

Introdução ao GNU/Linux + Shell Script

Shell Script

Otavio Goes¹, Ricardo Giacobbo¹ e Rogério A. Gonçalves¹

Em colaboração com:

Luiz Arthur Feitosa dos Santos¹, Rodrigo Campiolo¹ e João Martins de Queiroz Filho¹

¹Universidade Tecnológica Federal do Paraná (UTFPR)
Departamento de Computação (DACOM)
Campo Mourão, Paraná, Brasil

Minicurso GNU/Linux Básico + Shell Script

V Semana de Informática - SEINFO 2018

Agenda I

- 1 Introdução
- 2 Shell Script
- 3 Introdução ao Linux
- 4 Introdução a Programação com o Bash
- 5 Variáveis e o Ambiente do Shell
- 6 Execução Condicional
- 7 Repetição
- 8 Redirecionamento, Pipes e Ligações
- 9 Funções
- 10 Dúvidas
- 11 Referências

Objetivos

- Apresentar uma visão geral sobre o GNU/Linux.
- Apresentar um conjunto de comandos¹ básicos para a utilização com o terminal.
- Apresentar uma visão geral sobre Shell Script².

Fonte

Material baseado no tutorial: Linux Shell Scripting Tutorial (LSST) v2.0

Escrito por Vivek Gite (?)

http://bash.cyberciti.biz/guide/Main_Page

Material do Minicurso está disponível em:

[https:](https://)

[//github.com/rogerioag/minicurso-intro-gnu-linux-shell-script](https://github.com/rogerioag/minicurso-intro-gnu-linux-shell-script)

¹da Silva (2010)

²?

```
exec-omp-offload-all-configs.sh x
1#!/bin/bash
2
3benchmark=`basename $PWD`
4
5experiment_date='date +\'%d-%m-%Y-%H-%M-%S\''
6OUTPUT=output/${experiment_date}
7
8echo "Executing test for $benchmark, start at `date +'%d/%m/%Y-%T'`"
9
10mkdir -p ${OUTPUT}
11
12for size_of_data in TOY DATASET MINI_DATASET TINY_DATASET SMALL_DATASET MEDIUM_DATASET STANDARD_DATASET LARGE_DATASET EXTRALARGE_DATASET; do
13    for num_threads in 1 2 4 8 12 24; do
14        echo "Compiling ${benchmark} with dataset: ${size_of_data}, schedule: ${omp_schedule}, chunk: ${chunk_size}, threads: ${num_threads}." 
15        for omp_schedule in DYNAMIC; do
16            for chunk_size in 32 64 128 256; do
17                make POLYBENCH_OPTIONS="-DPOLYBENCH_TIME -D${size_of_data}" OMP_CONFIG="-DOPENMP_SCHEDULE_${omp_schedule} -DOPENMP_CHUNK_SIZE=${chunk_size}"
18                mv ${benchmark}-offloading-gpu.exe ${benchmark}-dataset-${size_of_data}-schedule-${omp_schedule}-chunk-${chunk_size}-threads-${num_threads}
19                for (( i = 1 ; i <= 10; i++ ))
20                do
21                    echo "Execution ${i} of ${benchmark} with dataset: ${size_of_data}, schedule: ${omp_schedule}, chunk: ${chunk_size}, threads: ${num_threads}." 
22                    echo "Execution = ${i}, benchmark = ${benchmark}, size_of_data = ${size_of_data}, schedule = ${omp_schedule}, chunk_size = ${chunk_size}, ./${benchmark}-dataset-${size_of_data}-schedule-${omp_schedule}-chunk-${chunk_size}-threads-${num_threads}-offloading-gpu.exe >> ${OUTPUT}/$benchmark.log"
23                done
24            done
25        done
26    done
27done
28done
29echo "End of tests at `date +'%d/%m/%Y-%T'`"
30
31NOHUP_FILE="nohup.out"
32
33if [ -f "$NOHUP_FILE" ]
34then
35    echo "Copy nohup.out to ${OUTPUT}."
36    cp $NOHUP_FILE ${OUTPUT}
37fi
```

#!/bin/bash

Shell Script

O que é o Linux?

- Sistema operacional de código aberto.
- SO baseado no Unix, possui diversas distribuições.
- Desenvolvido por Linus Torvalds, 1991.
- Características: estabilidade, segurança, multitarefa, código aberto.

O que é o Shell?

- O shell é um programa do usuário que permite a interação do usuário com o sistema.
- Há diversos shells para o Linux.
 - BASH (Bourne-Again SHell): projeto GNU compatível com o Bourne Shell (sh). É o mais comum.
 - CSH (C SHell): sintaxe e uso similar à linguagem C.
 - KSH (Korn SHell): baseado nas especificações POSIX e criado na AT&T.
 - TCSH: versão melhorada do shell CSH.

O que é o Shell?

- Prompt do Shell: Tem-se o terminal (Konsole, XTerm), console, ou shell remoto (SSH).
- A versão do shell pode ser verificada com o comando:
`echo $SHELL`

Comandos úteis

- CTRL + L : Limpa a tela.
- CTRL + W : Apaga conteúdo antes do cursor.
- CTRL + U : Limpa a linha.
- Seta Acima e Abaixo: Navegação de comandos.
- Tab : Completa nomes de arquivos e comandos.
- CTRL + R : Procura por comandos executados anteriormente.
- CTRL + C : Cancela um comando em execução

O que é shell script?

- Shell script é uma série de comandos armazenados em um arquivo texto.
- Shell script consiste de palavras-chaves (if, for), comandos (echo, pwd), binários (free, who), utilitários de texto (grep, awk), funções, controle de fluxo.
- Shell scripts são úteis para automatizar tarefas repetitivas, além de fornecer recursos para leitura de dados do usuário e de arquivos e enviar a saída para a tela e para arquivos.

Como vocês fariam?

Para cada item, digite o comando e a sintaxe.

- ① listar todos os arquivos .conf de um diretório.
- ② listar todos os arquivos com a extensão .conf de um diretório e seus subdiretórios.
- ③ encontrar no arquivo texto.txt a ocorrência da palavra linux.
- ④ imprimir na tela as interfaces de rede e o endereço de hardware.
- ⑤ contar as linhas do arquivo teste.txt.

O que é o Bash?

- Desenvolvido pelo projeto GNU.
- O padrão do shell Linux.
- Compatível com o shell original do Unix (sh).
- Incorpora características de outros shells.

Bash

- Viabiliza a execução comandos internos e externos.
- Um comando interno pertence ao próprio shell (Ex: echo)
- Um comando externo pertence ao sistema (Ex: ls)
- Para verificar se um comando é interno ou externo:
`type -a comando`

Scripts do Bash

- /etc/profile: Inicialização do sistema.
- /etc/bash.bashrc: Arquivo executado na inicialização do shell.
- /etc/bash.logout: Arquivo executado na finalização do shell.
- HOME/.bash_profile: Arquivo de inicialização de perfil do usuário.
- HOME/.bashrc: Arquivo de inicialização do shell do usuário.
- HOME/.bash_logout: Arquivo executado na finalização do shell do usuário.

Criando e executando um script

- Usar um editor de texto (nano, vi, kate, gedit, subl)
- Especificar o tipo do script (Shebang), digitar os comandos.
- Gravar o script de preferência com a extensão .sh.
- Alterar a permissão para execução:
`chmod +x nomearquivo.sh`
- Executar o arquivo do script:
`./nomearquivo.sh`

Bash

Exemplo - Olá Bash

```
1      #!/bin/bash
2      # acima não é comentário, é o Shebang
3
4      echo "Olá Bash."
5      echo "E sou o usuário $USER"
6      echo "Eu estou no diretório " $(pwd)
7      echo "Data: " `date +%d%tde%t%B%tde%t%Y`
```

Terminal

```
rogerio@chamonix:~/scripts$ ./hello_bash.sh
Olá Bash.
E sou o usuário rogerio
Eu estou no diretório /home/rogerio/scripts
Data: 29 de setembro de 2016
rogerio@chamonix:~/scripts$
```

Comentários

- O comentário de linha é o caracter #
- O comentário de bloco:

```
1      <<Comentario1
2          Programa simples de Ola Bash.
3          Elaborado para demonstrar comentário.
4          Eu prefiro ainda usar o #.
5      Comentario1
```

Depurando um script

- Para realizar o debug no bash, use o comando:
`bash -x nomescript.sh ou`
`bash -xv nomescript.sh`
- Pode-se ainda utilizar o comando interno set.
`set -x`: ativa o modo de debug
`set +x`: desativa o modo de debug
- E ainda pode ser utilizado diretamente no shebang:
`#!/bin/bash -x`

Atividades

- Faça um script para exibir o nome do usuário ativo.
- Faça um script para executar os comandos de rede ifconfig e route. Use o more para facilitar a visualização.
- Execute o script a seguir e descreva sua saída.

```
1   clear
2   echo "Olá $USER"
3   echo -e "Hoje é \c ";date
4   echo -e "Número de usuários: \c" ; who | wc -l
5   echo "Calendário"
6   cal
7   exit 0
```

Bash - Variáveis

Variáveis do sistema

- São criadas e mantidas pelo Bash.
- São definidas em letras maiúsculas.
- São exemplos PATH, DISPLAY, PS1, USER.
- Os comandos set, env ou printenv visualizam as variáveis de ambiente.

Variáveis do usuário

- São criadas e mantidas pelo programador.
- Devem ser definidas preferencialmente em letras minúsculas.

Variáveis comumente usadas

- BASH_VERSION: Versão do bash.
- HOSTNAME: Nome do computador.
- HISTFILE: O nome do arquivo de histórico de comandos.
- HOME: Diretório do usuário corrente.
- IFS: Internal Field Separator.
- LANG: Codificação dos caracteres.
- PATH: Caminho de busca de comandos.
- PS1: Configurações do prompt.
- SHELL: Define o caminho para o shell.
- DISPLAY: Configura o display gráfico.

Bash - Variáveis

Acesso ao conteúdo das variáveis

```
1 # Imprime o conteúdo de USER
2 echo "$USER"
3
4 # Sem o $ não é possível acessar o conteúdo
5 echo "USER"
6
7 # Pode-se usar {} no nome da variável
8 echo "${USER}"
9
10 # Usando o printf
11 printf "O usuário atual é %s." $USER
```

Bash - Variáveis

Criando e alterando variáveis

- Para criar uma variável:
nome_variavel=valor
- É obrigatório não usar espaço entre o nome da variável, o sinal de atribuição (=) e o valor.
- Exemplo de criação e acesso a variáveis:

```
1  codigo=3
2  usuario="Rodrigo"
3  diretorio="/home/rodrigo"
4  data="$(date)"
5
6  echo "Usuário $usuario, código $codigo, logado."
7  echo "Acesso ao $diretorio em ${data}."
```

Comando echo

- O comando `echo` exibe texto ou valor de variável.
- Não oferece opções de formatação.
- Usado para saída simples, para outros casos recomenda-se o `printf`.
- Exemplo:

```
1 echo "Começando um script."
2 echo "Hoje é $(date +"%d-%m-%Y")."
3 echo -e "Permite usar controles de formatação.\nLegal!"
4 echo $USER
5 echo /etc/*.conf
```

Comando printf

- O comando `printf` exibe saída formatada.
- Usa diretivas e controles de formatação similares ao C.
- Exemplo:

```
1     palavra="linux"
2     printf "%s\n" $palavra
3     printf "%10.4s\n" $palavra
4     num=10
5     printf "%d\n" $num
6     preco=1000.22
7     printf "%f ou %.2f\n" $preco $preco
```

Aspas, Apóstrofos e Barra

- Aspas duplas permitem a substituição de comandos e variáveis.

```
echo "$SHELL"; echo "$(date)"
```

- Aspas simples (apóstrofo) desabilita a interpretação de comandos e caracteres especiais.

```
echo '$SHELL' #imprime $SHELL
```

- A barra desabilita a interpretação de caracteres especiais ou visualiza caracteres.

```
echo "O valor é\R\$ 10,00\""
```

Bash - Ambiente

Continuação de comando

- A barra (\) também é usada para indicar continuação de comando na próxima linha.

```
1 echo "Estou escrevendo uma linha muito longa, \
2 quebrei para ficar mais simples. \
3 Acabei."
```

Proteção de comando

- A barra (\) ou as aspas duplas podem ser usadas para proteger comandos.

```
1 find $HOME -name *.png #erro produzido por *
2 #soluções:
3 find $HOME -name \*.png
4 find $HOME -name "*.png"
```

Bash - Ambiente

Exportando variáveis

- O comando export permite exportar variáveis para processos filhos.
- Exemplo:

```
1     export backup="$HOME/backup"
2     echo "Backup diretório: $backup"
3     bash
4     echo "Backup diretório: $backup"
```

Removendo variáveis

- O comando unset remove conteúdo de variáveis durante a execução do script.

```
1     valor="R$10 ,00 "
2     unset valor
```

Bash - Entrada

Entrada de dados via teclado

- O comando `read` permite ler um valor e armazená-lo em uma variável.
- Sintaxe:

```
1     read -p "Prompt"  variavel1 variavel2 variavelN
```

- Exemplo:

```
1     read -p "Digite seu nome: " nome
2     read -p "Digite sua idade: " idade
3     echo "$nome tem $idade anos de idade."
```

Bash - Entrada

Entrada com temporizador

- O usuário tem um temporizador em segundos para fornecer a entrada, senão o script é interrompido.

```
1     read -t 7 -p "Digite algo em 7 segundos:" conteudo
```

Entrada oculta

- Os caracteres digitados não são visualizados, útil para entrada de senhas.

```
1     read -s -p "Digite a senha:" senha
```

Processamento de entrada com o \$IFS

- O \$IFS é o separador interno ao shell usado para vários processamentos.
- Por padrão, o separador \$IFS é definido como espaço, tabulação e quebra de linha.
- Através da manipulação do \$IFS pode-se realizar processamento avançado na entrada de dados.

Bash - Entrada

Comando read e o \$IFS

- Pode-se separar o conteúdo de uma variável em várias partes usando o read e o \$IFS.
- Exemplo:

```
1     urls="www.ig.com.br www.google.com www.utfpr.edu.br"
2     echo $urls
3     # processamento baseado no padrão definido no $IFS
4     read -r url1 url2 url3 <<< "$urls"
5     echo $url1
6     echo $url2
7     echo $url3
```

Bash - Entrada

Comando read e o \$IFS

- Alterando o valor do \$IFS:

```
1 ipwd="rodrigo:x:1000:1000::/home/rodrigo:/bin/bash"
2 temp="$IFS"
3 IFS=":"
4 read -r login password uid gid info home shell <<< "$ipwd"
5 printf "Nome de login é %s, uid %d, gid %d,\n"
6     diretório é %s com o shell %s.\n"
7     $login $uid $gid $home $shell
8 IFS=$temp
```

Bash - Operadores Aritméticos

Expressões Aritméticas

- As expressões aritméticas são executadas pela sintaxe:
 $\$((\text{expressão}))$
- Exemplos:

```
1 echo $((10+5)) # adição -> 15
2 echo $((10-5)) # subtração -> 5
3 echo $((10*5)) # multiplicação -> 50
4 echo $((10/5)) # divisão -> 2
5 echo $((10%3)) # módulo -> 1
6 echo $((5**2)) # exponenciação -> 25
7 valor=3
8 valor=$((valor+1)) # resultado é 4
9 echo $valor
```

Bash - Variáveis e Constantes

Declarando variáveis e constantes

- As variáveis em Bash tem seu tipo definido na atribuição.
- O comando declare permite declarar uma variável com um tipo.
- Definindo uma variável inteira:

```
declare -i y=10
```

- Definindo uma constante:

```
declare -r pi=3.14
```

- Checando a existência de uma variável:

```
1 ${nome_variável:?variável não existe ou vazia}
```

Comandos úteis ao shell

- `env`: exibe as variáveis de ambiente.
- `which comando`: localiza o caminho completo de um comando.
- `whereis comando`: localiza o binário, fonte e a documentação de um comando.
- `whatis comando`: exibe uma descrição sucinta do comando.
- `history`: exibe o histórico de comandos. Use ! e a linha para reexecutar um comando.
- `alias apelido=comando`: cria um apelido para um comando.

Uso das chaves

- Simplificação na digitação de comandos.

```
ls {U*,P*}
```

```
rm -v teste.{sh,py,c}
```

- Geração de strings com um padrão.

```
echo arquivo{1,2,3}
```

```
echo arquivo{1..3}
```

```
echo arquivo{1..1000}.txt | xargs touch
```

Uso de Asterisco, Interrogação e Colchetes

- Asterisco (*): relaciona com qualquer padrão.

```
ls *.py
```

- Interrogação (?): relaciona com um único caracter.

```
ls arq_??_dezembro.txt
```

- Colchetes ([]): relaciona com qualquer caracter entre colchetes.

```
ls [ab]*.py
```

Bash - Atividades

- ① Escreva um comando para visualizar as variáveis de ambiente.
- ② Escreva um shell script que permite o usuário entrar com um nome de arquivo. Copie o arquivo para o diretório /tmp.
- ③ Escreva um shell script que permite o usuário entrar com um nome de diretório. Crie o diretório na home do usuário corrente.
- ④ Escreva um shell script que permite o usuário entrar com três nomes de arquivo. Copie os arquivos para o pen drive.
- ⑤ Escreva uma calculadora simples com as quatro operações básicas. O usuário digitará dois valores e o sinal da operação. Imprima o resultado.

Estrutura condicional

- Uma estrutura condicional permite a tomada de decisões e controlar o fluxo do programa.
- Há várias estruturas de condicional no bash. Por exemplo, o if (se) e o test.
- O condicional depende de expressões lógicas que retornam true ou false.
- No bash, false corresponde ao valor 0, e true ao valor 1 ou diferente de 0.

Bash - Condicional

Comando test

- É usado para comparações de atributos de arquivos, de strings e comparações aritméticas simples.
- Sintaxe:

```
1  test condição && comando se verdadeiro
2  test condição || comando se falso
3  test condição && comando se verdadeiro || comando se
   falso
```

- Exemplo para verificar se 10 é maior que 5:

```
1  test 10 -gt 5 && echo "É maior." || echo "É menor."
```

Bash - Condicional

Comparação Numérica

- Para comparação numérica tem-se os operadores:

Operador	Finalidade	Uso
eq	igualdade	num1 -eq num2
ge	maior ou igual	num1 -ge num2
gt	maior	num1 -gt num2
le	menor ou igual	num1 -le num2
lt	menor	num1 -lt num2
ne	diferente	num1 -ne num2

Bash - Condicional

Comparação entre Strings

- Para comparação entre strings tem-se os operadores:

Operador	Finalidade	Uso
=	igualdade	string1 = string2
==	igualdade	string1 == string2
!=	diferente	string1 != string2
-z	tamanho igual a zero	-z string

* Ao usar um operador de comparação entre strings, lembre-se de dar espaço antes e após o operador.

Bash - Condicional

Comparação de Atributos de Arquivos

- Operadores de comparação de atributos de arquivos:

Operador	Finalidade	Uso
a	arquivo existe?	-a arquivo
d	arquivo existe e é diretório?	-d arquivo
e	arquivo existe?	-e arquivo
f	arquivo existe e é regular?	-f arquivo
h	arquivo existe e é link simbólico?	-h arquivo
r	arquivo existe e permite leitura?	-r arquivo
s	arquivo existe e tamanho maior 0?	-s arquivo
w	arquivo existe e permite escrita?	-w arquivo
x	arquivo existe e é executável?	-x arquivo

Bash - Condicional

Exemplos com comando *test*

```
1 #!/bin/bash
2 num1=5
3 num2=10
4 palavra1="Linux"
5 palavra2="linux"
6 arquivo="/etc/resolv.conf"
7 dir="/etc"
8 test $num1 -eq $num2 && echo "$num1 e $num2 são iguais." \
9           || echo "$num1 e $num2 são diferentes."
10 test $num2 -gt $num1 && echo "$num2 é maior que $num1."
11 test $palavra1 = $palavra2 || echo "$palavra1 é diferente a
    $palavra2."
12 test "Aprovação" != "Reprovação" && echo "Aprovação é diferente
    de Reprovação."
13 test -f $arquivo && echo "$arquivo existe." || echo "$arquivo n
ão existe."
14 test -d $dir && echo "$dir é um diretório." || echo "$dir não é
    um diretório."
```

Bash - Condicional

Comando []

- Os colchetes tem a mesma finalidade do comando test.
- Ao usar [], deixar espaço antes e após a condição.
- Sintaxe:

```
1      [ condição ] && comando se verdadeiro
2      [ condição ] || comando se falso
3      [ condição ] && comando se verdadeiro || comando se
                      falso
```

- Exemplo para verificar se 10 é maior que 5:

```
1      [ 10 -gt 5 ] && echo "É maior." || echo "É menor."
```

Bash - Condicional

Comando *if*

- O comando *if* possui três variações:
if-then-fi, *if-then-else-fi* e *if-elif-fi*.
- Permite usar um bloco de comandos baseado na condição.
- Pode ser usado com o comando *test* ou com [condição].
- Forma padrão:

```
1      if test $num -eq 0
2      then
3          comando1
4          comando2
5      else
6          comando1
7          comando2
8      fi
```

```
1      if [ $num -eq 0 ]
2      then
3          comando1
4          comando2
5      else
6          comando1
7          comando2
8      fi
```

Bash - Condicional

Exemplo: if-then-else-fi

```
1 #!/bin/bash
2
3 read -s -p "Digite sua senha: " senha
4 if [ $senha == "segredo" ]
5 then
6     echo -e "\nSenha está correta."
7     echo "Você tem o poder."
8 else
9     echo -e "\nSenha não aceita."
10    echo "Não fique triste, tente novamente. :-)"
11 fi
```

Bash - Condicional

Exemplo: if-elif-fi

```
1      #!/bin/bash
2
3      echo "Conversor de Medidas (m->cm ou cm->m)"
4      read -p "Valor: " valor
5      read -p "Converter para (cm ou m): " medida
6
7      if [ $medida == "cm" ]
8      then
9          conversao=$(dc -e "2k$valor 100*pq")
10     elif [ $medida == "m" ]
11     then
12         conversao=$(dc -e "2k$valor 100/pq")
13     else
14         echo "Medida inválida."
15         exit 1
16     fi
17
18     echo "Valor convertido é $conversao."
19     exit 0
```

Bash - Condicional

Exemplo com if-elif-fi

```
1      #!/bin/bash
2
3      read -p "Digite um valor: " n
4      if [ $n -gt 0 ]; then
5          echo "$n é um valor positivo."
6      elif [ $n -lt 0 ]
7          then
8              echo "$n é um valor negativo."
9      elif [ $n -eq 0 ]
10         then
11             echo "$n é o zero."
12         else
13             echo "$n não é um número."
14     fi
```

Estado de Saída

- Todo comando executado no shell tem um estado de saída.
- O valor 0 indica uma execução sem erros.
- Os valores entre 1 e 255 indicam um estado de erro.
- Para saber o significado do retorno de um comando deve-se consultar a documentação.
- O comando exit permite retornar o estado de saída.
- Para obter o estado de saída usa-se o variável ?.

```
1      ps -aux
2      estado=$?
3      echo "Estado do comando ps é $estado."
```

Bash - Condicional

Exemplo: Condicional usando o estado de saída

```
1      #!/bin/bash
2
3      read -p "Usuário: " user
4      passwduser=$(grep "$user" /etc/passwd)
5
6      if [ $? -eq 0 ]
7      then
8          echo "Usuário existe."
9          echo "Linha no /etc/passwd: $passwduser"
10     else
11         echo "Usuário não existe"
12     fi
```

Bash - Condicional

Operador lógico && (e)

- Permite executar outro comando se o estado de saída do anterior é sucesso.
- Sintaxe:

```
1 — comando1 && comando2
```

Operador lógico || (ou)

- Permite executar outro comando se o estado de saída do anterior é erro.
- Sintaxe:

```
1      comando1 || comando2
```

Bash - Condicional

Exemplo usando E/OU Lógicos

```
1#!/bin/bash
2
3 usuario="admin"
4 grep "$usuario" /etc/passwd && echo "Usuário existe."
5
6 arquivo="/etc/resolv.config.falso"
7 cat $arquivo || echo "Arquivo não existe."
8
9 arquivo="/etc/resolv.conf"
10 cat $arquivo && echo "DNS GOOGLE:";grep "8.8.8.8" $arquivo \
11      || echo "Arquivo não existe ou não possui DNS GOOGLE."
```

Bash - Condicional

Operador lógico ! (not)

- Usado para negar uma expressão lógica.
- Sintaxe:

```
1      [ ! expressão ]
```

- Exemplo:

```
1      # se o diretório backup não existe, crie o diretó  
      rio  
2      [ ! -d $HOME/backup ] && mkdir $HOME/backup
```

Passagem de parâmetros

- Scripts em bash podem receber parâmetros via linha de comando.
- A quantidade de parâmetros é obtida por `$#`.
- Os parâmetros são: `$1 $2 ... $9`.
- Para exibir todos parâmetros usa-se `$@` ou `$*`.

Bash - Parâmetros

Exemplo de passagem de parâmetros

```
1 #!/bin/bash
2
3 if [ $# -eq 0 ]
4 then
5     echo "Uso: ${0} parametros"
6     exit 1
7 fi
8
9 echo "Nome do script: $0"
10 echo "Primeiro parâmetro: $1"
11 echo "Segundo parâmetro: $2"
12 echo "Terceiro parâmetro: $3"
13 echo "Número de parâmetros: $#"
14 echo "Todos os parâmetros (\$* version) : $*"
15 echo "Todos os parâmetros (\$@ version) : $@"
16
17 IFS="
18 echo "Todos os parâmetros (\$* version) : $*"
```

Bash - Parâmetros

Exemplo usando parâmetros

```
1      #!/bin/bash
2
3      [ "$#" -eq 0 ] && { echo "Uso: $0 diretório"; exit 1; }
4
5      read -p "Criar diretório $1 <S,N>: " dir
6      if [ $dir == "S" ]
7          then
8              mkdir $1
9              echo "Diretório criado."
10         else
11             echo "Operação cancelada."
12         fi
```

Bash - Condicional

Comando case

- A estrutura case permite comparar diversos valores para uma variável.
- Sintaxe:

```
1      case $variavel in
2          padrao1)
3              comandos
4          ;;
5          padrao2)
6              comandos
7          ;;
8          padraoN)
9              comandos
10         ;;
11         *)
12     esac
```

Bash - Condicional

```
1#!/bin/bash
2OPT=$1      # opção
3FILE=$2      # nome arquivo
4case $OPT in
5    -e|-E)
6        echo "Editando $2 ..."
7        [ -z $FILE ] && { echo "Sem nome do arquivo"; exit 1; }
8        || nano $FILE
9    ;;
10   -c|-C)
11        echo "Exibindo $2 ..."
12        [ -z $FILE ] && { echo "Sem nome do arquivo"; exit 1; }
13        || cat $FILE
14    ;;
15    -d|-D)
16        echo "Hoje é $(date)"
17    ;;
18    *)
19        echo "Uso : $0 -ecd nome_arquivo"
20        echo "__-e arquivo : Edita o arquivo."
21        echo "__-c arquivo : Visualiza o arquivo."
22        echo "__-d           : Exibe a data e a hora."
23    ;;
24esac
```

Bash - Condicional

Operações úteis para condicional

- Converter para minúsculas.

```
1     texto="TrEnZinho"
2     texto=$(tr '[upper:]' '[lower:]' <<< $texto)
```

- Identificar padrões.

```
1     opt=$1
2     case $opt in
3         [Ss][Ii][Mm])
4             echo "Reconhece S ou s seguido de I \
5                   ou i e de M ou m..."
6         ;;
7     esac
```

Bash - Atividades

- 1 Faça um script que leia um valor numérico entre 0 e 9. Imprima por extenso. Use o comando test.
- 2 Faça um script com menu que permita criar, renomear, mover, copiar, excluir um arquivo.
- 3 Faça um script que solicite uma senha. Valide a senha e verifique se o arquivo /etc/network/interfaces configura o dhcp.
- 4 Faça um script para localizar uma string em um arquivo. Leia a string e o nome do arquivo.
- 5 Faça um script para verificar se um nome de arquivo passado por argumento em linha de comando existe. Se existe, verifique se o arquivo é executável, um diretório ou um arquivo regular.
- 6 Faça um script para realizar backup de um diretório. O script deve receber o caminho do diretório e o nome do arquivo de backup. Faça com que o nome do arquivo de backup possua a data do backup.

Estruturas de laços

- As estruturas de laços permitem executar um bloco de comandos repetidas vezes até que uma condição seja satisfeita.
- O Bash possui várias estruturas de laços: for, while, until.
- As estruturas de laços são úteis para executar processamento sobre nomes e/ou linhas de arquivos.

Bash - Repetição

Estrutura for

- O laço *for* processa valores de uma lista.
- Os valores são em geral: strings, números, argumentos por linha de comando, nomes de arquivos, saída de um comando.
- Sintaxe:

```
1 for valor in lista
2 do
3   comando1
4   comando2
5   ...
6   comandoN
7 done
```

- lista pode ser uma variável, sequência numérica, conjunto de strings ou de comandos.

Bash - Repetição

Exemplo for - Sequência numérica

```
1      #!/bin/bash
2
3      echo "Exemplo FOR - Sequência numérica"
4      for num in {1..10}
5      do
6          echo "Execução número $num"
7      done
```

Bash - Repetição

Exemplo for - Retorno de comando

```
1      #!/bin/bash
2
3      echo "Conta e soma os arquivos .conf no diretório /etc"
4      soma=0
5      cont=0
6      for arquivo in $(find /etc -name "*.conf")
7      do
8          tamanho=$(du -s $arquivo | cut -f 1)
9          soma=$((soma+tamanho))
10         cont=$((cont+1))
11     done
12     echo "Total de arquivos: $cont arquivos."
13     echo "Espaço em disco: $soma KiB."
```

Bash - Repetição

Exemplo for - Processa arquivos em um diretório

```
1 #!/bin/bash
2
3 echo "Número de linhas dos arquivos .conf no diretório /etc"
4 soma=0
5 for arquivo in /etc/*.conf
6 do
7   linhas=$(wc -l $arquivo | cut -d " " -f 1)
8   soma=$((soma+linhas))
9 done
10 echo "Linhas: $soma"
```

Bash - Repetição

Exemplo for - Conteúdo de variável

```
1      #!/bin/bash
2
3      # Verifica a existencia de arquivos de senha
4      files="/etc/passwd /etc/group /etc/shadow /etc/gshadow"
5      for f in $files
6      do
7          [ -f $f ] && echo "$f encontrado." \
8                  || echo "$f não encontrado."
9      done
```

Bash - Repetição

Exemplo for - Sintaxe C/Java

```
1      #!/bin/bash
2
3      # for similar ao Java
4      echo "Tabuada do 2"
5      for (( i=0; i<=10; i++ ))
6      do
7          echo "2 x $i = $((2*i))"
8      done
```

Bash - Repetição

Um pouco mais sobre o for

- O for pode ser interrompido usando-se o comando break.
- O for pode ignorar o restante do processamento de um bloco e ir para o próximo usando-se o comando continue.
- Pode-se usar for aninhados.
- O for pode fazer uma execução infinita:

```
1  for ((; ;))
2  do
3      echo "INFINITO - Para parar digite Ctrl + C"
4  done
```

Bash - Repetição

Estrutura while

- O laço while permite executar blocos repetidas vezes baseado em uma condição, ou ainda, a leitura de linhas ou campos em arquivos textos.
- Sintaxe:

```
1  while [  
         condição  
         ]  
2  do  
3      comando1  
4      comando2  
5      ...  
6      comandoN  
7  done
```

```
1  while IFS= read -r linha  
2  do  
3      comando1 na $linha  
4      comando2 na $linha  
5      ...  
6      comandoN  
7  done < "/caminho/arquivo"
```

Bash - Repetição

Exemplo while - Condição

```
1      #!/bin/bash
2
3      i=1
4
5      #enquanto n for menor igual a 10
6      while [ $i -le 10 ]
7      do
8          echo "Repetição número $i"
9          i=$((i+1))
10     done
```

Bash - Repetição

Exemplo while - Leitura de arquivo texto

```
1  #!/bin/bash
2
3  arquivo=/etc/apt/sources.list
4  while IFS= read -r linha
5  do
6      echo $linha
7  done < "$arquivo"
```

Bash - Repetição

Exemplo while - Leitura de campos do arquivo

```
1  #!/bin/bash
2
3  file=/etc/passwd
4
5  while IFS=: read -r user enpass uid gid desc home shell
6  do
7      echo "Usuário $user ($uid) em \\"$home\\" com $shell."
8  done < "$file"
```

Bash - Repetição

Exemplo while - Laço infinito

```
1      #!/bin/bash
2
3      #Recomenda-se usar o : para loop infinito
4      while :
5      do
6          echo "INFINITO - Para parar digite Ctrl+C."
7      done
```

Bash - Repetição

Estrutura until

- Repete até a condição tornar-se verdadeira.
- Procura executar o bloco ao menos uma vez.
- Sintaxe:

```
1     until [ condição ]
2     do
3         comando1
4         comando2
5         ...
6         comandoN
7     done
```

Bash - Repetição

Exemplo until

```
1      #!/bin/bash
2
3      i=1
4      until [ $i -gt 5 ]
5      do
6          echo "Execução $i."
7          i=$((i+1))
8      done
```

Bash - Repetição

Estrutura select

- Permite selecionar itens em uma lista.
- A lista é exibida com números.
- O prompt é configurado na variável PS3.
- Sintaxe:

```
1      select item in lista
2      do
3          comando1
4          comando2
5          ...
6          comandoN
7      done
```

Bash - Repetição

Exemplo select

```
1      #!/bin/bash
2
3      # Configura o prompt do select
4      PS3="Digite um valor da lista: "
5
6      # Configura a lista
7      select selecao in abacaxi banana pera uva
8      do
9          echo "O número digitado corresponde a $selecao. "
10         echo "Para finalizar digite Ctrl+C."
11     done
12     echo ""
```

Bash - Crase ou \$()

Substituição de comando

- Há duas formas: `` ou \$().
- Exemplo:

```
1      #!/bin/bash
2
3      # Para substituir um comando pela resposta $()
4      data1=$(date +"%d-%m-%Y")
5      echo "$data1"
6
7      # Para substituir um comando pela resposta ``
8      data2='date +"%d-%m-%Y"`
9      echo "$data2"
```

Bash - Atividades

- 1 Faça um script que receba um diretório por parâmetro, procure pelos arquivos rar, extraia para um diretório de destino.
- 2 Faça um script que leia os arquivos .java de um diretório e conte a quantidade de if de cada arquivo.
- 3 Faça um script que processe os arquivos de um diretório e imprima estatísticas: maior arquivo, menor arquivo, quantidade de arquivos regulares, quantidade de arquivos executáveis, quantidade de diretórios e total de arquivos.
- 4 Faça um script que compare arquivos de duas pastas e encontre os arquivos comuns.
- 5 Faça um script com menu que permita selecionar: criar diretório, copiar arquivo, remover arquivo, renomear arquivo e sair. Para a operação escolhida execute as ações associadas.

Entrada/Saída

- A entrada/saída no Linux é controlada por arquivos.
- Os arquivos stdin (0) e stdout (1) definem a entrada e saída padrão respectivamente.
- Há também um arquivo padrão de erros, stderr (2).
- Os padrões definidos pelos arquivos são:
 - stdin: entrada padrão (teclado)
 - stdout: saída padrão (tela)
 - stderr: saída de erro padrão (tela)

Redirecionando os fluxos

- É possível redirecionar os fluxos de entrada, saída e erro para outros arquivos.
- Operador <: modifica o fluxo de entrada.
- Operador >: modifica o fluxo de saída.
- Operador 2>: modifica o fluxo de saída de erros.
- Exemplos:

```
1      cat < /etc/apt/sources.list
2      echo "Texto irá para o arquivo." > texto.txt
3      find /root -name "*" 2> erros.txt
```

Criação de arquivos vazios

- Há duas formas clássicas para criar arquivos vazios em Bash.
- Usando o operador >

```
>nome_arquivo
```

- Usando o comando touch

```
touch nome_arquivo
```

Descartando saídas

- Para descartar uma saída, ou seja, não visualizar na tela e nem encaminhar para um arquivo, deve-se redirecionar o fluxo para /dev/null ou /dev/zero.
- Redirecionando saída padrão

comando > /dev/null

- Redirecionando erros

comando 2> /dev/null

- Redirecionando saída padrão e erros

comando &> /dev/null

Bash - Redirecionamento

Redirecionador <<

- Denominado de here document.
- Sintaxe:

```
1     comando <<FIM
2     texto
3     texto
4     FIM
```

- Permite realizar a leitura de um fluxo de informações até que seja digitado o valor informado no redirecionamento. O fluxo é encaminhado como entrada do comando.
- Exemplo:

```
1     wc -w <<FIM
2     Conta a quantidade de palavras deste texto.
3     FIM
```

Bash - Redirecionamento

Redirecionador <<<

- Denominado de here string
- Sintaxe:

```
comando <<< "string ou variável"
```

- Permite processar uma string como se fosse um documento.
- Exemplo:

```
1      wc -w <<< "Conta as palavras desta string."
```

Bash - Redirecionamento

Redirecionador >>

- Permite redirecionar a saída padrão ou a saída de erro para o final de arquivos.
- Interessante para criar arquivos de log ou adicionar processamento que deve ser persistido.
- Exemplos:

```
1      >registro.txt
2      echo "$(date +"%d/%m/%Y")" >> registro.txt
3      for arq in /etc/*.conf
4      do
5          echo "$arq - linhas: $(wc -l $arq 2>>erros.txt | \
6              cut -d " " -f 1)" >> registro.txt
7      done
8      exit 0
```

Bash - Pipes

Usando pipes

- O símbolo | denomina-se pipe.
- O pipe permite obter a saída de um comando e usar como entrada de outro.
- Sintaxe:

comando1 | comando2

- Permite executar processamento avançado encadeando uma sequência de comandos.
- Exemplo:

```
1      #interfaces de redes ativas
2 netstat -i | cut -d " " -f 1 | egrep -v "Kernel|Iface|lo"
```

Bash - Múltiplos Comandos

Usando comandos encadeados

- Para executar uma sequência de comandos usa-se o ; (ponto-vírgula).
- Sintaxe

```
comando1 ; comando2 ; comando3
```

- Os comandos são executados em sequência, no caso, comando 1, comando 2 e comando 3.
- Exemplo:

```
1      # após 2 minutos, desligue o computador
2      sleep 2m; echo "boa noite"; halt &
```

Bash - Strings

Manipulando Strings

```
1      #!/bin/bash
2
3      s1="palavra"
4      s2="Linux Torvards"
5      s3="/home/rodrigo/script/string.txt"
6
7      echo "Tamanho da string \\"$s1\\": ${#s1}"
8      echo "Substring a partir da posicao 6 de \\"$s2\\": ${s2:6}"
9      echo "Substring a partir da posicao 6 com tamanho 3: \\"$s2\
       ": ${s2:6:3}"
10
11     echo "Remove do inicio ate primeira ocorrencia do padrao: $\
        {s3##*.}"
12     echo "Remove do inicio ate primeira ocorrencia do padrao: $\
        {s3##*/}"
13     echo "Remove do inicio ate ultima ocorrencia do padrao: ${\
        s3##*/}"
14
15     echo "Remove do fim ate primeira ocorrencia do padrao: ${s3\
        %.*}"
```

Bash - Atividades

- 1 Faça uma pesquisa sobre os comandos: awk, cut, sed, grep, sort, unique, wc.
- 2 Para os comandos anteriores, implementem 3 exemplos.
- 3 Faça um script que monitore um processo de 5 em 5 segundos e, quando o processo for terminado, desligue a máquina. O processo deve ser finalizado pelo usuário.

Funções

- Para diminuir a complexidade de scripts é possível modularizar o código fonte por meio de funções.
- Sintaxe:

```
1  nome_funcao() {                      function nome_funcao {  
2      comando1                         comando1  
3      comando2                         comando2  
4  }                                }  
                                         }
```

- A chamada da função ocorre por meio de seu nome. `nome_funcao`
- A função deve ser criada antes de ser chamada.

Bash - Funções

Passagem de argumentos

- A função utiliza as variáveis \$1 a \$9 para receber valores.
- Ao chamar as funções, basta escrever após o nome da função os argumentos.
- Exemplo:

```
1 #!/bin/bash
2 imprime() {
3     echo "Total de parâmetros: $#"
4     i=1
5     for arg in $*
6     do
7         echo "Argumento $i: $arg"
8         i=$((i+1))
9     done
10 }
11 imprime "Palavra" 3 100
12 exit 0
```

Bash - Funções

Variáveis globais versus locais

- Em Bash, as todas as variáveis são globais.
- Para declarar uma variável local em uma função usa-se a palavra reservada local.
- Exemplo:

```
1      #!/bin/bash
2      faz_algo() {
3          local data
4          data="10/10/2010"
5          echo "Data (variável local): $data"
6      }
7      data=$(date +"%d/%m/%Y")
8      echo "Data (variável global): $data"
9      faz_algo
10     echo "Data (variável global): $data"
11     exit 0
```

Retorno de funções

- As funções em bash utilizam o comando return para retornar um valor.
- O return permite retornar apenas valores numéricos.
- Se o return não é especificado, o valor de retorno é o estado de saída do último comando da função.
- Para retornar strings deve-se utilizar o comando echo.

Bash - Funções

Exemplo - Retorno de função

```
1      #!/bin/bash
2      site="www.UTFPR.edu.br"
3
4      function to_lower()
5      {
6          local str="$@"
7          local output
8          output=$(tr '[A-Z]' '[a-z]' <<< "${str}")
9          echo $output
10     }
11
12     # invoca a função to_lower
13     out=$(to_lower $site)
14
15     # exibe o resultado
16     echo "Endereço: $out"
```

Bash - Atividades

- ① Faça uma função para verificar se um email é válido. A função deve retornar 1 para verdadeiro ou 0 para falso.
- ② Faça uma função para verificar se um aplicativo passado como parâmetro está instalado. Retorne 1 para verdadeiro ou 0 para falso.
- ③ Faça uma função para verificar se uma máquina está ativa na rede. Retorne 1 para verdadeiro ou 0 para falso.
- ④ Faça uma função para retornar a extensão de um arquivo passado como parâmetro.

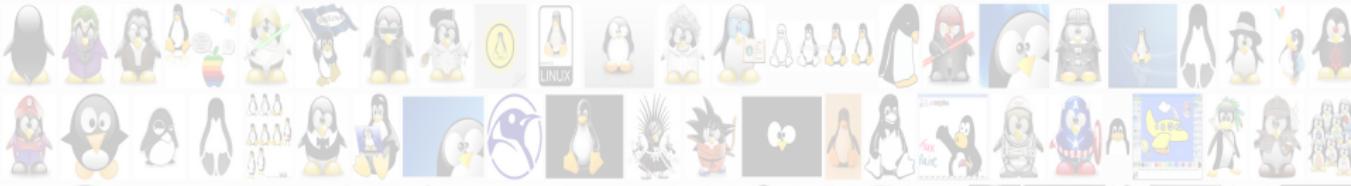


Obrigado!!!

Referências I

da Silva, G. M. (2010). Guia Foca GNU/Linux: Iniciante/Intermediário. Guia/manual, Guia Foca Linux. Versão 5.65.

Agradecimentos



Informações

O material desse minicurso foi preparado em colaboração com os Professores Luiz Arthur Feitosa dos Santos, Rodrigo Campiolo e com o acadêmico João Martins de Queiroz Filho.

Parte do material foi preparado e utilizado no curso de extensão Linux Módulo 1:Básico

(Registro DIREC-UTFPR-CM: 008/2018).

Material do Minicurso está disponível em:

<https://github.com/rogerioag/minicurso-intro-gnu-linux-shell-script>

