Curso:	Redes de Computadores / Noite		
Disciplina:	Desenvolvimento de Scripts II (5º Período)	Valor	
Professor (a):	João Paulo C. Aramuni		
Nome:		Nota	
Nº da Atividade/Nome:	Revisão		
Data:	19/03/2018		
Valor:			

1) Considere as seguintes definições e marque a alternativa INCORRETA:

A) Scripts shell são estruturas sofisticadas de programação. São arquivos que possuem dentro uma lista de comandos que você deseja executar.
B) Cada script está escrito dentro de um arquivo, e cada arquivo possui um nome diferente. Isso permite que exista uma combinação entre eles para que novos programas sejam gerados.
C) Scripts possuem como objetivos: Resolver um problema; Ajudar o administrador a tomar decisões e Criar funcionalidades.
D) O Shell é um interpretador de comandos que executa tarefas lidas da entrada padrão ou de um determinado arquivo e faz parte do Kernel Linux.
E) O Shell é um interpretador de comandos que possui características de uma linguagem de programação. Por isso é uma ferramenta poderosa para desenvolver scripts e programas rápidos para automatizar tarefas do dia-a-dia.

2) Considere as seguintes afirmativas sobre o Shell e marque a alternativa INCORRETA:

A) O Shell é um programa que recebe, interpreta, compila e executa os comandos de usuário, aparecendo na tela como uma linha de comandos.
B) O Shell padrão do UNIX, também chamado de Bourne Shell, foi desenvolvido por Stephen Bourne em meados dos anos 70.
C) A principal funcionalidade do Shell é ler os comandos que você digita, para depois passá-los ao Kernel para serem executados.
D) O Shell foi desenvolvido para executar programas no UNIX e possui ambiente de fácil utilização.
E) O Shell é considerado um dos mais poderosos programas padrão do UNIX.

3) O comentário #!/bin/bash que fica no topo de nossos scripts é um exemplo de comentário chamado de “shebang” (#!). Ele indica qual interpretador de comandos utilizar ao executar o script. Para que o script seja interpretado e executado pelo shell bash, utilizamos o comentário #!/bin/bash. São exemplos de comentários “shebang” válidos EXCETO:

A) #!/bin/csh — Executar o arquivo usando csh, o C shell, ou um shell compatível
B) #!/usr/bin/perl -T — Executar o arquivo usando Perl e option for taint checks
C) #!/usr/bin/java — Executar o arquivo usando o interpretador de linha de comando do Java
D) #!/usr/bin/python -O — Executar o arquivo usando Python com otimizações no código
E) #!/usr/bin/ruby — Executar o arquivo usando Ruby

4) Existem diversos tipos de loops diferentes presentes na linguagem. O shell nos permite trabalhar com estes loops de várias maneiras distintas. Analise o código abaixo e responda ao que se pede:

```bash
#!/bin/bash

variavel=0
while [ $variavel -lt 10 ]
do
    echo "MENSAGEM I"
    variavel=`expr $variavel + 1`
done

variavel2=20
until [ $variavel2 -lt 10 ]
do
    echo "MENSAGEM II"
    variavel2=`expr $variavel2 - 1`
done
```

Considerando o trecho de código acima, avalie as afirmações a seguir.

I. Suponha-se que a MENSAGEM I seja "$variavel eh menor do que 10."

O loop WHILE irá imprimir:
0 é menor do que 10.
1 é menor do que 10.
2 é menor do que 10.
3 é menor do que 10.
4 é menor do que 10.
5 é menor do que 10.
6 é menor do que 10.
7 é menor do que 10.
8 é menor do que 10.
9 é menor do que 10.

II. Suponha-se que a MENSAGEM II seja "$variavel2 nao eh menor do que 10."

O loop UNTIL irá imprimir:
20 não é menor do que 10.
19 não é menor do que 10.
18 não é menor do que 10.
17 não é menor do que 10.
16 não é menor do que 10.
15 não é menor do que 10.
14 não é menor do que 10.
13 não é menor do que 10.
12 não é menor do que 10.
11 não é menor do que 10.
10 não é menor do que 10.

III. A MENSAGEM I pode ser substituída por “$variavel eh menor do que 10. ”, e a MENSAGEM II pode ser substituída por “$variavel2 nao eh menor do que 10. ”.

IV. A válvula de escape do WHILE foi variavel=10, uma vez -lt 10 tem como resultado FALSO, e por isso, termina o loop. A válvula de escape do UNTIL foi variavel=9, uma vez que 9 -lt 10 tem como resultado VERDADEIRO, e por isso, termina o loop.

É correto o que se afirma em:
A) I, apenas.
B) I e II, apenas.
C) I, II e IV apenas.
D) II e IV, apenas.
E) I, II, III e IV.

5) Marque a alternativa que contenha as formas **corretas** de se executar um script através de um terminal linux:

A) ./meuscript.sh (Funciona mesmo sem setar as permissões)
sh meuscript.sh (Somente após setar as permissões)

B) ./meuscript.sh (Somente após setar as permissões)
sh meuscript.sh (Funciona mesmo sem setar as permissões)

C) /.meuscript.sh (Somente após setar as permissões)
sh meuscript.sh (Funciona mesmo sem setar as permissões)

D) /.meuscript.sh (Funciona mesmo sem setar as permissões)
sh meuscript.sh (Somente após setar as permissões)

E) meuscript.sh (Somente após as permissões)
sh meuscript.sh (Funciona mesmo sem setar as permissões)

6) O ShellScript nos permite extrair a lógica de negócio dos scripts para blocos de códigos separados, chamados de Funções. As funções permitem que o código fique mais organizado, legível e de fácil entendimento. Muitas vezes não apresenta diferença de resultado para o usuário, mas representa uma grande diferença para o administrador de redes que pretende realizar manutenção em um script. Dessa forma, dê dois exemplos práticos de utilização de funções em uma rede. Responda da forma mais detalhada o possível. **Não** é necessário codificar a função, apenas demonstrar seu funcionamento de forma explicativa.

7) O comando case (caixa de opções) é uma boa alternativa na substituição do if-then-else-fi de vários níveis. Ele nos possibilita associar diversas alternativas para um valor passado apenas para uma variável. Sendo assim, suponha que você é o administrador de uma rede e está construindo um script de backup automatizado. **Codifique** um menu para este script que contenha:

Opção 1 - Realizar Backup
Opção 2 - Deletar Backups anteriores
Opção 3 - Agendar novo Backup
Opção 0 - Sair
Opção * - Opção inválida

Não é necessário implementar as lógicas de cada funcionalidade, apenas codificar a estrutura do menu utilizando o comando **case**.

8) O script a seguir foi desenvolvido para testar uma determinada senha digitada pelo usuário.

```
#!/bin/bash

echo "Por favor digite a senha para logar no sistema"
read SENHA

 if [ "$SENHA" = "123" ];then
    echo "Senha correta, logado no sistema"
 else
    echo "Senha incorreta"
 fi
```

Reescreva o script acima utilizando **parâmetros** ao invés de variáveis dinâmicas (SENHA).