

# Git - Conflitos e mais Conflitos

Labenu\_



# O que vamos ver hoje?

- Breve revisão
- .gitignore
- Resolvendo conflitos



# Retomando alguns comandos

Labenu\_



# Retomando alguns comandos

- `git status`
  - Usado para verificar o status atual do repositório
  - Indica se está atualizado, quais foram as modificações e mais
- `git add`
  - Adiciona os arquivos na área de staging
  - Opção `--all` para pegar todos os arquivos
  - `git add .` para pegar tudo abaixo do diretório atual



# Retomando alguns comandos

- `git commit -m "mensagem"`
  - Persiste as modificações que estão em staging
  - É necessário inserir uma mensagem identificando com clareza o que foi feito pelo(a) dev
- `git push`
  - Manda modificações para um repositório remoto



# Retomando alguns comandos

- `git pull`
  - Pega as modificações de um repositório remoto
- `git branch`
  - Permite ver uma lista com as branches
  - Se colocar um `nome_da_branch` depois, ele cria uma branch com esse nome



# Retomando alguns comandos

- `git checkout nome_da_branch`
  - Muda para outra branch existente
- `git checkout -b nome_da_branch`
  - Cria uma nova branch e já entra nela
  - Basicamente junta os dois comandos mostrados anteriormente



# .gitignore

Labenu\_





# .gitignore 🖐️

- O .gitignore (o ponto no início é importantíssimo) é onde colocamos os nomes dos arquivos que devem ser ignorados pelo git
- Ou seja, é a lista dos arquivos cujas modificações **não entrarão nos commits** e nem no repositório remoto



# .gitignore 🖐️

- A sintaxe dele é bem simples, basta colocar o arquivo em questão no .gitignore
- O símbolo **\*** indica generalização (tudo)
  - **\*\*** -> ignora todos o conteúdo de um diretório. Ex: /pasta/\*\*
  - **\*** -> marca caracteres correspondentes. Ex: \*.js vai selecionar arquivos que contenham .js no nome. Ex: pasta/\*.js (ignora arquivos terminados em .js dentro da pasta).
- O símbolo **#** é usado para indicar comentários
- Para mais informações, consulte [esse link](#) e [esse](#)



# .gitignore 🖐️

- Usamos isto quando não queremos que certas informações estejam no repositório remoto:
  - node\_modules
  - arquivos de build
  - informações secretas
  - arquivos específicos da sua máquina



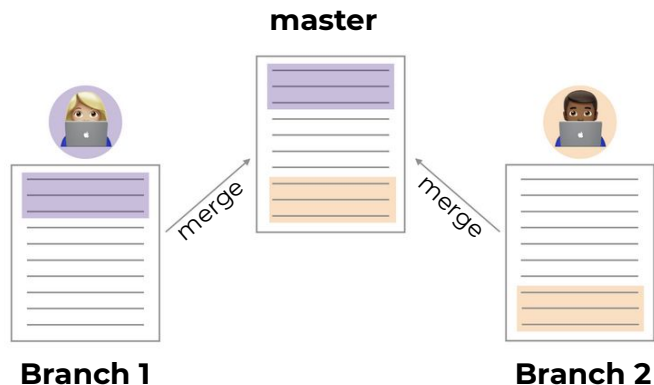
# Quando surgem os conflitos?

Labenu\_



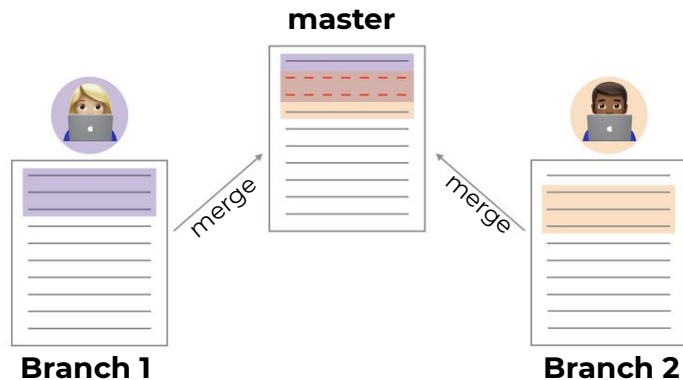
# Conflitos

- Normalmente os conflitos surgem quando estamos trabalhando com mais de uma pessoa no mesmo código.
- Quando isso acontece, é comum mexermos nos mesmos arquivos. O git consegue administrar esse processo de unir dois códigos escritos por duas pessoas no mesmo arquivo. 🙌🙌🙌



# Conflitos

- Porém, quando duas pessoas acabam editando as mesmas linhas de código, o git não consegue descobrir qual mudança ele deve manter, gerando o conflito.
- A primeira branch a ser mergeada na master não encontra problemas, mas na hora do merge da segunda branch o conflito vai surgir. 🙅🙅🙅



# Resolvendo Conflitos

Labenu\_



# Resolvendo Conflitos

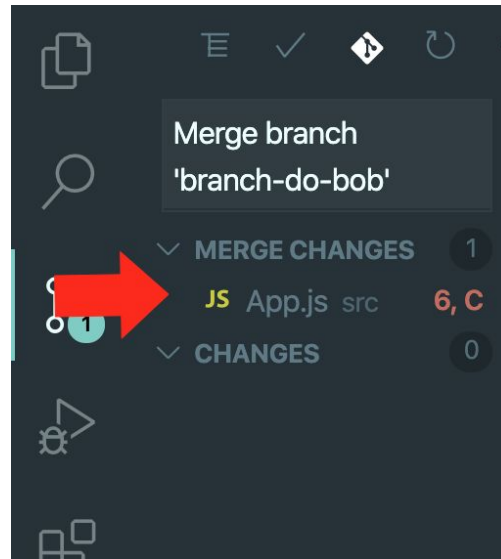
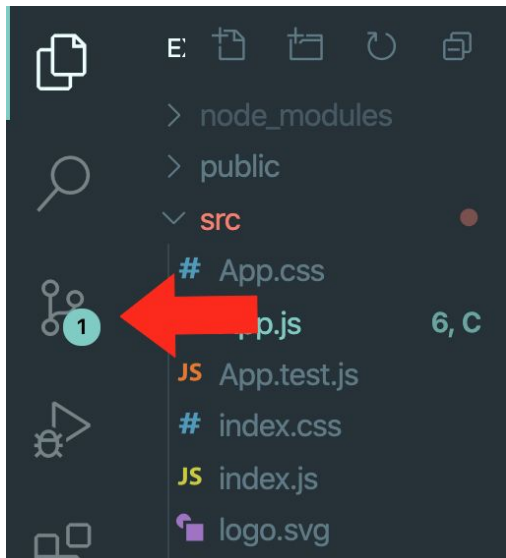
- Vamos resolver manualmente usando o comando `git merge`
  1. `git pull` na master
  2. Ir para a branch de destino (onde foi apontado o conflito)  
`git checkout branch2`
  3. Então, rodar o comando `git merge master` pois queremos puxar a master para a branch de destino





# Resolvendo Conflitos

- Abra o VSCode e vá na seção de git. Lá, ele mostra todos os arquivos com conflito



# Resolvendo Conflitos

- **Resolva todos os conflitos** editando os arquivos
- Se você não reconhece algum código, **converse com os colegas** que escreveram aquele código para resolver



# Theme

```
function App() {
```

```
  return (
```

```
    <div>
```

Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

```
<<<<<< HEAD (Current Change)
```



```
<h1>Login</h1>
```

```
<button>Clique para fazer login</button>
```

```
=====
```

```
<h1>Página de Login</h1>
```

```
<button>Login</button>
```

```
>>>>>> branch-do-bob (Incoming Change)
```

```
</div>
```

You, 7 minutes ago • Create project

```
);
```

```
}
```



# Resolvendo Conflitos

- Agora, você pode simplesmente dar um **add** e um **commit** nos seus arquivos normalmente
- O merge está concluído
- Lembre-se de rodar o git push para que o merge seja feito no GitHub também



# Resumo

- `git checkout master`
- `git pull`
- `git checkout branch-origem`
- `git merge master`
- `-- resolver conflitos --`
- `git add .`
- `git commit -m "Merge branch-origem"`
- `git push`





Obrigado(a)!