# Git e Github

**Git é um controle de versionamento distribuído e open source.**

No git para configurar existem 3 métodos,

```
--global (escopo global),
```

```
--system (escopo de sistema,
```

```
--local (escopo de algum ambiente)
```

**Configurando nome do usuário git global**

```
Geronimo@DESKTOP-9GNCJVS MINGW64 ~/Documents
$ git config --global user.name "Geronimonetto"
```

**Configurando e-mail global do git**

```
Geronimo@DESKTOP-9GNCJVS MINGW64 ~/Documents
$ git config --global user.email geronimomorais1617@gmail.com
```

**Verificando o nome do usuário**

```
Geronimo@DESKTOP-9GNCJVS MINGW64 ~/Documents
$ git config user.name
Geronimonetto
```

**Verificando o e-mail configurado no git**

```
Geronimo@DESKTOP-9GNCJVS MINGW64 ~/Documents
$ git config user.email
geronimomorais1617@gmail.com
```

**Verificando o nome padrão das branchs criadas**

```
Geronimo@DESKTOP-9GNCJVS MINGW64 ~/Documents
$ git config init.defaultBranch
master
```

**Configurando o nome da branch padrão para main**

```
Geronimo@DESKTOP-9GNCJVS MINGW64 ~/Documents
$ git config init.defaultBranch
master
```

**Configurando a branch padrão criada**

```
Geronimo@DESKTOP-9GNCJVS MINGW64 ~/Documents
$ git config --global init.defaultBranch main
```

**Listando configurações globais**

```
Geronimo@DESKTOP-9GNCJVS MINGW64 ~/Documents
$ git config --global --list
user.email=geronimomorais1617@gmail.com
user.name=Geronimonetto
init.defaultbranch=main
```

# Configurando git localmente

Configurando versionamento Local para armazenar repositório do github.

## Autenticando via Token

**Settings** > **Developer settings** > **Personal acess tokens** > **Tokens(Classic)** > **generate new token(classic)**

O Token vai ser gerado por um tempo e existem as permissões para o repositório, de acordo com o que você selecionar vão ser as opções válidas nesse repositório.

A senha será o token.

## Salvando as credenciais de token

Atualmente uma das formas mais seguras de conexão com o github é utilizando o token do github e a chave ssh.

```
git config credential.helper cache -- salva as configurações de
```

```
git config --global credential.helper store -- salva as credenci
```

```
git config --global --show-origin credential.helper -- mostrando
```

```
cat .git-credentials  -- verificando as credenciais
```

Caso as credenciais do github ja estejam configuradas podemos excluir em > gerenciar credenciais do windows e excluir a credencial do github.

## Configurando github via ssh

Conseguimos encontrar no nosso github a opção de conexão via ssh que também é de forma bastante segura, podemos encontrar em:

**settings > SSH and GPG keys**

## Verificando se há chaves ssh

```
ls -al ~/.ssh  -- verificando se existem chaves ssh
```

***id_rsa.pub***
***id_ecdsa.pub***
***id_ed25519.pub***

Provavelmente as chaves estarão nesse formato

## Gerando nova chave ssh

```
ssh-keygen -t ed25519 -C "geronimomorais1617@gmail.com"
```

**ed25519** é o formato da criptografia, assim como era **rsa** um pouco mais antigo.

## Executando o ssh agent

```
eval "$(ssh-agent -s)"  -- executando o ssh agent
Agent pid 1735
```

Após o comando será iniciado o **ssh agent,** contudo precisaremos adicionar ao nosso agent o caminho da nossa chave ssh.

```
ssh-add ~/.ssh/id_ed25519
```

Feito isso o próximo passo é adicionar a chave pública ao nosso github.

**Sign keys** - são chaves ssh para ver a assinatura dos seus commits

**Authentication keys** - são chaves para autenticação

## Encontrando as chaves privadas e públicas

```
/d/Bootcamp_DIO/Bootcamp_NTT/NTT_DATA/BOOTCAMP_NTT_DATA/BOOTCAMP
$ cd ~/.ssh
```

```
geron@Machine MINGW64 ~/.ssh
$ ls
id_ed25519  id_ed25519.pub  known_hosts  known_hosts.old
```

a chave adicionada ao github é a chave pública.

A partir disso agora podemos clonar e usar via ssh.

# Iniciando e clonando repositório

Para iniciar o repositório local podemos usar o comando init, que é responsável por manter todas as configurações do repositório localmente.

```
git init
```

Contudo podemos também clonar um repositório remoto quando precisarmos utilizar dentro de um projeto.

```
git clone url diretorio
```

```
git clone https://github.com/Geronimonetto/BOOTCAMP_NTT_DATA.gi
```

Dessa forma fazemos o clone do repositório de forma simples e no final passamos o nome do diretório que queremos manter.

o config dentro da pasta .git mostra onde estão nossas origens de repositórios, remoto, qual a branch e outras informações.

cd .git | cat config

## Adicionando repositório remoto ao repositório local

Quando iniciamos um repositório local, devemos adicionar um repositório remoto vinculado a ele, o comando para adicionar esse repo remoto é

```
git remote add nome_aleatorio url
```

**nome_aleatorio** por boas práticas é **origin** e a url é a url do repositório

```
# Adicionando um repositório remoto
git remote add torval https://github.com/Geronimonetto/BOOTCAMP_
```

```
# Verificando quais são os repo remotos
git remote -v
torval  https://github.com/Geronimonetto/BOOTCAMP_NTT_DATA.git (
torval  https://github.com/Geronimonetto/BOOTCAMP_NTT_DATA.git (
```

```
# Removendo repositorio remoto
git remote rm torval
```

## Clonando branch específica

Para clonar uma branch específica dentro do nosso repositório podemos passar uma flag e especificar qual branch queremos baixar para o nosso repositório remoto.

```
git clone url --branch nome_branch --single-branch
```

## Verificando Status

O comando git status verifica o status dos arquivos do diretório, que o git ainda não rastreou

```
git status
```

## Verificando logs

Para saber quais foram os commits feitos no repositório podemos usar o comando git log

```
git log
```

Quando encontrarmos uma pasta com algum arquivo chamado .gitkeep dentro é apenas por convenção para que a pasta não fique vazia.

## Desfazendo alterações

Caso você crie um repositório com git init dentro da pasta incorreta, podemos apenas excluir a pasta .git

Caso você tenha feito modificações em arquivos dentro do seu repositório local e deseja desfazer as modificações feitas podemos usar o comando:

```
git restore arquivo
```

## Modificando o ultimo commit

Caso você tenha enviado um commit de forma incorreta pode ser ajustado.

```
git commit --amend -m "commit teste"
```

Existe outra forma de fazer isso usando o comando git commit

```
git commit --amend
```

O editor será aberto para editar com o vi.

## Retornando para commits anteriores (GIT RESTORE)

Para você restaurar os commits de forma que você queira escolher quais arquivos quer remover e quais pretende adicionar existe a flag soft, os arquivos vão ficar no campo de staged.

```
git reset --soft hashlog
```

Para restaurar o commit de acordo quando os arquivos estavam untracked podemos usar o comando, você vai para o commit antes dos arquivos serem add

```
git reset --mixed hashlog
```

Para restaurar o commit de forma que não venha a ter nada dos commits seguintes, podemos usar o comando

```
git reset --hard hashlog
```

Caso você tenha adicionado um arquivo na área de preparação podemos remover usando 2 formas

```
git reset nomearquivo
git restore --staged nomearquivo
```

## Verificando alterações de commits

Para verificar todos os commits feitos e modificações podemos usar o comando

```
git reflog
```

## Tornando nossa branch main em upstream

```
git push -u origin main
```

O comando acima indica que queremos enviar nossas modificações para origin(caminho do nosso repo remoto) main (branch main), e a flag -u (upstream) indica que queremos que nossa principal seja a main.

# Atualizando repositório local com o repositório remoto

Para atualizar nosso repositório local podemos usar o comando

```
git pull
```

# Criando branch

A branch será criada de acordo com o commit que a branch atual estiver

```
git checkout -b nomebranch
```

## Alterando de branch

Para modificar de branch devemos usar o comando git branch e selecionar qual branch queremos ir inserindo o nome.

```
git checkout nome-branch
```

## Verificando último commit de  cada branch

com esse comando conseguimos verificar os commits em que as branchs estão apontando

```
git branch -v
```

## Juntando branchs

Para mergear uma branch com a outra, precisamos entrar na branch que queremos receber o merge e usamos o comando

```
git merge nome-branch
```

com isso a branch que está sendo mergeada será adicionada a branch que você se encontra

Podemos também mergear com o conteúdo da branch remota

```
git merge origin/mian
```

## Excluindo branch

Para excluir uma branch é um pouco mais fácil, podemos usar a flag -d

```
git branch -d nome-branch
```

## Baixando conteúdo da branch remota sem mergear com a branch local

Baixando alterações da branch remota

```
git fetch origin main
```

## Verificando diferença entre branchs

Para verificar a diferença entre branchs podemos usar o comando

```
git diff branch (main) branch/remota (origin/main)
```

## Arquivando modificação em branch

```
git stash  -- arquiva uma modificação
```

```
git stash apply -- Aplica a modificação na branch e mantém no st
```

```
git stash pop  -- Aplica a modificação na branch e excluir do st
```

## [BÔNUS]

Existe a possibilidade de rodar um vs code no github, é um editor muito parecido com o github, basta entrar no repositório e apertar .(ponto)