Programação Cunha

Índice

[GIT e GITHUB 1](#_Toc105849013)

[1. O que é o GIT e o GITHUB 1](#_Toc105849014)

[GIT-> Controlo de Versão 1](#_Toc105849015)

[GITHUB 1](#_Toc105849016)

[2. instalar o git 1](#_Toc105849017)

[Verificar a Instalação 2](#_Toc105849018)

[3. Criar conta no github 2](#_Toc105849019)

[Então vamos criar uma pasta 2](#_Toc105849020)

[4. git status 3](#_Toc105849021)

[5. Agora, vamos incluir este ficheiro no nosso controlo de versão: 3](#_Toc105849022)

[6. COMO CRIAR VERSÕES do codigo com COMMIT: 4](#_Toc105849023)

[7. Enviar o nosso codigo para a nuvem: 5](#_Toc105849024)

[8. Depois de definir para onde o nosso codigo vai ser enviado: 6](#_Toc105849025)

[9. CICRO DE ATUALIZAÇÃO DE CODIGO 7](#_Toc105849026)

[10. COMO VERIFICAR HISTÓRICO DE ATUALIZAÇÕES: 7](#_Toc105849027)

[11. O QUE SÃO BRANCHES E COMO CRIAR UMA: 8](#_Toc105849028)

[12. fazer MERGE 10](#_Toc105849029)

[13. PULL REQUEST UM MERGE DEMOCRÁTICO 12](#_Toc105849030)

[14. git ignore 13](#_Toc105849031)

[15. TODOS OS COMANDOS APRENDIDOS: 14](#_Toc105849032)

# GIT e GITHUB

## O que é o GIT e o GITHUB

### GIT-> Controlo de Versão

Num fluxo normal de trabalho, imaginemos que estámos a trabalhar num projeto

1: Erros de versões:

Se estivermos a trabalhar no nosso codigo e ele de repente parar de funcionar, com o controlo de versão, nós conseguimos voltar atrás e ver a diferença do codigo. Daí, ser mais facil identificar o erro.

2: Trabalho em equipa:

Se nós quisermos trabalhar em equipa sem ter um conflito de arquivos, nós precisamos do sistema de controlo de versão.

Caso o nosso parceiro edite o codigo de maneira errada, nós conseguimos selecionar qual codigo que deve ou não ser aprovado e mantido com o processo chamado "Merge"

ex: Eu quero trabalhar ao mesmo tempo, no mesmo projeto com outra pessoa, sem o sistema de controlo de versão, não iamos conseguir editar o mesmo ficheiro porque ia haver conflito.

Eu fiz asneira e coloquei um codigo totalmente contrario ao que era suposto, nós conseguimos voltar atras, identificar esse erro e selecionar o que queremos ou não.

GITHUB-> Local onde armazenámos o nosso codigo.

1. instalar o git: <https://git-scm.com/downloads>

### Verificar a Instalação

[linha de comandos] git --help

Se aparecer muitos comandos, é porque o git está instalado!

1. Criar conta no github: https://github.com/

Então vamos criar uma pasta e:

dentro da pasta (ainda vazia), carregamos no botao lado direito do rato e em [git bash]

git bash-> é uma linha de comandos do proprio git, onde podemos ver os comandos de forma mais facil.

Agora escrevemos:

git innit

Aparentemente, a nossa pasta está vazia. Para vermos o que foi criado, temos que ativar os ficheiros ocultos. [view-hidden items]

Agora conseguimos ver que uma pasta com o nome ".git" foi criada. Por default esta pasta está oculta, precisamente para ninguem mexer. Não é suposto modificarmos este arquivo manualmente, é criado automaticamente e não precisamos de mexer nele.

Agora criamos um ficheiro (pode ser codigo.txt) com qualquer coisa escrita.

1. git status-> Relatório de quais arquivos foram incluídos no controlo de versão e quais arquivos ainda não estão incluídos no controlo de versão. Tambem podemos ver se um arquivo foi atualizado.

No commits yet: commits são versões do codigo, nós não fizemos nenhuma versão do codigo. (ja vamos ver como é que fazemos isso)

untracked files: Mostra quais arquivos não estão no controlo de versão. (No nosso caso, é o nosso ficheiro codigo.txt)

## Agora, vamos incluir este ficheiro no nosso controlo de versão:

Só temos de escrever no git bash:

git add "codigo.txt" -> (nome do ficheiro que apareceu em cima)

Agora para perceber se está tudo bem, fazemos:

git status (relatorio)

e vamos perceber que aparece:

new file: meu codigo.txt

Isto é uma maneira de adicionar os nossos ficheiros ao controlo de versão,porem, num projeto nós podemos ter muitos ficheiros, então fazer tudo 1 a 1 é um pouco improlífico (improdutivo)

Então, na nossa pasta vamos arrastar para la alguns ficheiros como (.png(imagem), .py(ficheiro python))

Agora voltamos a rodar o comando: git status

Vamos reparar que os ficheiros que movemos, estão "untracked files" e para adicionar todos ao nosso controlo de versão, fazemos:

git add .

fazemos git status para ver se está tudo certo:

E sim. Passamos todos os ficheiros de uma só vez, de uma maneira mais rapida e eficiente para o nosso controlo de versão.

## COMO CRIAR VERSÕES do codigo com COMMIT:

Então com esses arquivos adicionados, vamos criar a primeira versão, escrevendo o seguinte comando:

git commit -m "commit inicial" #-m -> significa mensagem. "" mensagem do que foi feito naquela versão. Neste caso, como é a versão inicial, chamamos de commit inicial

Se estivermos a rodar este comando pela primeira vez, vai-nos ser pedido para configurar o nosso git no nosso pc:

git config --global user.email "cunhanik1@gmail.com"

git config --global user.name "Cláudio Cunha"

nós colocamos o nosso nome porque quando enviamos estas versões, o nosso nome vai estar junto a essas versões.

GIT CONFIGURADO.

Agora escrevemos o comando novamente:

git commit -m "commit inicial"

Agora diz-nos que:

criamos uma nova versão do codigo(novo commit), com o nome "commit inicial"

nome da brench (MASTER) para onde estámos a enviar as nossas alterações

arquivos adicionados

## Enviar o nosso codigo para a nuvem:

comando: git push

Vamos perceber que escrever só este comando vai dar erro, porque o git não sabe para onde vai fazer o push, ou seja, para onde enviar estas alterações. Para enviarmos o nosso codigo para a nuvem, precisamos de definir para onde enviar essas alterações. Neste caso, temos que definir o link do nosso projeto na nuvem.

Então vamos agora criar o projeto no github.

[+(canto superior direito,ao lado do icon) -> new repository] Agora damos o nome do nosso repositorio, ex: projeto1 , podemos opcionalmente dar uma descrição, publico ou privado: publico -> Caso seja para montar um portfolio que vai ser visto por toda a gente, para mostrar o que sabemos e provar que temos conhecimento. privado -> Caso seja um projeto pessoal com informações sensiveis, ou para vender, ou com intençoes de num futuro vender. Add a README FILE -> arquivo readme é um arquivo onde vai ter instruções de como as pessoas devem utilizar o nosso repositorio. (devemos fazer sempre)

feito isto, vamos agora copiar o link do nosso repositorio (acabado de criar) e escrever no git bash:

git remote add origin ColarOLink

Se tivermos um erro do genero: "remote origin already exists."

temos que fazer: git remote set-url origin ColarOLink

## Depois de definir para onde o nosso codigo vai ser enviado:

Se fizermos agora o: git push

Vamos perceber que vai dar erro porque nós não definimos para qual brench será enviado o nosso codigo

Inicialmente enviamo sempre para o brench master

git push --set-upstream origin master

fazendo isto, é muito provável que se abra uma janela no nosso ambiente de trabalho do github para nos connectarmos.

fazemos o nosso login

CONLUIDO ISTO, PODEMOS PERCEBER QUE O NOSSO PROJETO FOI ADICIONADO.

1. CICRO DE ATUALIZAÇÃO DE CODIGO:

A primeira versão do codigo está feita. Nós podemos criar várias versões, isso é um grande beneficio.

Agora, se fizermos uma alteração num dos nossos ficheiros, escrevemos:

git status

Isto diz-nos que um arquivo foi modificado e diz-nos o nome do arquivo.

Então repetimos o mesmo processo

git add .

Para confirmar que foi adicionado, fazemos:

git status

foi adicionado e está pronto para ser enviado para a proxima versão, e para fazer isso é so fazer:

git commit -m "permite registar usuarios" #A mensagem que vamos escrever é o que foi adicionado nesta nova versão. ex: registar novos usuarios

Podemos reparar que foi feito essa adição e agora podemos enviar as atualizações com:

git push

Se formos ao nosso repositorio, [nosso repositorio->brench master->nome do ficheiro que alteramos] podemos ver que o codigo foi alterado com sucesso.

## COMO VERIFICAR HISTÓRICO DE ATUALIZAÇÕES:

Com o comando:

git reflog

Nós conseguimos ver as versões que foram adicionadas até ao momento. A versão atual vai ser sempre a primeira, e assim por ordem. Ou seja, a ultima versão vai ser a versão com que nós começamos "commit inicial"

Podemos reparar que há um "id" antes das nossas versões.

Esse id serve para identificarmos as versões.

Caso haja algum erro com a nova versão e alguem queira voltar a uma versão anterior, fazemos:

git reset --hard id #Colamos o id

Agora se formos ao nosso arquivo, vamos ver que voltou à versão anterior

git reflog

Vai aparecer 3 versões, mesmo as que andamos para tras ou para a frente, aparece tudo.

Se quisermos voltar outra vez para a versão que estavámos, é so fazer a mesma coisa:

git reset --hard id

## O QUE SÃO BRANCHES E COMO CRIAR UMA:

BRANCH(ramo)

Podemos ter varias branch, mas só uma delas é que vai ser a Master(principal).

Branch master é aquela que recebe sempre todas as nossas alterações/versões.

Imaginemos que estamos a trabalhar com uma equipa:

Nós vamos querer trabalhar com outras branchs e manter a branch master estável !!

branch master é so codigo que funciona

branches paralelas é onde o desenvolvimento é feito e só depois de termos a certeza que o codigo está a funcionar, é que passamos esse codigo novo para a branch master. Com um merge

Nome do processo: MERGE

Então,

git branch

Aqui podemos ver quais branches estão disponíveis no momento, neste caso só temos a branch master

Para criar a paralela fazemos:

git branch staging #Nome de uma branch que está a receber atualizações que não estão testadas em produção. Podemos tambem dar o nome da funcionalidade que vamos criar.

git branch

Para ver quais branchs temos e conseguimos ver que a branch master está a verde e isso é porque a branch master é a que estámos a trabalhar no momento

Para mudar de branch, fazemos:

git checkout staging

Vai aparecer-nos "switch to branch 'staging'", agora estamos a trabalhar na branch staging e se quisermos ver, é so fazer:

git branch

a branch staging vai estar a verde

Agora se mudarmos o nosso codigo de um ficheiro qualquer e fizermos:

git status

Vai aparecer que um arquivo foi modificado e diz-nos o nome do arquivo. Para adicionar essas modificações, fazemos:

git add .

Depois um:

git commit -m "adicionada x modificação"

git status

ver se está tudo ok

git push

SEMPRE QUE É UMA BRANCH NOVA, E QUEREMOS GUARDAR NO GITHUB, NÓS TEMOS QUE FAZER:

git push --set-upstream origin staging

FEITO ISTO, podemos ir ao github ver

[Nosso repositorio->selecionar a branch staging] Agora vemos o codigo atualizado e é diferente do codigo da branch Master.

Com isto não corremos o risco de estragar o que está a funcionar e podemos testar novo codigo na branch paralela.

1. fazer MERGE:

Vamos unir o codigo de uma nova branch chamada "sistema de login" com a Master

Então MERGE significa unir

para unir, fazemos:

1º Entrar na branch master

git checkout master

2º git branch

ter a certeza que estamos na branch master

3º git pull da branch principal: git pull-> Traz as atualizações do servidor para o nosso pc para ter a certeza que estamos a unir os codigos nas versões mais recentes possiveis. Só depois de fazermos isso, é que fazemos o git merge

git pull

4º Criar uma nova branch com base na branch principal

git checkout -b sistema-de-login master

5º ver em que branch estamos

git branch

6º Agora criamos e editamos um ficheiro chamado "sistema de login" e imaginemos que tem la codigo para fazer o sistema de login.

7º Adicionar este codigo ao controlo de versão

git add .

8º git commit -m "criado sistema de login"

9º Ver se atualizou e se já podemos enviar o nosso ficheiro

git status

10º Dar commit para enviar o nosso ficheiro

git commit -m "Criado sistema de login"

AGORA SE O CODIGO ESTIVER EM CONDIÇÕES E QUISERMOS PASSAR PARA A MASTER:

11º Ir para a branch master

git checkout master

12º Ver se está tudo com as versões atualizadas

git pull

13º Unir a branch sistema de login com a master

git merge sistema-de-login

14º Mandar as alterações para o github

git push

1. PULL REQUEST UM MERGE DEMOCRÁTICO: Unir codigos com merge só se algum programador aprovar essa mudança

git checkout sistema-de-login

Depois disto vamos alterar o nosso codigo no sistema-de-login

git add . #Para adicionar as nossas alterações

git commit -m "adicionado idade ao sistema de login"

AGORA SE O CODIGO PRECISAR DE SER APROVADO, TEMOS QUE FAZER ISTO:

git push

git push --set-upstream origin sistema-de-login #enviar para o github

Agora entramos no github, e no nosso repositorio temos la que o nossos ficheiro sistema-de-login recebeu pushes recentes, ao lado diz "compare and pull request" e agora, temos que selecionar de onde é que queremos fazer o merge, ou seja: compare: sistema-de-login->base: master #Nota: compare e base já está escrito lá no site. Nós so temos que selecionar a ordem das branches e dizer que queremos passar de x para y ou de y para z.

Depois dizemos o que está a ser adicionado "adicionado idade ao sistema de login" e podemos deixar um comentario para explicar melhor o que está a ser feito " adicionado o requisito de idade para que possa ser validado que o usuario é maior de idade ou nao"

Depois é so dar "pull request".

Depois disso, no lado direito vamos ter la: Reviewrs, No reviewrs; No caso de termos alguem para rever o nosso projeto, vamos ter ali o nome das pessoas que vao rever e aprovar a nossa mudança no projeto.

Caso nao sejam aceites, as pessoas vão deixar um comentario com as alterações que acham necessarias.

Depois nós temos que fazer essas alterações e é so dar um COMMIT dizendo que as alterações que x ou y pediram estão feitas, não precisamos de criar e fazer pull request outra vez.

1. git ignore: Guarda ficheiros que não queremos que sejam enviados para o github.

ex: Se tivermos uma pasta com senhas e nao queremos enviar para o github

abrimos o git bash na nossa pasta, fazemos:

touch .gitingore

Esse comando vai fazer com que seja criado um ficheiro na nossa pasta chamado ".gitignore" e lá, usando o bloco de notas, metemos tudo o que queremos que não seja adicionado ao github.

ou seja: Se tivermos um ficheiro chamado "senhas.txt" e uma pasta chamada "segredos da aplicação"

abrimos o nosso bloco de notas do .gitignore e escrevemos la: senhas.txt

segredos da aplicação/

## TODOS OS COMANDOS APRENDIDOS:

git init-> Cria um novo repositório

git add . -> adiciona os arquivos atuais ao proximo commit

git status -> verificar o status atual do repositório git

git commit -m "mensagem do commit" -> cria um novo commit com uma mensagem

git push -> envia as atualizações para o github na branch atuamente ativa

git branch -> permite ver quantas branches temos e em qual delas estámos a trabalhar (a verde)

git checkout nome-da-branch -> permite mudar para uma nova branch

git checkout -b "nome da branch de origem" "nome da nova branch" -> permite mudar e criar uma nova branch com base noutra

git merge "branch a receber merge" -> permite fazer o merge da branch ativa atualmente com outra branch

git pull -> atualiza a branch que estamos a utilizar

touch .gitignore -> cria um ficheiro chamado ".gitignore" onde colocamos o nome dos ficheiros que queremos que não sejam enviados para o github.