**GIT**

Nos ajudam a manter um histórico de alterações;

Nos ajudam a ter controle sobre cada alteração no código;

Nos ajudam para que uma alteração de determinada pessoa não influencie na alteração realizada por outra;

Etc.

Baixar o programa git <https://git-scm.com/download/win>

Ou Gitkraken Programa Visual. Para trabalhar com git.

Sobre commit. Nunca comitar um código que não funciona.

**Comandos**:

Para registrar corretamente os dados do autor de cada alteração realizada, importante ter a configuração

--local serve para o projeto

--global serve para todos os projetos da minha máquina

git config --local user.name "Seu nome aqui"

git config --local user.email "seu@email.aqui"

“git init” Para começar a versionar um repositório na máquina,

“git status” Para visualizar o status do meu repositório (nome da branch, se possui commit, os arquivos que possui etc.

“git rm” Para remover o arquivo do monitoramento do git

“git add index.html” para adicionar o arquivo index.html no git(em um diretório temp antes de ser commitado) ou

“git add .” para adicionar o todos os arquivos que estão a partir da pasta que estou.

“git commit –m “mensagem do commit” ” para realizar o commit do arquivo ou alteração realizado.

“clear” limpa a tela do prompt de comando

“git log” poderemos verificar o histórico de alterações, cada mensagem de commits feitos, o andamento do nosso projeto

“git log –oneline” permite visualizar todos os commits, sendo que cada uma ocupa uma única linha

“git log –p” permite visualizar mais informações do commit

Se criar um arquivo “.gitignore” os arquivos ou pasta que informar dentro desse arquivo, serão ignorado pelo git. Basta então fazer o commit do gitignore, que os arquivos ou pasta inseridos dentro dele, não serão lidos pelo git.

“mkdir nomeDaPasta” cria uma pasta no endereço selecionado

**Utilização de um repositório remoto**

Neste exemplo vamos criar uma pasta local para simular um repositório remoto. Mas poderia ser um endereço da rede. Esse endereço é onde todos poderão pegar o projeto para editar local.

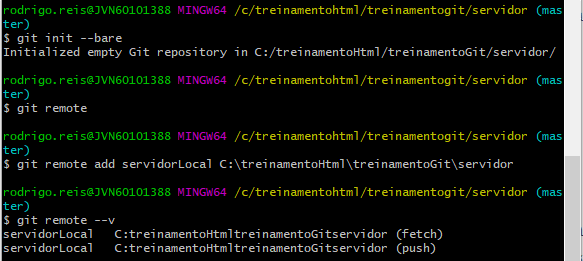
Criamos a pasta servidor para simulação

“git init --bare” para sinalizar que essa pasta não realiza alterações, apenas armazenará as alterações, ou seja é um diretório puro.

“git remote add servidorLocal /c/treinamentoHtml/treinamentoGit/servidor” Daremos o nome de servidorLocal para o repositório. Pode ser um diretório na nuvem, url etc...

“git remote” lista nossos repositório remoto.

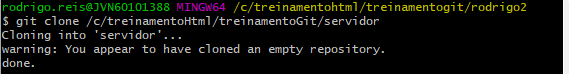
“git remote -v” exibe informações do repositório, bem como sinaliza o caminho que serão buscado (fetch) e enviado (push)



Continuando uma simulação, vamos criar uma pasta sinalizando que outra pessoa precisa do projeto. Então vamos dar um “mkdir rodrigo2”. Depois acessar a pasta rodrigo2.

“git clone /c/treinamentoHtml/treinamentoGit/servidor, para clonar este repositório na pasta rodrigo2 para que o outro usuário possa editar suas alterações localmente.

Obviamente se o servidor só foi criado e não foi feito o feito o commit dos arquivos para este servidor, após efetuar o clone, irá informar que o servidor clonado está vazio.



**Enviando/buscando/atualizando arquivos do servidor**

Na pasta local onde temos nosso projeto, vamos enviar os dados para o servidor para que outras pessoas possam pegar o projeto.

“git **push** servidorLocal master” **enviando** para o “servidorLocal” a minha branch master

“git remote rename <nomeAtual> <novoNome>” caso precise renomear a pasta servidor.

“git **pull** servidorlocal master” para **buscar** os dados do servidor para nossa pasta local na branch master.

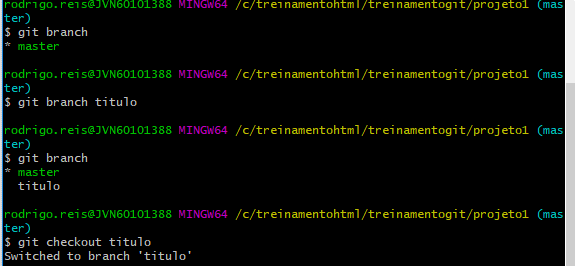
**Trabalhando com branchs**

“git branch” irá listar minhas branchs criadas

“git branch <nomeBranch>” criar nova branch

“git checkout titulo” passando a trabalhar na branch titulo

ou “git checkout -b <nomeBranch>” cria e já passa a trabalha na nova branch



“git merge titulo” na branch master eu executo este comando para pegar as alterações da branch titulo e colocar na master. Poderemos editar a mensagem exibida, mas caso não queiramos, para salvarmos e confirmarmos a mensagem, pressionaremos ":x + Enter" no editor Vim

“git rebase titulo” na branch master executando este comando, irá unir todos os commits da branch titulo na branch master

“git log --graph” serão exibidas linhas específicas representando o desenvolvimento, uma boa alternativa ao Visualizing Git.

O merge junta os trabalhos e gera um merge commit. O rebase aplica os commits de outra branch na branch atual. Com isso, evitamos os commits de merge. Há uma longa discussão sobre o que é "melhor": rebase ou merge. Estude, pesquise, e tire suas próprias conclusões. Aqui tem um artigo (de milhares outros) que cita o assunto: <https://medium.com/datadriveninvestor/git-rebase-vs-merge-cc5199edd77c>.

Sempre fazer o pull antes de iniciar um desenvolvimento.

**Desfazendo alteração**

“git checkout --<nomeArquivo>” Caso eu queira desfazer antes de do git add

“git reset HEAD” Caso eu queira desfazer após eu ter efetuado o git add. Irá voltar para o stado anterior (git add)

“git revert <has do commit>” Caso eu queira desfazer uma alteração já commitada. O has é o código gerado no commit. Esse código é possível ver ao executar o git log

Com o git checkout nós desfazemos uma alteração que ainda não foi adicionada ao index ou stage, ou seja, antes do git add. Depois de adicionar com git add, para desfazer uma alteração, precisamos tirá-la deste estado, com git reset. Agora, se já realizamos o commit, o comando git revert pode nos salvar.

**Guardando um código em alteração para depois**

“git stash” guarda os arquivos alterado em um repositório temp

“git stash list” lista as alterações guardadas

“git stash apply <indice>” Retorna para a minha branch os arquivos salvo

“git stash drop” Excluir as alterações que foram salvas no dir temporário

“git stash pop” Faz as duas ações “apply” e “drop” simultaneamente.

**Voltando “no tempo” em algum commit realizado**

“git checkout <hash>” sendo que hash é o código simplificado do commit. Porém isso faz com que vc esteja fora de qualquer branch.

Caso queira alterar alguma coisa em cima de um commit antigo e fazer este ser o atual, então:

Criar uma nova branch a partir do commit que vc precisa e executar “git checkout -b <novo-branch>”, de forma a não estarmos mais desassociados da linha de desenvolvimento, o que se confirma se realizarmos um novo commit.

**Analisando as diferenças entre commits/arquivos**

“git diff” mostra as diferenças entre os arquivos que foram editados mas ainda não subiram a partir do git add. Ou seja estão “localmente”

“git diff <hash>..<hash>” Mostra a diferença entre dois commits. Sendo que o “..” é para simbolizar o “entre”

git diff {hash do commit de merge com lista}..{hash do último commit realizado}

**Criando Tag no meu projeto**

“git tag -a v1.0.0 -m "Lançando a primeira versão do projeto" ” Dessa forma, geramos um produto entregável. Ou seja uma versão/release do sistema

“git tag” lista as tags já criadas

“git push servidorlocal v1.0.0” Empurrando a tag commitada para o servidor.

Fork é cópia de um repositório

“git rebase -i HEAD~3” selecionando 3 commits a serem unificados

Outra opção seria selecionar, copiar e colar manualmente o commit imediatamente anterior ao primeiro com o qual queremos trabalhar, acrescentando-o ao fim do comando git rebase -i.

Ao executarmos, receberemos a lista dos commits que queremos analisar. O primeiro comando, que por enquanto estará como pick, será responsável por definir se iremos juntar os commits ou não. Basicamente, teremos do commit mais antigo para o mais novo, de cima para baixo, e informaremos que queremos unificar "Quebras de linha" com o commit anterior, "Separando os títulos", e também com "Trocando UL por DL".

Para fazermos isto, substituiremos os pick por s, de squash, juntar, apertando:

pick 971dba0 Trocando UL por DL

s 3db095f Separando os títulos

s af37cf6 Quebras de linha

“git cherry-pick <hash>” Quando eu estou em na branch master corrigindo alguma coisa por exemplo, mas preciso de um commit específico de uma branch que estou trabalhando para entregar uma feature. Ou seja, Se uma implementação é necessária em sua branch e já foi feita em outra branch, pode fazer sentido trazer um commit específico, utilizando o git cherry-pick

**GIT BISECT**

Suponhamos que alguém da nossa equipe sobrescrevesse uma alteração que realizamos, modificando o título de "Lista de Cursos" para "Lista de Cursos de DevOps", inclusive commitando a alteração, e que, ainda, uma terceira pessoa da equipe, sabendo que esta alteração não deveria ter sido feita, modificasse o texto para "Lista de Cursos da Alura", commitando em seguida. Além disso, outra pessoa teria alterado o título depois de tudo isso, para "Cursos da Alura", e mais alguém voltaria o texto para "Lista de Cursos".

Muitas modificações foram feitas no título, e agora queremos encontrar o momento exato que determinada alteração foi aplicada. Isto quer dizer que queremos voltar ao estado em que o título esteve como "Lista de Cursos da Alura", por exemplo. Executando git log --oneline, notaremos que as mensagens de commits não são muito significativas e não nos ajudam nisso.

Seria necessário indicarmos ao Git que ele terá que buscar, dentre determinados commits — como os cinco últimos, por exemplo —, e ele teria que passar por cada um deles. Esta feature é chamada de bisect, e para executá-la digitaremos git bisect start, o que inicializará a busca do lado do Git. Inicialmente, é preciso informar a ele um estado, ou commit cuja parte do código que queremos não esteja boa, no caso, o título deverá ser "Lista de Cursos da Alura".

Vamos executar git bisect bad HEAD, e em seguida devemos informar o estado em que possivelmente estava bom, isto é, a partir de onde ele irá buscar o commit desejado. Então, copiaremos o hash referente à primeira alteração de título, e digitaremos git bisect good c17076a. Estes serão os limites da busca feita pelo Git. Obteremos a mensagem de que há uma revisão a mais para testarmos depois desta, e ele nos mostrará o estado de "Mudando título".

Neste estado, o título será aquele que queremos, e se ele ainda não tivesse encontrado a alteração desejada, continuaríamos executando git bisect bad, mas no nosso caso, como já estaremos no estado desejado, usaremos git bisect good. Após esta revisão, se tudo estiver bem, não precisaremos mais fazer nada. Para finalizar a busca, uma vez que o Git já nos entregou o hash e a descrição do commit, usaremos git bisect reset, com o qual voltaremos à master.

Com o hash do commit, poderíamos desfazer a alteração, analisar o porquê da inclusão dela, perguntar o que aconteceu à pessoa que realizou o commit. Para verificarmos todas as alterações referentes ao commit, aplicaremos o comando git show seguido do hash. Para reverter esta ação, pode-se utilizar git revert juntamente ao hash. Notem como é relativamente fácil encontrar um commit em que uma alteração específica foi implementada, sem que precisássemos, por exemplo, usar o git checkout ou git show em cada um dos commits existentes.

“git blame <nomearquivo>” permite saber quem comitou a alteração de um arquivo

“git show <hash>” mostra todas as alterações aplicadas pelo commit com o hash informado.

“git branch -d <nomeBranch>” remove a branch

Caso haja algum conflito, como o branch a ser deletado estar à frente do branch atual, ou seja, se novo-release tivesse alguns commits à frente do master, teríamos que utilizar git branch -D novo-release, com "D" em maiúsculo.

“git tag --delete tagname” excluir tag local

“git push origin --delete nome-da-branch” remover branch remota

“git push origin :nome-da-branch” remover branch remota ou remover tag remota

**Eventos automatizados - Git Hooks**

O Git possui eventos próprios, que são chamados de hooks. Assim, conseguimos criar os chamados Git Hooks para executarem uma ação quando um evento acontecer. Exe.: ao realizar o commit chamar um teste automatizado, ou alertar com uma mensagem, ou listar alguma coisa.... Criando um arquivo Shell Script, onde seu nome representa o evento, dentro da pasta .git/hooks. Para ver com mais detalhes os possíveis hooks (eventos), confira este site: https://githooks.com/.