03

Comparando parâmetro com regex

Transcrição

Em uma reunião com os diretores da *Multillidae*, nos informaram que existe um arquivo de log (https://drive.google.com/open?id=0BzmYQVmw4W7nZmxnenBZdVpRalE) de um servidor, onde eles precisam realizar alguns filtros para buscar resultados a fim de analisá-los.

Vamos abrir o terminar para checar esse arquivo de log, utilizando o comando `ls`.

Com o comando `cd apache-log`, podemos entrar no diretório, e logo depois encontramos o arquivo de log do servidor apache. Para melhorar a nossa análise, vamos trazer todo o resultado que consta nesse arquivo para o terminal.

```
$ cat apache.log
```

Como resultado, temos informações presentes que precisam ser analisadas pelos diretores da *Multillidae*, onde é necessário realizar um filtro. Esse filtro consiste em se basear pelo endereço IP de um usuário, para que seja possível ver os acessos desse determinado endereço.

Faremos um teste. Vamos supor que iremos verificar o acesso do respectivo ao endereço IP `47.86.228.66`. Nas etapas anteriores, já utilizamos algumas espécies de filtros através do **grep**, por isso, vamos utilizá-lo agora, redirecionando a saída para o próprio grep.

```
$ cat apache.log | grep 47.86.228.66
```

Ao redirecionar a saída para o grep, queremos que ele filtre somente os resultados respectivos ao endereço IP apresentado.

Após o "Enter", obtemos uma URL acessada pelo endereço IP. Maravilha! Vimos que esse comando funcionou corretamente e nos trouxe informações relevantes, por isso, vamos utilizá-lo em um novo script que será criado no diretório de scripts para os diretores da *Multillidae*.

Voltamos para a "Home", e depois entramos na pasta `/Scripts` com o comando `cd ~/Scripts/`.

Com o comando `nano`, criamos novos arquivos, no caso, chamaremos o novo script de **filtrando-resultado-apache.sh**.

A primeira linha do script deve conter o **interpretador**.

```
#!/bin/bash
```

O primeiro passo a ser executado é *entrar no diretório* `/apache-log`, pois é onde se encontra o arquivo.

```
#!/bin/bash

cd ~/apache-log
```

Após estar dentro do diretório `/apache-log`, queremos mostrar no terminal, o resultado do arquivo de log, entretanto, queremos redirecionar a saída para o **grep**, assim é possível realizar o filtro de acordo com o endereço IP passado pelo usuário.

Como esse endereço será passado pelo usuário, então ele será passado como um **parâmetro**.

Para pegarmos o resultado de um parâmetro, utilizamos o símbolo `$` e o número `1` (*fazendo a referência ao primeiro parâmetro*):

```
#!/bin/bash

cd ~/apache-log

cat apache.log | grep $1
```

Testaremos esse script. Usaremos "Ctrl + X" para sair e "Y" para salvar, utilizaremos o comando para executá-lo, passando o endereço IP como parâmetro:

```
$ bash filtrando-resultado-apache.sh 47.86.228.66
```

O nosso Script, foi capaz de retornar o resultado dos acessos ao respectivo endereço IP. Voltando ao Script, repare que nós simplesmente estamos pegando esse parâmetro que é passado pelo usuário, e tentamos realizar esse filtro. Porém, **o que nos garante que o parâmetro passado pelo usuário, é um formato de um endereço IP?**

Ele poderia passar uma palavra qualquer, não é mesmo? Não estamos fazendo nenhuma espécie de validação. E é justamente isso o que foi pedido para nós!

Vamos fazer uma **validação** para certificar que o formato padrão de entrada seja somente o *endereço IP*.

Usaremos as ***expressões regulares*** para conseguir validar essa entrada de dados. E como podemos usar as expressões regulares a fim de nos ajudar com essa tarefa?

Vamos abrir o editor de texto **gedit** para analisarmos o endereço IP.

Um endereço IP bem comum de se encontrar, é o `192.168.1.10`. Nesse endereço, todos os intervalos são formados por números, e os intervalos são separados por pontos (*temos que levar em conta essas duas informações para fazer a validação*).

A primeira etapa da validação será de permitir que a entrada seja somente **um número**.

Utilizando o `grep` para filtrar os dígitos de zero até nove, vamos colocar dessa forma: `[0-9]`, assim estamos dizendo que os números podem variar de **0** até o **9**. Mas repare que no primeiro e no segundo intervalo, existem não só 1, mas 3 algarismos, e no terceiro e quarto intervalos, temos respectivamente 1 e 2 algarismos.

Com isso, vemos que a nossa validação precisa aceitar uma *variação* de **um a três** algarismos. Temos que especificar que o valor mínimo de algarismos é `1` e o valor máximo de algarismos por cada intervalo é `3`. Representamos assim:

```
[0-9]{1,3}
```

Feita a validação do **primeiro intervalo**, temos que realizar a validação do segundo! Só que para chegar ao segundo intervalo, temos a separação do ponto.

O segundo intervalo segue a mesma regra de validação do primeiro, pois é formado por números que podem variar de um a três algarismos.

```
[0-9]{1,3}.[0-9]{1,3}
```

Para chegar ao terceiro intervalo, temos um ponto, e novamente, esse intervalo receberá a mesma validação, pois é formado por números que podem variar de um a três algarismos. O mesmo acontece com o último intervalo.

```
[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}
```

Bom, é claro que podemos melhorar a nossa expressão regular. Se repararmos, o trecho `[0-9]{1,3}.` se repete por 3 vezes. Poderíamos simplesmente agrupar esse valor entre `()` e colocar um **quantificador** para dizer que a parte agrupada irá se repetir por 3 vezes:

```
([0-9]{1,3}.){3}[0-9]{1,3}
```

Vamos ver como essa expressão regular irá se comportar como estamos esperando.

Para nos ajudar nessa verificação, utilizaremos o site regex101.com (https://regex101.com/). Existem outros sites que também conseguem essa validação.

Copiaremos a expressão criada, e vamos colar no campo **REGULAR EXPRESSION** nesse mesmo site. Depois, no campo **TEST STRING**, colocaremos o endereço IP que havíamos testado: `192.168.1.10`.

O site nos retornou `1 match`, significa que o resultado `([0-9]{1,3}.){3}[0-9]{1,3}` teve um padrão aceito em nossa expressão regular. Vamos fazer o seguinte. Tentaremos trocar o **ponto** por uma **vírgula**. O site nos retorna algo diferente de antes? Não! Continua sendo exibido `1 match`!

Isso acontece porque colocamos o ponto em nossa expressão regular, porém não realizamos o escape dele, dizendo que literalmente a divisão entre esses intervalos precisa ser somente o **ponto**. Para dizer que somente o ponto é aceito para dividir esses intervalos, colocamos a barra `\` antes do ponto, assim nós o "escapamos".

Sempre que fazemos isso, estamos dizendo que a divisão entre os intervalos só pode ser o ponto!

```
([0-9]{1,3}\.){3}[0-9]{1,3}
```

Para testar, temos o endereço IP, podendo variar de um a três algarismos. Tentaremos colocar um algarismo a mais do permitido no final, a fim de testá-lo. Então, o endereço IP ficará assim: `192.168.1.1000`.

Como podemos ver, esse endereço *é compatível* com a expressão regular. Ele só é compatível, porque o que está sendo considerado é somente **192.168.1.100**0. Mas se o usuário digitar dessa forma no script (com um dígito a mais), o padrão seria aceito, o que não deveria acontecer.

Sendo assim, é interessante colocarmos um **delimitador**, dizendo que nada no final ou no início pode existir!

O delimitador de palavras é o **Word Boundary**, simbolizado por `\b`. Adicionando esse delimitador nas extremidades da expressão, não será reconhecido nenhum algarismo que ultrapasse a quantidade de **3 por intervalo**.

```
\b([0-9]{1,3}\.){3}[0-9]{1,3}\b
```

Bom, com isso, sabemos que o endereço IP `192.168.1.1000` não será aceito pois ele não está respeitando as regras da expressão.

Se tentarmos, por exemplo, substituir um número por uma letra. `1a2.168.1.10` O que nos retornaria? Com certeza não foi aceito, pois a expressão regular não permite letras, somente números.

Vamos copiar a nossa expressão regular, para colocá-la no script.

É interessante que essa expressão regular esteja dentro de uma variável, assim ficará mais fácil de manuseá-la.

```
#!/bin/bash

cd ~/apache-log

regex="\b([0-9]{1,3}\.){3}[0-9]{1,3}\b"

cat apache.log | grep $1
```

Agora, validaremos o parâmetro passado pelo nosso usuário utilizando essa expressão regular através do `if`.

Dentro do `if`, iremos comparar o parâmetro passado pelo usuário com a expressão regular.

Utilizamos 2 colchetes (**[]**) para envolver a comparação, e utilizamos (**=~**) para fazer a comparação.

```
#!/bin/bash

cd ~/apache-log

regex="\b([0-9]{1,3}\.){3}[0-9]{1,3}\b"
if [[ $1 =~ $regex ]]
cat apache.log | grep $1
```

Se o parâmetro passado pelo usuário está de acordo com a expressão regular, será feita a verificação do arquivo de log.

```
if [[ $1 =~ $regex ]]
then
    cat apache.log | grep $1
```

Se não for um parâmetro compatível com a expressão regular, será impresso uma mensagem para o usuário.

```
if [[ $1 =~ $regex ]]
then
    cat apache.log | grep $1
else
```

```
    echo "Formato não é válido"
fi
```

Salvamos as alterações com "Ctrl + X" e "Y", e realizamos a execução do script passando o IP `47.86.228.66` como parâmetro.

```
$ bash filtrando-resultado-apache.sh 47.86.228.66
```

Se tudo estiver certo, o script será capaz de filtrar esse endereço IP.

Vamos supor que o quarto intervalo desse mesmo endereço IP tenha quatro algorismos, e assim, nós rodamos o script com o novo intervalo:

```
$ bash filtrando-resultado-apache.sh 47.86.228.1000
```

Quatro algarismos não é um formato válido. Por essa razão, temos como resultado a mensagem `Formato nao e valido .` Vamos deixar do jeito que estava, e agora, retiramos o ponto e colocarmos uma vírgula. O que vamos obter?

Novamente, obtemos a mesma mensagem! Isso significa que o formato não é válido pois a expressão regular não aceita vírgulas. Da mesma forma, se substituirmos qualquer número por uma letra qualquer, vamos obter a mesma mensagem de erro.

Então, com isso, nós conseguimos fazer uma verificação maior do que o nosso usuário pode estar passando como parâmetro.

Também podemos analisar um outro cenário, não menos interessante. Vamos rodar o script, passando como parâmetro o endereço IP `1.1.1.1` .

Esse é um endereço IP válido, pois ele está dentro das regras da expressão. Entretanto, esse endereço pode **não existir** em nosso endereço de log. Verificaremos se o parâmetro passado pelo usuário, existe ou não no arquivo de log. Como podemos realizar essa verificação?

Nas parte 1 do curso de Shell Scripting anteriores, abordamos os **status de saída**, e encontramos uma oportunidade de usá-los novamente nesta aula.

Os status de saída irão, justamente, checar se o parâmetro passado pelo usuário, existe ou não no arquivo de log.

> RELEMBRANDO: Se tudo ocorreu bem, o status será **0**. Se não, o status será **diferente de zero**.

Abriremos o script novamente. Validaremos o status de saída do comando `cat` .

```
if [[ $1 =~ $regex ]]
then
    cat apache.log | grep $1
    if [ $? -ne 0 ]
    then
        echo "O endereco IP procurado nao esta presente no arquivo"
    fi
else
```

```
        echo "Formato não é válido"
    fi
```

Vamos testar! Usaremos novamente o endereço IP `1.1.1.1`.

Recebemos a mensagem de que `O endereco IP procurado nao esta presente no arquivo`. E se usarmos o endereço que já existe no arquivo de log?

Como o endereço já existe dentro do arquivo de log, será filtrado e retornado para nós o resultado desse endereço IP!

Agora, o nosso script já consegue ajudar os diretores na tarefa de filtrar o resultado baseado no endereço IP, e consegue fazer uma validação um pouco melhor do parâmetro que está sendo passado pelo usuário.