

Introdução ao Shell Script no Linux

Veja neste artigo como criar códigos para automatização de tarefas rotineiras utilizando o interpretador de comandos bash em sistemas Unix-like. Serão dados os conceitos básicos e as sintaxes e explicações dos principais comandos.

Podemos utilizar a criação de arquivos de scripts para tornar mais simples as execuções de tarefas repetitivas no dia a dia. Muito tempo do programador é empregado em ações desse tipo, como abrir os mesmos programas todos os dias; esvaziar a lixeira e diretórios temporários para economizar espaço em disco; etc.

Um script nada mais é do que um **algoritmo** projetado para realizar uma determinada tarefa, utilizando os comandos específicos do bash e os executáveis do sistema operacional.

Lembre-se de executar os comandos como usuário comum e não como root, visto que, como root tudo será aceito e, dependendo do que você fizer, isto pode gerar danos ao sistema operacional. Uma maneira fácil de verificar é abrir o terminal e se o símbolo antes do cursor é o \$, você está como usuário comum, mas se é o #, você está como root. Para sair do modo root, digite `exit`

Criação do shell script

Em primeiro lugar precisaremos de um arquivo para escrever o nosso script. Podemos fazer isso via terminal ou via modo gráfico, sendo que, no último caso, basta apenas clicar com o botão direito do mouse em um diretório desejado e escolher criar novo arquivo de texto ou criar novo documento.

Para criar um arquivo via terminal, basta abrir o mesmo e digitar `vi exemplo1.sh`, também podemos digitar `touch exemplo1.sh`

O comando `vi` cria e abre um arquivo para leitura/escrita no terminal, enquanto o comando `touch` cria um arquivo, mas não o abre. Posteriormente é possível abri-lo com um editor de sua preferência.

Concedendo permissões ao arquivo

Para editar o arquivo, precisamos dar permissão de escrita a ele.

Para a primeira alternativa, em que o `vi` abriu direto o arquivo, precisamos pressionar `ESC` para editá-lo, assim, ao se fazer isso, o caractere `:` aparece, então digite `!chmod 777`

Para a segunda alternativa, em que o `touch` não abriu o arquivo criado, basta digitar:
`chmod 777 exemplo1.sh`.

O `chmod` é utilizado para setar permissões em arquivos e diretórios. O valor `777` concede todos os direitos (read, write, execute) para o usuário, o grupo e os outros. Ao invés de `777`, outro modo de fazer isso é digitando `+rwx`.

O caractere `!` força o `vi` a executar o que está sendo pedido (no caso, executar o `chmod`).

O caractere % faz referência ao arquivo atual. Pode-se também, ao invés de utilizá-lo, fornecer o nome do arquivo.

Edição e execução do arquivo

Neste artigo utilizaremos o vi, no terminal, mas você pode escolher qualquer outro editor, gráfico ou não.

Abra o arquivo com o comando vi exemplo1.sh, com o vi você precisa digitar i para colocá-lo no modo de inserção.

Um shell script começa (mas não necessariamente) definindo qual o interpretador de comandos que será utilizado para interpretar e executar o script. Existem outros além do bash, como o sh, o ksh e o csh.

```
#!/bin/SHELL_ATUAL
```

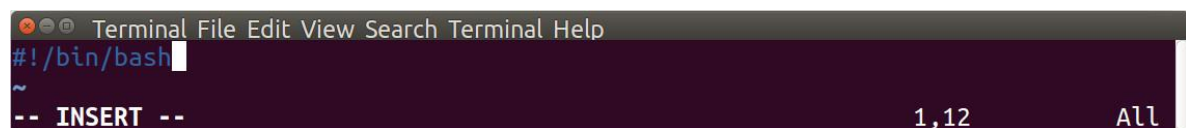


Figura 1. Execução do código bash

Como vemos na **Figura 1**, a primeira linha de um shell script define qual o interpretador de comandos será utilizado.

Note que utilizamos o path completo de onde se encontra o shell, no caso, no diretório /bin/.

Após isso, é hora de iniciarmos o nosso script. Para este artigo, o exemplo imprimirá na tela algumas informações sobre o usuário e o computador, conforme o código da **Listagem 1**.

```
#!/bin/bash
echo "Seu nome de usuário é:"
whoami
echo "Info de hora atual e tempo que o computador está ligado:"
uptime
echo "O script está executando do diretório:"
pwd
```

Listagem 1. Código do exemplo1.sh

Este código nos fornece algumas informações sobre o usuário, algumas informações da máquina e sobre o local de armazenamento do nosso script.

Para salvarmos o arquivo digitamos ESC e depois ":wq" para gravar as alterações e sair.

Para executar o script, há dois pontos a considerar:

1. Se você tiver salvo o seu arquivo no diretório atual, basta executá-lo digitando no prompt:
./exemplo1.sh
 2. Se você tiver salvo o seu arquivo em outro diretório qualquer, você precisará informar o path completo até ele. Considerando que ele esteja em /tmp/scripts: /tmp/scripts/exemplo1.sh
- O comando echo exibe na tela a string entre aspas duplas. Caso você não queira que ela fique entre aspas duplas, simplesmente não as forneça no comando echo;
 - O comando whoami exibe o usuário logado no sistema;
 - O comando uptime exibe a hora atual, o tempo decorrido desde que o computador foi ligado, o número de usuários logados e uma média do número de processos carregados nos últimos um, cinco e 15 minutos;

- O comando `pwd` exibe o diretório no qual o arquivo está rodando.

Toda string que contiver espaços deve estar entre aspas duplas.

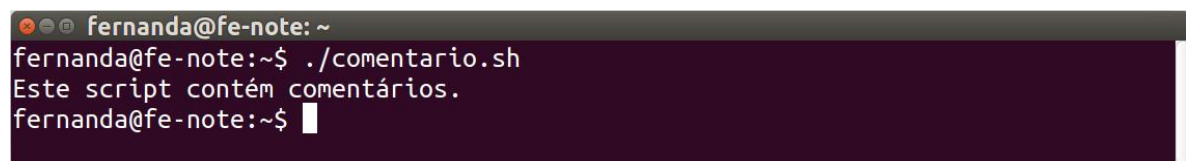
Inserindo comentários

Comentários em códigos são muito importantes. Explicar o que está sendo feito e dar informações sobre o funcionamento do código permite ao desenvolvedor economizar tempo para entendê-lo.

Para inserir comentários no seu script, basta iniciar a linha com o caractere `#`, como no código da **Listagem 2** e execução na **Figura 2**.

```
#!/bin/bash#Este é um comentário#Este é outro comentário echo "Este script contém comentários."
```

Listagem 2. Shell script com comentários



```
fernanda@fe-note: ~  
fernanda@fe-note:~$ ./comentario.sh  
Este script contém comentários.  
fernanda@fe-note:~$
```

Figura 2. Saída da execução do código da **Listagem 2**

Como pode ser observado, os comentários não são processados, portanto, não aparecem para o usuário.

Declarando e utilizando variáveis

Uma variável é um label (nome) que armazena um valor para ser utilizado posteriormente no código. Seu uso evita repetições de valores por parte do programador e torna o código mais informativo e limpo.

A linguagem do shell não é tipada, ou seja, pode-se armazenar qualquer tipo de valor em uma variável, desde strings a números.

Para declará-las basta seguir a sintaxe `nome_da_variavel=valor` onde:

- `nome_da_variavel`: sequência de caracteres que deve começar por qualquer letra maiúscula ou minúscula ou underscore (`_`);
- `valor`: qualquer dado que deva ser valorado à variável.

Nota: Observe que não deve haver espaços entre o sinal de igual e o nome e o valor da variável.

Para utilizarmos o valor da variável coloca-se o `$` (cifrão) na frente de seu nome, como mostra o exemplo da **Listagem 3**.

```
#!/bin/bash  
site=www.devmedia.com.br  
meu_numero_favorito=13  
_cidade="Porto Alegre"  
echo "Um ótimo site para você aprender a programar e se manter atualizado é: $site"  
echo "Meu número favorito é: $meu_numero_favorito"  
echo "Minha cidade natal é: $_cidade"
```

Listagem 3. Declarando e utilizando variáveis

Agora, se você deseja printar o nome da própria variável, basta colocar uma barra invertida \ antes do \$, assim, ela nega o caractere seguinte e normalmente é utilizada para caracteres de scape (ou seja, caracteres que já tem uma função específica, mas que você deseja somente utilizar em um nome ou valor, sem executá-los). Vejamos um exemplo na **Listagem 4**.

```
#!/bin/bash
nome=fernanda
echo "O nome da variável é \$nome"
```

Listagem 4. Printando o nome de uma variável ao invés de seu conteúdo

Atribuindo saídas de comandos a variáveis

É possível armazenar o resultado de um comando em uma variável. Isso é muito útil em situações em que se usará este resultado em mais de um lugar ao longo do script.

Há duas sintaxes para isso:

1. nome_da_variavel=\$(comando)
2. nome_da_variavel=comando

Você pode escolher a que melhor lhe agrada ou empregar as duas nos seus scripts.

O próximo exemplo lista as informações relativas a todos os discos e partições do sistema:

```
#!/bin/bash
system_info=`df -h` # Também poderia ser system_info=$(df -h)
echo "$system_info"
```

Veja que o comando df-h executará e a sua saída (resultado dessa execução) será armazenada na variável system_info.

Capturando a entrada de dados do usuário

Pode ser que o seu script precise interagir com o usuário, pedindo para ele fornecer algum dado de entrada para processamento. Neste caso, é necessário que se leia o que o usuário digitou e isso é feito através do seguinte comando readnome_da_variavel_para_armazenar_o_valor_a_ser_lido

Vejamos o exemplo da **Listagem 5**.

```
#!/bin/bash
echo "Qual o nome de uma de suas músicas favoritas?"
read nome_musica;
echo "Você gosta de ouvir $nome_musica!"
```

Listagem 5. Utilizando o comando read para ler entrada do usuário

Comandos de seleção ou de tomada de decisão

Na maioria das vezes precisamos seguir um determinado fluxo de execução baseado em alguma decisão tomada pelo usuário ou outro sistema que esteja utilizando o nosso. O comando mais simples que permite isso é o condicional, que tem a seguinte sintaxe:

```
if [ CONDICAO ];
then
    AÇÕES
fi
```

Listagem 1. NOME

Onde:

- **CONDICAO:** teste que, se verdadeiro, passará o controle para o bloco dentro do then;
- **AÇÕES:** comandos a serem executados se o resultado de CONDICAO for verdadeiro.

Nota: É muito comum o desenvolvedor esquecer de fechar o if. Lembre-se sempre que, para cada if aberto, você precisa fechá-lo com o fi.

Nota: Lembre-se que, se utilizar o `[`, você deve fechá-lo com o `]`, e deixando sempre espaços ao redor. Isso é muito importante, pois eles são um “atalho” para o comando 'test'. Isso significa que, alternativamente, você poderia querer não utilizar os colchetes:

```
if test CONDICAO;  
then  
    AÇÕES  
fi
```

Mas normalmente se utiliza os colchetes por ser mais compacto e para ficar mais semelhante ao formato em outras linguagens. De qualquer forma, a escolha é sua.

Nota: Em outras linguagens de programação o **if** testa uma condição, mas em shell script o **if** testa a saída de um comando.

Vamos a um exemplo em que o usuário deverá digitar um número e verificaremos se ele está em um determinado intervalo, como mostra a **Listagem 6**.

```
#!/bin/bash  
echo "Digite um número qualquer:"  
read numero;  
if [ "$numero" -gt 20 ];  
then  
    echo "Este número é maior que 20!"  
fi
```

Listagem 6. Utilizando o condicional if...then

Veja a seguir os parâmetros mais comuns utilizados com o comando test:

- **n** string1: o comprimento de string1 é diferente de 0;
- **z** string1: o comprimento de string1 é zero;
string1 = string2: string1 e string2 são idênticas;
string1 != string2: string1 e string2 são diferentes;
inteiro1 **-eq** inteiro2: inteiro1 possui o mesmo valor que inteiro2;
inteiro1 **-ne** inteiro2: inteiro1 não possui o mesmo valor que inteiro2;
inteiro1 **-gt** inteiro2: inteiro1 é maior que inteiro2;
inteiro1 **-ge** inteiro2: inteiro1 é maior ou igual a inteiro2;
inteiro1 **-lt** inteiro2: inteiro1 é menor que inteiro2;
inteiro1 **-le** inteiro2: inteiro1 é menor ou igual a inteiro2;
- **e** nome_do_arquivo: verifica se nome_do_arquivo existe;
- **d** nome_do_arquivo: verifica se nome_do_arquivo é um diretório;
- **f** nome_do_arquivo: verifica se nome_do_arquivo é um arquivo regular (texto, imagem, programa, docs, planilhas).

O comando else

Existe a possibilidade de também tratar o caso em que o nosso teste falha. Para isso temos o comando `else`, cuja sintaxe é:

```
if [ CONDICAÇÃO ];
then
  AÇÕES_1
else
  AÇÕES_2
fi
```

Onde:

- **CONDICAÇÃO** : teste que, se verdadeiro, passará o controle para o bloco dentro do `then`;
- **AÇÕES_1** : comandos a serem executados se o resultado de **CONDICAÇÃO** for verdadeiro;
- **AÇÕES_2** : comandos a serem executados se o resultado de **CONDICAÇÃO** for falso.

Vejamos um exemplo na **Listagem 7** que verifica se um número digitado pelo usuário é positivo ou negativo.

```
#!/bin/bash
echo "Digite um número qualquer:"
read numero;
if [ "$numero" -ge 0 ];
then
    echo "O número $numero é positivo!"
else
    echo "O número $numero é negativo!"
fi
```

Listagem 7. Utilizando o condicional `if...then...else`

O comando `elif`

Há casos em que temos mais de uma condição a ser testada, todas correlacionadas. Para isso temos o comando `elif`, cuja sintaxe é:

```
if [ CONDIÇÃO_1 ];
then
    AÇÕES_1
elif [ CONDIÇÃO_2 ];
then
    AÇÕES_2
elif [ CONDIÇÃO_3 ];
then
    AÇÕES_3
elif [ CONDIÇÃO_N ];
then
    AÇÕES_N
fi
```

Onde:

- **CONDICAÇÃO_1 ... CONDICAÇÃO_N** : teste que, se verdadeiro, passará o controle para o bloco dentro do respectivo `then`;
- **AÇÕES_1 ... AÇÕES_N** : comandos a serem executados se os resultados de **CONDICAÇÃO_1 ... CONDICAÇÃO_N** forem verdadeiros.

A seguir temos um exemplo que apresenta um menu para o usuário escolher uma opção. Baseado nesta escolha, a hora e a data serão exibidas; uma divisão será efetuada e seu resultado será exibido, e uma mensagem será exibida com o nome que o usuário fornecer, como mostra a **Listagem 8**.

```
#!/bin/bash
echo "Selecione uma opção:"
echo "1 - Exibir data e hora do sistema"
echo "2 - Exibir o resultado da divisão 10/2"
echo "3 - Exibir uma mensagem"
read opcao;
if [ $opcao == "1" ];
then
    data=$(date +%T, %d/%m/%y, %A")
    echo "$data"
elif [ $opcao == "2" ];
then result=$((10/2))
    echo "divisao de 10/2 = $result"
elif [ $opcao == "3" ];
then
    echo "Informe o seu nome:"
    read nome;
    echo "Bem-vindo ao mundo do shell script, $nome!"
fi
```

Listagem 8. Utilizando o comando `elif`

Nota: O `bash` não tem suporte nativo a divisões em ponto flutuante, apenas divisões inteiras. Caso queira efetuar este tipo de operação, precisará de um comando externo, como `dc` ou `bc`

Nota: Observe a linha: `result=$((10/2))`

Veja que utilizamos dois conjuntos de parênteses para encapsular a operação de divisão. Em shell script precisamos realizar operações matemáticas entre parênteses.

O comando `case`

O comando `case` tem a mesma funcionalidade do `if...then...elif`, com a diferença de sua sintaxe ser mais compacta e enxuta:

```
case VARIÁVEL in
    CASO_1)
        AÇÕES_1
        ;;
    CASO_2)
        AÇÕES_2
        ;;
    CASO_N)
        AÇÕES_N
        ;;
esac
```

Onde:

- `VARIÁVEL`: variável que terá seu valor verificado;
- `CASO_1 ... CASO_N`: possíveis estados da variável;
- `AÇÕES_1 ... AÇÕES_N`: ações a serem tomadas caso a variável combine com `CASO_1 ... CASO_N`, respectivamente.

Por exemplo, modificando o exemplo anterior temos o código da **Listagem 9**.

```
#!/bin/bash
echo "Selecione uma opção:"
echo "1 - Exibir data e hora do sistema"
echo "2 - Exibir o resultado da divisão 10/2"
echo "3 - Exibir uma mensagem"
read opcao;
case $opcao in
    "1")
        data=$(date +%T, %d/%m/%y, %A")
        echo "$data"
        ;;
    "2")
        result=$((10/2))
        echo "divisao de 10/2 = $result"
        ;;
    "3")
        echo "Informe o seu nome:"
        read nome;
        echo "Bem-vindo ao mundo do shell script, $nome!"
        ;;
esac
```

Listagem 9. Utilizando o comando case

LOOPS Condicionais

Loops são muito úteis para ficar iterando sobre determinadas ações até que uma condição seja satisfeita e interrompa o laço.

O primeiro deles é o **for**, cuja sintaxe é:

```
for VARIABEL in VALOR_1, VALOR_2 ... VALOR_N;
do
    AÇÕES
done
```

Onde:

1. **VARIABEL**: variável cujo valor será inicializado e incrementado, respeitando os limites dos valores do conjunto fornecido;
2. **VALOR_1**, **VALOR_2** ... **VALOR_N**: valores que **VARIABEL** poderá assumir durante o loop;
3. **AÇÕES**: ações a serem tomadas repetidamente até que o valor de **VARIABEL** ultrapasse o último valor informado no conjunto de valores fornecido.

Nota: A sequência **VALOR_1**, **VALOR_2** ... **VALOR_N**; na sintaxe pode ser substituída por: **{VALOR_1..VALOR_N};**

Observe que são apenas duas reticências.

Quando o loop for começa, a variável é inicializada com o primeiro valor do conjunto, e ocorre a primeira iteração (entrada no laço e execução dos comandos). Para as iterações seguintes, os valores do conjunto serão atribuídos à variável, sucessivamente, até que se alcance o último e o loop termine a execução.

Veja o exemplo da **Listagem 10**, que conta decrescendo de 10 a 0.


```
#!/bin/bash
echo "Testando o loop
for" for i in {10..0};
do
    echo "$i"
done
```

Listagem 10. Exemplo de uso do loop for

Outra forma de criarmos sequências de valores é com o comando seq, como mostra a **Listagem 11**.

```
#!/bin/bash
echo "Testando o comando seq"
for i in $(seq 1 5 100);
do
    echo "$i"
done
```

Listagem 11. Exemplo de uso do loop for com o comando seq com intervalo

Observe que foi criada uma sequência de 1 até 100, com intervalo de 5.

Agora, na **Listagem 12** temos um exemplo de loop sem intervalo.

```
#!/bin/bash
echo "Testando o comando seq"
for i in $(seq 1 100);
do
    echo "$i"
done
```

Listagem 12. Exemplo de uso do loop for com o comando seq sem intervalo

Observe que foi criada uma sequência de 1 até 100, de 1 em 1.

Loop while

Enquanto o loop for é mais ideal para quando sabemos até quanto contar, o loop while é bom para quando não temos essa noção, mas sabemos de uma condição que deverá ser atendida para que o laço termine. Sua sintaxe é:

```
while [ CONDICAO ];
do
    AÇÕES
done
```

Onde:

- **CONDICAO** : condição cuja veracidade determina a permanência no laço;
- **AÇÕES** : ações a serem tomadas enquanto **CONDICAO** for verdadeira.

Vamos na **Listagem 13** um exemplo que exibe ao usuário o que ele digitou, enquanto ele não informar -1.

```
#!/bin/bash
echo "Informe o que você quiser, -1 para sair"
read dado;
while [ $dado != "-1" ];
do
    echo "Você digitou $dado"
    read dado;
done
```

Listagem 13. Exemplo de uso do loop while

Vejamos um outro exemplo com contador na **Listagem 14**.

```
#!/bin/bash
echo "Informe até que valor positivo e maior que zero contar:"
read valor;
i=1
while [ $i -le $valor ];
do
    echo "$i"
    ((i=$i+1))
done
```

Listagem 14. Exemplo de uso do loop while com contador

Funções

O uso de funções é imprescindível para separar, organizar e estruturar a lógica de qualquer algoritmo, seja em shell script ou qualquer outra linguagem de programação. Sua sintaxe é muito simples:

```
nome_funcao()
{
    AÇÕES
}
```

Funções podem chamar outras funções existentes no script, simplesmente escrevendo-se o nome dela, como vemos no exemplo da **Listagem 15**.

```
#!/bin/bash

main() {
    echo "Escolha uma opção:"
    echo "1 - Esvaziar a lixeira"
    echo "2 - Calcular fatorial"
    read opcao;
    case $opcao in
        "1")
            esvaziar_lixeira
            ;;
        "2")
            calcular_fatorial
            ;;
        esac
    }

    esvaziar_lixeira(){
        echo "Esvaziando a lixeira..."
    }
```

```

path="${HOME}/.local/share/Trash/files"
cd "$path"
for file in *
do
rm -rf "$file"
done
echo "Done!"
}

calcular_fatorial(){
echo "Informe um número:"
read numero;
i=1
fat=1
while [ $i -le $numero ]
do
fat=$((fat*i))
i=$((i+1))
done
echo "fatorial de $numero é $fat"
}

main

```

Listagem 15. Exemplo de uso de funções

Nota: Lembre-se sempre de chamar a função principal (no nosso caso, main) no final do seu script, do contrário, nada acontecerá quando você o executar.

Argumentos

Normalmente um programa recebe argumentos como entrada, ou seja, dados fornecidos pelo usuário ou por outro programa, os quais devem ser “consumidos” para produzir as saídas desejadas.

Em shell script não poderia ser diferente: temos nomes especiais para designar os argumentos recebidos por um script:

- `$0` – contém o nome do script que foi executado;
- `$1 ... $n` – contêm os argumentos na ordem em que foram passados (1º argumento em `$1`, 2º argumento em `$2`, etc.).
- `$#` - contém o número de argumentos que foi passado (ou seja, não considera o nome do script em `$0`);
- `$*` - retorna todos os argumentos de uma vez só.

Vamos ao exemplo da **Listagem 16** e sua execução é exibida na **Figura 3**.

```

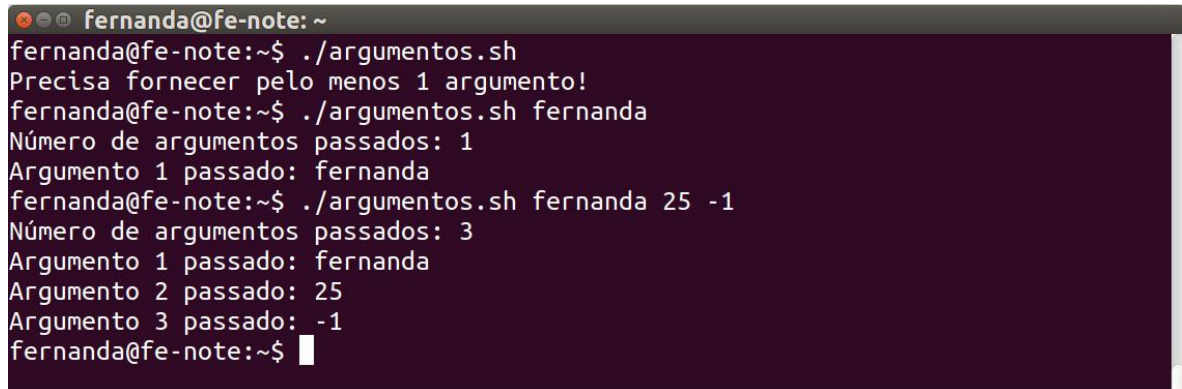
#!/bin/bash

if [ $# -lt 1 ];
then
echo "Precisa fornecer pelo menos 1 argumento!"
exit 1
fi

echo "Número de argumentos passados: $#"
```

```
for argumento in $*
do i=$((i+1))
    echo "Argumento $i passado: $argumento"
done
```

Listagem 16. Exemplo de uso de argumentos em scripts



```
fernanda@fe-note: ~
fernanda@fe-note:~$ ./argumentos.sh
Precisa fornecer pelo menos 1 argumento!
fernanda@fe-note:~$ ./argumentos.sh fernanda
Número de argumentos passados: 1
Argumento 1 passado: fernanda
fernanda@fe-note:~$ ./argumentos.sh fernanda 25 -1
Número de argumentos passados: 3
Argumento 1 passado: fernanda
Argumento 2 passado: 25
Argumento 3 passado: -1
fernanda@fe-note:~$
```

Figura 3. Saída da execução do código da **Listagem 16**

Conforme vimos no artigo, shell scripts são a melhor maneira de automatizar tarefas diárias em sistemas Unix-like. Além de práticos, nos poupam muito tempo, além de possuírem uma sintaxe simples e permitir processar desde pequenas quantidades de dados até executar tarefas mais robustas.