Módulo 03 - GIT

O que é GIT?

GIT - É um sistema de controle de versão distribuído. Significa que conseguimos ter o código fonte em vários computadores e todos conseguem compartilham o desenvolvimento. Por mais que a gente escolha um servidor para manter o código fonte, se todos os participantes também tiverem uma cópia do código fonte e o servidor se perder, os participantes não perdem o código fonte e suas versões.

Github/Bitbucket/Gitlab - Plataformas para compartilhar seus códigos.

Gif config

- Permite ver e atribuir variáveis de configuração como nome e e-mail.
- No windows, o arquivo .gitconfig estará no diretório $HOME, que é C:\User\$USER
- Para ver as variáveis criadas, executar: cat ~/.gitconfig

Criando a identidade do git, assim que instalar:

- Configurar nome
  git config --global user.name "SeuNome"
- Configurar e-mail
  git config --global user.email seuemail@dominio.com
- Configurar o VS Code como editor padrão
  git config --global core.editor "code -w"
- Ver as configurações
  git config --list

Conseguindo ajuda com o Git Help

- Executar: **git help**

Init, add e commit (+ log, status)

Init

- Selecione o diretório que deseja iniciar o versionamento através do terminal (D:\GitFiles).
- Execute: **git init.**

Add e Staging

- Crie um arquivo chamado index.html no mesmo diretório. Caminho no meu computador: D:/GitFiles/GitTest/.git/
- Execute para adicionar o arquivo ao git: **git add index.html**.

Commit

- Criar uma marca na linha do tempo do arquivo. A cada alteração, ele vai atualizando com marcas novas.
- Executar: **git commit -m "adicionado arquivo index no nosso site"**
- Para desfazer o último commit: **git reset --soft HEAD~1**

Log

- Serve para verificar o que aconteceu com o arquivo em outro momento, dando nome do autor, alteração, data etc.
- Executar: **git log**

- Para trazer o log dos últimos n commits: **git log -n 5** (5 commits - esse número pode ser o que você quiser).
- Para trazer o log dos commits após uma data: **git log - -since=aaaa-mm-dd**
- Para trazer o log dos commits antes de uma data (mostra tudo que aconteceu até o dia anterior ao passado): **git log - -until=aaaa-mm-dd**
- Para trazer o log dos commits por autor: **git log - - author=seunome (não tem espaço entre os hífens)**
- Para trazer o log dos commits por regular expressions (busca palavras que estão na descrição que você deu nos commits): **git log --grep="texto"**

Status

- Serve para verificar se houve alteração no código. O terminal mostra onde você está, quais arquivos foram alterados etc. Se tem um arquivo no diretório que não está no git, ele aparece quando você faz um **git status**. Isso quer dizer que o arquivo está só no seu computador local e ainda não consta no sistema de versionamento.
- Executar: git status
- Ele percebe se o arquivo foi modificado, mas se as alterações não foram "commitadas", ou seja, avisa que você ainda não "salvou" no git a última versão do código. Para salvar, precisa fazer um novo git add.

Commit

- Quando você faz qualquer alteração no projeto, não basta salvar ele no editor. É preciso fazer um novo commit no git.
- Executar: **git commit -m "update arquivo index.html"** (pode ser qualquer frase que ajude a entender a alteração).
- Depois, se fizer um git log, você vai poder ver todas as alterações.
- Você pode fazer o git olhar pra todos os arquivos num diretório ou para todos eles. Com o comando "git add .", você diz pro git olhar todos os arquivos no diretório que vai haver um commit daqui a pouco.

Log, status e show

Branch

- É uma ramificação de uma linha temporal. Existe uma linha do tempo principal (master) que pode se dividir em algumas outras criando algumas realidades alternativas pro nosso projeto.
- A primeira coisa é criar a ramificação/branch: **git branch feature/lista-produtos** (feature/ serve para categorizar os branches).
- Migrar para essa nova linha do tempo: **git checkout feature/lista-produtos**
- Criar um novo arquivo no diretório chamado listaprodutos.html
- Adicionar o arquivo ao git: git add listaprodutos.html. Sempre que você executa um comando add, significa que você tá pedindo pro git começar a monitorar esse arquivo. É comum dizer também que você colocou esse arquivo em staging.
- Se você, por algum motivo, você cometeu um erro depois de colocar o arquivo em staging (executar o git add). Executar: **git reset**
- Commitando (atualizando) o arquivo listaprodutos.html depois de uma alteração no código: git commit -m "lista de produtos adicionada"
- Vendo as últimas atualizações: git log
- Voltando pra linha do tempo principal (master branch): git checkout master
- Como eu fiz alterações num outro branch, se executar o git log no master branch, só vão aparecer os commits que fizemos nesse branch.
- Na prática, o que acontece é: quando você trabalha em um time, existem branches para determinadas funcionalidades. Ao final do desenvolvimento, tudo precisa ir pra linha do tempo principal para fazer um release. Para isso, de dentro do master branch, executar: **git merge feature/lista-produtos**

- Para inspecionar commits (ver o que foi alterado), é só copiar o código e executar: git show CÓDIGO
- **Git show** - Inspeciona o último commit.
- **Git branch** - Mostra todos as ramificações que foram criadas no repositório.
- Agora que já unimos os conteúdos, precisamos excluir a branch feature/lista-produtos: git branch -D feature/lista-produtos

Revisão

- Git init: inicia novo versionamento.
- Git add: adiciona ou modifica alterações elegíveis para o nosso ponto no tempo. O arquivo começa a ser visto pelo git para versionamento.
- Git commit: adiciona de fato todas as alterações na linha do tempo. É quando gera o identificador de alterações.
- Git log: visualiza os pontos na linha do tempo (quando commits tem, tudo que aconteceu na linha do tempo).
- Git status: informa o estado atual de alterações - arquivos criados, removidos, que não estão na área de staging.
- Git show: apresenta um ponto indicado na linha do tempo. Você pode passar o identificador de commit ou não passar nada (ele mostra o último commit).

Branch, checkout, merge, push

- Github - Rede social de código fonte. Você consegue compartilhar o seu, seguir outros códigos fontes, gerar discussão. É gratuito para projetos open source. É uma forma de criar repositórios remotos.
- Adicionando projeto iniciado localmente no github - Entra no github > new repository > dar um nome > executar o comando " **git remote add origin**

https://github.com/nathannieg/gama-no-github.git" > submeter o arquivo para o respositório remoto executando o comando "git push -u origin master" > atualizar a página do projeto para verificar se o código fonte já está adicionado.

- Desafio - Criar um arquivo README.md, escrever nele o que eu estou achando do curso, fazer o git olhar pra ele pra controlar versionamento e subir pro github. O que eu fiz: criei arquivo na pasta, fiz um git add README.md pra adicionar o arquivo ao git, fiz um git commit -m "adicionando README" para criar um ponto na linha do tempo e fazer o git começar a controlar o versionamento. Por último, fiz um git push -u origin master pra subir pro github.
- Desvinculando o repositório remoto no git do repositório local - Para desvincular o repositório remoto no github do seu repositório local (na máquina), escrever no terminal de vocês (recomendo o git bash): **git remote rm <<nome do repositório que querem excluir>>** . Depois basta dar um git remote add origin <<repositório remoto novo>>.

- A importância do README - É importante para dizer o que o seu projeto faz, qual o objetivo dele, porque outras pessoas deveriam contribuir com ele e ainda dar instruções.

- Revisão

Git branch - Gerencia nossas realidades alternativas.

Git checkout - Navela entre nossas realidades.

Git merge - Une nossas linhas do tempo.

Git push - Envia nossas alterações para um repositório remoto no github.

Revisão de tudo até aqui (realizada 2 dias depois de assistir ao vídeo)

Clone e pull

Clone - O clone serve para recuperar qualquer código no Github. Nada se perde uma vez que subimos pro github. É muito útil se você tiver o código fonte num novo computador ou se quiser baixar o código fonte de um projeto interessante que você viu). Tutorial: Vamos apagar a pasta do diretório do nosso computador > pegar o link do repositório no github (https://github.com/nathannieg/gama-no-github) > Executar o comando git clone https://github.com/nathannieg/gama-no-github > Agora o arquivo do projeto estará na pasta selecionada com o nome do repositório no github.

Git checkout -b nomedabranchnova - O git checkout -b cria uma branch e já migra para ela.

Git commit -am "textodedescricao" - Faz o git add e o git commit de uma vez só.

Resolvendo conflitos de commits - Surge quando você faz alterações no mesmo arquivo em momentos diferentes (lê-se: branches diferentes, linhas do tempo diferentes) com conteúdos diferentes no mesmo arquivo. Para resolver, você precisa voltar pro editor de código e ver onde está o conflito (mostra o estado atual do arquivo e qual é o conflito que tá sendo gerado). Você deve escolher qual deles vai considerar (no nosso caso, vai ser o h2) e apagar o outro. Agora, o arquivo terá sofrido uma alteração novamente (pode verificar dando um git status). Por isso, vamos ter que fazer um novo commit para finalizar a resolução do conflito. Ao final, faremos um novo git push para subir a alteração para o servidor remoto (git push).

Dica prática - É muito importante fazer commits aos poucos, com pedaços menores de código. Assim, é possível controlar melhorar as etapas de alterações para resolver mais rapidamente em caso de conflito.

Voltando na linha do tempo para um commit específico dentro do github - Vai no github > abre a lista de commits > copiar o código de verificação do commit (aqui

vamos pegar o endereço do commit anterior ao que fizemos agora - h2 - para mostrar como resolver conflito) > fazer um git checkout identificador.

Pegando alterações do servidor remoto (github) com git pull - editar o código do index.html no github > atualizar o repositório local > conferir se você está no branch master > git pull para atualizar o repositório local.

- Revisão

Git clone - clonar (baixar) um projeto de um repositório remoto.

Git pull - atualiza seu repositório local a partir do seu repositório remoto. É útil para fazer a atualização de conteúdo que outros desenvolvedores estão fazendo.

Ignore

.gitignore - Quando estamos trabalhando em projetos mais complexos, é comum que tenhamos arquivos/pastas que a gente não queira compartilhar com o restante da equipe. O conteúdo permanece no seu computador, mas não é submetido ao servidor github. Exemplo de arquivo que pode ir pro gitignore: usuários de mac geralmente tem um arquivo chamado .DS_Store, que é desnecessário para o projeto em si porque é parte da infraestrutura do SO. Em resumo, o .gitignore serve para avisar que você não quer que x arquivos sejam rastreados pelo git. Para verificar como funciona: cria um arquivo chamado .gitignore > criar uma pasta chamada node_modules > editar o .gitignore e incluir lá o nome "node_modules" > executar o comando git status > o git vai mostrar o .gitignore, mas não vai mostrar a pasta node_modules, que está dentro dela.

```
D:\GitFiles\gama-no-github> git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add" to track)

D:\GitFiles\gama-no-github>
```

Pull request

Acessar: https://github.com/jcbombardelli/gama-no-pullrequest

Essa aula tem como objetivo ensinar você a fazer uma alteração em projetos de terceiros.

Fork - É uma cópia, uma divisão do projeto original, onde você pode fazer suas próprias alterações sem afetar o projeto principal. Quanto mais forks tiver, quer dizer que tem mais gente contribuindo com o seu projeto. No canto superior direito, clicar no Fork e fazer a cópia para o seu repositório.

Pull request - Submeter contribuições para o projeto principal. O dono pode analisar e incorporar essas alterações no corpo do projeto principal. Na branch do projeto em que você fez a alteração, selecionar "new pull request" > direcionar para o repositório base (master) > dar um nome para a alteração > clicar em "create a pull request".

Desafio:

- Cópia com Fork.
- Fazer clone para o servidor local. Comando: https://github.com/nathannieg/gama-no-pullrequest
- Criar uma branch iniciata em gama/seunome. Comando: git checkout -b gama/nathannieg
- Editar o README para incluir suas impressões sobre o curso.
- Fazer o commit do arquivo. Comando: git commit -am "alteração nathannieg" (mas antes precisei dar um cd pra entrar na pasta gama-no-pullrequest)
- Fazer um git push com o comando git push --set-upstream origin gama/seunome. O --set-upstream resolve possíveis problemas de conflito. Comando: git push --set-upstream origin gama/nathannieg
- Voltar ao github para olhar as branches e encontrar a gama/nathannieg.
- Fazer um pullrequest com as suas alterações.

Gitflow

É uma das metodologias mais utilizadas hoje em desenvolvimento de software em time. O objetivo é manter a estabilidade do seu código fonte. Nesse modelo, é possível dar permissões específicas para que somente pessoas autorizadas tenham acesso à determinadas branches.

No Linux, existe um comitê bem específico que tem acesso para submeter códigos à branch master. Isso dá um controle um pouco maior ao que está sendo desenvolvido.

Para contribuir, é importante fazer o que fizemos no desafio: criar uma nova branch, fazer as contribuições e depois enviar por meio de um pull request.

Branch master - Linha master onde todo o código oficial está. Esse código final, que pode pode entrar em produção.

Branch hotfix - Onde ficam as pequenas alterações a partir do lançamento da master são realizadas.

Branch release - É onde juntamos todas as branches de funcionalidades desenvolvidas para depois submeter para a master.

Branch feature - Onde as funcionalidades estão de fato sendo desenvolvidas.

Branch develop - Pode ser usada para homologação, testes, controle etc. A partir da branch develop, nós quebramos em bugs e features. Se você vai fazer uma nova funcionalidade, eu crio ela a partir de develop, não de master. O ideal é que develop tenha sempre todas as novas funcionalidades completas.

Configurando chave SSH

Caminha de onde está a chave: C:\Users\SeuUser\.ssh

Hi seuuser! You've successfully authenticated, but gitHub doest not provide shell access

Curso que apoiou esse: https://app.rocketseat.com.br/node/o-guia-estelar-de-git/group/comecando/lesson/o-primeiro-commit