**GIT / GITHUB – Anotações dos Cursos**

Cursos e Tutoriais:

Controle de versão

https://king.host/wiki/artigo/controle-de-versoes/

GIT

Site oficial: https://git-scm.com/download

https://king.host/wiki/base-de-conhecimento/git/

\*

Youtube - Como usar Git e Github na prática: Guia para iniciantes | Mayk Brito:

https://www.youtube.com/watch?v=2alg7MQ6\_sI

Youtube - Entenda o GIT em 10 MINUTOS

https://www.youtube.com/watch?v=FV-hMoqHtcU

Youtube: Curso GIT e GITHUB - Como criar um repositório no GITHUB e clonar no Windows

https://www.youtube.com/watch?v=h1-XZ9Kh1H4

Youtube - O Git que será cobrado no seu 1º dia de trabalho - DevMedia

https://www.youtube.com/watch?v=CGTsIz5t0eY

Youtube - CURSO COMPLETO DE GIT (2 HORAS E 30 MINUTOS)

https://www.youtube.com/watch?v=OuOb1\_qADBQ

Youtube: GitHub | Guia Completo do Iniciante

https://www.youtube.com/watch?v=UbJLOn1PAKw

--------------------------------------------------------------------------

CURSOS:

Gratuitos:

UDEMY - GIT PARA INICIANTES

https://www.udemy.com/course/git-para-iniciantes/learn/lecture/11300204#overview

UDEMY - GIT E GITHUB PARA INICIANTES

https://www.udemy.com/course/git-e-github-para-iniciantes/learn/lecture/5120522#announcements

Pagos

UDEMY - Git e Github na Vida Real

https://www.udemy.com/course/git-e-github-na-vida-real/

--------------------------------------------------------------------------

**GIT – Anotações do Curso**

**GIT --VERSION**

Finalidade:

- Permite saber se o GIT está instalado;

- Permite saber qual a versão instalada.

Exemplos: **git --version**



-

**GIT CONFIG --GLOBAL <NOME\_DA\_CHAVE> “VALOR\_DA\_CHAVE”**

Permite criar variáveis

Globais → vale para todas as pastas;

Exemplos:

**git config --global user.name “carlos”**

**git config --global user.email “brhavos@gmail.com”**

Locais → vale apenas para a pasta (ou repositório) atual.

Exemplos:

**git config user.name “carlos”**

**git config user.email “brhavos@gmail.com”**

Apagando uma variável:

Global:

git config --global --unset user.teste

Local:

git config --unset user.teste

-

**git config --list**

Finalidade:

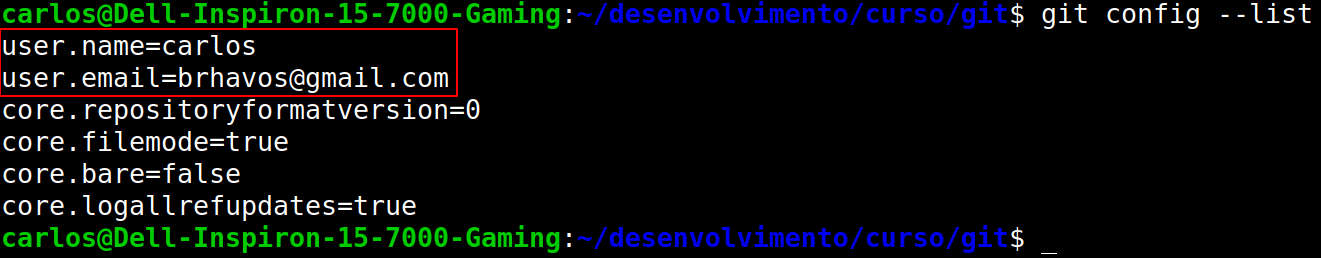
- Mostra as configurações do GIT, por exemplo:

– nome,

- e-mail,

- Editor-padrão,

- Etc.

Exemplo: **git config --list**

Em destaque estão as 2 variáveis INDISPENSÁVIS para o GIT trabalhar:

- **name**

- **email**

**Observação**: Notar que é “**e**mail” e não somente “mail”.

**git config <nome\_da\_variável>**

Finalidade: Lista o valor de uma variável específica.

Exemplos:

1-) **git config user.name**

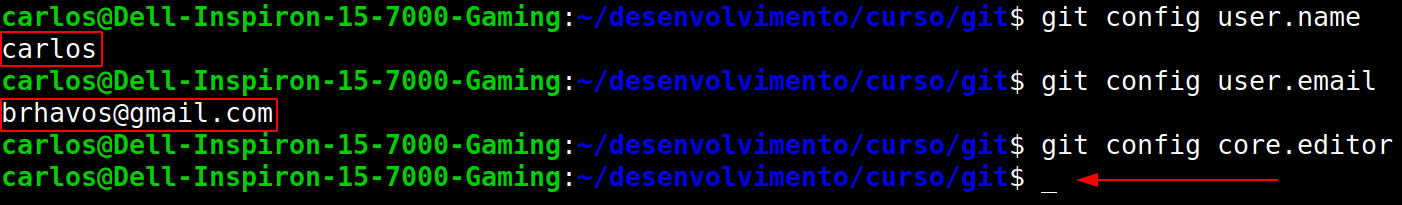
mostra o nome do usuário

2-) **git config user.email**

mostra o e-mail do usuário

3-) **git config core.editor**

mostra o editor global padrão



Em destaque:

- Os 2 primeiros comandos retornaram conteúdo, mas…

- O último não – isso significa que ainda não foi declarado um editor padrão.

-

**Definindo o VS como editor padrçao do GIT**

1-) Verifica se o VS está instalado:

**code -help**

2-) Faz a atribuição de uma variável:

**git config --global core.editor 'code --wait'**

3-) Teste – abrindo o arquivo de configuração do git:

**git config --global -e**

**GIT INIT**

Finalidade: Inicia um novo controle de versão de uma pasta local da estação de trabalho no GIT (repositório local):

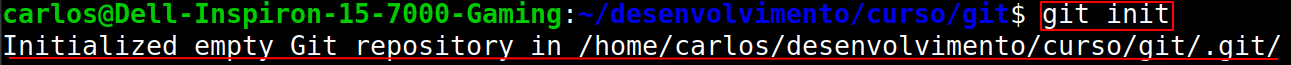
Passos para isto acontecer:

1-) Criar uma pasta na máquina local para o projeto que deseja controlar o versionamento;

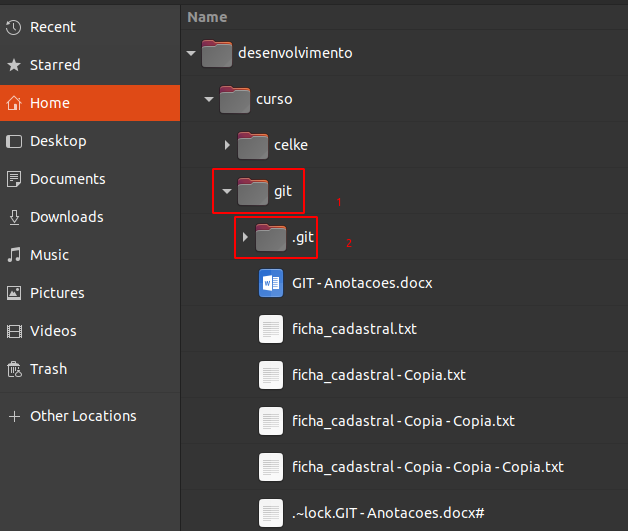
2-) Vai dentro da pasta;

3-) Aplicar comando “git init”.

Exemplo – observar o comando e a mensagem logo em seguida: **git init**



Obs.: Dentro da pasta monitorada pelo GIT é criada uma sub-pasta chamada “.git”, veja:



Em destaque:

1-) A pasta que foi versionalizada;

2-) A pasta oculta de controle de repositório.

Observação: Para desfazer o “git init” basta apagar a pasta “.git”. Com isso o GIT sessa o traquemento.

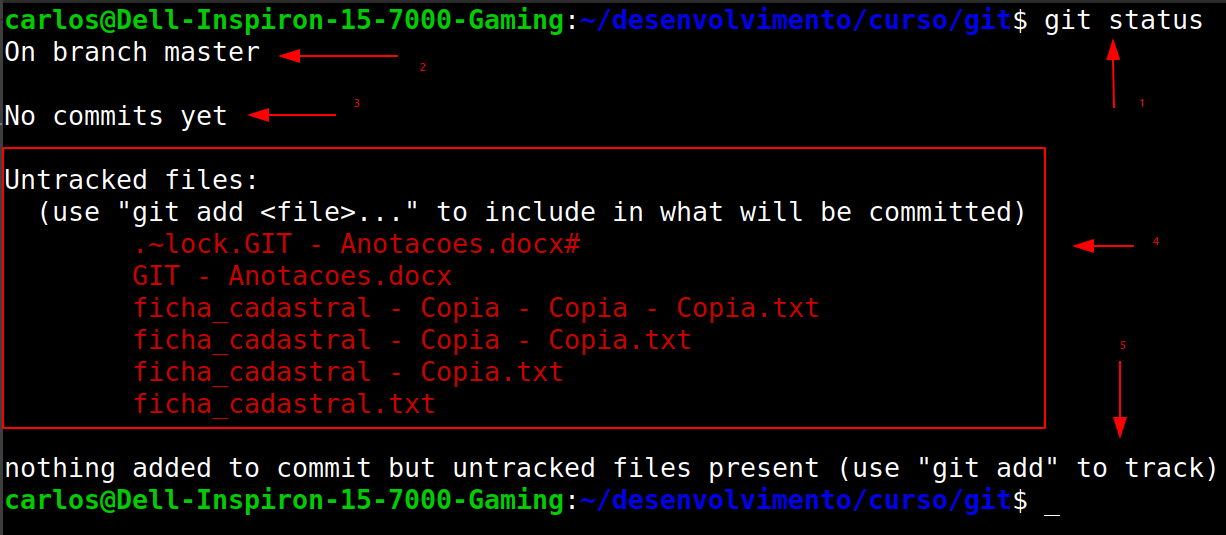
-

**GIT STATUS**

Mostra a situação dos arquivos da pasta que está sendo monitorada pelo GIT.

Sintaxe: **git status**

Exemplo:



Explicação das exibições:

1-) O comando;

2-) Nome da branch;

Essas branches são dividicas assim:

a-) Desenvolvimento;

b-) Homologação;

c-) Produção – no caso a “master” é essa.

3-) Não foram feitos ainda nenhum commit;

4-) Arquivo não traqueados (em fonte vermelha) e entre parênteses – uma dica – para usar o comando “add” para iniciar o traqueamento;

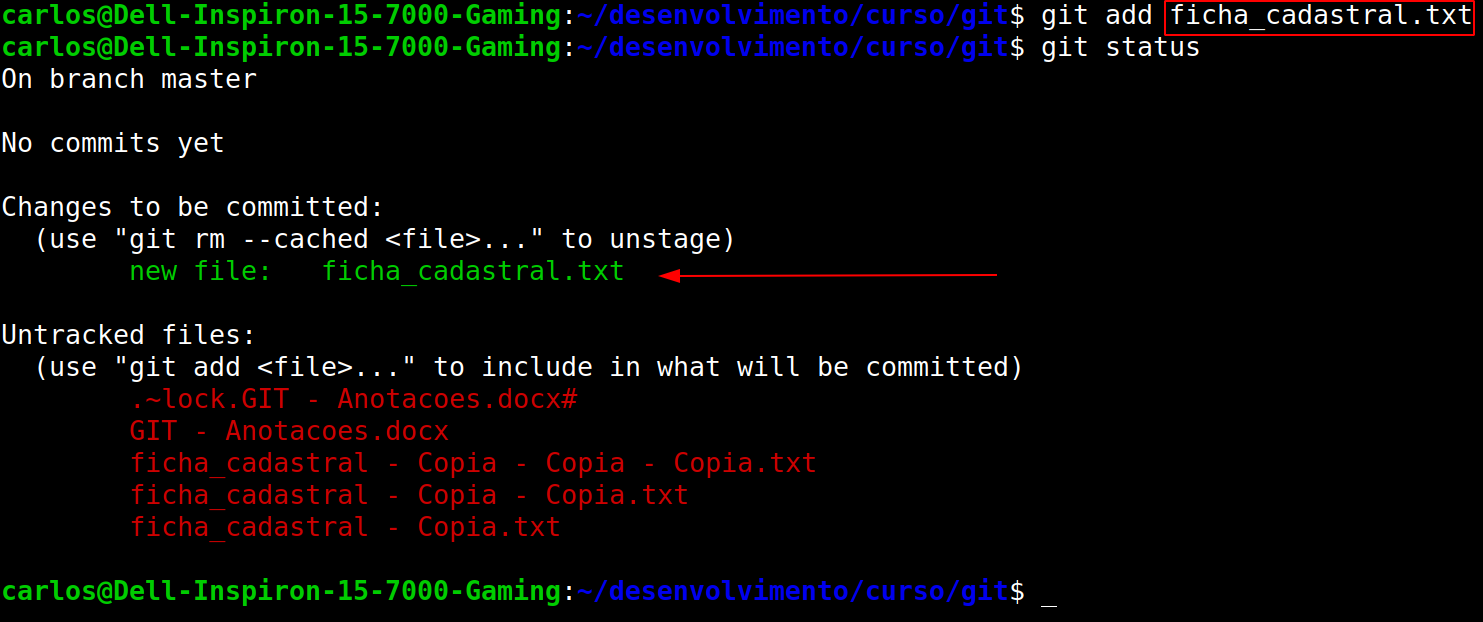
5-) Dica para iniciar o trackeamento dos arquivos.

-

**INICIANDO O TRACKING – “git add”...**

Um arquivo específico: **GIT ADD <NOME\_DO\_ARQUIVO>**

Exemplo: **git add ficha\_cadastral.txt**



No destaque estão:

- O nome do arquivo que vai ser monitorado, no caso é o “ficha\_cadastral.txt”;

- Depis do “git status”, a seta mostra que o nome dele ficou na cor “verde”, indicando que está sendo monitorado.

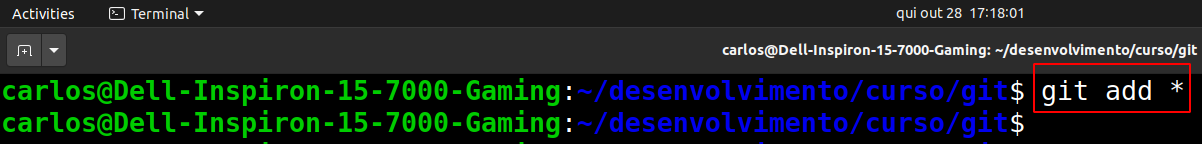
- Observar os demais arquivos com os nomes com as fontes em vermelho, indicando que não estão sendo monitorados.

-

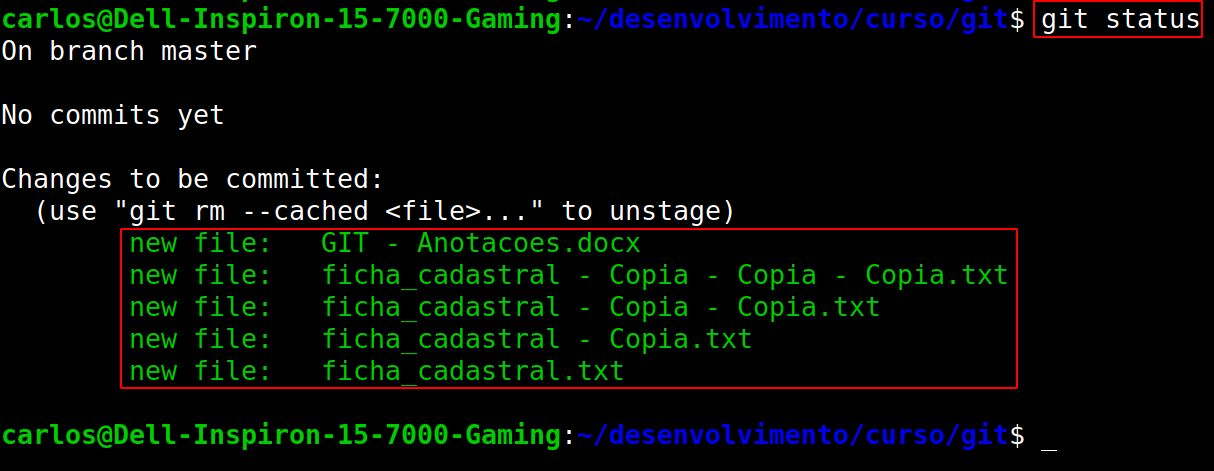
**Adicionando todos arquivos da pasta de uma só vez:**

**GIT ADD \***

Exemplo:



Usando o “git status” para ver a situação do monitoramento:



Observar que todos os arquivos da pasta estão com a fonte em verde – sinal que todos estão monitorados.

Ponto de atenção: Se no momento do “git add \*” algum arquivo da pasta estiver sendo editado, ele não será incluído na monitoração, logo, recomenda-se que todos arquivos da pasta estejam fechados, antes do comando. Estando um arquivo traqueado pode se dizer que ele está na área de STAGE.

-

**Removendo arquivos do monitoramento:**

**Um específico: GIT RM –CACHED <NOME\_DO\_ARQUIVO>**

Exemplo:

***git rm --chached ficha\_cadastral.txt***

**Todos da pasta: GIT RM –CACHED \***

Exemplo:

***git rm --chached \****

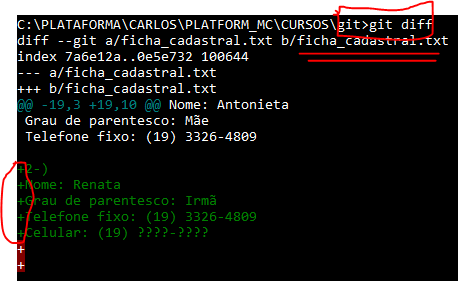
-

**GIT DIFF**

Mostra as diferenças nos arquivos que foram modificados.

Dica: Usar esse comando antes de commitar.

Exemplo:



No exemplo vemos:

- o comando;

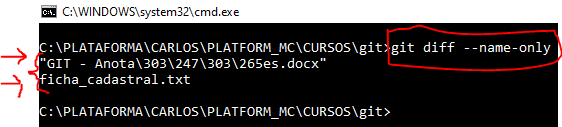
- o arquivo modificado;

- o conteúdo que no caso foi adicionado (ver que tem um sinal de “+”, indicando que foi acrescentado algo no arquivo).

**GIT DIFF --NAME-ONLY**

Vendo apenas os **nomes** dos arquivos modificados.

Exemplo:



No exemplo vemos:

- O comando;

- Os nomes dos arquivos modificados – no caso 2 arquivos foram modificados.

-

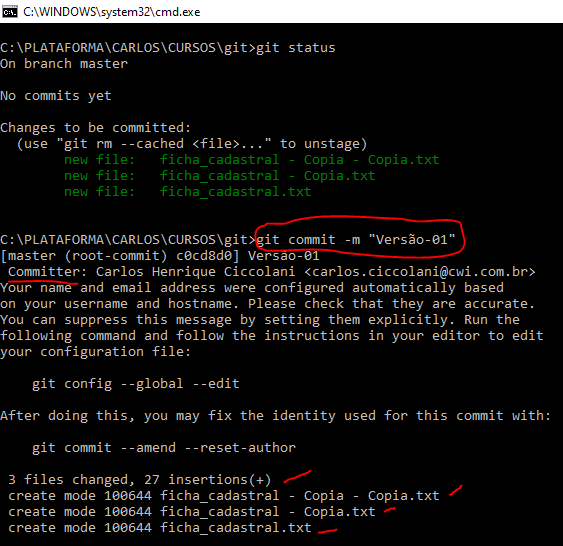
**GIT COMMIT –M “MENSAGEM\_IDENTIFICADORA\_DA\_VERSÃO”**

Gerando versões – **COMMIT**

Documentação: https://git-scm.com/docs/git-commit

Pré-requisito: Usar “**add**” antes, porque sem esse comando não tem como criar versão.

Criando uma primeira versão:



Para gerar outra versão do repositório:

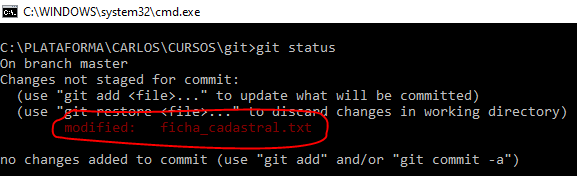
Sempre que arquivos foram modificados ou adicionados, executar:

git status

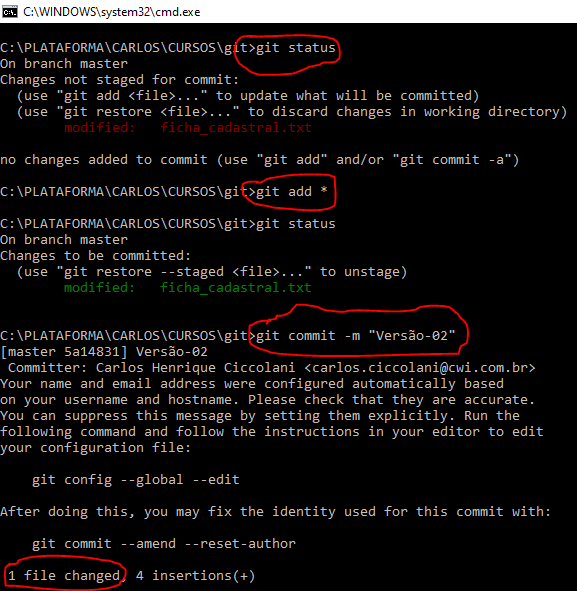
git add

git commit

Veja a situação depois de ter modificado um arquivo:



Gerando a segunda versão:



Quando for comitar arquivos que já existiam, pode usar o parâmetro “a”.

Exemplo: **git commit -am “mensagem\_do\_desenvolvedor”**

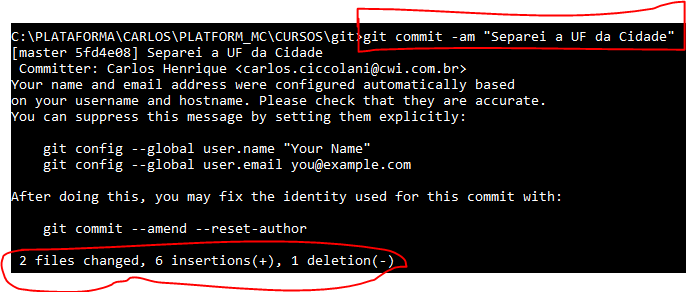
Esse parâmetro afeta 2 tipos de arquivos:

- Arquivos modificados;

- Arquivos excluídos (o que também não deixa de ser uma modificação).

Nesses 2 casos o parâmetro “a” prepara esses arquivos automaticamente.

*Observação: Isso não inclui os novos arquivos que ainda não foram submetidos ao “add” – estes precisam sim serem incluídos através do comando “****git add \*****”*



-

**GIT LOG**

1-) Mostra os logs de commit do Git em ordem decrescente de data.

Exemplo: **git log**

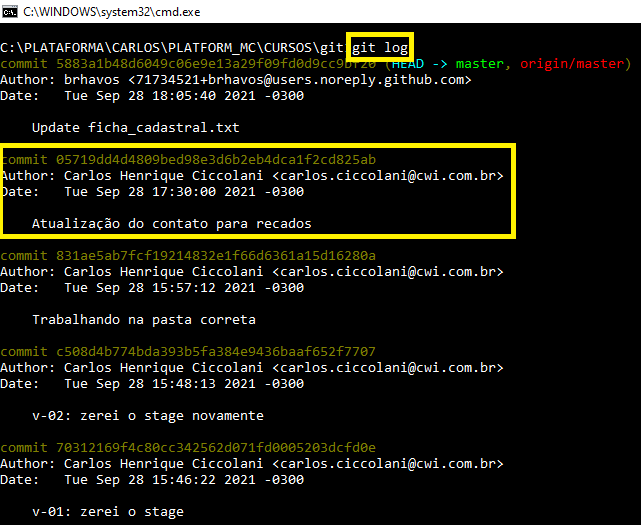
Atributos do commit:

hash,

autor,

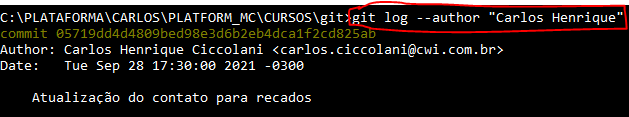
data,

comentário



2-) Filtrando todos os commits de um “author”

Exemplo: **git log - - author “Carlos Henrique”**



3-) Resumo dos commits

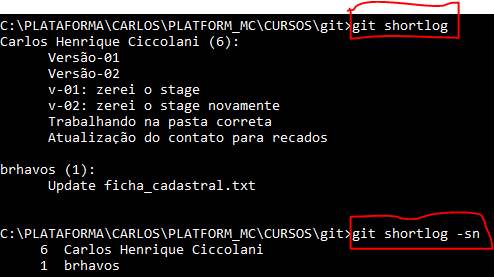
Mostra os commits em forma de sumário. Nesse caso temos 2 opções:

- Um com as mensagens.

Exemplo: **git shortlog**

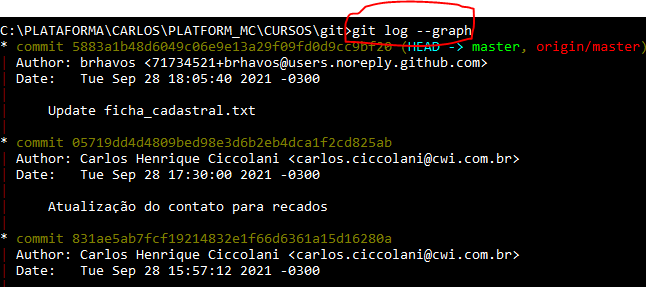
- E outro apenas com as quantidades de commits e o nome

Exemplo: **git shortlog -sn**



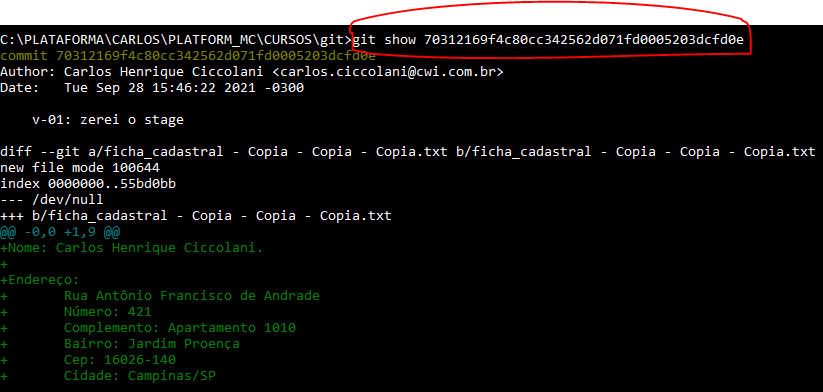
4-) Forma gráfica

Exemplo: **git log –graph**



5-) Vendo um hash específico

Exemplo: **git show <número do hash>**



-

**GIT REMOTE ADD ORIGIN <NOME\_DO\_REPOSITÓRIO\_REMOTO\_NO\_GITHUB>**

Configurando o servidor remoto (repositório remoto na nuvem).

Documentação: ***https://git-scm.com/docs/git-remote***

Esse comando permite assossiar a pasta monitorada do GIT na estação local (repositório local) a um repositório na nuvem.

**Pré-Requisitos:**

- Usar o comando “git remote” antes de tudo. Se já existir não precisa fazer novamente;

- É necessário que exista um repositório remoto.

No caso, usarei o GitHub

Dados da conta no GIT:

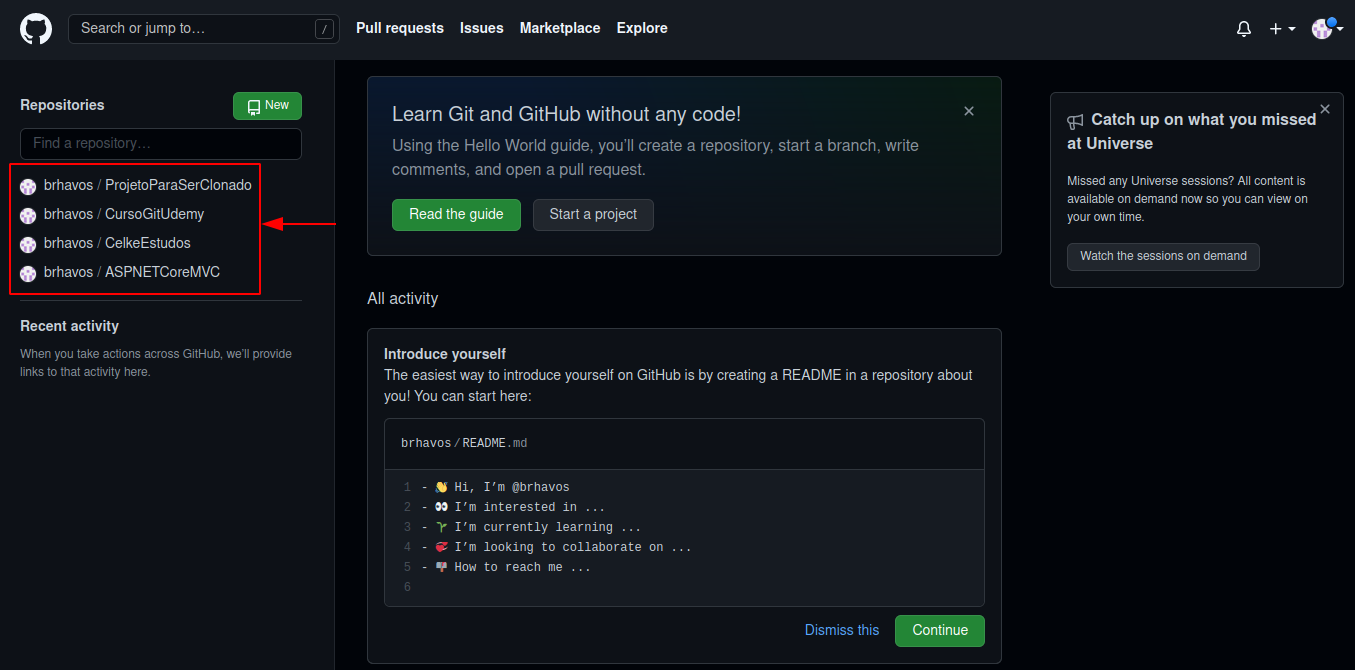
url: [www.github.com](http://www.github.com/)

escolher a opção **sign-in**

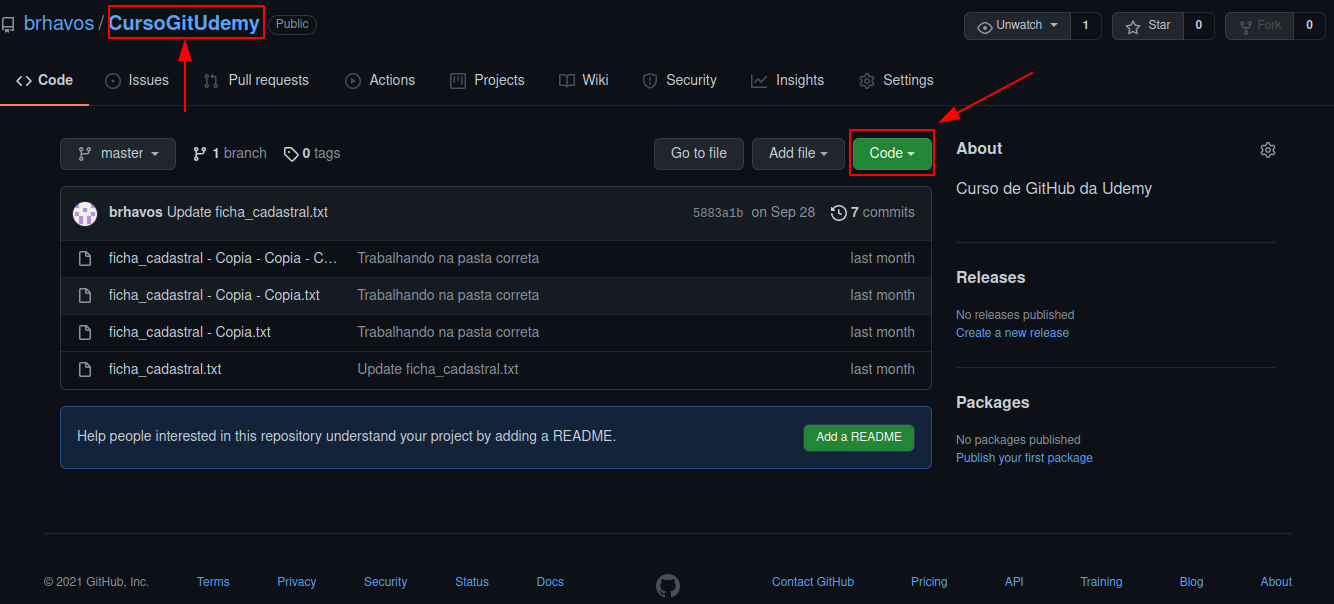
user: brhavos ou e-mail: [brhavos@gmail.com](mailto:brhavos@gmail.com)

password: *consultar arquivo de credenciais no Google Drive*.

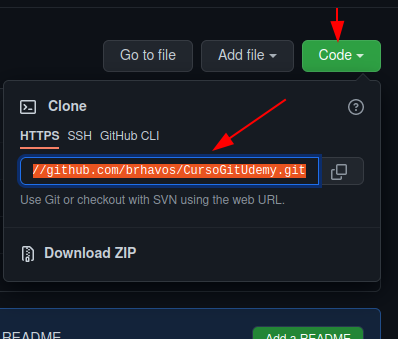
Ao acessar o Github aparecerá a seguinte tela – observar no destaque, em vermelho, que no canto superior esquerdo da tela aparecem os repositórios já criados. Caso queira criar um novo repositório, utilizar o botão verde chamado “**NEW**”.



No caso vou usar um dos repositórios já existentes chamado “**CursoGitUdemy**” (no menu à direita é a opção “brhavos / CursoGitUdemy”), clicando nessa opção vai aparecer a seguinte tela – observar os 2 destaques que indicam o nome do repositório selecionado e o botão verde que precisaremos clicar em seguida:



Clicando no botão verde, chamado “**Code**”, vai disponibilizar o nome do repositório remoto que deve ser informado no comando, veja:



O endereço do repositório remoto é o seguinte: **https://github.com/brhavos/CursoGitUdemy.git**

Agora, com todas as informações à mão, digita finalmente o comando abaixo na pasta monitorada:

**git remote add origin https://github.com/brhavos/CursoGitUdemy.git**

Exemplo:



Em destaque está o endereço do repositório remoto.

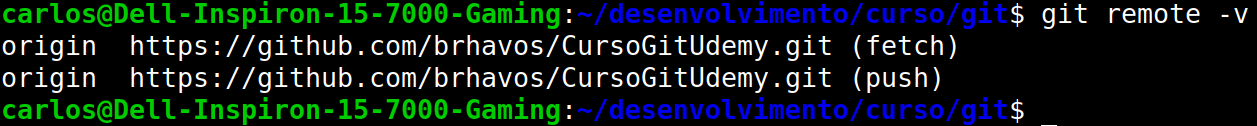
-

**GIT REMOTE –V**

Após executado o comando anterior, agora é o momento de verificar se a configuração está correta.

Esse comando mostra o nome do repositório remoto da pasta monitorada.

Exemplo: **git remote -v**



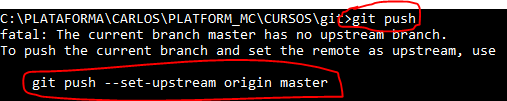
Mostra caminho para recuperação de conteúdo do repositório remoto (fetch/pull);

Mostra caminho para envio de conteúdo do repositório remoto (push).

**GIT PUSH**

Enviando conteúdo do repositório local para o repositório remoto.

**PRIMEIRA VEZ NO DIA:**



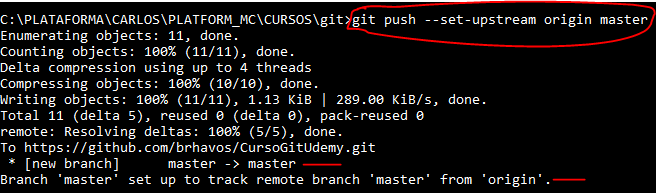
**git push -u <nome\_do\_repositório\_de\_destino> < nome\_do\_repositório\_de\_origem>**

**Exemplo: git push –u origin master**

**git push –-set-upstream origin master**

**Ou**

**git push –-set-upstream origin master -f**



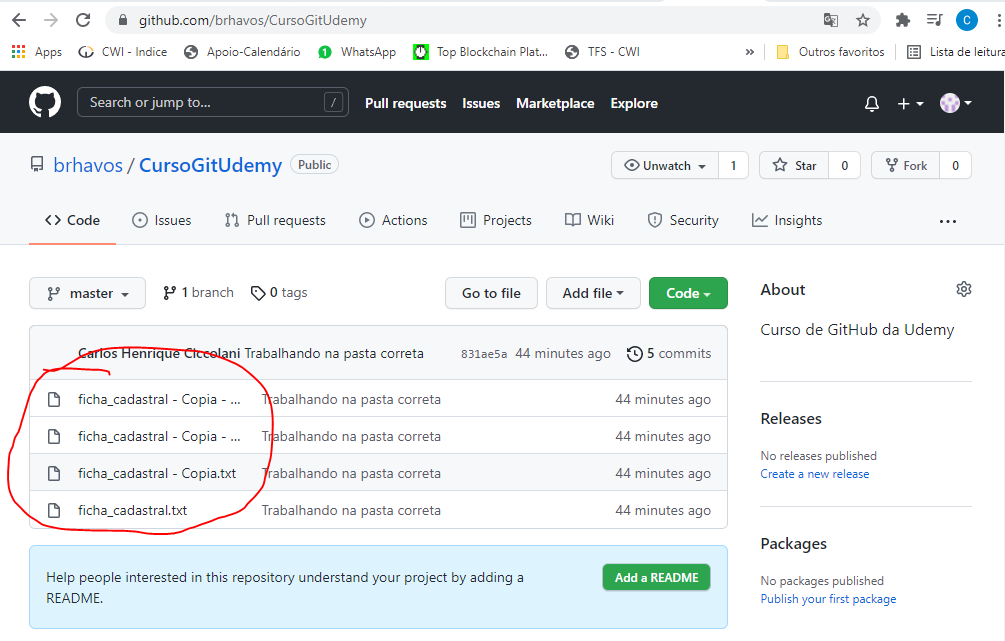
Sincronizando o conteúdo local com o servidor remoto.

Obs:

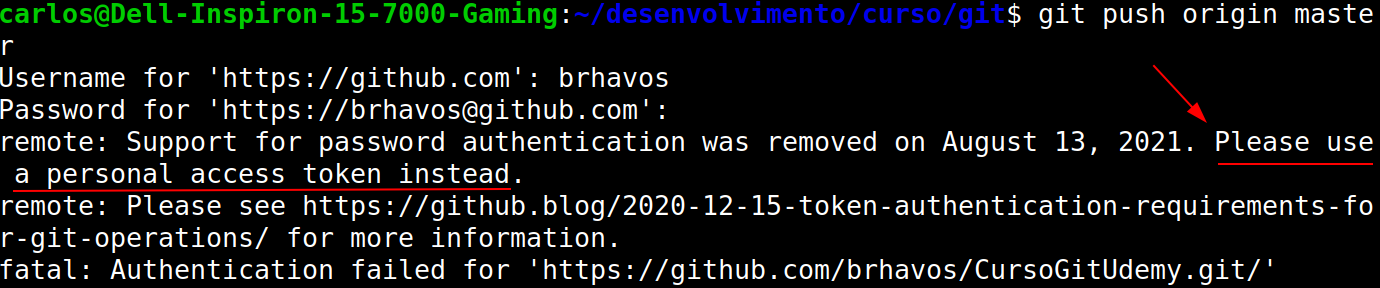
-origin: repositório remoto

-master: repositório local

... enquanto isso, no servidor, veja (obs.: atualize a página):



Observação: Se a autenticação falhar e solicitar o tokem aparecerá uma mensagem assim:



Caso não possua um token pessoal, seguem os passos para criá-lo:

DICA – Assista este vídeo: https://www.youtube.com/watch?v=yTkb3VtNjBU

Agora, siga os seguintes passos (descritos no vídeo acima):

vai no git

profile

settings

developer setings

personal access tokens

generate new token

note: descrição qualquer

expiration

marca todas as permissões

generate token

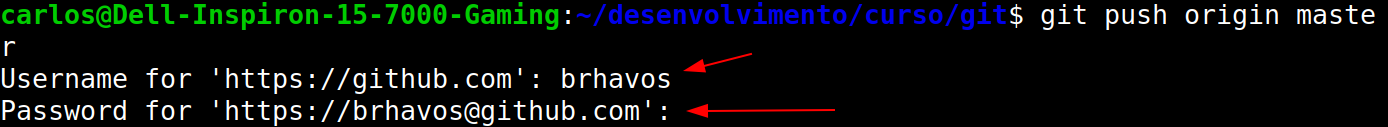
copia o número do token e guarda em local seguro;

Se perder pode gerar novamente, nesse caso será gerado um número diferente do anterior que funcionará normalmente.

>> No caso, esse foi o token gerado: **ghp\_AxX3Ks436ZU1SncAYw32XP54oatyBW3XGnH3**

- - - - -

Isto posto, devemos repetir a tentativa de push:



Username: é o do GitHub, no caso **brhavos**

Password: é o token

No caso: **ghp\_AxX3Ks436ZU1SncAYw32XP54oatyBW3XGnH3**

Obs.: Pode utilizar o sistema de copiar/colar

-

**MEMORIZAÇÃO DAS CREDENCIAIS**

Para memorizar as credenciais de push, fazer:

**git config --global credential.helper cache**

Como funciona isso?

Da segunda vez em diante que solicitar um push não será solicitada a credencial

Exemplo:

>>> Primeira vez:

git push origin master

>> Informa o login do git;

>> Informa o token

>>> Segunda vez e demais:

git push origin master

---------->>>>>>> Observação: Não vai solicitar as credenciais.

Para retirar a memorização do cache e fazer com que sejam solicitadas as credenciais novamente,

Fazer assim:

**git config --golbal --unset credential.helper**

>> A partir de agora sempre vai pedir as credencias (login do git e número do token).

- - - - -

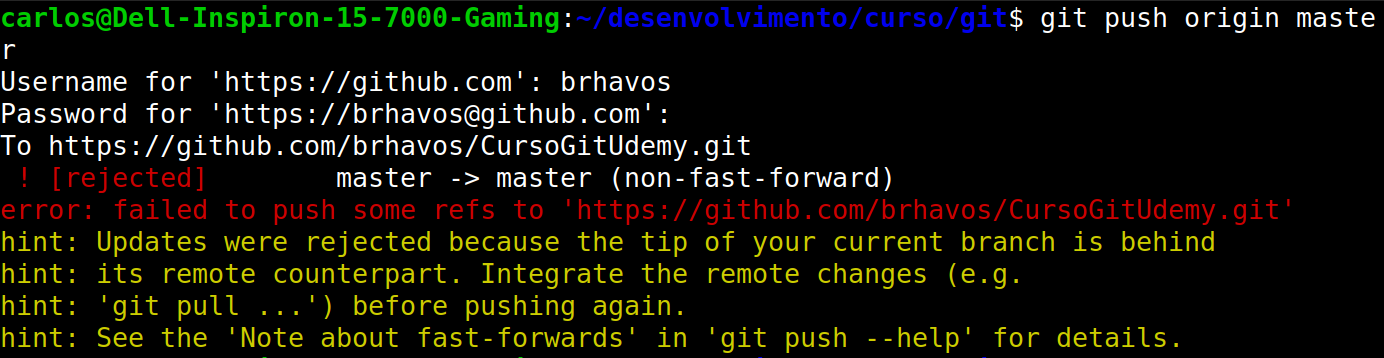
**PROBLEMAS QUE PODEM OCORRER**

Antes de prosseguir, veja esse fórum com a descrição de uma jornada problemática:

*https://cursos.alura.com.br/forum/topico-nao-consigo-dar-um-push-no-meu-projeto-git-45617*

Agora segue minha experiência pessoal:

Após tentar o “push” apareceu o seguinte:



**\*\*\* Isto quer dizer que antes do push precisa fazer um pull**

Digita o seguinte comando:

**git pull origin master**

Se ainda assim estiver dando problema, executa o seguinte comando:

**git pull origin master --allow-unrelated-histories**

Se ainda der problema é porque precisa commitar, então usa o seguinte comando:

**git add \***

**git commit -am “mensagem…”**

Agora executa o push:

**git push origin master**

Ponto de atenção: Após fazer um “push” é prudente verificar o repositório remoto.

- - - - -

**Quando um arquivo for:**

**- Criado;**

**- Alterado;**

**- Excluído;**

**Executar a seguinte sequência:**

**Na máquina local:**

**git status**

**git add \* (fazer esse comento somente se houver algum arquivo em vermelho)**

**git status**

**git commit –am “minha\_mensagem” (somente se não houver nenhum arquivo em vermelho)**

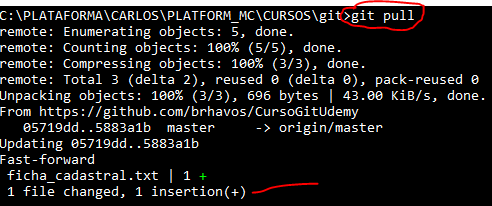
**git status**

**git push (somente isso. Não precisa fazer todo o processo da primeira vez)**

**Ao final, verifica no repositório remoto.**

**GIT PULL**

Para atualizar o repositório local a partir do servidor remoto (pull = pegando conteúdo).



Se der problema, utilize:

**git pull origin master**

Se ainda der problema, utilize:

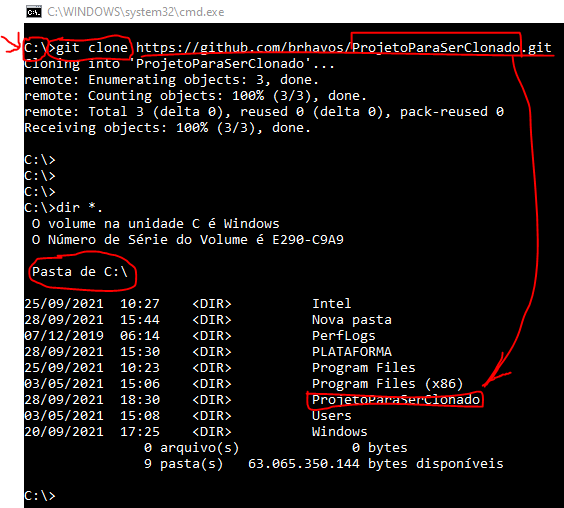
**git pull origin master –allow-unrelated-histories**

**GIT CLONE <ENDEREÇO\_DO\_PROJETO\_NO\_SERVIDOR\_REMOTO>**

**Obs.: Não colocar na pasta onde já tem controle do GIT.**

**Obs.: Sugestão – ir na pasta “raiz”**

Clonando (baixando) um projeto do servidor remoto para a máquina local.



**Próximas alterações ou implementações:**

**Na máquina local:**

**git status**

**git add \***

**got status**

**git commit –am “xxxxx”**

**git status**

**git push (somente isso. Não precisa fazer todo o processo da primeira vez)**

Desfazendo as coisas (voltando conteúdo dos arquivos modificados)

Supor que um determinado arquivo foi modificado.

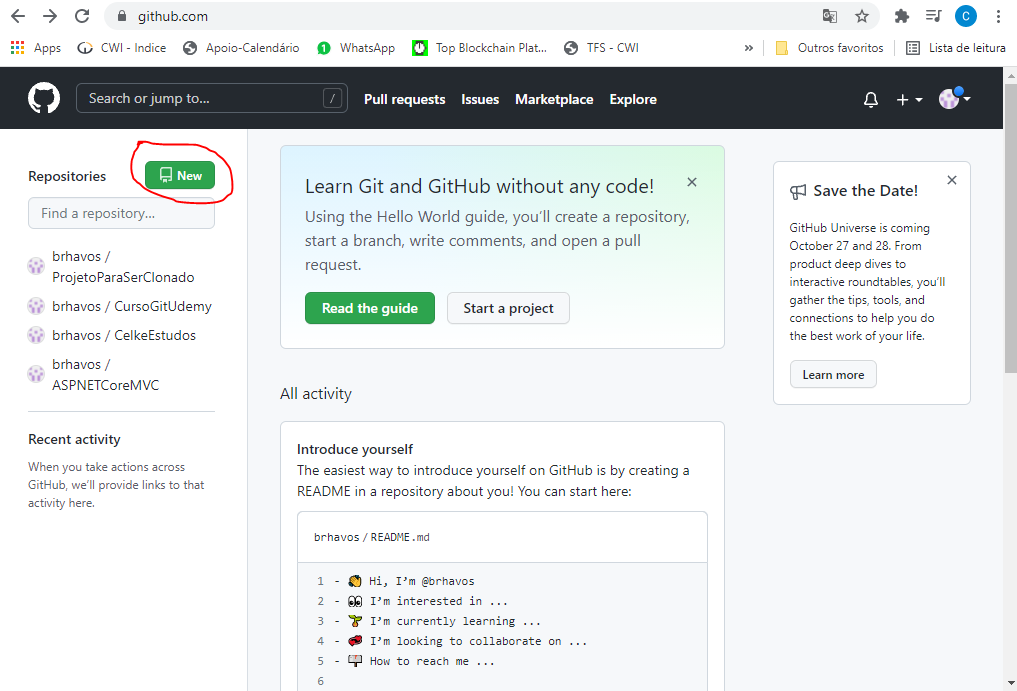
Posso desfazer as modificações dentro de 3 cenários:

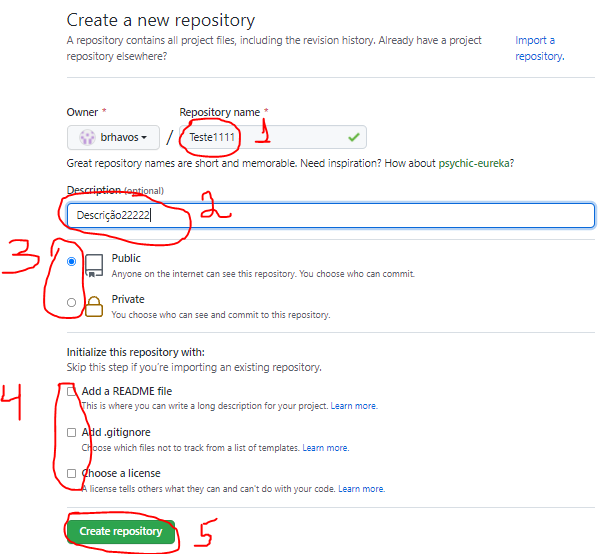
1. Antes do “add” (não está na área de stage)  **GIT CHECKOUT <NOME\_DO\_ARQUIVO>**
   1. Exemplo: **git checkout carta.txt**
2. Depois do “add” e antes do “commit” (está na área de stage)  **GIT RESET HEAD <NOME\_DO\_ARQUIVO>**
   1. Exemplo: **git reset HEAD carta.txt**
3. Depois do “commit”
   1. Soft: deixa no stage (como se tivesse dado um “add”)
      1. **GIT RESET --SOFT <NÚMERO\_DO\_HASH\_DESEJADO\_PARA\_RETORNO>**
         1. Exemplo: **git reset --soft 831ae5ab7fcf19214832e1f66d6361a15d16280a**
   2. Mixed: Coloca um passo antes do stage (como se fosse modificado e antes do “add”)
      1. **GIT RESET --MIXED <NÚMERO\_DO\_HASH\_DESEJADO\_PARA\_RETORNO>**
         1. Exemplo: **git reset --mixed 831ae5ab7fcf19214832e1f66d6361a15d16280a**
   3. Hard: Volta tudo como se nada tivesse feito
      1. **GIT RESET --HARD <NÚMERO\_DO\_HASH\_DESEJADO\_PARA\_RETORNO>**
         1. Exemplo: **git reset – hard 831ae5ab7fcf19214832e1f66d6361a15d16280a**

Obs.: Recomenda-se usar esse comando ANTES do “push”

Obs.: Se já tiver feito o “push”, quando for fazer o novo “push” precisará usar o parâmetro “f”, de “force”

GitHub – Criando um novo repositório:





CRIANDO TOKEN NO GITHUB

Dica: https://stackoverflow.com/questions/68775869/support-for-password-authentication-was-removed-please-use-a-personal-access-to

- - - - -

From August 13, 2021, GitHub is no longer accepting account passwords when authenticating Git operations. You need to add a PAT (Personal Access Token) instead, and you can follow the below method to add a PAT on your system.

Create [P]ersonal [A]ccess [T]oken on GitHub

From your GitHub account,

>>> go to Settings

>>> Developer Settings

>>> Personal Access Token

=> Botão "Generate New Token" (Give your password)

=> Preencha o formulário

=> Clica no botão "Generate token"

=> Faça uma cópia do token que foi gerado (no caso: ghp\_oheosxKMJgHjKt8DTiGjFMz4goZpP720onAD)

Copy the generated Token, it will be something like ghp\_sFhFsSHhTzMDreGRLjmks4Tzuzgthdvfsrta

- - - - -

For Linux, you need to configure the local GIT client with a username and email address,

Para Linux, você precisa configurar o cliente GIT local com um nome de usuário e endereço de e-mail,

git config --global user.name "your\_github\_username"

git config --global user.name "brhavos"

git config --global user.email "your\_github\_email"

git config --global user.email "brhavos@gmail.com"

git config -l

----------------------------------------------

Once GIT is configured, we can begin using it to access GitHub. Example:

Depois que o GIT estiver configurado, podemos começar a usá-lo para acessar o GitHub.

Exemplo:

git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY

> Cloning into `Spoon-Knife`...

$ Username for 'https://github.com' : username

$ Password for 'https://github.com' : give your personal access token here

----------------------------------------------

Now cache the given record in your computer to remembers the token:

Agora armazene em cache o registro fornecido em seu computador para lembrar o token:

git config --global credential.helper cache

----------------------------------------------

If needed, anytime you can delete the cache record by:

Se necessário, a qualquer momento você pode excluir o registro de cache:

git config --global --unset credential.helper

git config --system --unset credential.helper

----------------------------------------------

Now try to pull with -v to verify

Agora tente puxar com -v para verificar

git pull -v

----------------------------------------------

Linux/Debian (Clone as follows):

Linux / Debian (clone da seguinte forma):

git clone https://<tokenhere>@github.com/<user>/<repo>.git

----------------------------------------------

GitHubGitHub

GitHub: Where the world builds software

GitHub is where over 73 million developers shape the future of software, together. Contribute to the open source community, manage your Git repositories, review code like a pro, track bugs and feat... (490 kB)