GIT

O que é GIT?
Git é um sistema de controle de versão de arquivos. Através deles podemos desenvolver projetos na qual diversas pessoas podem contribuir simultaneamente no mesmo, editando e criando novos arquivos e permitindo que os mesmos possam existir sem o risco de suas alterações serem sobrescritas.

Instalando git
O git é um programa que pode ser baixado e instalado para Windows, Mac, ou então através do comando **sudo apt-get install git** para plataformas Linux/Debian, como o Ubuntu.

Configurando o git
Existem 2 pequenos passos para configurar o seu GIT para ter um acesso mais simplificado ao github. Aqui estaremos estabelecendo que, sempre que necessitar, você irá fornecer o seu login e senha ao GitHub.
git config --global user.name "YOUR NAME"
git config --global user.email "YOUR EMAIL ADDRESS"

GitHub

O que é?
O Github é um serviço web que oferece diversas funcionalidades extras aplicadas ao git, é um sistema de gerenciamento de projetos e versões de códigos assim como uma plataforma de rede social criado para desenvolvedores.

Se o Git é o coração do GitHub, então o Hub é a alma. O hub de GitHub é o que torna uma linha de comando, como o Git, a maior rede social para desenvolvedores do mundo.

Criando a conta no GitHub
O github não possui instalação, ele é um serviço, e caso você não tenha uma conta, chegou a hora de criá-la.

Para criar uma conta no GitHub, acesse o site **github.com** e preencha o formulário principal.

DEFINIÇÕES

Repositório
Repositório, ou repo, é um diretório onde os arquivos do seu projeto ficam armazenados. Ele pode ficar em um depósito do GitHub ou em seu computador. Você pode armazenar códigos, imagens, áudios, ou qualquer outra coisa relacionada ao projeto no diretório.
git init para criar um novo repositório

Vamos clonar!
Tendo o git configurado para utilizar o github e um projeto no github criado, precisamos trazer este projeto para o nosso git, e este processo se chama clonar. Então, quando você quiser começar um projeto utilizando git, você cria ele e clona na sua máquina. O comando para clonar o projeto é **git clone "url"**, veja:
git clone https://github.com/<username>/site.git

Dando um Forking em um Repositório
Dar Forking em um repositório significa que você vai criar um novo projeto baseado em repositórios existentes. Em termos simples, dar forking em um repositório quer dizer que você vai copiar um repositório existente, fazer as alterações necessárias, armazenar a nova versão como um novo repositório e chamar de seu projeto.

Essa é uma ótima ferramenta para desenvolvimento de projetos. Por ser um projeto totalmente novo, o diretório central não é afetado. Se o diretório master for atualizado, você também pode aplicar a atualização para seu *fork* atual.

Siga os passos abaixo para dar *fork* em um repositório no GitHub:
1. Encontre o repositório que você deseja dar *fork*.
2. Clique no botão ***Fork***

Comando git add
O comando **add** é usado para adicionar qualquer alteração de arquivo ao INDEX do git, que é uma área especial onde os arquivos estão sendo preparados para o commit. Quando usamos **add**, estamos dizendo que o arquivo estará adicionando ao próximo commit, quando este for realizado.

git add * ou **git add <files>**

Fazendo commit
Um commit é um registro de quais arquivos você alterou. Essencialmente, você faz alterações no seu repo e então diz ao git para colocar esses arquivos em um commit.

O comando utilizado para criar um commit em git é **git commit.** Com esse comando, todos os arquivos que estiverem marcados como **Commit candidate** serão incluídos no commit.

Uma coisa muito importante de qualquer commit é uma mensagem que explique o que significa aquele commit. Para isso, você pode passar a opção **-m "mensagem de commit".** Caso você não passe, o git vai abrir um editor de texto para que você digite a mensagem.

Exemplo:

- **git commit -m "commit"** cria um commit com todos os Commit candidate e a mensagem commit
- **git commit -a -m "segundo commit"** cria um commit com os arquivos marcados como Commit candidate e os modified e junta com a mensagem segundo commit

Errei a mensagem do commit, como arrumo?

Se você já fez o commit, mas ainda não fez o push das suas modificações para o servidor. Nesse caso você usa a flag **--amend**. Fica assim: **$ git commit --amend**

O **git commit --amend** modifica a mensagem do commit mais recente, ou seja, o último commit feito por você no projeto. Além de você mudar a mensagem do commit, você consegue adicionar arquivos que você se esqueceu ou retirar arquivos comitados por engano. O git cria um commit totalmente novo e corrigido.

Branch

Branch é uma cópia do diretório. Você pode usar o *branch* para desenvolver isoladamente.

Trabalhar em um *branch* não irá afetar o repositório central ou outros *branches*. Depois de finalizar o trabalho você pode combinar seu *branch* isolado com outros *branches* através de um *merge*. Para combinar o *branch* isolado ao repositório central utiliza-se o *Pull Request*.

Toda vez que for alterar algo em um projeto versionado com o Git, crie uma branch com o nome da tarefa que está fazendo. Para isso utilize o comando **git checkout -b nome_da_tarefa.**

- Criar um novo branch **git checkout teste**
- Criar e trocar de uma só vez **git checkout -b teste**
- Movendo o working directory para o ambiente de teste **git checkout teste**
- Para retornar **git checkout master**
- Transferindo as alterações do ‘teste’ para o ‘master’ **git merge teste**
- Deletar um branch **git branch -d teste**

Fazendo o merge

Com o commit feito a nova branch tem uma versão diferente da master. Podemos confirmar que estamos com diferença entre master e nossa branch rodando o seguinte comando (ainda na branch nova): **git diff master**

Sabendo que, realmente, a branch nova está com diferenças da master, é preciso fazer um merge para juntar os novos commits, feitos na nova branch, com a master. Para isso você precisa voltar para a master e rodar o comando **git merge.**

Para voltar para a master, usamos o comando **git checkout master** e para fazer o merge, o comando **git merge nome da branch**

Merge com conflitos

Caso tenha conflitos, os resolvemos “na mão” e depois salvamos as alterações com o comando **commit.**

Pull Request

Efetuar um *Pull Request* significa informar outros que você irá implementar as mudanças criadas no seu *branch* ao repositório master. Os colaboradores do repositório podem aceitar ou negar a *Pull Request*. Ao abrir um *Pull Request* você pode revisar e discutir seu trabalho com os colaboradores.

Siga os passos abaixo para criar uma *Pull Request* no GitHub:

1. Navegue até o repositório e encontre o menu *branch*.
2. No menu *branch*, selecione o *branch* que contém o seu trabalho.
3. Clique em **New pull request** ao lado do menu *branch.*
4. Coloque o título e a descrição da *Pull Request.*
5. Clique no botão **Create pull request**.

E o git pull?

Ainda existe um comando importante neste processo, que é o **git pull**. Ele é usado para trazer todas as modificações que estão no repositório para o seu projeto local. Isso é vital quando existem projetos mantidos por mais de uma pessoa, ou se você possui duas máquinas e precisa manter a sincronia entre elas.

Tag e Release

A **tag** é um ponteiro para um commit específico. Esse ponteiro pode ser super carregado com algumas informações adicionais (identidade do criador da tag, uma descrição ...).

A **tag** é um conceito de git enquanto que um **Release** é o conceito de nível mais alto do GitHub.

De acordo com o post de anúncio oficial do blog do GitHub: "Releases são objetos de primeira classe com changelogs e ativos binários que apresentam um histórico de projeto completo além dos artefatos do Git."

- Criar uma tag (no commit atual) **git tag -a <v1.0> -m "Versão 1.0"**
- Criar uma tag (num commit antigo) **git tag -a v0.0 CHAVE -m "Versão 0.0"**
- Para listar as tags **git tag**
- Ver detalhes da tag **git show v0.0**
- Reverter a versão **git checkout v0.0**

Comandos

git init : Server para criar um novo repositório.
Como usar : Criar uma pasta e abrir no terminal e digite o comando git init.

git clone : Serve para clonar um repositório já existente.
Como usar : Crie uma pasta e abra no terminal então digite git clone https://github.com/caminho/para/repositório.

Início do Fluxo:

git add : Serve para adicionar uma mudança em seu repositório.
Como usar : Abra a pasta do seu repositório no terminal e digite git add * (se for adicionar todas as mudanças feitas) ou git add <arquivo> para adicionar uma única mudança a ser feita.

git commit -m : Serve para você realmente confirma as alterações.
Como usar : Após usar o comando git add você usa o git commit -m para aprovar as mudanças e escreva uma mensagem falando sobre oque é esse commit exemplo git commit -m “frontend - navbar debug”.

git push : Serve para você enviar as alterações para o repositório.
Como usar : Após usar o comando git commit você usa o git push origin <branch que você está trabalhando> .

Fim do Fluxo:

git checkout -b <nome da branch> : Serve para você criar uma branch chamada test e usá la.

git checkout <branch> : Serve para você seleciona a branch que você quer usar.

git branch -d <nome da branch> : Serve para você deletar um branch.

git push origin <nome da branch> : Após você criar uma branch ela não existe para os outros membros então para aparecer você pode usar o comando git push origin <nome da branch>

git pull : Serve para você baixar as alterações feitas no repositório.

git merge <nome da branch> : Serve para você mesclar a branch com outra branch ativa.

git tag <nome> <id da issues> : Serve para você adicionar um tag é recomendado para rotular releases do software.

git log : Serve para você pegar o id das issues.

git checkout -- <arquivo> : Serve para sobrescrever algum arquivo com a sua versão mais recente ‘é usado quando é feito algo errado’

Caso você queira remover todas commits locais e recuperar o histórico mais recente do servidor use os comandos : **git fetch origin** depois **git reset --hard origin/master.**