

APOSTILA DE SISTEMAS OPERACIONAIS - I LABORATÓRIO

UNIX

COMANDOS

Prof. Renato Jensen

ÍNDICE

I - INTRODUÇÃO

1 - HISTÓRICO	1
2 - ESTRUTURA	1
2.1 - Características Gerais	2
2.2 - Acesso ao Sistema	2
2.3 - Shell	3
2.4 - Organização de Usuários no UNIX	3

II - PARTIÇÃO

1 - DEFINIÇÃO	4
---------------------	---

III - SISTEMA DE ARQUIVOS DO UNIX

1 - DEFINIÇÃO	5
2 - ESTRUTURA HIERÁRQUICA	5
3 - CAMINHOS	6
4 - ARQUIVOS	7
4.1 - CARACTERÍSTICAS	7
4.2 - TIPOS DE ARQUIVOS	8

IV - COMANDOS: NOTAÇÃO E DOCUMENTAÇÃO

1 - NOTAÇÃO GENÉRICA DE COMANDOS	9
2 - DOCUMENTAÇÃO ONLINE	9

V - REDIRECIONAMENTO DE E/S E PIPELINES

1 - DEFINIÇÃO	10
2 - COMANDOS DE REDIRECIONAMENTO DE E/S	11
2.1 - >	11
2.2 - <	11
2.3 - 2>	11
3 - COMANDOS DE PIPELINES	12
3.1 - 	12

VI - PRINCIPAIS COMANDOS

1 - COMANDOS DE USO GERAL	13
1.1 - clear	13
1.2 - cal	13
1.3 - date	13
1.4 - passwd	13
2 - COMANDOS DE MANIPULAÇÃO DE DIRETÓRIOS	14
2.1 - pwd	14
2.2 - ls	14
2.3 - cd	14
2.4 - mkdir	14
2.5 - rmdir	14
3 - COMANDOS DE MANIPULAÇÃO DE ARQUIVOS	15
3.1 - cat	15
3.2 - more	15
3.3 - tail	15
3.4 - cp	15
3.5 - mv	15
3.6 - rm	16
3.7 - find	16

VII - OUTROS COMANDOS

1 - COMANDOS DE FILTROS	17
1.1 - wc	17
1.2 - grep	17

VIII - PERMISSÕES E ACESSO A ARQUIVOS

1 - DEFINIÇÃO	18
2 - TIPOS DE ACESSOS	18
3 - PERMISSÕES	19
4 - COMANDOS DE MANIPULAÇÃO DE PERMISSÃO DE ACESSO	20
4.1 - chmod	20
4.2 - chown	20
4.3 - chgrp	20

I - INTRODUÇÃO

1 - HISTÓRICO

O UNIX é um dos mais populares sistemas operacionais pelo mundo inteiro por causa da grande base de suporte e distribuição. Ele foi originalmente desenvolvido como um sistema multitarefa para minicomputadores e *mainframes* em meados da década de 1970.

Qual a real razão da popularidade do UNIX? Os *hackers* achavam que o UNIX era a Coisa Certa - O Verdadeiro Sistema Operacional. Conseqüentemente o desenvolvimento do Linux por um grupo de *hackers* do UNIX que queriam colocar suas mãos no seu próprio sistema.

Versões do UNIX existem para muitos sistemas - indo de computadores pessoais a supercomputadores tal como o Cray Y-MP. A maioria das versões do UNIX para PCs são completamente caras.

O Linux é uma versão de distribuição livre do UNIX desenvolvido primeiramente por Linus Torvalds (torvalds@kruuna.helsinki.fi) da Universidade de Helsinki na Finlândia. O Linux foi desenvolvido com a ajuda de muitos programadores UNIX e especialistas através da Internet, permitindo que qualquer pessoa com conhecimento suficiente e iniciativa pudesse desenvolver e alterar o sistema. O kernel do Linux utiliza o código da AT&T (System V para PC 386) projetado na *Free Software Foundation* em Cambridge, Massachusetts. Assim, programadores do mundo inteiro tem contribuído para o crescimento do software.

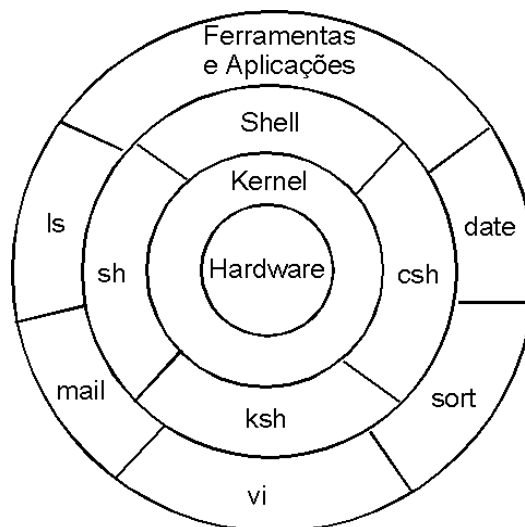
O Linux foi inicialmente desenvolvido como um projeto de lazer por Linus Torvalds. Ele foi inspirado por Minix, um pequeno sistema operacional desenvolvido por Andy Tanenbaum, e a primeira discussão sobre o Linux foi no *newsgroup* da USENET comp.os.minix.

Nenhuma propaganda foi feita com a versão 0.01 do Linux em Agosto de 1991. Esta versão ainda não era executável, pois continha somente os fontes do kernel ainda rudimentares, e assumia que você tinha acesso à máquina Minix para compilar e rodar o sistema.

Em outubro de 1991, Linux anunciou a primeira versão **oficial** do Linux, versão 0.02, quando já era possível executar bash (o GNU Bourne Again Shell) e o gcc (GNU C compiler). A partir de então mais pessoas passaram a participar do desenvolvimento do sistema.

2 - ESTRUTURA

A figura abaixo apresenta a arquitetura do UNIX de forma simplificada. Pode-se visualizar o sistema como um conjunto de níveis, desde o hardware até aplicações dos usuários, passando pelo núcleo (*kernel*) e programas básicos do sistema.



Arquitetura do sistema UNIX

De acordo com a figura, acima das aplicações do sistema (**as**, **cpp**, etc.) podem ser construídas outras aplicações do usuário, que utilizam os serviços das camadas inferiores para atingir seus propósitos. Um exemplo é o compilador padrão **C** (**cc**), que invoca o pré-processador **C** (**cpp**), o assembler (**as**) e um *link* editor (**ld**), todos programas separados da camada de nível imediatamente interior.

2.1 - Características Gerais

A portabilidade é uma característica marcante do UNIX, já que apenas uma pequena parte do código é dependente de especificações da máquina. Logo, a maior parte do sistema necessita apenas de uma recompilação quando é mudada a plataforma de hardware.

O UNIX é um sistema operacional de tempo compartilhado (*time sharing*), cujo código é escrito em sua grande maioria na linguagem C. Vários processos podem estar executando no sistema. Um processo pode gerar novos processos, criando as figuras de processos “pais” e “filhos”. O escalonamento do uso dos recursos de execução da CPU pelos vários processos é realizado por um algoritmo de prioridade bastante simples, porém eficiente.

Vários usuários podem estar usando o sistema ao mesmo tempo (*multiusuário*), e cada um deles pode ter num dado momento uma ou mais sessões abertas. Os processos gerados em cada sessão competem entre si e com outros processos no sistema pelo uso da CPU.

O sistema de arquivos do UNIX é organizado de forma hierárquica, numa estrutura de árvore. A partir de um diretório raiz vão sendo criados sub-diretórios, que também podem possuir sub-diretórios, e assim por diante.




2.2 - Acesso ao Sistema

Para acessar o sistema, o usuário deve ter sido previamente cadastrado pelo administrador do sistema, onde receberá um nome de usuário (*username*) e uma senha (*password*). Ao acessar o sistema o usuário será solicitado a digitar seu *username* no campo *login*, e a senha de acesso no campo *password*. Quando o usuário digita a senha ela não é ecoada na tela, por questões de segurança.

Uma vez identificado pelo sistema, o usuário recebe a linha de comando (*prompt*) onde poderá executar seus comandos. A partir deste momento o usuário passa a interagir com o programa Shell. As respostas geradas pela execução dos comandos serão mostradas na tela.

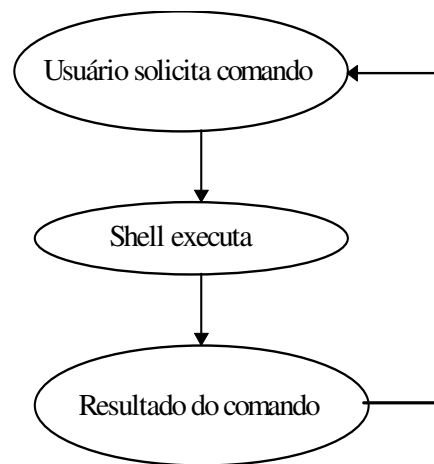
Para encerrar a sessão digite o comando *exit* na linha de comando ou as teclas Ctrl-D.

Atenção:

-  Quando a conexão for feita pelo administrador do sistema (*username = root*) o *prompt* a ser exibido é o #.
-  O sistema UNIX é sensível a letras maiúsculas e minúsculas tratando-as como letras diferentes. Por questão de padronização o nome de usuário e senha serão sempre minúsculas.
-  Quando a senha for informada pela primeira vez, o sistema pede para entrar novamente para confirmação. O sistema retorna ao *prompt* de *login* e então a sequência de *login* deverá ser completada com a nova senha.

2.3 - Shell

O mediador entre o usuário e o sistema UNIX é chamado de Shell ou interpretador de comandos. Esse programa lê os comandos digitados pelo usuário e os executa utilizando os serviços e/ou outros programas do sistema operacional. Como o UNIX, o MS-DOS também utiliza o conceito de Shell, geralmente através do *command.com*



Interação usuário-Shell

2.4 - Organização de Usuários no UNIX

2.4.1 - Identificadores de Usuários (UIDs)

Todo usuário de um sistema UNIX tem uma conta (*account*). A conta é dividida em duas partes: o nome do usuário (*username / logname*) e uma senha (*password*). O nome de usuário é um identificador, que informa ao sistema quem é o usuário que está abrindo uma sessão no UNIX. A senha é um autenticador, ou seja, uma maneira de o sistema operacional verificar se o usuário é quem ele diz ser (*username*). O UNIX também possui algumas contas especiais, associadas não a usuários, mas a funções ou processos do próprio sistema. Estas contas são usadas para administração.

O UNIX faz a tradução entre o nome de usuário e o UID consultando o arquivo */etc/passwd*. Cada usuário possui uma entrada nesse arquivo, onde o sistema pode mapear nomes de usuário (*username*) em UIDs. Caso seja fornecido o mesmo UID a dois usuários, o UNIX os verá como um único usuário, mesmo possuindo diferentes *usernames* e *passwords*.

O superusuário (*root*) é uma conta especial no sistema, pois detém todos os privilégios necessários à execução das tarefas de administração. Essa conta possui o *username* igual a *root* e seu UID é 0.

2.4.2 - Grupos

Todo usuário do UNIX pertence a um ou mais grupos. Assim como usuários tem *username* e UIDs, grupos possuem nomes (*group names*) e identificadores (GIDs).

Grupos permitem ao administrador do sistema organizar seus usuários de forma a permitir acesso comum a arquivos específicos, diretórios, dispositivos, etc.

Assim como o UID, o GID (*Group Identifier*) do usuário é armazenado no arquivo */etc/passwd*. A lista de grupos do sistema, juntamente com seus GIDs está armazenada no arquivo */etc/group*.

Os conceitos de UIDs e GIDs permitem definir o perfil do usuário dentro do sistema, ou seja, definir o espaço de armazenamento do usuário, permissões de acesso aos arquivos de dados e programas, compartilhamento de informações com outros usuários, etc.

II - PARTIÇÃO

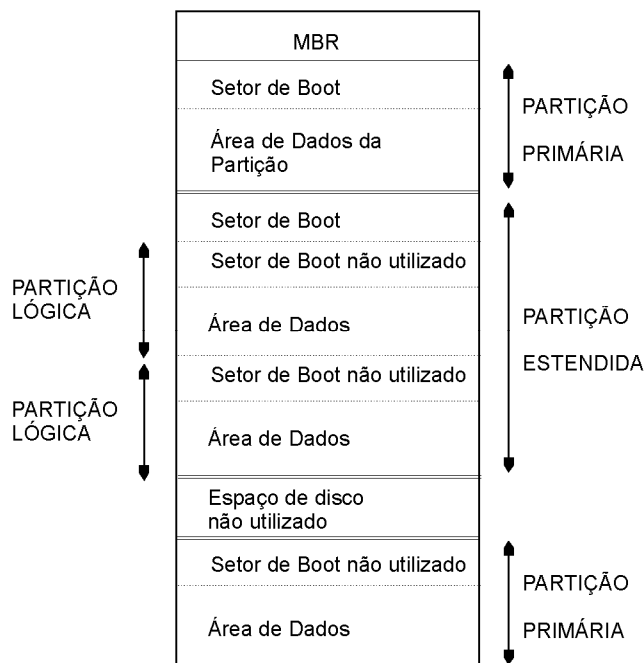
1 - DEFINIÇÃO

Discos são divididos em uma ou mais partições. Cada partição funciona como se fosse um disco separado, assim, diferentes sistemas operacionais podem ser instalados num mesmo disco sem que ocorra conflito.

A informação sobre como um disco foi particionado é armazenada no 1º setor (1º setor da 1ª trilha do 1º disco). Este setor é chamado **registro de boot principal** (MBR - *Master Boot Record*) do disco. O MBR contém um pequeno programa que lê a tabela de partição, verifica qual partição está ativa (como partição de *boot*) e lê o primeiro **setor de boot** daquela partição. O setor de boot também contém outro pequeno programa que lê a primeira parte do sistema operacional armazenado naquela partição e então o inicializa.

O esquema original de particionamento para discos de PC permitiam somente 4 partições. Para superar este problema, foram criadas as partições estendidas (extended partitions), permitindo que uma partição primária (*primary partition*) fosse dividida em sub-partições. A partição primária então subdividida é partição estendida.

Na figura abaixo o disco é dividido em 3 partições primárias, a segunda das quais é dividida em 2 partições lógicas. Parte do disco não é particionada. O disco como um todo e cada partição primária possuem um setor de *boot*.



Exemplo de Partição de Disco

Existem muitos programas para criar e remover partições. A maioria dos sistemas operacionais possuem o seus próprios programas, geralmente chamados *fdisk*. Quando estiver usando os discos IDE a partição de *boot* deve estar completamente dentro dos 1024 cilindros. Isto acontece porque o disco é utilizado pela BIOS durante o *boot*, e a maioria das versões de BIOS não consegue manipular mais do que 1024 cilindros. Cada partição deve ter um mesmo número de setores. O sistema de arquivos do UNIX utiliza blocos de 1 kbytes de tamanho, ou seja, 2 setores.

Cada partição, incluindo as partições estendidas, possuem o seu próprio arquivo de *device*. A convenção de nomes para estes arquivos que é um número de partição, é incluído após o nome do disco com a convenção: 1-4 são partições primárias e 5-8 são para as partições lógicas. Por exemplo, `/dev/hda` (*hd* - *hard drive*) refere-se ao primeiro disco como um todo, `/dev/hda1` é a primeira partição primária do primeiro disco IDE e `/dev/sdb7` é a terceira partição estendida no segundo disco SCSI. Os prefixos mais usados na definição de nomes de discos são:

- *fd* - floppy disk
- *hd* - hard drive
- *sd* - SCSI drive
- *st* - SCSI tape

III - SISTEMA DE ARQUIVOS DO UNIX

1 - DEFINIÇÃO

Um sistema de arquivos (*filesystem*) define os métodos e as estruturas de dados que um sistema operacional usa para manter o caminho de arquivos num disco ou partição, ou seja, é a maneira como os arquivos são organizados no disco.

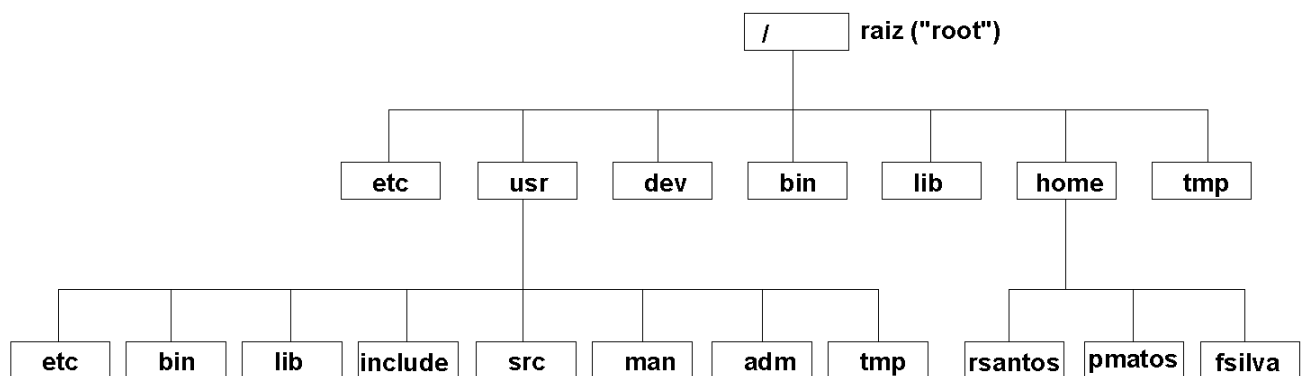
O sistema operacional UNIX usa um *filesystem* hierárquico para gerenciar e organizar seus arquivos e diretórios. Um arquivo geralmente é um recipiente para dados e diretório é um recipiente para arquivos e/ou outros diretórios. Um diretório contido dentro de outro diretório é também denominado de subdiretório.

Para trabalhar eficientemente com o sistema UNIX, é necessário compreender como o *filesystem* é organizado e os diferentes tipos de arquivos que podem ser criados. Existem limitações nos nomes de arquivos e diretórios.

O UNIX suporta vários tipos de sistemas de arquivos. Para descobrir quais são os sistemas de arquivos suportados consulte a árvore de origem do kernel no diretório `/usr/src/linux/fs`. É possível selecionar quais os tipos de sistemas de arquivos serão suportados no momento de construção do kernel (consulte o manual do sistema operacional).

2 - ESTRUTURA HIERÁRQUICA

Os arquivos são manipulados internamente pelo sistema operacional UNIX. Entretanto, para a visão do usuário, o sistema de arquivos é uma única estrutura hierárquica, onde pode-se ter várias partições, discos e arquivos associados em forma de um árvore.



Árvore de Diretório no UNIX

No sistema de arquivo do UNIX, arquivos e diretórios são agrupados por função. Usuários podem controlar seus próprios arquivos e diretórios enquanto que o administrador do sistema controla e gerencia o sistema de arquivos e diretórios. Assim fica mais fácil organizar as centenas de arquivos e programas que normalmente são instalados em uma máquina rodando o UNIX. O topo da hierarquia é denominado diretório raiz (*root*) e é indicado com uma barra (/).

O sistema UNIX também fornece comandos que permitem que você crie novos diretórios quando sua organização precisar de mudanças, como mover ou copiar arquivos de um diretório para outro, o que pode ser feito de forma extremamente fácil.

A estrutura de diretório pode variar de acordo com a implementação do sistema.

3 - CAMINHOS

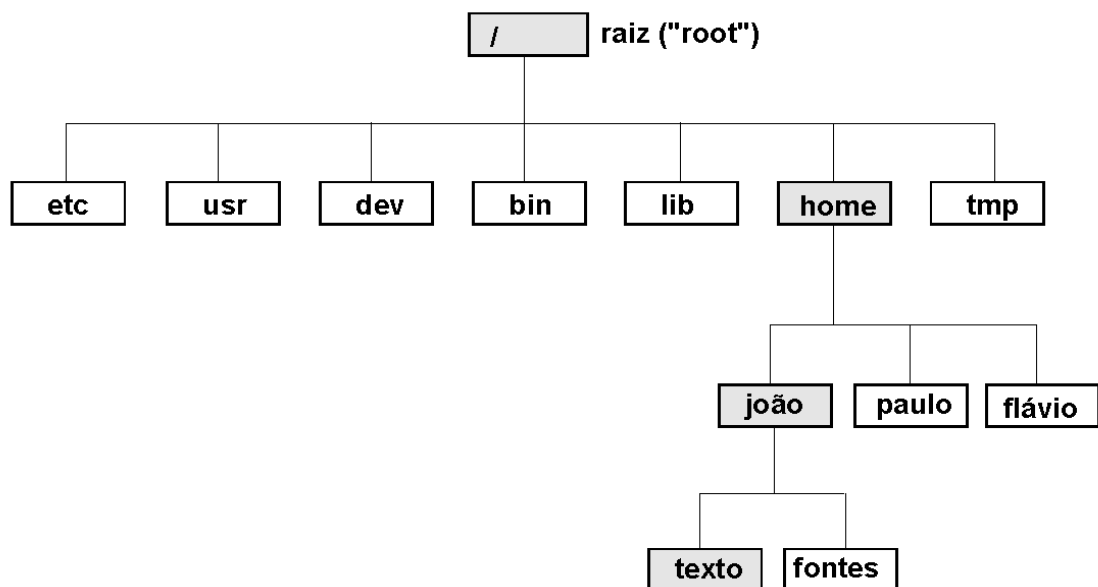
A maioria dos comandos do sistema UNIX operam sobre arquivos e/ou diretórios. Um caminho (*path*) representa os diretórios e subdiretórios que são passados para alcançar um arquivo ou diretório especificado.

O diretório inicial é chamado *diretório raiz*, representado por uma barra (/). Um *path* pode ser especificado de duas formas:

- absoluto;
- relativo.

Paths absolutos são aqueles que começam com uma barra (/). **Paths relativos**, ao contrário, sempre tomam o diretório atual como referência. Por exemplo:

- /dev absoluto
- /home/joao/texto absoluto
- .. relativo (diretório imediatamente acima)
- . relativo (diretório corrente)
- usr/lib relativo
- ../home/joao relativo



Caminho no UNIX

Quando um diretório é criado, duas entradas são automaticamente criadas, chamadas ponto (.) e ponto-ponto (..). Geralmente são utilizadas para designar caminhos relativos.

A entrada ponto (.) representa sua posição no diretório corrente. Se você está no diretório /home/joão então temos:

- . representa o diretório corrente /home/joão
- ./texto representa /home/joão/texto

A entrada ponto-ponto (..) representa o diretório imediatamente acima do seu diretório corrente, também chamado de **diretório-pai**. Se você está no diretório /home/joão então temos:

- .. representa o diretório / (raiz)
- ../.. também representa o diretório / (raiz)
- ../bin representa o diretório /bin

4 - ARQUIVOS

Um arquivo é simplesmente um nome com dados associados a ele, armazenados num periférico qualquer (disco rígido ou flexível). O sistema UNIX não impõe restrições ao formato de um arquivo. Um arquivo é visto pelo sistema como uma sequência não formatada de bytes, onde um byte segue o outro até que o fim do arquivo seja encontrado. Os programas de aplicação impõem formatos a essas sequências de bytes. Como exemplo, um editor de textos marca o fim de cada linha de texto com um caractere de nova-linha; um programa de banco de dados organiza os dados em registros, sendo que cada registro está dividido em campos.

Para o sistema UNIX tudo é considerado arquivo onde incluímos:

Arquivos Comuns - geralmente contém caracteres de texto ASCII.

Programas - arquivos comuns que contém instruções executáveis. Pode conter um código compilado (date, mkdir, ls) ou pode conter comandos shell do sistema UNIX (.profile, .logout).

Diretório - arquivo especial contendo os nomes dos arquivos e diretórios que ele mantém. O diretório guarda um número de inodo para cada item, que identifica aonde as informações de arquivo e os endereços de armazenamento de dados podem ser encontrados no sistema de arquivos.

Dispositivo - arquivo que garante o interfaceamento entre o kernel e o hardware de um periférico (disco, terminal, impressora, memória). Como estes arquivos têm por finalidade o interfaceamento, eles nunca contém dados verdadeiros. Estes arquivos são geralmente guardados no diretório /dev, e existe um arquivo para cada dispositivo com o qual o computador precisa se comunicar.

4.1 - Características

Um arquivo possui muitas características que podem ser visualizadas com o comando **ls -la**, dentre as quais destacamos:

```
$ ls -la
```

-rw-r--r--	1	ricardo	computacao	37	Ago	15	12:10	readme.txt
-rwxr-xr--	1	ricardo	computacao	127	Ago	15	12:20	.profile
drwxr-xr-x	2	ricardo	computacao	1024	Ago	15	12:34	aula

↑ ↑ ↑	↑	↑	↑	↑	↑	↑	↑
permissões	links	proprietário	grupo	tamanho	data e hora		nome

tipo de arquivo

Tipo de Arquivo -	normal ou especial
Permissão -	definição do modo de acesso ao arquivo
Links -	número de nomes de arquivos associados ao mesmo conjunto de dados
Proprietário -	identificação do usuário proprietário do arquivo
Grupo -	identificação do grupo para acesso ao arquivo
Tamanho -	número de bytes que o arquivo contém
Data -	data e hora da última alteração do arquivo
Nome -	nome do arquivo

Um ponto muito importante a saber a respeito dos nomes de arquivos no UNIX é que letras maiúsculas e minúsculas são interpretadas diferentemente. Geralmente os nomes de arquivos são formados por letras minúsculas. As regras para a criação de arquivos são:

- máximo de 14 caracteres;
- máximo de 255 caracteres se o sistema suporta nomes longos;
- normalmente contém caracteres alfa (a-z, A-Z), numéricos (0 - 9), ponto (.), traço (-) e subscrito (_).

Ponto (.) - por ser considerado um caracter normal, o ponto pode aparecer em qualquer lugar num nome de arquivo e várias vezes. Quando aparece no início do nome do arquivo (.profile) indica que este é oculto.

Existem caracteres especiais que não podem ser utilizados como parte de um nome de arquivo pois frequentemente produzirão resultados inesperados, são eles: ?, @, #, [,], *, <, >, \, \$ e |.

4.2 – Tipos de Arquivos

Os tipos de arquivos no sistema UNIX são identificados através do primeiro caracter de saída do comando ls -l. Os arquivos geralmente são tratados de duas formas:

- **Comuns** arquivos de dados ou programas;
- **Diretórios** arquivo que contém nome de outros arquivos, juntamente com ponteiros para seus *i-nodes*.
- **Especiais** um tipo de arquivo utilizado por dispositivo de caractere ou de bloco;
- **Link** um tipo de arquivo que aponta para outro arquivo.

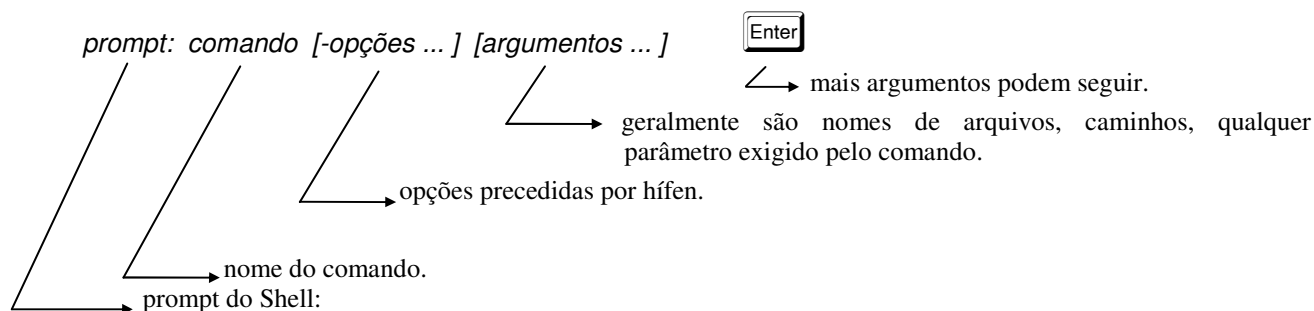
Símbolo	Tipo do Arquivo
-	Comun
d	Diretório
c	Especial – caractere
b	Especial – bloco
l	Link

Tipos de Arquivos

IV – COMANDOS: NOTAÇÃO E DOCUMENTAÇÃO

1 – NOTAÇÃO GENÉRICA DE COMANDOS

Os comandos no UNIX são muitos e variados, contudo guardam entre si algumas características comuns. Um comando, de um modo geral, é composto do nome seguido de opções e argumentos. Assim, um comando pode vir seguido de uma ou mais opções e um ou mais argumentos, nesta ordem.



Os elementos da linha de comando que estão entre [] são opcionais. Em alguns casos somente o nome do comando é obrigatório.

Espaços em branco (definidos com a tecla **Space** ou **Tab**) são usados como delimitador de comandos, opções e argumentos. Após todo comando sempre deverá ser pressionada a tecla **Return** para que o mesmo seja executado pelo computador. Para entrar com vários comandos em uma mesma linha utilize o carácter **ponto-e-vírgula** (;) como delimitador.

2 – DOCUMENTAÇÃO ONLINE

O UNIX possui uma documentação on-line bastante abrangente, conhecida como *man pages*. Cada item documentado é apresentado numa estrutura como mostrado abaixo.

```
NOME
    nome - função do comando
SINOPSE
    nome [-opções ...] [argumentos ...]
DESCRIÇÃO
    apresenta a descrição do comando
OPÇÕES
    -X : descreve a opção X
EXEMPLO
    $ nome -X argumento(s)
```

Para utilizar a referência *on-line*, o usuário deve utilizar o comando **man**. O comando **man** apresenta todas as entradas do manual UNIX *on-line*.

man tópico

O *tópico* é o comando, chamada de sistema, função, ou qualquer outro item que se queira consultar. O argumento *seção* especifica em qual seção [1-8] no manual se deseja procurar pela ocorrência do tópico.

Ao encontrar o tópico solicitado, o **man** apresenta o conteúdo na tela e aguarda comandos do teclado. Os principais comandos são:

q	finaliza
<ENTER>	próxima linha
<ESPAÇO>	próxima tela
w	volta uma tela

V - REDIRECIONAMENTO DE E/S E PIPELINES

1 - DEFINIÇÃO

Um comando do UNIX é um programa (conjunto de instruções) que geralmente lê dados de um ou vários arquivos de entrada, efetua algum tipo de processamento sobre eles (computação) e finalmente produz os resultados em arquivo(s) de saída. Na arquitetura do UNIX, note-se, dispositivos como terminal, discos e impressoras também são tratados como arquivos.

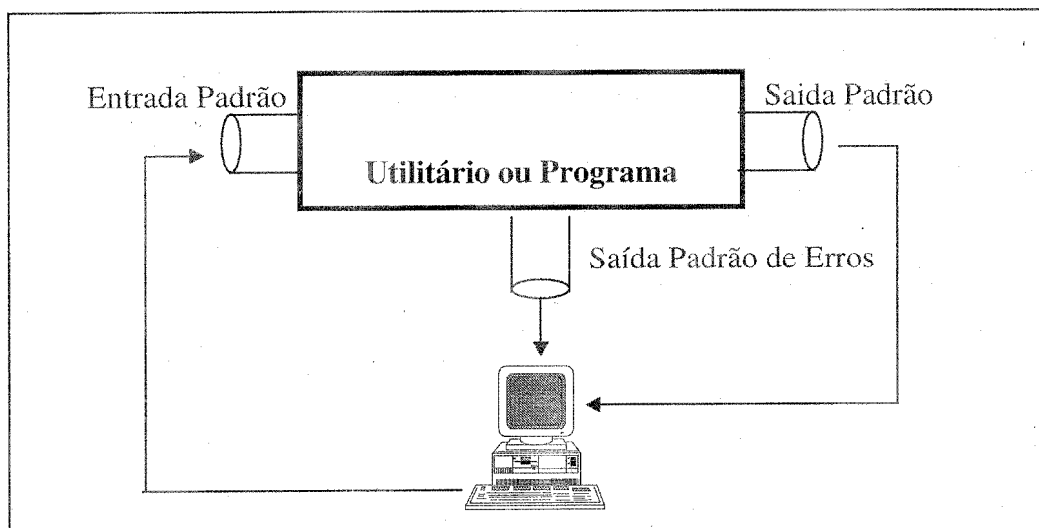
Ao criar um processo, o UNIX predefine e automaticamente lhe associa três arquivos padrões:

- `stdin` (0) - entrada padrão;
- `stdout` (1) - saída padrão;
- `stderr` (2) - saída padrão para erros.

A entrada padrão (*stdin*) é um arquivo de onde os programas obtêm dados de entrada para o seu processamento. O sistema associa a esta entrada padrão o teclado do terminal, caso o usuário não especifique uma alternativa. Essa redefinição é um recurso oferecido pelo Shell conhecido como redirecionamento.

A saída padrão (*stdout*) é o arquivo para onde usualmente são enviados os resultados do programa. O monitor do terminal é o arquivo de saída padrão associado ao processo pelo sistema. Assim como a entrada padrão, a saída padrão pode ser redirecionada para outros arquivos (impressora, disco, etc).

Além da saída e entrada padrão, o UNIX reserva uma conexão de I/O para mensagens de erros do programa. Se um usuário redireciona a saída padrão de um programa, mensagens de erros emitidas por este programa serão ainda enviadas para o terminal, pois a saída padrão de erros também é o terminal. O Shell permite redirecionar a saída padrão de erros para outros arquivos no sistema.



Entrada e Saídas Padrões

2 - COMANDOS DE REDIRECIONAMENTO DE E/S

Redirecionamento é uma facilidade que o UNIX oferece, permitindo que o usuário altere as saídas e a entrada padrão do programa, especificando novos arquivos.

2.1 - REDIRECIONANDO A SAÍDA PADRÃO (> e >>)

No exemplo do comando **cal**, a saída padrão do comando foi o terminal. O usuário pode definir que a saída do programa (um determinado calendário) seja realizada para um arquivo em disco, para posterior utilização. Esse redirecionamento é feito utilizando-se o símbolo “>” (maior que) e o nome do arquivo que armazenará a saída do programa.

```
$ cal 10 1997 > calendario97.dat
```

Nesse exemplo, a saída do comando **cal** será redirecionada para o arquivo `calendario97.dat`. O arquivo será criado, caso não exista, ou sobrescrito se já existir um arquivo com mesmo nome no diretório.

Uma outra forma de redirecionar a saída padrão é feita utilizando o símbolo “>>”. Nesse caso, a saída é colocada no final (*append*) de um arquivo existente, não destruindo os dados do arquivo. Caso o arquivo não exista, ele será criado para armazenar a saída do programa.

```
$ cal 11 1997 >> calendario97.dat
```

2.2 - REDIRECIONANDO A ENTRADA PADRÃO (<)

O redirecionamento de entrada pode ser feito em todos os comandos que lêem seus dados de uma entrada padrão. O redirecionamento de entrada é realizado utilizando o símbolo “<” (menor que), indicando ao Shell para trocar a entrada padrão (terminal) para um novo arquivo especificado.

Como exemplo, pode-se ter o comando **write** com a entrada padrão redirecionada para outro arquivo. No caso, a entrada estará contida no arquivo em disco, chamado `calendario97.dat`.

```
$ write antonio < calendario97.dat
```

Desse modo, o conteúdo do arquivo `calendario97.dat` será transmitido para a tela do terminal do usuário `antonio`.

2.3 - REDIRECIONANDO A SAÍDA DE ERROS (2> e 2>>)

A saída padrão de erros é muito utilizada para a depuração de programas e emissão de relatórios de erros aos usuários. O sistema operacional pode utilizar a saída padrão de erros, associada ao processo, para relatar a seu usuário algum erro que ocorreu na execução do processo. A saída de erros é redirecionada utilizando a notação: `2>`

```
$ ls arquivo_nao_existente > saida_erro.dat
ls: arquivo_nao_existente: No such file or directory
$ cat saida_erro.dat
$ ls arquivo_nao_existente 2> saida_erro.dat
$ cat saida_erro.dat
ls: arquivo_nao_existente: No such file or directory
```

Da mesma forma que a saída padrão, a saída padrão de erros pode ser redirecionada para o final (*append*) de um arquivo existente, não destruindo os dados do arquivo. Caso o arquivo não exista, ele será criado para armazenar a saída do programa.

```
$ ls arquivo_nao_existente 2> saida_erro.dat
$ cat saida_erro.dat
ls: arquivo_nao_existente: No such file or directory
$ ls arquivo_nao_existente2 2>> saida_erro.dat
$ cat saida_erro.dat
ls: arquivo_nao_existente: No such file or directory
ls: arquivo_nao_existente2: No such file or directory
```

3 - COMANDOS DE *PIPELINES*

Pipes são uma velha forma de comunicação entre processos no UNIX, e são fornecidos por todos os sistemas UNIX. Um *pipe* (duto) é uma conexão que conduz a saída padrão de um programa para a entrada padrão de outro programa. Esse conceito facilita aos programas UNIX trabalharem juntos, sem a necessidade de arquivos intermediários.

3.1 - *PIPELINE* (|)

O símbolo para a criação de *pipelines* é a barra vertical (|). Quando em uma linha de comando, dois programas são separados por uma barra vertical (|), a saída do primeiro é transferida para a entrada do segundo.

```
$ who | wc -l  
6
```

A resposta da linha de comando acima é o número de usuário conectados no sistema naquele instante.

Os *pipes* são *half-duplex*, isto é, os dados fluem somente em uma direção. Eles somente podem ser usados entre processos que possuam o mesmo ancestral comum (processo pai). No caso anterior, o processo “pai” comum aos dois processos era o Shell do usuário.

VI – PRINCIPAIS COMANDOS

1 - COMANDOS DE USO GERAL

O UNIX diferencia entre letras maiúsculas e minúsculas. Por convenção, todos os comandos listados aqui devem ser digitados em letras minúsculas.

As palavras em **negrito** constituem o nome do comando, devendo ser digitadas exatamente como estão escritas; em *itálico* representam os parâmetros que o usuário especifica para o comando (o nome de um arquivo, por exemplo); [entre colchetes] é opcional, podendo ser omitido (na execução do comando os colchetes não devem ser digitados. Quando as opções estiverem separadas por uma barra vertical (|), indica que somente uma delas deve ser escolhida.

1.1 - CLEAR

Sintaxe: **clear**
Descrição: o comando **clear** limpa a tela atualmente ativa
Exemplo: **\$ clear**

1.2 - CAL

Sintaxe: **cal** [mês [ano]]
Descrição: **cal** apresenta o calendário do ano e mês especificados. Na falta dos argumentos, são assumidos o mês e o ano corrente.
Exemplo: **\$ cal 6 1997**

June 1997						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
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
29	30					

1.3 - DATE

Sintaxe: **date**
Descrição: sem nenhum argumento, são apresentadas a hora e a data corrente. somente o administrador do sistema pode alterar a data.
Exemplo: **\$ date**
Tue Jun 17 17:01:02 EST 1997

1.4 - PASSWD

Sintaxe: **passwd**
Descrição: **passwd** altera a senha de um usuário. O superusuário (*root*) pode usar **passwd** para alterar a senha de qualquer usuário.
Exemplo: **\$ passwd**

2 - COMANDOS DE MANIPULAÇÃO DE DIRETÓRIOS

Um diretório é uma maneira de organizar os arquivos. Os diretórios também são considerados como sendo arquivos com formato pré-definido. Podem conter os nomes dos arquivos e diretórios e seus números inode associados. A seguir apresentamos os comandos básicos para manipulação de diretórios.

2.1 - PWD

Sintaxe: **pwd**

Descrição: pesquisa a árvore de diretório, em direção a raiz para informar o *path corrente*.

Exemplo: **\$ cd /home/antonio**

\$ pwd
/home/antonio

2.2 - LS

Sintaxe: **ls [-l R] [diretório]**

Descrição: o comando **ls** é usado para listar arquivos. Na ausência do parâmetro *arquivo* representando o nome de um arquivo ou diretório, o comando **ls** lista o diretório corrente.

Opções: **-R** lista a árvore de diretórios recursivamente;
-l listagem com formato longo

Exemplos: **\$ ls -l**
total 2
-rw-r--r-- 1 antonio users 32 Jun 17 22:40 arquivo.txt
drwx----- 3 antonio users 512 Jun 17 21:05 teste/

2.3 - CD

Sintaxe: **cd [diretório]**

Descrição: *diretório* torna-se o novo diretório corrente. Na execução do comando o sistema verifica a permissão de acesso ao diretório especificado. Sem argumento, o comando **cd** retorna ao diretório *home* do usuário.

Exemplo: **\$ cd /home/antonio**
\$ cd ..
\$ cd ../home/ana

2.4 - MKDIR

Sintaxe: **mkdir nome_diretório**

Descrição: cria um diretório na árvore de diretórios.

Exemplos: **\$ mkdir ../textos**

2.5 - RMDIR

Sintaxe: **rmdir nome_diretorio**

Descrição: remove diretórios (somente se estes estiverem vazios).

Exemplos: **\$ rmdir /usr/ana/textos**
\$ rmdir arquivo.txt

3 - COMANDOS DE MANIPULAÇÃO DE ARQUIVOS

O sistema UNIX disponibiliza vários comandos para manipulação de arquivos. Os principais são:

3.1 - CAT

Sintaxe: **cat** arquivo [arquivo ...]
Descrição: o comando **cat** concatena de lista conteúdo de arquivos.
Exemplos: \$ **cat** arq.dat
Linha 1
Linha 2

3.2 - MORE

Sintaxe: **more** [arquivo ...]
Descrição: **more** é o paginador de arquivos padrões para UNIX. Após cada tela cheia, o comando apresenta uma mensagem solicitando ao usuário para continuar a apresentação do arquivo.
Exemplo: \$ **more** arq.dat
Linha 1
Linha 2

3.3 - TAIL

Sintaxe: **tail** - D opção [arquivo]
onde: D=deslocamento
Descrição: o comando **tail** escreve o conteúdo do arquivo especificado para a saída padrão, iniciando no ponto (deslocamento) definido pelo usuário. Se o deslocamento é precedido do sinal “-” significa que o ponto de referência para o deslocamento é o final do arquivo.
Opções: -b o arquivo é interpretado como blocos de 512-bytes;
-c o arquivo é interpretado como uma sequência de bytes;
-l o arquivo é interpretado como linhas, que devem terminar com <CR>.
Exemplo: \$ **tail** -8c arq.dat
Linha 2

3.4 - CP

Sintaxe: **cp** arq-origem arq-destino
Descrição: o comando **cp** copia o conteúdo de um arquivo/diretório para outro.
Exemplos: \$ **cp** arq001 /usr/src/bkp

3.5 - MV

Sintaxe: **mv** arq-origem arq-destino
Descrição: o comando **mv** move arquivos e diretórios pelo sistema de arquivos. Permite também renomear um arquivo ou diretório, caso o nome destino seja igual da origem.
Exemplos: \$ **mv** /home/antonio /home/bkp
\$ **mv** teste.txt teste1.txt

3.6 - RM

Sintaxe:	rm [-r] <i>arquivo</i>
Descrição:	o comando rm é usado para remover arquivos e diretórios (inclusive aqueles não vazios). Para apagar um arquivo ou diretório, o usuário não precisa ter direitos de leitura ou escrita sobre ele, mas precisa ter direito de escrita no diretório onde ele se encontra.
Opções:	-r é a opção para apagar um diretório, removendo todos arquivos e sub-diretórios;
Exemplos:	\$ rm teste.txt \$ rm -r teste

3.7 - FIND

Sintaxe:	find [caminho ...] [expressão]
Descrição:	o comando find procura arquivos recursivamente em cada diretório especificado em <i>caminho</i> , confrontando-os com a expressão booleana <i>expressão</i> .
Expressões:	-name <i>arquivo</i> verdadeiro se <i>arquivo</i> confronta com o nome do arquivo corrente; -print sempre verdadeiro, causa a impressão do <i>path</i> corrente.
Exemplos:	\$ find -name arq -print ./arq

VII - OUTROS COMANDOS

1 - COMANDOS DE FILTROS

Filtro é um programa que lê de sua entrada padrão, executa um processamento e escreve para sua saída padrão. Filtros são normalmente conectados linearmente em Shell *pipelines*.

1.1 - WC

Sinopse: **wc** [-lwc] [arquivo ...]

Descrição: o comando **wc** conta o número de linhas, palavras e caracteres de um arquivo especificado como parâmetro. Se nenhum nome de arquivo for especificado, o arquivo de entrada do comando **wc** será a entrada padrão. Caso nenhuma opção seja definida, o comando imprime o número de linhas, palavras e caracteres.

Opções:

- -l conta linhas;
- -w conta palavras (*words*);
- -c conta caracteres.

Exemplos:

```
$ wc arq.dat
2      4      16
$ wc -c arq.dat
16
```

1.2 - GREP

Sinopse: **grep** *expressão* [arquivos de entrada]

Descrição: o comando **grep** procura por um padrão especificado em arquivos, escrevendo a linha do arquivo que possui o padrão equivalente.

Expressão: expressões podem ser cadeias de caracteres ou expressões regulares, que, embora também sejam cadeias de caracteres.

Exemplos:

```
$ cat > arquivo.dat
Ana
Antonio
Maria
Marco
^Z

$ grep "an" arquivo.dat
Ana
Antonio
```

VIII - PERMISSÕES E ACESSO A ARQUIVOS

1 - DEFINIÇÃO

O UNIX permite que múltiplos usuários armazenem e tenham acesso às informações do disco ao mesmo tempo. Desta forma, é fundamental proteger seus arquivos de outros usuários. Para isto faz-se uso de mecanismos para acesso de arquivos e diretórios.

Para acessar arquivos e executar programas, o sistema UNIX tem que saber suas identificações. Para cada arquivo ou diretório, tem-se três categorias de comunidade:

Nome	Descrição
usuário (<u>u</u> ser/owner)	Usuário que criou o arquivo
grupo (<u>g</u> roup)	Membros de grupos. Grupos são usuários que concordam em partilhar certos arquivos e diretórios e são geralmente formados no decorrer de um projeto ou por seções/departamentos de uma empresa.
outros (<u>o</u> thers)	Outros usuários do sistema (o usuário e os membros do grupo estão fora desse conjunto de usuários).

Propriedade de Arquivos

O computador mantém identificadores numéricos para usuário (UID) e grupo (GID). Esta identificação é definida inicialmente quando você entra no sistema. Para consultar os identificadores ativos, utilize o comando **id** diretamente no prompt do UNIX.

Todas as identificações de usuários reconhecidas pelo computador são armazenadas no arquivo `/etc/passwd`, e todas as identificações de grupo são armazenadas no arquivo `/etc/group`.

Usuários a serem incluídos num grupo específico são definidos pelo administrador do sistema, e cada usuário pode ser um membro de um ou mais grupos. Grupos são normalmente formados a partir dos grupos de trabalho já definidos numa organização.

2 - TIPOS DE ACESSOS

O sistema de arquivos do UNIX permite que se defina para cada arquivo ou diretório, três tipos básicos de controle de acesso.

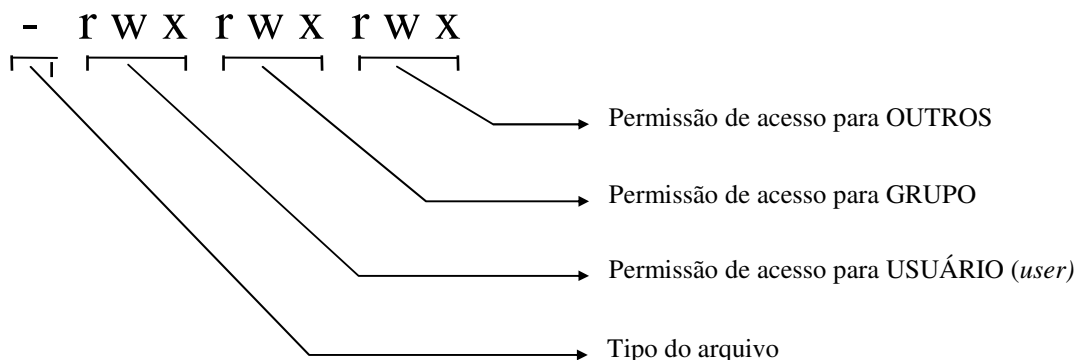
Tipo de Acesso	Aplicado em Arquivos	Aplicado em Diretórios
r	permite ler o conteúdo do arquivo	permite listar o conteúdo do diretório
w	permite alterar o conteúdo do arquivo	permite alterar o conteúdo do diretório
x	permite executar um arquivo (programa)	permite pesquisar o diretório

Tipos de Acesso para Arquivos/Diretórios

Diferentes comandos do sistema UNIX requerem algumas permissões para acessar um programa ou arquivo. Por exemplo, para executar o comando **cat** num arquivo a permissão *read* é necessária, porque este comando deve ser capaz de ler o conteúdo do arquivo para mostrá-lo na tela. Da mesma forma um diretório requer a permissão *read* para listar seu conteúdo com o comando **ls**.

A utilização de tipos de acesso para cada arquivo e diretório depende da política adotada no gerenciamento desses recursos. Permissões de acesso de escrita para diretórios liberam seu conteúdo para alteração, permitindo a remoção de seus arquivos e sub-diretórios.

A cada classe de usuários, são aplicadas as três permissões de acesso:



A combinação entre os três tipos de permissão (r,w,x) com cada classe de usuários é denominado modo do arquivo. A configuração do modo, de arquivo e diretórios é realizada pelo usuário através do comando **chmod**.

Os valores de permissão de arquivos podem ser expressos em vários formatos. Uma outra forma é a representação como números octais.

Modo	Valor Octal
---	0
--x	1
-w-	2
-wx	3
r--	4
r-x	5
rw-	6
rwX	7

3 - PERMISSÕES

O acesso a um arquivo é definido por sua identificação de usuário, sua identificação de grupo e as permissões associadas ao arquivo. As permissões para um arquivo estão especificadas no modo de acesso. O modo de um arquivo é um campo de 9 caracteres que define as permissões para o proprietário do arquivo, para o grupo ao qual pertence e para todos os outros usuários do sistema. As permissões podem ser visualizadas no comando **ls -l**.

-r w -	r - -	r - -	1	ricardo	computacao	37	Ago	15	12:10	readme.txt
-r w x	r - x	r - -	1	ricardo	computacao	127	Ago	15	12:20	.profile
d r w x	r - x	r - -	2	ricardo	computacao	1024	Ago	15	12:34	aula

↓	↓	↓		↑	↑					
Usuário	Grupo	Outros		proprietário	grupo					

4 - COMANDOS DE MANIPULAÇÃO DE PERMISSÃO DE ACESSO

Apresentamos a seguir os principais comandos para a manipulação de permissão de acesso a arquivos.

4.1 - CHMOD

Sintaxe: **chmod** nova-permissão *arquivo*

Descrição: Altera a permissão de acesso de um arquivo.

nova-permissão: 3 posições numéricas indicando o dono, o grupo e os outros respectivamente

Exemplos:

```
$ ls -l
total 2
-rw-r--r--    1 antonio  users   32   Jun 17 22:40 arquivo.txt
drwx-----   3 antonio  users  512   Jun 17 21:05 teste/

$ chmod 000 *
$ ls -l
total 2
-----      1 antonio  users   32   Jun 17 22:40 arquivo.txt
d-----      3 antonio  users  512   Jun 17 21:05 teste/

$ chmod 421 teste
$ ls -l
total 2
-r---w---x    1 antonio  users   32   Jun 17 22:40 arquivo.txt
dr---w---x    3 antonio  users  512   Jun 17 21:05 teste/

$ chmod 221 teste
$ ls -l
total 2
-r---w---x    1 antonio  users   32   Jun 17 22:40 arquivo.txt
d-w---w---x   3 antonio  users  512   Jun 17 21:05 teste/
```

4.2 - CHOWN

Sintaxe: **chown** novo_dono *arquivo*

Descrição: o comando **chown** muda o dono de um arquivo para o *novo_dono* especificado. O *novo_dono* pode ser um UID válido ou um nome de *login* válido. Somente o superusuário (*root*) pode usar o comando **chown**

Exemplos: **\$ chown antonio /home**

4.3 - CHGRP

Sintaxe: **chgrp** novo_grupo *arquivo*

Descrição: o comando **chgrp** muda o grupo associado a um determinado arquivo.

Exemplos: **\$ chown users arquivo.txt**