# Caderno de Estudos de GIT

**Anotações dia 1 – 12/05/24: (Livro da O’Reilly)**

**TABELA DE COMANDOS**

**Azul é comando bash e laranja comando git.**

|  |  |  |  |
| --- | --- | --- | --- |
| **COMANDO** | **FUNÇÃO** | **OPÇÕES** | **ARGUMENTOS** |
| **pwd**  (print working directory) | Mostra o caminho do diretório atual |  | Não |
| **clear** | Limpa o terminal |  | Não |
| **ls**  (list) | Mostra todos os arquivos e diretórios dentro do diretório atual | **-a** : ocultos também | Não |
| **cd**  (change directory) |  | **..** : volta para o diretório pai | Sim, diretório para onde quer ir |
| **mkdir**  (make directory) | Criar diretório |  | Sim, nome do diretório novo |
| **git version** | Mostra a versão atual do git |  | Não |
| **git config**  **git config --global --list**  **git config --global user.name “<nome>”**  **git config --global user.email “<email>”** | Visualizar ou editar configurações do git | Uma infinidade, as principais:  --global  --list  --global --list | As vezes sim, principalmente se for editar algo, ex:  --global user.name “seuNome” |
| **git init** | Transforma um diretório qualquer em repositório git |  |  |
| **git add .** | Adiciona todos os arquivos do diretório editados para a área stage (pré-commit) | substituir o . por outra coisa, o . significa todos os arquivos |  |
| **git commit** | Cria um | -m : adiciona uma mensagem de commit  -am : faz o trabalho do git add e da mensagem juntos | Coloca a mensagem entre aspas depois do -m |
| **git remote add origin git@github.com:usuario/repositorio.git** | conecta seu repositório local com o remoto |  |  |
| **git push -u origin main** | envia os dados do repositório local pra branch main do repositório remoto, só precisa usar uma vez |  |  |

Git é uma tecnologia usada para controle e compartilhamento de versionamentos de projetos.

Para instalar o git basta acessar o site oficial e baixar, a versão recomendada deve ser acima da versão 2.28.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Para verificar a versão instalada do git basta digitar

git version

O git pode ser utilizado através tanto de uma janela de comando chamada Shell como através de uma interface gráfica chamada Git GUI (no caso do livro é na verdade apenas o gerenciador de arquivos).

Normalmente um git bash (shell do git) mostra isso para descrever o local no computador que está sendo acessado no momento:

 Sinceramente ainda não entendi o MINGW64 ali.

Mas a parte verde é o usuário e o ~ é o diretório atual, naquele caso é a pasta do usuário, se mudar aparece outro caminho.

Alguns comandos possuem opções e argumentos

git commit -m “Message”

Nesse caso git commit é o comando, -m é a opção e “Message” é o argumento.

O comando clear limpa a tela.

Para visualizar o caminho atual, digite o comando pwd, esse comando significa Print working directory (exibir diretório de trabalho).

É importante lembrar que existem arquivos normais e arquivos ocultos (meio transparentes no gerenciador de arquivos e geralmente começam o nome com .

O comando ls, que significa list exibe todos os diretórios e arquivos visíveis dentro do diretório local. Sua variação, o ls -a, mostra os ocultos também.

Para começar a trabalhar com git é necessário configurar seu ambiente através do comando git config. Uma opção interessante para esse comando é --global --list, com essa opção são exibidas as informações sobre o git local (caso existam), caso não existam, ocorrerá um erro. As duas principais configurações no momento são o user.name e o user.email, para alterá-las basta digitar git config –global –list user.email e user.name passando como parâmetro entre aspas os valores desejados.

Terminal integrado é uma janela de linha de comando integrada à um editor de rich text (código basicamente).

**Anotações dia 2 – 18/05/24: (curso do Nélio Alves)**

Um repositório é um projeto controlado pelo git. Repositórios podem ser remotos (github, gitlab, bitbucket, etc) ou locais (no seu pc).

**Conceito de SSH**

SSH é um protocolo para comunicação de dados com segurança. É possível configurar quais dispositivos podem acessar seu GitHub gerando uma chave SSH no seu PC e enviando para o GitHub.

**Git Status**

Esse comando exibe as alterações dentro do repositório que estão prontas para serem adicionadas na área de stage ou commitadas.

**Anotações dia 3 – 06/06/24: (curso do Nélio Alves)**

**Gerar chave SSH**

Para permitir que um PC possa fazer commits na sua conta, é preciso gerar uma chave SSH para ele, para isso, basta digitar esse comando:

ssh-keygen -t ed25519 -C your\_email@example.com

Dar enter na parte do local do arquivo, dar enter na senha (ou colocar).

Depois disso, para permitir acesso ao github, basta ir no site, em configurações, selecionar SSH:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Depois Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente e então adicionar a chave e dar um nome pro PC.

**Anotações dia 4 – 16/06/24: (curso do Nélio Alves)**

**Guia para criar repositório e subir no GitHub**

**Passo 1**

**Configurar usuário**

git config --global user.name “<Seu Nome>” – Define o nome do usuário do git

git config --global user.email “<Seu Email>” – Define o email do usuário do git

git config --list – Exibe as configurações

**Passo 2**

**Gerar chave SSH e permitir acesso à conta**

ssh-keygen -t ed25519 -C “<Seu Email>” – Gera chave de acesso SSH no padrão de criptografia ed25519

**ou**

ssh-keygen -t rsa -b 4096 -C “<Seu Email>” – Gera chave de acesso SSH no padrão de criptografia rsa4096

Depois disso aparecerá a opção de mudar o diretório onde a chave SSH será salva e a opção de colocar senha, a do diretório é bom nem mexer, a da senha é opcional, para ignorar basta só dar enter.

Após isso basta ir no GitHub, configurações, new SSH Key e colar a senha pública gerada (para encontrar ela basta ir no diretório que aparece no terminal.

**Passo 3**

**Criar repositório local e fazer primeiro commit**

git init – Transforma a pasta onde você está no git bash em um repositório git

git status – Mostra se existem modificações pendentes e quais são

git add . ou git add <arquivos> - Envia determinados arquivos para a área stage

Obs: Adicionar mais de um arquivo basta separar com espaço, uma dica é usar o TAB para ele sugerir o nome dos arquivos, facilita muito.

git branch -M main – Muda a branch usada no repositório para main, nome usado para a branch principal (útil se o git estiver impondo a branch master por padrão)

git commit -m “<mensagem>” – Junta todas as modificações e cria um marco na linha do tempo do repositório (uma nova versão), é recomendado que a mensagem descreva o que ocorreu de mudanças no repositório e existem padrões de nomes de commits

**Passo 4**

**Criar repositório remoto e linkar com repositório local**

Vou pular a parte do github de criar o repositório porque é fácil, basta lembrar que tem o Read.me, o gitignore e a licença que tem que tomar cuidado pq pode ter problema pra dar o primeiro push, é bom dar pull primeiro.

Depois de criar o repositório remoto:

git remote add origin git@github.com:<usuario>/<repositorio>.git: Adiciona um repositório remoto para um repositório local, lembrando que esse formato usando git@github é quando temos chave ssh e afins, dá pra usar o formato https também.

git push -u origin main: Envia os commits para o repositório remoto.

**Outros comandos e conceitos fundamentais**

**Clonar repositório remoto**

É possível clonar um repositório usando o seguinte comando:

git clone git@github.com:<usuario>/<repositorio>.git: Clona o repositório remoto para o local, inclusive com branches, logs, etc.

Depois disso basta continuar os comandos normalmente.

**Verificar histórico de commits**

Para visualizar o histórico de commits, basta usar o comando:

git log: Esse comando exibe todo o histórico de versões do repositório, além disso é possível fazer várias customizações nele usando modificadores.

**Verificar alterações em arquivos de um commit para o outro**

git diff: Exibe as alterações (o que foi apagado ou inserido) de um commit para outro, é bem melhor usar no VS Code, indo em Source Control, pois lá é bem mais gráfico.

**O que é HEAD**

HEAD é essencialmente o último commit da branch atual.

**Anotações dia 5 – 20/06/24: (curso do Nélio Alves)**

**Git checkout**

Git checkout é um comando muito útil para várias coisas, dentre elas está a possibilidade de fazer seu código retornar ao estado que estava em determinado commit, para isso basta escrever ‘git checkout <código do commit>’, esse código de commits pode ter a versão longa ou curta.

Vale lembrar que esse tipo de comando não deve ser feito para alterar commits antigos, se você usa o checkout para voltar no tempo e altera o commit de antigamente as coisas começam a ficar meio malucas pois esse commit vai ficar separado da HEAD (último commit de todos). Para especificar commits sem precisar colocar o código deles também podemos usar justamente a palavra chave HEAD e um ~x logo depois, sendo x um número de commits ‘para trás’ da HEAD, exemplo:

git checkout HEAD~1 retorna para o commit logo antes do HEAD (último)

git checkout HEAD~2 retorna para o antepenultimo commit

Para voltar para o commit final, o jeito certo é na verdade dar um:

git checkout <nome da branch>

Dessa forma voltamos para o momento atual que estavamos antes do checkout (acredito que do último commit, é importante se certificar de não voltar no tempo sem salvar as alterações).

Para resolver o problema de ter alterado o passado, devem ser digitados os seguintes comandos:

git reset

git clean -df

git checkout -- .

**.gitignore**

O arquivo .gitignore descreve todos os arquivos e pastas que devem ser ignorados pelo git e não serem commitados, isso pode acontecer por n motivos, por exemplo: arquivos de configuração ou de log que variam de acordo com cada máquina; arquivos muito pesados que não devem ser commitados; bibliotecas externas que devem ser baixadas quando outro desenvolvedor for executar na própria máquina, e não mandadas para o github; e assim por diante. Cada tipo de aplicação tem seu próprio gitignore tanto na seleção dele no github quanto em outros locais.

**Como remover arquivos da área de Stage**

Basta usar o comando **git reset**, lembrando que é sempre interessante usar o comando git status antes sempre para conferir.

**Como desfazer mudanças não salvas**

Imagina que você tá fazendo várias alterações no projeto e percebe que fez besteira, se quiser voltar para o último commit, basta fazer os seguintes comandos:

git status

git reset

git clean -df

git checkout -- .

**Como fechar o editor de texto Vim**

Se você fizer algumas coisas tipo merge triplo e commit sem mensagem ele vai abrir o Vim, editor de texto padrão do Linux (Nano sola); para escrever algo no Vim basta teclar **‘i’** e escrever, esse ‘i’ representa ‘insert’. Para salvar e sair do Vim, basta digitar Esc, depois **:wq**, esse wq representa write and quit (escrever e sair) e então Enter. Se você quer sair sem salvar basta digitar **:q!**

**Como apagar um commit mas manter as alterações**

Se você fez um monte de alterações e deu um commit mas percebeu que faltou uma, é possível apagar o commit mas manter as alterações que você fez, para isso basta digitar o seguinte comando:

git reset --soft HEAD~1

Esse comando faz com que resete as alterações para o commit anterior ao que foi feito por último e errado (HEAD) mas de forma que não sejam perdidas as alterações (por isso o –soft).

**Como apagar um ou mais commits de vez**

Imagina que você meteu o louco e quer apagar os últimos 3 commits e também as alterações feitas nesses commits, você pode usar o comando:  
git reset --hard <código do commit onde quer parar>

Esse comando força o repositório a voltar para o commit citado, apagando quaisquer commits depois dele, é uma ação muito perigosa, deve ser feita com cuidado.

**Como atualizar o repositório local em relação ao remoto**

Caso seu repositório local esteja desatualizado em relação ao remoto, basta que você use o comando:

git pull <remoto> <branch>

ou basicamente

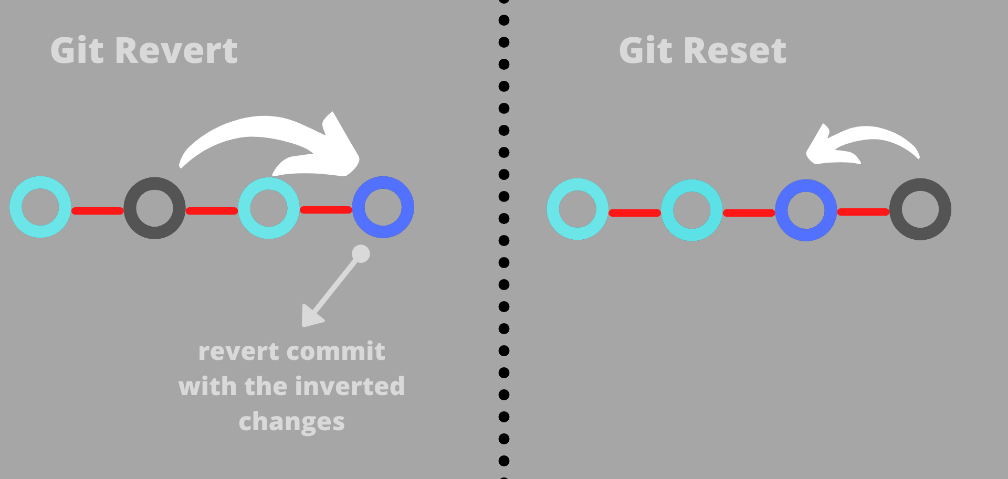
git pull origin main

**Anotações dia 6 – 23/06/24: (curso do Tiago Matos)**

**Formas de restaurar commits anteriores**

**Git** **checkout**: Existem três formas de restaurar commits anteriores, a primeira é o git checkout que já vi antes, ela restaura para um momento anterior apenas para visualizar como estava o código antes, não é para reverter alterações e começar de novo.

**Git revert**: Cria um commit novo na linha do tempo que removeu (deu revert) na alteração de um commit anterior.



**Git reset:** Volta no tempo apagando os commits anteriores, dependendo da flag colocada ele pode manter as alterações ou desfazer elas também, é bem perigoso.