GIT e GIT-HUB

2022

# SUMÁRIO:

# COMANDOS BÁSICOS TERMINAL

## Comandos básicos em Windows do Prompt de comando:

### dir = listar todas as pastas do diretório atual.

* dir –a mostra arquivos ocultos

### cd = Navegar entre as pastas (change directory)

* cd / Leva você para a pasta c:
* cd nome da pasta – Leva para essa pasta, verifique no dir
* cd .. retrocede um nível

### cls = Limpar a tela do prompt de comando

### mkdir nome da pasta = Cria uma pasta no diretório atual.

### echo palavra = printa no terminal um texto.

* echo palavra > arquivo.extensão = o terminal checa se existe o arquivo, caso não, cria e coloca o texto dentro.

### del nome pasta = deleta tudo que tem dentro da pasta

### rmdir nome pasta = deleta a própria pasta

### move nomearquivo.txt ./pasta = move o arquivo para uma pasta

## Comandos básicos em Linux :

### ls = listar todas as pastas do diretório atual.

### cd = Navegar entre as pastas (change directory)

* cd / Leva você para a pasta parte inicial
* cd nome da pasta – Leva para essa pasta, verifique no dir
* cd .. retrocede um nível

### clear = Limpa a tela do prompt de comando

### mkdir nome da pasta = Cria uma pasta no diretório atual.

### echo palavra = printa no terminal um texto.

* echo palavra > arquivo.extensão = o terminal checa se existe o arquivo, caso não, cria e coloca o texto dentro.

# GIT – INFORMAÇÃO

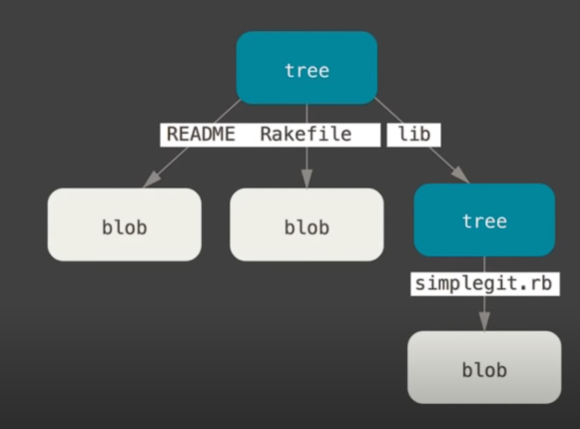
## Atalhos de usabilidade:

### Botão direito na pasta e usar o **Use git bash here** é uma forma de abrir o arquivo no próprio git.

### Mover arquivo dentro do terminal bash:

Mv nomearquivo.extensao ./nomedapasta

## Objetos internos do GIT:



### BLOBS

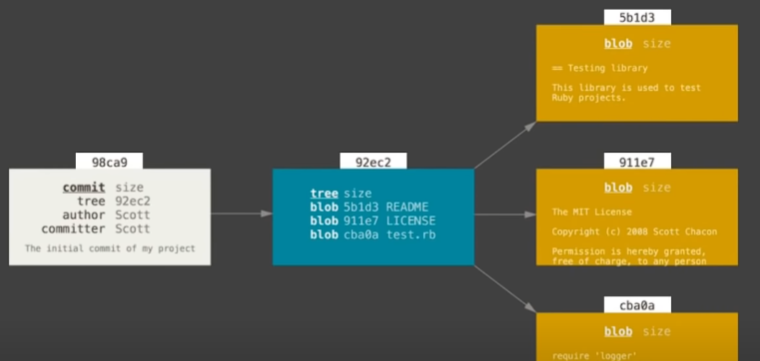
A maneira como o git manipula com esses objetos faz com que eles fiquem dentro de um objeto blob, esse blob contem metadados e esses metadados tem a quantidade de string dentro dele, uma barra e um zero e de fato, o conteúdo.

### TREES

As trees armazenam os blobs, contem metadados assim como os anteriores, além disso, montam a estrutura do arquivo e contem o nome do arquivo o que o anterior não tem.

### COMMITS

Objeto que junta tudo e como os demais, é criptografado pelo SHA1.



Isso mostra que o git é um sistema distribuído e muito seguro, qualquer alteração no arquivo faz uma mudança na SHA1.

## Chaves SSH

É uma forma segura de estabelecer uma conexão segura entre duas maquinas por meio de encriptação.

Dessa forma, a chave SSH permite que o github entenda que seu computador seja confiável.

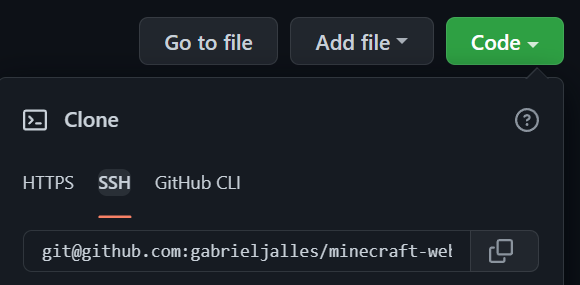
### Comandos de criação da chave SSH

No gitbash:

* ssh-keygen –t ed25519 --C email (enter – digite a senha –novamente#)
* Vá para a pasta criada do ssh ,
* cat id\_ed25519.pub
* Copie a chave e cole dentro do github.
* Para uma entidade ficar responsável por pegar as chaves digite o comando:
  + eval $(ssh-agent –s)
* Como o agente foi gerado, passe a chave para ele , mas agora não a pub, mas sim a privada :
  + ssh-add id\_ed25519

### Quando for clonar uma pasta do github, use o caminho SSH e não os demais.

* Exemplo: git clone (caminho ssh)



## Comandos para criação de Autor e email

### git config - -global user.email “[gabriel.jalleswork@hotmail.com](mailto:gabriel.jalleswork@hotmail.com)”

### git config - -global user.name “gabrieljalles”

### Para saber o email linkado:

git config - - list

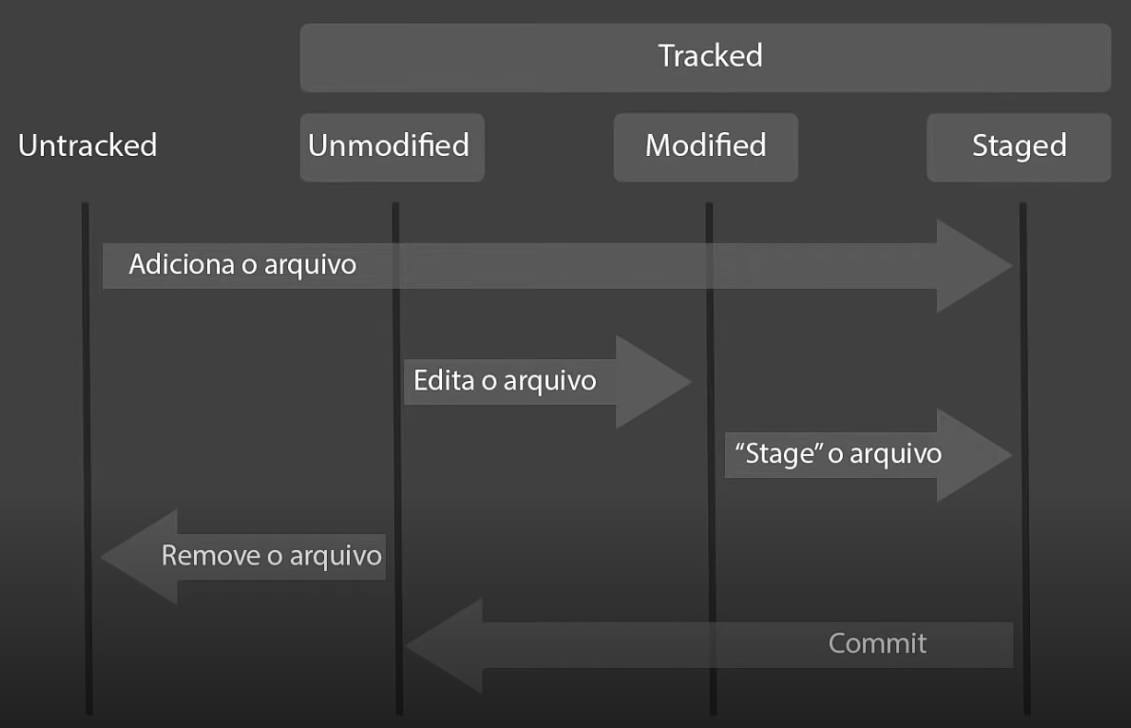
### Para desvincular email:

git config - - global - - unset user.email

gir config - - global - -unset user.name

# GIT – PRÁTICO

## Primeiros comandos com git



staged = se preparando para entrar no palco , ser commitado.

Quando usamos o **git add** os arquivos untracked passam a ficar no staged

Vamos dizer que você tenha um arquivo não modificado e tracked, quando você faz uma modificação ele passa pro modified.

Quando o arquivo, agora modified, passa por git add ele vai para staged.

Se um arquivo que era unmodified, é removido ele passa a ser untracked novamente. O git não olha mais para ele. **git ignore**

Quando o arquivo sofre um commit, ele sai do staged para o unmodified.

### git init

Cria um repositório no git dentro da pasta principal do projeto.

### git add

Serve para dizer ao git para ele passar a ver o arquivo

git add nomearquivo

git add \* (adiciona tudo)

### git commit –m “descrição breve da modificação”

### git status

Mostra o status atual do arquivo na master.

### git restore nomedoarquivo.extensao

tira ele do staged não será mais commitado

## Comandos para enviar para o GITHUB

### git remote add origin nome do site dado pelo repositório criado

Usado para adicionar o repositório da nuvem na aplicação.

Usado só no inicio, para a primeira vez que vai enviar o arquivo.

### Git remote –v

Usado para ver os nomes de git adicionados.

### Git push origin main

Mandando a nossa aplicação para a nuvem

## Comandos para receber o arquivo do GITHUB

### Clonando um repositório do github

Git clone site do repositório

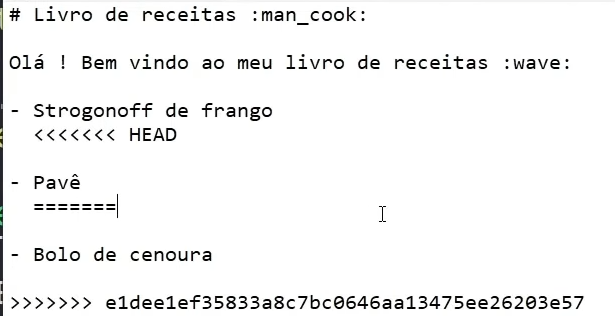
Não precisa mais iniciar o git, ele já vem como um repositório completo para trabalhar.

### Consertando erro :

Caso você esteja trabalhando em um código, e outra pessoa também, ela envia pro git e depois de algum tempo você também envia, irá aparecer um erro. Esse erro pede para que você pegue a versão dela primeiro antes de acrescentar a sua.

Git pull origin master

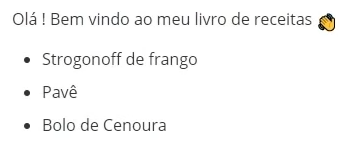
Algumas vezes pode aconter o conflito de merge, duas linhas modificadas por pessoas diferentes. Para resolver isso, o git pede para que você manualmente faça a modificação abrindo o arquivo manualmente.



O git acrescentas <<<<HEAD para mostrar que a atualização mais recente é a sua (PAVE)

O que foi pego do github foi depois de =============

Você tem a liberdade de pegar e modificar você mesmo, depois salvar.



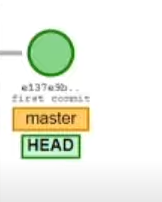
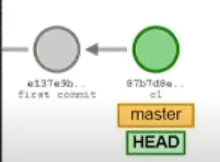
# BRANCHES

Branch é uma bifurcação do versionamento em que você passa a atuar em uma nova linha do tempo, salvando o código na nuvem até que ele esteja pronto para ser colocado na main.

## HEAD

Head ele mostra onde você está nas branchs, cada commit ele muda. Mostra o ponto de modificação mais recente.

Aqui ele usa master mas é main:

 (após commit) (veja que o head vai pro ponto mais recente)

## Diferença entre branch Main e master

Branch master está ficando ultrapassado, as pessoas usam mais Main como branch principal.

## Como criar um bifurcação no código (branches) :

Normalmente as pessoas criam uma nova branch quando querem acrescentar uma nova funcionalidade na aplicação.

### Criando branch -> git checkout –b nomebranch

Um comando que não só movimenta para a branch, mas também cria ela (-b) , ou seja o HEAD passa para lá.

### Movendo entre branchs ->it checkout nomebranch

um comando usado para se movimentar pelas branchs

## Como enviar para a nova branch

### Git push origin nomedabranchcriada

## Como juntar branch main com a branch criada

Após você criar todas as funcionalidades na nova branch e achar que deve ser implementado a branch main, primeiro volte para a branch main.

Git checkout main

Git merge nomedabranchcriada

– juntar a branch criada com a que você está atualmente

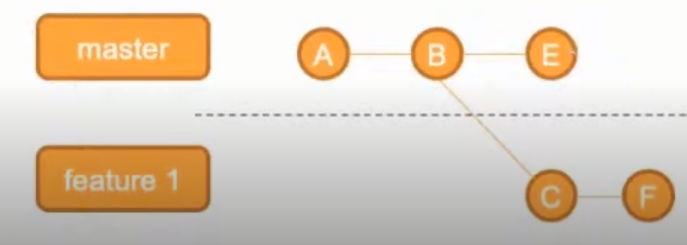
Agora, se der git status verá que :



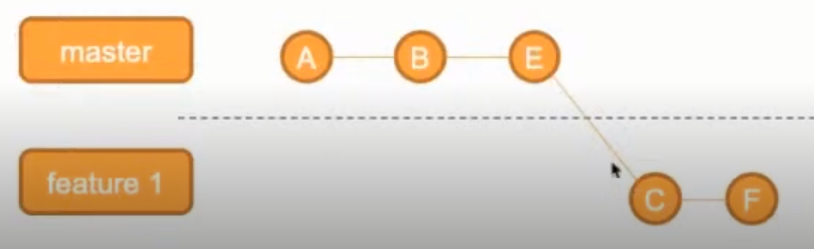
A sua branch está a frente em 1 commit da branch main no site, o seu main atual está no futuro e o main do site no passado. Precisamos mandar para o site agora. **git push origin main**. Agora seu código já está atualizado para continuar ou criar novas branchs.

## Usando RABASE para atualizar o branch atual com a main

Sem usar rabase



Usando rebase



Para executar:

git checkout feature1

git rebase main

## Como saber em qual branch estou e quais existem no repositório local

Usado para caso não esteja no gitbash

### git branch

## Como mudar o nome da branch atual

### git branch –m novo nome para branch

## Como mudar o nome da branch de qualquer outra branch

### git branch –m nomedabranch nomenovabranch

## Como deletar uma branch

### git branch –d nomebranch

## Coisas muitos importantes em uma branch

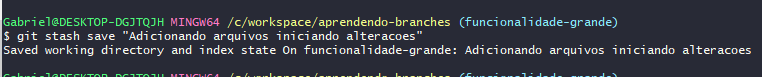
Quando você passa para uma branch, caso esteja fazendo algo em outra branch mas você não realizou o commit antes. Os aquivos que estão sendo trabalhados passam para a nova branch quando você faz a movimentação.

### STASH

O stash é usado como se fosse uma caixa para guardar alterações, ele conserta o problema anterior.

### git stash save “coloque o contexto”

Nesse momento, você está guardando as alterações feitas em uma caixa, dentro do branch atual. Agora você pode se mover tranquilamente, sem que os arquivos andem com você.



### git stash list = usado para ver quantas stash você possui e quais são



Vamos dizer que agora eu queira usar novamente, os stash guardados basta:

### Git stash pop 1 = o índice nesse caso é o 1

### git stash clear = usado para limpar o stash

# GIT – HISTÓRICO

## Git log

Conhecendo uma forma de ver todo o histórico do repositório:

Caso o histórico seja muito grande, use a tecla Q para sair da visualização.

### git log pasta = Você vê o log de uma pasta específica

### git log - - oneline = mostra tudo em apenas uma linha

### git log - - graph = Usado para mostrar uma maneira mais ilustrado

### gitk = abre aplicativo que mostra detalhado todo histórico

# REVERTER COMMIT

## Git reset

### git reset sha1 = você volta o código para aquele momento

Todos os arquivos commitados depois daquele momento voltam pra área de unstaged.

### git reset HEAD~1 = Olha onde a tag HEAD está e mova 1 para trás.

### git reset - -soft = Você desfaz o commit , depois do add

volta para o momento depois do git add \*

outro exemplo : git reset - -soft HEAD~1

### git reset - -mixed = Você desfaz o commit , antes do add

O arquivo volta a ser unstaged

### git reset - -hard = Apaga tudo, destrói o último cód criado

## Git revert (NÃO ENTENDI)

### git revert shkjas1 = Revertendo referenciando o SHA1

### git revert HEAD~1 = Revertendo referenciando o Head

# ESTRUTURANDO COMMITS

A melhor forma de usar o commit é pelo **git commit**

Ele abre o vin, um editor de código commit que te ajuda, mas é difícil de mexer.

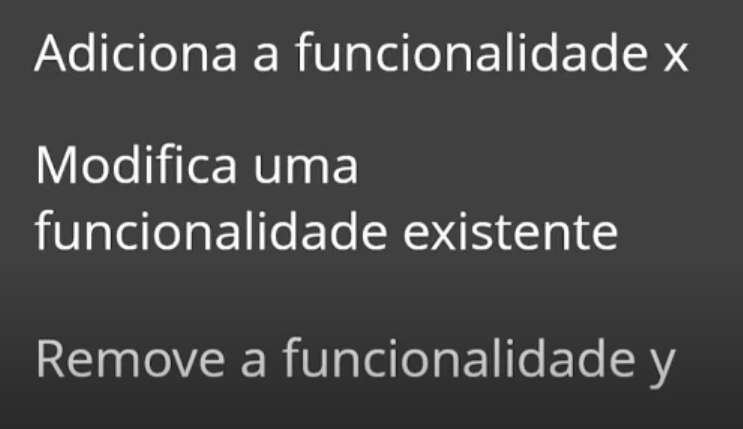
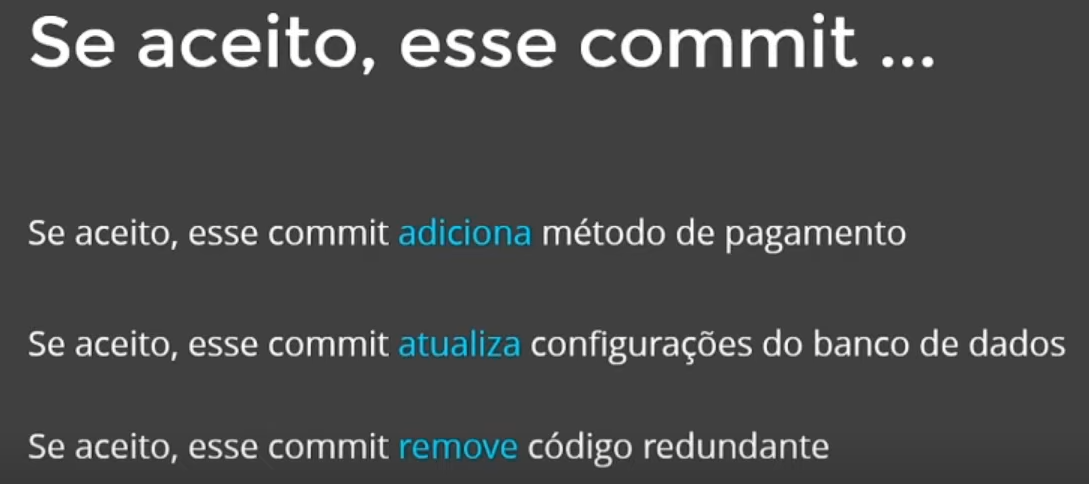
Insert passa a escrever no text

Closes #1 você referencia a issue com o commit, mostrando que o problema foi resolvido

Dois pontos wq fecha e salva executando o commit

### CAPA

É muito importante que comece com letra maiúscula, até 50 caracteres, não terminar com ponto escrito de forma imperativa. A forma imperativa é fazer a utilização do presente mandando uma ordem.

### CORPO

Deve-se adicionar detalhes ao commit, a cada 75 caracteres quebre a linha , explique tudo. Sempre crie o código pensando que a pessoa tem 0 de conhecimento.

### RODAPÉ

Opcional,

## Commits atômicos

Idealmente o seu commit precisa ser uma unidade, início meio e fim. Caso exista testes no código, é bom colocar nesse commit também.

## Semantic Versioning



Toda vez que fazemos uma modificação que quebra a compatibilidade, é uma **major**

Uma **minor** alteration é reservado para implementação podendo ser até grande, mas não quebra compatibilidade.

Uma patch são pequenas alterações, correções de bugs.

# CONTRIBUINDO PARA OUTROS REPOSITÓRIOS

## Pull request

Mover um repositório de uma pessoa para o seu use o fork na parte principal para copiar para sua conta, após isso, pode clonar e salvar quando necessário no seu perfil.

Após enviar para seu repositório, você pode solicitar o pull request para mostrar ao repositório do dono, ele irá analisar e dirá se você fez um pull request benefíco para o site e adicionar ele.

## Permissões

No site, setings -> manage access -> adicione a pessoa para que ela seja um colaborador no seu código.

Agora, a pessoa pode fazer pull sem ter que pedir todas as vezes, se torna uma pessoa de confiança para mudar o código.

# APELIDOS PARA O CÓDIGO

## Criando apelidos para o código

git config - - global alias.comando funçãochamada

exe: git config –global alias.s status -> agora, chamando : git s = git status

## Apagando o apelido do código

git config - - global - - unset alias.s

# GIT LAB

Gerenciamento de repositórios

Organização de repositórios em grupos

## Versionamento de código – controle de versões

O GIT é um sistema de versionamento distribuído, cada cópia possui todas as versões do código. (DVCS – Distributed version control system )