**LAPORAN PRAKTIKUM**

**REKAYASA PERANGKAT LUNAK**

**Judul : Git Lanjut Bagian 3 dan Git Lanjut Bagian 4**



**Disusun Oleh:**

**Alfath Roziq Widhayaka – L0122012**

**PROGRAM STUDI INFORMATIKA  
FAKULTAS INFORMASI DAN SAINS DATA  
UNIVERSITAS SEBELAS MARET**

**2023**

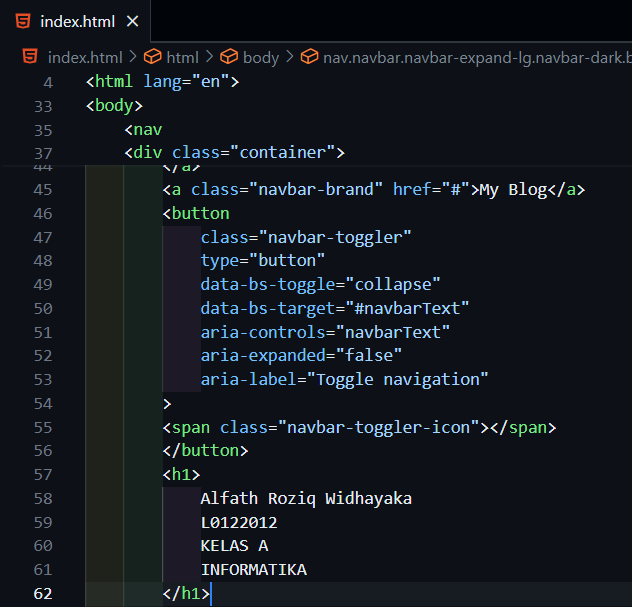
1. **Git Lanjut Bagian 3**
2. **Perintah Untuk Membatalkan Revisi**

Pada praktikum sebelumnya, sudah belajar cara melihat perbedaan di setiap revisi. Sekarang akan belajar, cara membatalkan sebuah revisi. Terkadang pada perubahan yang dilakukan terjadi kesalahan dan ingin mengembalikannya seperti keadaan sebelumnya. Maka perlu menyuruh git untuk megembalikannya. Ada beberapa perintah yang digunakan diantaranya : **git checkout, git reset,** dan **git revert.**

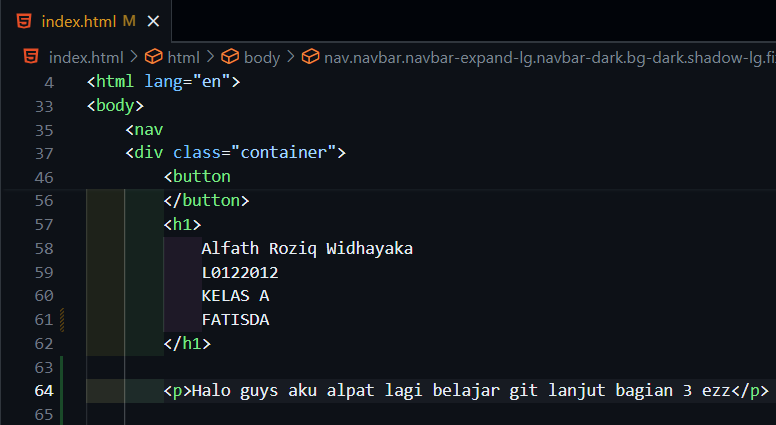
1. **Membatalkan perubahan**

Jika revisi belum staged ataupun committed, bisa mengembalikannya menggunakan perintah **git checkout nama\_file.html**. Contoh: Misalkan saya akan merubah isi dari file index.html pada repositori proyek-01.

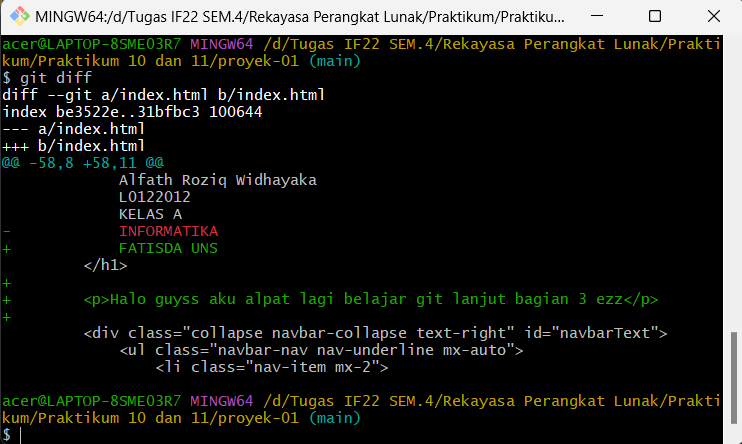
Sebelum diubah :



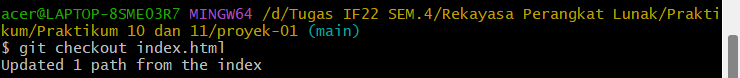
Setelah diubah :



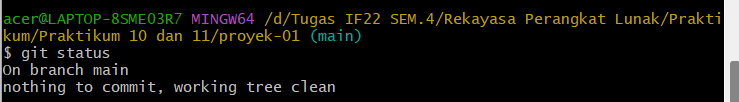
Hasil git diff :



Berdasarkan perintah **git diff** di atas dapat dilihat perbedaan isi file index.html. Tulisan berwarna merah diganti dengan tulisan berwarna hijau dalam file index.html. Sekarang akan membatalkan perubahan tersebut. Karena belum melakukan stage dan commit, maka bisa menggunakan perintah **git checkout index.html** :

****

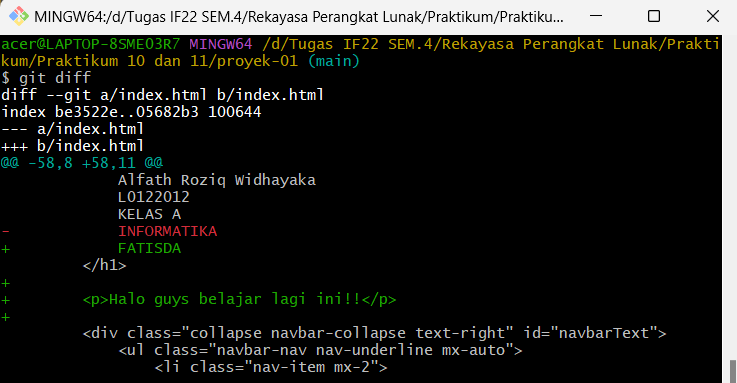
Perubahan yang baru saja lakukan akan dibatalkan. Untuk mengeceknya, silahkan coba periksa file yang sudah dirubah tadi atau cek dengan perintah **git status**.



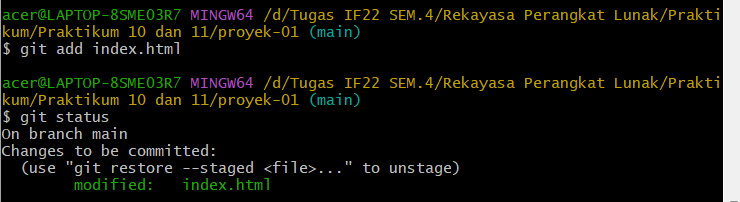
Hati-hati! Terkadang perintah ini sangat berbahaya, karena akan menghapus perubahan yang baru saja dilakukan. Bila sudah merubah banyak hal, maka itu akan sia-sia setelah menjalankan perintah ini

1. **Membatalkan perubahan file yang sudah dalam kondisi staged**

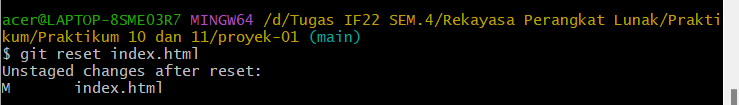
Kondisi staged merupakan kondisi file yang sudah di add (**git add**), namun belum disimpan (**git commit**) ke dalam Git. Sebagai contoh, lakukan perubahan lagi di file index.html seperti pada contoh sebelumnya.



Setelah itu, ubah kondisi file menjadi staged dengan perintah **git add index.html**



Nah, file index.html sudah masuk ke dalam kondisi staged. Untuk mengubahnya menjadi kondisi **modified**, bisa menggunakan perintah **git reset index.html**.

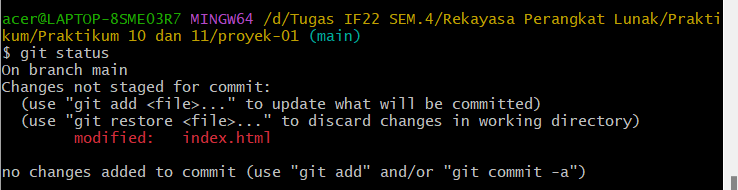


Setelah dilakukan perintah **git reset** maka file index.html yang sudah berada di kondisi staged kini berubah lagi dan kembali ke dalam kondisi modified yang dilambangkan dengan huruf M. Perintah **git reset** sukses dilakukan dengan diikuti output berikut.

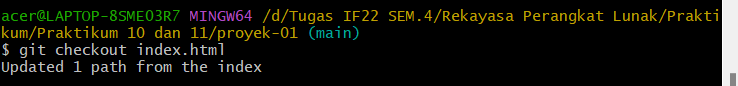
**Unstaged changes after reset:**

**M index.html**

Cek statusnya lagi :



Sekarang file index.html sudah dalam kondisi modified, bisa membatalkan perubahannya dengan perintah **git checkout** seperti contoh sebelumnya.

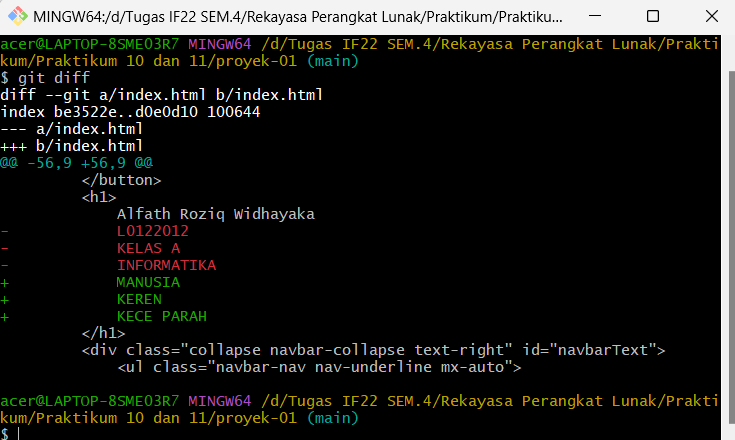


Maka perubahan yang lakukan akan dibatalkan. Perintah **git checkout** digunakan untuk membatalkan revisi atau perubahan pada file yang belum masuk ke dalam tahap staging. Dengan dijalankan perintah **git checkout index.html** maka perubahan yang sebelumnya dilakukan akan dibatalkan dan isi file akan sama seperti semula.

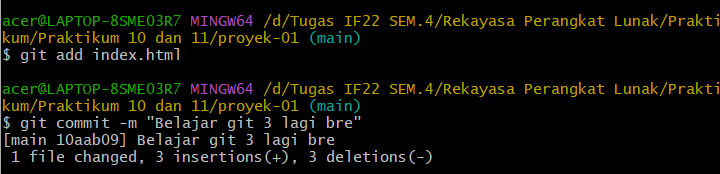
1. **Membatalkan perubahan file yang sudah dalam kondisi Commited**

Sekarang bagaimana kalau filenya sudah dalam kondisi commited dan ingin mengembalikannya? Untuk melakukan ini, harus mengetahui nomer commit, kemudian mengembalikan perubahannya seperti pada nomer commit tersebut.

Misalkan, ubah kembali file **index.html**

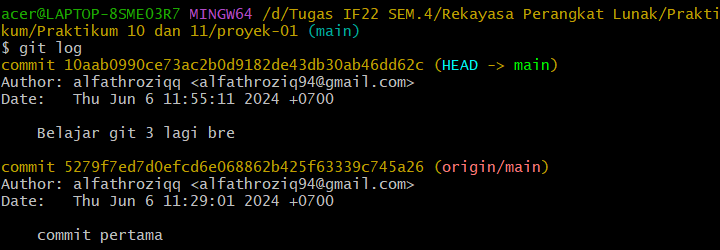
****

Kemudian melakukan commit.



Untuk melakukan commit, terlebih dahulu dilakukan perintah **git add** untuk mengubah kondisi file modified menjadi staged. Setelah berada dalam kondisi staged, maka lakukan commit untuk mengubah kondisi file menuju kondisi committed.

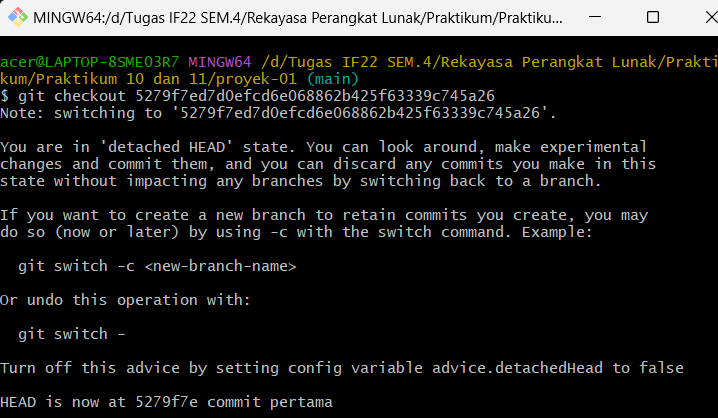
Sekarang akan melihat nomer commit dengan perintah **git log**.



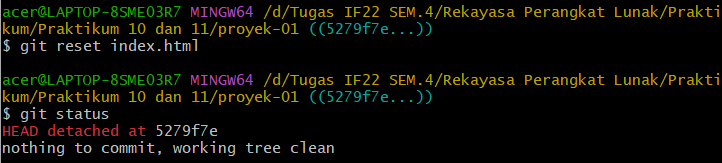
Ccommit yang sebelumnya, kita ingin kembali kesini

Commit yang baru saja dilakukan

akan mengembalikan kondisi file index.html, seperti pada commit sebelumnya. Maka bisa menggunakan perintah **git checkout <id>**.



Seperti mesin waktu, sudah mengembalikan keadaan file **index.html** seperti keadaan saat commit tersebut. Namun, saat ini kondisi **index.html** dalam keadaan staged. bisa kembalikan ke dalam kondisi modified dengan perintah **git reset**.



Pada contoh tersebut, sudah berhasil mengembalikan file index.html ke dalam keadaan seperti commit sebelumnya.

Apabila ingin mengembalikan seluruh file dalam commit, cukup melakukan checkout ke nomer commit saja, tanpa diikuti nama file. Contoh:



*Catatan:* Perintah ini akan mengembalikan semua file dalam kondisi pada nomer commit yang diberikan, namun bersifat temporer.

1. **Kembali ke 3 commit sebelumnya**

Untuk kembali ke 3 commit sebelumnya, bisa menggunakan perintah berikut.



1. **Membatalkan semua perubahan yang telah ada**

Jika ingin mengembalikan semua file ke suatu commit, bisa melakukannya dengan perintah:

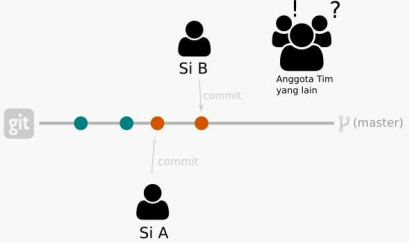


Contoh :



1. **Menggunakan Percabangan Untuk Mencegah Konflik**

Bayangkan anda sedang bekerja dengan tim pada suatu repositori Git. Repositori ini dikerjakan secara bersama-sama. Kadang akan terjadi konflik, karena kode yang tulis berbeda dengan yang lain. Misalnya, Si A menulis kode untuk fitur X dengan algoritma yang ia ketahui. Sedangkan si B menulis dengan algoritma yang berbeda. Lalu mereka melakukan commit, dan kode sumber jadi berantakan. Anggota tim yang lain menjadi pusing.



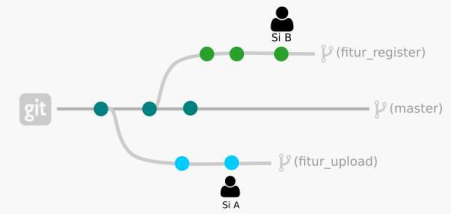
Agar tidak terjadi hal yang seperti ini, harus membuat cabang (branch)tersendiri. Misalnya, si A akan mengerjakan fitur X, maka dia harus membuat cabang sendiri. Si A akan bebas melakukan apapun di cabangnya tanpa menggangu cabang utama (main).

1. **Cara membuat cabang baru**

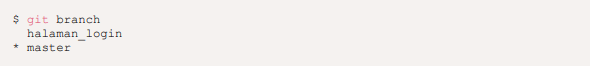
Perintah untuk membuat cabang adalah **git branch**, kemudian diikuti dengan nama cabangnya. Contoh:



Maka Git akan membuat cabang bernama **fitur\_register.**



Sekarang setiap orang memiliki cabangnya masing-masing. Mereka bebas bereksperimen. Untuk melihat cabang apa saja yang ada di repositori, gunakan perintah **git branch.** Contoh:

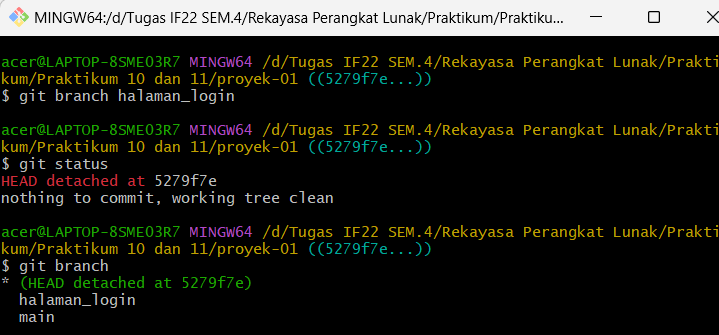


Tanda bintang (\*) artinya cabang yang sedang aktif atau sedang berada di sana.

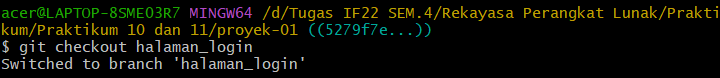
1. **Latihan**

Untuk memantapkan pemahaman tentang percabangan Git, mari coba praktik. Pada repositori, buatlah sebuah cabang baru.

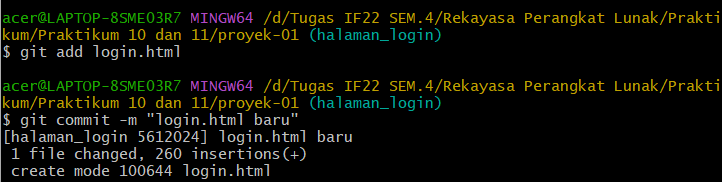
Dengan menjalankan perintah **git branch halaman\_login** maka git akan membuat branch baru dengan nama **halaman\_login**.



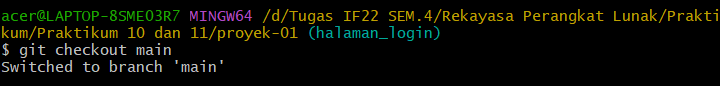
Setelah itu, pindah ke cabang yang baru saja buat dengan perintah **git checkout halaman\_login**.



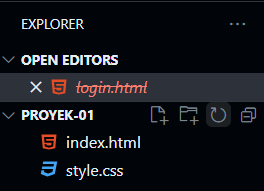
Perintah **git checkout halaman\_login** akan menyebabkan git pindah ke cabang **halaman\_login**. Git melakukan checkout dari cabang main dan pindah ke cabang **halaman\_login**. Lalu tambahkan file **login.html**, isinya bebas.



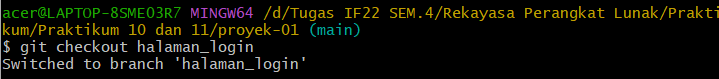
Untuk melakukan commit, terlebih dahulu dilakukan perintah **git add** untuk mengubah kondisi file modified menjadi staged. Setelah berada dalam kondisi staged, maka lakukan commit untuk mengubah kondisi file menuju kondisi committed. Revisi pada cabang **halaman\_login** sudah disimpan. Sekarang coba kembali ke cabang **main**.



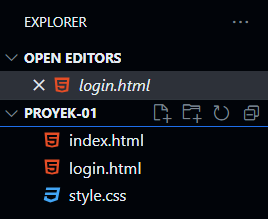
Perintah **git checkout main** akan menyebabkan git pindah ke cabang main. Git melakukan checkout dari cabang **halaman\_login** dan pindah ke cabang **main**. Apakah anda menemukan file **login.html**? Pasti tidak (file tercoret)



Sekarang kembali lagi ke cabang **halaman\_login**.



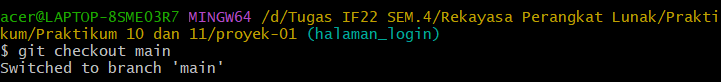
Perintah **git checkout halaman\_login** akan menyebabkan git pindah ke cabang **halaman\_login**. Git melakukan checkout dari cabang main dan pindah ke cabang **halaman\_login**. Cek lagi, apakah sekarang file login.html sudah ada?



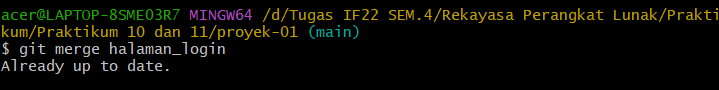
Ternyata ada. Dapat diambil kesimpulan, kalau perubahan pada cabang **halaman\_login** tidak akan berpengaruh di cabang **main**.

1. **Menggabungkan cabang**

Anggaplah sudah selesai membuat fitur login di cabang halaman\_login. Sekarang ingin Menggabungkannya denga cabang **main** (utama). Pertama, harus pindah dulu ke cabang **main**.



Setelah itu, barulah bisa menggabungkan dengan perintah **git merge**.



Perintah git merge halaman\_login akan menggabungkan cabang **main** dan cabang **halaman\_login** sehingga file **login.html** juga akan ada di cabang main.

Sekarang lihat, file login.html sudah ada di cabang **main**.



Hati-hati! kadang sering terjadi bentrok ketika menggabungkan cabang. saat menjalankan perintah `**git merge**` lalu ada kode yang bentrok.

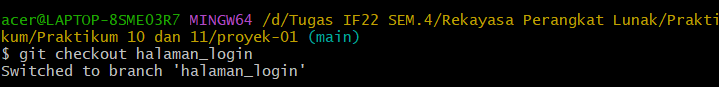
1. **Mengatasi Bentrok**

Bentrok biasanya terjadi jika ada dua orang yang mengedit file yang sama. Kenapa bisa begitu, padahal mereka sudah punya cabang masing-masing?

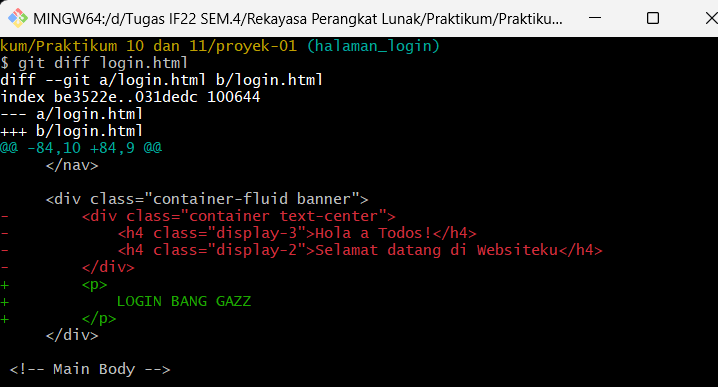
Bisa jadi, di cabang yang mereka kerjakan ada file yang sama dengan cabang lain. Kemudian, saat digabungkan terjadi bentrok. Mengatasi bentrok adalah tugas dari pemilik atau pengelola repostiri. Dia harus bertindak adil, kode mana yang harus diambil. Biasanya akan ada proses diskusi dulu dalam mengambil keputusan.

Sekarang akan coba membuat bentrokan.

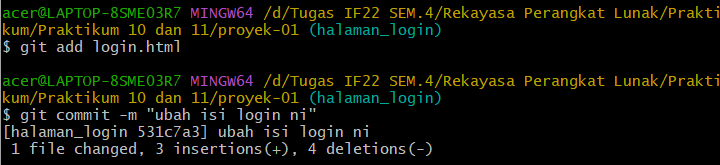
Pindah dulu ke branch **halaman\_login**…



Setelah itu, edit file login.html atau index.html, karena kedua file tersebut ada di kedua cabang yang akan gabungkan.

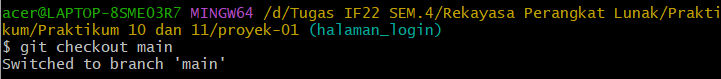


Setelah itu, lakukan commit lagi:

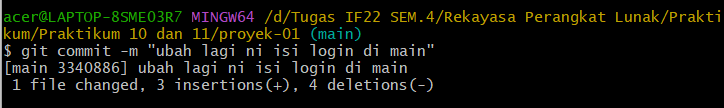


Untuk melakukan commit, terlebih dahulu dilakukan perintah **git add** untuk mengubah kondisi file modified menjadi staged. Setelah berada dalam kondisi staged, maka lakukan commit untuk mengubah kondisi file menuju kondisi committed.

Selanjutnya pindah ke cabang main dan lakukan perubahan juga di cabang ini. Ubah file yang sama seperti di cabang **halaman\_login**.

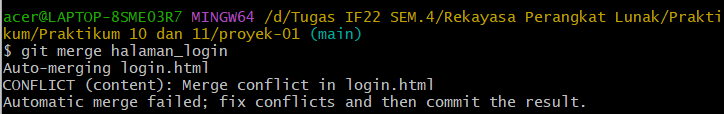


Setelah itu, lakukan commit di cabang **main**



Untuk melakukan commit, terlebih dahulu dilakukan perintah git add untuk mengubah kondisi file modified menjadi staged. Setelah berada dalam kondisi staged, maka lakukan commit untuk mengubah kondisi file menuju kondisi committed.

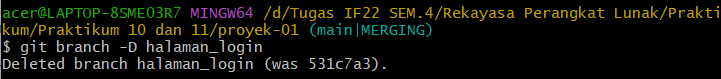
Terakhir, coba gabungkan cabang **halaman\_login** dengan cabang **main**, maka akan terjadi bentrok.



1. **Menghapus Cabang**

Cabang yang sudah mati atau tidak ada pengembangan lagi, sebaiknya dihapus. Agar repositori bersih dan rapi. Cara menghapus cabang, gunakan perintah git branch dengan argumen -d dan diikuti dengan nama cabangnya.

Contoh :



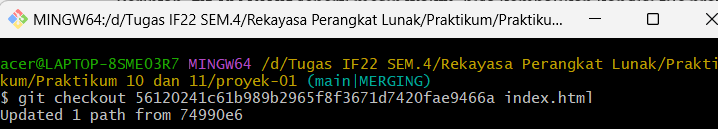
Perintah **git brach -D halaman\_login** akan menghapus branch **halaman\_login** dari repository.

1. **Git Lanjut Bagian 4**
2. **Perbedaan Git Checkout, Git Reset, dan Git Revert**

Pada praktikum git lanjut bagian 3, sudah membahas cara membatalkan revisi menggunakan perintah **git checkout, git reset,** dan **git revert**. Sekilas, tiga perintah tersebut melakukan hal yang sama. Akan tetapi ada perbedaannya. Berikut perbedaanya.

1. **Git Checkout**

Perintah **git checkout** seperti mesin waktu, bisa kembalikan kondisi file proyek seperti waktu yang dituju.



Maka semua file akan dikembalikan seperti keadaan pada nomer commit tersebut.

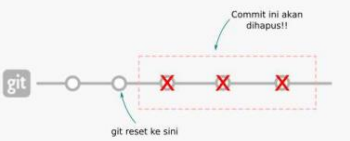


Akan tetapi, ini bersifat temporer (sementara). Pengembalian ini tidak disimpan dalam database Git. Bisa sebut perintah **git checkout** sebagai perintah untuk mengecek kondisi file di setiap commit.

1. **Git Reset**

Perintah git reset sering disebut sebagai perintah berbahaya yang dapat menghancurkan catatan histori perubahan.

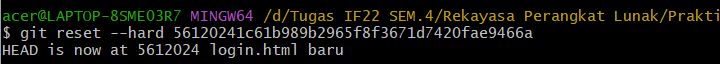
Hati-hati! Perintah ini membuat tidak bisa kembali lagi ke masa depan. Mau tidak mau, harus menulis ulang histori.



Perintah ini memiliki tiga argumen atau opsi utama, yaitu **--soft, --mixed,** dan **--hard.**

* **--soft** akan mengebalikan dengan kondisi file dalam keadaan *staged*
* **--mixed** akan mengebalikan dengan kondisi file dalam keadaan *modified*
* **--hard** akan mengebalikan dengan kondisi file dalam keadaan *commited*

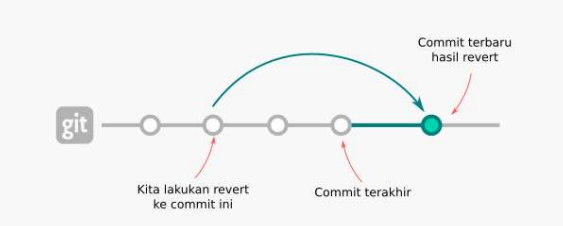
Contoh penggunaan:



Hati-hati! Jangan lakukan git reset pada repositori yang sudah di bagikan ke publik, karena dapat merusaknya.

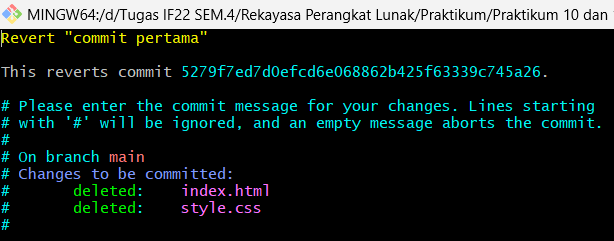
1. **Git Revert**

***Revert*** artinya mengembalikan. Perintah ini lebih aman daripada **git rese**t, karena tidak akan menghapus catatan sejarah revisi. *Revert* akan akan mengambil kondisi file yang ada di masa lalu, kemudian menggabungkannya dengan commit terakhir.



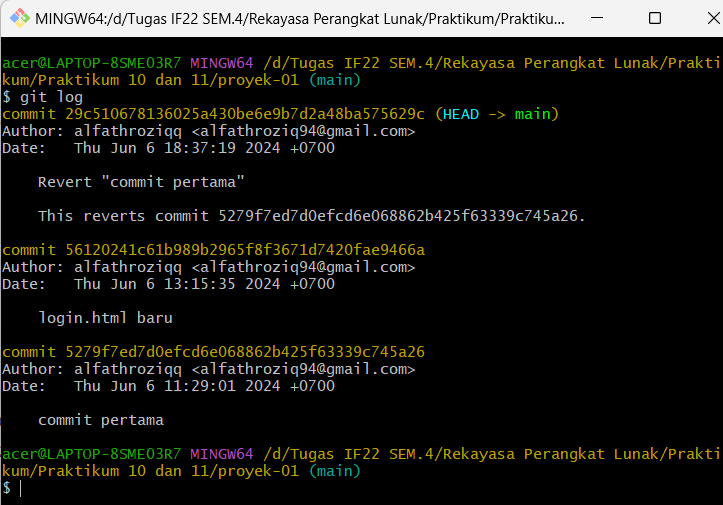
coba pada repository masingmasing. Sebelumnya, berikut ini adalah kondisi repositori yang dijadikan bahan percobaan.

Kali ini coba revert ke commit pertama pada log yang ada pada git log. Tuliskan **git revert <id>**.



Gambar diatas merupakan hasil dari revert menjadi ke commit pertama pada folder proyek-01.

Ketik **git log** untuk melihat perubahan yang terjadi pada log.



Kesimpulan:

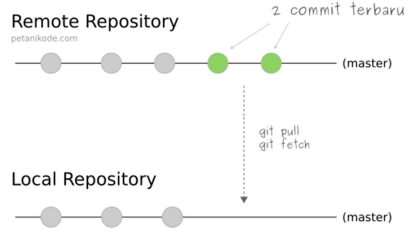
* Perintah **git checkout** mengembalikan file dalam kondisi sebelumnya, tapi bersifat sementara.
* Perintah **git reset**, akan mengembalikan file ke kondisi sebelumnya, kemudian menghapus catatan sejarah commit beikutnya.
* Perintah **git revert** mengembalikan file dengan tidak menghapus sejarah commit.

1. **Studi Kasus : Kapan menggunakan git pull dan git fetch**

Perintah **git pull** dan **git fetch** adalah dua perintah untuk mengambil commit terbaru dari remote repository. Apa perbedaan dari kedua perintah tersebut? kapan waktu yang tepat menggunakan git pull dan git fetch?

1. **Apa perbedaan Git Pull dengan Git Fetch**

Sebelum menjawab pertanyaan “kapan waktu yang tepat menggunakan **git pull** dan **git fetch**?” bahas dulu perbedaanya. Perintah **git pull** dan **git fetch** fungsinya sama. Yaitu mengambil commit terbaru dari ***remote repository***



Perbedaanya perintah **git pull** dan **git fetch** :

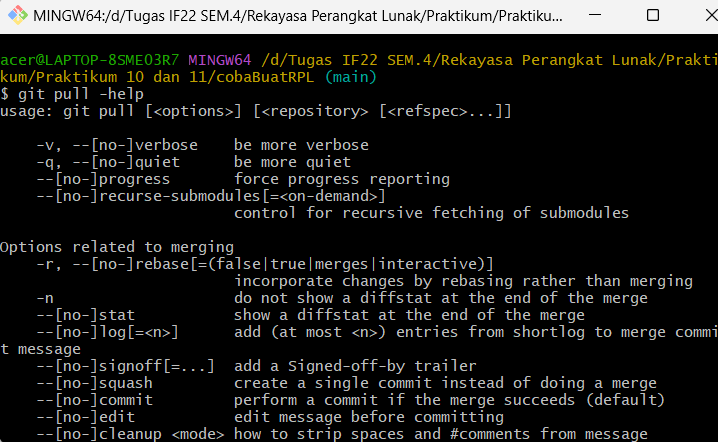
* Perintah **git pull** akan mengambil commit terbaru lalu otomatis menggabungkan (merge) dengan branch yang aktif.
* Sedangkan **git fetch** akan mengambil commit saja. Perintah **git fetch** tidak akan langsung melakukan *merge*.

1. **Cara menggunakan git pull dan git fetch**

Perintah git pull dan git fetch dapat gunakan seperti ini:

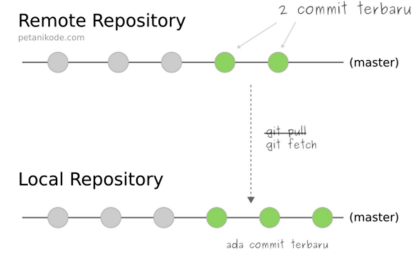


Artinya akan mengambil commit dari branch main pada repository remote. origin adalah nama remote-nya. Untuk penjelasan yang lebih detail, bisa membaca di dokumentasi dengan mengetik perintah git pull **--help** atau **git fetch --help**.



1. **Kapan waktu yang tepat menggunakan git pull dan git fetch**

Apabila sudah melakukan commit di repository lokal, maka yang gunakan adalah git fetch.



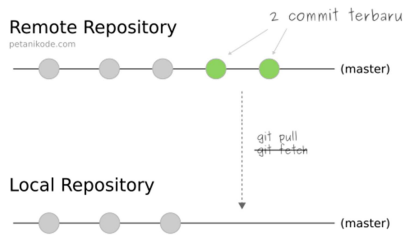
Menggunakan **Git Fetch**

Kenapa menggunakan **git fetch**?

Karena untuk mencegah terjadinya bentrok.

Perintah **git fetch** akan mengambil commit terbaru dan menyimpannya di branch origin/main.

Sedangkan apabila tidak pernah melakukan apa-apa di lokal repository, bisa menggunakan **git pull**.

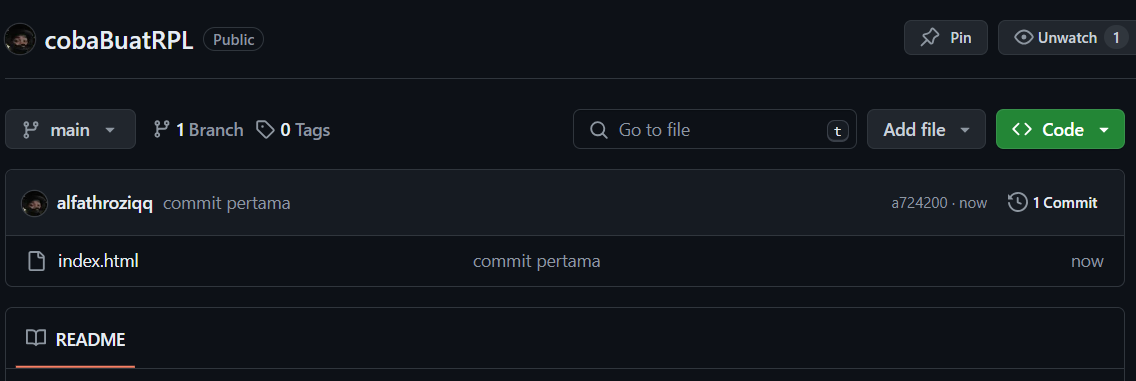


Menggunakan **Git Fetch**

Perintah **git pull** akan mengambil commit terbaru ke branch origin/main dan langsung menggabungkannya dengan branch main (lokal).

1. **Studi kasus penggunaan git pull dan git fetch**

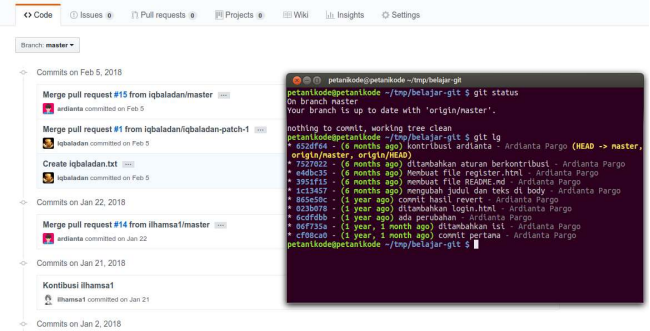
Untuk mencoba penggunaan **git pull dan git fetch** akan menggunakan repository **cobaBuatRPL** yang sudah saya dibuat di Github.



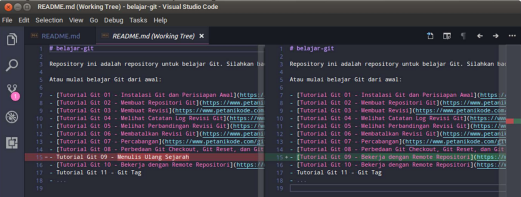
Repositori di Github

Saat ini sudah terdapat 14 kontributor di dalam repositori tersebut. Sedangkan pada repositori lokal, saya belum melakukan apa pun.

Ini perbedaan commit repo lokal dengan remote:



Perbedaan commit di repo remote dengan lokal Pada repository remote sudah terjadi 48 commit, sedangkan pada lokal hanya 10 saja. Sekarang saya akan melakukan sebuah commit di repository lokal. Saya akan mengubah file README.md menjadi seperti ini:

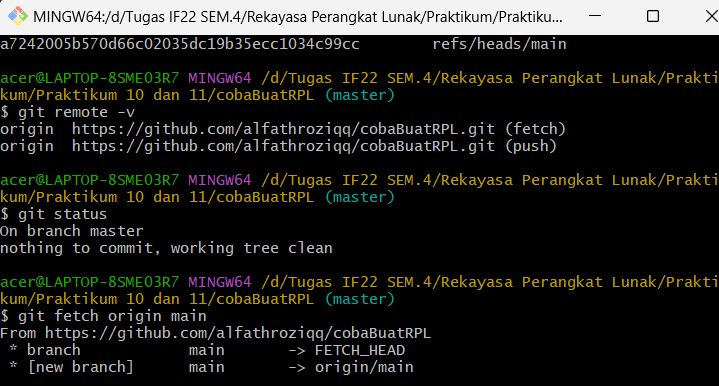


Jadi sekarang di repositori lokal ada 11 commit.

Lalu perintah yang mana yang harus gunakan untuk mengambil 48 commit di Github?

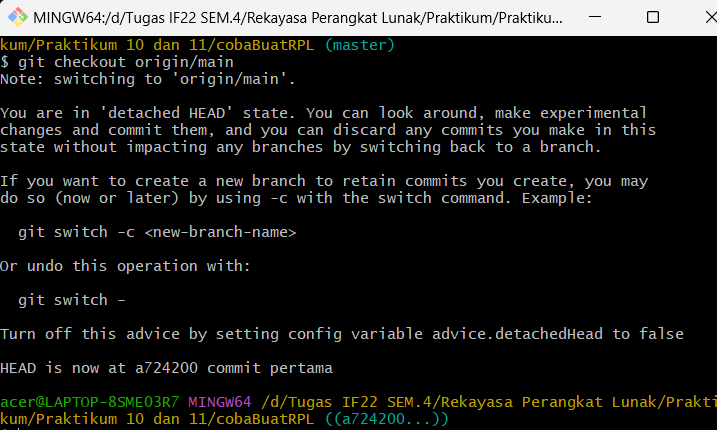
Tentu saja akan menggunakan git fetch, karena saya sudah melakukan perubahan di repo lokal.



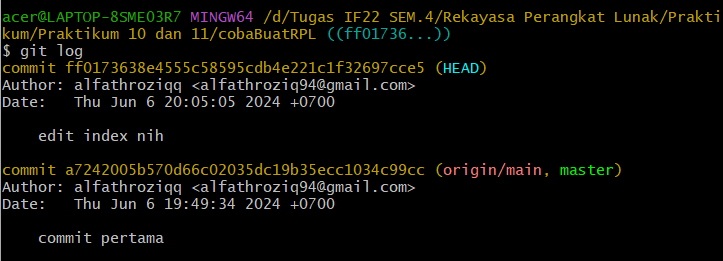


bisa masuk ke branch origin/main dengan perintah git checkout:





Lalu bisa lihat commit apa saja yang ada di sana dengan git log:



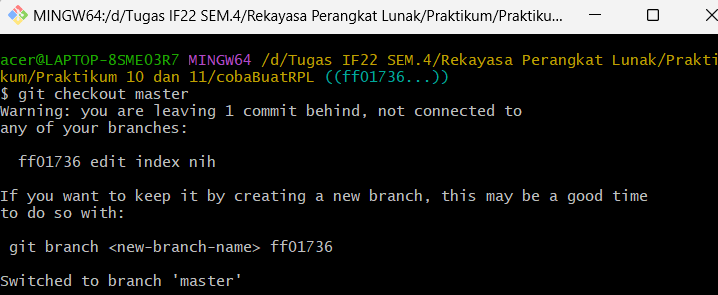
Ada dua branch dalam log tersebut:

* **master** adalah branch master di lokal.
* **origin/master** adalah branch master di remote yang sudah fetch.

Apabila sudah yakin, bisa menggabungkan (merge) dua branch tersebut. Caranya:



Menggabungkan branch **master** dengan **origin/master**



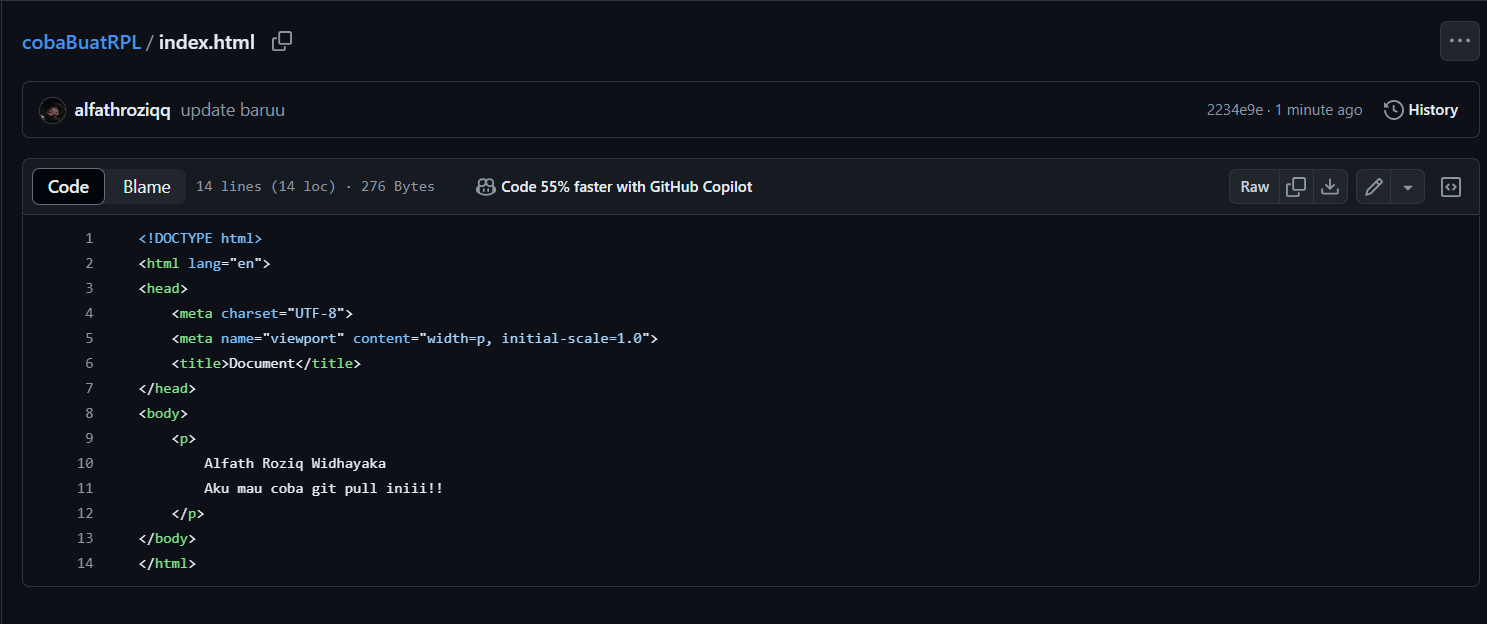
Kemudian lihat histori commit nya dengan git log

Lalu Bagaimana Penggunaan Git Pull?

Perintah git pull digunakan saat tidak pernah melakukan commit apapun di repo lokal. Cara kerja git pull:

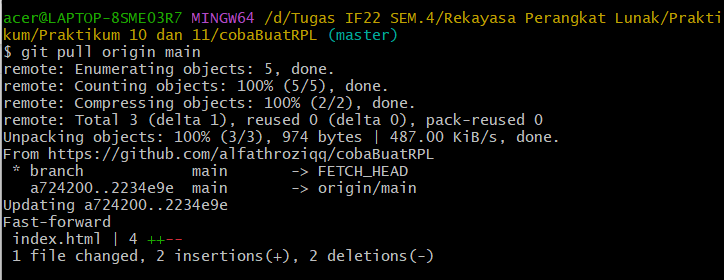
* Ambil semua commit dari repo remote
* Gabungkan branch main (lokal) dengan origin/main (remote)
* Selesai

Untuk mencoba **git pull**, saya akan mengubah file **index.html** dari Github.



Artinya sekarang di repositori remote (Github) ada 1 commit baru. Bagaimana cara mengambilnya dengan git pull?

Caranya: Gunakan perintah **git pull origin main**.



Maka semua commit terbaru dari branch origin/main di Github akan diambil ke lokal dan langsung digabungkan dengan branch main.

1. **Kesimpulan**

Pada praktikum **Git Lanjutan bagian 3**, mempelajari berbagai cara untuk membatalkan perubahan dan mengelola revisi di Git menggunakan perintah **git checkout**, **git reset,** dan **git revert**. Perintah ini memungkinkan pembatalan perubahan yang belum di-staged, mengembalikan file dari staged ke modified, serta mengembalikan file ke kondisi commit sebelumnya. Selain itu, bagian ini juga menekankan pentingnya penggunaan percabangan (branching) dalam Git untuk menghindari konflik dan memastikan kolaborasi yang lebih aman dan terstruktur dalam tim. Praktikum ini juga mencakup teknik penggabungan cabang dan cara mengatasi bentrok saat penggabungan.

Pada praktikum **Git Lanjutan bagian 4**, difokuskan pada perbedaan antara perintah git **checkout, git reset,** dan **git revert** serta penggunaannya yang tepat. Selain itu, praktikum ini membahas perbedaan antara **git pull** dan **git fetch** serta situasi yang tepat untuk menggunakan masing-masing perintah tersebut. **Git pull** digunakan untuk mengambil dan menggabungkan commit terbaru dari remote repository secara otomatis, sedangkan **git fetch** hanya mengambil commit tanpa langsung menggabungkannya, sehingga lebih aman untuk mencegah konflik ketika sudah ada perubahan di repository lokal.