**Git: Trabalho em Equipe com Controle e Segurança**

* **Minha Conta GitHub:**

Usuário: [michel\_soares@outlook.com](mailto:michel_soares@outlook.com)

Senha: mitzero0

* **Introdução a Tecnologia GIT**

Git: É uma tecnologia de versionamento de dados usado para facilitar, por exemplo, o desenvolvimento de um projeto de software onde por sua vez temos diversas pessoas trabalhando em equipe, muitas vezes alterando um mesmo arquivo simultaneamente. Com o uso correto do Git temos a certeza que as partes do projeto desenvolvidas por cada pessoa da equipe integram-se entre si fazendo que cada integrante da equipe tenha sempre uma versão atualizada do arquivo.

* **Instalação do GIT**

Muito simples instalar!

O GIT fornece um repositório de código fonte na web grátis para projetos públicos (GitHub), ou seja, o GitHub é uma plataforma online de repositórios;

* **Comandos Básicos GIT:**

*git clone*  - Esse comando clone criará uma pasta com o mesmo nome padrão do repositório.

*git tag* - Visualizamos todas as tags (versões do software) criadas.

*git checkout v0.1* - retornamos a um uma versão anterior do projeto.

*git diff v0.1 v0.2* - esse comando permiti visualizar as alterações realizadas da versão v0.1 para v0.2.

*git blame -* *<nome\_arquivo.extensao>* - podemos visualiza quem alterou os arquivo recentemente.

*git init* - esse comando inicializar um repositório no diretorio especificado.

*git add* *<file name>* - esse comando nos permite adicionar um arquivo para lista do git, de arquivos rastreados, monitorados pelo mesmo. podemos adicionar varios arquivos ao mesmo tempo

*git add* <file name> <file name> (...)

*git add \*.xhtml -* para todos os arquivos na pasta que tenha o formato .xhtml

*git add nomePasta/ -* adicionando todos os arquivos de um diretório especificado.

*git commit -m* *"exemplo"* - esse comando cria um commit do nosso projeto. OBS.: sempre é preciso passa uma mensagem para identificação.

*git whatchanged* - permite verificar quais arquivos foram alterados por cada commit realizado.

*git whatchanged -p* - permite visualizar detalhes de todos os arquivos alterados por cada commit realizado.

*git remote* - esse comando mostra todos os repositórios remotos que possuo no GitHub.

*git remote add* *<exemplo-repositorio>* https://github.com/MichelSoares/git-alura.git - esse comando serve para integrar repositório locais com um repositório remoto.

*git push* *<name repositorio remoto> <name branch>*: exemplo - git push origin master - esse comando manda as alterações realizadas para o repositório remoto.

* **Integração com GitHub**

Depois de criada nossa conta no GitHub, precisamos gerar uma senha (chave de segurança) que será responsável por identificar nossa máquina quando fizermos as interações entre nosso projeto e o serviço do Github. Caso não exista essa configuração entre nossa máquina e a conta do Github, o acesso ao serviço pelos comandos do Git será negado.

Especificamente no Windows (Git Bash) e não no ***prompt de comando***digitar o seguinte comando:

*ssh-keygen -t rsa -C* [*seu\_email@hotmail.com*](mailto:seu_email@hotmail.com)

Com esse comando geramos nossa chave pública/privada para realizamos operações especificas no GitHub.

Enter- para não alterarmos o local que o mesmo salvará a chaves (geralmente salvo no diretório com nome do Usuário na pasta .ssh)

Enter- para não atribuímos uma senha para abrir nossa chave public/private.

Agora no site do GitHub em Settings> SSH keys> ADD SSH key> title (titulo da chave) abaixo a chave propriamente dito (id\_rsa.pub).

* **Criando e interagindo com o repositório remoto**

Um repositório remoto nada mais é do que um "espelho" do nosso repositório local. Quando executamos o comando git init - estamos inicializando um repositório em determinado diretório, porem a principio ele é somente LOCAL.

* Criamos um novo repositório no GITHUB por padrão com o mesmo nome do projeto. Copiamos um endereço HTTP.
* No GitBash (ou Prompt de Comando do Windows caso na instalação o tenha habilitado) -

*git remote add* *<exemplo-repositorio>* *https://github.com/MichelSoares/git-alura.git*

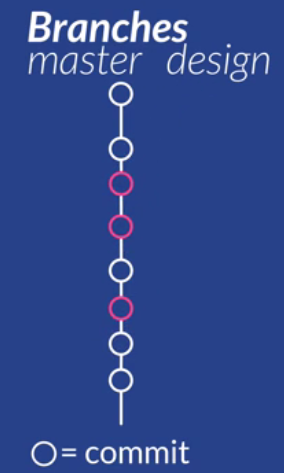
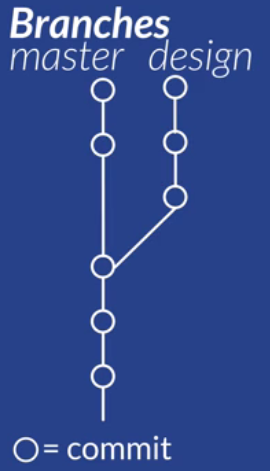
Esse comando serve para integrar repositório locais com um repositório remoto.

* Mandando as alterações dos meus COMMIT'S para o repositório remoto:

*git push* *<nome repositorio> <branch>*

* **Conceitos do BRANCH**

Branch é basicamente uma linha de produção sendo a principal por default a MASTER, onde o sistema está rodando. Quando trabalhamos em equipe temos diversas pessoas modificando vários arquivos no projeto, muitas vezes o mesmo arquivo simultaneamente, agora imagine se ambas realiza-se o commit no repositório em uma BRANCH ao mesmo tempo e ainda mais com o sistema em produção, ocorreria um erro! Para evitarmos o ocorrido geralmente trabalhamos com BRANCH'S "SECUNDÁRIAS", paralela (uma bifurcação) referente à BRANCH principal. Conforme imagem abaixo:



Observação: Quando trabalhamos em uma branch paralela temos a vantagem de não alteramos o código que está em execução! o que evita de *quebrarmos* o código e o sistema cair. Terminado todas as modificações na nossa branch paralela devemos integrar com nossa branch principal, para efetivar as modificações realizadas. No Entanto antes devemos ATUALIZAR nosso repositório local com as modificações que eventualmente os desenvolvedores programaram, no caso usamos o comando: *git pull* . Não se preocupe quando atualizamos nosso repositório local referente ao remoto não perdemos nossas modificações pelo contrário o GIT realiza um MERGE AUTOMATICAMENTE, integra nossas modificações com as existentes no nosso repositório, caso haja conflito o mesmo avisará e será necessário realizar um MERGE manualmente.

* **Como realizar MERGE Manualmente.**
* Abrir o arquivo onde ocorreu o conflito, atento no detalhe que indica: <<<<HEAD (conteúdo) ==== *commitado* por mim e ======= (conteúdo) >>>>>>>> commitado por outro desenvolvedor no mesmo trecho de código. Nesse momento definimos com qual modificação optar.
* **Comandos Branch’s**

*git branch* - esse comando mostra as branch’s existentes.

*git branch nomebranch* - criando um Branch.

*git checkout nomebranch* - trocando de branch.

*git checkout -b* *<nome branch>* - esse comando criar e já troca para a branch recé criada.

*git branch -d <nome branch>* - deletando uma branch;

*git push -u* *<nome repositorio> <nome branch>* - criando uma branch remota e vinculado-a com a local com o mesmo nome.

*git branch -r* - esse comando retorna todos as branch's remotas.

*git branch -a* - esse comando retorna todas as branch's locais e remotas.

*git checkout -t* *<nome repositorio>/<nome branch>* - esse comando executa: A criação de uma nova branch local chamada <nome branch> é criada, muda-se para essa branch, copiamos todo o conteúdo da branch remota <nome branch> do repositório referente ao <nome repositorio> e trackeamos as duas branches.

*git push* *<nome repositorio>:<nome branch>* - deletando uma branch remota.

*git fetch* *<nome repositorio>* - Verificando as atualizações do repositorio citado.verificamos a possível criação de novas branchs.

*git push* *<nome repositorio> <branch>*: passando minha nova branch local para meu repoditório remoto. Para evitar tal trabalho, podemos indicar o caminho (track) da branch remota para a nossa branch local. Isso pode ser feito no instante em que criamos a branch remota através da opção "-u". No nosso caso, temos:

*git push -u origin design*

*git merge* *<nome branch>*: Salva meus commits da minha branch AUXILIAR na branch MASTER (Principal).

* **Resoluções de conflitos**

E quando em um projeto diversas pessoas estão alterando um mesmo arquivo ao mesmo tempo quem irá commitar o projeto primeiro? caso commita como saberei que o mesmo já estará atualizado com as modificações dos outros desenvolvedores.

Uma abordagem possível é permitir que vários desenvolvedores manipulem o mesmo arquivo ao mesmo tempo. No entanto, o envio do arquivo para o repositório só pode ser feito se o desenvolvedor que deseja enviá-lo estiver devidamente sincronizado (atualizado) com o repositório. Essa é justamente a maneira seguida pelo Git.