Atividade 1: Git Local

Pré-requisitos:

- Instalar o python
- Instalar o GitHub Desktop

Para instalar o python, siga o procedimento indicado no site: https://www.python.org/downloads/

Para instalar o GitHub Desktop, siga o procedimento indicado no site: https://docs.github.com/pt/desktop/installing-and-authenticating-to-github-desktop/installing-github-desktop

Ambiente:

Defina um local para colocar os seus arquivos deste estudo.

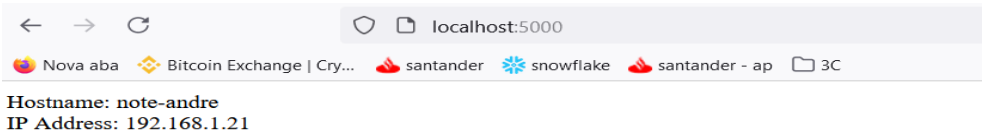
Vou usar o Desktop do Windows como exemplo, mas adapte para o seu sistema operacional conforme necessário.

Arquivos que vamos trabalhar:

Vamos reaproveitar a aplicação web em python utilizada na aula anterior.

Caso não tenha os arquivos, utilize o que foi disponibilizado no portal (atividade_git.zip)

- Crie uma pasta `Impacta` no seu Desktop;
- Dentro desta pasta, crie uma nova pasta com o nome `atividade_git`;
- Coloque o conteúdo da aplicação (ou do arquivo zip) neste diretório;
    - Obs.: o arquivo zip não contém o diretório do ambiente virtual (venv). Siga os passos abaixo para criar este:
        - Abra o terminal
        - Navegue até o diretório da aplicação. EX(C:\Users\andre.moraes\Desktop\Impacta\atividade_git)
        - Crie o ambiente virtual: `python -m venv venv`
        - Ative o ambiente virtual: `venv\Scripts\activate.bat`
        - Instale os pacotes necessários da aplicação: `pip install -r requirements.txt`
        - Execute a aplicação python app.py
        - Se tudo der certo, e não tiver nenhuma mensagem de erro, será possível abrir o navegador no endereço http://localhost:5000/



Iniciando o GIT:

No diretório da aplicação, digite: `git init`

Observe que foi criado o diretório `.git`

Verificando o status:

Digite: `git status`

Observe que todos os arquivos não foram mapeados pelo git, ou seja, são novos!

Adicionando arquivos:

Digite: `git add .` (não esqueça do ponto no final)

Verifique o status agora: `git status`

Observe que todos os arquivos, inclusive o diretório das variáveis de ambiente foram adicionadas e isto não queremos.

Removendo os arquivos, antes do commit.

Digite: `git rm –cached -r venv/*`

Verifique o status: `git status`

Observe que agora nossos arquivos estão no git conforme gostaríamos e o venv não.

Fazendo o commit.

Digite: `git commit -m “primeiro commit”`

Observe o log do commit

Verifique o status: `git status`

Verificando o log.

Digite: `git log`

Observe como ficou o log do primeiro commit

Criando uma Branch

Crie uma Branch com o nome gitignore, aonde vamos criar um arquivo .gitignore e informar que não qureremos enviar para o git e, portanto, não queremos fazer o controle de versão do diretório com as variáveis de ambiente do git.

Digite o comando: `git checkout -b gitignore`

Verifique o status da Branch aonde você se encontra: `git status`

Observe que no status ainda temos o diretório venv como mapeado mas não adicionado ao git

Crie o arquivo `.gitignore` e coloque em seu conteúdo:

```
venv
```

Verifique o status da Branch: `git status`

Observe que o diretório que pedimos para ignorar foi removido do status e que o arquivo .gitignore está disponível para adicionarmos ao git e fazer o commit.

Adicione e faça commit dos arquivos:

```
git add .
git commit -m “adicionando o gitignore”
```

Verifique o LOG:

```
git log
git log –-graph
git log --pretty=oneline
git log -p
git log -2
git log -1
```

Merge com a master

Liste as Branch do seu repositório: `git branch`

Volte para a Branch master: `git checkout master`

Faça o merge: `git merge gitignore`

Se tiver algum conflito nos arquivos é agora que aparece!

Verifique o status: `git status`

Atividade 2: Git Remoto

Salvando todo o seu projeto no github.

Faça login no https://github.com/

Crie um repositório novo com o nome Impacta_DataOps

Volte ao seu terminal e adicione a origem do seu repositório:

Ex: git remote add origin https://github.com/andrericsouza/Impacta_DataOps.git

Confira se foi cadastrado ok: git remote -v

Faça o envio dos arquivos para o github:

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

Irá surgir uma tela perguntando se deseja fazer login no github com o navegador.

Feito o login, observe no github que os arquivos já estão disponíveis.

Adicione o README ao repositório.

Baixe o arquivo README da atividade disponibilizada.

Garanta que esteja na Branch master: git branch

Se precisar: git branch master

Crie uma Branch nova para incluir o arquivo readme: git checkout -b readme

Inclua o arquivo

Adicione e envie o arquivo para o github

```
git add .

git commit -m “adicionando readme”

git push
```

Se necessário, execute o comando: git push --set-upstream origin readme

Observe que no git irá surgir uma mensagem que temos uma atualização na Branch readme e o botão para fazer o PR.

Click no botão “Compare & pull request”

Veja as diferenças que o sistema inferiu. Estando ok clique em “Create pull request”

Faça o MERGE com o click no botão “Merge pull request” e “Confirm merge”

Volte ao inicio do repositório e observe o readme.

No terminal, volte à Branch master: git checkout master

Sincronize o repositório local com o remoto: git pull

Observe o log

Automatizando tarefas no GIT

Crie uma nova Branch para incluirmos a rotina automática de validação de códigos python (black)

`git checkout -b automacao`

Crie os diretório conforme abaixo:

`mkdir .github` (sim, tem um ponto no inicio do nome do diretório)

`cd .github` (não esqueça do ponto)

```
mkdir workflows
cd workflows
```

crie o arquivo de automação do black: `black.yml`

coloque este conteúdo nele:

```
name: Black Code Style Check

on:
  push:
    branches:
      - '**'
  pull_request:
    branches:
      - '**'

jobs:
  black:
    runs-on: ubuntu-latest

    steps:
      - name: Check out repository
        uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
```

```
      python-version: '3.x'

  - name: Install dependencies
    run: |
      pip install black

  - name: Run Black
    run: |
      black --check .
```

volte ao diretório inicial do seu projeto git e digite o comando: black . --check

observe o log.

Vamos executar o comando agora para corrigir o arquivo antes de enviar para o github.

Digite: black .

Para verificar as mudanças que foram feitas, digite: git diff <nome do arquivo corrigido>

Verifique o status da sua Branch: git status

Adicione, faça o commit e envie para o git:

```
Git add .
Git commit -m “fix black + automacao black”
Git push
```

Se necessário, digite o comando: git push --set-upstream origin automacao

No github, faça o merge.

Acesse no github agora o menu Actions

Observe na esquerda que temos a nossa etapa de automação já disponível.

Acesse o menu Settings do seu repositório.

Vá na opção Rules.

Crie uma nova Rulesets

Coloque o nome que desejar.

Deixe com o status Active

Em target coloque a Branch Default

Marque a opção Require status checks to pass

Adicione a role black criada anteriormente.

Salve e retorne as configurações do repositório.

Mude a visibilidade para public

Vamos forçar o black a causar um erro agora.

Altere o arquivo app.py colocando um espaço à mais na linha 15 depois do ==

Troque também as aspas duplas por aspas simples.

No terminal, verifique com o black, mas não faça a correção: black .--check

Envie os arquivos para o github

```
Git add .
Git commit -m “teste do black”
Git push
```

No github navegue até sua Branch automação e crie uma PR. Observe o que ocorre.

Volte ao seu terminal e execute o black: black .

envie os arquivos para o github

```
Git add .
Git commit -m “teste do black”
Git push
```

Observe que agora é possivel fazer o merge.

Navegue no log do actions e veja o que ele fez.

Em seu terminal, retorne para a Branch master.

Execute o comando git pull

Agora o seu git local está sincronizado com o git remoto.

Observe o log