Guia Rápido - Git

arthursilva@alu.ufc.br
4r7hu3.github.io

Este guia busca apresentar da maneira mais rápida e sucinta possível, sendo portanto um “manual de consulta” para qualquer um (especialmente iniciantes), os usos do Git e seus principais comandos básicos.

O que é Git?

Desenvolvido em 2005 por Linus Torvalds, o criador do kernel Linux, o Git é um sistema de controle de versão distribuído e open source criado para facilitar a manutenção de pequenos e grandes projetos, sendo bastante utilizado de maneira colaborativa, permitindo maior velocidade e eficiência em tarefas de desenvolvimento e gerenciamento de projetos.

A prática do controle de versão, seja para grandes ou pequenas tarefas, permite aos usuários rastrear e gerenciar todas as alterações em arquivos dentro de um repositório (uma pasta), dando maior flexibilidade, eficiência, velocidade e segurança aos projetos.

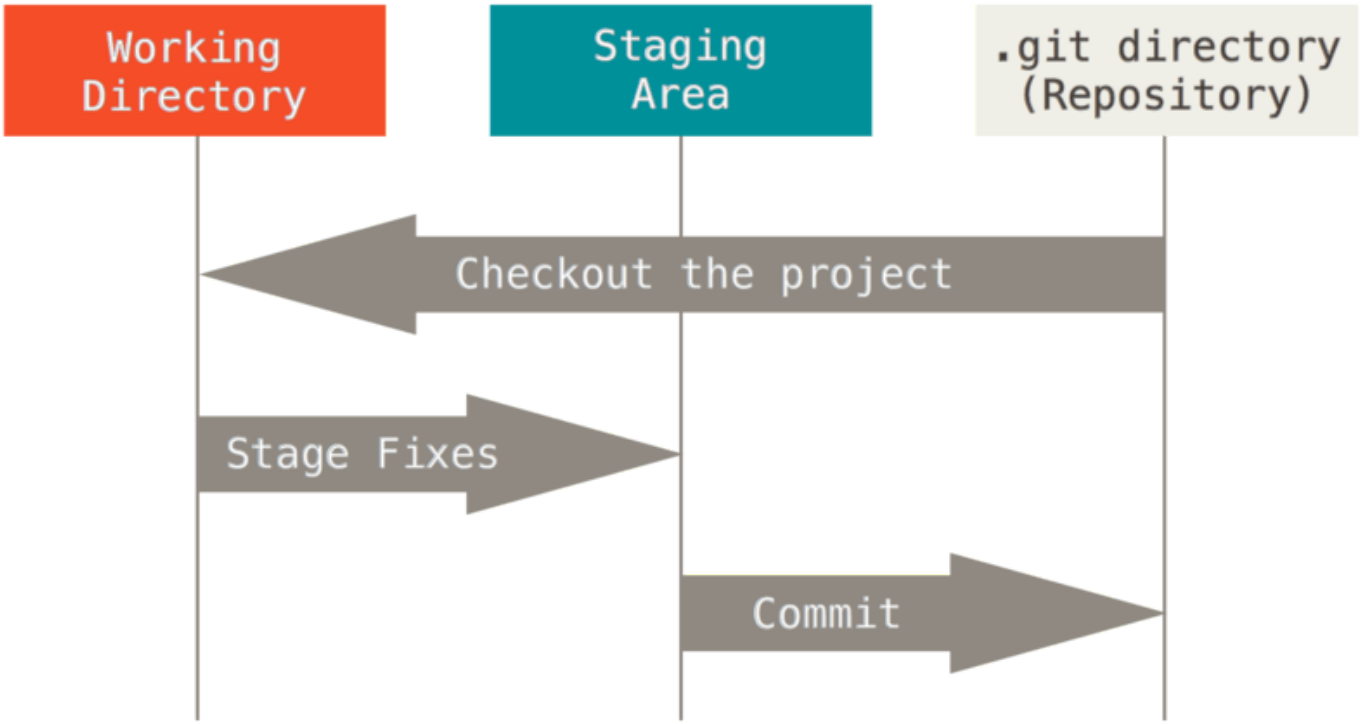
Como funciona

De forma breve, o Git funciona através de capturas de estados (**snapshots**) de arquivos em forma de fluxo, que são armazenados em uma base de dados local (**.git**) contendo todos os metadados do repositório. Dessa forma, praticamente todas as ações que o sistema faz são uma adição de novos dados na base, que registra cada nova alteração, fazendo isso de forma segura, por meio de *checksums* do tipo SHA-1.

Os estados, que formam o fluxo do Git, são:

- **Modified**: você modificou um arquivo, mas não o marcou nesta versão, nem fez o commit
- **Staged** : você modificou um arquivo e o marcou, nessa versão, para o commit
- **Commited**: você fez o snapshot para a base local

Esses estados é que formam o fluxo do Git, ilustrado na figura abaixo.



Assim, o que ocorre ao longo do fluxo é:

1. modificações em arquivos do seu diretório
2. seleção de quais mudanças manter para o snapshot
3. captura das mudanças selecionadas anteriormente
4. retorno ao diretório na sua nova versão

Git e GitHub

É muito comum que ingressantes na área de tecnologia confundam o Git com o GitHub, porém, é importante deixar claro que são dois projetos diferentes, mas com aspectos em comum.

O GitHub é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão, que funciona através de repositórios remotos e com o Git. Essencialmente, este último é usado de forma local (na própria máquina), por meio de linhas de comando em um terminal, enquanto o GitHub é acessado na web, fornece uma interface gráfica atrativa, de fácil compreensão e uso.

A aplicação do Git independe de qualquer tipo de repositório remoto. Por outro lado, usar repositórios remotos sem o Git, apesar de possível, não é nada vantajoso. O essencial e recomendável, é a

combinação dos dois, pois permite um gerenciamento muito mais poderoso e seguro, em praticamente qualquer projeto.

Conectando ao GitHub

Para realizar a conexão do repositório local com um repositório remoto, primeiro, é necessário fazer uma autenticação, que pode ser via HTTPS, ou SSH, que são dois protocolos de rede seguros bastante usados na internet. Abaixo, explico como configurar.

Configurando via HTTPS

Para conectar um repositório remoto a um local, basta copiar a url dada em “<> Codes” e colá-la após o comando `git clone`. Porém, a cada ação de troca local-remota, serão solicitadas as credenciais do GitHub do usuário por questões de segurança, o que acaba tornando o processo maçante.

Há algumas formas de evitar tal solicitação, mas nem sempre irão funcionar, ou podem ser mais demoradas de implementar. Assim, uma alternativa ao uso do HTTPS, é o protocolo SSH, que necessita de apenas uma configuração, mas viabiliza maior comodidade ao fazer um push, além de ser mais seguro, por usar criptografia ao invés de senhas.

Configurando via SSH

Aqui, omitirei como o SSH funciona, partindo diretamente para a configuração, e considerando uma máquina com sistema Linux (Debian Based). No Windows, você pode rodar esses comandos dentro do Git Bash.

No terminal, execute os seguintes comandos:

1. `ssh -V` (verifique se o SSH está instalado, caso não, execute `sudo apt install openssh-client`)
2. `ssh-keygen -t rsa -C seu@email.com` (gera as chaves privada e pública no diretório .ssh)
3. `ls .ssh` (verifica as chaves criadas)
4. `cat .ssh/id_rsa.pub` (imprime o conteúdo da chave pública no terminal, copie-a!)

Por fim, vá até o seu *GitHub => Settings => SSH & GPG Keys => New SSH Key*.
Dê um título a sua chave, e cole-a no espaço indicado.
Confirme a conexão com o comando `ssh -T git@github.com`, no terminal.

Tudo pronto! Agora, para clonar repositórios na sua máquia com o SSH, basta copiar o link fornecido em “<> Code”, e colá-lo após o comando `git clone`.

Comandos

- `git config --global user.name ['name']`: seta o nome do usuário
- `git config --global user.email [email]`: seta o e-mail do usuário

- `git config --list`: lista todas as configurações do git no diretório
- `git init [project_name]`: inicia um repositório local e o nomeia (opcional)
- `git clone [url]`: clona um repositório remoto e todo seu histórico
- `git remote add [remote_name] [url]`: adiciona um repositório remoto
- `git status`: mostra o status do repositório antes do commit
- `git add [files]`: adiciona os arquivos à area de stage
- `git commit -m [message]`: faz o snapshot dos arquivos e adiciona uma mensagem
- `git diff`: mostra as modificações nos arquivos antes do commit
- `git log`: lista o histórico de versões (commits) do repositório
- `git reflog`: lista o histórico de versões de maneira compacta
- `git show`: mostra o conteúdo de um ou mais commits
- `git push`: envia os commits para o repositório remoto
- `git pull`: baixa as mudanças no repositório remoto para o repositório local
- `git reset [option] [commit_id]`: retorna para um commit específico, mantendo ou não as alterações
  - `git reset --soft [commit_id]`: retorna ao commit, mantendo as modificações em staged
  - `git reset --mixed [commit_id]`: retorna ao commit, mantendo as modificações em modified
  - `git reset --hard [commit_id]`: retorna ao commit, descartando todas as alterações no diretório
- `git revert [commit_id]`: cria um novo commit desfazendo as mudanças do commit apontado
  - `git revert --no-edit [commit_id]`: cria novo commit, sem mensagem, desfazendo o commit apontado
  - `git revert -n [commit_id]`: desfaz o commit apontado, mas sem criar novo commit para isso
- `git restore`: restaura arquivos específicos, descartando mudanças, sem atualizar o históricos de commits
  - `git restore --staged [file]`: remove um arquivo da área de stage, mantendo as mudanças
  - `git restore --source [commit_id] [file]`: restaura um arquivo na versão de um commit apontado
- `git clean`: remove arquivos e/ou diretórios que não são rastreados pelo git
- `git branch`: lista todas as branches do repositório
- `git branch -m [name]`: muda o nome da branch corrente
- `git branch [name]`: cria uma nova branch
- `git switch [branch]`: muda para uma branch

- `git branch -d [branch]`: exclui uma branch
- `git merge [branch]`: faz o merge da branch corrente com a branch apontada, criando novo commit

Arquivos

- `.gitignore`: diz ao git quais arquivos ignorar, e é visto pelos usuários
- `.git/info/exclude`: diz ao git quais arquivos ignorar, mas não é visto pelos usuários

Links úteis

- Documentação Oficial
- Pro Git
- Reset vs Revert
- reset --hard
- GitHub com SSH
- Autenticação no GitHub