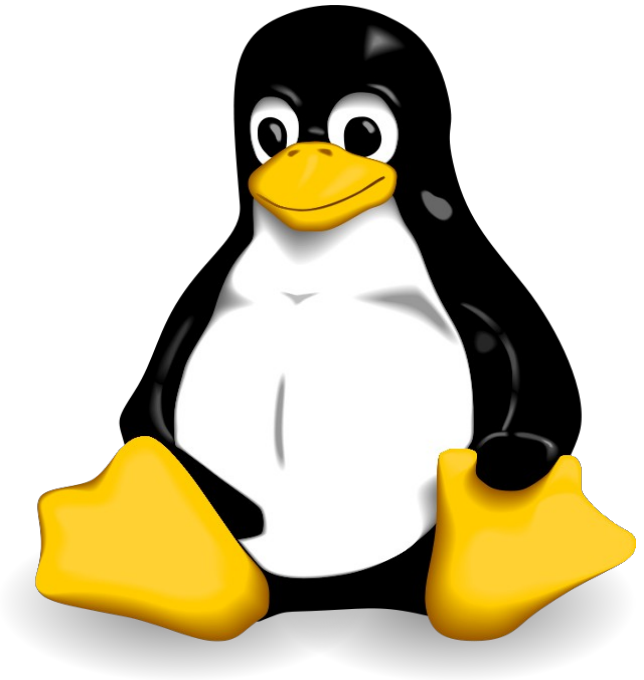
Introdução ao Linux



Prof. Glauber Lopes Mariano

FMet / UFPel

Material de apoio fornecido pelo Prof. Dr. Mateus Teixeira

UNIDADE 1 – INTRODUÇÃO AO SISTEMA OPERACIONAL LINUX

- Introdução

- Estrutura organizacional

- Kernel

- Shell

- Processos

- Estrutura de arquivos

- Comandos básicos

- Redirecionamento e “pipes”

O que é um Sistema Operacional (SO)?

É um conjunto de programas que faz a interface do usuário e seus programas com o computador. É responsável pelo gerenciamento de recursos e periféricos, interpretação de mensagens e execução de programas.

Classificação dos SO

- MONOUSUÁRIO / MULTIUSUÁRIO
- MONOTAREFA / MULTITAREFA

CLASSIFICAÇÃO DOS SO

MONOUSUÁRIO

- Processador somente pode ser utilizado por um único usuário por vez;
- Possui processamento centralizado;

Ex.: MS-DOS.

MULTIUSUÁRIO

- Mais de um usuário usando o processador ao mesmo tempo;
- Um computador (servidor) executa os programas e os gerencia;

Ex.: Unix, Linux, Win NT e 2000...

CLASSIFICAÇÃO DOS SO

MONOTAREFA

- Só consegue executar uma atividade de cada vez.

Ex.: MS-DOS.

MULTITAREFA

- Gerencia a execução de mais de uma tarefa ao mesmo tempo (por execução simultânea ou compartilhamento do tempo entre as tarefas).

Ex.: Win 95, 98, NT e 2000, Unix, Linux.

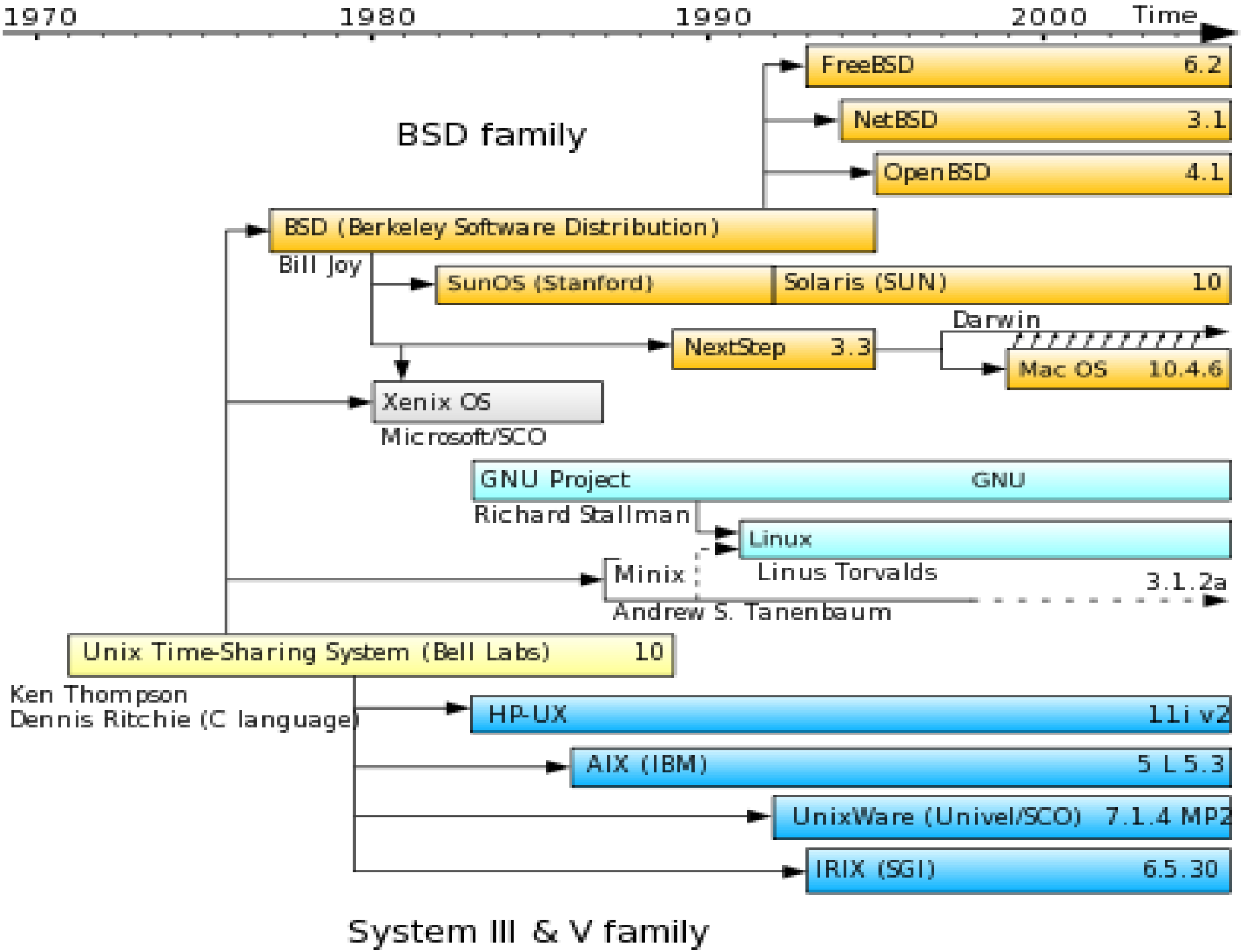
CLASSIFICAÇÃO DOS SO

Sistema Operacional Android??

Mono/Multi usuário
Mono/Multi tarefa

O Linux: a sua história (*short version*).

- Inspirado no MINIX, um pequeno sistema UNIX, desenvolvido por Andy Tanenbaum em 1987;
- O MINIX tinha a sua distribuição e modificação restritas;
- Em outubro de 1991, Linus Torvalds (22 anos), na Universidade de Helsinki, Finlândia, libera um substituto não comercial para o MINIX => Nascia o Linux em sua versão 0.02! (o seu kernel)
- A grande cartada de Linus Torvalds => distribuir o Linux por meio da Licença Pública Geral do GNU (GNU GPL).


1970
1980
1990
2000
Time
BSD family
FreeBSD 6.2
NetBSD 3.1
OpenBSD 4.1
BSD (Berkeley Software Distribution)
Bill Joy
SunOS (Stanford)
Solaris (SUN) 10
Darwin
NextStep 3.3
Mac OS 10.4.6
Xenix OS
Microsoft/SCO
GNU Project
GNU
Richard Stallman
Linux
Linus Torvalds
Minix
3.1.2a
Andrew S. Tanenbaum
Unix Time-Sharing System (Bell Labs) 10
Ken Thompson
Dennis Ritchie (C language)
HP-UX 11i v2
AIX (IBM) 5 L 5.3
UnixWare (Univel/SCO) 7.1.4 MP2
IRIX (SGI) 6.5.30
System III & V family

Software Livre

- O software é livre quando o usuário tem quatro liberdades fundamentais (GPL baseia-se em 04 liberdades):
    - Executar o programa para qualquer propósito;
    - Estudá-lo e adaptá-lo às suas necessidades;
    - Redistribuí-lo, da forma como foi recebido;
    - E modificá-lo e distribuir as modificações;

Software Livre

- Perguntas:

- GRADS seria um software? Linguagem de programação?
- Caso seja software, seria um software livre?

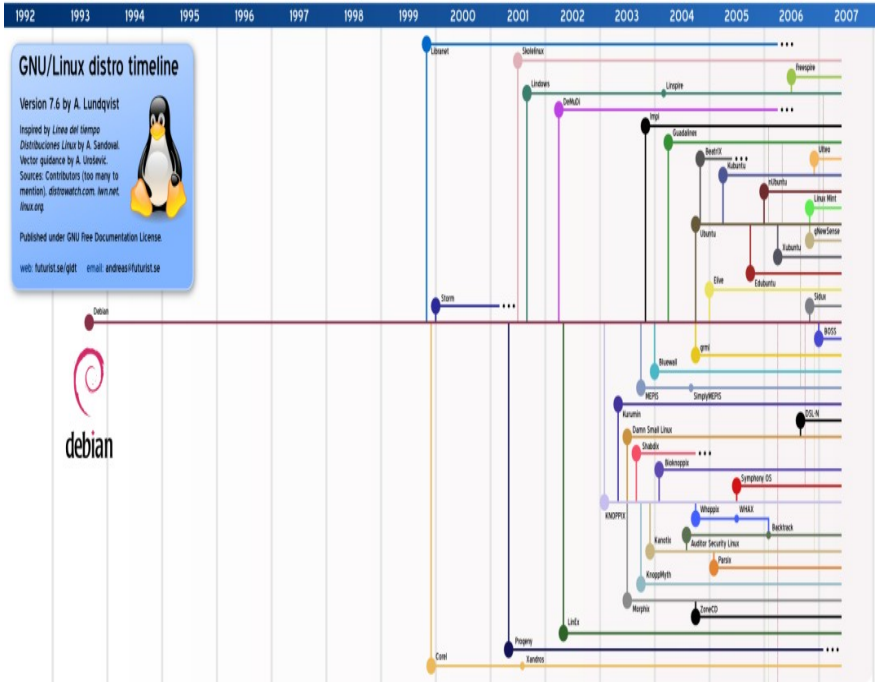
Algumas características (LINUX) ...

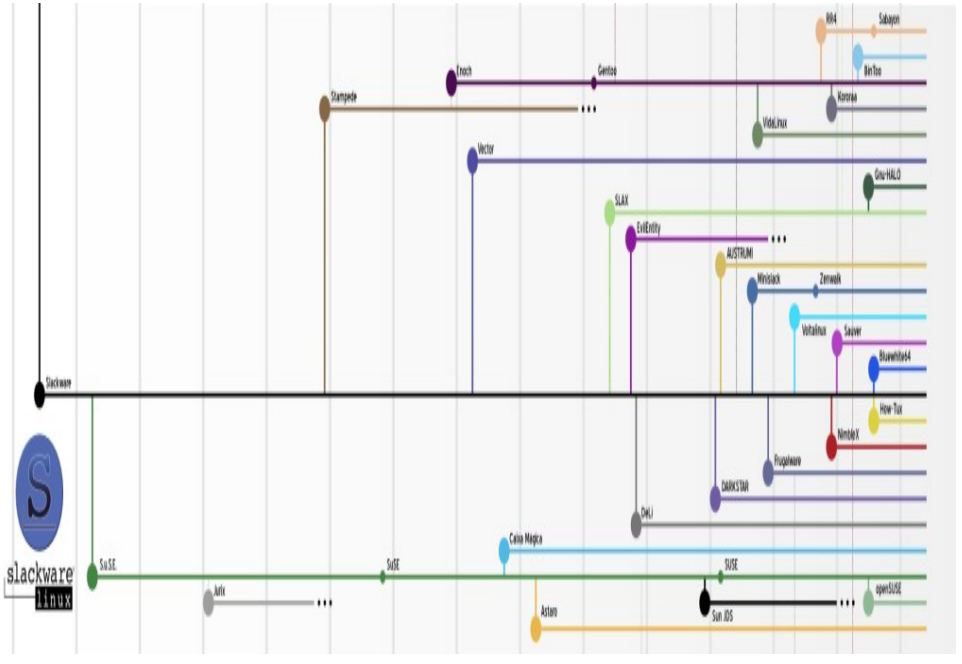
- Livre e desenvolvido por voluntários;
- Convive sem nenhum conflito com outros SO (MS-DOS, MS-Windows, OS/2) no mesmo computador;
- Multitarefa real;
- Multiusuário;
- Suporte a nomes extensos (255 caracteres);
- Conectividade com outras plataformas: Apple, Sun, Macintosh, Sparc, Alpha, Unix, DOS, Windows etc;
- Modularização – somente carrega para a memória o que é usado durante o processamento;
- Não é preciso reiniciar o computador ao modificar configuração de periféricos ou parâmetros de rede;
- Não precisa-se de processador potente: um 386 SX 25, com 4 Mb de RAM roda bem o linux (modo texto);

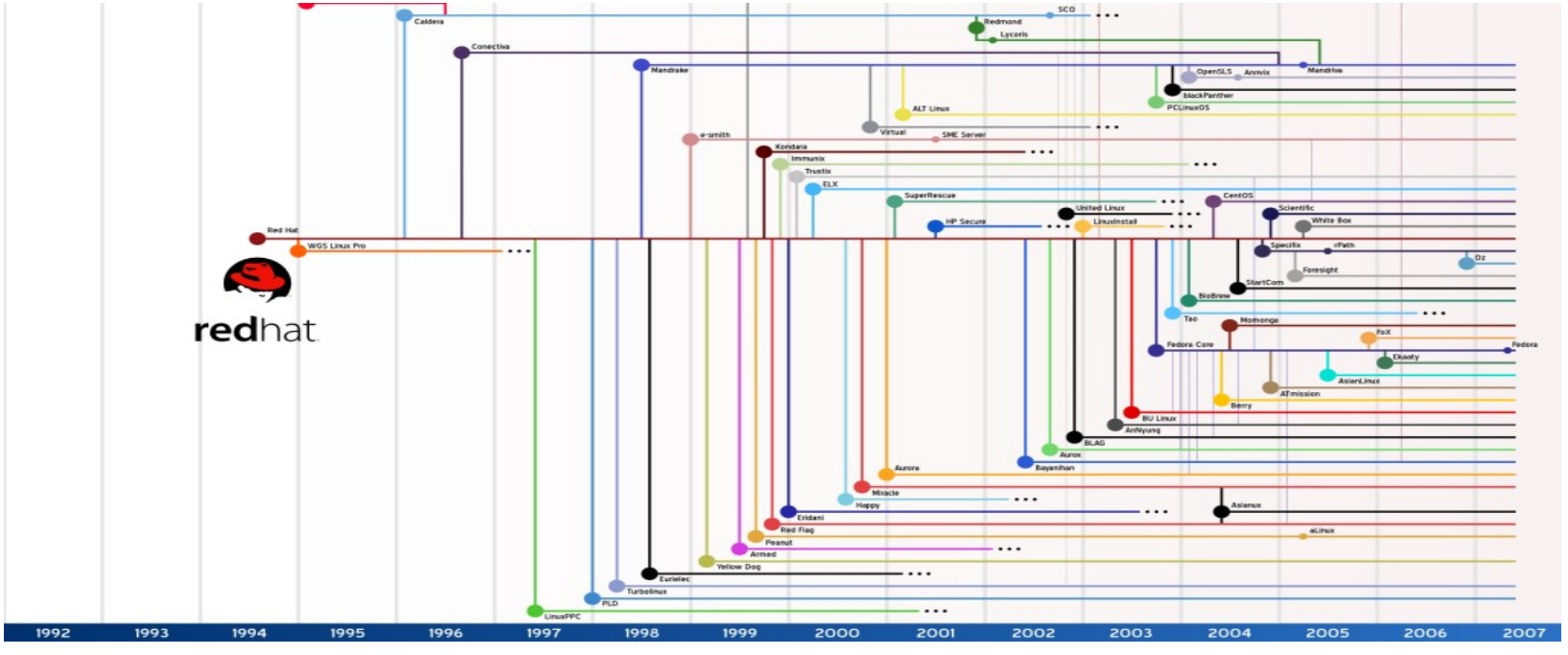
Mais algumas características ...

- Não é requerida licença para uso;
- Utiliza permissões de acesso a arquivos, diretórios e programas em execução na memória RAM;
- Não é vulnerável a vírus! TOTALMENTE VERDADE? O que seria "virus"?
- Roda aplicações DOS e Windows por meio dos emuladores DOSEMU e Wine ("*Wine is not an emulator*"), respectivamente;
- Suporte a dispositivos infravermelho, a rede via rádio amador, a dispositivos Plug-and-Play, a dispositivos USB, a dispositivos Wireless;
- Montagem de servidor Web, E-Mail, News, etc;
- Suporte a diversos dispositivos e periféricos;
- Pode ser executado em 10 arquiteturas diferentes (Intel, Macintosh, Alpha etc)
- E muito mais ...

Linha do Tempo de distribuições do Linux







Distribuições Linux

- Qual seria a “melhor distribuição”?

- O que uma distribuição linux precisa ter para ser a “escolhida” para uso?

- Tamanhos diferentes (disquetes até DVDs)
- Pacote inicial “completo”
- Interface gráfica padrão
- Organização e pré-configuração de software
- Algo mais?

Programando no Linux

- Suporta várias linguagens de programação;
- GNU Compiler Collection (GCC) => C, C++, Java, Ada e FORTRAN;
- Também inclui suporte a Perl, Ruby, Python e outras linguagens dinâmicas;
- Programação gráfica em GNOME e KDE;
- Além de vários outros compiladores proprietários disponíveis;

Interfaces

Interação do usuário com o sistema

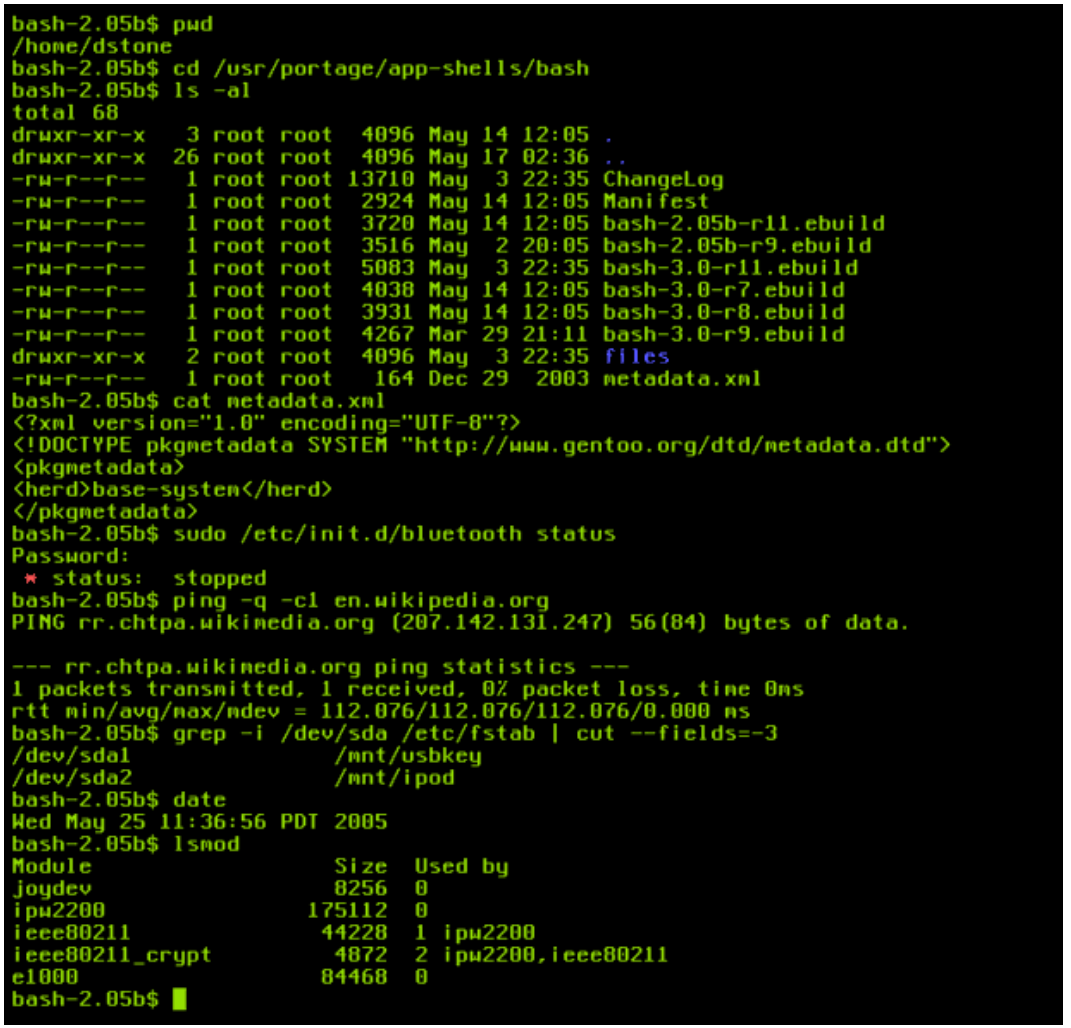
- Modo linha de comando (console de comandos ou *Shell*)

- Modo Gráfico

Para alterar modo gráfico e modo texto:
Ctrl+Alt+F1 (até F4)

Interface por linha de comando

- Interface fornecida através de um Shell;
- Normalmente está “escondida” pelo ambiente gráfico;
- Particularmente apropriada para automação de tarefas repetitivas ou agendadas;
- Acessível no ambiente gráfico através de emuladores de terminal.



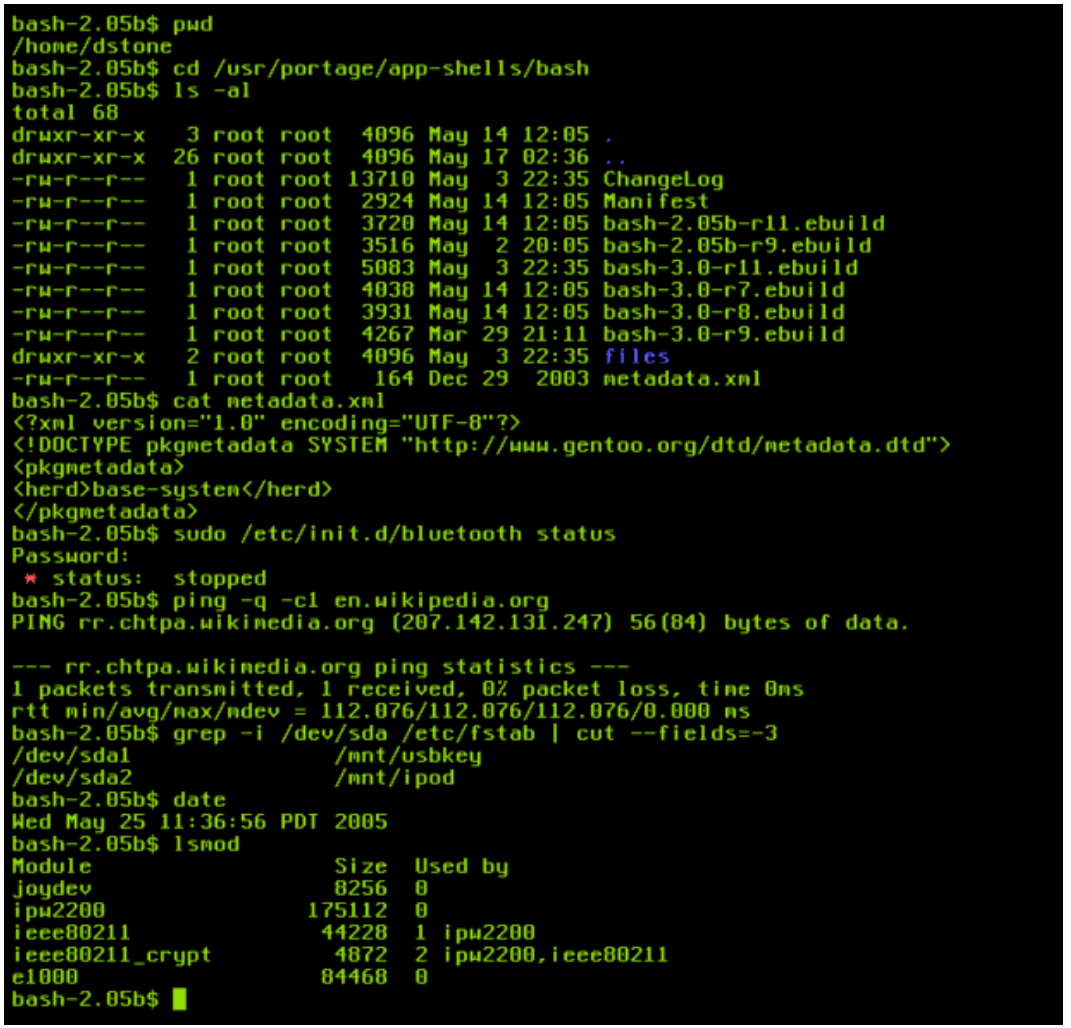
Interface por linha de comando

- Exibe pouco ou nenhum alerta e instruções na tela (prompt);

- É uma interface abstrata, não familiar. Muitos nomes de comandos não tem significado para os usuários, dificultando o entendimento e identificação pelo usuário. ;

- Exige grande carga de memória do usuário para lembrar os comandos e sua syntax;

- A curva de aprendizado inicial é grande para esse tipo de interface;

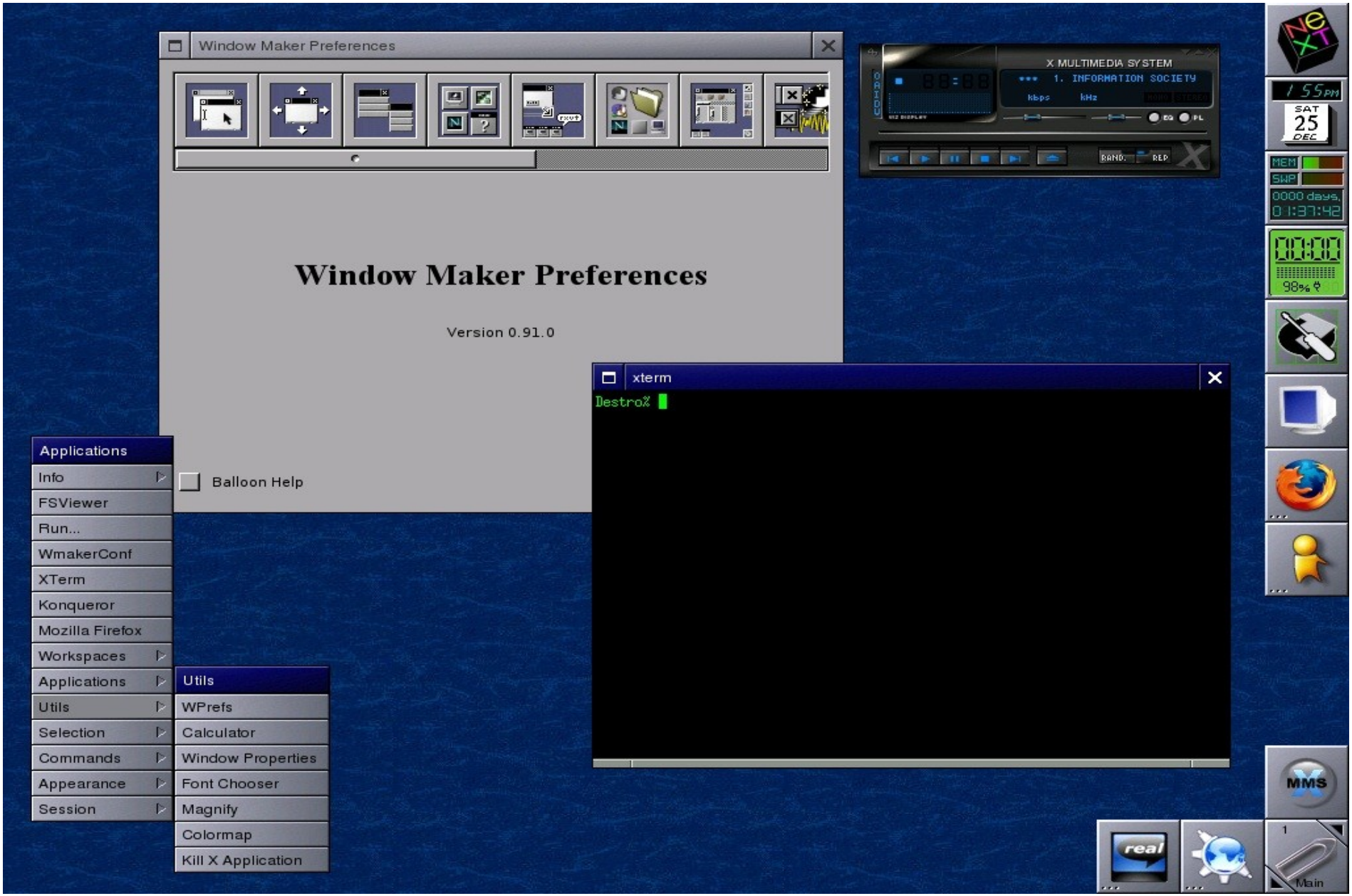


Interfaces gráficas

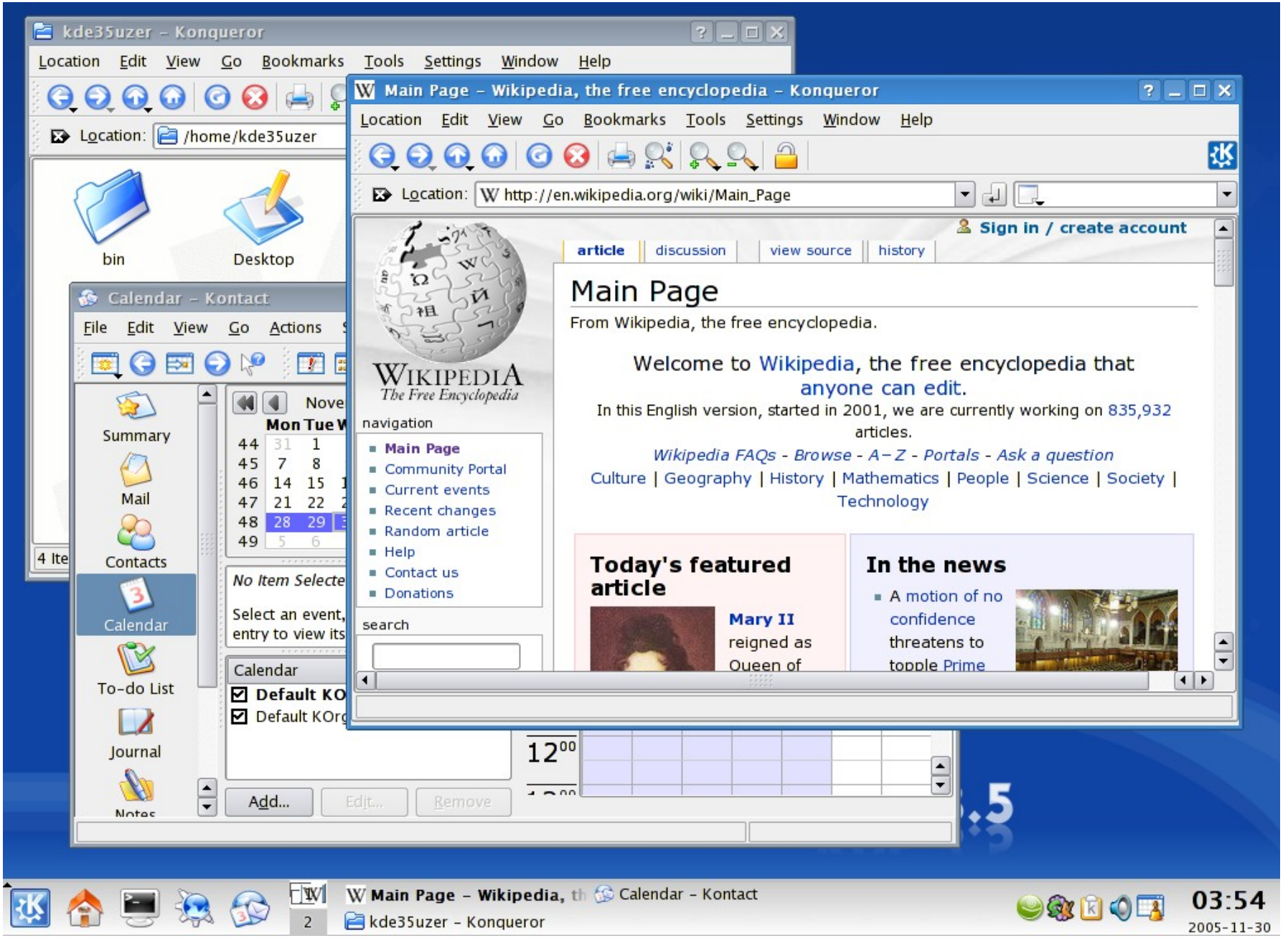
GUI – Graphical User Interface

- X Window System (X11 ou X) é o subsistema gráfico predominante no Linux;
- Acessa à placa de vídeo, memória, teclado, etc...;
- As primeiras interfaces gráficas do Linux eram simplesmente gerenciadores de janelas;
- Atualmente a interface gráfica fornece todo um ambiente gráfico homogêneo.

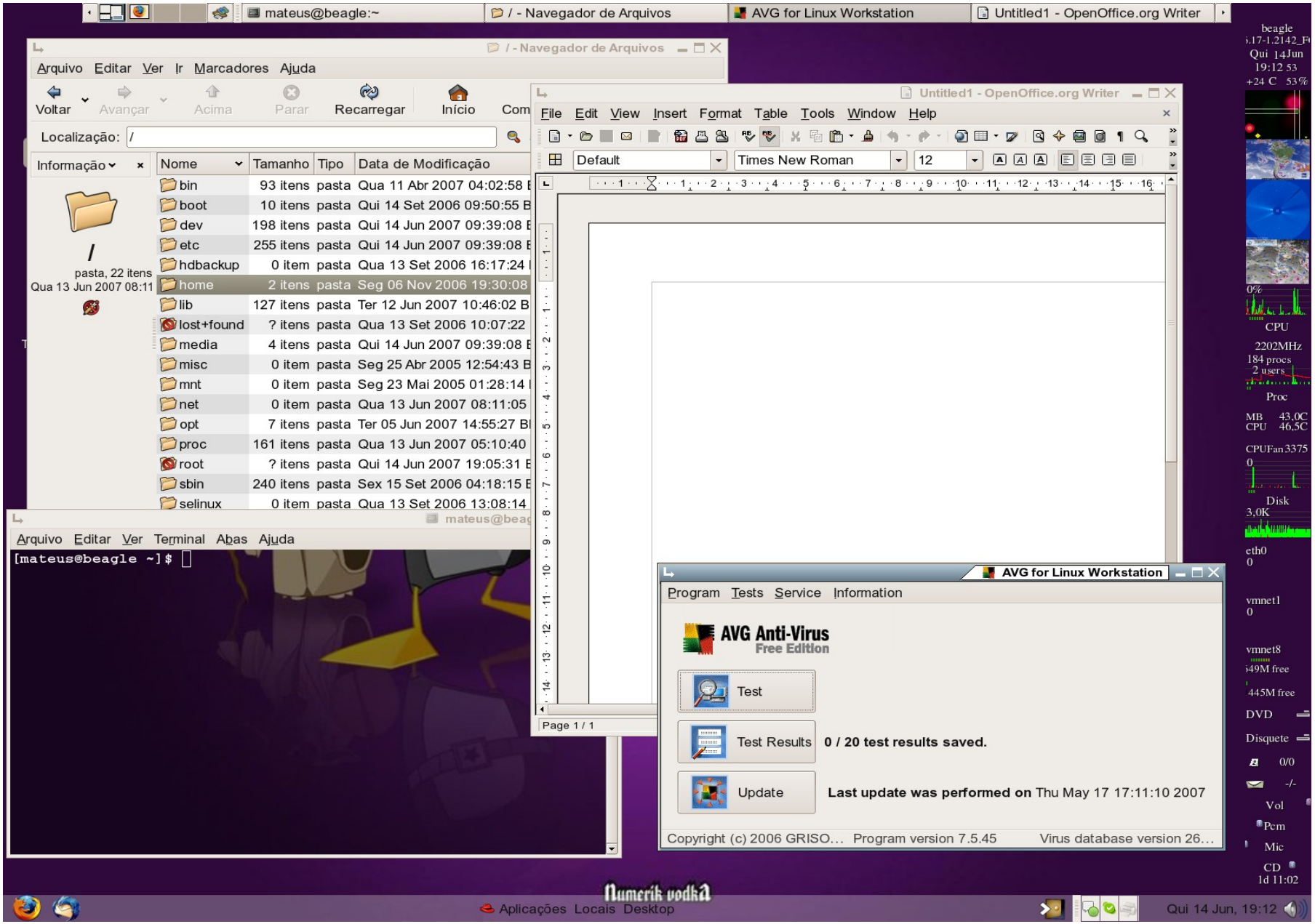
Interfaces gráficas – Window Maker



Interfaces gráficas – KDE



Interfaces gráficas – GNOME



Interfaces gráficas

KDE x GNOME

- Interface gráfica.
- Disponibilização das opções de configurações.
- O gerenciador de arquivos.
- Os aplicativos padrões preferênciais.

Usualmente considera-se:

KDE mais parecido com a interface do Windows
GNOME mais parecido com MacOS

Teste: Mude a interface gráfica do seu sistema

Conectando ao sistema

- Para se trabalhar com o Linux é necessário fornecer um usuário e uma senha;
- Atualmente, o modo gráfico é o mais usado para acessar o Linux (*opção de acessar via modo texto*);
- Tanto no modo gráfico quanto no modo texto tem-se as características de multiusuário, multitarefa e acesso ao mouse.

O usuário Root (*Super Usuário ou "administrador"*)

Tenha cuidado ao se conectar ao sistema como usuário Root. Este usuário tem privilégios totais sobre o sistema. Um descuido pode ser fatal ao sistema, podendo ocasionar desde a perda de dados até a parada completa do sistema.

Atenção ao desligar o sistema

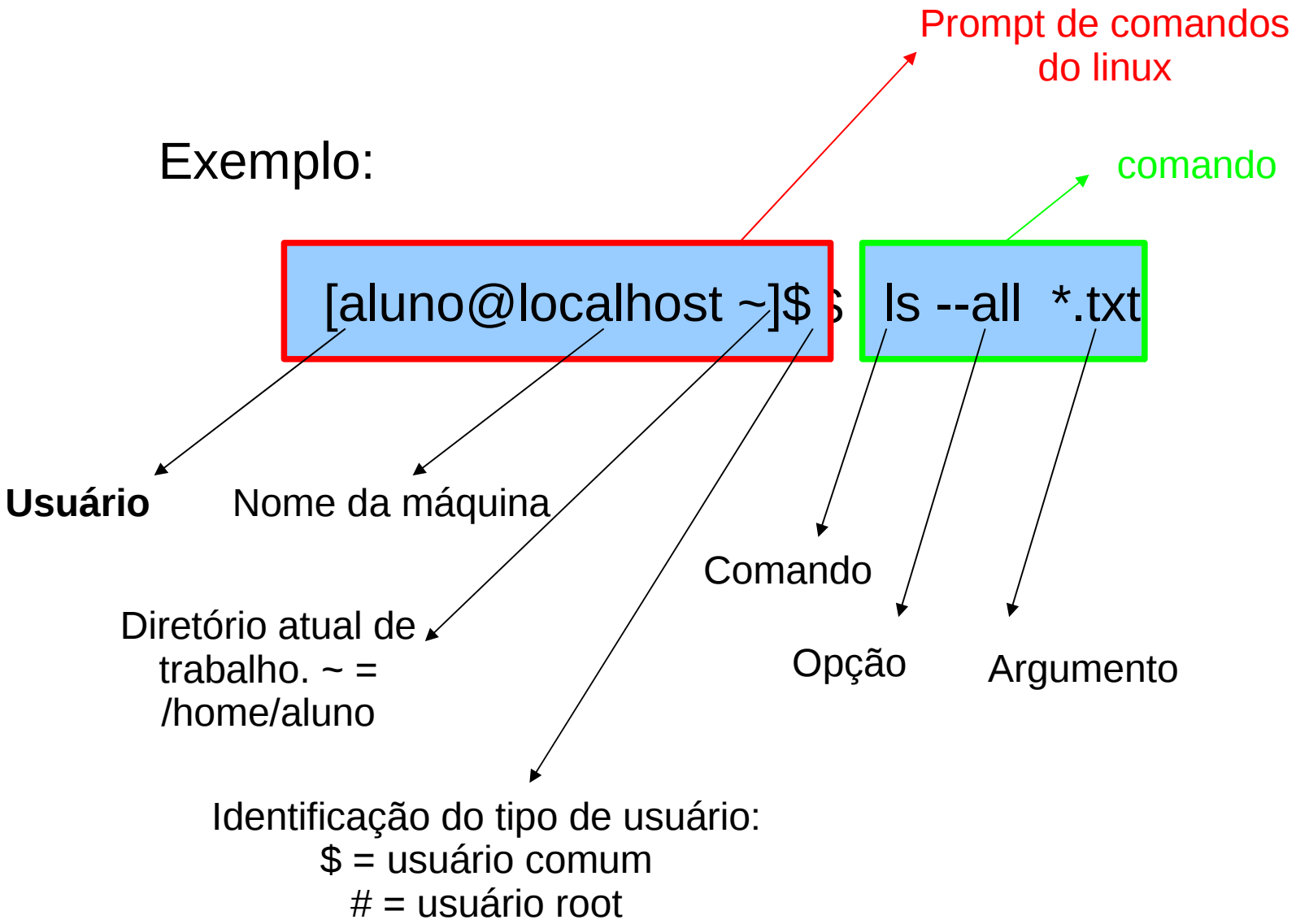
Nunca desligue o sistema diretamente pelos botões do painel frontal do computador. Isto poderá danificar o sistema de arquivos do Linux.

Comandos básicos

- ls => exibe uma lista de arquivos do diretório atual ou especificado;
- cd <diretório> => muda de diretório;
- passwd => muda a senha do usuário atual;
- file <arquivo> => mostra o tipo do arquivo;
- cat <arquivo texto> => exibe o conteúdo do arquivo texto;
- pwd => exibe o diretório de trabalho;
- exit / logout => sai da sessão;
- man <comando> => mostra as páginas manuais do comando;
- info <comando> => mostra as páginas Info do comando;
- apropos <string> => procura a <string> no banco de dados de 'whatis'.

Executando comandos

Os comandos são executados em um terminal, através de um Shell. O Shell padrão do GNU/Linux chama-se Bash.



Usando teclas especiais do Bash

- CTRL+A = move o cursor para o início da linha;
- CTRL+C = finaliza a executação de um programa;
- CTRL+D = igual a digitar exit ou logout no terminal;
- CTRL+E = move o cursos para o final da linha;
- CTRL+H = backspace;
- CTRL+L = limpa o terminal;
- CTRL+R = procura no histórico de comandos;
- CTRL+Z = suspende um programa;
- Setas p/ direita e p/ esquerda = movimenta o cursos na linha de comando;
- Setas p/ cima e p/ baixo = navega no histórico de comandos;
- Tab = completa nome de arquivos ou comandos;
- Tab Tab = mostra possibilidades de completamento de nomes de arquivos ou comandos.

Obtendo ajuda – páginas de manual

- Sintaxe:

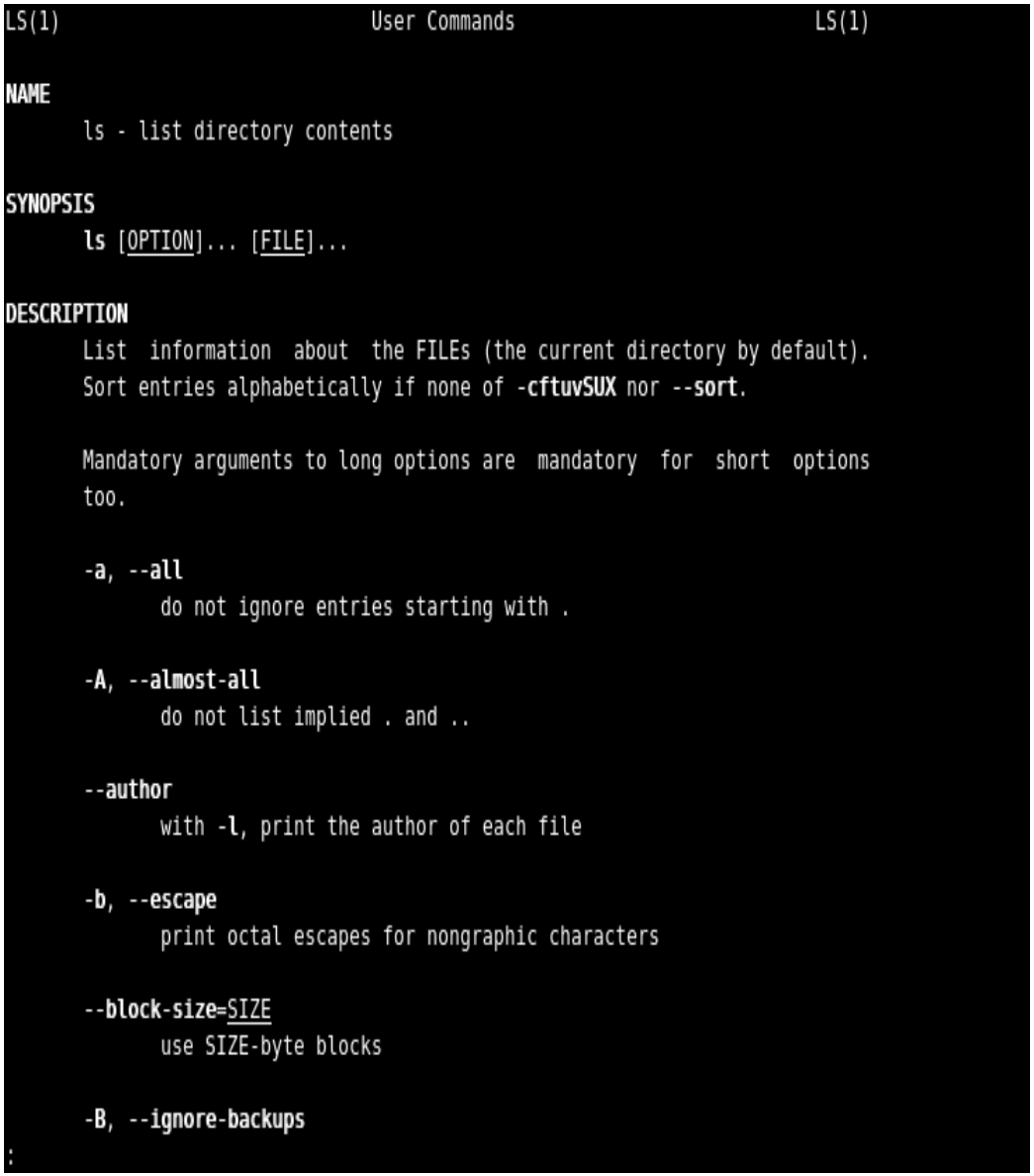
  man [seção] [comando]

  Ex.: man ls =>

- Acompanham quase todos os programas GNU/Linux;

- Mais informações sobre o comando man, digite:

man man



```
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort.

       Mandatory arguments to long options are  mandatory  for  short  options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print octal escapes for nongraphic characters

       --block-size=SIZE
              use SIZE-byte blocks

       -B, --ignore-backups
:
```

Obtendo ajuda – páginas Info

- Sintaxe:

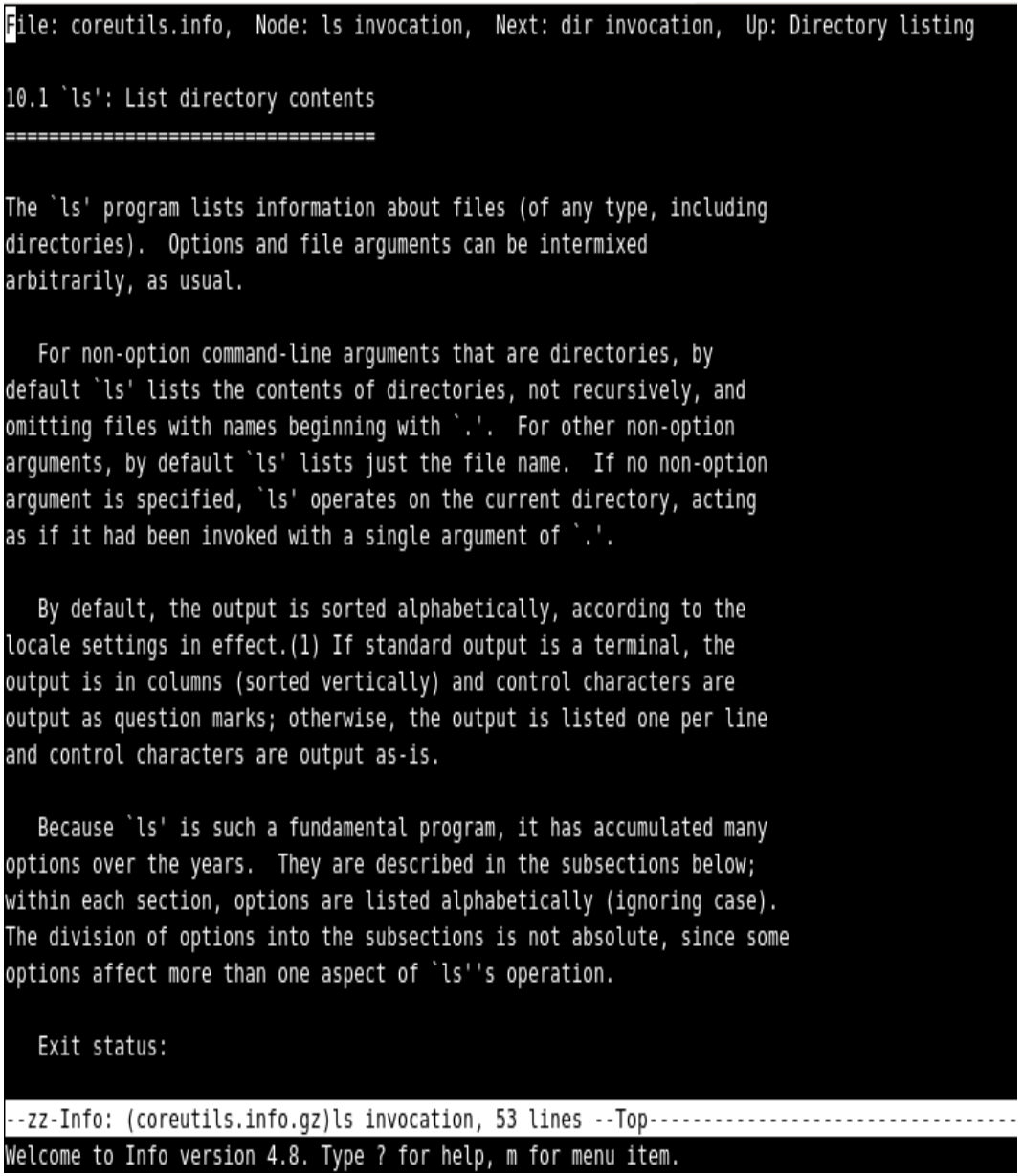
  info [comando]

  Ex.: info ls =>

- Idêntico às páginas de manual, mas permite navegação.

- Mais informações sobre o comando info, digite:

man info



Obtendo ajuda – ajuda on-line

- Sintaxe:

  [comando] --help

  Ex.: ls --help =>

- Ajuda rápida, útil para conhecermos as opções do comando.

```
[mateus@localhost ~]$ ls --help
Uso: ls [OPÇÃO]... [ARQUIVO]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort.

Argumentos obrigatórios para opções longas também o são para opções curtas
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
      --author               with -l, print the author of each file
  -b, --escape               print octal escapes for nongraphic characters
      --block-size=SIZE      use SIZE-byte blocks
  -B, --ignore-backups       do not list implied entries ending with ~
  -c                         with -lt: sort by, and show, ctime (time of last
                               modification of file status information)
                               with -l: show ctime and sort by name
                               otherwise: sort by ctime
  -C                         list entries by columns
      --color[=WHEN]         control whether color is used to distinguish file
                               types.  WHEN may be `never', `always', or `auto'
  -d, --directory            list directory entries instead of contents,
                               and do not dereference symbolic links
  -D, --dired                generate output designed for Emacs' dired mode
  -f                         do not sort, enable -aU, disable -lst
  -F, --classify             append indicator (one of */=>@|) to entries
      --file-type            likewise, except do not append `*'
      --format=WORD          across -x, commas -m, horizontal -x, long -l,
                               single-column -1, verbose -l, vertical -C
      --full-time            like -l --time-style=full-iso
  -g                         like -l, but do not list owner
  -G, --no-group             like -l, but do not list group
  -h, --human-readable       with -l, print sizes in human readable format
                               (e.g., 1K 234M 2G)
```

Obtendo ajuda – comandos internos

- Sintaxe:

  help [comando]

  Ex.: help cd =>

- Ajuda rápida, útil para conhecermos as opções do comando.

```
[mateus@localhost ~]$ help ls
bash: help: no help topics match `ls'.  Try `help help' or `man -k ls' or `info ls'.
[mateus@localhost ~]$ help cd
cd: cd [-L|-P] [dir]
    Change the current directory to DIR.  The variable $HOME is the
    default DIR.  The variable CDPATH defines the search path for
    the directory containing DIR.  Alternative directory names in CDPATH
    are separated by a colon (:).  A null directory name is the same as
    the current directory, i.e. `.'.  If DIR begins with a slash (/),
    then CDPATH is not used.  If the directory is not found, and the
    shell option `cdable_vars' is set, then try the word as a variable
    name.  If that variable has a value, then cd to the value of that
    variable.  The -P option says to use the physical directory structure
    instead of following symbolic links; the -L option forces symbolic links
    to be followed.
[mateus@localhost ~]$
```

Obtendo ajuda – apropos e whatis

- Sintaxe:

  apropos [string]

- Procura uma string no banco de dados 'whatis'.

```
[mateus@localhost ~]$ apropos ls
aa_imgheight         (3)  - returns height of the emulated image in pixels
aa_imgwidth          (3)  - returns width of the emulated image in pixels
acl_cmp              (3)  - compare two ACLs
acl_extended_file    (3)  - test for information in ACLs by file name
aconnect             (1)  - ALSA sequencer connection manager
afs_syscall [unimplemented] (2)  - unimplemented system calls
alsactl              (1)  - advanced controls for ALSA soundcard driver
```

- Sintaxe:

  whatis [palavra]

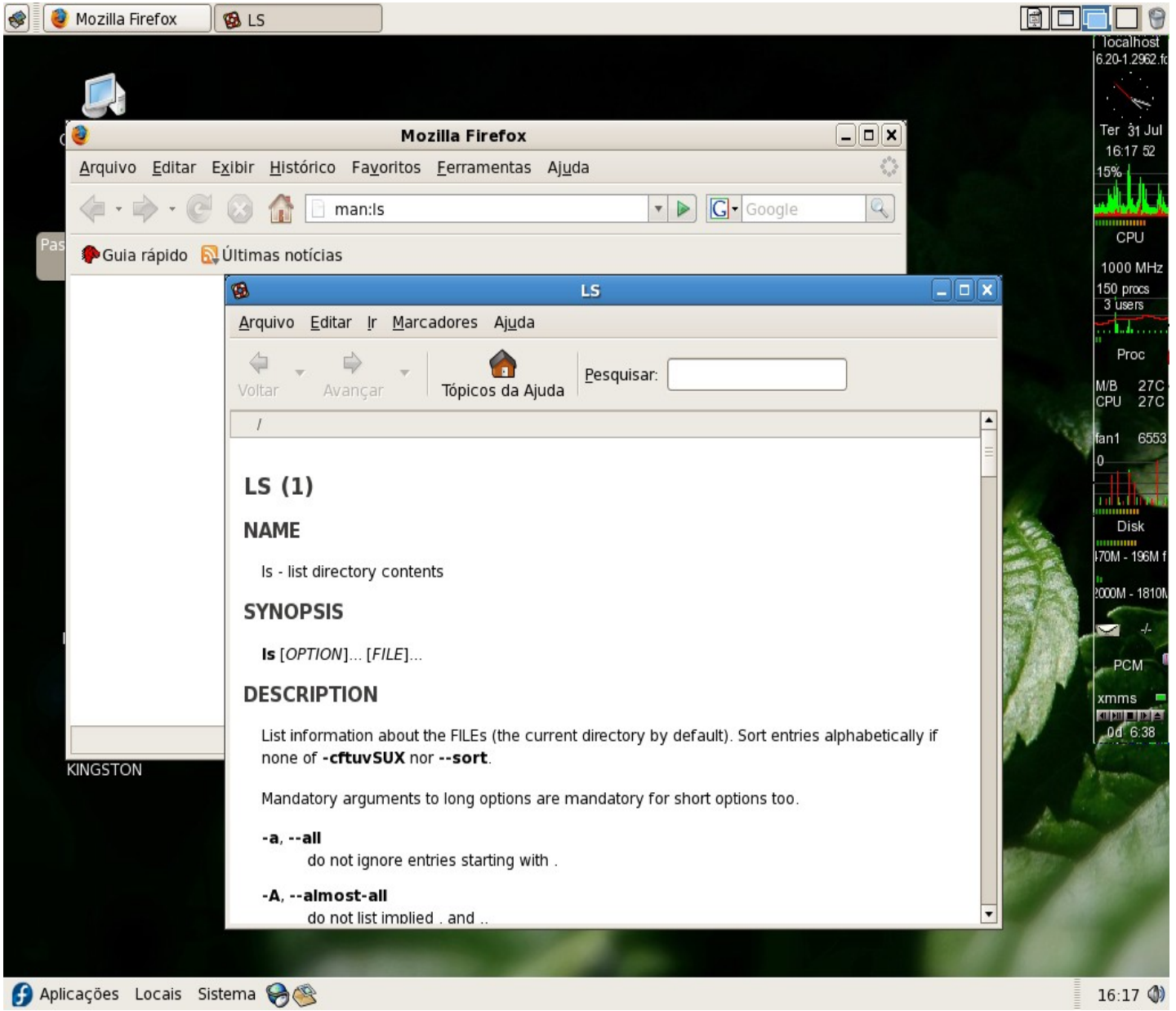
- Procura uma palavra num banco de dados próprio.

```
[mateus@localhost ~]$ whatis ls
ls                   (1)  - list directory contents
ls                   (1p)  - list directory contents
[mateus@localhost ~]$
```

Obtendo ajuda – ajuda gráfica

- Se você prefere o mundo gráfico, digite 'man:ls' ou 'info:ls' na barra de endereço do seu browser.

- Lembrando que alguns browser estão configurados para procurar na internet.



Exercícios

CONECTANDO E DESCONECTANDO

1. Independente do modo – gráfico ou texto – entre no sistema com o seu nome de usuário e a sua senha (login) ;

2. Saia do sistema (logout);

3. Tente acessar o sistema com um usuário não existente;

Exercícios

SENHAS (passwd)

1. Mude a sua senha para *aluno* e tecle Enter;

O que acontece?

2. Tente novamente, agora entre com uma senha muito fácil, como *123* ou *aaa*;

O que acontece?

3. Agora, não defina senha alguma e tecle Enter;

O que acontece?

4. Tente o comando **psswd** ao invés de **passwd**;

O que acontece?

Exercícios

DIRETÓRIOS

1. Entre com o comando ***cd blah***

O que acontece?

2. Entre com o comando ***cd ..***

Há um espaço entre 'cd' e '..'!
O que acontece?

3. Entre com o comando ***ls***

O que você vê?
O que você supõe ser esta informação?
Como saber qual diretório voce está no momento?

4. Entre com o comando ***cd ..***

O que acontece?
Exiba o conteúdo deste diretório.

5. Repita o passo mais uma vezes

O que acontece?

Exercícios

DIRETÓRIOS

5. Entre com o comando ***cd root***

O que acontece?
A quais diretórios você tem acesso?

6. Entre no diretório /etc/sysconfig

Como saber os subdiretórios que estão nessa página?

Exercícios

OBTENDO AJUDA

1. Leia ***man intro***

2. Leia ***man ls***

3. Leia ***info passwd***

4. Entre com o comando ***apropos pwd***

5. Tente ***man*** ou ***info*** no comando ***cd***

6. Leia ***ls --help*** e experimente.

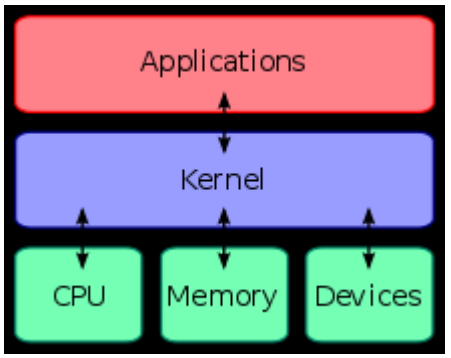
Estrutura Organizacional do Linux

Pode ser dividida em 4 partes:

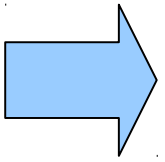
1) Kernel
2) Shell
3) Processos
4) Estrutura de arquivos

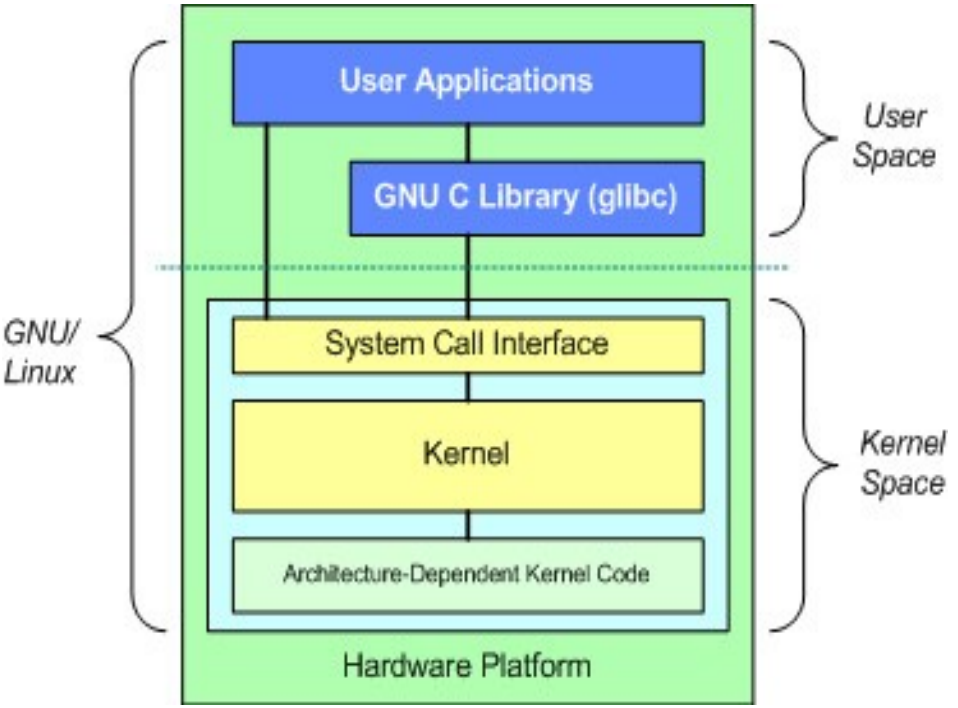
Kernel

- Base do sistema operacional
  - Escalonamento de processos
  - Gerenciamento de memória
  - Gerenciamento de dispositivos
  - Gerenciamento de arquivos
  - Interface de chamada do sistema
  - Interface de controle do operador

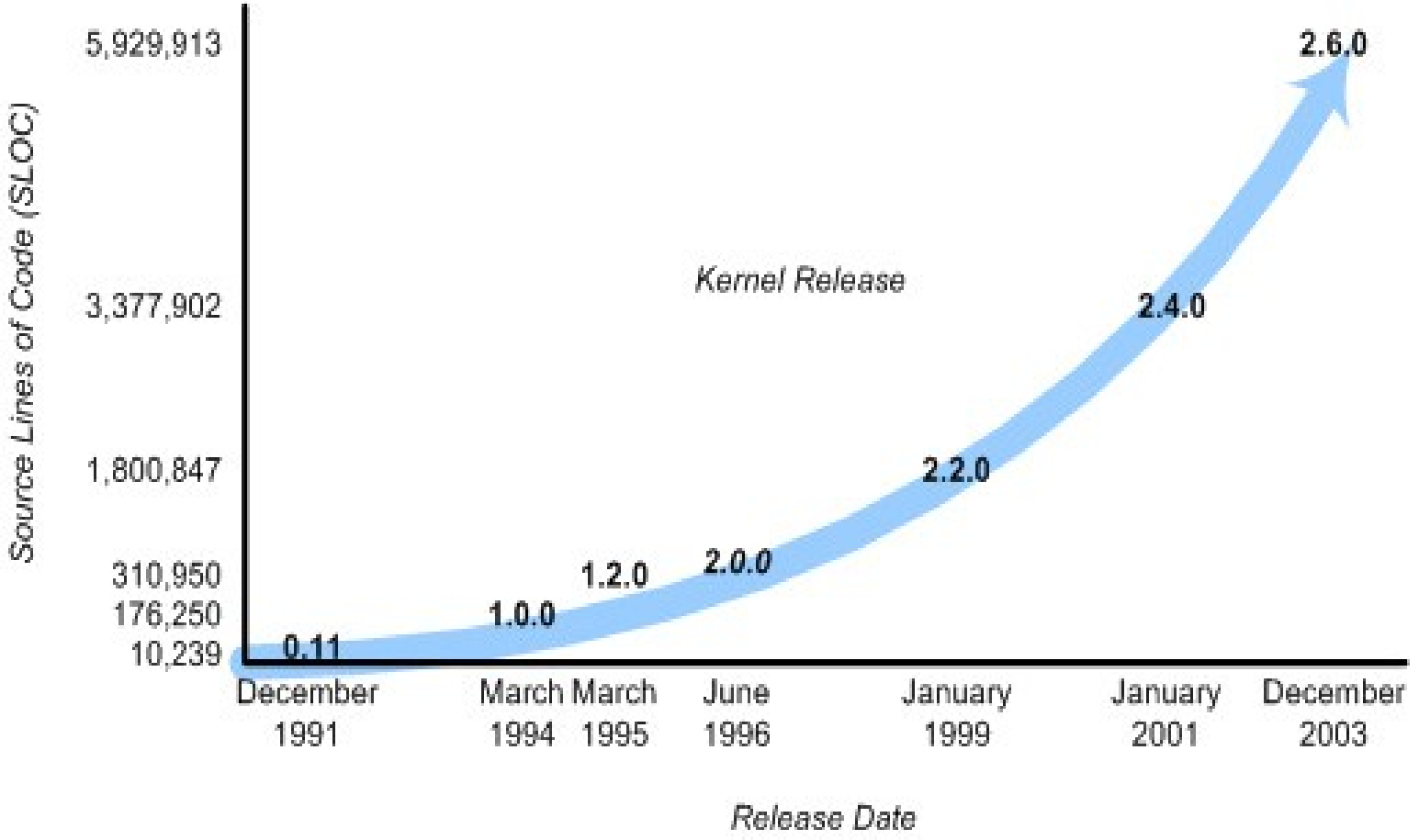


Arquitetura fundamental do Sistema Operacional GNU/Linux





Evolução do Kernel Linux



Tá, qual é o meu kernel??? Comando *uname*

Shell

- Programa que recebe, interpreta e executa os comandos do usuário.
- Mediador usuário-sistema

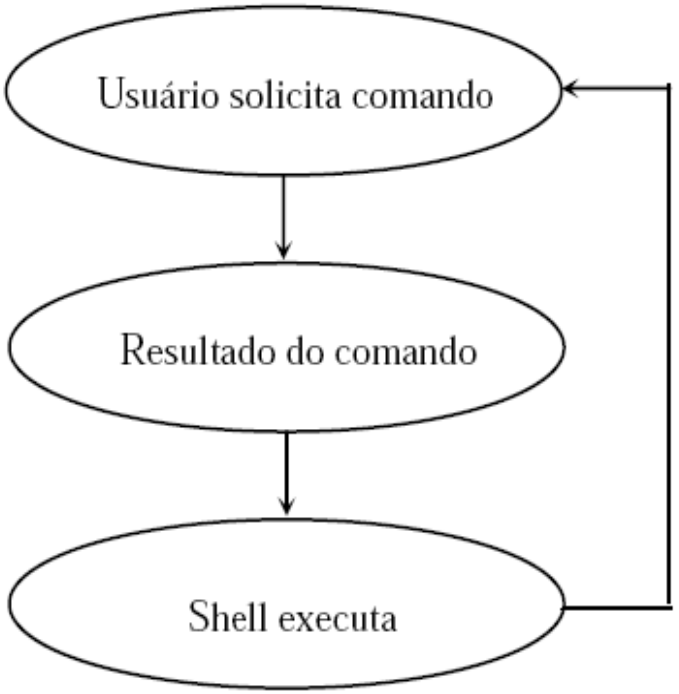


Figura 1: Interação Usuário - Shell

- Shells mais conhecidos (echo $0)
  - **sh**: Bourne Shell
  - **Bash**: Bourne-Again Shell
  - **ksh**: Korn Shell
  - **csh**: C Shell
  - **rsh**: Remote Shell
  - **Rsh**: Restricted Shell

Processos

- Programas em execução

- Podem ser executados em:
  - **Primeiro plano (foreground) -** enquanto o processo especificado na linha de comando não termina, o shell não libera o prompt para o usuário, impedindo o disparo de novos processos.
  - **Segundo plano (background) -** vários programas ou comandos podem ser executados, independente do término de cada um deles. Este modo de execução é especificado pelo símbolo & no final da linha de comando.

- Podem ser divididos em três grandes grupos:
  - Interativos
  - Batch, ou em lote
  - Daemons

Processos

- Interativos:

  iniciados e controlados a partir de uma sessão terminal

- Batch:

  não estão associados a nenhum terminal. (Normalmente associados com *Cron*)

  – são submetidos a uma fila

- Daemons

  processos servidores, iniciados no “boot” (Normalmente iniciados junto com o sistema – placa de rede, etc..)

  – rodam em “background”, esperando requisição do usuário.

Processos

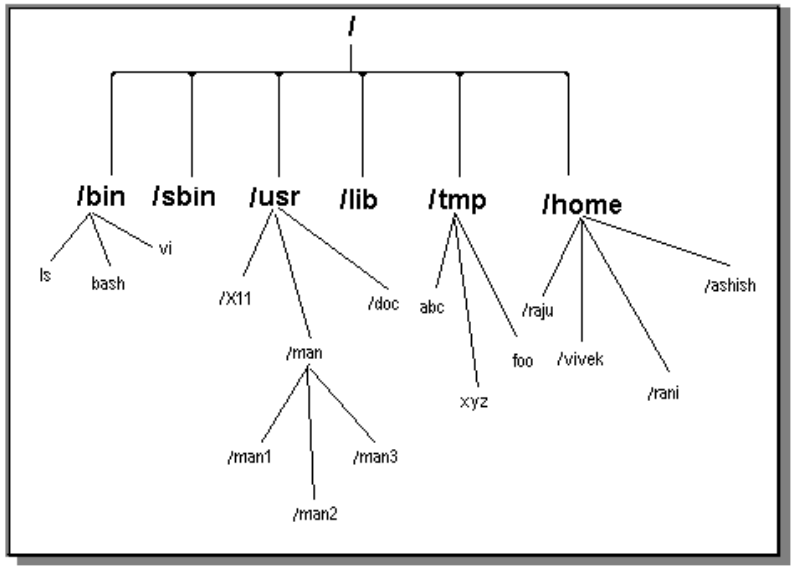
- Atributos
  - **Process ID (PID)**: número que identifica o processo (nro. único).
  - **Parent Process ID (PPID)**: é o processo pai de um processo.
  - **TTY**: terminal associado ao processo
  - **UID real (RUID)**: ID do usuário que criou o processo
  - **UID efetiva (EUID)**: ID do usuário que determina acesso aos recursos do processo.
  - **GID real e efetiva (RGID e EGID)**: grupo primário ou corrente do usuário.

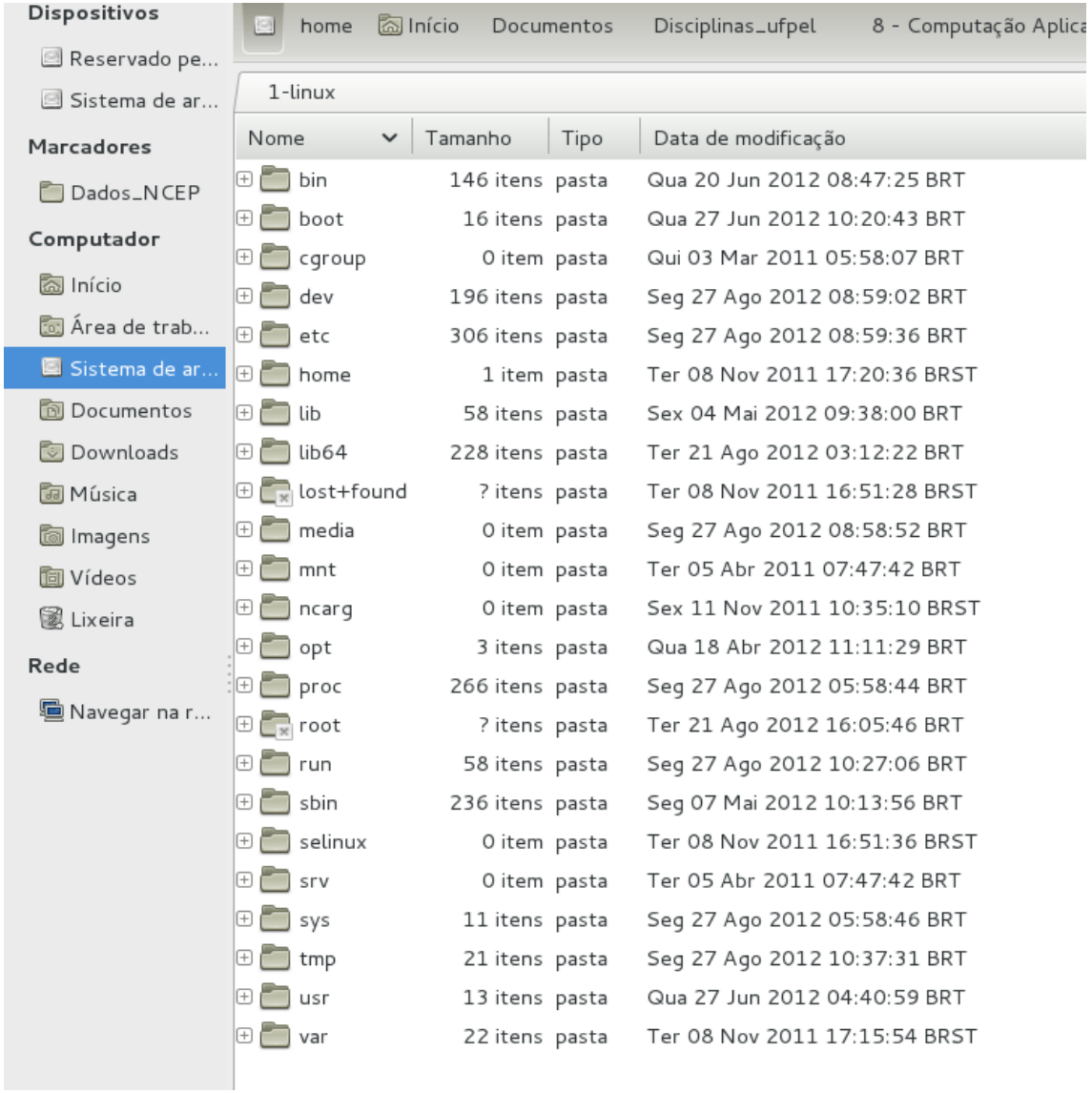
  Comando: **ps -ef**

Estrutura de Arquivos

- Arquivos são centrais ao Linux/UNIX
- Organizados em diretórios, em forma de árvore
- Acesso organizado por propriedades e permissões
- Há três tipos de arquivos:
  - Arquivos ordinários (comuns)
  - Arquivos diretórios
  - Arquivos especiais (p/ dispositivos)

Estrutura de Arquivos

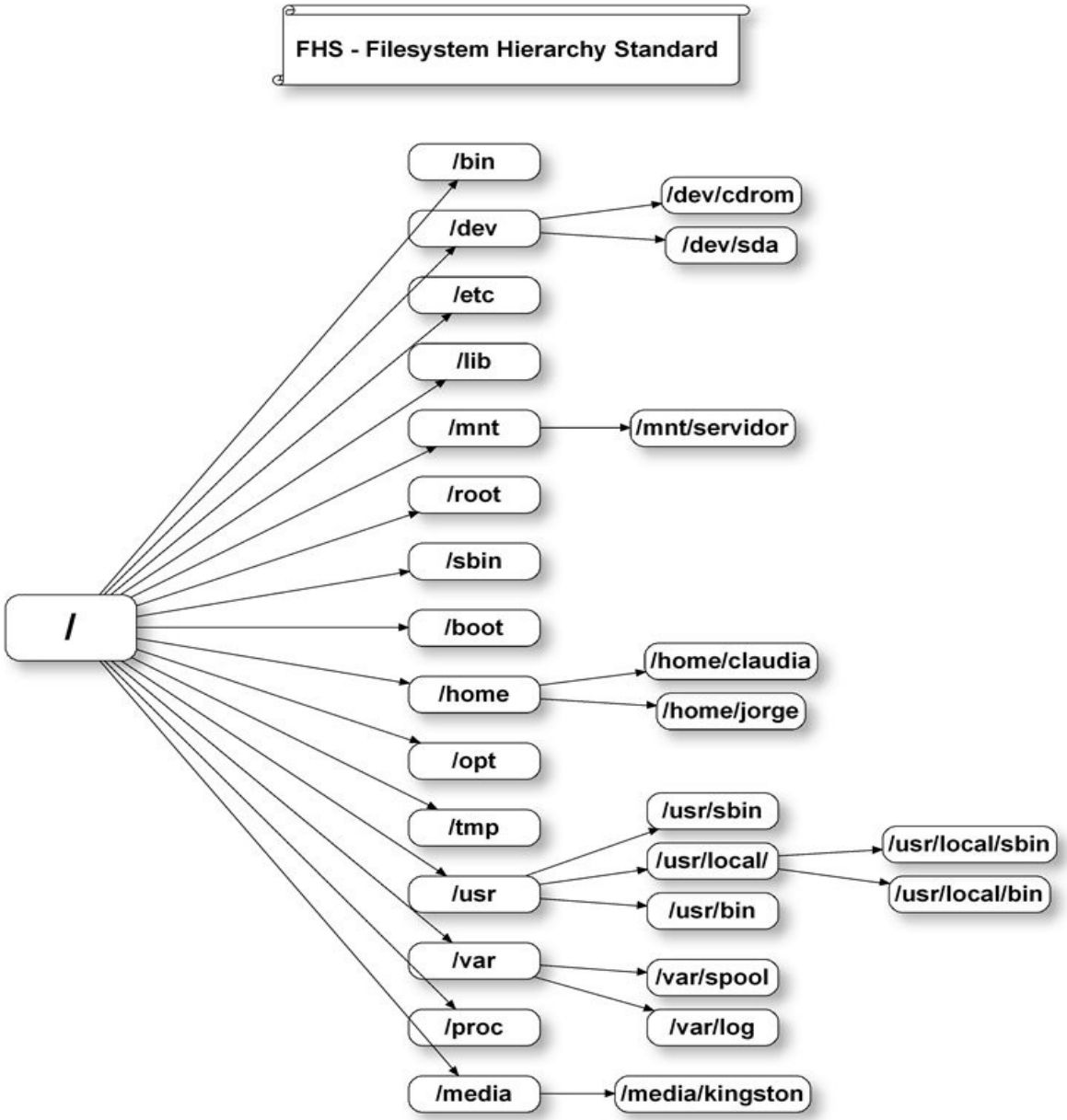




O sistema de arquivos do Linux

- Um sistema UNIX/Linux não faz diferença entre um diretório e um arquivo (o diretório é um arquivo contendo o nome de outros arquivos);
- Programas, serviços, textos e assim por diante são todos arquivos;
- Dispositivos de entrada e saída, e geralmente todos os dispositivos, são considerados como arquivos;
- É útil vermos a organização dos arquivos como uma árvore.

Sistema de arquivos



Sistema de arquivos

Os subdiretórios do diretório raiz (podem variar um pouco conforme distribuição/versão):

/bin	Arquivos binários, Programas comuns, compartilhados pelo sistema, administrador e usuários
/boot	Arquivos de boot (inicialização; boot-loader; Grub); kernel do Linux.
/dev	Dispositivos (devices) de entrada/saída: floppy, hardisk, cdrom, modem .
/etc	Arquivos de configuração (scripts) e inicialização.
/home	Diretórios pessoais (local) dos usuários
/lib	Bibliotecas e módulos(drives): compartilhadas com freqüência.
/lost+found	Cada partição tem um diretório destes. Aqui ficam os arquivos salvos durante falhas
/misc	Para propósitos diversos
/mnt	Diretório de montagem de dispositivos, sistemas de arquivos e partição.
/net	Ponto de montagem padrão para sistemas de arquivos remotos
/opt	Para instalação de programas não oficiais da distribuição (terceiros)
/proc	Diretório virtual (RAM) onde rodam os processos ativos.
/root	Diretório pessoal (local) do administrador do sistema
/sbin	Arquivos de sistema essenciais (binários do superusuário).
/tmp	Espaço temporário para o sistema gerados por alguns utilitários
/usr	Programas, bibliotecas, docs etc de programas dos usuários
/var	Armazenamento de todos os arquivos variáveis, logs, arquivos baixados da internet, etc.

Segurança de arquivos

- Permissões de arquivo: A PRIMEIRA LINHA DE DEFESA DO LINUX!

- Num sistema Linux, todo arquivo pertence a um usuário e a um grupo de usuários;

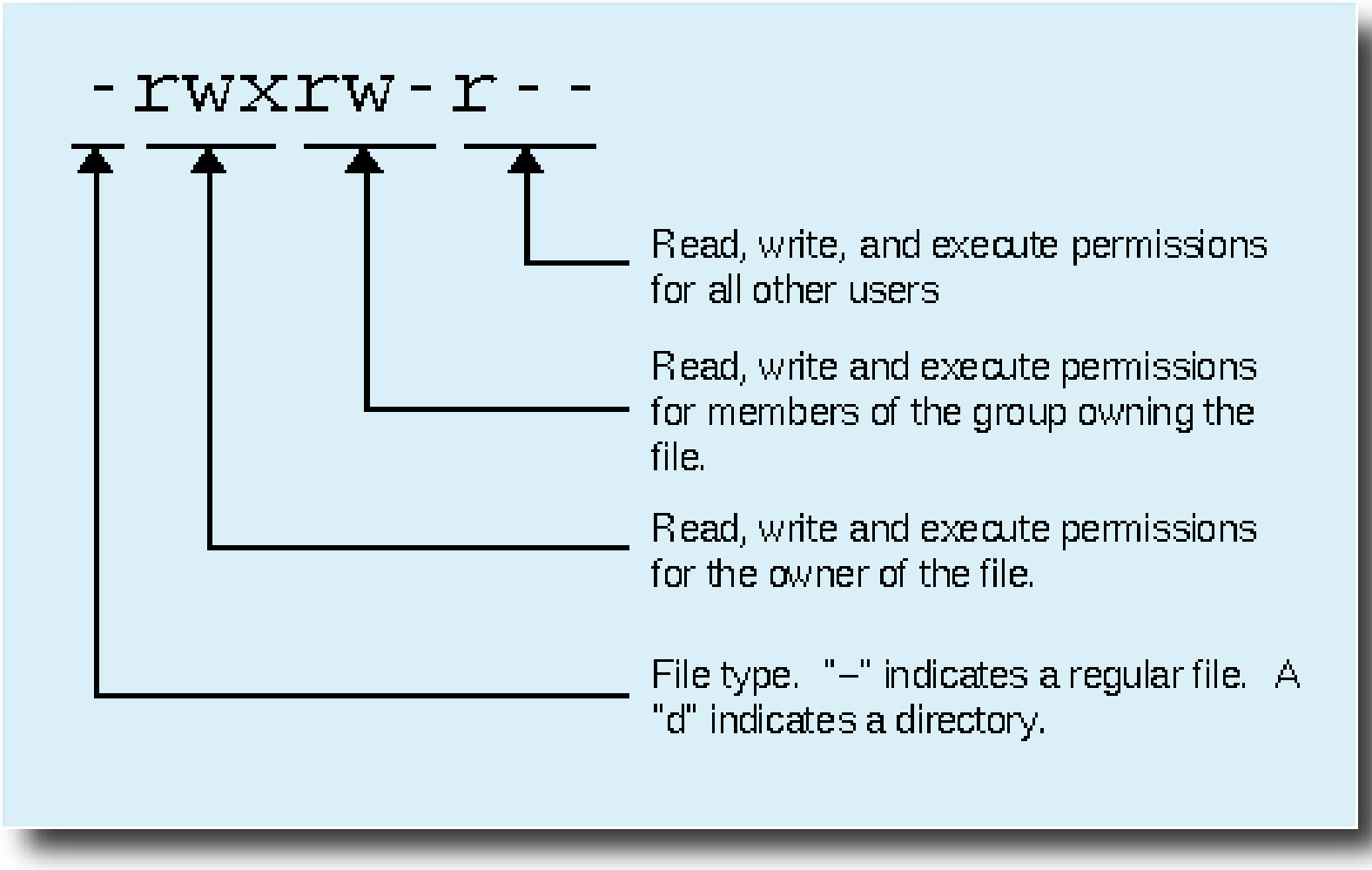
- Aqueles que não são donos e nem fazem parte do grupo de usuários donos do arquivo, são considerados como outros.

- **Donos, grupos de usuários e outros** têm seu acesso a arquivos definido pelas **permissões de arquivo.**

CUIDADO COM COMANDO rm -rf COMO ROOT

Segurança de arquivos (ls -l)

As permissões de arquivo:



Segurança de arquivos (ls -l)

- O comando ***chmod*** possibilita a mudança de permissões. Ex. (válido para **u, g** ou **o**):

```
chmod u+x .bash_history
chmod u-x .bash_history
```

- A alteração das permissões faz-se utilizando os seguintes sinais:

u -> dono do arquivo
g -> grupo ao qual o arquivo pertence
o -> outros usuários
\+ -> fornece a permissão especificada
\- -> retira a permissão especificada
r -> permissão de leitura
w -> permissão de escrita
x -> permissão de execução

Comando "groups"

- addgroup -delgroup
- groupadd -groupdel
- groupmod

Comando: id "usuario"

Segurança de arquivos

- A alteração das permissões também pode ser feita com argumentos numéricos (octal / binários):

Nível	u	g	o
Permissão	rwx	r-x	---
Binário	111	101	000
Octal	7	5	0

Para arquivos

rwx		
000	0	(zero) permissão negada
001	1	permissão de execução
010	2	permissão de gravação
011	3	permissão de gravação e execução
100	4	permissão de leitura
101	5	permissão de leitura e execução
110	6	permissão de leitura e gravação
111	7	soma de todas as permissões

Processos

“Depois dos arquivos, os processos são as coisas mais importantes em um sistema Linux/UNIX.”

Multi-usuário e multi-tarefa

Nem todo comando inicia um único processo. Alguns comandos iniciam uma série de processos, tal como o **firefox**; outros, como o **ls**, são executados como um único comando.

Num ambiente Linux/UNIX é comum:

- Usuários rodarem múltiplos comandos ao mesmo tempo e no mesmo sistema;
- Usuários desconectam-se do sistema mas seus processos continuam sendo executados;
- Reativação de processos interrompidos.

Tipos de processos

Processos interativos:

São controlados e inicializados por meio de um terminal, ou seja, alguém precisa iniciar estes processos;

Podem ser executados em primeiro e em segundo planos;

- Primeiro plano: o terminal fica indisponível ao usuário para outros comandos (ex.: gedit);
- Segundo plano: liberação do terminal para outras ações

Processos interativos:

Até agora só havíamos lidado com comandos em primeiro plano – apesar deles serem rápidos;

Enquanto um processo roda em segundo plano, o usuário não fica impedido de fazer outras coisas no terminal onde iniciou o programa;

O Shell oferece algo chamado como Controle de Tarefas que permite a fácil manipulação de múltiplos processos;

Este controle permite que programas sejam passados do primeiro para o segundo plano e vice-versa, inclusive serem iniciados diretamente em segundo plano;

Processos interativos:

Rodar processos em segundo plano é útil quando o mesmo não precisa que o usuário entre com informações;

Roda-se em segundo plano quando espera-se que o programa levará um longo tempo para terminar a tarefa;

Coloca-se um programa em segundo plano por meio do caractere &. Adicionando-o ao final do comando, ex:

» gedit &
» ls

Processos interativos:

Todas os mecanismos de controle de tarefas são explicados em detalhes nas páginas Info do **bash**;

Controlando processos:

(parte do) comando	**Significado**
comando_normal	Roda comando em primeiro plano
comando &	Roda comando em segundo plano
jobs	Mostra comandos no segundo plano
CTRL+Z	Suspende (para, mas não sai) programa
CTRL+C	Interrompe (para e sai) programe
%n	Referência ao processo *n*
bg	Reativa um programa suspenso em segundo plano
fg	Coloca um programa no primeiro plano
kill	Finaliza um processo

Processos automáticos:

Não são conectados a um terminal;

São tarefas que podem ser enfileirados, sendo executados em uma base FIFO (First-In, First-Out);

Podem ser executados de acordo com um de 2 critérios:

- Em uma certa data e hora, com o comando **at**
- Em horas que a carga total do sistema está baixa o suficiente para aceitar tarefas extras, com o comando **batch**. OBS.: Por padrão, tarefas são postas numa fila onde esperam para ser executadas quando a carga do sistema é menor que 80%.

Daemons (Disk And Execute MONitor)

São processos de servidor, que rodam continuamente;

Também são normalmente “processos-pai”

Na maioria das vezes eles são iniciados quando o sistema inicia e, então, esperam em segundo plano até serem chamados;

Exemplos: serviços de rede, servidores de e-mail, ftp e http etc.

Listar: ps -ef

Atributos dos processos

ID do processos (PID): número único que identifica o processos;

ID do processo pai (PPID): número do processo que iniciou o processo;

Nice number: o quanto o processo é amigável a outro processo;

Terminal (TTY): terminal ao qual o processo está conectado;

Nome de usuário dos usuários real e efetivos (RUID e EUID): o dono do processo

Obs.: o dono real é aquele que executa o comando e o usuário efetivo é aquele que determina o acesso aos recursos do sistema.

Atributos dos processos:

Quando o usuário inicia um programa (**gimp**), o próprio processo e todos os outros iniciados por este são de propriedade de *usuário* e não do administrador do sistema.

Quando **gimp** precisa acessar certos arquivos, este acesso será determinado pelas permissões do usuário e não do *root*.

Exibindo informações dos processos

O comando **ps** é um das ferramentas para visualizar processos. Ele possui diversas opções que podem ser combinadas para exibir diferentes atributos dos processos. Veja a documentação para maiores detalhes.

```
[mateus@localhost ~]$ ps
  PID TTY          TIME CMD
 3033 pts/0    00:00:00 bash
 3064 pts/0    00:00:00 ps
[mateus@localhost ~]$
```

O comando **ps** sem opções mostra apenas informações sobre o shell atual e seus processos -> **ps au**

Exibindo informações dos processos

Podemos usar o comando **grep** em um **pipe** (encadiamento de processos – ex.: “ls | grep bash”) para filtrar os resultados do comando **ps**:

```
[mateus@localhost ~]$ ps auxw |grep bash
mateus    3033  0.0  0.2   4876  1432 pts/0    Ss   00:49   0:00 bash
mateus    3234  0.0  0.1   4876   548 pts/0    R+   00:57   0:00 bash
[mateus@localhost ~]$
```

| => este operador será explicado em detalhes posteriormente

Exibindo informações dos processos

Note que o comando **ps**: dá apenas o estado momentâneo (instantâneo) dos processos ativos. O comando **top** (-d 5) dá uma visão mais precisa atualizando os resultados dados pelo comando **ps**

```
top - 01:08:15 up 35 min,  1 user,  load average: 0.68, 0.55, 0.46
Tasks: 107 total,   4 running, 101 sleeping,   0 stopped,   2 zombie
Cpu(s):  9.6%us,  1.9%sy,  0.0%ni, 83.8%id,  4.1%wa,  0.4%hi,  0.2%si,  0.0%st
Mem:    481516k total,   460724k used,    20792k free,     5896k buffers
Swap:  2048276k total,    24720k used,  2023556k free,   188440k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2922 mateus    15   0  246m 118m  46m R  5.9 25.3   1:20.84 soffice.bin
 3490 mateus    15   0  2196  916  712 R  3.9  0.2   0:00.02 top
 2942 mateus    15   0  133m  60m  25m R  2.0 12.9   0:16.57 acroread
    1 root      15   0  2040  640  552 S  0.0  0.1   0:00.63 init
    2 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    3 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    4 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    5 root      10  -5     0    0    0 S  0.0  0.0   0:00.02 events/0
    6 root      11  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
    7 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kthread
   56 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kblockd/0
   57 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
  168 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 cqueue/0
  169 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 ksuspend_usbd
  172 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 khubd
  174 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kseriod
  198 root      15   0     0    0    0 S  0.0  0.0   0:00.07 pdflush
```

Exibindo informações dos processos

A primeira linha da saída do comando **top** contém as informações exibidas pelo comando **uptime** (hora do sistema e tempo logado, qts usuários conectados e carga de trabalho nos últimos 1, 5 e 15min).

Exibindo informações dos processos

As relações entre os processos podem ser visualizadas por meio do comando **pstree**

Encerrando um processo

Um processo termina porque ele recebe um sinal. Existem vários sinais que podem ser enviados a um processo. Use o comando **kill** para enviar um sinal a um processo. O comando **kill -l** mostra uma lista de sinais. A maioria dos sinais são de uso interno do sistema. Como usuário, você precisará dos seguintes sinais:

Nome do sinal	**Nro. sinal**	**Significado**
SIGTERM	15	Termina o processo ordenadamente
SIGINT	2	Interrompe o processo (pode ser ignorado)
SIGKILL	9	Interrompe o processo (não pode ser ignorado)
SIGHUP	1	Para daemons: leia o arquivo de configuração.

Encerrando um processo

Suponha que o resultado do comando ps seja como mostrado abaixo.

PID	TTY	TIME	CMD
841	pts/0	00:00:00	bash
1314	pts/0	00:00:00	teste

Para finalizar o processo teste, basta digitar **kill 1314**

O comando acima corresponde a enviar o sinal SIGTERM (ou 15) ao processo.

Caso o processo não seja encerrado, você pode forçar o término do processo com o seguinte comando **kill -9 1314**

O comando acima corresponde a enviar o sinal SIGKILL (**9**) ao processo.

Esta opção informa ao sistema que o comando kill não pode ser ignorado, ele deve ser imediatamente processado. Neste caso, o sistema não se preocupa em salvar dados ou apagar arquivos temporários criados durante a execução do processo.

Tempo de execução

O Bash oferece um comando interno que exibe o tempo que um comando levou para ser executado. A cronometragem é bastante apurada e pode ser usada com qualquer comando:

```
[mateus@localhost ~]$ time ls
Artista Desconhecido  cd2ogg.pl   doutorado  GPL.txt  pagina-internet    treinos
casoCaragua           concurso    DOUTORADO  mm5      STP_purple.tracks
casoCaragua1967       Desktop     downloads  MP3      tese
casoCubatao1994       documentos  fotos      musicas  teste_mm5

real    0m0.012s
user    0m0.000s
sys     0m0.007s
[mateus@localhost ~]$
```

Há um outro comando, fornecido pelo Linux, que exibe maiores informações -> **/usr/bin/time**

Agendando processos

Existem 3 tipos de agendamento de processos:

- Esperando um pequeno tempo e, então, retomando a execução do trabalho, usando o comando **sleep**;
- Rodando um comando em um horário específico, usando o comando **at**;
- Rodando regularmente um comando, diariamente, semanalmente, horariamente etc, usando as facilidades do **cron**

Agendando processos

O comando **sleep**:

Ex.:

sleep 20; echo “Testando...”

sleep 100; gnome-terminal

Mais informações? Veja as páginas de manual do comando **sleep**.

Agendando processos

O comando **at:**

Ex.:

at now + 2 minutes
tar -cvf documentos.tar /home/glauber/Documentos
<CTRL>+<D>

at HH:MM MM/DD/YYYY
ls -ltra > testeat.txt
<CTRL>+<D>

- Para listar jobs agendados **atq**

- Para finalizar algum job agendado **atrm** ***#n***

Mais informações na documentação do comando **at**.

Agendando processos

Cron e **crontab:**

- É manipulado pelo daemon **cron** (carregado junto com o boot do sistema);
- Ele obtém as informações de quais programas e quando (somente uma vez ou repetidas vezes) eles devem ser executados, de tabelas do sistema e usuários;
- O cron usa uma espécie de tabela conhecida como crontab. O arquivo crontab geralmente fica localizado no diretório /etc, mas também pode estar em um diretório que cria um crontab para cada usuário do sistema (geralmente em /var/spool/cron/)

Agendando processos

1ª coluna = minutos (0-59);

2ª coluna = hora (0-23);

3º coluna = dia (1-31);

4ª coluna = mês (1-12 ou jan;feb.);

5ª coluna = semana (0-7). 0 e 7 são Domingo.

6ª coluna = usuário dono do processo.

7ª coluna = comando a ser executado

```
[mateus@localhost ~]$ more /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
[mateus@localhost ~]$
```

Asterisco = aceita todo o intervalo.

Para executar uma tarefa de Seg. a Sex. entre com 1-5 na última coluna, para executar um comando na Seg., na Qua. e na Sex. entre com 1,3,5.

Agendando processos

Comando	**Função**
crontab -e	Edita o crontab atual do usuário
	DICA: export VISUAL=gedit
crontab -l	Exibe o atual conteúdo do crontab do usuário
crontab -r	Remove o crontab do usuário

Existe a possibilidade de informar intervalos no preenchimento, separando os números de início e fim através de - (hífen).

Ex.: em **horas** 2-5, o comando será executado às 2, 3, 4 e 5 horas.
Ex2.: em **horas** 2,15-18,22 ?

#tarefa teste

*** 11 * * * tar -cvf backup.tar /home/glauber/Desktop**

Comandos Básicos vi

ESC Passa para o modo de comando

i Insere texto antes do cursor

o Adiciona linha abaixo da linha atual

Ctrl + h Apaga o ultimo caracter

x Deleta o caracter que esta sob o cursor

u Desfaz a ultima modificacao

:wq Salva o arquivo e sai do editor

:w nome_do_arquivo Salva o arquivo corrente com o nome especificado

:w! nome_do_arquivo Salva o arquivo corrente no arquivo especificado

:q Sai do editor

:q! Sai do editor sem salvar as alteracoes realizadas

Redireção de entrada e saída

A maioria dos comandos do Linux/UNIX lê uma entrada – um arquivo ou uma opção – e escrevem um resultado;

Por padrão, a entrada é o teclado e a saída é a tela do monitor;

Eles são chamados como *entrada (stdin)* e *saída (stdout) padrões*, respectivamente;

Entretanto, estes padrões podem ser modificados, por exemplo, mudando a saída padrão para uma impressora.

Operadores de redireção

Algumas vezes é útil passar a saída de um comando para um arquivo ou para um outro comando;

Este procedimento é conhecido como redireção da saída;

A redireção é feita com o símbolo “>” ou com o operador “|” (pipe) que manda a saída padrão de um comando para outro comando como entrada padrão;

Operadores de redireção

```
[mateus@localhost cursoLinux]$ cat teste1
algumas palavras
[mateus@localhost cursoLinux]$ cat teste2
algumas outras palavras
[mateus@localhost cursoLinux]$ cat teste1 teste2 > teste3
[mateus@localhost cursoLinux]$ cat teste3
algumas palavras
algumas outras palavras
[mateus@localhost cursoLinux]$
```

Operadores de redireção

CUIDADO: a redireção para arquivos já existentes causa a sobre-escrita. No shell Bash, a sobre-escrita de arquivos existentes pode ser evitada adicionando **set -o noclobber** ao arquivo *.bashrc*.

A redireção de nada é igual a esvaziar o arquivo:

```
[mateus@localhost cursoLinux]$ ls -l lista
-rw-rw-r-- 1 mateus mateus 17 Set  2 19:58 lista
[mateus@localhost cursoLinux]$ > lista
[mateus@localhost cursoLinux]$ ls -l lista
-rw-rw-r-- 1 mateus mateus 0 Set  2 19:58 lista
[mateus@localhost cursoLinux]$
```

A redireção de nada a um arquivo inexistente cria um arquivo novo e vazio.

Operadores de redireção

Alguns exemplos com o operador “|” (pipe):

ls *-l /etc/* | ***grep*** *tab*
Exibe uma lista de arquivos do diretório */etc*, cujos nomes contêm *“tab”* no nome.

ls *-la* | *less*
Exibe uma lista de arquivos uma página por vez.

ls | ***grep*** *init /etc/inittab* | ***grep*** *-v default*
Exibe todas as linhas que têm “init”, excluindo da saída as linhas que têm “default”.

Operadores de redireção

Exemplo de uso do operador “|” (pipe):

$ **ls** | **grep** b | **sort** -r | **tee** arquivo.out | **wc** -l

(1) O comando "ls" lista o conteúdo do diretório, porém, devido ao pipe ele não envia o resultado para tela e sim ao comando "grep b".
(2) O comando "grep b" por sua vez filtra os nomes de arquivos que contém a letra "b". Devido ao segundo pipe a saída do comando "grep b" é enviada para "sort -r", que classifica os nomes em ordem crescente.
(3) A saída do "sort -r " é então passada pra o comando "tee", que divide os dados em dois, como se fosse uma conexão em t, fazendo com que as informações processadas pelo comando "sort -r" sejam escritas no arquivo "arquivo.out".
(4) Então o comando "wc -l" conta (wc) as linhas (-l) do arquivo "arquivo.out". Assim obtemos como resultado a quantidade de arquivos que contém a letra "b" impresso na tela e o nome desses arquivos em "arquivo.out".

Operadores de redireção

Em outro caso, você pode querer que um arquivo seja a entrada para um comando que normalmente não aceitaria um arquivo como uma opção;

Esta redireção de entrada é feita com o operador “<”;

Abaixo está um exemplo:

mail *fulano@teste.com.br < meu_arquivo.txt*

Operadores de redireção

Ao invés de sobre-escrever um arquivos, você pode adicionar texto a um arquivo existente usando o operador “>>”:

```
[mateus@localhost cursoLinux]$ cat teste1
algumas palavras
[mateus@localhost cursoLinux]$ date >> teste1
[mateus@localhost cursoLinux]$ cat teste1
algumas palavras
Dom Set  2 20:34:52 BRT 2007
[mateus@localhost cursoLinux]$
```

Filtros

Quando um programa realiza operações com a sua entrada e escreve os resultados na saída padrão, ele é chamado de filtro.

Um pouco mais sobre o comando **grep**:

- Procura linhas que se igualam a um padrão;
- Este comportamento é revertido com a opção *-v*;
- Use a opção *-i* para tornar a procura não sensível à caixa de texto;
- A opção *–colour* ressalta o padrão procurado;
- Leia a documentação do comando **grep** para saber mais.

Filtros

O comando **sort** organiza as linhas em ordem alfabética (padrão):

cat ~/.bash_history | **sort**

Este comando pode ser usado em conjunto com o comando **uniq** (ou com a opção *-u*) para retirar resultados duplicados:

```
[mateus@localhost cursoLinux]$ cat numeros
1
2
3
1
2
5
[mateus@localhost cursoLinux]$ sort numeros | uniq
1
2
3
5
```

Shell Script

Shell – Normalmente conhecido como o “prompt” da linha de comando do Unix ou Linux e que recebe os comandos dos usuários do Linux em modo texto (terminal)

Ligação do usuário com o kernel do sistema que por sua vez acessa o hardware

Por que Shell Script?

O que é um script?

Script

Arquivo que guarda uma sequência de comandos e tarefas a serem executados. Indicado, normalmente, para automação de tarefas que devem ser repetidas.

Não possui estrutura rígida, oferecendo flexibilidade ao programador. Porém, deve-se possuir :

1) lógica correta: o script fará o que deve ser feito

2) controle de fluxo correto:

3) eficiência: deve-se usar da melhor forma possível os recursos do sistema (memória, disco ...)

Como escrever um script em shell

1) Usar um editor de texto ASCII (vi, gedit, nedit) para escrever o script

2) Fornecer permissão de execução ao arquivo que contém o script (chmod +x *arquivo*)

Para executar o arquivo (no terminal) - 2 opções:

$ bash <nome do arquivo>

$ ./ <nome do arquivo>

Onde ./ significa (no diretório atual)

Primeiro Script

Ex.:

- Um *root* quer guardar as informações de tempos em tempos dos usuários que estão logados e a ocupação em disco:

date

df

w

TAREFA: Descubra antes o que cada comando faz!

Primeiro Script

IMPORTANTE: TODO o script deverá conter inicialmente o comando:

#!/bin/bash

COMANDO: which bash

Esse comando servirá para que o sistema saiba que é o shell (bash) que irá executar os comandos seguintes

Com um editor de texto insira as linhas abaixo e salve com o nome de **testeshell**:

#!/bin/bash

date

df

w

echo Será que isso vai cair na prova?

Como executar um script shell?

- Permissão de execução ao script: **chmod +x testeshell**

- Executá-lo (se no diretório corrente): **./testeshell**

DICA:

echo $PATH

Arquivo .bashrc (visualizado no diretório /home/**usuario** com ls -a)

PATH=$PATH:/home/glauber/Documentos

export PATH

Variáveis

- Não há tipo de dados (float, real...)
- Nomes sensíveis à caixa de texto (semelhante ao linux)
- A variável é um nome simbólico à uma área da memória

-Para se **definir uma variável** basta usar “**=**” (sem espaços)

Ex.:

teste = 1, 2, 3

hoje=$(date)

- Para **visualizar o conteúdo** atual da variável “**echo $**”

Ex.:

echo $teste

echo Hoje é: $hoje

Variáveis

- Para **apagar** uma variável “**unset**”

Ex.:

unset $teste

- Todo comentário (no início ou no meio de uma linha) deve ser com # - EXCETO A PRIMEIRA LINHA

- Para saber as variáveis que o shell já usa por padrão (não podendo ser usada novamente):

env

Interação entre usuário e variáveis

- Comando **read**

Ex.:

```
#!/bin/bash
echo Qual eh o seu nome?
read meu_nome
echo Olah $meu_nome. O sistema estah pronto.
```

Escopo das variáveis

- As variáveis em shell não precisam ser declaradas, mas se tentar ler uma variável não declarada resultará em um string vazio

Não haverá avisos ou erros ao se executar o script!

Ex.:

1)

```
#!/bin/bash

echo MVAR eh: $MVAR

MVAR='Tudo bem'

echo MVAR eh: $MVAR
```

Colocar em um arquivo com o nome de mvar.sh

No primeiro caso ele não irá mostrar o valor de $MVAR

2) NO TERMINAL:

MVAR='Olah'

./mvar.sh

Escopos diferentes.

O script abre um novo shell

3)

export MVAR

./mvar.sh

Comando **export** permite visualização de variáveis por outros programas, incluindo shell

4)

echo $MVAR

Não exibe o último valor de MVAR alterado pela 3ªlinha do script

Quando o shell é terminado, o seu ambiente é destruído

5)

```
MVAR='Olah'
echo $MVAR
Olah
. ./mvar.sh
MVAR eh: Olah
MVAR eh: Tudo bem
echo $MVAR
Tudo bem
```

Para receber as mudanças do script é necessário usar o comando **source** (.) que não cria um outro shell para o rodar o script

6)

```
#!/bin/bash
echo Qual eh o seu nome?
read USUARIO
echo Olah $USUARIO
echo Serah criado um arquivo chamado $USUARIO_arquivo
touch $USUARIO_arquivo
```

Isto causará um erro, a menos que exista uma variável $USUARIO_arquivo. O shell não sabe onde o nome da variável termina e o resto começa

7)

```
#!/bin/bash
echo Qual eh o seu nome?
read USUARIO
echo Olah $USUARIO
echo Serah criado um arquivo chamado ${USUARIO}_arquivo
touch ${USUARIO}_arquivo
```

O Shell sabe, agora, onde termina o nome da variável

As aspas duplas são importantes na última linha pois mais de 01 arquivo pode ser criado, dependendo do nome digitado

8)

criar script

#!/bin/bash

cd ..

ls

Executar de 02 maneiras:

./script

source ./script

FIM