



escola  
britânica de  
artes criativas  
& tecnologia

## Módulo | Análise de Dados: Controle de Versão II

Caderno de Aula

Professor [André Perez](#)

### Tópicos

1. Adicionar e salvar;
2. Visualizar e reverter;
3. Persistir e atualizar.

### 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 \br/>https://${GITHUB_USER}:${GITHUB_TOKEN}@github.com/andre-marcos-perez/da-ebac.
```

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

## 1. Adicionar e salvar

### 1.1. Fluxo

O `git` define um fluxo de trabalho para manter o rastreamento das ações (criar, modificar, salvar, etc.) realizadas nos arquivos de um repositório. Neste fluxo, arquivos são movidos entre **áreas** ou **zonas** dependendo da ação que é realizada. São elas:

1. **working**: trabalho;
2. **staging**: preparação;
3. **repository**: alterações salvas localmente;
4. **remote**: alterações salvas remotamente.

### 1.2. Adicionar

O comando `git add` (`doc`) move arquivos da *working* para *staging* área. Se um arquivo for alterado/removido após ter sido adicionado, este deve ser adicionado novamente. Os usos mais comuns do comando são:

```
git add <nome-do-arquivo-1> <nome-do-arquivo-2> ...  
git add <nome-do-dir>
```

Exemplo:

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

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

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

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

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

O arquivo `.gitignore` é utilizado para indicar ao `git` quais arquivos devem ser ignorados pelo comando `git add`.

```
In [ ]: !head -n 10 .gitignore
```

### 1.3. Salvar

O comando `git commit` ([doc](#)) move arquivos da *staging* para *repository* área. A todo `commit` é atribuído uma chave identificadora única para rastreamento (*hash*). Ações nos arquivos "comitados" são salvas no repositório local dentro do diretório `.git`. O uso mais comum do comando é:

```
git commit -m "<mensagem-descrevendo-as-alterações>"
```

Exemplo:

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

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

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

## 2. Visualizar e reverter

### 2.1. Visualizar

O comando `git log` ([doc](#)) lista os últimos *commits* (id, data, autor, mensagem, etc.) em ordem cronológica. Os usos mais comuns do comando são:

```
git log
git log <nome-do-arquivo>
```

Exemplo:

```
In [ ]: !git log
```

```
In [ ]: !git log --oneline
```

Já o comando `git diff` ([doc](#)) mostra as diferenças entre um arquivo na *working* com a *staging* ou *repository* área, ou seja, entre a versão recentemente alterada com a última versão salva. O uso mais comum do comando é:

```
git diff <nome-do-arquivo>
```

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

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

## 2.2. Reverter

O comando `git reset` ([doc](#)) move arquivos da *staging* de volta para a *working* área, essencialmente desfazendo o comando `git add`. Os usos mais comuns do comando são:

```
git reset  
git reset <nome-do-arquivo>
```

Exemplo:

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

```
In [ ]: !git reset
```

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

Já o comando `git checkout` ([doc](#)) move arquivos da *repository* de volta para a *working* área, essencialmente desfazendo qualquer alteração feita nos arquivos. Os usos mais comuns do comando são:

```
git checkout  
git checkout <nome-do-arquivo>
```

Exemplo:

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

```
In [ ]: !git checkout
```

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

## 3. Persistir e atualizar

### 3.1. Persistir

O comando `git push` ([doc](#)) move arquivos da *repository* para *remote* área, salvando assim as alterações "comitadas" localmente no servidor `git` remoto, como o GitHub. O uso mais comum do comando é:

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

Exemplo:

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

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

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

## 3.2. Atualizar

O comando `git pull` ([doc](#)) faz o movimento contrário do `git push`, movendo arquivos da *remote* para *repository* área, atualizando assim o projeto localmente. O uso mais comum do comando é:

```
! git pull
```

Exemplo:

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

```
In [ ]: !git pull
```

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