

# Introdução ao GNU/Linux + Shell Script

## Comandos Básicos e Shell Script

Otavio Goes<sup>1</sup>, Ricardo Giacobbo<sup>1</sup> e Rogério A. Gonçalves<sup>1</sup>

Em colaboração com:

Luiz Arthur Feitosa dos Santos<sup>1</sup>, Rodrigo Campiolo<sup>1</sup> e João Martins de Queiroz Filho<sup>1</sup>

<sup>1</sup>Universidade 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 Distribuições
- 3 Shell Gráfico (KDE, GNOME, XFCE...)
- 4 Gerenciamento de Arquivos
- 5 Terminal (Console)
- 6 Comandos de Manipulação de Diretórios
- 7 Comandos de Manipulação de Arquivos
- 8 Comandos Diversos
- 9 Informações do Sistema
- 10 Informações dos Usuários
- 11 Busca e Localização
- 12 Manipulação de Processos
- 13 Compactadores
- 14 Básicos de Rede
- 15 Gerenciador de Serviços
- 16 Desligando/Reiniciando a Máquina

# Agenda II

- 17 Permissões de acesso a arquivos e diretórios
- 18 Shell Script
- 19 Introdução ao Shell
- 20 Introdução a Programação com o Bash
- 21 Variáveis e o Ambiente do Shell
- 22 Execução Condicional
- 23 Repetição
- 24 Redirecionamento, Pipes e Ligações
- 25 Funções
- 26 Dúvidas
- 27 Referências

# Objetivos

- Apresentar uma visão geral sobre o GNU/Linux.
- Apresentar um conjunto de comandos<sup>1</sup> básicos para a utilização com o terminal.
- Apresentar uma visão geral sobre Shell Script<sup>2</sup>.

## Fonte

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

Escrito por Vivek Gite (?)

[http://bash.cyberciti.biz/guide/Main\\_Page](http://bash.cyberciti.biz/guide/Main_Page)

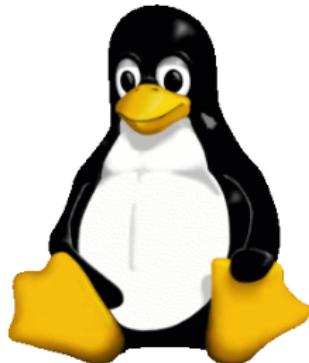
Material do Minicurso está disponível em:

# O que já sabemos sobre o Linux?



O que já sabemos sobre o Linux?

Vocês já usam Linux?



## O que é o Linux?

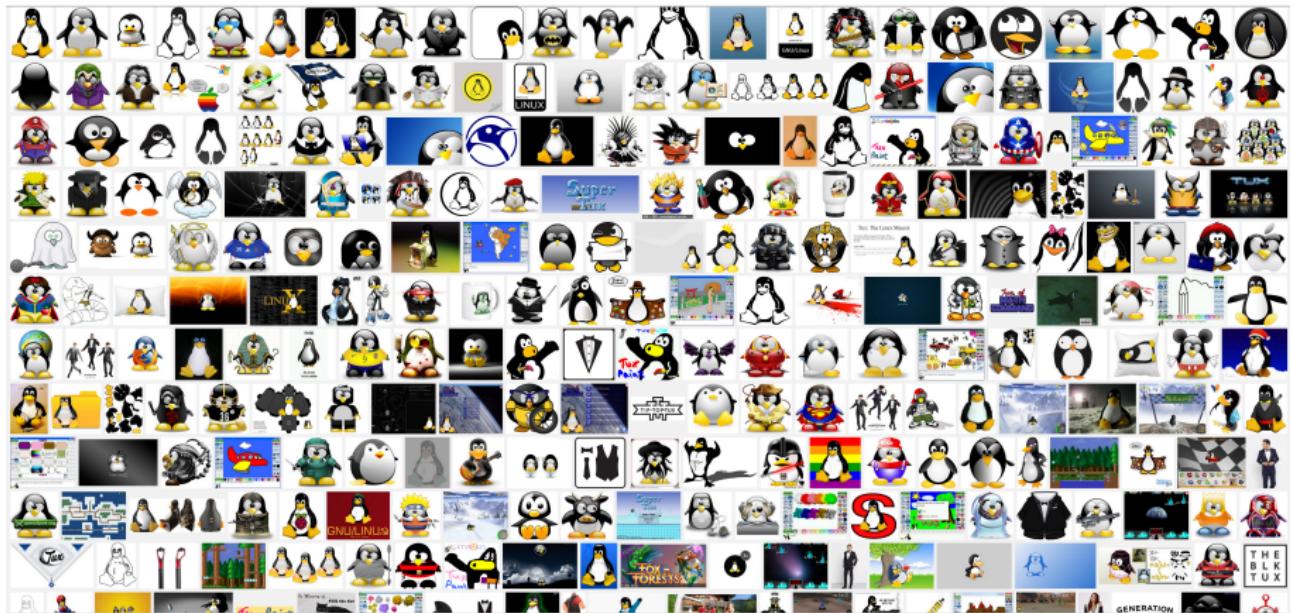
- Sistema operacional de código aberto.
- SO baseado no Unix, possui diversas distribuições.
- Desenvolvido por Linus Torvalds, 1991.
- Características: estabilidade, segurança, multitarefa, código aberto.

# Vocês usam Linux?

- Vocês usam Internet?
  - A maioria dos servidores web são Linux.
  - Sistema Acadêmico, Google...
- Vocês usam Smartphones com Android?
  - Kernel do Linux.
- MacBooks com iOS, iPhones...
  - Darwin BSD
- PlayStation 4
  - FreeBSD
- Computação de Alto Desempenho
  - Unix, Linux [www.top500.org/statistics/sublist](http://www.top500.org/statistics/sublist)

# Por que não usar?

# Linux



<https://www.google.com.br/search?q=tux>

W Linux – Wikipédia, a ... +

<https://pt.wikipedia.org/wiki/Linux>

Pesquisar

Rogerio.rag

Discussão Testes Preferências Beta Páginas vigiadas Contribuições Sair

Artigo Discussão Ler Editar Editar código-fonte Ver histórico Mais Software Livre

Wiki Loves Monuments: Participe do maior concurso fotográfico do mundo enviando suas imagens e concorra a R\$6.000 em prêmios!

WIKIPÉDIA A encyclopédia livre

Página principal Conteúdo destacado Eventos atuais Esplanada Página aleatória Portais Informar um erro Colaboração Boas-vindas Ajuda Página de testes Portal comunitário Mudanças recentes Manutenção Criar página Páginas novas Contato Faça uma doação

Imprimir/exportar Criar um livro Baixar como PDF Versão para impressão Em outros projetos Wikimedia Commons Wikilivros Ferramentas Páginas afiliadas Alterações relacionadas Carregar arquivo Páginas especiais Ligação permanente

6 Código-fonte e documentação

Linux

Origem: Wikipédia, a encyclopédia livre.

# Por que não contribuir?

“Linux” é o nome de um sistema operacional baseado no núcleo Linux, desenvolvido por Linus Torvalds, inspirado no sistema Minix. O seu código fonte está disponível sob a licença GPL (versão 2) para que qualquer pessoa o possa utilizar, estudar, modificar e distribuir livremente de acordo com os termos da licença. A Free Software Foundation e seus colaboradores usam o nome GNU/Linux para descrever o sistema operacional, o que tem gerado controvérsias.<sup>[3][4]</sup>

Inicialmente desenvolvido e utilizado por grupos de entusiastas em computadores pessoais, os sistemas operativos ou sistemas operacionais (português europeu) ou sistemas operacionais (português brasileiro) com núcleo Linux passaram a ter a colaboração de grandes empresas como IBM, Sun Microsystems, Hewlett-Packard (HP), Red Hat, Novell, Oracle, Google, Mandriva e Canonical.<sup>[5]</sup>

Apoiado por pacotes igualmente estáveis e cada vez mais versáteis de software livres para escritório (LibreOffice, por exemplo) ou de uso geral (Mozilla Firefox) e por programas para micro e pequenas empresas que na maioria deles são de código aberto (OpenOffice, LibreOffice, OpenERP, entre outros), o Linux é hoje uma alternativa amigável e econômica ao sistema operacional Microsoft Windows, conhecido por sua estabilidade e robustez, tem gradualmente caído no domínio popular, encontrando-se cada vez mais presente nos computadores de uso pessoal atuais. Mas já há muito que o Linux se destaca como o núcleo preferido em servidores, ainda mais quando se considera que é o único sistema de grande escala usado em computadores espaciais, como o lado norte do Isto mundo, o Tianhe-2, chinês (lista TOP500).

**Documentação, tradução, wikipedia...**

<https://pt.wikipedia.org/wiki/Linux>

[https://pt.wikipedia.org/wiki/Software\\_livre](https://pt.wikipedia.org/wiki/Software_livre)

Tux, a mascote do Linux

Produção	Comunidade
Modelo	Software Livre
Lançamento	1991 (25 anos)
Versão estável	4.2 (23 de agosto de 2015; há 11 meses) <sup>[1]</sup>
Versão em teste	3.9-rc8 (21 de abril de 2013; há 3 anos) <sup>[2]</sup>
Mercado-alvo	Geral
Arquitetura(s)	Diversas
Núcleo	Linux
Licença	GNU GPL & Linux.org
Página oficial	<a href="http://kernel.org">kernel.org</a> & <a href="http://linux.org">linux.org</a>
Estado de desenvolvimento	

# Linux



[http://www.libre-en-touraine.org/UserFiles/File/maquette\\_tux\\_linux-en-touraine\\_logo\\_inside.jpg](http://www.libre-en-touraine.org/UserFiles/File/maquette_tux_linux-en-touraine_logo_inside.jpg)

- Criado por Linus Torvalds.
- 1991
- GNU: GNU/Linux



## Linus Torvalds

[Seguir](#)

Linus Benedict Torvalds é o criador do Linux, núcleo do sistema operacional GNU/Linux. Torvalds nasceu em Helsínquia, na Finlândia. É ateu e filho dos jornalistas Anna e Nils Torvalds, e neto do poeta Ole Torvalds. [Wikipédia](#)

**Nascimento:** 28 de dezembro de 1969 (46 anos), Helsínquia, Finlândia

**Conjuge:** [Tove Torvalds](#)

**Obras:** [Só por Prazer](#), [Linux, c'est gratuit !](#), [mais](#)

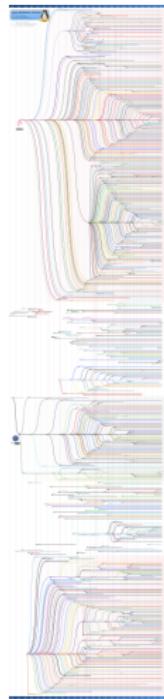
**Nacionalidades:** Finlandês, Americano

**Filhas:** [Daniela Yolanda Torvalds](#), [Celeste Amanda Torvalds](#), [Patricia Miranda Torvalds](#)

**Filiação:** [Anna Torvalds](#), [Nils Torvalds](#)

## Distribuições

- Existem muitas distribuições.
  - <https://distrowatch.com>



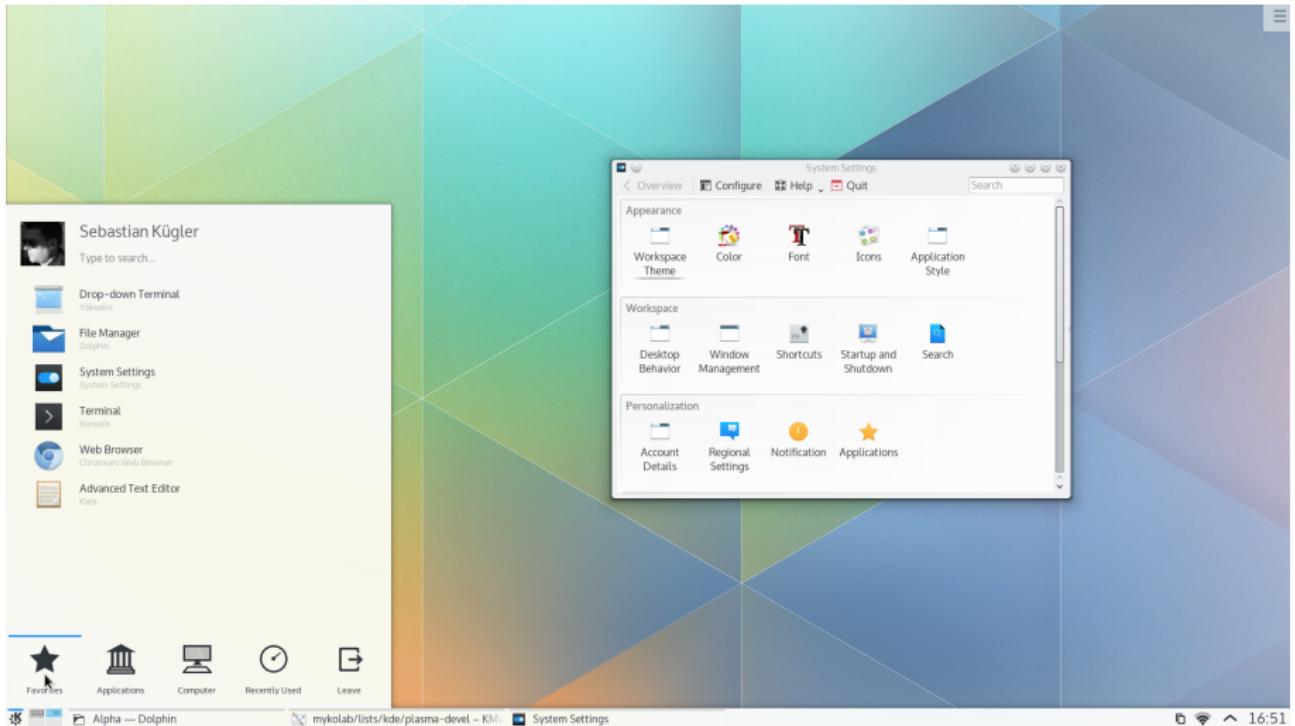
[https://upload.wikimedia.org/wikipedia/commons/1/1b/Linux\\_Distribution\\_Timeline.svg](https://upload.wikimedia.org/wikipedia/commons/1/1b/Linux_Distribution_Timeline.svg)

# Shell Gráfico (KDE, GNOME, XFCE...)

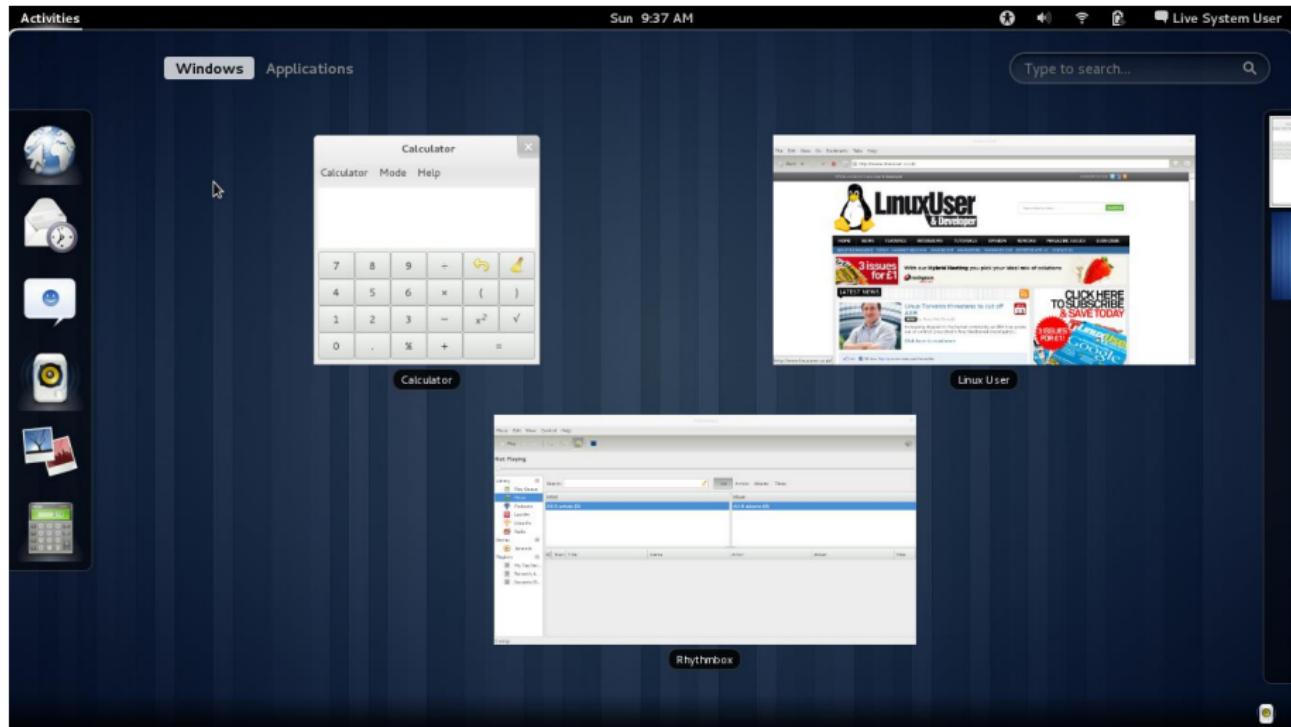
- Modo terminal há mais recursos e flexibilidade.
- Para que haja uma melhor interação do usuário com o sistema, há o modo gráfico.
  - KDE.
  - GNOME
  - XFCE
  - LXDE



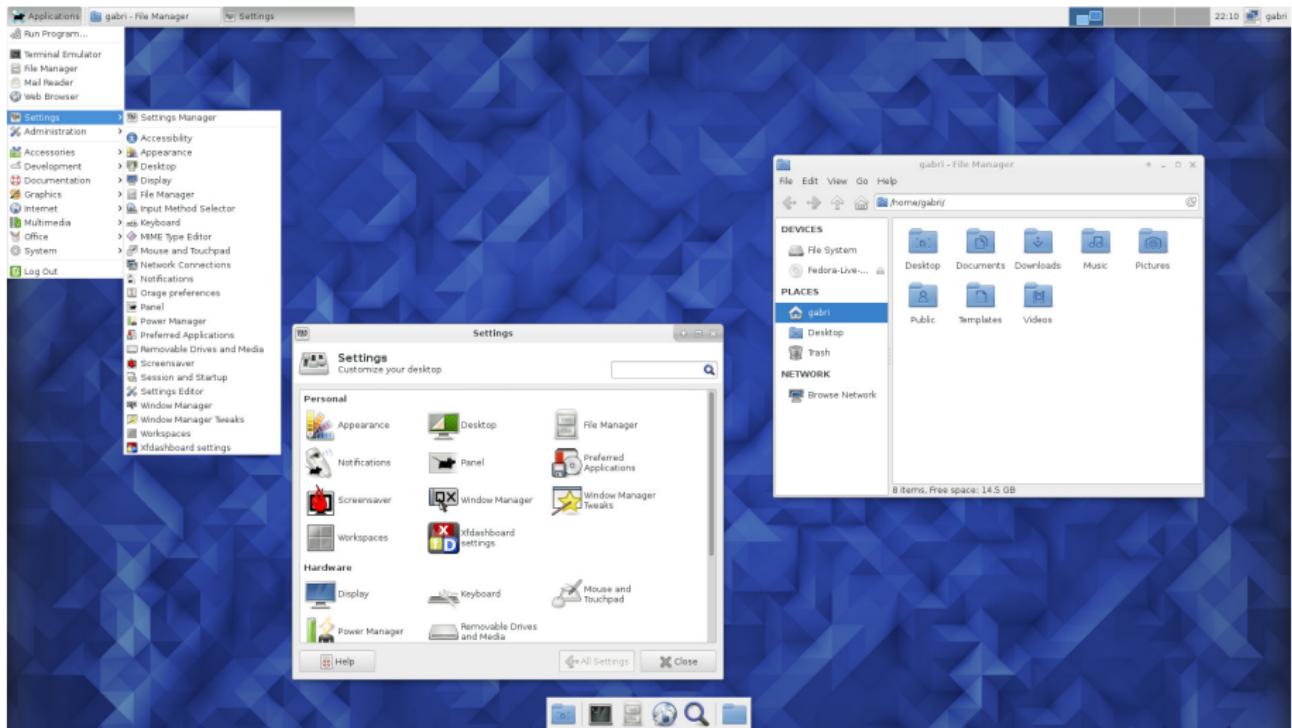
# KDE



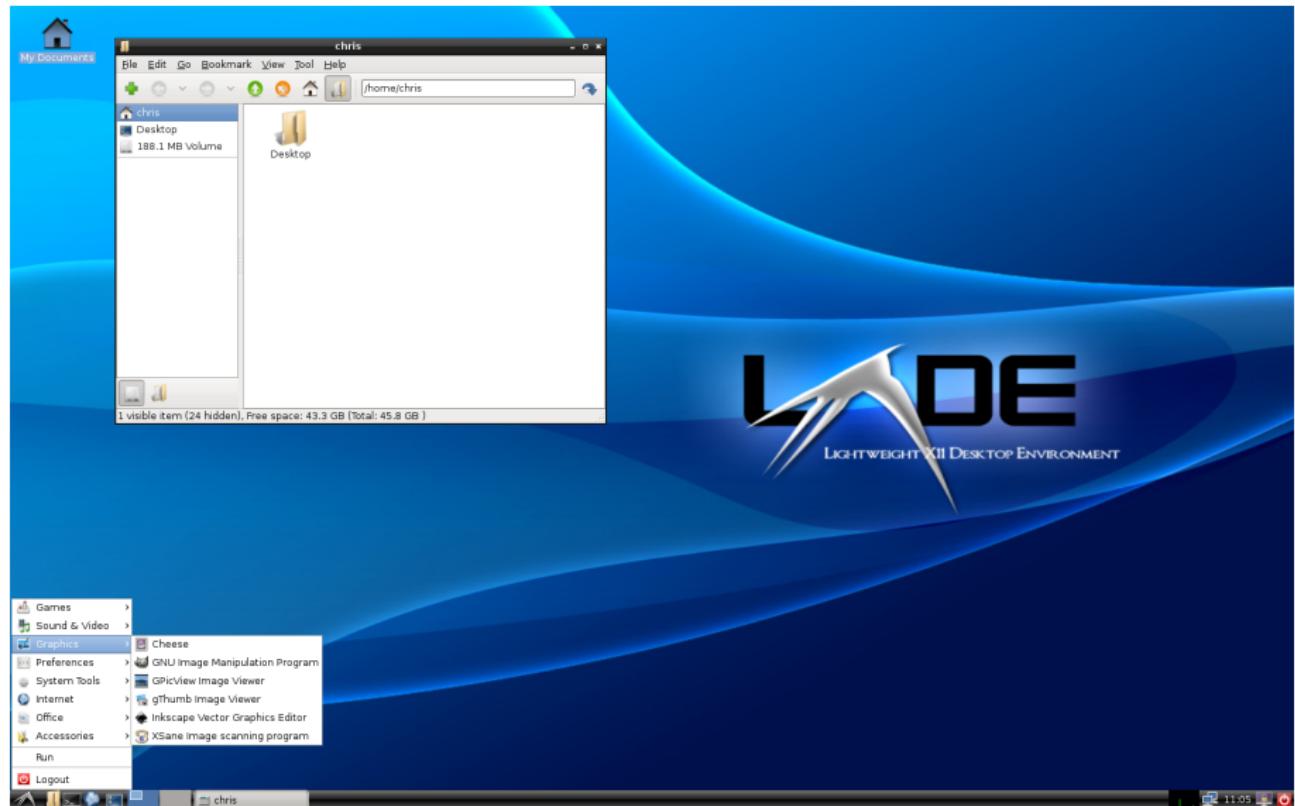
# GNOME



# XFCE



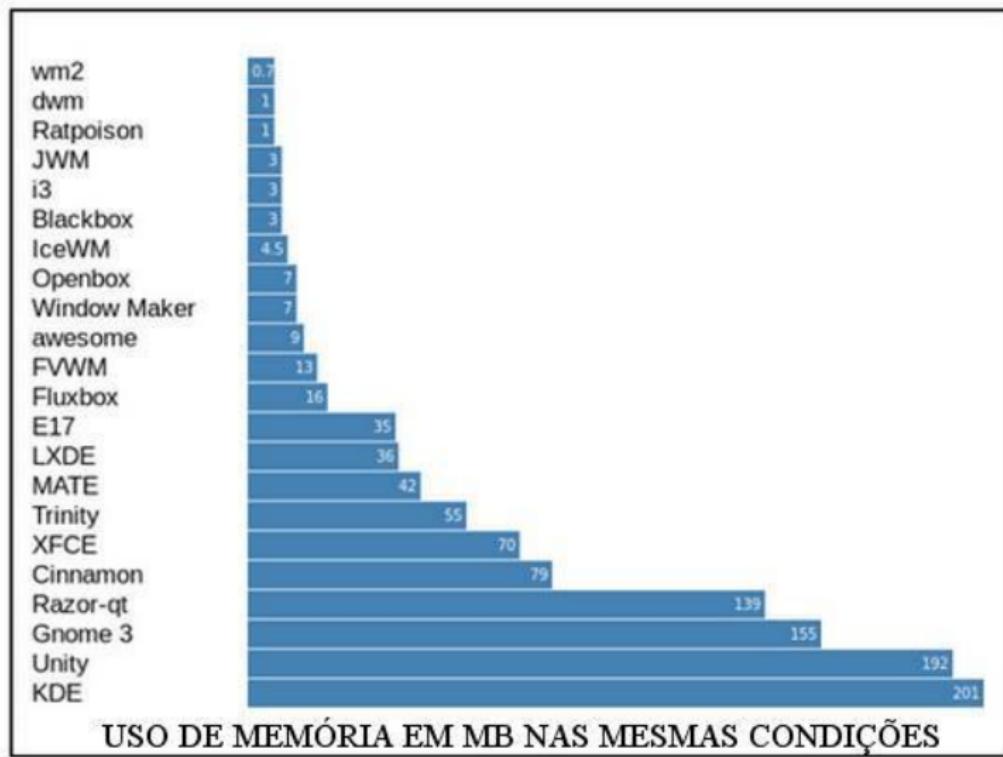
# LXDE



# Outros Modos Gráficos

- MATE.
- Blackbox.
- Unity.
- Cinnamon.

# Uso de Memória por Modo Gráfico



# Linux na UTFPR-CM: Fedora

Atividades

Qua, 16:12



## Atenção

Quando utilizar o usuário **convidado**, salve os arquivos em um pendrive ou envie para o seu e-mail, pois seus dados **serão apagados ao encerrar a sessão**.

Utilize seu **usuário institucional** para salvar localmente os seus arquivos.

A UTFPR não se responsabiliza por arquivos deixados nos computadores.



UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CAMPUS CAMPO MOURÃO  
Departamento Acadêmico de Computação

# Linux na UTFPR-CM: Fedora

The screenshot displays a Fedora desktop environment with the following components:

- File Manager:** Shows a sidebar with "Pasta pessoal" selected. Other locations listed include Recentes, Documentos, Downloads, Imagens, Música, Videos, Lixeira, RAG16GB, and Outras localizações. The main area shows icons for Área de trabalho, Documentos, Downloads, Imagens, Modelos, Música, Público, and Vídeos.
- Terminal:** Shows a terminal window with the following session:

```
[rogerioag@localhost ~]$ pwd  
/home/usuarios/pessoas/rogerioag  
[rogerioag@localhost ~]$ ls  
Área de trabalho Downloads Modelos Público  
Documentos Imagens Música Vídeos  
[rogerioag@localhost ~]$
```
- Web Browser:** Shows the UTFPR website in Mozilla Firefox. The URL is [www.utfpr.edu.br/campomourao](http://www.utfpr.edu.br/campomourao). The page features the UTFPR logo, navigation links for ALUNOS, FUTUROS ALUNOS, EX-ALUNOS, OUTROS CÂMPUS, and links for CHINÊS, VENDE-SE, Acesse o PORTAL do SISU, and FUTURO ALUNO.

# Qual é a Música?



```
rogerioag@localhost:~$ pwd  
/home/usuarios/pessoas/rogerioag  
[rogerioag@localhost ~]$ ls  
Área de trabalho Downloads Modelos Público  
Documentos Imagens Música Vídeos  
[rogerioag@localhost ~]$ █
```

<https://www.youtube.com/watch?v=bXiSCgyvmVE>

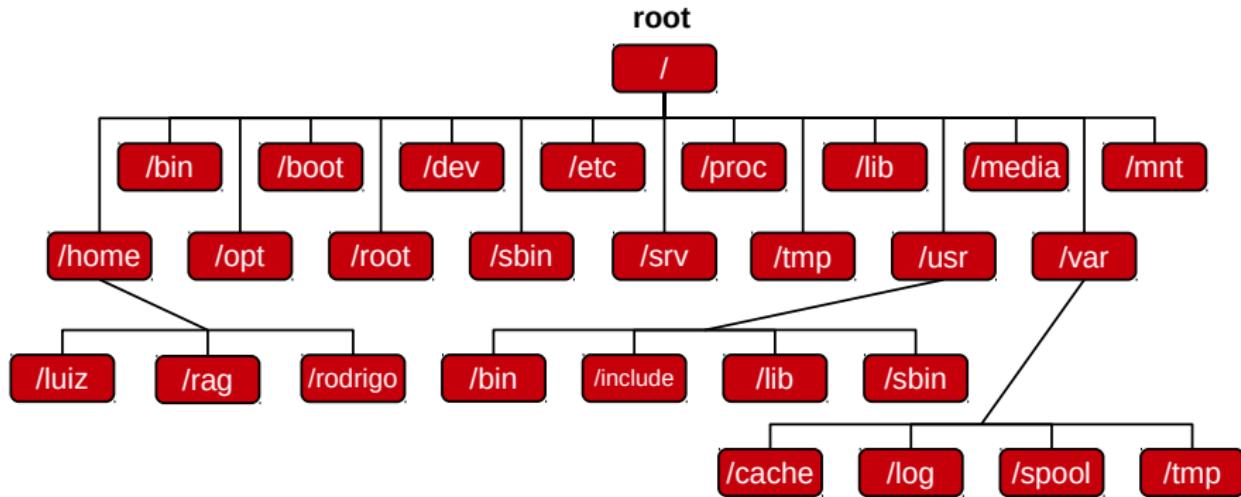
## Diretório (Pasta)

- Local utilizado para armazenar um conjunto de arquivos para melhor organização e localização.
- O diretório, como o arquivo, também é "Case Sensitive".
  - $/teste \neq /Teste \neq /TeStE\dots$
- No Linux/Unix diferentemente do Windows os diretórios são especificados por / e não como \.
- Diretório Raiz: Diretório principal do sistema.
  - Dentro estão todos os diretórios do sistema.
  - Representado por uma "/".
  - Comando `cd /` você acessa esse diretório.

## Diretório (Pasta)

- A *home* do usuário é representado por: ~
- Diretório Atual pode ser recuperado com o comando *pwd* e é representado por: .
- Diretório Superior é representado por: ..
- Diretório Anterior é representado por: -

# Árvore de Diretórios



# Estrutura Básica de Diretórios

Diretório	Conteúdo
/bin	Programas do sistema que são usados com frequência pelos usuários.
/boot	Arquivos necessários para inicialização do sistema.
/dev	Arquivos usados para acessar dispositivos (periféricos) existentes no computador.
/etc	Arquivos de configuração de seu computador local
/home	Diretórios contendo os arquivos dos usuários

# Estrutura Básica de Diretórios

/lib	Bibliotecas compartilhadas pelos programas do sistema e módulos do kernel
/mnt	Ponto de montagem temporário
/proc	Sistema de arquivos do Kernel. Este diretório não existe em seu disco rígido, ele é colocado lá pelo kernel e usado por diversos programas que fazem sua leitura, verificam configurações do sistema ou modifica o funcionamento de dispositivos do sistema através das alterações em seus arquivos.
/root	Diretório do usuário root.

# Estrutura Básica de Diretórios

/sbin	Diretório de programas usados pelos superusuários para administração e controle do funcionamento do sistema.
/tmp	Diretório para o armazenamento de arquivos temporários criados por programas.
/usr	Contém maior parte de seus programas. Normalmente acessível somente como leitura.
/var	Contém a maior parte dos arquivos que são gravados com frequência pelos programas do sistema, e-mail, cache, etc.

# Comandos Básicos

# Comandos básicos I

- Comandos<sup>a</sup> são ordens que passamos ao Sistema Operacional para executar uma determinada tarefa.
- Cada comando tem uma função específica, devemos saber a função de cada comando e escolher o mais adequado para fazer o que desejamos.
- KISS (Keep It Simple Stupid)



---

<sup>a</sup>da Silva (2010)

# Comandos básicos I

- man: manual do sistema.
- info: informações sobre comandos, programas...
- ls: lista o conteúdo do diretório (pasta).
- Seta para Cima: mostra comandos digitados anteriormente.
- Tab: completa palavras. Exemplo digite ls e alguma letra e aperte TAB.
- Ctrl+Shift+r: pesquisa o históricos de comandos já executados

# Terminal I

- `man <nome-do-comando>`
- `info <nome-do-comando>`

## Terminal

```
rogerio@chamonix:~$ man ls
...
rogerio@chamonix:~$ info ls
...
rogerio@chamonix:~$ ls
...
```

# Comandos de Manipulação de Diretórios I

Comando: pwd

Mostra o nome e o caminho do diretório atual.

Terminal

```
rogerio@chamonix:~$ pwd  
/home/rogerio  
rogerio@chamonix:~$
```

# Comandos de Manipulação de Diretórios I

Comando: cd

(change dir) Entrar/Acessar um diretório.

Terminal

```
rogerio@chamonix:/$ cd ~  
rogerio@chamonix:~$ pwd  
/home/rogerio  
rogerio@chamonix:~$
```

# Comandos de Manipulação de Diretórios I

Comando: `mkdir`

(make dir) Criar um diretório no sistema.

## Terminal

```
rogerio@chamonix:~$ mkdir teste
rogerio@chamonix:~$ cd teste
rogerio@chamonix:~/teste$ pwd
/home/rogerio/teste
rogerio@chamonix:~/teste$
```

# Comandos de Manipulação de Diretórios I

Comando: rmdir

(remove dir) Remove um diretório do sistema, porém, o mesmo deve estar vazio.

Terminal

```
rogerio@chamonix:~/teste$ pwd  
/home/rogerio/teste  
rogerio@chamonix:~/teste$ cd ..  
rogerio@chamonix:~/$ rm teste  
rogerio@chamonix:~/$ ls
```

# Comandos de Manipulação de Arquivos I

Comando: cat

Mostra o conteúdo de um arquivo binário ou texto.

Supondo a existência de um arquivo teste.txt com as linhas:

- A
- B
- C
- D

Terminal

```
rogerio@chamonix:~$ ls
teste.txt
rogerio@chamonix:~$ cat teste.txt
A
B
C
D
rogerio@chamonix:~$
```

# Comandos de Manipulação de Arquivos I

Comando: tac

Mostra o conteúdo de um arquivo binário ou texto só que na ordem inversa.

Supondo a existência de um arquivo teste.txt com as linhas:

A  
B  
C  
D

Terminal

```
rogerio@chamonix:~$ ls
teste.txt
rogerio@chamonix:~$ tac teste.txt
D
C
B
A
rogerio@chamonix:~$
```

# Comandos de Manipulação de Arquivos I

Comando: head

Mostra o conteúdo das linhas iniciais de um arquivo texto.

Supondo a existência de um arquivo teste.txt com as linhas:

A  
B  
C  
D

Terminal

```
rogerio@chamonix:~$ ls
teste.txt
rogerio@chamonix:~$ head -n 2 teste.txt
A
B
rogerio@chamonix:~$
```

# Comandos de Manipulação de Arquivos I

Comando: tail

Mostra o conteúdo das linhas finais de um arquivo texto.

Supondo a existência de um arquivo teste.txt com as linhas:

- A
- B
- C
- D

## Terminal

```
rogerio@chamonix:~$ ls  
teste.txt  
rogerio@chamonix:~$ tail -n 3 teste.txt  
B  
C  
D  
rogerio@chamonix:~$
```

# Comandos de Manipulação de Arquivos I

Comando: rm

Apaga arquivos. Também pode ser usado para apagar diretórios e sub-diretórios vazios ou que contenham arquivos.

Supondo a existência de um arquivo teste.txt.

Terminal

```
rogerio@chamonix:~$ ls  
teste.txt  
rogerio@chamonix:~$ rm teste.txt  
rogerio@chamonix:~$ ls  
rogerio@chamonix:~$
```

# Comandos de Manipulação de Arquivos I

## Outros exemplos

- rm teste.txt - Apaga o arquivo teste.txt no diretório atual.
- rm \*.txt - Apaga todos os arquivos do diretório atual que terminam com .txt.
- rm \*.txt teste.novo - Apaga todos os arquivos do diretório atual que terminam com .txt e também o arquivo teste.novo.
- rm -rf /tmp/teste/\* - Apaga todos os arquivos e sub-diretórios do diretório /tmp/teste mas mantém o sub-diretório /tmp/teste.
- rm -rf /tmp/teste - Apaga todos os arquivos e sub-diretórios do diretório /tmp/teste, inclusive /tmp/teste.

# Comandos de Manipulação de Arquivos I

Comando: cp

Copia arquivos e diretórios.

Supondo a existência de um arquivo teste.txt.

## Terminal

```
rogerio@chamonix:~$ ls  
teste.txt  
rogerio@chamonix:~$ cp teste.txt teste2.txt  
rogerio@chamonix:~$ ls  
teste.txt teste2.txt  
rogerio@chamonix:~$
```

# Comandos de Manipulação de Arquivos I

Comando: mv

Move arquivos e diretórios. Outra funcionalidade é que pode ser usado para renomear arquivos.

Supondo a existência de um arquivo teste.txt.

## Terminal

```
rogerio@chamonix:~$ ls  
teste.txt  
rogerio@chamonix:~$ mv teste.txt teste2.txt  
rogerio@chamonix:~$ ls  
teste2.txt  
rogerio@chamonix:~$ mv teste2.txt ~/Documentos/  
rogerio@chamonix:~$ cd Documentos  
rogerio@chamonix:~/Documentos$ ls  
teste2.txt  
rogerio@chamonix:~/Documentos$
```

# Comandos de Manipulação de Arquivos I

## Outros exemplos

- `mv teste.txt testel.txt` - Muda o nome do arquivo `teste.txt` para `testel.txt`.
- `mv teste.txt /tmp` - Move o arquivo `teste.txt` para `/tmp`. Lembre-se que o arquivo de origem é apagado após ser movido.
- `mv teste.txt teste.new` - (supondo que `teste.new` já existe) Copia o arquivo `teste.txt` por cima de `teste.new` e apaga `teste.txt` após terminar a cópia.

# O que estudamos até agora?

- Distribuições.
- Tipos de modos gráficos.
- Gerenciamento de arquivos.
- Terminal.



```
rogerioag@localhost:~$ pwd
/home/usuarios/pessoas/rogerioag
[rogerioag@localhost ~]$ ls
Área de trabalho Downloads Modelos Público
Documentos Imagens Música Vídeos
[rogerioag@localhost ~]$ █
```

A screenshot of a Linux terminal window titled 'rogerioag@localhost:~'. The window has a menu bar with 'Arquivo', 'Editar', 'Ver', 'Pesquisar', 'Terminal', and 'Ajuda'. The terminal itself shows the command 'pwd' followed by the path '/home/usuarios/pessoas/rogerioag'. Then, the command 'ls' is run, listing the contents of the home directory: 'Área de trabalho', 'Downloads', 'Modelos', 'Público', 'Documentos', 'Imagens', 'Música', and 'Vídeos'. A cursor is visible at the end of the command line.

- Comandos de manipulação de diretórios e arquivos.

# Informações de armazenamento I

Comando: clear

Limpa a tela.

Terminal

```
root@chamonix:~# clear  
root@chamonix:~#
```

# Informações de armazenamento I

Comando: date

Permite ver ou modificar a data e hora do sistema.

## Terminal

```
rogerio@chamonix:~$ date +%d/%m/%Y-%H:%M:%S  
17/09/2016-10:58:08  
rogerio@chamonix:~$
```

# Informações de armazenamento I

Comando: df

(disk free) Mostra a capacidade utilizada de um sistema de arquivos exibindo informações de espaço.

## Terminal

```
root@chamonix:/# df -h
Sist. Arq.      Tam. Usado Disp. Uso% Montado em
/dev/sda1        902G  98G   758G  12% /
udev            10M    0    10M   0% /dev
tmpfs           1,6G  9,1M  1,6G   1% /run
tmpfs           3,9G  69M   3,8G   2% /dev/shm
tmpfs           5,0M  4,0K  5,0M   1% /run/lock
tmpfs           3,9G    0   3,9G   0% /sys/fs/cgroup
tmpfs           791M  8,0K  791M   1% /run/user/119
tmpfs           791M   36K  791M   1% /run/user/1000
root@chamonix:/#
```

# Informações de armazenamento I

## Comando: du

(disk usage) Mostra uma lista detalhada sobre a utilização do disco.  
Opções:

- -a Mostra todos os arquivos e não somente diretórios.
- -c Mostra um total no final da listagem.
- -h Mostra as informações de uma forma mais amigável.
- -s Mostra um sumário do diretório especificado e não o total de cada subdiretório.
- -S Exclui os subdiretórios da contagem.

## Terminal

```
root@chamonix:/# du -hsm Documentos
116(Documentos)
root@chamonix:/#
```

# Comandos para Informações do sistema I

## Comando: free

Mostra detalhes sobre a utilização da memória RAM do sistema

### Terminal

```
rogerio@chamonix:/$ free -t
total        used        free      shared      buffers
Mem:    8098408     7506936      591472     396328     237372
-/+ buffers/cache:  2153964      5944444
Swap:   16579580        3932    16575648
Total:  24677988     7510868    17167120
rogerio@chamonix:/$
```

# Informações do Usuário I

Comando: w

Fornece um sumário de cada utilizador ativo no sistema.

## Terminal

```
rogerio@chamonix:/$ who
rogerio  tty7          2016-09-26  08:08  (:0)
rogerio@chamonix:/$
```

# Informações do Usuário I

Comando: lastlog

Mostra o último login dos usuários cadastrados no sistema.

## Terminal

```
rogerio@chamonix:/$ lastlog
rogerio    pts/0    172.16.255.153  Qui Set 22 17:32:53 -0300 2016
rag        pts/26   172.16.255.197  Sab Set 24 20:57:41 -0300 2016
rogerio@chamonix:/$
```

# Informações do Usuário I

Comando: passwd [usuario]

Modifica a parametros e senha de usuário.

## Terminal

```
rogerio@chamonix:/$ passwd rogerio
```

```
rogerio@chamonix:/$
```

# Comandos de Busca I

## Comando: which

Mostra a localização de um arquivo executável no sistema. Muito usado para descobrir qual versão de um determinado comando.

### Terminal

```
rogerio@chamonix:/$ which gcc  
/usr/bin/gcc  
rogerio@chamonix:/$ which gcc-4.8  
/usr/bin/gcc-4.8  
rogerio@chamonix:/$
```

# Comandos de Busca I

## Comando: find

Localiza a partir de um caminho ou diretório recursivamente uma expressão diretamente no sistema de arquivos.

### Terminal

```
rogerio@chamonix:/$ find / -name bash
/bin/bash
rogerio@chamonix:/$ find / -iname python
# Desconsidera maiusculo e minusculo
/usr/bin/python
/usr/lib/wx/python
/usr/lib/libreoffice/share/Scripts/python
/usr/lib/gimp/2.0/python
/usr/share/python
/usr/share/lintian/overrides/python
rogerio@chamonix:$
```

# Comandos de Busca I

Opção: `-ctime`

Mostra os arquivos modificados nos últimos dias

Terminal

```
rogerio@chamonix:/$ find /home -ctime 30  
/home/aula/texto1.txt  
rogerio@chamonix:/$
```

Opção: `-size`

Localiza arquivos com tamanho menor que o declarado

Terminal

```
rogerio@chamonix:/$ find / -size 30k  
/home/aula/texto1.txt  
rogerio@chamonix:/$
```

# Comandos de Busca I

## Comando: locate

Localiza uma palavra na estrutura de um arquivo/diretório do sistema.  
É útil quando queremos encontrar um programa ou comando.

## Terminal

```
rogerio@chamonix:/$ locate string.h
/usr/include/string.h
/usr/include/bsd/bitstring.h
/usr/include/bsd/string.h
/usr/include/bsd/sys/bitstring.h
/usr/include/c++/4.8/bits/basic_string.h
rogerio@chamonix:/$
```

# Comandos de Busca I

Comando: apropos/whatis

Procura programas/comandos através da descrição.

## Terminal

```
rogerio@chamonix:/$ whatis ifconfig
ifconfig (8)           - configura uma interface de rede
rogerio@chamonix:/$
```

# Comandos de Busca I

Comando: grep

Procura por um texto dentro de um arquivo(s).

Terminal

```
rogerio@chamonix:/$ grep "teste" teste.txt
teste
rogerio@chamonix:/$
```

# Comandos para Informações do Sistema I

Comando: top

Mostra detalhes sobre os processos que estão em execução.

```
rogerio@chamonic: ~
top - 23:48:46 up 12 days, 8:12, 4 users, load average: 0,14, 0,22, 0,28
Tasks: 279 total, 1 running, 278 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2,0 us, 0,3 sy, 0,0 ni, 97,3 id, 0,4 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem: 16346884 total, 8573084 used, 7773800 free, 12764 buffers
KiB Swap: 15625212 total, 445432 used, 15179780 free. 3738828 cached Mem

      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
28754 rogerio   20   0 2658492 1,117g 188212 S 12,0 7,2 70:43.37 firefox
29270 rogerio   20   0 2120304 518644 40404 S 4,3 3,2 39:07.65 gnome-shell
1324 root      20   0 1101232 294232 238188 S 2,0 1,8 209:31.14 Xorg
7645 rogerio   20   0 339216 26892 21476 S 1,0 0,2 0:00.20 gnome-screensho
2293 rogerio   20   0 43568 3396 1708 S 0,3 0,0 0:13.58 dbus-daemon
2312 rogerio   20   0 1415472 48816 10832 S 0,3 0,3 0:52.39 gnome-settings-
24249 rogerio   20   0 2531300 371628 86068 S 0,3 2,3 10:05.86 soffice.bin
28919 rogerio   20   0 607176 34068 14648 S 0,3 0,2 0:30.16 GoogleTalkPlugi
1 root       20   0 177616 5168 2468 S 0,0 0,0 0:09.49 systemd
2 root       20   0     0     0     0 S 0,0 0,0 0:00.24 kthreadd
3 root       20   0     0     0     0 S 0,0 0,0 0:29.45 ksoftirqd/0
7 root       20   0     0     0     0 S 0,0 0,0 3:53.45 rcu_sched
8 root       20   0     0     0     0 S 0,0 0,0 0:00.00 rcu_bh
9 root       rt   0     0     0     0 S 0,0 0,0 0:01.03 migration/0
10 root      rt   0     0     0     0 S 0,0 0,0 0:00.96 watchdog/0
11 root      rt   0     0     0     0 S 0,0 0,0 0:00.77 watchdog/1
12 root      rt   0     0     0     0 S 0,0 0,0 0:00.66 migration/1
```

# Comandos para Informações do sistema I

Comando: htop

Versão melhorada do comando top para informações de processos e do sistema.

```
rogerio@chamonic: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
1 [          0.0%]   5 [|          0.5%
2 [| |        2.4%]   6 [|          0.5%
3 [| | |      5.7%]   7 [| | |      0.9%
4 [          0.0%]   8 [|          0.5%
Mem[|||||:||||| 4937/15963MB] Tasks: 139, 422 thr; 1 running
Swp[|          434/15258MB] Load average: 0.26 0.36 0.34
Uptime: 12 days, 08:25:38
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
1640 Debian-gd 20 0 53524 2448 0 S 0.0 0.0 0:00.00 (sd-pam)
2243 rogerio 20 0 53524 2088 0 S 0.0 0.0 0:00.00 (sd-pam)
2559 rogerio 20 0 25276 5960 2728 S 0.0 0.0 0:00.39 -
2532 rogerio 20 0 5968 544 520 S 0.0 0.0 0:00.00 /bin/cat
1091 root 20 0 4340 1380 1228 S 0.0 0.0 0:00.01 /bin/sh /usr/bin/mysqld
2353 rogerio 20 0 4340 104 0 S 0.0 0.0 0:00.00 /bin/sh /usr/bin/start-
25225 rogerio 20 0 4667M 355M 13348 S 0.0 2.2 0:02.66 /home/rogerio/.dropbox-
25226 rogerio 20 0 4667M 355M 13348 S 0.0 2.2 0:00.10 /home/rogerio/.dropbox-
25231 rogerio 20 0 4667M 355M 13348 S 0.0 2.2 0:05.87 /home/rogerio/.dropbox-
25232 rogerio 20 0 4667M 355M 13348 S 0.0 2.2 0:01.86 /home/rogerio/.dropbox-
25233 rogerio 20 0 4667M 355M 13348 S 0.0 2.2 0:08.00 /home/rogerio/.dropbox-
25234 rogerio 20 0 4667M 355M 13348 S 0.0 2.2 0:12.83 /home/rogerio/.dropbox-
25235 rogerio 20 0 4667M 355M 13348 S 0.0 2.2 0:01.85 /home/rogerio/.dropbox-
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice -F8Nice +F9Kill F10Quit
```

# Comandos para Informações do sistema I

## Comando: ps

Lista informações sobre os processos que estão em execução.

### Terminal

```
rogerio@chamonix:/$ ps aux
rogerio 12093  0.0  0.0  14692  1728 ?          S    22:45 0:00 gnome
rogerio 12094  0.0  0.0  24092  6124 pts/0      Ss   22:45 0:00 /bin/
      bash
rogerio 13707 18.3  1.9 1130760 154000 ?        S1   23:54 0:01 /opt/
      google/
rogerio 13734  0.0  0.0  19100  2544 pts/0      R+   23:54 0:00 ps
      aux
rogerio@chamonix:/$
```

# Recuperando o pid de um processo I

Comando: pidof

Recupera o pid do processo.

Terminal

```
rogerio@chamonix:~$ pidof firefox  
28754  
rogerio@chamonix:~$
```

# Matando um Processo I

Comando: kill

Envia um sinal para finalizar o processo.

## Terminal

```
rogerio@chamonix:~$ kill -9 13707
rogerio@chamonix:~$ kill -SIGKILL 13707
rogerio@chamonix:~$ killall firefox
rogerio@chamonix:~$
```

# Compactadores I

## Compactadores

São programas especializados em gerar uma representação mais eficiente de vários arquivos dentro de um único arquivo de modo que ocupem menos espaço no seu armazenamento.

Tabela 1: Extensão de arquivos compactados

Extensão	Significado
.zip	Arquivo compactado pelo programa zip e descompactado pelo unzip.
.rar	Arquivo compactado pelo programa rar.
.tar.gz	Arquivo compactado pelo gzip no utilitário tar

# Compactadores I

Comando: zip

zip [opções] [arquivo-destino] [arquivos-origem]

Comando: unzip

unzip [opções] [arquivo.zip] [arquivos-extrair]

## Terminal

```
rogerio@chamonix:~$ zip -r textos.zip ~/Documentos/*.txt
rogerio@chamonix:~$ cd Documentos
rogerio@chamonix:/Documentos$ unzip textos.zip
rogerio@chamonix:/Documentos$
```

# Compactadores I

Comando: rar

rar [opções] [arquivo-destino.rar] [arquivos-origem]

## Terminal

```
rogerio@chamonix:~$ rar a textos.rar ~/Documentos/*.txt
rogerio@chamonix:~$ cd Documentos
rogerio@chamonix:/Documentos$ rar x textos.zip
rogerio@chamonix:/Documentos$
```

# Compactadores I

Comando: tar

tar [opções] [arquivo-destino] [arquivos-origem]

## Terminal

```
rogerio@chamonix:~$ tar -czf textos.tar.gz textos.txt  
rogerio@chamonix:~$ cd Documentos  
rogerio@chamonix:/Documentos$ tar -xzf textos.tar.gz  
rogerio@chamonix:/Documentos$
```

# Compactadores I

Comando: tar

tar [opções] [arquivo-destino] [arquivos-origem]

## Terminal

```
rogerio@chamonix:~$ tar -czf textos.tar.gz textos.txt  
rogerio@chamonix:~$ cd Documentos  
rogerio@chamonix:/Documentos$ tar -xzf textos.tar.gz  
rogerio@chamonix:/Documentos$
```

# Comandos Básicos de Rede I

## Comando: who

Mostra quem está atualmente conectado no computador. Este comando lista os nomes de usuários que estão conectados em seu computador, o terminal e data da conexão.

### Terminal

```
rogerio@chamonix:~$ who
rogerio :0          2016-09-18 13:55 (:0)
rogerio pts/0       2016-09-18 14:26 (:0)
rogerio@chamonix:~$ who -b
system boot 2016-09-18 13:54
rogerio@chamonix:~$
```

# Comandos Básicos de Rede I

## Comando: ping

Verifica se um computador está disponível na rede. Este comando é muito utilizado por alguns programas de conexão e administradores para verificar se uma determinada máquina está conectada na rede e também para verificar o tempo de resposta de cada máquina da rede.

### Terminal

```
rogerio@chamonix:~$ ping www.google.com
PING www.google.com (172.217.29.36) 56(84) bytes of data.
64 bytes from rio01s20-in-f36.1e100.net (172.217.29.36): icmp_seq
=1 ttl=53 time=47.6 ms
64 bytes from rio01s20-in-f36.1e100.net (172.217.29.36): icmp_seq
=2 ttl=53 time=48.8 ms
--- www.google.com ping statistics ---
5 packets transmitted, 2 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 47.130/48.065/49.090/0.775 ms
rogerio@chamonix:~$
```

# Comandos Básicos de Rede I

## Comando: w

Mostra quem está conectado no sistema e o que cada um está fazendo.

### Terminal

```
rogerio@chamonix:~$ w
14:35:22 up 41 min,  2 users,  load average: 0,12, 0,30, 0,51
USER   TTY   FROM      LOGIN@   IDLE   JCPU   PCPU WHAT
rogerio :0    :0   13:55 ?xdm? 10:01 0.03s gdm-session-worker
rogerio pts/0 :0   14:26   1.00s 0.05s 0.00s w

rogerio@chamonix:~$
```

# Comandos Básicos de Rede I

## Comando: traceroute

Mostra o caminho percorrido por um pacote para chegar ao seu destino. Este comando mostra na tela o caminho percorrido entre os Gateways da rede e o tempo gasto de retransmissão.

### Terminal

```
rogerio@chamonix:~$ traceroute www.google.com
traceroute to www.google.com (172.217.29.36), 30 hops max, 60
    byte packets
1  192.168.1.1 (192.168.1.1)  3.418 ms  5.096 ms  6.260 ms
2  * * *
3  * * *
4  * * *
5  * * *
6  * * *
7  72.14.198.181 (72.14.198.181)  78.115 ms 78.882 ms  82.762 ms
rogerio@chamonix:~$
```

# Comandos Básicos de Rede I

## Comando: ifconfig

Utilizado para atribuir um endereço a uma interface de rede ou configurar parâmetros de interface de rede.

### Terminal

```
rogerio@chamonix:~$ ifconfig
eth0      Link encap:Ethernet  Endereco de HW f0:76:1c:fa:af:0b
          UP BROADCAST MULTICAST  MTU:1500  Metrica:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          colisoes:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  Endereco de HW a4:c4:94:5f:89:71
          inet end.: 192.168.0.101  Bcast:192.168.0.255  Masc:255.255.255.0
          endereco inet6: fe80::a6c4:94ff:fe5f:8971/64 Escopo:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metrica:1
          RX packets:303299 errors:0 dropped:0 overruns:0 frame:0
          TX packets:185961 errors:0 dropped:0 overruns:0 carrier:0
          colisoes:0 txqueuelen:1000
          RX bytes:300865340 (286.9 MiB)  TX bytes:88900429 (84.7 MiB)
```

# Comandos Básicos de Rede - ifconfig |

Comando: ifconfig [interface-rede] down/up

Desabilitando e habilitando uma interface de rede.

## Terminal

```
root@chamonix:/# ifconfig eth0 down  
root@chamonix:/# ifconfig eth0 up  
root@chamonix:/#
```

Comando: ifconfig [interface-rede] [endereço-ip]

Atribuindo IP a uma interface de rede.

## Terminal

```
root@chamonix:/# ifconfig eth0 192.168.1.1  
root@chamonix:/#
```

# Comandos Básicos de Rede I

## Comando: ssh

Ferramenta de acesso remoto bastante poderosa, que permite acessar máquinas Linux remotamente de forma segura.

## Terminal

```
rogerio@chamonix:~$ ssh
usage: ssh [-1246AaCfgKkMNnqsTtVvXxYy] [-b bind_address] [-c
           cipher_spec]
           [-D [bind_address:]port] [-E log_file] [-e escape_char]
           [-F configfile] [-I pkcs11] [-i identity_file]
           [-L [bind_address:]port:host:hostport] [-l login_name] [-m
           mac_spec]
           [-O ctl_cmd] [-o option] [-p port]
rogerio@chamonix:~$
```

# Comandos Básicos de Rede - ssh I

Comando: ssh usuario@endereço-ip

Acessando uma máquina remotamente.

## Terminal

```
rogerio@chamonix:~$ suporte@172.18.0.175
ECDSA key fingerprint is e3:37:bc:62:8d:17:6c:f3:bf:63:5e:b2:9d:0
b:b3:d6.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added "[localhost]:2224" (ECDSA) to the list
of known hosts.
suporte@localhost's password:
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 4.2.0-42-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

Last login: Mon Aug 29 10:31:10 2016 from sshserver.coint
suporte@172.18.0.175:~$
```

# Comandos Básicos de Rede I

Comando: scp

Utilizado quando você quer copiar dados entre máquinas/servidores.

Comando: scp [diretório-origem]

usuario@endereço-ip: [diretório-destino]

Utilizado quando você quer copiar dados entre seu computador com a máquina de acesso remoto.

## Terminal

```
rogerio@chamonix:~$ scp ~/Documentos/teste.txt rogerio@192
.168.100.1:/home/rogerio
rogerio@chamonix:~$
```

# Comandos Básicos de Rede I

Comando: scp usuario@endereço-ip:[diretório-origem]  
[diretório-destino]

Utilizado quando você quer copiar dados entre o computador remoto e o destino seja sua máquina

## Terminal

```
rogerio@chamonix:~$ scp rogerio@192.168.100.1:/home/rogerio/teste  
.txt /home/rogerio/  
rogerio@chamonix:~$
```

## Systemd

Sistema de gestão, gerenciamento e inicialização de todos os processos no sistema de forma centralizada, isso vai desde o processo de carregar scripts, todo processo de inicialização do sistema, até o seu desligamento.

- `systemctl`: utilizado para inspecionar e controlar o estado do sistema `systemd` e gerenciador de serviços.
- `systemd-cgls`: exibe recursivamente o conteúdo da árvore de hierarquia de um determinado grupo de controle do Linux.
- `systemadm`: interface gráfica pra gerenciamento de serviços no `systemd` que permite inspecionar e controlar o `systemd`. Não utilize a não ser que você seja um desenvolvedor, pois está em uma versão muito recente.

# Gerenciador de Serviços I

Comando: `systemctl -t service`

Lista todos os serviços em execução.

## Terminal

```
root@chamonix:/# systemctl -t service
UNIT                  LOAD    ACTIVE SUB     DESCRIPTION
accounts-daemon.service loaded  active running  Accounts Service
acpid.service          loaded  active running  ACPI event daemon
atd.service            loaded  active running  execution scheduler
avahi-daemon.service  loaded  active running  Avahi mDNS/DNS-SD
binfmt-support.service loaded  active exited   support for
bluetooth.service     loaded  active running  Bluetooth service
root@chamonix:/#
```

# Gerenciador de Serviços I

Comando: `systemctl stop [nome-service]`

Parar um serviço em execução.

## Terminal

```
rogerio@chamonix:/# systemctl stop bluetooth.service  
rogerio@chamonix:/#
```

Comando: `systemctl start [nome-service]`

Para inicializar um serviço.

## Terminal

```
rogerio@chamonix:/# systemctl start bluetooth.service  
rogerio@chamonix:/#
```

# Gerenciador de Serviços I

Comando: `systemctl disable [nome-service]`

Desabilitar um serviço.

## Terminal

```
rogerio@chamonix:/# systemctl disable bluetooth.service  
rogerio@chamonix:/#
```

Comando: `systemctl enable [nome-service]`

Para habilitar um serviço.

## Terminal

```
rogerio@chamonix:/# systemctl enable bluetooth.service  
rogerio@chamonix:/#
```

# Outros Comandos I

Comando: reboot

Envia um sinal para reiniciar o computador.

Comando: shutdown

Envia um sinal para o computador desligar.

## Terminal

```
rogerio@chamonix:~$ reboot now
rogerio@chamonix:~$ shutdown -r now
rogerio@chamonix:~$ shutdown -h now
```

# Funcionamento I

## Comando: chmod

Configura permissões de acesso de duas maneiras diferentes;  
Simbolicamente e Numericamente.

Quais usuários ?

- u : dono
- g : grupo
- o : outros
- A : todos

Tipo de gravação

- r : leitura
- w : escrita
- x : execução

Combinação

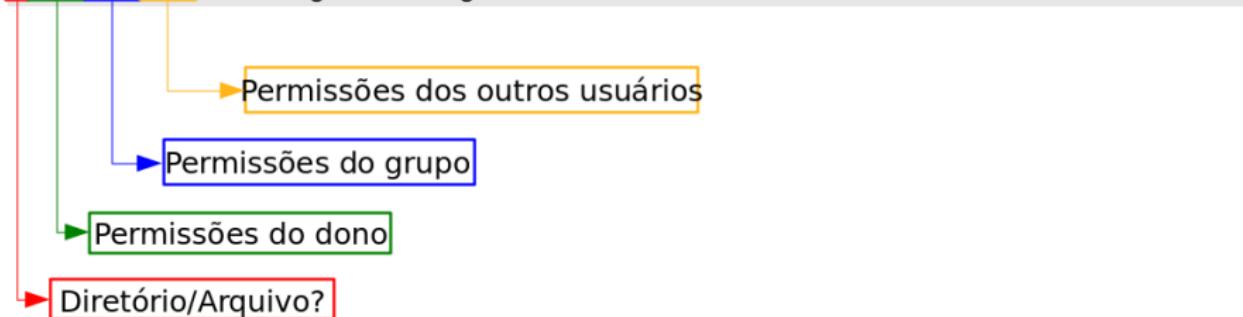
- + : adicionar permissão
- - : remover permissão
- = : definir permissão

# Exemplo I

Comando: ls -lha

Mostra informações do arquivo, nome do proprietário, nome do grupo, tamanho em bytes, rótulo de tempo e o nome do arquivo

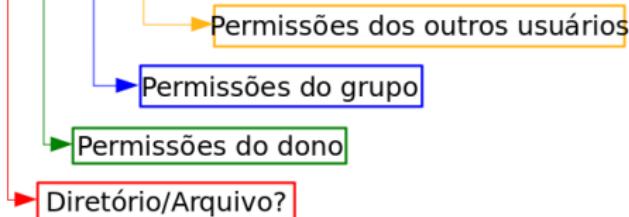
```
rogerio@ragnote:~/teste$ ls -lha
total 12K
drwxr-xr-x  3 rogerio rogerio   36 2009-03-24 17:58 .
drwxr-xr-x 70 rogerio rogerio 4,0K 2009-03-24 17:39 ..
-rw-r--r--  1 rogerio rogerio  170 2009-03-16 16:59 arquivo.txt
drwxr-xr-x  2 rogerio rogerio    6 2009-03-24 17:58 teste2
```



# Exemplo I

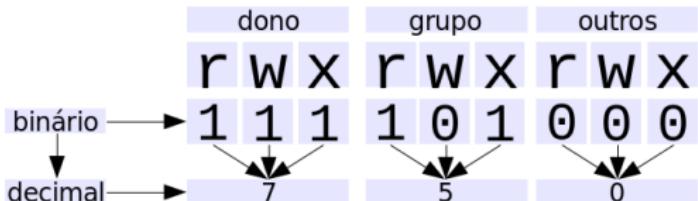
## Funcionamento da permissão em formato simbólico e numérico

```
rogerio@ragnote:~/teste$ ls -lha
total 12K
drwxr-xr-x  3 rogerio rogerio  36 2009-03-24 17:58 .
drwxr-xr-x 70 rogerio rogerio 4,0K 2009-03-24 17:39 ..
-rw-r--r--  1 rogerio rogerio 170 2009-03-16 16:59 arquivo.txt
drwxr-xr-x  2 rogerio rogerio   6 2009-03-24 17:58 teste2
```



r → leitura  
w → escrita  
x → execução

Números Binários
0 1 0
2 1 0
$2^2 \ 2^1 \ 2^0$
$4x0 + 2x1 + 1x0 = 2$



# Exemplo I

Comando: chmod

Funcionamento numérico.

```
rogerio@ragnote:~/teste$ ls -lh
total 12K
-rw-r--r--  1 rogerio rogerio  170 2009-03-16 16:59 arquivo.txt
drwxr-xr-x  2 rogerio rogerio      6 2009-03-24 21:09 teste2
rogerio@ragnote:~/teste$ chmod 700 teste2/
rogerio@ragnote:~/teste$ ls -lh
total 12K
-rw-r--r--  1 rogerio rogerio  170 2009-03-16 16:59 arquivo.txt
drwx-----  2 rogerio rogerio      6 2009-03-24 21:09 teste2
rogerio@ragnote:~/teste$ chmod 777 arquivo.txt
rogerio@ragnote:~/teste$ ls -lh
total 12K
-rwxrwxrwx  1 rogerio rogerio  170 2009-03-16 16:59 arquivo.txt
drwx-----  2 rogerio rogerio      6 2009-03-24 21:09 teste2
rogerio@ragnote:~/teste$
```

# Exemplo I

Retirando todas as permissões

```
rogerio@ragnote:~/teste$ chmod a-rwx arquivo.txt
rogerio@ragnote:~/teste$ ls -lh
total 4,0K
----- 1 rogerio rogerio 170 2009-03-16 16:59 arquivo.txt
drwx----- 2 rogerio rogerio 6 2009-03-24 21:09 teste2
rogerio@ragnote:~/teste$ chmod u+rwx arquivo.txt
rogerio@ragnote:~/teste$ ls -lh
total 4,0K
-rw----- 1 rogerio rogerio 170 2009-03-16 16:59 arquivo.txt
drwx----- 2 rogerio rogerio 6 2009-03-24 21:09 teste2
rogerio@ragnote:~/teste$ chmod g+rwx arquivo.txt
rogerio@ragnote:~/teste$ ls -lh
total 4,0K
-rw-rw---- 1 rogerio rogerio 170 2009-03-16 16:59 arquivo.txt
drwx----- 2 rogerio rogerio 6 2009-03-24 21:09 teste2
rogerio@ragnote:~/teste$ chmod o+x arquivo.txt
rogerio@ragnote:~/teste$ ls -lh
total 4,0K
-rwxr-x--x 1 rogerio rogerio 170 2009-03-16 16:59 arquivo.txt
```

Colocando  
permissões  
para cada um

```
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 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 n o     coment rio ,      o Shebang
3
4      echo "Ol Bash."
5      echo "E sou o usu rio $USER"
6      echo "Eu estou no diret rio " $(pwd)
7      echo "Data: " date +%d%tde%t%B%tde%t%Y
```

## Terminal

```
rogerio@chamonix:~/scripts$ ./hello_bash.sh
Ol Bash.
E sou o usu rio rogerio
Eu estou no diret rio /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 comentario.
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 \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 conteudo de USER
2      echo "$USER"
3
4      # Sem o $ n o     possivel acessar o conteudo
5      echo "USER"
6
7      # Pode-se usar {} no nome da variavel
8      echo "${USER}"
9
10     # Usando o printf
11     printf "%s\n" $USER
```

## 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}."
```

# Bash - Saída

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

# Bash - Saída

## 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 \"R\$ 10,00\""
```

## 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     #solues:
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    %s, uid %d, gid %d,\n"
6         diret rio    %s com o shell %s."\\
7         $login $uid $gid $home $shell
8     IFS=$temp
```

# Bash - Operadores Aritméticos

## Expressões Aritméticas

- As expressões aritméticas são executadas pela sintaxe:  
`$((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      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_variavel:?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 condi o && comando se verdadeiro
2     test condi o || comando se falso
3     test condi o && comando se verdadeiro || comando se
               falso
```

- Exemplo para verificar se 10 é maior que 5:

```
1     test 10 -gt 5 && echo "    maior." || echo "    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 $num2 s o iguais." \
9           || echo "$num1 e $num2 s o diferentes."
10 test $num2 -gt $num1 && echo "$num2 maior que $num1."
11 test $palavra1 = $palavra2 || echo "$palavra1 diferente a
$palavra2."
12 test "Aprova o" != "Reprova o" && echo "Aprova o
diferente de Reprova o."
13 test -f $arquivo && echo "$arquivo existe." || echo "$arquivo
n o existe."
14 test -d $dir && echo "$dir um diret rio." || echo "$dir
n o um diret rio."
```

# Bash - Condicional

## Comando [ ]

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

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

- Exemplo para verificar se 10 é maior que 5:

```
1      [ 10 -gt 5 ] && echo "    maior." || echo "    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 est    correta."
7     echo "Voc   tem o poder."
8 else
9     echo -e "\nSenha n  o aceita."
10    echo "N  o 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 inv lida."
15         exit 1
16     fi
17
18     echo "Valor convertido      $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      um valor positivo."
6      elif [ $n -lt 0 ]
7      then
8          echo "$n      um valor negativo."
9      elif [ $n -eq 0 ]
10     then
11         echo "$n      o zero."
12     else
13         echo "$n      n      um n mero."
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      $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."
```

## Operador lógico ! (not)

- Usado para negar uma expressão lógica.
- Sintaxe:

```
1      [ ! express o ]
```

- Exemplo:

```
1      # se o diretório backup não existe, crie o
          diretório
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 par metro: $1"
11 echo "Segundo par metro: $2"
12 echo "Terceiro par metro: $3"
13 echo "Número de par metros: $#"
14 echo "Todos os par metros (\$* version) : $*"
15 echo "Todos os par metros (\$@ version) : $@"
16
17 IFS="
18 echo "Todos os par metros (\$* version) : $*"
```

# Bash - Parâmetros

## Exemplo usando parâmetros

```
1      #!/bin/bash
2
3      [ "$#" -eq 0 ] && { echo "Uso: $0 diret rio"; exit 1; }
4
5      read -p "Criar diret rio $1 <S,N>: " dir
6      if [ $dir == "S" ]
7      then
8          mkdir $1
9          echo "Diret rio criado."
10     else
11         echo "Opera o 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
2 OPT=$1      # op      o
3 FILE=$2     # nome arquivo
4 case $OPT in
5     -e|-E)
6         echo "Editando $2 ..."
7         [ -z $FILE ] && { echo "Sem nome do arquivo"; exit 1; }
8             || nano $FILE
9
10    ;;
11    -c|-C)
12        echo "Exibindo $2 ..."
13        [ -z $FILE ] && { echo "Sem nome do arquivo"; exit 1; }
14            || cat $FILE
15
16    ;;
17    -d|-D)
18        echo "Hoje      $(date)"
19
20    *)
21        echo "Uso : $0 -ecd nome_arquivo"
22        echo "____-e arquivo : Edita o arquivo."
23        echo "____-c arquivo : Visualiza o arquivo."
24        echo "____-d           : Exibe a data e a hora."
25
26    ;;
27
28 esac
```

# 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 - Sequencia num rica"
4      for num in {1..10}
5      do
6          echo "Execu    o n mero $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 n o 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 [  
2       condi o  
3   ]  
4   do  
5       comando1  
6       comando2  
7       ...  
8       comandoN  
9   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 no mero $i"
9          i=$((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 [ condi   o ]
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 "Execu    o $i."
7          i=$((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 numero 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 ir 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 varivel"
```

- 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      # ap s 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 at     primeira ocorrencia do padr o:
        \${s3#\${.*}}"
12     echo "Remove do inicio at     primeira ocorrencia do padr o:
        \${s3#\${*/}}"
13     echo "Remove do inicio at     ultima ocorrencia do padr o: \${s3#\${*/}}"
14
15     echo "Remove do fim at      primeira ocorrencia do padr o: \${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 par metros: $#"
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.

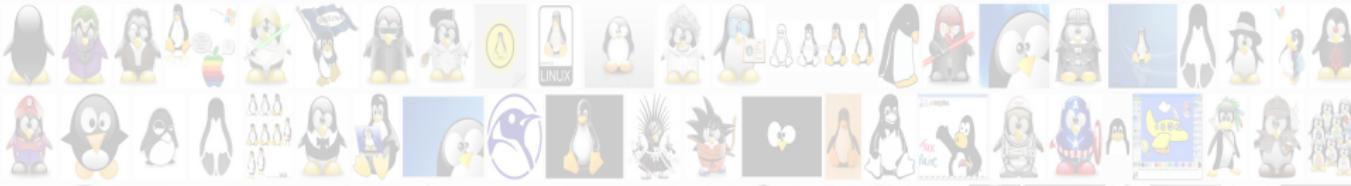


Obrigado!!!

# 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>

