

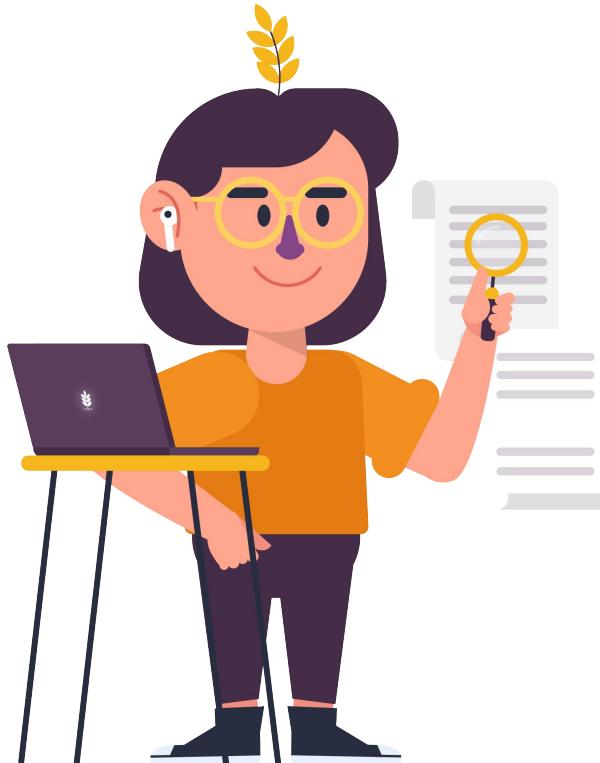


**Selamat datang di Chapter 3 Topik 2 online
course Full Stack Web dari Binar Academy!**



Setelah kamu selesai dijelaskan tentang dasar-dasar membuat halaman web dan juga bahasa pemrograman Javascript, **Chapter 3 ini bakalan ngajak kamu buat belajar cara menjalankan kodingan dan cara berkolaborasi dengan developer lain dalam mengembangkan sebuah sistem.** Mulai dari penjelasan terkait terminal, IDE, GIT, membuat web layout dan responsive design.

Pada topik kedua ini, kita bakal bahas tentang **GIT**. Cus langsung aja~



Detailnya, kita bakal bahas hal-hal berikut ini:

- Definisi, Fungsi, dan Sejarah Git
- Tahap Penyimpanan di Git
- Perintah (Command) Dasar dalam Git
- Praktek Penggunaan Git



Pertama, coba kita liat apa sih Git itu?



Eitss, sebelum kita mulai, ada satu hal yang perlu kamu ingat!

Directory/direktori itu memiliki arti yang sama dengan Folder.

Jadi, kalo selama pembelajaran kamu menemukan kata direktori, itu artinya adalah sebuah folder.



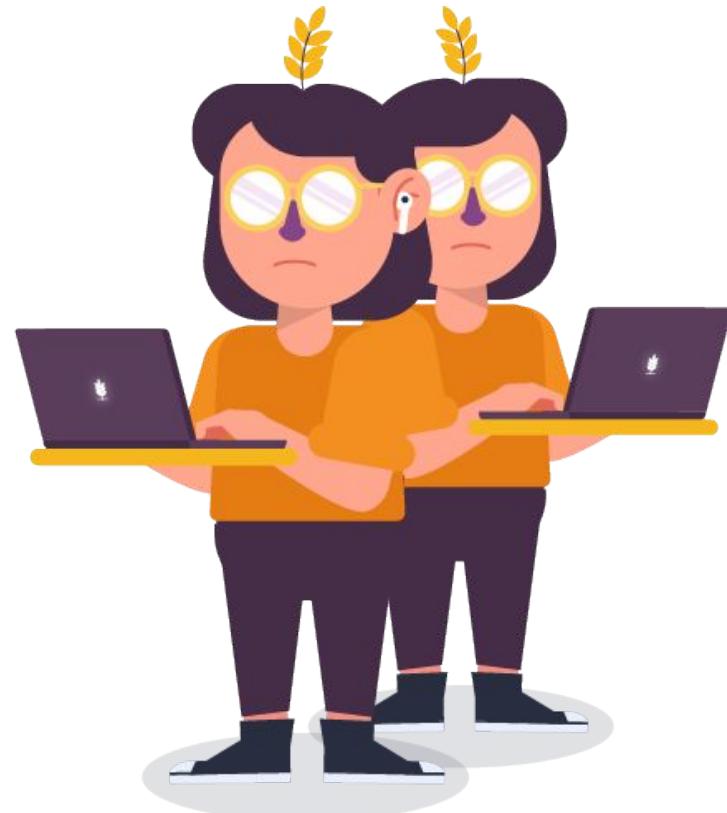


Git adalah sebuah version control. Maksudnya, Git adalah sebuah aplikasi yang berguna untuk mengatur versi dari kode/aplikasi kita.

Dalam dunia pemrograman, tentu kita akan berkolaborasi dengan programmer lain. Nah, untuk berkolaborasi, tentu kita perlu sesuatu yang bertanggung jawab untuk mengatur tim kita.

Dengan adanya Git, kita dapat dengan mudah berkolaborasi dengan orang lain. Karena di dalam Git, semua perubahan di dalam kode kita, akan tersimpan ke dalam history. Jadi, ketika kita menemui kode, kita bisa melihat history-nya dan siapa yang melakukan perubahan tersebut. Sehingga, kita bisa dapat dengan mudah memperbaikinya.

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



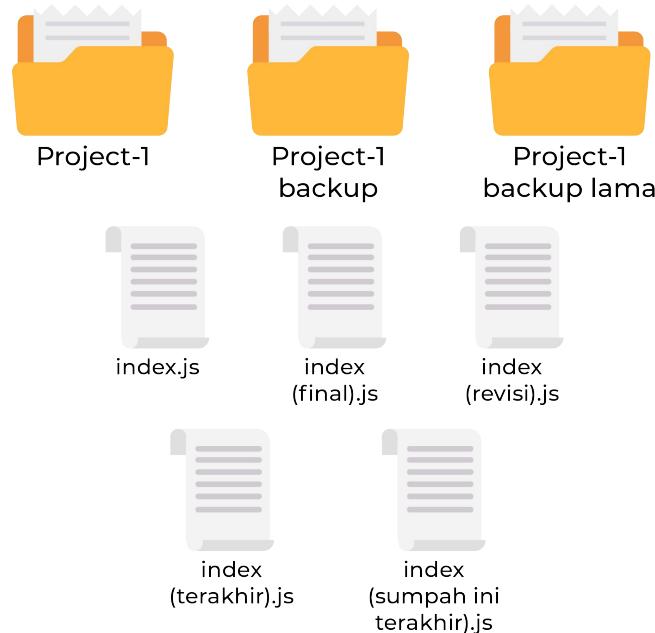


Bayangan pas kamu lagi bikin tugas kuliah~

Ketika kamu ngerjain tugas kuliah atau skripsi, pasti kamu bakal bikin backup-an data, kan? Atau bakal dikerjakan di google docs supaya setiap perubahan bisa di-track. Jadi, kalo tiba-tiba ada yang kehapus atau salah input, kamu dapat dengan mudah meng-undo nya!

Nah, programmer juga butuh sesuatu yang bisa nge-backup dan nge-track pekerjaan mereka. Apalagi pekerjaan itu dilakukan bareng-bareng.

Dengan menggunakan Git, pekerjaan akan ke-track dan ter-backup di dalam Git dalam bentuk history.





Udah kebayang kan fungsi Git? Sekarang, kita lanjut bahas sejarahnya!

Git ini dibuat oleh **Linus Torvalds** (Pencipta Linux). Bermula ketika beliau membuat project linux-nya menjadi open-source. Banyak orang yang ingin berkolaborasi dalam project tersebut. Karena saking banyaknya orang yang berkolaborasi, sering terjadi perubahan kode dan tidak jelas siapa yang di balik perubahan tersebut. Permasalahan itu cukup mengganggu penggerjaan project Linux.

Nah, untuk mengatasi masalah tersebut, Linux membuat sebuah aplikasi bernama Git, yang digunakan untuk mengelola perubahan kode dalam project Linux.



**Sebelum kita mempraktekkan penggunaan
Git, kita perlu tahu dulu, gimana cara Git
menyimpan pekerjaan kita?**

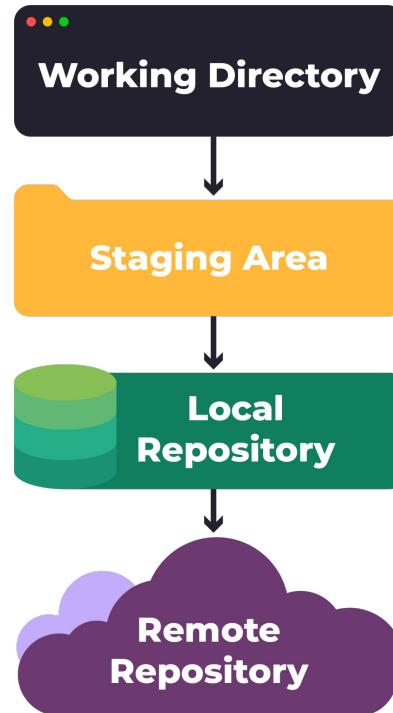




Ketika bekerja dengan Git, Git akan menyimpan folder atau pekerjaan kita dalam empat tahap.

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

Kita bahas satu-persatu, yuk!

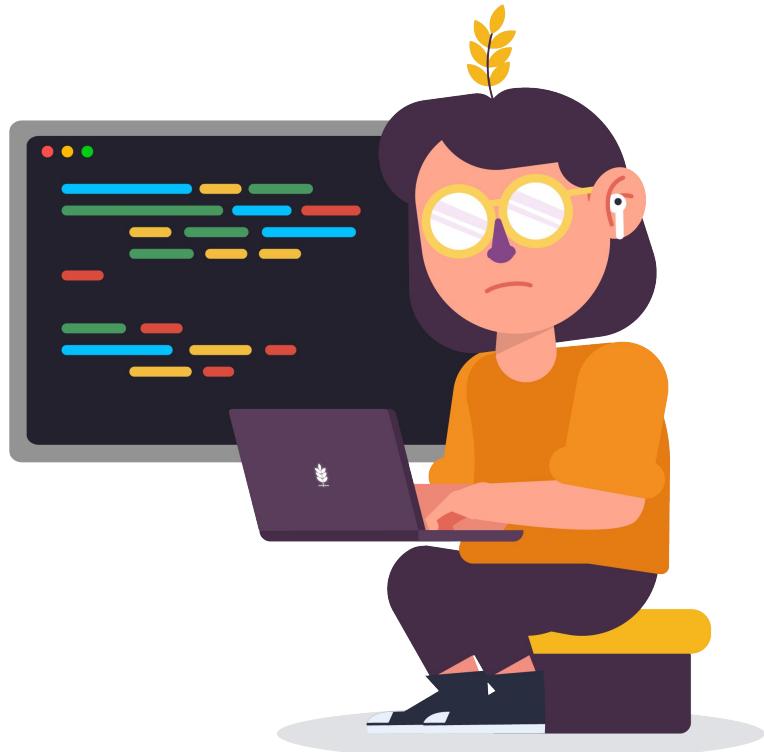




1. Working Directory

Sesuai namanya, working directory ini adalah **sebuah direktori untuk menyimpan kode yang masih dalam tahap penggerjaan**, alias belum selesai.

Misal, kita sedang mengerjakan sederet baris kode dalam file index.js lalu kita save. Maka, file index.js tersebut akan berada dalam working directory Git.





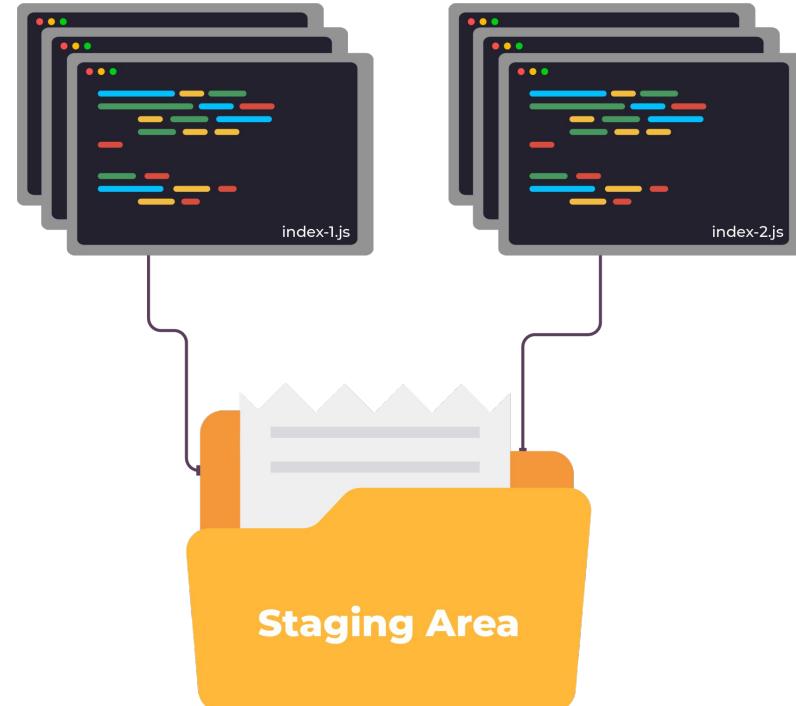
2. Staging Area

Tahap dimana perubahan yang sudah dibuat siap untuk di-commit (dibuatkan berita acara).

Misal, kita sudah selesai menambahkan sederet baris kode dalam *index.js* dan ingin melaporkannya kepada developer lain.

Nah, untuk melaporkannya, kita harus memindahkan *index.js* dari *working directory* ke *staging area* terlebih dahulu.

Intinya, **staging area merupakan tempat untuk mengorganisasikan perubahan yang dilakukan untuk dimasukkan dalam berita acara**.



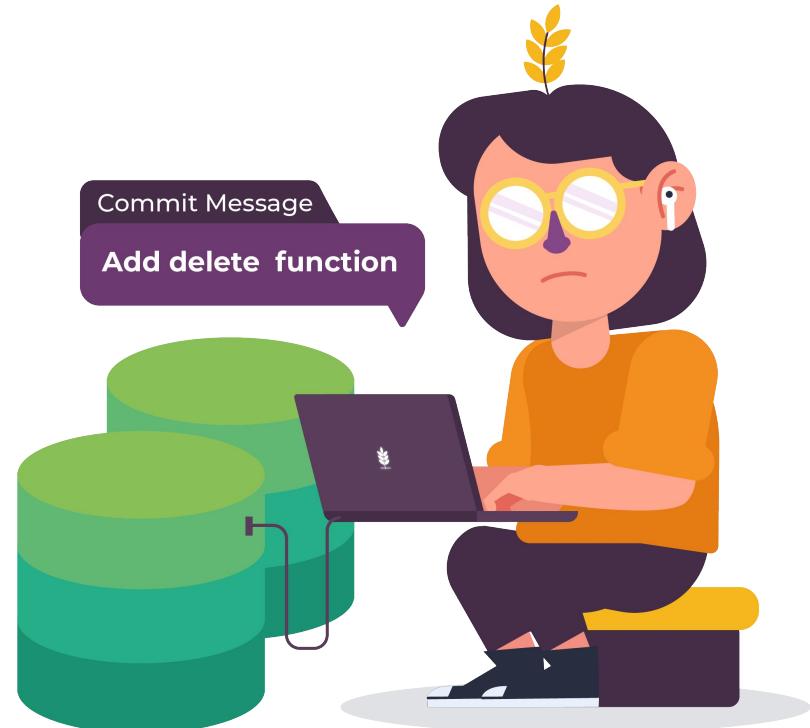


3. Local Repository

Tahap dimana perubahan sudah ditetapkan (fixed), tapi perubahan tersebut hanya terjadi di dalam komputer (belum di-upload ke cloud).

Perubahan yang masuk dalam local repository memiliki yang namanya **commit message**. Commit message adalah pesan yang menginformasikan apa yang telah dikerjakan atau apa yang diubah dari sebuah file.

Misal, kamu menambahkan sebuah function delete dalam file index.js. Setelah selesai dikerjakan dan siap untuk disimpan dalam local repository, jangan lupa untuk menambahkan keterangan perubahannya atau commit message.



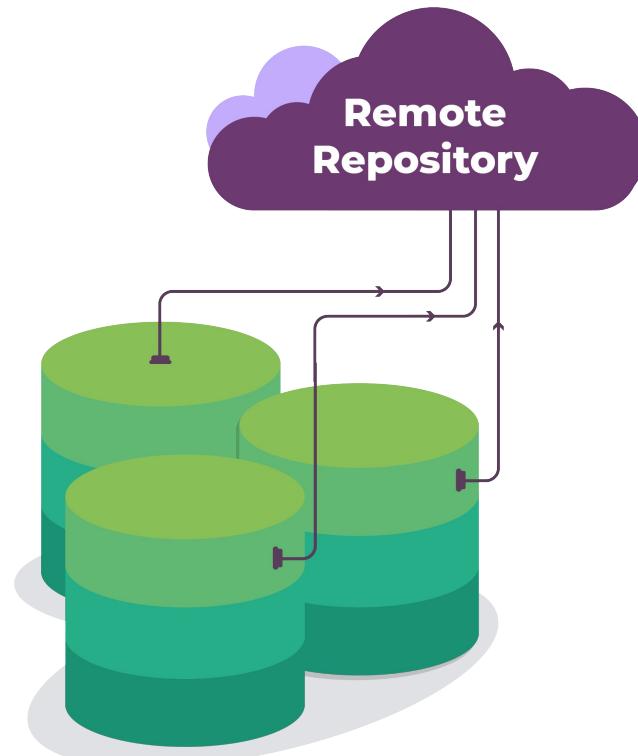


4. Remote Repository

Tahap dimana perubahan yang dilakukan sudah berada di cloud dan dapat diakses oleh developer lain.

Misal, file index.js di local repository bakal kita upload ke remote repository. Nah, perubahan yang terjadi di file index.js tersebut akan otomatis berada di cloud.

Ya kayak pas kita upload tugas ke Google Drive gitu deh~



Sampai disini, kamu udah cukup paham kan tentang Git?

Biar lebih paham, mari kita perdalam melalui praktik langsung penggunaan Git!

Ingat kata pepatah,
Practice makes perfect!





Sebelum kita praktek langsung, **Sabrina mau ngasih tau dulu nih beberapa istilah atau perintah-perintah (command) dasar dalam Git yang masing-masing punya fungsi beda-beda.** Ada yang berfungsi membuat branch, menambahkan file, mengupload file, dan lain-lain.

Cus langsung cek di halaman selanjutnya~





BRANCHES.

git branch	Untuk melihat list semua branch lokal
git branch -a	Untuk melihat list semua branch lokal dan branch pada remote repo
git checkout -b branch_name	Membuat branch lokal baru dan langsung pindah ke branch baru tersebut
git checkout branch_name	Pindah ke sebuah branch
git push origin branch_name	Upload semua perubahan pada lokal branch ke remote branch
git branch -m new_name	Mengubah nama sebuah local branch
git branch -d branch_name	Menghapus local branch
git push origin :branch_name	Menghapus remote branch

LOGS.

git log --oneline	Melihat riwayat commit secara singkat
git diff	Melihat semua changes
git diff myfile	Melihat changes pada file tertentu

CLEANUP.

git clean -f	Menghapus semua file yang tidak ke track
git clean -df	Menghapus semua file dan folder yang tidak ke track
git checkout -- .	Undo semua changes pada local

Terus, penggunaannya kaya gimana ya? Yuk langsung aja kita praktokin stey-by-step!





1. Instal Git terlebih dahulu

Tahap pertama yang harus kamu lakukan adalah menginstal Git di Linux. Karena Git ini adalah aplikasi di terminal, jadi, untuk instalasinya bisa dilakukan di terminal. **Buka terminal di Linux, lalu ketik perintah di bawah ini!**

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

Untuk MacOS, bakal dijelasin di halaman selanjutnya ya!



Untuk MacOS

Buka terminal, lalu ketik seperti di bawah ini untuk menginstal brew dan Git.

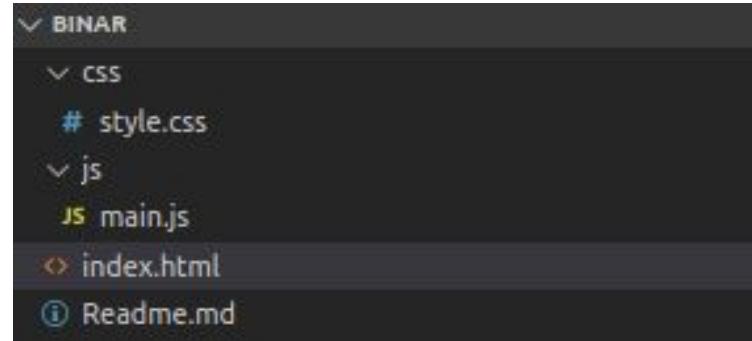
```
# Install brew dulu, yah
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
brew doctor

# Install Git nya
brew install git
```



2. Inisialisasi Git di folder project kamu

Misal kamu sedang menjalankan project bernama **Binar**. Nah, semua file project itu kamu simpan dalam folder bernama **Binar** seperti gambar di bawah ini.





Terus, pas kamu mau mengimplementasikan Git dalam project tersebut, yang harus kamu lakukan adalah: **Masuk ke terminal, buka folder Binar lewat terminal, lalu inisialisasi Git dengan mengetik perintah seperti di bawah ini:**

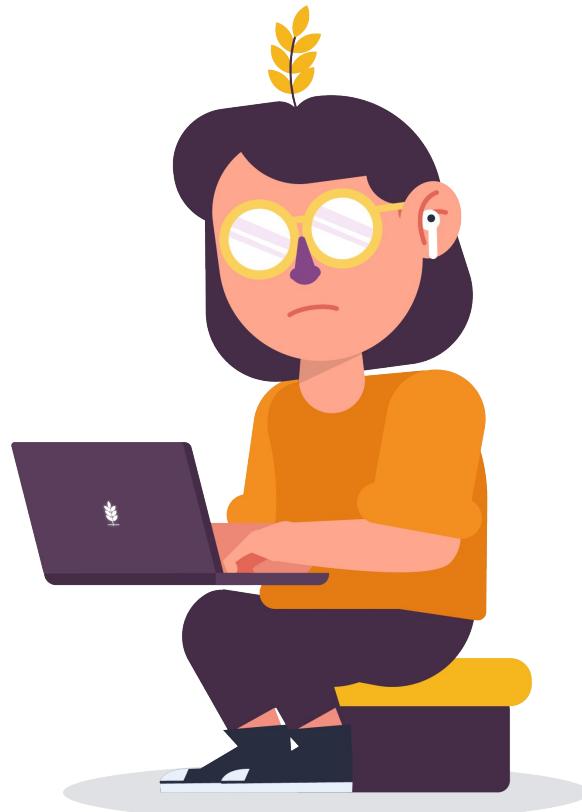
```
cd lokasi/folder/pekerjaan/mu # Untuk pindah lokasi ke folder-mu  
git init # Untuk menginisialisasi git
```



Setelah kamu menginisialisasi Git, maka terminal akan berada dalam folder project kamu.

Selanjutnya, **kamu bisa menjalankan perintah-perintah Git di dalam terminal seperti berikut ini:**

- **git status** - untuk melihat apa saja dokumen update yang ada di repository.
- **git push** - untuk mengirimkan coding yang sudah dikerjakan.
- **git pull** - untuk menarik coding yang ditulis developer lain.
- **git commit** - sebagai penanda sebelum dilakukan git push.





Sekarang, coba kamu jalankan 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 inisialisasi project, semua file yang berada dalam folder itu akan masuk ke tahap Working Directory. Nah, ada baiknya kamu langsung pindahkan file ke dalam Local Repository dengan melakukan **commit** terlebih dahulu.

Commit pertama ini, disebut **Initial Commit**. Untuk melakukan **commit**, kamu hanya perlu menjalankan dua perintah di bawah 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 kamu perhatikan wujud file dan folder kamu di dalam project, keliatannya nggak ada perbedaan sama sekali. Karena, pemindahan stage hanya terjadi di dalam Git itu sendiri, dan kita nggak perlu tau gimana Git melakukan itu. Tapi intinya, perubahan yang kamu lakukan udah ke catat kok!



3. Lakukan perubahan, lalu commit

Setelah selesai melakukan inisialisasi, hal selanjutnya adalah melakukan perubahan di dalam pekerjaan tersebut. Entah menambah kode baru, merubah kode, atau bahkan menghapus kode, pokoknya sesuai dengan kebutuhan pengembangan sistem.

Misal kamu punya file **index.html** dan ingin melakukan perubahan pada file tersebut.

```
<!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>
```



Setelah melakukan perubahan, kamu harus melakukan **commit**. Tujuannya, untuk menandakan bahwa perubahan tersebut telah dicatat oleh Git.

Sebelum melakukan **commit**, pastikan apa yang kamu commit itu **benar**. Kamu bisa jalankan **git status** untuk mengecek apa saja yang berubah. Nanti outputnya akan seperti di bawah ini!

```
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")
```

Ini menunjukkan bahwa Git udah melakukan tracking di folder kamu. Silahkan kamu cek ulang apakah file index.html yang kamu ubah sudah benar.



Kalo udah benar, baru deh sekarang kamu lakukan commit. Untuk melakukannya, kamu harus pindahkan perubahan yang kamu lakukan dari **working area** ke **staging area**. Cara memindahkannya mudah, kok. Kamu cukup jalankan perintah ini~

```
git add namafileita
```

Karena yang kamu ubah adalah file **index.html**, maka kamu harus **pindahkan** file **index.html** ke **staging area**.

```
git add index.html
```

Nah, selanjutnya kamu bisa cek apakah file index.html tersebut udah dipindah ke staging area. Caranya, **cek menggunakan git status**.

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

    modified:   index.html
```

Kalo warnanya udah **hijau**, itu artinya file kamu udah masuk ke **staging area**.



Setelah perubahan masuk ke staging area, selanjutnya kamu harus meng-commit perubahan tersebut. Agar perubahannya dipindah ke **Local Repository**.

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

Kamu bisa menggunakan git status untuk memastikan apakah perubahan tersebut sudah di-commit.

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

Kita udah bahas penggunaan Git di Local Repository, sekarang kita lanjut gimana penggunaan Git di Remote Repository!

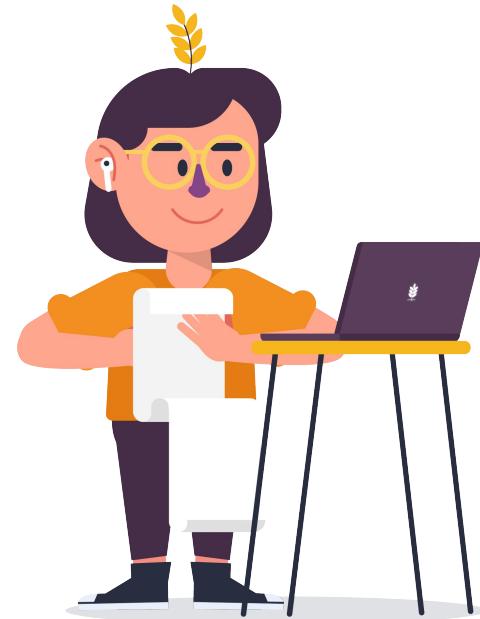




Apa sih bedanya local repository dan remote repository?

Sebelum kita bahas gimana penggunaan Git di remote repository, kamu perlu tahu dulu apa itu remote repository dan apa perbedaannya dengan local repository.

Perbedaan keduanya terletak di lokasi penyimpanan. Kalo **Local repository**, artinya folder tersimpan di device kamu. Kalau **remote repository**, folder tersimpan di cloud dan bisa diakses oleh developer lain.





Berikut ini tiga layanan remote repository yang paling terkenal:

- Github
- Gitlab
- Bitbucket

Nah, kita bakalan praktek menggunakan **Gitlab**. So, pastikan kamu telah membuat akun Gitlab, ya! Kalo udah punya, kamu bisa langsung membuat repository untuk proyek kamu di Gitlab tersebut.

Bingung caranya? Cek slide berikutnya deh~





Blank project Create from template Import project CI/CD for external repo

Project name
Test

Project URL
 

Project slug
test

Want to house several dependent projects under the same namespace? [Create a group](#).

Project description (optional)
This a test

Visibility Level 

 Private
Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.

 Public
The project can be accessed without any authentication.

Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project 



Setelah berhasil **create project**, maka kamu perlu menyalin alamat dari repository itu dengan menekan tombol **Clone**, lalu pilih *Clone with HTTPS* kalo belum setup SSH Key.

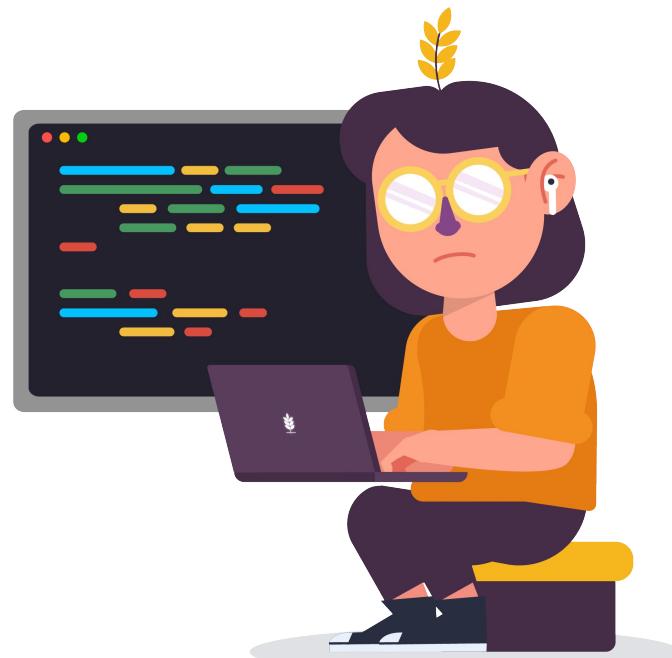
Untuk menyambungkan ke remote repository, kita akan menggunakan perintah Git remote, perintah ini akan mengurusi segala sesuatu yang berbau konfigurasi remote repository. Perintahnya seperti ini: **git remote add origin git@gitlab.com:namamu/nama-project.git**

Fungsi perintah di atas adalah memberi tahu local repository tentang alamat remote repository yang akan menjadi tempat tujuan mengupload file.

Abis itu, kamu bisa menjalankan suatu perintah untuk mengetahui alamat remote repository, yaitu: **git remote -v**

Untuk menghapus alamat remote repository, kamu bisa menggunakan: **git remote remove origin**

Untuk mengubah alamat remote repository, gunakan perintah ini: **git remote set-url origin git@gitlab.com/alamat/baru.git**





Git Push dan Git Pull

Wih apaantuh? Kayak di pintu minimarket aja~

Git Push dan Git Pull ini digunakan agar directory project yang ada di setiap developer merupakan directory yang up-to-date.

Git Push berfungsi untuk mengupload pekerjaan ke remote repository. Caranya, menggunakan perintah: **git push origin master**

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

Tapi... sangat tidak dianjurkan untuk mengupload langsung ke branch master, yaa!

Kita bakal bahas lebih lanjut soal branch setelah ini.





Sekarang, coba kita bahas dulu apa itu Git Pull.

Ketika developer lain mengupload pekerjaan mereka di remote repository, maka pasti akan terjadi perubahan kan? Sebagai sesama tim, sudah semestinya perubahan itu relevan dengan pekerjaan kita. Yaa biar sama-sama up-to-date gitu deh!

Nah, disitulah fungsi Git Pull. **Yaitu mensinkronisasikan perubahan dari remote repository ke local repository.** Caranya, dengan menggunakan perintah: **git pull origin master**

Maksud dari perintah di atas adalah, kita menginstruksikan Git untuk mengunduh perubahan terbaru yang terjadi di branch master ke local repository.



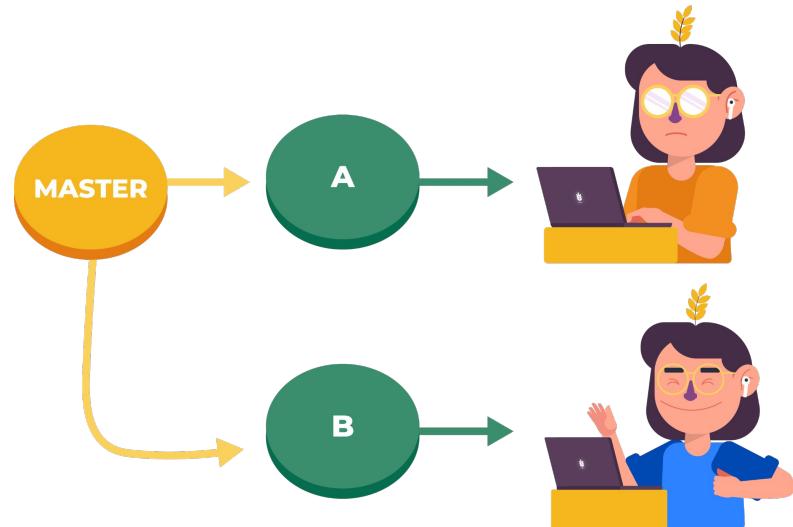


Berkolaborasi lewat branching~

Branch berguna untuk menduplikasi kode di tempat baru, dimana kamu bisa dengan mudah menggabungkan branch A dengan branch B.

Dengan **sistem branching** inilah, kamu **bisa melakukan kolaborasi dengan developer lain**. Jika developer A mengerjakan fitur A dan developer B mengerjakan fitur B, mereka berdua akan mengerjakannya di branch masing-masing. Dengan begitu, nggak akan terjadi tumpang tindih antara kode fitur A dengan fitur B.

Ketika kamu ingin membuat suatu fitur atau memperbaiki bug, sangat disarankan untuk mengerjakannya di branch lain. Tujuannya supaya tidak terjadi konflik dan lebih mudah untuk memonitor perubahan.

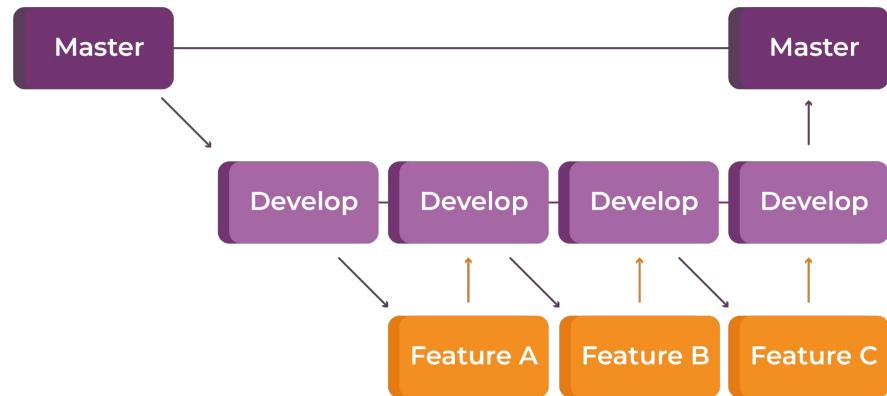




Ketika kamu membuat project, sangat disarankan untuk membuat satu branch bernama "develop". Soalnya, **branch master hanya digunakan untuk kode yang sudah stabil dan udah nggak ada lagi major bugs.** Sedangkan **branch develop berguna untuk menyimpan kode-kode terbaru dan menjadi tempat untuk melakukan tes kode.**

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

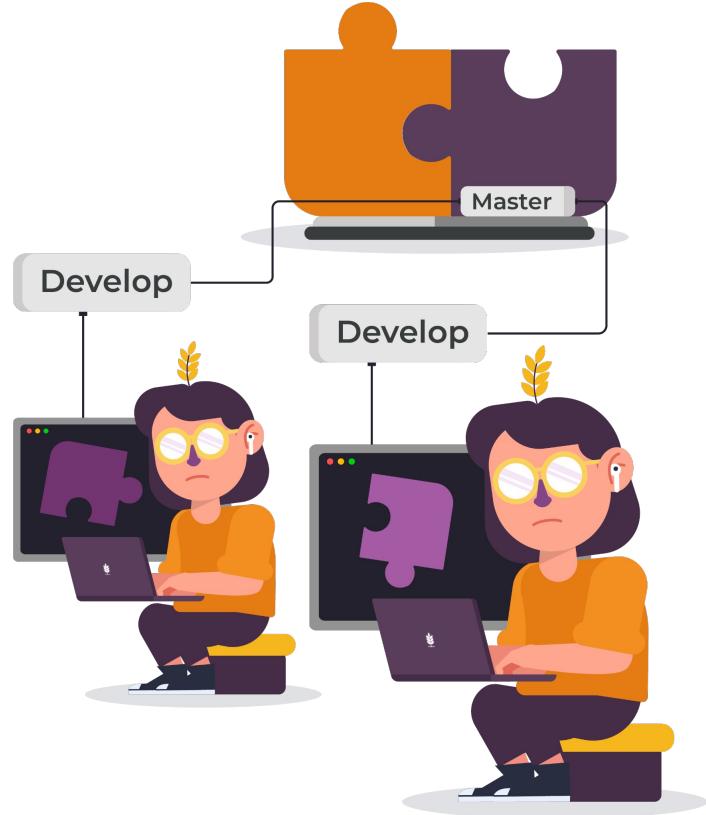
Kenapa harus kayak gitu? Soalnya, branch master merupakan branch dimana kode akan dipakai langsung oleh User. Bisa gawat kan kalo user menemukan eror dalam kode kita~





Pada fase development, biasanya branch develop digunakan sebagai tempat untuk menyimpan update fitur terbaru dari kode kamu. Misal, kamu punya fitur registrasi. Ketika kamu akan mengerjakan fitur registrasi tersebut, maka langkah yang harus dilakukan adalah:

1. Membuat branch baru bernama **feature/register** dan melakukan perubahan di branch tersebut.
2. Setelah selesai, kamu harus gabungkan branch **feature/register** ke branch **develop**, jangan langsung ke master.
3. Setelah fase development selesai, gabungkan branch **develop** dengan **master**. **Tapi, pastiin kode kamu** udah stabil.





Commit behind dalam branching.

Ketika bekerja dengan branch, sangat mungkin terjadi **commit behind**.

Apa itu commit behind? Ketika mengajukan penggabungan branch, ada kemungkinan source branch ketinggalan update dari target branch. Untuk 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**.

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





Soal commit behind, ada cerita menarik nih!

Alkisah, developer A mengerjakan fitur A dan developer B mengerjakan fitur B di waktu yang bersamaan. Ternyata, developer B kelar duluan. Setelah itu, developer B melakukan penggabungan branch fitur B ke develop. Branch develop pun mendapat perubahan dari fitur B.

Nggak lama berselang, developer B juga kelar. Abis itu, ia mengajukan penggabungan branch. Namun, developer B menyarankan agar developer A untuk tidak memaksakan penggabungan tersebut.

Alasannya? Karena udah ada perubahan dari fitur B pada branch develop, sedangkan pada fitur A belum memiliki perubahan. Kalo dipaksain, nanti kodennya bisa berantakan~



Saatnya kita Quiz!





1. Git Pull adalah command yang digunakan untuk...

- A. Mendownload commit terbaru dari directory.
- B. Mengupload commit terbaru ke directory.
- C. Melempar commit terbaru ke server database.



1. Git Pull adalah command yang digunakan untuk...

- A. Mendownload commit terbaru dari directory.
- B. Mengupload commit terbaru ke directory.
- C. Melempar commit terbaru ke server database.

Git Pull adalah command yang digunakan untuk mendownload perubahan terbaru dari directory.



2. Manakah perintah berikut yang tepat untuk menginisialisasi directory project dengan Git?

- A. Git create
- B. Git init
- C. Git make



2. Manakah perintah berikut yang tepat untuk menginisialisasi directory project dengan Git?

- A. Git create
- B. Git init
- C. Git make

Untuk menginisialisasi directory project dengan git, menggunakan perintah: **git init**



3. Perintah git yang digunakan untuk menyambungkan local repository dan remote repository 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



3. Perintah git yang digunakan untuk menyambungkan local repository dan remote repository 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

Untuk menyambungkan local repository dan remote repository, digunakan perintah git remote, yaitu: `git remote add origin git@gitlab.com:namamu/nama-project.git`



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



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

Untuk menyimpan hasil pekerjaan ke remote repository di branch master, digunakan perintah git push yaitu: **git push origin master**



- 5. Salah satu masalah yang sering terjadi ketika menggabungkan pekerjaan di branch tertentu adalah adanya commit behind. Commit behind terjadi ketika...**
- A. Source branch tertinggal update dari branch target.
 - B. Target branch tertinggal update dari source branch.



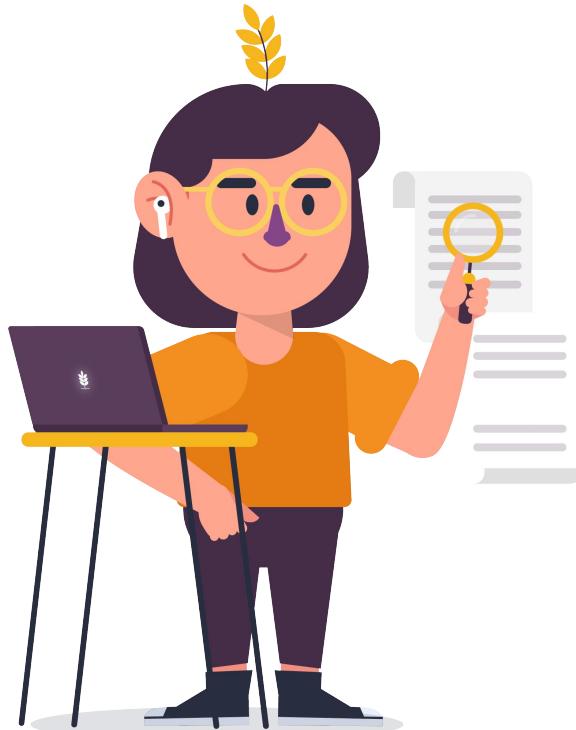
- 5. Salah satu masalah yang sering terjadi ketika menggabungkan pekerjaan di branch tertentu adalah adanya commit behind. Commit behind terjadi ketika...**
- A. Source branch tertinggal update dari target branch.
 - B. Target branch tertinggal update dari source branch.

Commit behind adalah masalah yang terjadi dalam penggabungan branch, dimana source branch ketinggalan update dari target branch.



Yuk eksplor lebih jauh!

-





Nah, selesai sudah pembahasan kita di Chapter 3 Topic 2 ini.

Selanjutnya, kita bakal bahas **Web Layout**.

Penasaran kayak gimana? Cus langsung ke topik selanjutnya~



Terima Kasih!



Next Topic

loading...