10

Validando parâmetros vazios

Transcrição

Conseguimos montar um script capaz de filtrar o parâmetro passado pelo usuário, desta forma:

```
#!/bin/bash

cd ~/apache-log

if [ $1 == "GET" ]
then
    cat apache.log | grep GET
elif [ $1 == "POS bashT" ]
then
    cat apache.log | grep POST
elif [ $1 == "PUT" ]
then
    cat apache.log | grep PUT
elif [ $1 == "DELETE" ]
then
    cat apache.log | grep DELETE
fi
```

Mas como você pode ver, esse código não está tão elegante, pois ainda temos alguns problemas a serem resolvidos sobre a validação da entrada do parâmetro. Vamos focar em como podemos melhorá-lo.

Se traduzirmos esse código, teremos algo como:

- Se o parâmetro for GET, então faça o filtro com o GET.
- Se o parâmetro for POST, então faça o filtro com POST.
- Se o parâmetro for PUT...

Perceba que o código está bem verboso em sua leitura. O que podemos fazer para mudá-lo? Nós simplesmente queremos validar as possíveis entradas que podem ter por parâmetro na entrada passada pelo usuário.

Caso o parâmetro seja **GET**, caso seja **POST**, caso seja **PUT**, caso seja **DELETE**, e também caso não seja nenhum deles, simplesmente mostre uma mensagem dizendo que o parâmetro passado não é válido.

Usaremos o ***CASE***, uma outra alternativa para realizar a validação de parâmetros! Abriremos o **gedit**, assim conseguiremos fazer uma manipulação um pouco maior:

```
$ gedit filtro-requisicao.sh
```

Apagaremos todos os `if`'s, pois usaremos a partir de agora o `case`.

```
#!/bin/bash

cd ~/apache-log
```

```
case $1 in
        GET)
        cat apache.log | grep GET
        ;;
```

Essa foi a primeira validação, onde caso o parâmetro seja do tipo GET, faremos o filtro no `apache.log`. Com o `| grep`, consideramos somente a entrada das requisições GET. Logo depois, utilizamos o *terminador* (`;;`).

Usaremos a mesma lógica para os outros tipos de requisições.

```
case $1 in
        GET)
        cat apache.log | grep GET
        ;;

        POST)
        cat apache.log | grep POST
        ;;

        PUT)
        cat apache.log | grep PUT
        ;;

        DELETE)
        cat apache.log | grep DELETE
        ;;
```

E se o usuário colocar qualquer outra palavra que não seja uma dessas quatro requisições? Colocaremos uma mensagem para ele, dizendo que o parâmetro não é válido. Como podemos pegar **qualquer outro parâmetro**? Isso mesmo, utilizando o `*)`!

```
case $1 in
        GET)
        cat apache.log | grep GET
        ;;

        POST)
        cat apache.log | grep POST
        ;;

        PUT)
        cat apache.log | grep PUT
        ;;

        DELETE)
        cat apache.log | grep DELETE
        ;;

        *)
        echo "O parametro passado nao e valido"
        ;;
esac
```

Uma vez que abrimos o `case`, temos que fechá-lo digitando "case" (**esac**) ao contrário. Vamos clicar em "Save" e fechar o gedit.

Agora é a hora de executar o script.

```
$ bash filtro-requisicao.sh GET
```

Foi possível filtrar somente as requisições do tipo GET. Passando a requisição POST, também vamos ter os resultados das requisições POST, e assim por diante. Se nós passarmos um parâmetro diferente dessas requisições, será mostrado a mensagem dizendo que o parâmetro não é válido.

Com isso, conseguimos verificar o que o usuário irá passar como parâmetro. Entretanto, ainda temos o problema onde o usuário pode inserir uma requisição em minúsculo, e pelo fato do `case` considerar somente entradas com as palavras em **maiúsculo**. Dessa forma, o parâmetro cairá no último case, mostrando a mensagem de que o parâmetro passado não é válido.

Qual estratégia podemos adotar nesse caso? Bom, sabemos que nos arquivos, as requisições estão em maiúsculas. Uma das formas de abordagem é que, sempre que recebermos o parâmetro do usuário, simplesmente transformamos essas palavras em maiúsculo! Pois se tratarmos assim, também será possível fazer a verificação.

Para essa tarefa, usaremos o **awk**!!

Então, redirecionaremos o "get" para que o awk possa tratar. Queremos que o `awk` transforme o parâmetro que será passado, em maiúscula. Para isso, temos que utilizar uma função do `awk` chamada de **toupper()**.

```
$ echo get | awk '{ print toupper($1) }'
```

Mesmo que o usuário passe como parâmetro palavras em minúsculo, através desse redirecionamento para o `awk`, estamos transformando-a em maiúscula, que é exatamente o tipo de requisições que encontraremos no arquivo.

Vamos copiar esse comando, para podermos utilizá-lo no script. Depois colamos logo abaixo do comando `cd ~/apache-log`. Não podemos nos esquecer de que, como ele é um comando, temos que envolvê-lo entre `$()`. Após envolvê-lo entre os parênteses, armazenaremos o resultado em uma variável chamada `letra_maiuscula`

```
#!/bin/bash

cd ~/apache-log

letra_maiuscula=$(echo get | awk '{ print toupper($1) }')
```

No lugar do `get`, pegaremos o parâmetro passado pelo usuário.

```
letra_maiuscula=$(echo $1 | awk '{ print toupper($1) }')
```

Com essa linha criada, sempre estaremos pegando o parâmetro do usuário e transformando em maiúsculas. A verificação do `case` vai ser já considerando esse caso com a transformação para o maiúsculo.

```
case $letra_maiuscula in
```

Testaremos agora. Para sair e salvar, usamos o "Ctrl + X" e "Y".

No terminal, mandaremos o seguinte comando com o parâmetro:

```
$ bash filtro-requisicao.sh get
```

Mesmo que tenhamos mandado uma requisição em minúsculo, foi retornado para nós todos os registros dessa requisição. Mesmo que o usuário mande uma requisição com a primeira letra maiúscula, e o resto em minúsculo, também irá funcionar! Pois o `awk` sempre irá colocar as letras dos parâmetros em maiúsculas.

Só que é possível melhorarmos um pouco mais o código. Vamos supor que o usuário se esqueceu de passar o parâmetro. Nesse caso, o que é retornado é a mensagem `O parametro passado nao e valido`. Mas, seria mais interessante para o usuário, se ele fosse lembrado de passar o parâmetro, não é mesmo?

Vamos realizar essa verificação! Abriremos novamente o script com o comando `nano filtro-requisicao.sh`.

Quando o usuário esquece de passar algum parâmetro, o conteúdo do `$1` será **vazio**.

```
#!/bin/bash

cd ~/apache-log

if [ -z $1 ]

letra_maiuscula=$(echo $1 | awk '{ print toupper($1) }')
```

Com o comando `-z`, verificamos se o conteúdo é vazio ou não. Se for vazio, vamos colocar uma mensagem para que ele insira uma requisição. Assim que ele inserir esse parâmetro, pegaremos esse parâmetro de volta, fazendo uma leitura do parâmetro.

```
#!/bin/bash

cd ~/apache-log

if [ -z $1 ]
then
    read -p "Voce esqueceu de colocar o parametro (GET,PUT,POST,DELETE): " requisicao
```

Nós esperamos que o usuário coloque esse parâmetro esquecido, para que possamos armazená-lo na variável **`requisicao`**. Nessa situação, é possível que o usuário coloque letras minúsculas, sendo necessário transformar todo esse parâmetro para **letras maiúsculas**. Simplesmente, copiaremos o código que faz essa validação para nós.

```
#!/bin/bash

cd ~/apache-log

if [ -z $1 ]
```

```
then
    read -p "Voce esqueceu de colocar o parametro (GET,PUT,POST,DELETE): " requisicao
    letra_maiuscula=$(echo $1 | awk '{ print toupper($1) }')

letra_maiuscula=$(echo $1 | awk '{ print toupper($1) }')
```

É necessário trocar de `echo $1` para `echo $requisicao` , e o código ficará assim:

```
#!/bin/bash

cd ~/apache-log

if [ -z $1 ]
then
    read -p "Voce esqueceu de colocar o parametro (GET,PUT,POST,DELETE): " requisicao
    letra_maiuscula=$(echo $requisicao | awk '{ print toupper($1) }')

letra_maiuscula=$(echo $1 | awk '{ print toupper($1) }')
```

Com isso, garantimos que mesmo que o usuário esqueça de passar um parâmetro, mostramos uma mensagem para ele, e assim ele colocaria esse parâmetro que será armazenado na variável `requisicao` . Depois pegamos o conteúdo dessa variável, e armazenamos na variável `letra_maiuscula` . Tudo isso acontecerá, CASO o usuário se esqueça de passar o parâmetro.

Caso o `$1` **não** esteja vazio, ou seja, caso o usuário se lembrou de inserir um parâmetro, acrescentamos o `else` :

```
#!/bin/bash

cd ~/apache-log

if [ -z $1 ]
then
    read -p "Voce esqueceu de colocar o parametro (GET,PUT,POST,DELETE): " requisicao
    letra_maiuscula=$(echo $requisicao | awk '{ print toupper($1) }')
else

    letra_maiuscula=$(echo $1 | awk '{ print toupper($1) }')
fi

case $letra_maiuscula in
```

Vamos sair para testar, com "Ctrl + X" e "Y".

Faremos uma simulação agora, quando o usuário acaba se esquecendo de mandar o parâmetro. Após rodar o comando `bash filtro-requisicao.sh` , é mostrado a mensagem:

```
$ bash filtro-requisicao.sh
Voce esqueceu de colocar o parametro (GET,PUT,POST,DELETE):
```

Legal, agora colocaremos a requisição `get` , por exemplo:

```
Voce esqueceu de colocar o parametro (GET,PUT,POST,DELETE): get
```

Como resultado, recebemos somente as requisições GET! Mas, não necessariamente o usuário irá mandar uma requisição assim como pede a mensagem. Pode ser que ele dê "Enter" uma segunda vez. E o que foi retornado, é a mensagem antiga "O parametro passado nao e valido". Fazendo isso, a variável requisição está com um conteúdo diferente dos quatro parâmetros de requisição. Então, queremos forçar sempre que a variável requisição estiver vazia, para ela forçar o usuário colocar alguma requisição, não permitindo que ele passe a diante. Vamos ver como podemos fazer isso?

Enquanto a variável `requisicao` estiver vazia, forçaremos o usuário a passar um parâmetro, não vamos seguir a diante.

```
if [ -z $1 ]
then
    while [ -z $requisicao ]
    do
        read -p "Voce esqueceu de colocar o parametro (GET,PUT,POST,DELETE): " requisicao
        letra_maiuscula=$(echo $requisicao | awk '{ print toupper($1) }')
    done
else
    letra_maiuscula=$(echo $1 | awk '{ print toupper($1) }')
fi
```

Ou seja, enquanto o usuário não passar a requisição, ele ficará eternamente nesse loop, e será forçado a passar algum parâmetro para o comando. Vamos testar?

Executamos o comando `bash filtro-requisicao.sh` , e damos o "Enter", pois esquecemos do parâmetro na primeira vez. O que nos retornou foi a mensagem de que nos esquecemos de colocar o parâmetro. Se tentarmos colocar o "Enter" mais uma vez, a requisição estará vazia, e se ela estiver vazia, será retornado a mesma mensagem até que ele coloque mande alguma requisição.

Dessa forma conseguimos filtrar os casos que o usuário pode esquecer de passar um parâmetro, e com o `while` garantimos que ele tem que passar um parâmetro aqui em algum momento. Conseguimos também fazer a verificação de um parâmetro que não existe (diferente de GET, POST, PUT e DELETE), onde o usuário pode passar alguma outra palavra utilizando o `case` , e também, transformamos as palavras em minúsculo para maiúsculo com o `awk` , assim a tarefa foi resolvida.