05

Criando arquivos de log completo

Transcrição

Com o comando `ps -p 14009 -o size` é impresso o tamanho da memória alocada do processo informado. Entretanto, o resultado que temos é um tamanho em **kilobytes** mostrado a seguir:

```
  SIZE
931348
```

Ao dividir esse valor em kilobytes por **1024**, conseguimos ter o valor em **megabytes**!

O primeiro passo é eliminar a linha de cabeçalho `SIZE`, pois queremos somente o número. Como podemos solucionar esse problema?

Como já visto, utilizamos o **grep** para redirecionar a saída desse resultado, mostrando somente o valor numérico.

```
$ ps -p 14009 -o size | grep [0-9]
```

E temos o seguinte resultado:

```
931348
```

Agora que temos o resultado com números, podemos pensar em fazer a divisão com o comando `echo 931348/1024`. Porém, quando fazemos isso, o echo imprime literalmente o valor `931348/1024`, mas esperamos que uma operação de divisão seja feita.

Para que possamos ter essa operação de divisão, utilizaremos a ferramenta `bc` que vai realizar a divisão entre os valores e passaremos os parâmetros da divisão, através de `>>>`:

```
$ bc <<< 931348/1024
```

E temos o resultado da divisão:

```
909
```

Se calcularmos essa operação na calculadora, veremos que o valor será de `909,51953125`. Podemos recalcular de uma forma que tenha uma precisão maior. Suponhamos que valor tenha duas casas após a vírgula.

Para isso, trabalharemos com **escala de 2**, e envolveremos a operação entre aspas.

```
$ bc <<< "scale=2;931348/1024"
```

E como resultado, obtemos esse valor `909.51` , com duas casas após a vírgula. Vamos copiar os dois comandos criados, para que nós coloquemos no script `processos-memoria.sh` .

```
for pid in $processos
do
    nome_processo=$(ps -p $pid -o comm=)
    echo -n $(date +%F,%H:%M:%S,) >> $nome_processo.log
    tamanho_processo=$(ps -p $pid -o size | grep [0-9])
done
```

Então, colamos o comando testado no terminal, o envolvemos no $(), apagamos o valor do processo Firefox `14009` para que seja possível pegar o número do processo respectivo da variável `$pid` e ,assim, armazenamos o resultado na variável `tamanho_processo` .

O próximo passo é colar o segundo comando no Script:

```
for pid in $processos
do
    nome_processo=$(ps -p $pid -o comm=)
    echo -n $(date +%F,%H:%M:%S,) >> $nome_processo.log
    tamanho_processo=$(ps -p $pid -o size | grep [0-9])
    echo $(bc <<< "scale=2;$tamanho_processo/1024") >> $nome_processo.log
done
```

Redirecionamos a impressão para o arquivo `.log` . Como toda a linha é um comando, o envolvemos em `$()` . O valor numérico **931348** é o valor estático do processo do Firefox. Por essa razão, pegamos o conteúdo do `$tamanho_processo` que será dividido por 1024.

Para melhorar o script, sabemos que o resultado será em megabytes, por isso, colocamos o comando entre aspas e dizemos que seu resultado é um valor em megabytes:

```
echo "$(bc <<< "scale=2;$tamanho_processo/1024") MB">> $nome_processo.log
```

Ao chegar até aqui, fizemos um grande avanço. Mas, vamos fazer uma melhoria. Para que os arquivos não fiquem espalhados por todos os lugares, vamos colocá-los dentro de um diretório. Aqui mesmo dentro do diretório `/Scripts` e verificaremos se existe um diretório chamado de `/log` , onde salvaremos todos esses arquivos. Caso ele não exista, vamos criá-lo.

Logo no início do script, faremos essa verificação.

```
#!/bin/bash

if [ ! -d log ]
then
    mkdir log
fi
```

Agora com a certeza de que o diretório `/log` vai existir, podemos salvar os arquivos dentro do `/log` que foi criado.

```
#!/bin/bash

if [ ! -d log ]
then
    mkdir log
fi

processos=$(ps -e -o pid --sort -size | head -n 11 | grep [0-9])
for pid in $processos
do
    nome_processo=$(ps -p $pid -o comm=)
    echo -n $(date +%F,%H:%M:%S,) >> log/$nome_processo.log
    tamanho_processo=$(ps -p $pid -o size | grep [0-9])
    echo $(bc <<< "scale=2;$tamanho_processo/1024") >> log/$nome_processo.log
done
```

É importante saber se os arquivos foram salvos com sucesso ou se houve algum problema. Então, podemos fazer essa verificação por meio do **status de saída**. Caso ele seja `0` é porque está tudo "ok". Se for diferente de `0`, é porque um problema ocorreu.

Como já vimos, podemos envolver todo o código em uma função e, assim, verificar qual será o status de saída dessa função.

Chamaremos essa função de `processos_memoria()`:

```
processos_memoria(){
processos=$(ps -e -o pid --sort -size | head -n 11 | grep [0-9])
for pid in $processos
do
    nome_processo=$(ps -p $pid -o comm=)
    echo -n $(date +%F,%H:%M:%S,) >> log/$nome_processo.log
    tamanho_processo=$(ps -p $pid -o size | grep [0-9])
    echo $(bc <<< "scale=2;$tamanho_processo/1024") >> log/$nome_processo.log
done
}
```

Após criar a função vamos invocá-la no final do código.

```
processos_memoria(){
processos=$(ps -e -o pid --sort -size | head -n 11 | grep [0-9])
for pid in $processos
do
    nome_processo=$(ps -p $pid -o comm=)
    echo -n $(date +%F,%H:%M:%S,) >> log/$nome_processo.log
    tamanho_processo=$(ps -p $pid -o size | grep [0-9])
    echo $(bc <<< "scale=2;$tamanho_processo/1024") >> log/$nome_processo.log
done
}

processos_memoria
```

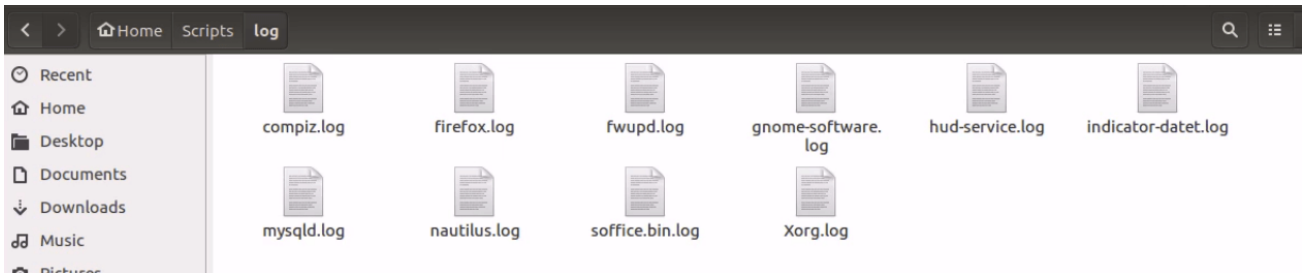
Uma vez chamada a função, vamos verificar seu status de saída:

```
processos_memoria
if [ $? -eq 0 ]
then
    echo "Os arquivos foram salvos com sucesso"
else
    echo "Houve um problema na hora de salvar os arquivos"
fi
```

Hora de testar o script!

Não esqueça de sair e salvar: "Ctrl + X" "Y"

Ao executar o script `bash processos-memoria.sh`, temos o status de saída: os arquivos foram salvos sem nenhum problema. Ao acessar o diretório `/log`, encontramos arquivos com os nomes dos processos junto de suas respectivas extensões `.log`.



Clicando em um desses arquivos, por exemplo, vamos encontrar informações parecidas com essa:

```
2017-07-21,16:57:47,909.51 MB
```

Temos a **data**, o **horário** e a **locação em MB**. Comentamos em um momento anterior sobre ter colocado o símbolo `>>` antes de salvar o processo na pasta log. Isso quer dizer que agora eles têm o conteúdo que acabamos de executar. Se executarmos novamente, as informações serão mantidas e, ainda, serão acrescentadas as novas informações. Vamos ver se realmente ocorre isso! Vamos rodar novamente o script.

Clicando em algum dos arquivos, vamos ter uma linha a mais de informações e vemos que a informação anterior se manteve, assim como havíamos planejado.

```
2017-07-21,16:57:47,909.51 MB
2017-07-21,16:59:05,909.51 MB
```

Tarefa concluída com sucesso!