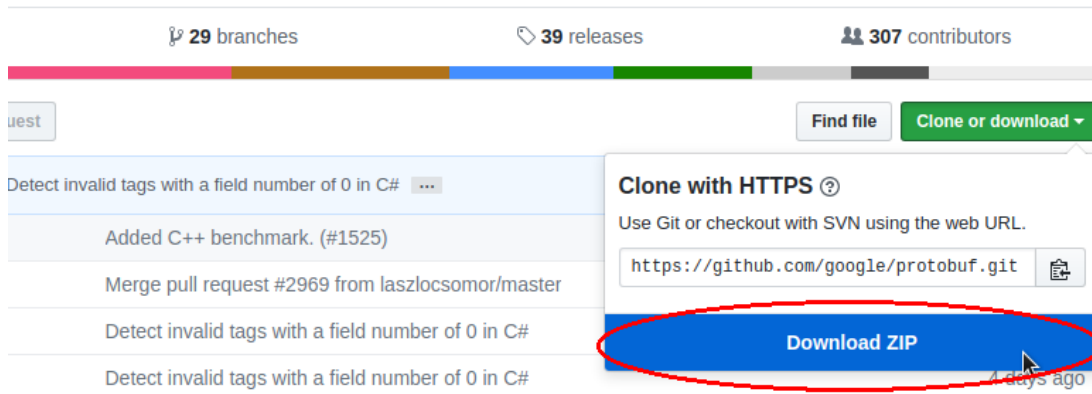


# PROTOCOL BUFFERS - TUTORIAL DE INSTALAÇÃO



1º: Fazer o download da pasta do projeto, presente no github, pelo seguinte link: <https://github.com/google/protobuf>



2º: Extrair a pasta do projeto.

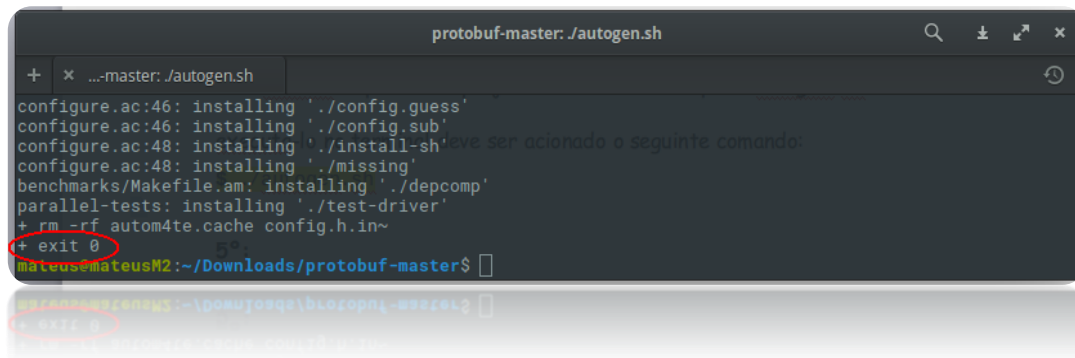
3º: Instalar alguns programas que serão necessários para a compilação do projeto Protocol Buffers. As instruções utilizadas agora estão presentes no arquivo **readme.md** na pasta src. No debian e derivados instale os programas com o seguinte comando:

```
$ sudo apt install autoconf automake libtool curl make g++ unzip
```

4º: Acesse a pasta do projeto no terminal e execute o arquivo **autogen.sh**. Para executá-lo no terminal deve ser acionado o seguinte comando:

```
$ ./autogen.sh
```

O comando deve retornar status 0, indicando sucesso na execução.



```
protobuf-master: ./autogen.sh
+ x ...-master: ./autogen.sh
configure.ac:46: installing './config.guess'
configure.ac:46: installing './config.sub'
configure.ac:48: installing './install-sh'
configure.ac:48: installing './missing'
benchmarks/Makefile.am: installing './depcomp'
parallel-tests: installing './test-driver'
+ rm -rf autom4te.cache config.h.in~
+ exit 0
MateusMateusM2:~/Downloads/protobuf-master$
```

5º: Para instalar o executor (*runtime*) e o compilador, devem ser executados os seguintes comandos:

```
$ ./configure
$ make
$ make check
$ sudo make install
$ sudo ldconfig # refresh shared library cache
```

A execução de alguns dos comandos acima pode ser um pouco demorada. Não encerre o processo, aguarde até que esteja terminado.

No fim da execução dos cinco passos acima, você terá os programas necessários para compilar arquivos **.proto**. Prosseguiremos o tutorial ensinando como configurar essa tecnologia para linguagem Java.

# PROTOCOL BUFFERS - INSTALAÇÃO PARA JAVA



**1º:** A configuração de Protocol Buffers para Java necessita do software Apache Maven. A instalação do maven pode ser realizada da seguinte forma:

**1.1:** Visite a página do maven em <https://maven.apache.org/download.cgi> e baixe a opção binária compactada em **tar.gz**.

**1.2:** Descompacte o arquivo para a pasta **/usr/local** conforme o seguinte comando:

```
$ sudo tar -zxf ./apache-maven-3.5.0-bin.tar.gz -C /usr/local/
```

O comando acima funciona com a versão 3.5.0 do maven. Caso você tenha baixado outra versão, adapte o comando a ela.

**1.3:** O próximo passo é criar um link simbólico no linux para que o maven seja visível para todos os usuários. O link pode ser criado com o seguinte comando:

```
$ sudo ln -s /usr/local/apache-maven-3.3.9/bin/mvn /usr/bin/mvn
```

**1.4:** Teste sua instalação do maven. Execute o comando:

```
$ mvn -v
```

Diogo D. Moreira em seu blog tem um tutorial completo sobre a instalação

do apache maven, presente em:

<http://diogodmoreira.com/blog/2015/08/25/maven-3-ubuntu-14-04.html>

**2º:** Considerando que a pasta do projeto Protocol Buffers está em sua pasta de downloads, prossiga para a subpasta específica da linguagem Java com o seguinte comando:

```
$ cd Downloads/protobuf-master/java
```

**3º:** Verifique se a instalação do compilador foi bem sucedida com o comando:

```
$ protoc --version
```

**4º:** Você deve executar alguns testes. A execução deles é realizada pelo maven, com o seguinte comando:

```
$ mvn test
```

Se algum dos testes falhar é um sinal que alguma das bibliotecas não está instalada corretamente em seu sistema. Você deve reiniciar a instalação caso tenha algum erro, ou prossiga por seu risco.

**5º:** Você deve instalar a biblioteca no seu repositório do maven. A instalação deve ser realizada da seguinte forma:

```
$ mvn install
```

**6º:** Se você não for utilizar o maven para gerenciar sua própria compilação, é possível criar um arquivo **.jar** para utilizar com o seguinte comando:

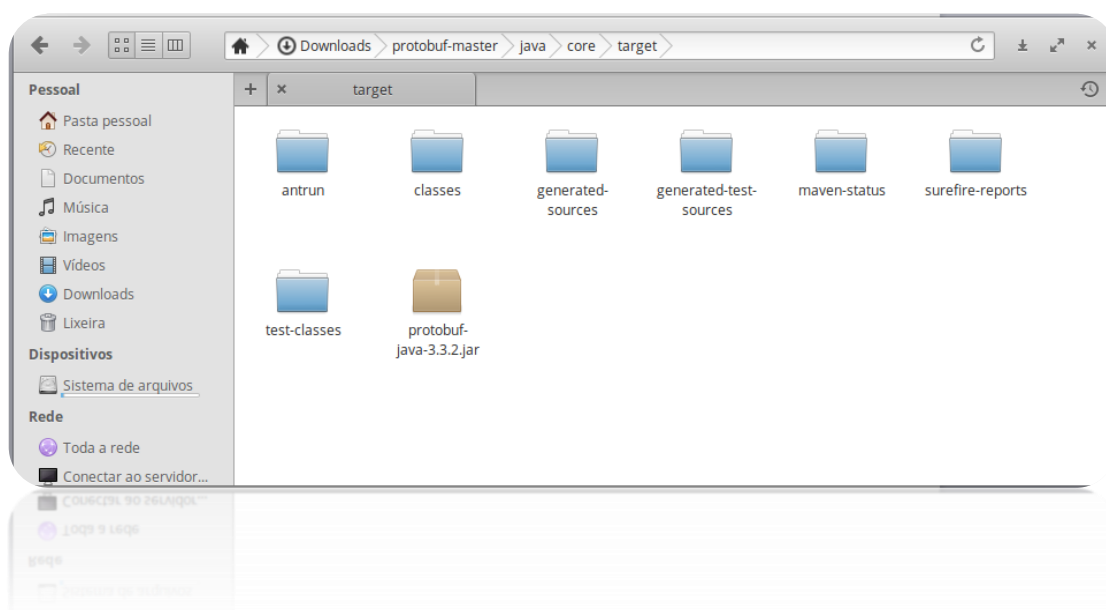
```
$ mvn package
```

O arquivo **.jar** gerado deve ser colocado no diretório alvo, ou seja, na pasta de seu projeto Java que irá utilizar Protocol Buffers.

Podem ser gerados três diferentes arquivos Java, sendo eles:

- **protobuf-java:** A biblioteca principal para Java do Protocol Buffers. A maioria dos usuários precisa apenas dessa biblioteca.
- **protobuf-lite:** A versão leve da biblioteca principal do Protocol Buffers. Essa é uma subversão da biblioteca citada acima, mais leve, geralmente utilizada para gerar códigos mais enxutos para dispositivos móveis.
- **protobuf-java-util:** Ele contém o suporte JSON, bem como utilitários para trabalhar com os tipos bem conhecidos do proto3.

Ao executar todos os passos acima, o arquivo **.jar** referente a protobuf-java foi gerado:



# gRPC - TUTORIAL DE INSTALAÇÃO



## Sobre o gRPC

gRPC é um sistema de código aberto de chamada de procedimento remoto (remote procedure call - RPC) inicialmente desenvolvido pela Google. Ele usa o protocolo HTTP/2 para o transporte, Protocol Buffers como linguagem de descrição da interface e fornece recursos como: autenticação, transmissão bidirecional, controle de fluxo e tempo limite. Ele gera conexões multi-plataforma para clientes e servidores de várias linguagens distintas.

## Instalação

Como foi descrito acima o gRPC fornece conexões de cliente e servidores em várias linguagens de programação. Para cada caso é necessário instalar dependências ou bibliotecas. O foco deste tutorial se concentra nas seguintes linguagens: Java, Python e C++. Para demais linguagens acesse <https://grpc.io/docs/>.

## Pré requisitos

Antes da instalação é necessário instalar algumas dependências, que no caso do Linux são: build-essential, autoconf e libtool.

Para instalar tais dependências use o comando:

```
$ sudo apt-get install build-essential autoconf libtool
```

Por padrão, gRPC usa o protocol buffers, é necessário do compilador protoc, na qual foi demonstrado a sua instalação acima, para gerar o servidor e o código do cliente.

- **Linguagem Python:** É necessários ter uma versão do pip (8 ou superior):

```
$ sudo python3 -m pip install --upgrade pip  
$ sudo python3 -m pip install grpcio  
$ sudo python3 -m pip install grpcio-tools
```

Para realizar a compilação dos arquivos de tipo ".proto" que contenham definições de serviços gRPC.

```
$python3 -m grpc_tools.protoc -I./ --python_out=./../python  
--grpc_python_out=./../python application.proto
```

- **Linguagem Java:** JDK (versão 7 ou superior):

Baixe os .JAR: [Clicando aqui](#) e os adicione em seu projeto. Note que junto aos arquivos de biblioteca que foram baixados existe um "protoc-gen-grpc-java.exe". Modifique-o para permitir execução. Ele será necessário no momento de gerar arquivos protoc com definições gRPC.

Para realizar a compilação dos arquivos de tipo ".proto" que contenham definições de serviços gRPC.

```
$ protoc --plugin=protoc-gen-grpc-java=protoc-gen-grpc-java.exe  
--grpc-java_out ./ --java_out ./ nomeArquivo.proto
```

- **Linguagem C++:**

```
$ git clone -b $(curl -L https://grpc.io/release)
https://github.com/grpc/grpc

$ cd grpc

$ git submodule update -init

$ make

$ [sudo] make install
```

### **gRPC depois da instalação:**

#### **Definindo o serviço**

O primeiro passo é definir o serviço gRPC, os pedidos e os tipos de resposta do método usando o protocol buffers.

Para definir um serviço, especifica-se um serviço nomeado em um arquivo .proto:

```
service SomaVet {
    ...
}
```

Em seguida, é necessário definir métodos rpc dentro do serviço, especificando seus tipos de solicitação e resposta. O gRPC permite definir quatro tipos de métodos de serviço:

- Um **RPC simples** onde o cliente envia uma solicitação ao servidor usando o stub e aguarda uma resposta de retorno, assim como uma chamada de função normal.

```
rpc GetVetpos(Pos) returns (Num) {}
```



- Um **RPC de transmissão do lado do servidor** onde o cliente envia uma solicitação ao servidor e recebe um fluxo para ler uma seqüência de mensagens de volta. O cliente lê do fluxo retornado até que não haja mais mensagens. Para isso especifica-se um método de transmissão do lado do servidor colocando a palavra-chave *stream* antes do tipo da resposta.

```
rpc RandoValores(Qtd) returns (stream Vet) {}
```

- Um **RPC de transmissão do lado do cliente** onde o cliente escreve uma seqüência de mensagens e as envia para o servidor, novamente usando um fluxo fornecido. Uma vez que o cliente terminou de escrever as mensagens, espera que o servidor as leia e devolva sua resposta. Para isso especifica-se um método de transmissão do lado do servidor colocando a palavra-chave *stream* antes do tipo da solicitação.

```
rpc SomaVet(stream Vet) returns (Num) {}
```

- Um **RPC de transmissão bidirecional** onde ambos os lados enviam uma seqüência de mensagens usando um fluxo de leitura e gravação. Os dois fluxos operam de forma independente, para que os clientes e servidores possam ler e escrever em qualquer ordem que desejam: por exemplo, o servidor pode aguardar para receber todas as mensagens do cliente antes de escrever suas respostas, ou pode ler uma mensagem e, em seguida, escrever uma mensagem, ou alguma outra combinação de leituras e gravações. A ordem das mensagens em cada

fluxo é preservada. Você especifica esse tipo de método colocando a palavra-chave *stream* antes da solicitação e da resposta.

```
rpc DobraElemVet(stream Vet) returns (stream Vet) {}
```

### **Comandos para compilar serviços com o gRPC:**

Gerando arquivos de comunicação. Rode os comandos para gerar as mensagens e as interfaces gRPC

#### **Python:**

```
$ python3 -m grpc_tools.protoc -I./  
--python_out=. --grpc_python_out=. application.proto
```

#### **Java:**

```
$ protoc  
--plugin=protoc-gen-grpc-java=protoc-gen-grpc-java.exe  
--grpc-java_out . --java_out . application.proto
```

Precisa da biblioteca `protoc-gen-grpc-java.exe` (baixe aqui a versão compatível:

<https://repo1.maven.org/maven2/io/grpc/protoc-gen-grpc-java/1.0.1/>)

#### **C++:**

```
$ protoc -I --grpc_out=. --plugin=protoc-gen-grpc='which  
grpc_cpp_plugin' Aplocacao.proto
```