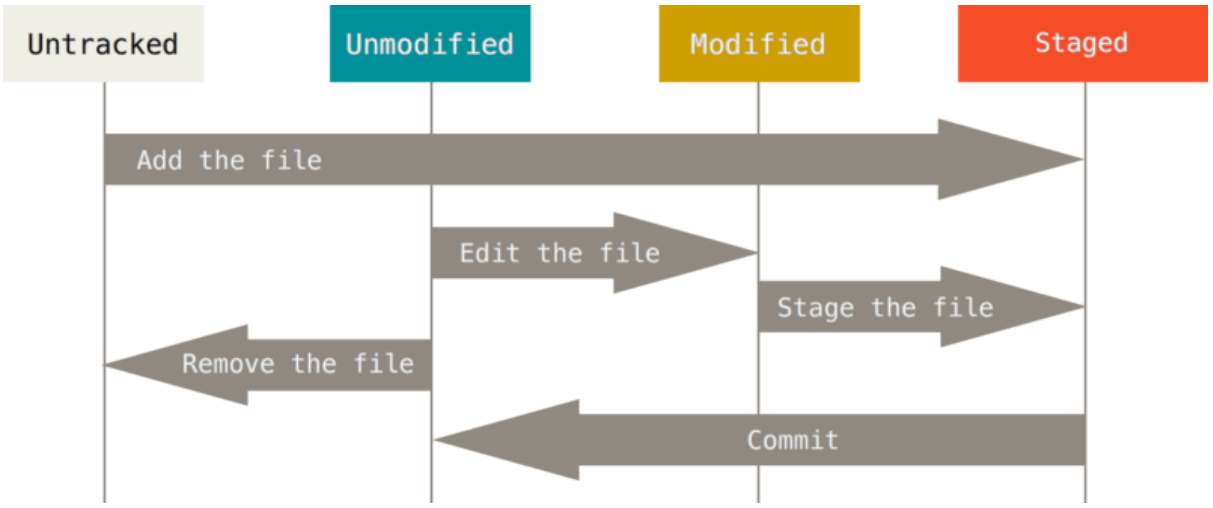
Ciclo de vida de um arquivo no Git



git init: Cria um repositório no diretório corrente.

Primeiramente, quando um repositório é iniciado, se existir arquivos, eles não estarão sendo monitorados, isso significa que o estado desses arquivos serão **Untracked**. O mesmo estado será atribuído para arquivos criados/adicionados em um repositório existente. Para arquivos **Untracked** serem monitorados pelo git é necessário usar o comando **git add**.

git add + nome do arquivo: Altera os arquivos para o **Staged**, o que significa que estarão prontos para serem comitados.

Commit: "*...refere-se ao processo de tornar permanente um conjunto de alterações, ou seja, de efetivar as alterações.*" - wikipedia.

git commit -m "descrição do commit": Comando para registrar uma versão do repositório. Quando o commit é realizado uma mensagem é retornada dizendo o local e o identificador desse registro.

Como resultado de um versionamento realizado após um commit, os arquivos de um repositório ficaram no estado **Unmodified**, o que

significa que eles apenas estarão sendo monitorados aguardando uma alteração.
Em razão de uma alteração feita em um ou mais arquivos no repositório, os arquivos alterados serão alterados para o estado **Modified** podendo ser alterados novamente ou serem comitados.

Configurações

git config --global username “nome do usuário”: Configura o nome do usuário.

git config --global user.email “user@provedor.com”: Configura o email do usuário.

git config --global core.editor + comando do editor: Configura o editor padrão que o git irá usar.

git config user.name: Exibe o nome do usuário.

git config user.email: Exibe o email do usuário.

Visualizando os logs

git log: Mostra os registros dos commits. Cada registro possui um identificador, autor, data que o commit foi realizado e a descrição.

git log --decorate: Exibe informações adicionais do registro.

git log --author “nome do autor”: Exibe uma lista com todos os commits feitos pelo autor determinado no valor do comando.

git log --author + “nome do autor”: Exibe uma lista com todos os commits feitos pelo autor determinado no valor do comando.

git shortlog: Exibe uma lista em ordem alfabética quais foram os autores, quantos e quais commits fizeram.

git shortlog -sn: Exibe a quantidade de commits e o nome do usuário que commitou.

git log --graph: Mostra os registros com detalhes gráficos.

git show + identificador do commit: Exibe os detalhes de um commit específico através do identificador passado no valor do comando.

Visualizando os logs

git diff: Exibe cada mudança realizada no repositório antes do commit ser feito.

git diff --name only: Exibe apenas o nome dos arquivos que foram modificados.

git diff --name only: Exibe apenas o nome dos arquivos que foram modificados.

Desfazendo ações

git checkout + nome do arquivo: Reseta o arquivo para o estado anterior de alguma edição.

git reset HEAD: Retorna arquivos para o estado anterior ao **Staged**.

git reset --soft + identificador: Desfaz o commit identificado pelo identificador passado como valor no comando, retornando-o para o estado **Staged.**

git reset --mixed + identificador: Desfaz o commit identificado pelo identificador passado como valor no comando, retornando-o para o estado **Modified.**

git reset --hard + identificador: Desfaz o commit identificado pelo identificador passado como valor no comando, retornando-o para o estado anterior ao **Modified, i**sso significa que as alterações no arquivo

também serão desfeitas. Os arquivos ficarão no estado **Staged** após o comando.

Ligando o repositório local com o remoto

git remote add origin + link do repositório remoto: Comando usado para associar o repositório local com o repositório remoto. O nome *“origin”* é um pseudônimo padrão que determina a origem do repositório remoto, mas pode ser usado qualquer outro nome.

git remote -v: Exibe informações sobre o repositório remoto.

Enviando mudanças para o repositório remoto.

git push -u origin master: Comando usado para enviar todo conteúdo do repositório local para o remoto. O parâmetro “-u” é usado para salvar o caminho que indica a origem e o destino e não ser necessário informar o caminho na próxima vez que houver um *“push”.*

Clonando um repositório.

git clone + link do repositório remoto: Faz uma cópia de um repositório remoto criando um repositório local no diretório corrente.

Fork de um projeto.

Fork é uma cópia de um projeto de uma pessoa para que seja possível fazer contribuições no projeto e posteriormente enviar as alterações para a pessoa. O fork se diferencia do clone pois este não permite enviar as modificações para o proprietário original do projeto.

O que é um branch e por que usar?

Branch é um ponteiro móvel que leva a um commit.

Master: É o branch original que existe desde o primeiro commit.

Frequentemente é necessário criar outros branches separados do branch master com o propósito de preservar uma versão original sendo facilmente possível, criar ou excluir alterações evitando conflitos.

Criando um branch

git checkout -b + nome do branch: Cria um branch com o nome passado no valor do comando.

git branch: Exibe os branches existentes no repositório destacando o branch corrente.

Movendo entre os branches e deletando

git checkout + nome do branch: Seleciona o branch desejado através do nome passado no valor do comando.

git branch -D + nome do branch: Apaga o branch referente ao nome passado no valor do comando.

Entendendo o merge

Ao passo que um branch é criado de forma dependente ao branch antecessor, é provável que a criação de vários branches irá gerar várias ramificações.

git merge + nome da branch: Gera um novo commit unindo a branch que possui o nome que foi passado no valor do comando. O objetivo

principal do merge é unir as versões em destruir as versões independentes do branch principal.

Entendendo o rebase

Tendo em conta que o merge une branches em um novo commit, existe uma maneira de restabelecer a estrutura ramificada realocando branches em sequência de fila.

git rebase + nome da branch: Adiciona na branch corrente o branch que possui o nome passado no valor do comando, evitando a criação de um novo commit com o preço da perda do histórico cronológico.

Criando um .gitignore

Quando pensamos sobre a necessidade de repositórios possuírem arquivos, mas que esses arquivos não devem ser subidos para repositórios remotos ou há a necessidade que um commit seja feito com exceção de certos arquivos, é necessário criar um arquivo com a extensão .**gitignore**. O conteúdo desse arquivo deve conter as regras indicando quais arquivos devem ser ignorados.

Entendendo o git stash

git stash: Transfere o estado de um repositório do **Modified** para o estado **WIP**, isso significa que o repositório ainda está para ser alterado mas permite o usuário mudar de branch.

git stash apply: Modifica o estado **WIP** para o estado **Modified.** Isso permite que a edição dos arquivos seja possível.

git stash list: Exibe uma lista de todos os repositórios no estado **WIP**.

git stash clear: Lima todos arquivos que estiverem no estado **WIP**.