Universitas Pertamina

Author: Ahmad, Farel, Hafiz 02/10/2024

GIT & GITHUB

VERSION CONTROL

PREPARED BY

UNIVERSITAS PERTAMINA

MADE FOR

PRAKTIKUM PEMROGRAMAN WEB

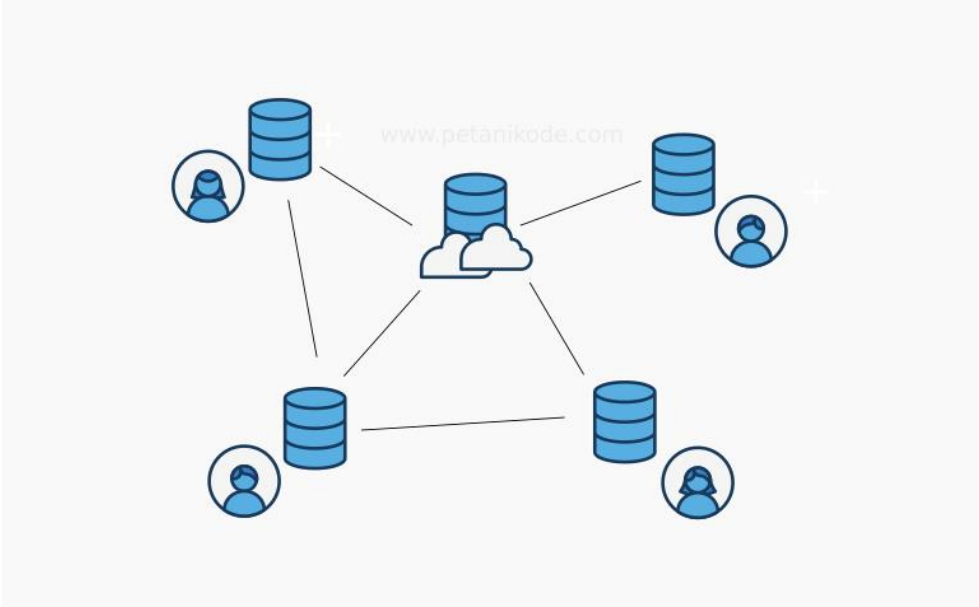
Universitas Pertamina

Pendahuluan Git

Git adalah salah satu sistem pengontrol versi (Version Control System) pada proyek perangkat lunak yang diciptakan oleh Linus Torvalds.

Pengontrol versi bertugas mencatat setiap perubahan pada file proyek yang dikerjakan oleh banyak orang maupun sendiri.

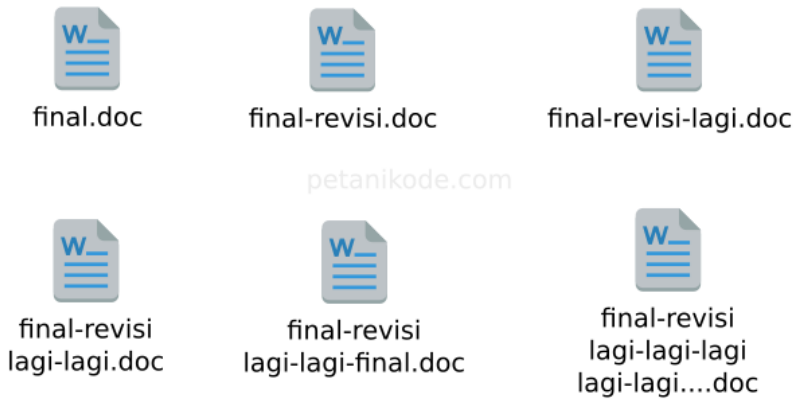
Git dikenal juga dengan distributed revision control (VCS terdistribusi), artinya penyimpanan database Git tidak hanya berada dalam satu tempat saja.



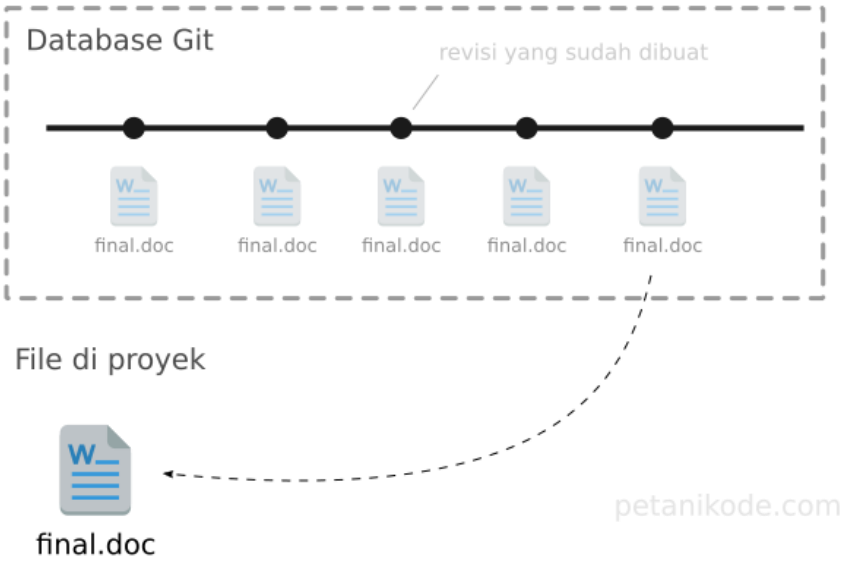
Apa yang dilakukan Git

Git sebenarnya akan memantau semua perubahan yang terjadi pada file proyek. Lalu menyimpannya ke dalam database.

Sebelum menggunakan Git:



Setelah menggunakan git:



Dengan begitu kita hanya memiliki 1 berkas *final.doc* namun kita bisa menyimpan tiap perubahan yang telah kita lakukan.

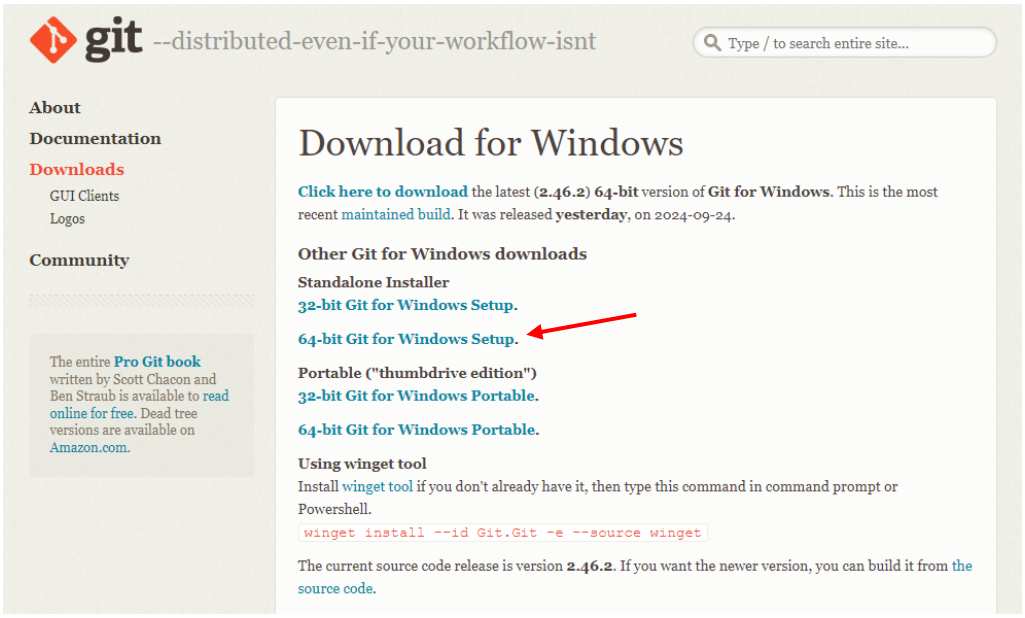
Kenapa Git itu penting

Selain dari pada mengatur versi dari suatu dokumen, dengan git kita juga bisa berkolaborasi dengan orang lain. Fitur itu lah yang menjadi kenapa Git itu penting bagi seorang programmer. Juga pada era modern ini, ada beberapa cara deploy yang meng-*invoke* Git dalam proses CI/CD atau *devops*, sehingga Git menjadi *tool* yang wajib dikuasai oleh para programmer.

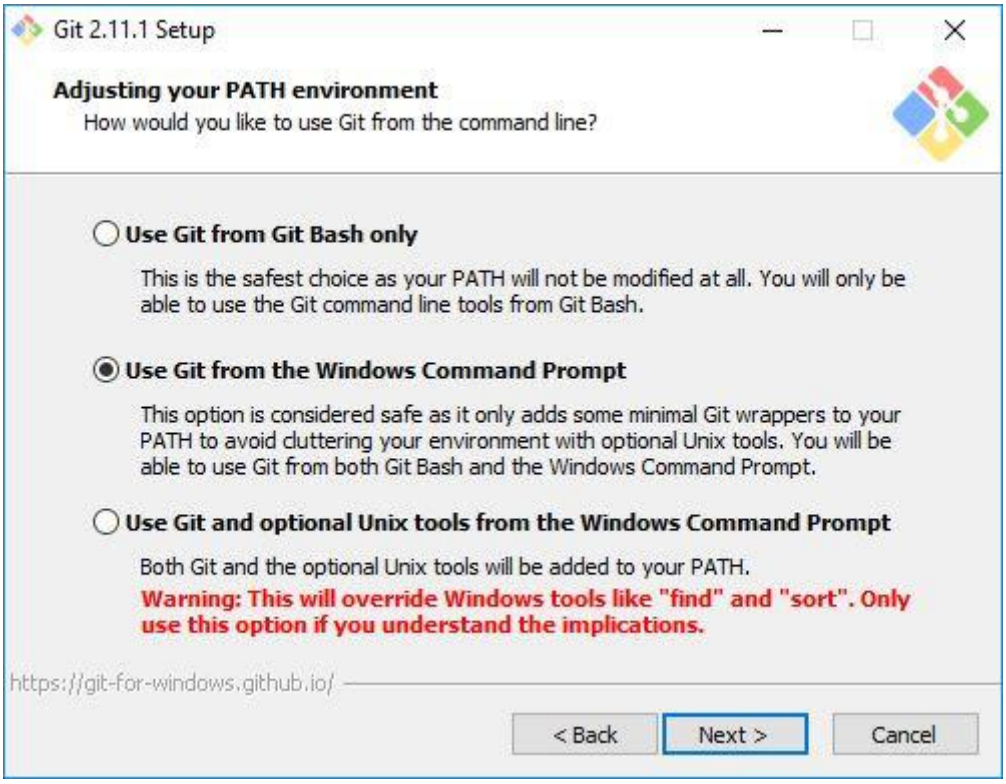
Instalasi dan Setup Git

1. Windows

   a. Unduh Git di https://git-scm.com/downloads/win , pilih yang standalone dan sesuaikan dengan bit laptop kalian (umumnya 64 bit)



   b. Install git cukup dengan klik 2x pada file .exe hasil download dari langkah pertama. Kemudian klik next sampai selesei, dan pastikan jika kalian melihat jendela seperti yang di bawah, maka sesuaikan seperti yang ada di gambar.



2. Mac

   a. Buka terminal

   b. Jalankan perintah seperti di bawah

   ```
   brew install git
   ```

   c. Jalankan perintah seperti di bawah untuk memastikan git telah terinstall

   ```
   git --version
   ```

3. Setup Git

   a. Jalankan perintah di bawah di dalam terminal / git bash, akan lebih baik jika menggunakan email yang aktif

   ```
   git config --global user.name "jhon doe"
   git config --global user.email jhon@gmail.com
   ```

   b. Jalankan perintah di bawah ini untuk mengatur nama branch default dari *master* menjadi *main*

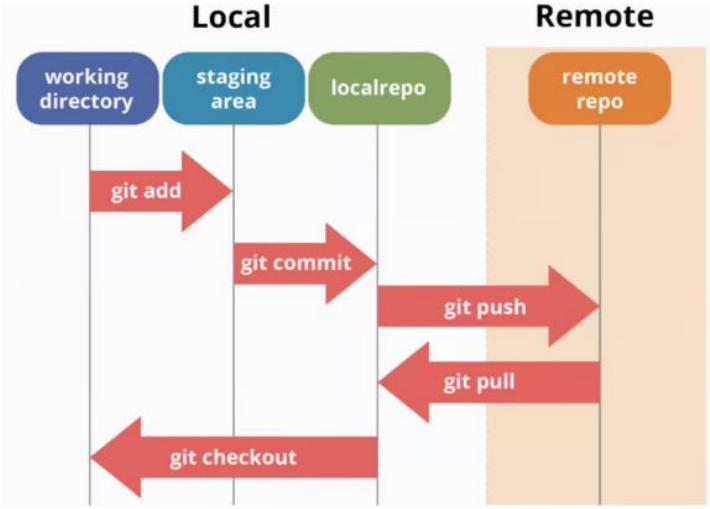
   ```
   git config --global init.defaultBranch main
   ```

   c. Jalankan perintah di bawah ini untuk memastikan konfigurasi sudah sesuai dengan yang diinginkan

```
git config --list
```

Bekerja dengan Git

1. State dalam Git



Sebelum kita bekerja dengan Git, kita harus memahami dulu konsep 3 state dalam Git. Tiga state tersebut di antaranya adalah *working stage / modified*, *staged,* dan *committed.*

Working stage / modified

Modified adalah kondisi di mana revisi atau perubahan sudah dilakukan, tetapi belum ditandai dan belum disimpan di *version control*.

Staged

Staged adalah kondisi di mana revisi sudah ditandai, tetapi belum disimpan di *version control*. Untuk mengubah kondisi file dari *modified* ke *staged* gunakan perintah **git add nama_file**.

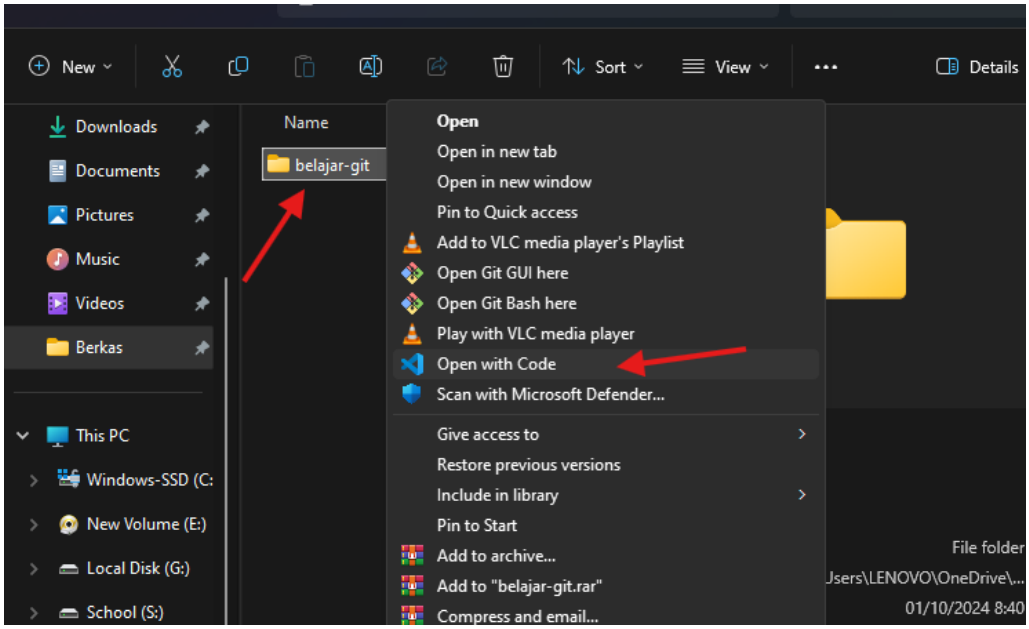
Committed

Committed adalah kondisi di mana revisi sudah disimpan di version control. perintah untuk mengubah kondisi file dari *staged* ke *committed* adalah **git commit -m "message_for_commit".**

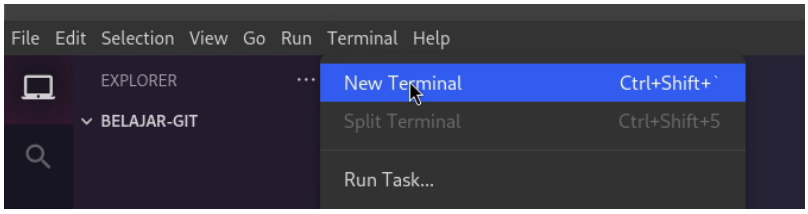
2. Workflow

Bekerja dengan git merupakan serangkaian proses sinkronus yang harus dilakukan oleh programmer. Berikut adalah tahapan lengkap yang harus dilakukan untuk bekerja dengan Git.

a. Membuat folder projek *belajar-git* di file manager kalian lalu membukanya dengan vs code.



b. Membuka terminal di VS Code dengan menekan tombol *CTRL* + ` atau di tabs terminal > new terminal.

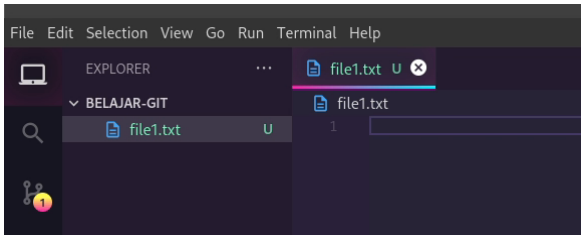


c. Membuat repository

Repository adalah folder projek, tempat di mana kita akan bekerja. Cara meembuat repository bisa dengan menjalankan perintah *git init.*

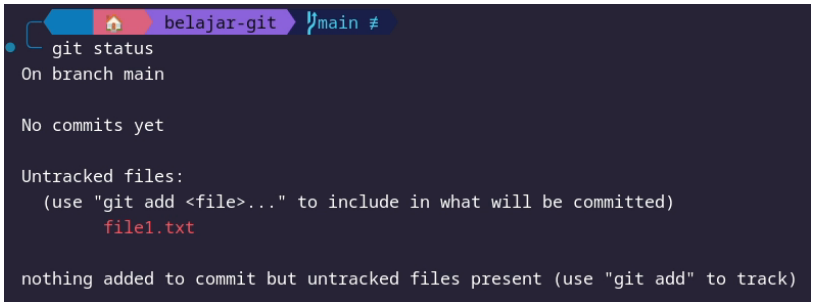
d. Mulai bekerja dengan menambahkan file yang berkaitan

Setelah kita meeng-inisiasi git di dalam repository kita, maka git akan secara otomatis mendeteksi segala macam perubahan yang terjadi di dalam folder projek. Termasuk di antaranya menambah, menghapus, me-*renamee file* maupun perubahan di dalam file itu sendiri yang bisa berupa menambah baris, menghapus dan mengganti baris kode.



e. Melihat status file dengan menjalankan perintah *git status* di dalam terminal

Untuk melihat status file atau perubahan yang kita lakukan itu berada di section mana dan di dalam kondisi apa, kita bisa menggunakan perintah *git status* untuk menampilkan status dari repository kita.



Untracked files artinya ada file baru yang terdeteksi oleh git namun belum ditambahkan ke repository atau bisa dibilang belum dikenali sehingga perubahan yang dilakukan di dalam *file1.txt* tidak akan di-*tracking.*

Kondisi semacam ini termasuk di dalam *modified state* pada *working directory section.*

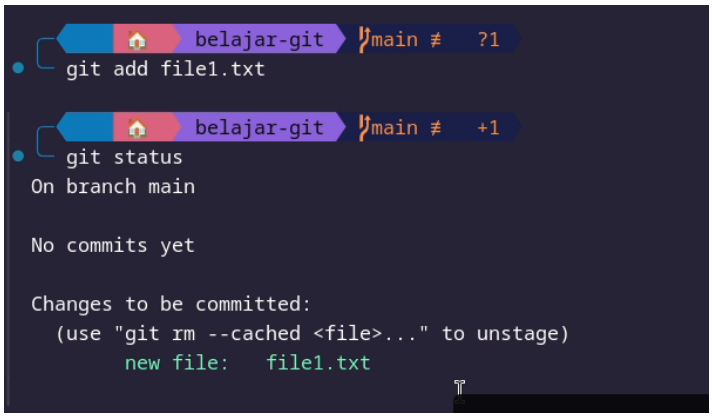
f. Membuat perubahan pada file yang berkaitan

Menambah, menghapus, dan me-*rename* dan mengubah isi *file* termasuk tindakan merubah/*modification*. Git akan mendeteksi perubahan tersebut dan memasukkannya dalam *working directory section.*

g. Memindahkan state dari *modified* ke *staged* dengan menjalankan perintah *git add nama_file* di dalam terminal

Setelah kita selesei melakukan suatu perubahan pada file projek kita, maka langkah selanjutnya adalah menandai perubahan tersebut. Sederhananya seperti

kita membuat *checkpoint* dan mengabari ke Git kalau perubahan yang kita lakukan sampai di titik itu. Perubahan yang telah kita lakukan belum disimpan di dalam repository tapi sudah berada di *staging area* dan siap untuk disimpan di dalam repository.



h. Memindahkan state dari *staged* ke *committed* dengan menjalankan perintah *git commit -m "message_for_change"*

Commit itu seperti histori atau catatan revisi yang sudah kita buat dan kita simpan ke dalam repository. Ada 4 bagian penting yang ada pada struktur *commit*, yaitu *hash code*, *author*, *date* dan *message*.

i. Melihat log dari revisi perubahan kita dengan menjalankan perintah *git log* di dalam terminal

Semua perubahan yang telah kita commit akan tersimpan di dalam repository. Untuk bisa melihat daftar commit yang telah disimpan bisa menggunakan perintah *git log*. Informasi yang ada commit sangat penting, terutama *hash code* karena itu merupakan id dari commit tersebut dan jika kita ingin mengakses histori commit pada commit sebelumnya maka kita wajib mengetahui *hash code* tersebut.

```
belajar-git main ≠
git log
commit 057c55121c747aa23853474f81bb6328549dcdaa (HEAD -> main)
Author: Arroziqi <ahmadarroziqi@gmail.com>
Date:   Mon Sep 30 08:40:16 2024 +0700

    menambah baris 1 pada ketiga file

commit 55f490509dd81f9b60e197586246bb6a71c612be
Author: Arroziqi <ahmadarroziqi@gmail.com>
Date:   Mon Sep 30 08:36:30 2024 +0700

    menambah file2.txt dan file3.txt ke dalam repository

commit b35f9113d9fbc7de99118bf881e23a2467e5905f
Author: Arroziqi <ahmadarroziqi@gmail.com>
Date:   Mon Sep 30 08:32:13 2024 +0700

    menambah file1.txt
```

3. Git reset

Dengan adanya catatan histori dari tiap perubahan yang telah kita lakukan membuat kita tidak khawatir lagi jika perubahan yang kita lakukan gagal dan ingin kembali ke kondisi sebelum dilakukannya perubahan. Untuk membatalkan suatu commit, kita bisa menggunakan perintah *git reset <mode> <hash-code>*. Ada tiga mode yang biasa digunakan yaitu *--soft, --mixed, --hard.* Ketika kita tidak jadi untuk mereset suatu commit, kita bisa kembali lagi ke commit semula dengan syarat sesaat setelah kita melakukan reset ke commit tertentu, kita belum melakukan commit pada perubahan itu.

- --soft

  Akan membatalkan commit dan berpindah ke commit yang dituju, tetapi perubahan yang telah dilakukan akan tetap ada di *staging area.*

```
belajar-git main ≠
git reset --soft 057c551

belajar-git main ≠  ~1
git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file3.txt


belajar-git main ≠  ~1
git log --oneline
057c551 (HEAD -> main) menambah baris 1 pada ketiga file
55f4905 menambah file2.txt dan file3.txt ke dalam repository
b35f911 menambah file1.txt
```

- --mixed

  Akan membatalkan commit dan berpindah ke commit yang dituju, tetapi perubahan yang telah dilakukan akan tetap ada di *working directory*.

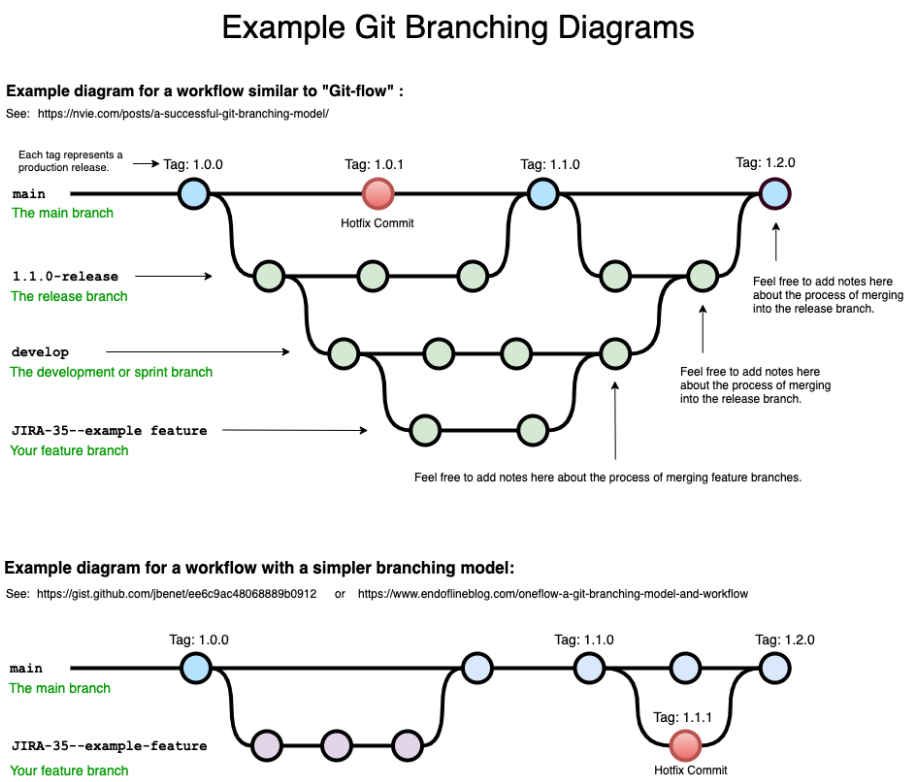
```
belajar-git  main ≠
git reset --mixed 057c551
Unstaged changes after reset:
M       file3.txt

belajar-git  main ≠  ~1
git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file3.txt

no changes added to commit (use "git add" and/or "git commit -a")

belajar-git  main ≠  ~1
git log --oneline
057c551 (HEAD -> main) menambah baris 1 pada ketiga file
55f4905 menambah file2.txt dan file3.txt ke dalam repository
b35f911 menambah file1.txt
```

- --hard

  Akan membatalkan commit dan berpindah ke commit yang dituju, tetapi perubahan yang telah dilakukan akan hilang sehingga bisa dikatakan seperti kembali ke kondisi repository awal sesuai yang dituju.

```
belajar-git  main ≠
git reset --hard 057c551
HEAD is now at 057c551 menambah baris 1 pada ketiga file
```

4. Branch



Sebelumnya kita telah bisa mengontrol versi dari aplikasi kita hanya mengandalkan commit yang selalu kita lakukan tiap ada perubahan. Namun ketika perubahan atau projek yang kita kerjakan sudah mulai kompleks. Maka akan menjadi sangat sulit untuk mencatat log dari tiap revisinya. Oleh karenanya Git menjawab kesulitan itu dengan menghadirkan fitur *branch*.

Dengan menggunakan *branch* kita bisa mengelompokan atau membuat percabangan untuk tiap kelompok perubahan yang kita lakukan pada projek kita. Contoh kasusnya misal kita bekerja dengan Agile dan sudah selesei dengan versi betanya. Kemudian pada sprint berikutnya kita ingin mengembangkan fitur baru tanpa mempengaruhi versi betanya. Maka dengan begitu kita bisa membuat branch baru untuk proses pengerjaan fitur baru kita dan versi betanya tetap dibiarkan berjalan di branch utamanya. Ketika nanti kita sudah selesei dengan fitur barunya, kita bisa menggabungkan perubahan yang dilakukan pada branch fitur baru itu ke dalam branch utamanya. Sehinggal dalam proses berjalannya aplikasi, aplikasi utamanya tidak akan terkena dampak selama proses develop fitur baru tersebut.

Langkah-langkahnya sebagai berikut:

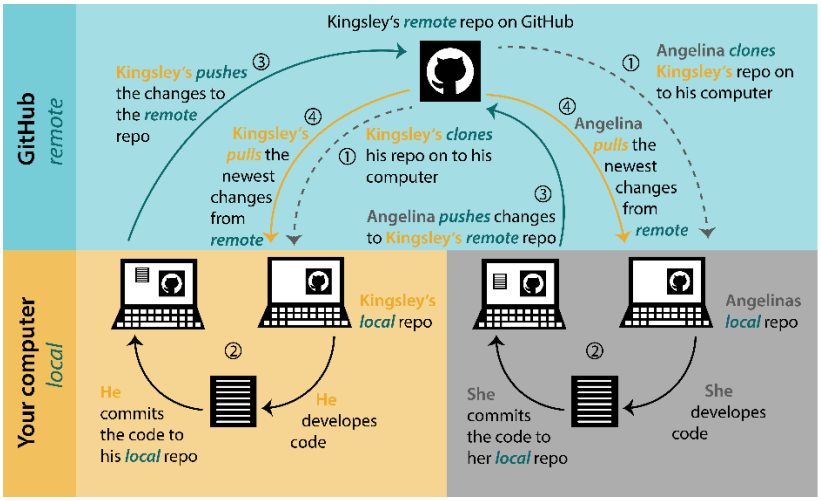
a. Masuk ke terminal di mana tempat kita bekerja
b. Pastikan kita ada di branch utama,
   Jika nama branch masih *master*, maka ganti menjadi *main* dengan menjalankan perintah *git branch -M main*
c. Jalankan perintah *git branch new_fiture* untuk membuat branch baru dengan nama *new_fiture* yang akan kita gunakan nantinya dalam proses develop fitur baru.
d. Pindah dari branch *main* ke branch *new_fiture* dengan menjalankan perintah *git switch new_fiture*
e. Mulai bekerja
f. Jika sudah selesei dengan proses develop pada fitur baru kita, dan memastikan tidak ada lagi bug yang ada pada fitur baru kita. Maka kita bisa kemudian menggabungkan perubahan yang ada pada branch *new_fiture* ke branch *main* dengan menjalankan perintah:
   git switch main
   git merge new_fiture
   untuk bisa menggabungkan 2 branch yang berbeda kita wajib pindah terlebih dahulu ke branch mana yang ingin menerima perubahan. Baru setelah itu kita gabung dengan perintah *git merge*
g. Hapus branch *new_fiture* jika sudah tidak digunakan lagi.
   git branch -d new_fiture

Berkolaborasi dengan GitHub

1. Setup akun GitHub

Buat akun pada https://github.com/ , akan lebih baik jika menggunakan akun email yang sama saat konfigurasi git.

2. GitHub Flow



Dalam berkolaborasi menggunakan Git dan GitHub, maka perlu dipahami *flow* yang harus dilalui oleh pengembang. GitHub secara sederhana merupakan repository online yang bisa diakses secara bersama. Jika kita hanya menggunakan Git, maka pola kolaborasinya berbatas pada Local Area Network. Sehingga tidak dapat di akses secara online oleh device lain. Dengan bantuan GitHub maka projek yang telah kita simpan di dalam Git, bisa kita up ke internet sehingga kita bisa akses secara bersama via online internet.

Membuat repository GitHub

a. Masuk ke akun GitHub
b. Membuat repository baru
c. Copy link repository-nya
d. Masuk ke terminal di mana tempat kita bekerja
e. Hubungkan Git Local kita dengan repository GitHub kita dengan menambahkan *remote repository* menggunakan perintah *git remote add origin <link_repository>*. *Origin* merupakan nama remote yang biasa dipakai secara umum.
f. Upload *commit* yang telah kita lakukan di repository local dengan menjalankan perintah *git push -u origin main* . Perintah tersebut akan mem-*push* perubahan yang telah kita buat di local ke dalam repository *GitHub* dengan nama branch *main.*

Clone repository GitHub ke repository local

a. Masuk ke terminal di mana tempat kita mau menaruh berkas projek kita
b. Clone repository GitHub ke local dengan menjalankan perintah *git clone <link_repo>*

Mengambil perubahan dari repository GitHub

a. Masuk ke terminal di mana tempat kita mau menaruh berkas projek kita
b. Ambil tiap perubahan yang ada pada repository GitHub dengan menjalankan perintah *git pull origin <nama-branch>*

3. .gitignore

File *.gitignore* adalah file yang menentukan berkas apa saja yang tidak akan di upload ke repository GitHub. Seperti file *.env* yang mana berisikan data *credential* yang akan berbahaya kalau kita upload ke repository *GitHub.*

4. LICENSE

File khusus ini sangat penting untuk menjaga karya kita agar terhindar dari kasus plagiarisme. Ada banyak jenis license yang disediakan oleh GitHub, namun yang standard dan free adalah MIT licencse.

Conflict Handling

Hal yang wajib dikuasai oleh seorang pengembang jika ingin bekerja bersama dengan tim adalah *conflict handling.* Konflik pasti akan selalu terjadi dalam setiap pekerjaan, karena pasti ada beberapa file yang dikerjakan bersama. Oleh karenanya perlu dipahami dengan benar-benar *conflict handling* ini. GitHub sudah menyediakan fitur *auto solve* tapi tetap dalam beberapa kasus tertentu mewajibkan penanganan dari kita secara manual.