
Dave Hamilton - www.sxc.hu

Curso de Shell Script

Papo de Botequim
Parte VIII

Chegou a hora de fazer como Jack e dividir os programas em pedacinhos. Com funções e chamadas externas os scripts ficam menores, a manutenção mais fácil e ainda por cima reaproveitamos código.

por Júlio Cezar Neves

– Eaê, cara, tudo bem?

– Tudo beleza! Eu queria te mostrar o que fiz mas já sei que você vai querer molhar o bico primeiro, né?

– Só pra te contrariar, hoje não quero. Vai, mostra logo aí o que você fez.

– Poxa, o exercício que você passou é muito grande. Dá uma olhada na **listagem 1** e vê como eu resolvi:

– É, o programa tá legal, tá todo estruturadinho, mas gostaria de fazer alguns poucos comentários: só para relembrar, as seguintes construções: `[ ! $Album ] &&` e `[ $Musica ] ||` representam a mesma coisa, isto é: no caso da primeira, testamos se a variável `$Album` não (`!`) tem nada dentro, então (`&&`)... Na segunda, testamos se `$Musica` tem algum dado, senão (`||`)...

Se você reclamou do tamanho do programa, é porque ainda não te dei algumas dicas. Repare que a maior parte do script é para mostrar mensagens centralizadas na penúltima linha da tela. Repare ainda que algumas mensagens pedem um `S` ou um `N` como resposta e outras são só de advertência. Isso é um caso típico que pede o uso de funções, que seriam escritas somente uma vez e executadas em diversos pontos do script. Vou montar duas funções para resolver esses casos e vamos incorporá-las ao seu programa para ver o resultado final.

– Chico! Agora traz dois chopes, um sem colarinho, para me dar inspiração. E você, de olho na **listagem 2**.

Como podemos ver, uma função é definida quando digitamos `nome_da_função ()` e todo o seu corpo está entre chaves (`{}`). Já conversamos aqui no boteco sobre passagem de parâmetros e as funções os recebem da mesma forma, isto é, são parâmetros posicionais (`$1, $2, ..., $n`). Todas as regras que se aplicam à passagem de parâmetros para programas também valem para funções, mas é muito importante realçar que os parâmetros passados para um programa não se confundem com aqueles que são passados para suas funções. Isso significa, por exemplo, que o `$1` de um script é diferente do `$1` de uma de suas funções internas.

Repare que as variáveis `$Msg, $TamMsg` e `$Col` são de uso restrito dessa rotina e, por isso, foram criadas como variáveis locais. A razão é simplesmente a economia de memória, já que ao sair da rotina elas serão devidamente detonadas, coisa que não aconteceria se eu não tivesse usado esse artifício.

A linha de código que cria a variável local `Msg` concatena ao texto recebido (`$1`) um parêntese, a resposta padrão (`$2`) em caixa alta, uma barra, a outra resposta (`$3`) em caixa baixa e finaliza fechando o parêntese. Uso essa convenção para, ao

Listagem 1: *musinc5.sh*

```
01 $ cat musinc5.sh
02 #!/bin/bash
03 # Cadastra CDs (versao 5)
04 #
05 clear
06 LinhaMesg=$((`tput lines` - 3)) # Linha onde serão mostradas as msgs para o operador
07 TotCols=$(tput cols)           # Qtd colunas da tela para enquadrar msgs
08 echo "
                                   Inclusão de Músicas
                                   ======= == =======
                  Título do Álbum:
                                        | Este campo foi
                  Faixa:               < criado somente para
                                        | orientar o preenchimento
                  Nome da Música:
                  Intérprete:"           # Tela montada com um único echo
09 while true
10 do
11     tput cup 5 38; tput el    # Posiciona e limpa linha                    ➡
```

```
12    read Album
13    [ ! "$Album" ] &&         # Operador deu <ENTER>
14        {
15            Msg="Deseja Terminar? (S/n)"
16            TamMsg=${#Msg}
17            Col=$(((TotCols - TamMsg) / 2))        # Centraliza msg na linha
18            tput cup $LinhaMesg $Col
19            echo "$Msg"
20            tput cup $LinhaMesg $((Col + TamMsg + 1))
21            read -n1 SN
22            tput cup $LinhaMesg $Col; tput el      # Apaga msg da tela
23            [ $SN = "N" -o $SN = "n" ] && continue # $SN é igual a N ou (-o) n?
24            clear; exit                            # Fim da execução
25        }
26    grep "^$Album\^" musicas > /dev/null &&
27        {
28            Msg="Este álbum já está cadastrado"
29            TamMsg=${#Msg}
30            Col=$(((TotCols - TamMsg) / 2))        # Centraliza msg na linha
31            tput cup $LinhaMesg $Col
32            echo "$Msg"
33            read -n1
34            tput cup $LinhaMesg $Col; tput el      # Apaga msg da tela
35            continue                               # Volta para ler outro álbum
36        }
37    Reg="$Album^"                        # $Reg receberá os dados para gravação
38    oArtista=                            # Variável que guarda artista anterior
39    while true
40    do
41        ((Faixa++))
42        tput cup 7 38
43        echo $Faixa
44        tput cup 9 38                    # Posiciona para ler música
45        read Musica
46        [ "$Musica" ] ||                 # Se o operador tiver dado <ENTER>...
47            {
48               Msg="Fim de Álbum? (S/n)"
49               TamMsg=${#Msg}
50               Col=$(((TotCols - TamMsg) / 2))        # Centraliza msg na linha
51               tput cup $LinhaMesg $Col
52               echo "$Msg"
53               tput cup $LinhaMesg $((Col + TamMsg + 1)
54               read -n1 SN
55               tput cup $LinhaMesg $Col; tput el      # Apaga msg da tela
56               [ "$SN" = N -o "$SN" = n ] && continue # $SN é igual a N ou (-o) n?
57               break                                  # Sai do loop para gravar
58            }
59        tput cup 11 38                               # Posiciona para ler Artista
60        [ "$oArtista" ] && echo -n "($oArtista) "  # Artista anterior é default
61        read Artista
62        [ "$Artista" ] && oArtista="$Artista"
63        Reg="$Reg$oArtista~$Musica:"         # Montando registro
64        tput cup 9 38; tput el               # Apaga Música da tela
65        tput cup 11 38; tput el              # Apaga Artista da tela
66    done
67    echo "$Reg" >> musicas                   # Grava registro no fim do arquivo
38    sort musicas -O musicas                  # Classifica o arquivo
69 done
```

mesmo tempo, mostrar as opções disponíveis e realçar a resposta oferecida como padrão.

Quase no fim da rotina, a resposta recebida (`$SN`) é convertida para caixa baixa (minúsculas) de forma que no corpo do programa não precisemos fazer esse teste. Veja na **listagem 3** como ficaria a função para exibir uma mensagem na tela.

Essa é uma outra forma de definir uma função: não a chamamos, como no exemplo anterior, usando uma construção com a sintaxe `nome_da_função ()`, mas sim como `function nome_da_função`. Em nada mais ela difere da anterior, exceto que, como consta dos comentários, usamos a variável `$*` que, como já sabemos, representa o conjunto de todos os parâmetros passados ao script, para que o programador não precise usar aspas envolvendo a mensagem que deseja passar à função.

Para terminar com esse blá-blá-blá, vamos ver na **listagem 4** as alterações no programa quando usamos o conceito de funções:

Repare que a estrutura do script segue a ordem *Variáveis Globais*, *Funções* e *Corpo do Programa*. Esta estruturação se deve ao fato de Shell Script ser uma linguagem interpretada, em que o programa é lido da esquerda para a direita e de cima para baixo. Para ser vista pelo script e suas funções, uma variável deve ser declarada (ou inicializada, como preferem alguns) antes de qualquer outra coisa. Por sua vez, as funções devem ser declaradas antes do corpo do programa propriamente dito. A explicação é simples: o interpretador de comandos do shell deve saber do que se trata a função antes que ela seja chamada no programa principal.

Uma coisa bacana na criação de funções é fazê-las tão genéricas quanto possível, de forma que possam ser reutilizadas em outros scripts e aplicativos sem a necessidade de reinventarmos a roda. As duas funções que acabamos de ver são bons exemplos, pois é difícil um script de entrada de dados que não use uma rotina como a `MandaMsg` ou que não interaja com o operador por meio de algo semelhante à `Pergunta`.

Conselho de amigo: crie um arquivo e anexe a ele cada função nova que você criar. Ao final de algum tempo você terá uma bela biblioteca de funções que lhe poupará muito tempo de programação.

O comando source

Veja se você nota algo de diferente na saída do *ls* a seguir:

```
$ ls -la .bash_profile
-rw-r--r--  1 Julio  unknown   4511 Mar 18 17:45 .bash_profile
```

Não olhe a resposta não, volte a prestar atenção! Bem, já que você está mesmo sem saco de pensar e prefere ler a resposta, vou te dar uma dica: acho que você já sabe que o `.bash_profile` é um dos scripts que são automaticamente executados quando você se "loga" (ARRGGHH! Odeio esse termo!) no sistema. Agora olhe novamente para a saída do comando *ls* e me diga o que há de diferente nela.

Como eu disse, o `.bash_profile` é executado durante o *logon*, mas repare que ele não tem nenhuma permissão de execução. Isso acontece porque se você o executasse como qualquer outro script careta, no fim de sua execução todo o ambiente por ele gerado morreria junto com o shell sob o qual ele foi executado (você se lembra de que todos os scripts são executados em *sub-shells*, né?). Pois é para coisas assim que existe o comando *source*, também conhecido por "." (ponto). Este comando faz com que o script que lhe for passado como parâmetro não seja executado em um *sub-shell*. Mas é melhor um exemplo que uma explicação em 453 palavras. Veja o scriptzinho a seguir:

```
$ cat script_bobo
cd ..
ls
```

Ele simplesmente deveria ir para o diretório acima do diretório atual. Vamos executar uns comandos envolvendo o `script_bobo` e analisar os resultados:

```
$ pwd
/home/jneves
$ script_bobo
jneves    juliana   paula     silvie
$ pwd
/home/jneves
```

Se eu mandei ele subir um diretório, por que não subiu? Opa, peraí que subiu sim! O sub-shell que foi criado para executar o script tanto subiu que listou os diretórios dos quatro usuários abaixo do diretório `/home`. Só que assim que a execução do script terminou, o sub-shell foi para o beleléu e, com ele, todo o ambiente criado. Agora preste atenção no exemplo abaixo e veja como a coisa muda de figura:

```
$ source script_bobo
jneves    juliana   paula     silvie
$ pwd
/home
$ cd -
/home/jneves
$ . script_bobo
jneves    juliana   paula     silvie
$ pwd
/home
```

Listagem 2: Função *Pergunta*

```
01 Pergunta ()
02     {
03       #  A função recebe 3 parâmetros na seguinte ordem:
04       #  $1 - Mensagem a ser mostrada na tela
05       #  $2 - Valor a ser aceito com resposta padrão
06       #  $3 - O outro valor aceito
07       #  Supondo que $1=Aceita?, $2=s e $3=n, a linha a
08       #  seguir colocaria em Msg o valor "Aceita? (S/n)"
09       local Msg="$1 (`echo $2 | tr a-z A-Z`/`echo $3 | tr A-Z a-z`)"
10       local TamMsg=${#Msg}
11       local Col=$(((TotCols - TamMsg) / 2))  # Centra msg na linha
12       tput cup $LinhaMesg $Col
13       echo "$Msg"
14       tput cup $LinhaMesg $((Col + TamMsg + 1))
15       read -n1 SN
16       [ ! $SN ] && SN=$2                    # Se vazia coloca default em SN
17       echo $SN | tr A-Z a-z                 # A saída de SN será em minúscula
18       tput cup $LinhaMesg $Col; tput el     # Apaga msg da tela
19       return                                # Sai da função
20     }
```

Listagem 3: Função *MandaMsg*

```
01 function MandaMsg
02     {
03          # A função recebe somente um parâmetro
04          # com a mensagem que se deseja exibir.
05          # para não obrigar o programador a passar
06          # a msg entre aspas, usaremos $* (todos
07          # os parâmetros, lembra?) e não $1.
08          local Msg="$*"
09          local TamMsg=${#Msg}
10          local Col=$(((TotCols - TamMsg) / 2)) # Centra msg na linha
11          tput cup $LinhaMesg $Col
12          echo "$Msg"
13          read -n1
14          tput cup $LinhaMesg $Col; tput el     # Apaga msg da tela
15          return                                # Sai da função
16     }
```

Listagem 4: *musinc6.sh*

```
01 $ cat musinc6
02 #!/bin/bash
03 # Cadastra CDs (versao 6)
04 #
05 # Área de variáveis globais
06 # Linha onde as mensagens serão exibidas
07 LinhaMesg=$((`tput lines` - 3))
08 # Quantidade de colunas na tela (para enquadrar as mensagens)
09 TotCols=$(tput cols)
10 # Área de funções
11 Pergunta ()
12     {
13     #  A função recebe 3 parâmetros na seguinte ordem:
14     #  $1 - Mensagem a ser dada na tela
15     #  $2 - Valor a ser aceito com resposta default
16     #  $3 - O outro valor aceito
17     #  Supondo que $1=Aceita?, $2=s e $3=n, a linha
18     #  abaixo colocaria em Msg o valor "Aceita? (S/n)"
19     local Msg="$1 (`echo $2 | tr a-z A-Z`/`echo $3 | tr A-Z a-z`)"
20     local TamMsg=${#Msg}
21     # Centraliza a mensagem na linha
22     local Col=$(((TotCols - TamMsg) / 2))
23     tput cup $LinhaMesg $Col
24     echo "$Msg"
25     tput cup $LinhaMesg $((Col + TamMsg + 1))
26     read -n1 SN
27     [ ! $SN ] && SN=$2     # Se vazia, coloca default em SN
28     echo $SN | tr A-Z a-z  # A saída de SN será em minúsculas
29     tput cup $LinhaMesg $Col; tput el   # Apaga msg da tela
30     return                              # Sai da função
31     }
32 function MandaMsg
33     {
34     # A função recebe somente um parâmetro
35     # com a mensagem que se deseja exibir;
36     # para não obrigar o programador a passar
37     # a msg entre aspas, usaremos $* (todos
38     # os parâmetro, lembra?) e não $1.
39     local Msg="$*"
40     local TamMsg=${#Msg}
41     # Centraliza mensagem na linha
42     local Col=$(((TotCols - TamMsg) / 2))
43     tput cup $LinhaMesg $Col
44     echo "$Msg"
45     read -n1
46     tput cup $LinhaMesg $Col; tput el  # Apaga msg da tela
47     return                             # Sai da função
48     }
49 # O corpo do programa propriamente dito começa aqui
50 clear
51 # A tela a seguir é montada com um único comando echo
52 echo "
                                        Inclusão de Músicas
                                        ======== == =======
                    Título do Álbum:
                    Faixa:
                    Nome da Música:
                    Intérprete:"
53 while true
54 do
55     tput cup 5 38; tput el      # Posiciona e limpa linha
56     read Album
57     [ ! "$Album" ] &&           # Operador deu <ENTER>
58         {
59         Pergunta "Deseja Terminar" s n
60         # Agora só testo caixa baixa
61         [ $SN = "n" ] && continue
62         clear; exit             # Fim da execução
63         }
64     grep -iq "^$Album\^" musicas 2> /dev/null &&
65         {
66         MandaMsg Este álbum já está cadastrado
67         continue                # Volta para ler outro álbum
68         }
69     Reg="$Album^"       # $Reg receberá os dados de gravação
70     oArtista=           # Guardará artista anterior
71     while true
72     do
73         ((Faixa++))
74         tput cup 7 38
75         echo $Faixa
76         tput cup 9 38           # Posiciona para ler música
77         read Musica
78         [ "$Musica" ] ||   # Se o operador teclou <ENTER>...
79             {
80             Pergunta "Fim de Álbum?" s n
81             # Agora só testo a caixa baixa
82             [ "$SN" = n ] && continue
83             break           # Sai do loop para gravar dados
84             }
85         tput cup 11 38          # Posiciona para ler Artista
86         # O artista anterior é o padrão
87         [ "$oArtista" ] && echo -n "($oArtista) "
88         read Artista
89         [ "$Artista" ] && oArtista="$Artista"
90         Reg="$Reg$oArtista~$Musica:" # Montando registro
91         tput cup 9 38; tput el       # Apaga Música da tela
92         tput cup 11 38; tput el      # Apaga Artista da tela
93     done
94     # Grava registro no fim do arquivo
95     echo "$Reg" >> musicas
96     # Classifica o arquivo
97     sort musicas -o musicas
98 done
```

Listagem 5: *musinc7.sh*

```
01 $ cat musinc7.sh
02 #!/bin/bash
03 # Cadastra CDs (versao 7)
04 #
05 # Área de variáveis globais
06 LinhaMesg=$((`tput lines` - 3))  # Linha onde serão mostradas as msgs para o operador
07 TotCols=$(tput cols)            # Qtd colunas da tela para enquadrar msgs
08 # O corpo do programa propriamente dito começa aqui
09 clear
10 echo "
                                  Inclusão de Músicas
                                  ======== == =======
                    Título do Álbum:
                                       | Este campo foi
                    Faixa:             < criado somente para
                                       | orientar o preenchimento
                    Nome da Música:
                    Intérprete:"  # Tela montada com um único echo
11 while true
12 do
13     tput cup 5 38; tput el               # Posiciona e limpa linha
14     read Album
15     [ ! "$Album" ] &&                    # Operador deu <ENTER>
16         {
17         source pergunta.func "Deseja Terminar" s n
18         [ $SN = "n" ] && continue        # Agora só testo a caixa baixa
19         clear; exit                      # Fim da execução
20         }
21     grep -iq "^$Album\^" musicas 2> /dev/null &&
22         {
23         . mandamsg.func Este álbum já está cadastrado
24         continue                         # Volta para ler outro álbum
25         }
26     Reg="$Album^"                        # $Reg receberá os dados de gravação
27        oArtista=                            # Guardará artista anterior
28     while true
29     do
30         ((Faixa++))
31         tput cup 7 38
32         echo $Faixa
33         tput cup 9 38                    # Posiciona para ler música
34         read Musica
35         [ "$Musica" ] ||                 # Se o operador tiver dado <ENTER>...
36             {
37             . pergunta.func "Fim de Álbum?" s n
38             [ "$SN" = n ] && continue    # Agora só testo a caixa baixa
39             break                        # Sai do loop para gravar dados
40             }
41         tput cup 11 38                           # Posiciona para ler Artista
42         [ "$oArtista" ] && echo -n "($oArtista) "  # Artista anterior é default
43         read Artista
44         [ "$Artista" ] && oArtista="$Artista"
45         Reg="$Reg$oArtista~$Musica:"             # Montando registro
46         tput cup 9 38; tput el                   # Apaga Música da tela
47         tput cup 11 38; tput el                  # Apaga Artista da tela
48     done
49     echo "$Reg" >> musicas                 # Grava registro no fim do arquivo
50     sort musicas -o musicas              # Classifica o arquivo
51 done
```

Ahh! Agora sim! Quando passado como parâmetro do comando *source*, o script foi executado no shell corrente, deixando nele todo o ambiente criado. Agora vamos rebobinar a fita até o início da explicação sobre este comando. Lá falamos do `.bash_profile` e, a esta altura, você já deve saber que sua incumbência é, logo após o login, preparar o ambiente de trabalho para o usuário. Agora entendemos que é por isso mesmo que ele é executado usando esse artifício.

E agora você deve estar se perguntando se é só para isso que esse comando serve. Eu lhe digo que sim, mas isso nos traz um monte de vantagens – e uma das mais usadas é tratar funções como rotinas externas. Veja na **listagem 5** uma outra forma de fazer o nosso programa para incluir CDs no arquivo musicas.

Agora o programa deu uma boa encolhida e as chamadas de função foram trocadas por arquivos externos chamados `pergunta.func` e `mandamsg.func`, que assim podem ser chamados por qualquer outro programa, dessa forma reutilizando o seu código.

Por motivos meramente didáticos, as chamadas a `pergunta.func` e `mandamsg.func` estão sendo feitas por *source* e por `.` (ponto) indiscriminadamente, embora eu prefira o *source* que, por ser mais visível, melhora a legibilidade do código e facilita sua posterior manutenção. Veja na listagem 6 como ficaram esses dois arquivos.

Em ambos os arquivos, fiz somente duas mudanças, que veremos nas observações a seguir. Porém, tenho mais três observações a fazer:

1. As variáveis não estão sendo mais declaradas como locais, porque essa é uma diretiva que só pode ser usada no corpo de funções e, portanto, essas variáveis permanecem no ambiente do shell, poluindo-o;
2. O comando *return* não está mais presente, mas poderia estar sem alterar em nada a lógica do script, uma vez que só serviria para indicar um eventual erro por meio de um código de retorno previamente estabelecido (por exemplo `return 1`, `return 2`, ...), sendo que o `return` e `return 0` são idênticos e significam que a rotina foi executada sem erros;

3. O comando que estamos acostumados a usar para gerar um código de retorno é o *exit*, mas a saída de uma rotina externa não pode ser feita dessa forma porque, como ela está sendo executada no mesmo shell do script que o chamou, o *exit* simplesmente encerraria esse shell, terminando a execução de todo o script;
4. De onde veio a variável `LinhaMesg`? Ela veio do script `musinc7.sh`, porque havia sido declarada antes da chamada das rotinas (nunca esqueça que o shell que está interpretando o script e essas rotinas é o mesmo);
5. Se você decidir usar rotinas externas não se envergonhe, exagere nos comentários, principalmente sobre a passagem dos parâmetros, para facilitar a manutenção e o uso dessa rotina por outros programas no futuro.

– Bem, agora você já tem mais um monte de novidades para melhorar os scripts que fizemos. Você se lembra do programa *listartista.sh* no qual você passava o nome de um artista como parâmetro e ele devolvia as suas músicas? Ele era como o mostrado aqui embaixo na **listagem 7**.

– Claro que me lembro!

– Para firmar os conceitos que te passei, faça-o com a tela formatada e a execução em *loop*, de forma que ele só termine quando receber um ***Enter*** no lugar do nome do artista. Suponha que a tela tenha 25 linhas; a cada 22 músicas listadas o programa deverá dar uma parada para que o operador possa lê-las. Eventuais mensagens de erro devem ser passadas usando a rotina `mandamsg.func` que acabamos de desenvolver. Chico, manda mais dois!! O meu é com pouca pressão...

Listagem 6: *pergunta.func* e *mandamsg.func*

```
01 $ cat pergunta.func
02 #  A função recebe 3 parâmetros na seguinte ordem:
03 #  $1 - Mensagem a ser dada na tela
04 #  $2 - Valor a ser aceito com resposta default
05 #  $3 - O outro valor aceito
06 #  Supondo que $1=Aceita?, $2=s e $3=n, a linha
07 #  abaixo colocaria em Msg o valor "Aceita? (S/n)"
08 Msg="$1 (`echo $2 | tr a-z A-Z`/`echo $3 | tr A-Z a-z`)"
09 TamMsg=${#Msg}
10 Col=$(((TotCols - TamMsg) / 2))    # Centraliza msg na linha
11 tput cup $LinhaMesg $Col
12 echo "$Msg"
13 tput cup $LinhaMesg $((Col + TamMsg + 1))
14 read -n1 SN
15 [ ! $SN ] && SN=$2                # Se vazia coloca default em SN
16 echo $SN | tr A-Z a-z             # A saída de SN será em minúscula
17 tput cup $LinhaMesg $Col; tput el  # Apaga msg da tela
18 $ cat mandamsg.func
19 # A função recebe somente um parâmetro
20 # com a mensagem que se deseja exibir;
21 # para não obrigar o programador a passar
22 # a msg entre aspas, usaremos $* (todos
23 # os parâmetro, lembra?) e não $1.
24 Msg="$*"
25 TamMsg=${#Msg}
26 Col=$(((TotCols - TamMsg) / 2))    # Centraliza msg na linha
27 tput cup $LinhaMesg $Col
28 echo "$Msg"
29 read -n1
30 tput cup $LinhaMesg $Col; tput el  # Apaga msg da tela
```

Não se esqueça: em caso de dúvida ou falta de companhia para um chope, é só mandar um e-mail para o endereço julio.neves@gmail.com que terei prazer em lhe ajudar. Vou aproveitar também para mandar minha propaganda: diga aos amigos que quem estiver a fim de fazer um curso porreta de programação em Shell deve mandar um e-mail para julio.neves@tecnohall.com.br para informar-se. Até mais! ■

Listagem 7: *listartista.sh*

```
01 $ cat listartista.sh
02 #!/bin/bash
03 # Dado um artista, mostra as suas musicas
04 # versao 2
05 if  [ $# -eq 0 ]
06 then
07     echo Voce deveria ter passado pelo menos um parametro
08     exit 1
09 fi
10 IFS="
11 :"
12 for ArtMus in $(cut -f2 -d^ musicas)
13 do
14     echo "$ArtMus" | grep -i "^$*~" > /dev/null && echo $ArtMus | cut -f2 -d~
15 done
```

Informações

[1] *Bash*, página oficial: http://www.gnu.org/software/bash/bash.html

[2] Manual de referência do *Bash*: http://www.gnu.org/software/bash/manual/bashref.html

Sobre o autor

Julio Cezar Neves é Analista de Suporte de Sistemas desde 1969 e trabalha com Unix desde 1980, quando participou do desenvolvimento do SOX, um sistema operacional similar ao Unix produzido pela Cobra Computadores. Pode ser contatado no e-mail julio.neves@gmail.com