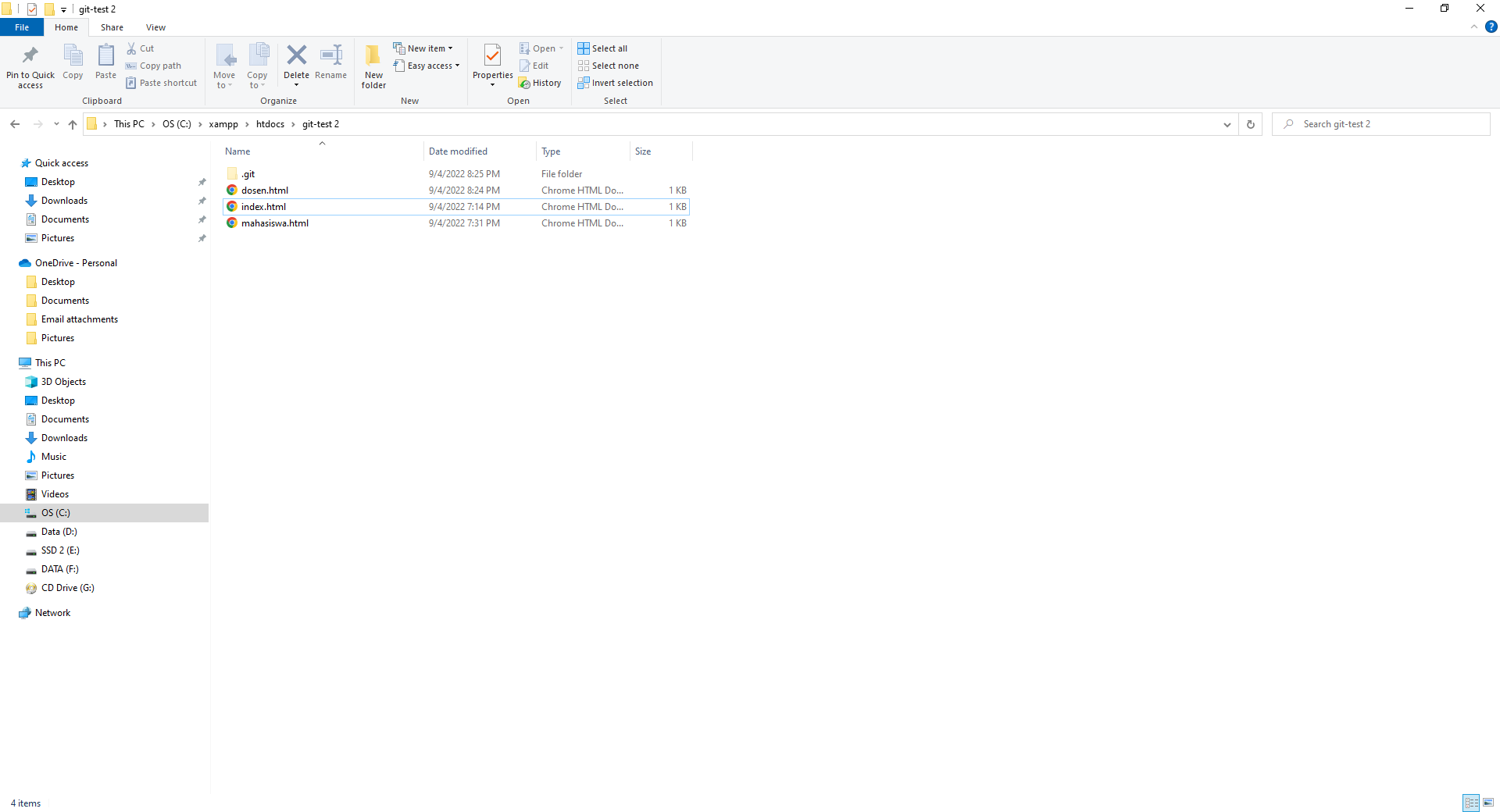
Gambar 46 Perubahan head dalam repository

1. Selanjutnya akan ditambahkan 2 *branch* pada GIT yaitu *branch* dosen dan staff dengan menggunakan perintah **git branch *nama\_branch*** sehingga **branch** menjadi 3 yaitu master, dosen, dan staff. **TIPS**: Perintah **git branch** dapat digunakan untuk memantau seluruh branch yang ada.
2. Masukkan perintah **git branch dosen** dan **git branch staff** utuk menambahkan *branch* dosen dan staff.



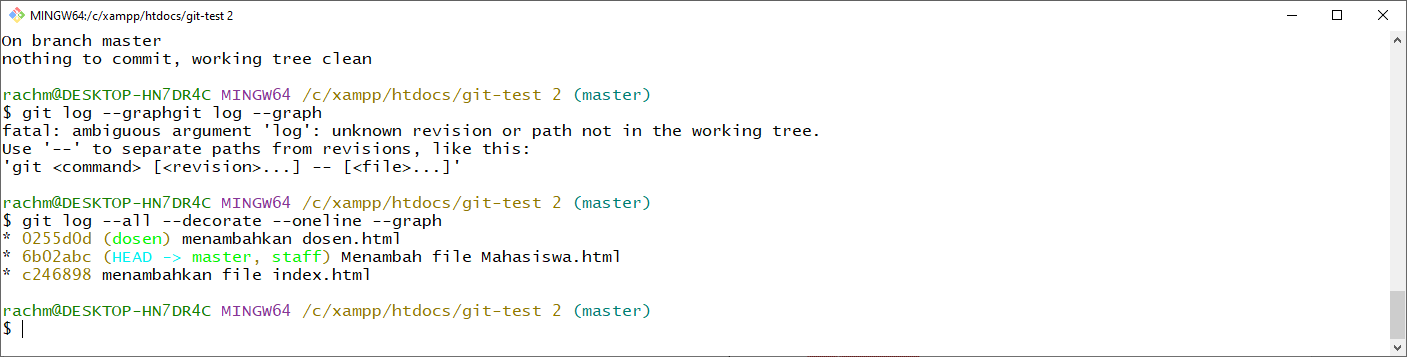
Gambar 47 Menambahkan branch

1. Gunakan perintah **git checkout** ***nama-branch*** untuk mengubah head kedalam branch yang diinginkan untuk aktif. Aktifkan branch dosen dengan memasukkan perintah **git checkout dosen**.
2. Tambahkan file baru sehingga pada branch dosen terdapat 3 file yaitu index.html, mahasiswa.html, dan file baru yang ditambahkan. Pada contoh, file baru diberikan nama dosen.html



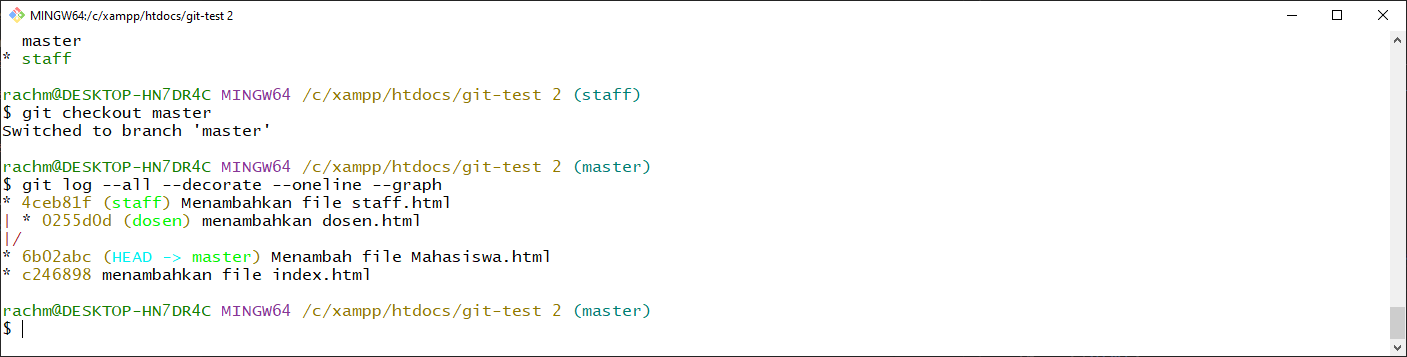
Gambar 48 File pada branch telah ditambahkan

1. Gunakan perintah **git checkout master** untuk kembali kedalam branch master. Perhatikan pada *file explorer* hanya terdapat 2 file yaitu index.html serta mahasiswa.html, dapat disimpulkan bahwa *snapshot* yang ditampilkan pada repo bersumber dari commit terakhir yang terjadi pada *branch master*.

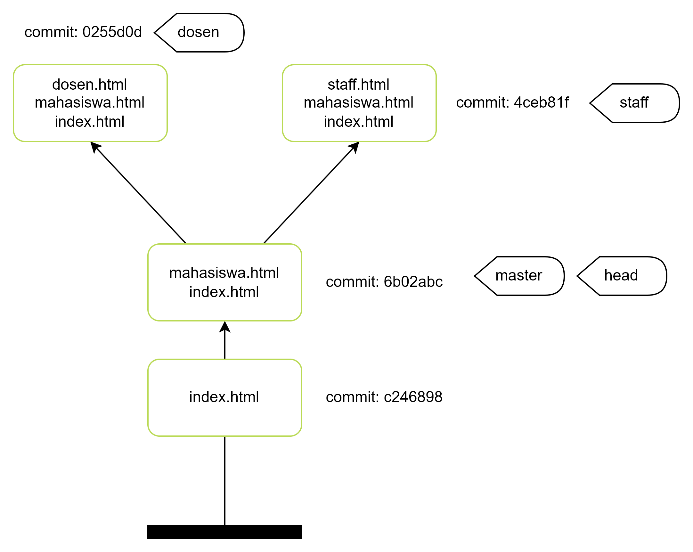


Gambar 49 Branch master aktif

1. Arahkan head ke *branch* staff kemudian tambahkan file dalam branch staff. Gunakan perintah **git log --all --decorate --oneline --graph** untuk menampilkan bagan dari commit yang telah tersusun.

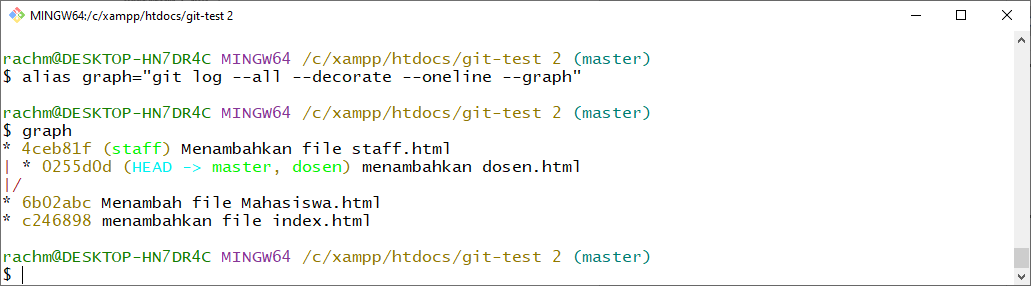


Gambar 50 Bagan dari repo

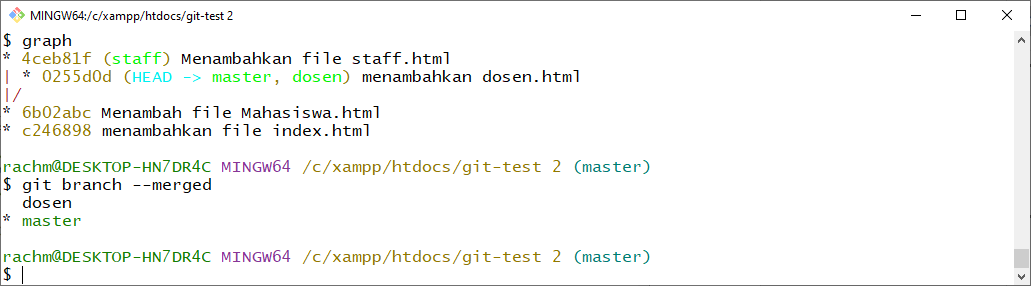


Gambar 51 Bagan dari branch pada repo

1. Langkah berikutnya adalah menggabungkan *branch* yang terpisah menjadi satu dimulai dengan penggabungan antara *branch master* dengan *branch dosen* yang kemudian digabung dengan *branch staff.*
2. Penggabungan Branch Master dengan Dosen: Aktifkan branch master terlebih dahulu kemudian masukkan perintah **git merge dosen**. Cek dengan menggunakan perintah graph sehingga muncul bagan seperti Gambar52. Penggabungan antar dosen dan master disebut **fast forward merge**
3. Cek sekali lagi branch yang telah tergabung dengan menggunakan perintah **git branch --merged** sehingga terlihat luaran seperti Gambar 53.



Gambar 52 Bagan branch dalam repo



Gambar 53 Branch dosen dan master terbagung

1. Gabungkan antara branch master dengan branch staff sehingga branch master memiliki 4 file yaitu mahasiswa.html, index.html, dosen.html, dan staff.html

# GIT MERGE CONFLICT

Pada bagian ini akan dijabarkan langkah-langkah dalam menyelesaikan permasalahan *merge conflict* dalam GIT. Konflik *Merge* terjadi karena terdapat dua branch yang mengerjakan komponen/baris yang sama pada satu file didalam suatu repo.

Langkah awal untuk dapat menerapkan *Git Merge Conflict* adalah membuat dua cabang atau *branch* kemudian melakukan perubahan dalam file pada bagian yang sama di dua *branch* yang telah dibuat. Kondisi ini dapat dicapai dengan melakukan langkah-langkah sebagia berikut:

1. Masuk dalam folder *repository* pada GIT dengan GIT Bash
2. Setelah mengakses folder *repository* maka buat satu *branch* dengan menggunakan perintah **Git branch** **revisi***.*
3. Aktifkan branch yang telah dibuat dengan perintah **Git checkout revisi.** Kemudian rubah salah satu dari file yang telah dibuat pada repository. Pada contoh terdapat empat file yaitu index.html, dosen.html, mahasiswa.html, dan staf.html dengan perubahan dijalankan pada file mahasiswa.html terutama merubah bagian header menjadi h4
4. Simpan perubahan pada *staging area,* dilanjutkan dengan commit perubahan.
5. Pindah ke *branch* master dengan perintah **git checkout *nama\_branch****.* Sehingga pada saat dieksekusi perintah Git log dapat terlihat pada bagan bahwa *branch master* teraktifasi atau head telah menunjuk pada *branch master*.
6. Rubahfile yang sama dengan file yang telah dirubah pada *branch* revisi, terutama pada bagian yang sama. Misal: rubah bagian header menjadi h1.
7. Simpan pada *staging area* dilanjut dengan *commit*.
8. Coba perhatikan bagan branch yang telah disusun dengan menginputkan perintah **git log --all --decorate –oneline –graph.** Perhatikan apa yang telah terjadi!
9. Coba lakukan merge branch dengan perintah **git merge revisi,** makaakan muncul pesan bahwa terjadi konflik pada file yang sama. Hal ini terjadi karena satu file memiliki bagian yang sama-sama dirubah sebelumnya
10. Perbaiki atau *resolve* konflik yang terjadi dengan cara edit konten file dengan editor kemudian simpan.
11. Pada Git Bash dapat dilakukan penyimpanan file yang telah dimodifikasi pada staging area, dilanjutkan dengan commit.

Chapter

3

GITHUB

GITHUB merupakan website yang memiliki layanan utama untuk mengelola project GIT

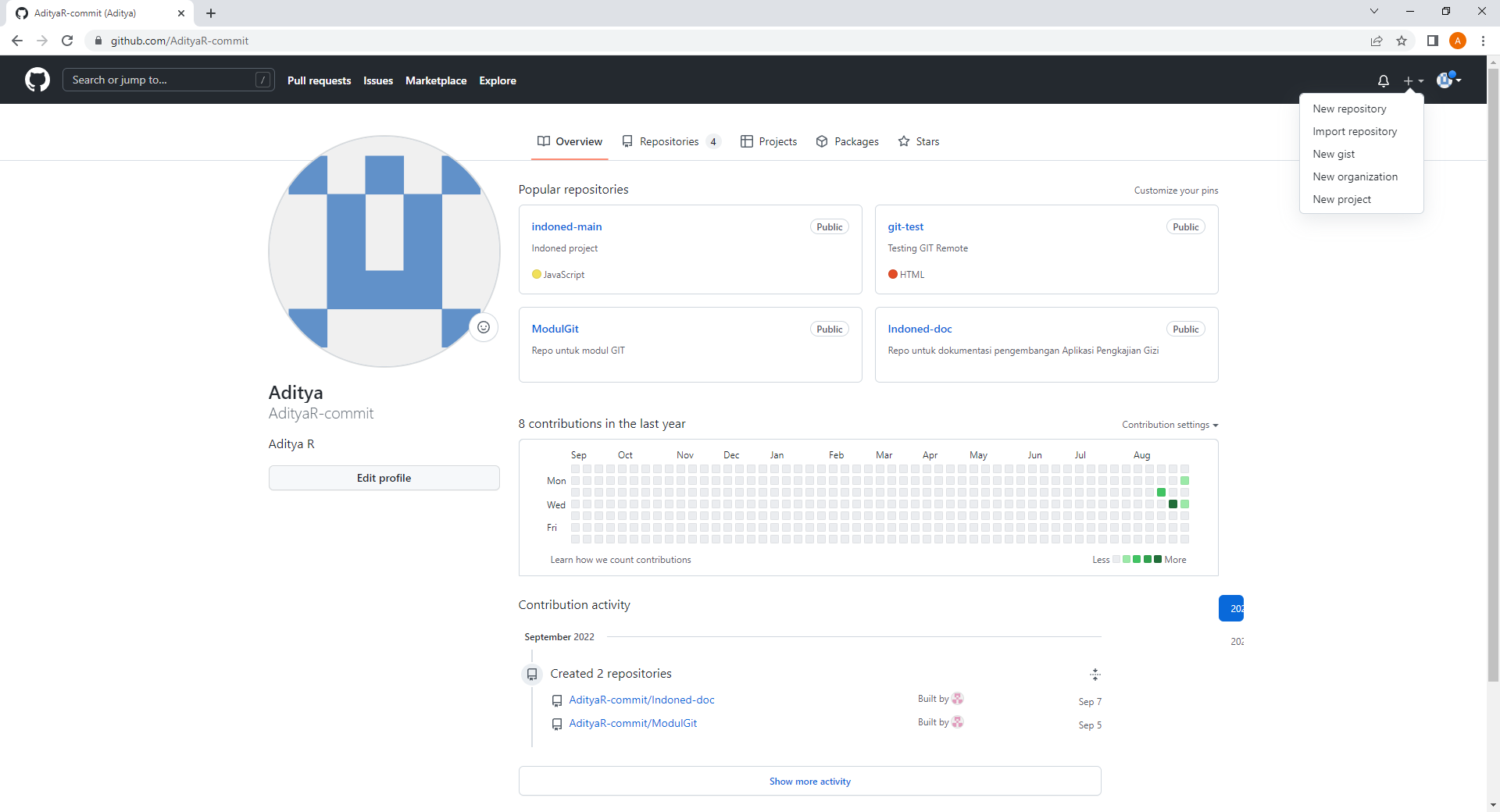
P

ada bagian ini akan dijelaskan penggunaan GITHUB sebagai *version control system* serta pembuatan repository/repo dengan menggunakan GITHUB. Penjelasan mekanisme *version control* melalui GITHUB dalam bab ini dibagi menjadi tiga subbagian yaitu: A) Bekerja dengan github. Langkah-langkah untuk membuat repository dengan GITHUB serta melakukan *versioning* pada file dijelaskan pada sub bagian ini. B) Mekanisme pembuatan branch serta langkah menyatukan branch didefinisikan pada sub bagian BRANCH. C) Proses duplikasi *repository* orang lain beserta history commitnya dijelaskan pada bagian FORK.

# Bekerja dengan GITHUB

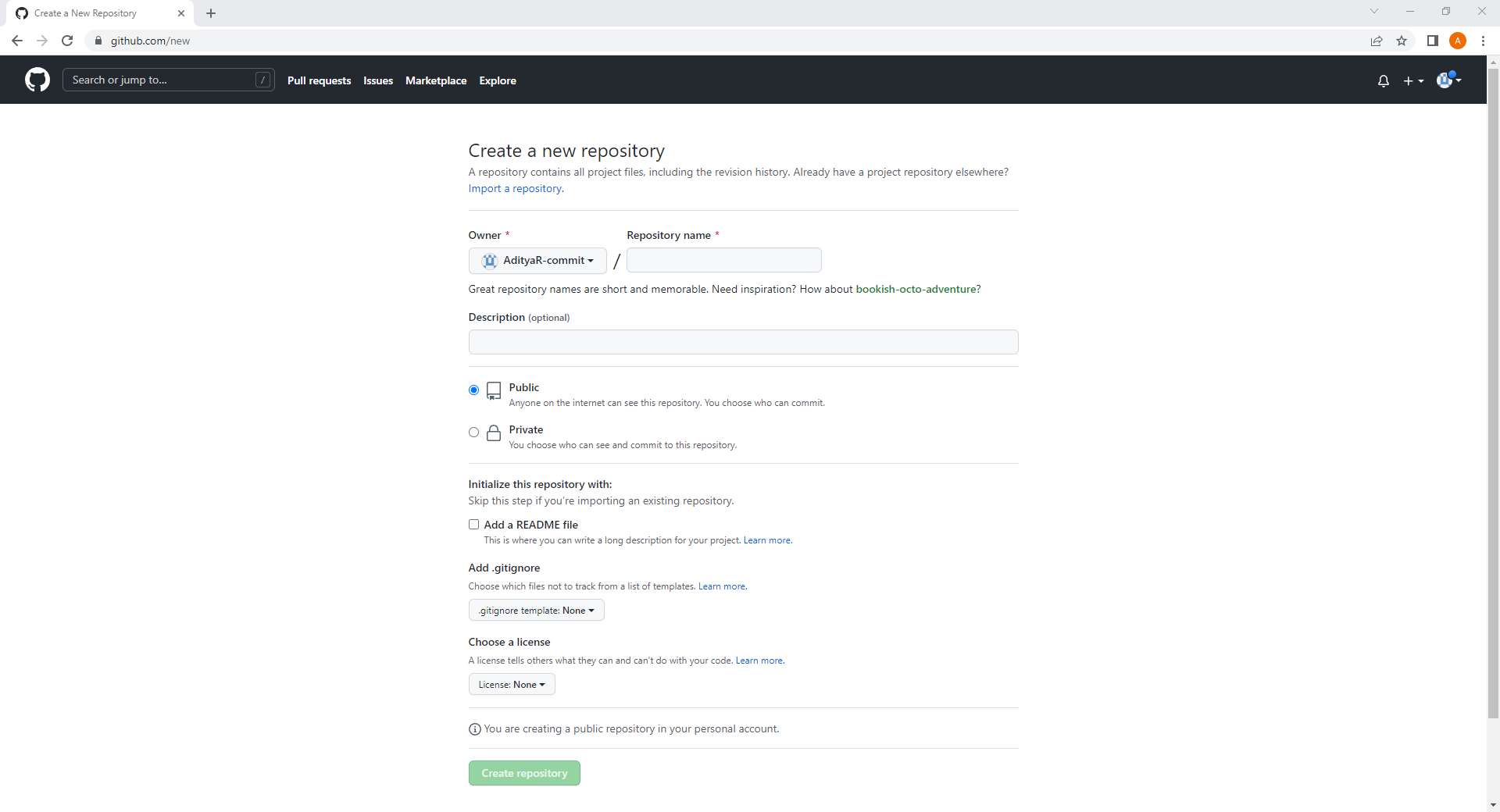
Pada bagian ini akan dijelaskan langkah-langkah dalam membuat repository dengan menggunakan GITHUB.

1. Masuk GITHUB dengan data akun yang telah dibuat.
2. Klik tombol plus pada halaman akun GITHUB dan pilih *New repository* untuk membuat *repository* baru. Fungsi New repository diilustrasikan pada Gambar 54.



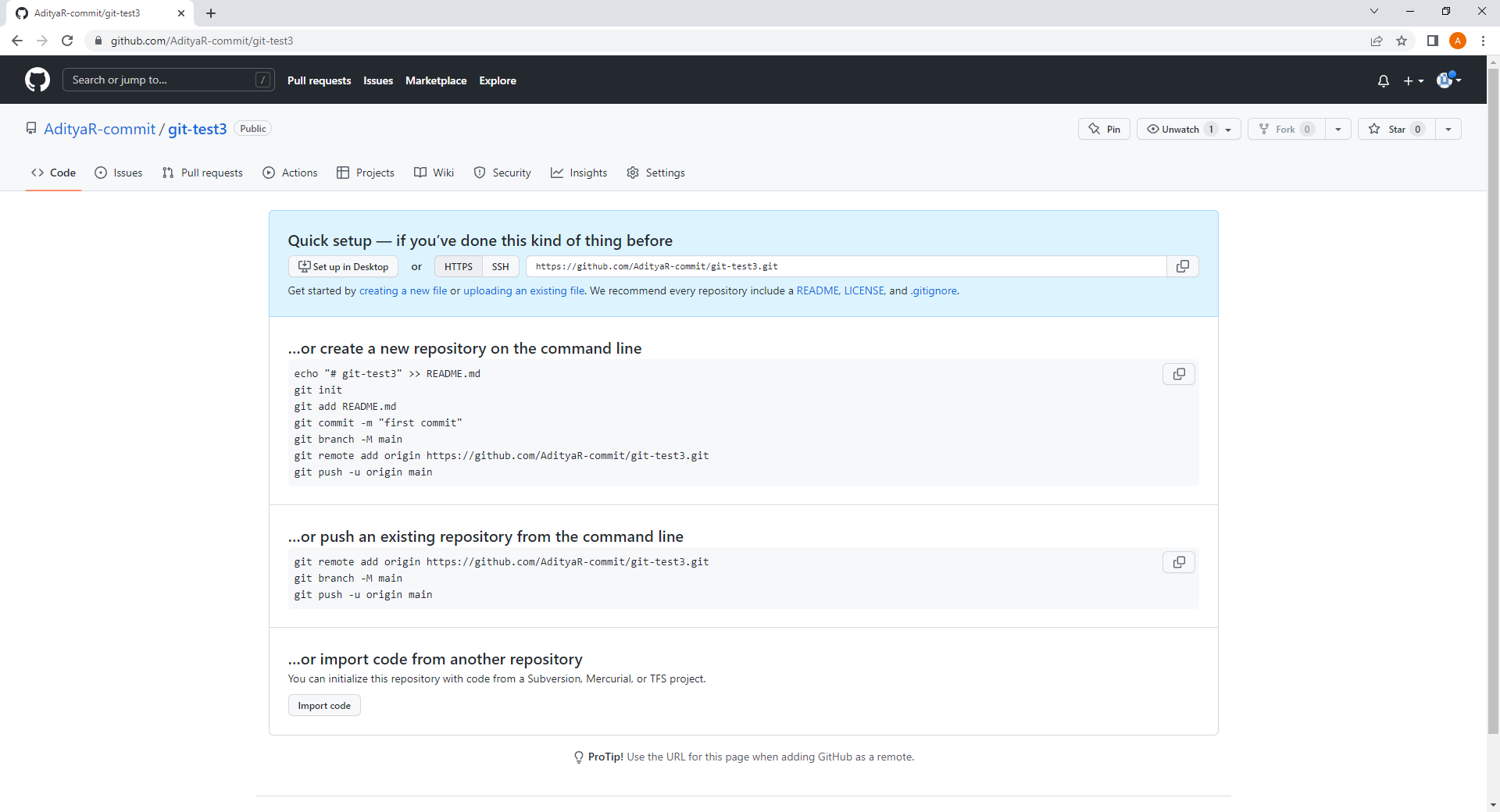
Gambar 54 Menu *repository* baru GITHUB

1. Buat repository baru dengan langkah diatas, kemudian GITHUB menampilkan form untuk membuat repo baru. Isikan nama repo kemudian berikan deskripsi yang sesuai pada *repository* yang akan dibangun. Pilih apakah repository akan dapat diakses oleh orang umum atau tidak melalui opsi Public dan Private. Jika dipilih *public* maka konten dari repo akan dapat dilihat oleh umum, namun jika dipilih private maka konten hanya dapat diakses oleh akun pribadi dan referral.
2. Pilih apakah akan menambahkan file *readme* atau tidak, jika dipilih tidak ditambahkan *file readme* maka repo tidak berisi file apapun. Jika sudah yakin dengan pilihan yang ditentukan maka klik *Create Repository.* Form repo GITHUB diilustrasikan pada gambar 55.

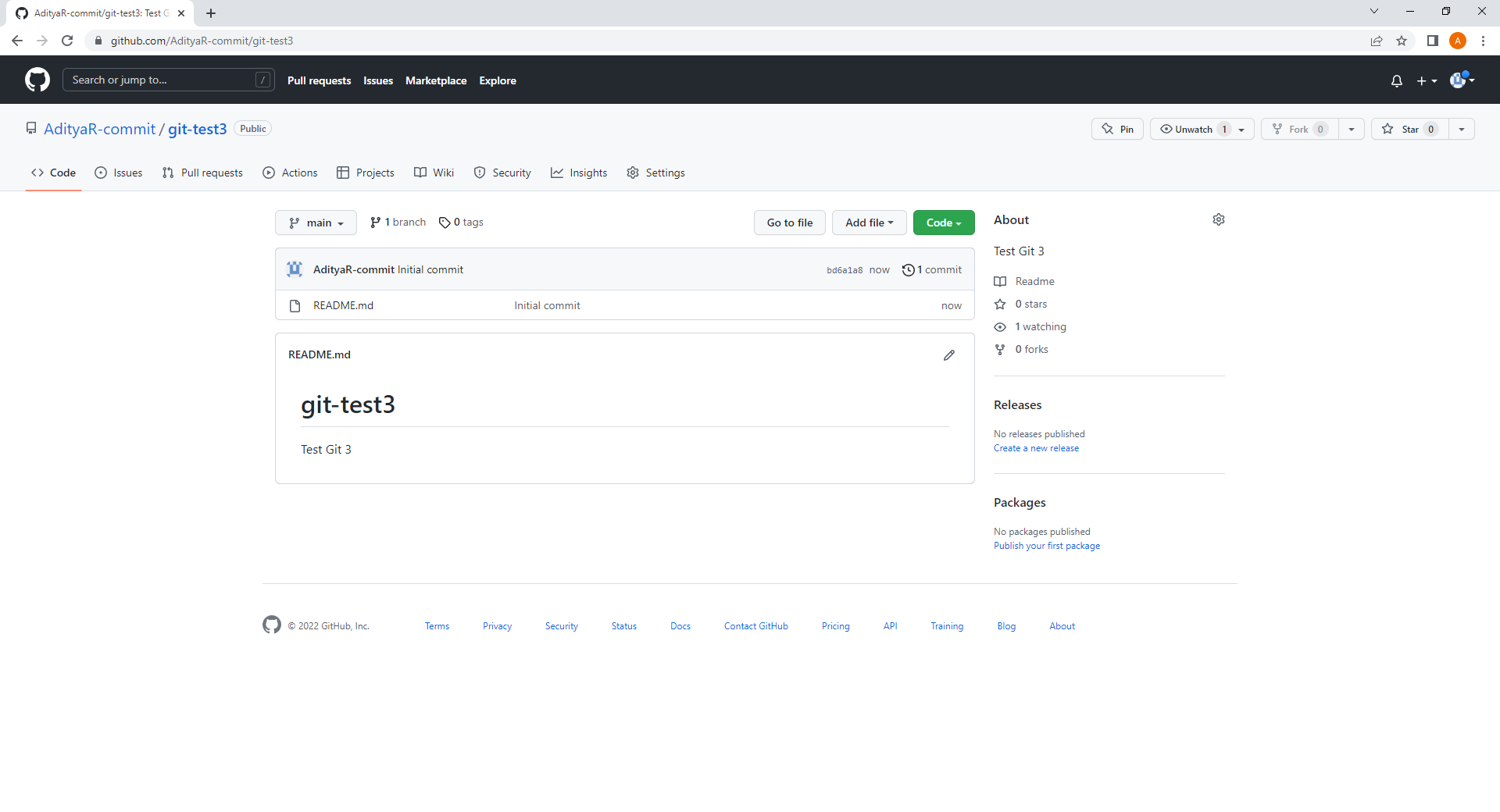


Gambar 55 Form buat *repository* baru GITHUB

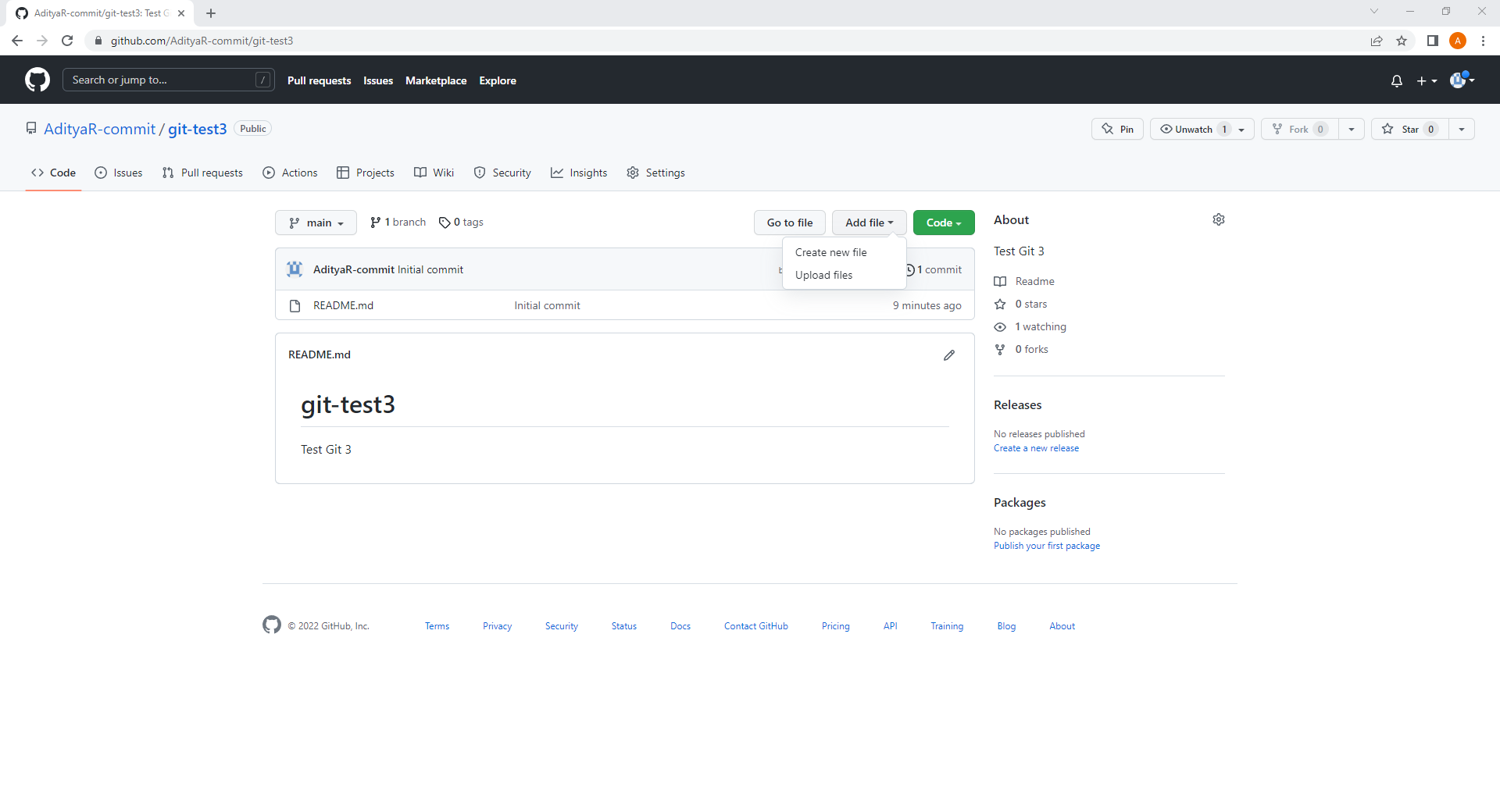
1. Jika tidak dipilih untuk membuat dokumen readme maka akan ditampilkan perintah-perintah yang dapat digunakan untuk melakukan push dan membuat *repository* pada computer pribadi. Gambar 56 menampilkan halaman *repository* jika dokumen readme tidak dibuat.
2. Gambar 57 merupakan ilustrasi dari halaman *repository* dengan penambahan dokumen readme.
3. Dokumen baru dapat ditambahkan dengan langkah klik tombol Add file 🡪 Create new file seperti pada Gambar 58.



Gambar 56 Repository GITHUB tanpa file readme

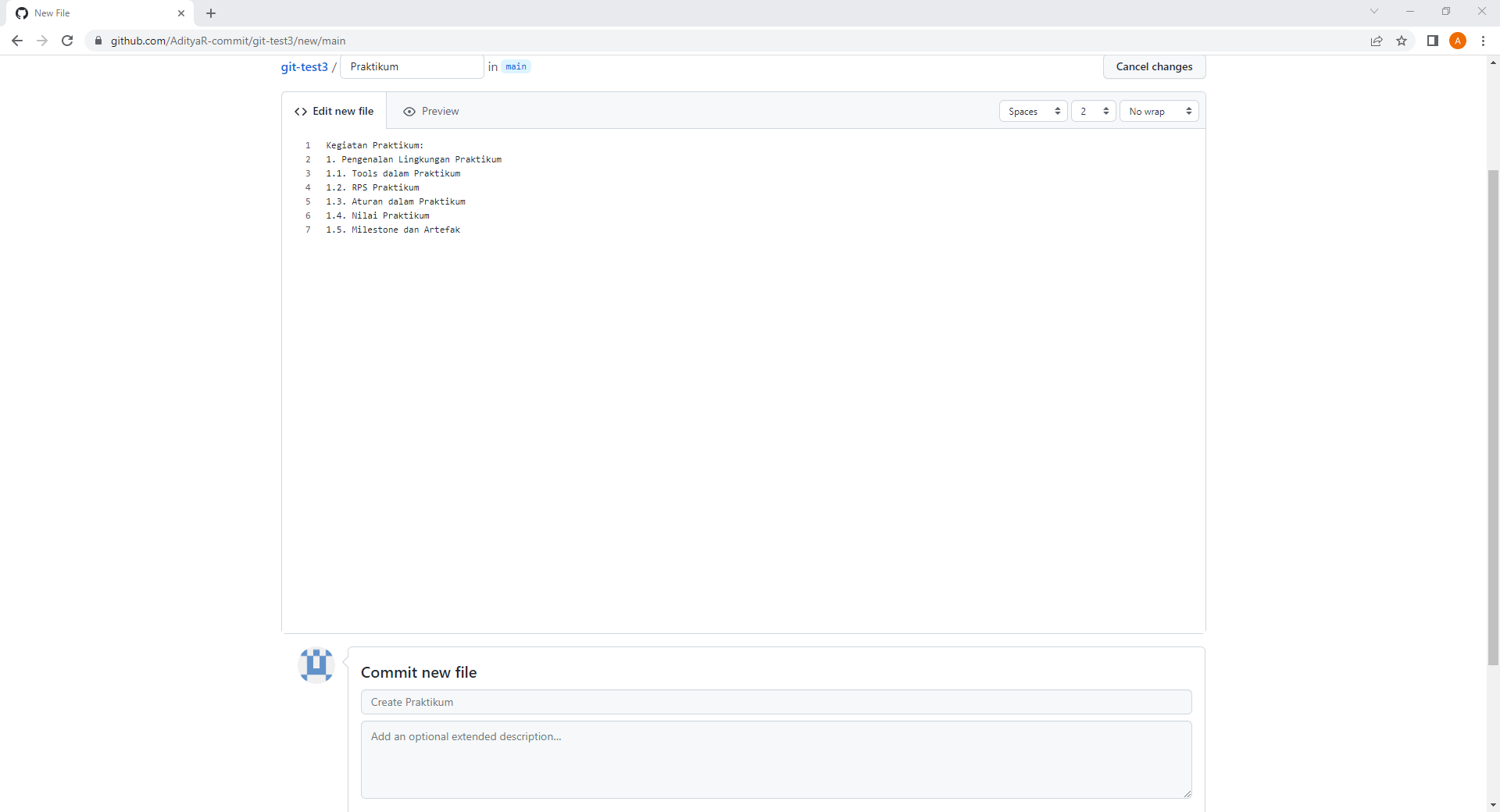


Gambar 57 Repository GITHUB tanpa file readme



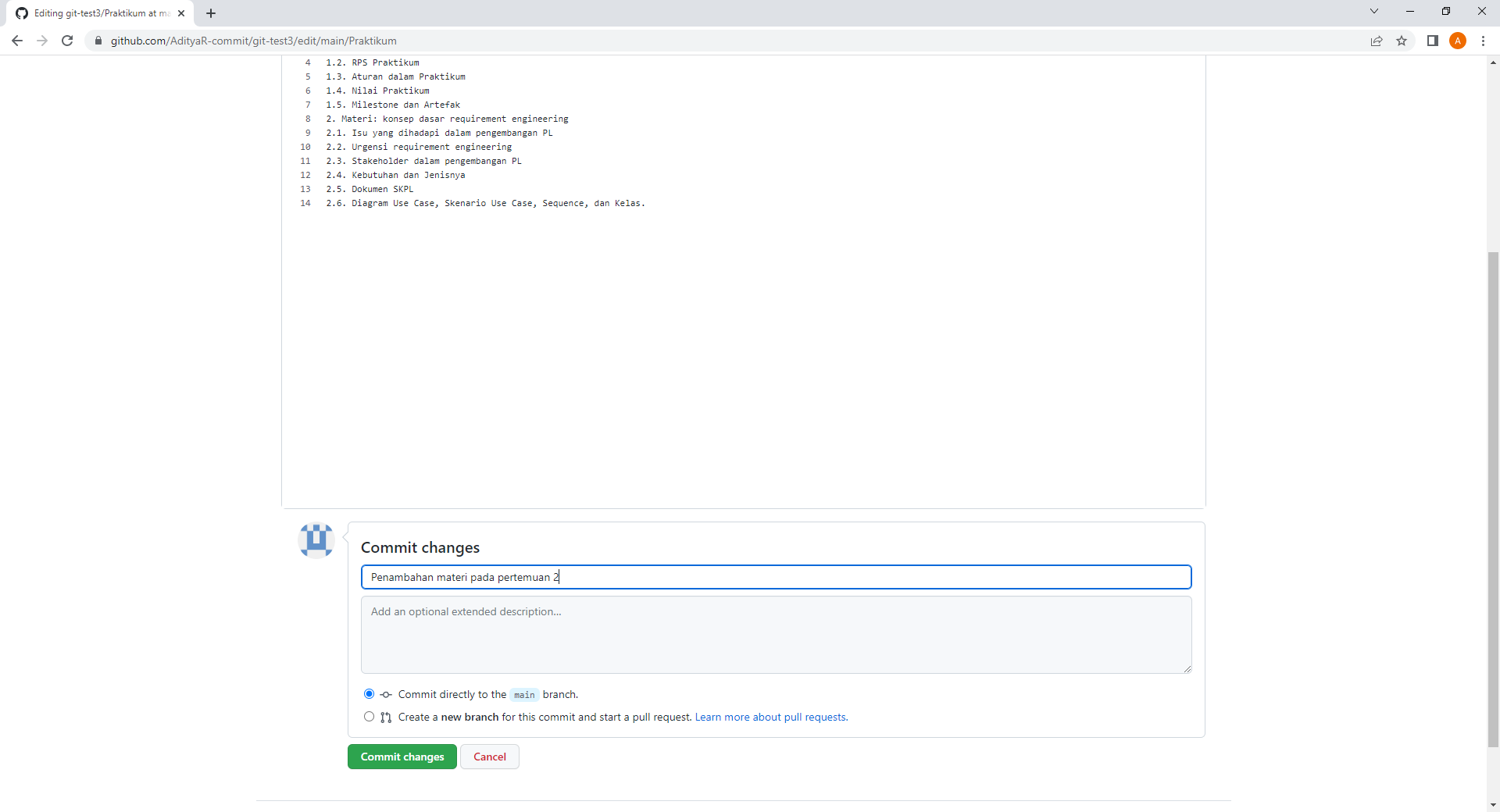
Gambar 58 Create New File GITHUB

1. Tambahkan isi pada file baru yang ditampilkan oleh GITHUB 🡪 berikan nama pada file tersebut 🡪 Pada form *commit new file* berikan narasi yang menggambarkan deskripsi dari file tersebut terutama isi konten dan perubahannya 🡪 Pilih Commit directly to the main branch 🡪 Commit new file 🡪 File telah ditambahkan pada *repository*. Langkah ini dideskripsikan pada Gambar 59.



Gambar 59 Create New File GITHUB

1. Ubah dokumen dengan cara pilih dokumen yang akan dirubah 🡪 tekan tombol pensil yang menandakan dokumen akan diedit 🡪 edit dokumen. History pada dokumen digunakan untuk memantau log dari commit yang telah dibuat, melalui history juga dapat dipantau perubahan yang terjadi didalam dokumen.

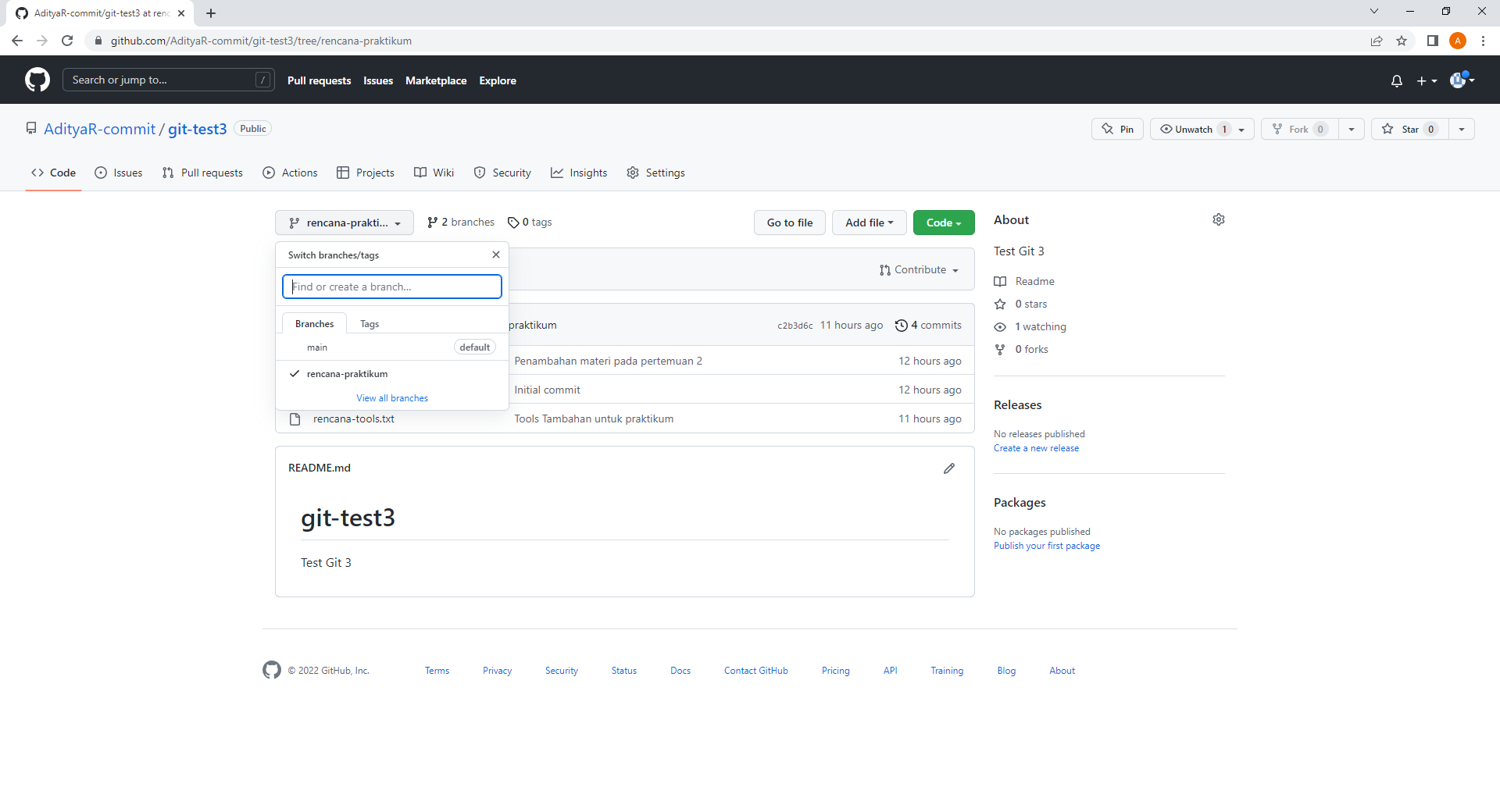


Gambar 60 Edit file di GITHUB

# Branch

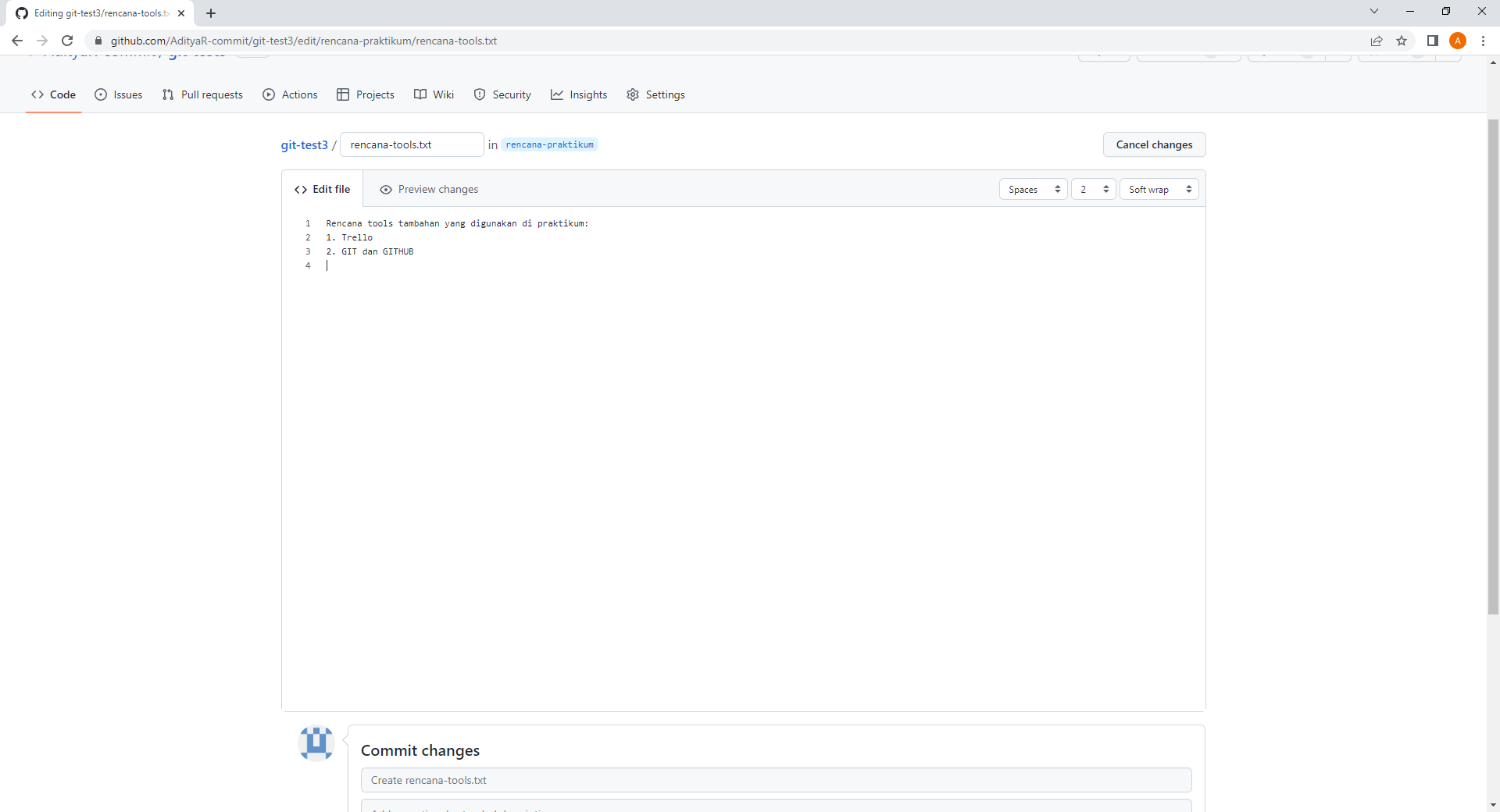
Pada bagian ini akan dijelaskan langkah-langkah dalam melakukan branching dan merge pada branch dengan menggunakan GITHUB.

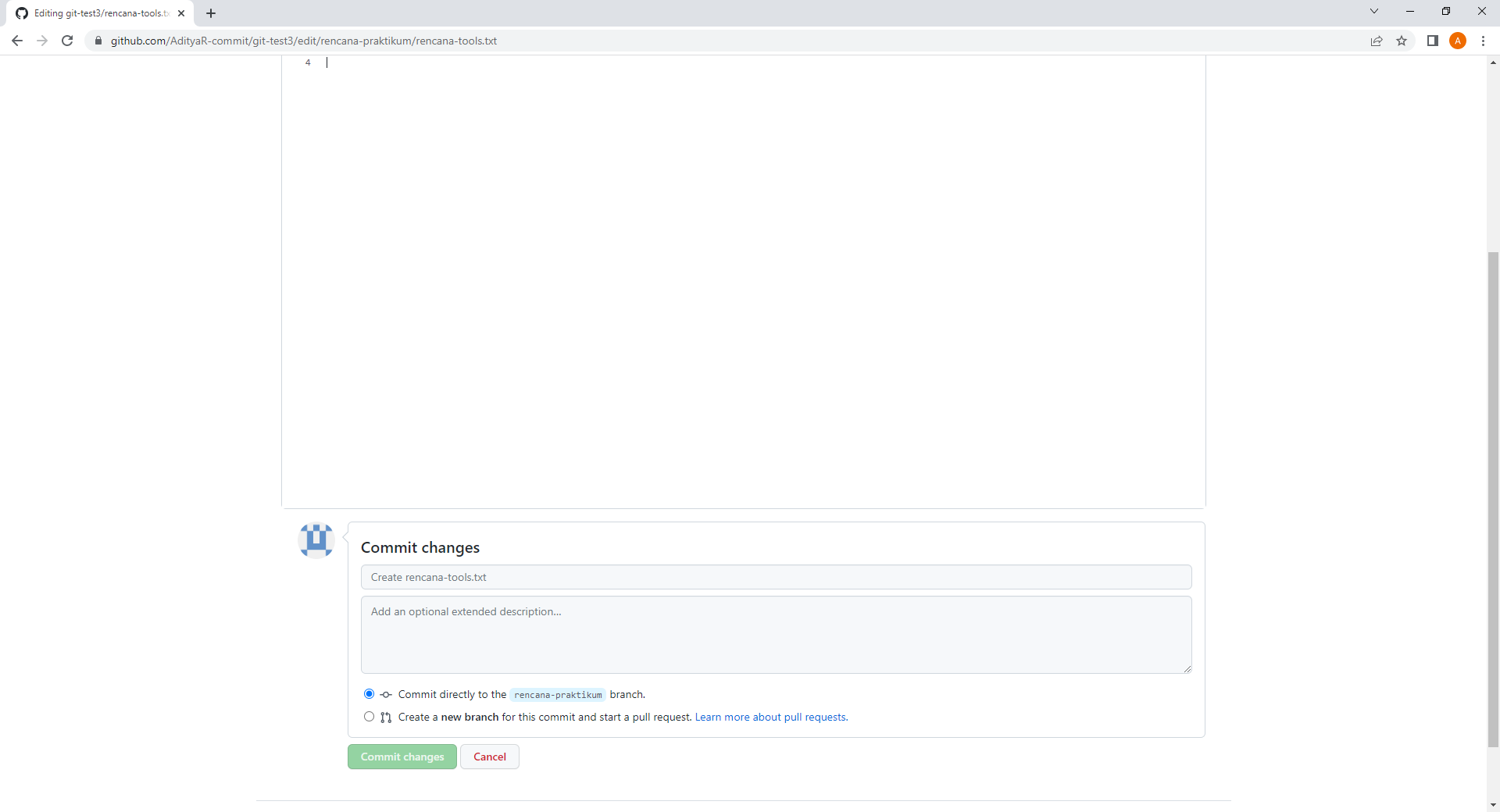
1. Masuk GITHUB dengan akun yang telah dibuat.
2. Pilih repository yang akan menjadi tempat kerja dari pembuatan *branch*. Pada contoh digunakan repo dengan nama git-test3
3. Tambahkan *branch* yang akan menjadi cabang dari *branch master.* Branch master merupakan branch utama yang secara otomatis ada ketika dibuat *repository* di GITHUB. Berikan nama yang sesuai pada branch baru yang telah dibuat 🡪 aktifkan branch baru untuk selanjutnya dibuat file baru pada branch tersebut. Pada contoh nama branch baru adalah *rencana-praktikum*.



Gambar 61 Membuat *branch* baru di GITHUB

1. Buat file baru pada branch baru yang telah diaktifkan, kemudian isi file tersebut serta berikan komentar ketika akan melakukan commit. Perhatikan bahwa sekarang file akan disimpan pada branch baru yang telah dibuat. Pada contoh file baru disimpan denngan nama *renana-tools.txt* pada branch dengan nama *rencana-praktikum.*



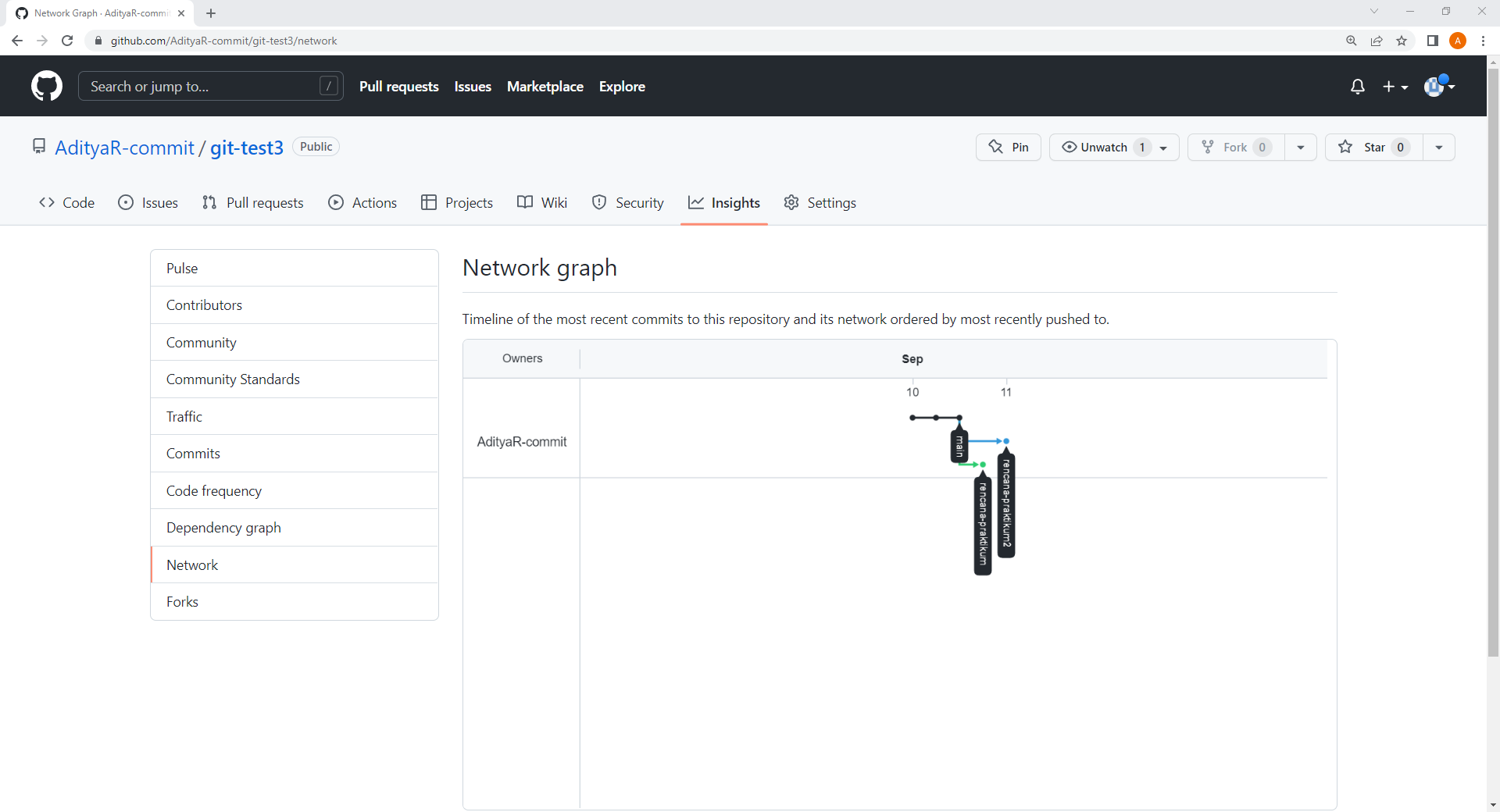


Gambar 62 Simpan file pada *branch* baru

1. Tambahkan *branch* baru lagi yang akan menjadi cabang dari *branch master.* Berikan nama yang sesuai konteks pada branch baru yang telah dibuat 🡪 aktifkan branch baru untuk selanjutnya dibuat file baru pada branch tersebut. Pada contoh nama branch baru adalah *rencana-praktikum2*.
2. Pada branch ketiga buat sebuah file dengan nama namun memiliki konten berbeda dengan file baru pada branch kedua 🡪 berikan komentar ketika akan melakukan commit.

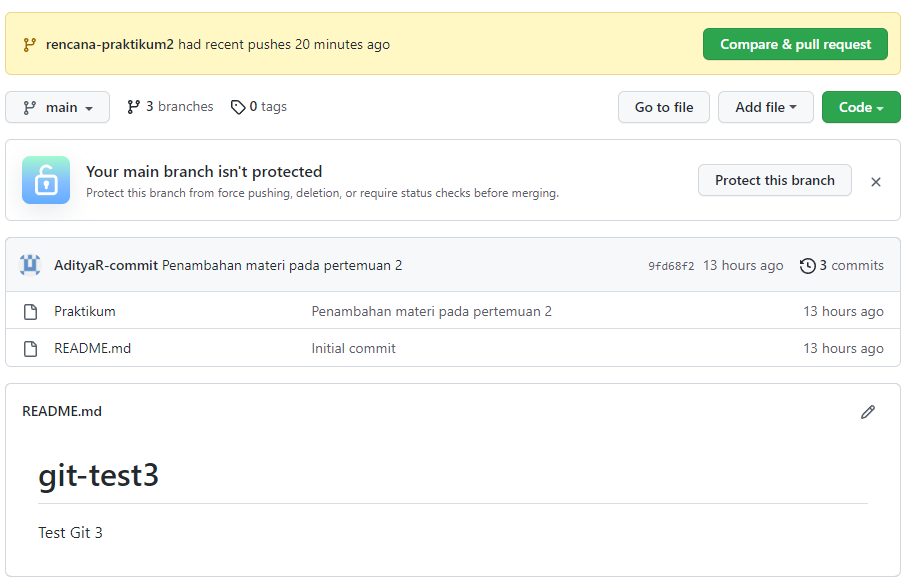
Perhatikan bahwa sekarang file akan disimpan pada branch ketiga. Pada contoh, file baru disimpan dengan nama *rencana-tools.txt* pada branch dengan nama *rencana-praktikum2.*

1. Bagan yang menunjukkan keterkaitan antara 3 branch dapat dipantau pada *tab insight* dengan menu *network*.



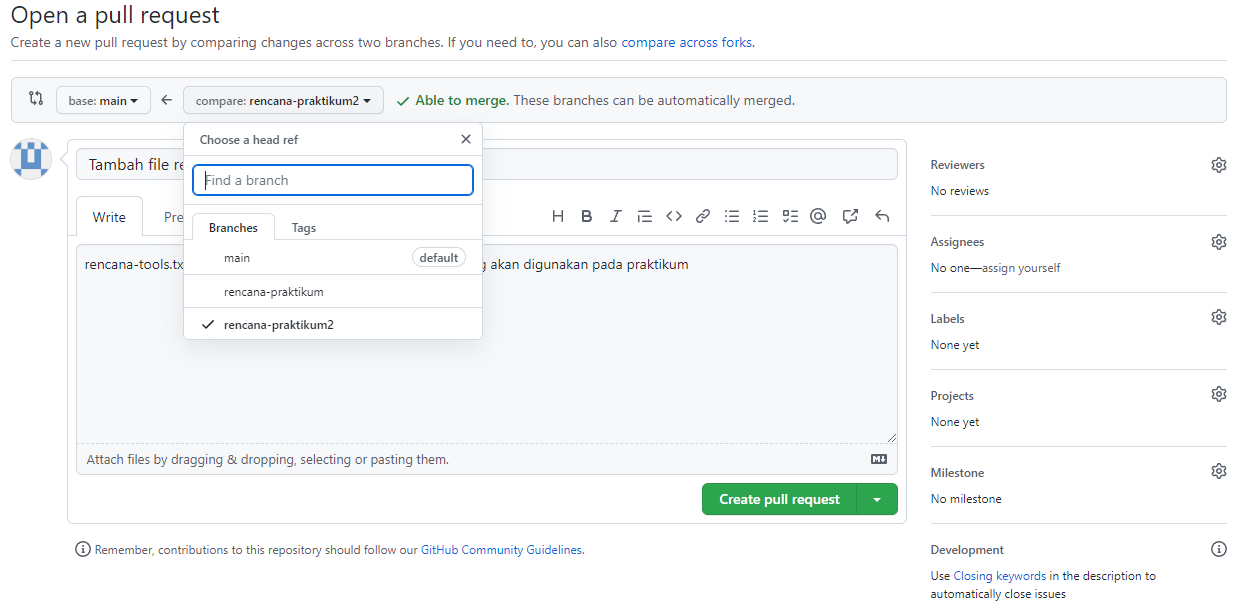
Gambar 63 Network pada tab insight

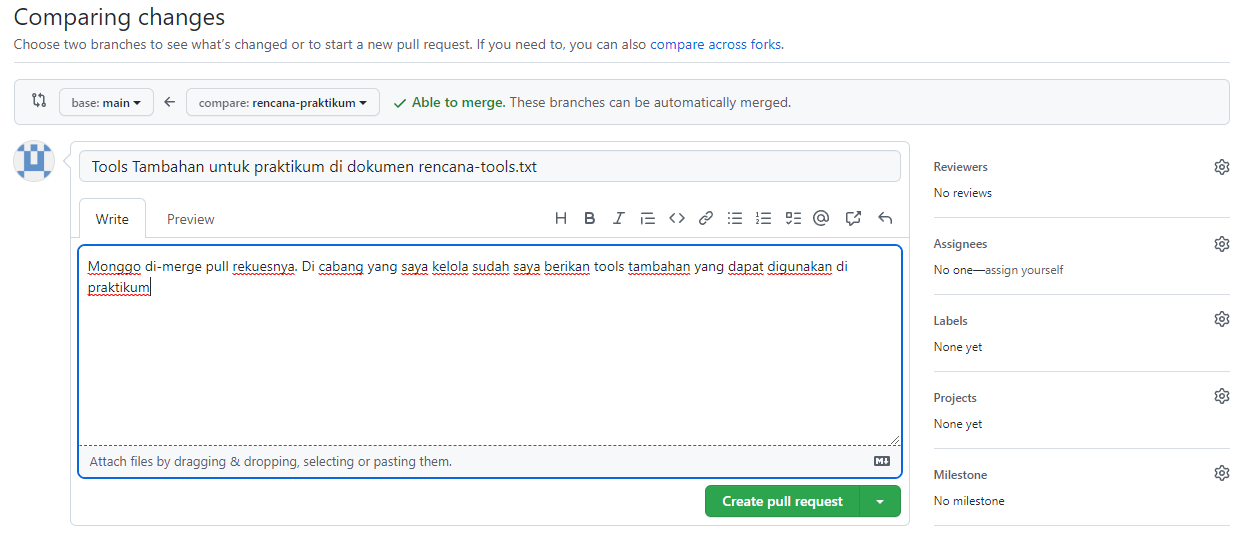
1. Lakukan merge dengan mengaktifkan terlebih dahulu *branch* master/main 🡪 Jalankan Compare & Pull Request seperti pada Gambar 64.



Gambar 64 Compare & Pull Request

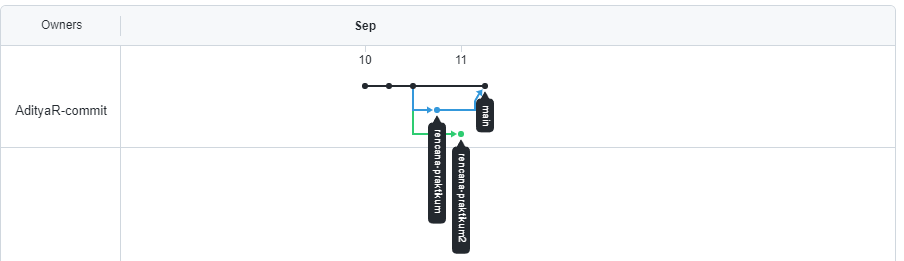
1. GITHUB akan menampilkan laman opsi merge, pastikan bahwa base telah menunjuk pada cabang utama/*main branch* sedangkan compare menunjuk pada cabang kedua dengan status **Able to Merge** 🡪 Berikan komentar 🡪 Pilih Create Pull Request.



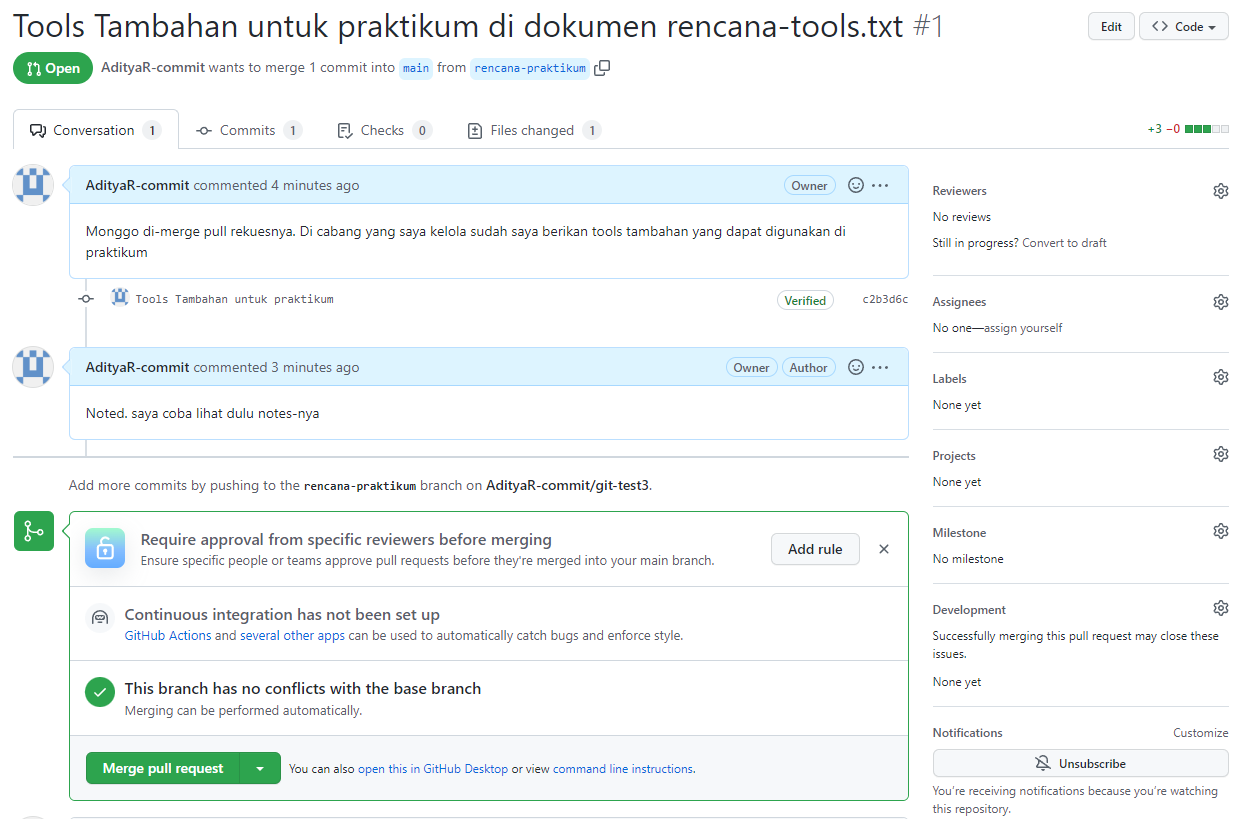


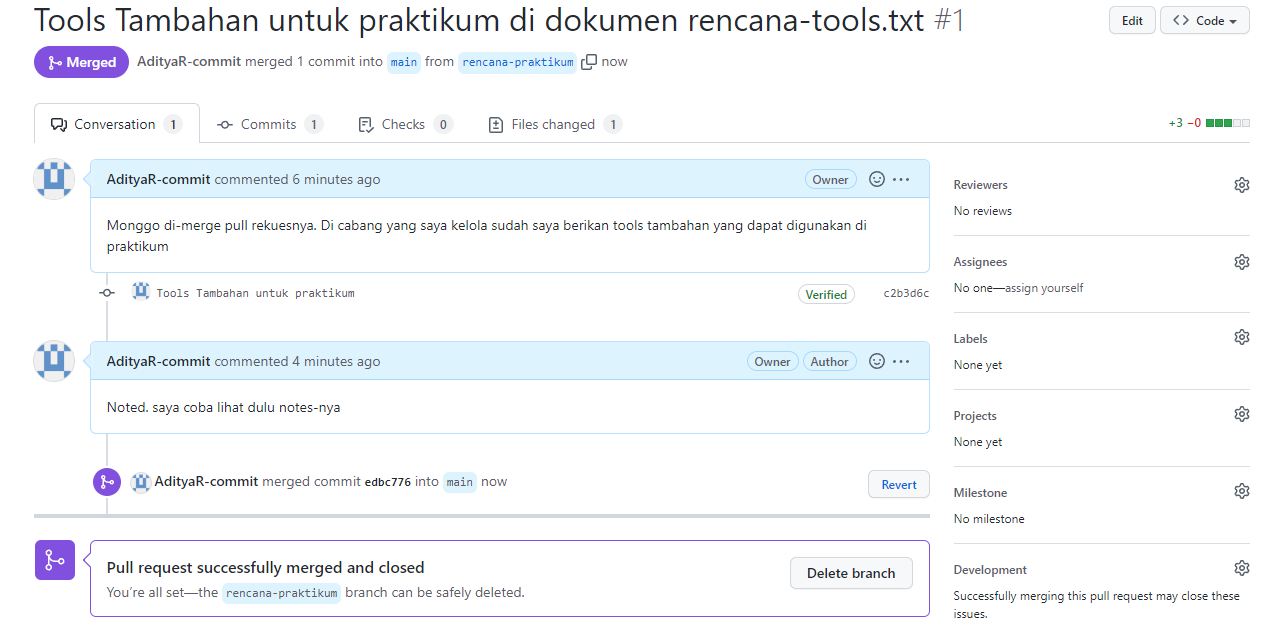
Gambar 65 Compare & Pull Requests

1. Klik pada **merge pull request** dengan opsi berikan komentar pada permintaan pull request. Jika berhasil melakukan merge maka akan muncul status yaitu **Pull Request Successfully Merged and Closed**. Langkah ini diilustrasikan pada Gambar 66.
2. Lihat dan perhatikan pada bagian *network*!



Gambar 66 Network setelah merge





Gambar 66 Merge Pull Requests

 **Task**: Hapus cabang atau *branch* kedua yang telah dilakukan merge, kemudian jalankan merge antara cabang utama dengan cabang ketiga. Solusikan *conflict* yang terjadi antar dua cabang tersebut serta perhatikan pada menu insight bagan yang merepresentasikan keadaan antar 3 cabang.

# Fork

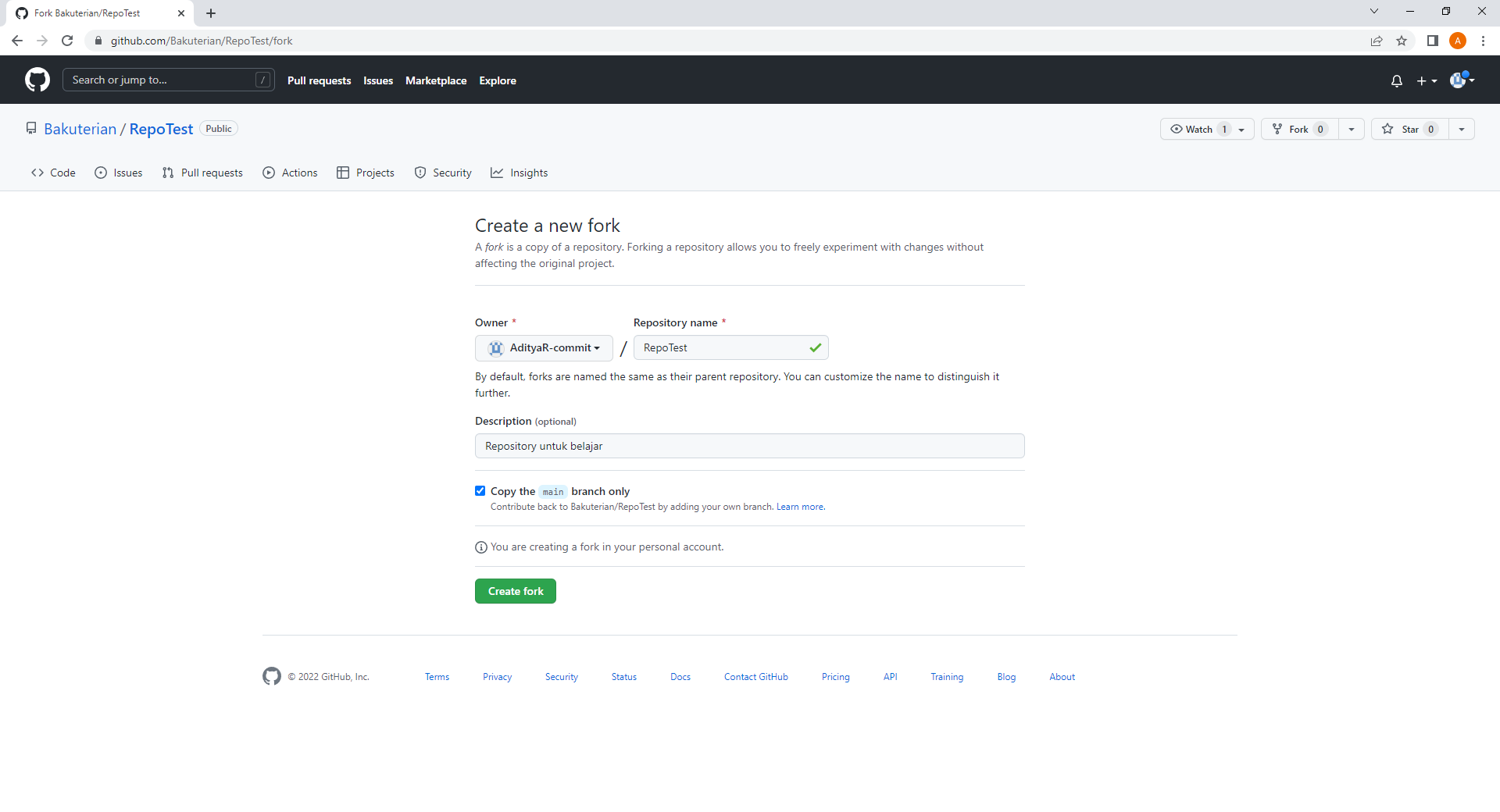
Pada bagian ini akan dijelaskan langkah-langkah dalam membuat salinan repo lain pada repository milik pribadi dengan menggunakan fitur Fork dalam GITHUB. Fork hanya dapat dijalankan bila telah dipilih target duplikasi *repository* lain dalam GITHUB, dengan melakukan *fork* maka dapat dilakukan perubahan konten dalam repo lain tanpa mengganggu cabang utama dari repo orisinalnya. Berikut adalah langkah yang dapat dijalankan:

1. Masuk GITHUB dengan akun yang telah dibuat.
2. Pilih *repository* lain yang akan menjadi target duplikasi 🡪 tekan tombol FORK seperti pada Gambar 68.



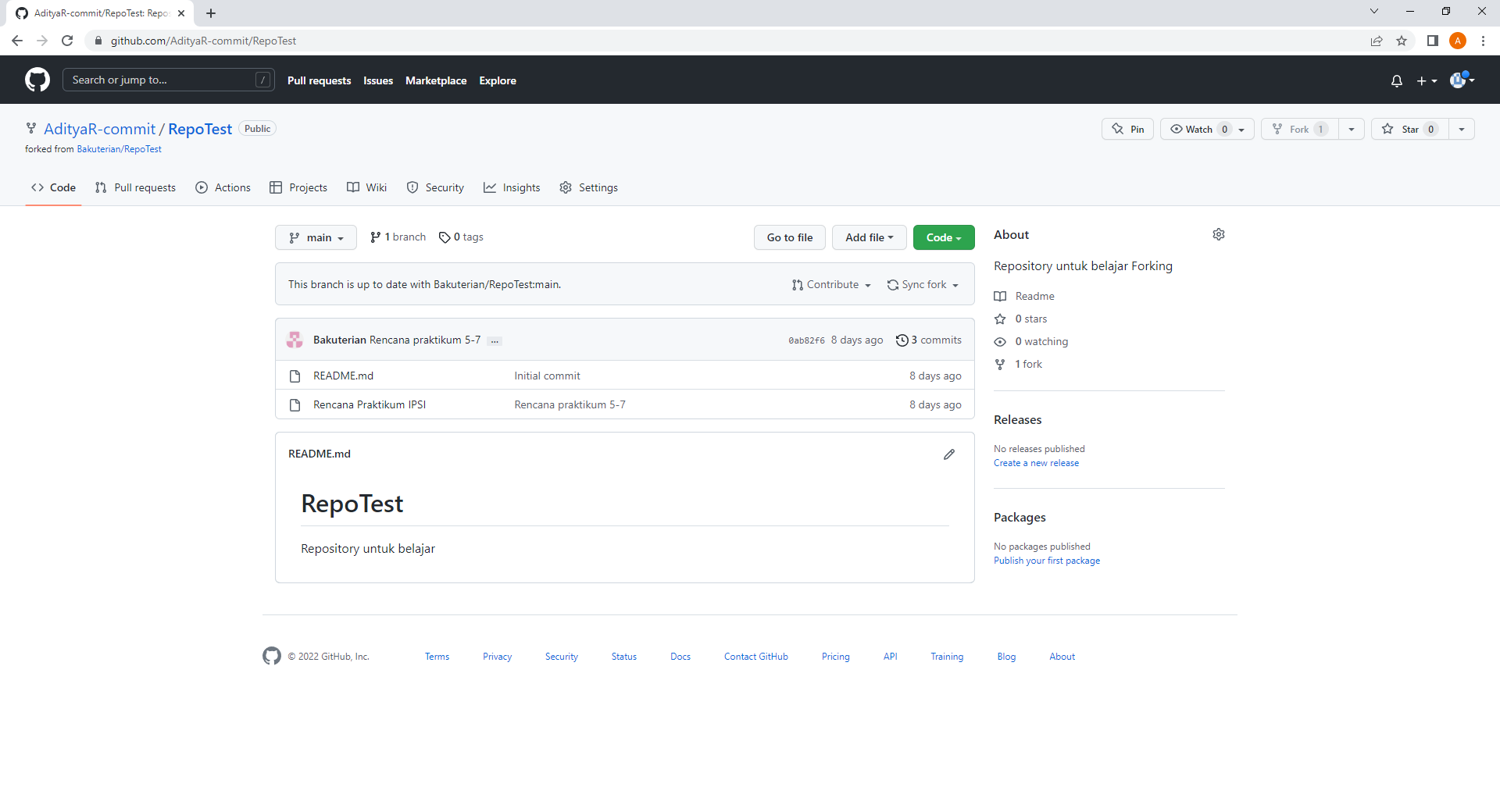
Gambar 68 Fork pada repo contoh

1. Tekan tombol **Create Fork** untuk konfirmasi aksi fork yaitu duplikasi snapshot dan history commit repo orisinal ke repo tujuan.



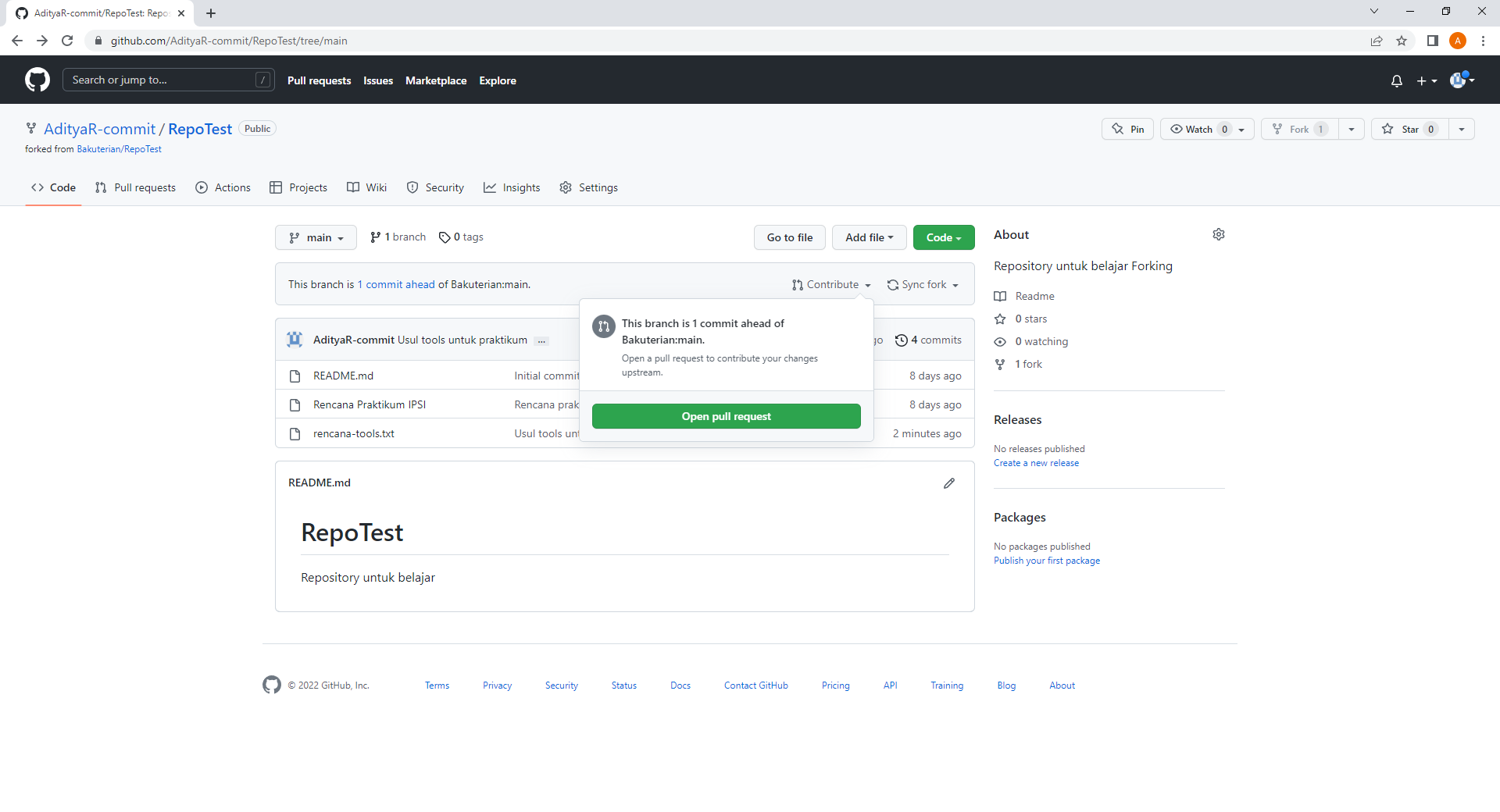
Gambar 69 Fork pada repo contoh

1. Tekan tombol **Create Fork** untuk konfirmasi aksi fork yaitu duplikasi snapshot dan history commit repo orisinal ke repo tujuan. Buat file pada repo hasil *forking* 🡪 simpan dan commit pada jalur utama.



Gambar 70 Hasil repo fork

1. Tekan tombol **Contribute –** **Open Pull Request** untuk meminta repo orisinal menyetujui usulan file baru yang telah dibuat dalam repo lokal.



Gambar 71 Pull request ke repo orisinal

1. Sistem akan menampilkan tombol **Contribute –** **Open Pull Request** untuk meminta repo orisinal menyetujui usulan file baru yang telah dibuat dalam repo lokal.

Pastikan bahwa repo yang menjadi *base repository* berada pada akun orisinal sedangkan *head repository* adapada repo akun kita, serta status merge antara repo orisinal dengan repo hasil *forking* memiliki status *Able to Merge*

.



Gambar 72 Konfirmasi repo untuk pull request

1. Tekan tombol **Create Pull Request**. Pada repo orisinal tekan tab Pull Request dan pilih request dari repo local kemudian klik **Merge Pull Request**.

Chapter

4

GIT REMOTE GITHUB

Remote merupakan website yang memiliki layanan utama untuk mengelola project GIT

P

ada bagian ini akan dijelaskan penggunaan GITHUB sebagai *version control system* serta pembuatan repository/repo dengan menggunakan GITHUB.

DAFTAR ISTILAH

|  |  |
| --- | --- |
| **repo** | **Folder dalam project** |
| **commit** | **Rekaman/snapshot dari repo** |
| **hash** | **Penanda unik pada sebuah commit** |
| **checkout** | **Berpindah ke sebuah commit** |
| **branch** | **Cabang dari commit** |
| **merge** | **Penggabungan branch** |
| **remote** | **Sumber yang didalamnya terdapat folder repository/repo** |
| **clone** | **Mengambil konten dari repository dari remote** |
| **push** | **Mengirimkan commit ke repo** |
| **pull** | **Mengambil commit dari repo** |
| **master** | **Nama default dari *main branch* pada GIT** |
| **head** | **Sebuah penanda atas *branch* yang sedang aktif pada GIT** |
| **Working tree** | **Wadah penyimpanan file yang akan dirubah di GIT** |
| **Staging area** | **Tempat sebuah file disimpan untuk selanjutnya dilakukan *commit*** |
| **Git directory** | **Tempat penyimpanan *snapshots* dari file yang telah *commit*** |

DAFTAR PERINTAH DI GIT

|  |  |
| --- | --- |
| **Git init** | **Mengubah folder menjadi *repository*/repo** |
| **Git status** | **Menampilkan status teraktual dari *folder* yang menjadi *repository*** |
| **Git add *nama\_file*** | **Menambahkan *file* dengan nama *nama\_file* kedalam *staging area*** |
| **Git add .** | **Menambahkan banyak *file* pada *repository* ke *staging area*** |
| **Git commit -m “*pesan*”** | **Melakukan *commit* yaitu membuat dan menyimpan snapshots dari perubahan dan memberikan pesan terhadap perubahan yang dijalankan** |
| **Git commit -a -m “*pesan*”** | **Melakukan penambahan file kedalam *staging area* bersamaan dengan menjalankan dan menambahkan pesan commit** |
| **Git restore** |  |
| **Git clone *path*** | **Menyalin *snapshot* kedalam folder *repository* di GIT** |
| **Git push** |  |
| **Git pull** |  |
| **Git ls-files** |  |
| **Git branch** | **Menambahkan branch pada GIT** |
| **Git merge** | **Menyatukan branch yang terpisah** |
| **Git checkout *nama\_branch*** | **Menentukan branch yang sedang aktif** |
| **Git log** | **Menampilkan catatan riwayat yang telah dijalankan** |
| **pwd** | **Menampilkan *path* dari *folder* yang menjadi area kerja** |