escola britânica de artes criativas & tecnologia

Analista de Dados

Controle de Versão III

Módulo | Análise de Dados: Controle de Versão III

Caderno de Aula

Professor André Perez

Tópicos

1. Sistema de branchs;
2. Trabalhando com branchs;
3. Mover código entre branchs.

Aulas

0. Setup

0.1. Autenticação

In [ ]:

```
import os

username = "andre-marcos-perez"
os.environ["GITHUB_USER"] = username

!git config --global user.name "${GITHUB_USER}"
```

In [ ]:
```
import os
from getpass import getpass

usermail = getpass()
os.environ["GITHUB_MAIL"] = usermail

!git config --global user.email "${GITHUB_MAIL}"
```

In [ ]:
```
import os
from getpass import getpass

usertoken = getpass()
os.environ["GITHUB_TOKEN"] = usertoken
```

0.2. Projeto

In [ ]:
```
!git clone https://${GITHUB_USER}:${GITHUB_TOKEN}@github.com/andre-marcos-
```

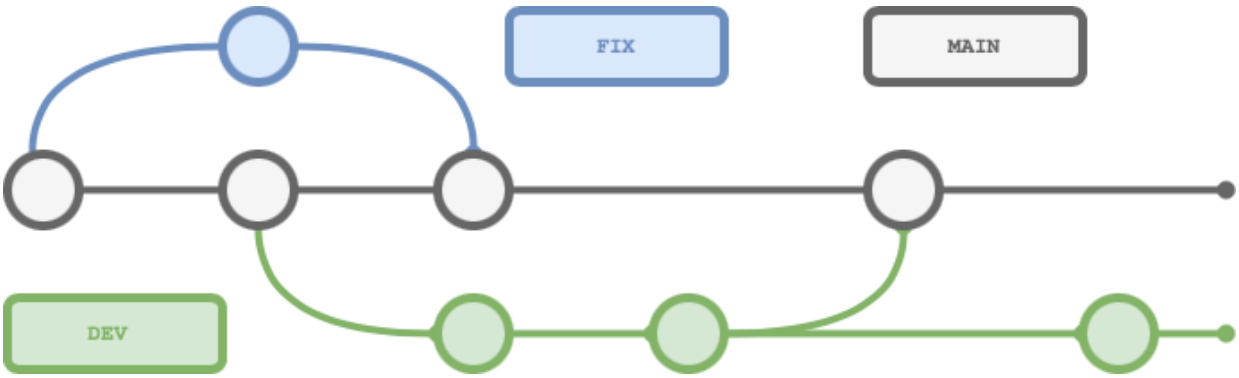
In [ ]:
```
%cd /content/da-ebac/
```

1. Sistema de branchs

O sistema de *branchs* ou ramificações é uma funcionalidade do `git` que permite a criação de uma versão (ou ramo) do código do repositório. De maneira geral, todo novo repositório possui uma *branch* base com o nome *main* (voce também pode encontra-la com o antigo nome de *master*).

A vantagem do uso desse sistema é que mudanças no código principal ou *branch* main, como novas funcionalidades e correções, podem ser desenvolvidadas e testadas em **paralelo** e de maneira **isolada**.

> O processo de **criação** de *branchs* chama-se **checkout**
>
> O processo de **junção** de *branchs* chama-se **merge**



2. Trabalhando com branchs

2.1. Listar

O comando `git branch` (doc) lista as *branchs* do repositório. Os usos mais comuns do comando são:

> `git branch`
>
> `git branch -a`

Exemplo:

In [ ]:
```
!git branch
```

In [ ]:
```
!git branch -a
```

Já o comando `git fetch` (doc) atualiza os metadados das *branchs* no repositório local. O uso mais comunm do comando é:

> `git fetch --prune`

Exemplo:

In [ ]:
```
!git fetch --prune
```

2.2. Criar

O comando `git checkout` (doc) criar novas *branchs*. Ele também é utilizado para mudar o contexto do desenvolvimento entre *branchs*. *Branchs* remotas são criadas através do comando `git push`, caso não existam. Os usos mais comuns do comando são:

> `git checkout <nome-da-branch>`
>
> `git checkout -b <nome-da-nova-branch> <nome-da-branch-referencia>`

Exemplo:

In [ ]:
```
!git branch -a
```

In [ ]:
```
!git checkout -b dev main
```

In [ ]:
```
!git branch -a
```

In [ ]:
```
# alterar o arquivo hello.py
```

In [ ]:
```
!git status
```

In [ ]:
```
!git add hello.py
```

In [ ]:
```
!git commit -m "arquivo hello.py alterado em dev"
```

In [ ]:
```
!git status
```

In [ ]:
```
!git push origin dev
```

In [ ]:
```
!git checkout main
!git branch -a
```

In [ ]:
```
!git checkout dev
!git branch -a
```

3. Movendo código entre branchs

3.1. Remoto

Mover códigos entre as áreas *remote* de *branchs* é feito através de um ação conhecida como *pull request* (PR) no GitHub ou *merge request* (MR) no GitLab. O PR (ou MR) é feito através da interface web da ferramente online de `git` e é a forma **mais comum** de mover código entre *branchs*. Em geral, o fluxo é o seguinte:

1. `git checkout` da nova *branch* a partir da original;
2. Trabalho local + `git add` + `git commit`;
3. `git push` da nova *branch*;
4. Criar o PR (ou MR);
5. Discute o PR com pares;
6. Aprova o MR;
7. `git checkout` + `git pull` na *branch* de original.

3.2. Local

O comando `git merge` (doc) move arquivos entre branchs. O uso mais comum do comando é:

> `git merge <nome-da-branch>`

Exemplo:

In [ ]:
```
!git checkout main
!git branch -a
```

In [ ]:
```
!git merge dev
```

3.3. Excluir

Para excluir uma *branch* no repositório local, utiliza-se o comando:

> `git branch -d <nome-da-branch>`

Exemplo:

In [ ]:
```
!git branch -d dev
```

Já Para excluir uma *branch* no repositório remoto, utiliza-se o comando:

> `git push origin --delete <nome-da-branch>`

Exemplo:

In [ ]:
```
!git push origin --delete dev
```