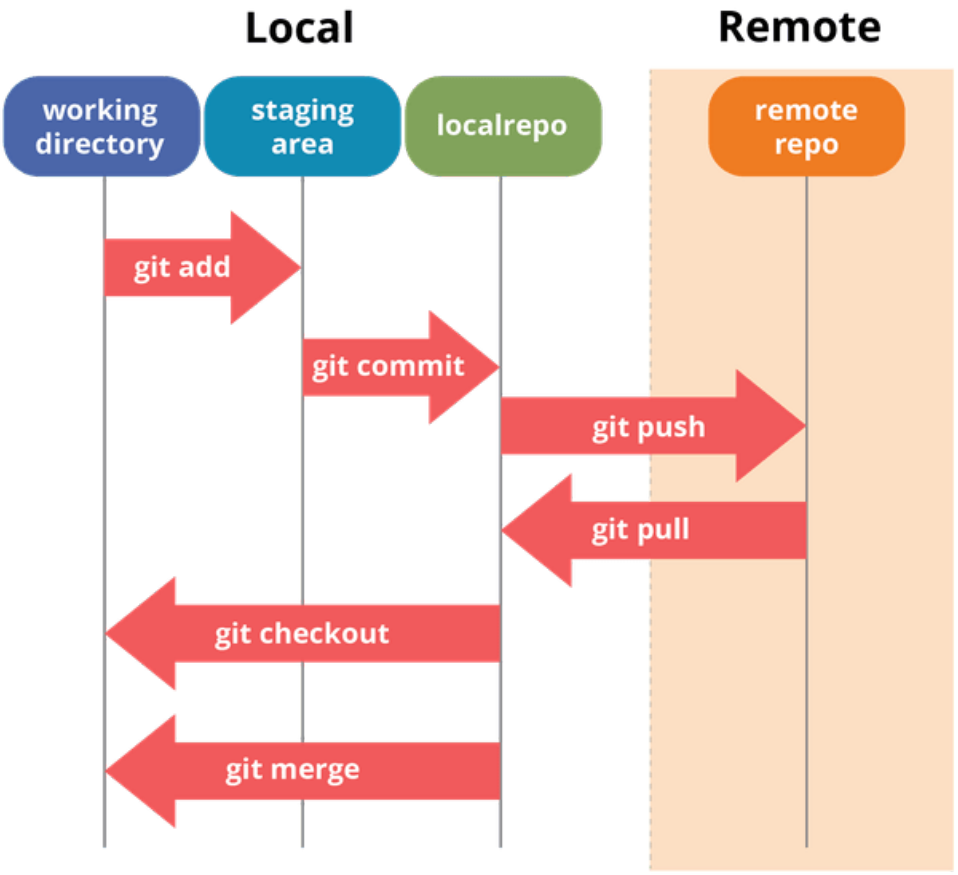
Guia básico do Git

Elaborado por Matheus Moroti.

> 💡 Aqui estão alguns **comandos básicos** que você deve conhecer ao usar Git, assim como o respectivo fluxo de trabalho. Tenha em mente que o **Git** irá trabalhar como o "*software*" no seu **repositório local** que irá versionar seu projeto e que ficará armazenado no seu **repositório remoto**, o **GitHub**.



Fonte da imagem

- `git init`: Inicializa um novo repositório Git.
    - Contexto de uso: Ao criar um novo projeto, quando você já tiver o Git instalado, ao executar isso no terminal você irá criar um novo projeto para ser versionado. **Lembre-se sempre de criar um ambiente virtual para cada projeto**.
- `git clone [url]`: Clona um repositório existente a partir de uma URL.
    - Contexto de uso: Você já trabalha com um colaborador em um projeto versionado no GitHub e gostaria de trabalhar naquele projeto. Outra opção comum de uso, é baixar para explorar algum repositório público de um artigo ou método de interesse.
- `git status`: Mostra o estado atual do repositório, incluindo mudanças não commitadas.
    - Contexto de uso: Útil para identificar quais arquivos o Git está versionando no seu repositório local e quais arquivos que já estão no seu repositório remoto (GitHub) e sofreram modificações no repositório local.
- `git add [arquivo]`: Adiciona um arquivo ao *staging area*.
    - Contexto de uso: Aqui você irá adicionar o que você quer versionar. Fique atento que **arquivos grandes (≥50MB**) devem ser evitados!
- `git commit -m "mensagem"`: Comita (ou envia) as mudanças no repositório com uma mensagem descritiva.
    - Contexto de uso: Ao fazer esse comando, você irá adicionar uma mensagem para o seu eu do futuro, revisor, colaborador, ou qualquer pessoa que tenha interesse em ter acesso ao seu projeto. Seja **sucinto e informativo**, evite “adicionei” tal coisa, uma vez que isso é redundante.
- `git push`: Envia os commits (arquivo + mensagem) locais para o repositório remoto.
    - Contexto de uso: Aqui seu repositório remoto receberá todas suas alterações locais e atualizará o repositório remoto.

- `git pull`: Atualiza o repositório local com as mudanças do repositório remoto.
    - Contexto de uso: Seu colaborador fez modificações, enviou essas alterações, mas seu projeto local não é atualizado automaticamente, afinal, é seu repositório local. Para receber essas modificações no seu repositório local, é só executar esse comando. Fique atento sempre à informar todos colaboradores que modificações foram enviadas ao remoto, pois se você está em uma linha do tempo distinta do colaborador, podem surgir problemas de conflitos. O Git é robusto para resolver esses problemas e existem diversas soluções, como criar linhas do tempo próprias ou mesmo resolver esses conflitos ao comparar as modificações. No entanto, considero isso como um uso intermediário e não irei entrar em detalhes aqui. Para mais informações, leia aqui.
- `git branch`: Cria uma *branch* no repositório local.
    - Contexto de uso: Uma das opções de trabalhar em um projeto colaborativo, ou caso você use isso como uma boa prática ao trabalhar em um mesmo projeto. Aqui é como se você criasse "linhas do tempo" distintas do mesmo projeto, onde não necessariamente as modificações irão aparecer em todo histórico do Git. No entanto, considero isso como um uso intermediário e não irei entrar em detalhes aqui. Para mais informações, leia aqui.
- `git checkout [branch]`: **Muda** para uma *branch* específica.
    - Contexto de uso: Aqui você alterna nas *branchs* disponíveis.
- `git merge [branch]`: Mescla uma *branch* específica na *branch* atual.
    - Contexto de uso: Mesclagem é o jeito do Git de unificar um histórico bifurcado. O comando `git merge` permite que você pegue as linhas de desenvolvimento independentes criadas pelo `git branch` e as integre em uma ramificação única, no caso, a que está em uso no seu terminal.