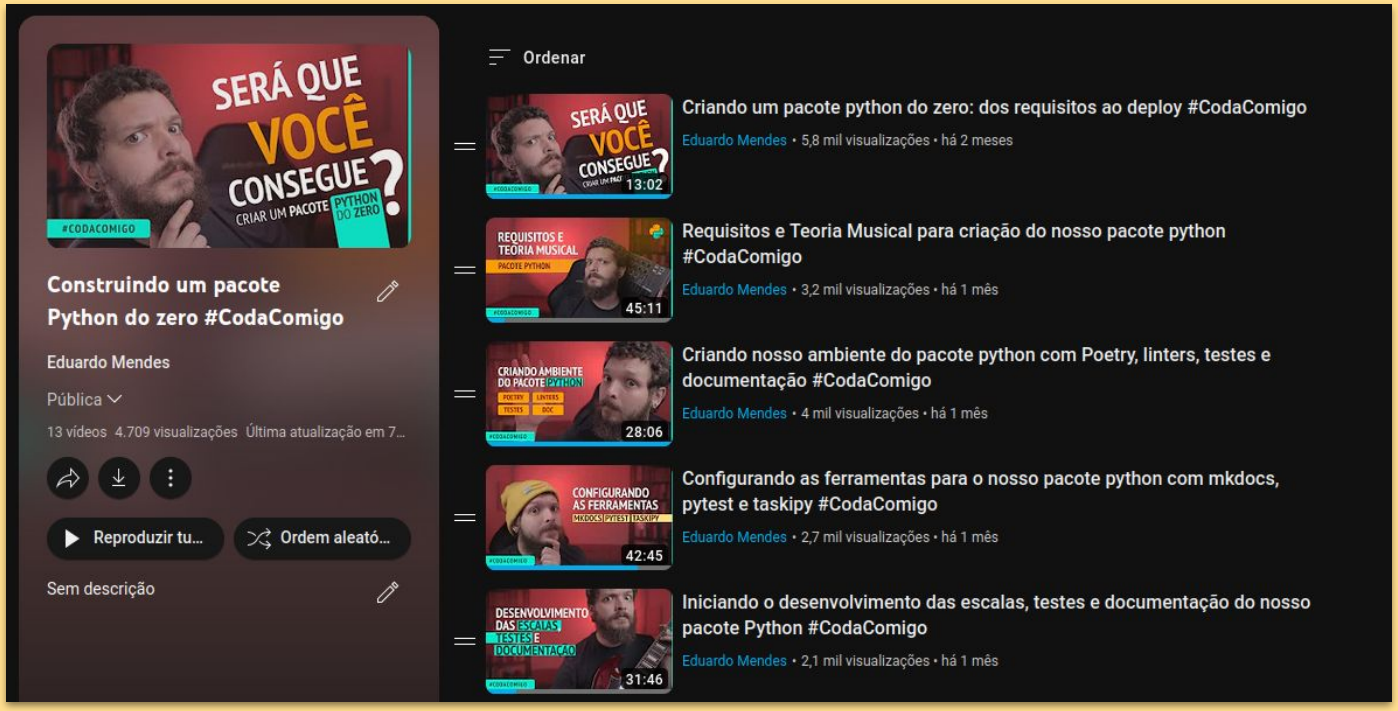
Aplicações de linha de comando

Live de Python #233

Gostaria que nos atentássemos hoje mais a conceitos do que implementações.

Nota 1



Um conteúdo totalmente prático relacionado a Typer foi elaborado no canal recentemente!

Nota 2 - Link na descrição!

Roteiro

1. Aplicações de linha de comando

O que são?

2. Um pouco sobre os padrões

Dando nome as coisas

3. Conhecendo o Typer

E os conceitos de CLI

4. Criando uma aplicação

Se der tempo!

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Ademar Peixoto, Adilson Herculano, Adriano Ferraz, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alynne Ferreira, Alysson Oliveira, Ana Carneiro, Andre Azevedo, Andre Mesquita, Aquiles Coutinho, Arnaldo Turque, Aslay Clevisson, Aurelio Costa, Bernardo At, Bruno Almeida, Bruno Barcellos, Bruno Barros, Bruno Batista, Bruno Freitas, Bruno Lopes, Bruno Ramos, Caio Nascimento, Christiano Morais, Damianth, Daniel Freitas, Daniel Wojcickoski, Danilo Boas, Danilo Segura, Danilo Silva, David Couto, David Kwast, Davi Goivinho, Delton Porfiro, Denis Bernardo, Dgeison Peixoto, Diego Farias, Diego Guimarães, Dilenon Delfino, Diogo Paschoal, Edgar, Edinilson Bonato, Eduardo Tolmasquim, Elias Silva, Emerson Rafael, Érico Andrei, Everton Silva, Fabiano, Fabiano Tomita, Fabio Barros, Fábio Barros, Fabio Castro, Fábio Thomaz, Felipe Rodrigues, Fernanda Prado, Fernando Celmer, Firehouse, Flávio Meira, Francisco Neto, Gabriel Barbosa, Gabriel Espindola, Gabriel Mizuno, Gabriel Moreira, Gabriel Nascimento, Gabriel Paiva, Gabriel Simonetto, Gabriel Souza, Geandreson Costa, Gilberto Abrao, Giovanna Teodoro, Giuliano Silva, Guilherme Felitti, Guilherme Gall, Guilherme Ostrock, Guilherme Piccioni, Guilherme Silva, Guionardo Furlan, Gustavo Pereira, Gustavo Suto, Harold Gautschi, Heitor Fernandes, Helvio Rezende, Hugo Cosme, Igor Riegel, Italo Silva, Janael Pinheiro, Jhonatan Martins, Joelson Sartori, Jônatas Silva, Jon Cardoso, Jorge Silva, Jose Alves, José Gomes, Joseíto Júnior, Jose Mazolini, Juan Felipe, Juan Gutierrez, Juliana Machado, Julio Franco, Júlio Gazeta, Julio Silva, Kaio Peixoto, Kaneson Alves, Leandro Miranda, Leandro Silva, Leo Ivan, Leonardo Mello, Leonardo Nazareth, Leon Solon, Luancomputacao Roger, Lucas Adorno, Lucas Carderelli, Lucas Mendes, Lucas Nascimento, Lucas Schneider, Lucas Simon, Lucas Valino, Luciano Filho, Luciano Ratamero, Luciano Silva, Luciano Teixeira, Luis Alves, Luiz Duarte, Luiz Lima, Luiz Paula, Luiz Perciliano, Mackilem Laan, Marcelo Campos, Marcio Moises, Marco Mello, Marcos Gomes, Maria Clara, Marina Passos, Mateus Lisboa, Matheus Silva, Matheus Vian, Mauricio Fagundes, Mauricio Nunes, Mírian Batista, Mlevi Lsantos, Murilo Viana, Natan Cervinski, Nathan Branco, Nicolas Teodosio, Osvaldo Neto, Otávio Carneiro, Patricia Minamizawa, Patrick Felipe, Paulo D., Paulo Tadei, Pedro Henrique, Pedro Pereira, Peterson Santos, Priscila Santos, Pydocs Pro, Pytonyc, Rafael Lino, Rafael Lopes, Rafael Romão, Raimundo Ramos, Ramayana Menezes, Regis Santos, Renato Oliveira, Rene Bastos, Ricardo Silva, Riverfount, Rjribeiro, Robson, Robson Maciel, Rodrigo Freire, Rodrigo Oliveira, Rodrigo Quiles, Rodrigo Ribeiro, Rodrigo Vaccari, Rodrigo Vieira, Rogério Lima, Rogério Nogueira, Ronaldo Silva, Ronaldo Silveira, Rui Jr, Samanta Cicilia, Thaynara Pinto, Thiago Araujo, Thiago Borges, Thiago Curvelo, Tiago Minuzzi, Tony Dias, Tyrone Damasceno, Valcilon Silva, Valdir Tegon, Vcwild, Vicente Marcal, Vinicius Stein, Vladimir Lemos, Walter Reis, Willian Lopes, Wilson Duarte, Wilson Neto, Xico Silvério, Yuri Fialho, Zeca Figueiredo

Obrigado você

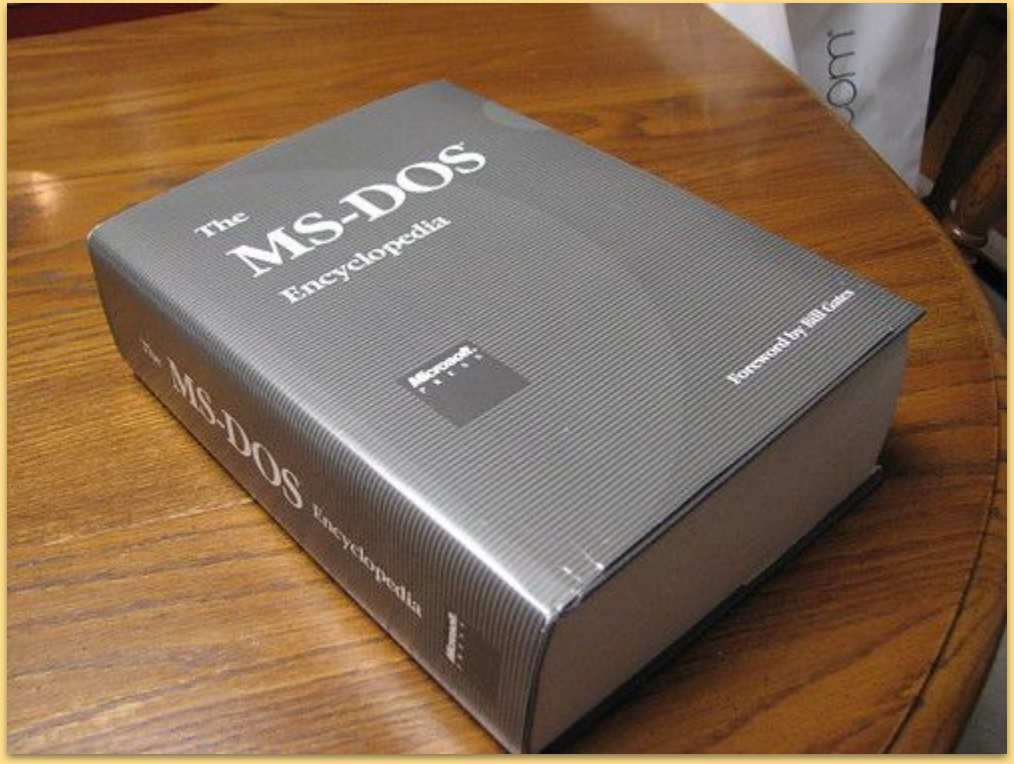
Apps CLI

O que são?

Um pouco de história, mas bem pouco

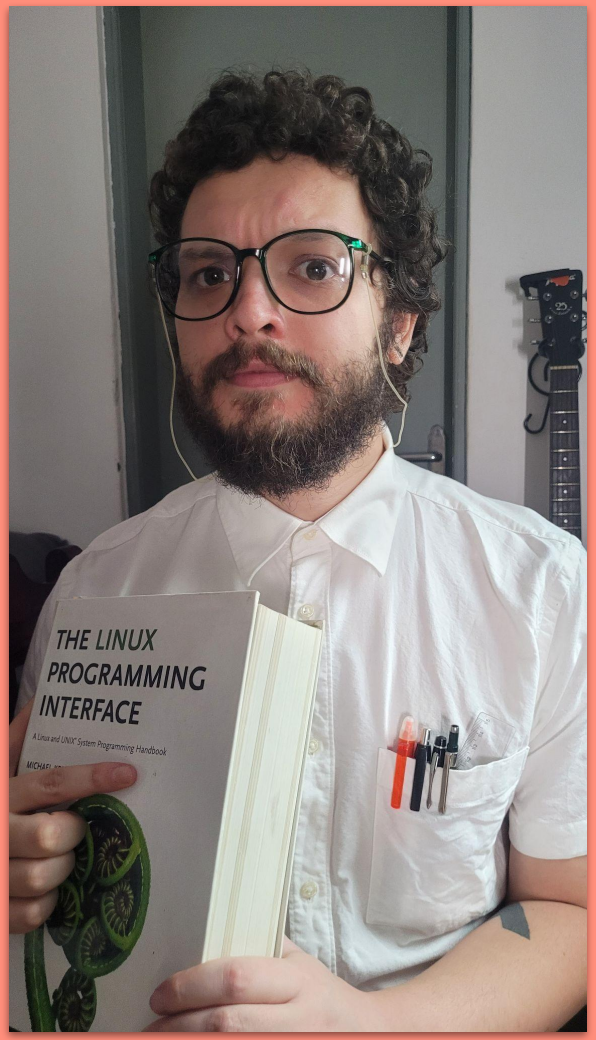
https://flashbak.com/paleotechnology-a-curious-glimpse-into-an-80s-computer-book-2714/

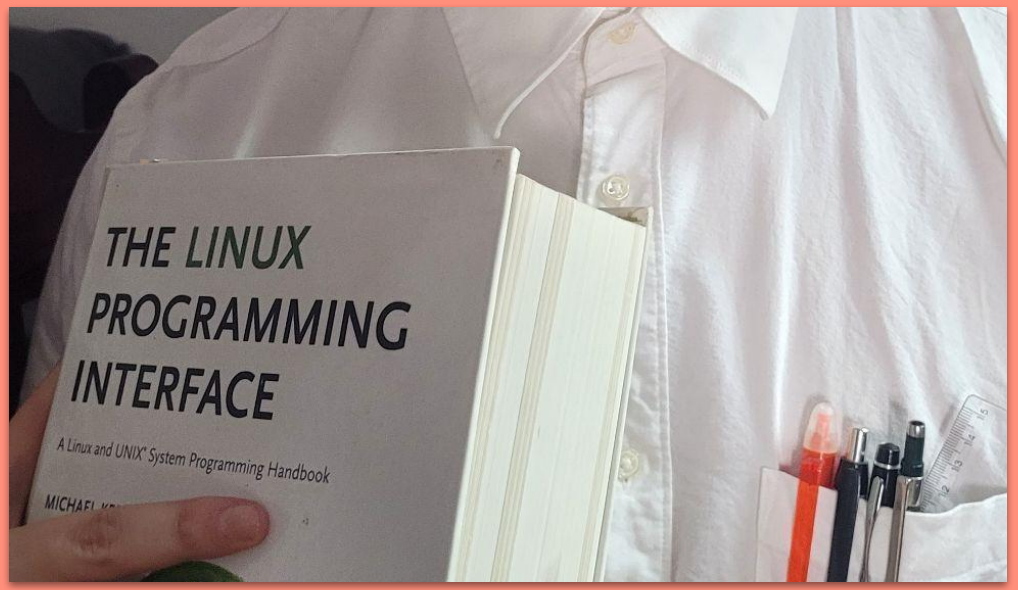
Por isso



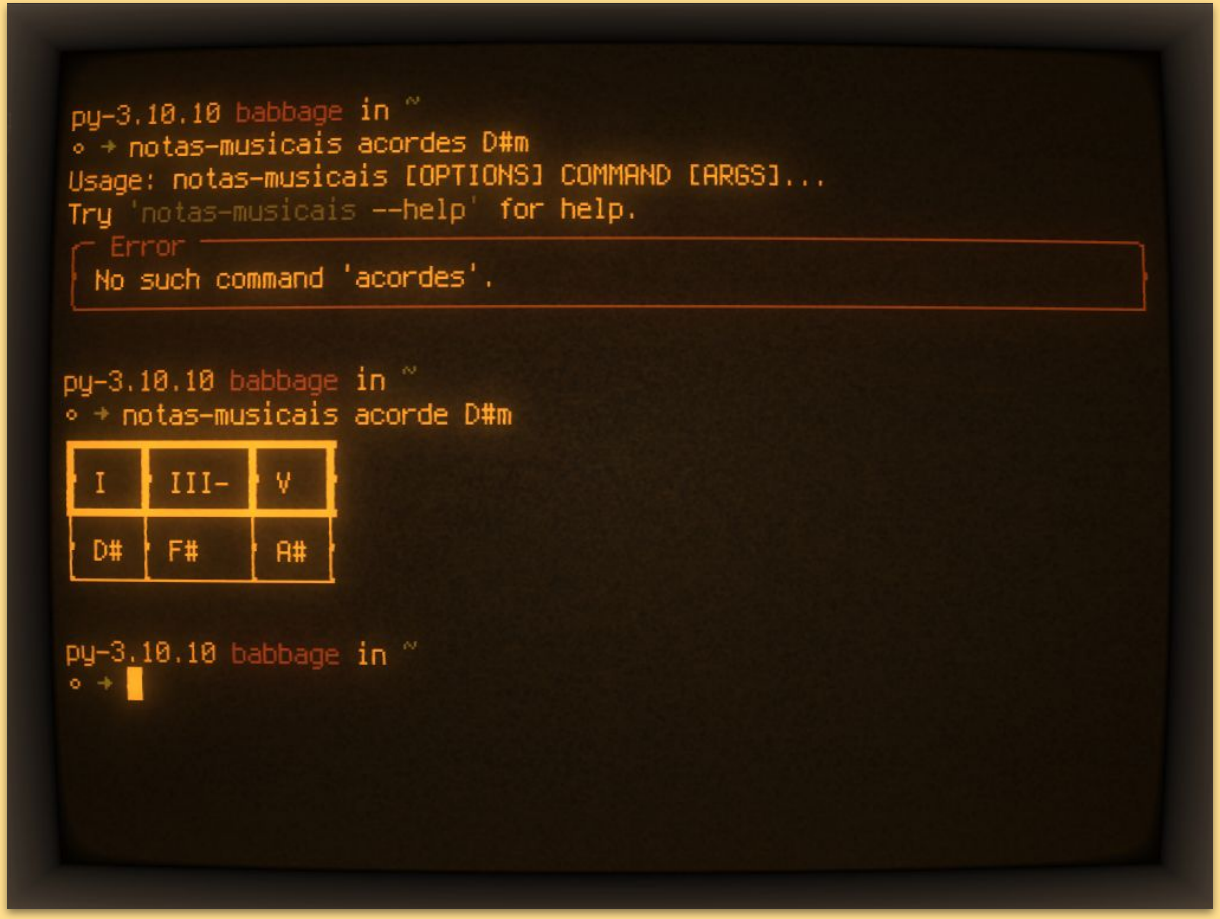
https://www.flickr.com/photos/markbirkle/2983251428

Ou isso


THE LINUX
PROGRAMMING
INTERFACE


THE LINUX
PROGRAMMING
INTERFACE
A Linux and UNIX® System Programming Handbook

Essa era a forma de interagir com o PC



https://github.com/Swordfish90/cool-retro-term

Exemplo de aplicações de terminal

- pip / poetry / hatch / pyenv / pipx / conda
- venv
- python
- git
- httpie
- ...

Uma grande conversa

Lidar com a linha de comando é conversar com o computador, diferente de uma interface gráfica que dispões todos os seus "atributos" de forma visual.

```
$ httpie
usage: httpie [-h] [--debug] [--traceback] [--version] {cli,plugins} ...
httpie: error: Please specify one of these: 'cli', 'plugins'

This command is only for managing HTTPie plugins.
To send a request, please use the http/https commands:

  $ http POST pie.dev/post hello=world

  $ https POST pie.dev/post hello=world
```

Lidar com o terminal, é como conversar com o shell

A segunda pergunta

Lidar com o terminal, é como conversar com o shell

```
$ http
usage:
    http [METHOD] URL [REQUEST_ITEM ...]

error:
    the following arguments are required: URL

for more information:
    run 'http --help' or visit https://httpie.io/docs/cli
```

Continuando a conversa

Se sabemos que o uso é `http [METHOD] URL [REQUEST_ITEM ...]`

```
$ http GET dunossauro.com
HTTP/1.1 301 Moved Permanently
Content-Length: 38
Content-Type: text/plain; charset=utf-8
Date: Sat, 29 Apr 2023 18:03:08 GMT
Location: https://dunossauro.com/
Server: Netlify
X-Nf-Request-Id: 01GZ72XNMG8MGMDM6SFK19WZWW

Redirecting to https://dunossauro.com/
```

Mais um pouco de papo

Como ainda não conseguimos chegar onde queríamos, poderíamos pedir ajuda!

```
$ http --help
```

Executemos juntos, como sei que quero que ele "siga" [follow] (é um termo padrão na web), também poderia perguntar sobre isso:

```
$ http --help | grep "follow"
      The "auto" style follows your terminal's ANSI color styles.
      The following are the default options:
      followed redirects (with --follow), the first unauthorized request when
      See the following page to find out your default HTTPIE_CONFIG_DIR:
  --follow, -F
      By default, requests have a limit of 30 redirects (works with --follow).
      3xx (Redirect) and --follow hasn't been set, then the exit status is 3.
      A string in the OpenSSL cipher list format. By default, the following
```

Obtendo o resultado esperado

Depois de um bom papo com a máquina, chegamos lá!

```
$ http GET dunossauro.com -F
HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 102441
Cache-Control: public, max-age=0, must-revalidate
Content-Encoding: gzip
Content-Length: 1360
Content-Type: text/html; charset=UTF-8
Date: Fri, 28 Apr 2023 13:43:24 GMT
Etag: "d50b6978f7fa8f3f9df5f6e164a89aec-ssl-df"
Server: Netlify
Strict-Transport-Security: max-age=31536000
Vary: Accept-Encoding
X-Nf-Request-Id: 01GZ73BJXE1XBMN7SJWRP25BPN

<html>
  <head>
    <meta charset="utf-8" />
    <title>Dunossauro</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.3.0/css/all.min.css">
    <link rel="stylesheet" href="https://unpkg.com/terminal.css@0.7.2/dist/terminal.min.css" />
```

Por que isso é legal?

- "Composível"
  - É possível colocar aplicações diferentes para trabalhar juntas
  - `httpie | grep`
  - `cat arquivo | less`
- Atômico
  - Indivisível
  - Caso uma coisa falhe, tudo falha
  - Resposta para um único input
- Indepotente
  - Sempre de obtêm a mesma resposta
  - Pode ser executado mais de uma vez

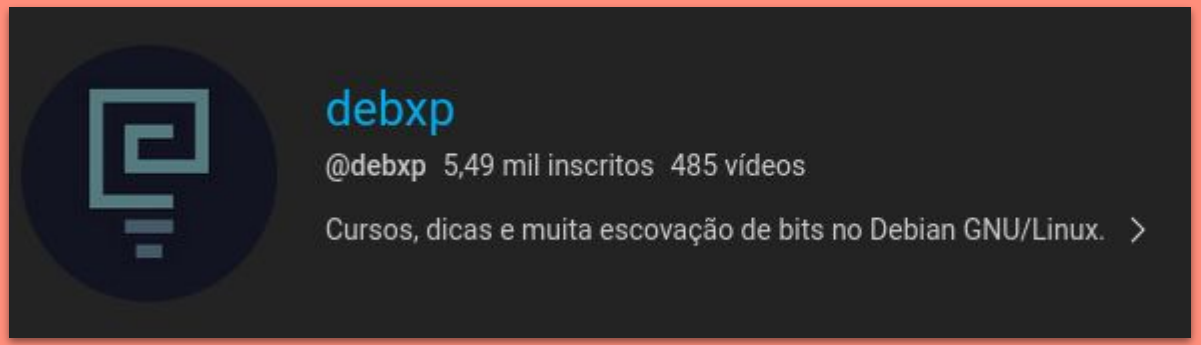
Claro!

Se quiserem, podemos fazer uma live somente sobre conceitos sobre aplicações de linha de comando, sem nos preocuparmos com implementação de código.

Me empolguei?

Para dominar o shell

Recomendo conhecer o canal do Blau Araujo!



https://www.youtube.com/@debxp
blauaraujo.com

Dando nome as coisas

Padrões

12 Factor CLI Apps

CLIs are a fantastic way to build products. Unlike web applications, they take a small fraction of the time to build and are much more powerful. With the web, you can do whatever the developer programmed. With CLIs, you can easily mash-up multiple tools together yourself to perform advanced tasks. They require more technical expertise to use, but still work well for admin tasks, power user tasks

Jeff Dickey

1.5K Followers

New Blog: jdxcode.com

Follow

https://medium.com/@jdxcode/12-factor-cli-apps-dd3c227a0e46

Command Line Interface Guidelines

https://clig.dev/

Vocabulário de nicho

- Comando (geralmente o nome da aplicação)
  - git
  - poetry
  - pip
- Subcomando
  - git commit
  - poetry add
  - pip install
- Argumentos
  - git switch main
  - pip install typer
  - poetry remove pandas

Vocabulário de nicho

- Flag / sinalizador
  - pip install `--upgrade` pip
  - git push `-f`
  - ls `-lhF`
- Redirecionamento (pipes)
  - cat <arquivo> `|` grep teste
  - notas-musicais escala `>` arquivo.txt
- Códigos de saída
  - 0: Quando deu certo
  - 1: Quando deu errado
  - 2: Problema de permissão
  - ...

Vocabulário de nicho

- Condutores
  - stdin: Entrada padrão
  - stdout: Saída padrão
  - stderr: Saída de erro

Algumas opções que """todos""" os CLI têm

- Flags de ajuda
  - git `-h`
  - git `–help`
  - git `–version`
- Manuais no terminal
  - `info` http
  - `man` http
- Obtendo mais informações
  - pipx upgrade-all `--verbose`
  - pytest `-vvv` [verbose x 4]

Typer

E os conceitos de CLI

Typer

- **Typer** é um projeto desenvolvido pelo **Sebastián Ramírez**
- É baseado no **Click**, mantido pelo time **Pallets**
  - Lives de python #88 e #89
- Dezembro de 2019
- Versão atual: **0.8.0** (saiu hoje xD)
- Licença: **MIT**
- Usa como extensões:
  - **Rich**: Para prints incríveis (live de python #224)
  - **Shellingham**: Detecta em qual shell está sendo executado

pip install typer[all]

Instalação

Um olá mundo! [exemplo_01.py]

```python
from typer import run

def olar(nome: str):
    print(f'Olá {nome}')

run(olar)
```

Um olá mundo! [exemplo_01.py]

```python
from typer import run

def olar(nome: str):
    print(f'Olá {nome}')

run(olar)
```

No terminal

```
$ python exemplo_01.py
Usage: exemplo_01.py [OPTIONS] NOME
Try 'exemplo_01.py --help' for help.
╭─ Error ──────────────────────────────────────────────────────────────────────╮
│ Missing argument 'NOME'.                                                     │
╰──────────────────────────────────────────────────────────────────────────────╯
```

O início do papo

```
$ python exemplo_01.py
Usage: exemplo_01.py [OPTIONS] NOME
Try 'exemplo_01.py --help' for help.
╭─ Error ──────────────────────────────────────────────────────────────────────╮
│ Missing argument 'NOME'.                                                     │
╰──────────────────────────────────────────────────────────────────────────────╯
```

0 –help

```
$ python exemplo_01.py --help

 Usage: exemplo_01.py [OPTIONS] NOME

╭─ Arguments ──────────────────────────────────────────────────╮
│ *    nome       TEXT  [default: None] [required]            │
╰──────────────────────────────────────────────────────────────╯
╭─ Options ────────────────────────────────────────────────────╮
│ --help          Show this message and exit.                  │
╰──────────────────────────────────────────────────────────────╯
```

12 fatores para CLI

1. **Um bom help é essencial**
2. Prefira flags a argumentos
3. Em qual versão estou?
4. Tenha condutores em mente (pipes)
5. Lide com as coisas dando errado
6. Seja chique
7. Avise, se puder
8. **Use tabelas**
9. Seja rápido
10. Encoraje contribuições
11. Seja claro sobre subcomandos
12. Siga o XDG

12 fatores para CLI

1. **Um bom help é essencial**
2. Prefira flags a argumentos
3. Em qual versão estou?
4. Tenha condutores em mente (pipes)
    a. Expor formatores diferentes (--plain, –json, etc..) [clig]
5. Lide com as coisas dando errado
    a. Erros para humanos (clig)
6. Seja chique
7. Avise, se puder
8. **Use tabelas (continuação do 6)**
9. Seja rápido
10. Encoraje contribuições (continuação do 1)
11. Seja claro sobre subcomandos (continuação do 1)
12. Siga o XDG

Argumentos de funções são argumentos do CLI

O typer organiza os argumentos posicionais como argumentos do CLI e os nomeados como flags opcionais:

```python
from typer import run

def cli(nome: str, email: str, senha: str = None):
    print('Cadastro efetuado com sucesso!')
    ...

run(cli)
```

0 – help

```
$ python exemplo_02.py --help

  Usage: exemplo_02.py [OPTIONS] NOME EMAIL

┌─ Arguments ──────────────────────────────────────────────────────────┐
│ *    nome        TEXT  [default: None] [required]                    │
│ *    email       TEXT  [default: None] [required]                    │
└──────────────────────────────────────────────────────────────────────┘
┌─ Options ────────────────────────────────────────────────────────────┐
│ --senha          TEXT  [default: None]                               │
│ --help                 Show this message and exit.                   │
└──────────────────────────────────────────────────────────────────────┘
```

Valores de flags precisam ser explícitos

```
$python exemplo_02.py a b c
Usage: exemplo_02.py [OPTIONS] NOME EMAIL
Try 'exemplo_02.py --help' for help.
╭─ Error ──────────────────────────────────────────────────────────────╮
│ Got unexpected extra argument (c)                                    │
╰──────────────────────────────────────────────────────────────────────╯
```

O uso das flags

```
$ python exemplo_02.py a b --senha c
Cadastro efetuado com sucesso!
```

Argumentos opcionais e obrigatórios

O Typer conta com um parâmetro um tipo chamado `Argument`.
Arguments oferecem algunas opções sintáticas interessantes:

- Argument(...): Requerido
- Argument('valor'): Valor default
- Argument(uma_func): Algo que gera o argumento padrão
- Argument(..., help='Mensagem de ajuda')
- Argument(..., metavar='customização de exibição')
- Argument(..., envvar='VARIAVEL_DE_AMBIENTE')

Exemplos de uso [exemplo_03.py]

```python
from typer import Argument, run

def cli(
    nome: str = Argument(..., help='Seu primeiro nome'),
    email: str = Argument('teste@email.com', metavar='✉️email✉️'),
    senha: str = Argument(
        '123',
        help='Sua senha, ela pode ser uma variável de ambiente',
        envvar='SENHA',
    ),
):
    print(f'{nome=}, {email=}, {senha=}')


run(cli)
```

Nosso help

```
$ python exemplo_03.py --help

 Usage: exemplo_03.py [OPTIONS] NOME 📧email📧 [SENHA]

┌─ Arguments ──────────────────────────────────────────────────────────────────┐
│ *    nome        TEXT        Seu primeiro nome [default: None] [required]    │
│      email       📧email📧   [default: teste@email.com]                      │
│      senha       [SENHA]     Sua senha, ela pode ser uma variável de ambiente│
│                              [env var: SENHA]                                │
│                              [default: 123]                                  │
└──────────────────────────────────────────────────────────────────────────────┘
┌─ Options ────────────────────────────────────────────────────────────────────┐
│ --help           Show this message and exit.                                 │
└──────────────────────────────────────────────────────────────────────────────┘
```

O uso de flags

Para o uso de flags o Typer nos fornece o objeto Option. Option fornece várias formas de uso bastante interessantes:

- Option(False, help='Uma flag qualquer')
- Option(...): Flag requerida
- Option(..., prompt=True, configrmation_pront=True, hide_input=True)
- Option(False, '--version', '-v')
- Option(..., callback=uma_funcao_para_processar)
- Option(..., callback=uma_funcao_para_processar, is_eager=True):

  Executa antes da função de fato

Um exemplo de uso [exemplo_04.py]

```python
from typer import Argument, Exit, Option, run

def version(value: bool):
    if value:
        print('0.0.1a0')
        raise Exit(code=0)

def cli(
    nome: str = Argument(..., help='Seu primeiro nome'),
    email: str = Argument('teste@email.com', metavar='📧email📧'),
    senha: str = Option(
        ...,
        prompt=True,
        confirmation_prompt=True,
        hide_input=True,
        help='Senha será cobrada no prompt',
    ),
    version: bool = Option(
        False, '--version', '-v', callback=version, is_eager=True
    ),
):
    print(f'{nome=}, {email=}, {senha=}')

run(cli)
```

12 fatores para CLI

1. **Um bom help é essencial**
2. Prefira flags a argumentos
3. **Em qual versão estou?**
4. Tenha condutores em mente (pipes)
5. Lide com as coisas dando errado
6. **Seja chique**
7. Avise, se puder
8. **Use tabelas**
9. Seja rápido
10. Encoraje contribuições
11. Seja claro sobre subcomandos
12. Siga o XDG

Subcomandos

Para que o Typer trabalhe com subcomandos tempos que criar um aplicativo:

```python
from typer import Typer

app = Typer()

@app.command()
def comando_a():
    ...

@app.command()
def comando_b():
    ...

app()
```

Um exemplo

```
$ python exemplo_05.py --help

 Usage: exemplo_05.py [OPTIONS] COMMAND [ARGS]...

╭─ Options ────────────────────────────────────────────────────────────────────╮
│ --install-completion          Install completion for the current shell.      │
│ --show-completion             Show completion for the current shell, to copy it or │
│                               customize the installation.                    │
│ --help                        Show this message and exit.                    │
╰──────────────────────────────────────────────────────────────────────────────╯

╭─ Commands ───────────────────────────────────────────────────────────────────╮
│ comando-a                                                                    │
│ comando-b                                                                    │
╰──────────────────────────────────────────────────────────────────────────────╯
```

Aqui começa o meu caso com o typer

O fator 11 é "**Seja claro sobre subcomandos**" e o typer exige um help para mostrar esses comandos.

- Quando app é criado, você perde o comando inicial
- É necessário um callback, se quisermos a flag –version
- O App suprime as mensagens base
- É necessário importar o contexto do Click

[exemplo_05.py]

```python
from typer import Context, Exit, Option, Typer

app = Typer()


def version(arg):
    if arg:
        print('0.0.1a0')
        raise Exit(code=0)


@app.callback(invoke_without_command=True)
def typer_callback(
    ctx: Context,
    version: bool = Option(
        False, '--version', '-v', is_eager=True, is_flag=True, callback=version
    )
):
    if ctx.invoked_subcommand:
        return
    print('Use um dos comandos! `comando_a` ou `comando_b`')

app()
```

23 linhas de boilerplaite

12 fatores para CLI

1. **Um bom help é essencial**
2. Prefira flags a argumentos
3. **Em qual versão estou?**
4. Tenha condutores em mente (pipes)
5. Lide com as coisas dando errado
6. **Seja chique**
7. Avise, se puder
8. **Use tabelas**
9. Seja rápido
10. Encoraje contribuições
11. **Seja claro sobre subcomandos**
12. Siga o XDG

Testes [exemplo_06.py]

Uma das coisas mais legais do Typer é o seu cliente de testes. Para que ele funcione, você precisa de um app.

```python
from typer import Typer
from typer.testing import CliRunner

app = Typer()

test_runner = CliRunner()

@app.command()
def version():
    print('0.0.1a0')

def test():
    result = test_runner.invoke(app)
    assert result.stdout == 'batatinha', result.stdout

test()
```

Disponibilizarei o conteúdo em "projeto.py" no repositório!

Caso não tenha dado pra codar o projetinho

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Referências

- CLIG – CLI Guidelines: clig.dev
- 12 Factor CLI-apps: medium.com/@jdxcode/12-factor-cli-apps-dd3c227a0e46
- GNU Standards para CLI: gnu.org/prep/standards/html_node/Command_002dLine-Interfaces.html
- Documentação do typer: typer.tiangolo.com