



GIT

Silver- Chapter 2 - Topic 5

**Selamat datang di Chapter 2 Topic 5 online course
Android dari Binar Academy!**





Haaii Binarian 🙌

Masih di Chapter 2 niih~

Pada Topik-topik sebelumnya, kita sudah bahas banyak tentang penggunaan Android Studio, aturan penulisan kodanya, hingga pembuatan layout sederhana.

Nah, untuk **Topik 5**, topik terakhir chapter ini, kita akan menyentuh salah satu tools lainnya, yaitu **version control GIT**, sebagai sarana untuk berkolaborasi dengan developer lainnya dalam membangun aplikasi.

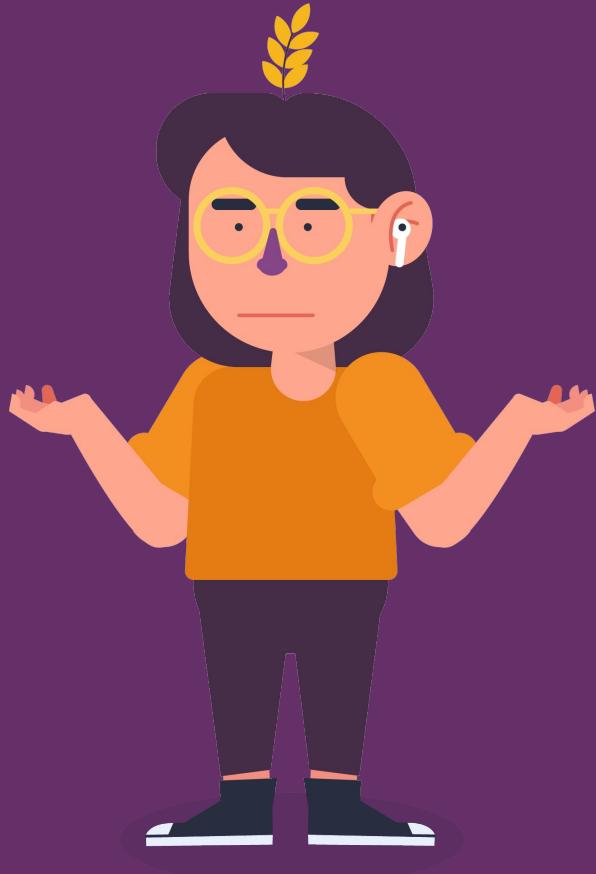
So, let's start~



Detailnya, kita bakal bahas hal-hal berikut ini :

- Pengenalan GIT
- Konsep GIT Flow





Okeey, kita mulai petualangan kita.
Bagi developer, memahami **Version Control** itu sangat penting.
Apa alasannya?



Kita mulai dengan cerita kita sehari-hari 📝

Coba deh inget-inget waktu sekolah/kuliah, siapa yang pernah melakukan revisi dengan model kayak gambar disamping ini?

Kalau diliat-liat lagi, ternyata cukup berantakan dan nggak teratur, kan? Selain itu, kita juga nggak bisa langsung tahu perubahan dari satu revisi ke revisi yang lain.



Makalah Bab V



Makalah Bab V Rev



Makalah Bab V Rev 2



Makalah Bab V Rev 3



Makalah Bab V Rev 3 + DaPus



Makalah Bab V Rev 3 + DaPus Revisi Final



Makalah Bab V Rev 3 + DaPus Revisi Final Banget



Makalah Bab V Rev 3 + DaPus Revisi Final Banget Kuadrat



Sama kayak revisi makalah/skripsi, pada pengembangan software Android, **perubahan atau revisi adalah hal yang sangat umum kita hadapi di dalam prosesnya.**

Bedanya, seorang developer wajib tahu kondisi software sebelum atau setelah dilakukan perubahan.

Dalam kondisi tertentu, bisa aja perubahan yang sempet dicoba ternyata batal diterapin dan harus kembali lagi ke versi sebelumnya.



Teammate 1

17:21

Eh tambahin fitur ini ya..

Teammate 2

17:22

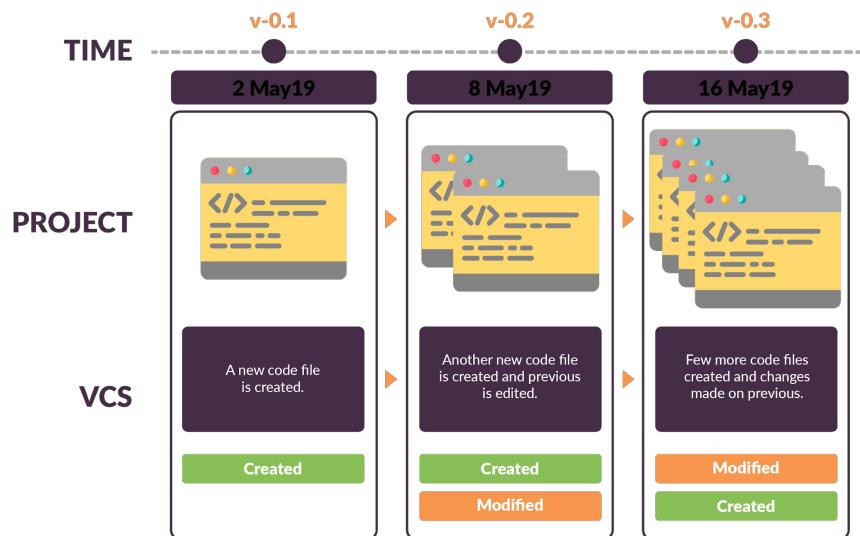
Kayaknya lebih bagus
kalau begini deh...

Teammate 1

17:22

Balikin lagi seperti yang
awal aja...

typing...



Kayak gini contohnya~

Makin banyak perubahan, makin membingungkan, bukan?

Karena itu, setiap perubahan yang terjadi harus kita simpan dengan baik.

Naah, bagaimana cara kita menyimpan perubahan versinya yang mudah ditarik saat kita butuh?



Nah, disinilah peran GIT 😊

GIT hadir sebagai pahlawan kita semua karena penggunaan **version control** yang ada di GIT membuat kita terhindar dari masalah-masalah ketika proses revisi dalam pengembangan Android dilakukan.



GIT akan berperan dalam hal kayak gini :

1. Menelusuri perubahan pada software yang sedang dikembangkan
2. Membuat beberapa versi sebagai opsi penawaran kepada customer/client
3. Mengulang atau membatalkan suatu perubahan.
4. Mengarsipkan kode-kode program sesuai histori perubahan.

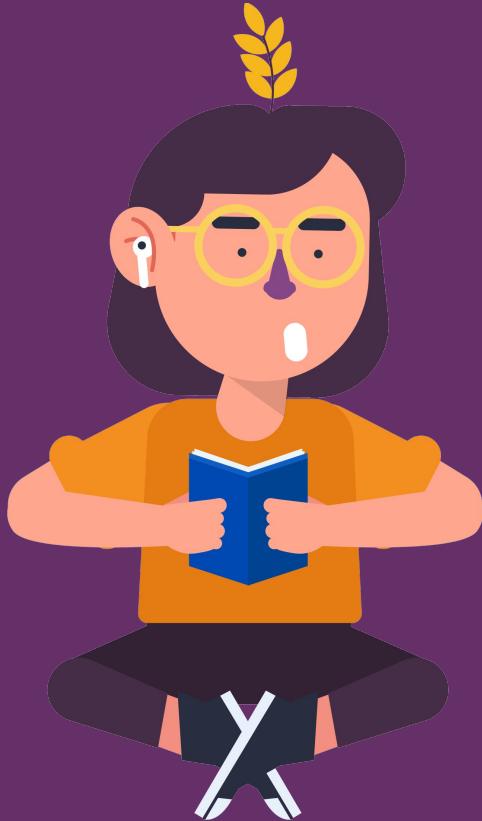
Dengan begitu, kita hanya perlu **satu file saja untuk setiap revisi!**



Supaya makin kebayang, kamu bisa simak video berikut untuk penjelasan GIT lebih lengkapnya :



Video Link Klik disini: [What is Git? Explained in 2 minutes](#)



Kalau kamu tadi sempet penasaran tentang **Version Control** yang ada di GIT, kamu tenang aja 😊

Setelah ini kita bakal bahas **Version Control** beserta tiga jenisnya.

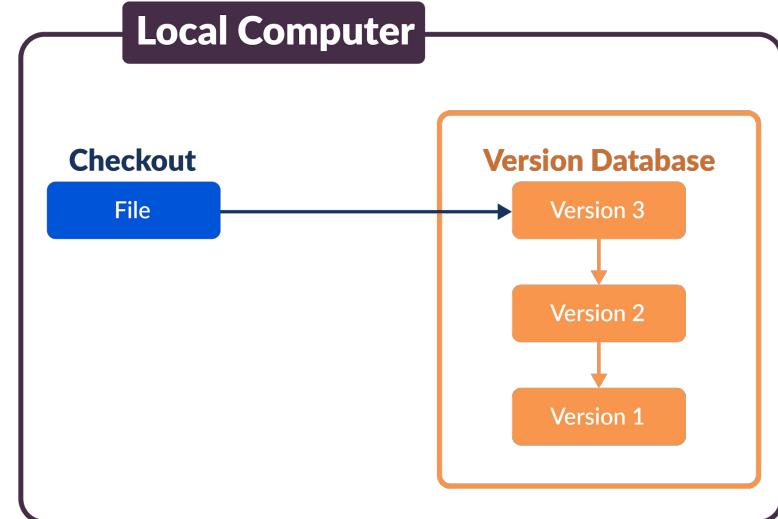
Ada apa aja tuh? Markijut, mari kita lanjut~

1. Local Version Control System

Version control jenis ini **hanya ada pada komputer pengguna**. Dimana di dalamnya, berkas **tidak** dibagikan dengan pengguna di komputer lain.

Dalam sistem ini, kolaborasi jadi nggak efektif karena hanya melibatkan satu pengguna pada satu komputer.

Karena itu, disarankan untuk menggunakan sistem ini kalo tujuan kamu adalah menyimpan dan menelusuri perubahan yang dilakukan oleh satu orang.



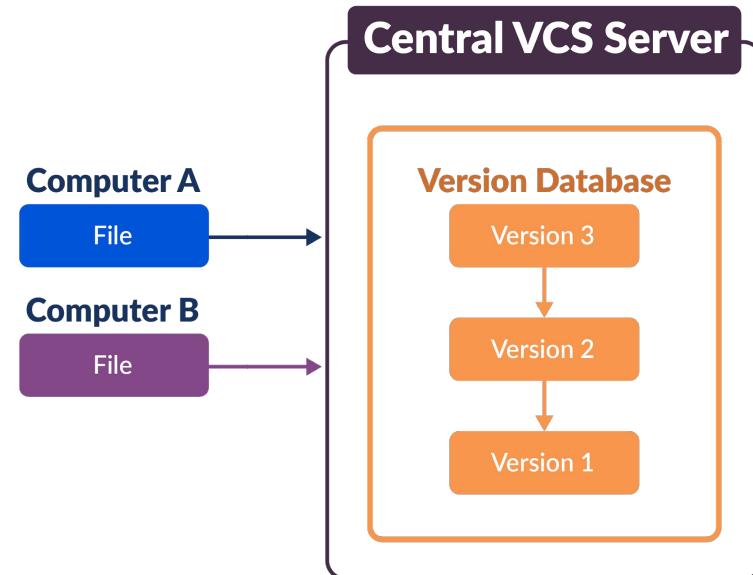


2. Central Version Control System

Version control yang bisa **menghubungkan dua komputer atau lebih untuk berkolaborasi**.

Biasanya, sistem ini digunakan pada **jaringan lokal** yang tidak melibatkan komputer client dalam jumlah besar.

Namun, central version control system tidak memiliki portabilitas yang baik. Berkas hanya bisa diakses melalui jaringan lokal. Sama seperti namanya “Central”, kalau tim developer tinggal di kota yang berbeda, versi ini tidak bisa diterapkan.



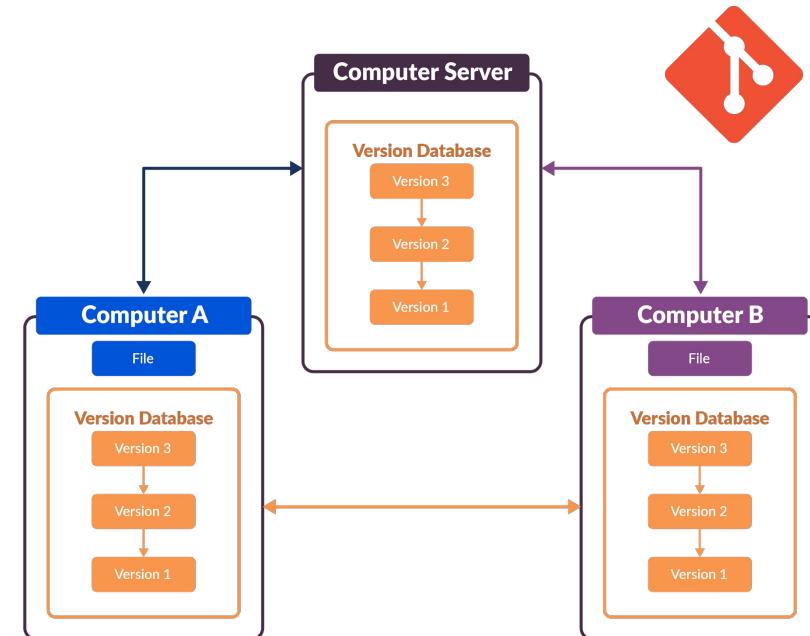


3. Distributed Version Control System

Sistem ini mengharuskan pengguna untuk mempunyai berkas version control pada masing-masing komputer. Tiap komputer akan **dihubungkan oleh server yang menangani version control secara keseluruhan.**

Server ini biasanya ada di cloud atau internet sehingga setiap pengguna bisa mengakses berkas dimana aja asalkan terhubung dan memiliki akses ke server.

Nah, versi ini lah yang akan kita bahas lebih dalam nantinya~





**Ngomong-ngomong tentang
Distributed Version Control
System, ada satu sistem
andalan yang bisa digunakan
pada Android Developer.**

**Dia adalah Git. Ciyeet ketemu
lagi~**



Kita coba review lagi pemahaman kita tentang Git 😊

Git merupakan salah satu perangkat *version control system* yang paling banyak digunakan oleh developer. Di dalamnya, terdapat banyak fitur, antara lain:

1. Version Control yang terdistribusi.
2. Operasi bersifat atomik (berhasil seluruhnya atau gagal seluruhnya).
3. Semua file disimpan pada folder .git.
4. Data model yang mendukung integritas data.
5. Staging area / index untuk mereview aplikasi sebelum ditayangkan untuk umum.



Google Cloud
Source Repository

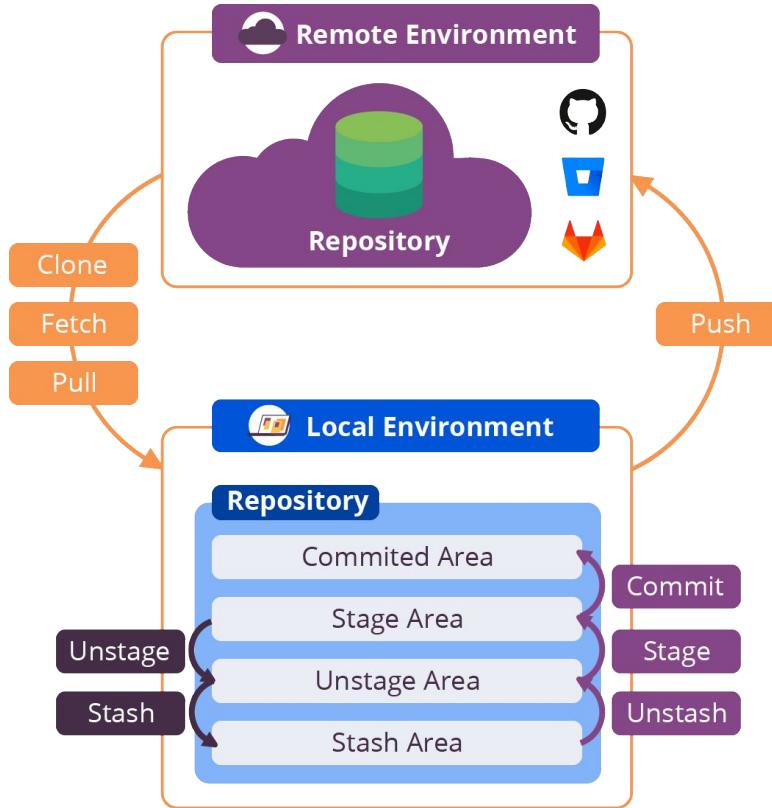


Tapi, nggak cuma Git aja kok~

Masih ada *version control system* lain, yang bisa kita pake juga! Ini beberapa contohnya:

- **Subversion** -> version control yang didukung oleh CollabNet.
- **Google Cloud Source Repository** -> version control yang didukung oleh Google.
- **Mercurial** -> version control yang didukung oleh Microsoft dan Unix.

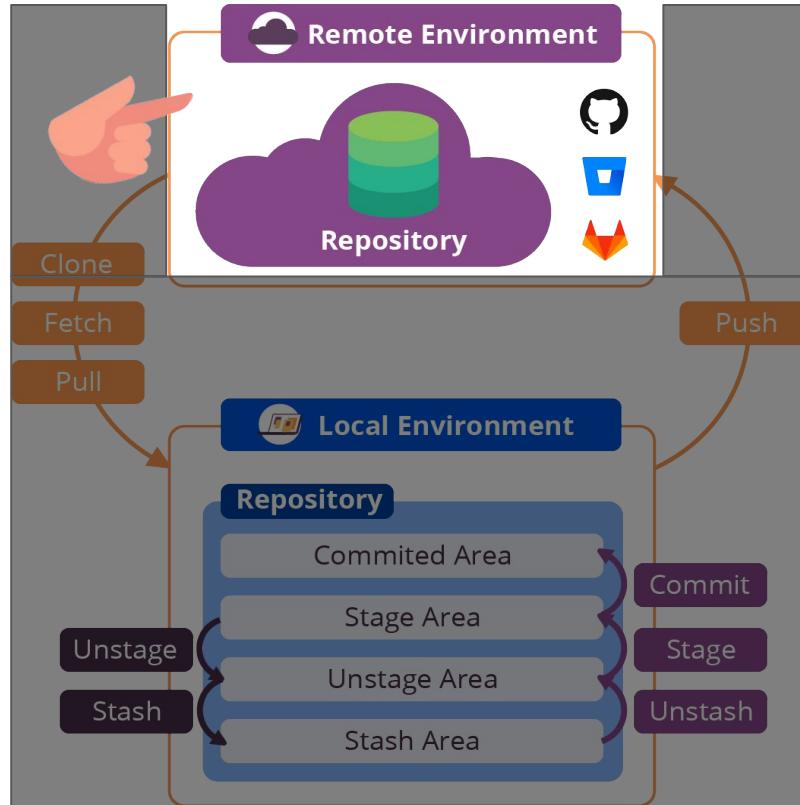
Namun, khusus untuk topik kali ini kita bakal bahas tentang penggunaan GIT secara lebih mendalam.



Kembali lagi ke pembahasan Git

Gambar disamping ini adalah seluruh ekosistem dalam **version control Git**. Ada Remote Environment ada Local Environment.

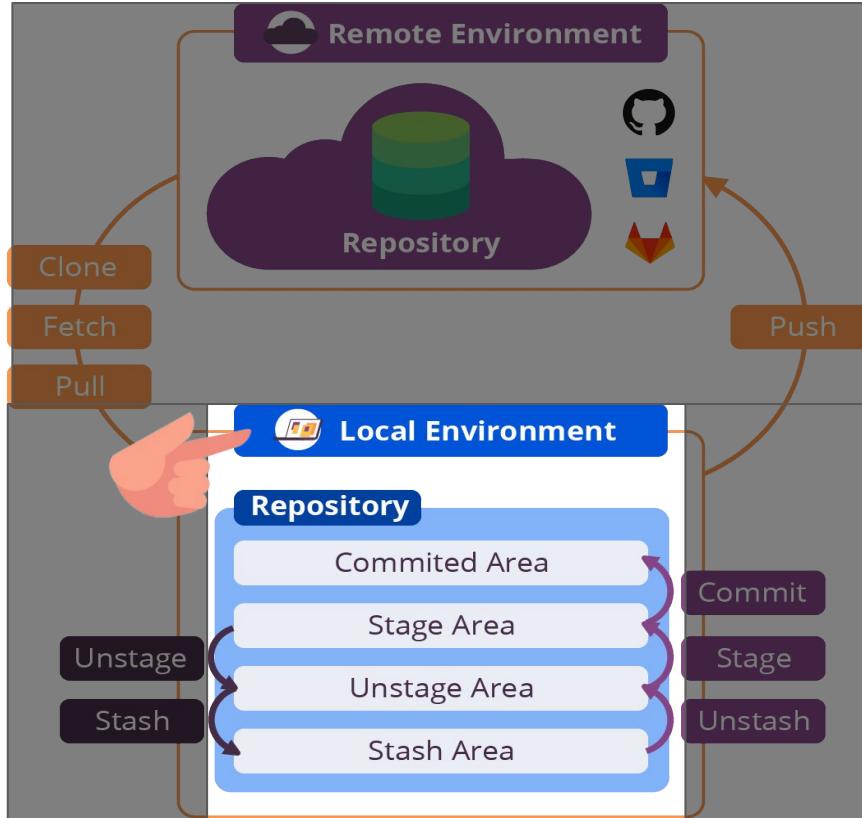
Kita bahas konsepnya satu per satu dulu yaa~



Remote Environment

Ini adalah penyedia layanan *Version Control* yang ada di internet. Contohnya kayak **Github**, **Gitlab**, **Bitbucket**, dll.

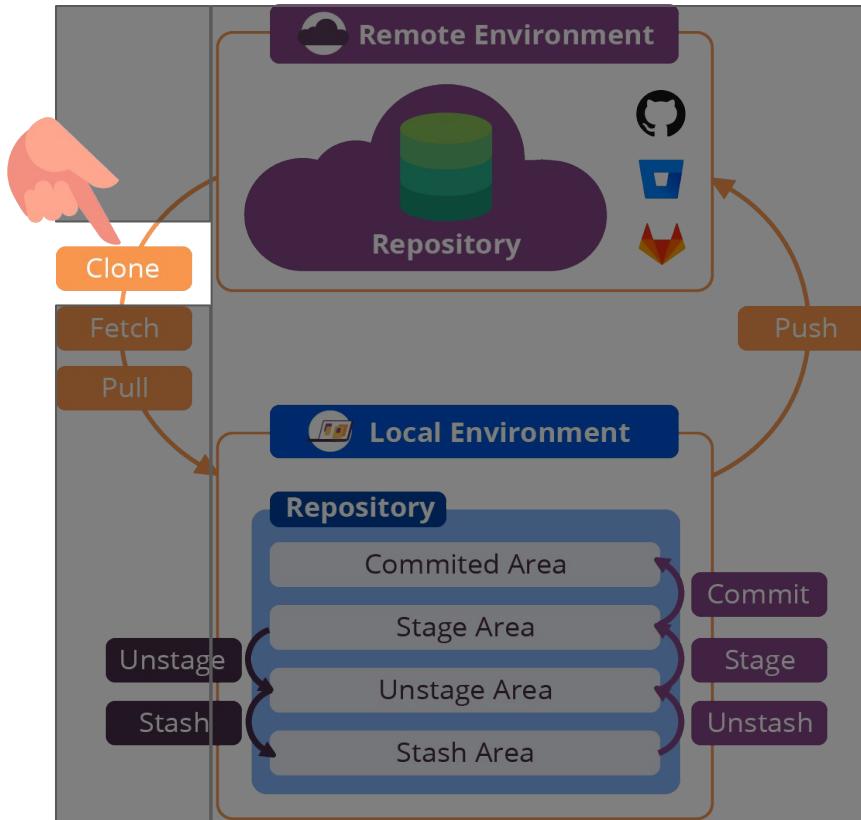
Pada *remote environment*, kita bisa menyimpan *repository* atau *project* aplikasi kita. *Repository* inilah yang membuat kita bisa mengakses *version control* dari mana aja asalkan terhubung dengan internet.



Local Environment

Ini adalah *version control environment* yang ada pada komputer kita. Di dalam Local Environment, bisa terdapat banyak repository (*project*).

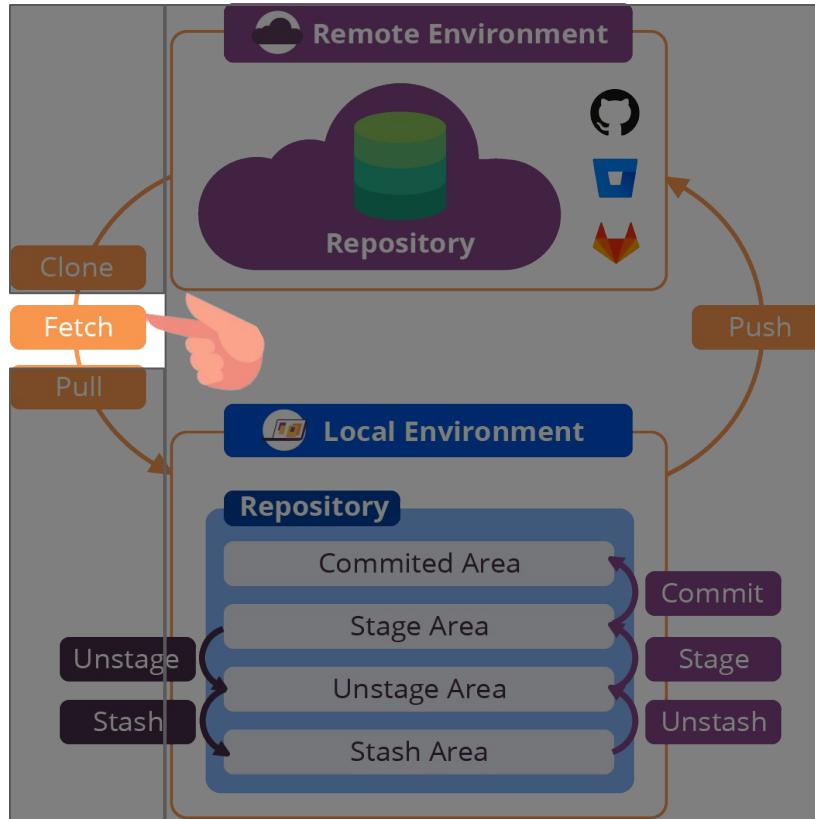
Kita bisa melakukan berbagai aksi Git terhadap *repository* yang ada pada *local environment* tanpa terhubung ke Internet.



Clone

Clone adalah perintah untuk mereplikasi *repository* yang ada di *remote environment* ke *local environment*.

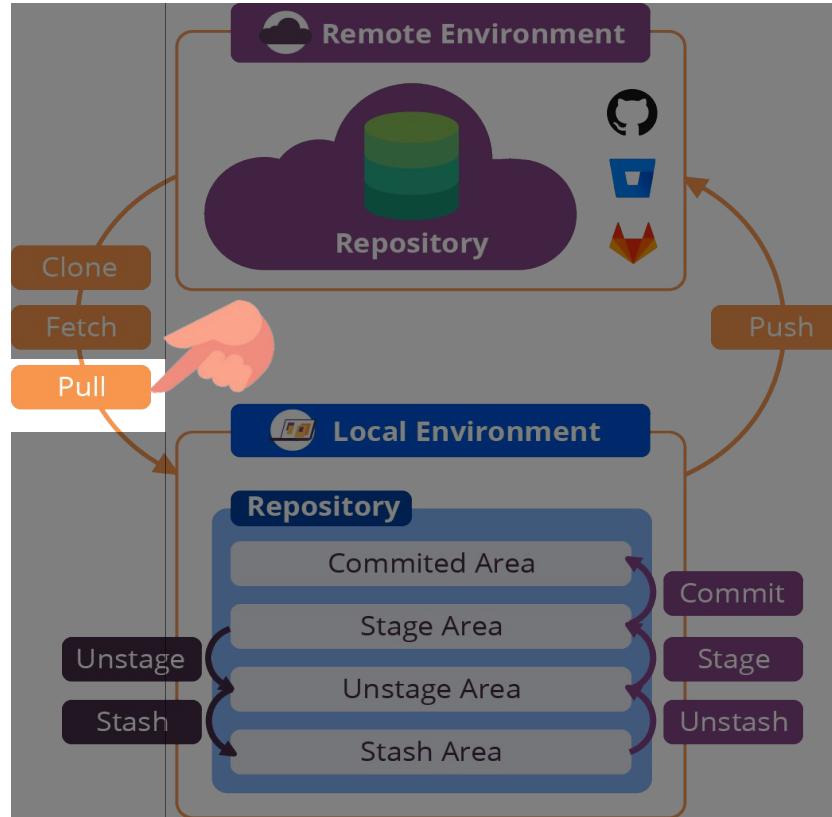
Dengan begitu, kita bisa menerapkan perintah-perintah lainnya.



Fetch

Fetch terjadi dari Remote ke Local. Perintah untuk mendapatkan detail tentang apa saja yang berubah pada remote repository.

Namun perubahan yang didapatkan tidak akan berpengaruh pada repository local kita sampai kita melakukan perintah **pull**.

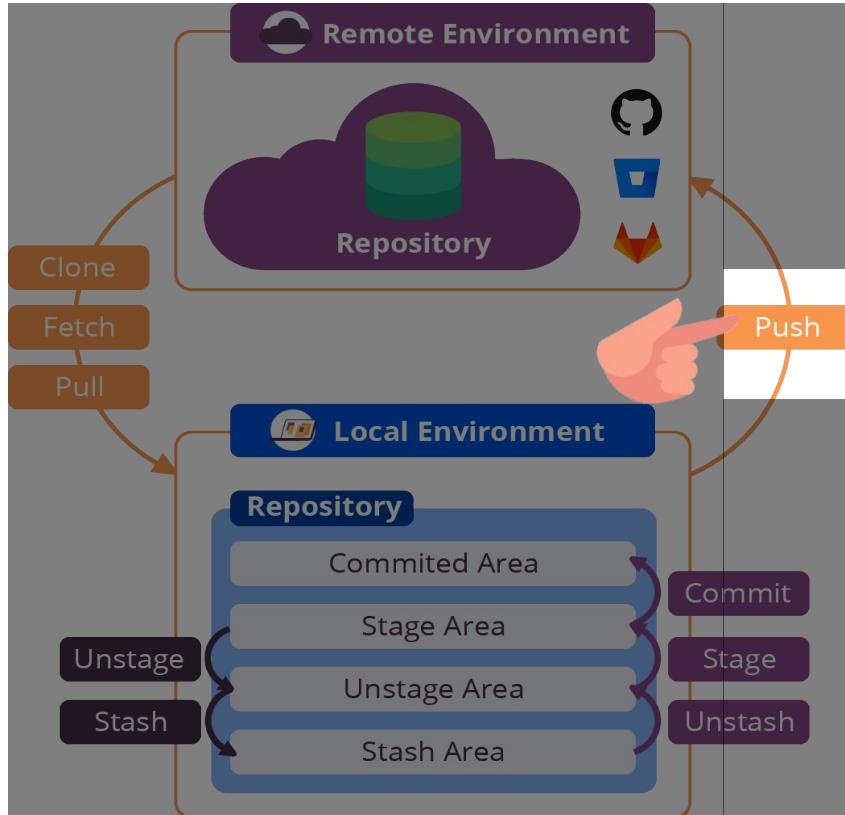


Pull

Pull terjadi dari Remote Env ke Local Env. Merupakan **perintah untuk ‘menarik’ berbagai macam perubahan yang terjadi di remote environment**, meliputi perubahan file, penghapusan file, maupun penambahan file.

Sebelum memberikan perintah pull, kondisi local repository harus bersih dari perubahan.

Setelah melakukan perintah pull, akan terjadi conflict jika perubahan yang ada dalam commit local kita berbeda dengan perubahan yang ada pada commit remote.



Push

Push terjadi dari Local ke Remote. Kebalikan dari pull, perintah **push** mendorong perubahan dari local environment ke remote environment.

Semua perubahan file, pembuatan branch, dan hal-hal lain yang kamu lakukan di level local akan dikirimkan ke remote.



Nah, tadi sering dibahas tuh operasi Git yang dilakukan dari Local Environment ke Remote Environment maupun sebaliknya.

Supaya pengetahuan kita seluas samudera, sekarang kita kerucutin operasi Git yang terjadi di Local Environment!



Coba kita review ulang dulu ...

Jadi GIT itu punya 2 environment; Remote/online environment, dan Local environment.

Remote environment ini yang bisa lihat semua team member, seperti yang sudah dijelaskan sebelumnya.

Sedangkan Local environment ini yang bisa lihat hanya kita.

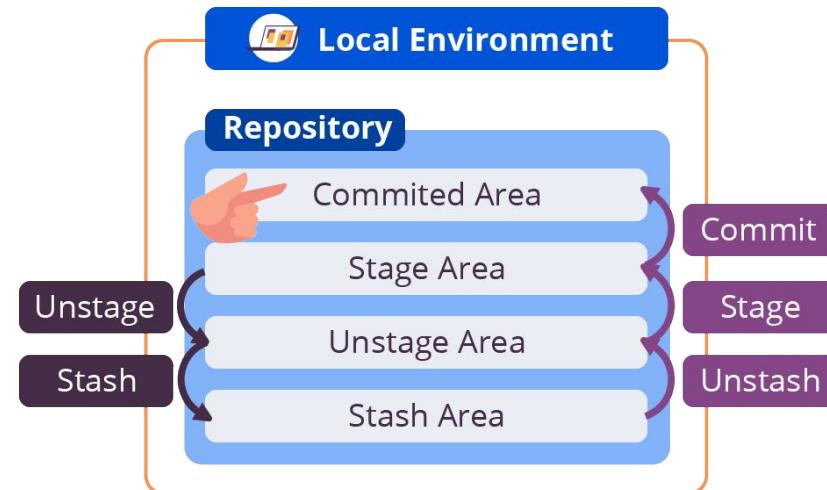


Sebelum lanjut, kita cari tahu dulu tempat-tempat yang ada di Git!

1. Committed Area

Area tempat commit dikumpulkan. Commit adalah kumpulan perubahan yang direkam, atau tempat di mana developer bisa masukin checkpoint.

Ini casenya masih local (di device kita) ya, belum bisa dilihat oleh team member lain akan tetapi akan tercatat di history pada GIT.



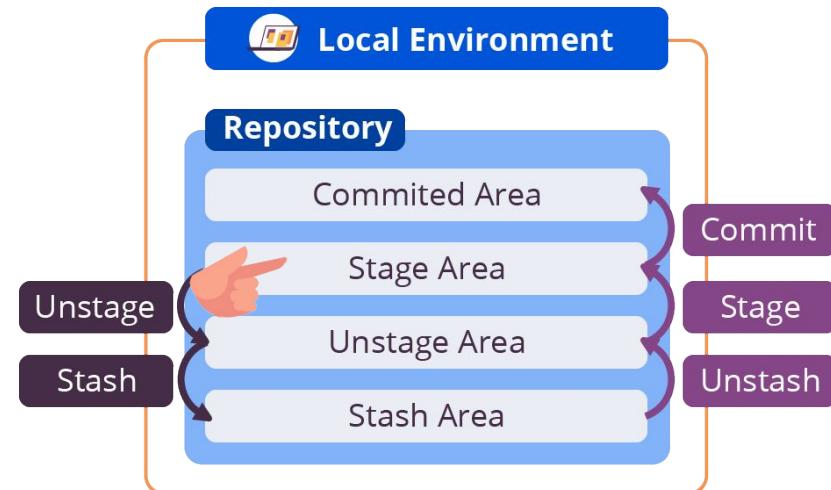


2.

Stage Area

Area tempat file-file kita diawasi perubahannya oleh Git.

Area ini merupakan tempat dimana kita mempunyai perubahan/penambahan dan kita memberitahu ke Git bahwa betul semua perubahan/penambahan itu akan dibawa masuk ke Git.



3. Working Space / Unstage Area

Tempat file yang tidak terawasi atau belum masuk ke Stage Area.

Unstage area tempat dimana kita mempunyai perubahan/penambahan tetapi kita belum/tidak memberitahu GIT bahwa itu akan dibawa masuk ke GIT.

Area ini kayak waktu kamu ngerjain Ms Word, tapi blm di save (dan belum ada autosave nya).



4.

Stash Area

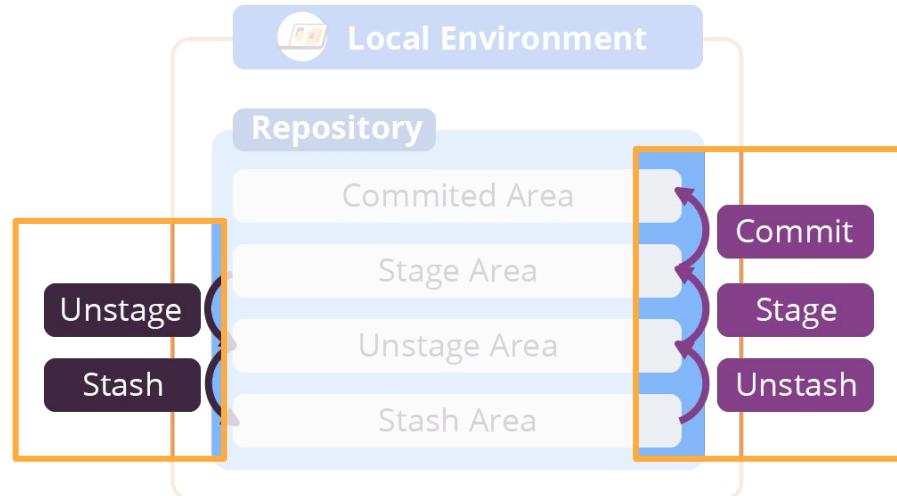
Tempat berkumpulnya stash. Stash adalah perubahan yang disimpan tanpa melakukan commit.

Stash ini sama seperti commit, hanya saja dia cuman bersifat local (di device kita). Jadi ketika distash, akan menyimpan layaknya draft.



Kalau proses pemindahan file antar area itu, gimana ya?

Setelah tahu empat definisi diatas, kita bakal bahas proses-proses pemindahan antar tempat penyimpanan~

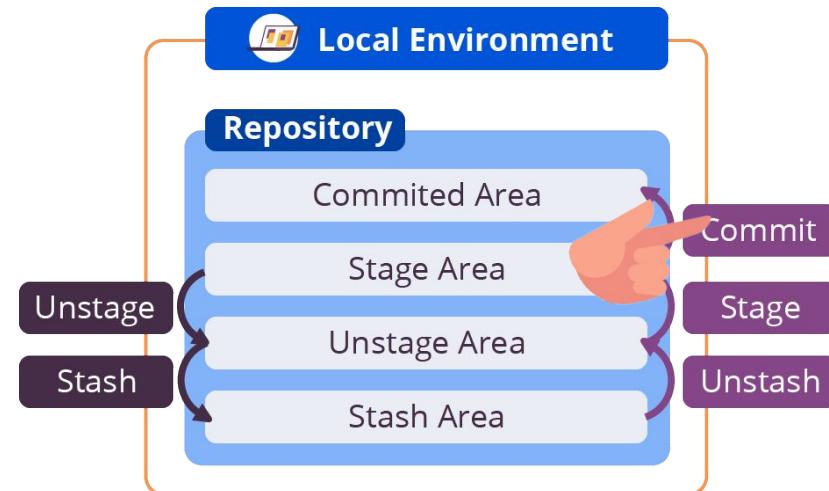


Commit

Commit adalah perintah untuk menyimpan rekaman perubahan yang terjadi pada *file* yang ada di *Stage Area*.

Untuk melakukan commit bisa menggunakan perintah

```
git commit -m "Pesanan Commit"
```



Stage

Stage adalah perintah untuk **menambahkan file dari Unstage Area atau tidak terawasi oleh Git ke Stage Area**.

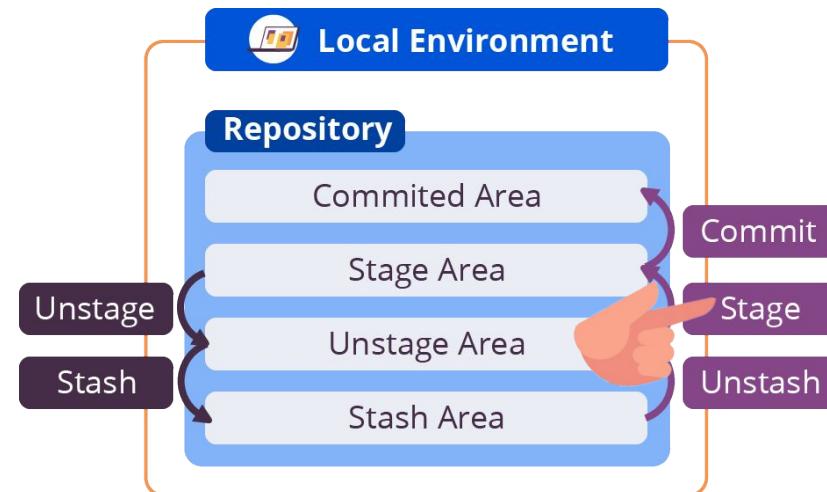
Dengan begitu, setiap perubahan yang terjadi pada file akan terdeteksi oleh git dan memungkinkan untuk dilakukan *commit*.

Cara mengetahui status perubahan pada Git Repository adalah dengan *command*:

git status

Sedangkan untuk menambahkan file ke Stage Area, bisa dengan *command*

git add



Unstage

Unstage adalah perintah untuk **menghapus file dari Stage Area dan memindahkannya ke Working Space atau Unstage Area**.

Perintah ini membuat *file* tidak lagi diawasi oleh Git. Karena itu, kita tidak bisa melakukan *commit* pada *file*.

Untuk melakukan unstage bisa menggunakan perintah

`git reset`





Stash

Perintah untuk **menyimpan rekaman perubahan pada Working Space / Unstage Area**. Berbeda dengan **commit**, perintah **stash** akan tersimpan di dalam **stash area**.

Perintah ini penting untuk diterapkan jika sedang membangun kode dengan progress yang belum selesai namun terlalu sayang untuk dihapus (*drafting*).

Untuk melakukan stash bisa menggunakan perintah

git stash



Unstash

Perintah untuk **menerapkan kembali** draft kode yang kita simpan di *Stash Area* dengan perintah *stash* sebelumnya.

Proses ini biasa disebut dengan **Apply Stash**.

Untuk melakukan stash bisa menggunakan perintah.

git stash pop

atau

git stash apply

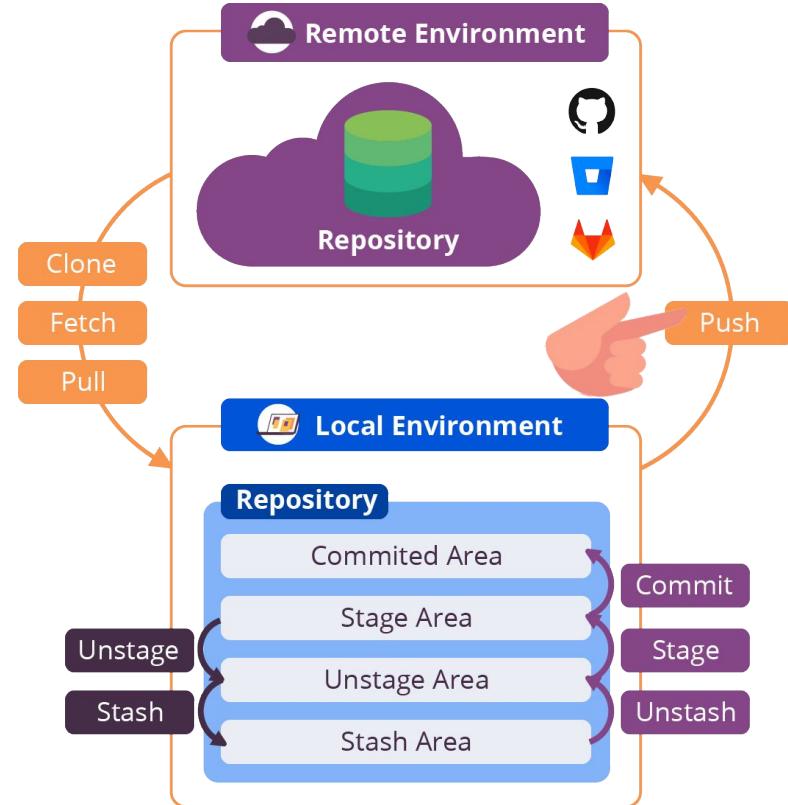


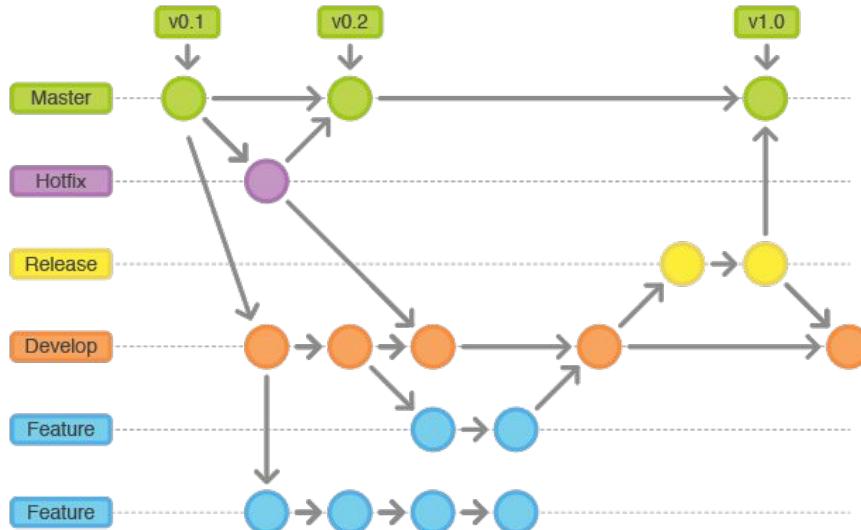
Kalau Update Perubahan dari Local ke Remote Environment?

Kamu bisa update perubahan kita di local environment kamu ke remote environment (misal ke gitlab atau github) dengan menjalankan command “git push ...”

Seperti yang dijelaskan sebelumnya yah~

git push



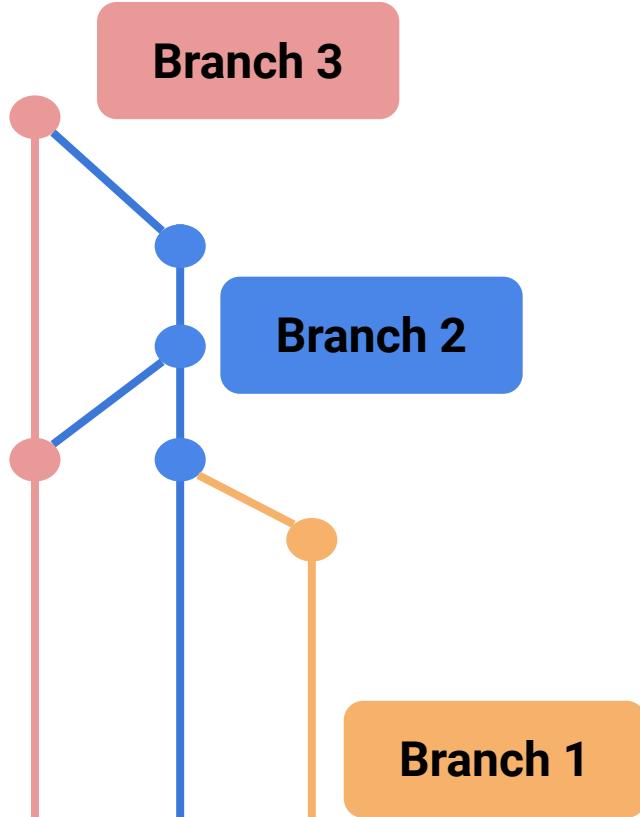


Gimana sih Flow penggunaan Git ini?

Di sebelah kiri, kamu sedang mengamati apa yang disebut dengan **Gitflow Workflow**.

Seperti *workflow* pada umumnya, ia menjadi panduan cara kerja kita dalam menggunakan Git.

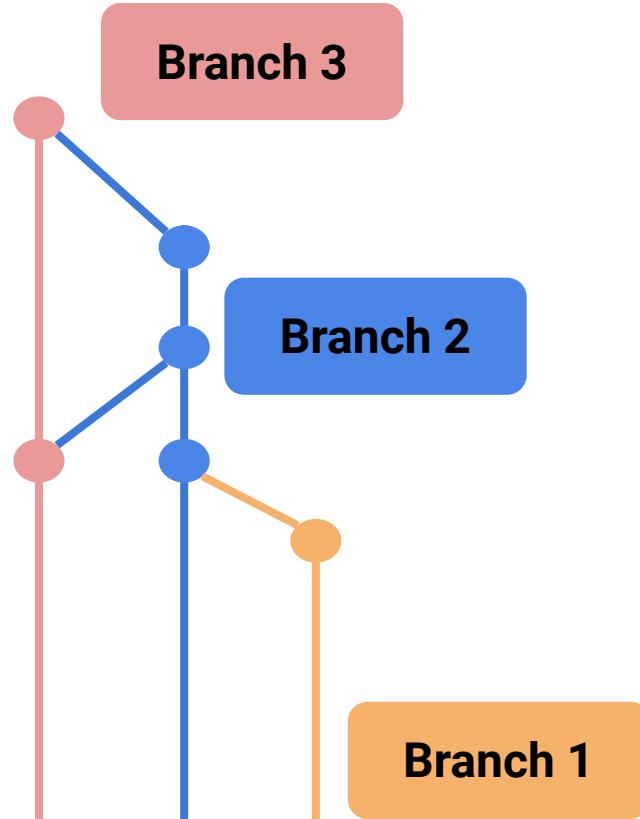
Gitflow sendiri bekerja berdasarkan *branch* yang sudah ditentukan. Namun, apa sih *branch* itu?



Apa sih maksudnya Branch?

Branch adalah komponen utama pada *version control*. Ketika kita membuat suatu *repository*, setidaknya ada satu branch yang dibuat, biasanya bernama **branch Master**.

Selanjutnya, ia akan memiliki cabang-cabang baru sesuai dengan perubahan yang kita lakukan dan catat.



Fungsi branch sangat beragam. Tapi, salah satu fungsi utamanya adalah kita bisa **beralih code dengan sangat mudah hanya dalam sekejap**. Sulap!

Sebagai contoh, pada aplikasi kita punya dua rancangan *style* dengan tampilan yang berbeda:

- warna biru, dan
- warna pink.

Dengan adanya *branch*, kita bisa berganti-ganti *style* tampilan tanpa melakukan banyak perubahan 😊



GITFLOW WORKFLOW

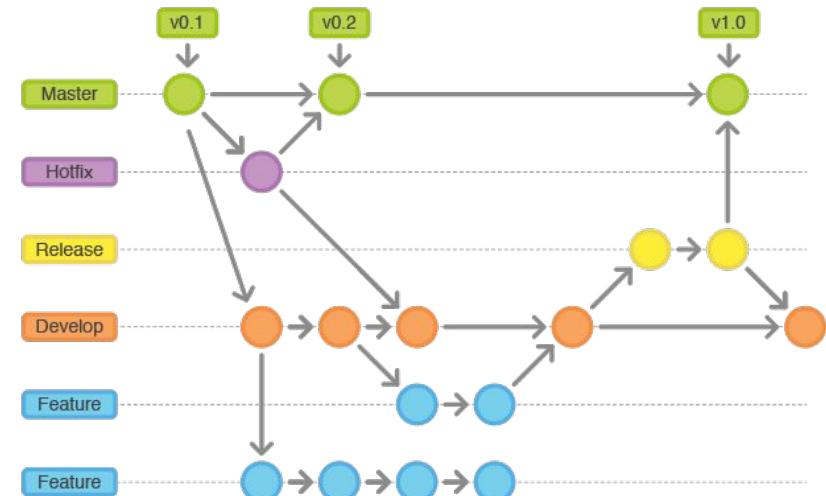
Karena tadi kita udah tahu definisi branch, sekarang kita balik lagi ke Gitflow Workflow. Kayak gambar disamping, ada banyak branch yang udah ditentuin dalam Gitflow :

BRANCH MASTER

Branch induk tempat **berbagai perubahan aplikasi terjadi**. Cabang ini bakal merekam perubahan yang terjadi, baik yang dirilis maupun tidak untuk dirilis.

BRANCH HOTFLIX

Branch ini digunakan ketika ada **bug atau kesalahan** pada versi aplikasi yang tercatat dalam *branch master*.





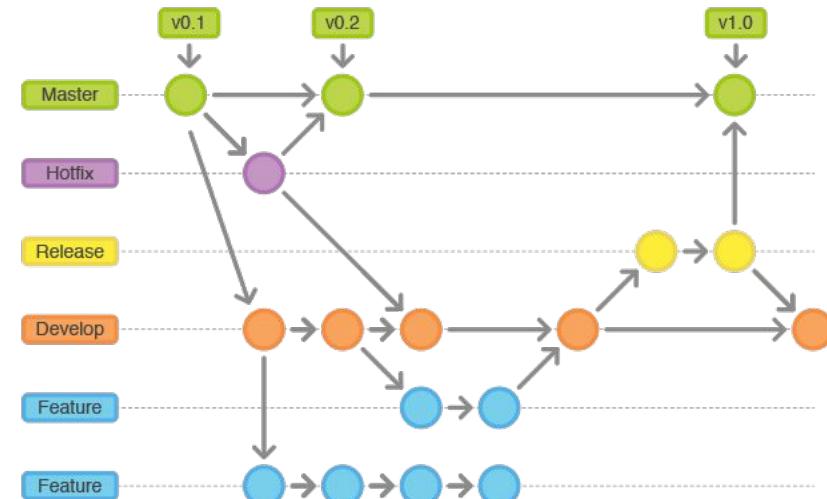
BRANCH RELEASE

Sesuai namanya, *branch* ini berfungsi jika terdapat **perbaruan ataupun penambahan fitur** pada aplikasi yang sudah dirilis.

BRANCH DEVELOP

Branch ini digunakan untuk **mencatat kode-kode yang masih dalam tahap percobaan**.

Branch develop sangat penting digunakan di tahap-tahap awal karena sangat rentan terjadi kesalahan. Dengan begitu, kode nggak tercampur dengan yang lain.



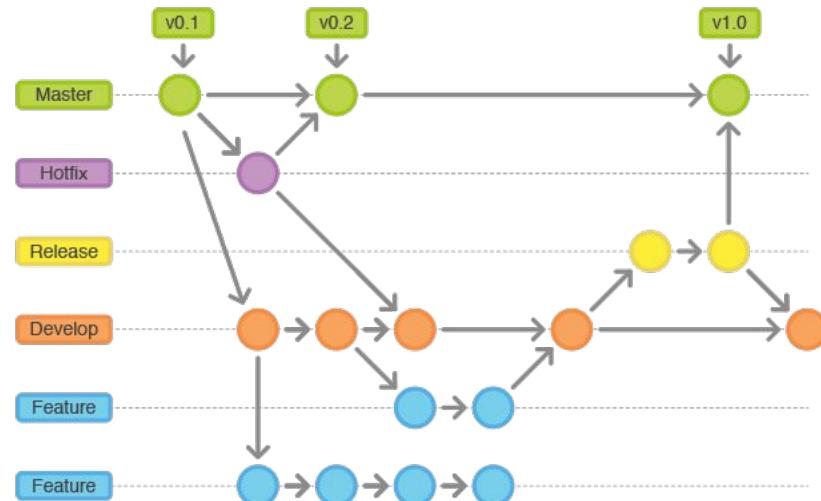


BRANCH FEATURE

Setiap fitur baru dalam aplikasi akan berada di cabang ini.

Sebagai catatan, branch feature tidak menjadikan branch master sebagai induknya, tetapi pada branch develop.

Ketika suatu fitur selesai, flow akan bergabung kembali ke track develop. Track feature tidak boleh berinteraksi langsung dengan flow master.





Wah beragam banget kan ternyata 😊

Tenang aja gengs, kalau kamu takut lupa ada banyak Cheat sheet di Internet buat bantu kamu belajar Git.

Bisa diunduh dengan [klik link ini](#) yah Cekidot 👉

Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new_branch

```
$ git branch new_branch
```

Delete the branch called my_branch

```
$ git branch -d my_branch
```

Merge branch_a into branch_b

```
$ git checkout branch_b
```

```
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$ git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

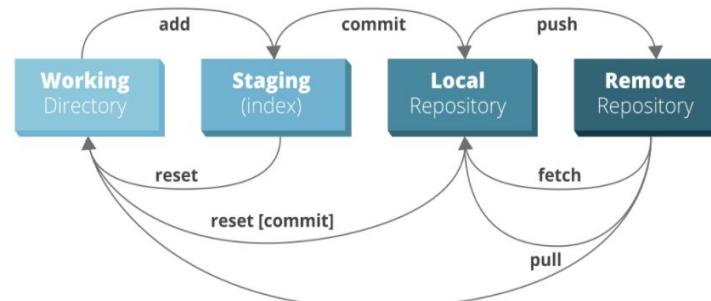
```
$ git push
```

Finally!

When in doubt, use git help

```
$ git command --help
```

Or visit <https://training.github.com/> for official GitHub training.





Wahh.. sekarang kita sudah berubah menjadi perwakilan Putra dan Putri Git

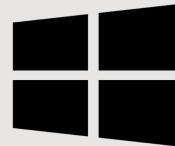


Supaya pemahaman kita terus tertanam dalam jiwa, perlu praktik langsung dengan [install Git di device kita.](#)



Windows

[Click Here](#)



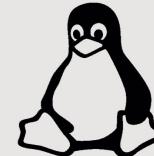
MAC OS

[Click Here](#)



Linux

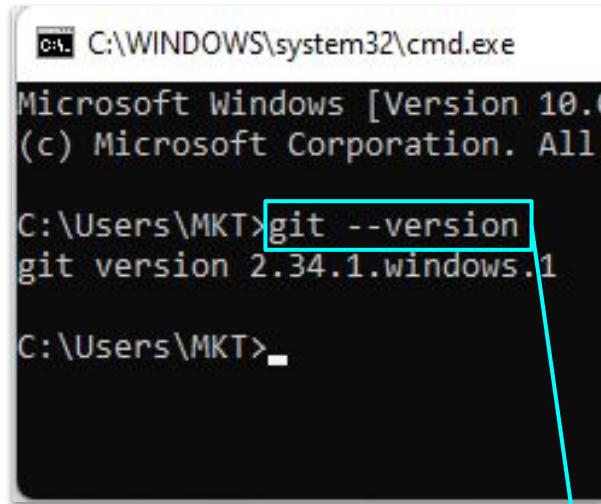
[Click Here](#)



Install GIT

Untuk install GIT pada device, kita dapat mengikuti link di samping ini untuk mulai install-nya





```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.22621.14]
(c) Microsoft Corporation. All rights reserved.

C:\Users\MKT>git --version
git version 2.34.1.windows.1

C:\Users\MKT>
```

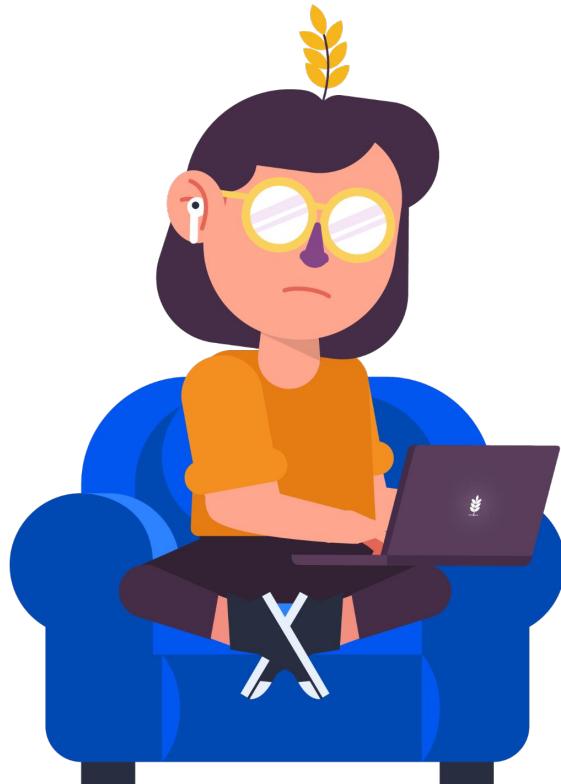
Perintah untuk
mengecek versi
GIT

Setelah kita install GIT, untuk memastikan GIT sudah terinstall dengan benar, kita perlu masukkan perintah berikut pada CMD (Windows) :

Git -- version



Install Git udah super ngoggey banget nih. Selanjutnya meluncur ke cara penggunaan Git-nya!



Udah kita install nih, terus gimana cara gunainnya?

Setelah kita berhasil install GIT pada device masing-masing, kita perlu **login** dulu ke GIT supaya bisa menggunakan seluruh fitur-nya.

ATLASSIAN



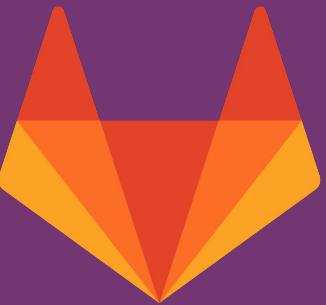
GitLab



Untuk login ke GIT, kita bisa menggunakan berbagai platform repository online kayak gini :

- Github,
- Gitlab, atau
- Bitbucket.

Kalau belum punya akun dari ketiga platform tersebut, kita bisa mendaftarkan diri dulu.



GitLab

Karena ada tiga platform yang bisa dipake, kita pilih satu yaitu **Gitlab**.

Selama proyek kedepannya, kita akan menyambungkan Project Android Studio kita dengan **Gitlab**.

Gimana sih caranya? Yuk kita simak slide selanjutnya~

Apa itu Gitlab?

Kalo sebelumnya kita mengenal dengan yang namanya GIT, kali ini kita akan berkenalan dengan apa yang dimaksud dengan Gitlab.

Gitlab merupakan **platform web-based yang menyediakan repository online gratis** secara terbuka maupun private, dan terintegrasi dengan GIT.



Lalu apa aja sih manfaat dari penggunaan Gitlab :

- Dapat berjalan di environment yang **dihosting sendiri** maupun **on-premises**.
- Source Code Management (**SCM**) yang memungkinkan developer untuk me-review, track, dan menggabungkan beberapa branch dengan mudah. (Cont.)



- Continuous Integration (**CI**) yang menyediakan pipeline otomatis untuk compile, testing, dan validasi pembuatan perangkat lunak.
- Permission yang detail dan memungkinkan kita membatasi penggabungan (merge) dan push ke pengguna tertentu.

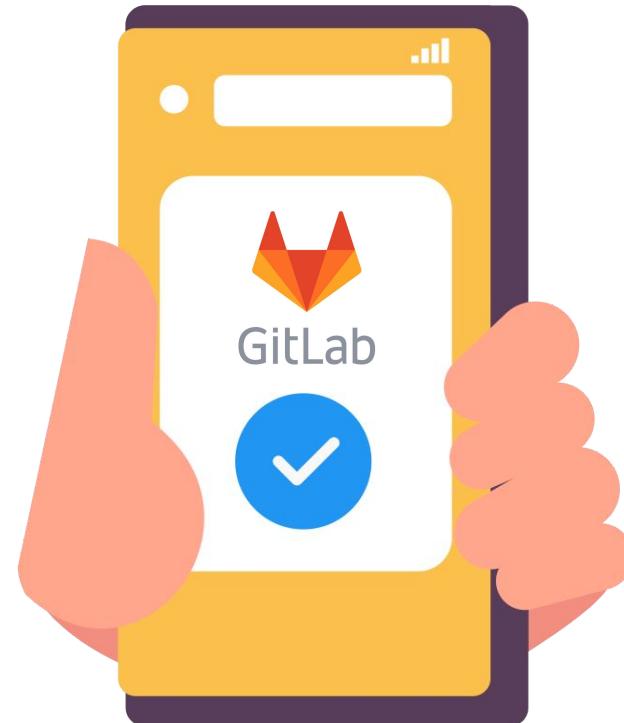
Selain itu, Gitlab memiliki paket premium yang menawarkan fitur untuk tingkat perusahaan seperti analytics, group dan project insights, laporan code quality, dan juga pelacakan compliance.



Kalau kamu pakai Gitlab versi premium, terdapat fitur-fitur tambahan khusus untuk penggunaan tingkat perusahaan, seperti :

- Analytics,
- Group dan Project insights,
- Laporan code quality, dan
- Pelacakan compliance.

Tapi untuk praktek kali ini, kita akan coba versi gratisnya dulu yah 😊





Masuk ke Gitlab

Pertama bin utama, untuk masuk ke Gitlab, kita bisa mengakses link berikut:

https://gitlab.com/users/sign_in

Kalau udah punya akun, kita bisa sign in dengan memasukkan username/email serta password.

Atau, kita dapat masuk dengan memilih satu pilihan pada section "Sign in with".

GitLab.com

GitLab.com offers free unlimited (private) repositories and unlimited collaborators.

- [Explore projects on GitLab.com](#) (no login needed)
- [More information about GitLab.com](#)
- [GitLab Community Forum](#)
- [GitLab Homepage](#)

By signing up for and by signing in to this service you accept our:

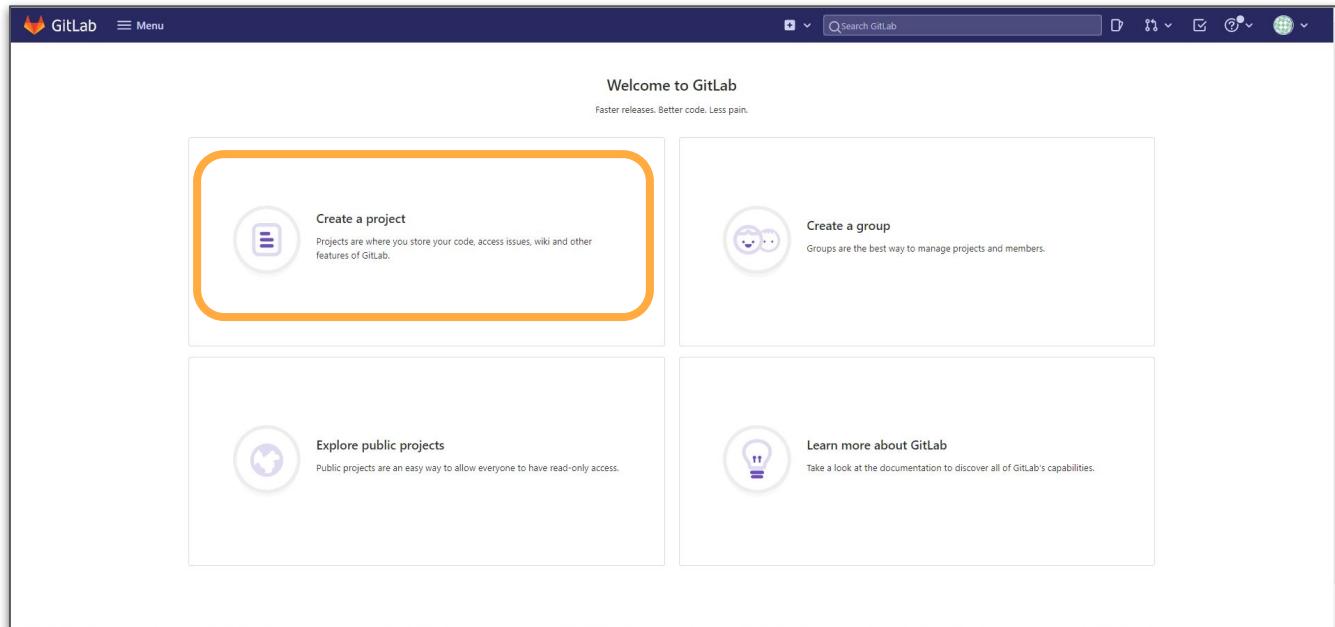
- [Privacy policy](#)
- [GitLab.com Terms](#).

The form consists of several input fields: 'Username or email' (with placeholder 'gitlab.com/johndoe'), 'Password' (with placeholder 'password123'), and a 'Remember me' checkbox. Below the form are two buttons: a blue 'Sign in' button and a smaller 'Forgot your password?' link.

Don't have an account yet? [Register now](#)

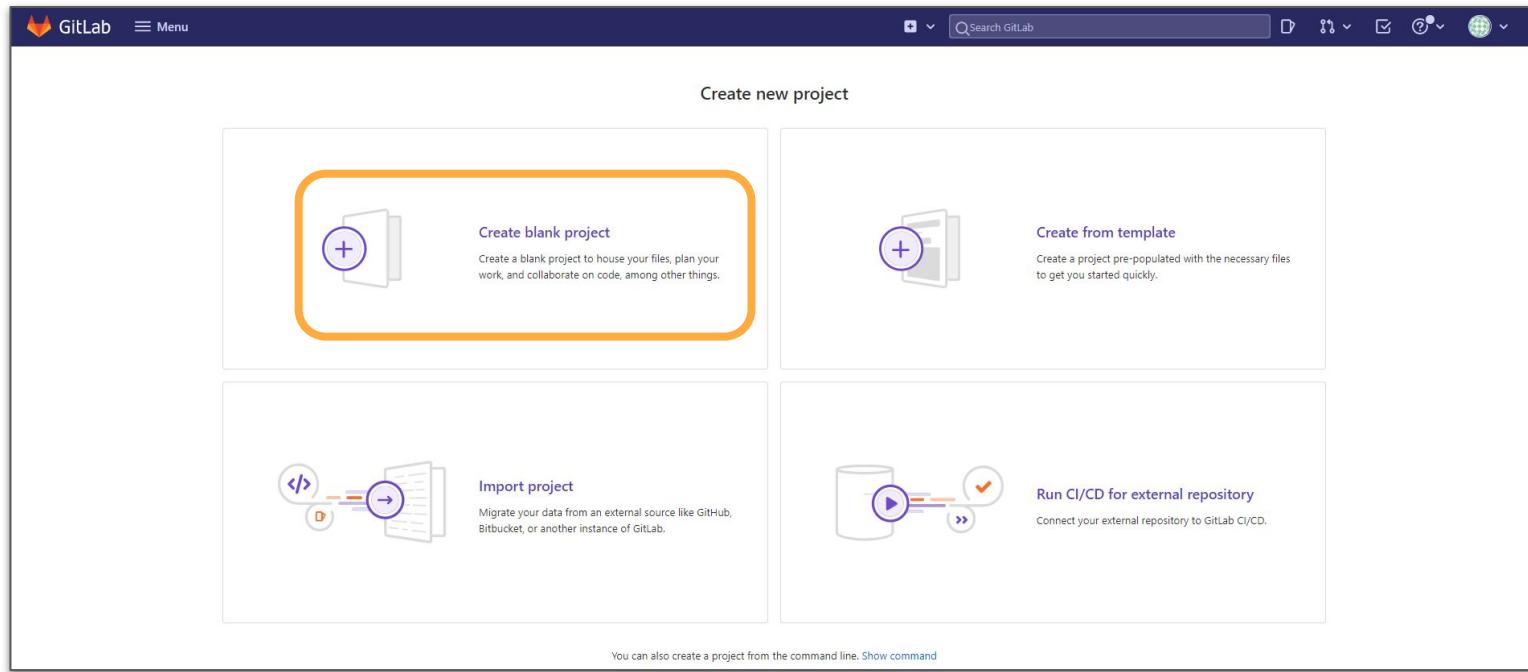
The 'Sign in with' section displays five social media and service logos: Google (G), GitHub (octocat), Twitter (bird), Bitbucket (key), and Salesforce (blue dot). Below each logo is a corresponding input field. A 'Remember me' checkbox is located at the bottom of this section.

Kalau udah berhasil masuk, maka tampilannya akan menjadi seperti berikut.



Tampilan tersebut, merupakan tampilan awal ketika kita belum pernah sama sekali membuat akun ataupun project. Nah untuk membuat sebuah project baru, kita bisa menekan menu "**Create a project**" yang terdapat pojok kiri atas.

Setelah itu, kita akan masuk ke halaman baru untuk menentukan seperti apa project yang akan kita buat. Di sini kita bisa memilih menu “**Create blank project**” yang terdapat pada kiri atas.



Bimsalabim, muncul halaman berikutnya yang mengarahkan kita untuk mengisi beberapa field sebelum membuat sebuah project baru.

New project > Create blank project

Create blank project

Project name: MyFirstProject

Project URL: https://gitlab.com/Abikayusri/

Project slug: myfirstproject

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

Project description (optional)

Description format

Project deployment target (optional)

Select the deployment target

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.

Project Configuration

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.

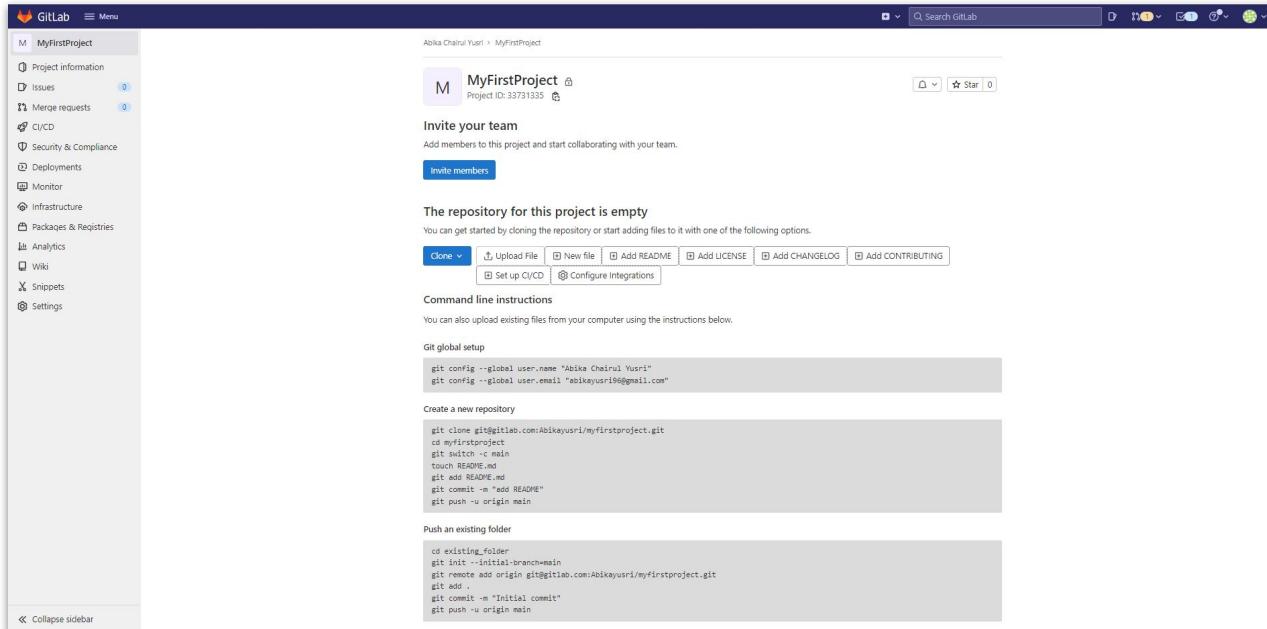
Enable Static Application Security Testing (SAST)
Analyze your source code for known security vulnerabilities. [Learn more](#).

[Create project](#) [Cancel](#)

Kita akan membuat sebuah project dengan nama "MyFirstProject" kayak gambar diatas sebagai latihan membuat project pada Gitlab. Kalau udah, tekan tombol "Create project"



Setelah berhasil, maka tampilannya akan berubah menjadi seperti di bawah ini. Lalu apakah udah selesai?



The screenshot shows a GitLab project page for 'MyFirstProject'. The sidebar on the left contains links for Project information, Issues (0), Merge requests (0), CI/CD, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, Snippets, and Settings. The main content area shows the project details for 'MyFirstProject' (Project ID: 3373135). It includes sections for 'Invite your team' (with a 'invite members' button), 'The repository for this project is empty' (with options to 'Clone', 'Upload File', 'New file', 'Add README', 'Add LICENSE', 'Add CHANGELOG', 'Add CONTRIBUTING', 'Set up CI/CD', and 'Configure Integrations'), 'Command line instructions' (with a code block for cloning the repository), 'Git global setup' (with a code block for setting global git config), 'Create a new repository' (with a code block for initializing a new repository), and 'Push an existing folder' (with a code block for pushing an existing folder to the repository).

Belum yaa. Supaya project Android Studio kita bisa masuk ke dalam repository yang udah kita buat, langkah selanjutnya adalah membuat sebuah project **dummy** yang bisa kita gunakan untuk latihan integrasi Android Studio dengan Gitlab.



Tadaaaa~ kalau udah bikin project dummy, kita balik lagi ke halaman Gitlab.

Perhatiin Command Line instructions, disitu ada cara untuk mengintegrasikan project Android Studio ke repository yang telah kita buat.

Cara tersebut, dapat kita implementasikan pada project Android Studio kita. Sekarang, kita buka kembali project Android Studio~

Git global setup

```
git config --global user.name "Abika Chairul Yusri"  
git config --global user.email "abikayusri96@gmail.com"
```

Create a new repository

```
git clone git@gitlab.com:Abikayusri/myfirstproject.git  
cd myfirstproject  
git switch -c main  
touch README.md  
git add README.md  
git commit -m "add README"  
git push -u origin main
```

Push an existing folder

```
cd existing_folder  
git init --initial-branch=main  
git remote add origin git@gitlab.com:Abikayusri/myfirstproject.git  
git add .  
git commit -m "Initial commit"  
git push -u origin main
```

Push an existing Git repository

```
cd existing_repo  
git remote rename origin old-origin  
git remote add origin git@gitlab.com:Abikayusri/myfirstproject.git  
git push -u origin --all  
git push -u origin --tags
```

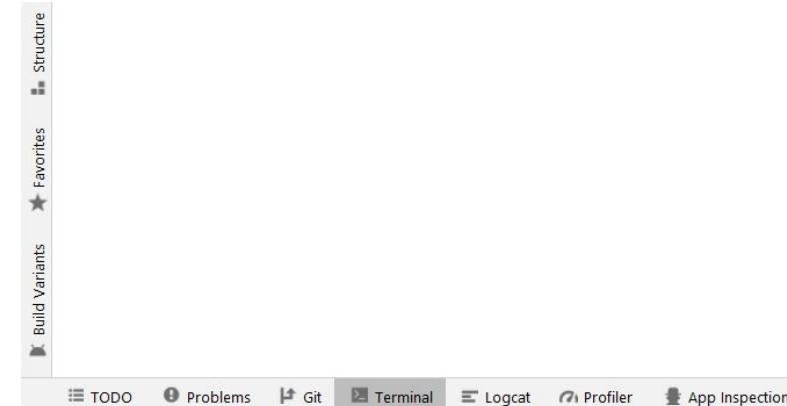


Pada Android Studio, bukalah tab **terminal** yang berada pada bawah Android Studio.

Setelah itu, kita bisa masukan perintah kayak gambar samping, ke terminal.

Jika berhasil, maka pada terminal akan mengeluarkan output kayak gambar dibawah. Kemudian, kalau kita perhatikan lagi, pada bagian kanan bawah Android Studio akan muncul sebuah branch yang telah kita buat.

```
Enumerating objects: 71, done.  
Counting objects: 100% (71/71), done.  
Delta compression using up to 8 threads  
Compressing objects: 100% (53/53), done.  
Writing objects: 100% (71/71), 97.03 KiB | 1.56 MiB/s, done.  
Total 71 (delta 1), reused 0 (delta 0), pack-reused 0  
To gitlab.com:Binarian/myfirstproject.git  
 * [new branch]      main -> main
```



```
git init --initial-branch=main  
  
git remote add origin  
git@gitlab.com:Binarian/myfirstproject.git  
  
git add .  
  
git commit -m "Initial commit"  
  
git push -u origin main
```

Pas kita cek kembali repository Gitlab kita, maka repository tersebut akan berubah menjadi gambar di bawah ini. Yang artinya kita sudah berhasil mengintegrasikan repository Gitlab dengan project Android Studio kita~

M MyFirstProject

Project ID: 33731335

1 Commit 1 Branch 0 Tags 635 KB Files 635 KB Storage

Auto DevOps
It will automatically build, test, and deploy your application based on a predefined CI/CD configuration.
Learn more in the Auto DevOps documentation

Enable in settings

master myfirstproject /

History Find file Web IDE Clone

first commit Abika Chairul Yusri authored 5 minutes ago d4951d59

Upload File Add README Add LICENSE Add CHANGELOG Add CONTRIBUTING Add Kubernetes cluster

Set up CI/CD Configure Integrations

Name	Last commit	Last update
.idea	first commit	5 minutes ago
app	first commit	5 minutes ago
gradle/wrapper	first commit	5 minutes ago
.gitignore	first commit	5 minutes ago
build.gradle	first commit	5 minutes ago
gradle.properties	first commit	5 minutes ago
gradlew	first commit	5 minutes ago
gradlew.bat	first commit	5 minutes ago
settings.gradle	first commit	5 minutes ago

<< Collapse sidebar



Wohoooo~

**Kita sudah berhasil mengintegrasikan
Android Studio dengan Gitlab**

**Integrasi proyek kita ke Gitlab ini akan
banyak membantu dalam penggerjaan
challenge kedepannya~**

Saatnya kita Quiz!





1. Di bawah ini yang merupakan manfaat paling tepat dari penggunaan version control system, adalah ...

- A. Membantu melacak semua perubahan kode yang dilakukan oleh para engineer dan membantu mencegah terjadinya konflik
- B. Melakukan kontrol versi version pada system app yang sedang dikembangkan
- C. Membantu mengecek version pada system



1. Di bawah ini yang merupakan manfaat paling tepat dari penggunaan version control system, adalah ...

- A. Membantu melacak semua perubahan kode yang dilakukan oleh para engineer dan membantu mencegah terjadinya konflik
- B. Melakukan kontrol versi version pada system app yang sedang dikembangkan
- C. Membantu mengecek version pada system

Benar! Version Control memungkinkan kita untuk melacak semua perubahan kode yang dilakukan oleh para engineer dan juga membantu mencegah terjadinya konflik.



2. Kamu adalah seorang android developer yang bekerja secara freelance mengerjakan sebuah project besar, dan bekerja bersama dengan tim yang jumlahnya lebih dari ratusan orang baik dari dalam dan luar negeri.

Dari kasus di atas, kira-kira version control system dengan tipe apakah yang cocok digunakan?

- A. Local Version Control System
- B. Central Version Control System
- C. Distributed Version Control System

2. Kamu adalah seorang android developer yang bekerja secara freelance mengerjakan sebuah project besar, dan bekerja bersama dengan tim yang jumlahnya lebih dari ratusan orang baik dari dalam dan luar negeri.

Dari kasus di atas, kira-kira version control system dengan tipe apakah yang cocok digunakan?

- A. Local Version Control System
- B. Central Version Control System
- C. Distributed Version Control System

Pada Distributed Version Control System terdapat penyedia layanan Version Control yang ada di Cloud / Internet. Sehingga kita bisa bekerja sama dan mengakses code kita dari mana saja.



3. Berikut ini merupakan penjelasan tentang branch yang benar, kecuali...

- A. merupakan sebuah fitur yang disediakan oleh VCS dan digunakan untuk memisahkan pekerjaan satu dengan yang lainnya.
- B. penamaan branch secara default akan dinamakan sebagai branch.
- C. sebuah percabangan yang digunakan untuk pengembangan sebuah aplikasi.



3. Berikut ini merupakan penjelasan tentang branch yang benar, kecuali...

- A. merupakan sebuah fitur yang disediakan oleh VCS dan digunakan untuk memisahkan pekerjaan satu dengan yang lainnya.
- B. penamaan branch secara default akan dinamakan sebagai branch.
- C. sebuah percabangan yang digunakan untuk pengembangan sebuah aplikasi.

Penamaan branch secara default akan dinamakan dengan master atau main. Untuk membuat bahasanya lebih sesuai, komunitas Git saat ini menggunakan nama main daripada master.



4. Berikut ini, manakah perintah dalam GIT yang digunakan untuk membuat branch baru

- A. git branch <branch_name>
- B. git branch -m <branch_name>
- C. git checkout <branch_name>



4. Berikut ini, manakah perintah dalam GIT yang digunakan untuk membuat branch baru?

- A. git branch <branch_name>
- B. git branch -m <branch_name>
- C. git checkout <branch_name>

Dalam GIT, ketika kita ingin membuat branch baru. Kita dapat menggunakan perintah `git branch <branch_name>`



5. Di antara syntax di bawah ini, manakah yang digunakan untuk mengirimkan hasil perubahan kita ke dalam GIT?

- A. git push origin master
- B. git commit -m "kirim hasil"
- C. git clone origin master



5. Di antara syntax di bawah ini, manakah yang digunakan untuk mengirimkan hasil perubahan kita ke dalam GIT?

- A. git push origin master
- B. git commit -m "kirim hasil"
- C. git clone origin master

Tepat!!! Untuk mengirimkan hasil perubahan yang telah kita lakukan ke repository online dengan GIT. Kita dapat menggunakan syntax “git push origin nama_branch” untuk mengirimkannya

Referensi dan bacaan lebih lanjut~

1. [S.O.L.I.D Principles: The Kotlin Way](#)
2. [The S.O.L.I.D Principles in Pictures](#)
3. [GIT Cheat Sheet](#)



Nah, selesai sudah pembahasan kita di **Chapter 2 Topic 5** ini.

Daan, Topik ini juga mengakhiri bahasan **Chapter 2** kita.

Selanjutnya, kita akan bahas Challenge yang akan kita kerjakan sebagai penutup Chapter ini~



Terima Kasih!



Chapter ✓

completed