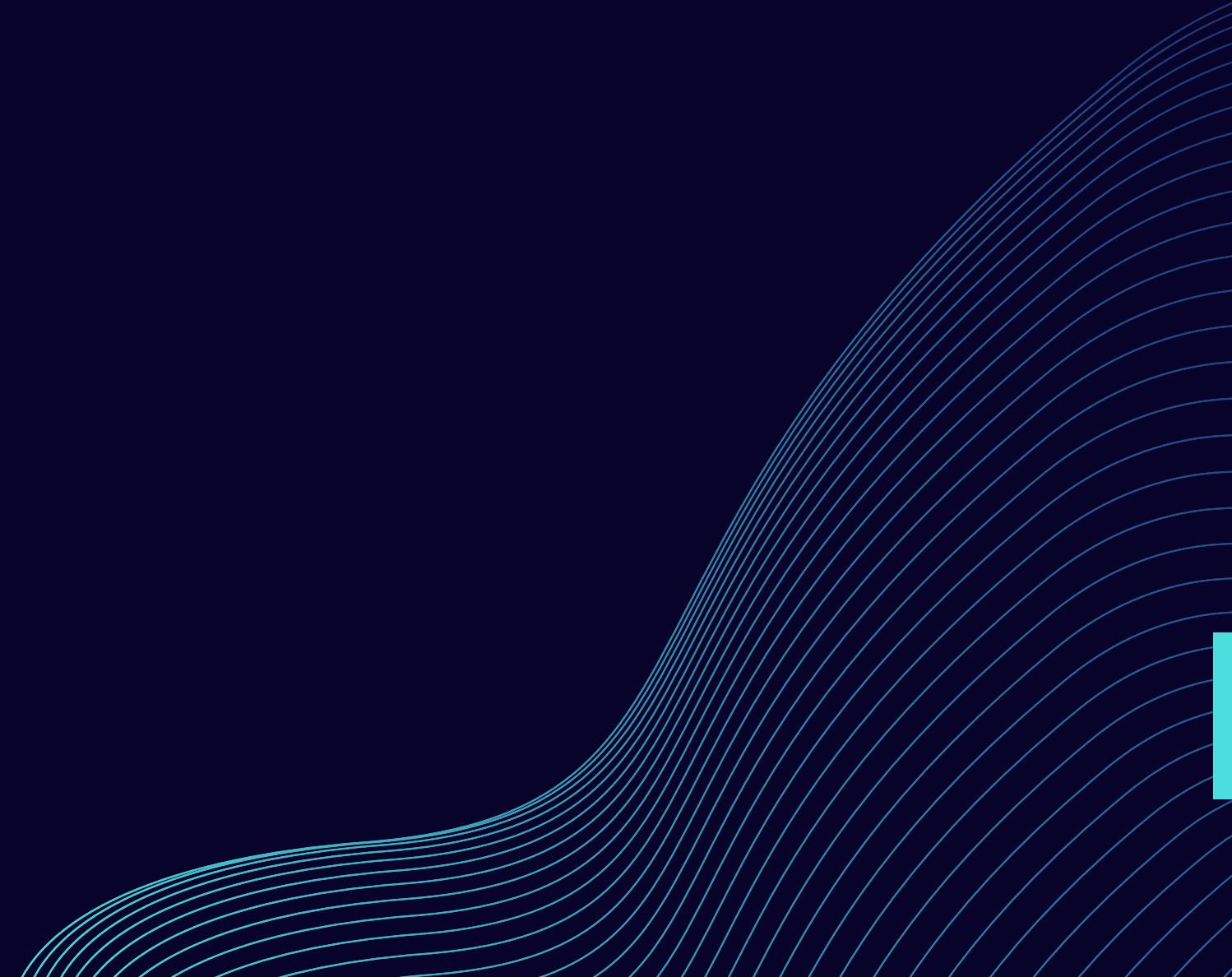




GIT PRESENTATION





PORQUÊ GIT?

01

Não mandar zips por Discord - é a melhor maneira de partilhar código

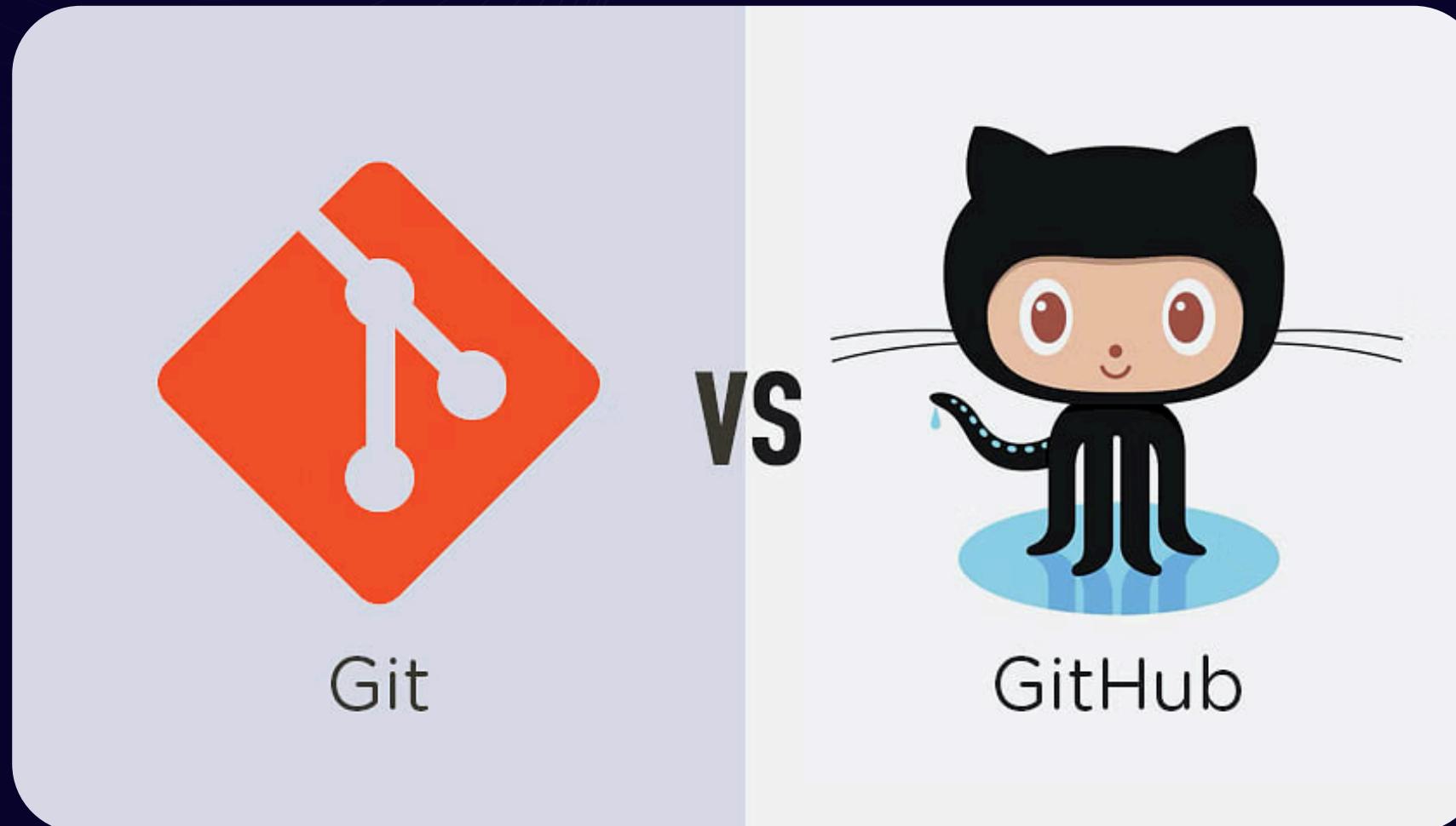
02

Não perder trabalho - se há algo a funcionar, pode-se guardar permanentemente

03

Fazer desenvolvimento incremental

GIT VS GITHUB



01

Muito relacionados, mas diferentes

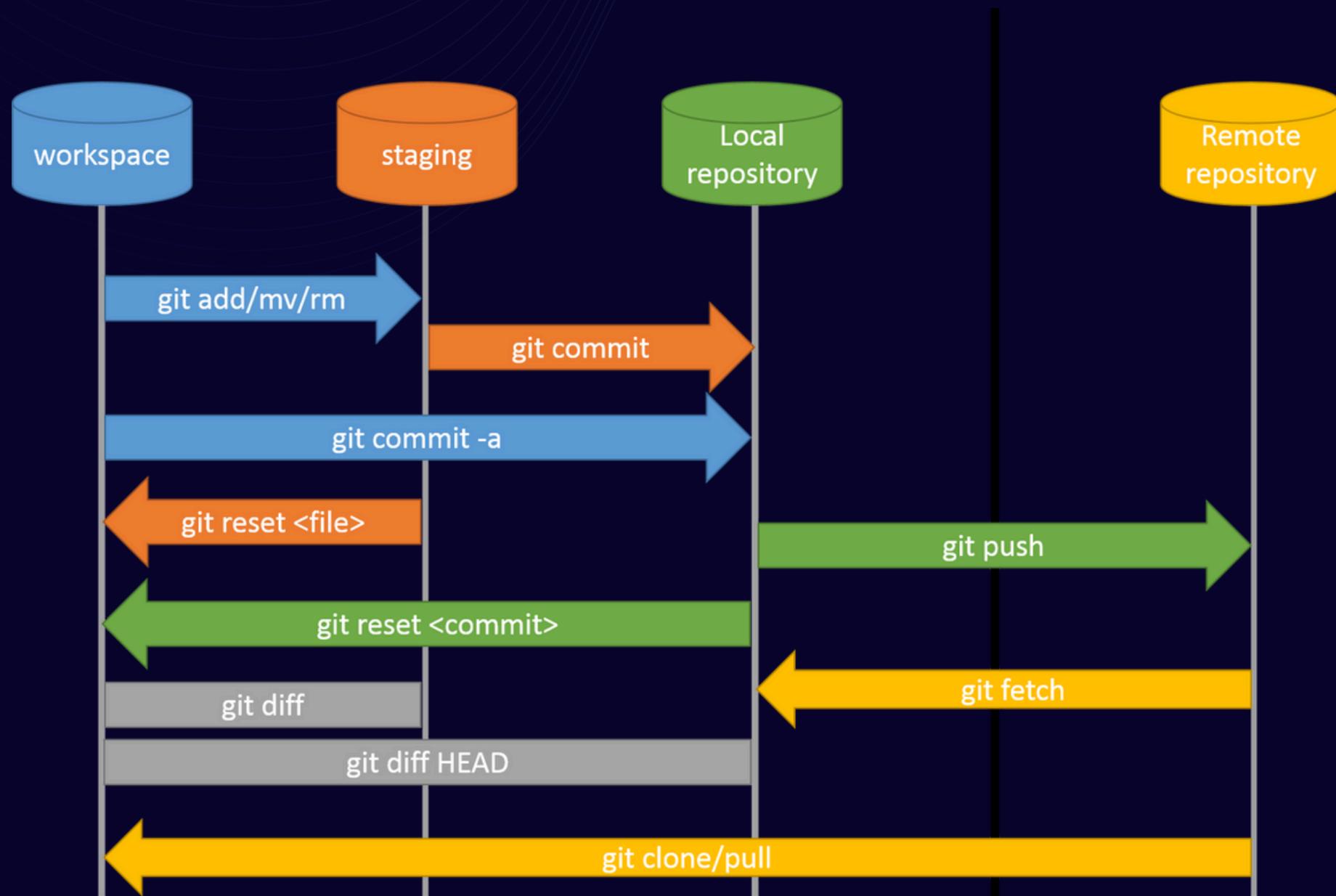
02

GitHub: plataforma colaborativa onde podemos pôr os nossos **repositórios Git**

03

Git: sistema que dá track a mudanças em ficheiros ao longo do tempo - **version control**

KEY CONCEPTS



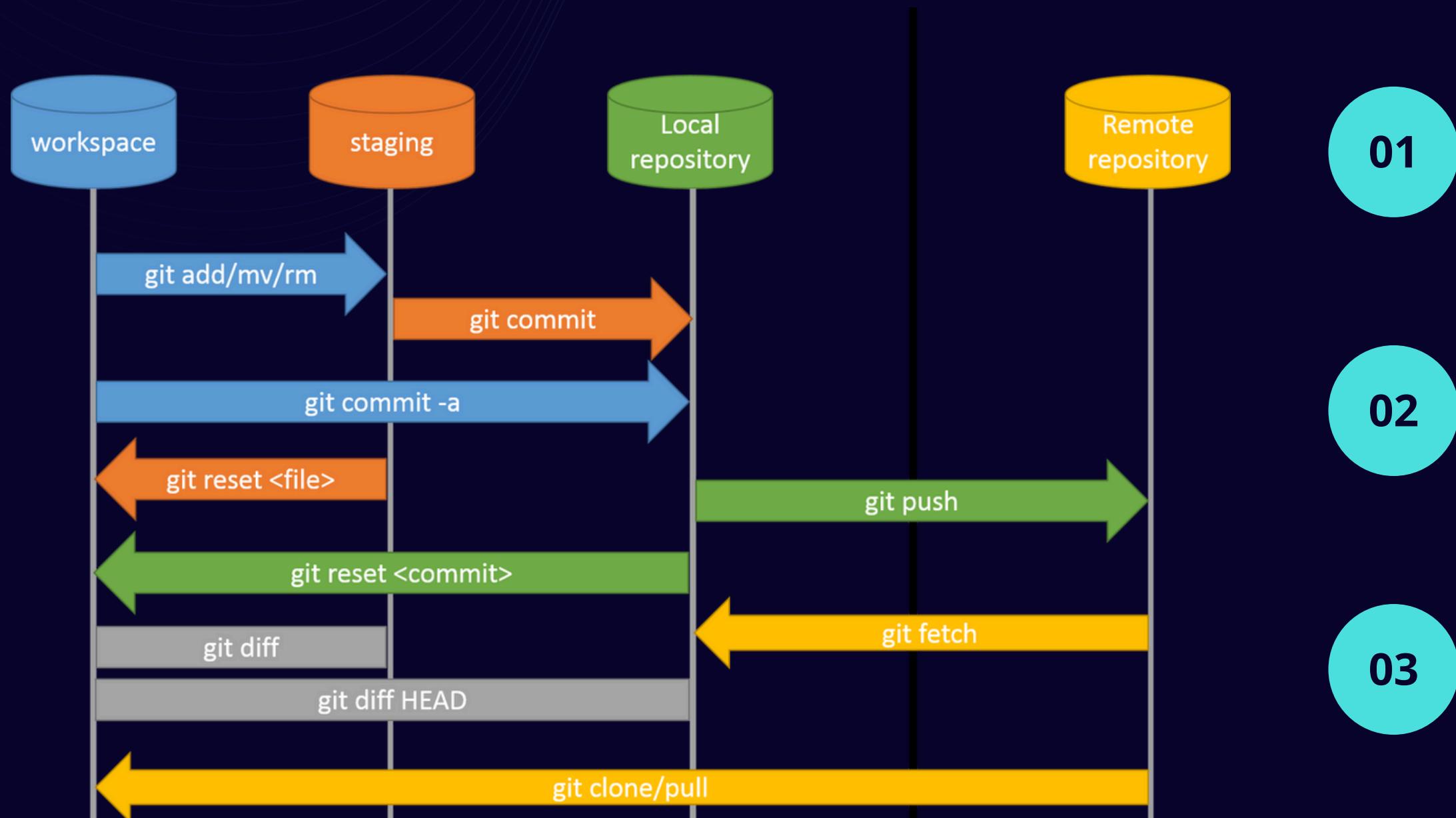
01
Untracked files: ficheiros que não estão a ser geridos pelo Git

02
Tracked Files: ficheiros que adicionamos ao git (via *git add*)

03
HEAD: Ponteiro para o estado atual do repositório

04
Commits: Salvaguardas do estado atual da árvore de ficheiros num dado momento (identificado por um **hash**)

KEY CONCEPTS

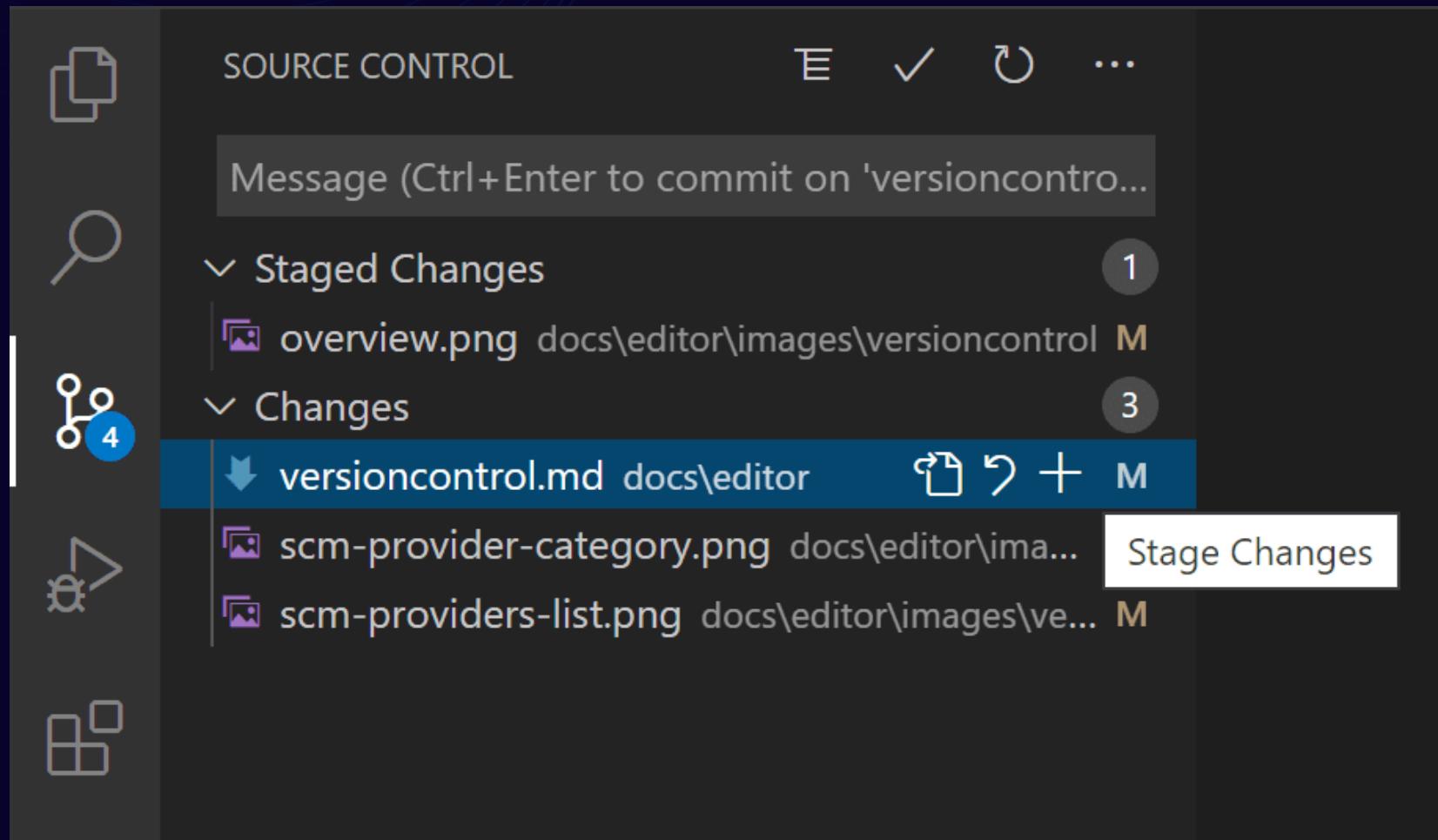


Sandbox (Working Directory): Pasta local onde estamos a trabalhar e editar ficheiros

Staging Area: É um armazenamento temporário para as mudanças que queremos guardar no próximo commit

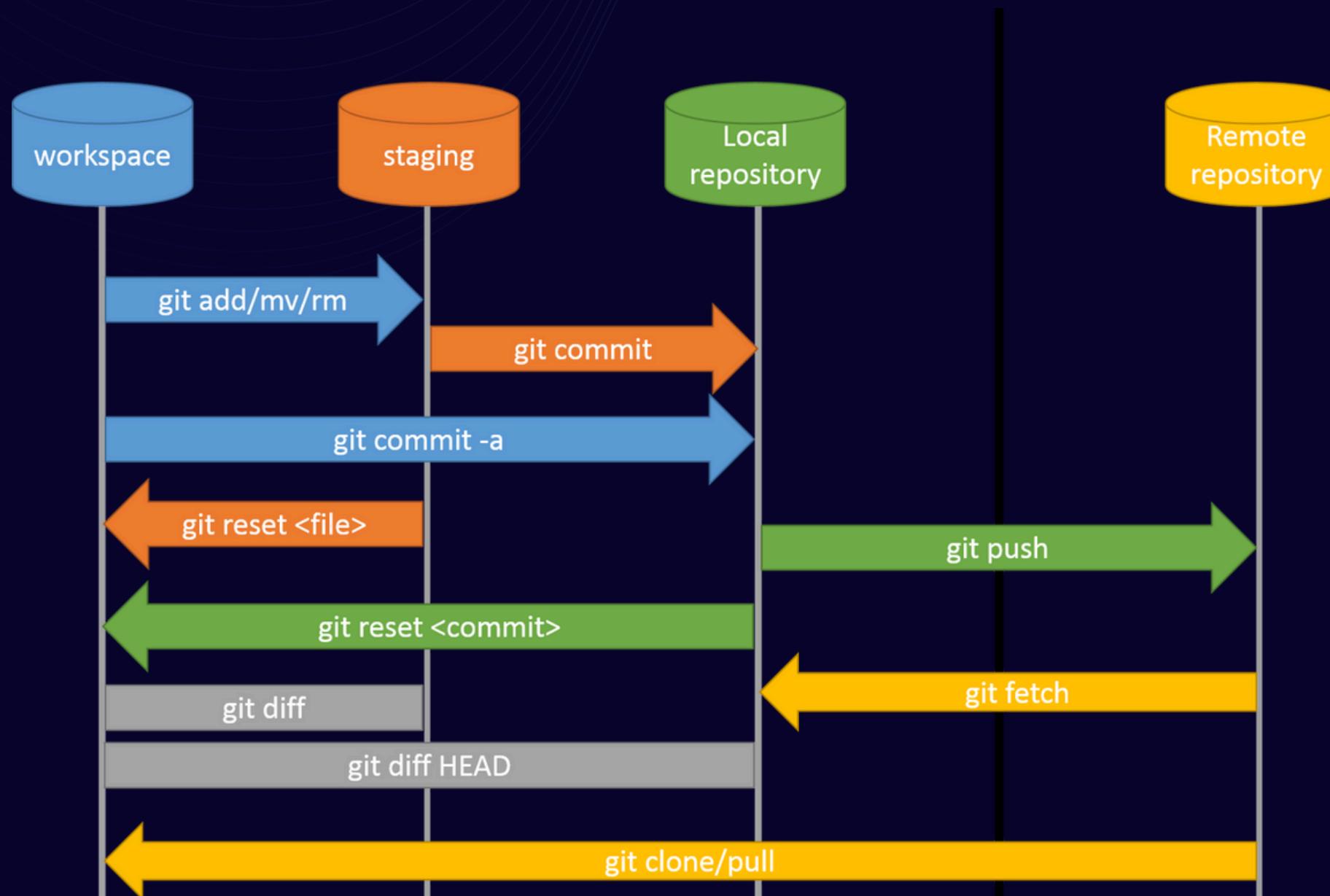
Repository: Os repositórios guardam as alterações permanentes durante o tempo, chamadas commits. Podem ser locais ou remotos

VSCODE GUI



- A **GUI do VSCode** é excelente e intuitiva para novos utilizadores de Git -> vou-vos ensinar a utilizá-la na **parte prática**
- Para entenderem o que está de facto a acontecer, vou-vos obrigar primeiro a correrem os comandos na consola por vários motivos:
 - Para **averiguarem o estado do repositório** e da workspace e staging area
 - Para **entenderem o que está a acontecer "under the hood"**
 - É **mais fácil de corrigir erros** e problemas que inevitavelmente surgem

KEY COMMANDS



git add <file_y> ...: adicionar ficheiros à staging area

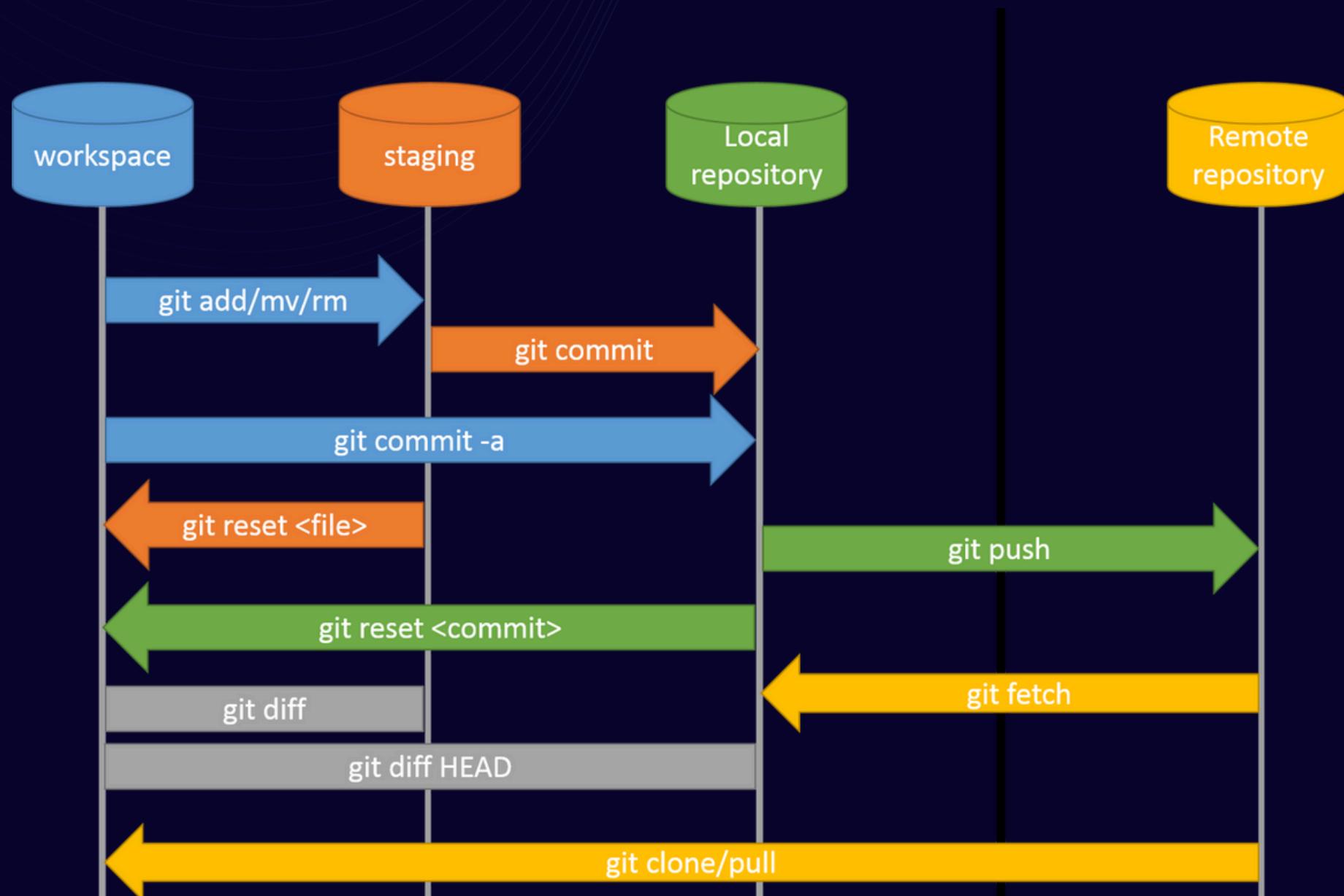
git rm <file_x>--cached: remove ficheiros da staging area, mas mantém a sua existência

git status: ver o estado do working directory e da staging area

git commit -m “<commit_message>”: guarda o estado dos ficheiros que foram adicionados à **staging area** no **repositório local**

git push: Envia as alterações do nosso repositório **local** para o repositório **remoto**

KEY COMMANDS



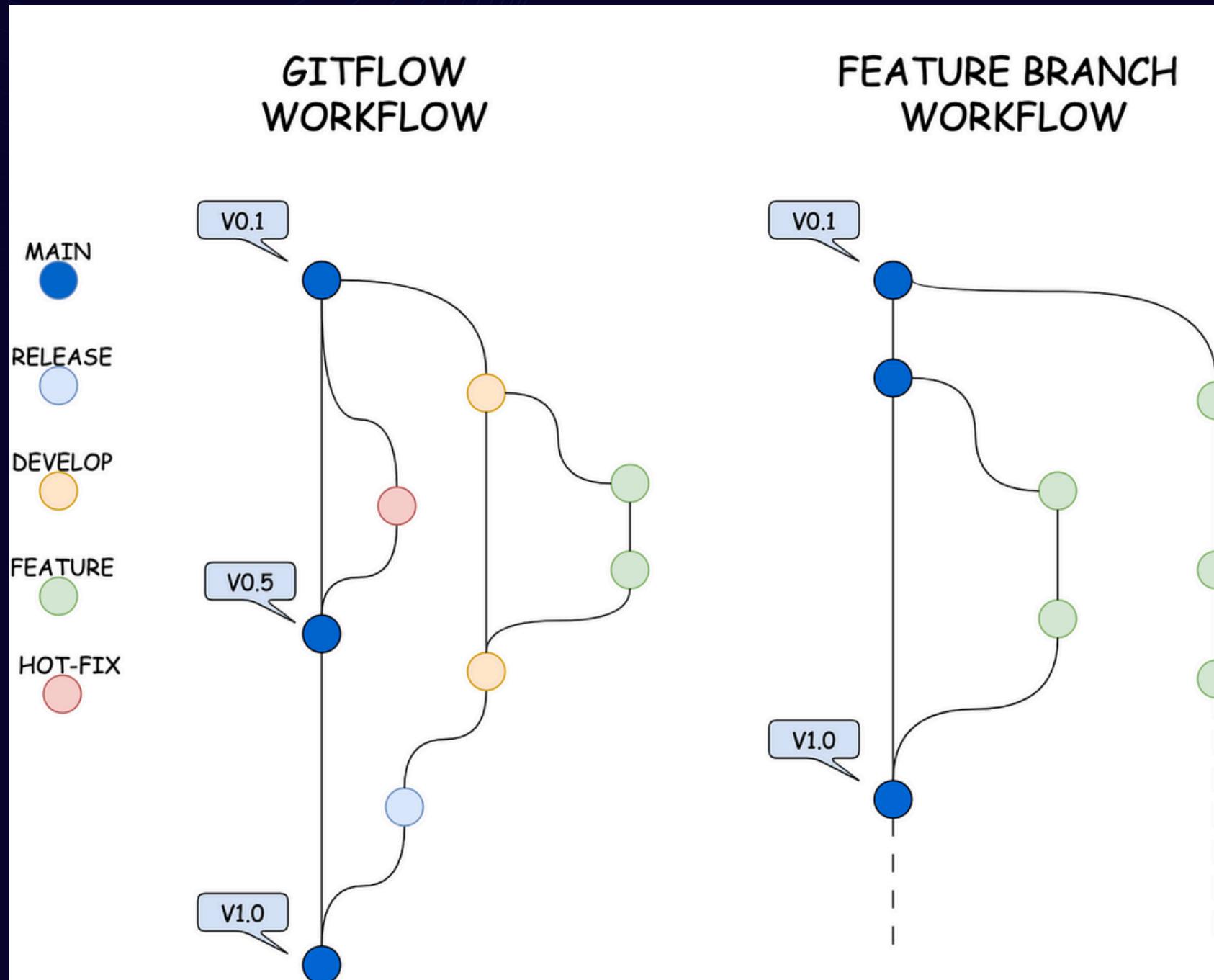
git pull: faz download das alterações do repositório remoto para o nosso local, e faz merge

git log: ver o histórico de commits num branch num dado instante

git init: inicializa um repositório git na diretoria onde estamos

git clone: clona um repositório git existente para a diretoria onde estamos

GIT WORKFLOW



Main branch: a linha principal de desenvolvimento - é crucial que só tenha código que foi testado e funciona!

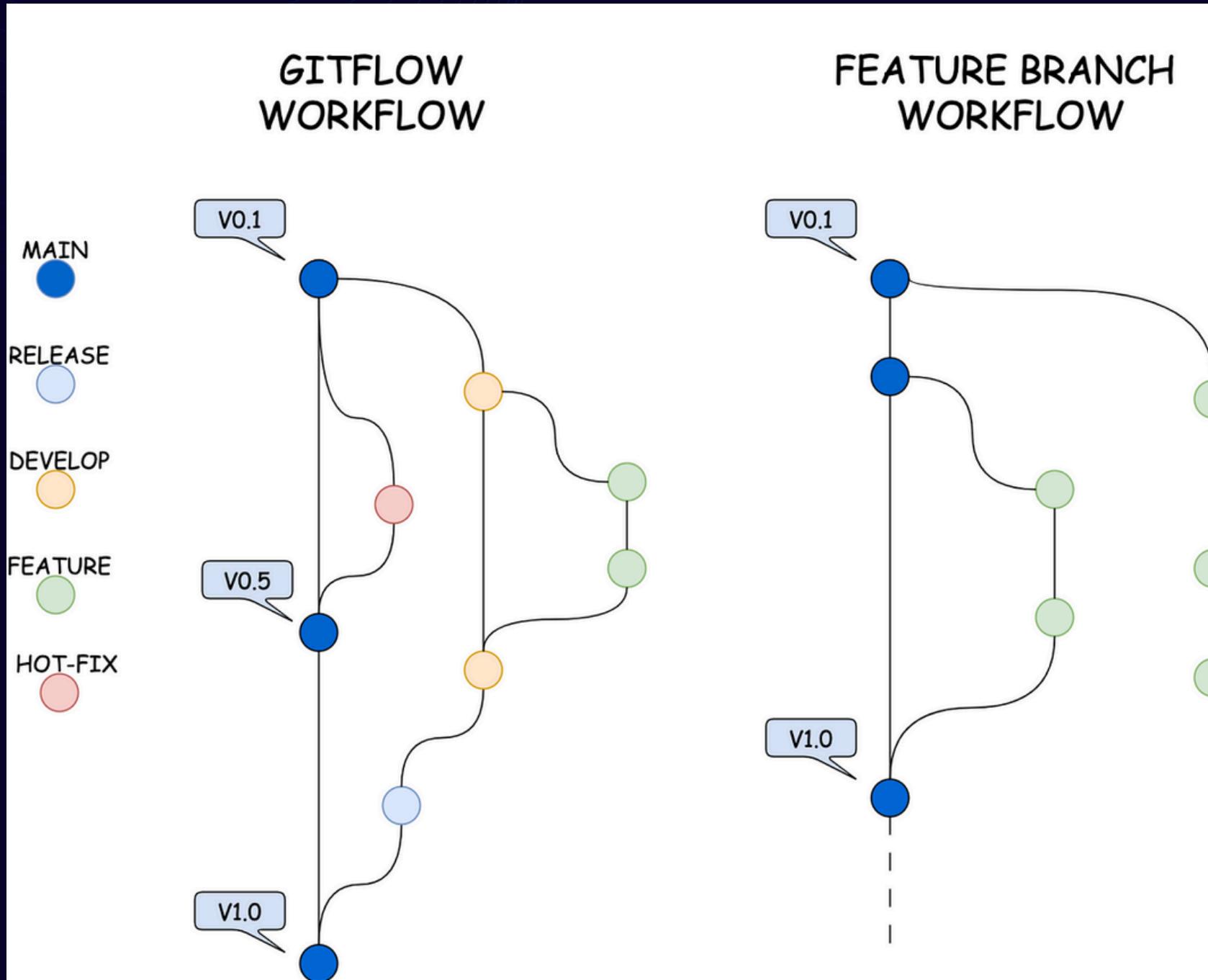
- main

02

Feature branches: são utilizados para desenvolver funcionalidades isoladas

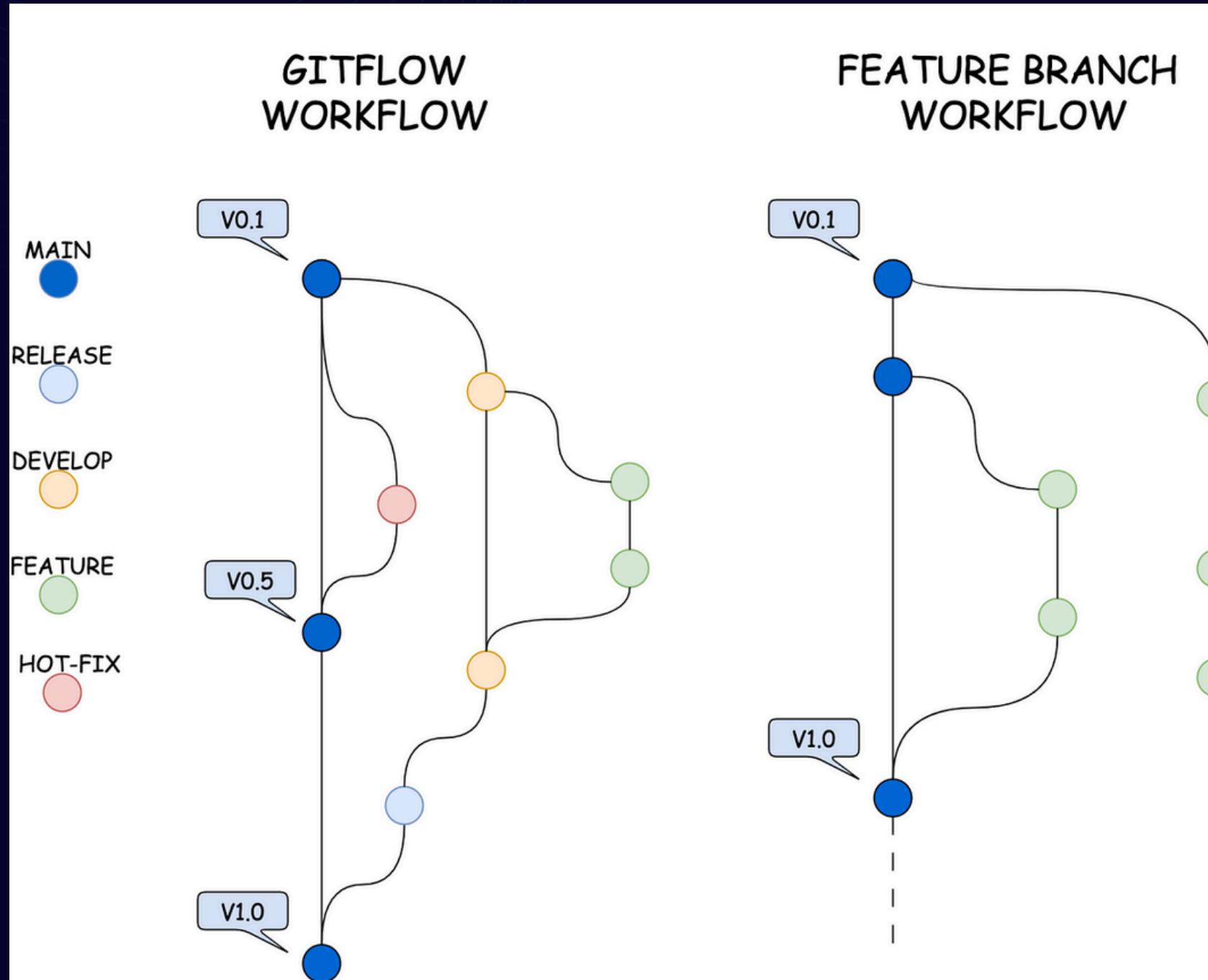
- feat/Pedal
- test/VESC

GIT WORKFLOW - COMMANDS



- 01 **git branch feat/X:** cria um novo branch com nome feat/X
- 02 **git checkout feat/X:** troca do branch atual para o branch feat/X
- 03 **git merge <branch-name>**

GIT WORKFLOW - COMMANDS



- Geralmente depois de acabarmos uma feature:

(Estamos no branch feat/X)

git checkout main: vamos para o branch “main”

(Estamos no branch main)

git merge feat/X: damos merge do branch “feat/X” no branch “main”

git log: podemos ver que os commits que o branch “feat/X” tinha a mais foram para o main!

PRACTICAL PART

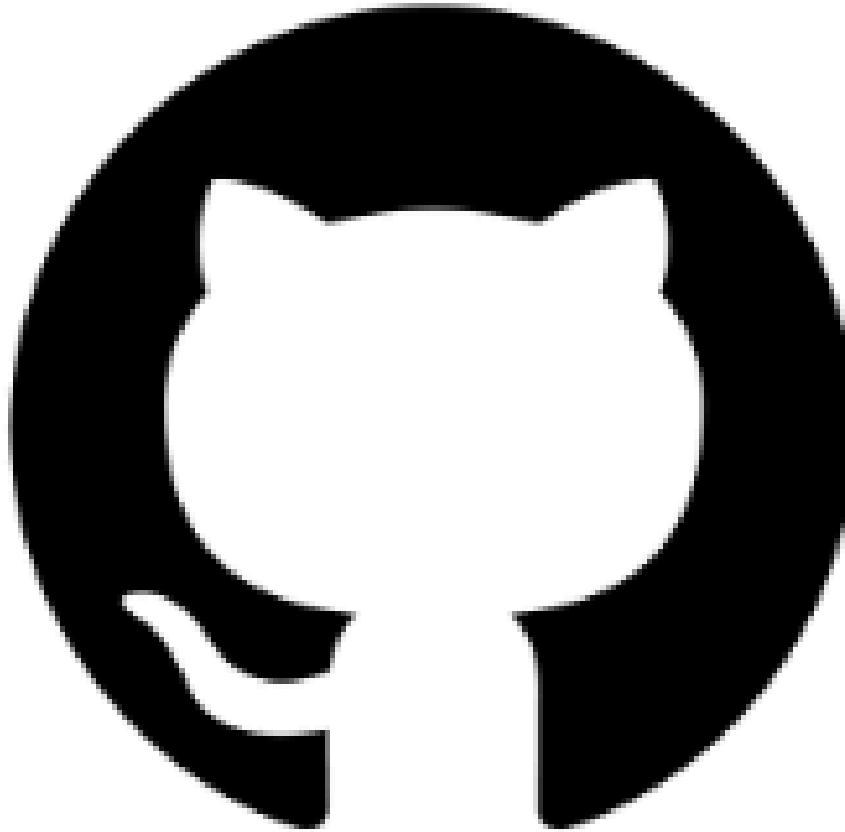
01

Setup para
Windows

02

Clonar um
repositório, criar
um nosso e
correr comandos

github.com/pedro-curto/Git-Workshop



GitHub