**Curso de Git e GitHub:**

**COMANDOS DO GIT:**

**-cd(..):** o cd serve para entrar num diretório, já o cd.. serve para sai do mesmo que você está ou acabou de entrar.

**-mkdir:** comando no windows para criar pastas.

**-init**: comando no git para criar um repositório.

**-status:** comando do git que mostra o status da sua pasta, caso tenha algum arquivo que esteja no estágio de unmodified, modified(foi feita alguma alteração nele), staged e commited, para que assim você possa fazer as alterações como quiser.

**-add**: comando do git que muda o estágio do arquivo de modified(ele não está salvo na base do git, ele poderá entrar para o controle de versionamento do git) para staged.

**-commit**(ele salva essa versão, é bom também sinalizar o que foi alterado)**:** comando do git que passa de staged para unmodified automaticamente, porém ele passa pela fase de commited(nessas fase o arquivo já entrou para o controle de versionamento).**é como uma foto do estado atual do projeto.**

**-checkout:** esse comando do git ele volta o arquivo para fase de unmodified, ou seja, pra fase em que não houve nenhuma alteração, isso é interessante, pois caso você faça alguma alteração, mas depois se arrependa ou até não goste você possa recuperar a versão anterior. Isso pode ser feito antes mesmo de ter realizado o commit.

**-clone**: opção de clonar o repositório, quando você clona um repositório você traz com ele todas as informações de conexão local e remota desse repositório, a partir disso toda vez que você faz um commit no seu computador eu conseguirei enviar essas informações que serão automaticamente carregadas e atualizadas no github. É melhor você copiar o link e depois coloca-lo no terminal.

**-push:** esse comando é utilizado para enviar os arquivos, alterações, pastas que você adicionou ao repositório do github, porém você consegue realizar isso através do terminal.

-**pull:** com esse comando você puxa os arquivos atuais do github para o seu repositório local, ou seja, caso você esteja com o código desatualizado ,utilizando esse comando, todas as pastas que foram criadas/alteradas serão baixadas para o seu repositório.

**-remote add:** comando que adiciona um novo repositório remoto ao seu repositório local, porém é preciso que você dê um nome que vai ser “apelido” do URL do repositório remoto assim você não precisa ficar digitando o toda hora. Nome padrão que normalmente usam é origin. Com isso você faz com que seu repositório local saiba que há um repositório remoto que ele pode se ligar.

**BRANCHES (RAMOS/BRAÇOS):**

Com a criação de novas branches você consegui continuar produzindo sem afetar a branch máster, pois o commits serão adicionados a um linha alternativa, mas quando você finalizar os seus teste e estiver tudo correto você poderá alocar as suas alterações na branch master novamente e isso se chama **MERGE**.

As branches funcionam como linhas alternativas para o controle do seu projeto, sendo possível você criar ramos para trabalhar no mesmo projeto e evitar que aconteçam conflitos com a branch master que é a linha principal. Cada commit entra no branch e funciona como se fosse uma foto do atual estado do projeto, ou seja, a branch é uma linha do tempo com várias fotos de como o projeto estava ou está. Dentro do commit nós temos um código específico para esse commit, autor do commit, horário da realização do commit e comentário do commit.

Após a realização de um próximo commit, estarão as mesmas informações do outro commit,, como o seu código específico, quem o fez/alterou, horário de alteração, mensagem e um a + que é o parentesco com o commit anterior e assim sucessivamente. É assim que o git controla todas as alterções que são realizadas no projeto/código, pois assim cada commit terá uma ligação com o commit anterior. Porém isso acontece assim sempre na mesma branch.

Quando você cria uma nova branch os commits continuam com uma ligação com os commits anteriores, ou seja, você ainda tem acesso a tudo que aconteceu, todas as alterações, sendo assim você consegue trablhar nessa branch como se estivesse trabalhando na branch master.

**Tipos de MERGE:**

**-Fast-Foward:**  é quando você junta as branches e o github não irá precisa criar novo commits, ou seja, ocorreu tudo “bem”, ele não irá precisar fazer grandes alterções, é como se ele projetasse os commits da branch nova na branch master sem maiores complicações. Porém, isso só é possível quando não há nenhuma alteração reliazada na branch master enquanto houve alteração na nova branch. Após a junção das branches é possível eliminar a branch que foi criada deixando apenas a branch master, dando a impressão que todas as alterções foram realizadas na branch master.

**-Recursive-strategy:**  esse caso é diferente. Nele há a criação de uma nova branch e nessa branch são realizados novos commit, porémocorre a realização de novos commits na branch master ao mesmo tempo, para sanar isso o git ao junta a branches faz um commit chamado **merge(esse terá dois pais a branch master e a branch nova que foi criada[relaçao de parentesco com os commits])**, mas nesse caso não há grandes complicações, pois nesses commits não houveram nenhuma sobreposição de dados ou conflito de alteração de códigos, por isso ele usa o recursive-strategy para a realização desse merge, ele junta as linhas de branch de maneira automática e sem afetar nenhum dos arquivos alterados.

**-Conflict:** ocorre quando alguém fez uma alteração na branch master e também foi realizado alguma alterção na mesma parte do projeto na branch nova, dessa forma antes do git realizar o commit merge ele informa que houve algum conflito nas alterções, ou seja, mesma linha de código foi alterada nos commits, sendo assim o git abre o arquivo de conflito para assim ser possível remover os conflitos e fazer novamente o commit