

# Documento de especificação funcional Tevec



# Sumário

Introdução.....	1
Dados pessoais.....	1
Detalhes.....	1
Objetivo.....	1
Funcionalidades.....	1
Dicionário.....	1
Estrutura.....	1
1.    Classes:.....	1
1.1. Variaveis globais:.....	1
1.2. Métodos:.....	1
Restrições.....	2
Utilização.....	2

# Introdução

Este documento tem como objetivo detalhar as regras de negocio, requisitos, detalhes e estrutura de código e utilização do programa **Instalador**.

O **Instalador** é utilizado para clonar um projeto alocado no github e instalar o mesmo dentro do ambiente python. Onde este pode ser chamado como um modulo em qualquer programa python.

A tarefa de trazer o projeto para dentro do ambiente python é muito interessante para por exemplo: realizar testes facilmente fora do ambiente de desenvolvimento.

É possível tanto instalar quanto desinstalar um projeto do github no ambiente python, apenas setando o nome do projeto alocado no github. O programa também suporta instalar a versão (tag) de um release específico de um projeto.

## Glossário dos termos

1. Parâmetro: Valores inseridos no terminal (cmd no Windows) antes de rodar o programa. Ex: "python **Instalador.py** -h", neste caso "-h" é um parâmetro.
2. Usuário: Utilizador do programa (**Instalador.py**).
3. Ambiente python, ambiente: Ambiente gerenciado pelo conda para desenvolvimento utilizando python.

## Regras de negócio

### Detalhes

**Título:** Implementação do instalador

**Nome do elaborador:** Junior

**Data de criação da especificação:** 30/08/2016

**Nome do revisor:** Wanderson

**Data de revisão da especificação:** 31/08/2016

**Nome do(s) Projeto(s) envolvido(s):** moura, wickbold, ampm, gerdal, danone, crm.

**Branch(s) utilizada(s):** [MRA\_20160802\_01], [WICK\_20160802\_01], [AMPM\_20160802\_01], [GRD\_20160802\_01], [DAN\_20160802\_01], [CRM\_20160802\_01]

**Prazo:** 26/08/2016

**Dificuldade:** ## (Pontuação no Sprint Planning (SCRUM))

**Prioridade:**

[ ] - Alta

☒ - Média

☐ - Baixa

**Complexidade:**

☐ - Alta

☒ - Média

☐ - Baixa

**Camada:**

☐ - Front-end

☒ - Back-end

☐ - Serviços

☐ - Banco de Dados

☐ - Outro

**Motivo:**

☐ - Nova característica

☐ - Alterar característica

☐ - Remover característica

☐ - Novo projeto

☒ - Novo Componente

☐ - Outro

## Objetivo

Disponibilizar um projeto alocado no github dentro do ambiente python (podendo setar a versão do release) ou remover o mesmo do ambiente.

## Solicitação

Tendo em vista a quantidade de problemas gerados relacionados com versionamento em testes em produção e QAS surgiu a necessidade de equalizar as versões para testes, dê certa forma a geração de um ambiente para teste em produção e QAS é um tanto quanto burocrática, pois há a necessidade de clonar os projetos manualmente do github e configurar o mesmo (onde se dedica um tempo a mais), ou seja o trabalho para testar é manual e sua configuração também.

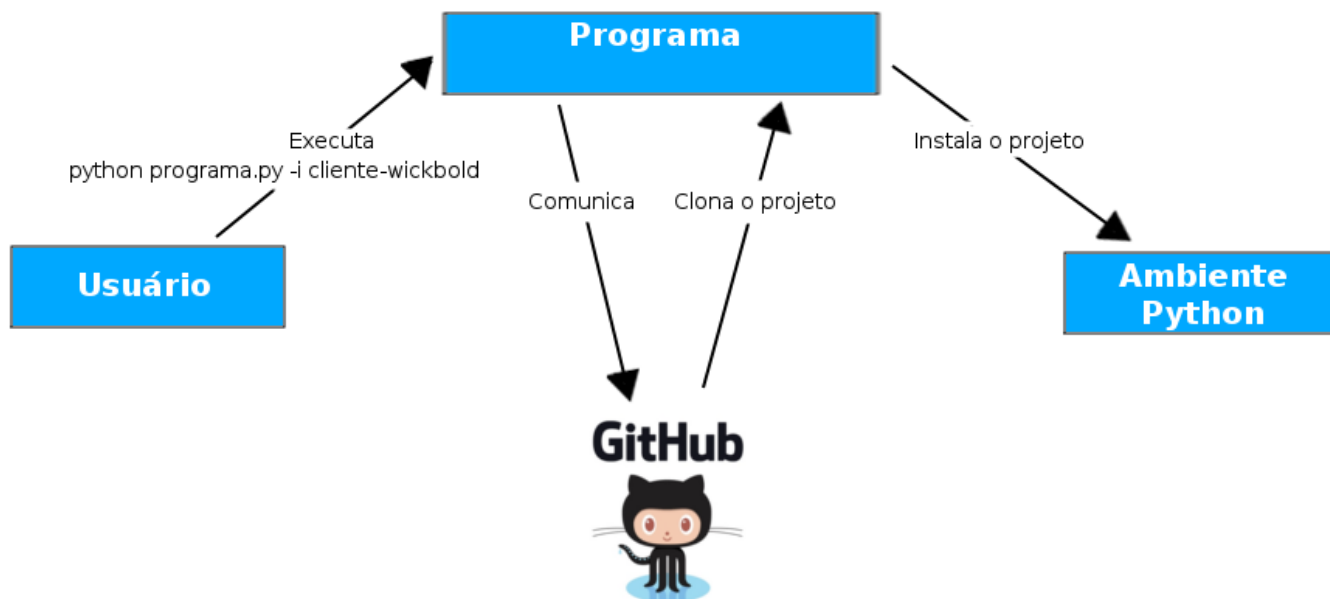
Então surgiu a ideia de criar um automatizador, para clonar os projetos do github e instalar o mesmo dentro do ambiente python.

Este automatizador deve ser uma aplicação console em python, onde é passado parâmetros para o programa e este por sua vez tomar decisões em cima dos parâmetros passados, por exemplo: "python programa.py -i nome\_do\_projeto" para instalar o projeto que está alocado no github no ambiente.

O programa também deverá ter a funcionalidade de instalar determinada versão do projeto também alocada no github, pois pode haver a necessidade de instalar uma versão anterior do projeto.

Em poucas palavras este automatizador deve ser uma facilitador para realização de testes em produção e QAS.

## Mapa conceitual



## Cenário atual

Hoje a pessoa que vai realizar testes em produção ou QAS precisa clonar localmente o projeto que está alocado no github.

## Requisitos funcionais

ID	Requisito	Descrição	Caso de uso
1	Instalar projeto no ambiente python.	Instalar projeto alocado no github dentro do ambiente python setando o nome do mesmo.	Instalar projeto no ambiente python.

2	Desinstalar projeto do ambiente python.	Desinstalar projetos no ambiente python setando o nome do mesmo.	Desinstalar projeto do ambiente python.
3	Instalar determinada versão do projeto no ambiente python.	Instalar versão (tag) de um determinado projeto alocado no github dentro do ambiente python setando o nome e a versão (tag) do mesmo.	Instalar determinada versão do projeto no ambiente python.
4	Listar ambientes.	Listar todos os ambiente python locais com um simples comando.	Lembrar/descobrir os ambientes python locais disponiveis.
5	Exibir ajuda.	Exibir um texto de ajuda ao usuário quando o mesmo solicitar através de um comando.	Dúvidas por parte do usuário.

## Requisitos não funcionais

ID	Requisito	Descrição
1	Validações.	Validar se o ambiente python está ativo, validar a versão do python.
2		

## Usabilidade

O usuário deverá executar o programa no terminal (cmd no Windows) utilizando o python, ex: “python **Instalador.py**”. Mas o programa deverá aceitar parâmetros e tomar decisões apartir dos parâmetros inseridos. Ficaria algo como “python **Instalador.py** -h” para exibir uma ajuda ao usuario, ou “python **Instalador.py** -i cliente-wickbold” para instalar um projeto alocado no github dentro do ambiente python.

## Plano de implantação

Fase	Objetivo	Prioridade
Fase 1	Implementar <b>instalador</b> .	Média
Fase 2		

Prioridades: Alta, Média, Baixa.

# Biblioteca

- <http://stackoverflow.com/questions/19042389/conda-installing-upgrading-directly-from-github>

## Especificação técnica

### Regras

- Deverá ser passado no mínimo um e no máximo quatro parâmetros ao executar o programa.
- É necessário ter um ambiente python a parte.
- O ambiente python precisa necessariamente ser gerenciado pelo conda.
- O ambiente python gerenciado pelo conda deve estar ativo antes de instalar ou desinstalar um projeto utilizando o programa.

### Restrições

- Não há restrições.

### Funcionalidades

1. Instalação de projetos no ambiente python: O programa possui a feature de instalar um projeto que está alocado no servidor do github dentro do ambiente python, para isto basta apenas setar o nome do projeto no github.
2. Desinstalação de projetos do ambiente python: O programa possui a feature de desinstalar um projeto que foi instalado anteriormente dentro do ambiente python, para isto basta apenas setar o nome do projeto alocado no github.
3. Atualização do python: O programa também verifica se a versão do python é igual a versão do python utilizada pelo strike. Caso a versão seja diferente o programa tem a funcionalidade de atualizar a versão do python, deixando igual a versão utilizada pelo strike.
4. Validações: O programa possui também uma série de validações para verificar: Se o ambiente python está ativo (o ambiente python precisa estar ativo para a instalação do projeto dentro do mesmo), versão do python, quantidade de parâmetros inseridos na execução do programa.
5. Listar os ambientes python: Com apenas um simples comando o programa consegue listar todos os ambientes python locais. Esta feature é interessante para consultar/lembrar os ambientes python.
6. Instalar a versão de um release específico: O programa também pode instalar uma versão específica de um projeto alocado no github dado o nome da tag (versão) no github.

7. Ajuda: O programa possui a funcionalidade de exibir uma ajuda com um pequeno manual de utilização do programa.
8. (A fazer) Dependências: Antes de instalar um projeto no ambiente python o programa pergunta ao usuário se o mesmo deseja instalar todas as dependências do projeto, ou não. Isto é útil se o usuário possui um ambiente com as versões dos pacotes python intencionalmente diferentes do strike (ex: para realizar provas de conceito) e não deseja que as versões de seus pacotes sejam substituídas.

## Estrutura

Veja abaixo toda a estrutura de classes, metodos e variaveis do **instalador**.

### 1. Classes:

#### 1.1. Principal:

Classe reponsável pelo comportamento de todo o programa.

##### 1.1.1. Variaveis globais:

- 1.1.1.1. versao\_release: Guarda a versão do release setado pelo usuário como parâmetro.
- 1.1.1.2. versao\_python: Guarda a versão do python utilizada pelo usuário.
- 1.1.1.3. nome\_cliente: Guarda o nome do cliente setado pelo usuário como parâmetro.
- 1.1.1.4. lista\_mensagens: Uma lista de todas as mensagens utilizadas pelo sistema.
- 1.1.1.5. nome\_programa: Guarda o nome do programa, neste caso **Instalador**.
- 1.1.1.6. versao\_python\_strike: Guarda a versão do python utilizada pelo corporativo.

##### 1.1.2. Métodos:

###### 1.1.2.1. \_\_init\_\_():

Esse é um método construtor, nele é executada toda a lógica do programa.

O método não possui parâmetros.

<b>Detalhe</b>	Neste momento o metodo verifica se há algum parametro com o valor '--atualizar-python', caso haja é chamado o metodo 'atualizarVersaoPython()' para atualizar a versão do python.
<b>Código</b>	<pre># Atualizar versao do python for indice, valor_parametro in enumerate(lista_parametros):     if valor_parametro == '--atualizar-python':         concluido = Instalador.atualizarVersaoPython(self, lista_parametros[indice + 1])         exit()</pre>
<b>Detalhe</b>	Verifica se há algum parametro com o valor '-h', se houver o programa chama o método 'exibirAjuda()' que imprime um texto para ajuda de utilização do programa.
<b>Código</b>	<pre># Exibir ajuda for indice, valor_parametro in enumerate(lista_parametros):</pre>



	<pre> if valor_parametro == '-h':     print(Instalador.exibirAjuda(self))     exit() </pre>
<b>Detalhe</b>	Verifica se há algum parametro com o valor '-l', se houver o programa chama o método 'listarAmbientes()' que imprime uma lista de todos os ambientes python locais.
<b>Código</b>	<pre> # Exibir lista de ambientes for indice, valor_parametro in enumerate(lista_parametros):     if valor_parametro == '-l':         concluido = Instalador.listarAmbientes(self)         exit() </pre>
<b>Detalhe</b>	Verifica se há algum parametro com o valor '-v', se houver o programa atribui o próximo valor da lista de parametros para uma variavel global chamada 'versao_release'.
<b>Código</b>	<pre> # Setando a versão do release for indice, valor_parametro in enumerate(lista_parametros):     if valor_parametro == '-v':         # Verifica se o indice existe na lista         if (indice + 1) in lista_parametros:             self.versao_release = (lista_parametros[indice + 1]) </pre>
<b>Detalhe</b>	Verifica se há algum parametro com o valor '-i' ou 'instalar', se houver o programa verifica se existe um próximo valor na lista de parametros (este próximo valor deve ser o nome do projeto no github), caso haja, o próximo valor é atribuido a uma veriaavel chamada 'nome_cliente'. Em seguida é chamado o método 'instalarPacote()' que tem a função de instalar o projeto do github no ambiente python.
<b>Código</b>	<pre> # Instalando o projeto do cliente e suas dependencias no ambiente python for indice, valor_parametro in enumerate(lista_parametros):     if valor_parametro == '-i' or valor_parametro == 'instalar':         if len(lista_parametros) &gt; 2 and len(lista_parametros) &lt; 6:             self.nome_cliente = lista_parametros[indice + 1]              # Instalando o projeto cliente e suas dependencias             concluido = Instalador.instalarPacote(self)         else:             print(self.lista_mensagens[8] + self.lista_mensagens[9])             exit() </pre>
<b>Detalhe</b>	Verifica se há algum parametro com o valor '-r' ou 'remover', se houver o programa verifica se existe um próximo valor na lista de parametros (este próximo valor deve ser o nome do projeto no github), caso haja, o próximo valor é atribuido a uma veriaavel chamada 'nome_cliente'. Em seguida é chamado o método 'desinstalarPacote()' que tem a função de desinstalar o projeto do github do ambiente python.
<b>Código</b>	<pre> # Desinstalando o projeto do cliente e suas dependencias do ambiente python </pre>

```

for indice, valor_parametro in enumerate(lista_parametros):
    if valor_parametro == '-r' or valor_parametro == 'remover':
        if (indice + 1) in lista_parametros:
            self.nome_cliente = lista_parametros[indice + 1]

            # Desinstalando o projeto cliente e suas dependencias
            concluido = Instalador.desinstalarPacote(self)
        else:
            print(self.lista_mensagens[8] + self.lista_mensagens[9])
            exit()

```

#### 1.1.2.2. InstalarPacote():

Esse método tem como objetivo instalar um projeto do github no ambiente python dado o nome do projeto no github.

O método não possui parâmetros.

<b>Detalhe</b>	O programa faz uma pergunta ao usuário, perguntando se ele deseja realmente instalar o projeto, se o usuario digitar 'y' e der enter o programa entende que ele deseja instalar o projeto no ambiente python.
<b>Código</b>	<pre> if input(self.lista_mensagens[2]) == 'y': </pre>
<b>Detalhe</b>	Neste momento o programa pega o caminho do ambiente python atravez do atributo 'prefix' da biblioteca 'sys'. Após é concatenado com o caminho até a paste '/site-packages/' onde são instalador os pacotes no ambiente python.
<b>Código</b>	<pre> # Pegando o caminho do ambiente python print(self.lista_mensagens[5])  caminho_ambiente = str(sys.prefix) + '/lib/python' + self.versao_python + '/site-packages/' </pre>
<b>Detalhe</b>	<p>O programa verifica se a variavel global 'versao_release' está preenchida. Se sim ela é concatenada com o valor atribuido a variavel 'comando_pip', caso contrario ela não é concatenada.</p> <p>A variavel 'comando_pip' recebe o comando utilizado pelo 'pip (instalador de pacotes python)' para realizar a instalação do projeto no ambiente python.</p>
<b>Código</b>	<pre> # Gerando o comando para o 'pip install' print(self.lista_mensagens[11])  if self.versao_release == None:      comando_pip = 'pip install -e git+http://github.com/TEVEC/' + str(self.nome_cliente) + '#egg=' + str(self.nome_cliente) </pre>

	<pre> else:      comando_pip = 'pip install -e git+https://github.com/TEVEC/' + str(self.nome_cliente) + '@' + str(self.versao_release) + '#egg=' + str(self.nome_cliente) </pre>
<b>Detalhe</b>	O programa abre uma sessão no terminal (ou cmd, no caso do windows) e vai até o diretório armazenado na variável 'caminho_ambiente', após acessar o diretório executa o comando armazenado na variável 'comando_pip'. Se um dos comandos falharem é exibida uma mensagem de erro e o programa é encerrado.
<b>Código</b>	<pre> # Utilizando os comandos  if os.system('cd ' + caminho_ambiente + ' &amp;&amp; ' + comando_pip):      exit() </pre>

#### 1.1.2.3. desinstalarPacote():

Esse método tem como objetivo desinstalar projetos do ambiente python dado o nome do projeto no github.

O método não possui parâmetros.

<b>Detalhe</b>	O programa faz uma pergunta ao usuário, perguntando se ele deseja realmente instalar o projeto, se o usuário digitar 'y' e der enter o programa entende que ele deseja instalar o projeto no ambiente python.
<b>Código</b>	<pre> if input(self.lista_mensagens[3]) == 'y': </pre>
<b>Detalhe</b>	O programa pega o caminho do ambiente python através do atributo 'prefix' da biblioteca 'sys'. Após é concatenado com o caminho até a pasta '/site-packages/' onde são instalados os pacotes no ambiente python.
<b>Código</b>	<pre> # Pegando o caminho do ambiente python  print(self.lista_mensagens[5])  caminho_ambiente = str(sys.prefix) + '/lib/python' + self.versao_python + '/site-packages/' </pre>
<b>Detalhe</b>	Passando o caminho armazenado na variável 'caminho_ambiente' como parâmetro do método 'remove()' da biblioteca 'os', onde este realiza a exclusão do diretório do projeto. Desinstalando o projeto do ambiente python.
<b>Código</b>	<pre> # Pegando o caminho do ambiente python  print(self.lista_mensagens[12])  os.remove(caminho_ambiente + 'src/' + str(self.nome_programa)) </pre>

#### 1.1.2.4. listarAmbientes():

Esse método tem como objetivo listar todos os ambientes python existentes localmente.

O método não possui parâmetros.

<b>Detalhe</b>	O programa abre uma sessão do terminal (ou cmd, no caso do windows) e executa o comando 'conda env list', utilizado para listar os ambientes python.
<b>Código</b>	<pre># Listando ambientes python  if os.system('conda env list'):     exit()</pre>

#### 1.1.2.5. atualizarVersaoPython():

Esse método tem como objetivo atualizar o versão do python quando o usuário solicitar.

O método receberá como parâmetro:

1. versao: Utilizado para setar a versão desejada do python.

<b>Detalhe</b>	O programa abre uma sessão do terminal (ou cmd, no caso do windows) e executa o comando 'conda install python=' + str(versao)', onde é setado o parâmetro 'versao' para determinar a versão do python desejada. O python é instalado ou atualizado com este comando.
<b>Código</b>	<pre># Atualizando versão do python  if os.system('conda install python=' + str(versao)):     exit()</pre>

#### 1.1.2.6. validacoes():

Esse método tem como objetivo agrupar e executar validações para o programa.

O método receberá como parâmetro:

1. lista\_parametros: Lista de parâmetros recebidos no console.

<b>Detalhe</b>	Valida se a variavel 'versao_python' que contém a versão do python instalada é diferente da variavel 'versao_python_strike' que guarda a versão do python utilizada pelo strike. Se for diferente é exibida uma mensagem de erro e o programa é encerrado.
<b>Código</b>	<pre># Validando versão do python  if self.versao_python != self.versao_python_strike:     print(self.lista_mensagens[4])     exit()</pre>
<b>Detalhe</b>	Valida se o tamanho da lista 'lista_parametros' (que contém uma lista dos parâmetros setados no console) é menor do que 2. Se verdadeiro é exibida uma mensagem de erro e o programa é encerrado.

<b>Código</b>	<pre># Validando se falta argumentos  if len(lista_parametros) &lt; 2:      print(self.lista_mensagens[7] + self.lista_mensagens[9])      exit()</pre>
<b>Detalhe</b>	Valida se o tamanho da lista 'lista_parametros' (que contém uma lista dos parâmetros setados no console) é maior do que 5. Se verdadeiro é exibida uma mensagem de erro e o programa é encerrado.
<b>Código</b>	<pre># Validando se há excesso de argumentos  if len(lista_parametros) &gt; 5:      print(self.lista_mensagens[10] + self.lista_mensagens[9])      exit()</pre>

#### 1.1.2.7.    exibirAjuda():

Esse método tem como objetivo exibir uma ajuda ao usuário, quando o mesmo solicitar.

O método não possui parâmetros.

<b>Detalhe</b>	Retorna uma string contendo o texto de ajuda.
<b>Código</b>	return texto_ajuda

#### 1.1.2.8.    carregarListaMensagens():

Esse método tem como objetivo carregar uma lista com as mensagens utilizadas pelo programa.

O método não possui parâmetros.

<b>Detalhe</b>	Carrega uma lista com as mensagens utilizadas pelo programa.
<b>Código</b>	self.lista_mensagens = ['\nInstalação concluída.', ...

## Observações

O programa não agrupa/separa as dependências de cada projeto. Ou seja não será possível, caso o usuário desejar instalar dois projetos distintos com dependências diferentes, ex: “projeto-X” com dependências “A” e “B” e “projeto-Y” com dependências “A” e “C” no mesmo ambiente e criar um agrupamento/separação de dependências para cada

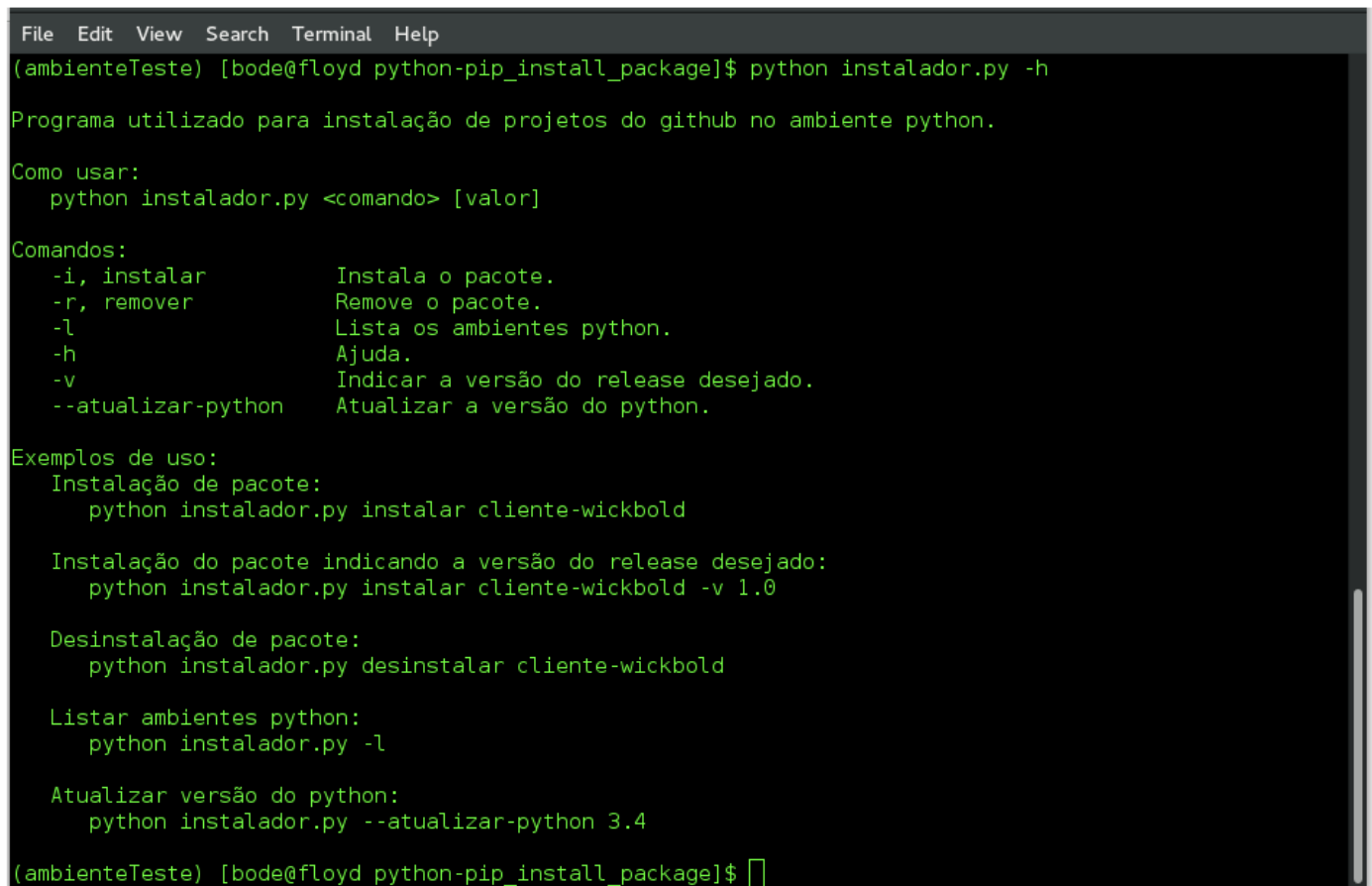
projeto, onde as dependências do “projeto-X” não interfiram nas dependências do “projeto-Y”. Isto não funciona pois o gerenciador de pacotes do ambiente python (conda) não aceita este tipo de funcionalidade, todas as dependências são instaladas em uma mesma instância. Uma solução para este caso seria criar ambientes diferentes para projetos distintos onde se deseja separar as dependências, e rodar o **Instalador.py** setando o projeto desejado em cada ambiente. Esta observação vale também para dependências com versões diferentes, é possível ter apenas uma versão de determinado pacote dentro do ambiente.

## Utilização

Deverá ter em máquina o programa **instalador.py** e rodar o mesmo com o comando 'python **instalador.py** <comando> [valor]'

No caso de instalação de projetos no ambiente. Como o repositório no github é privado, há a necessidade de autenticação antes de clonar um projeto do github para dentro do ambiente python. Neste caso a aplicação em tempo de execução solicitará o login e a senha utilizados pelo usuário no github para realizar a autenticação.

O programa possui um pequeno manual integrado. Para acessá-lo basta ter o programa em máquina e executar o seguinte comando no terminal (cmd no Windows): “python **instalador.py** -h”.



```
File Edit View Search Terminal Help
(ambienteTeste) [bode@floyd python-pip_install_package]$ python instalador.py -h

Programa utilizado para instalação de projetos do github no ambiente python.

Como usar:
  python instalador.py <comando> [valor]

Comandos:
  -i, instalar      Instala o pacote.
  -r, remover      Remove o pacote.
  -l               Lista os ambientes python.
  -h               Ajuda.
  -v               Indicar a versão do release desejado.
  --atualizar-python  Atualizar a versão do python.

Exemplos de uso:
  Instalação de pacote:
    python instalador.py instalar cliente-wickbold

  Instalação do pacote indicando a versão do release desejado:
    python instalador.py instalar cliente-wickbold -v 1.0

  Desinstalação de pacote:
    python instalador.py desinstalar cliente-wickbold

  Listar ambientes python:
    python instalador.py -l

  Atualizar versão do python:
    python instalador.py --atualizar-python 3.4

(ambienteTeste) [bode@floyd python-pip_install_package]$
```

Imagem 1 – Exemplo de utilização do programa.