

Google Developer Group  
Universitas Sriwijaya

Learning 1

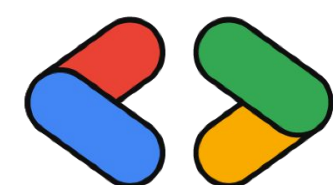
# Git & Github Fundamentals

*Version Control & Collaboration Workflow*



```
filterByOrg = filterByOrg ? study.lead_organization === filterByOrg : true  
filterByStatus = filterByStatus ? study.status === filterByStatus : true  
filterByStatus = filterByStatus ? study.status === filterByStatus : true  
filterByStatus = filterByStatus ? study.status === filterByStatus : true  
filterByStatus = filterByStatus ? study.status === filterByStatus : true
```

```
function filterStudies({ studies, filterByOrg, filterByStatus }) {  
  return studies.filter(study => {  
    return filterByOrg === study.lead_organization &&  
    filterByStatus === study.status
```



Google Developer Group  
Universitas Sriwijaya

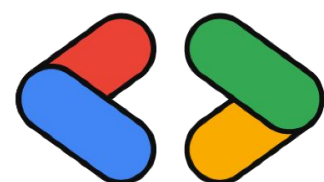
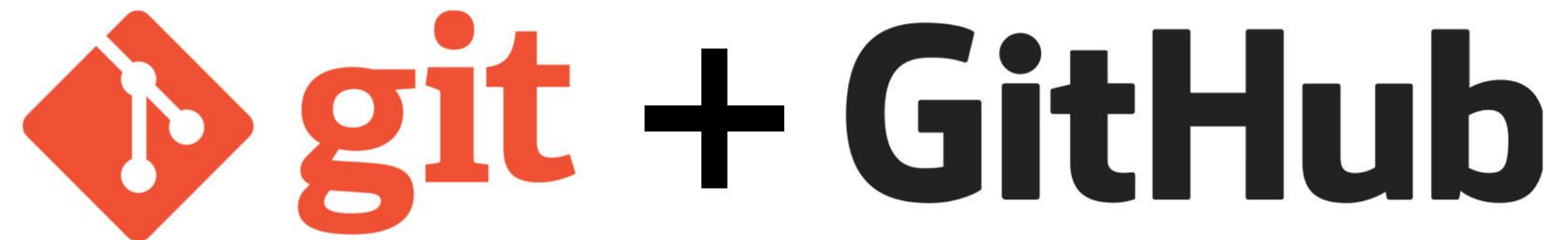
# Git vs Github

## **Git** - *Version Control System*

- Mengelola perubahan kode
- Berjalan di komputer lokal

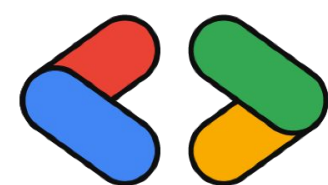
## **Github** - *Platform hosting repository Git*

- Menyimpan repository secara online
- Mendukung kolaborasi tim

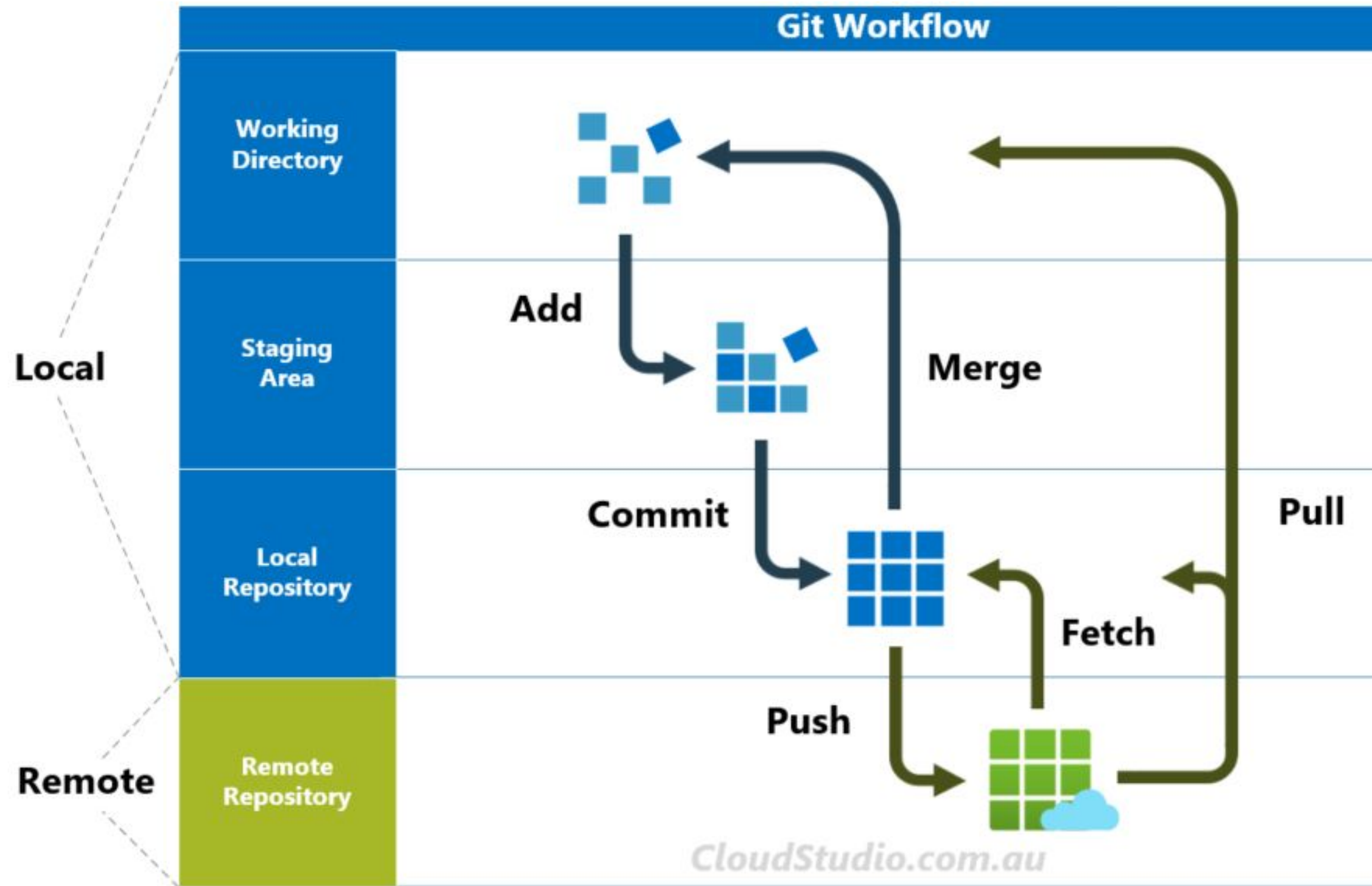


# Why Git & GitHub ?

- Melacak perubahan kode
- Menghindari kehilangan progress
- Memudahkan kolaborasi tim
- Standar Industri Software (*Bare Minimum*)

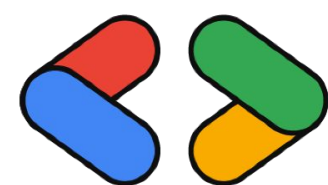
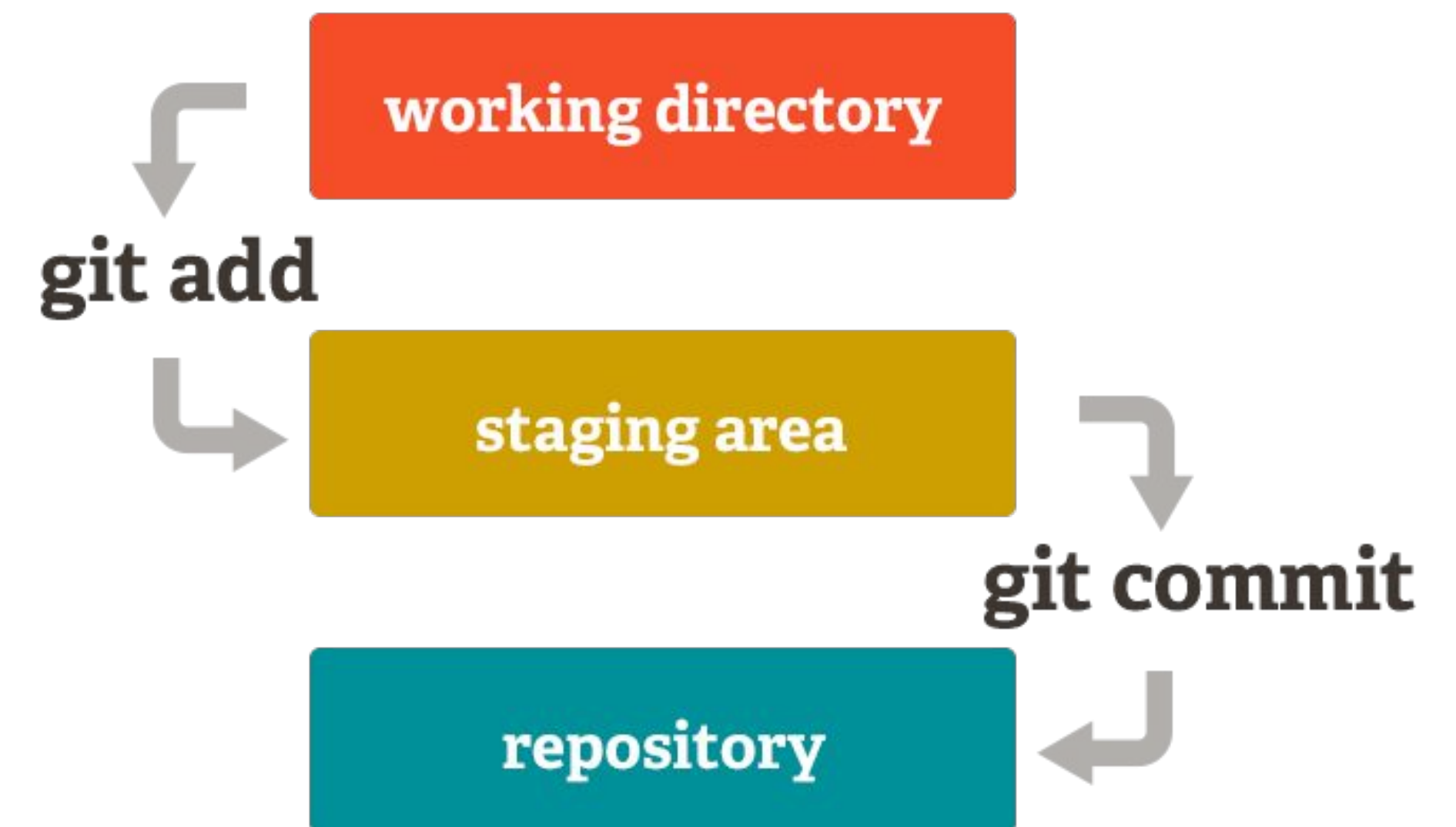


# Git Workflow



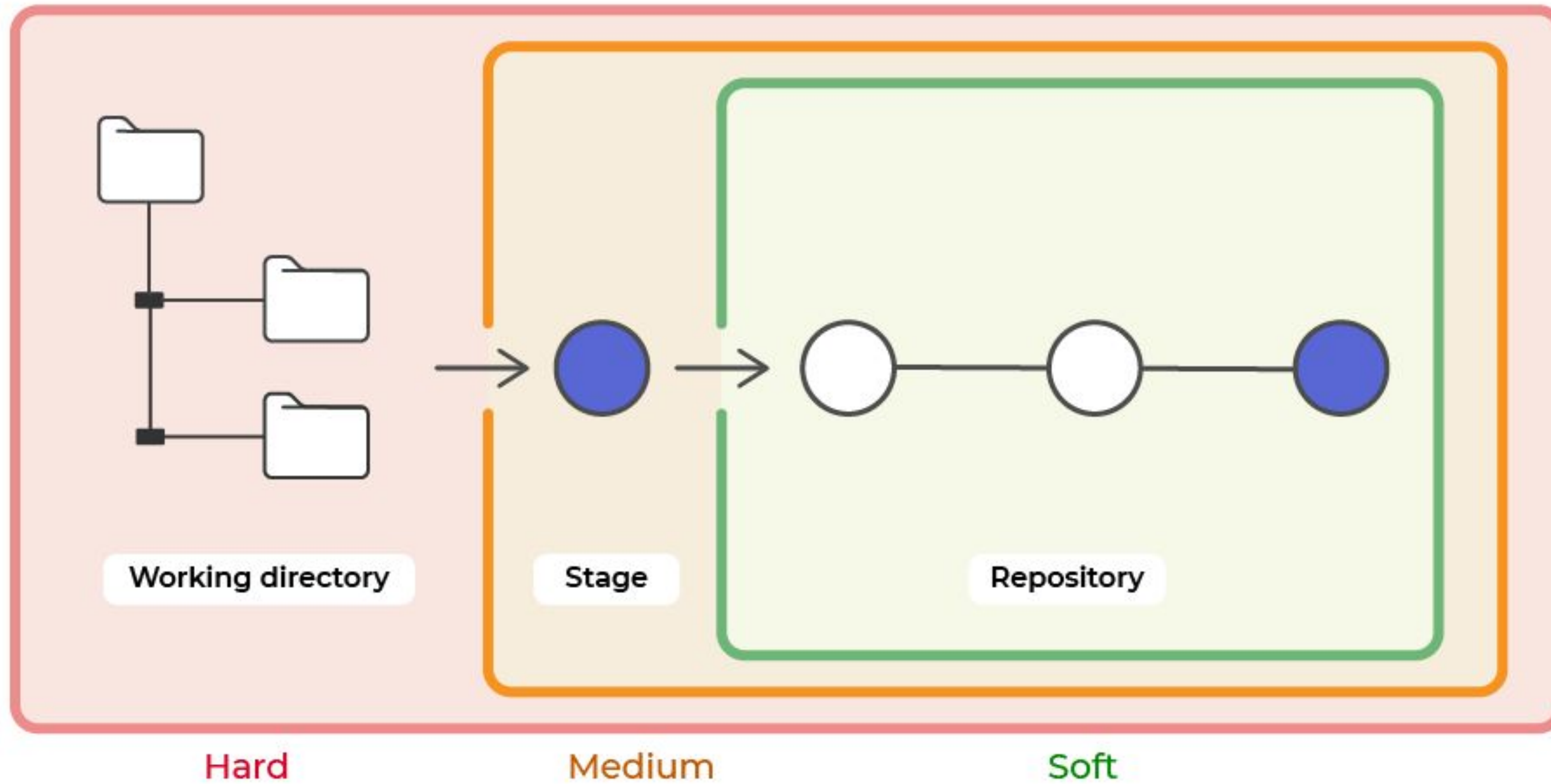
# Git Basic Commands

- **git init:** membuat repository Git
- **git status:** cek kondisi repository
- **git add:** memasukkan perubahan ke staging
- **git commit:** menyimpan perubahan
- **git log:** melihat riwayat commit



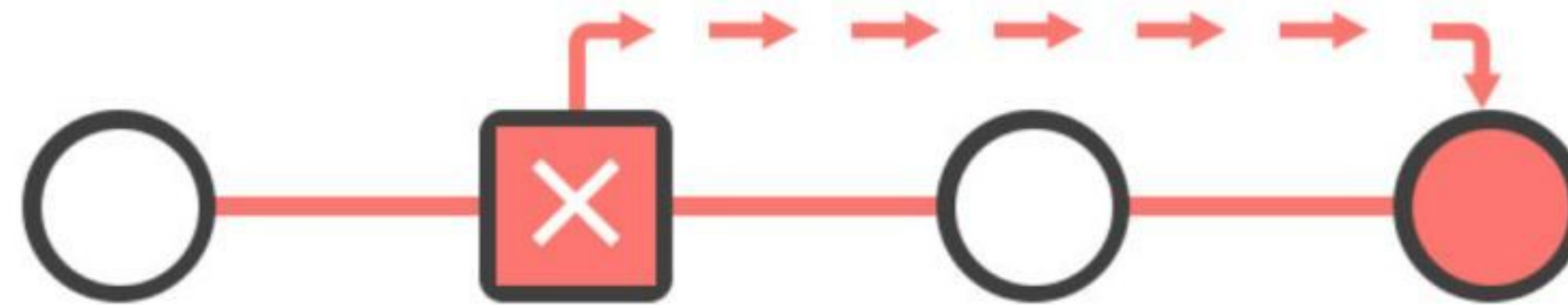
# Git Reset

- Mundur ke commit tertentu



# Git Reset vs Git Revert

Reverting



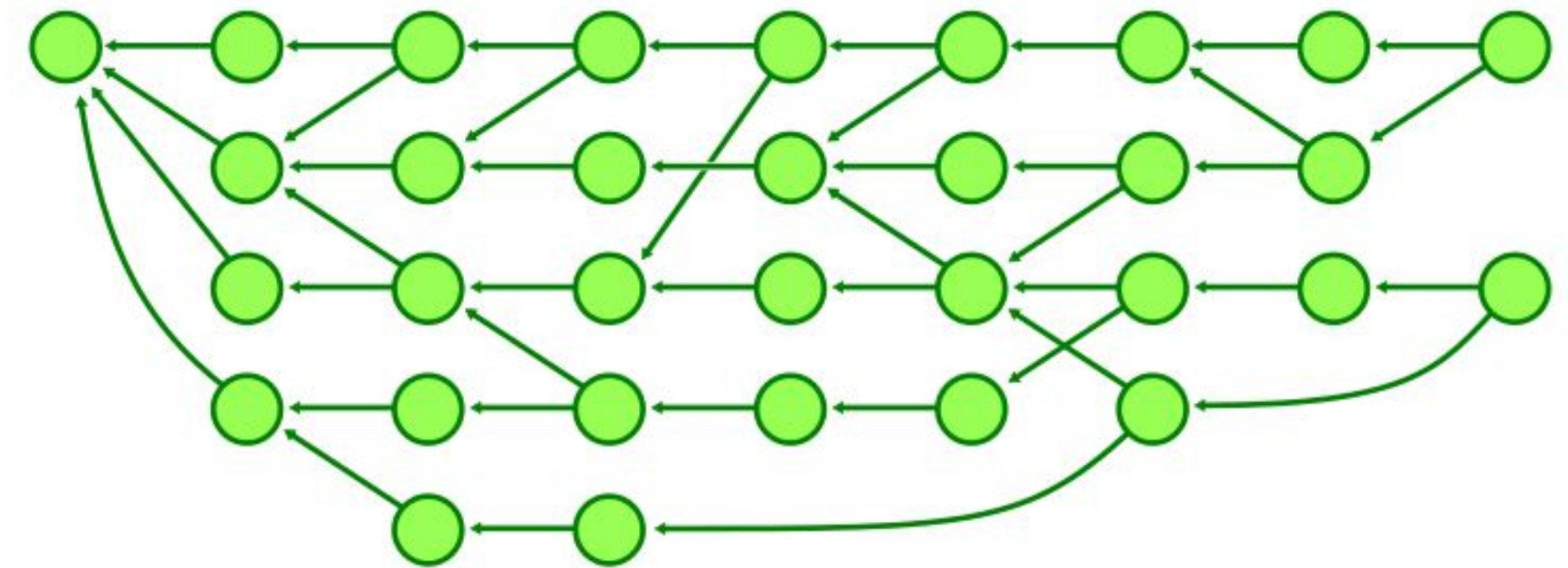
Resetting



Git Reset & Git Revert Explanation

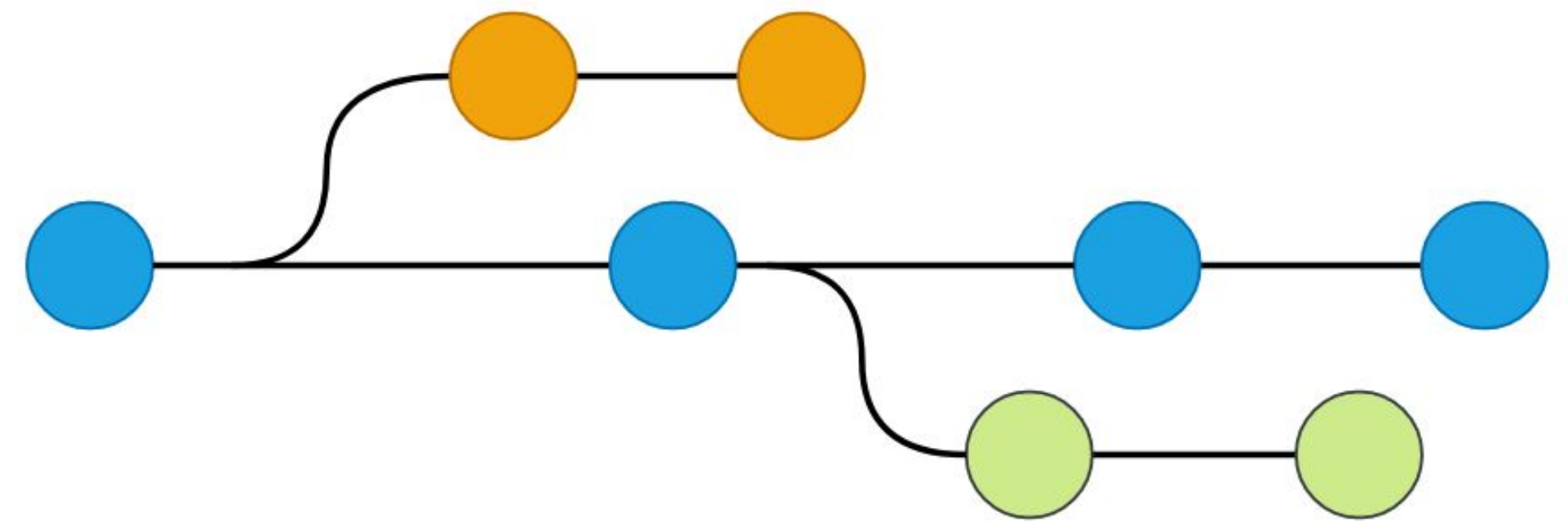
# Git Branching

- Cabang dari kode utama
- Bekerja tanpa mengganggu branch lain
- Dasar dalam kolaborasi tim



# Why Branching Matters ?

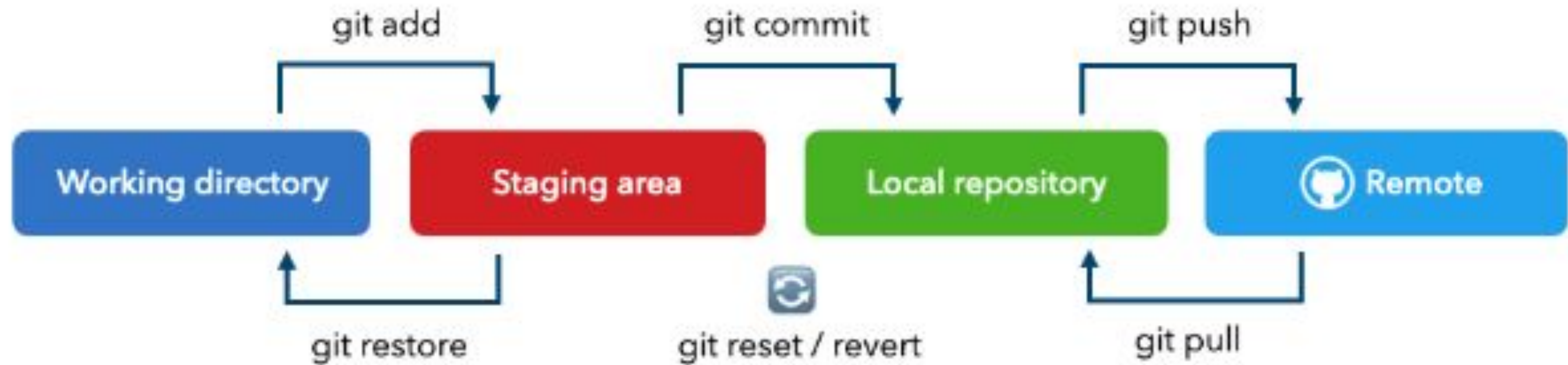
- Development lebih aman
- Menghindari konflik langsung
- Dapat bekerja secara paralel



# Git Branch Commands

- **git branch:** lihat daftar branch
- **git branch nama-branch:** membuat branch
- **git checkout nama-branch:** pindah branch
- **git merge nama-branch:** menggabungkan branch

# Github Personal Workflow




# Github Commands

- **git remote add origin:** mendaftarkan dan memberi nama alamat GitHub
- **git push:** kirim perubahan ke GitHub
- **git fetch:** cek dan ambil update dari GitHub tanpa langsung menggabungkannya.
- **git pull:** ambil update dari GitHub
- **git clone:** ambil repo dari github ke local

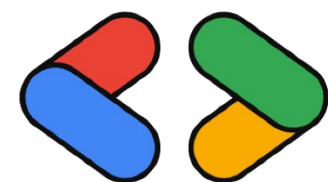
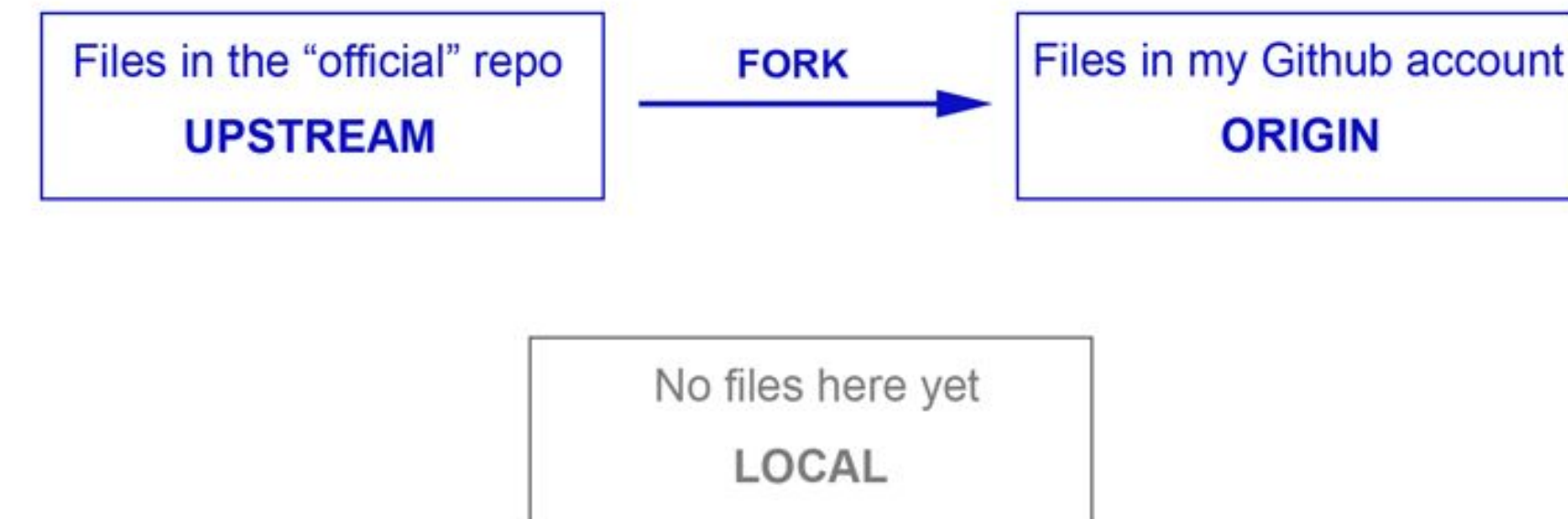
# Git Fork vs Clone

## Comparison Chart

Fork	Clone
A fork of a repository is nothing but a copy of that repository that you can work on.	A clone is basically a local copy of a remote repository that is stored on your computer.
It allows you to contribute code to the repositories where you aren't the owner or a collaborator.	It allows you to work on the projects, fix some issues or contribute changes to the code.
You do not need the owner's permission to fork their repository.	You can push the changes back to the remote repo only if you have the push rights to the repo.
	

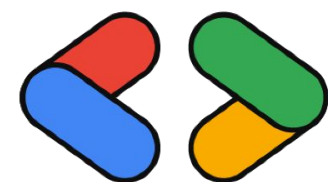
# When Do We Use Fork?

- Open source project
- Repository bukan milik kita
- Tidak punya akses *push*

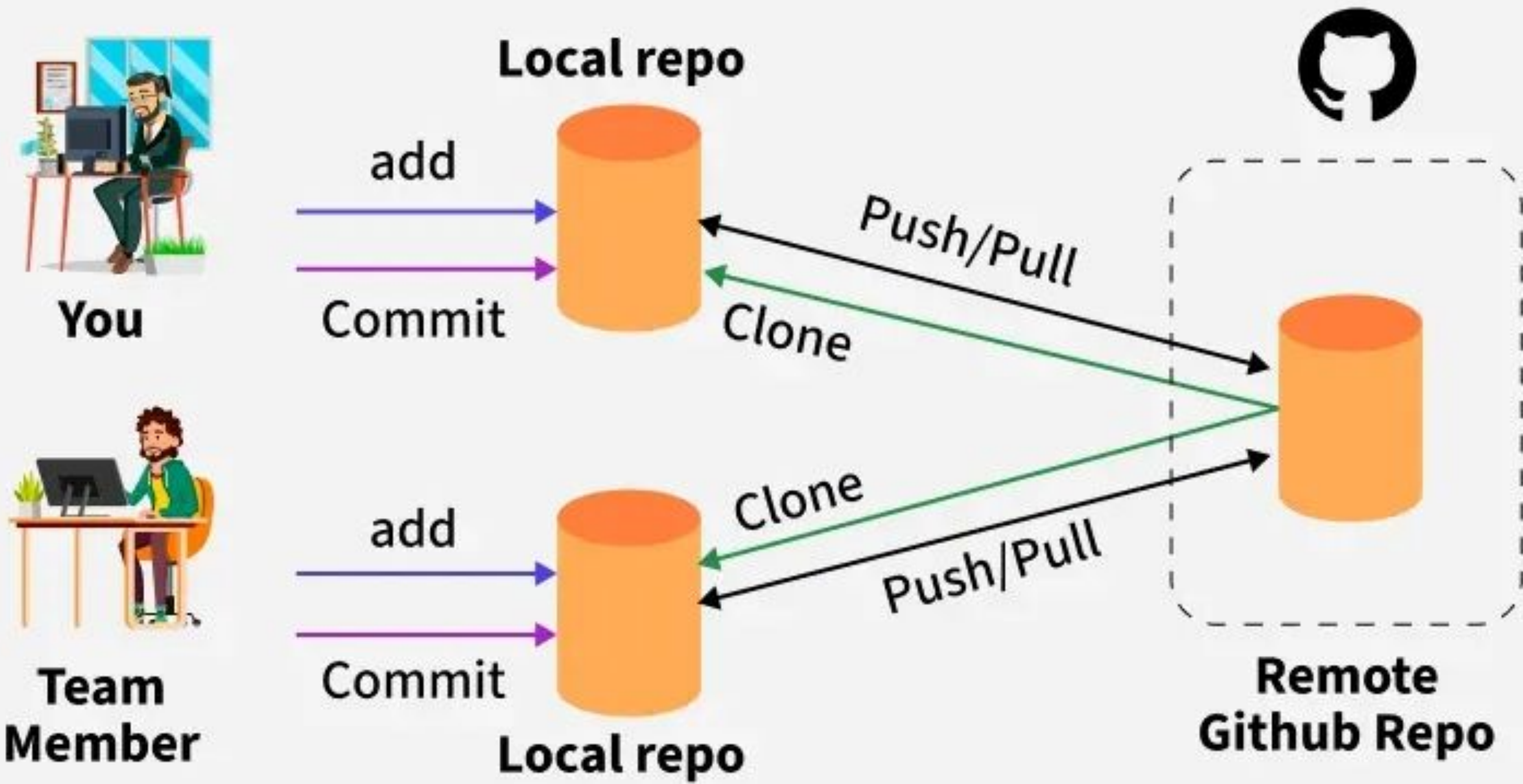


# Forking Workflow

- Forking repository
- Clone repository fork ke ke local
- Set remote *upstream* ke repository Asli
- Lakukan perubahan
- Push ke repository origin/fork
- Buat pull request di github



# Github Collaboration Workflow



# Github Collaboration

## Workflow

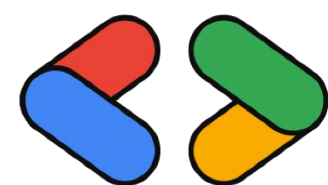
Clone → Branch → Commit → Push → Pull Request → Review → Merge

# Conventional Commit Message

Standar penulisan (*Best Practice*) untuk commit message agar perubahan kode mudah dipahami oleh tim.

*Why ?*

- Riwayat commit lebih jelas
- Mudah review & tracking
- Standar kerja profesional

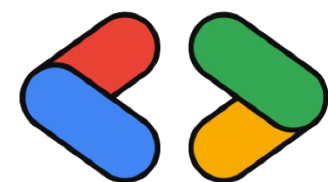


Google Developer Group  
Universitas Sriwijaya

# Commit Message Structure

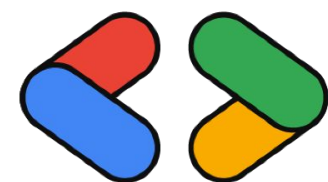
type(optional scope): short description

- chore: initial commit
- feat(auth): add login form
- refactor: optimized fetch logic
- fix(navbar): fix responsive issue
- style: change button to secondary color
- docs: add README.md



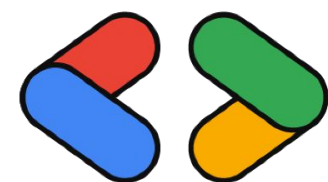
# Conventional Commit Types

- **feat:** menambahkan fitur baru
- **fix:** memperbaiki bug atau error
- **refactor:** merapikan struktur kode tanpa mengubah fitur
- **chore:** perubahan teknis non-fitur (config, dependency, tooling)
- **style:** perubahan tampilan atau formatting kode tanpa mengubah logika
- **docs:** perubahan atau penambahan dokumentasi



# Branch Naming Best Practices

- **feature/** → fitur baru → *feature/dashboard-page*
- **fix/** → perbaikan bug → *fix/navbar-responsive*
- **refactor/** → perapihan kode → *refactor/auth-logic*



# Challenge! (FE Member Only\*)

1. Buka repository Frontend Development GDGoC UNSRI 25/26
2. Lakukan forking repository
3. Lakukan forking workflow (*clone* → *set remote origin* → *set remote upstream*)
4. Buka folder *1-version-control- system*, cari file *members.js*
5. Buat branch member/nama-member
6. isi data-mu sebagai member ke dalam *Array Members*
7. lakukan push dan pull request ke repository *upstream*
8. pastikan mengimplementasikan *conventional commit messages*
9. pastikan tidak terdapat conflict saat pull request

*Selamat Mencoba ~*



# Thank You !