**GIT / GITHUB – Anotações dos Cursos**

Netlify → pesquisar conhecimento

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 dos Cursos**

**GIT --VERSION**

Finalidade:

- Permite saber se o GIT está instalado;

- Permite saber qual a versão instalada.

Exemplols:



-

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

Finalidade:

- Mostra as configurações globais, por exemplo:

– nome,

- e-mail,

- Editor-padrão,

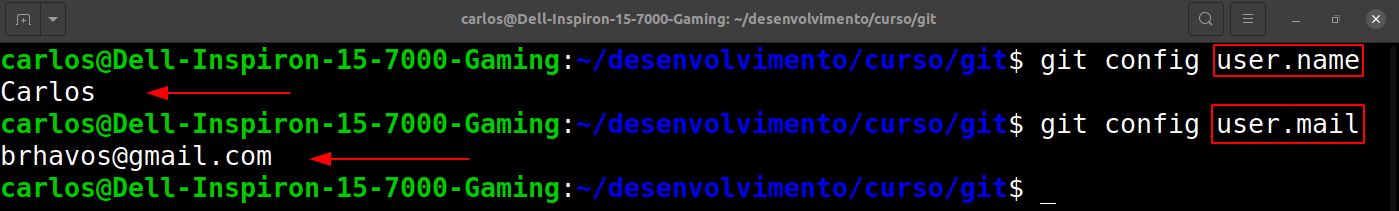
- Etc.

Exemplos:

1-) git config user.name

2-) git config user.mail

3-) git config core.editor



Checando todas as configurações globais que foram definidas:

git config --list



**GIT INIT**

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

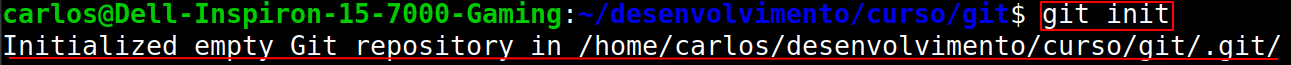
Passos para isto acontecer:

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

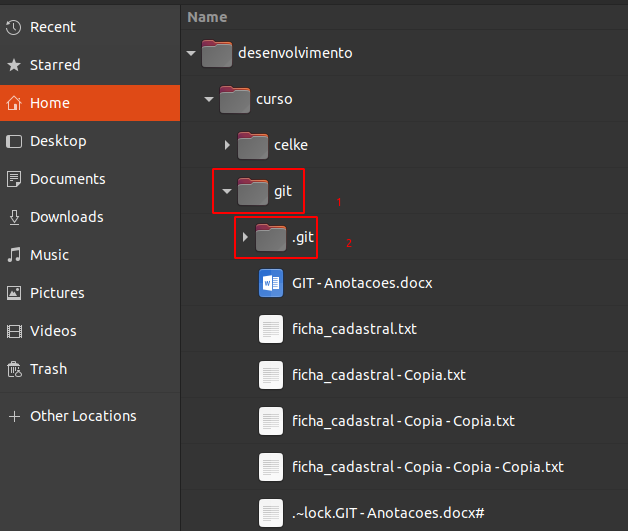
2-) Vai dentro da pasta;

3-) Aplicar comando “git init”.

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



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.

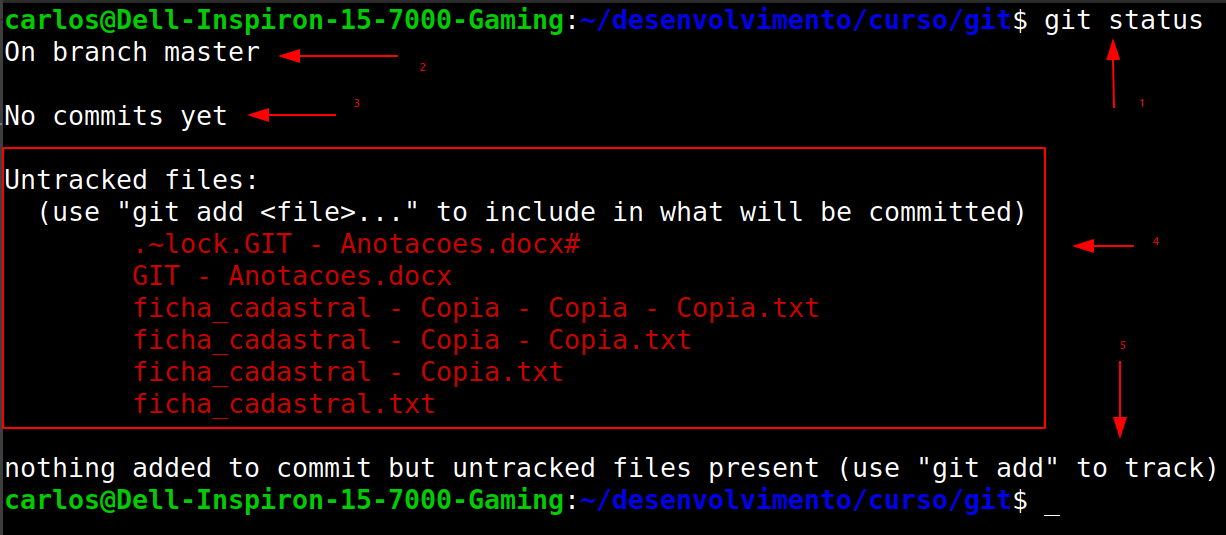
-

**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;

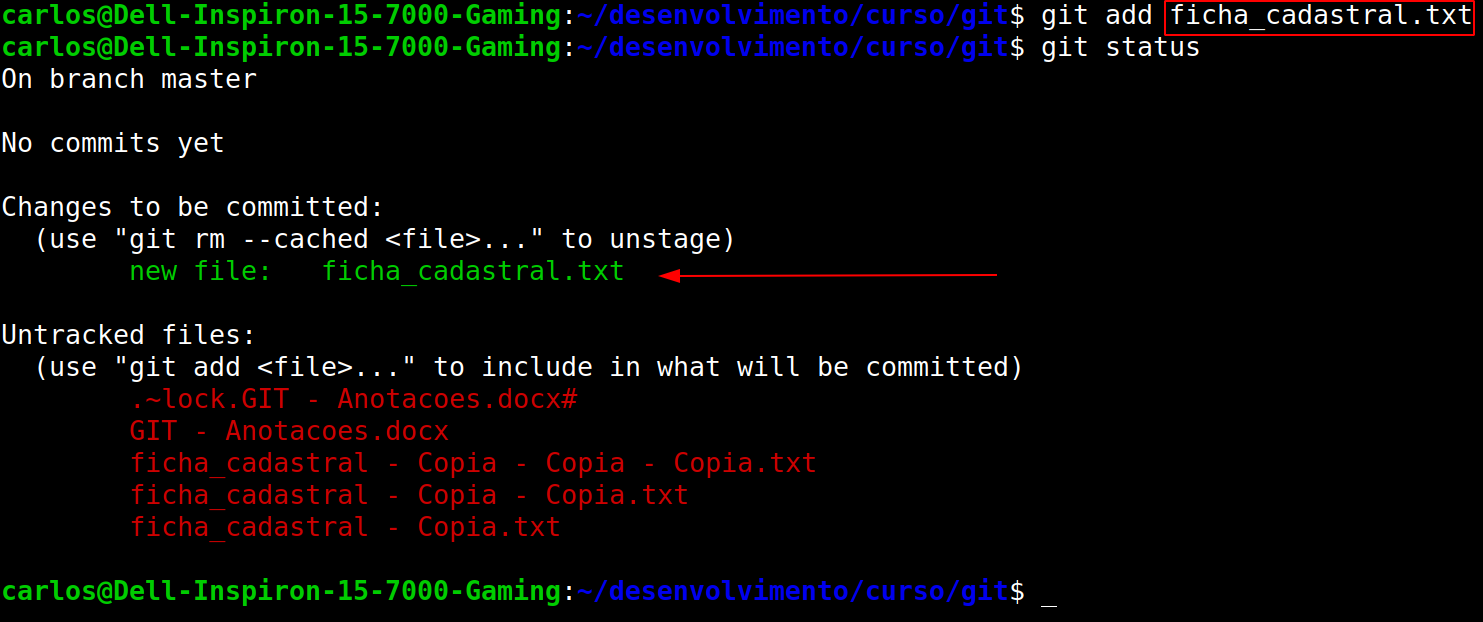
5-) Dica para iniciar o trackeamento dos arquivos.

-

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”;

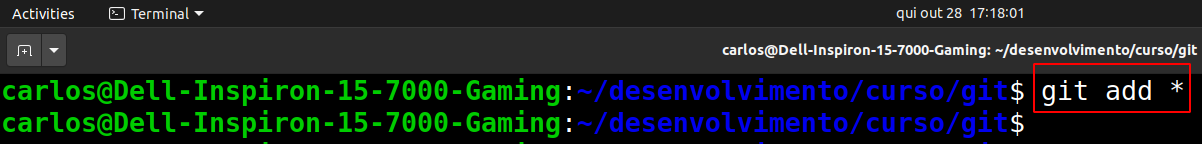
- Depis do “git status”, a seta mostra que o nome dele ficou na cor “verde”, indicando que está sendo monitorado.

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

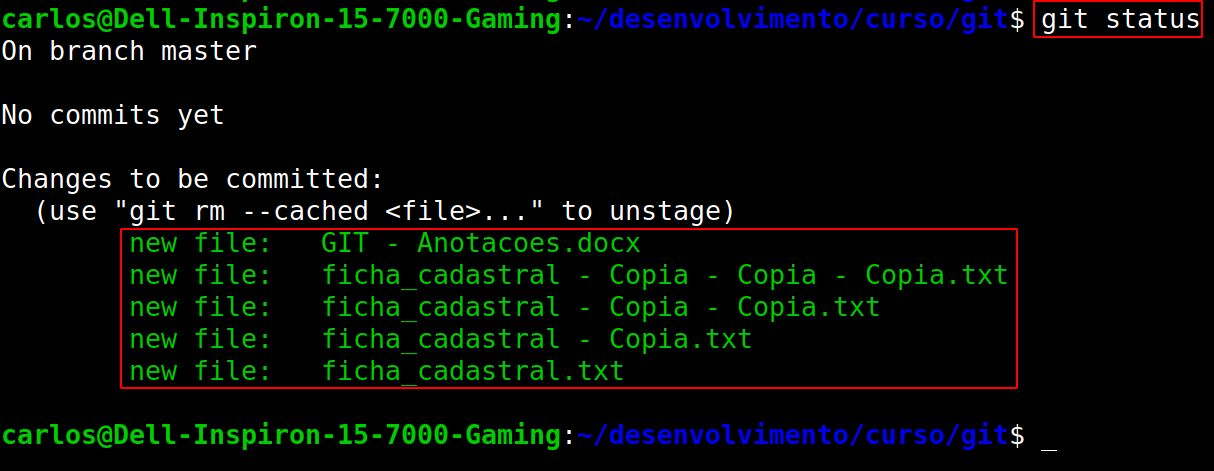
**Adicionando todos arquivos da pasta de uma só vez:**

**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 \**

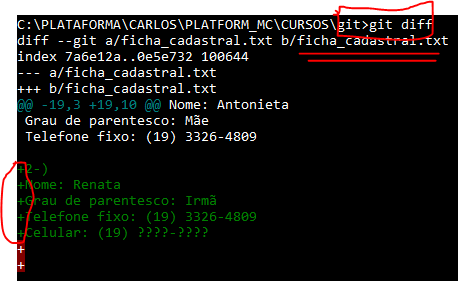
-

**GIT DIFF**

Mostra as diferenças nos arquivos que foram modificados.

Dica: Usar esse comando antes de commitar.

Exemplo:



No exemplo vemos:

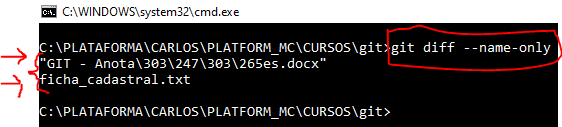
- o comando;

- 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 arquivos modificados.

Exemplo:



No exemplo vemos:

- O comando;

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

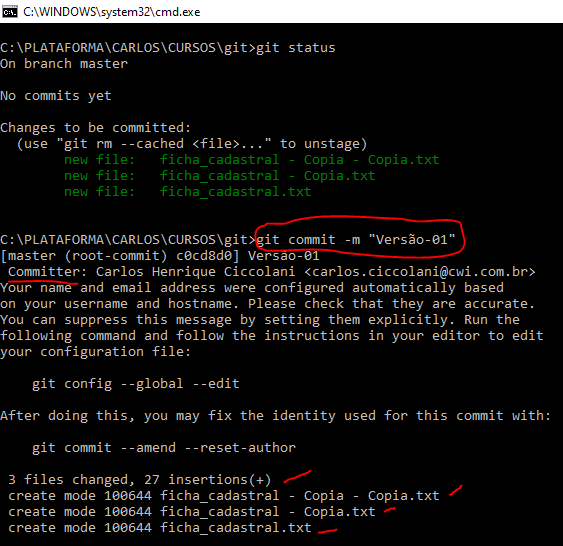
-

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

Gerando versões – **COMMIT**

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

Criando uma primeira versão:



Demais Versões:

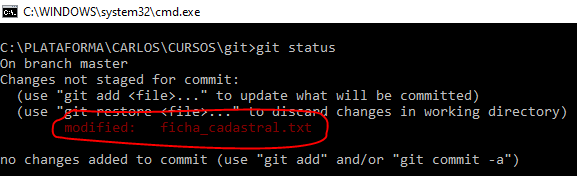
Sempre que arquivos foram modificados ou adicionados, executar:

git status

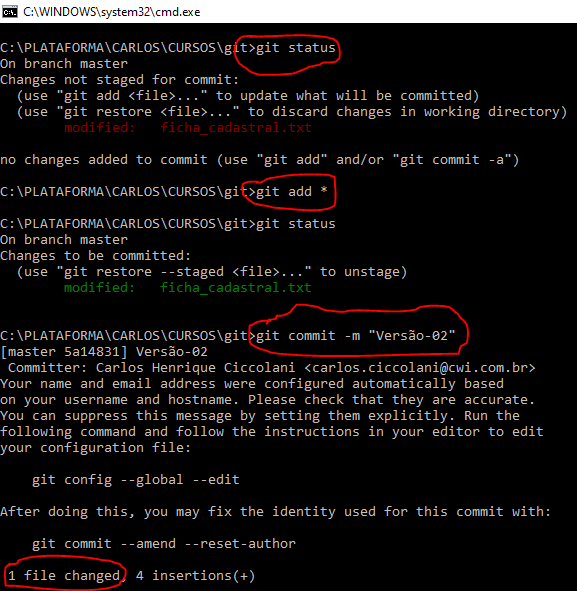
git add

git commit

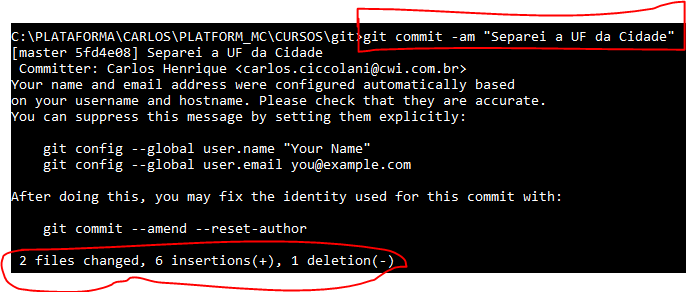
Veja a situação depois de ter modificado um arquivo:



Gerando a segunda versão:



Quando for comitar arquivos que já existiam, pode usar o seguinte comando:



**GIT LOG**

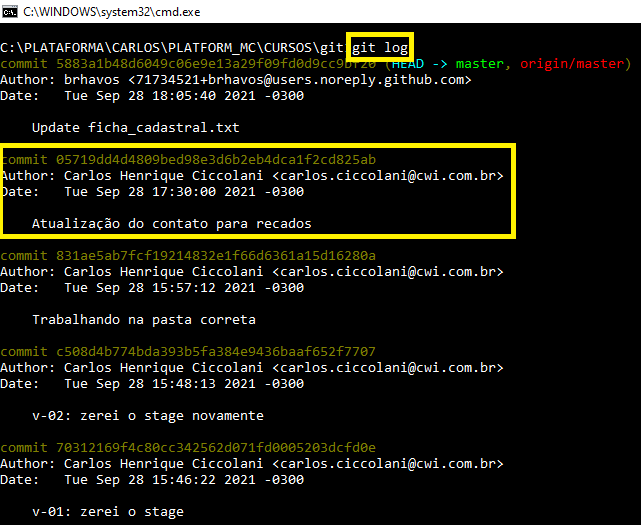
Verificando logs do Git

1-) Vai na pasta inicializada pelo Git;

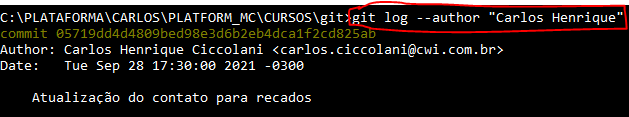
2-) Digite o comando desejado.

Exemplos:

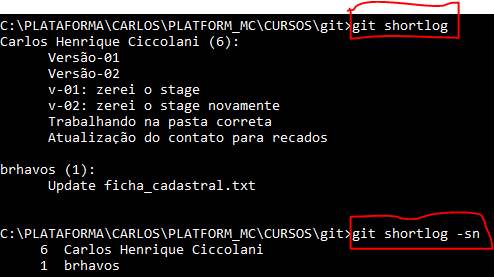
1-) Destacando os dados do commit – hash, autor, data e comentário



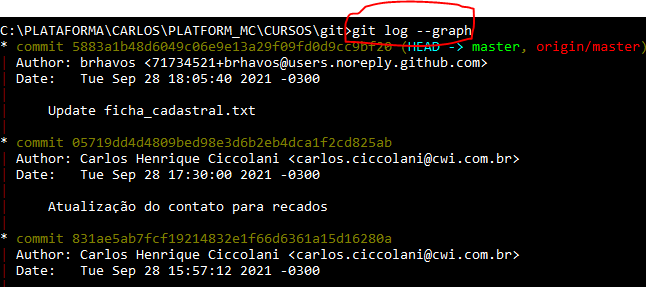
2-) Ações de um determinado usuário:



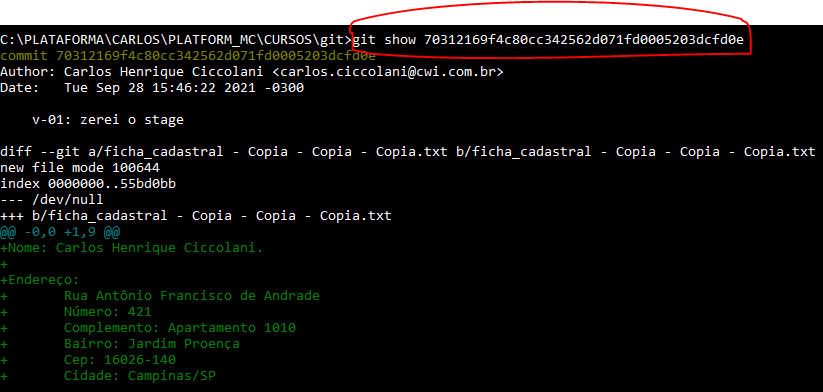
3-) Resumo dos commits



4-) Forma gráfica



5-) Vendo um hash específico



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

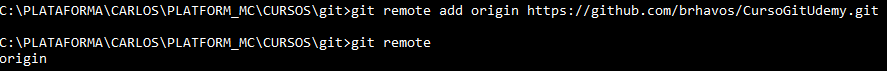
Configurando o servidor remoto (repositório remoto, na nuvem).

Pré-Requisitos:

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

- Existir um repositório remoto  no caso, no GitHub

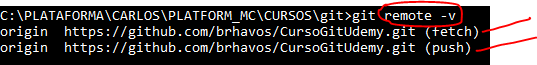
- Fazer apenas uma vez por repositório e pasta



**GIT REMOTE –V**

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

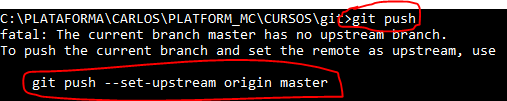
Mostra caminho para envio de conteúdo (push).



**GIT PUSH**

Enviando conteúdo para o servidor remoto

**PRIMEIRA VEZ:**



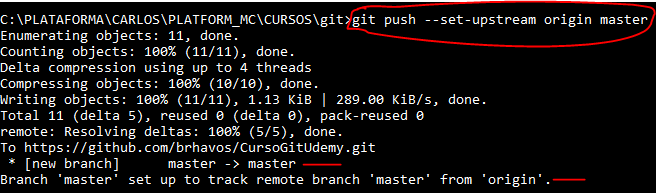
**GIT PUSH –U <nome\_do\_repositório\_de\_destino> < nome\_do\_repositório\_de\_origem>**

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

**GIT PUSH –SET-UPSTREAM ORIGIN MASTER**

**Ou**

**GIT PUSH –SET-UPSTREAM ORIGIN MASTER -F**



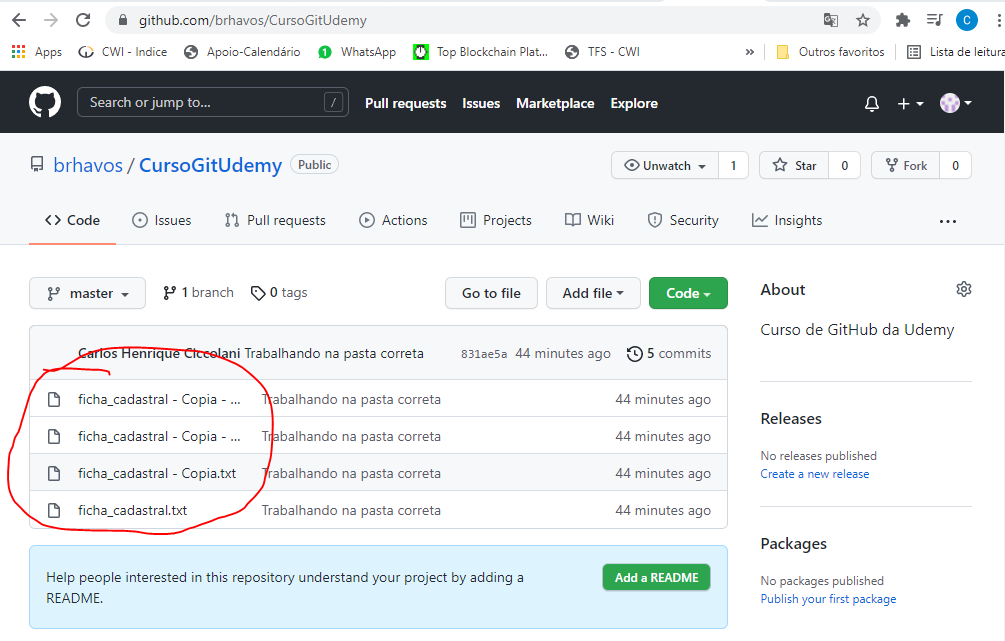
Sincronizando o conteúdo local com o servidor remoto.

Obs:

-origin: destino (repositório remoto)

-master: origem (repositório local)

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



**Próximas alterações ou implementações:**

**Na máquina local:**

**git status**

**git add \***

**got status**

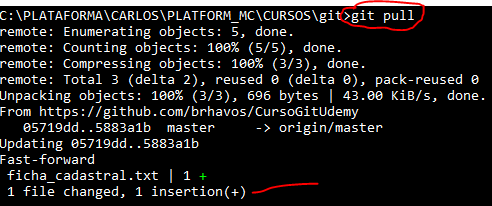
**git commit –m “xxxxx”**

**git status**

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

**GIT PULL**

Para atualizar os trabalhos a partir do servidor remoto (pegando conteúdo do servidor).

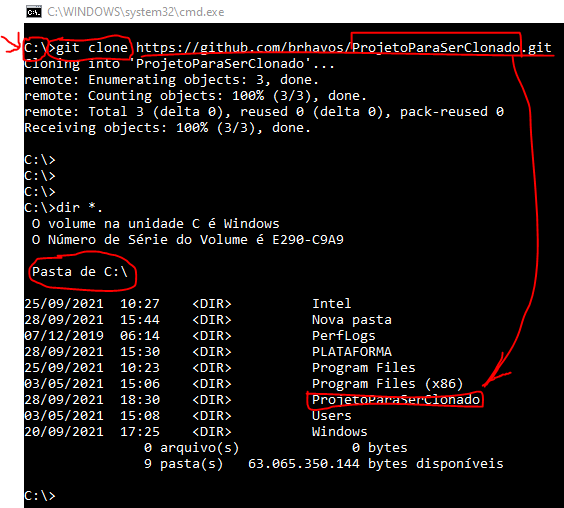


**GIT CLONE <ENDEREÇO\_DO\_PROJETO\_NO\_SERVIDOR\_REMOTO>**

**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 –m “xxxxx”**

**git status**

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

Desfazendo as coisas (voltando conteúdo dos arquivos modificado)

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:

