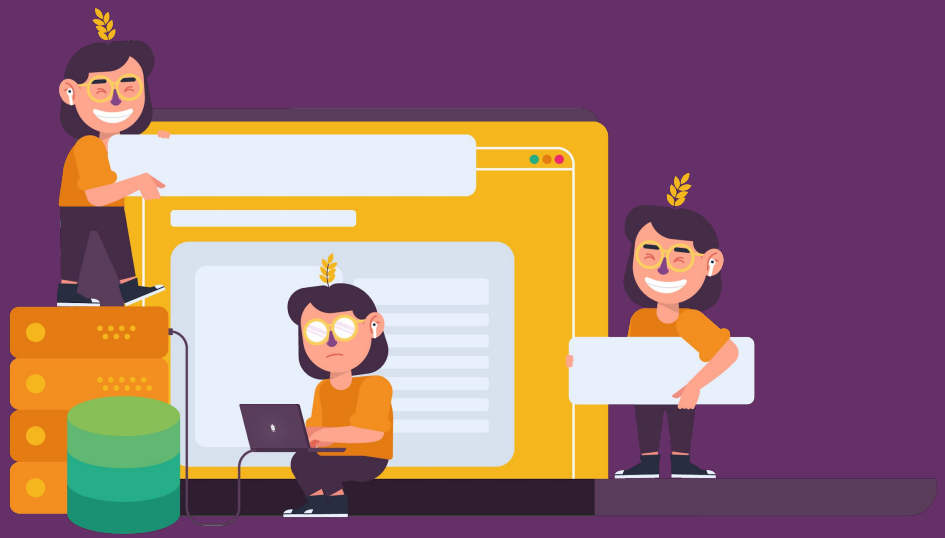
BINAR ACADEMY

Git

Gold - Chapter 3 - Topic 3

Hai! Ini adalah Topik ketiga dari Chapter 3 *online course* Full-Stack Web dari Binar Academy!

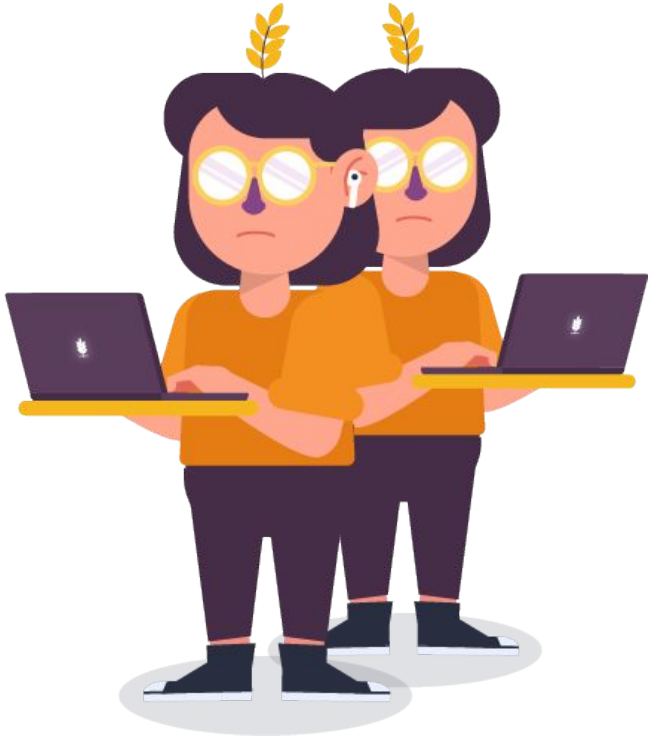


Pada materi kali ini, kita akan belajar lebih jauh tentang Git.

Secara detail, ini poin-poin yang akan kita bahas hari ini:

1. Mengenal apa itu Git
2. Mengenal apa itu version control
3. Bagaimana bekerja dengan Git
4. Bagaimana berkolaborasi dengan Git
5. Bagaimana menyelesaikan konflik di dalam Git

Apa sih sebenernya Git?

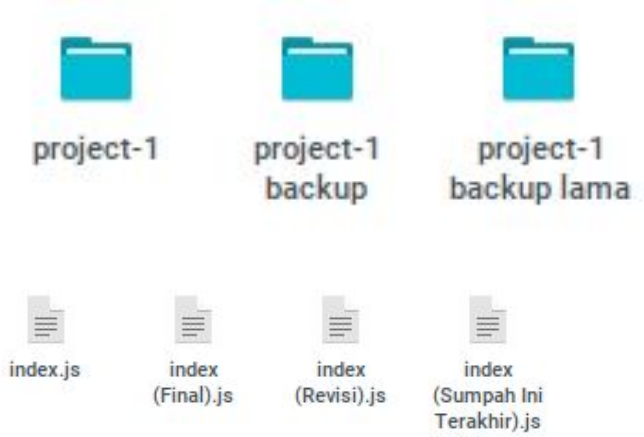


Dalam dunia pemrograman, kita dianjurkan untuk bekerja sama dengan orang lain. Nah, demi mempermudah proses kolaborasi tersebut, tentu kita butuh ruang menulis kode yang bisa untuk digunakan oleh dua orang atau lebih di waktu yang bersamaan.

Nah, Git adalah jawaban dari permasalahan di atas!

Git merupakan sebuah *version control,* atau aplikasi yang berguna untuk mengatur versi dari kode/aplikasi kita.

In a nutshell, Git itu seperti Google Drive-nya Programmer.



Cara seperti ini sudah nggak zaman lagi di dunia pemrograman. Soalnya, kita sudah punya **Git!**

Ketika kita mengerjakan tugas, *project*, atau skripsi, kita ingin memiliki file backup.

Dengan membuat file backup, kita bisa dengan mudah kembali ke kondisi sebelumnya jika kerjaan kita ada yang rusak.

Nah, Git juga memiliki fungsi backup seperti yang dijelaskan di atas. Di dalam Git, semua perubahan pada kode kita akan tersimpan ke dalam history. Jadi, kita bisa *tracking history* untuk mengetahui siapa yang membuat atau mengubah kode. Sehingga, kita bisa dapat dengan mudah memperbaikinya.

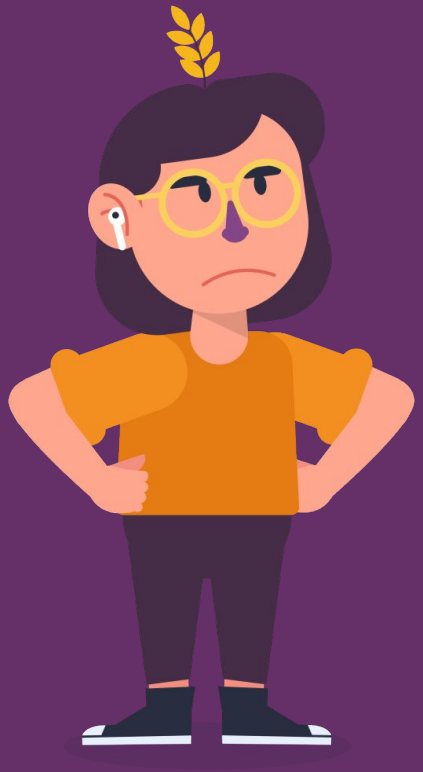
Kita bahas sejarah dikit ya!

Git ini dibuat oleh Linus Torvalds (Pencipta Linux).

Ceritanya, saat beliau membuat project linux-nya menjadi *open-source* (semua orang bisa melihat kodenya dan berkolaborasi), banyak orang yang ingin berkolaborasi ke projectnya.

Tapi, saking banyaknya orang yang berkolaborasi di project tersebut, konflik antara developer satu dengan yang lain tidak bisa dibendung lagi karena kodenya berubah terus dan nggak ada yang ngaku soal siapa dalang di balik perubahan tersebut.

Nah, untuk mengatasi masalah ini, Linus membuat aplikasi baru yang bernama Git, yang fungsinya mengontrol dan mendokumentasikan kode dari project kita. Sehingga, jika terdapat konflik kita bisa dengan mudah memecahkannya.

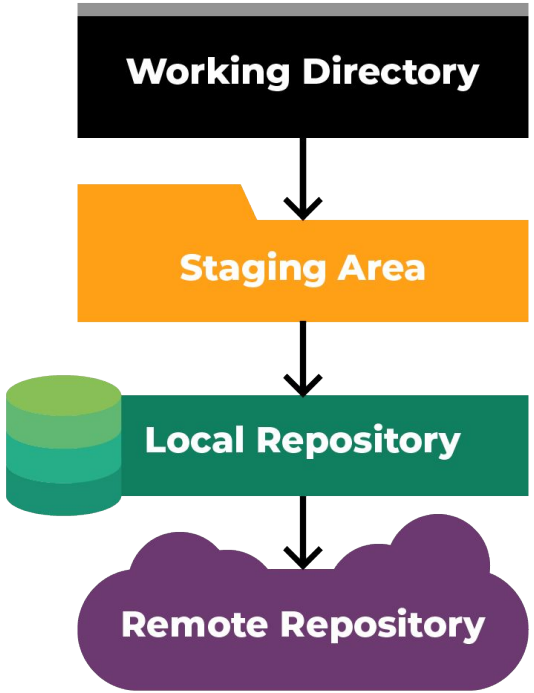


Sebelum kita mempraktikkan Git, kita perlu tahu dulu tentang satu hal ini:

Bagaimana sih Git menyimpan pekerjaan kita?

Jadi, ketika kita bekerja dengan Git, ia akan menyimpan folder/pekerjaan kita dalam 4 tahap. Tahapannya bisa kamu cek pada gambar di samping.

Setiap perubahan akan berada di salah satu tahap/area tersebut, jadi tidak semua perubahan yang kita lakukan akan masuk ke tahap yang sama.



Working Directory

Sesuai namanya, ini adalah sebuah direktori untuk menyimpan perubahan yang masih dalam tahap pengerjaan, alias belum selesai.

Contoh: Kita sedang menambahkan sederet baris kode di dalam file *index.js* lalu kita save. Maka file *index.js* tersebut berada di dalam *working directory* pada Git.

Staging Area

Suatu tahap dimana perubahan yang sudah kita buat siap untuk di-commit (dibuatkan berita acara).

Contoh: Kita sudah selesai menambahkan sederet baris kode di index.js dan kita ingin menegaskan atau melaporkan bahwa terdapat perubahan di dalam *index.js* kepada developer lain.

Untuk melakukan hal tersebut, kita harus memindahkan *index.js* dari *working directory* ke *staging area* terlebih dahulu.

Intinya, staging area itu hanyalah daftar perubahan apa saja yang akan kita masukkan ke dalam berita acara nantinya.

Local Repository

Suatu tahap dimana perubahan kita sudah ditetapkan *(fixed)*, tapi perubahan tersebut hanya terjadi di dalam komputer kita saja (belum di-*upload* ke cloud).

Nah, perubahan yang sudah masuk ke Local Repository perlu memiliki yang namanya ***commit message***.

Commit message adalah pesan untuk memberi tahu apa yang telah kamu kerjakan atau apa yang kamu ubah dari sebuah file.

Contoh:

Misal kamu menambahkan sebuah function delete di file index.Nah, setelah selesai dan siap untuk diupload kamu harus menyimpan perubahan tadi dan jangan lupa untuk menambahkan pesan perubahannya atau *commit message*.

Commit message ini harus relevant dengan apa yang kamu kerjakan, dalam kasus kita, commit message bisa seperti ini, *“add delete function”*

Remote Repository

Suatu tahap dimana perubahan kita sudah berada di cloud dan dapat diakses oleh developer lain.

Contoh: File *index.js* di *local repository* bakal kita upload ke *remote repository*. Nah, perubahan yang terjadi di file *index.js* tersebut akan otomatis diterapkan di cloud juga.

Yah kayak kita upload tugas ke Google Drive gitu deh~

Sudah cukup penjelasannya.

Sekarang langsung aja kita coba pakai **Git**!

Instalasi di Linux

Buka terminal kita,

lalu ketik perintah ini di terminal

```
# Ubuntu/debian based Distro
sudo apt update
sudo apt install git -y
```

1. Install Git terlebih dahulu

Jadi, Git itu sebenarnya adalah aplikasi di terminal. Maka dari itu, untuk instalasinya kita bisa pake terminal juga.

Memang, ada versi GUI dari Git. Tapi, untuk install dan pengoperasiannya kita perlu bersakit-sakit dahulu. Jadi, pakai yang mudah saja ya~

Oke, langsung aja kita, buka terminal dan instal linux-nya

← Coba tulis perintah di samping di dalam terminal

“Kalau kita pengen Install di MacOS, caranya gimana gan?”

Buka terminal juga, lalu ketik kode di bawah untuk install Brew dan install Git.

Oh iya, Brew adalah package manager untuk Mac OS. Dengan alat ini, kamu bisa menginstall aplikasi yang kamu inginkan dengan sangat mudah.

```
# Install brew dulu, yah
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
brew doctor

# Install Git nya
brew install git
```

Kalau sudah, pastikan Git sudah terinstall dengan baik di dalam sistem kita.

Caranya, dengan menjalankan perintah ini:

```
git --version
> git version 2.17.1
```

2. Inisialisasi Git di folder project-mu

Katakanlah kamu punya project bernama "Binar" yang tersimpan dalam directory documents. Nah, semua file project-mu kamu akan kamu simpan di dalam folder tersebut.

Nah, kalau kita pengen implementasi Git di dalam project kita, maka yang harus dilakukan adalah seperti ini:

1) Masuk ke terminal.
2) buka folder mu itu lewat terminal. Lalu, inisialiasasi Git di sana.
3) Ketik perintah di bawah ini di dalam terminal:
   git init

Perhatikan contoh kodenya di gambar ya~

```
cd Documents/binar # Untuk directory binar
git init # Untuk menginisialisasi git
```

Setelah kita inisialisasi git, maka ketika terminal kita berada di dalam folder project-mu, kita dapat menjalankan perintah-perintah git di dalam terminal kita. Seperti:

- git status
- git push
- git pull
- git commit

Nah, mungkin kita bisa coba menjalankan perintah **git status**

Maka hasilnya akan seperti gambar di samping →

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what
will be committed)

    Readme.md
    css/
    index.html
    js/

nothing added to commit but untracked files
present (use "git add" to track)
```

Setelah kita inisialisasi, semua file yang berada di dalam folder itu akan masuk ke tahap Working Directory. Nah, ada baiknya kita langsung pindahkan file kita ke dalam Local Repository dengan melakukan *commit* terlebih dahulu.

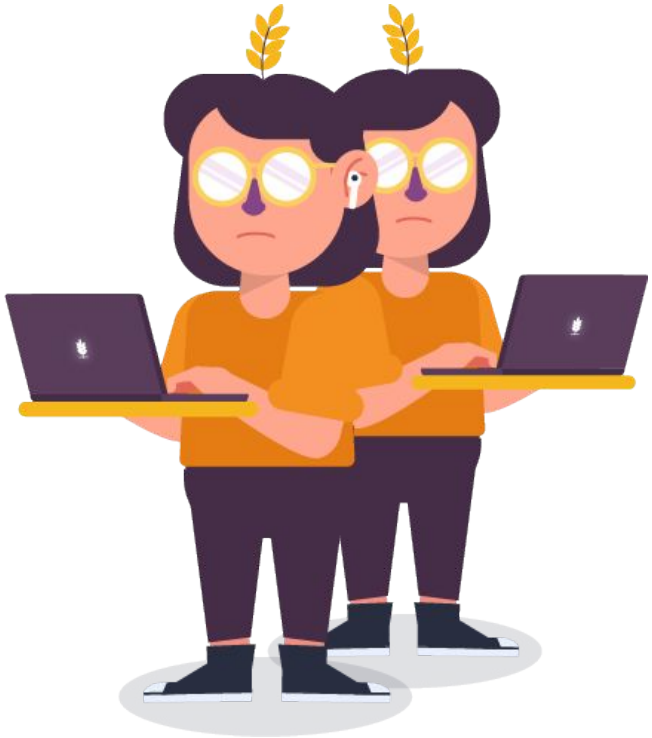
Commit pertama ini, biasa kita sebut dengan **Initial Commit**.

Untuk melakukan **commit** kita hanya perlu menjalankan dua perintah ini:

```
# Ini akan memindahkan file kita ke Staging Area
git add .
# Ini akan memindahkan file kita ke Local Repository
git commit -m "[Fikri] Initial Commit"
```

Nah, kalau kita perhatikan wujud file dan folder mu di dalam project, ga akan ada perbedaan sama sekali. Karena pemindahan stage itu hanya terjadi di dalam Git itu sendiri.

Dan, kita tidak perlu mempelajari lebih dalam tentang bagaimana Git melakukan itu. Yang penting, perubahannya sudah tercatat oleh Git, kok!



3. Lakukan Perubahan dan Commit

Setelah kita selesai dengan inisialisasi, maka hal yang perlu kamu lakukan adalah melakukan perubahan di dalam pekerjaanmu. Baik itu menambah kode baru, merubah kode, atau bahkan menghapus kode; tergantung tindakan mana yang perlu dilakukan.

Katakanlah kita punya file **`index.html`** dan kita pengen tambahin perubahan di situ.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width, initial-scale=1.0">
    <title>Belajar Git</title>
  </head>
  <body>
    <h1>Hello World</h1>
    <h5>Lagi belajar git nih</h5>
  </body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Belajar Git</title>
  </head>
  <body>
    <h1>Hello World</h1>
    <h5>Lagi belajar git nih</h5>
    <h6>Ini perubahan Baru</h6>
  </body>
</html>
```

Nah setelah kita melakukan perubahan, yang kita perlukan adalah melakukan **commit**. Fungsinya untuk menegaskan bahwa perubahan itu dicatat oleh git.

Sebelum kita melakukan **commit** pastikan yang akan kita commit itu benar.

Kita bisa jalankan `git status` lagi untuk mengecek apa saja yang berubah. Nanti outputnya akan seperti di gambar.

Tampilan tersebut menunjukkan bahwa Git sudah melakukan tracking di folder kita. Silakan kamu cek ulang; benarkah file index.html yang berubah. Kalau sudah yakin, baru kita lakukan commit.

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be
committed)
  (use "git checkout -- <file>..." to discard
changes in working directory)

      modified:   index.html

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

Untuk melakukan commit, maka kita harus pindahkan perubahan tersebut dari **working area** ke **staging area**. Cara memindahkannya mudah kita cukup jalankan perintah ini

```
git add namafileita
```

Karena yang kita rubah adalah file **index.html**, maka kita bisa **pindahkan index.html** ke **staging area.**

```
git add index.html
```

Untuk mengecek apakah perubahan kita sudah dipindah ke staging area, kita bisa cek menggunakan **git status** lagi**.**

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

      modified:   index.html
```

Jika warnanya sudah hijau, maka artinya sudah masuk ke **staging area.**

Setelah perubahan tersebut kita masukkan ke staging, maka yang perlu kita lakukan adalah commit perubahan tersebut. Kita perlu melakukan commit agar perubahan kita dipindah ke local repository.

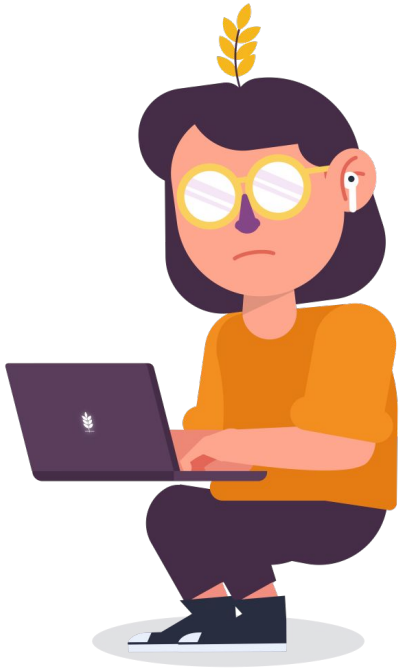
```
git commit -m "[Fikri] Perubahan Baru dengan h6"
```

Kita bisa pastikan lagi kalau perubahan tersebut sudah di-commit dengan menggunakan git status.

```
On branch master
nothing to commit, working tree clean
```

Nah, sejauh ini kita telah bekerja menggunakan git di local repository kita.

Sekarang kita akan bekerja dengan **remote repository**.



Berikut 3 service yang paling terkenal untuk remote repository :

- Github
- Gitlab
- Bitbucket

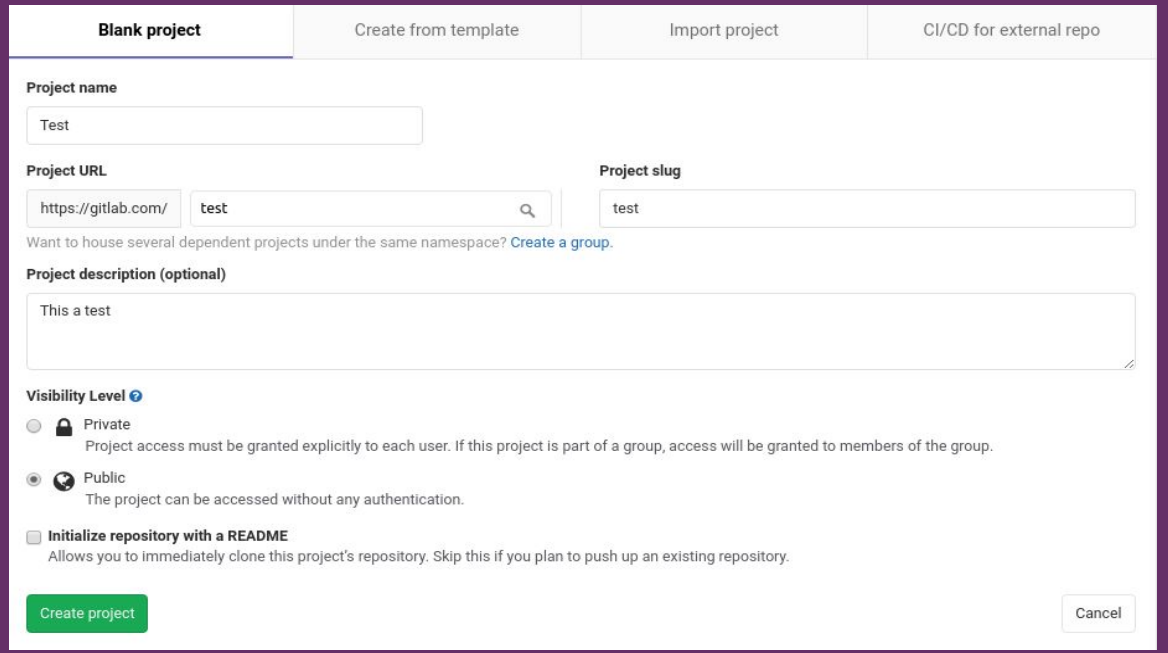
Sepanjang meteri ini kita akan menggunakan Gitlab.

Pastikan kamu membuat telah membuat akun di gitlab ya.

Pada gambar, kamu bisa lihat kotak dialog yang muncul saat membuat repository baru di Gitlab.

Nah, kamu tinggal mengisi form yang disediakan.

Terakhir tekan tombol "create project" untuk membuat project-nya.



Setelah berhasil membuat repository baru, maka yang perlu kita lakukan adalah menyalin alamat dari repository itu dengan menekan tombol "Clone". Selanjutnya, pilih "Clone with HTTPS" kalau belum setup SSH Key.

Untuk menyambungkan ke remote repository, kita akan menggunakan perintah git remote, perintah ini akan mengurusi segala sesuatu yang berbau konfigurasi remote repository.

Bentuknya nanti seperti ini:
git remote add origin git@gitlab.com:namamu/nama-project.git

Fungsi perintah tersebut adalah memberi tahu local repository kita soal alamat remote repository yang akan menjadi tempat tujuan kita dalam mengupload filenya.

Selain perintah untuk menambah remote repository tadi, ada lagi beberapa perintah yang akan sangat berguna saat bekerja dengan remote repository, di antaranya:

- git remote -v , digunakan untuk mengetahui alamat remote repository
- git remote remove origin, digunakan untuk menghapus remote
- git remote set-url origin git@gitlab.com/alamat/baru.git, digunakan untuk mengganti remote

Untuk mengupload pekerjaan kita, kita bisa menggunakan perintah **git push origin master**

Maksud dari perintah di atas adalah kita upload perubahan kita ke remote repository yang bernama origin di branch master.

Tapi,,,

Sangat tidak dianjurkan untuk upload langsung ke branch master

Kita akan bahas lebih lanjut soal apa itu branch di bagian lain modul ini

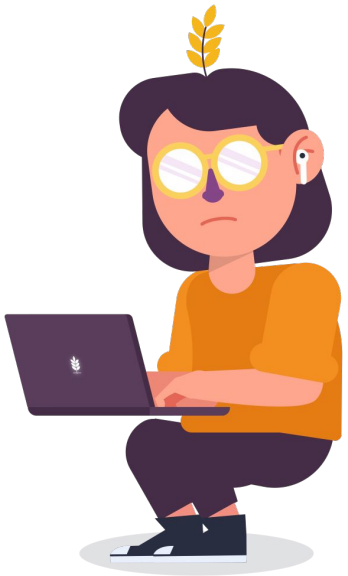
Ketika developer lain mengupload pekerjaan mereka di remote repository, maka akan terjadi perubahan di sana.

Supaya pekerjaan tim bisa lancar, kita harus sinkronisasi perubahan tersebut ke local repository kita. Caranya, dengan perintah **git pull origin master**

Maksud dari perintah tersebut adalah kita menyuruh git untuk mengunduh perubahan terbaru yang terjadi di branch master ke local repository.

Bagaimana? Sudah paham kan?

Oke, sekarang kita bahas **Branching**, ya!

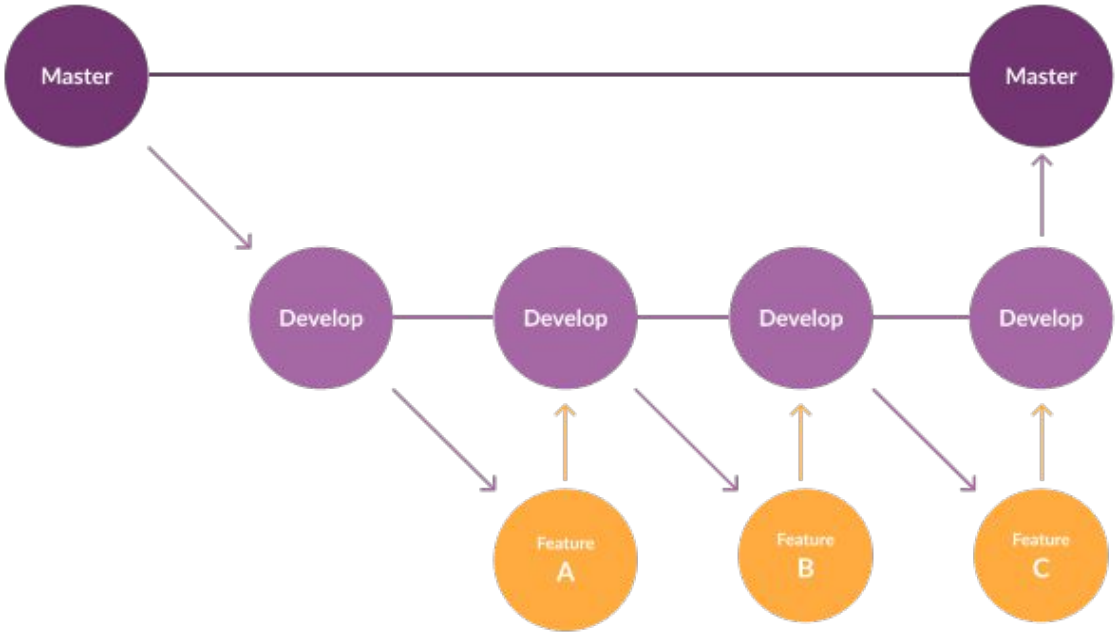


Apa itu Branch?

Branch berguna untuk menduplikasi kode kita di tempat baru. Dengan sistem branching ini, kita bisa melakukan kolaborasi dengan developer lain.

Misal, developer A mengerjakan fitur A dan developer B mengerjakan fitur B. Mereka berdua akan mengerjakan tugasnya di tempat masing-masing. Nah, setelah mengerjakan fitur masing-masing, tiba waktunya bagi developer A dan developer B untuk menggabungkan hasil pekerjaan mereka.

Dengan branch, kode untuk fitur A akan bergabung dengan kode untuk fitur B tanpa menyebabkan kekacauan di kode mereka.



Gambar di atas adalah contoh flow saat bekerja dengan branch

Ketika kita membuat project, sangat disarankan untuk membuat satu branch bernama "develop". Branch master hanya digunakan untuk kode yang sudah stabil dan tidak ada major bugs. Sedangkan, branch develop berguna untuk menyimpan kode-kode terbaru dan dijadikan tempat melakukan tes kode.

Cara ini termasuk umum dilakukan dalam praktik coding, lho. Jadi, sebelum melakukan penggabungan branch develop ke master, biasanya akan dilakukan beberapa tes untuk memastikan bahwa kode yang kita bikin sudah stabil.

Mengapa begitu? Karena, branch master adalah branch dimana kode kita akan dipakai langsung oleh User. Bisa gawat bila User yang menemukan kendala dalam kode kita...

Biasanya, pada fase *development*, branch develop digunakan sebagai tempat menyimpan update dari fitur terbaru dari kode kita.

Contoh, kita ingin memiliki fitur registrasi. Ketika hendak mengerjakan fitur tersebut, maka kita harus membuat *branch* baru bernama “feature/register” dan melakukan perubahan di *branch* tersebut.

Setelah selesai, maka kita akan gabungkan branch feature/register ke *branch develop* untuk dites terlebih dahulu. Setelah fase development selesai, baru kita gabungkan *branch develop* dengan *master*.

Ketika kita bekerja dengan Branch, maka akan sangat dimungkinkan terjadi *commit behind*.

Apa itu *commit behind*? Ketika kita mengajukan penggabungan branch, ada kemungkinan *source branch* ketinggalan update dari target branch. Maka dari itu, source branch harus melakukan **git pull origin target-branch** terlebih dahulu untuk mendapatkan update dari *target branch*.

Nah, itulah yang dimaksud dengan *commit behind*.

Sangat tidak direkomendasikan kita memaksakan penggabungan ketika ada commit behind. Kenapa? Karena update dari target branch bisa saja terhapus karena source branch belum memilikinya.

Soal commit behind sendiri, ada contoh menarik nih:

Developer A mengerjakan fitur A dan developer B mengerjakan fitur B di waktu yang bersamaan. Ternyata,Developer B mampu selesai duluan. Karena itu, ia melakukan penggabungan branch fitur B ke develop. Selanjutnya, branch develop pun mendapat perubahan dari branch B.

Tak lama berselang, Developer A selesai dengan fitur A dan mengajukan penggabungan branch. Namun, Developer B menyarankan Developer A untuk tidak memaksakan hal tersebut.

Apa alasannya? Hal ini dikarenakan sudah ada perubahan dari fitur B pada branch develop, sedangkan di fitur A belum memiliki perubahan itu. Kalau dipaksakan, nanti kodenya jadi berantakan deh...

- Git merupakan sebuah *version control,* atau aplikasi yang berguna untuk mengatur versi dari kode/aplikasi kita.
- 4 tahapan menyimpan pekerjaan dengan saat menggunakan git :

    1. Working directory
    2. Staging Area
    3. Local Repository
    4. Remote Repository

- Layanan untuk membuat remote repository diantaranya Github, Gitlab, dan Bitbucket
- Branching berguna untuk menduplikasi kode kita di tempat baru. Dengan sistem branching ini, kita bisa melakukan kolaborasi dengan developer lain.

BINAR

Saatnya

Quiz

BINAR

1

Apa itu git ?

A. Aplikasi version control

B. Framework

C. Bahasa pemrograman

BINAR

2

Manakah perintah berikut yang tepat untuk menginisialisasi directory project dengan git ?

A. git create

B. git init

C. git make

BINAR

3

Perintah git yang digunakan untuk menyambungkan local dan sebuah remote repository yang benar adalah ...

A. **git remote add origin git@gitlab.com:namamu/nama-project.git**
B. git add remote origin git@gitlab.com:namamu/nama-project.git
C. git origin add remote git@gitlab.com:namamu/nama-project.git

BINAR

4

Perintah git yang digunakan untuk menyimpan hasil pekerjaan ke remote repository di branch master adalah ...

A. git fetch origin master
B. git pull origin master
C. git push origin master

BINAR

5

Salah satu masalah yang sering terjadi ketika menggabungkan suatu pekerjaan di branch tertentu adalah adanya commit behind.

Commit behind ini terjadi ketika ...

A. **Source branch ketinggalan update dari target branch**
B. Target branch ketinggalan update dari source branch
C. Source branch tidak ada isinya

Referensi

- https://www.petanikode.com/tutorial/git/

BINAR

Terima

Kasih