workshop

# Montando seu primeiro portfolio online **gratuito**

# Tópicos principais

- Introdução;
- Linha de comando linux;
- Comandos GIT básicos;
- Criar conta no **Github**;
- Contribuindo no Github (projetos **Open Source**);
- Criando domínio e hospedando com **Github Pages**;
- Estruturando portfólio e layout;
- Codificando em HTML, CSS e JS (landing page).

# Olá

**Meu nome é Rebeca**

Sou Full Stack Web Developer

Formada em Redes de computadores e cursando Análise e Desenvolvimento de Sistemas.

Meu github é https://github.com/rebecagrn

Meu site/portfolio é https://rebecag.com

## **DICA 01** – Use o Terminal

Se você é programador, deve passar boa parte do seu tempo atrás de uma tela de terminal. Aprenda a usá-lo.

# **DICA 02** – Aprenda Linux/Unix

A maioria dos ambientes (reais) de desenvolvimento e produção estão abaixo de um ecossistema Linux/Unix. Você irá precisar disso, mais cedo ou mais tarde.

Windows? Máquina Virtual ou Bash.

## “ **DICA 03** - Aprenda um pouco sobre Design

Isso fez muita diferença durante todo o tempo que trabalhei. Programadores normais são ótimos, mas programadores que manjam de design conseguem entregar algo bonito, que é (as vezes) mais importante.

## **DICA 04** - Crie o seu GitHub hoje!

Empresas estão sempre de olho no que vocês andam “codando”. Exponha todos seus estudos. Crie repositório com links, tutoriais e coisas do tipo. Sabe aquele problema que você demorou para resolver? Faça um repositório com esse tutorial. Além de ajudar outras pessoas, pode ajudar você mesmo no futuro.

## **DICA 05** - Contribua com a comunidade

Trabalhar em projetos Open Source é uma ótima maneira de mostrar o que você sabe. Contribua sempre que possível com qualquer tipo de projeto que achar que pode melhorar. Seja traduzindo, refatorando, etc...

## “ **DICA 06** - A melhor linguagem é o Inglês

Sim, isso é importante. Quando tiver que mostrar seu código, pode ser para uma empresa de fora… Sempre faça tudo em Inglês! Aprenda a ler e escrever em inglês, falar também é bom. A maioria do conteúdo relevante na internet está em Inglês.

# 1. GITHUB

O GitHub foi adotado pela comunidade de desenvolvimento;
Com o GitHub é possível criar um perfil pessoal, para organizar seus projetos pessoais;
Ou ainda, criar uma **organization** (um grupo que pode ter vários colaboradores) e também seus próprios projetos.

# 1.1 GITHUB

O **GitHub** é uma rede social, por isso, é possível seguir e ser seguido dentro da plataforma;
Você consegue acompanhar em um feed com a atividade de cada um dos usuários que você segue;

# 1.2 GITHUB

**As Stars valem ouro** → no GitHub os “likes” são medidos em estrelas, quanto mais estrelas seu projeto tiver, mais relevante ele é visto pela comunidade do GitHub;

Em seu perfil, estão: seus repositório principais, suas organizações, um resumo de duas contribuições, e uma timeline de suas atividades;

É possível navegar entre seus repositórios, as estrelas que você marcou, seus seguidores, e as pessoas que você segue.

# 1.3 GITHUB

- É nele que a maioria dos projetos (inclusive os grandes) estão organizados
- Utiliza notações MarkDown para criação de sua documentação
- Segundo a fonte científica wikipedia → Markdown
  é uma linguagem simples de marcação originalmente criada por John Gruber e Aaron Swartz Markdown converte seu texto em XHTML válido.

# COMANDOS LINUX

| | |
|---|---|
| `du -hs *` | mostra o espaço usado por cada arquivo/diretório de usuário |
| `ls` | lista conteúdo do diretório local |
| `ls -l` | mostra conteúdo detalhado |
| `ls -a` | mostra arquivos escondidos |
| `ls -la` | combinação de ambos |
| `mkdir` | cria diretório |
| `cp` | copia arquivo |
| `cp -r` | copia recursivamente (para copiar diretórios) |

# COMANDOS LINUX

| | |
|---|---|
| `mv` | mover ou renomear arquivo/diretório |
| `rm` | apaga arquivo |
| `rm -r` | apaga recursivamente |
| `rm -rf` | apaga recursivamente sem confirmação (use com cuidado!) |
| `pwd` | mostra o diretório atual |
| `cat, more ou less` | mostram conteúdos de arquivo |
| `tail` | mostra final de arquivo |
| `head` | mostra começo de arquivo |
| `ssh` | acessa outra máquina Linux via protocolo seguro SSH |
| `nano` | abre o editor *nano* |
| `touch` | cria um arquivo em branco |

# 2 – O que é controle de versão?

- Já se deparou com cópias e mais cópias do mesmo projeto?
- Cada cópia com pequenas modificações?
- Já apagou um arquivo sem querer e depois descobriu que não dava para recuperar?
- Já teve que trabalhar em equipe e trocar o projeto por e-mail ou whatsapp?
- Já desenvolveu algo que ficou pior que a versão anterior?

# 2.1 – Controle de versão

- Solução: **Controle de Versão**!
- O que é? → é um sistema com a finalidade de gerenciar diferentes versões do mesmo arquivo;
- Cada modificação (no projeto) gera uma nova versão;
- Podemos imaginar que a todo momento o versionador de arquivos tira uma foto de todo nosso sistema e guarda isso;
- Caso seja preciso, podemos recuperar o estado no momento em que a foto foi tirada;

# 2.2 – Ecossistema

Existem sistemas que fazem o controle desse ecossistema:

- **Git**
- Svn
- Mercurial
- Bazaar

# 2.3 – Vantagens

Existem sistemas que fazem o controle desse ecossistema:

- Vocês podem trabalhar em diferentes branchs e depois juntar tudo;
- Se perder alguma coisa, ou fizer algo errado… é só voltar;
- Vocês saberão o que cada um fez e em que parte o arquivo foi modificado;
- Podem clonar o repositório no ambiente de produção.

# 2.4 – Git é o Github?

Não! **GIT** é uma tecnologia para versionamento de arquivos;

- GitHub é uma plataforma que utiliza o git para armazenar os projetos;
- Ou ainda... uma rede social para programadores?
- Mostra seus códigos para todos;
- Participe de projetos OpenSource;
- Repositórios privados gratuitos.

# Instalando o GIT

Instale pelo site oficial: **https://git-scm.com/download**

# GIT - Configuração básica

Depois de instalado, podemos fazer algumas configurações básicas, em um terminal:

- `git config --global user.name "Seu nome"`
- `git config --global user.email "email@gmail.com"`
- Se algo der errado, será mostrado na tela, são essas informações que serão utilizadas para enviar o projeto para o repositório;
- Possivelmente você pode configurar chaves SSH para permitir uma autenticação automática com seu ambiente remoto Git.

# GIT – Iniciando um projeto

- Crie uma pasta: `mkdir project`
- Entre nessa pasta: `cd nome-do-projeto`
- Para inicializar o repositório: `git init`
- Listar arquivos existentes: `ls`

# GIT – Adicionando arquivos

Quando se cria um novo arquivo ele não está sendo visto pelo **Git** → ele se encontra no status (untracked).

Quando se adiciona um arquivo:

`git add file.ext` → adiciona um único arquivo

`git add –all` ou `git add .` → adiciona todos arquivos que foram modificados;

Nesse momento ele passa para o estágio → (stage)

# GIT - básico

Pode-se observar os estados dos arquivos com o comando: git status;

Um commit só "leva" arquivos que estão no status (stage); nesse momento (stage) o git sabe de sua existência, mas não existe nenhum "commit" para essa versão.

Um commit é a criação de um snapshot do sistema, ou seja, um *"printscreen"* da situação atual do seu sistema.

# GIT - básico

para realizar um commit: `git commit -m "mensagem"`

quando se faz um *commit*, tudo está ok, mas só na sua máquina.

`git remote add origin <endereço-do-repositório`

*origin* pode ser qualquer nome, mas é utilizado como **padrão**;

Um mesmo repositório pode ter diferentes locais remotos;

para ver a lista de repositórios remotos: `git remote`

# GIT - Push no repositório

Já temos nosso commit, e um repositório linkado; precisamos enviar isso!

```
git push -u origin master
```

-u “trackeia” o comando e possibilita que os próximos comandos sejam apenas *git push*;

***origin*** é o nosso repositório destino;

***master*** é o branch que estamos;

***master*** é sempre o branch default;.

Podemos também, ao invés de criar um repositório e ligá-lo a uma origem, cloná-lo:

```
git clone <endereço-do-repositório>
```

isso deixa nosso repositório pronto para realizar os *commit’s* e *push’s*;

# GIT – Clonando repositórios

Com o comando:

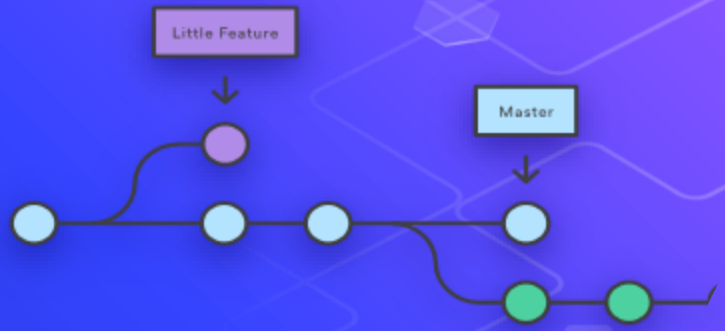
git clone <nome-do-repositorio>, podemos ligá-lo para o mesmo ou outro repositório remoto.

# PRÁTICA GIT – CRIANDO E LIGANDO A REPO

```
mkdir femtec-project
git init
touch femtec-project.html
git add .
git commit -m “Primeiro commit”
git remote add origin ...
git push -u origin master
```

```
git clone ...
touch myproject.html
git add .
git commit -m “Enviando arquivo”
git push
```

# 3 - Branch




O que é?

É um ramo para determinada *snapshot* do seu sistema de arquivos;

Por que usar?

Modifica arquivos sem alterar o fluxo principal;

Pode-se corrigi um bug em determinada funcionalidade sem interferir no fluxo original do projeto.

# 3.1 - Branch


E como juntar com o projeto principal?

- Através do comando *merge;*
- Branch master é a *branch* com o código final do projeto, estável.
- Criando uma nova branch, ao submeter o *pull request* para o repositório original, caso não for aceito, as alterações não estarão na branch *master*.

# GIT – Criando uma branch

Para criar um novo branch: primeiro tenha certeza que você está no master: `git branch`;

Depois cria-se uma nova branch: `git branch minha_branch` ;

Entre na nova branch: `git checkout minha_branch`.

Para navegar entre os branchs: `git checkout minha_branch`;

`git checkout master`; Para remover um branch dê o comando: `git branch -D minha_branch`.

# GIT – Revertendo erros

**git reset --soft**

Mata o commit mas o arquivo estará em staged; deixa o arquivo pronto para ser commitado novamente.

**git reset --mixed**

Mata o commit mas deixa os arquivos antes do staged; será preciso dar o add novamente.

**git reset --hard**

Mata o commit e todas as suas modificações.

# 4 - Merge e Rebase

Depois que já criamos nossas funcionalidades em nosso branch separado, é hora de fazer a união entre os branchs;

Existem 2 métodos: **merge** e o **rebase,** fazem basicamente a mesma coisa; de formas diferentes; o merge é interessante em *pull requests* e novas features, pois sabemos exatamente de onde veio.

# 4.1 - Merge e Rebase

O **rebase**:

É utilizado quando não é necessário saber quando a alteração foi feita *(cronologicamente)*;

Ambos podem ser usados para juntar diversas funcionalidades timeline principal de seu projeto;

# 4.2 - Merge e Rebase

```
$ mkdir rebase-merge
$ cd rebase-merge
$ git init
$ nano teste.txt
$ git add teste.txt
$ git commit -m “add att”
```

```
$ git checkout -b teste
$ nano bar
$ git add exemplo.html
$ git commit -m “Add updates”
```

# 5 - Fork

O que é? é uma cópia de um projeto para sua conta do GitHub.

Isso é ideal para realizar contribuições. Por quê?

Você pode fazer um *pull-request* ao dono do repositório que quer contribuir;

Como fazer isso? basta ir ao repositório no **GitHub**, estando logado, e clicar em **fork**;

Com o *fork* é possível contribuir em repositórios de terceiros.

# 6.1 – Fazer o fork do repositório

Clonar o repositório para a sua máquina

Criando um branch → neste momento estamos criando um fluxo separado do principal, para que suas modificações não impactem diretamente no projeto;

Primeiro tenha certeza que você está no master: `git branch`

Cria-se uma nova branch: `git branch add_my_name`

Entra-se nesta branch: `git checkout add_my_name`

Agora pode-se criar, editar e modificar os arquivos de acordo com sua necessidade;

# 6.2 – Enviando o Pull-Request

Edite o arquivo **README.MD** e adicione o seu nome completo;

```
git add .
```

```
git commit -m "Alterações"
```

Enviando a branch: `git push origin add_my_name`

Acesse sua conta no GitHub

Basta clicar no botão verde: **Compare & Pull Request**;

Será direcionado para uma tela onde irá poder criar de fato o seu pull-request ● Coloque um título, uma descrição, e pronto!

# 7- Hospedando sua página no Github Pages

- Só é possível hospedar arquivos estáticos: **HTML, CSS e JS**;
- vamos a uma passo a passo de como fazer isso: **passo 1:** crie um repositório ou pode ser um repositório já existente
- **passo 2:** vá até settings > GitHub Pages;
- **passo 3:** selecione um branch (você pode usar o *master*);

# 7.1 - Hospedando sua página no Github Pages

- Feito isso sua página já pode ser vista em: **seuuser.github.io/seu-repositorio**;
- Podemos ainda definir um domínio para apontar para o GitHub;  apenas criaremos um arquivo chamado **CNAME** na raiz do seu repositório com seu próprio domínio e depois disso, só apontar para os servidores do GitHub no gerenciador do seu domínio e pronto. temos um site grátis para sempre!

# Obrigada!

**Dúvidas?**

Pode me encontrar em:

https://github.com/rebecagrn

rebecagpacheco@outlook.com