[**Tutorial no notion**](https://awake-andesaurus-fde.notion.site/Introdu-o-Git-GitHub-a3742fbeb70b4e079b2908a926e9e7c4?pvs=4)

**Git**

Podemos utitlizar o comando “git help ‘nome do comando’” para acessar a sua documentação.

**clear** >> limpa o terminal

Config inicial do git, 3 passos:

**git config --global user.name “Matheus Gomes”**

**git config --global user.email matheus.recin@outlook.com**

**git config --global init.default branch main** >> alterar o nome da branch padrão

outros comandos:

**git branch -m master main** >> Renomear a Branch Localmente

**git push -u origin main** >> Atualizar o Branch Upstream no Repositório Remoto

**git push origin --delete master** >> Remover a Branch Antiga do Repositório Remoto

Em um diretório, temos os arquivos untracked e os tracked.

Os arquivos ‘untracked’, não recebem um ‘SNAPSHOT’ que seria o stagio pronto para o commit.

Os arquivos ‘tracked’, estão prontos para receber um commit.

Então vamos ter os arquivos ‘staged’ e ‘unstaged’, onde o status ‘staged’ siginifica que arquivo está ‘tracked’, pronto para o commit.

Sempre que executamos um commit em um ou mais arquivos, o git faz um snapshot, é gerada uma primeira versão.

**git diff** >> mostra a diferença entre as versões não commitadas;

**git diff --staged** >> ver mudanças que estão no indice, mas ainda não commitadas;

**git diff <commit-hash-1> <commit-hash-2> >>** comparar dois commits

o arquivo pode estar em 3 estágios:

working stage

staging stage

commit stage

**git log** >> mostra os logs de operações realizadas

**git log --oneline** >> mostra um resumo dos logs em apenas uma linha

alterar a mensagem do último commit (amend)

**git log --online** >> mostra os logs das ultimas alterações realizadas;

para alterar somente a mensagem do ultimo commit podemos utilizar o comando:

**git commit -m “Nova mensagem do ultimo commit realizdo” --amend**

podemos pular o estágio de staging, saindo do working direto para o commit:

**git commit -am “pular o staging”**

removendo e restaurando arquivos:

**git rm hamburger.jpg** >> exclui

**git restore hamburger.jpg** >> recupera

renomear arquivos:

arquivo1 🡪 arquivo2

**git mv nomeDoArquivo1.jpg nomeDoArquivo2.jpg**

log detalhado, de todas as alterações dos ultimos commits:

**git log -p**

**git reset**

utilizamos o reset para retornar uma versão de determinada linha do tempo ou corrente de histórico de alterações. A HEAD e a MAIN ficam sempre no ultimo commit realizado, que seria o final da corrente ou linha do tempo de alterações. Ao voltar uma versão, movimentamos a head e a maind para um ponto anterior do linha. Temos 3 tipos, SOFT, MIXED e HARD

**soft** >> torna os commits anteriores com o status STAGED, basta fazer o commit;

**mixed** >> torna os commits anteriores com o status UNNSTAGED, necessário fazer o add e o commit;

**hard** >> remove os commits anteriores, não sendo possível a recuperação das alterações realizadas.

Texto preto sobre fundo branco

Descrição gerada automaticamente

**git alias** (atalho de acesso a comandos)

**git config alias.log1 “log --oneline”** >> alteração refletida somente no terminal do proprietario do projeto

**git config --global alias.log1 “log --oneline”** >> alteração refletida para todos os usuários do projeto

o comando acima gera um atalho de “log1” para “log --oneline"

podemos alterar direto no arquivo .gitconfig os alias que desejamos criar:

No Linux e macOS:

O arquivo .gitconfig geralmente está em ~/.gitconfig.

Alternativamente, pode estar em ~/.config/git/config.

No Windows:

O arquivo .gitconfig geralmente está em C:\Users\<seu\_usuario>\.gitconfig.

Branch, semelhante a uma filial, imagem, grupo de processos ou linhas de tempo.

**git branch** >> mostra as branchs criadas, e a branch atual destacada

Um proejto sempre inicia na MAIN e a partir dela, é uma boa prática criar novas branchs para fazer complementos em determinado projeto. Novas branchs criadas a partir da MAIN serão cópias identicas, prontas para serem editadas, complementadas. Depois podemos unir as alterações da branch criada para a branch principal (merge).

Texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média

**git branch NomeDaNovaBranch** >> cria uma nova branch;

**git switch NomeDaNovaBranch** >> altera/seleciona o ambiente de trabalho para a branch descrita;

**git merge -m “Merge da NovaBranch para a Branch Main” NomeDaNovaBranch**

faz o merge(mescla) da branch atual com a descrita após o comentario

**git branch -d NomeDaNovaBranch** >> remove a nova branch criada;

é uma boa prática remover uma branch depois de finalizada uma feature.

**GitHub**

Como criar um token?

O que temos na interface do github?

**git push --all** >> envia a Head/Main atual para o GitHub de todas as branchs

**O que são issues(problemas)?**

Problemas que são abertos no nosso projeto;

No github ao abrirmos uma issue, basta colocar uma descrição e enviar, ela será direcionada ao proprietário ou aos adm do projeto;

são ferramentas para rastrear tarefas, melhorias ou problemas (bugs) em um projeto. Elas são discussões abertas que ajudam a planejar e organizar o trabalho.

**O que são pull-requests?**

permite aos desenvolvedores solicitar que suas alterações em uma branch sejam revisadas e possivelmente integradas (ou "mescladas") em outra branch, geralmente a branch principal do projeto, como a main ou master.

**Como funciona um Pull Request?**

**Criação da Branch:** O desenvolvedor cria uma nova branch a partir da branch principal ou outra branch de trabalho.

**Desenvolvimento:** O desenvolvedor faz as alterações de código necessárias nessa nova branch.

**Criação do Pull Request:** Após finalizar as alterações, o desenvolvedor abre um Pull Request, que é uma solicitação para que essas alterações sejam revisadas e, se aprovadas, mescladas na branch de destino.

**Revisão e Discussão:** Outros desenvolvedores, ou os mantenedores do projeto, revisam o código, deixam comentários, sugerem mudanças ou pedem esclarecimentos. Durante essa etapa, o autor do PR pode fazer ajustes com base no feedback recebido.

**Mesclagem:** Se o Pull Request for aprovado, ele é mesclado na branch de destino. Se houver conflitos, eles precisam ser resolvidos antes da mesclagem.

**Fechamento:** Após a mesclagem ou caso o PR seja rejeitado, ele é fechado.

**Release**

Lançamento de novas versões.

Na plataforma do GitHub podemos criar novas versões:

na aba Code do github no menu lateral da direita, temos a opção de Create a new release;

na pagina que abrir selecionamos a tag como a version v1.0.0;

adicionamos um título e uma descrição e publicamos a nova release.

Sempre que feitas alterações no repositório remoto GitHub, precisamos trazer as alterações para o repositorio local.

**git pull** >> baixa as atualizações do repositório remoto para o local.

VSCode e github

Se foi criado um token de acesso para o Git, e vinculada a conta do github, o VSCode vai reconhecer o git automaticamente e não será necessário nenhuma config adicional...

Settings

Git enable, check box true

Sinceramente é bem intuitivo...