

Professor: Otávio Reis

Por Quê estou aprendendo a tecnologia e como será relevante na minha carreira?

Criado por **Linus Torvalds**

Git é um sistema de controle de versão de arquivos de código distribuído, também conhecido pela sigla VCS – *version control system*.

Github = Um servidor remoto para armazenamento de código.

Git e ao GitHub são tecnologia independente mas complementa um a outro

Os VCS são uma ótima forma de otimizar o trabalho ao analisar as alterações feitas em um código de projeto compartilhado, além de ser um componente fundamental do sistema de gerenciamento de configuração de software que cuida das mudanças que precisam ser feitas em um projeto.

As alterações/revisões/ atualizações feitas são identificáveis através de códigos de alfanumérico (Código de criptografia SHA1). Informações como data e a identidade do autor da alteração também são mantidas.

1. o GIT segue uma abordagem **peer to peer**, contrário de outros como o Subversion (SVN) que segue um modelo baseado em **cliente-servidor**.
2. GIT permite aos desenvolvedores ter uma infinidade de ramos de código completamente independente. Criação, exclusão e fusão desses ramos é simples e não leva tempo.

3. No GIT, todas as operações são atômicas. Isso significa que uma ação pode ter sucesso ou falhar (sem fazer nenhuma alteração). Isso é importante porque em alguns sistemas de controle de versão (como o CVS) onde as operações não são atômicas, se uma operação de repositório é suspensa, ela pode deixar o repositório em um estado instável.
4. No GIT, tudo é armazenado dentro da pasta **.git** . Isso não é o mesmo em outros VCS como SVN e CVS onde os metadados de arquivos são armazenados em pastas ocultas (por exemplo, .cvs, .svn, etc.)
5. GIT usa um modelo de dados que ajuda a garantir a integridade criptográfica de qualquer coisa presente dentro de um repositório.

Cada vez que um arquivo é adicionado ou **um commit** é feito, suas somas de verificação são geradas. Da mesma forma, eles são recuperados através de suas somas de verificação também, analisando código de criptografia SHA1.

6. Outra característica presente no GIT é sua **área de teste** ou **índice**. Na área de preparação, os desenvolvedores podem formatar *commits* e receber feedback antes de aplicá-los.

GIT pode ser instalado tanto no Windows, linux, Mac os.

Aqui iremos aprender instalar no sistema Windows:

1. Instalar o GIT no Windows é tão simples como baixar um instalador e executá-lo. Execute os seguintes passos para instalar o GIT no Windows:
2. Acesse [o site oficial](#) e faça o download do instalador do **GIT para Windows**.

3. Depois de baixado, clique duas vezes no arquivo para iniciar o assistente de instalação. Basta seguir as instruções na tela, clicando em **Next**. Ao término, clique em **Finish** para concluir com êxito a instalação.



4.

Logo no início da instalação, recomendo observar se opção , (GIT BASH HERE e GIT GUI HERE) esteja assinalado , depois e só seguir com next.



Com botão Windows + R, Digite (CMD) e Abra o prompt de comando e digite os seguintes comandos no terminal:

```
git config --global user.name "SEU NOME"  
git config --global  
user.email"exemplo@seuemail.com.br"
```

Nota: Lembre de substituir SEU NOME e exemplo@seuemail.com.br com seus dados. Qualquer commit criado posteriormente será associado à esses dados.

Recomendo antes de usar GIT, já ter conhecimento básico em Navegação via command line interface e instalação, ou seja pesquisar, criar e excluir arquivos via comando no CMD.

Navegação via command line interface e instalação

Comandos Windows	Comandos Unix
cd / (change Director ou navegar entre as pasta)	cd
dir (Lista de Diretório da pasta)	ls
cd.. (Voltar pasta)	cd..
Mkdir + nome (comando p/ criar pasta)	mkdir
del /rmdir (deletar só arquivos) / (remove diretorio)	rm-rf
Cls (clear cleaner ou limpar tela)	clear ou ctrl L
Atalho TAB (auto completa nome da pasta)	
Silence on sucesso: termo que, se não dizer nada no sistema está tudo certo.	
Echo (simplesmente printa de volta a frase que escreveu)	
> (redirecionador de fluxo)	
-a (mostra arquivos ocultos)	

Link de tutorial:

Recomendo ver as aulas disponível no curso

<https://web.digitalinnovation.one>

Ou link abaixo:

[\(75\) MS-DOS - Vídeo Aula com os principais comandos -
www.professorramos.com - YouTube](#)

[\(22\) Prompt de Comando // CMD - YouTube](#)

Entendendo como o Git funciona por baixo dos panos

-O que é SHA1 (Algoritmo de HASH seguro) é um conjunto de hash criptográficas projetados pela NSA (Agencia de segurança Nacional EUA)

As alterações/revisões/ atualizações feitas, são identificáveis através desse códigos de 40 alfanumérico Criptografado.

Vamos ao exemplo:

Crie uma pasta em área de trabalho (Desk top):

Nomeia como: **exp: workspace**

Crie um bloco de nota, e nomeia como : tets1

Escolher arquivo que deseja criptografar com sha1:

(nesse exemplo arquivo test1.txt)

- Abra o programa git bash :
- Digite comando: **"Dir"** para encontrar diretório do seu arquivo :
- Digite comando: **"CD (nome do arquivo)"**, para navegar até o arquivo

Exp: cd desktop

- Digite comando: **"ls"** para visualizar o que tem de arquivos na pasta desktop
- Ao encontrar pasta workspace criado anteriormente, repita comando

"CD " para acessar essa pasta.

Exp: **cd workspace** (obs: comando, será inserido a partir desta pasta)

- Repita comando: **"ls"** para visualizar o que tem na pasta desktop

Digite comando: **"openssl sha1, nome do arquivo com extensão".**

Exp: **openssl sha1 test1.txt**

Pronto, você consegue visualizar o identificador do arquivo (SHA1) com 40 alfanumérico Criptografado.

A cada alteração que efetuar e salvar o arquivo, irá gerar um novo SHA1.

Exp: Dentro do arquivo de texto, se no conteúdo estiver escrito "ola, seja bem vindo" e salvar , °1 código é gerado.

se alterar conteúdo, estiver escrito "bem vindo! " e salvar , °2 código é gerado.

Se arquivo voltar ser = ao anterior "ola, seja bem vindo", então git irá reconhecer o °1 código novamente.

atenção: mais a frente terá prática do exemplo para entender melhor...

-Quais são os objetos interno fundamentais?

Blobs (Os arquivos fica armazenado dentro desse objeto Blobs. Armazena meta dados do git como tamanho do strings, tipo de

objeto, \0 e conteúdo escrito "obs: Blobs não armazena os nomes").

Esse metadados armazenados, podem gerar código sha1 diferente do código gerados por Blobs.

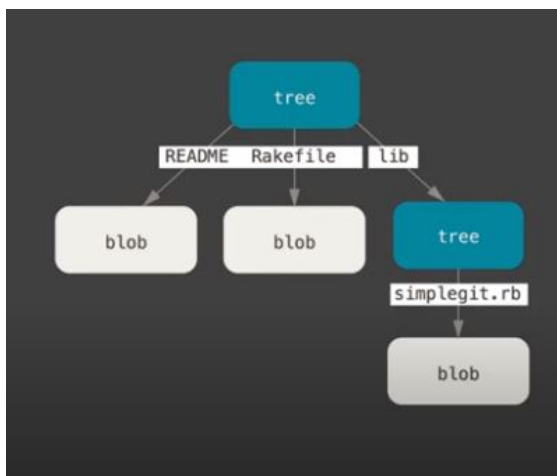
Exp: Um código sha1 para arquivo, outro código para Blobs, strings e conteúdo interno .

Continue lendo que longo em breve, fica mais claro a explicação



Tree (Armazena e aponta para tipos Blobs diferente "que por sua vez também tem sha1", e também armazena o nome desse arquivo.

Ela é uma árvore responsável em montar toda estrutura de onde está os arquivos que podem também apontar para outras árvores)



Commits

(É objeto mais importante de todos, objeto que vai ter sentido nas alterações feitas.)

Commit aponta para > arvore(Tree),

Aponta para > parente(último alteração feito no commit)

Aponta também para > Autor e > Menssagem, então é onde irá fazer sentido para quem está sendo apontado.

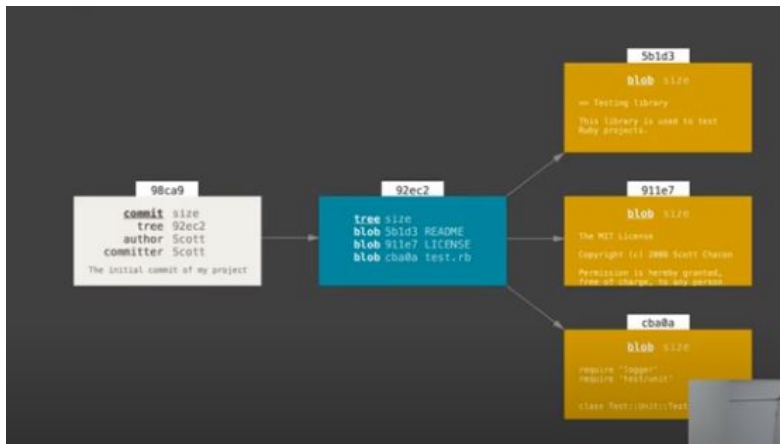
Timestamp = a data e horário criado



Resumindo, uma vez que altera o arquivo, você irá alterar toda estrutura de codificação sha1 daquele **Commit**, ele é único para cada autor que realiza alterações.. Ele cria uma time line de commit, que por sua vez é a forma mais seguro de se trabalhar e compartilhar arquivos.

O que é commit?

Basicamente seria um envelopamento de todos arquivos que contem no projeto ou pasta. Um **commit** é o ato de envelopar e enviar, ou seja, fazer um upload dos códigos para um banco de dados.



Primeiros comandos com Git:

Vamos deixar já instalado bloco de nota typora: você pode utilizar bloco da sua preferência mas nesse exemplo vamos usar typora
Baixar > [Typora — a markdown editor, markdown reader.](#)

Git init (comando para poder iniciar e criar um repositório)

Git add (comando para poder mover arquivos e adicionar no repositório)

Git commit (comando para poder criar o primeiro commit)

Commit = "Tipo Envelopamento de arquivos"

Vamos mover a pasta workspace da área de trabalho para >
Diretorio C:\

Exp: C:\workspace

- Com botão direito clicando em qualquer área branca da pasta c:
abra opção git bash.
- Digite **ls** para listar pasta. Encontre a sua pasta workspace
- Digite **cd** para entrar na pasta

```
MINGW64:/c/workspace

Samsung-w10@DESKTOP-VEC658C MINGW64 /c
$ ls
'$Recycle.Bin'/' PerfLogs/' Windows/
'Arquivos de Programas'@ 'Program Files'/' bootTel.dat
'Arquivos de Programas RFB'/' 'Program Files (x86)'/ found.000/
Config.Msi/ ProgramData/ hiberfil.sys
'Documents and Settings'@ Recovery/ pagefile.sys
Intel/ 'System Volume Information'/' swapfile.sys
MSOCache/ Tryd5/ workspace/
Microsoft/ Users/
OneDriveTemp/ VTRoot/

Samsung-w10@DESKTOP-VEC658C MINGW64 /c
$ cd workspace

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace
$ |
```

- Limpe a tela digitando **clear**
- Digite **mkdir** para criar uma pasta.
Exemplo: **mkdir livro-receita** . Pode conferir que existe uma nova pasta.
- Digite **cd livro-receita** para entrar na pasta.
- De o comando **git init** .

Podem verificar que ainda não tem nada nesta pasta com comando **ls**, ou indo direto na pasta manualmente.

Mas se analisar direito, dentro dessa pasta

C:/workspace/livro-receita/.git/

Tem uma pasta gerencial do **.git** em formato oculto

```
MINGW64:/c/workspace/livro-receita

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace
$ mkdir livro-receita

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace
$ ls
livro-receita/

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace
$ cd livro-receita/

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita
$ git init
Initialized empty Git repository in C:/workspace/livro-receita/.git/

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ |
```

- Para visualizar podem dar comando **ls -a** , ou visitando a pasta do livro-receita > lá em cima na opção exibir > assinar e habilitar Itens ocultos
(na aba mostrar / ocultar)

```
MINGW64:/c/workspace/livro-receita

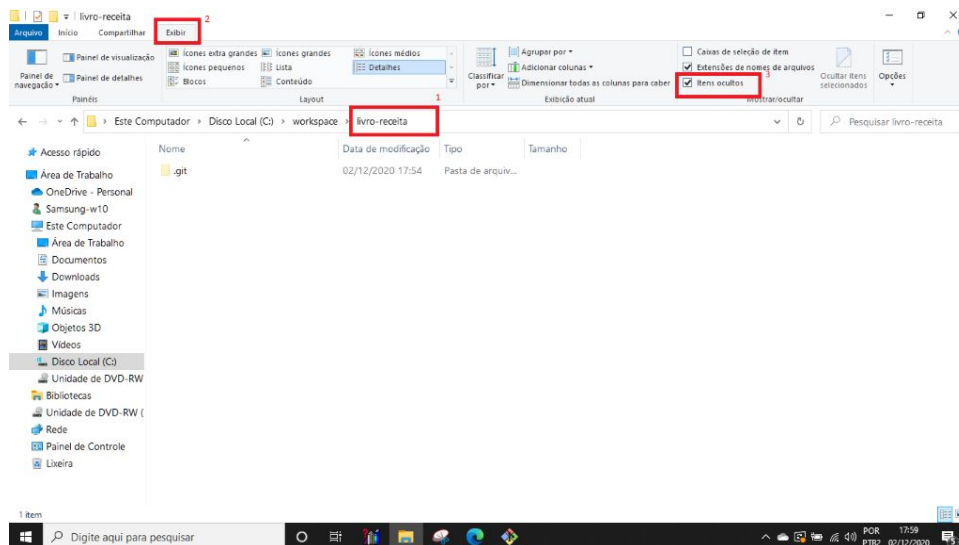
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ cd livro-receita/
bash: cd: livro-receita/: No such file or directory

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ git init
Reinitialized existing Git repository in C:/workspace/livro-receita/.git/

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ ls

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ ls -a
./ ../ .git/

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ |
```



Dentro desta pasta .git tem uma pasta **objects**, do assunto que comentamos onde todas alterações com o passar do tempo será armazenado nesta pasta.

Agora vamos criar primeiro arquivo dentro desta pasta livro-receita: Nesse exemplo vamos usar um arquivo no formato Markdown com extensão .md, em vez do tipo .txt que é bloco tradicional.

- **Markdown** é uma forma mais humana para escrever mesmo sem entender html, é uma linguagem de marcação leve que você pode usar para adicionar elementos de formatação a documentos de texto.

O html é nada mais que um esqueleto de qualquer página da web.



No html para escrever com tamanho de fonte diferente é inserido tags como h1,h2,h3...

E no markdown e só acrescentar # .##. ###

- Criando arquivo **strogonoff.md**

Já com typora instalado, na pasta livro-receita aperte com botão direita em qualquer área branca:

> em seguida, Novo

> Documento de texto

> crie um arquivo de nota com nome "**strogonoff.md**"

Aperte com botão direita clicando novamente sob arquivo "strogonoff.md"

> Abrir com

> E marque a opção : sempre usar esse aplicativo para abrir extensão .md

Agora abra o arquivo e digite: # strogonoff de frango :chicken:

> e de ENTER.

Veja como ele já foi formatado.

Caso queira conhecer mais funções do markdown, então abra aba ajuda > markdown reference, será listado diversos comandos.

Abaixo do título, copie e cole qualquer receita de strogonoff.

Salve e feche o arquivo.

Digitando comando **git add ***

Caso tenha fechado as janelas, abra novamente o git bash e vá até a pasta livro-receitas:

```
MINGW64:/c/workspace/livro-receita
Samsung-w10@DESKTOP-VEC658C MINGW64 /c
$ ls
'$Recycle.Bin'/' PerfLogs/ Windows/
'Arquivos de Programas'/' Program Files'/' bootTel.dat
'Arquivos de Programas RFB'/' Program Files (x86)'/' found.000/
Config.Msi/ ProgramData/ hiberfil.sys
'Documents and Settings'/' Recovery/ pagefile.sys
Intel/ 'System Volume Information'/' swapfile.sys
MSOCache/ Tryd5/ workspace/
Microsoft/ Users/
OneDriveTemp/ VTRoot/

Samsung-w10@DESKTOP-VEC658C MINGW64 /c
$ cd workspace/

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace
$ ls
livro-receita/

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace
$ cd livro-receita/

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ |
```

Digite comando:

git add *

git commit -m "commit msg inicial"

(Seria um breve resumo para identificar esse código)

ATENÇÃO! : em caso aparecer seguinte msg : please tell me who are you

```
MINGW64:/c/workspace/livro-receita
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ git add *

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ git commit -m "commit msg inicial"
Author identity unknown

*** Please tell me who you are.

Run

$ git config --global user.email "you@example.com"
$ git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'Samsung-w10@DESKTOP-VEC658C.(no
ne)')

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ |
```

Siga as instruções da tela e repita os procedimentos anteriores:


```
MINGW64:/c/workspace/livro-receita
to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'Samsung-w10@DESKTOP-VEC658C.(no
ne)')

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ git config --global user.email "[redacted].com"

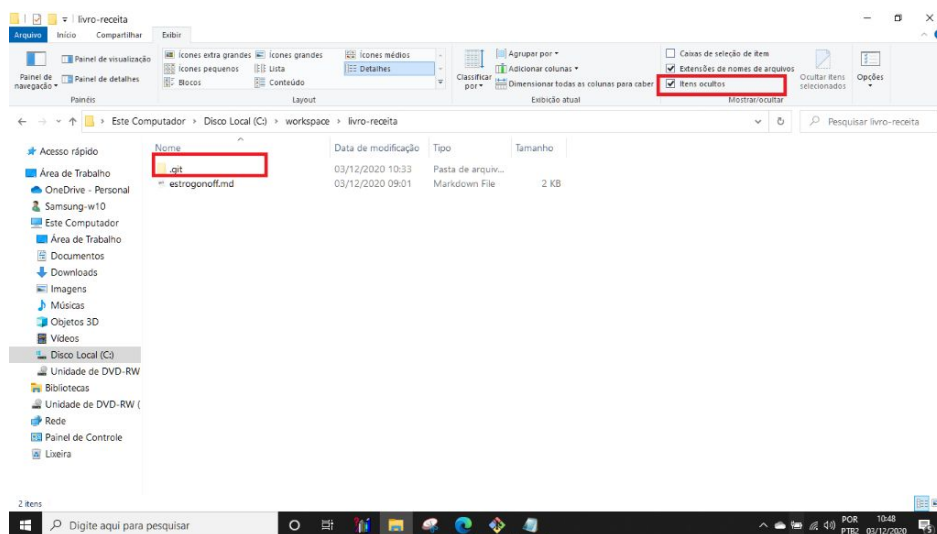
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ git config --global user.name "[redacted]"

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ git add *

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ git commit -m "commit msg inicial"
[master (root-commit) c860cce] commit msg inicial
1 file changed, 26 insertions(+)
create mode 100644 estrogonoff.md

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$
```

! OPS !: Eu tentei mas não deu certo... e agora?
Então, nesse caso é necessário habilitar e abrir a pasta .git que está oculto



Abra pasta .git >
abra o arquivo de **config** em bloco de nota. > Editar arquivo config.
Verifique se a seção existe.
> Se a seção do usuário estiver faltando, adicione-a. [user]
Registre seu nome e seu email.

Exp:

```
[user]
```

```
name = "your name"
```

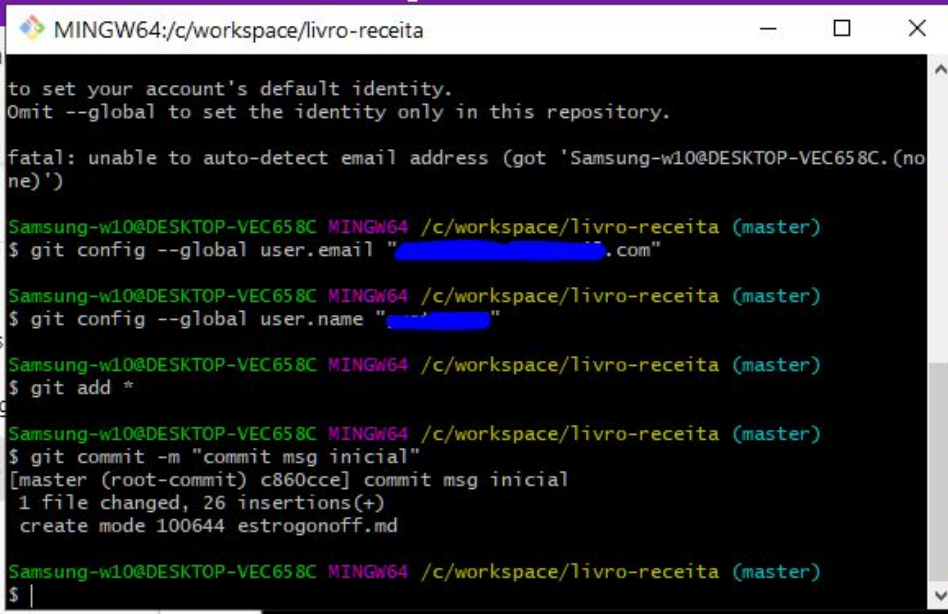
```
email = "your email"
```

Repita novamente no gitbash comando:

git add *

git commit -m "commit msg inicial"

Parabéns, geramos o primeiro commit :

A screenshot of a terminal window titled 'MINGW64:/c/workspace/livro-receita'. The terminal shows the following commands and output:

```
to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'Samsung-w10@DESKTOP-VEC658C.(no
ne)')

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ git config --global user.email " [REDACTED]@gmail.com"

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ git config --global user.name "[REDACTED]"

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ git add *

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$ git commit -m "commit msg inicial"
[master (root-commit) c860cce] commit msg inicial
1 file changed, 26 insertions(+)
create mode 100644 estrogonoff.md

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/livro-receita (master)
$
```

Ciclo de vida dos arquivos no Git:

Teoria por trás da prática do código anterior para entender o que está acontecendo no Git, quais são os estados desses arquivos, como git identifica, se o arquivo foi alterado ou não.

Comando `Git init` como já mencionado, tem capacidade em iniciar e criar um repositório dentro da pasta.

Entendendo melhor conceito de `Tracked` ou `Untracked`:

`Tracked`: Pode se dividir em três estágios diferentes.

Un modified = (Arquivos que ainda não foi modificado)

Modified = (Arquivos que sofreu modificações)

Staged = (Traduzindo seria encenar ou preparar para exibir, também pode se entender como preparar para outro tipo de agrupamento)

`Untracked`: São arquivos que ainda não temos Consciência

Quando inserimos comando `git init` na prática, o arquivo `estrogonof.md` era desconhecido pelo Git. Pertencia ao primeiro estágio `Untracked`

Após inserir outro comando `git add *`, o arquivo foi movido direto para estágio **Staged**. Aguardando para fazer parte de um outro agrupamento.

Os arquivos **unmodified**, são os arquivos que está dentro do repositório git mas ainda não sofreu modificações . Após editar o arquivo, ele passa pertencer ao estágio **modified**.

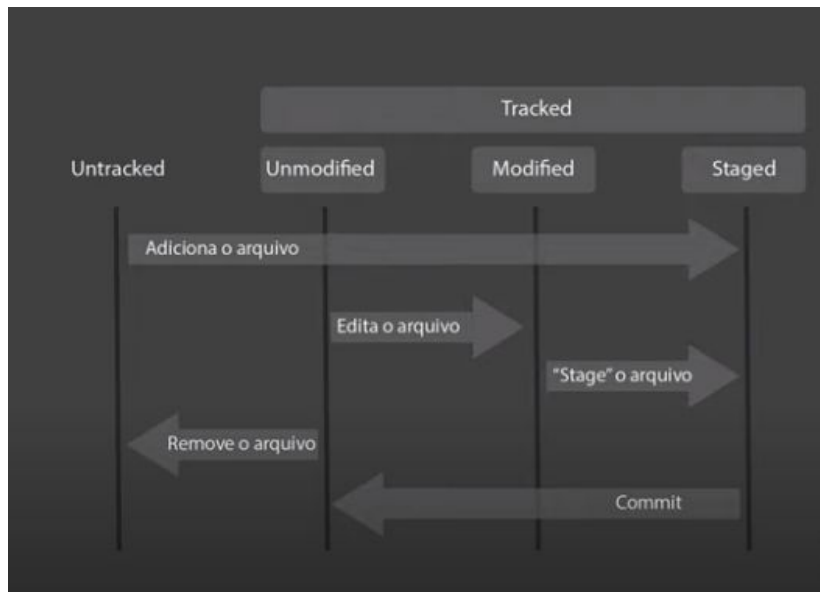
(Como e feito essa mudança? O git consegue comparar o código sha1, se houve ou não modificação)

Se rodar novamente nesse mesmo arquivo, comando `git add`, agora passa pertencer ao estágio **Staged**, aguardando para fazer parte de um outro agrupamento.

Em outro caso, se o arquivo **unmodified**, que não sofreu alterações for removido, então passa pertencer ao estágio de **Untracked** .

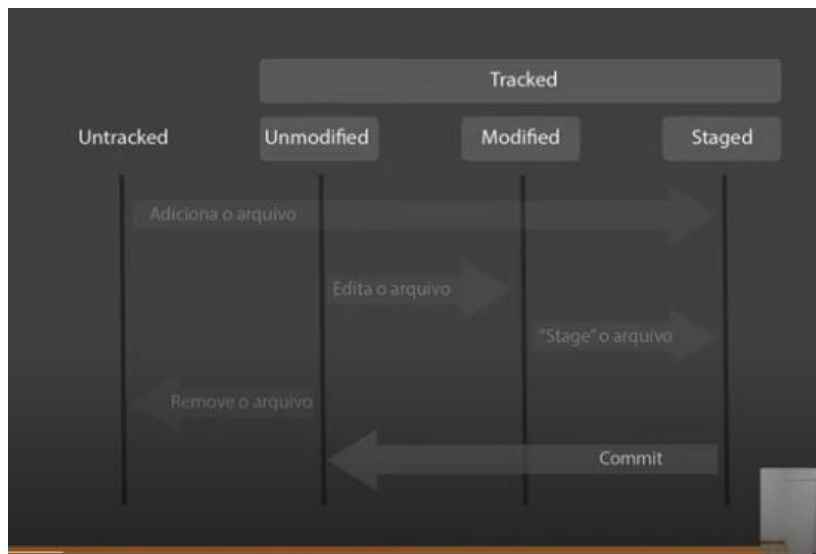
O arquivo movido para estágio **Staged**.

Está aguardando para fazer parte de um outro agrupamento, chamado **Commit**.



Quando inserimos o comando **COMMIT**, está envelopando toda mudança feito anteriormente, em seguida e movido para estágio **unmodified**, para começar o ciclo novamente.

Motivo de ser mover para **unmodified**, e para sinalizar que o arquivo está sendo salvo, então será feito um printscreen (uma foto daquele arquivo) e gerar um código Sha1 para ficar protegido.



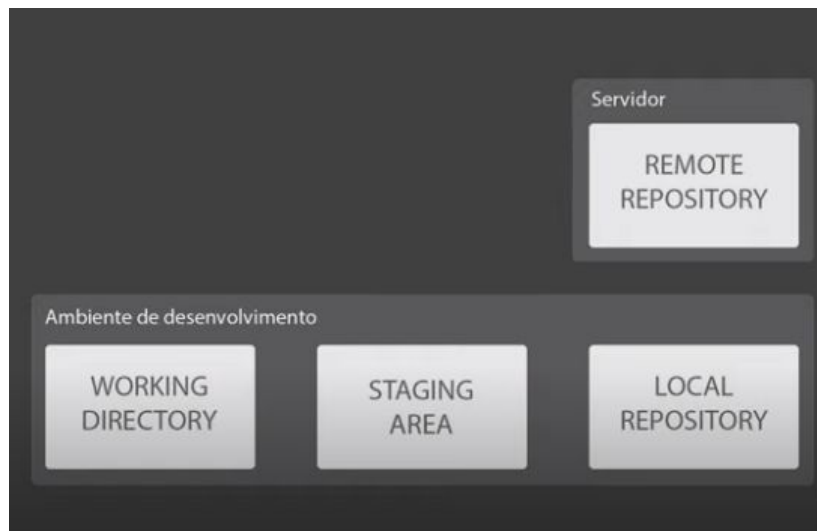
Agora vamos entender, o que significa e o que acontece dentro do repositório:

Temos dois ambiente, (Servidor / Desenvolvimento).

Tudo que pertence na sua máquina representa **Ambiente do Desenvolvimento**.

Como git é um sistema de controle de versão de arquivos de código distribuído, então ele vai ter a versão dele no **servidor (Github)**

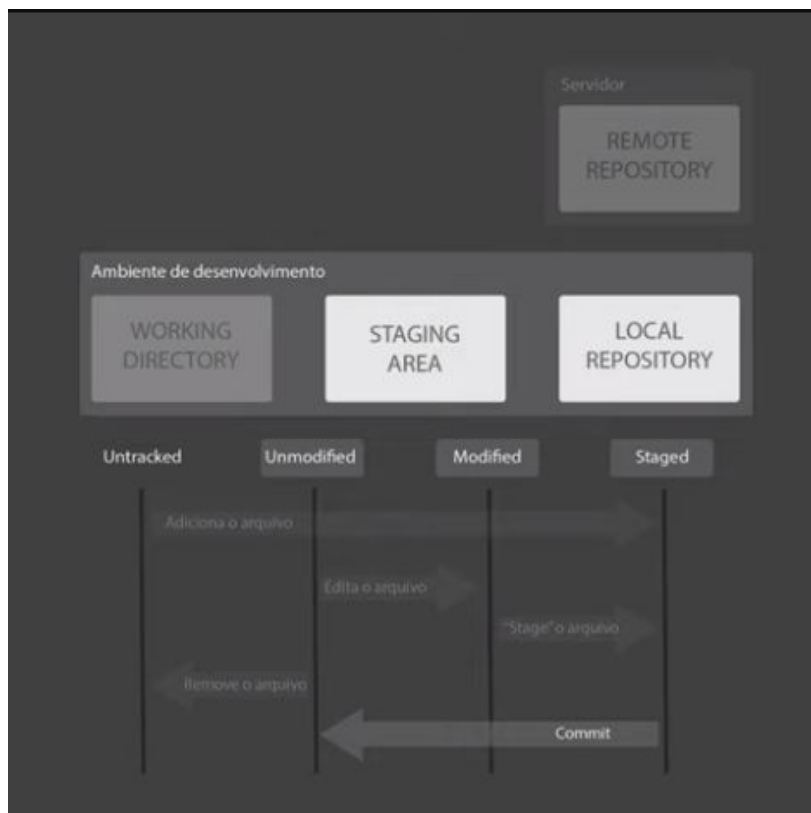
Os código gerado na sua máquina, não será refletido imediatamente na versão que está disponível em sistema remoto .
(A não ser que você programe e execute um conjunto de codificação, que faça com que você altere a codificação local para ambiente remoto)



Quando arquivo do exercício da receita foi criado, pertencia ao seu diretório de trabalho (**working directory**) >

Após inserir comando **git add ***, o arquivo foi movido direto para estágio **Staged** ou **staging**.

Aguardando para fazer parte de um outro agrupamento > só depois que efetuar comando **COMMIT**, pertence no estágio repositório local (**Local repository**), por sua vez pode ser empurrado para **REPOSITORIO REMOTO**.



Até aqui, já aprendemos como editar alguns comando básico, e também já aprendemos a navegar entre as pastas.

Relembrando os comandos : Git bash
cd / (change Director ou navegar entre as pasta)
dir (Lista de Diretório da pasta) ou LS
cd.. (Voltar pasta)
Mkdir + nome (comando p/ criar pasta)
del /rmdir (deletar só arquivos) / (remove diretorio)
Clear (clear cleaner ou limpar tela)
Atalho TAB (auto completa nome da pasta)
Silence on sucesso:

termo que, se não dizer nada na tela git bash, está tudo certo.
Echo (simplesmente printa de volta a frase que escreveu)
> (redirecionador de fluxo)
-a (mostra arquivos ocultos)
Git init (comando para poder iniciar e criamos um repositório)
Git add * (comando para poder mover arquivos e adicionar no repositório)
Git commit (comando para poder criar o primeiro commit)
Git commit -m " msg " (Seria um breve resumo para identificar esse código)
Openssl sha1 nome do arquivo com extensão para ver código
• Git status (dentro da pasta do projeto, será possível de verificar se os arquivos já foi adicionado ou não)
• Git add "nome do arquivo" (para adicionar no repositório)
• Git add -A (para adicionar todos arquivos no repositório)
• Git log (histórico do projeto)
• Git branch (Descobre em qual branch esteja trabalhando)
• Git checkout (nome) para alterar de um Branch para outro

• Git branch -m (novo-nome). Renomear Branch
• Git branch (nome) cria um novo branch
• Git branch -d (nome do branch) deletar Branch
• Git diff em caso queira saber o que exatamente foi alterado no arquivo, antes de dar comando commit
• Git diff --name only (para saber apenas o nome de quem fez alteração)

Oque e (Branch) e para que serve?

Uma **ramificação no git** é um ponteiro para as alterações feitas nos arquivos do projeto. É útil em situações nas quais você deseja adicionar um novo recurso ou corrigir um erro, gerando uma nova ramificação garantindo que o código instável não seja mesclado nos arquivos do projeto principal. Depois de concluir a atualização dos códigos da ramificação, você pode mesclar a ramificação com a principal, geralmente chamada de **master**.

Imagine que você esteja trabalhando no meio de uma grande funcionalidade, pode levar até 2 meses para terminá-la. Em uma bela manhã de sol seu chefe resolve pedir urgentemente uma alteração na versão em produção da aplicação, ou seja, você não pode utilizar o código em que está trabalhando pois o mesmo possui features inacabadas. Como resolver?

As **ramificações ou branches no Git** são formas de termos uma mesma versão do código sofrendo alterações e recebendo *commits* de diferentes fontes e inclusive por diferentes desenvolvedores.

Dessa forma, nós podemos manter um ramo para nossa funcionalidade que irá levar mais tempo e trabalhar em outro *branch* com a versão em produção para realizar alterações mais urgentes. E fica tranquilo, no fim

de tudo o Git ainda vai nos ajudar a unir os códigos desses dois ramos de forma muito simpática.

Por padrão, você sempre está trabalhando em um ramo no Git, e mesmo quando você não cria um *branch*, o Git cria automaticamente um *branch* chamado master como padrão.

Na imagem abaixo podemos ver um exemplo de trabalho com vários ramos e *commits* aplicados. Veja que em alguns pontos da história os ramos são unidos para que as alterações de um ramo sejam aplicadas a outro.



Muito utilizados para trabalhar com equipe:

Mas para isso é necessário aprender conceito do Pull Request muito utilizado no GIT HUB.

Pull Request é quando você envia uma **sugestão de melhoria para o repositório**. Quando você quer trazer, pegar, puxar algo para o seu repositório usando o git, você usa o comando git pull. Então, um pull request nada mais é do que uma requisição, ou pedido para que aquele repositório faça um pull com as suas alterações.

Dentro do git pode ter vários ramos ou (Branch), onde o ramo principal se chama master. Raramente quando tem diversos programadores, fazemos os commit e push diretamente no branch master.

Terá uma pessoa responsável pelo **merge (mesclagem)** e os demais fazem esse push nos demais ramos no Pull Request.

Grande maioria das vezes você envia seu código, alguns programadores analisam seu **código (codigo review)** e fazem alguns comentários a respeito para melhoria. Depois de corrigido e ajustado e incorporado no Branch master do projeto.

Recomendado colocar (ramificações) branch específico para cada projeto. Vamos retomar sobre esse assunto mais a frente

Arquivo README.md

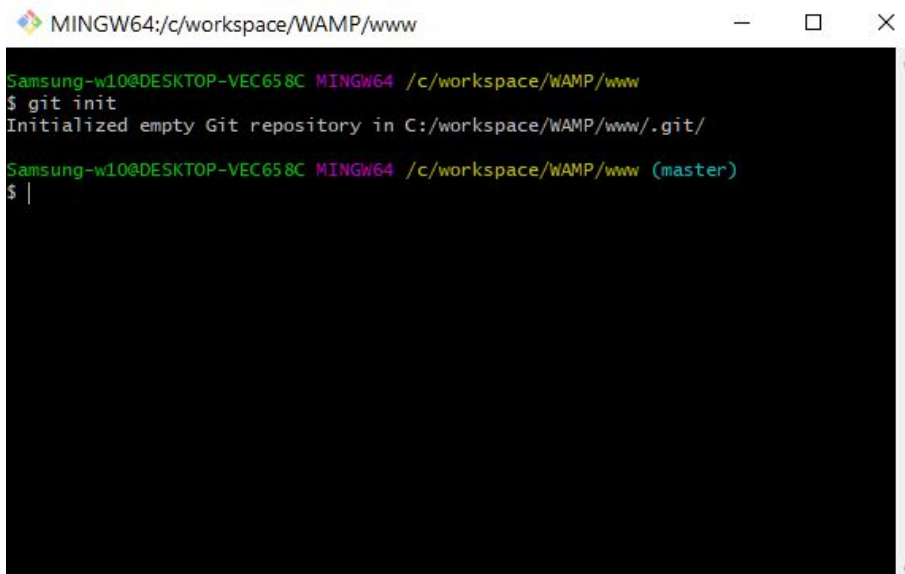
O **README.md** é um **arquivo** de texto utilizado para descrever o seu projeto. Ele é essencial se o seu projeto for público, ou seja, visa a criação e a participação de uma comunidade. Ele nada mais é do que um **arquivo** de texto onde ficam as instruções e informações descritivas. Você poderá descrever, documentar e exemplificar, o seu projeto.

Praticando mais com Git: Agora é com vocês.

- Vamos abrir git bash, e navegar até a pasta workspace.
- Agora crie uma pasta com nome: **WAMP** usando comando **mkdir**
- Dentro da pasta **WAMP**, crie uma outra pasta com nome **WWW**
- Navegue até a pasta www, vamos iniciar repositório com comando **Git Init**

Toda vez que iniciamos repositório, se cria uma pasta .git de forma oculto.

Todos arquivos e objetos alterados dentro desta pasta, será armazenado em .git



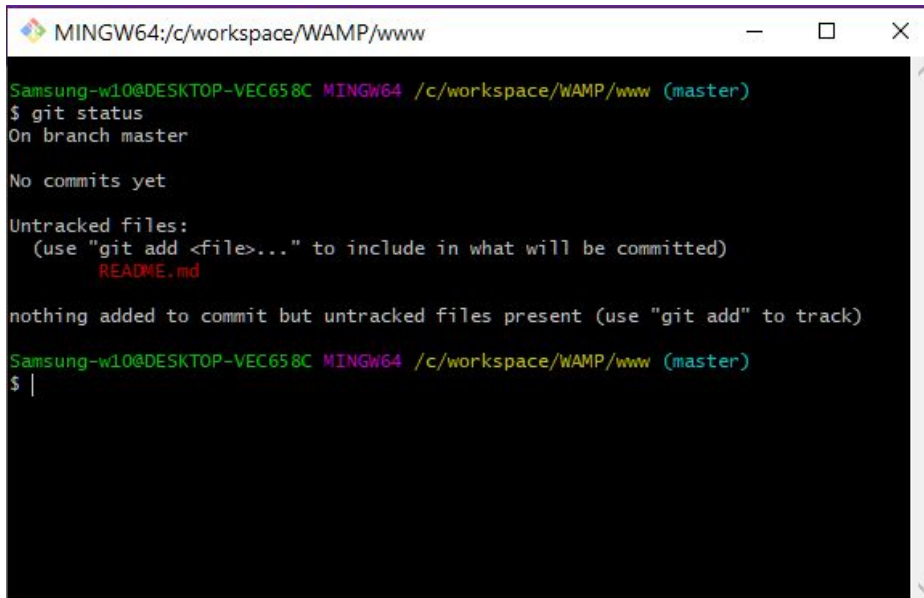
```
MINGW64:/c/workspace/WAMP/www
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www
$ git init
Initialized empty Git repository in C:/workspace/WAMP/www/.git/
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ |
```

- Vamos criar um arquivo **README.md** dentro dessa pasta www
- > Novo documento de texto, renomear para **README.md**
- > Abra arquivo readme.md e digite número de 1 a 10, de forma vertical.

Pode salvar e fechar arquivo.

- Digite o comando **Git status**, veja o que aparece: ou seja, ainda está no estágio **Untracked**.

Git não tem conhecimento desse arquivo no repositório.

A screenshot of a Windows terminal window titled 'MINGW64:/c/workspace/WAMP/www'. The terminal shows the output of the 'git status' command. The output indicates that the user is on the 'master' branch and has no commits yet. It lists 'README.md' as an untracked file. A message at the bottom states: 'nothing added to commit but untracked files present (use "git add" to track)'.

```
MINGW64:/c/workspace/WAMP/www
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md

nothing added to commit but untracked files present (use "git add" to track)
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ |
```

- **git add "nome do arquivo"** (para adicionar no repositório)
Ou **git add -A** (para adicionar todos arquivos desta pasta, no repositório)
exp: git add README.md

- Entender que o git trabalha com as modificações, não com os arquivos.
A maioria dos sistemas de controle de versão trabalham com arquivos. Você adiciona o arquivo no controle de código e o sistema acompanha as mudanças a partir daquele momento..
O Git se concentra nas mudanças de um arquivo, não no arquivo em si. Um comando `git add readme.md` não fala para o git adicionar o arquivo no repositório, mas para perceber o atual estado do arquivo para que ele seja parte de um commit depois.

- Digite novamente git status:

```
MINGW64:/c/workspace/WAMP/www
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  README.md

nothing added to commit but untracked files present (use "git add" to track)

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ git add README.md

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   README.md

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ |
```

- Dê o commit, para informar alteração que efetuamos:

Git commit -m "etiqueta com breve msg"

Parabéns, geramos commit no branch Master:

```
MINGW64:/c/workspace/WAMP/www

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ git add README.md

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   README.md

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ git commit -m "criamos arquivo READ.me e inserimos nº 1 a 10 verticalmente"
[master (root-commit) 2299900] criamos arquivo READ.me e inserimos nº 1 a 10 verticalmente
1 file changed, 20 insertions(+)
create mode 100644 README.md

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ |
```

- Limpa tela e digite o comando: **Git Log**
(permite visualizar histórico de todo registro feito naquele branch)

```
MINGW64:/c/workspace/WAMP/www
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ git log
commit 2299900a469ff1529cd07b2ae9b6ee3fe1dcf223 (HEAD -> master)
Author: g... <...@...com.br>
Date: Tue Dec 8 15:55:41 2020 -0300

    criamos arquivo READ.me e inserimos nº 1 a 10 verticalmente
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ |
```

Agora vamos supor que criamos mais dois arquivos na pasta www e efetuamos alteração acrescentando qualquer informações no arquivo READ.me

Exp:

msg READ.me "mais dois arquivos sendo adicionados"

Arquivo index.html

Arquivo style.css

Repita o comando **git status**, veja o que aparece:

Dessa vez, consta que o arquivo foi modificado e tem dois arquivos no estágio Untracked, git não tem conhecimento desses arquivos no repositório.

```
MINGW64:/c/workspace/wamp/www

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        "index.html\302\240"
        "style.css\302\240"

no changes added to commit (use "git add" and/or "git commit -a")

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www (master)
$
```

Dessa vez dê o comando **git add -A** e veja novamente com git status.

Agora reconhece que está registrado na sua máquina.

```
MINGW64:/c/workspace/wamp/www

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   "index.html\302\240"
        new file:   "style.css\302\240"

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www (master)
$ |
```

Atenção: enquanto não der comando **COMMIT**, não será reconhecido pelo git, apenas na sua máquina.

```
MINGW64:/c/workspace/wamp/www

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   "index.html\302\240"
        new file:   "style.css\302\240"

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www (master)
$ git commit -m "etiqueta 2: inserido arquivo index.html + style.css"
[master 34f10d0] etiqueta 2: inserido arquivo index.html + style.css
3 files changed, 7 insertions(+)
create mode 100644 "index.html\302\240"
create mode 100644 "style.css\302\240"

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www (master)
$ |
```

Digite comando **git log** e veja como ficou registrado, aparece histórico :

```
MINGW64:/c/workspace/wamp/www

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www (master)
$ git log
commit 34f10d04a80fd0623fc477ab3ec56245c5788e98 (HEAD -> master)
Author: [redacted] <[redacted]@outlook.com.br>
Date:   Wed Dec 9 09:24:53 2020 -0300

    etiqueta 2: inserido arquivo index.html + style.css

commit 2299900a469ff1529cd07b2ae9b6ee3fe1dcf223
Author: [redacted] <[redacted]@outlook.com.br>
Date:   Tue Dec 8 15:55:41 2020 -0300

    criamos arquivo READ.me e inserimos nº 1 a 10 verticalmente

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www (master)
$ |
```

Comando **git branch** permite saber qual branch esteja trabalhando: (o que estiver sinalizando com * é o atual branch)

Revertendo Modificações

Imagine uma situação onde você precisa reverter o projeto. Nesse caso deve usar comando **Git reset** - (escolher umas 3 tipos de resete) **nº código commit anterior** para reverter o projeto:

Mas antes é necessário entender alguns conceitos como:

3 tipos de resete: **--soft** **--mixed** **--hard**

--soft : Descarta a última confirmação, e volta para estágio anterior do último commit, mantendo os arquivos. Ou seja, Alterações e últimos arquivos atual são mantidas em etapas (*índice / index*) .

(comando mais seguro e ideal para usar com equipe)

Use isso quando perceber que fez alguns erros, mas o trabalho é bom - tudo o que você precisa fazer é recommitar de forma diferente.

O index está intocado, então você pode dar commit imediatamente se quiser - o resultado commit terá todo o mesmo conteúdo que você estava antes de reiniciar.

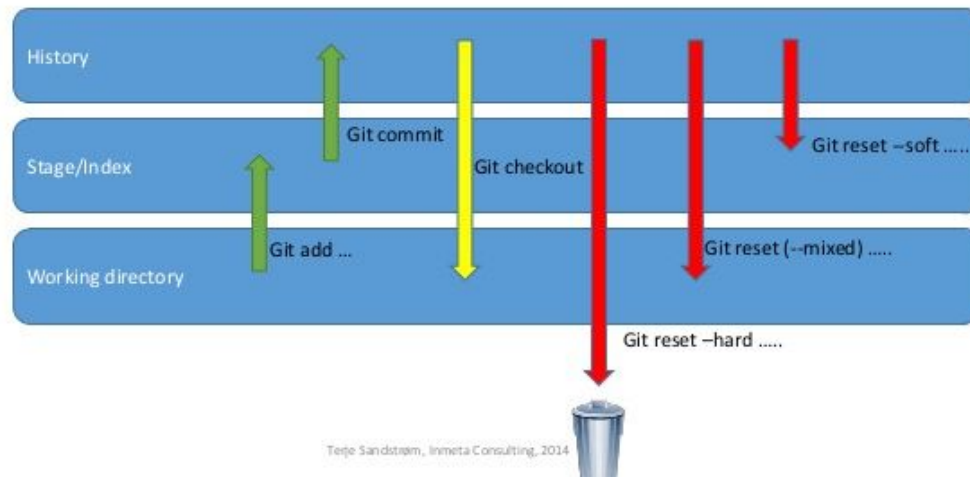
--mixed : Está descartando a última confirmação e adição (commit + add) ou seja, as alterações são deixadas na *árvore de trabalho* e não está preparado para dar commit. Será necessário dar comando add * primeiro.

Use isso quando perceber que fez alguns commit ruins, mas você quer manter todo o trabalho que você fez para que você possa corrigi-lo e recommitar.

--hard : Está descartando a última confirmação, adição e qualquer alteração feita nos códigos. Ou seja as alterações não confirmadas são removidas e desaparecem para sempre.

(Não é muito recomendado quando esteja trabalhando com equipe)

Git tree movements visualized



Pode ser útil para pessoas visuais que usam git no terminal com cores:

1. **git reset --soft** A e você verá as coisas de **B e C em verde** (faseado)
2. **git reset --mixed** A e você verá as coisas de **B e C em vermelho** (unstaged)
3. **git redefinir --hard** A e você já **não verá mudanças B e C** de qualquer lugar (será como se nunca tivessem existido)

Você pode recuperar quaisquer alterações *confirmadas* com o reflog;

Exemplo na prática:

Digite comando **git log** e veja como ficou registrado, aparece histórico :

```
MINGW64:/c/workspace/wamp/www
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www (master)
$ git log
commit 34f10d04a80fd0623fc477ab3ec56245c5788e98 (HEAD -> master)
Author: [redacted] <[redacted]@outlook.com.br>
Date:   Wed Dec 9 09:24:53 2020 -0300

    etiqueta 2: inserido arquivo index.html + style.css

commit 2299900a469ff1529cd07b2ae9b6ee3fe1dcf223
Author: [redacted] <[redacted]@outlook.com.br>
Date:   Tue Dec 8 15:55:41 2020 -0300

    criamos arquivo READ.me e inserimos nº 1 a 10 verticalmente
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www (master)
$ |
```

Para retornar no primeiro commit .

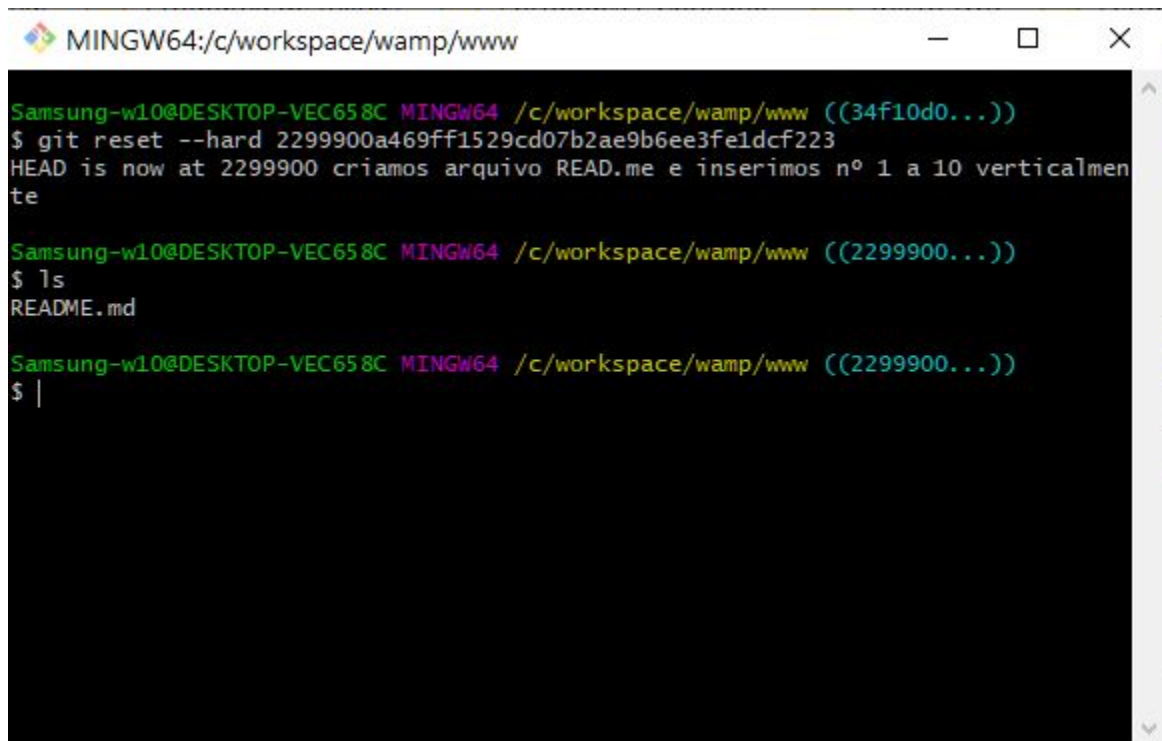
Digite o comando **git reset --hard (id, código do 1º commit)**

Exp: git reset --hard 2299900a469ff1529cd07b2ae9b6ee3fe1dcf223

Atenção! :

Você lembra que havíamos criado dois arquivos qualquer no último commit na pasta www?

Agora vá lá na pasta e verifique, esses arquivos já não existe mais!
E na tela recebe msg dizendo que HEAD (Branch) , agora pertence ao primeiro commit



```
MINGW64:/c/workspace/wamp/www

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((34f10d0...))
$ git reset --hard 2299900a469ff1529cd07b2ae9b6ee3fe1dcf223
HEAD is now at 2299900 criamos arquivo READ.me e inserimos nº 1 a 10 verticalmente

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((2299900...))
$ ls
README.md

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((2299900...))
$ |
```

Na mesma pasta www, crie um novo arquivo de test1, e adicione qualquer coisa no arquivo READ.MD ... exp: inserido arquivo test1

Agora digite comando **git commit -am "msg"**
digite comando **git log**: Foi criado um novo commit novamente.

```
MINGW64:/c/workspace/wamp/www
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((2299900...))
$ git commit -am "criando arquivo test1 e adicionado qualquer coisa no Readme"
[detached HEAD a08d156] criando arquivo test1 e adicionado qualquer coisa no Readme
1 file changed, 4 insertions(+)

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((a08d156...))
$ git log
commit a08d156d8df20a884079e7cb97de8143d12ffcd8 (HEAD)
Author: <[redacted]@outlook.com.br>
Date: Thu Dec 10 12:32:43 2020 -0300

    criando arquivo test1 e adicionado qualquer coisa no Readme

commit 2299900a469ff1529cd07b2ae9b6ee3fe1dcf223
Author: <[redacted]@outlook.com.br>
Date: Tue Dec 8 15:55:41 2020 -0300

    criamos arquivo READ.me e inserimos nº 1 a 10 verticalmente

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((a08d156...))
$ |
```

Agora vamos testar com comando `--soft` :

Para retornar no primeiro commit .

Digite o comando `git reset --soft (id, código do 1º commit)`

Exp: `git reset --soft 2299900a469ff1529cd07b2ae9b6ee3fe1dcf223`

Atenção! :

Se consultar a pasta www, o arquivo criado test1 permanece.

Conteúdo adicionado no arquivo README.md também permanece.

Mas se digitar comando `git log`, veja o que aparece:

Retornamos ao primeiro commit.

```
MINGW64:/c/workspace/wamp/www

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((a08d156...))
$ git reset --soft 2299900a469ff1529cd07b2ae9b6ee3fe1dcf223

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((2299900...))
$ git log
commit 2299900a469ff1529cd07b2ae9b6ee3fe1dcf223 (HEAD)
Author: <[redacted]@outlook.com.br>
Date: Tue Dec 8 15:55:41 2020 -0300

    criamos arquivo READ.me e inserimos nº 1 a 10 verticalmente

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((2299900...))
$ |
```

Se digitar comando **git status**, veja o que aparece: Git já reconhece o arquivo `readme.md`.

Só falta adicionar arquivo `test1` porque nós tivemos que criamos um novo arquivo para poder mostrar em prática.

Mas ao contrário, se os arquivos já existirem dentro do projeto como no caso `README`, outros arquivos também estaria na cor Verde.

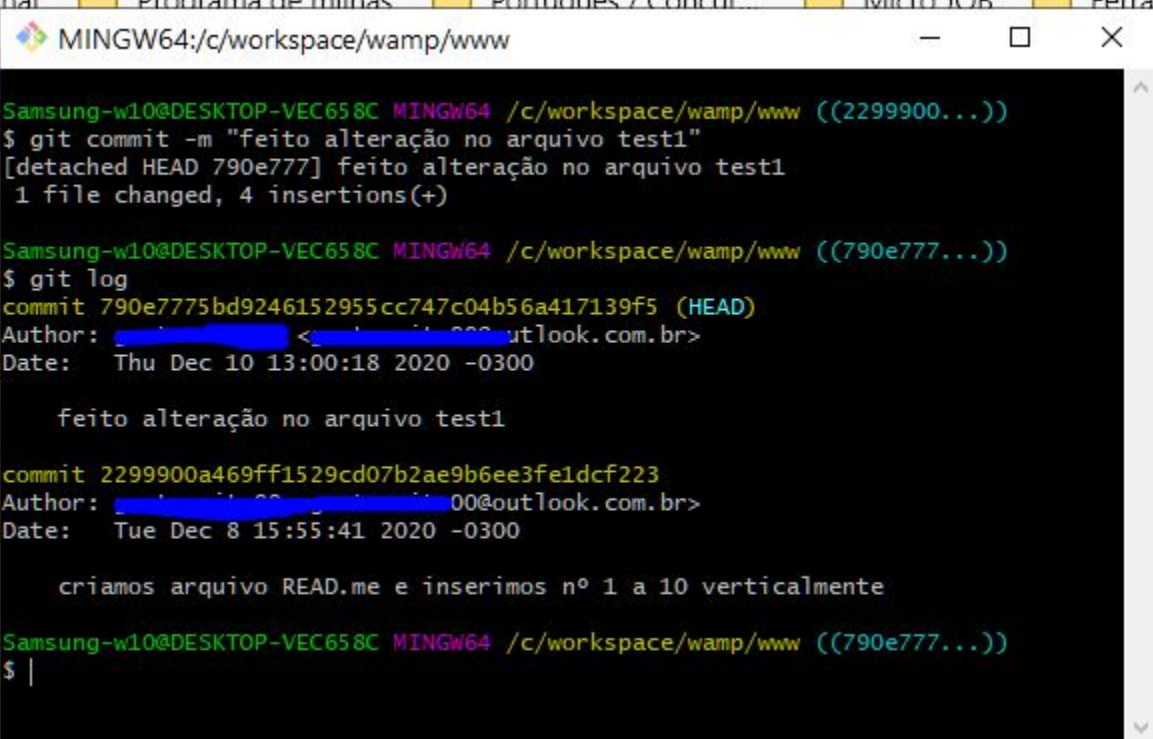
```
MINGW64:/c/workspace/wamp/www

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((2299900...))
$ git status
HEAD detached from 34f10d0
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test1.txt

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((2299900...))
$ |
```


Assim você já pode commitar direto usando comando `git commit -m "msg"` ou `git commit -am "msg"`



```
MINGW64:/c/workspace/wamp/www
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((2299900...))
$ git commit -m "feito alteração no arquivo test1"
[detached HEAD 790e777] feito alteração no arquivo test1
1 file changed, 4 insertions(+)

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((790e777...))
$ git log
commit 790e7775bd9246152955cc747c04b56a417139f5 (HEAD)
Author: <[redacted]@outlook.com.br>
Date: Thu Dec 10 13:00:18 2020 -0300

    feito alteração no arquivo test1

commit 2299900a469ff1529cd07b2ae9b6ee3fe1dcf223
Author: <[redacted]@outlook.com.br>
Date: Tue Dec 8 15:55:41 2020 -0300

    criamos arquivo READ.me e inserimos nº 1 a 10 verticalmente

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/wamp/www ((790e777...))
$ |
```

Importante dar comando `git status` para verificar se não existe mais nada de pendência, quando deletamos commit usando comando `-hard`, as vezes pode acontecer do HEAD continuar no commit que deletamos.

Para regularizar, será necessário alterar Branch para o atual.

Como faço para trocar de Branch?

Digite comando `Git checkout (nome)`

Exp: `git checkout master`

Como conferir Branch atual?

Exp: `git branch` (o que estiver com sinal * é o Branch atual.)

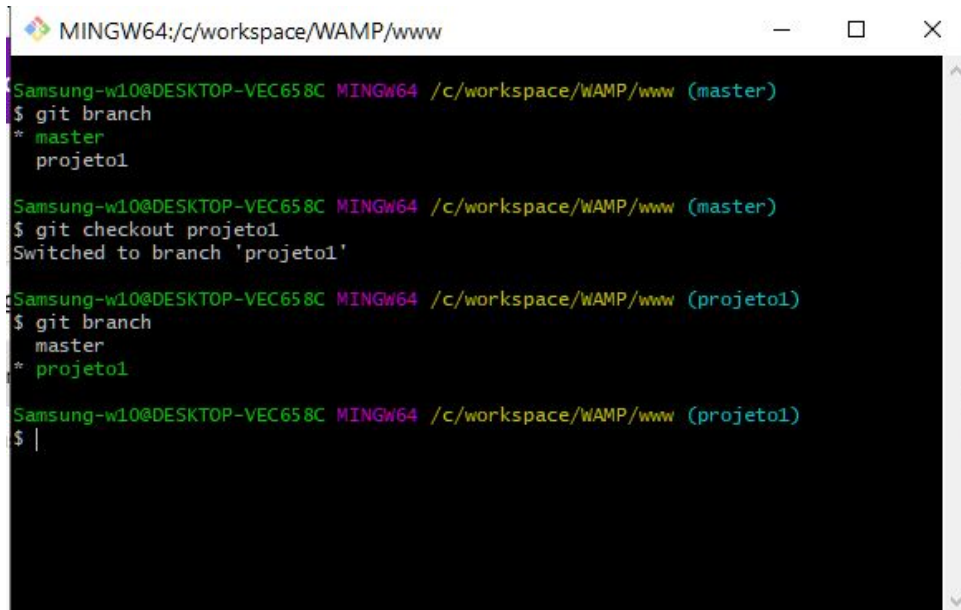
Como criar novo Branch?

Lembra quando recomendei a colocar (ramificações) branch específico para cada projeto? Então, deve digitar seguinte comando:

Exp: **git branch (nome que deseja)**

git branch projeto1

E se digitar : **git checkout projeto1**, veja o que acontece:

A screenshot of a terminal window titled 'MINGW64:/c/workspace/WAMP/www'. The terminal shows the following sequence of commands and output:
1. Command: `$ git branch`
Output: `* master`
 `projeto1`
2. Command: `$ git checkout projeto1`
Output: `Switched to branch 'projeto1'`
3. Command: `$ git branch`
Output: `* master`
 `* projeto1`
4. Command: `$`
Output: `$ |`

Preste muita Atenção! :

Agora temos dois branch: Master / Projeto1

Todos arquivos e alterações feito em cada Branch, significa que permanece naquele Branch trabalhado .

Exp: No arquivo Readme.md digitamos números 1 a 10 verticalmente.

Se editar mesmo arquivo acrescentando até 20 e der comando commit no branch projeto1. Essas alterações não será possível de visualizar no branch Master,e sim apenas no branch projeto1 .


```
MINGW64:/c/workspace/WAMP/www

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (projeto1)
$ git branch
  master
* projeto1

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (projeto1)
$ git commit -am "adicionaei numero ate 20, testando novo branch"
[projeto1 b4a924e] adicionaei numero ate 20, testando novo branch
1 file changed, 16 insertions(+), 3 deletions(-)

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (projeto1)
$ git status
On branch projeto1
nothing to commit, working tree clean

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (projeto1)
$ |
```

```
MINGW64:/c/workspace/WAMP/www

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (projeto1)
$ git log
commit b4a924e3d56eff866859462bde40a9cfd03dd260 (HEAD -> projeto1)
Author: <[redacted]@outlook.com.br>
Date: Thu Dec 10 15:56:11 2020 -0300

    adicionaei numero ate 20, testando novo branch

commit 34f10d04a80fd0623fc477ab3ec56245c5788e98 (master)
Author: <[redacted]@outlook.com.br>
Date: Wed Dec 9 09:24:53 2020 -0300

    etiqueta 2: inserido arquivo index.html + style.css

commit 2299900a469ff1529cd07b2ae9b6ee3fe1dcf223
Author: <[redacted]@outlook.com.br>
Date: Tue Dec 8 15:55:41 2020 -0300

    criamos arquivo READ.me e inserimos nº 1 a 10 verticalmente

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (projeto1)
$ |
```

Altere branch para master, veja que commit anterior não é registrado.

```
MINGW64:/c/workspace/WAMP/www

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (projeto1)
$ git checkout master
Switched to branch 'master'

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ git log
commit 34f10d04a80fd0623fc477ab3ec56245c5788e98 (HEAD -> master)
Author: <[redacted]@outlook.com.br>
Date:   Wed Dec 9 09:24:53 2020 -0300

    etiqueta 2: inserido arquivo index.html + style.css

commit 2299900a469ff1529cd07b2ae9b6ee3fe1dcf223
Author: <[redacted]@outlook.com.br>
Date:   Tue Dec 8 15:55:41 2020 -0300

    criamos arquivo READ.me e inserimos nº 1 a 10 verticalmente

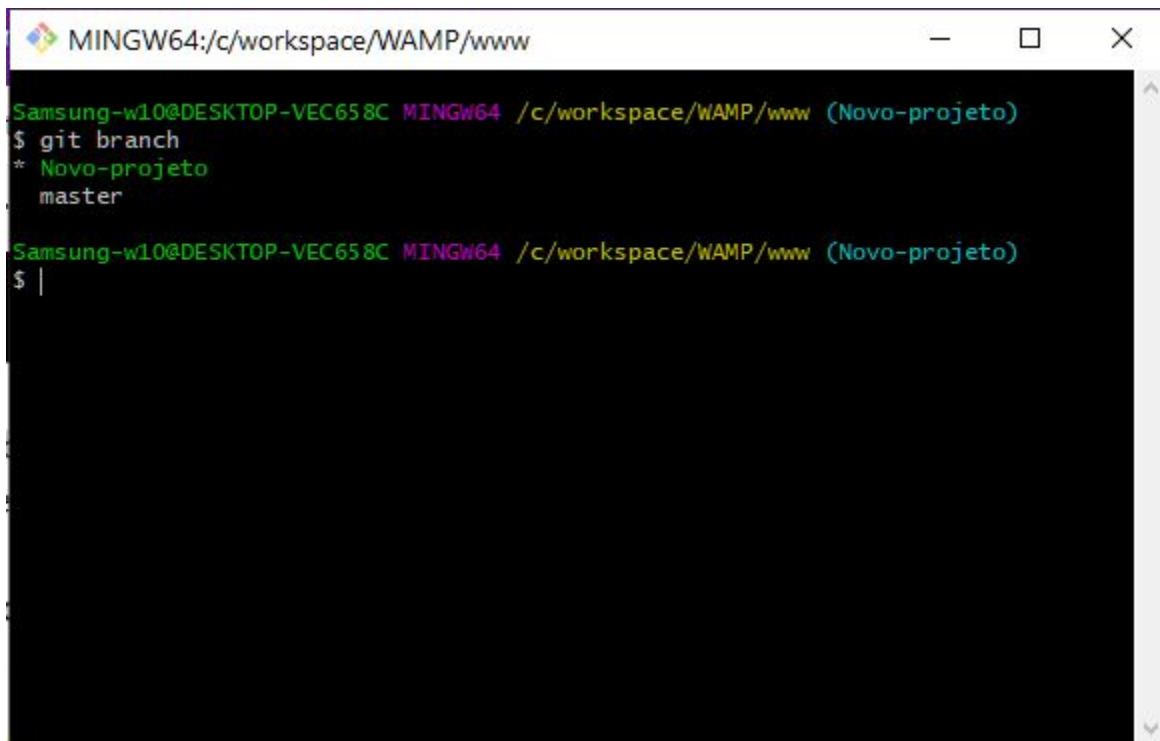
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ |
```

Faz um teste e abra o arquivo Readme.md: número adicionado até 20 não ficará mais visível no branch master.

Mas , se voltar acesso para branch projeto1, será possível de visualizar números adicionados até 20.

Como renomear Branch?

Digite comando: `Git branch -m (novo-nome)`.

A screenshot of a Windows command prompt window. The title bar shows the path 'MINGW64:/c/workspace/WAMP/www'. The prompt is 'Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (Novo-projeto)'. The user has entered the command '\$ git branch'. The output shows two branches: 'Novo-projeto' (marked with an asterisk) and 'master'. The prompt is now '\$ |'.

Comando **git status** permite visualizar estatus do arquivo
Mas em caso queira saber o que exatamente foi alterado no
arquivo, antes de dar comando commit então, digite o comando :
git diff

Ou no caso para saber apenas o nome de quem fez alteração: **git diff --name only**

Vamos supor que vou remover número 1 a 5 no arquivo
readme.md
E acrescento qualquer coisa no final da linha.

Veja como aparece na tela, em vermelho sinaliza que foi removido
número de 1 a 5 . Em verde com sinal + é o que foi acrescentado.

Recomendado a utilizar mesmo cadastro feito no git local:
Para consultar dados já registrado no git local, abra git bash e insira
seguinte comando : `git config --list`

Para alterar os dados da configuração, siga seguinte comando:
Abra git bash > vai até onde encontra arquivo criado no exemplo (workspace) > `git config --global --unset user.email` >
> `git config --global --unset user.nickname`

Se der comando `git config --list`, será possível de verificar que os dados cadastrados não estão mais disponíveis.

Pronto, agora só cadastrar novamente

> `git config --global user.email "seuemail@gmail.com"`
> `git config --global user.nickname "seu nome cadastrado no github"`

- **Como criar um repositório remoto**

Após efetuar cadastro no Github > clique no seu perfil > your repositories (seus repositórios) > New (Novo) > Agora e só preencher os dados como:

Nome do repositório : (Tudo minúsculo, sem caracteres , e tudo junto)

Descrição (opcional):

Escolher se e público ou não:

Marca umas das alternativas: Inicializar com Readme

(Nessa opção, como já criamos um readme em nossa máquina local não precisa assinar)

Cria repositórios

Após efetuar esse procedimento vai ser direcionado a uma página.

E logo de cara vai aparecer um link como:

https://github.com/nome_do_usuario/no_do_repositório.git

Você pode divulgar seu repositório através desse link.

- **Como se aponta nosso código da nossa máquina do repositório local , para repositório remoto ?**

Para você ter uma autorização e efetuar alterações nos arquivos que consta no repositório local para remoto, é necessário gerar uma chave.

Segue o link tutorial de como gerar essa chave:

[Generating a new SSH key and adding it to the ssh-agent - GitHub Docs](#)

Se você seguir orientação corretamente vai aparecer uma tela semelhante a esse:

OBS: no momento que pedir a senha, não é obrigatório registrar..
Aperte ENTER duas vezes então está tudo ok.

```
MINGW64:/c/workspace/WAMP/www
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ ssh-keygen -t rsa -b 4096 -C "[redacted]@outlook.com.br"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Samsung-w10/.ssh/id_rsa):
Created directory '/c/Users/Samsung-w10/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Samsung-w10/.ssh/id_rsa
Your public key has been saved in /c/Users/Samsung-w10/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:w1xbjkcQfz0ePB984iiyH6/EfqihlzRCsvx0bZTNX18 [redacted]@outlook.com.br
The key's randomart image is:
+---[RSA 4096]-----+
|
|  o
|  o
| . + +
| .o.. X + =
| . ++ S = * + E
| o .+o+ . * =
| ..o=0*.. +
| ..oo*+.
| .o.oo+o.
+---[SHA256]-----+

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ |
```

No exemplo acima consta que foi criado uma pasta .ssh

Navegue até essa pasta .

Exp: c:/Users/Samsung-10/.ssh

Dentro desta pasta terá acesso a duas chaves.

id_rsa : Privada

id_rsa.pub : Pública

O nosso interesse é o público, então abra arquivo com bloco de nota e copie o conteúdo da chave.

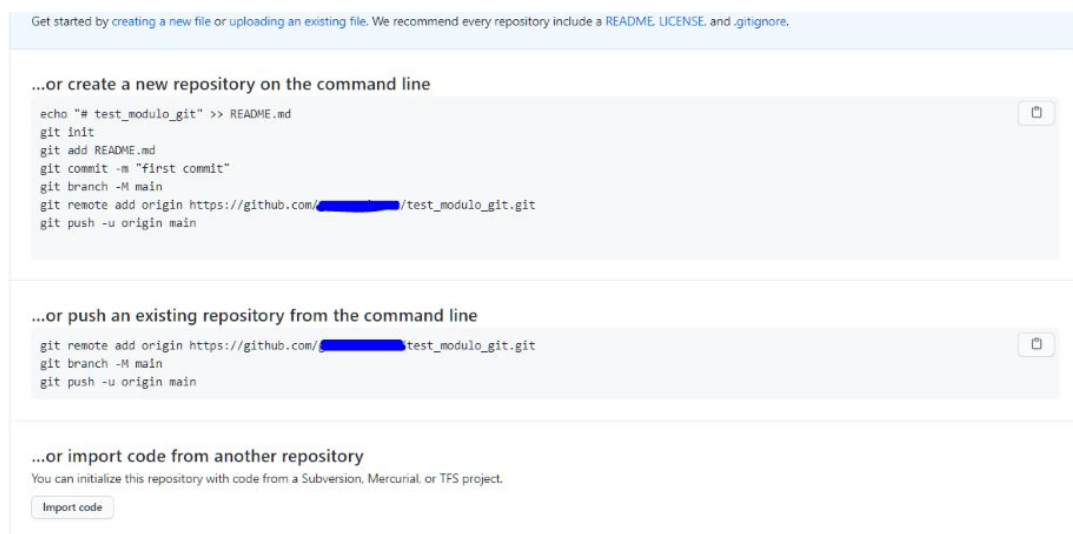
Vá até o github > Clique no seu perfil > em seguida na opção setting

No lado esquerdo, encontre aba SSH and GPG key > new ssh key

Crie um título de identificação da chave:
exp, Meu acesso Home
E cole a chave no campo disponível abaixo.

Pronto, agora está autorizado para edição e alterações no repositório local para remoto.

Continuando com a configuração:
Retorne novamente o acesso no github no seu repositório:
Se fez tudo corretamente, vai aparecer uma tela semelhante a essa:



Em caso for criar uma nova repositório, siga a primeira instrução

Mas como já temos repositório criado localmente então, siga a segunda instrução .

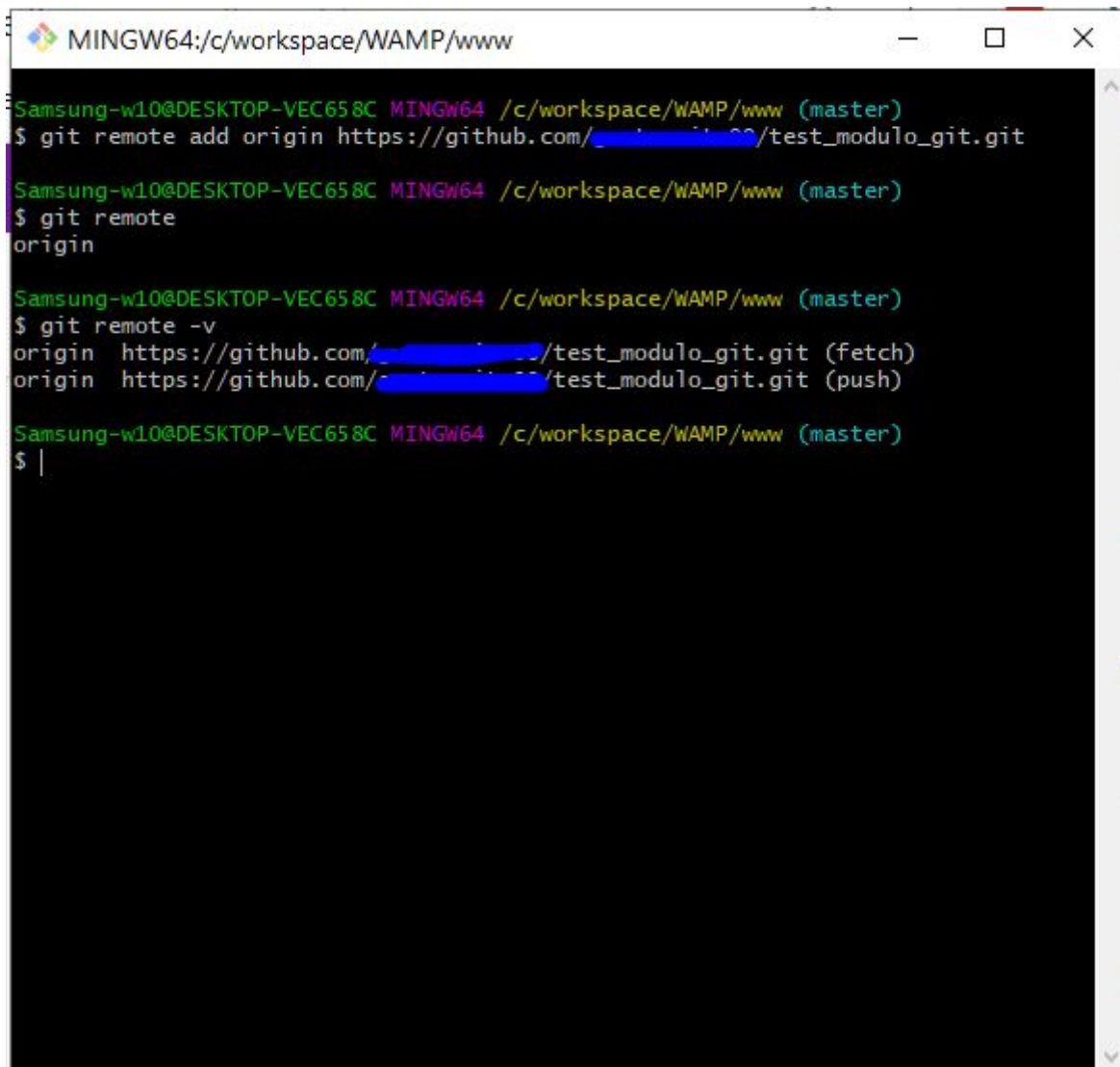
Git remote add origin
https://github.com/nome_do_usuario/no_do_repositório.git

Após efetuar comando e caso queira consultar registro.

Digite comando: `git remote`

Versão origin remoto já consta registrado.

Comando `git remote -v`, permite visualizar duas opções disponíveis:

A screenshot of a Windows command prompt window titled "MINGW64:/c/workspace/WAMP/www". The window shows a series of Git commands and their outputs. The user is in the "master" branch. The commands and outputs are as follows:

```
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ git remote add origin https://github.com/[redacted]/test_modulo_git.git

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ git remote
origin

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ git remote -v
origin https://github.com/[redacted]/test_modulo_git.git (fetch)
origin https://github.com/[redacted]/test_modulo_git.git (push)

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (master)
$ |
```

Opção fetch é a capacidade que tenho de eu puxar conteúdo de um usuário que editou arquivos na Espanha ou em outros lugares, então com esse link e comando consigo e permite puxar alteração feito pelo outro usuário para meu código que está no meu computador.

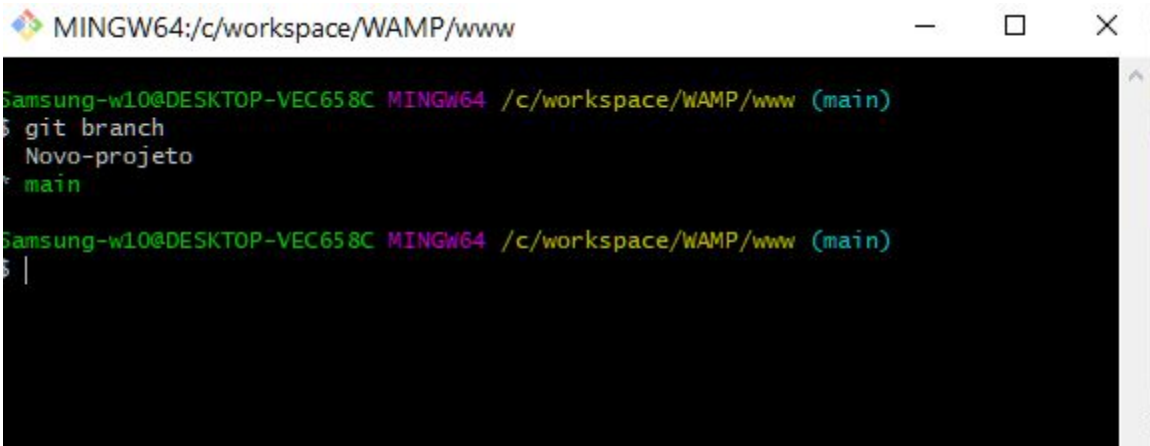
Opção push: seria o processo inverso, levar coisas do meu repositório local para remoto.

Após der próximos comando

```
git branch -M main
```

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

E se der um `git branch`, novo branch chamado `main`(principal) foi registrado.

A screenshot of a terminal window titled 'MINGW64:/c/workspace/WAMP/www'. The terminal shows the command 'git branch' being executed, followed by the output 'Novo-projeto' and 'main'. The prompt '\$' is visible at the end of the line.

```
MINGW64:/c/workspace/WAMP/www
$ git branch
Novo-projeto
main
$
```

Retorne novamente no Git Hub e atualize a página .

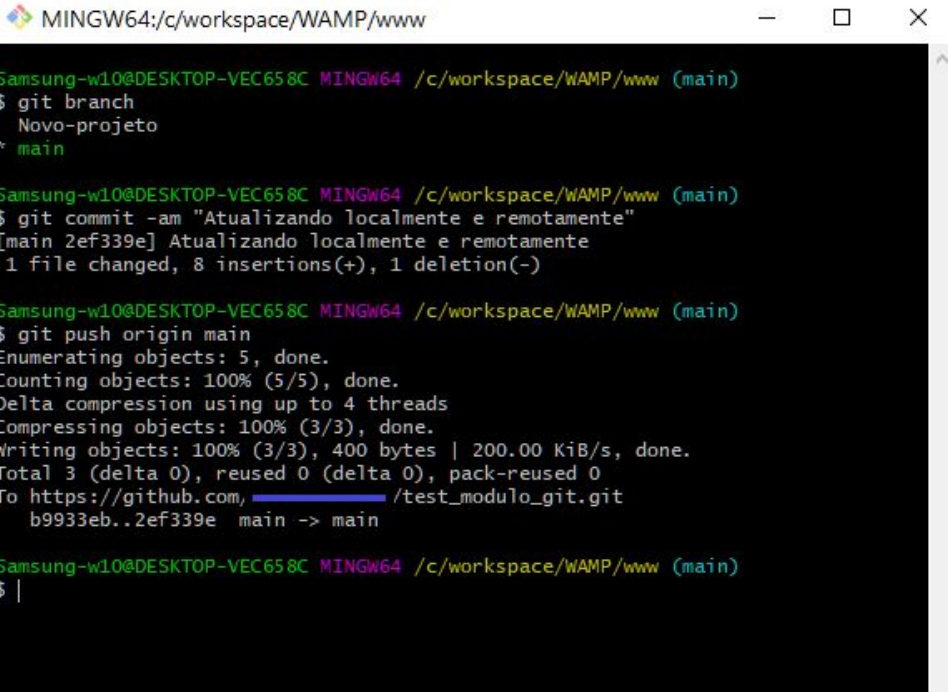
Como que vou atualizar pelo meu git hub as atualizações feitas?

Abra arquivo `Readme.md` que utilizamos desde início, acrescente nova informações

De o comando `commit` novamente no repositório local, mas isso não significa que procedimento efetuara `commit` remotamente.

Então digite comando: git push(empurrar repositório) origin (origem do destino) (e qual branch será enviado)

Exp: **git push origin main**

A screenshot of a terminal window titled 'MINGW64:/c/workspace/WAMP/www'. The terminal shows a series of git commands and their outputs. First, 'git branch' is run, showing 'Novo-projeto' and '* main'. Then, 'git commit -am "Atualizando localmente e remotamente"' is run, resulting in '[main 2ef339e] Atualizando localmente e remotamente' and '1 file changed, 8 insertions(+), 1 deletion(-)'. Finally, 'git push origin main' is run, showing progress for enumerating, counting, compressing, and writing objects, and finally pushing to 'https://github.com, /test_modulo_git.git' with the commit 'b9933eb..2ef339e main -> main'.

```
MINGW64:/c/workspace/WAMP/www
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ git branch
  Novo-projeto
* main

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ git commit -am "Atualizando localmente e remotamente"
[main 2ef339e] Atualizando localmente e remotamente
1 file changed, 8 insertions(+), 1 deletion(-)

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 400 bytes | 200.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com, /test_modulo_git.git
   b9933eb..2ef339e  main -> main

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ |
```

Se atualizar página do GitHub, atualizações consta disponível. Experimente agora copiar arquivo estrogonof da pasta livro de receita e coloque dentro da pasta www e dê os comando já ensinados .

Git status

git add * estrogonof.md

Git -A

Git commit -am "msg atualizado"

Git push origin main

Atualizar página do GitHub

Parabéns!, já consta todos arquivos...

Como ignorar alguns arquivos que você não deseja sincronizar para repositório remoto?

Para esse exemplo: vamos criar três arquivos na pasta www
1º de backup.sql , dentro desse arquivo digite qualquer coisa
2º de senhas.txt , digite algumas senhas qualquer
3º com nome **.gitignore**

Digite o comando status, verá que tem dois arquivos esperando para adicionar e commitar. Mas eu não quero que apareça esses arquivos.

Abra o arquivo **.gitignore** e registre pasta, arquivo, documentos, qualquer arquivo que deseja ignorar.

Exp:

backup.sql

Senhas.txt

Ou

*.sql

*.txt

*Antes do ponto representa para selecionar todos arquivos com extensão selecionada.

Repita comando **git status**: veja que os arquivos ficou oculto e aparece apenas o arquivo **.gitignore**

```
MINGW64:/c/workspace/WAMP/www

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        backupbanco.sql
        senhas.txt

nothing added to commit but untracked files present (use "git add" to track)

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add" to track)

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ |
```

Só adicionar, comitar com msg, e dar push e sincronizar com remoto.

```
MINGW64:/c/workspace/WAMP/www
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ git -A
unknown option: -A
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
      [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
      [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
      [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
      <command> [<args>]

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ git add -A

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ git commit -am "adicionando arquivo gitignore"
[main f6582b5] adicionando arquivo gitignore
1 file changed, 2 insertions(+)
create mode 100644 .gitignore

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 308 bytes | 308.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/[redacted]/test_modulo_git.git
67f6a6d..f6582b5 main -> main

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ |
```

Nesse link será possível de encontrar diversos gitignore de acordo com vários projetos: [github/gitignore: A collection of useful .gitignore templates](https://github.com/gitignore-io)

Vamos sincronizar outro Branch para remoto e excluir

Digite comando git branch:

Em nosso exemplo tem mais um Branch chamado Novo-projeto.

Digite comando git push origin Novo-projeto


```
MINGW64:/c/workspace/WAMP/www
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ git branch
  Novo-projeto
* main

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ git push origin Novo-projeto
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 361 bytes | 180.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'Novo-projeto' on GitHub by visiting:
remote:   https://github.com/_____/test_modulo_git/pull/new/Novo-proje
to
remote:
To https://github.com/_____/test_modulo_git.git
* [new branch]      Novo-projeto -> Novo-projeto

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ |
```

Pode atualizar git hub que vai constar dois branch.
Para excluir um dos branch, digite o comando:

git push origin :Novo-projeto
Branch deletado.

```
Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ git push origin :Novo-projeto
To https://github.com/_____/test_modulo_git.git
- [deleted]          Novo-projeto

Samsung-w10@DESKTOP-VEC658C MINGW64 /c/workspace/WAMP/www (main)
$ |
```

Até agora usamos comando git push para sincronizar do repositório local para remoto, agora vamos fazer o inverso.

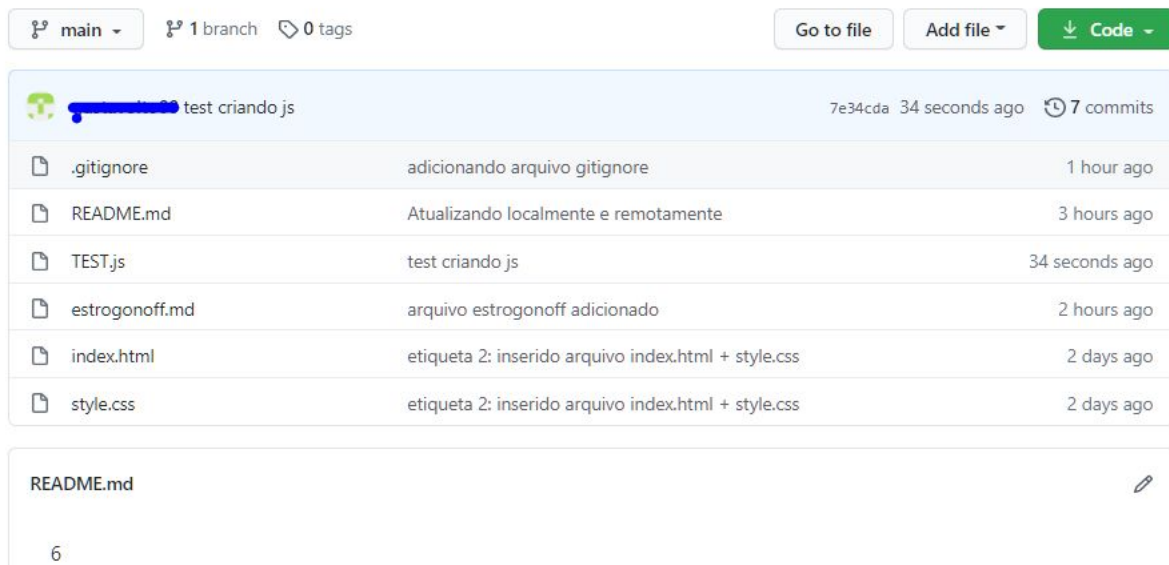
Abra git hub, vai na opção da aba **Add file**, em seguida **Creat new file**

Vamos nomear de TEST.js

Preencha o arquivo com qualquer coisa

Lá em baixo na opção **Commit new file**, digite test criando js

Aperte botão **commit new file** , Atualize git hub



Esse TEST.js não existe no meu repositório local.

Então como atualizar de repositório remoto para local?

Atenção ! Antes de realizar qualquer alterações no seu projeto, baixe sempre as atualizações feitas pela equipe ou por você mesmo em outro momento.

Digite comando: **git pull** para trabalhar sempre com a última versão.

Exp: **git pull origin main**

Como clonar um projeto?

Navegue até a pasta do interesse e digite seguinte comando:

Git clone (url do projeto)

Como permitir comentário no seu projeto em git hub?

No git hub existe uma aba chamado **pull request**, só clicar na opção criar novo descrevendo a linha de pensamento de forma clara, apenas isso...

-

Link de referência para aprender sobre comando GIT

[Git How To: Tutorial Guiado de Git](#)

[Comandos Git - Aprenda Git do básico ao avançado](#)

Bom aprendizado a todos!