

Boas-vindas

Esteja confortável, pegue uma água
e se acomode em um local tranquilo
que já começamos.

Como você **chega?**

1



2



3



Esta aula será

- **gravada**

Resumo

da aula anterior

✓ Programação Funcional X Programação Orientada a Objetos

✓ Classe:

✓ Objeto

✓ Importação de bibliotecas

```
class Cachorro:
    def __init__(self, nome, idade, raca):
        self.nome = nome
        self.idade = idade
        self.raca = raca

    def latir(self):
        print("Au!")
```

```
scooby = Cachorro("Scooby-Doo", 6, "Dogue Alemão")
```

```
import requests
import pandas as pd
from numpy import mean
```

Perguntas?

Aula 06. PYTHON

GIT e Controle de Versão

Objetivos da aula

- O que é Controle de Versão?;
- Qual a diferença de Git e GitHub;
- Comandos principais do Git;
- Deploy;
- Criando um repositório.

Controle de Versão

Controle de Versão

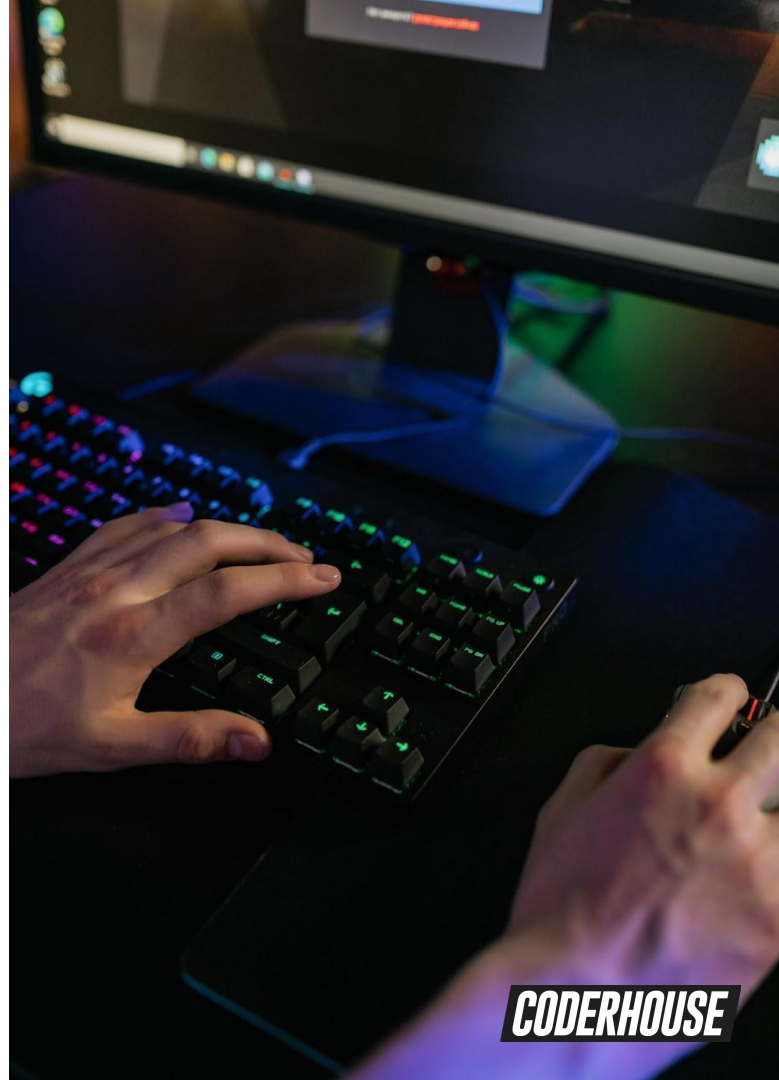
Ou **Version control (VC)** é o processo de gerenciar e controlar as mudanças no código fonte de um software ao longo do tempo. É um sistema que permite aos desenvolvedores monitorar e registrar as alterações realizadas no código-fonte, e ter a capacidade de voltar a versões anteriores se necessário.

Quem nunca fez isso!!!

-  codigo FINAL .py
-  codigo Final Final.py
-  codigo Final.py
-  codigo v2 - Copia.py
-  codigo v2.py
-  codigo v3.2.py
-  codigo v3.py
-  codigo.py

Vantagens do uso de VC

- ✓ Facilidade de uso
- ✓ Velocidade
- ✓ Colaboração
- ✓ Histórico de alterações
- ✓ Ramificação
- ✓ Suporte para grandes projetos
- ✓ Comunidade ativa
- ✓ Integração



CODERHOUSE

Controle de Versão

1. **Facilidade de uso:** relativamente fácil de aprender e usar, principalmente para desenvolvedores que já estão familiarizados com outros sistemas de controle de versão.
2. **Velocidade:** os sistemas normalmente são muito rápido e eficiente no gerenciamento de arquivos, mesmo em projetos grandes com muitos arquivos e histórico de alterações.

Controle de Versão

3. **Colaboração:** permite que várias pessoas trabalhem no mesmo código simultaneamente, com a possibilidade de mesclar as mudanças de diferentes desenvolvedores em um código final funcionando.
4. **Histórico de alterações:** mantém um histórico completo de todas as mudanças feitas em um arquivo ao longo do tempo, permitindo que os desenvolvedores voltem a versões anteriores ou comparem diferentes versões de um arquivo.

Controle de Versão

5. **Ramificação:** permite que os desenvolvedores criem ramificações ou branches para experimentar novas funcionalidades ou correções de bugs sem afetar o código principal, e depois mesclar as mudanças apenas quando estiverem prontos.
6. **Suporte para grandes projetos:** É capaz de lidar com projetos grandes com muitos arquivos e muitos desenvolvedores trabalhando simultaneamente.

Controle de Versão

7. **Comunidade ativa:** Possui uma comunidade ativa de desenvolvedores que trabalham constantemente para melhorar o sistema e fornecer suporte aos usuários.
8. **Integração:** Pode ser facilmente integrado a outras ferramentas de desenvolvimento, como sistemas de automação de construção e deploy, facilitando o desenvolvimento de software e implantações.

GIT x Github

O que é GIT?

Com o **Git**, os desenvolvedores podem criar um repositório ou diretório para armazenar todos os arquivos de um projeto, rastrear as mudanças feitas nesses arquivos ao longo do tempo, criar ramificações ou branches para experimentar novas funcionalidades sem afetar o código principal, mesclar as mudanças feitas por diferentes desenvolvedores em um código final funcionando, e muito mais.

É como uma máquina do tempo para o seu código-fonte. Ele permite que você salve várias versões do seu código em diferentes momentos, para que você possa voltar no tempo e ver o que foi alterado e quando.



História do GIT?

Como muitas coisas na vida, o Git começou com um pouco de destruição criativa e uma ardente controvérsia.

Em 2005, a relação entre a comunidade que desenvolveu o núcleo do Linux e a empresa que desenvolveu BitKeeper (ferramenta até então utilizada) quebrou em pedaços, e a ferramenta passou a ser paga. Isto alertou a comunidade que desenvolvia o Linux (e especialmente Linux Torvalds, o criador do Linux) a desenvolver a sua própria ferramenta baseada em lições aprendidas ao usar o BitKeeper.



CODERHOUSE

História do GIT?

Algumas metas do novo sistema era os seguintes:

- ✓ Velocidade
- ✓ Projeto simples
- ✓ Forte suporte para desenvolvimento não-linear (milhares de ramos paralelos)
- ✓ Completamente distribuído
- ✓ Capaz de lidar com projetos grandes como o núcleo o Linux com eficiência (velocidade e tamanho dos dados)



GIT e GitHub são a mesma coisa?

Não! 🙅

O que é GitHub?

GitHub é um serviço baseado em nuvem que hospeda um sistema de controle de versão do tipo GIT.

Assim como o **Google Fotos** é um “Drive” específico para fotos, possuindo visualizações, edições, álbuns e gerenciamento de fotos.

O GitHub é como um “Drive” específico para arquivos GIT, possuindo funcionalidades específicas para arquivos GIT.



Outras plataformas

Existem várias outras plataformas baseadas em nuvem semelhantes ao Github. Alguns exemplos incluem:

- ✓ GitLab
- ✓ Bitbucket
- ✓ SourceForge



GitLab



Bitbucket



SOURCEFORGE

CODERHOUSE

Instalação

Primeiro precisamos instalar o **GIT**, seguindo as instruções do mesmo:

Começando – Instalando o Git

O que usar?

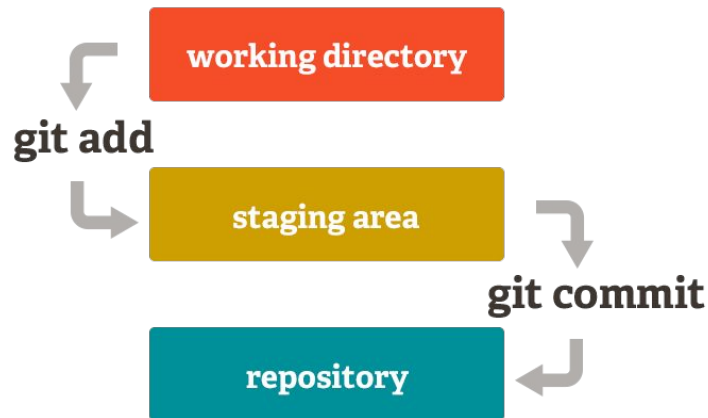
Utilizaremos o **Git Bash** nos exemplos da aula 😊



Comandos principais

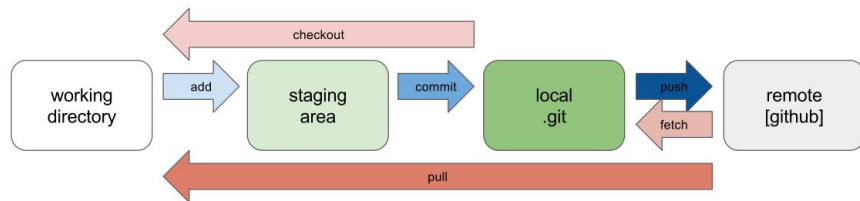
Comandos principais

- ✓ **git init:** Cria um novo repositório Git local.
- ✓ **git add:** Adiciona um arquivo ao índice (staging area).
- ✓ **git commit:** Grava as alterações no repositório.



Comandos principais

- ✓ **git status:** Exibe o status do repositório, incluindo arquivos não rastreados e alterações não confirmadas.
- ✓ **git pull:** Obtém e incorpora mudanças do repositório remoto no repositório local.
- ✓ **git push:** Envia as alterações feitas no repositório local para o repositório remoto.



Comandos principais

Decore!! Estes são o comandos que você vai mais utilizar:

```
git add .  
git commit -m "adicionado exercícius 3"  
git push origin main
```



IN CASE OF FIRE

 > GIT COMMIT

 > GIT PUSH

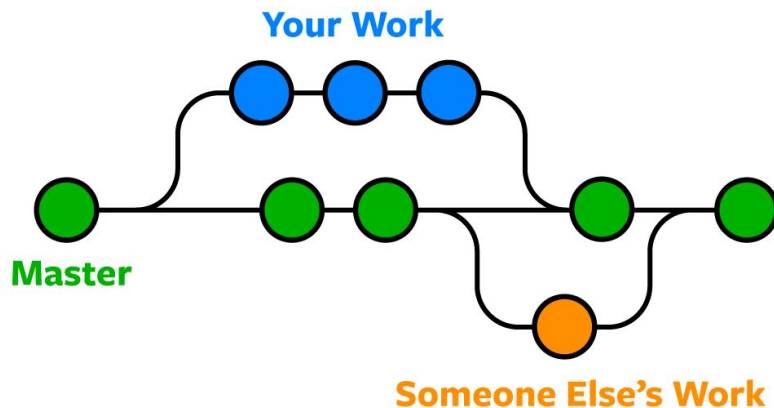
 > LEAVE BUILDING



Comandos de Branch

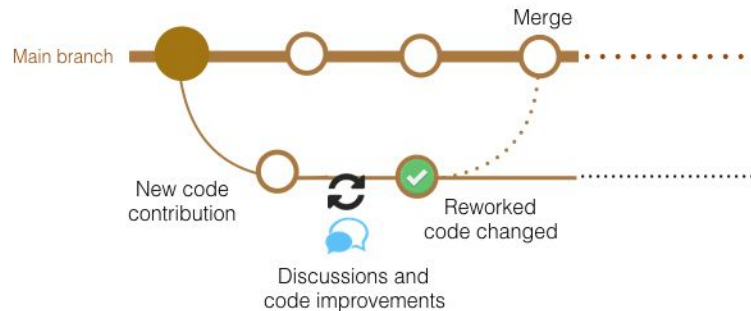
- ✓ **git branch:** Lista as branches do repositório e permite criar novas branches.
- ✓ **git checkout:** Permite alternar entre branches e também mover arquivos para um determinado commit.
- ✓ **git merge:** Une as mudanças de uma branch em outra.

Branches são ramificações do código-fonte que permitem trabalhar em diferentes versões do seu projeto ao mesmo tempo, sem afetar o código principal.



Pull Request

Um **pull request** é uma solicitação para que um ramo de código em um repositório Git seja mesclado com outro ramo, geralmente o ramo principal ou uma ramificação mais estável.



Simplified Pull Request process

Outros comandos

- ✓ **git clone:** Cria uma cópia de um repositório Git existente.
- ✓ **git log:** Exibe o histórico de commits do repositório.

```
$ TZ=PTST8PDT git log-compact --decorate --graph -n 17 v2.6.1
```

```
== 2015-09-28 ==  
* 22f698cb 19:19 jch (tag: v2.6.1) Git 2.6.1  
* 3adc4ec7 19:16 jch Sync with v2.5.4  
\   
* 24358560 15:34 jch (tag: v2.5.4) Git 2.5.4  
* 11a458be 15:33 jch Sync with 2.4.10  
  
* a2558fb8 15:30 jch (tag: v2.4.10) Git 2.4.10  
* 6343e2f6 15:28 jch Sync with 2.3.10  
  
* 18b58f70 15:26 jch (tag: v2.3.10, maint-2.3) Git 2.3.10  
* 92cdfd21 14:59 jch Merge branch 'jk/xdiff-memory-limits' into maint-2.3  
\   
* 83c4d380 14:58 jk merge-file: enforce MAX_XDIFF_SIZE on incoming files  
* dcd1742e 14:57 jk xdiff: reject files larger than ~1GB  
* 3efb9880 14:57 jk react to errors in xdi_diff  
* f2df3104 14:46 jch Merge branch 'jk/transfer-limit-redirection' into maint-2.3  
  
== 2015-09-25 ==  
* b2581164 15:32 bb http: limit redirection depth  
* f4113cac 15:30 bb http: limit redirection to protocol-whitelist  
* 5088d3b3 15:28 jk transport: refactor protocol whitelist code  
  
== 2015-09-28 ==  
* df37727a 14:33 jch Merge branch 'jk/transfer-limit-protocol' into maint-2.3  
  
== 2015-09-23 ==  
* 33cfccbb 11:35 jk submodule: allow only certain protocols for submodule fetches
```

exemple de git log

Resumos dos comandos

git init: Cria um novo repositório Git local.

git add: Adiciona um arquivo ao índice (staging area).

git commit: Grava as alterações no repositório.

git clone: Cria uma cópia de um repositório Git existente.

git pull: Obtém e incorpora mudanças do repositório remoto no repositório local.

git push: Envia as alterações feitas no repositório local para o repositório remoto.

git status: Exibe o status do repositório, incluindo arquivos não rastreados e alterações não confirmadas.

git branch: Lista as branches do repositório e permite criar novas branches.

Resumos dos comandos

git checkout: Permite alternar entre branches e também mover arquivos para um determinado commit.

git merge: Une as mudanças de uma branch em outra.

git log: Exibe o histórico de commits do repositório.

Cheat sheet dos comandos: [GitHub Education](#)



Break

5 minutos e voltamos!





Break

10 minutos e voltamos!

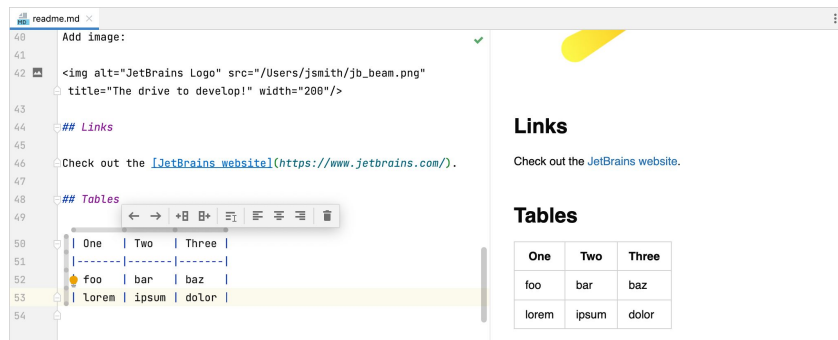


Arquivos Auxiliares

README.md

Geralmente é incluído na raiz de um repositório do Github, fornece informações sobre o projeto para os usuários e desenvolvedores que visitam o repositório. O README.md é escrito em Markdown, que permite a formatação de texto e a inclusão de imagens e links.

O README.md é frequentemente a primeira coisa que os usuários e desenvolvedores verão quando visitarem o repositório do Github, por isso é importante que seja bem escrito e informativo.



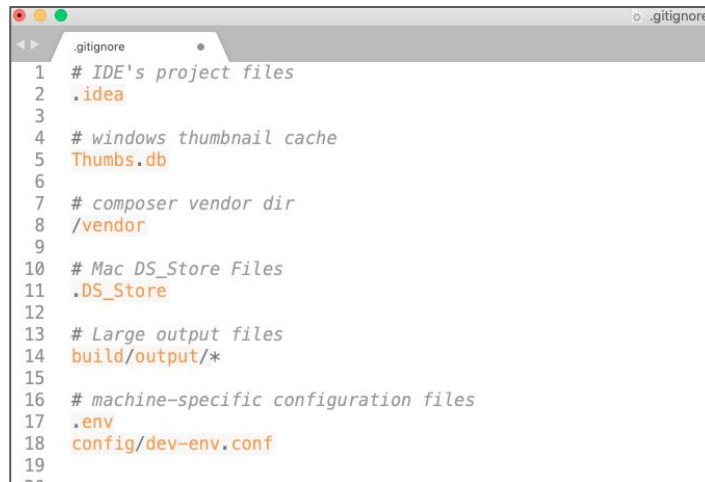
.gitignore

É usado para especificar quais arquivos e pastas devem ser ignorados pelo Git durante o processo de controle de versão. Isso é útil quando há arquivos gerados automaticamente ou dados sensíveis que não devem ser compartilhados no repositório.

O arquivo .gitignore é colocado na raiz do projeto e permite especificar padrões de nome de arquivo ou pasta para serem ignorados.

Exemplo de gitignore para projetos em Python:

[Python.gitignore](#)

A screenshot of a code editor window showing a .gitignore file. The file contains several lines of text, each preceded by a line number from 1 to 19. The text specifies patterns for files and directories to be ignored by Git. The patterns include IDE project files, Windows thumbnail cache, composer vendor directory, Mac DS_Store files, large output files, and machine-specific configuration files. The patterns are: # IDE's project files, .idea, # windows thumbnail cache, Thumbs.db, # composer vendor dir, /vendor, # Mac DS_Store Files, .DS_Store, # Large output files, build/output/*, # machine-specific configuration files, .env, and config/dev-env.conf.

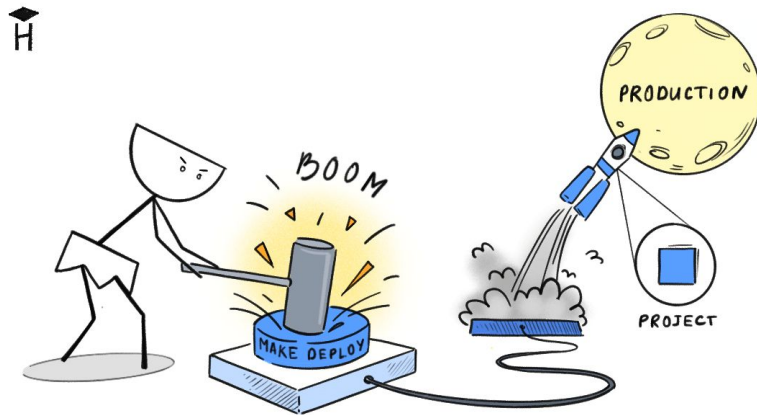
```
1 # IDE's project files
2 .idea
3
4 # windows thumbnail cache
5 Thumbs.db
6
7 # composer vendor dir
8 /vendor
9
10 # Mac DS_Store Files
11 .DS_Store
12
13 # Large output files
14 build/output/*
15
16 # machine-specific configuration files
17 .env
18 config/dev-env.conf
19
```

Deploy

O que é deploy

O verbo deploy, em inglês, quer dizer implantar.

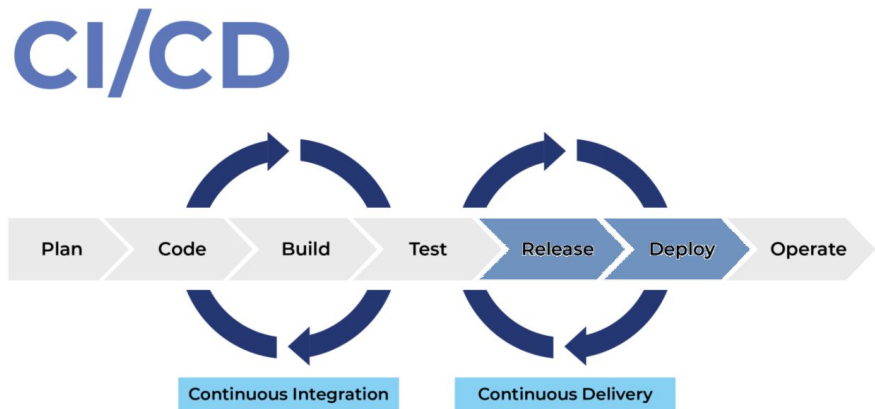
Deploy é o processo de implantar um software ou aplicativo em um ambiente de produção, tornando-o disponível para uso pelos usuários finais.



O que é deploy

O deploy pode ser realizado manualmente ou automaticamente por meio de ferramentas de integração.

Durante o processo de deploy, o código-fonte é compilado, testado e implantado em um ambiente de produção, onde os usuários finais podem acessá-lo e usá-lo.



Deploy e GIT

Git ou Github são ferramentas que podem ser usadas em conjunto com outras ferramentas de integração contínua e implantação contínua (CI/CD) para automatizar o processo de deploy.

O Github também fornece recursos para integração com várias ferramentas de Deploy, como mencionado anteriormente, permitindo que os desenvolvedores configurem pipelines de implantação automatizados para seus projetos.



Criando um repositório no GitHub

Duração: 10 minutos



ATIVIDADE EM SALA

Criando um repositório no GitHub

Vamos seguir o passo a passo descrito em:

<https://docs.github.com/pt/get-started/quickstart/create-a-repo>

Perguntas?

Como foi a aula?

1

Que bom

O que foi super legal na aula e podemos sempre trazer para as próximas?

2

Que pena

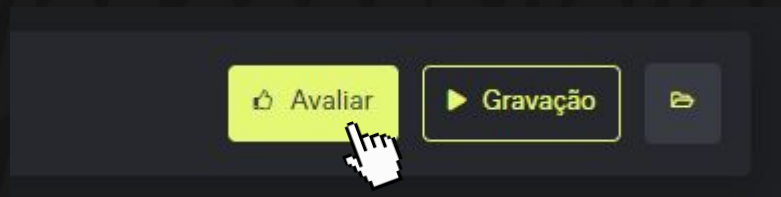
O que você acha que não funcionou bem e precisamos melhorar?

3

Que tal

Qual sugestão deveríamos tentar em próximas aulas?

O que você achou da aula?



Seu feedback vale pontos para o Top 10!! 🕶️



Deixe sua opinião!

1. Acesse a plataforma
2. Vá na aula do dia
3. Clique em **Avaliar**



Primeiro commit

DESAFIO COMPLEMENTAR



DESAFIO COMPLEMENTAR

Primeiro commit

Descrição

- ✓ Criar um repositório no github com os exercícios desenvolvidos até agora.

Aspectos a incluir

- ✓ Crie uma conta no github (se já não possui);
- ✓ Crie um repositório;
- ✓ Crie uma pasta na sua máquina e vincule ao repositório;
- ✓ Crie um commit inicial e de um Push;

Formato

- ✓ A entrega é o link do repositório
- ✓ Crie um commit com todos os exercícios e de um Push;
- ✓ Adicione no readme o nome do curso e o seu nome;
- ✓ Adicione os tutores e o professor como colaboradores do repositório;

Resumo

da aula de hoje

- ✓ Controle de Versão?
- ✓ Git e GitHub
- ✓ Deploy
- ✓ Comandos Git (add, commit, push, ...)
- ✓ README.md e .gitignore
- ✓ Repositório

Serviços de talento 🚀

Potencialize sua jornada! Na Coderhouse você pode adquirir **serviços de desenvolvimento profissional** com foco em **currículo, LinkedIn e entrevistas!**

No seu perfil, acesse a barra lateral e clique em ★ **Serviços**. Agora é só escolher o serviço que deseja adquirir.



Ainda quer saber mais?
Recomendamos o
seguinte material



MATERIAL AMPLIADO

Recursos multimídia

Git

- ✓ [Git cheat sheet](#) | **GitHub**
- ✓ [Criando um repositório](#) | **GitHub**
- ✓ [Python.gitignore](#) | **GitHub**



**Obrigado por estudar
conosco!**