GIT E GITHUB

No curso de git e github aprendemos que essas ferramentas apesar de serem distintas, complementam uma a outra, pois o git, serve como uma ferramenta onde podemos organizar nossos códigos, atualizar, criar pastas e compartilhar em algum repositório de forma remota, sem precisarmos de fazer manualmente todas essas tarefas, e o github funciona como uma rede social, onde podemos depositar nossos códigos e temos a facilidade de trabalhar em conjunto com vários programadores de várias áreas do mundo de forma simultânea, servindo também como área de marketing place, e uma comunidade para programadores.

GIT: desde já, vimos que CMD e o GIT servem para finalidades parecidas, mas antes de irmos mais a fundo, temos de ter ciência que esses são partes da programação que não tem uma interação gráfica, dessa forma, não funciona atalhos de teclados, clics com mouse dessa forma, esse programam só responde com comandos no teclado.

E nesses programas, vamos aprender a criar diretórios, organizar documentos com o propósito no fim de fazer tudo isso porém usando o github como detentor de todos os nossos arquivos. Sabido isso, o nosso primeiro contado com algum terminal desse tipo, é como CMD. O CMD pode mudar de acordo com o sistema operacional, os sistemas operacionais derivados do UNIX tem comandos diferendos dos derivado do Windows. (note que o git que instalamos, é como se fosse a versão do Linux, além disso, o CMD do Windows 10 emula uma versão do cmd do Linux, e pelo que testamos o git que instalamos funciona com os 2 comandos).

-DOS COMANDOS –

Dir (cmd) / ls(Linux) = lista as pastas, nos mostra onde estamos situados dentro do sistema operacional;

Cd(cmd/linux) = Esse comando funciona para ambos os sistemas operacionais e serve para navegarmos entre as pastas do sistema operacional. O significado deste é Chang-direction, então, basta escrevermos “cd” dar enter, depois disso, o programa vai nos jogar para o diretório base, dessa forma, escrevemos “/” que podemos chamar de barra de navegação, que indica que “queremos ir ao algum lugar” e depois digitamos o nome da pasta em questão.

cd..(cmd/Linux) = Esse comando serve para retroceder uma pasta, quando se está dentro de outra pasta, esse comando serve tanto para sistema Linux e Windows.

Cls(cmd) / clear(Linux) = quando estamos configurando algo no terminal, inevitavelmente acontece de ficar muitas informações na tela, podendo causar certo tipo e confusão, dessa forma, esses 2 comandos serve para limpar a tela, tornando tudo mais limpo na hora de trabalhar.

Do sistema Linux, temos o atalho CTRL + L que funciona de forma idêntica ao comando “clear”.

No Windows esse comando não funciona, porém temos um autocomplete de pasta, isto é, toda vez que estivermos digitando o nome de uma pasta, basta, que apertemos “tab” e o sistema completa o nome que supostamente você iria digitar, dessa forma, evitamos erros humanos.

Mkd (cmd/Linux) na criação de pasta, utilizamos mkdir (make directori) e para nomear a pasta, basta que digitemos o nome logo após, o mkdir. Um conceito bastante útil em terminais é que silêncio é sinônimo de sucesso, isto é, se digitarmos algum comando e o terminal não retornar nenhuma mensagem de erro, significa que tudo funcionou com sucesso.

Del(cdm)/ = esse comando serve para deletar arquivos dentro de um diretório, no curso criamos um arquivo de texto pelo cdm “ echo hello > hello.text” dentro da pasta workspace que criamos, dessa forma, usamos o comando del, para apagar o conteúdo da pasta, note que apagamos o conteúdo da pasta, não a pasta em si, para apagar o diretório da pasta, usamos de outro comando.

Rmdir (cmd)/ = esse comando serve para excluir o repositório com absolutamente tudo que há nele rmdir = remove repositor. No exemplo do curso usamos, devemos digitar esse comando e o nome da pasta que queremos apagar, além disso, usamos também duas flags s/ q/ (seja lá o que isso significa).

Rm (Linux) = para deletar um diretório no Linux é um pouco diferente, nesse caso basta que usemos o rm (remove), é interessante que usemos também o a flag -rf ( r= recursivamente f = force) nesse caso, a flag “r “ indica que quer apagar a pasta e tudo que há nela, pois pode haver um encadeamento de pastas, isto é, uma pasta dentro de uma pasta, dentro de uma pasta e assim por diante, e com o f deletamos tudo, já o “f” indica que não queremos que o sistema nos pergunte sobre esse procedimento e delete de uma vez.

Logo o procedimento para excluir de vez o diretório é rm -rf (nome da pasta) / rm -rf workspace

-SHA1-

A Sigla sha1 significa Secure Hash Algorithm, esse é um sistema de criptografagem usado pela agência nacional dos Estados Unidos. Esse sistema é um dos motivos pelo qual o git é amplamente usado, justamente pela segurança. Trazendo pra um contexto prático, toda vez que salvamos um arquivo pelo git, este gera um arquivo identificador com 40 caracteres, isto ocorre toda vez que um arquivo é salvo e se esse mesmo arquivo for modificado. Esse identificador permanece o mesmo, se o documento salvo não sofrer nenhuma alteração.

-tipos de objetos-

O git trabalha com 3 tipos de objetos internos, que isto é, elementos que armazenam algum tipo de dado, a teoria destes é importante para entendermos o porquê do git ser tão seguro. O primeiro deles é o mais basilar de todos, que é o “BLOB”. Esse objeto armazena basicamente um elemento de texto o “sha1” deste elemento, isto é, o identificador de 40 caracteres, essa é a estrutura base de qualquer dado que vamos lidar usando o git.

Trees – ou arvores, são objetos que fazem referência a um “blob” ou a um conjunto de blobs, ou a outras árvores, vale salientar que as trees também possuem seu sha1, que são modificadas toda vez que a tree em questão é modificada ou quando um blob dessa tree é modificada. Em outras palavras os blobs estão diretamente ligadas com as trees, e quando modificamos um blob, este tem sem sha1 mudado, assim como a tree que está conectado a esse blob. Essa estrutura de objetos é importante porque sempre quando vamos montar algum sistema ou algoritmo, sempre utilizamos pastas dentro de pastas, e garantindo que qualquer modificação destas faz com que o identificador mude, garantimos a segurança do arquivo.

É importante saber que, as trees, diferente do blob, armazenam como metadados o nome dos blobs e arquivo destes.

E por fim, nós temos o Commit, este objeto também tem a sua própria “sha1” e aponta para outro commid, assim com outras trees e outros blobs e segue a mesma lógica e relação, entre as trees e os blobs, caso um blob que esteja relacionado a uma tree, que por usa vez, está atrelada a um commit, tenha seu conteúdo mudado, todo o objeto subsequente e o próprio blob terá seu “sha1” mudado , o que torna todo o procedimento ainda mais seguro.

No caso do commit, ele armazena as trees e os blobs, além disso, como meta dados, ele armazena a data em que o arquivo foi modificado, e o autor, e vale salientar que o arquivo só pode ser mudado pelo mesmo autor.

-GERANDO CHAVE SSH-

Gerando uma chave ssh, além de aumentarmos a segurança, liberamos funcionalidades no github que sem esta seria difícil termos, como clonar códigos e atualizar simultaneamente, compartilhar nossos códigos com terceiros a fim de trabalhar juntos de forma remota, etc.

Então, para cadastrarmos essa chave, o processo consistem em 3 basicamente, gerar uma chave no git e cadastrar um email, saber qual é a chave através do código “cat”, ainda dentro e do git, e por fim, adicionar essa senha, tanto no git hub quanto num gerenciador de segurança dentro do git, esses passos serão melhor explicados as seguir.

1 passo : Aqui vamos gerar o código lá no git com o comando “ssh-keygen -t ed25519 -C (email de cadastro do github)” depois aperte enter;

Em seguida o sistema vai pedir que você cadastre uma senha para esta chave, basta digitar normalmente e apertar enter, os caracteres não vão aparecer git, mas não tem problema, funciona desta forma.

Após isso, devemos navegar para a pasta onde está a nossa chave que já foi criada, lembre-se que ao criar uma chave de acesso, o sistema cria 2, uma chave pública e outra privada, a pública vem com o sufixo “pub” no fim. Note que o git também vai mostrar onde a chave foi criada dentro do seu sistema geralmente é algo como (C/user/jefer/.ssh/id\_ed25519) verifique no topo das informações do comando.

Devemos acessar a pasta com o comando cd (change Direction) que vimos anteriormente

2 passo: depois que estamos dentro da pasta, temos de acessar a nossa chave ver quis são os caracteres que a compõem, dessa forma, vamos utilizar o comando “cat”+ “id\_ed25519.pub” que é o comando mais o nome do arquivo que vai ser aberto. Note que nessa etapa, vamos usar a chave pública para configurar o nosso github.

Depois de copiar a chave de acesso, vamos ao github, configurações, keyssh, ir na opção de usar uma nova chave, dar um nome a essa chave e depois colar a chave que copiamos no git na área de texto. Note que temos a opção de deletar uma chave já existente ou de cadastrar outra chave simultaneamente, lembre-se que cada chave representa um dispositivo, e nada impede de haver muitos dispositivos logado numa mesma conta com várias chaves de segurança ao mesmo tempo

3 passo: Já depositada a chave no github, agora devemos depositar a chave no agente de segurança do git, e para isso, usamos comando “eval $(ssh-agent -s)” após isso, apertamos enter, e será mostrado no número do agente logo em baixo.

Após isso, devemos adicionar a chave a esse agente, e para isso utilizaremos o código

“ssh-add id\_ed25519” E com isso ele vai pedir a senha que cadastramos la no começo que digitamos ( não confundir com a chave gerada pelo sistema), basta digitar e apertar enter.

Dessa forma, a chave já está validade, no sistema windowns, como é o primeiro acesso, o sistema vai avisar que se trata do primeiro acesso e que não há nenhum nome cadastrado na chave e vai perguntar se deseja continuar mesmo assim, basta digitar “yes” e continuar, dessa forma a chave será validade da mesma forma.

E para clonar um repositório na sua máquina de forma que você consiga atualizar simultaneamente no github, basta ir no github, criar um repositório, ir na opção clone e pegar o endereço ssh. Depois ir no git, usar o comando git clone, colar o endereço ssh e dar enter.

-COMANDOS DO GIT-

Git init – Para usar alguns comandos do git, note que sempre vamos inicia-los pelo nome “git”. Esse nome faz referência ao programa, a tecnologia que está sendo usada, dessa forma, o nosso primeiro comando, git init, serve para inicializar um repositório, ou seja, vamos criar uma pasta normalmente com o comando “mkdir + nome” da pasta e em seguida vamos usar esse comando para inicializar o diretório, e só depois desse comando a pasta fica apta para versionar códigos e fazer qualquer modificação através do git, pois esta será o nossa pasta gerenciadora, é uma lógica parecida que será aplicada quando trabalharmos com github.

Git config --global user.email /user.name – para que o git funcione corretamente quando formos fazer um commit, isto é, montar um objeto e eniva-lo para o github é necessário que configuremos o git primeiro e para isso precisamos configurar um email e um nome, para essa questão usamos o “git config –global user.email “nome do email” e depois “git config –global user.name + nome”. Podemos definir essas configurações toda vez que enviarmos um commit e criarmos um repositório novo, porém usamos a propriedade global, que engloba todas as nossas necessidades sem precisar de ficar pondo essa informação todas as vezes eu fazermos um commit

Git add \* = e para selecionarmos/ adicionarmos o arquivo que vai ser enviado, na pasta onde o arquivo está, utilizamos este comando “git add \*”.

Git commit -m “comentário” = e por fim temos o comando commit que envia nossos arquivos, que é composto pelo comando “git commit -m” + “comentário sobre o commit” dessa forma, com o comentário vamos ter ideia do que foi modificado no nosso projeto, e junto dele, vamos ter informação da data e hora e o arquivo que foi mudado, e assim já começamos a ter uma ideia de como é o compartilhamento de dados entre o git e github.

Git status = Aprendemos que os arquivos passam por alguns estados antes de receberem um commit, que são não listados (untrack), esse é o estado do arquivo quando a pasta não recebe o git init para gerenciar o arquivo e quando o arquivo não é adicionado na pasta com o git add \* depois disso, e em seguida temos o “unmodified” que é quando o arquivo não é modificado e o “modified” que é quando o arquivo foi modificado. O arquivo ainda passa pelo status “stage” que é onde ele fica na espera depois de ser modificado ou não, pronto para recebe o commit, e depois do commit, o arquivo retorna ao status de unmodified e repete o processo e dessa forma vamos atualizando nossos arquivos.

O git status serve justamente para monitorarmos como está o status dos nossos arquivos para nos situarmos melhor antes de realizar o commit.

Mv “nome do arquivo” + “./nome da pasta/” = dessa forma podemos mover os nossos arquivos para qualquer pasta do sistema ex: mv texto.txt ./download/

OBS: Note que ao criarmos uma pasta e adicionarmos um arquivo que foi que foi commitado e adicionado no git, através do comando add, essa pasta nova junto com o arquivo detro passam a ser objetos estranhos dentro do git, ele reconhece como se o arquivo que antes estava na pasta “mestre” tivesse sido deletado e nem se quer nota a nova pasta que foi criada, dessa forma, é necessário adicionar tanto a pasta quanto o arquivo através do git add ex:

git add texto.text download/

dessa forma os arquivos ficam prontos para dar um commit novamente e ficam aguardando o próximo comando na área de staged

OBS: o git add \* : move todos os arquivos untratckd de uma vez só, seja pasta ou qualquer outro tipo de arquivo e o git add + nome arquivo/ou pasta move um arquivo específico.

Git config - -list = esse comando é interessante para saber todas as configurações do seu git , email cadastrado, nome etc.

Caso queiramos mudar a alguma configuração que fizemos errado ou desejemos mudar por qualquer motivo, usamos o comando “git config - -global - -unset + “propriedade que vai ser mudada”. Essa propriedade que vais e ser mudada pode ser o “user.name” o “user.email” entre outros elementos. E depois de apertar enter, basta reescrever novamente os comandos correspondentes a configuração que você quer fazer, “git config - -global user.name” por exemplo.

-ENVIANDO ARQUIVO PARA GITHUB-

Primeiro temos de criar um repositório lá no github e a partir daí, temos de copiar o endereço do nosso repositório que é uma url que indica o caminho.

O próximo passo é passar a nossa pasta com o os nossos arquivos de desenvolvimento, que estão no nosso computador para o repositório do github, e para isso utilizamos o comando

“git remote add origin + endereço do nosso repositório”

Após isso, é estabelecido uma conexão entre a nossa pasta física e a do github, e agora basta enviarmos o nosso arquivo com o comando “git push origin master”, após isso, basta digitar a senha e enviar.

-resolvendo conflitos-

Pelo git ter a proposta de ser um sistema de versionamento de códigos onde compartilhamos os nossos códigos e trabalhamos em conjunto para resolver problemas, pode acontecer de alguns conflitos surgirem, isso acontece pelo fato de outro usuário clonar a sua pasta colaborar com o seu código e ao mesmo tempo em que você o seu colaborador desenvolvem acontece de ambos os programadores modificarem a mesma linha, dessa forma, ao usar o “git add” e o “git push origin master” o git reconhece o conflito e acaba abortando o push.

Quando isso acontece a procedência é baixar o código e o git vai dizer em qual arquivo está o conflito e para tal proceder, usamos o “git pull origin master” que vai fazer o download do arquivo conflitante e temos de muda-lo manualmente.

Depois de mudar e definirmos para o git que a nossa versão é a versão correta, e definitiva, basta enviarmos normalmente, git add > git commit -m “comentário” > git push origin master.

-Git clone-

O git clone é o comando para clonarmos o diretório de pessoas alheias ou mesmo o nosso próprio diretório coma finalidade de irmos atualizando o nosso próprio projeto ou de irmos colaborando com projeto alheio.

A grande vantagem de se clonar uma pasta é que ela já vem programada como um diretório e o git já reconhece como um ambiente diretório e não precisamos passar por aquele processo do de usar os comandos git init e o git remote add origin, para estabelecermos uma conexão para aí sim clonarmos nossos objetos.

Ao clonar a pasta, estamos baixando automaticamente todo o conteúdo desta, tão é interessante criar uma pasta e inicializar o git dessa pasta para poder modificarmos e organizar tudo de forma mais organizada.

Pra clonar, basta irmos no diretório criado no github e lá pegar a url do repositório, e usar no git o comando git clone + url do diretório e apertar enter.