Uma introdução ao Linux

Laboratório de Informática I

Luís Rodrigo de O. Gonçalves

Sumário

II Part 2 - Comandos

Part 1 - Introdução

1. Introdução ao Linux

Este material tem por objetivo apresentar de forma prática e clara os principais conceitos e comandos relacionados ao *Linux*.

1.1 O Linux

Quando nos propomos a estudar o Linux, nos deparamos com muito mais do que comandos, serviços e arquivos de configuração. Estudar o Linux é entender como o sistema operacional funciona, como ele interage com os serviços e com os usuários, é conhecer um pouco da filosofia por de traz do sistema, é entender o seu surgimento e sua evolução até os dias atuais.

O Linux atual continua bem fiel as suas raízes, ao eternos espírito livre, corajoso e audacioso de seus fundadores, desenvolvedores e usuários.

Desde o seu surgimento humilde, por volta do ano de 1994, ele sempre teve como proposta ser um sistema operacional que pudesse evoluir e receber os aperfeiçoamentos selecionados pela sua comunidade de desenvolvedores e usuários, como um organismo vivo que cresce e se modifica de acordo com as necessidades das células que o compõem.

Além de trazer o poder do Unix e aproximar-lo dos usuários de micro computadores, o Linux influenciou e vem influenciando de forma ativa outros vários sistemas operacionais que já existiam quando do seu surgimento e baseado no próprio Linux vários outros foram desenvolvidos.

1.2 Histórico do Unix

Quando estudamos o Linux, precisamos conhecer um pouco da história do sistema operacional que inspirou tudo. O sistema operacional *Unix* foi criado em meados de 1969 nos Laboratórios da *BELL TELEPHONE (BTL)*, ele foi inicialmente escrito utilizando-se a linguagem de programação

Assembler, que lhe conferia maior velocidade, porém baixo nível de portabilidade; desta forma, com o surgimento e popularização da linguagem de programação "c" o seu *Kernel* foi reescrito, permitindo que o mesmo fosse portado para várias plataforma de *hardware*;

O Unix, fora inicialmente desenvolvido para computadores de grande porte, *mainframes*, mas atualmente suas variações já estão sendo largamente utilizada em microcomputadores, em celulares e até mesmo em vários tipos de sistemas embarcados.

Um Sistema *Unix* é composto basicamente de uma coleção de aplicativos para os usuários, bibliotecas, utilitários e do próprio ambiente operacional que é o responsável por fazer a interface entre o *hardware* e o usuário final.

Como mencionado anteriormente, em meados 1994 iniciou-se o desenvolvimento de um sistema operacionais baseado no *UNIX* porém que "rodava" em microcomputadores, na época em máquinas da arquitetura ***i386***, este sistema ficaria conhecido posteriormente como *Linux* e sendo distribuído até hoje de forma gratuita.

Desde o seu surgimento, o *Unix* tem ganho cada vez mais espaço, não somente na comunidade acadêmica, como também no mercado. No final da década de 80 até meados da década de 90 ele perdeu parte de seu espaço de atuação para os sistemas operacionais da *Microsoft*, mas com o advento do *Linux* este processo tem se invertido, e atualmente ele é um dos sistema operacionais mais utilizados.

1.3 Características do Unix

O *UNIX*, inicialmente, era um sistema operacional voltado para ambientes robustos, os quais geravam um elevado nível de serviço, os quais deveriam ser atendidos com alto nível de qualidade. Com o surgimento do *Linux*, assim como de outros sistemas da família *BSD* e com seu contínuo porte para arquiteturas derivadas da *i386*, o Unix, continua atendendo as mesmas necessidades.

Dentre as principais características e funcionalidades, da família de sistemas operacionais baseados na arquitetura do Unix, podemos destacar:

- **Funcionalidade**: capacidade de se adaptar a novas necessidades à medida que elas são desenvolvidas.
- **Performance**: estes sistemas são considerados como os sistemas operacionais mais rápidos do mercado, mesmo para ambiente de pequeno porte.
- **Portabilidade**: eles suportam vários tipos de *hardware* e geralmente oferecem isolamento da camada de dependência, o que permite que somente a parte dependente do *hardware* precise ser portada para as várias plataformas
- **Qualidade/Disponibilidade**: eles oferecem um processo de melhoramento constante de sua estrutura, de forma a garantir o aumento da qualidade dos serviços oferecidos; eles fornecem maior resiliencia através de soluções de *Cluster* e Alta Disponibilidade.
- ***Networking***: todos estes sistemas dão suporte, de forma nativa à uma vasta gama de

protocolos dos quais podemos destacar: *TCP/IP*, *ATM*, *FDDI* dentre outros; Assim como uma vasta gama de serviços de rede, tais como: *NFS* (*Network File System*), *NIS* (*Network Information Service*), *E-mail* (*IMAP*, *POP3*, *SMTP*), *HTTP*(S), (S)*FTP*, acesso remoto e outros. Além disto estes sistemas suportam vários tipo de de interfaces e podem utilizar vários meios de acesso.

- ☑ Ainda pode-se destacar outras características importantes, tais como: a segurança, a estabilidade, a capacidade Multitarefa & Multiusuário, possibilidade de permitir a emulação de terminais, o acesso remoto, a criação de *Cluster* e possui um padrão de regulamentação denominado "*Posix*".

1.4 Shell x Kernel

O sistema operacional *UNIX* é dividido basicamente em duas camadas: *kernel* e *shell*. O *Kernel* é o responsável pela interface entre o *hardware* e as aplicações, sendo, desta forma, considerado como o "núcleo central" do sistema operacional. Acima deste é executado o *shell* que realiza a interface entre os aplicativos e os usuários. Atualmente, há uma série de aplicações *shell* disponíveis, porém, a mais utilizada, no *Linux*, é o *BASH*.

Devido a arquitetura do *Unix*, e do *Linux*, um aplicativo pode rodar em dois modos: ou em **modo *kernel***, no qual ele tem acesso direto aos dispositivos; ou em **modo usuário**, no qual ele pede serviços à camada do *kernel*.

1.5 Processo de Instalação do Linux

O processo de instalação do *Linux* pode ser dividido em vários passos, dos quais podemos destacar os seguintes:

- ☑ Inventário do *Hardware*;
- ☑ Alocação espaço em disco;
- ☑ Criação e ativação dos sistemas de arquivos;
- ☑ Seleção dos pacotes;
- ☑ Processo de instalação dos pacotes e do *kernel*;
- ☑ Configuração do gerenciador de *Boot* (*GRUB*).

1.5.1 Inventário

Nesta etapa, o *Linux* irá realizar uma busca pelos *hardwares* instalados na máquina, bem como as suas configurações. Geralmente toda a placa ou dispositivo possuem associado à eles alguns valores que definem a sua configuração. Dentro destes valores, os mais importantes são o número de *IRQ*, de *I/O* e de *DMA*.

1.5.2 Alocando espaço em disco

Uma unidade de disco pode ser dividida em vários pedaços, denominados **partições de disco**. Uma partição pode ser primária ou secundária; dependendo do seu tipo, e da sua posição no disco ela recebe uma nomenclatura.

O **Disco 01** e composto de 4 partições primárias, que normalmente são utilizadas para a instalação de sistemas operacionais, já o **Disco 02** possui uma partição primária e três secundárias. Sendo que, um disco rígido pode conter quantas partições secundárias couberem no mesmo, e não somente três como ocorre em nosso exemplo.

Analisando os discos percebemos que ambos possuem a mesma quantidade de partições, mas a nomenclatura muda; o motivo da alteração da nomenclatura está relacionado à regra utilizada para nomear as partições, a seguir temos uma breve explicação acerca desta regra.

- Se o disco rígido, ainda, for do tipo ***IDE***, seu nome começa com as letras **hd**. Quando disco é do tipo ***SCSI*** ou do tipo ***SATA*** seu nome começa com as letras **sd**;
- De acordo com a posição do disco ele assume uma letra, de **a** até **z**; como na tabela abaixo:
  - 1º Disco **hda** **sda**
  - 2º Disco **hdb** **sdb**
  - 3º Disco **hdc** **sdc**
  - 4º Disco **hdd** **sdd**
- De acordo com a posição da partição, dentro do disco, ela assume um número, ou seja, se é uma das partições primárias este número vai de **1** até **4**, porém, quando estamos trabalhando com uma partição secundária este número deve ser maior ou igual a **5**.

Podemos utilizar várias ferramentas para nos ajudar a gerenciar as partições de um disco, dentre elas podemos destacar o "**fdisk**", que, mesmo não sendo uma aplicação simples sempre pode ser encontrada nas distribuições de Linux.

Uma outra ferramenta disponível é o "**fips**" cujo objetivo é o auxiliar no redimensionamento do disco, ou seja, alteração do tamanho das partições existentes.

Além destas, ainda podemos utilizar outras como por exemplo o (i) **cfdisk** e o (ii) **gparted**.

OBS: Adicionar mais informações sobre o cfdisk e o gparted

Além de particionarmos o disco ainda é necessário que as partições sejam "formatada". Quando formatamos uma determinada partição do disco, estamos na realidade associando à esta partição um determinado sistema de arquivo.

A seguir estão listados alguns dos sistemas de arquivos mais utilizados no *Windows* e no *Linux*

- **DOS/Windows**
  - FAT16
  - FAT32
  - VFAT
  - NTFS
- **Linux**

- ☑ Ext2
- ☑ Ext3
- ☑ Ext4
- ☑ Reiserfs
- ☑ Linux Swap

Quando instalado o *Linux*, podemos utilizar várias partições, cada uma com um uso específico. Mas independentemente da quantidade de partições utilizadas, duas partições são indispensáveis:

- **/**: A partição raiz é aquela na qual todos os pacotes serão instalados
- **swap**: Ao contrário do Windows o Linux não trabalha, por padrão, com arquivo de swap, mas sim como uma partição inteira dedicada para esta finalidade. Geralmente a capacidade, em Gbytes, partição de swap é o dobro da quantidade de memória *RAM*.

1.5.3 Seleção de pacotes

Nesta parte do processo de instalação devemos selecionar quais aplicativos devem ser instalados; de acordo com o tipo de instalação a ser realizada, existe um conjunto de pacotes padrão. A maioria das distribuições já possuem algumas pré-seleções de pacotes que o usuário pode escolher para facilitar a sua tarefa de instalação, evitando assim uma seleção manual.

Feita a seleção de pacotes, o processo de instalação cópia todos os pacotes para a partição correta e as configurações dos serviços são realizadas de forma quase que automática. Apos o processo de cópia e configuração dos serviços, fica ao encargo do usuário apenas mais alguns detalhes, que variam de distribuição para distribuição.

OBS: explicar um pouco mais

1.5.4 Configuração do Gerenciador de *Boot*

Terminada a fase de instalação, precisamos definir como o *Linux* será iniciado e caso haja outros sistemas operacionais instalados, como estes também serão carregados. Além disto devemos determinar qual sistema operacional deverá ser carregando caso o usuário não selecione um, de forma explicita, ou seja qual será o sistema operacional *default*, assim como, quanto tempo o sistema deve aguardar entes de carregar o sistema *default*.

Para resolver as questões apresentadas anteriormente, a maioria das distribuições *Linux* oferecem um aplicativo denominado gerenciador de *boot*, que permite escolher qual sistemas será iniciado, bem como os parâmetros de inicialização; inclusive definir uma senha que deverá ser fornecida para que o sistema seja carregado, impedindo que o mesmo seja utilizado por um usuário não autorizado.

O gerenciador de *boot* mais utilizado, atualmente, é o "*Grub*", ele possui suporte à interface gráfica, que pode ser customizada de acordo com a necessidade do usuário.

OBS : adicionar mais informações sobre o GRUB

1.6 Estrutura hierárquica dos diretórios

Os diretório no *Linux* são organizados na forma de arvore, só que de cabeça para baixo, ou seja, o primeiro diretório é o '**/**', que recebe o nome de diretório **raiz** ele recebe este nome pois é o primeiro elemento da arvores. Abaixo do diretório raiz, há vários outros tais como: '*bin*', '*sbin*', '*etc*', '*usr*' e etc. Na figura 1.1 temos a representação do primeiro nível da estrutura de diretórios de uma máquina com o ***Ubuntu Linux* 18.04**.

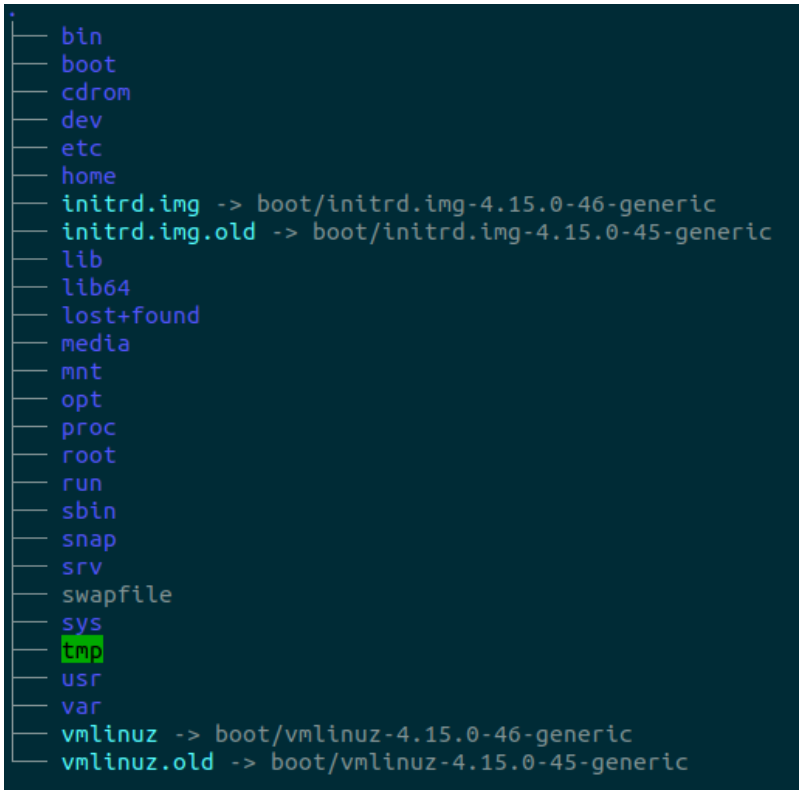

Figura 1.1: Estrutura de Diretórios - *Ubuntu Linux* 18.04

O motivo do uso de uma estrutura em arvore, é que ela fornece apenas um caminho desde a raiz até um determinado ponto da estrutura, o que evita o a existência de um *loop* infinito.

Cada diretório da estrutura utilizada pleo *Linux* possui um uso e significado específico; dos vários diretórios existentes na estrutura *default* do *Linux* podemos destacar:

- **/bin**: armazena os principais comandos utilitários; geralmente os comandos encontrados neste diretório podem ser executados por qualquer usuário registrado;
- **/dev**: contem os dispositivos suportados pelo sistema, tais como: partições de disco, unidade de *DVD*, *modem*, etc;
- **/etc**: contem os arquivos de configuração da maioria das aplicações instaladas;
- **/lib**: armazena as bibliotecas e funções utilizadas por outros aplicativos;
- **/tmp**: hospeda os arquivos temporários gerados pelos aplicativos que estão em execução.

- **/home**: este contem os vários diretórios que guardam os dados pessoais dos usuários cadastrados;
- **/usr/bin**: contem vários utilitários que podem ser utilizados pelos usuários cadastrados no sistema;
- **/usr/lib**: hospeda as principais bibliotecas (*libs*) utilizadas pelo sistema;
- **/usr/spool**: contem, por exemplo os *spools* de *e-mail* e impressora.

1.6.1 Path Absoluto e Relativo

O *PATH* é o caminho que descreve a localização de um objeto, seja ele um arquivo ou um diretório, dentro do sistema de arquivo, por exemplo, o *PATH* "**/home/lrodrigo/aula3.txt**", representa o caminho desde a raiz até o arquivo **aula3/txt**

Existem dois tipos de *path*: o *path* absoluto e o ***path*** relativo. O *path* absoluto é todo e qualquer caminho desde a raiz até o objeto desejado (**/home/lrodrigo/aula3.txt**)

Já o *path* relativo, é qualquer caminho que parte de um determinado ponto da árvore, que não seja a raiz; geralmente este ponto é o diretório corrente. Supondo que atualmente estamos no diretório "**/home**" um *path* relativo, válido, poderia ser "**lrodrigo/aula3.txt**"

1.7 Usuários

No Linux, existem dois tipos básicos de usuários, os **Usuários comuns**, que podem utilizar os recursos do sistema dentro das limitações estabelecidas para eles, através da estrutura de proteção de arquivos e da concessão de permissões; e os **Superusuários ou Administradores do Sistema**, que, normalmente, representam um grupo reduzido de indivíduos, os quais podem acessar todo o sistema de maneira irrestrita, não estando sujeitos ao sistema de proteção de arquivos. O uso das contas administrativas devem ser reservadas exclusivamente para as atividades de Administração do Sistema.

1.7.1 Ambiente do Usuário

Para um usuário acessar o sistema é necessário que o mesmo possua uma conta ativa. O processo de cadastro dos usuários somente pode ser realizado por um dos Administrador do Sistema. Sendo que, para adicionar uma nova conta é necessário obter as seguintes informações :

- ☑ Nome completo do usuário e sugestão para o nome de *login*
- ☑ Descrição de suas atividades e o relacionamento com outros usuários, para determinar o(s) grupo(s) ao(s) qual(is) fará parte.
- ☑ Estimativa do espaço em disco necessário para armazenamento dos seus dados.

Com base nestas informações o Administrador do Sistema definirá os seguintes atributos para a conta dos usuário:

- ☑ *User name (login)*;
- ☑ Senha de acesso (*passwd*);

- Identificação numérica do usuário (*uid*);
- Identificação numérica do grupo principal (*gid*);
- Diretório de trabalho (diretório *home*);
- *Shell* do usuário (normalmente **/bin/bash**).

Estas informações são armazenadas no diretório **/etc**, nos arquivos :

- /etc/passwd - Informações aos usuários
- /etc/group - Informações referentes aos grupos e seus membros

OBS: O usuário com **uid** 0 é chamado de **root** ou superusuário e os usuários com *gid* 0 fazem parte do grupo de Administradores do Sistema, que possuem privilégios especiais não fornecidos à outros grupos. Logos, todos os usuários devem ser cadastrados em outros grupos e o sistema deve possuir um único usuário com *uid* 0 e um grupo com *gid* 0

1.8 Variáveis de Ambiente

O sistema mantém um grupo de variáveis que utiliza para definir o ambiente de trabalho do usuário; por convenção, estas variáveis tem seus nomes em letras maiúsculas. Algumas destas variáveis são listadas abaixo.

- ***PATH*** : armazenas a lista de diretórios que o shell do usuário utiliza para localizar um comando:

  Exemplo 1.8.1 PATH="/sbin:/bin:/usr/bin:/usr/local/bin:/snap/bin"

- ***HOME*** : Guarda caminho absoluto do diretório de trabalho original do usuário (homedir);

  Exemplo 1.8.2 HOME="/home/luisrodrigoog"

- ***LOGNAME*** : Guarda a identificação do usuário utilizada no processo de login;

  Exemplo 1.8.3 LOGNAME="luisrodrigoog"

- ***TERM*** : Guarda o nome do tipo de terminal utilizado pelo usuário;

  Exemplo 1.8.4 TERM="xterm"

- ***PS1*** – determina como será o *prompt* fornecido pelo *shell* ao usuário;

  Exemplo 1.8.5
  ```
  PS1="\\[\\e[0;32m\\]\\u\\[\\e[m\\]@\\[\\e[1;33m\\]
  \\h\\[\\e[m\\]:\\[\\e[1;34m\\]\\W\\[\\e[m\\]\\|
  [\\e[0;37m\\]\\\\\$\\[\\e[m\\] "
  ```

  Com a variável ***PS1*** setada para o valor apresentado anteriormente, devemos obter um *prompt* semelhante à:

  Saída 1.8.5.1 luisrodrigoog@matedesk:/$

1.9 Referencias

Mais informações sobre o sistema operacional *Linux*, sua estrutura interna, seus comandos, serviços e algumas de suas principais distribuições podem ser obtidas utilizando um dos livros da seção 1.9.1 ou em um dos sites da seção 1.9.2.

1.9.1 Livros

Há vários livros disponíveis no mercado sobre o *Linux*, destes podemos destacar alguns que foram utilizados para montar este material:

- Danesh, Arman; Dominando o Linux – Red Hat Linux 6.0, A Bíblia; Ed. Makron Books
- Welsh, Matt; Dominando o Linux; Ed. Ciência Moderna
- Siever, Ellen; Linux, O Guia Essencial; Editora Campus
- Petrovsjy, Michele e Parkinson, Tom; Guia de Referência do Unix; Quark Books

1.9.2 Sites

Dentre os vários sites disponíveis na *Internet*, relacionados com o sistema operacional *Linux*, podemos destacar os seguintes:

- Site oficial do Debian Linux
- Site oficial do Ubuntu Linux
- Site oficial do CentOs
- Site oficial do Kali Linux
- Site oficial do Fedora
- Site do Guia Foca Linux GNU/Linux
- Linux Security
- Linux Security for Beginners

II Part 2 - Comandos

2. Comandos Básicos

Este capítulo visa apresentar a lista de comandos mais utilizados no *Linux*; para cada comando teremos uma breve descrição, sua sintaxe, seus principais argumentos e alguns exemplos.

OBS: Nosso objetivo não é explorar todos os argumentos e opções disponíveis para cada um dos comandos, mas sim apresenta-los de forma clara e sucinta.

2.1 Conectando-se e desconectando-se do sistema

Depois de iniciado, o *Linux* realiza um processo de validação do usuário que é executado através dos mecanismos de autenticação, mais especificamente pelo comando *login*. Toda vez que se deseja fazer uso de uma máquina *Linux*, deve realizar o processo de conexão, algo semelhante ao que fazemos quando usamos o caixa eletrônico dos bancos, porém, no *Linux*, fazemos uso de duas informações, a nossa identificação pessoal conhecida normalmente como ***Login Name*** ou ***User Name*** e a **Senha** ou ***Password***.

Para gerenciar o processo acesso e desconexão do sistema, podemos fazer uso dos seguintes comandos:

2.1.1 login

Este comando permite nos conectar ao sistema, mas para tal precisamos de um *User Name* válido, que é passado como o primeiro argumento.

Sintaxe 2.1.1 **login [username]**

Onde:

- **username** : é o nome do usuário o qual deseja se conectar

Caso fosse necessário utilizar a conta do usuário **smorgana**, podemoríamos utilziar o comando

Exemplo 2.1.1

```
login smorgana
```

2.1.2 logout e exit

Estes comandos são utilizado para desconectar-se do sistema; a princípio eles não necessitam de nenhum argumento.

Sintaxe 2.1.2

logout

Sintaxe 2.1.3

exit

2.1.3 halt

Utilizamos o comando ***halt***, quando desejamos desligar o computador. Em alguns sistemas, este comando só pode ser executado com privilégios administrativos.

Sintaxe 2.1.4

halt

2.1.4 reboot

Assim como o *halt*, o comando *reboot*, pode necessitar de privilégios administrativos para ser executado e pode ser utilizado quando desejamos reiniciar a máquina.

Sintaxe 2.1.5

reboot

2.1.5 shutdown

O comando *shutdown*, dependendo do argumento passado, via linha de comando, nos permite tanto desligar quanto reiniciar o computador, assim permite determinar o momento em que a ação deve ser executada.

Sintaxe 2.1.6

shutdown [-r] [-h] now

Onde:

- ☑ **-r** : para reiniciar o computador
- ☑ **-h** : para desligar o computador

Vejamos alguns exemplos de uso para este comando:

Exemplo 2.1.2 : Para reinicia o sistema, assim que o comando for executado, podemos utilizar a seguinte linha de comando:

```
shutdown -r now
```

Exemplo 2.1.3 : E para desligar a máquina, podemos utilizar:

```
shutdown -h now
```

2.2 Gerenciamento de Arquivos

O *Linux* oferece uma vasta gama de comandos e utilitário, que auxiliam no gerenciamento dos arquivos e da estrutura de diretórios; dos quais, alguns serão apresentados nas próximas seções.

2.2.1 Apresentando o conteúdo do diretório

O comando **ls** pode ser utilizado para listar o conteúdo de um determinada pasta ou diretório e possui a seguinte sintaxe:

Sintaxe 2.2.1 : **ls [parâmetros] [diretório]**

Onde :

- ☑ **diretório**: é o caminho relativo ou absoluto do diretório, cujo conteúdo deve ser listado;
- ☑ Dentre os **parâmetros** suportados destacam-se:
  - ☑ **-l** : lista os arquivos, contidos no diretório, utilizando um formato longo, ou seja, junto com o nome dos arquivos também serão apresentadas as suas propriedades.
  - ☑ **-a**: lista todos os arquivos, incluindo os ocultos, ou seja, aqueles que começam por um ponto (“.”)
  - ☑ **-h**: exibe o tamanho dos arquivos em *Kbytes*, *Mbytes*, *Gbytes*, ou seja, na unidade mais próxima.

Vejamos alguns exemplos:

Exemplo 2.2.1 Para listar o conteúdo do diretório **/var**, podemos utilizar o comando

```
ls -l /var
```

Exemplo 2.2.2 Para listar o conteúdo do diretório **/var**, no formato longo e exibindo os arquivos ocultos, podemos utilizar:

```
ls -l /var
```

2.2.2 metacaracterees ou Caracteres Coringa.

Os metacaracterees, ou caracteres coringa, geralmente são utilizados na abreviação de nomes de arquivos, e seu uso é muito difundido no mundo *Linux*. Os principais metacaracterees são: ***?*** ** [ ]*.

Para entender melhor o funcionamento dos *metacaracterees*, vamos supor que dentro do nosso diretório pessoal (*homedir*) existam os seguintes arquivos:

```
teste1 teste2 teste3 teste4
texto01 texto02 texto03 texto04
```

Exemplo 2.2.3 Podemos utilizar o comando **ls** para visualizar estes arquivos:

```
ls
```

Saída 2.2.3.1 O comando acima deve gerar a seguinte saída:

```
teste01 teste02 teste03 teste04 teste1 teste2 teste3 teste4
```

Com base nestes arquivos utilizaremos alguns exemplos para demonstrar como os caracteres coringa funciona, começando pelo "? ", pode ser utilizado para substituir qualquer carácter, apenas uma vez.

Exemplo 2.2.4 Neste exemplo, combinaremos o comando **ls** com o metacaractere "?", utilizando este combinação obteremos a listagem de todos os arquivos que começam por **teste** e possuem apenas um carácter apos.

```
ls teste?
```

Saída 2.2.4.1 Obteremos como resposta a seguinte saída:

```
teste1 teste2 teste3 teste4
```

O metacaractere "*" pode ser utilizado para substituir qualquer sequencia de caracteres.

Exemplo 2.2.5 Neste exemplos utilizamos o comando **ls** e o metacaractere "*" para listar todos os arquivos que começam por "**te**":

```
ls te*
```

Saída 2.2.5.1 Obteremos como resposta:

```
teste1 teste2 teste3 teste4 texto01 texto02 texto03 texto04
```

Já o metacaractere "**[ab]**", define um **conjunto** de caracteres que poderá ser utilizado no processo de substituição.

Exemplo 2.2.6 Para listar todos os arquivos que começam com a palavra "**teste**" e terminam com **1** ou **4**, podemos utilizar:

```
ls teste[14]
```

Saída 2.2.6.1 O comando anterior retornará a seguinte saída:

```
teste1 teste4
```

Podemos utilizar "**[a-b]**" para definir um **intervalo** de caracteres que poderão ser utilizados no processo de substituição.

Exemplo 2.2.7
Para listar todos os arquivos que começam com a palavra "**teste**" e terminam com um valor de **1** até **4**, podemos utilizar:

```
ls teste[1-4]
```

Saída 2.2.7.1 O comando anterior deve retornar:

```
teste1 teste2 teste3 teste4
```

Finalmente, o carácter ";" separa vários comandos em uma única linha:

Exemplo 2.2.8 Podemos alterar o diretório corrente para o *homedir* do usuário, listar todos os arquivos que começam por "**teste0**" e terminam com um dos valores de **1** até **4**, utilizando:

```
cd ~; ls -lh teste0[1-4]
```

Saída 2.2.8.1 O comando anterior deve retornar:

```
-rw-rw-r- 1 luisrodrigoog luisrodrigoog 0 Mar 2 13:22 teste01
-rw-rw-r- 1 luisrodrigoog luisrodrigoog 0 Mar 2 13:22 teste02
-rw-rw-r- 1 luisrodrigoog luisrodrigoog 0 Mar 2 13:22 teste03
-rw-rw-r- 1 luisrodrigoog luisrodrigoog 0 Mar 2 13:22 teste04
```

2.2.3 Manipulando diretórios

Além de listar os arquivos contidos em um diretório, o *Linux* nos disponibiliza comandos para manipular a arvore de diretórios, alguns destes comandos serão apresentados nas próximas seções.

mkdir

O primeiro comando que estudaremos é o **mkdir**, o qual podemos utilizar para criar novos diretórios

Sintaxe 2.2.2 : **mkdir -p [nome_do_diretório]**

Onde:

- ☑ **nome_do_diretório** : representa o caminho no qual será criado o novo diretório; é possível criar mais do que um diretório ao mesmo tempo, basta que informemos na mesma linha de comando o nome de todos os diretórios.
- ☑ **-p** : força a criação de todos diretórios que estão ausentes no caminho.

Vejamos alguns exemplos:

Exemplo 2.2.9 : Neste exemplos serão criados os diretórios **teste1** e **teste2** dentro do **homedir** do usuário, utilizando apenas uma linha de comando:

```
mkdir ~/teste1 ~/teste2
```

cd

Utilizamos o comando **cd** para alterar o diretório corrente, ou seja, para mudar a nossa posição dentro da arvore de diretórios.

Sintaxe 2.2.3 : **cd [path_para_o_diretório]**

Onde:

- ☑ **path_para_o_diretório** : representa o caminho para o qual seremos direcionados.

Exemplo 2.2.10 : Neste exemplo utilizamos o comando **cd** para entrar no diretórios "**/etc/apt**", onde ficam os arquivos de configuração do comando **apt**:

```
cd /etc/apt
```

rmdir

O comando **rmdir** é utilizado para remover um dado diretório, o qual deve estar vazio, ou seja, não deve conter nenhum tipo de arquivo, caso contrário recebemos uma mensagem de erro.

Sintaxe 2.2.4 : **rmdir [path_para_o_diretório]**

Onde:

- ☑ **path_para_o_diretório** : representa o caminho do diretório a ser removido

Exemplo 2.2.11 Para remover o diretório denominado "**fotos**" que está dentro do **homedir** do usuário, podemos utilizar:

```
rmdir ~/fotos
```

Caso o diretório esteja vazio ele será removido com sucesso, caso contrário recebemos uma mensagem semelhante a apresentadas a seguir.

Saída 2.2.11.1 : Mensagem de erro gerada quando o diretório não está vazio:

```
rmdir:  falhou em remover ‘‘fotos/’’:  Diretório não vazio
```

pwd

Este comando é utilizado para informar o diretório corrente, ou simplesmente o diretório atual. Geralmente não utilizamos parâmetros com este comando:

Sintaxe 2.2.5 : **pwd**

Exemplo 2.2.12 : A sequencia de comandos a seguir, move o usuário para o diretório "**/var/log**", lista o conteúdo deste diretório, move o usuário para o seu **homedir** e imprime a sua posição na arvore de diretórios:

```
cd /var/log
ls -l
cd ~
pwd
```

2.3 Copiando, Removendo, Movendo e Renomeando Arquivos

Assim como podemos manipular nossos diretórios, também podemos manipular nossos arquivos; sendo que os três principais comandos disponíveis para esta finalizada são: (i) **cp**, (ii) **mv** e o (iii) **rm**.

2.3.1 cp

O comando **cp** é utilizado para copiar arquivos, quando o Linux realiza o processo de cópia, ele mantém a versão original e cria um novo arquivo idêntico ao original no ponto de destino.

Sintaxe 2.3.1 : **cp -Rv [origem] [destino]**

Onde:

- ☑ **-R** : realiza uma cópia recursiva; neste caso, se houver subdiretórios, o conteúdo dos mesmos também será copiado.
- ☑ **-v** : exibe o que está sendo copiado.
- ☑ **-f** : modo forçado, sobrescreve o arquivo sem solicitar confirmação.

Observem os exemplos abaixo:

Exemplo 2.3.1 Para copiar o arquivo **bash** localizado no diretório **/bin**, para o diretório local (./), podemos utilizar:

```
cp /bin/bash ./
```

Exemplo 2.3.2 Para copiar todos os arquivos que começam com a letra "**b**", localizados no diretório **/bin** e em seu subdiretórios, para o diretório "**~/binarios**", podemos utilizar:

```
mkdir ~/binarios
cp -R /bin/b* ~/binarios
```

2.3.2 mv

Este comando é utilizado tanto para mover arquivos quando para renomeá-los, nestes casos o arquivo original deixa de existir.

Sintaxe 2.3.2 : **mv –Rv [origem] [destino]**

Onde:

- ☑ **origem**: determina o nome do arquivo ou diretório que deverá ser movido.
- ☑ **destino**: permite especificar para onde o arquivo, ou diretório, deve ser movido. Geralmente utilizamos como destino um caminho absoluto ou relativo. Caso seja especificado um nome diferente do original o arquivo ou diretório será renomeado.

- **-R** : move, os arquivos ou diretórios de forma recursiva, ou seja, se no **path**, indicado como origem, houver subdiretórios o conteúdo dos mesmos também serão movidos
- **-v** : exibe o nome do aquivo que está sendo movido.
- **-f** : habilita o modo forçado, no qual os arquivos serão sobrescritos sem solicitar a confirmação do usuário. Use com cuidado

Vejamos alguns exemplos de como podemos utilizar este comando:

Exemplo 2.3.3 : O comando abaixo, pode ser utilizado para **renomear** o diretório "**fotos**", que está dentro do **homedir** do usuário **aluno**, para "**bkp**"

```
mv /home/aluno/fotos /home/aluno/bkp
```

Exemplo 2.3.4 : O comando abaixo, pode ser utilizado para **mover** todo o conteúdo do diretório "**/home/aluno**", de forma recursiva, para o diretório "**/opt/bkp**". E a medida que os arquivos forem sendo movidos seus nomes serão listados.

```
mv -Rv /home/aluno/ /opt/bkp
```

2.3.3 rm

Utilizamos o comando **rm** para remover um ou vários arquivos, ou diretórios.

Sintaxe 2.3.3 : **rm –Rvf [origem]**

Onde :

- **-R** : remove recursivamente, ou seja, se houver subdiretórios o conteúdo dos mesmos também serão removidos.
- **-v** : exibe a identificação do que está sendo removido.
- **-f** : modo forçado, remove o arquivo sem solicitar confirmação.

A seguir temos um exemplo do uso deste comando:

Exemplo 2.3.5 : Para remover todos os arquivos contidos no diretório "**/home/aluno/bkp**", podemos utilizar o comando:

```
rm -Rv /home/aluno/bkp
```

2.4 Exibindo o conteúdo de arquivos

Durante a administração de um sistema *Linux*, geralmente, precisamos visualizar o conteúdo de vários arquivos texto; que normalmente podem conter dados sobre as configurações ou informações com os *logs* o sistema. O objeto desta seção é apresentar alguns comandos que podem nos ajudar a visualizar o conteúdo dos nossos arquivos.

2.4.1 cat

O comando **cat**, pode ser utilizado para exibir o conteúdo de um ou vários arquivos, ou pode ser utilizado para concatenar seus conteúdos em um novo arquivo.

Sintaxe 2.4.1 : **cat [lista_de_arquivos]**

Onde:

- ☑ **lista_de_arquivos** : é a lista de nome dos arquivos a serem exibidos ou concatenados

Para facilitar a compreensão sobre o funcionamento do comando, observem os próximos exemplos:

Exemplo 2.4.1 O arquivo "**/etc/hosts**" contem as informações referente à resolução de nomes realizada localmente, ele funciona como um *DNS* local. Para exibir o conteúdo deste arquivo, podemos utilizar o comando abaixo:

```
cat /etc/hosts
```

Que deve gerar uma saída semelhante à:

Saída 2.4.1.1

```
# The following lines are desirable for IPv4 capable hosts
127.0.0.1       localhost
127.0.1.1       phil

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

A seguir temos um exemplo de como combinar dois ou mais arquivos.

Exemplo 2.4.2 Podemos utilizar o comando abaixo para combinar os arquivos **lista1.txt** e **lista1.txt** em um único com o nome de **lista-completa.txt**

```
cat lista1.txt lista2.txt » lista-completa.txt
```

2.4.2 more

Quando precisamos listar o conteúdo de arquivos grandes, os quais possuem mais linhas do que cabem na tela do terminal, podemos utilizar o comando **more**, que exibe o conteúdo de um arquivo, uma tela por vez.

Sintaxe 2.4.2 : **more [lista_de_arquivos]**

Onde:

- ☑ **lista_de_arquivos**: contem o nome de todos os arquivos a serem exibidos.
- ☑ **-d**: exibe a mensagem "[Pressione espaço para continuar, 'q' para sair.]", e exibe "[Pressione 'h' para obter instruções.]". Em vez de tocar o *beep* quando uma tecla ilegal é pressionado.

Este comando é interativo e aceita vários comandos internos, como por exemplo, podemos utilizar a tecla "**ENTER**" para nos movimentar pelo arquivo. Além deste há vários outros, dos quais podemos destacar:

- 👉 **h ou ?**: exibe um resumo dos comandos contidos neste lista.
- 👉 **SPACE ou z** : exibe as próximas **k** linhas de texto. O padrão é o tamanho da tela atual.
- 👉 **RETURN**: exibe as próximas **k** linhas de texto. O padrão é **1**.
- 👉 **d** ou **ctrl+D**: percorre **k** linhas. O padrão é o tamanho atual é **11**.
- 👉 **q** ou **Q**: finaliza o processo de paginação.
- 👉 **f**: avançar **k** telas cheias.
- 👉 **b** ou **ctrl+B**: pula para trás **k** telas cheias de texto.
- 👉 **=** : Exibir o número da linha atual.
- 👉 **:f** Exibe o nome do arquivo e a linha atual.

A seguir temos um exemplo de uso deste comando:

Exemplo 2.4.3 : Para listar o conteúdo do arquivo "**/var/log/syslog**", uma tela por vez, podemos utilizar:

```
more /var/log/syslog
```

2.4.3 less

O comando **less** permite que um arquivo possa ser exibido uma página por vez, e ainda que o mesmo seja movimentado para cima e para baixo. O comando **less** é semelhante ao comando **more**, porem com mais funcionalidades

Sintaxe 2.4.3 : **less [arquivo]**

Onde:

- ☑ **arquivos** : é a lista de nome dos arquivos a serem exibidos.

Podemos utilizar a tecla "ENTER" ou as setas direcionais para nos movimentar pelo arquivo. Para sair deste comando pressione "q". Os mesmos comandos interativos que podemos utilizar no **more** também estão disponíveis no **less**.

Exemplo 2.4.4 : Para listar o conteúdo do arquivo "**/var/log/syslog**", uma linha por vez ou uma tela por vez, podemos utilizar:

```
less /var/log/syslog
```

2.5 Links

O comando **ln** nos permite criar *links*, ou seja, permite que vários atalhos apontem a um mesmo arquivo real, ou posição da arvore de diretórios. No *Linux*, existem basicamente dois tipo de *links*:

- ***Hard Links*** : são arquivos que apontam para o mesmo *i-node*.
- ***Link Simbólico*** : é arquivo especial que contem o caminho completo para o arquivo destino

No *Linux*, utilizamos o comando **ln** para criar ambos os tipos de *links*, a sintaxe deste comando pode ser observada logo a seguir.

Sintaxe 2.5.1 : **ln -s [origem] [link]**

Onde:

- ☑ **-s** - cria link simbólico
- ☑ **origem** - arquivo ou diretório a se referido
- ☑ **link** - nome a ser dado para o arquivo de link

A seguir temos um exmeplo do uso deste comando.

Exemplo 2.5.1 Podemos utilizar próxima sequencia de comandos para: (i) entrar no *homedir* do usuário; (ii) criar e entrar na pasta **aula3**; (iii) copiar o arquivo **bash**, que está localizado no diretório **/bin** para o *homedir*; (iv) criar um *hardlink* do arquivo com o nome **hteste**; (v) em seguida criar um *link* simbólico com nome de **steste** e (vi) listar o conteúdo do diretórios.

```
cd ~
mkdir aula3; cd aula3
cp /bin/bash ./
ln bash hteste
```

```
ln -s bash steste
ls -la
```

Saída 2.5.1.1 : O comando **ls** devm gerar uma saída semelhante à:

```
total 2040
drwxrwxr-x  2 aluno aluno    4096 Mar  3 16:48 .
drwxr-xr-x 28 aluno aluno    4096 Mar  3 16:48 ..
-rwxr-xr-x  2 aluno aluno 1037528 Mar  3 16:47 bash
-rwxr-xr-x  2 aluno aluno 1037528 Mar  3 16:47 hteste
lrwxrwxrwx  1 aluno aluno       4 Mar  3 16:48 steste -> bash
```

2.6 Comparando arquivos

Durante o processo de desenvolvimento de aplicações e da configuração de um sistema *Linux*, muitas vezes precisamos comparar e localizar as diferenças contidas nos arquivos; os comandos desta seção auxiliam na localização de mudanças entre as várias versões de um arquivo e nos permite identificar, por exemplo, erros que seriam difíceis de serem encontrados de outra forma.

2.6.1 cmp

O comando **cmp** pode ser utilizado para comparar o conteúdo de dois arquivos quaisquer. Sua sintaxe é bem simples e pode ser observada abaixo:

Sintaxe 2.6.1 : **cmp [arquivo1] [arquivo2]**

Onde:

- ☑ **arquivo1**, **arquivo2**: é o nome dos arquivos a serem comparados;

O comando **cmp** exibe apenas a primeira diferença entre os dois arquivos. Observe o exemplo abaixo.

Exemplo 2.6.1 : Como sabemos, o comando **cat** pode ser utilizado para listar o conteúdo de arquivos texto; desta forma, os dois primeiros comandos nos apresentam o conteúdo dos arquivos **arq1.txt** e **arq2.txt**; já o comando **cmp** apresenta a primeira diferença localizada nos arquivos:

cat arq1.txt

```
1234567890
```

cat arq2.txt

```
1234567890X
```

cmp arq1 arq2

```
arq1 arq2 differ:  char 11, line 1
```

2.6.2 diff

O comando **diff** exibe todas a diferença entre doi ou mais os arquivos. Devemos passar para o **diff** dois arquivos: o **arquivo1** e **arquivo2**; caso o **arquivo1** seja um diretório e **aquivo2** não, o comando **diff** compara o **arquivo2** com o primeiro da lista dos arquivos contidos no diretório. Se ambos o nomes fornecidos são diretório, o comando **diff** compara os arquivos correspondentes em ambos diretórios, neste caso, os arquivos serão selecionados em ordem alfabética; por padrão esta comparação não é recursiva a menos que a opção seja fornecida **-r** ou **–recursive**. O comando **diff** nunca compara o conteúdo atual do diretório como se ele fosse um arquivo.

Como podemos observar, a sintaxe deste comando é um pouco mais complexa do que aquela utilizada pelo **cmp**.

Sintaxe 2.6.2 : **diff -arc [arquivo1] [arquivo2]**

Onde :

- ☑ **-a**: Trata todos os arquivos como texto e compara-os linha por linha, mesmo se eles não se parecem com texto.
- ☑ **-c**: retorna os dados no formato de contexto.
- ☑ **-r** ou **–recursive**: Quando comparando diretórios, compara os sub-diretórios encontrados de forma recursiva.

A seguir temos um exemplo de como podemos utilizar o comando **diff**.

Exemplo 2.6.2 Neste exemplo temos dois arquivo denominados **arq1.txt** e **arq2.txt**; o primeiro comando exibe o conteúdo de **arq1.txt** e o segundo o conteúdo de **arq2.txt**; finalmente o ultimo comando apresenta a diferença entre os dois:

```
cat arq1.txt
   123456789
   1234567890
cat arq2.txt
   1234567890
   123456789
diff arq1 arq2
   *** teste1 2003-02-03 23:21:20.000000000 -0200
   -- teste2 2003-02-03 23:21:31.000000000 -0200
   ***************
   *** 1,2 ****
   - 123456789
   1234567890
   -- 1,2 --
   1234567890
   + 123456789
```

2.7 Pedindo ajuda

Até este ponto já tivemos acesso à vários comandos; dos quais estudamos apenas alguns de seus argumentos, mas cada um dos comandos estudados podem ter vários outro argumentos. Para obter *ajuda* sobre um comando ou um utilitário do *Linux* podemos utilizar um comandos apresentados nesta seção.

2.7.1 man

Este comando exibe a página de manual, *help*, de um determinado comando. Algumas destas páginas de manual já estão traduzidas para nosso idioma, mas a grande maioria está disponível apenas na língua inglesa.

Sintaxe 2.7.1 : **man -L [comando]**

Onde:

- ☑ **comando** : é o nome do comando para qual desejamos obter informações
- ☑ **-L** : permite determinar o idioma no qual a página de manual deve ser exibida

A seguir temos um exemplo de como podemos utilizar este comando:

Exemplo 2.7.1 : Com a linha de comando abaixo será apresenta a página de manual do **cp**, utilizando nosso idioma (**pt**).

```
man -L pt cp
```

2.7.2 help

O comando **help** exibe informações sobre comandos internos do *shell*, sua sintaxe é bem simples e pode ser observada logo abaixo:

Sintaxe 2.7.2 **help [comando]**

Onde:

- ☑ **comando** : é o nome do comando para qual desejamos obter informações

A seguir tem um exemplo de como utilizar o comando:

Exemplo 2.7.2 : Pomos utilizar o comando **help** para obter ajuda sobre ele mesmo:

```
help help
```

Saída 2.7.2.1 O comando acima deve gerar uma saída semelhante à:

```
help:  help [-dms] [padrão ...]
```

```
        Exibe informação sobre comandos builtin.

        Exibe um breve resumo dos comandos builtin.  Se PATTERN for
especificado, retorna ajuda detalhada de todos os comandos que
coincidirem com PATTERN. Caso contrário a lista de tópicos de ajuda é
impressa.

        Opções:
          -d imprime uma breve descrição para cada tópico
          -m mostra o uso no formato de uma pseudo-manpage
          -s imprime apenas uma pequena sinopse de uso para cada
               tópico correspondente

        Argumentos:
          PATTERN Padrão especificando um tópico de ajuda

        Estados de saída:
          Retorna sucesso a menos que o PATTERN não seja encontrado ou
          uma opção inválida é inserida.
```

2.7.3 info

Este comando apresenta várias informações sobre os comandos do *Linux*, ele é muito semelhante ao *man*, mas ele é orientado a menus, o que facilita localizar informações relacionadas.

Sintaxe 2.7.3 info [comando]

Onde:

- ☑ **comando**: é o nome do comando para qual desejamos obter informações

Exemplo 2.7.3 : Para obter informações sobre o comando *bash* podemos utilizar :

```
info bash
```

2.8 Permissões e Controle de Acesso aos arquivos

No sistema de arquivos do *Linux*, residem grande parte das configurações dos serviços e dos registros do funcionamento do ambiente. Uma forma de proteger estas informações e garantir o bom funcionamento do sistema é aplicar os devidos mecanismos de controle de acesso aos arquivos e diretórios do sistema operacional. Nesta seção serão apresentados alguns dos comandos que podemos utilizar para esta finalidade.

2.8.1 chmod

O comando *chmod* permite especificar/modificar as permissões de um determinado arquivo ou diretório. Com ele podemos definir o nível de acesso baseado mas permissões atribuídas ao **dono** (**U**) (usuário que retem a posse do arquivo), ao **grupo** (**G**) ao qual o arquivo está associado e nos **demais** (**O**) usuários que não são seu proprietário, nem fazem parte do mesmo grupo ao qual o arquivo foi associado.

Sintaxe 2.8.1 : **chmod -R [permissões] [arquivo]**

Onde:

- **arquivo** : é o nome do arquivo ou diretório para o qual desejamos definir as permissões
- **-R** : ativa a função recursiva que irá ajustar a permissão dos arquivos e diretórios das subpastas.
- **permissões** : define os direitos que devem atribuídos à um determinado arquivo. A seguir temos as classes que podem ser utilizadas:
  - **u** = usuário
  - **g** = grupo
  - **o** = outros

  As **três permissões** podem ser utilizadas são:
  - **r** – leitura
  - **w** – escrita
  - **x** – execução

  Finalmente, os **modificadores** que podemos utilizar são:
  - **+** : adiciona
  - **-** : remove
  - **=** : iguala

A seguir temos alguns exemplo de como utilizar este comando.

Exemplo 2.8.1 : Neste exemplo copiados o arquivo **bash**, localizado no diretório **/bin** para a pasta **~/aula8/teste** e em seguida utilizamos o comando **ls** para listar suas propriedades.

```
mkdir -p ~/aula8/teste      cp /bin/bash ~/aula8/teste
ls -la
     drwxrwxr-x 2 user1 user1 4096 Mar 5 10:22 .
     drwxrwxr-x 5 user1 user1 4096 Mar 5 09:53 ..
```

```
-rwxr-xr-x 1 user1 user1 1037528 Mar 5 10:22 teste
```

Como podemos perceber, o **proprietário** deste arquivo possui as permissões de leitura, escrita e gravação; assim como os usuários deste **grupo**, já os usuários que **não fazem parte do grupo** possuem apenas as permissões de leitura e escrita. Utilizando o comando abaixo, adicionamos a permissão de escrita para os usuários que ainda não possuem.

```
chmod g+w teste
ls -la
    drwxrwxr-x 2 user1 user1 4096 Mar 5 10:22 .
    drwxrwxr-x 5 user1 user1 4096 Mar 5 09:53 ..
    -rwxrwxr-x 1 user1 user1 1037528 Mar 5 10:22 teste
```

Observando a saída do comando **ls**, percebemos que, agora, todos os usuários possuem permissão de leitura, de escrita e execução; estas correspondem ao valor 777.

Utilizando o comando **chmod**, apresentado abaixo, atribuímos, aos usuários pertencentes ao **grupo**, apenas a permissão de escrita, logo o arquivo **não** pode ser **nem lido nem executado** por estes usuários.

```
chmod g=w teste
ls -la
    drwxrwxr-x 2 user1 user1 4096 Mar 5 10:22 .
    drwxrwxr-x 5 user1 user1 4096 Mar 5 09:53 ..
    -rwx-w-r-x 1 user1 user1 1037528 Mar 5 10:22 teste
```

2.8.2 umask

Quando criamos nossos arquivos, o *Linux* associa aos mesmos um conjunto padrão de permissões; para definir ou atualizar este padrão temos o comando ***umask***, responsável por ajustar esta mascara. Com o comando **umask** não utilizamos o mesmo valor que no ***chmod***, ele precisa que seja fornecida o valor invertido; por exemplo para definir a permissão padrão como **750**, devemos utilizar a mascara **0027**.

Sintaxe 2.8.2 umask [mascara]

Onde:

- ☑ **mascara** - é o inverso do valor absoluto utilizado no comando **chmod**.

Exemplo 2.8.2 Neste exemplo, utilizando o comando ***touch*** criaremos três arquivos, o primeiro será criado utilizando a mascara padrão, desta forma a permissão atribuída será : $-rw-rw-r--$; antes de criarmos o segundo arquivo, ajustamos a mascara para 0027, logo o mesmo ficou com a seguinte permissão: $-rw-r-----$; finalmente no ultimo arquivo utilizamos a mascara 0077, que gerou as permissão $-rw-------$

```
touch teste1
ls -l teste1
      -rw-rw-r- 1 aluno aluno 0 mar 7 10:05 teste1
umask 0027
touch teste2
ls -l teste2
      -rw-r--- 1 aluno aluno 0 mar 7 10:06 teste2

umask 0077
touch teste3
ls -l teste3
      -rw---- 1 aluno aluno 0 mar 7 10:07 teste3
```

Como podemos observar, no exemplo anterior, nenhum dos arquivos recebeu a permissão de execução ("x"); isto ocorre, pois nenhum deles é um arquivo binário ou um *script*. Mesmo quando definido via ***umask***, o *Linux* só irá atribuir a permissão de execução quado for necessário.

2.8.3 chown

Utilizamos este comando quando precisamos alterar o proprietário, dono, de um ou vários arquivos. Todo arquivo do *Linux* deve estar associado à um **UID** de usuário. Quando baixamos arquivos compactados da *Internet*, geralmente precisamos ajustar a posse destes novos arquivos, pois, provavelmente o **UID** associado a estes arquivo não está cadastrado na maquina.

Sintaxe 2.8.3 **chown [-fhR] [user_name] [arquivo]**

Onde:

- **-f**: ativa o modo forçado, o que em alguns casos evita que erros sejam reportados.
- **-h**: se o arquivo for um *link* simbólico, altera a propriedade do *link* simbólico e não do arquivo.
- **-R**: opção recursiva, ou seja, será aplicada as subpastas e seu conteúdo.
- **user_name**: pode ser o *user name* ou o **UID** do usuário ao qual será atribuído à posse dos arquivos.

Exemplo 2.8.3 : O próximo comando altera a posse de todos os arquivos da pasta "/home/aluno/copia/uther", de seus arquivos e subpastas para o usuário **aluno**.

```
chown -R aluno /home/aluno/copia/uther/*
```

2.8.4 chgrp

O comando ***chgrp*** nos permite alterar o grupo ao qual um ou vários arquivos estão associados. Em alguns casos precisamos compartilhar uma pasta, ou um conjunto de arquivos, com um grupos de usuários, neste caso, podemos ajustar o grupo ao qual a pasta está associada para o mesmo que desejamos compartilhar.

Sintaxe 2.8.4 chgrp [-fhR] [gid] [arquivo]

Onde:

- **gid**: é o número decimal que identifica o grupo, ou o próprio nome do grupo.
- **-f**: entra em modo forçado; o que evita que algumas mensagens de erro sejam geradas e capturadas.
- **-h**: se o arquivo for um *link* simbólico, altera a propriedade do *link* simbólico e não do arquivo.
- **-R**: ativa a opção recursiva no processo de atualização das configurações.

Exemplo 2.8.4 Para atualizar o grupo ao qual a pasta "/opt/projetos/sgs/" e todo o seu conteúdo está associado para o grupo **sgs**, podemos utilizar:

```
chgrp -R sgs /opt/projetos/sgs/*
```

2.9 Redirecionadores de E/S

Os redirecionadores de entrada e saída nos permite alterar o local para o qual o *Linux* encaminha a saída de um programa e a partir de onde ele obtém os dados de entrada. Os redirecionadores são muito úteis quando precisamos guardar a saída de um comando para uso posterior ou para encaminhar a saída de um comando como estrada de outro. Os principais redirecionadores de entrada e saída são:

- `>` : Envia a saída padrão do comando ou do programa para um determinado arquivo.

Exemplo 2.9.1 : Vamos exemplificar o uso deste redirecionador utilizando o comando **ls**, para listar o conteúdo de um determinado diretório:

```
ls -lah
     total 8,0K
     drwxrwxr-x 2 aluno aluno 4,0K mar 7 10:41 .
     drwxrwxr-x 3 aluno aluno 4,0K mar 7 10:05 ..
     -rw-rw-r- 1 aluno aluno 0 mar 7 10:05 teste1
     -rw-r--- 1 aluno aluno 0 mar 7 10:40 teste2
     -rw---- 1 aluno aluno 0 mar 7 10:41 teste3
```

Podemos perceber que a saída do comando **ls** foi apresentado na tela, que é o dispositivo de saída padrão. No próximo comando utilizamos o redirecionador de entrada e saída anterior para encaminhar a saída do comando para o arquivo ***exemplo.txt***:

```
ls -lah > exemplo.txt
cat exemplo.txt
     total 8,0K
     drwxrwxr-x 2 aluno aluno 4,0K mar 7 10:41 .
     drwxrwxr-x 3 aluno aluno 4,0K mar 7 10:05 ..
     -rw---- 1 aluno aluno 0 mar 7 14:38 exemplo.txt
     -rw-rw-r- 1 aluno aluno 0 mar 7 10:05 teste1
     -rw-r--- 1 aluno aluno 0 mar 7 10:40 teste2
     -rw---- 1 aluno aluno 0 mar 7 10:41 teste3
```

Observe que, caso o arquivo de destino (**exemplo.txt**) já existia ele será sobrescrito pelo novo arquivo, ou seja o "`>`" é um redirecionar destrutivo.

>> : Envia as informações da saída padrão de um comando para um determinado arquivo, sem destruir o conteúdo original.

Exemplo 2.9.2 : Utilizaremos novamente o comando **ls** para demonstrar o funcionamento do redirecionado:

```
ls -lah
     total 8,0K
     drwxrwxr-x 2 aluno aluno 4,0K mar 7 10:41 .
     drwxrwxr-x 3 aluno aluno 4,0K mar 7 10:05 ..
     -rw---- 1 aluno aluno 0 mar 7 14:38 exemplo.txt
     -rw-rw-r- 1 aluno aluno 0 mar 7 10:05 teste1
     -rw-r--- 1 aluno aluno 0 mar 7 10:40 teste2
     -rw---- 1 aluno aluno 0 mar 7 10:41 teste3
```

Novamente, podemos perceber que a saída do comando foi enviada para a tela, mas nos próximo comando conseguimos direcionar para o arquivo **exemplo2.txt** o conteúdo do comando **ls**, e como utilizamos o redirecionador >>, o novo conteúdo será adicionado ao final do arquivo original.

```
ls -lah >> exemplo2.txt      cat exemplo2.txt
     total 8,0K
     drwxrwxr-x 2 aluno aluno 4,0K mar 7 10:41 .
     drwxrwxr-x 3 aluno aluno 4,0K mar 7 10:05 ..
     -rw---- 1 aluno aluno 0 mar 7 14:39 exemplo2.txt
     -rw---- 1 aluno aluno 0 mar 7 14:38 exemplo.txt
     -rw-rw-r- 1 aluno aluno 0 mar 7 10:05 teste1
     -rw-r--- 1 aluno aluno 0 mar 7 10:40 teste2
     -rw---- 1 aluno aluno 0 mar 7 10:41 teste3
```

Para comprovar o comportamento não destrutivo deste redirecionador, vamos repetir o comando anterior e verificar como ficou o arquivo.

```
ls -lah >> exemplo2.txt
cat exemplo2.txt
     total 12K
     drwxrwxr-x 2 aluno aluno 4,0K mar 7 14:49 .
     drwxrwxr-x 3 aluno aluno 4,0K mar 7 10:05 ..
     -rw---- 1 aluno aluno 0 mar 7 14:49 exemplo2.txt
     -rw---- 1 aluno aluno 403 mar 7 14:38 exemplo.txt
     -rw-rw-r- 1 aluno aluno 0 mar 7 10:05 teste1
```

```
-rw-r--- 1 aluno aluno 0 mar 7 10:40 teste2
-rw---- 1 aluno aluno 0 mar 7 10:41 teste3
total 16K
drwxrwxr-x 2 aluno aluno 4,0K mar 7 14:49 .
drwxrwxr-x 3 aluno aluno 4,0K mar 7 10:05 ..
-rw---- 1 aluno aluno 474 mar 7 14:49 exemplo2.txt
-rw---- 1 aluno aluno 403 mar 7 14:38 exemplo.txt
-rw-rw-r- 1 aluno aluno 0 mar 7 10:05 teste1
-rw-r--- 1 aluno aluno 0 mar 7 10:40 teste2
-rw---- 1 aluno aluno 0 mar 7 10:41 teste3
```

&> : este redireciona, ao mesmo tempo, a saída padrão e a saída de erro.

Este redirecionador é muito utilizado, por exemplo, quando estamos compilando um programa e desejamos obter todas as mensagens.

Exemplo 2.9.3 Neste exemplo, estamos compilando o código fonte contido no arquivo "**teste.c**" e gerando o arquivo binário "**teste**"; durante o processo usamos o redirecionador para enviar tanto a saída padrão quanto a saída de erro para o arquivo "**mensagens.txt**"

```
gcc -o teste teste.c &> mensagens.txt
```

2> : realiza apenas o redirecionamento da saída de erro para um arquivo.

Exemplo 2.9.4 Para capturar apenas as mensagens de erro geradas durante o processo de compilação de um código, podemos utilizar:

```
gcc -o teste teste.c 2> erro.txt
```

Logo, com o uso dos redirecionadores de E/S é possível gerar dois arquivos independentes, ou seja, um contendo a saída padrão e um contendo a saída de erro.

Exemplo 2.9.5 Neste exemplo, estamos enviando as mensagens de erro para o arquivo **erro.txt** e as mensagens que seriam enviadas para a saída padrão estão sendo descartadas, ou seja, estão sendo enviadas para "**/dev/null**"

```
gcc -c teste teste.c 2> erro.txt > /dev/null
```

< : Faz com que os comandos obtenham a sua entrada a partir de um arquivo. Geralmente utilizamos este redirecionador em alguns *scripts* especiais.

Exemplo 2.9.6 Na linha de comando abaixo, estamos obtendo os dados, a serem ordenados, a partir do arquivo **origem.txt** e enviando para o arquivo **saida.txt** as mensagens geradas pelo comando **sort**.

```
sort < origem.txt > saida.txt
```

O comando acima organiza o conteúdo do arquivo "**origem.txt**" e envia o resultado para o arquivo "**saida.txt**"

| : este redirecionador de entrada e saída permite que um dado comando opere com a saída de um outro, ou seja, redireciona a saída de um comando para a entrada de outro. Ele funciona como um sistema de encanamento que interliga ambos os comandos.

Exemplo 2.9.7 Neste conjunto de exemplos, utilizamos o pipe ("|") para encaminhar a saída do comando **du** como a entrada do comando **sort**, e no ultimo comando adicionamos um terceiro redirecionador entre o comando **sort** e o comando **more**.

Para listar espaço em disco utilizado por todos os arquivos do diretório "**/bin**" começados pelas letras "**a**" ou "**b**", podemos utilizar:

```
du /bin/[a-b]*
     1088 /bin/bash
     732 /bin/brltty
     36 /bin/bunzip2
     2016 /bin/busybox
     36 /bin/bzcat
     0 /bin/bzcmp
     4 /bin/bzdiff
     0 /bin/bzegrep
     8 /bin/bzexe
     0 /bin/bzfgrep
     4 /bin/bzgrep
     36 /bin/bzip2
     16 /bin/bzip2recover
     0 /bin/bzless
     4 /bin/bzmore
```

Para ordenar a listagem, podemos redirecionar a saída do comando **du** para a entrada do comando **sort**.

```
du /bin/[a-b]* | sort -n
     0 /bin/bzcmp
     0 /bin/bzegrep
     0 /bin/bzfgrep
     0 /bin/bzless
     4 /bin/bzdiff
     4 /bin/bzgrep
     4 /bin/bzmore
     8 /bin/bzexe
     16 /bin/bzip2recover
     36 /bin/bunzip2
     36 /bin/bzcat
     36 /bin/bzip2
     732 /bin/brltty
     1088 /bin/bash
     2016 /bin/busybox
```

Para apresentar a listagem, da utilização do espaço de cada arquivo do mesmo diretório podemos utilizar o comando abaixo, mas como esta listagem tende a ocupar mais de uma tela adicionaremos um outro redirecionador que irá encaminhar a saída do comando **sort** para a entrada do comando **more**.

```
du /bin/* | sort -rn | more
```

2.10 Editores de Texto

As distrições *Linux* oferecem várias opções de editores de texto, que geralmente são utilizados no processo de edição dos arquivos de configuração. Dentre os editores mais utilizados e conhecidos podemos destacar pelo menos quatro, o primeiro é o ***vi***, um clássico do mundo *Unix*; o segundo é o ***vim***, que funciona da mesma forma que o bom e velho *vi*, mas possui alguns melhoramentos; o terceiro é o ***namo***, muito comum nas distribuições mais recentes e entre os usuários mais novos do *Linux* e finalmente o ***joe***, tão bom quando o nano porem menos conhecido.

2.10.1 vi

O primeiro editor que vamos conhecer é o ***vi***, ele é um dos mais usados no mundo *Linux*, sendo composto, basicamente, em dois modos de operação:

- **Modo Edição**: no qual é realizada a "inserção" do texto; para sair deste modo, basta pressionar a tecla "**Esc**";
- **Modo Comando**: utilizado para inserção dos comandos; é neste modo que o **vi** está quando ele é iniciado;

O **vi** suporta os seguintes comandos de **inserção**:

- **i**: insere o texto no início da linha.
- **A**: insere texto no final da linha.
- **o**: adiciona uma linha abaixo da atual.
- **O**: adiciona uma linha acima da atual.

Além destes, ainda temos os comandos utilizados para nos **movimentar pelo arquivo**:

- **ctrl+f**: pula para a tela seguinte.
- **ctrl+b**: pula para a tela anterior.
- **H**: salta para a primeira linha da tela.
- **L**: salta para a última linha da tela.
- ^: move para o primeiro carácter não branco da linha.
- **$**: move para o final da linha corrente.
- **nG**: move para a linha "**n**".
- **G**: move para a última linha do arquivo.
- **k**: move uma posição acima.
- **j**: move uma posição abaixo.
- **h**: move uma posição a esquerda.
- **l**: move uma posição a direita.

Ainda temos os comandos para **localizar textos** dentro do documento:

- **/palavra**: move para a próxima ocorrência da palavra (para repetir a busca pressione "**n**").
- **?palavra**: move para a ocorrência anterior da palavra ("**n**" para repetir).
- **ctrl+g**: exibe o nome do arquivo, o número da linha corrente e o número total de linhas.

O **vi** ainda fornece alguns comandos para **controlar as alterações** realizadas no texto:

- **u**: desfaz a ultima alteração.
- **U**: desfaz todas as modificações feitas na linhas.
- **rx**: substitui o texto corrente pelo indicado.
- **Rtexto**: substitui o texto corrente pelo indicado.
- **cw**: substitui a palavra corrente.

Finalmente, temos os comandos utilizados para **salvar as modificações**:

- **:wq**: salva o arquivo e sai.
- **:w [arquivo]**: salva o arquivo com um nome específico.
- **:w!**: salva o arquivo corrente de forma forçada.
- **:q**: sai da programa, se há mudanças envia advertência.
- **:q!**: sai do editor sem salvar as mudanças.
- **:wq!**: salva o arquivo e sai do programa

2.11 Localizando arquivos

Em uma distribuição *Linux* temos uma vasta quantidade de arquivos, sejam eles: binários, imagens, texto e etc. Devido a sua quantidade, localizar estes arquivos nem sempre é tarefa fácil, mas podemos contar com a ajuda de alguns comandos como o ***find***, o ***slocate***, o ***which*** e o ***whereis***.

Nas próximas seções estudaremos um pouco sobre cada um destes comandos e aprenderemos a distinguir quando utilizar cada um deles.

find

Este comando pode ser utilizado para localizar arquivos dentro da arvore de diretórios do *Linux*, esta busca é baseada nas características do arquivo como seu *nome*, a *data de criação*, a *data de ultima edição* e outras características. Apesar de ser extremamente versátil, abordaremos somente alguns de seus principais argumentos.

Sintaxe 2.11.1 : **find [caminho] [expressão]**

Onde:

- **caminho** : determina o ponto de partida da busca; geralmente um *path* relativo ou absoluto; quando não sabemos onde o arquivo pode ser localizado, um bom ponto de partida é a própria raiz da arvore de diretórios (“/”).
- **expressão** : determina o que deve ser ser “**buscado**”, dentre os valores que este argumento pode assumir destacam-se:
  - −**name “nome”** : realiza a busca baseada no nome do arquivo.
  - −**usr “usuario”** : realiza a busca baseada no nome do proprietário, ou seja do usuário que criou ou que mantém o arquivo.
  - −**group “grupo“** : realiza a busca baseada no grupo do proprietário.
  - −**perm “num“** : realiza a busca baseada na permissão absoluta do arquivo; útil para localizar arquivos com um permissões específicas.
  - −**type c** : realiza a busca baseada no tipo do arquivo, neste caso um arquivo comum.
  - −**type d** : realiza a busca de diretórios.
  - −**type l** : realiza a busca de arquivos do tipo **link**.
- −**print** : imprime, na saída padrão, a lista dos arquivos encontrados.

Exemplo 2.11.1 : Para localizar e imprimir a lista de todos os arquivos cujo nome seja **.bash**, localizados a partir da raiz da arvore de diretórios podemos utilizar:

```
find / -name .bash -print
```

2.11.1 slocate

Utilizamos este comando para localizar os arquivos armazenados localmente. O ***slocate*** trabalha de forma diferente do comando ***find***, pois em vez de realizar uma busca no sistema de arquivos, ele realiza uma busca em um banco de dados que contem a listagem dos arquivos; logo é importante manter as informações contidas no banco de dados atualizadas.

Sintaxe 2.11.2 : **slocate -u -n [numero] [string]**

Onde:

- ☑ **-u** : é utilizado para atualizar a base de dados da aplicação.
- ☑ **-n [numero]** : especifica um número máximo de resultados a serem retornados.
- ☑ **string**: especifica o nome do arquivo, ou parte do nome do arquivo, que se deseja localizar.

Exemplo 2.11.2 : Para localizar os cinco primeiros arquivos com o nome **find**, podemos utilizar:

```
slocate −n5 find
```

2.11.2 updatedb

O comando ***updatedb***, atualiza a base de dados utilizada pelo comando ***slocate*** para localizar arquivos disponíveis no sistema.

Sintaxe 2.11.3 : **updatedb −u**

Onde:

- ☑ **−u** : inicia a atualização, do banco de dados, a partir dos arquivos contidos do diretório "/" e recursivamente analisa cada um dos subdiretórios.

Exemplo 2.11.3 Para atualizar o banco de dados do ***slocate*** com a informações de todos os arquivos contidos na maquina, podemos utilizar:

```
updatedb −u
```

2.11.3 which

Este comando é extremamente útil para localizar programas executáveis ou *shell scripts*, que estão em um dos diretórios do caminho de busca, ou seja, os diretórios que estão presentes na variável ***PATH***.

Sintaxe 2.11.4 : **which [comando]**

Onde:

- ☑ **comando**: é o nome do comando ou *shell script* que se deseja localizar

Exemplo 2.11.4 A seguir utilizamos o comando ***which*** para localizar os binários do ***bash*** e do ***password***.

```
which bash
     /usr/bin/bash

which passwd
     /usr/bin/passwd
```

OBS: O conteúdo variável ***PATH*** determina em quais diretórios o *Linux* deve buscar os comandos que digitamos no *prompt* do *shell*. Dependendo da distribuição e do usuário logado o seu conteúdo pode ser diferente, ou seja, pode conter mais ou menos diretórios. Para exibir o conteúdo da variável, podemos utilizar o comando:

```
echo $PATH
```

2.11.4 whereis

Utilizamos este comando quando precisamos localizar: (i) o arquivo binário, (ii) o código fonte e (iii) as páginas de manual de um ou mais programas.

Sintaxe 2.11.5 : **whereis [-bms] [programa]**

Onde:

- ☑ **-b** = localiza o arquivo binário
- ☑ **-m** = localiza as páginas de manual
- ☑ **-s** = localiza o código fonte
- ☑ **programa** = nome do programa para o qual desejamos as informações

Exemplo 2.11.5 : Para localizar o arquivo binário, as páginas de manual e, se estiver instalado, o código fonte do programa ***reset***, podemos utilizar:

```
whereis -bms reset
     reset:  /usr/bin/reset /usr/share/man/man1/reset.1.gz
```

2.12 Manipulação de Texto

A forma como os dados são apresentados na saída de alguns comandos nem sempre atende a necessidade de todos usuários, muitas vezes precisamos que as informações sejam organizadas, em outros casos precisamos que apenas determinadas colunas sejam utilizadas e assim por diante.

Nesta seção serão apresentados alguns comandos, fornecidos pelo *Linux*, que nos auxiliam à retirar o máximo das informações obtidas de outros comandos.

2.12.1 sort

O primeiro comando que abordaremos é o ***sort***, que ordena as linhas de um arquivo, ou aquelas que foram recebias via redirecionadores de **e/s**, utilizando um *pipe* ("|"), e as retorna na a saída padrão.

Sintaxe 2.12.1 : **sort <arquivo>**
Onde:[leftmargin=0.6in]
- ☑ **arquivo** = nome do arquivo que precisa ser ordenado

Exemplo 2.12.1 Para exibir o conteúdo ordenado do arquivo **/etc/passwd**, podemos utilizar:

```
sort /etc/passwd
```

2.12.2 wc

Este comando realiza a contagem de (i) caracteres, (ii) palavras e (iii) linhas de um determinado arquivo, ou de um conteúdo recebido via redirecionadores de *e/s*.

Sintaxe 2.12.2 **wc -clw [arquivo]**
Onde:
- ☑ **arquivo** : é o nome do arquivo a ser analisado.
- ☑ **-c** : retorna a quantidade de caracteres.
- ☑ **-l** : retorna a quantidade de linhas.
- ☑ **-w** : retorna a quantidade de palavras.

Exemplo 2.12.2 Para exibir o número de usuários cadastrados no sistema, podemos contar a quantidade de linhas do arquivo que funciona como o banco de dados dos usuários, ou seja o "/etc/passwd"

```
wc -l /etc/passwd
```

2.12.3 head

Este comando imprime as "**n**" primeiras linhas de um arquivo.

Sintaxe 2.12.3 : **head –n num [arquivo]**

Onde:

- ☑ **arquivo** : é o nome do arquivo que desejamos imprimir as primeiras linhas. Este argumento pode ser suprimido quando utilizamos como entrada de dados as informações redirecionadas via um *pipe*.
- ☑ **-n** : número de linhas a serem exibidas

Exemplo 2.12.3 Para listar as quinze primeiras linhas do arquivo **/etc/passwd**, podemos utilizar o comando:

```
head -n 15 /etc/passwd
```

2.12.4 tail

Utilizamos este comando quando precisamos imprimir as últimas "**n**" linhas de um arquivo.

Sintaxe 2.12.4 : **tail -n [arquivo]**

Onde:

- ☑ **arquivo** : é nome do arquivo do qual desejamos extrair as ultimas "**n**" linhas.
- ☑ **-n** : quantidade de linhas a serem retornadas.

Exemplo 2.12.4 Para listar as ultimas cinco linhas do arquivo **/etc/passwd**, podemos utilizar o comando:

```
tail -n 5 /etc/passwd
```

2.12.5 cut

O comando ***cut***, nos permite selecionar quais colunas do texto desejamos apresentar; mesmo que o arquivo não pareça ser um arquivo formado por colunas muitas vezes podemos extrair suas colunas.

Sintaxe 2.12.5 : **cut -f -d [arquivo]**

Onde:

- ☑ **arquivo** : nome do arquivo do qual desejamos extrair as colunas.
- ☑ **-f** : define as colunas a serem exibidas.
- ☑ **-d** : especifica o delimitador de colunas a ser utilizado.

Exemplo 2.12.5 : Para listar o **Nome**, o **UID**, o **GID** e o **Shell** dos cinco primeiros usuários cadastrados no sistema, podemos utilizar o comando:

```
cut -f1,3,4,7 -d:  /etc/passwd | head -n5
     daemon:1:1:/usr/sbin/nologin
     bin:2:2:/usr/sbin/nologin
     sys:3:3:/usr/sbin/nologin
     games:5:60:/usr/sbin/nologin
     man:6:12:/usr/sbin/nologin
```

2.12.6 grep

O comando ***grep*** pode ser utilizado para localizar padrões de texto dentro de um ou vários arquivos. Este comando também pode ser utilizado para filtrar a saída de um determinado comando, que foi redirecionada utilizando-se do *pipe*

Sintaxe 2.12.6 : **grep [opções] [padrão] [arq1[arq2 [arq3 [... [arqn]]]]]**

Onde:

- ☑ **[arq1[arq2 [arq3 [... [arqn]]]]]** : representa a lista de nomes dos arquivos no qual desejamos realizar a pesquisa.
- ☑ **padrão** : representa a informação que desejamos localizar.
- ☑ **opções** : dentre as opções suportadas pelo **grep** podemos destacar:
  - ☑ **-C N**, **-N**, **–context=N**: exibe "N" linhas antes e "N" linhas depois da linha onde o padrão foi encontrado; útil quando estamos procurando algo dentro de um arquivo de configuração.
  - ☑ **-c**, **–count** : conta o total de linhas que coincidiram com o padrão;
  - ☑ **–color**, **–colour**: ativa a exibição de cores, destacando o trecho que coincide com o padrão;
  - ☑ **-i**, **–ignore-case** : Modo *case-insensitive*; quando ativo, o comando **grep** não diferencia as letras maiúsculas das minúsculas;
  - ☑ **-l**, **–files-with-matches** : em vez de exibir a saída normal, exibe apenas os nomes dos arquivos que coincidem com o padrão;
  - ☑ **-n**, **–line-number** : exibe o número da linha onde o padrão foi encontrado;
  - ☑ **-o**, **–only-matching** : exibe apenas o trecho da linha que coincide com o padrão;
  - ☑ **-R**, **-r**, **–recursive**: realiza busca recursiva;

Caso seja necessário utilizar um padrão de busca que não possa ser expresso com um texto simples, o **grep** nos permite utilizar expressões regulares, que nos podem realizar buscas mais sofisticadas e avançadas.

Quando da escrita deste material, estava disponível, disponível gratuitamente na Internet, um

excelente guia intitulado: Expressões Regulares – Guia de Consulta Rápida[1], escrito pelo Aurélio Marinho Jargas, que pode ser utilizado como base para o estudo das expressões regulares e de como podemos enriquecer nossos filtros no **grep**

Exemplo 2.12.6 : Para verificar se existe um usuário com o *login name* "**postfix**" cadastrado no sistema, podemos utilizar o comando ***grep*** para filtrar o conteúdo de arquivo **/etc/passwd**.

```
grep postfix /etc/passwd

    postfix:x:125:131::/var/spool/postfix:/usr/sbin/nologin
```

[1] https://aurelio.net/regex/guia/

2.13 Sistema de Arquivo

Assim como outros sistemas operacionais, as distribuições *Linux* fornecem várias ferramentas para o gerenciamento de discos, partições e do sistema de arquivo associado à elas. Nesta seção, apresentaremos alguns dos comandos que podemos utilizar em nosso dia à dia para esta finalidade.

2.13.1 fdisk

Utilizamos o comando *fdisk* para gerenciar as partições existentes no disco.

Sintaxe 2.13.1 : **fdisk -lus [particao]**

Onde:

- ☑ **partição** : é a descrição da partição de disco, ou do próprio disco, que se deseja examinar;
- ☑ **-l** : lista todas as partições de disco existentes na máquina;
- ☑ **-u** : exibe o tamanho em setores em vez de cilindros;
- ☑ **-s** : exibe o tamanho em blocos de partição;

Exemplo 2.13.1 : Para listar as informações sobre o primeiro disco *SATA*, instalado na máquina e suas partições, podemos utilizar o comando:

```
fdisk -l /dev/sda

Disco /dev/sda: 465,8 GiB, 500107862016 bytes, 976773168 setores
Unidades: setor de 1 * 512 = 512 bytes
Tamanho de setor (lógico/físico): 512 bytes / 4096 bytes
Tamanho E/S (mínimo/ótimo): 4096 bytes / 4096 bytes
Tipo de rótulo do disco: gpt
Identificador do disco: 2C6B0145-B429-4120-A3CE-7C1FE2CC29EF

Dispositivo  Início       Fim   Setores Tamanho Tipo
/dev/sda1      2048   1050623   1048576    512M Sistema EFI
/dev/sda2   1050624 976771071 975720448  465,3G Linux sistema de arquivos
```

O **fdisk**, foi um dos primeiros comandos desenvolvidos e desde então vem sendo largamente utilizado; mas atualmente temos opções mais intuitivas de serem utilizadas.

2.13.2 cfdisk

Assim como o **fdisk**, o **cfdisk** é um programa que roda de modo texto e que acompanha grande parte das distribuições. Quando o assunto é particionamento de disco ele podemos dizer que ele bem mais simples que o **fdisk** e por isto tem sido tão utilizado. O **cfdisk** é baseado em **curses**, o que lhe confere uma **tui** (*text user interface*) intuitiva, que aceitas os seguintes comandos:

Listagem dos comandos suportados pelo cdfisk

- **-b** : alterna o sinalizador de inicialização da partição atual, indicando se ela pode ou não "**bootavel**"
- **-d** : exclui a partição atual.
- **-h** : mostra a tela de ajuda (*help*).
- **-n** : cria uma nova partição a partir do espaço livre; por padrão o **cfdisk** utilizará todo o espaço livre disponível.
- **-q** : finaliza o programa sem escrever quaisquer alteração no disco.
- **-t** : altera o tipo de partição. Por padrão, novas partições são criadas como partições **Linux**.
- **-u** : salva a tabela atual de partições em um *script* **sfdisk**; que são compatíveis com o **cfdisk**, **fdisk**, **sfdisk** e outros aplicativos baseados na biblioteca **libfdisk**. Estes *scripts* podem ser utilizados para replicar o particionamento do disco em várias maquinas.
- **-W** : grava a tabela de partições em disco. Como isso pode destruir todos dados no disco, o usuário deve confirmar ou recusar a escrita digitando "**yes**" ou "**no**".
- **Seta para cima, seta para baixo** : move o cursor para a partição anterior ou seguinte
- **Seta para a esquerda, seta para a direita** : seleciona o item anterior ou seguinte do menu. Pressionando "**Enter**" executa-se o item selecionado.

2.13.3 mkfs

Este comando é utilizado para criar os sistemas de arquivo nas partições de disco, ou seja, ele realiza a formatação da partição e a prepara para uso.

Sintaxe 2.13.2 : **mkfs -t tipo -c partição**

Onde :

- **-f [tipo]**: especifica o tipo de sistema de arquivo a ser criado;
- **-c** : realiza a verificação de blocos ruins (*bad blocks*) antes de criar o sistema de arquivo
- **partição** : é a partição a ser manipulada.

Durante o processo de formatação, determinados o tipo do sistema de arquivo que utilizaremos, para cada uso há um sistema mais apropriado.

Exemplo 2.13.2 : Para formatar a **primeira** partição estendida do **primeiro** disco **sata**, utilizando o sistema de arquivo **ext4**, que é atual padrão utilizado no **Linux**, podemos utilizar:

```
mkfs -t ext4 -c /dev/sda5
```

2.13.4 fsck

Utilizamos o comando ***fsck*** quando precisamos verificar e/ou recuperar um sistema de arquivo no *Linux*.

Sintaxe 2.13.3 : **fsck -Aa [filesystem]**

Onde:

- ☑ **-A** : verifica todos os sistemas de arquivo existentes no arquivo "**/etc/fstab**", este arquivo mantém a listagem dos sistemas de arquivos que devem ser montados durante a carga do sistema.
- ☑ **-a :** repara o sistema de arquivo sem fazer nenhuma pergunta.
- ☑ **filesystem** : é a partição a ser manipulada

Algumas vezes, quando desligamos o sistema operacional de modo forçado e o mesmo não consegue fechar todos os arquivos, o sistema de arquivos pode ser corrompido, logo, ele não ficará acessível até que seja corrigido.

Exemplo 2.13.3 : Utilizando o comando abaixo, realizamos a verificação do sistema de arquivos da primeira partição estendida do primeiro disco *SATA*.

```
fsck -a /dev/sda5
```

2.13.5 du

O comando ***du*** exibe as informações sobre o consumo de espaço em disco de um determinado diretório ou arquivos.

Sintaxe 2.13.4 : **du -hks [objeto]**

Onde :

- ☑ **-h** : exibe o tamanho em formato humano, ou seja, na unidade de medida mais próxima (TBytes, GBytes, MBytes, KBytes, bytes).
- ☑ **-k** : exibe o tamanho em KBytes.
- ☑ **-h** : realiza a soma do espaço em disco de um determinado segmento, ou diretório.

Exemplo 2.13.4 : Para apresentar o quanto que cada subpasta contida no *homedir* do usuário está consumindo de espaço em disco, podemos utilizar:

```
du -hs ~/*

   5,2M    /home/aluno/Área de Trabalho
   23M     /home/aluno/Biblioteca do calibre
```

```
4,0K    /home/aluno/Desktop
20M     /home/aluno/Documentos
7,0G    /home/aluno/Downloads
6,9G    /home/aluno/Dropbox
4,0K    /home/aluno/Imagens
4,0K    /home/aluno/Modelos
4,0K    /home/aluno/Música
4,0K    /home/aluno/Público
24K     /home/aluno/snap
4,0K    /home/aluno/Vídeos
```

2.13.6 df

Este comando exibe a quantidade de espaço livre disponível em um determinando (i) disco, (ii) partição ou em (iii) todas as partições montadas.

Sintaxe 2.13.5 : **df -hk [objeto]**

Onde:

- **-h**: exibe as informações no formato humano.
- **-k**: exibe as informações em kbytes
- **objeto**: geralmente um diretório, que em alguns casos pode está associado a um sistema de arquivo, ou seja, à um ponto de montagem.

Exemplo 2.13.5 : Para listar as informações sobre o espaço livre em tdas as partições do disco, podemos utilizar:

```
df -h
```

```
Filesystem      Size  Used Avail Use% Mounted on
udev            3.0G  4.0K  3.0G   1% /dev
tmpfs           597M  692K  597M   1% /run
/dev/dm-0        95G  2.4G   87G   3% /
/dev/sda1       236M   72M  152M  33% /boot
```

Exemplo 2.13.6 : Para verificar qual o espaço livre na partição "/boot", podemos utilizar:

```
df -h /boot/
```

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       236M   72M  152M  33% /boot
```

2.13.7 mount

O comando ***mount*** anexa um sistema de arquivo à um diretório local, permitindo que os arquivos de uma determinada partição, ou areá compartilhada, por um servidor, sejam acessados de forma transparente.

Sintaxe 2.13.6 : **mount [-arw] [-t file_system] [dispositivo] [diretório]**

Onde:

- **-a** : monta todos os sistemas de arquivo que estão declarados no arquivo **/etc/fstab**;
- **-r** : monta o sistema de arquivo em no modo **read-only**, ou seja, somente leitura;
- **-w** : monta o sistema de arquivo no modo ***read-write***, permitindo que os arquivos sejam alterados, desde que o usuário tenha as permissões necessárias;

- **-t file_system** : indica o tipo do sistema de arquivo a ser montado. Os principais tipos sistemas de arquivo, suporados, são:
  - **vfat** : utilizado para montar as partições formatadas pelo *Windows* e pelo *DOS*, que usam os padrões **FAT16** e **FAT32** ;
  - **ext**: utilizado, principalmente, pelas primeiras versões do *Linux*; foi o primeiro sistema de arquivos criados unicamente para o ele, em 1992;
  - **ext2**: utilizado, principalmente, pelas versões mais antigas do *Linux*; já suportava discos com até **2 TB** e não suportava *journaling*;
  - **ext3**: este sistema fornece as mesmas funcionalidade do **Ext2**, mas já suportava *journaling*;
  - **ext4**: é a atual versão do padrão *ext*, sendo utilizada pela grande maioria das maquinas baseadas em *Linux*, principalmente aquelas que rodam a distribuição *Ubuntu Linux*.
  - **btrfs** : criando inicialmente pela *Oracle* e ao que tudo indica será o sucessor do *Ext4*.
  - **reiserfs** : A sua criação foi um avanço para sistemas *Linux*, mas foi substituído pelo *Reiser4*.
  - **zfs**: criado pela *SUN* para o sistema *Solaris*. Todo arquivo gravado por esse sistema possui um *checksum* e com isso o sistema de arquivos consegue identificar quando um arquivo está corrompido ou não.
  - **xfs**: lida muito bem com arquivos grandes, mas não se sai tão bem ao trabalhar com arquivos pequenos.
  - **jfs ou Journaled File System**: este sistema trabalha tão bem com arquivos grandes quanto com os pequenos, ele faz pouco uso do processador; pode ter seu

tamanho aumentando, mas não diminuído. Porém não está totalmente testado para *Linux*, logo o padrão **ext4** ainda é uma melhor opção.

- ☑ **nfs**: sistema de arquivo distribuído pela rede; baseia-se em um servidor que disponibiliza alguns de seus diretórios e de um cliente que monta este diretório remoto, como um diretório local;
- ☑ **iso9660**: sistema de arquivo padrão utilizados pelos **CD-R** e **CD-RW**.
- ☑ **ntfs**: este é o padrão utilizado no sistema de aquivo das versões mais recentes do *Windows*; O *NTFS* está disponível desde as versões do *Windows NT* e do *XP*.

☑ **dispositivo**: geralmente assume um valor semelhante à **/dev/sdxn**, onde:

- ☑ **x**: representa a letra associada a ordem do disco e pode assumir os valores a, b, c, d e etc.
- ☑ **n**: é o número que determina a partição de disco a ser utilizada; os valores de 1 até 4 identificam as partições primárias e os valores > 4 são usados para partições estendias;

☑ **diretório** : é o diretório, local, no qual o dispositivo deve ser montado; para determinar o diretórios que funcionará como ponto de montagem, podemos utilizar tanto um *path* relativo quanto um absoluto

Exemplo 2.13.7

```
mount -a
mount -t ntfs /dev/hda1 /mnt/win_c
mount -t vfat/dev/hda2 /mnt/wii_d
mount /mnt/cdrom
mount /mnt/floppy
mount -t nfs tux-pet:/home /home
```

2.13.8 umount

O comando ***umount***, "desmonta" um dado sistema de arquivo, que está anexado ao diretório especificado.

Sintaxe 2.13.7 umount -a -t file_system [diretório]

Onde:

- **-a** : desmonta todos os sistemas de arquivo que estão declarados no arquivo **/etc/fstab**; use com **cuidado**.
- **-t file_system** : indica tipo do sistema de arquivo a ser desmontado;
- **diretório** : é o diretório no qual o sistema de arquivo esta montado ou o próprio sistema de arquivo.

Exemplo 2.13.8

```
umount -a
umount /mnt/win_c
umount /mnt/win_d
umount /mnt/cdrom
umount /mnt/floppy
```

2.14 Gerenciamento de Processos

Todas as aplicações e serviços que estão sendo executadas em uma maquina rodando o *Linux*, recebe o nome de processo. Para cada aplicação temos pelo menos um processo, mas uma aplicação pode ser subdivida em vários processos.

Cada processo recebe uma identificação única, denominada de ***PID*** (*Process IDentification*), sendo que esta identificação geralmente é utilizada no processo de gerencia dos processos. Além do **PID**, um processo possui vários outros atributos, os quais serão estudados e apresentados posteriormente.

Nesta seção, serão apresentados alguns comandos que podemos utilizar para gerenciar os processos ativos em uma máquinas rodando o sistema operacional *Linux*

2.14.1 ps

Este comando é utilizado para listar "todos" os processos que estão atualmente rodando na máquina. Ele funciona de forma análoga ao comando ***ls***, mas em vez de listar os arquivos, ela apresenta as informações sobre os processos.

Sintaxe 2.14.1 : **ps -aeflmuxw**

Onde:

- **-a** : lista todos processos que pertencem ao usuário.
- **-e** : exibe as variáveis de ambiente relacionadas aos processos.
- **-f** : exibe a árvore de execução dos processos.
- **-l** : exibe mais campos no resultado.
- **-m** : mostra a quantidade de memória ocupada por cada processo.
- **-u** : apresenta as informações utilizando um modo de saída mais amplo; exibe o nome do usuário que iniciou determinado processo e a hora em que isso ocorreu;
- **-x** : lista os processos que não possuem um terminal de controle associado, como por exemplo os *daemons* do sistema e dos serviços ativos.
- **-w** : se o resultado de processo não couber em uma linha, essa opção faz com que o restante seja exibido na linha seguinte.

A saída do comando **ps** possui um formato tabular, o que facilita a leitura e entendimento das informações exibidas. Dentre as informações apresentadas pelo ***ps***, podemos destacar:

- **USER** : nome do usuário, dono, do processo;
- **UID** : número de identificação do usuário, dono, do processo;
- **PID** : número de identificação do processo;
- **PPID** : número de identificação do processo pai, ou seja, aquele que criou o processo atual;
- **%CPU** : porcentagem de uso de CPU;

- **%MEM** : porcentagem da memória usada;
- **VSZ** : indica o tamanho virtual do processo;
- **RSS** : sigla de *Resident Set Size*, indica a quantidade de memória usada (em KB);
- **TTY** : identifica o terminal ao qual o processo está associado;
- **START** : informa a hora em que o processo foi iniciado;
- **COMMAND** : indica nome do comando que gerou o processo;
- **PRI** : valor da prioridade associada processo; esta valor pode ser alterado pelo comando **renice**.
- **NI** : valor, preciso, da prioridade (geralmente igual aos valores de PRI);
- **WCHAN** : mostra a função do ***kernel*** onde o processo se encontra em modo suspenso;
- **STAT** : indica o estado atual do processo, sendo representado por uma das letras:
  - **R** : executável;
  - **D** : em espera no disco, aguardando o fim do troca de dados (*i/o*) com o disco;
  - **S** : suspenso;
  - **T** : interrompido;
  - **Z** : zumbi.

A seguir temos alguns exemplos de uso do comando **ps**:

Exemplo 2.14.1 : A forma mais simples de se utilizar o **ps** é executa-lo sem passar nenhum parâmetro, neste caso ele retornará apenas os processos ativos no terminal corrente:

```
ps
```

```
  PID TTY          TIME CMD
 3412 pts/0    00:00:00 bash
 4609 pts/0    00:00:00 ps
```

Exemplo 2.14.2 : Para listar todos os processos ativos no sistema, podemos utilizar ou o **-e** ou o **-A**:

```
ps -e
```

```
  PID TTY          TIME CMD
    1 ?        00:00:02 systemd
    2 ?        00:00:00 kthreadd
    4 ?        00:00:00 kworker/0:0H
    6 ?        00:00:00 mm_percpu_wq
    7 ?        00:00:00 ksoftirqd/0
    8 ?        00:00:03 rcu_sched
    9 ?        00:00:00 rcu_bh
   10 ?        00:00:00 migration/0
                [...continua...]
```

Exemplo 2.14.3 : Para obter mais informações sobre os processos podemos utilizar os argumentos **-f** ou **-F**

```
ps -ef
```

```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 08:46 ?        00:00:02 /sbin/init splash
root         2     0  0 08:46 ?        00:00:00 [kthreadd]
root         4     2  0 08:46 ?        00:00:00 [kworker/0:0H]
root         6     2  0 08:46 ?        00:00:00 [mm_percpu_wq]
root         7     2  0 08:46 ?        00:00:00 [ksoftirqd/0]
root         8     2  0 08:46 ?        00:00:03 [rcu_sched]
root         9     2  0 08:46 ?        00:00:00 [rcu_bh]
                    [...continua...]
```

```
ps -eF
```

```
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
root         1     0  0 56389  9256   2 08:46 ?        00:00:02 /sbin/init splash
root         2     0  0     0     0   3 08:46 ?        00:00:00 [kthreadd]
root         4     2  0     0     0   0 08:46 ?        00:00:00 [kworker/0:0H]
root         6     2  0     0     0   0 08:46 ?        00:00:00 [mm_percpu_wq]
root         7     2  0     0     0   0 08:46 ?        00:00:00 [ksoftirqd/0]
root         8     2  0     0     0   2 08:46 ?        00:00:03 [rcu_sched]
root         9     2  0     0     0   0 08:46 ?        00:00:00 [rcu_bh]
                    [...continua...]
```

Exemplo 2.14.4 : Para listar todos os processos de um determinado usuário podemos utilizar o parâmetro **-u** e para listar todos os processos do usuário que está conectado ao terminal, podemos utilizar o argumento **-x**:

```
ps -u
```

```
  PID TTY          TIME CMD
 2557 ?        00:00:00 systemd
 2558 ?        00:00:00 (sd-pam)
 2569 ?        00:00:00 x-session-manag
 2570 ?        00:05:07 Xorg
 2573 ?        00:00:00 xrdp-chansrv
 2601 ?        00:00:02 dbus-daemon
 2631 ?        00:00:00 ssh-agent
 2647 ?        00:00:00 gvfsd
 2652 ?        00:00:00 gvfsd-fuse
 2661 ?        00:00:00 at-spi-bus-laun
               [...continua...]
```

Exemplo 2.14.5 : Para listar todos os processos de um determinado grupo podemos utilizar o argumento **-g**:

```
ps -g root
```

```
  PID TTY          TIME CMD
    1 ?        00:00:02 systemd
    2 ?        00:00:00 kthreadd
    4 ?        00:00:00 kworker/0:0H
    6 ?        00:00:00 mm_percpu_wq
    7 ?        00:00:00 ksoftirqd/0
    8 ?        00:00:03 rcu_sched
    9 ?        00:00:00 rcu_bh
   10 ?        00:00:00 migration/0
               [...continua...]
```

Exemplo 2.14.6 : Para listar todos os processos associados à um determinado executável podemos utilizar o argumento **-C**:

```
ps -C chrome
```

```
  PID TTY          TIME CMD
 4766 ?        00:00:03 chrome
 4777 ?        00:00:00 chrome
 4781 ?        00:00:00 chrome
 4809 ?        00:00:00 chrome
 4874 ?        00:00:00 chrome
 4963 ?        00:00:00 chrome
 4967 ?        00:00:00 chrome
```

Exemplo 2.14.7 : Para listar os processos de forma hierarquica, podemos utilizar o argumento **-H**:

```
ps -eH
```

```
  PID TTY          TIME CMD
 1281 ?        00:00:00   kerneloops
 1313 ?        00:00:00   xrdp-sesman
 2556 ?        00:00:00     xrdp-sesman
 2569 ?        00:00:00       x-session-manag
 2631 ?        00:00:00         ssh-agent
 2680 ?        00:00:01         mate-settings-d
 2685 ?        00:00:07         marco
 2691 ?        00:00:00         mate-panel
 3404 ?        00:00:56           mate-terminal
 3422 pts/1    00:00:00             bash
 4755 pts/1    00:00:03               htop
 4747 pts/0    00:00:00             bash
 5165 pts/0    00:00:00               ps
 5084 pts/2    00:00:00             bash
 5163 pts/2    00:00:00               ssh
 4766 ?        00:00:04           chrome
               [...continua...]
```

Exemplo 2.14.8 : Para listar todos os processos associados à um determinado terminal podemos utilizar o argumento **-t**:

```
ps -t pts/2

  PID TTY          TIME CMD
 5084 pts/2    00:00:00 bash
 5163 pts/2    00:00:00 ssh
```

Exemplo 2.14.9 : Para determinar quais processos que mais estão utilizando **CPU**, podemos utilizar o usar:

```
ps -eo pid,ppid,cmd,%mem,%cpu -sort=-%cpu | head -n 15

   PID  PPID CMD                         %MEM %CPU
  4442  1437 /usr/sbin/xrdp               0.2 12.4
  2570  2556 /usr/lib/xorg/Xorg :10 -aut  1.4  1.5
  4755  3422 htop                         0.0  0.9
  4766  2691 /opt/google/chrome/chrome    1.9  0.4
  3015     1 /home/luisrodrigoog/.dropbo  1.4  0.2
  3404  2691 mate-terminal                0.2  0.2
  1219  1210 /usr/lib/xorg/Xorg -core :0  0.3  0.1
  1611  1610 /usr/sbin/slick-greeter      0.2  0.1
     1     0 /sbin/init splash            0.0  0.0
     2     0 [kthreadd]                   0.0  0.0
     4     2 [kworker/0:0H]               0.0  0.0
     6     2 [mm_percpu_wq]               0.0  0.0
     7     2 [ksoftirqd/0]                0.0  0.0
     8     2 [rcu_sched]                  0.0  0.0
```

Exemplo 2.14.10 : Para determinar quais processos estão utilizando mais memória, podemos utilizar o usar:

```
ps -eo pid,ppid,cmd,%mem,%cpu -sort=-%mem | head -n 15
```

```
  PID  PPID CMD                         %MEM %CPU
 4766  2691 /opt/google/chrome/chrome    1.9  0.3
 2570  2556 /usr/lib/xorg/Xorg :10 -aut  1.4  1.5
 3015     1 /home/luisrodrigoog/.dropbo  1.4  0.2
 2888  2569 /usr/bin/nextcloud           0.7  0.0
 4874  4781 /opt/google/chrome/chrome -  0.7  0.0
 2714  2569 caja                        0.5  0.0
 4809  4766 /opt/google/chrome/chrome -  0.5  0.0
 2879  2569 megasync                    0.4  0.0
 1219  1210 /usr/lib/xorg/Xorg -core :0  0.3  0.1
 4777  4766 /opt/google/chrome/chrome -  0.3  0.0
 2569  2556 x-session-manager           0.3  0.0
 2870  2569 /usr/bin/python3 /usr/bin/b  0.3  0.0
 4963  4781 /opt/google/chrome/chrome -  0.2  0.0
 3404  2691 mate-terminal               0.2  0.2
```

Exemplo 2.14.11 : Para listar todos os processos, com e sem terminal, utilizando o formato logo, podemos utilizar:

```
ps -aux
```

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
systemd+   950  0.0  0.0 143984  3128 ?        Ssl  08:46   0:00 /lib/systemd/systemd-tim
root       957  0.0  0.0 110476  3412 ?        Ssl  08:46   0:01 /usr/sbin/irqbalance --f
root       959  0.0  0.1 170828 17372 ?        Ssl  08:46   0:00 /usr/bin/python3 /usr/bi
root       960  0.0  0.0 288432  7296 ?        Ssl  08:46   0:00 /usr/lib/accountsservice
root       961  0.0  0.0  31628  3116 ?        Ss   08:46   0:00 /usr/sbin/cron -f
root       962  0.0  0.0 427260  9176 ?        Ssl  08:46   0:00 /usr/sbin/ModemManager
syslog     963  0.0  0.0 263036  4508 ?        Ssl  08:46   0:00 /usr/sbin/rsyslogd -n
root       965  0.0  0.0   4552   756 ?        Ss   08:46   0:00 /usr/sbin/acpid
root       966  0.0  0.0  26420  4960 ?        Ss   08:46   0:00 /usr/sbin/smartd -n
root       967  0.0  0.0  70708  6260 ?        Ss   08:46   0:00 /lib/systemd/systemd-log
root       972  0.0  0.0 503408 11508 ?        Ssl  08:46   0:04 /usr/lib/udisks2/udisksd
```

2.14.2 kill

Utilizamos o comando ***kill*** quando precisamos enviar sinais (mensagens) para os processos; dentre os sinais um dos mais utilizados é o **–9**, que serve para matar o processo que é o alvo do sinal.

Sintaxe 2.14.2 : **kill -l [sinal] [pid]**

Onde:

- ☑ -l : lista os nomes dos sinais.
- ☑ -sinal : número do sinal,os mais importantes são:
  - ☑ 9 : Mata o processo;
  - ☑ 1 (HUP): recarrega um processo
- ☑ pid : é a identificação numérica do processo que deve receber o sinal.

A seguir temos alguns exemplos de como podemos utilizar o comando **kill**.

Exemplo 2.14.12 : Para obter a lista de todos os sinais suportados pelo comando, podemos usar:

```
kill -L

 1) SIGHUP  2) SIGINT  3) SIGQUIT  4) SIGILL  5) SIGTRAP
 6) SIGABRT  7) SIGBUS  8) SIGFPE  9) SIGKILL 10) SIGUSR1
11) SIGSEGV 12) SIGUSR2 13) SIGPIPE 14) SIGALRM 15) SIGTERM
16) SIGSTKFLT 17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP
21) SIGTTIN 22) SIGTTOU 23) SIGURG 24) SIGXCPU 25) SIGXFSZ
26) SIGVTALRM 27) SIGPROF 28) SIGWINCH 29) SIGIO 30) SIGPWR
31) SIGSYS 34) SIGRTMIN 35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```

Exemplo 2.14.13 : Para recarregar o *daemon* do servidor **mysql**, podemos utilizar o comando abaixo:

```
kill -hup ‘cat /var/run/mysqld/mysqld.pid‘
```

Exemplo 2.14.14 : Para encerrar todos os processos que o usuário possui controle, podemos usar:

```
kill -9 -1
```

Exemplo 2.14.15 : Para obter o significado do sinal 11, podemos usar:

```
kill -l 11
     SEGV
```

2.14.3 killall

O comando ***killall***, envia um sinal para todos os processos que possuem o mesmo nome; assim, este comando permite enviar o mesmo sinal para vários processos distintos, ou seja, que possuem *PIDs* diferentes, mas que possuem o mesmo nome.

Este comando é extremamente útil quando precisamos encerrar um grupo de processos semelhantes.

Sintaxe 2.14.3 : **killall [-ei] [-l] [-sinal] [nome]**

Onde:

- ☑ **-e** : envia um sinal para os processos que combinam exatamente com o nome informado;
- ☑ **-i** : solicita uma confirmação antes de enviar um sinal;
- ☑ **-l** : lista os nomes dos sinais;
- ☑ **-sinal** : indica o número do sinal que será enviado; sendo que os mais comuns são:
  - ☑ **1 (HUP)** : Recarrega o processo;
  - ☑ **9** : Mata o processo;
- ☑ **nome** : é o nome do processo que deve receber o sinal.

Exemplo 2.14.16 : Para recarregar o daemon do named e todos os processos associados, podemos utilizar o comando:

```
killall -HUP named
```

2.14.4 top

O comando ***top*** exibe a lista dos processos, ativos no sistema, de acordo com a utilização de *CPU*; por este motivo, é um dos comandos mais utilizados para monitorar a carga d sistema. Ele é um comando tipo interativo, ou seja, uma vez carregado ele aceita vários comandos internos.

Sintaxe 2.14.4 : **top -dn -i -n -n qt_interacoes -u username -o fieldname -O**

Onde:

- **-d n** : atualiza a lista de processos à cada "n" segundos.
- **-i** : exibe apenas os processos que estão executando.
- **-n qt_interacoes**: determina quantas interações serão realizadas entes de finalizar o programa.
- **-u username** : lista apenas os processos do usuário especificado
- **-o fieldname** : especifica o nome do campo utilizado para classificar a listagem; podemos acrescentar um "**+**" ou "**-**" ao nome do campo, o "**+**" forçará a classificação de alta para baixa, enquanto um "**+**" garantirá uma ordem de baixa para alta.
- **-O** : imprime os nomes dos campo disponíveis para serem utilizados com a opção -o

O ***top*** suporta os seguintes **comandos interativos**:

- **h** ou **?**: Help.
- **ENTER** ou **Barra de Espaço**: Atualizar a lista dos processo.
- **d** : permite alterar o tempo de atualização.
- **e** : altera a unidade de medida que representa a quantidade de memória utilizada por cada processo.
- **E** : altera a unidade de medida que representa, no resumo do sistema, a quantidade de memória utilizada.
- **f** : permite definir quais camos devem ser exibidos.
- **i** : lista apenas os processos em execução.
- **k**: mata os processos definidos pelo identificador (ID)
- **r** : altera a prioridade de um processo;
- **W** : salva as opções em uso para serem utilizadas sempre que o comando for carregado;
- **q** : sai do programa;

Exemplo 2.14.17 A seguir temos o exemplo de uma saída gerada pelo comando **top**:

```
top - 16:19:06 up  7:33,  0 users,  load average: 0,23, 0,07, 0,03
Tarefas: 225 total,   1 em exec., 173 dormindo,   0 parado,   0 zumbi
%CPU(s):  1,7 us,  0,4 sis,  0,0 ni, 97,8 oc,  0,1 ag,  0,0 ih,  0,0 is  0,0 tr
KB mem : 16298344 total, 13185088 livre,  1074088 usados,  2039168 buff/cache
KB swap:  2097148 total,  2097148 livre,        0 usados, 14781856 mem dispon.

  PID USUARIO   PR  NI    VIRT    RES    SHR S  %CPU %MEM    TEMPO+ COMANDO
 4442 xrdp      20   0   81168  39432  19116 S   4,3  0,2  13:57.44 xrdp
 2570 luisrod+  20   0  895280 237592 115024 S   1,0  1,5   6:05.74 Xorg
 4755 luisrod+  20   0   34244   5120   3816 S   1,0  0,0   0:39.64 htop
 3404 luisrod+  20   0  652724  43192  30512 S   0,7  0,3   1:04.80 mate-terminal
 2888 luisrod+  20   0 1966888 128888  98516 S   0,3  0,8   0:09.36 nextcloud
 5529 luisrod+  20   0   44692   4280   3508 R   0,3  0,0   0:00.09 top
    1 root      20   0  225556   9256   6652 S   0,0  0,1   0:02.51 systemd
    2 root      20   0       0      0      0 S   0,0  0,0   0:00.00 kthreadd
    4 root       0 -20       0      0      0 I   0,0  0,0   0:00.00 kworker/0:0H
    6 root       0 -20       0      0      0 I   0,0  0,0   0:00.00 mm_percpu_wq
    7 root      20   0       0      0      0 S   0,0  0,0   0:00.15 ksoftirqd/0
    8 root      20   0       0      0      0 I   0,0  0,0   0:04.18 rcu_sched
    9 root      20   0       0      0      0 I   0,0  0,0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S   0,0  0,0   0:00.00 migration/0
   11 root      rt   0       0      0      0 S   0,0  0,0   0:00.04 watchdog/0
   12 root      20   0       0      0      0 S   0,0  0,0   0:00.00 cpuhp/0
```

Na primeira linha temos as informações sobre:

- ☑ A hora atual - **16:19:06**
- ☑ A quanto tempo o sistema está ligado - **7:33**
- ☑ A quantidade de usuários conectados localmente - **0 users**. O usuário estava conectado via Remote Desktop Procol, logo não havia usuários conectados localmente
- ☑ A média de carga de 5, 10 e 15 minutos respectivamente - **0,23, 0,07, 0,03**

Na segunda linha temos as informações sobre os processos, ou seja: o número total de processos; destes, quais estão nos estados de: (i) executando (***running***), (ii) dormindo (***sleeping***), (iii) parado (***stopped***) ou (iv) modo zumbi (***zombie***).

Na terceira linha temos as informações sobre o estado da CPU e seus diferentes tempos uso:

- **us** : usuário (*user*) - tempo de CPU gasto na execução dos processos de usuário;
- **Sy** : sistema (*sytem*) - tempo de CPU gasto na execução de processos do *kernel*;
- **Id** : ocioso (*idle*) - tempo de CPU em inatividade
- **Wa** : tempo para *I/O* - tempo de CPU esperando a conclusão de operação de entrada/-saída no disco (*I/O*)
- **hi** : tempo gasto servindo interrupções de *hardware*
- **Si** : tempo de CPU servindo interrupções de *software*.

As duas próximas linhas apresentam o uso de memória, a quarta linha refere-se a memória física e a quinta linha à memória virtual (*swap*). A memória física é apresentada como: memória total disponível, memória usada, memória livre e memória usada para *buffers*. Da mesma forma, a *swap*: total, usada, espaço de troca livre e em cache. Todas em unidade *Kilobyte*.

Após as linhas de cabeçalho temos as informações sobre os processos que estão dividias nas seguintes colunas:

- ***PID*** (Identificador do processo) : é a identificação do processo (identificador único)
- **USUÁRIO (*USER*)** : é o usuário proprietário do processo.
- **PR (Prioridade)** : é a prioridade de agendamento do processo. Alguns valores neste campo são "***RT***". Isso significa que o processo está sendo executado em tempo real (*Real Time*).
- **NI** : os valores mais baixos significam maior prioridade.
- **VIRT** : é a quantidade de memória virtual usada pelo processo.
- **RES** : é o tamanho da memória usada. Residente na memória física e não na área de troca (*swap*).
- **SHR (*Share* – compartilhada)** : SHR é a memória compartilhada usada pelo processo.
- **S (State – estado)** : é o estado do processo. Ele pode ter um dos seguintes valores:
  - **D** – ininterrupto
  - **R** – executando
  - **S** – dormindo
  - **T** – rastreado ou parado
  - **Z** – zumbi
- **% CPU** : é a porcentagem de tempo de CPU que a tarefa tem usado desde a última atualização.
- **% MEM** : é a percentagem de memória física disponível usada pelo processo.
- **TEMPO +** : é o tempo total de CPU que a tarefa tem usado desde o início (precisão

de centésimo de segundo)

- ☑ **COMANDO** : é a descrição do comando que foi utilizado para iniciar o processo.

2.14.5 &

Quando adicionamos o simbolo & após o nome de um comando e pressionamos "*ENTER*", o comando será enviado para ser executado, diretamente em segundo plano; ao ser enviado para *background*, o *prompt* do terminal é liberado e o usuário pode continuar a executar outros aplicativos.

Sintaxe 2.14.5 : **[comando] &**

Onde:

- ☑ **comando** : representa o comando que deve ser enviado para o segundo plano, assim como ocorre com outros comandos podemos utilizar tanto o *path* relativo quanto o absoluto para determinar a localização do comando que deverá ser executado.

Alguns comandos, como o *find*, podem demorar um pouco para retornarem alguma resposta ou gerarem alguma saída para o usuário, uma forma de trabalhar com estes comandos é executa-los em segundo plano e capturar toda a saída que eles irão produzir para um arquivos texto, que poderá ser consultado posteriormente.

Exemplo 2.14.18 : A linha de comando a seguir, utiliza o comando ***find*** para pesquisar pelo arquivo **sources.list**, mas em vez de esperar por uma resposta o comando será enviado para o segundo plano e o **stdout** e o ***stderr*** serão enviados para o arquivo **local.txt.**

```
find / -name sources.list &> local.txt &
```

OBS: Mesmo quando enviamos os processos para segundo plano, eles ficam associados ao terminal no qual foram iniciados, finalizando-se o terminal o processo também será finalizando

2.14.6 control+c e control+z

Utilizamos a combinação de teclas "**control+c**" quando desejamos matar/encerrar um processo que está rodando no primeiro plano.

á a combinação de teclas "**control+z**" é utilizada quando precisamos parar, ou melhor, pausar, um processo que está rodando no primeiro plano. Neste caso, posteriormente, podemos trazer o processo de volta para o primeiro plano ou podemos envia-lo para o segundo plano.

2.14.7 jobs

Lista todos os trabalhos que estão sendo executados ou que estão suspensos/parados.

Sintaxe 2.14.6 jobs [-l]

Onde:

- ☑ **-l** : inclui os *ids* dos processos

A seguir temos um exemplo de uso do comando ***jobs***, combinado à outros.

Exemplo 2.14.19 : Neste exemplo utilizamos o comando ***find*** para realizar duas buscas, a primeira foi "**pausada**", com o **ctrl+z**, e a segunda foi enviada, diretamente para o segundo plano. Depois de pausar o primeiro **find** executamos o comando **jobs** e recebemos a informação que o processo **11587** estava **Parado**; depois que enviamos o segundo **find** para o segundo plano, executamos novamente o comando **jobs**, que nos informou que há dois processos na fila, o primeiro (**11587**) parado e o segundo (11652) sendo executado.

```
find / -name sources.list &> saida.txt
ctrl+z
     ^Z
     [1]+ Parado find / -name sources.list &> saida.txt
jobs -l
     [1]+ 11587 Parado find / -name sources.list > saida.txt

find / -name sources.list &> saida2.txt &
     [2] 11652
jobs -l
     [1]+ 11587 Parado find / -name sources.list &> saida.txt
     [2]- 11652 Executando find / -name sources.list > saida2.txt &
```

2.14.8 fg

Utilizamos o comando **fg** para trazer um processo que estava sendo executado no segundo plano de volta para o primeiro plano.

Sintaxe 2.14.7 fg

Exemplo 2.14.20 Neste exemplo utilizamos o comando **find** para localizar o arquivo **faillog**, como este comando geralmente demora para dar uma resposta ele foi enviado para o segundo plano, após algum utilizamos o comando **fg** para traze-lo de volta para o primeiro plano.

```
find / -name faillog -print &
fg find
```

2.14.9 bg

Faz com que um comando, que foi "**pausado**", seja executado em segundo plano.

Sintaxe 2.14.8 bg

Este comando é útil quando sabemos que programa tomará muito tempo para ser executado, logo podemos capturar a sua saída e envia-lo para o segundo plano

Exemplo 2.14.21

```
find / -name faillog > local.txt 2> erro.txt
CTRL+Z
bg
```

2.15 Compactação e Backup

Durante a administração de um maquina, seja ela uma servidora ou o *desktop* de um usuário, nos deparamos com a necessidade de realizarmos alguns *backups*, ou compactar algumas pastas antes de iniciarmos alguma modificação no sistema; em outros momentos temos a necessidade de extrair o conteúdo de um arquivo compactado que recebemos via *e-mail* ou que baixamos da *Internet*. Independentemente da necessidade, esta seção apresenta alguns comandos que nos ajudam no processo de compactação e extração de dados.

2.15.1 gzip e gunzip

Utilizamos o comando ***gzip*** para compactar arquivos, neste processo, geralmente, o comando substitui o arquivo original pelo compactado; já o comando ***gunzip*** realiza o processo de extração dos arquivos que foram compactados utilizando-se o comando anterior.

Sintaxe 2.15.1 Tanto o comando ***gzip*** quanto o ***gunzip*** possuem uma sintaxe muito semelhante e aceitam quase que os mesmos argumentos:

gzip [-cdflrt] [arquivo]
gunzip [-cflrt] [arquivo]

Onde:

- ☑ **-c** = deixa o arquivo original intacto.
- ☑ **-d** = descompacta o arquivo.
- ☑ **-f** = força a compactação, mesmo quando já existe um arquivo com o mesmo nome.
- ☑ **-l** = lista os arquivos que estão dentro do arquivo compactado.
- ☑ **-r** = compacta recursivamente descendo na estrutura de diretórios.

Exemplo 2.15.1 : Neste exemplo, criamos e acessamos o diretório **~/LabInfo/Aula16**, em seguida realizamos uma cópia do arquivo **passwd** com o nome **senhas**, que foi compactado com o comando **gzip** e depois descompactado pelo **gunzip** e finalmente geramos uma cópia compactada o arquivo **senhas** a qual demos o nome de **senhas_compactadas.gz** :

```
mkdir -p ~/LabInfo/Aula16
cd ~/LabInfo/Aula16
cp /etc/passwd ./senhas
ls -l
gzip senhas
ls -l
gunzip senha.gz
ls -l
gzip -c senhas > senhas_compactadas.gz
ls -l
gunzip senhas_compactadas.gz
ls -l
```

2.15.2 compress e uncompress

Utilizamos o comando ***compress*** como uma outra opção para compactar arquivos, e o comando ***uncompress*** é utilizado para extrair o conteúdo destes arquivos. Os arquivos compactados por estes comandos geralmente possuem a extensão **.Z** e uma taxa de compressão menor que os compactados pelo **gzip**.

Sintaxe 2.15.2 : Estes dois comandos utilizam a mesma sintaxe :

compress [-vr] [arquivo]

uncompress [-rv] [arquivo]

Onde:

- ☑ **-v** = apresenta a porcentagem de redução no tamanho do arquivo.
- ☑ **-r** = opção recursiva, somente deve ser utilizada quando o arquivo especificado for um diretório.

Exemplo 2.15.2 :

```
compress -rv ~/teste
unconpress -rv ~/teste
```

2.15.3 bzip2 e bunzip2

O comando ***bzip2*** realiza uma compressão de arquivos com uma taxa superior ao ***gzip***, e o ***bunzip2*** é o comando utilizado para descompacta-los. Os arquivos com extensão **.bz2**, geralmente foram criados utilizando-se este mecanismo de compactação.

Sintaxe 2.15.3 : A seguir temos a sintaxe de ambos os comandos :

bzip2 -dtv19 [arquivo]

bunzip2 [arquivo]

Onde:

- ☑ **arquivo**: é o nome do arquivo a ser manipulado.
- ☑ **-d**: descompacta o arquivo.
- ☑ **-t**: testa a integridade do arquivo.
- ☑ **-v**: mostra a taxa de compressão obtida.
- ☑ **-9**: utiliza a melhor taxa de compressão.
- ☑ **-1**: realiza uma compressão rápida.

Vejamos como podemos utilizar o comando **bzip2** e **bunzip2** para compactar e descompactar uma série de arquivos:

Exemplo 2.15.3 : Neste exemplo, criamos e entramos na pasta "**~/LabInfo/Aula16/teste-bzip**", em seguida copiamos todos os arquivos que começam com a letra "**u**" e que estão dentro da pasta "**/bin**" para o diretório corrente; com o comando **ls** verificamos quais foram os arquivos copiados e com o comando **bzip2** compactamos todos os arquivos e para cada um dos novos arquivos adicionados a extensão "**.bz2**"

```
mkdir -p ~/LabInfo/Aula16/teste-bzip
cd ~/LabInfo/Aula16/teste-bzip
cp -v /bin/u* ./
ls -l
bzip2 *
ls -l
```

Exemplo 2.15.4 : Utilizando os comandos abaixo, extraímos todos os arquivos compactados pelo comando **bzip2** e que possuem a extensão "**.bz2**".

```
mkdir -p ~/LabInfo/Aula16/teste-bzip
cd ~/LabInfo/Aula16/teste-bzip
bunzip *.bz2
```

2.15.4 zip e unzip

O ***zip*** é o compactador de arquivos *default* do *Windows*, mas desde alguns anos ele tem se tornado muito comum no mundo *Linux*; o parceiro deste comando no processo de descompressão é o ***unzip***, geralmente os arquivo compactados por estes utilitários possui a extensão "**.zip**"

Sintaxe 2.15.4 : A seguir temos a sintaxe de ambos os comandos :

zip -FRT -P [senha] [mone_do_arq_compactado] [arquivos]

unzip [mone_do_arq_compactado]

Onde:

- ☑ **arquivo**: é o nome do arquivo a ser (des)compactado.
- ☑ **-F**: caso o arquivo esteja corrompido tanta corrigi-lo.
- ☑ **-P [senha]**: acrescenta uma senha no arquivo.
- ☑ **-R**: compactação recursiva.
- ☑ **-T**: testa a integridade do arquivo.

A seguir temos alguns exemplos de uso dos comandos *zip* e *unzip*:

Exemplo 2.15.5 : Utilizando os comandos a seguir, criamos e entramos na pasta "~/LabInfo/Aula16/teste-zip", em seguida utilizamos o comando zip para criar um arquivo compactado e com senha contendo todos os arquivos da pasta "/bin":

```
mkdir -p ~/LabInfo/Aula16/teste-zip
cd ~/LabInfo/Aula16/teste-zip
zip -P senha binarios.zip /bin/*
```

Exemplo 2.15.6 : Para extrair o *backup*, criado no exemplo anterior, podemos utilizar o comando:

```
unzip binarios.zip
```

2.15.5 tar

Originalmente o comando ***tar*** foi criado para empacotar vários arquivos e/ou diretórios em um único pacote, que seria enviado para uma unidade de *backup* do tipo *DAT* ou *DLT*; com o tempo ele passou a ser utilizado como uma ferramenta de *backup* que utiliza a própria máquina para guardar os arquivos. Por padrão, o **tar** não realiza a compactação dos pacotes, mas podemos solicitar que ele use um dos comandos estudados anteriormente para esta finalidade.

Sintaxe 2.15.5 : **tar -crtxzjpv -f [destino] [origem]**

Onde:

- ☑ **−r**, **−−append** : anexa arquivos no final de um pacote **tar**;
- ☑ **−t**,**−−list** : lista o conteúdo de um arquivo;
- ☑ **−c** : cria um arquivo **tar**;
- ☑ **−t**, **−−list** : listar o conteúdo de um arquivo
- ☑ **−x**, **−−extract**, **−−get** : extrai o arquivo **tar**;
- ☑ **−z** : compacta, o arquivo **tar**, utilizando o **gzip**;
- ☑ **−v** : modo "falador" ou verbose;
- ☑ **−f** : especifica o nome do pacote a ser criado, ou extraído.
- ☑ **−j**, **−−bzip2** : (des)compacta o arquivo utilizando **bzip2**.
- ☑ **−p**, **−−preserve-permissões** : extrair informações sobre permissões de arquivo (padrão para superusuário)
- ☑ **−z**, **−−gzip** : (des)compacta o arquivo utilizando **gzip**
- ☑ **−−lzip** : (des)compacta o arquivo utilizando **lzip**
- ☑ **−−lzma** : (des)compacta o arquivo utilizando **lzma**

A seguir temos um exemplo do comando tar para geração de um pacote contendo o backup da pasta "teste-backup".

Exemplo 2.15.7 : Iniciamos o exemplo criando o diretório "**~/LabInfo/Aula16/teste-backup**", em seguida mudamos o diretórios corrente para "**~/LabInfo/Aula16**", iniciamos a cópia de todos os arquivos contidos no diretórios "**/bin**" para a "**~/LabInfo/Aula16/teste-backup**" e finalmente, utilizamos o comandos *tar* para gerar o arquivo de backup com o nome de "**pacote.tar**"

```
mkdir -p ~/LabInfo/Aula16/teste-backup
cd ~/LabInfo/Aula16
cp -v /bin/* ./teste-backup
tar -cvf pacote.tar ~/teste/
```

Exemplo 2.15.8 : Para verificar se o arquivo de *backup* foi criado com sucesso utilizamos o o comando **ls**, em seguida removemos todo o conteúdo do diretório "~/LabInfo/Aula16/teste-backup", removemos a pasta vazia utilizando o comando **rmdir**; finalmente realizamos a extração do arquivo de *backup* utilizando o comando **tar**. Para garantir que os arquivos foram restaurados com sucesso entramos da sub-pasta recém criada e listamos o seu conteúdo e voltamo ao diretórios original.

```
ls -l
cd ./teste ; rm * ; cd ..
rmdir ./teste
tar -xvf pacote.tar
cd ./teste ; ls -l ; cd ..
```

Exemplo 2.15.9 : Este ultimo comando, gera uma versão compactada do *backup* da sub-pasta "**teste**" e de seu conteúdo.

```
tar -xzvf pacote2.tgz ./teste/
```

2.16 Data e Hora

Grande parte do bom funcionamento de um sistema operacional depende da sua capacidade de gerenciar a contagem do "tempo". Neste seção, conheceremos alguns comandos que o *Linux* nos oferece para esta finalidade.

2.16.1 cal

Este comando pode ser utilizado para exibir o calendário de um determinado mês ou ano; quando não fornecemos o mês e/ou o ano que desejamos os atuais serão utilizados

Sintaxe 2.16.1 **cal –y [opcoes] [[mes] ano]**

Onde:

- **mes** : é o valor numérico do mês do calendário
- **ano** : é o ano do calendário
- Dentre as opções aceitas podemos destacar:
  - **-h**: Desativa o destaque, geralmente aplicado a data atual.
  - **-j** : exibe os dias julianos, ou seja, os dias são numerados a partir de 1 de janeiro.
  - **-m mês** : exibe o calendário do mês especificado.
  - **-y** : exibe um calendário para o ano especificado.
  - **-3** : exibe o mês anterior, atual e próximo;
  - **-1** : exibe apenas o mês atual. Este é o padrão.
  - **-C** : muda para o modo **cal**.
  - **-N** : alterna para o modo **ncal**.
  - **-M** : as semanas começam na segunda-feira.
  - **-S** : as semanas começam no domingo.

A seguir temos alguns exemplos de uso do comando **cal**:

Exemplo 2.16.1 : Para listar o calendário do mês de fevereiro em 1994, podemos utilizar

```
cal 02 1994

      Fevereiro 1994
   do se te qu qu se sá
          1  2  3  4  5
    6  7  8  9 10 11 12
   13 14 15 16 17 18 19
   20 21 22 23 24 25 26
   27 28
```

Exemplo 2.16.2 : Para listar o calendário do mês atual, do anterior e do próximo, podemos utilizar:

```
cal -3
```

```
   Fevereiro 2019          Março 2019           Abril 2019
do se te qu qu se sá  do se te qu qu se sá  do se te qu qu se sá
                1  2                   1  2      1  2  3  4  5  6
 3  4  5  6  7  8  9   3  4  5  6  7  8  9   7  8  9 10 11 12 13
10 11 12 13 14 15 16  10 11 12 13 14 15 16  14 15 16 17 18 19 20
17 18 19 20 21 22 23  17 18 19 20 21 22 23  21 22 23 24 25 26 27
24 25 26 27 28        24 25 26 27 28 29 30  28 29 30
                      31
```

2.16.2 date

Utilizamos este comando para exibir e/ou alterar a data e a hora atual do sistema.

Sintaxe 2.16.2

date [-u|–utc|–universal] [MMDDhhmm[[CC]YY][.ss]]

date [OPTION]... [+FORMAT]

Onde:

- ☑ **MM** : mês com dois dígitos.
- ☑ **DD** : dia com dois dígitos.
- ☑ **hh** : hora com dois dígitos.
- ☑ **mm** : minuto com dois dígitos.
- ☑ **CCYY** : ano com quatro digito.

- ☑ Dentre as **opções** suportadas temos:
  - ☑ **-u**, **–utc**, **–universal** : imprimie ou define a data utilizando o Tempo Universal Coordenado (**UTC**)

- ☑ Dentre os **formatos** suportados destacam-se
  - ☑ **%%** : para apresentar o simbolo %;
  - ☑ **%a** : nome do dia da semana de forma abreviada;
  - ☑ **%A** : nome do dia da semana;
  - ☑ **%b** : nome do mês abreviado;
  - ☑ **%B** : nome do mês;
  - ☑ **%c** : data e hora local - ter 26 mar 2019 10:11:45 -03;
  - ☑ **%C** : seculo utilizando dos dígitos;

- **%d** : dia do mês;
- **%D** : data no formato mês/dia/ano;
- **%e** : dia do mês, representado utilizando-se sempre dois caracteres, quando ele é menor do que 10 será inserido um espaço antes dele;
- **%F** : data no formato ano-mes-dia;
- **%g** : últimos dois dígitos do ano;
- **%G** : ano utilizando 4 dígitos
- **%H** : hora (00..23)
- **%I** : hora (01..12)
- **%j** : dia do ano (001..366)
- **%k** : hora utilizando sempre dois caracteres (0..23);
- **%l** : hora utilizando sempre dois caracteres (1..12);
- **%m** : mês (01..12)
- **%M** : minuto (00..59)
- **%n** : pula linha
- **%N** : nanosegundos (000000000..999999999)
- **%p** : AM or PM;
- **%P** : o mesmo que %p, mas em letras minusculas
- **%q** : *quarter* do ano (1..4)
- **%r** : hora local no formato - **10:13:06**
- **%R** : hora e minutos no formato de 24h
- **%s** : quantidade de segundos desde 01/01/1970 às 00:00:00 UTC
- **%S** : segundos (00..60)
- **%t** : insere um *tab*
- **%T** : hora no formato %H:%M:%S
- **%u** : dia da semana, sendo que 1 representa a segunda
- **%U** : semana do ano, sendo domingo o primeiro dia da semana (00..53) - **1553606010**
- **%V** : semana do ano, sendo segunda o primeiro dia da semana (01..53)
- **%w** : dia da semana (0..6); sendo domingo igual a 0
- **%x** : data utilizando a representação local (15/03/2019)
- **%X** : hora utilizando a representação local (15:23:06)
- **%y** : ultimos digitos do ano (00..99)
- **%Y** : ano
- **%z** : valor do time zone (-0300)
- **%:z** : valor do time zone (-03:00)
- **%::z** : valor do time zone (-03:00:00)
- **%:::z** : valor do time zone com a precisão necessária (e.g., -03, -03:30)
- **%Z** : abreviação do nome do Time zone (EDT)

- Por padrão o comando **date**, quando necessário, preenche os valores numéricos com zeros (0), mas podemos alterar este padrão utilizando um dos **modificadores**:
  - **- (hifem)** : não preenche o campo;
  - **_ (underline)** : preenche com espaços;
  - **0 (zero)** : preenche com zeros
  - **^** : se possível, utiliza letra maiúsculas
  - **#** : se possível, inverte de maiúscula para minusculas; e vice-e-versa.

Vejamos alguns exemplos de como podemos utilizar o comando *date*.

Exemplo 2.16.3 : Para alterar a data e a hora atual para **15/02/1994** e **16:30:01**, podemos utilizar o comando:

```
date 021516301994
```

Exemplo 2.16.4 : Para apresentar a data atual utilizando três formatos distintos, podemos utilizar :

```
date +%D
     03/18/19
date +%y-%m-%d
     19-03-18
date +%Y-%m-%d
     2019-03-18
```

2.17 Comandos de Vídeo

Nesta seção apresentamos alguns comandos genéricos relacionados ao *display*.

2.17.1 clear

Utilizamos este comando para limpar a tela do nosso terminal.

Sintaxe 2.17.1 **clear**

Exemplo 2.17.1

clear

2.17.2 echo

O comando *echo* é utilizando quando desejamos exibir mensagens na tela, ou quando precisamos exibir o conteúdo de uma variável; geralmente empregamos este comando em *scripts*, para informar ao usuário sobre uma determinada condição.

Sintaxe 2.17.2

echo -neE "Texto"
echo -neE $VARIAVEL

Onde:

- **"Texto", $VARIAVEl** : é o texto ou a variável cujo conteúdo deve ser impresso.
- **-n** : não gera a quebra de linha após a impressão da mensagem ou da variável.
- **-e** : habilita a interpretação dos "*backslash escapes*"
- **-E** : desabilita a interpretação dos "*backslash escapes*", este é o comportamento padrão.

- Dentre os "***backslash escapes***", suportados, podemos destacar:
  - **** : imprime uma barra invertida.
  - **\a** : emite um *beep*.
  - **\b** : *backspace*.
  - **\c** : não produz mais nenhuma saída.
  - **\f** : *form feed*.
  - **\n** : pula linha.
  - **\r** : retorno de carro.
  - **\t** : insere uma tabulação horizontal (*tab*)
  - **\a** : insere uma tabulação vertical
  - **\0NNN** : imprime o carácter cujo valor está expresso no padrão **octal**.
  - **\xHH** : imprime o carácter cujo valor está expresso no padrão **hexadecimal**.

A seguir temos alguns exemplos do uso deste comando:

Exemplo 2.17.2 : Para apresentar o conteúdo da varável ***PATH***, que contem a lista dos diretórios, nos quais o *shell* consulta pelos comandos digitados no seu *prompt*, podemos utilizar :

```
echo $PATH
     /sbin:/bin:/usr/bin:/usr/local/bin:/snap/bin
```

Exemplo 2.17.3 : Neste exemplo utilizamos o comando *echo* para gerar a mensagem “Ola, Mundo . ”

```
echo -ne "Ola, \t Mundo \t.  \n"
     Ola, Mundo .
```

3. Comandos Básicos - Lista de Exercícios

Esta parte da apostila contem alguns exercícios , que visam auxiliar a fixação dos comandos apresentados no capítulo anterior.

3.1 Lista – Gerenciando Arquivos I

1. Construa a seguinte estrutura de diretório:

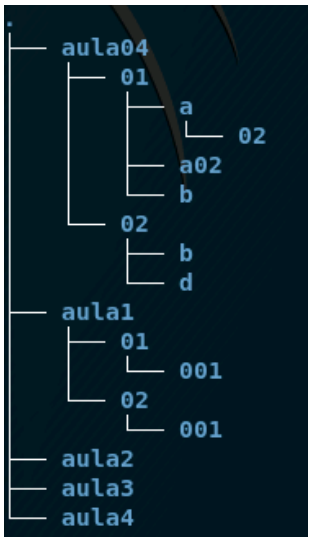

Figura 3.1: Estrutura de Diretórios.

2. Informe qual é a sequência a de comandos necessária para criar o arquivo **listagem.txt**, contendo a listagem de todos os arquivos do diretório atual, incluindo os arquivos ocultos, no formato longo.

3. Informe qual é o comando que copia para o diretório **/home/aluno/backup** todos os arquivos que estão do diretório **/etc**, bem como de todos os que estão em seus subdiretórios.
4. Informe qual é o comando que apaga todos os arquivos do diretório **/home/aluno/backup**, sem que nada seja questionado ao usuário e que ainda gere o arquivo **apagados.txt**, com a listagem de todos os arquivos que foram removidos.
5. Informe qual é a sequência de comandos necessária para: armazenar no arquivo **calendario.txt** o calendário do mês de agosto de 1976.
6. Informe qual é o comando utilizado para exibir, na tela, o conteúdo do arquivo **/etc/inittab**, uma tela por vez.
7. Informe qual é o comando necessário para exibir, na tela, o conteúdo do arquivo **/etc/passwd**, sem que o mesmo seja paginado

3.2 Lista - Gerenciamento de Arquivo II

1. Informe o comando que lista as **10** primeiras linhas do arquivo **/etc/passwd**.
2. Informe o comando que lista as **5** últimas linhas do arquivo **/var/log/message**.
3. Informe qual é a sequência de comandos que lista da 3^a à 5^a linha do arquivo **/etc/inittab**.
4. Informe o comando necessário para se criar um *link* simbólico com o nome de **teste.lnk** para o arquivo **/bin/bash**.
5. Informe a sequência de comandos necessária para exibir na tela da linha **10** até a linha **11** do arquivo **/var/log/messages**.

3.3 Lista - Gerenciamento de Arquivos III

1. Informe qual e a sequência de comandos para: procurar por todos os arquivos que estão no computador do usuário e possuem o nome igual a "**exemplo.txt**".
2. Informe qual e a sequência de comandos para: procurar por todos os arquivos que estão dentro do "**homedir**" do usuário e que possuem como extensão "**c**" e em seguida redirecionar o nome dos arquivos encontrados para o arquivo "**arquivos.txt**" e as mensagens de erro para o arquivo "**erros.txt**" ambos localizados no "**homedir**" o usuário.
3. Informe qual é a sequência de comandos para: armazenar no arquivo "**comando.txt**" a localização do arquivo binário, do código fonte e das paginas de manual do comando "**slocate**".
4. Qual é o comando que cria o arquivo "**/home/aluno/log.txt**", contento apenas a data e a hora atual.
5. Informe qual é a sequência de comandos necessários para se criar o arquivo **usuarios.txt**, contendo a listagem, em ordem alfabética, de todos os usuários cadastrados na máquina.
6. Informe a sequência de comandos necessários para armazenar no arquivo **usuarios2.txt** a primeira e a segunda coluna do arquivo **/etc/passwd**.
7. Informe a sequência de comandos necessária para ordenar o arquivo **/etc/passwd**, na ordem alfabética inversa, e exibir, na tela, a 3^a e a 4^a linha.

8. Informe a sequência de comandos necessária para criar o arquivo **./10primeiros.txt** com as **10** primeiras linhas do arquivo **/etc/passwd**, ordenado alfabeticamente.
9. Informe a sequência de comandos utilizada para criar o arquivo **./lista-ordenada.txt** com o conteúdo do arquivo **/etc/group** ordenado alfabeticamente. Em seguida copie a 1^a, 5^a, 2^a, 9^a e 4^a. linhas, deste arquivo, que acabou de ser criado e nesta ordem, para o arquivo **./nova-lista.txt**.
10. Informe a sequência de comandos necessária para:
    (a) Entrar no **homedir** do usuário;
    (b) Criar um diretório chamado **copia**;
    (c) Entrar no diretório **copia**;
    (d) Copiar para o diretório atual, de forma recursiva, todos os arquivos que estão no diretório **/var/log**;
    (e) Mudar o **proprietário** de todos os arquivos do diretório atual, bem como de seus subdiretórios, para a usuária "**ssattler**";
    (f) Mudar o grupo de todos os arquivos do diretório atual, bem como de seus subdiretórios, para o grupo "aluno";
    (g) Mudar a permissão de todos os arquivos do diretório atual, bem como de seus subdiretórios, de tal forma que o proprietário tenha acesso total, o grupo tenha acesso de leitura e as demais pessoas não tenham nenhum tipo de acesso;

3.4 Lista - Gerenciamento de Processos

1. Qual é o comando que lista, na tela, todos os processos, com e sem terminal no formato amplo, que pertencem ao usuário **root**?
2. Qual é o comando que lista, na tela, todos os processos, cujo nome é **bash**, no formato amplo, ordenados pelo nome do usuário e não pelo **PID**?
3. Qual é o comando que lista, na tela, todos os processos, com e sem terminal, no formato amplo uma tela por vez?
4. Qual é a linha de comando que envia para o arquivo "~/processos.txt" todos os processos, com e sem terminal, no formato amplo e ordenados pelo nome do usuário que o disparou?
5. Quais são os comandos necessários para se descobrir o **PID**, somente dos processos cujo nome é **bash**, e então enviar um sinal para matar todos os processos **bash** que pertencem ao usuário **aluno**?
6. Qual é o comando que envia um sinal para interromper, parar a execução, de todos os processos cujo nome é **wget**?
7. Qual é o comando que acorda, volta a fazer o processo rodar, todos os processos cujo nome é **wget**?
8. Qual é o comando que gera um processo, que irá rodar somente no segundo plano e no final de sua execução terá gerado o arquivo "~/mp3.txt" contendo uma listagem de todos os arquivos "**.mp3**" que existem na máquina?
9. Qual é a sequência de passos necessários para executar o comando "**wget** `http://www.`

`dicasl.unicamp.br/`" inicialmente no primeiro plano, e em seguida enviar este processo para o segundo plano?

10. Qual é o comando que gera o arquivo **~/trabalhos_na_fila.txt** contendo os dados de todos os processos que estão na fila de execução ou na fila de espera.

3.5 Lista – Compressão de Arquivos e Backup

1. Qual é o comando necessário para criar um pacote "**tar**", com o nome de "**backup.tar**" e contendo todos os arquivos do **homedir** do usuário? A saída deste comando deve ser redirecionada para o arquivo **~/arquivos.txt**
2. Qual é o comando necessário para criar um pacote "**tar**", com o nome de "**backup.tar**", contendo todos os arquivos do **homedir** do usuário. A saída deste comando deve ser redirecionada para o arquivo **~/arquivos.txt** de tal forma que todos os arquivos fiquem na ordem alfabética.
3. Quais são os comandos necessários para realizar o **backup** do diretório **/etc** para o pacote "**~/backup.tar**", de tal forma que a sua saída seja direcionada para o arquivo "**~/lista_arquivo.txt**". Em seguida, devemos incluir os arquivos do diretório **/bin** no pacote "**~/backup.tar**" e a saída deste segundo comando também deve ser direcionado para o arquivo "**~/lista_arquivo.txt**"?
4. Qual é o comando necessário para gerar o pacote "**backup.tgz**" contendo todos os arquivos do diretório "**/usr/local/bin**"?
5. Utilizando-se o **gunzip** e o **tar**, como proceder para conseguir acessar os arquivos que estão dentro do pacote "**backup.tar.gz**"
6. Qual é o comando necessário para gerar o pacote **~/bmp.tar.Z** contendo todos os arquivos **.jpg** do diretório atual?
7. Qual é o comando utilizado para localizar todos os arquivos que pertencem ao usuário "**Lab1**", que estão dentro do diretório **/home** e em seguida gerar um pacote "**.tar**" contendo todos estes arquivos
8. Qual é o comando utilizado para localizar todos os arquivos que pertencem ao grupo "**alunos**", que estão dentro do diretório **/home** e em seguida gerar um pacote "**.tar**" contendo todos estes arquivos. Feito isto compacte este arquivo com o comando **gzip**?
9. Qual é o comando utilizado para localizar todos os arquivos do tipo **MP3**, que estão dentro do diretório **/home** e em seguida gerar um pacote "**.tar**" contendo todos estes arquivos. Finalmente compacte este arquivo dando origem a um arquivo com terminação "**.Z**"
10. Qual é o comando utilizado para localizar todos os arquivos que não pertencem a **nenhum usuário válido**, que estão dentro do diretório **/home** e em seguida gerar um pacote "**.tar**" contendo todos estes arquivos. Finalmente compacte este arquivo com o comando **bzip**?
11. Informe qual é a sequência de comandos necessários para:
    - ☑ Entrar no **homedir** do usuário;
    - ☑ Criar um diretório chamado **copia**;
    - ☑ Entrar no diretório **copia**;
    - ☑ Copiar para o diretório atual de forma recursiva todos os arquivos que estão no diretório

/usr/local;

- ☑ criar um pacote com o nome de "**programas.tar**" contendo todos os arquivos do diretório atual, bem como dos seus subdiretórios;
- ☑ E em seguida compactar este arquivo como o **bzip2**

12. Informe qual é a sequência a de comandos necessários para:
    (a) Entrar no **homedir** do usuário;
    (b) Criar um diretório chamado "**~/restore**";
    (c) Descompactar o arquivo "**~/backup/programas.tar.gz**"
    (d) Entrar no diretório "**restore**" e extrair os arquivos que estão dentro do pacote que foi descompactado pelo comando anterior

3.6 Lista - Revisão

1. Qual é a vantagem do **Unix** sobre os sistemas operacionais anteriores a ele?
2. Em qual linguagem foi desenvolvida a primeira geração do **Unix**?
3. Qual foi a vantagem de se utilizar a linguagem "**C**" na segunda geração do **Unix**?
4. O processo de instalação de um **Linux** pode ser dividido em quais etapas?
5. Quais são as informações necessárias para se ter acesso a uma determinada partição de disco? Mostre um exemplo de como se realiza esta identificação.
6. Quais são os comandos associados aos "diretórios" ? Qual é a função de cada um?
7. Quais são todos os comandos associados à copia, remoção e à movimentação de arquivos? Dê um exemplo do funcionamento de cada um e explique o que ele está fazendo.
8. Quais são todos os comandos que podemos utilizar para visualizar um determinado arquivo? Dê um exemplo de cada um.
9. Quais são todos os redirecionadores de entrada e saída? Qual é a função de cada um? Dê um exemplo para uso de cada.
10. Quais são todos os metacaracteres? Qual é a função de cada um? Dê um exemplo do uso para cada.

Índice Remissivo