# **Daftar isi**

[**Daftar isi** i](#_Toc203468808)

[**Panduan Git** 1](#_Toc203468809)

[1. git –version 1](#_Toc203468810)

[2. pwd 2](#_Toc203468811)

[3. cd 2](#_Toc203468812)

[4. mkdir 2](#_Toc203468813)

[5. ls 3](#_Toc203468814)

[6. git init 3](#_Toc203468815)

[7. git config –global user.name “nama username git hub kamu” 4](#_Toc203468816)

[8. git config –global user.email “alamat email git hub kamu” 4](#_Toc203468817)

[9. git config –list 4](#_Toc203468818)

[10. touch 5](#_Toc203468819)

[11. code . 5](#_Toc203468820)

[12. git status 6](#_Toc203468821)

[13. git add 6](#_Toc203468822)

[14. git commit 7](#_Toc203468823)

[15. buat repositori baru di github 8](#_Toc203468824)

[16. git remote add 9](#_Toc203468825)

[17. git remote -v 9](#_Toc203468826)

[18. git branch -M 10](#_Toc203468827)

[19. git push 10](#_Toc203468828)

[20. git branch 11](#_Toc203468829)

[21. git branch (nama-branch-baru) 11](#_Toc203468830)

[22. git switch 12](#_Toc203468831)

[23. git checkout -b (nama-branch-baru) 12](#_Toc203468832)

[24. menambahkan perubahan di branch fitur-1 12](#_Toc203468833)

[25. Pull Request ke branch main via GitHub 14](#_Toc203468834)

[26. git pull 18](#_Toc203468835)

[27. git branch -d 19](#_Toc203468836)

[28. git push origin --delete (nama-branch) 20](#_Toc203468837)

[29. git merge 21](#_Toc203468838)

[30. git log 22](#_Toc203468839)

[31. git log --oneline 23](#_Toc203468840)

[32. git log -p 23](#_Toc203468841)

[33. git reset --soft HEAD~1 24](#_Toc203468842)

[34. git reset --mixed HEAD~1 25](#_Toc203468843)

[35. git reset --hard HEAD~1 26](#_Toc203468844)

[36. git revert HEAD 28](#_Toc203468845)

[37. rm 29](#_Toc203468846)

[38. git clone 30](#_Toc203468847)

[39. git stash 33](#_Toc203468848)

[40. git fetch 35](#_Toc203468849)

[41. git rebase (nama-branch) 37](#_Toc203468850)

[42. git stash list 37](#_Toc203468851)

[43. git stash drop stash@{nomor-stash} 38](#_Toc203468852)

[44. git stash pop 38](#_Toc203468853)

[45. git merge --squash (nama-branch) 41](#_Toc203468854)

# **Panduan Git**

## git –version

Memastikan Git sudah terinstal.



Jika muncul pesan versi Git seperti git version seperti gambar di atas, maka Git sudah terinstal dan siap digunakan, tetapi jika saat menjalankan perintah git --version muncul pesan seperti:

'git' is not recognized as an internal or external command, operable program or batch file.

Atau

git: command not found

Itu artinya Git belum terinstal di komputer kamu, atau sistem operasi belum mengenali Git sebagai perintah (belum terdaftar di environment variable PATH).

**Solusi: Install Git**

Silakan ikuti panduan resmi instalasi Git di halaman berikut:

<https://github.com/git-guides/install-git>

**Git Bisa Digunakan di CMD, PowerShell, dan VS Code**

Setelah Git berhasil diinstal, kamu **tidak harus selalu menggunakan Git Bash** untuk menjalankan perintah Git.

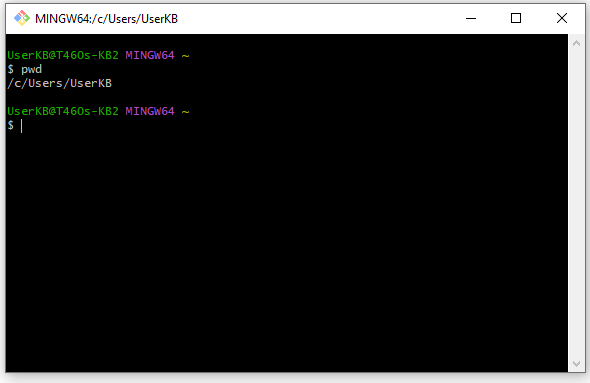
Git bisa dijalankan dari berbagai terminal, seperti:

* **Command Prompt (CMD)**
* **PowerShell**
* **Terminal di Visual Studio Code**

Yang penting, Git sudah dikenali oleh sistem (terdaftar di PATH). Kamu bisa mengetik git --version di terminal manapun untuk memastikannya.

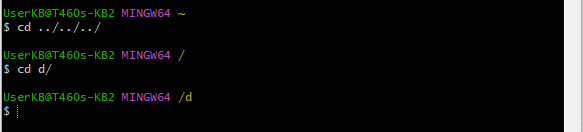
## pwd

untuk melihat posisi kita saat ini.



## cd

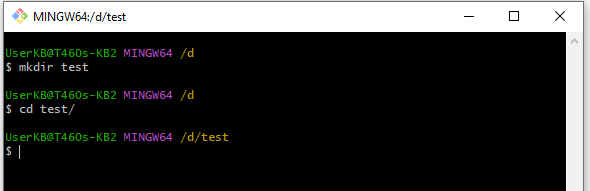
kemudian kita menggunakan cd untuk pindah direktori, cd .. untuk pindah ke direktori sebelumnya. Di sini saya pindah ke direktori d/



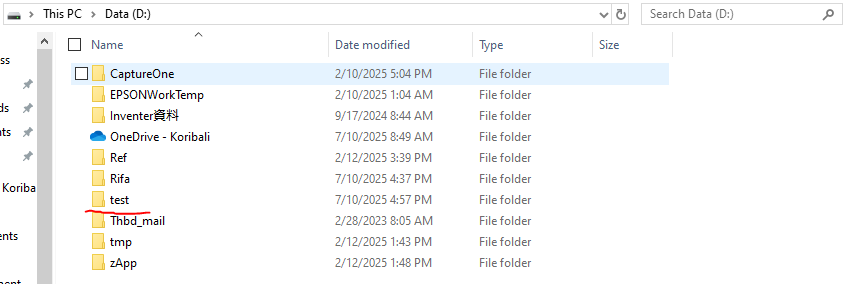
## mkdir

untuk membuat folder baru. Di sini saya membuat folder test. Kemudian menggunakan perintah cd untuk pindah ke folder yang baru saja saya buat.

***\*\*tampilan git bash\*\****



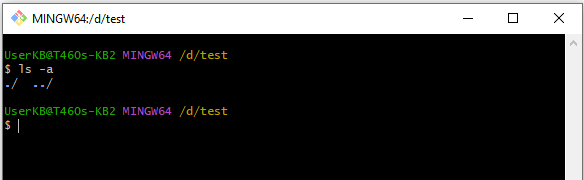
***\*\*tampilan file explorer\*\****



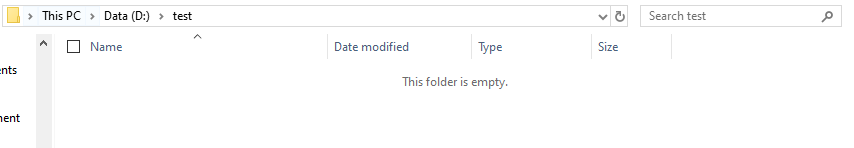
## ls

untuk menampilkan daftar isi file apa saja yang ada pada folder tersebut, di sini saya menggunakan ls -a untuk menampilkan semua file termasuk yang ter-hide.

***\*\*tampilan git bash\*\****

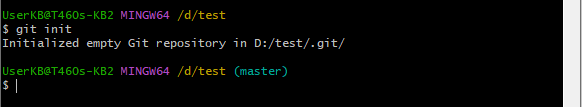


***\*\*tampilan file explorer\*\****

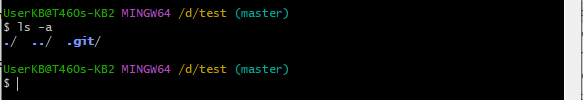


## git init

Ini akan membuat folder .git tersembunyi, tandanya folder sudah menjadi repositori Git.

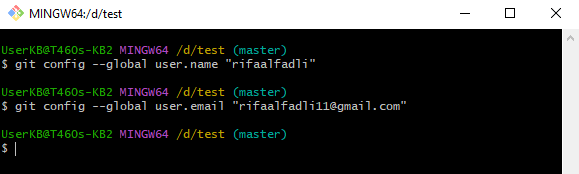


Kita juga bisa cek menggunakan ls -a, untuk cek apakah sudah terbuat atau belum.



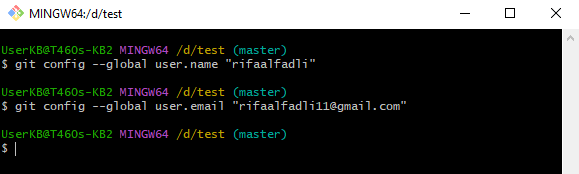
## git config –global user.name “nama username git hub kamu”

Sebelum mulai menggunakan Git, kamu harus mengatur identitasmu agar setiap commit tercatat dengan benar. Perintah ini akan menyimpan nama kamu secara global di komputer.



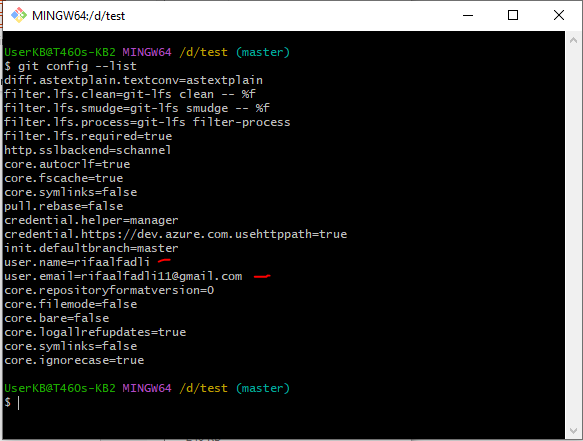
## git config –global user.email “alamat email git hub kamu”

Sebelum mulai menggunakan Git, kamu harus mengatur identitasmu agar setiap commit tercatat dengan benar. Perintah ini akan menyimpan nama email kamu secara global di komputer.



## git config –list

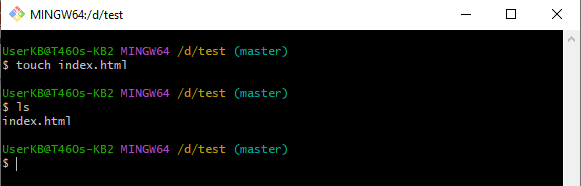
Gunakan git config --list untuk mengecek konfigurasi Git kamu.



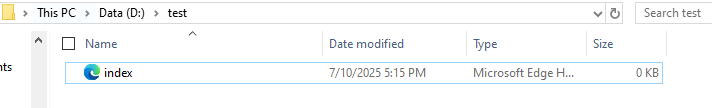
## touch

untuk membuat file baru, di sini saya membuat index.html. dan ketik perintah ls untuk menampilkan isi file dari folder test.

***\*\*tampilan git bash\*\****



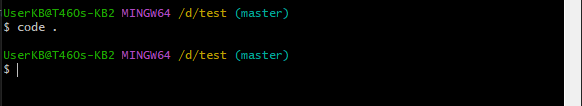
***\*\*tampilan file explorer\*\****



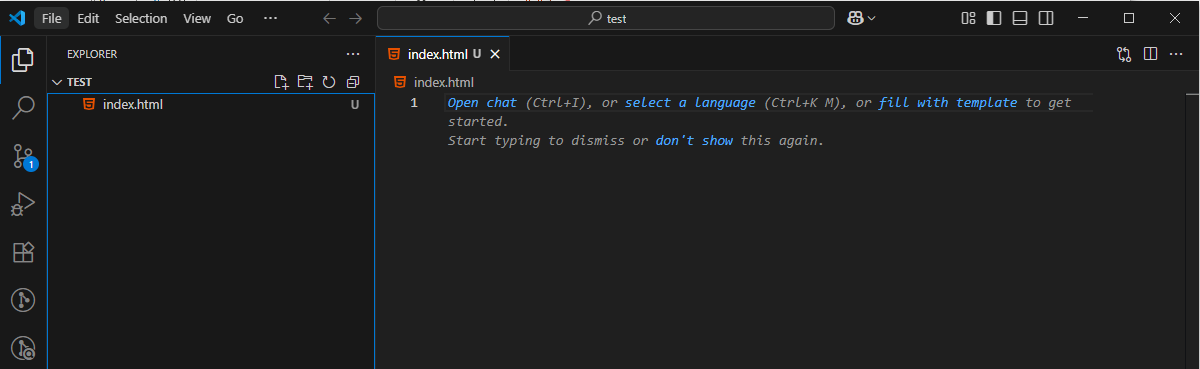
## code .

untuk buka folder test di vs code.

***\*\*tampilan git bash\*\****

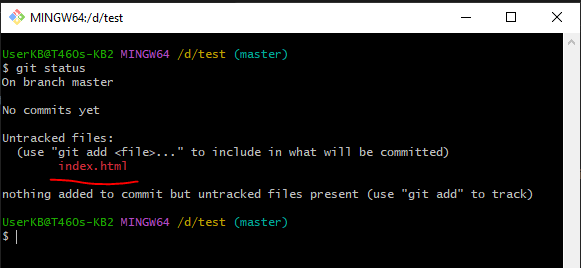


***\*\*tampilan vs code\*\****



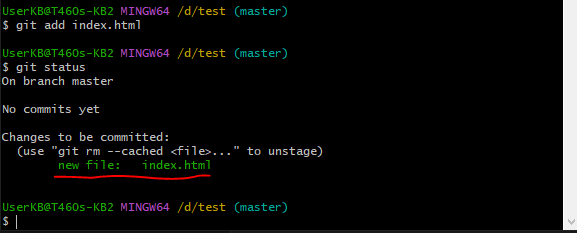
## git status

untuk melihat perubahan apa saja yang terjadi di file kamu. Karena sebelumnya saya menambahkan file baru yaitu index.html maka kita cek. Bisa kita lihat di sini index.html berwarna merah yang artinya menandakan file baru ditambahkan atau diubah, tapi belum kamu tambahkan ke staging area (belum git add).



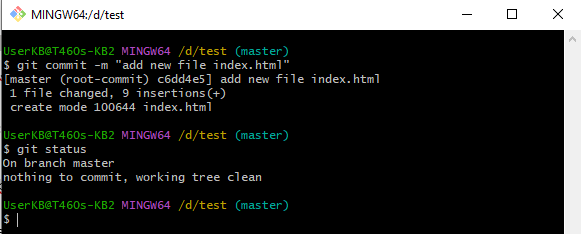
## git add

untuk tambahkan ke staging area kita tambahkan perintah git add (nama-file) atau bisa juga git add . (artinya semua file perubahan yang ada kita add). Kemudian saya ketik git status kembali untuk cek, karena file sebelumnya sudah di add maka dari itu index.html berubah menjadi warna hijau, artinya File tersebut sudah masuk ke *staging area* dan siap untuk di-commit.



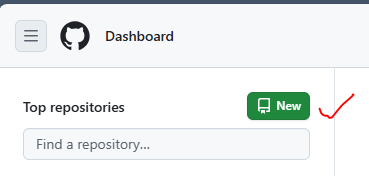
## git commit

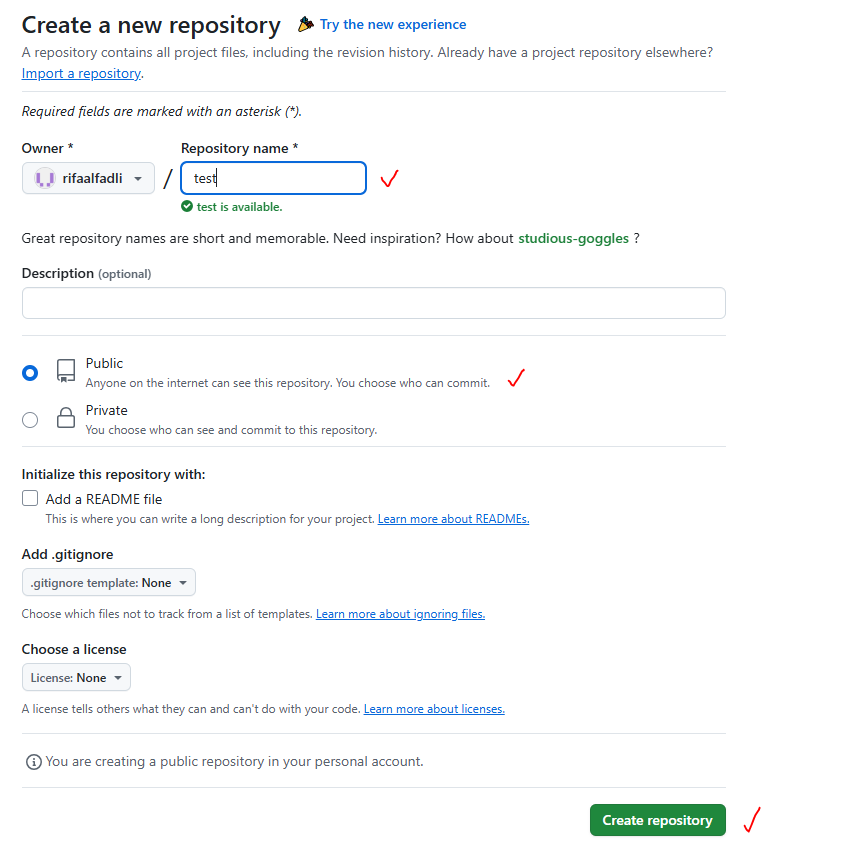
lalu commit dengan pesan yang menjelaskan perubahanmu. Lalu cek kembali dengan perintah git status, hasilnya semua perubahan sudah kamu commit, dan tidak ada file baru atau file yang diubah di direktori kerja kamu.



## buat repositori baru di github

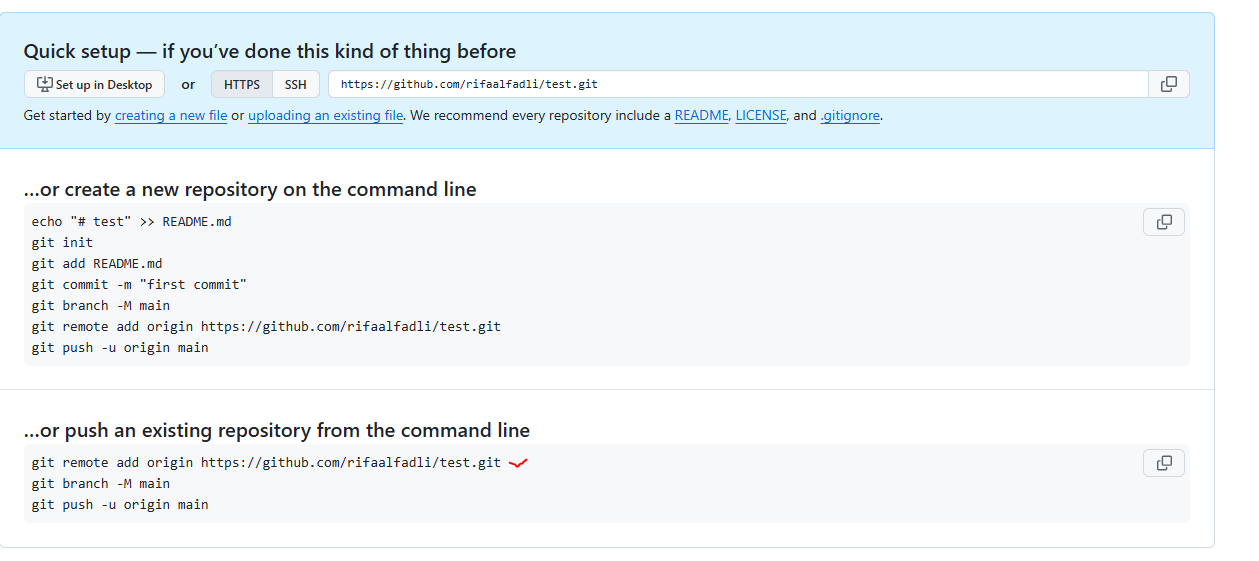
Setelah commit, kamu ingin menyimpan ke GitHub (remote). Pertama, sambungkan repositorimu dengan GitHub. **Caranya** : login github, klik new, lalu isi nama repositori, samakan dengan nama folder yang sudah saya buat di awal, yaitu “test”, opsional di sini saya buat public, kemudian klik create repository.



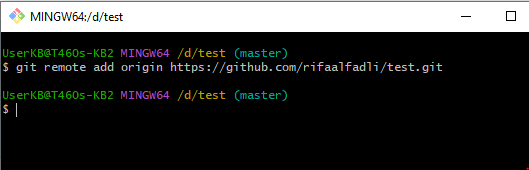


## git remote add

Perintah ini digunakan untuk menambahkan koneksi ke repository jarak jauh (remote). Ketika sudah dibuat salin perintah di baris pertama.

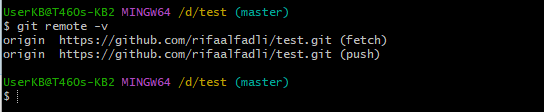


Kemudian paste perintah tadi ke git bash kamu.



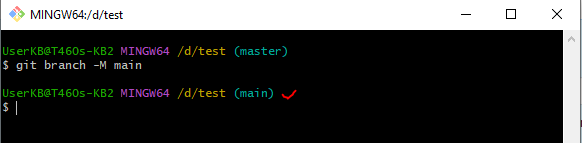
## git remote -v

digunakan untuk menampilkan daftar remote repository yang terhubung ke repository Git lokal kamu, lengkap dengan URL-nya.



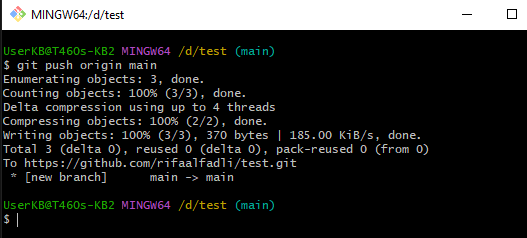
## git branch -M

Mengganti nama branch aktif saat ini menjadi main, secara paksa. Di sini saya mengganti branch utama master dengan main.



## git push

Mengirim (push) branch lokal main ke repository remote (GitHub) yang bernama origin.



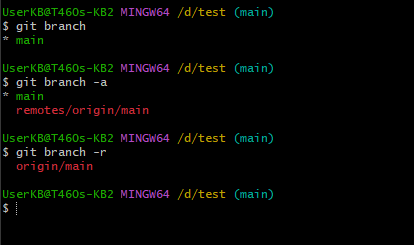
## git branch

***git branch***untuk menampilkan daftar isi branch yang ada di local kamu.

***git branch -a*** digunakan untuk menampilkan semua branch yang ada baik di local maupun repositori.

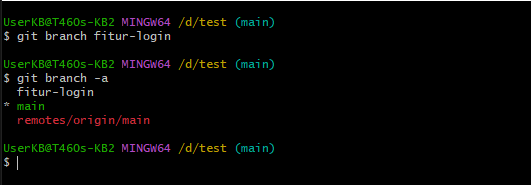
***git branch -r*** digunakan untuk menampilkan daftar isi branch yang ada di repositori.

(tanda bintang menunjukkan posisi kamu ada di branch mana)



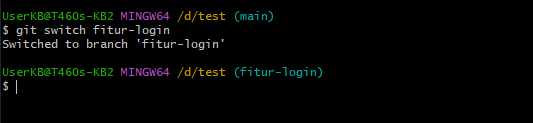
## git branch (nama-branch-baru)

Perintah git branch nama-branch-baru digunakan untuk membuat branch baru di Git, tanpa langsung berpindah ke branch tersebut. Di sini saya membuat branch baru bernama fitur-login. Lalu cek menggunakan perintah git branch -a, untuk menampilkan branch baru.



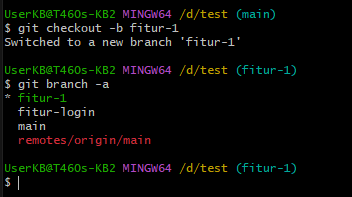
## git switch

Perintah git switch digunakan untuk berpindah branch. Di sini saya berpindah ke branch fitur-login.



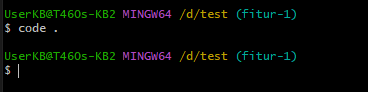
## git checkout -b (nama-branch-baru)

Dengan perintah ini, kita juga bisa membuat branch baru, dan langsung pindah ke branch baru tersebut. Di sini saya membuat branch baru dengan nama fitur-1. Cek juga dengan git branch -a untuk menampilkan semua branch yang ada.

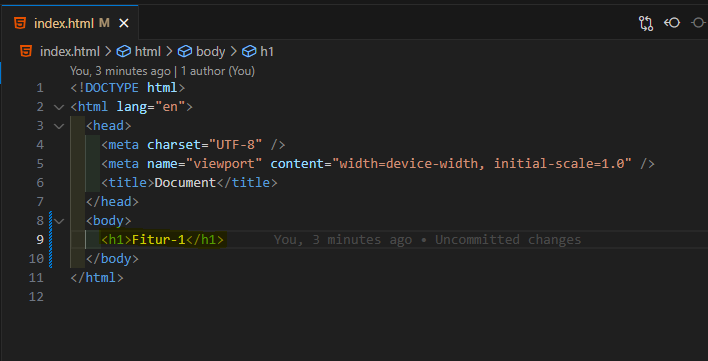


## menambahkan perubahan di branch fitur-1

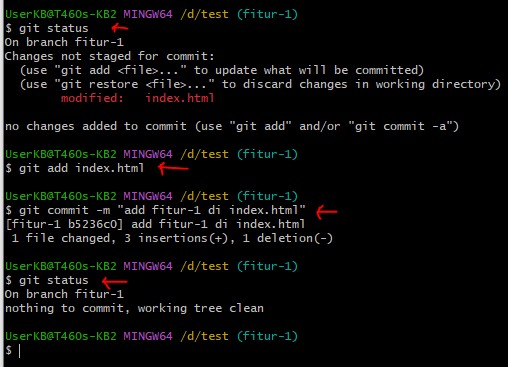
gunakan perintah code . untuk membuka folder test di vs code.



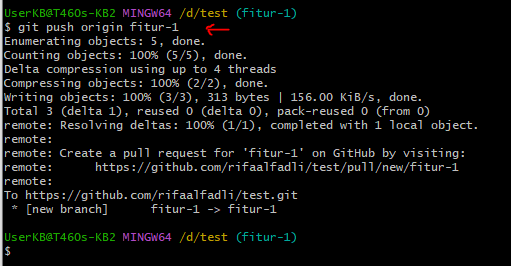
Kemudian tambahkan perubahan apapun.



Jika sudah, cek dengan git status, kemudian masukkan perintah git add, kemudian git commit, dan cek kembali dengan perintah git status.



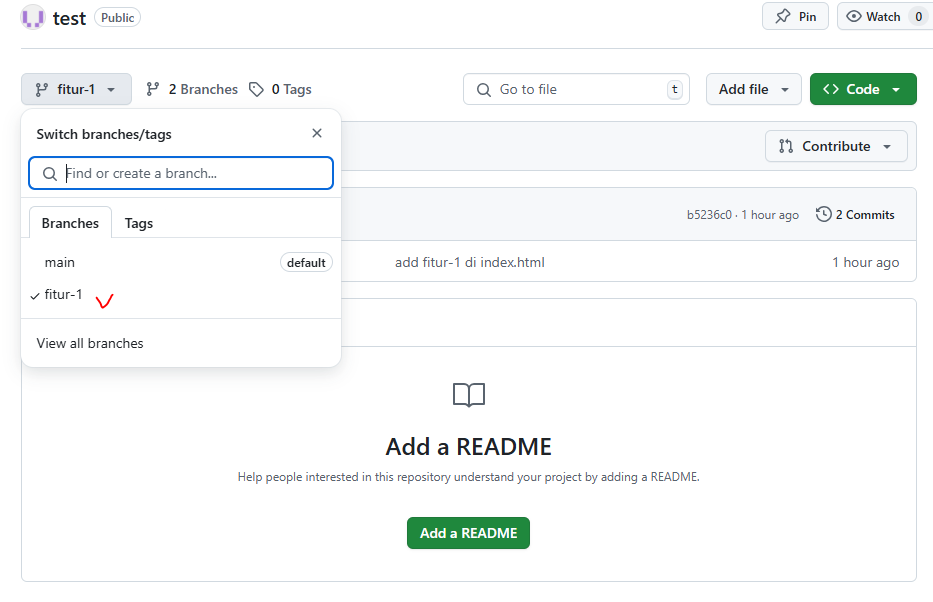
Masukkan perintah git push origin (nama-branch-yang-kamu-gunakan-saat-ini)



## Pull Request ke branch main via GitHub

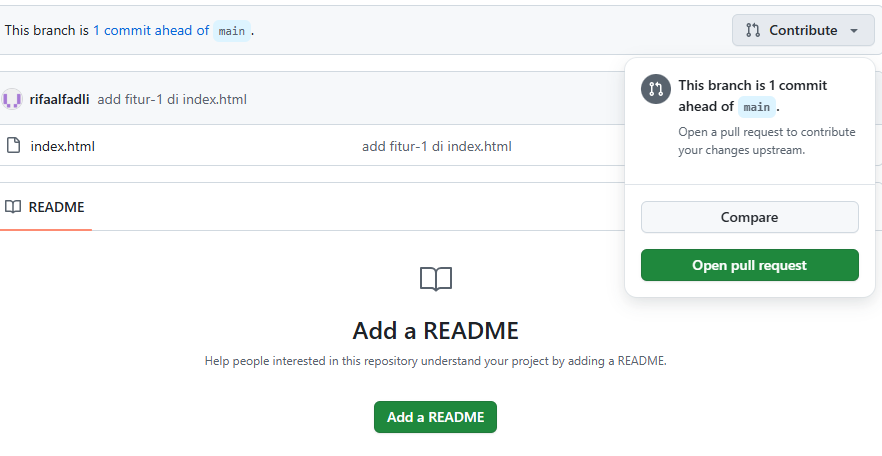
Pull Request (PR) adalah permintaan untuk menggabungkan perubahan dari satu branch ke branch lain, biasanya dari branch fitur (seperti fitur-1) ke branch utama (main), melalui GitHub.

Pertama kita buka GitHub, lalu klik branch, dan pilih branch yang ingin diajukan sebagai Pull Request (PR)

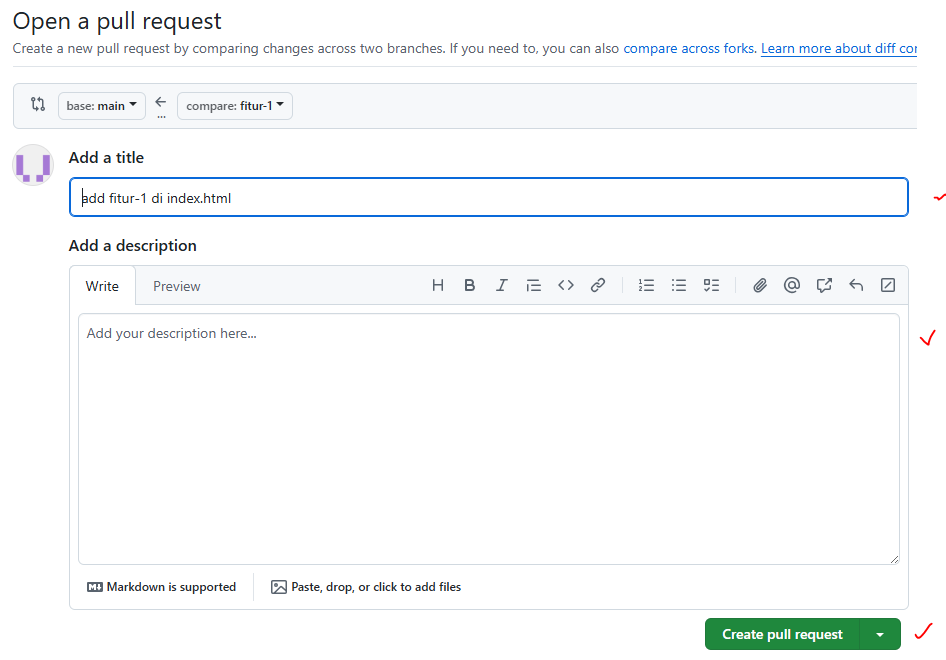


Klik Contribute, di sini ada dua pilihan yaitu compare dan open pull request.

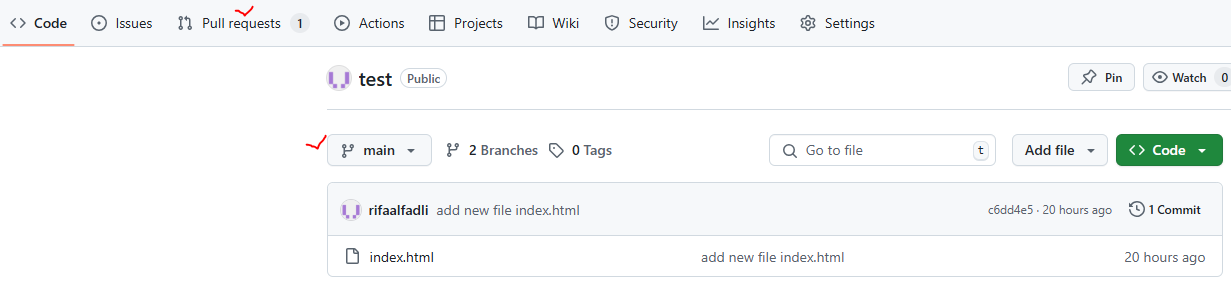
* Kalau kamu masih ingin **lihat apa aja sih bedanya file** antara dua branch → klik **Compare** dulu.
* Kalau kamu sudah yakin dan siap untuk **ajukan perubahan ke main**, langsung klik **Open pull request**.



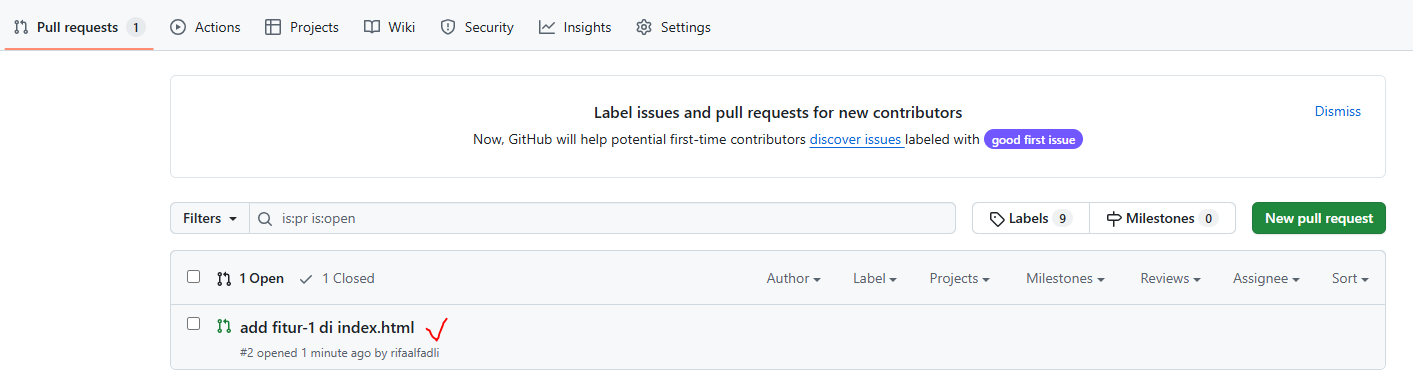
Ketika sudah klik Open pull request, maka akan tampil seperti gambar di bawah ini, kamu bisa menambahkan title, description, dan terakhir klik create pull request.



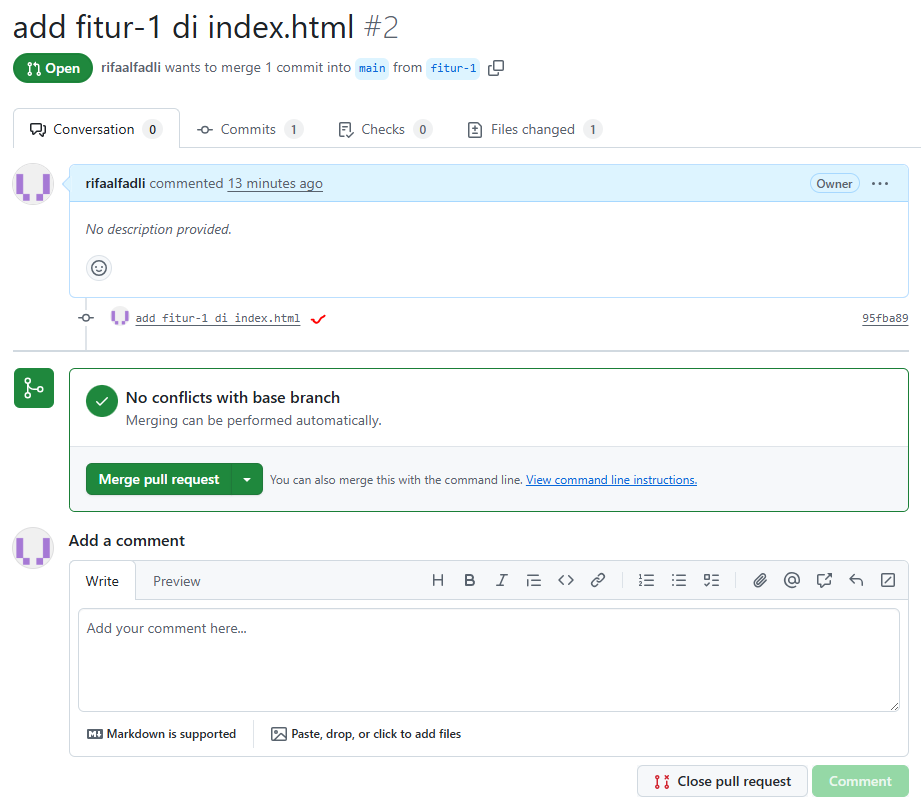
Masuk kembali ke branch main, dan nantinya akan ada notifikasi Pull Request.



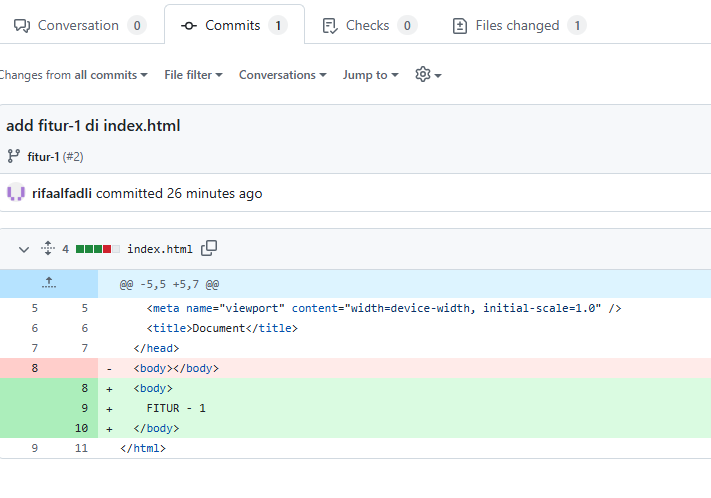
Ketika klik tab Pull request, akan muncul daftar isi yang menampilkan semua PR yang ada. Tinggal kita pilih dan klik pull request yang ingin kita periksa terlebih dahulu.



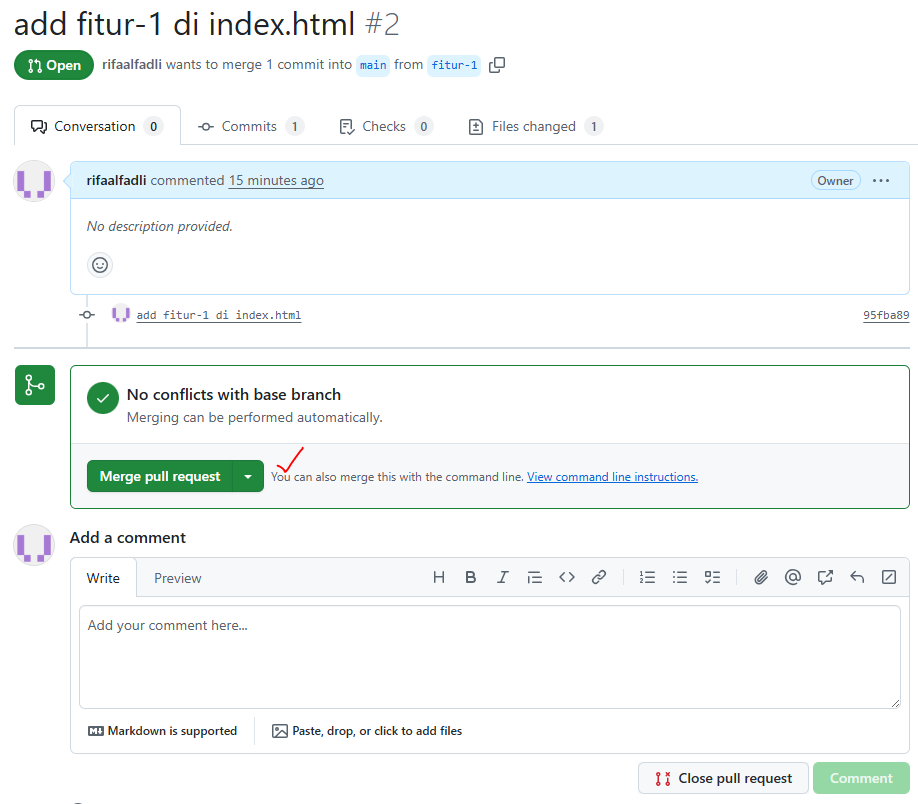
Kemudian akan menampilkan halaman seperti gambar dibawah ini, di sini kita bisa lihat perubahan apa saja yang dilakukan sebelum dimerge dengan klik link commit ini.



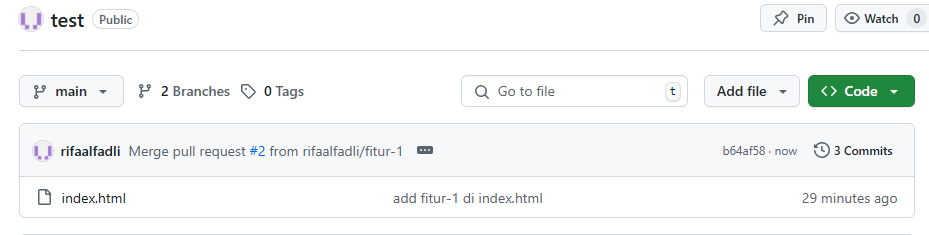
Bisa kita lihat perubahan apa saja yang terjadi.



Setelah semua sudah di rasa oke, kita bisa langsung klik Merge pull request.  
(di sini juga kita bisa menambahkan description dan comment)



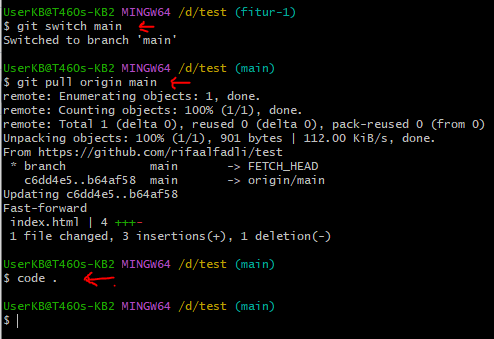
Perubahan sudah tersimpan di branch utama pada repositori GitHub.



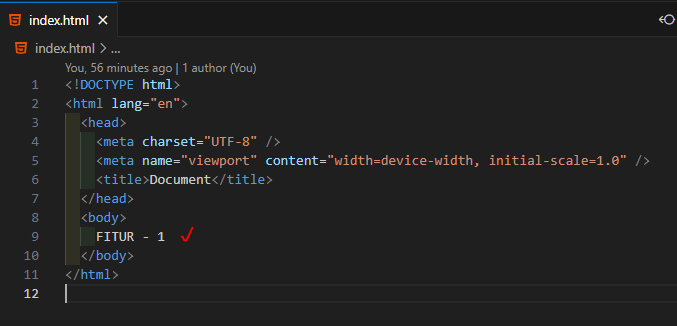


## git pull

Tarik (pull) semua perubahan terbaru dari branch main yang ada di remote (GitHub) ke branch lokal kamu saat ini. Sebelumnya ubah dulu posisi branch kita ke branch main. Untuk cek perubahan bisa buka vs code, dengan perintah code .

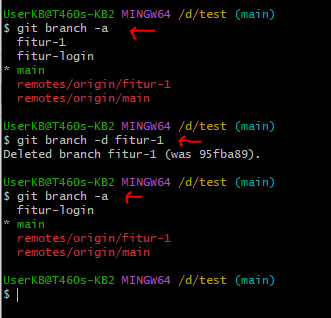


Ter-update:



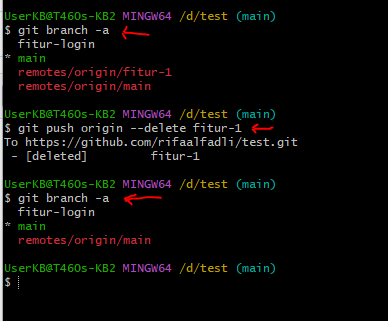
## git branch -d

perintah ini digunakan untuk hapus branch di lokal, kita bisa lihat dulu daftar isi branch kita dengan memasukkan perintah git branch -a, untuk melihat branch apa yang mau kita hapus (note: kamu *tidak boleh* berada di branch yang ingin dihapus).

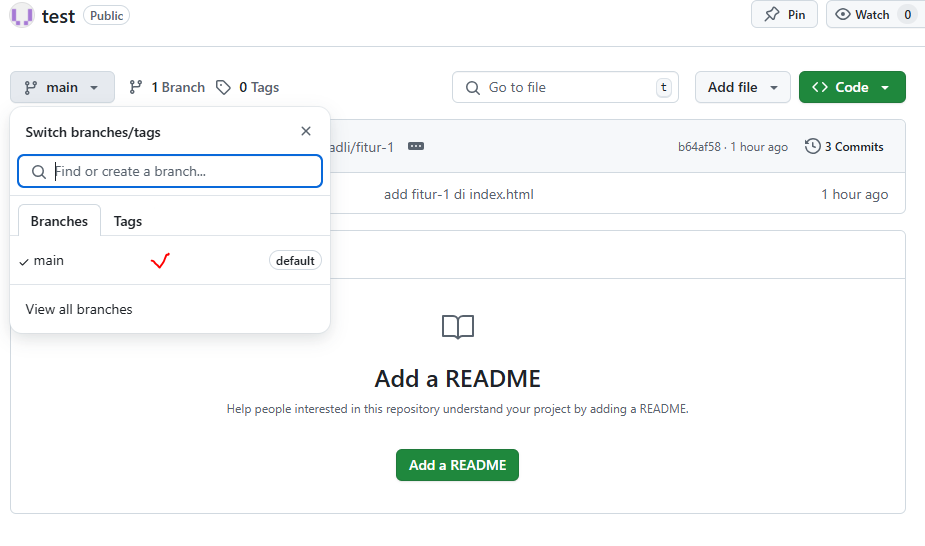


## git push origin --delete (nama-branch)

Untuk menghapus branch di GitHub (remote repo) lewat Git Bash.

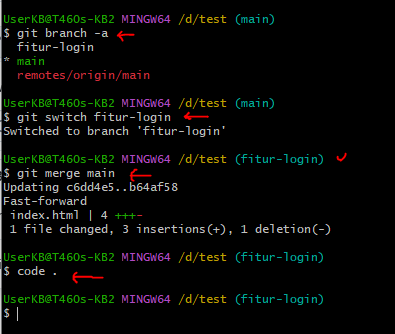


Cek di GitHub, bisa dilihat branch fitur-1 sudah tidak ada.

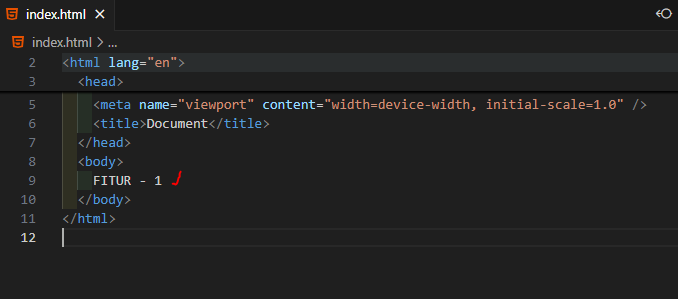


## git merge

Perintah ini bisa kita gunakan untuk menggabungkan isi terbaru dari branch main ke branch yang sedang aktif sekarang. Sebelum itu, saya menggunakan branch fitur-login yang sudah saya buat di awal, kita bisa cek daftar branch dengan git branch -a, kemudian kita pindah branch menggunakan git switch (nama-branch). Untuk cek kita bisa menggunakan perintah code . untuk melihatnya di vs code.

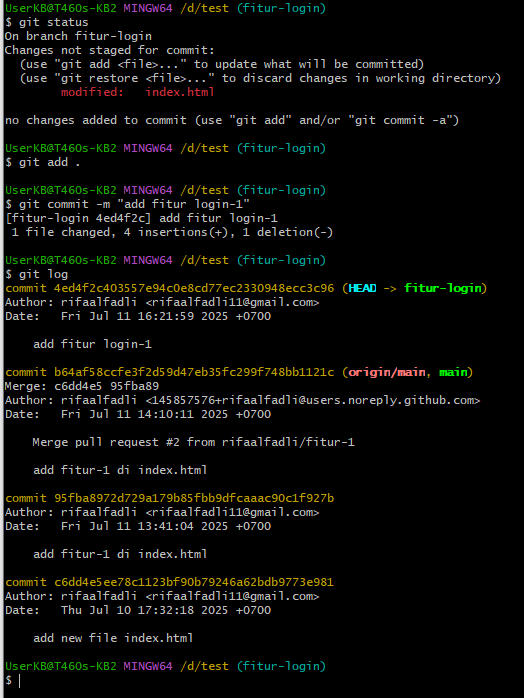


Tampilan vs code:



## git log

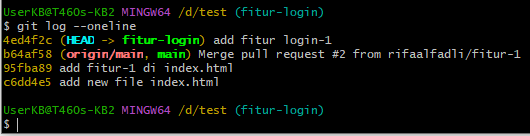
Fungsi git log adalah untuk melihat riwayat commit dalam sebuah repository Git. Pada gambar di bawah ini saya menambahkan perubahan di branch fitur-login, dengan menambahkan isi “fitur login – 1” di file index.html. kemudian lakukan seperti biasa yaitu, git status, git add . , git commit -m, kemudian cek riwayat commit dengan git log.



## git log --oneline

Perintah ini digunakan untuk melihat daftar riwayat commit secara ringkas, yaitu 1 baris per commit, hanya menampilkan:

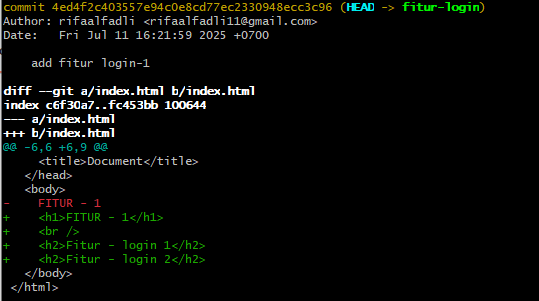
* ID commit singkat (hash pendek)
* Pesan commit



## git log -p

untuk tampilkan perubahan isi file tiap commit.





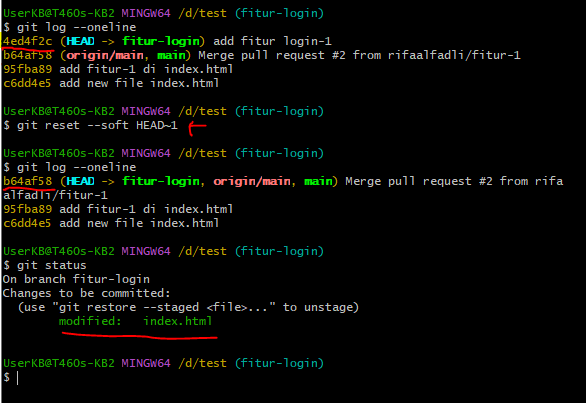
Untuk kembali ketik :wq

## git reset --soft HEAD~1

(Note: Perintah reset tidak direkomendasikan untuk rollback Ketika perubahan tersebut sudah di push ke remote GitHub) .

Perintah git reset --soft HEAD~1 digunakan untuk membatalkan commit terakhir, tapi isi file tetap dan masih dalam staging area (siap di-commit ulang). Pada gambar ini bisa dilihat Ketika menjalankan perintah tersebut, commit sebelumnya yang sudah di dicommit telah ter-reset. Cek dengan git status pun commit “add fitur login-1” kembali berwarna hijau/belum dicommit.

(Note: HEAD~1 artinya satu commit sebelum yang sekarang. HEAD~2, HEAD~3, dst = makin mundur ke belakang.)

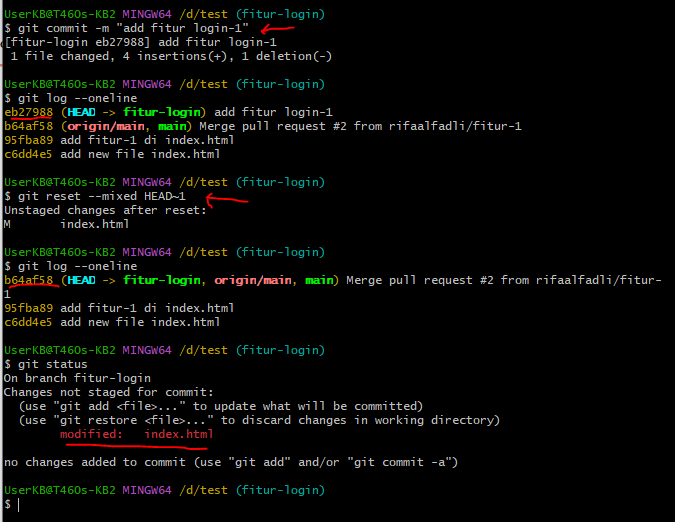


## git reset --mixed HEAD~1

(Note: Perintah reset tidak direkomendasikan untuk rollback Ketika perubahan tersebut sudah di push ke remote GitHub) .

Perintah git reset --mixed HEAD~1 digunakan untuk membatalkan commit dan juga menghapus dari staging (git add), tapi file tetap ada di folder kerja. Pada gambar di bawah ini saya lakukan commit ulang, dan kemudian menjalankan perintah git reset --mixed HEAD~1.

(Note: HEAD~1 artinya satu commit sebelum yang sekarang. HEAD~2, HEAD~3, dst = makin mundur ke belakang.)

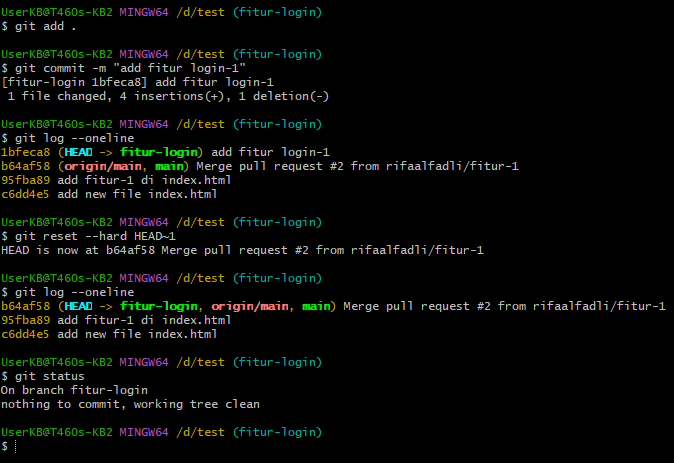


## git reset --hard HEAD~1

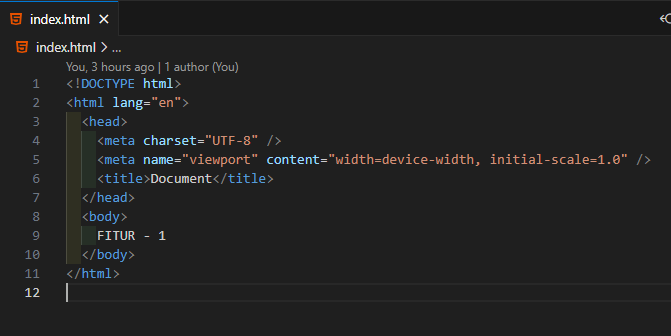
(Note: Perintah reset tidak direkomendasikan untuk rollback Ketika perubahan tersebut sudah di push ke remote GitHub) .

Perintah git reset --hard HEAD~1 digunakan untuk membatalkan commit dan menghapus semua perubahan, termasuk file yang sudah diubah. Pada gambar di bawah ini saya lakukan add dan commit ulang, dan kemudian menjalankan perintah git reset --hard HEAD~1. Hasilnya git status pun bersih artinya tidak ada perubahan sama sekali. Terakhir cek menggunakan code .,apakah benar-benar hilang.

(Note: HEAD~1 artinya satu commit sebelum yang sekarang. HEAD~2, HEAD~3, dst = makin mundur ke belakang.)



Bisa dibandingkan dengan gambar di No. 32, semua perubahan sudah dihapus.

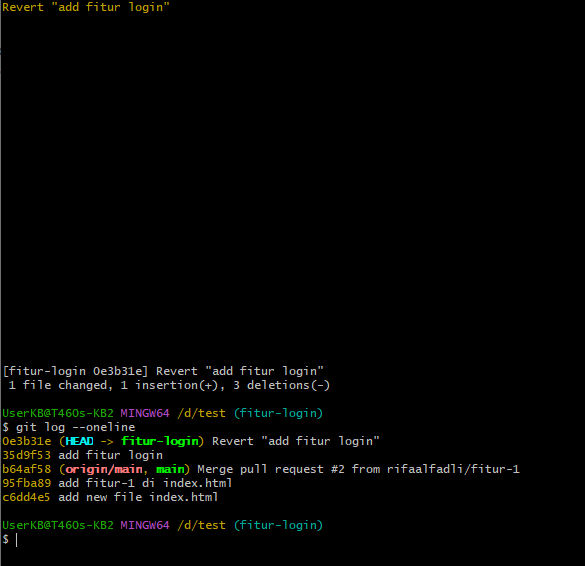


## git revert HEAD

Revert aman digunakan walaupun commit-nya sudah di-push, karena tidak menghapus riwayat, hanya menambah commit pembalik.

Perintah git revert HEAD digunakan untuk membuat commit baru yang membatalkan efek dari commit terakhir.

Berbeda dengan reset, revert ini tetep menyimpan commit yang lama, dan membuat commit baru dengan kondisi perubahan pada commit yang sebelumnya itu tidak ada.

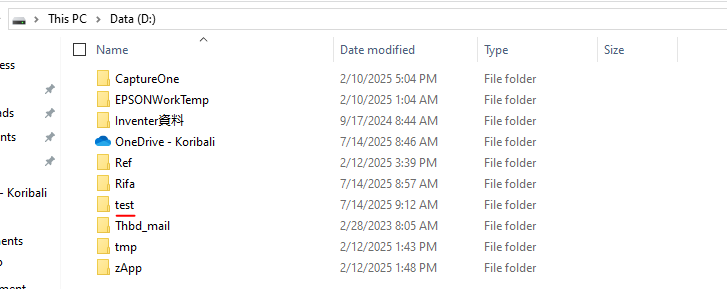


## rm

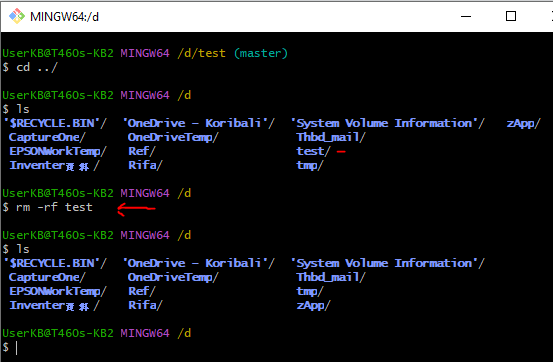
Perintah **rm – rf** digunakan untuk menghapus folder dan isinya secara rekursif tanpa konfirmasi apa pun, bahkan jika ada file yang dilindungi. (langsung dihapus secara paksa dan tidak masuk recycle bin.)

Jika ingin lebih aman, bisa menggunakan perintah **rm -r**, yaitu menghapus folder dan isinya secara rekursif, **tapi akan memberi peringatan atau meminta konfirmasi jika tidak punya izin**.

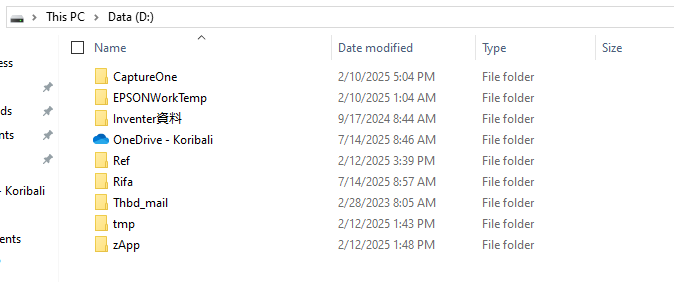
Di sini saya ingin menghapus folder test, dengan menggunakan rm-rf. Berikut tampilan file explorer sebelum folder test dihapus.



Berikut adalah tampilan git bash, pada saat saya menjalankan perintah rm -rf kepada folder test.



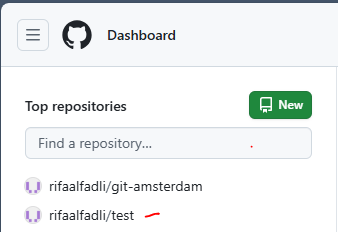
Berikut tampilan file explorer Ketika perintah rm -rf sudah dijalankan, dapat dilihat folder test sudah tidak ada.

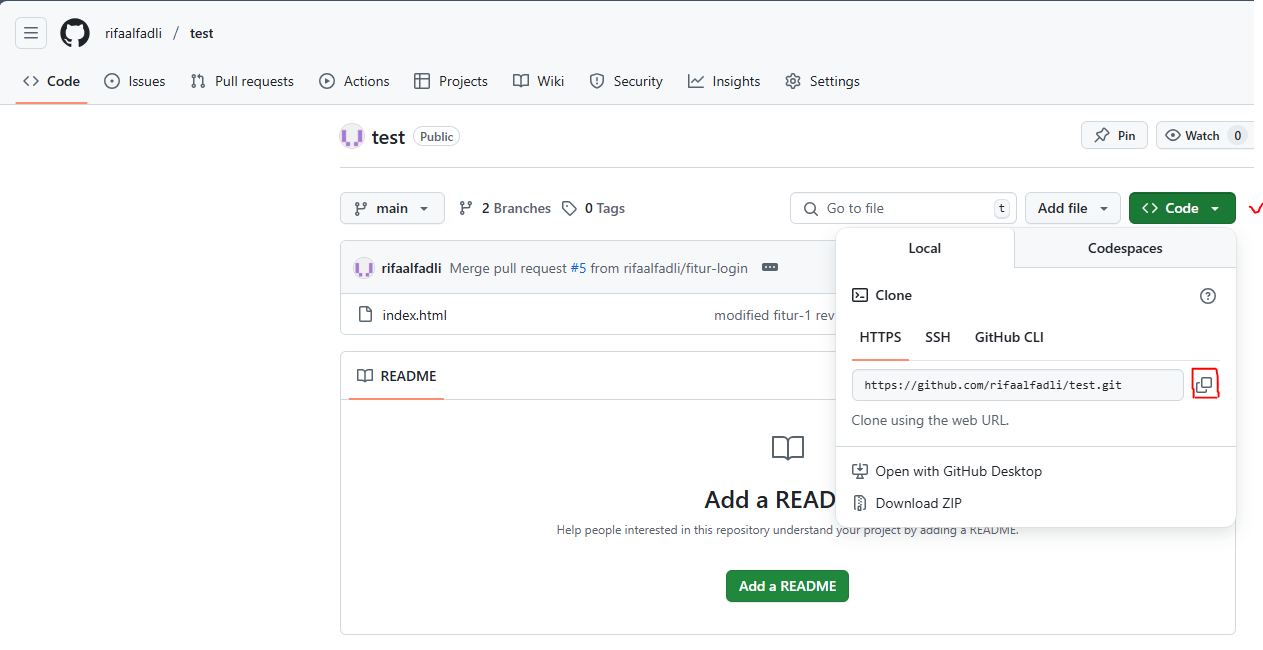


## git clone

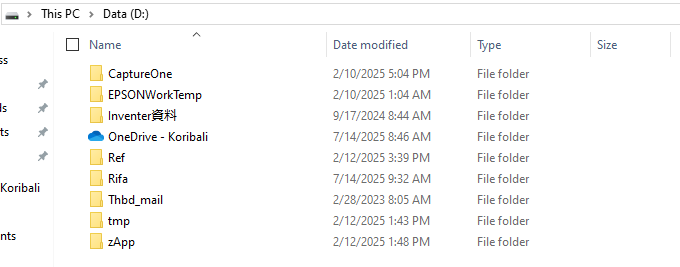
git clone adalah perintah untuk mengambil (mengunduh) salinan lengkap dari repository Git yang ada di remote (seperti GitHub) ke komputer lokal kamu. Di sini saya ingin mengambil kembali folder test yang sebelumnya sudah saya hapus. Karena folder tersebut sudah di-push ke GitHub sebagai repository, saya bisa mengunduhnya kembali menggunakan perintah git clone.

Pertama, buka git hub, di menu Home terdapat tampilan Top repositories, yang menampilkan semua daftar isi repositori yang ada, lalu pilih repositori yang ingin kita clone atau ambil

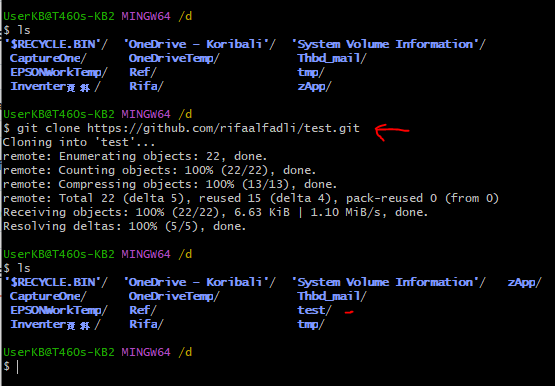


Kemudian, klik tombol Code, lalu salin URL HTTPS dari repository tersebut.

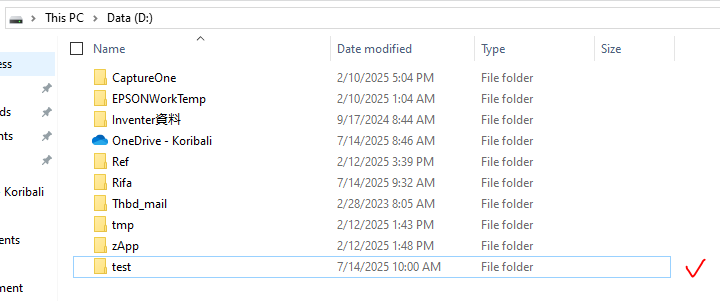
Berikut adalah tampilan File Explorer sebelum menjalankan perintah git clone, di mana folder test belum ada.



Selanjutnya, buka Git Bash dan jalankan perintah git clone diikuti dengan URL HTTPS yang telah disalin sebelumnya.



Berikut adalah tampilan File Explorer setelah menjalankan perintah git clone, di mana folder Folder hasil clone akan muncul di direktori kerja, dan sudah berisi file serta riwayat commit dari repositori asal.



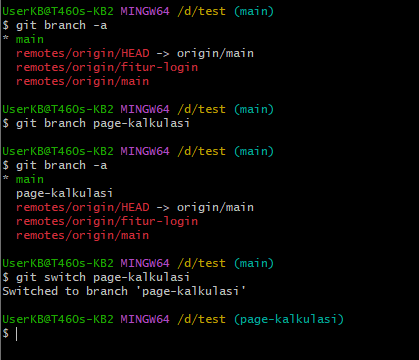
## git stash

git stash digunakan untuk menyimpan sementara perubahan kerja, agar kamu bisa berpindah branch atau mengerjakan hal lain tanpa kehilangan perubahan tersebut. Perubahan yang disimpan tidak akan ikut di-commit, dan bisa dikembalikan nanti dengan git stash pop atau git stash apply.

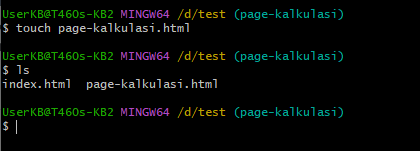
Jika **kamu mengubah file yang sudah pernah di-track Git**, kamu **bisa langsung git stash tanpa git add.**

Namun, jika **kamu membuat file baru (belum pernah di-commit atau di-add**), maka **kamu perlu menjalankan git add . terlebih dahulu**, atau gunakan **opsi git stash -u agar file baru tersebut ikut disimpan.**

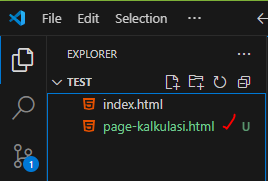
Pertama, saya membuat branch baru bernama page-kalkulasi pada repository yang baru saja saya clone menggunakan Git Bash.



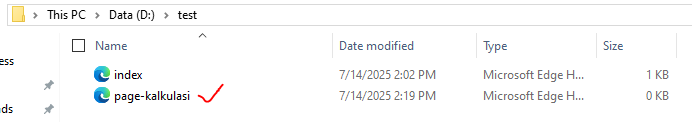
Kemudian, saya membuat file HTML baru dengan nama page-kalkulasi.html.



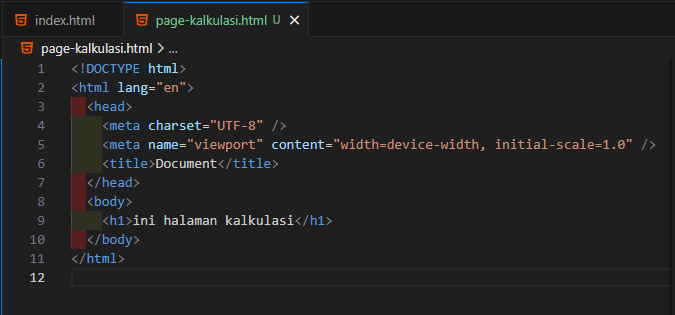
Berikut tampilan di vs code, setelah file page-kalkulasi.html ditambahkan.



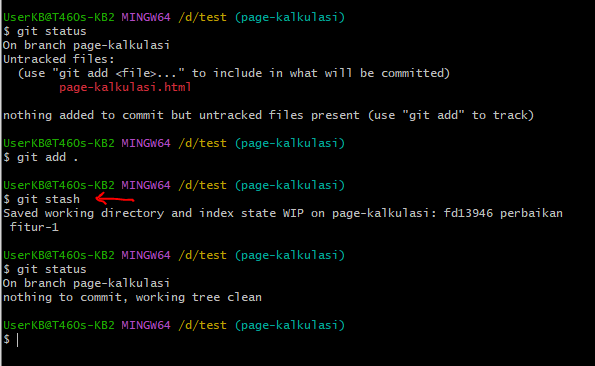
Berikut tampilan di file explorer, setelah file page-kalkulasi.html ditambahkan.



Selanjutnya, saya menambahkan isi kode ke dalam file page-kalkulasi.html.



Buka kembali Git Bash, jalankan git add . untuk menambahkan file baru, lalu gunakan git stash untuk menyimpan perubahan. Setelah itu, jika dicek dengan git status, hasilnya akan bersih karena tidak ada perubahan yang perlu di-commit.



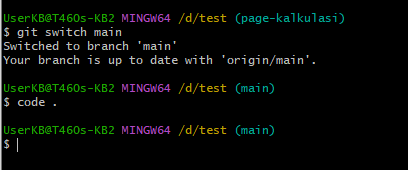
## git fetch

git fetch digunakan untuk mengambil update terbaru dari remote repository (seperti GitHub) tanpa langsung menggabungkannya ke branch lokal.

Secara default, **perintah git fetch origin akan mengambil semua branch dari remote bernama origin**.

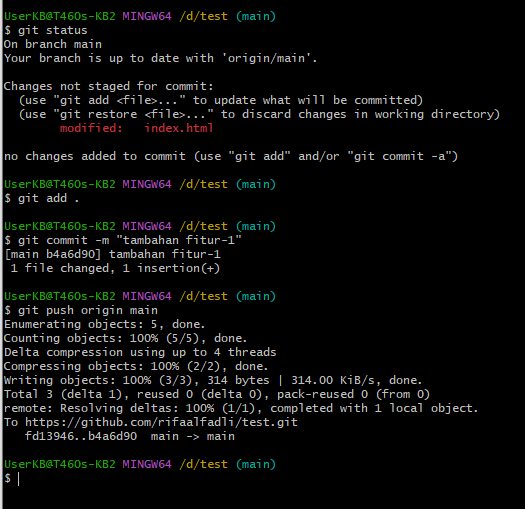
Kalau **ingin mengambil branch tertentu saja**, kamu bisa **gunakan git fetch origin nama-branch.** Setelah fetch, kamu bisa memilih untuk menggabungkan update tersebut menggunakan git merge atau git rebase.

Melanjutkan proses sebelumnya, setelah menjalankan git stash, saya berpindah ke branch main karena ada kebutuhan mendesak. Perubahan di branch page-kalkulasi saya simpan terlebih dahulu menggunakan stash, lalu melakukan perubahan di branch main.

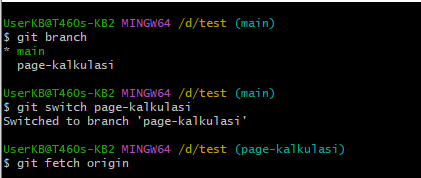




Setelah melakukan perubahan, jalankan perintah git add ., git commit -m, dan git push seperti biasa untuk menyimpan dan mengirim perubahan ke remote repository.



Selanjutnya, pindah ke branch page-kalkulasi menggunakan git switch, lalu jalankan git fetch origin untuk mengambil update terbaru dari remote bernama origin.

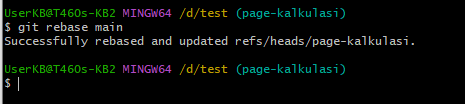


## git rebase (nama-branch)

git rebase digunakan untuk memindahkan basis commit dari suatu branch ke commit terbaru di branch lain, seolah-olah branch tersebut dibuat dari versi terbaru.

Ini menjaga riwayat commit tetap bersih dan linear, berbeda dengan git merge yang menciptakan commit gabungan. Biasanya digunakan saat kamu ingin sinkronisasi branch fitur dengan update terbaru dari main tanpa membuat riwayat bercabang.

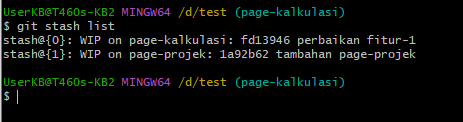
Melanjutkan proses sebelumnya setelah melakukan fetch, langkah selanjutnya adalah menggabungkan update dari branch main. Di sini, saya menggunakan git rebase main untuk menyinkronkan branch aktif dengan perubahan terbaru dari main.



## git stash list

git stash list digunakan untuk melihat daftar **semua perubahan yang pernah disimpan sementara menggunakan git stash.**

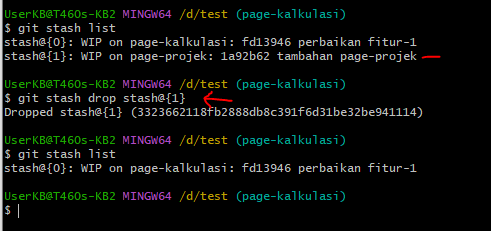
Perintah ini akan menampilkan urutan stash seperti stash@{0}, stash@{1}, dan seterusnya, lengkap dengan informasi branch asal dan pesan singkatnya. Ini sangat berguna untuk melacak perubahan apa saja yang pernah kamu simpan dan memilih stash mana yang ingin kamu pulihkan atau hapus.



## git stash drop stash@{nomor-stash}

Perintah git stash drop stash@{nomor-stash} digunakan untuk **menghapus stash tertentu dari daftar stash**.

Pada gambar di bawah ini, saya ingin menghapus stash@{1}.

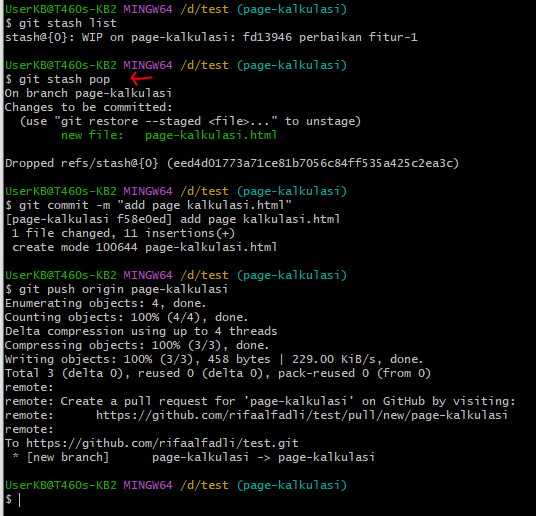


## git stash pop

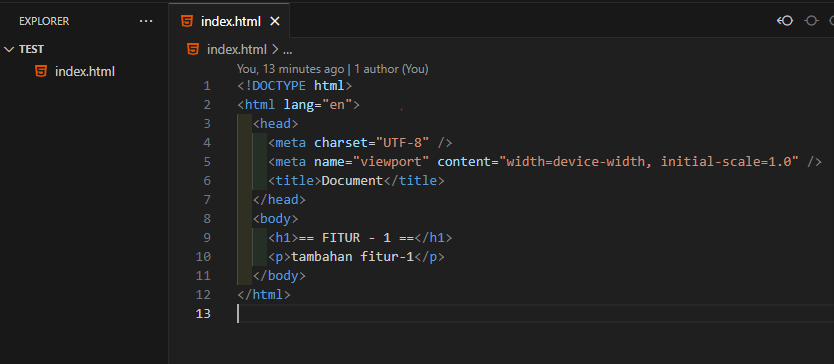
git stash pop digunakan untuk **mengambil dan menerapkan perubahan terakhir yang disimpan dengan stash**, lalu **menghapusnya dari daftar stash**.

Sementara itu, git stash apply juga menerapkan stash ke direktori kerja, tapi **tidak menghapus stash tersebut**, sehingga masih bisa digunakan lagi. **(git stash apply stash@{nomor-stash})**

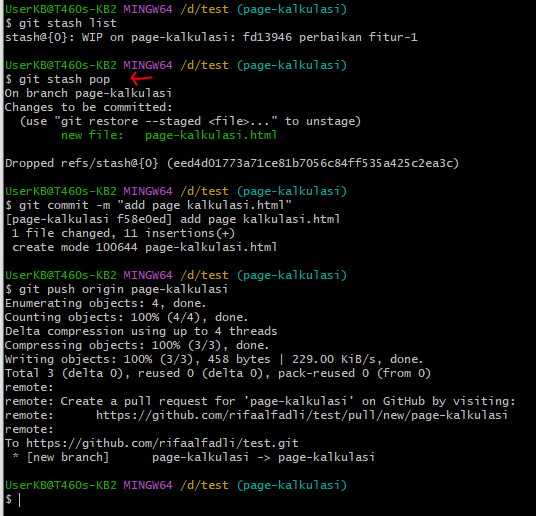
Di sini saya menggunakan git stash pop untuk mengambil dan menerapkan perubahan terakhir yang sebelumnya disimpan dengan stash. Hasilnya dapat terlihat, di mana perubahan muncul kembali namun belum di-commit. Nama file ditandai dengan warna hijau, yang berarti file sudah masuk ke staging area (sudah di-*add*).



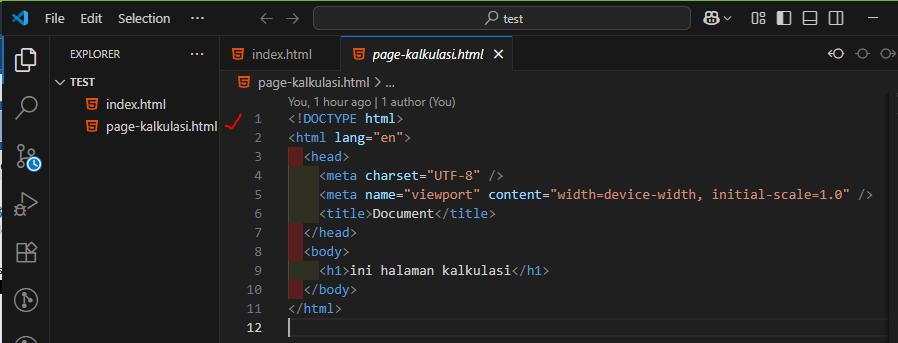
Berikut adalah tampilan di VS Code sebelum file page-kalkulasi.html di-commit.



Setelah menjalankan perintah git stash pop, karena perubahan belum di-commit, saya melakukan git commit dan git push untuk menyimpan dan mengirim perubahan tersebut ke remote repository pada branch page-kalkulasi.



Berikut adalah tampilan di VS Code sebelum file page-kalkulasi.html di-commit.



## git merge --squash (nama-branch)

git merge --squash (nama-branch) digunakan untuk menggabungkan semua perubahan dari branch tertentu ke branch aktif, namun hanya menjadi satu commit. Riwayat commit dari branch tersebut tidak ikut dibawa. Setelah menjalankan perintah ini, **kamu perlu menjalankan git commit secara manual untuk menyimpan hasil penggabungan.**

Melanjutkan proses sebelumnya, setelah perubahan pada branch page-kalkulasi berhasil di-*push* ke repository, saya menggabungkannya ke branch utama (main). Langkah pertama adalah berpindah ke branch main menggunakan git switch main, lalu menjalankan perintah git merge --squash page-kalkulasi. Terakhir, saya menjalankan git commit -m untuk menyimpan hasil penggabungan tersebut sebagai satu commit.

