Universidade do Estado de Santa Catarina – Bacharelado em Ciência da Computação
TE em Ciência da Computação – Prof. Rafael R. Obelheiro

Exercícios – Objetivo 105.2

1. Qual o efeito de usar

   ```
   #! /usr/bin/tac
   ```

   como primeira linha de um script?

2. Crie um arquivo `hello.py` com o seguinte conteúdo:

   ```
   #! /usr/bin/env python3

   nome = input("Qual o seu nome? ")
   print(f"Ola', {nome}!")
   ```

   (a) Qual o interpretador de comandos usado por esse script?
   (b) Ajuste as permissões do arquivo para que o script possa ser executado com `./hello.py`.
   (c) Existe alguma diferença entre executar o script com `./hello.py` e com `bash hello.py`?

3. Faça um script shell que liste todos os argumentos recebidos na linha de comando, em ordem. Exemplo de uso:

   ```
   $ ./lista-args badejo baiacu borriquete
   1: badejo
   2: baiacu
   3: borriquete
   ```

   Teste o script passando, entre os argumentos, um que contenha espaço (como `congro rosa`).

4. Faça um script shell que execute o script do exercício 3 quatro vezes, usando como argumentos:

   (a) `$*`
   (b) `$@`
   (c) `"$*"`
   (d) `"$@"`

   Teste este script passando, entre os argumentos, um ou mais contendo espaços.

5. Como você pode atribuir a lista de argumentos de um script a um array, em que cada posição do array corresponde a um argumento?

6. Faça um script shell que imprima se o PID do processo (correspondente ao próprio script) é par ou ímpar.

7. O usuário `bob` pertence apenas aos grupos `bob` e `staff`. Os arquivos `scr?` são scripts shell, que estão em um diretório no `PATH`. De acordo com a saída do comando `ls -l` abaixo, quais scripts podem ser executados por `bob` invocando apenas o seu nome? E quais podem ser executados usando `sh` *nome*?

   ```
   -rw-rw-r-- 1 bob  bob    23 Aug 13 12:41 scr1
   -rwxr--r-- 1 bob  bob    23 Aug 13 12:41 scr2
   --wx-w---- 1 bob  bob    23 Aug 13 12:41 scr3
   -rwxr-xr-x 1 root staff  23 Aug 13 12:41 scr4
   -rwxr----- 1 root staff  23 Aug 13 12:41 scr5
   ```

8. Um programa C chamado `prog.c` geralmente é compilado em um executável chamado `prog` usando a seguinte linha de comando:

```
$ cc -o prog prog.c
```

Escreva um script shell chamado `comp` para compilar um programa C recebido como argumento na linha de comando, i.e., invocar `comp prog.c` seria equivalente à linha de comando acima. Seu script deverá produzir uma mensagem de erro, sem invocar o compilador, quando nenhum argumento for fornecido.

DICA: o comando `basename` pode fornecer o nome de um arquivo sem uma determinada extensão.

9. O compilador C aceita várias flags; exemplos comuns incluem `-pthread`, que indica que o código usa Pthreads, `-Wall`, que ativa a exibição de vários avisos (*warnings*), e `-g`, que gera um executável com símbolos de depuração. Modifique o script do exercício anterior para que o primeiro argumento seja interpretado como um programa C, e os demais, caso existam, sejam repassados para o compilador.

10. Alice escreveu um script shell para realizar um teste simples de soma:

```
#! /bin/bash

# Sorteia dois numeros entre 00 e 99
n1=$((RANDOM % 100))
n2=$((RANDOM % 100))

read -p "Quanto e' $n1 + $n2? " resp
if [ "$resp" == $n1+$n2 ]; then
    echo "Certo!"
else
    echo "Errado..."
fi
```

Dois usuários, Bob e Mallory, testaram o script. Bob foi perguntado "`Quanto e' 20 + 31?`" e forneceu a resposta correta (`51`), mas o script disse que ela estava errada. Mallory foi perguntado "`Quanto e' 14 + 18?`" e forneceu uma resposta incorreta, mas o script disse que ela estava certa.

(a) Qual foi a resposta informada por Mallory?

(b) Corrija o(s) erro(s) no script de Alice.

(c) Modifique o script para que ele produza códigos de retorno sinalizando se o usuário acertou a resposta ou não.

11. Bob escreveu um script shell que continha o seguinte trecho:

```
echo -n "Executando 'cmd'..."
cmd
echo "feito."
if [ $? -eq 0 ]; then
    echo "'cmd' executado com sucesso."
else
    echo "'cmd' executado com ERRO."
fi
```

No entanto, a mensagem indicando o erro nunca é exibida, mesmo com `cmd` retornando um valor diferente de zero para sinalizar o erro. Qual o problema?

12. O que acontece quando a sequência de comandos abaixo é executada? Você consegue identificar a inconsistência?

```
if [ 1 > 2 ]; then
    echo "maior"
else
    echo "menor ou igual"
fi
```

13. Reescreva o trecho do exercício anterior em uma única linha, usando `||` e `&&`. Não se esqueça de remover a inconsistência.

14. O que você espera que seja mostrado pelos comandos abaixo?

```
$ [ -e / ] && ls /a || echo "/ nao existe"    ## 1

$ [ -e /a ] || echo "/a nao existe" && ls /a  ## 2
```

    Execute os comandos e verifique se eles produzem os resultados esperados. Caso algum resultado não seja o esperado, tente explicar o porquê.

15. Adapte a função `show_usage()` mostrada nos slides para que aceite dois parâmetros opcionais, o código (numérico) de retorno e uma mensagem específica de erro. Se não for passado nenhum argumento, a função comporta-se como no original. Se for passado apenas um argumento, ele será usado como código de retorno. Se forem passados dois argumentos, o primeiro será o código de retorno e o segundo a mensagem específica, que deve ser exibida antes da mensagem genérica (que mostra como invocar o programa). Exemplos de uso:

```
show_usage            # msg padrao
show_usage 2          # msg padrao + codigo de retorno 2
show_usage 3 "erro!"  # "erro!" + msg padrao + cod retorno 3
```

16. O script abaixo requer um parâmetro, e sinaliza um erro caso esse parâmetro não seja passado corretamente. No entanto, ele contém nada menos do que três problemas. Quais são eles? (Considere não apenas erros que impeçam a execução do script, mas também características que fujam das boas práticas para construção de scripts.)

```
#! /bin/bash

if [ $# -eq 0 ]; then
    echo "ERRO! Parametro nao informado!"
    exit 0
else
    cmd $1
    exit 0
fi
```

17. Após corrigir os problemas do script do exercício 16, complemente-o, verificando se o arquivo passado como parâmetro é um arquivo executável antes de invocar `cmd`. Caso não seja, o script deverá exibir uma mensagem de erro e encerrar sem executar `cmd`.

18. Uma operação frequente no shell é criar um diretório `X` e imediatamente mudar o diretório corrente para `X`. Crie um script, alias ou função `mkcd` que junte esses dois passos. Considere que:

    i. `mkcd X` equivale à sequência

    ```
    mkdir X
    cd X
    ```

    quando `X` não existe;

    ii. `mkcd X` equivale a `cd X` quando `X` já existe;

    iii. se houver erro na criação do diretório, o diretório corrente não deve ser alterado;

    iv. `mkcd` deve sinalizar erro caso não receba o nome do diretório como parâmetro.

    DICA: você pode usar `test -d` para testar a existência de um diretório.

19. Mostre como as condições abaixo poderiam ser testadas no shell usando `test`:

    (a) O usuário tem permissão para ler o arquivo `/etc/hosts`.

    (b) O usuário tem permissão de leitura e execução no diretório `/etc`.

    (c) O conteúdo da variável `$teste` é igual à string "`sucesso`".

    (d) O conteúdo da variável `$teste` é numericamente igual ao conteúdo da variável `$result`.

    (e) O conteúdo da variável `$teste` é igual à string "`sucesso`" e o arquivo `/etc/hosts` existe.

    (f) O conteúdo da variável `$teste` é igual à string "`sucesso`", ou ao número 5, ou ao conteúdo da variável `$result`.

20. O que acontece se você omitir os caracteres "`;;`" em um comando `case`? Isso funciona como na construção `switch`...`case` da linguagem C, em que `break` marca o final de uma opção e a sua omissão faz com que as opções seguintes também sejam testadas?

21. Faça um script shell que liste todos os argumentos recebidos na linha de comando, em ordem **inversa**. Exemplo de uso:

    ```
    $ ./lista-args-inv badejo baiacu borriquete
    1: borriquete
    2: baiacu
    3: badejo
    ```

    Teste o script passando, entre os argumentos, um que contenha espaço (como `congro rosa`).
    ATENÇÃO: note que processar a saída do script do exercício 3 com `tac` **NÃO** produz a resposta correta.

22. O sistema de inicialização (*init system*) é o primeiro processo lançado em um sistema Linux após o boot. Sua principal responsabilidade é completar a inicialização do sistema: definir data/hora, iniciar serviços, montar sistemas de arquivos, configurar rede, entre outras tarefas. Existem diferentes sistemas de inicialização que podem ser usados no Linux; os mais conhecidos são SysV init, systemd e OpenRC, e cada distribuição Linux escolhe qual deles usar. A tabela abaixo lista os sistemas de inicialização usado por algumas distribuições conhecidas:

distribuição	sistema de inicialização	distribuição	sistema de inicialização
Alpine	OpenRC	Mint	systemd
Debian	systemd	OpenSUSE	systemd
Fedora	systemd	PCLinuxOS	SysV init
Gentoo	OpenRC	Slackware	SysV init
MX	SysV init	Ubuntu	systemd

Faça um script shell que pergunte ao usuário qual a distribuição Linux, e imprima o sistema de inicialização usado por ela (com base na tabela acima). Caso a distribuição não esteja na tabela acima, informe que o sistema de inicialização é desconhecido. Não faça distinção entre letras maiúsculas e minúsculas. Sua solução deve usar o comando `case` do shell.

23. Embora o material da certificação LPIC-1[1] sugira que o comando `case` casa uma variável com padrões fixos, é possível usar padrões com metacaracteres (*globbing*). Por exemplo, a função abaixo determina se o seu argumento é um número inteiro sem sinal válido:

```
# is_uint() - https://stackoverflow.com/a/61835747
is_uint() {
    case $1 in
        '' | *[!0-9]*) return 1;;
    esac
}
```

Um exemplo de uso da função:

```
if [ is_uint "$1" ] ; then
    echo "argumento e' um numero valido"
else
    echo "argumento nao e' um numero valido"
fi
```

(a) Explique a lógica usada pela função `is_uint`.

(b) Usando a função `is_uint` como modelo, escreva duas funções `iseven` e `isodd` que retornam verdadeiro quando um número recebido é par ou ímpar, respectivamente. Você pode supor que a função recebe como argumento um número válido, ou usar `is_uint` para validar o argumento recebido.

[1] `https://learning.lpi.org/pt/learning-materials/102-500/105/105.2/105.2_02/`