# Unix/Linux

**Contempla: visão geral Unix e Linux**

**Versão 1.8**
**Agosto de 2014**

Prof. Jairo

jairo@uninove.br
professor@jairo.pro.br

http://www.jairo.pro.br/

# Unix/Linux

O conteúdo apresentado em "Unix/Linux" tem como finalidade servir de guia didático e visa fornecer conhecimentos básicos em sistemas operacionais. Sua origem vem das notas de aula do Prof. Jairo, portanto é um conteúdo acadêmico com intenção de auxiliar no ensino da disciplina "Prática em Sistemas Operacionais" ministrado nos cursos de Ciência da Computação e Sistemas de Informação.



O arquivo "**jairo-unix.pdf**" está em:

http://www.jairo.pro.br/praticas_em_sist_oper/jairo-unix.pdf

Qualquer crítica ou sugestão, favor entrar em contato com o Prof. Jairo no endereço eletrônico "jairo@uninove.br" ou "professor@jairo.pro.br".

São Paulo, 10 de agosto de 2014.

# Unix/Linux

## Sumário


.....4
.....4
1 - Introdução.....5
1.1 – Histórico.....5
1.2 – Família Unix.....6
1.3 – Linux.....8
1.4 - Distribuição Linux.....8
1.5 - Arquivos em Unix.....10
1.6 - Características do Unix.....10
2 - Processo de boot e runlevel.....12
3 - Processo de logon.....14
4 - Linha de comando.....16
4.1 - Interpretador de comandos: shell.....16
4.2 - Conceitos básicos de comandos.....16
4.3 - Conceitos de processos.....17
5 - Comandos básicos.....18
6 - Pipes.....27
7 - Redirecionamentos.....28
7.1 – Saída padrão.....28
7.2 – Saída de erro.....28
7.3 – Entrada padrão.....28
8 - Permissões do sistema de arquivo.....30
8.1 – chmod.....31
8.1.1 - Modo numérico ou absoluto (octal).....31
8.1.2 - Modo Simbólico.....32
8.1.3 - Bits s e t.....33
8.1.3.1 – Bit s.....33
8.1.3.2 – Bit t.....36
8.2 - chown.....37
8.3 - chgrp.....37
9 - Umask.....38
10 - Scripts.....42
10.1 – Editor vi/vim.....42
10.2 – Scripts.....44
11 – Comandos de rede.....51
11.1 – lspci.....51
11.2 – ifconfig.....51
11.3 – ping.....52
11.4 – route.....53
11.5 – nslookup.....53
11.6 – netstat.....54
12 – Aplicações clientes de serviços.....56

12.1 – wget....56
12.2 – telnet....57
12.3 – ftp....57
12.4 – ssh....58
12.5 – sftp....58
12.6 – scp....59
13 – Serviços em Unix....61
13.1 – Processo daemon....61
13.2 – Serviço Cron....61
13.3 – Serviço Samba....64
13.3.1 – Samba server....65
13.3.2 – Samba client....68
14 – Gerenciamento de pacotes....71
14.1 – Gerenciamento de pacotes no Linux CentOS....71
14.1.1 - rpm....72
14.1.2 - yum....73
14.2 – Gerenciamento de pacotes no Linux Ubuntu....74
14.3 – Gerenciamento de pacotes em sistemas em geral....76
15 - Anexos....78
15.1 - EUID/EGID....78

# 1 - Introdução

## 1.1 – Histórico

O Unix teve origem no projeto MULTICS[1], que por ser complexo demais nunca saiu do ambiente acadêmico. O termo UNICS vem, inicialmente, pela simplificação do MULTICS num UNICS [de Uniplexed e não Multiplexed]. Como a palavra Unics é uma paródia [e também uma brincadeira], logo o nome evoluiu para Unix.

No início, por volta de 1969, Unix era um sistema operacional desenvolvido por um grupo de funcionários da AT&T[2] no Bell Labs[3]. Entre esses funcionários estavam Ken Thompson, Dennis Ritchie e Douglas Mcllroy.

A primeira versão do Unix veio em 1971, e rodava em máquinas DEC[4] e Mainframes. A popularidade veio com a versão V6, de 1975, a primeira disponibilizada fora dos domínios da Bell Laboratories. Nessa época, a Universidade de Berkeley havia comprado o código fonte do Unix.

Em 1973 o Unix foi reescrito em C, linguagem essa criada especificamente para reescrever o código fonte do Unix.

Em 1979 foi portado para máquinas VAX da DEC.

Atualmente, o termo Unix é uma marca registrada do *The Open Group*[5], e os sistemas Unix se dividem em vários ramos de sistemas criados e suportados por diversos vendedores.

Inicialmente o Unix era um sistema aberto, apropriado ao ambiente acadêmico, daí a sua popularidade. Posteriormente diversas empresas surgiram, cada uma suportando o seu Unix. E para garantir a não fragmentação da arquitetura Unix em diversos sistemas operacionais incompatíveis entre si, foi criado o padrão ou comitê POSIX.

O POSIX, Portable Operating System Interface for Unix, de 1988, é um conjunto de definições e convenções que padroniza a interface [linha de comando], o conjunto de bibliotecas dos sistemas Unix e as suas chamadas de sistema [system calls].

A partir do POSIX, é mais correto classificar os sistemas Unix como membros de uma família, isto é, família Unix.

1 MULTICS: Multiplexed Information and Computing Service foi um computador bastante avançado para a sua época. O projeto teve início em 1964.
2 AT&T: American Telephone & Telegraph.
3 Bell Labs: Também conhecido como AT&T Bell Laboratories e Bell Telephone Laboratories.
4 DEC: Digital Equipment Corporation, posteriormente adquirida pela Compaq, que foi comprada pela HP.
5 The Open Group: É um consórcio da indústria de software para prover padrões abertos e neutros para a infraestrutura de informática.

Posteriormente o padrão POSIX foi extendido pelo *The Open Group*, pela publicação da *Single Unix Specification*, que é uma família de padrões para sistemas operacionais qualificados para o nome Unix. Os sistemas não qualificados em acordo com a *Single Unix Specification* são chamados de Unix-like. Por exemplo, os sistemas AIX, HP-UX e Mac OS X são sistemas Unix registrados, já Linux e FreeBSD são Unix-like.

Em 1992 os sistemas Unix foram adaptados para a arquitetura RISC[6].

# 1.2 – Família Unix

Após o POSIX, tecnicamente o Unix passa a ser tratado como uma família de sistemas operacionais. Essa família é composta tanto por membros proprietários quanto membros de código fonte aberto.

Os principais membros dessa grande família são:

| Sistema Operacional | Fabricante | Arquitetura do computador |
|---|---|---|
| Solaris [SunOS] | Sun[7] Microsystems [proprietário] | RISC - processador Sparc |
| AIX | IBM [proprietário] | RISC - processador PowerPC |
| HP-UX | HP [proprietário] | RISC - processador PA-RISC |
| Linux | Código fonte aberto [open source] | todas |
| FreeBSD | Código fonte aberto [open source] | todas |
| Mac OS X | Apple [proprietário]. Porém, o kernel XNU é open source | Atual: x86[8]. Anterior: RISC PowerPC |

Da tabela acima, a arquitetura do computador vem da época da adaptação do Unix para a arquitetura RISC, porém atualmente temos também Solaris rodando em CISC [x86] e HP-UX em EPIC [Itanium].

Linux foi portado para as seguintes arquiteturas de computador: Intel x86 [CISC], Alpha [DEC], Sparc, Motorola 68000, Power, MIPS [MIPS Technologies], PA-RISC, EPIC, AMD X86-64 e ARM.

Apesar da definição de Unix como família de sistemas, no ambiente corporativo costuma-se ainda usar o termo "Unix" para os membros proprietários e Linux/FreeBSD para os de código fonte aberto.

6 RISC: Reduced Instruction Set Computer.
7 Em 2010 a Sun Microsystems foi comprada pela Oracle.
8 x86: é a linha de processadores da Intel, que é de arquitetura CISC.

Mais recentemente, novos sistemas operacionais surgiram baseados nos sistemas Unix-like de código fonte aberto. É o caso do iOS e Android.

| Sist. Operacional | Fabricante | Arquitetura do computador |
|---|---|---|
| iOS [iPhone OS] | Apple [proprietário]. Porém, o kernel XNU[9] é open source | ARM[10] |
| Android | Open Handset Alliance[11] [open source] | ARM, x86 |

O iOS é baseado no Mac OS X. O kernel [núcleo do sistema operacional] do Mac OS X é XNU, que é um híbrido entre kernel FreeBSD e kernel Mach. Darwin também é considerado kernel, porém trata-se do kernel XNU acrescido de outras porções do sistema.

Tanto Free BSD quanto Mach são Unix-like, já o Mac OS X é um sistema Unix registrado. Por sua vez, o iOS é Unix-like.

O Android usa o kernel Linux, com algumas modificações. No entanto, Android não é Unix-like, mas pode-se dizer que é baseado no Linux.

Tanto iOS quanto Android são sistemas desenvolvidos para equipamentos portáteis como celulares, tablets e ultrabooks[12]. Em julho de 2014, o Android já vendeu um bilhão de smartphones e tem uma fatia de 80% no mundo. Atualmente, já são vendidos mais smartphones do que celulares comuns, e esse percentual está aumentando rapidamente.

O crescimento do segmento portátil tem sido muito rápido nos últimos anos. Entre outras funções, os portáteis podem ser usados para navegar na internet. No início de 2012, os portáteis representavam 9% do tráfego na web global [9% de browser market share], no final de 2013 representavam 26%. Em 2014, nos Estados Unidos, já representam mais de 50% do tráfego na web. A persistir esse crescimento, dentro de poucos anos irá desaparecer a figura do desktop usado quase que exclusivamente para acessar a internet.

Voltando à família Unix e considerando os novos sistemas que estão surgindo, é fácil concluir que continua atual, embora tenha mais de 40 anos de idade.

9 XNU: é um acrônimo e significa X is Not Unix.
10 ARM: significa Advanced RISC Machine e pertence à empresa ARM Holdings. Difere do RISC tradicional por ter apenas 32 bits e consumir pouca energia. É uma arquitetura específica para equipamentos portáteis. 98% dos celulares do mundo usam processador ARM.
11 Open Handset Alliance: Google, HTC, Dell, Intel, Motorola, Qualcomm, Texas Instruments, LG, Samsung, T-Mobile e Nvidia.
12 Ultrabook é um conceito de notebook/netbook mais fino, leve, de baixo consumo de energia e armazenamento de dados em memória flash.

# 1.3 – Linux

Apesar de tudo que divulgam na mídia, Linux é apenas um kernel[13] de sistema operacional, mantido em www.kernel.org.

O nome Linux vem em homenagem a Linus Torvalds, o estudante de Ciência da Computação da Universidade de Helsinki na Finlândia que criou esse sistema em 1991. Na verdade, Linus partiu do Minix[14] e tentou "melhorá-lo". A idéia era ter um Unix-like completo no PC, uma máquina bastante acessível já nessa época. O mesmo não podia ser dito dos RISC onde rodavam os Unix.

No entanto, Linus teve problemas para conseguir finalizar o seu intento, então disponibilizou o projeto na internet e convidou algumas pessoas a ajudá-lo. Esse fato trouxe um grande número de colaboradores e soluções, além de uma massa de usuários ao redor do globo.

Esses fatores reunidos permitiram rapidamente tornar o Linux um sistema enxuto, rápido, moderno e eficiente. Posteriormente obteve relativa popularização e uso.

O grande mérito de Linus então foi ter inaugurado o modelo de desenvolvimento colaborativo, posteriormente copiado por vários outros desenvolvedores.

A licença de uso do Linux é GNU General Public License.

# 1.4 - Distribuição Linux

Sendo Linux apenas um kernel, não tem como um leigo baixar o código fonte, compilar, instalar e usar na sua máquina. Desse modo, surgiram empresas e projetos comunitários com a intenção de facilitar essa tarefa de instalar e usar Linux.

Numa definição, distribuição Linux é o conjunto formado pelo kernel Linux, software aplicativo, utilitários mais um instalador. Esse conjunto é criado e mantido por organizações comerciais ou mesmo projetos comunitários. Em geral, é usado apenas software na categoria open source, mas em alguns casos também entra software proprietário.

De um modo geral, as distribuições comerciais tem por intenção vender suporte ao sistema. Ou seja, partem de um produto de custo zero [open source] e o que puderem vender de suporte será

13 Kernel: núcleo do sistema operacional.
14 Minix: é o sistema operacional Unix-like criado pelo professor Tanenbaum em 1987, com propósitos educacionais. O Minix roda em PCs, ao contrário dos Unix proprietários que rodam na arquitetura RISC.

lucro. Mas essas empresas também desenvolvem software, em acordo com as suas necessidades [por exemplo, suporte a harware] e contribuem com www.kernel.org. Também contribuem com doações para kernel.org continuar mantendo as suas atualizações no kernel.

O desenvolvimento do Linux atualmente continua sendo colaborativo [como no início em 1991], porém agora o desenvolvedor é basicamente um profissional pago por alguma empresa. Isso mostra que a filosofia open source não é incompatível com o mundo dos negócios.

As principais distribuições são:

- Red Hat [www.redhat.com], é voltada para o segmento servidor. A versão para desktop é o Fedora.
- Suse [www.novell.com/linux], é voltada para o segmento servidor. Para entrar nesse segmento, a Novell comprou a distribuição Suse.
- Mandriva [www.mandriva.com] é voltada para o segmento desktop. Essa distribuição surgiu da fusão da Mandrake [França] com a Conectiva [Brasil].
- Debian [www.debian.org] voltada para o segmento desktop. Tem a tradição de não incluir software proprietário.
- Slackware [www.slackware.com] voltada para o segmento desktop. Tem a tradição de usar muito a linha de comando.
- Ubuntu [www.ubuntu.com] voltada para o segmento desktop.

Existem centenas de distribuições Linux. A maior parte delas é baseada em alguma citada acima.

Para manter todas as distribuições Linux compatíveis entre si foi criado o **LSB** [Linux Standard Base]. LSB padroniza a estrutura interna da distribuição. De certo modo, LSB é uma extensão do POSIX que é aplicada especificamente ao Linux.

O objetivo da LSB é promover um conjunto de padrões que aumentarão a compatibilidade entre as distribuições Linux, permitindo que uma mesma aplicação para Linux seja instalada e rode em qualquer distribuição.

A LSB especifica bibliotecas padrão, comandos, utilitários, hierarquia do sistema de arquivo e níveis de execução, entre outros.

NOTA:
Não são apenas as distribuições Linux que contribuem com www.kernel.org. Por exemplo a Google, no processo de porte [adaptação] do Linux para um celular, precisou desenvolver soluções, que posteriormente foram incorporadas ao código do kernel Linux. É essa lógica de economia no desenvolvimento que efetivamente garante um rápido crescimento em uso dos sistemas de código fonte aberto.

# 1.5 - Arquivos em Unix

Em Unix, os arquivos são classificados como regulares, diretórios e especiais.

Os arquivos que guardam conteúdo são os **regulares**, e podem ser binários ou texto. Por exemplo, o arquivo */etc/passwd* é um arquivo de texto, já o arquivo */bin/ls* é binário.

Os arquivos que tratam da organização dos dados no sistema de arquivo são os **diretórios**. Por exemplo, */etc*.

Já os arquivos do tipo **especial** são os que associam dispositivos de hardware ao sistema hierárquico de arquivo. No Unix não existe, por exemplo, unidade de disco, portanto o dispositivo físico está associado a um arquivo do tipo especial. No diretório */dev* é onde estão os arquivos especiais, por exemplo */dev/sda1*. Desse modo, no Unix tudo passa a ser tratado como arquivo, por exemplo, acessar a um dispositivo é acessar a um arquivo do tipo especial.

# 1.6 - Características do Unix

As características do Unix são: multiusuário, multitarefa, portável e sistema de arquivo hierárquivo e montável.

Multiusuário é a capacidade de permitir a mais de um usuário acessar ao sistema simultaneamente.

Multitarefa é a capacidade de executar mais de uma tarefa simultaneamente.

Portável pois foi escrito na lingugam C, linguagem essa criada para (re)escrever o código do Unix. Desse modo, para portar o Unix para outra arquitetura de computador, o pré-requisito básico é que nessa máquina tenha um compilador C. Numa visão simplificadora, basta compilar o código do sistema Unix nessa máquina que ele terá sido portado.

Sistema de arquivo hierárquico e montável diz respeito à estrutura hierárquica do sistema de arquivo, com seus diretórios e sub-diretórios associados aos dispositivos [devices] com sistema de arquivo, os quais são montados [anexados] a determinados pontos no sistema hierárquico.

A figura abaixo mostra o sistema hierárquico de arquivo associado [montado] em dois pontos de montagem: dispositivo /dev/sda1 no / [barra] e /dev/sda2 no /home.

Os principais diretórios básicos são: /etc/, /bin, /usr, /lib, /tmp, /sbin, /home e /dev. De um modo geral, esses diretórios básicos [além de outros] aparecem em todos os sistemas da família

Unix.

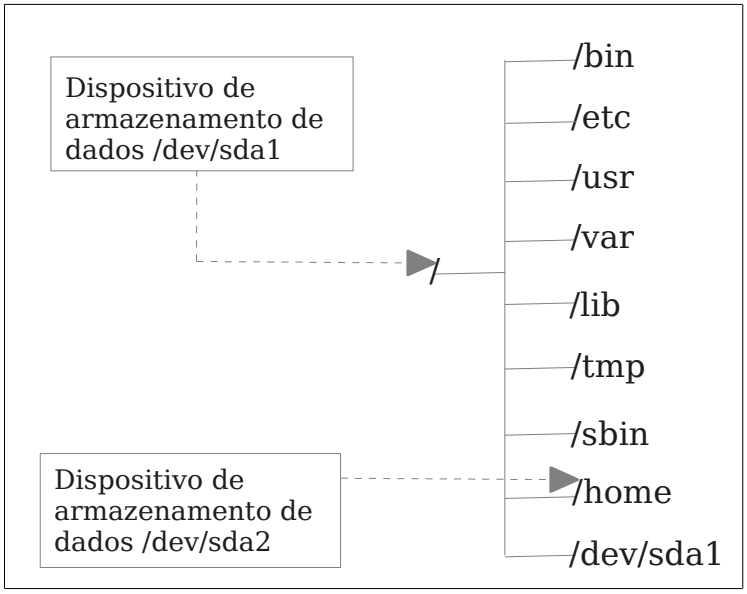
No **/etc** é onde estão as configurações do sistema, na forma de centenas de arquivos de texto.

No **/bin** é onde estão os executáveis básicos do sistema. Esses executáveis são os comandos Unix.

No **/usr** é onde estão os recursos, o nome vem de *unix system resources*.

No **/var** é onde são guardados dados que variam à medida que o sistema roda. Nesse diretório estão, entre outros, os arquivos de log do sistema.

No **/lib** é onde estão as bibliotecas do sistema.




O **/tmp** é um diretório público, onde todo usuário tem permissão de escrita.

No **/sbin** estão executáveis do sistema, que normalmente não são executados pelos usuários e sim pelo root [administrador do sistema].

No **/home** estão os diretórios homes dos usuários, por exemplo */home/aluno*.

No **/dev** é onde estão os arquivos do tipo especiais. Os arquivos especiais relacionam os dispositivos [hardware] ao sistema hierárquico de arquivos.

# 2 - Processo de boot e runlevel

Quando a máquina é ligada, o sistema BIOS[15] carrega e executa o código do boot loader no MBR, que é o Master Boot Record ou Registro Mestre de Boot. Boot loader é quem inicia o processo de carregamento do kernel de sistema operacional.

É o MBR que decide qual sistema operacional será carregado, pois ele aponta para alguma partição com setor de boot que na sequência carrega o sistema operacional. No Linux, exemplos de boot loaders são LILO e GRUB. LILO significa LInux LOader e GRUB é GRand Unified Bootloader.

Depois de carregado o kernel, o sistema executa o programa /sbin/init que é o primeiro processo a ser disparado, portanto tem PID[16] igual a 1. O processo initi faz algumas verificações básicas como integridade do sistema de arquivos e inicia alguns programas necessários para que o sistema operacional funcione corretamente.

Ao final do carregamento do kernel, ele inspeciona o arquivo /etc/inittab para determinar o modo de operação ou runlevel.

O arquivo /etc/inittab normalmente tem as seguintes configurações:

```
=== /etc/inittab ========================================
# Default runlevel. The runlevels used by RHS are:
# 0 - halt (Do NOT set initdefault to this)
# 1 - Single user mode
# 2 - Multiuser, without NFS (The same as 3, if you do not have networking)
# 3 - Full multiuser mode
# 4 - unused
# 5 - X11
# 6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:                # Console Text Mode
# id:5:initdefault:              # Console GUI Mode
=========================================================
```

Com base no /etc/inittab acima, esse sistema irá iniciar no runlevel 3, que é o modo texto sem interface gráfica.

15 BIOS: Basic Input/Output System [Sistema Básico de Entrada/Saída] é um programa de computador pré-gravado em memória permanente [firmware] executado por um computador quando ligado.

16 PID: Process Identifier ou identificador do processo, que é um número inteiro atribuído a cada processo no sistema.

Baseado na seleção de runlevel, o processo init passa a executar os scripts[17] de inicialização localizados nos subdiretórios do diretório /etc/rc.d/. Scripts de inicialização são scripts que iniciam processos ou serviços.

No diretório /etc/rc.d existem os subdiretórios /etc/rc.d/rc0.d/, /etc/rc.d/rc1.d/ e assim por diante, até /etc/rc.d/rc6.d/. Cada um desses subdiretórios corresponde a um runlevel.

A descrição de cada runlevel segue abaixo:

| Modo | Diretório do runlevel | Descrição |
|---|---|---|
| 0 | /etc/rc.d/rc0.d | halt, desliga o sistema |
| 1 | /etc/rc.d/rc1.d | single user mode, ou modo de manutenção [sem usuários nem serviços] |
| 2 | /etc/rc.d/rc2.d | não é usado [mas poderia ser definido algum uso particular pelo usuário] |
| 3 | /etc/rc.d/rc3.d | full multi-user mode, ou modo completo de serviço sem interface gráfica |
| 4 | /etc/rc.d/rc4.d | não é usado [mas poderia ser definido algum uso particular pelo usuário] |
| 5 | /etc/rc.d/rc5.d | full multi-user mode, ou modo completo de serviço com interface gráfica |
| 6 | /etc/rc.d/rc6.d | reboot, reinicia o sistema |

Em cada um dos diretórios do runlevel existem linque simbólicos que apontam para os respectivos scripts de inicialização. A notação adotada é usar o "S" para iniciar o serviço no runlevel específico e um "K" para parar o serviço quando desligar ou reiniciar o sistema.

Por exemplo, o diretório /etc/rc.d/rc3.d/:

```
================================
...
K90network          ->      ../init.d/network
K92ip6tables        ->      ../init.d/ip6tables
K92iptables         ->      ../init.d/iptables
K95firstboot        ->      ../init.d/firstboot
S06cpuspeed         ->      ../init.d/cpuspeed
S11portreserve      ->      ../init.d/portreserve
S12rsyslog          ->      ../init.d/rsyslog
S13irqbalance       ->      ../init.d/irqbalance
S14nfslock          ->      ../init.d/nfslock
...
================================
```

17 script: um script é um roteiro, que contém comandos ou mesmo instruções em linguagem de programação específica do interpretador do script em específico. Um script é um arquivo de texto. Exemplo: shell script, que contém comandos shell ou intruções na sintaxe de programação shell.

# 3 - Processo de logon

Para logar numa máquina Unix, o usuário precisa estar previamente cadastrado no sistema. Se for um usuário local, terá uma entrada no arquivo **/etc/passwd**, que é o catálogo ou cadastro de usuários locais. Cada usuário local é uma linha no arquivo, por exemplo o usuário root tem a seguinte linha:

---
root:x:0:0:root:/root:/bin/bash
---
onde:
o 1° campo é o ***username***: root;
o 2° campo está vazio [x], pois a *hash*[18] de senha [digest] está no arquivo /etc/shadow [Linux];
o 3° campo é o ***UID*** [User Identifier]: 0;
o 4° campo é o ***GID*** [Group Identifier]: 0;
o 5° campo é a ***descrição*** do usuário: root;
o 6° campo é a ***home*** do usuário: /root.
o 7° campo é o ***shell*** do usuário: /bin/bash

Para o usuário aluno, tem a seguinte linha:

---
aluno:x:1000:1000:uninove:/home/aluno:/bin/bash
---
onde:
o **username** é aluno, **UID** é 1000, **GID** é 1000, **descrição** é "uninove", a **home** é /home/aluno e o **shell** é /bin/bash.

Os **GID** estão cadastrados no arquivo **/etc/group**, que é o catálogo ou cadastro de grupos locais.

Para o grupo de GID 1000 tem a seguinte entrada:

---
aluno:x:1000:
---
onde:
o 1° campo é o ***groupname***: aluno;
o 2° campo é a hash de senha do grupo;
o 3° campo é o **GID;**

18 Hash: é o mesmo que *message digest,* que é uma função que produz um código de tamanho fixo como saída para uma dada mensagem de entrada [a senha].

o 4° campo é para incluir os usernames que participam como grupo secundário.

O usuário também precisa ter uma entrada no arquivo de senhas, que no Linux é o **/etc/shadow**. Por exemplo:

---
aluno:$1$GXJzit5J$vS4wC8AW6hV8zvLu6Dtxc.:14713::::::
---
onde:
o 1° campo é o ***username***: aluno
o 2° campo é ***hash de senha*** [no caso, MD5 digest]
o 3° campo é o dia da alteração dessa senha [desde o 1° de janeiro de 1970]: 14713
do 4° campo em diante, se não vazios, servem para aplicar a política de senha e sua expiração

O usuário então cadastrado e com uma senha válida pode logar no sistema. Ao logar, receberá o seu **shell**, que é a interface usual e que permite interagir com o sistema em linha de comando.

Nesse processo, o usuário também carrega as suas variáveis de ambiente. Por exemplo, $SHELL, $HOME, $PATH.

# 4 - Linha de comando

## 4.1 - Interpretador de comandos: shell

Ao final do processo de logon o usuário ganha o seu processo shell, que serve para disparar comandos. O **shell** é um interpretador de comandos. A cada comando que o usuário dispara, gera pelo menos um processo filho do processo shell.

A única interface comum a todos os sistemas membros da família Unix é a linha de comando, portanto toda a administração desses sistemas é feita a partir de comandos.

Os comandos são disparados no shell. Ao contrário do DOS/Windows, no mundo Unix existem dezenas de interpretadores de comandos, por exemplo:

**bash**: Bourne Shell Again;
**ksh**: Korn Shell;
**csh**: C shell;
**sh**: Bourne Shell.

No arquivo **/etc/shells** precisam estar declarados todos os shell possíveis de serem usados no sistema.

Para a linha de comando, basicamente todo shell é igual. Porém, para escrever *shell script* precisa usar a linguagem específica de cada shell.

## 4.2 - Conceitos básicos de comandos

Todo comando envolve o disparo de pelo menos um executável. Por exemplo, ao ser comandado ***ls***, o usuário está disparando o executável /bin/ls, que gera um processo.

De um modo geral, todo comando admite que seja passado pelo menos um parâmetro ou opção, além de arquivo.

Por exemplo, o comando ***ls -l /etc*** lista o conteúdo do diretório ***/etc*** no formato longo. Foi passado o parâmetro ***-l*** e o arquivo ***/etc***. Se não tivesse sido passado parâmetro nem arquivo, o comando seguiria o seu default [padrão], que no caso do ***ls*** é listar o diretório atual.

De forma análoga ao ***ls***, a maioria dos comandos admite receber parâmetros e arquivos.

# 4.3 - Conceitos de processos

Numa definição simples, processo é o executável enquanto estiver rodando. Do ponto de vista do sistema operacional, processo é uma estrutura responsável pela manutenção das informações necessárias a essa execução.

Os processos são independentes uns dos outros. Cada processo tem seu próprio PID (process identifier), que é um número. Para cada processo, no diretório "/proc" existe um subdiretório com esse número. Os comandos ***ps***, ***pstree*** e ***top*** costumam ser usados para obter informações sobre processos em execução. O comando ***kill*** envia sinais ao processo.

Para controlar a sua ação, um processo possui vários atributos. Esses atributos são:

- **PID**: Process IDentifier - é o número que identifica o processo;
- **PPID**: Parent Process IDentifier – é o processo pai, que gerou um processo filho;
- **UID**: User IDentifier – é o usuário que criou o processo;
- **GID**: Group IDentifier – é o grupo (primário) do usuário que criou o processo;
- **EUID**: Effective User ID – é o usuário efetivo no processo, independente de quem tenha disparado o executável. Ocorre quando se usa o bit ***s*** no campo dono do arquivo: num exemplo, se aluno é o dono desse arquivo e joca tem permissão para executá-lo, a identificação dono do processo (UID) será joca, mas o EUID será aluno;
- **EGID**: Effective Group ID – é o grupo (primário) efetivo no processo, independente de quem tenha disparado o executável. Ocorre quando se usa o bit ***s*** no campo grupo do arquivo: num exemplo, se aluno é o grupo desse arquivo e joca tem permissão para executá-lo, a identificação grupo do processo (GID) será joca, mas o EGID será aluno.

# 5 - Comandos básicos

A notação usada nesses exemplos será:

| | |
|---|---|
| o comando: | shell$ **ls** |
| a saída do comando | *arquivo1  dir3* |

onde a *string* **shell$** representa o shell do usuário.

Para saber onde se encontra no sistema hierárquico de arquivo:

```
shell$ pwd
/home/aluno
```

Comando para se deslocar no sistema hierárquico de arquivo:

```
shell$ cd  /etc
shell$ pwd
/etc
shell$ cd
shell$ pwd
/home/aluno
shell$ cd  ../../bin
shell$ pwd
/bin
```

NOTA:
Todo nome de arquivo precedido de uma barra direita [ex: **/etc**] representa um caminho [path] **absoluto**. Se não for precedido da barra, é um caminho **relativo**.

Listagem do diretório **/bin**:

```
shell$ ls  /bin
alsaunmute    dbus-monitor   false        link      nice          rvi        tracepath6
arch          dbus-send      fgrep        ln        nisdomainname rview      traceroute
awk           dbus-uuidgen   find         loadkeys  ntfs-3g       sed        traceroute6
basename       dd            fusermount   login     ntfs-3g       probe      setfont
....
```

Listagem no formato longo:

```
shell$ ls  -l  /bin
-rwxr-xr-x      1  root  root      123 Mai       15 09:38 alsaunmute
-rwxr-xr-x      1  root  root    36608 Abr       1 08:20 arch
lrwxrwxrwx      1  root  root        4 Jun       14 15:19 awk -> gawk
-rwxr-xr-x      1  root  root    34720 Abr       1 08:20 basename
-rwxr-xr-x      1  root  root   838824 Abr       8 07:46 bash
-rwxr-xr-x      1  root  root    57096 Abr       1 08:20 cat
-rwxr-xr-x      1  root  root    63804 Abr       1 08:20 chgrp
-rwxr-xr-x      1  root  root    59652 Abr       1 08:20 chmod
-rwxr-xr-x      1  root  root    66608 Abr       1 08:20 chown
-rwxr-xr-x      1  root  root   110996 Abr       1 08:20 cp
....
```

Mostrar o conteúdo de arquivo de texto:

```
shell$ more  /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
--Mais--(51%)
```

Pegar e mostrar todo o conteúdo de um arquivo de uma única vez:

```
shell$ cat  /etc/inittab
# inittab is only used by upstart for the default runlevel.
# ADDING OTHER CONFIGURATION HERE WILL HAVE NO EFFECT ON YOUR SYSTEM.
# System initialization is started by /etc/event.d/rcS
# Individual runlevels are started by /etc/event.d/rc[0-6]
# Ctrl-Alt-Delete is handled by /etc/event.d/control-alt-delete
# Terminal gettys (tty[1-6]) are handled by /etc/event.d/tty[1-6] and
# /etc/event.d/serial
# For information on how to write upstart event handlers, or how
# upstart works, see init(8), initctl(8), and events(5).
# Default runlevel. The runlevels used are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault:
```

Buscar expressão regular no conteúdo do arquivo:

```
shell$ grep  operator  /etc/passwd
operator:x:11:0:operator:/root:/sbin/nologin
```

Com a opção **-v**, **grep** faz uma busca reversa, isso para ignorar todas as linhas que contenham o padrão de busca:

```
shell$ grep  -v  "#"  /etc/inittab
id:5:initdefault:
```

Mostrar as 10 últimas linhas de um arquivo:

```
shell$ tail   /etc/group
stapdev:x:491:
stapusr:x:490:
wbpriv:x:88:squid
smolt:x:489:
torrent:x:488:
haldaemon:x:68:
squid:x:23:
gdm:x:42:
aluno:x:500:
jackuser:x:487:
```

Mostrar as 4 últimas linhas de um arquivo:

```
shell$ tail  -4  /etc/group
squid:x:23:
gdm:x:42:
aluno:x:500:
jackuser:x:487:
```

NOTA:
**tail** com opção **-f** mostra as últimas linhas de um arquivo e permanece tentando ler novas linhas, à medida que forem sendo escritas. Isso é muito útil para acompanhar acessos em arquivos de logs.
Exemplo: **tail -f /var/log/messages**

Mostrar as linhas do início de um arquivo:

```
shell$ head  /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

Contar o número de linhas, palavras e caracteres de um arquivo:

```
shell$ wc  /etc/passwd
40   65  1986  /etc/passwd
```

que tem 40 linhas, 65 palavras e 1986 caracteres.

Para determinar apenas o número de linhas, usar a opção ***-l***. Exemplo: ***wc -l /etc/passwd***.

Determinar o tipo de um arquivo:

```
shell$ file  /etc/group
/etc/group: ASCII text

shell$ file  /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs),
for GNU/Linux 2.6.18, stripped

shell$ file  /usr
/usr: directory
```

NOTA:
Em Unix não existe o conceito de extensão para determinar o tipo do arquivo, daí a necessidade do comando ***file***.

Copiar um arquivo:

```
shell$ cp  /etc/passwd  /tmp
```

Mover um arquivo:

```
shell$ mv  /tmp/passwd  /tmp/passwd2
```

Criar um diretório:

```
shell$ mkdir  /home/aluno/teste
```

Remover diretório vazio:

```
shell$ rmdir  /home/aluno/teste
```

Remover arquivo regular:

```
shell$ rm  /tmp/passwd2
```

NOTA:
Para remover diretório com conteúdo, usar o comando **rm** com opção **-r** ou **-R** [recursivo].
Exemplo: **rm -r /home/aluno/teste**

Pedir o menu de ajuda [help] do comando:

```
shell$ ls  --help
```

Usar o manual de ajuda do comando:

```
shell$ man  ls
```

Para ordenar o conteúdo de um arquivo na apresentação:

```
shell$ sort  /etc/passwd
```

Para recortar campos do arquivo na apresentação:

```
shell$ cut  -d:  -f1  /etc/passwd
```

Para encontrar todos os arquivos de nome *passwd* no diretório /etc, usar find:

```
shell$ find  /etc  -type  f  -name  passwd  -print
```

O comando find também pode ser usado para encontrar arquivos que contenham determinado conteúdo. Por exemplo, para encontrar os arquivos que contenham a palavra aluno no diretório /etc, comandar:

```
shell$ find  /etc  -type  f  -exec  grep  -i  aluno  {} \;  -print
```

Para criar o usuário juca, antes precisa ganhar privilégios de root:

```
shell$ sudo  su
[sudo] password for aluno:
shell# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
```

```
shell# /usr/sbin/useradd  -m  -s  /bin/bash juca
```

onde a opção **-m** é para criar a home do usuário e **-s** especifica o shell.

Como não foi feito referência a grupo, foi criado o grupo juca:

```
shell# tail  -1  /etc/group
juca:x:1002:
```

Para criar o usuário joca, no grupo primário juca, comandar:

```
shell# /usr/sbin/useradd  -m  -s  /bin/bash  -g  juca  joca
```

onde a opção **-g** é o groupname do grupo onde será criado o usuário joca.

Para excluir o usuário joca, comandar **userdel**:

```
shell# userdel  joca
```

Para criar um grupo, usar o comando **groupadd**:

```
shell# groupadd  jeca
```

Para excluir um grupo, usar o comando **groupdel**:

```
shell# groupdel  jeca
```

Para retornar o shell de aluno, comandar **exit**:

```
shell# exit
shell$
```

Comandos adicionais:

**uptime**: diz a quanto tempo o sistema está rodando;
**history**: mostra o histórico de comandos disparado no shell;
**ps**: mostra status de processos;
**pstree**: mostra a árvore de processos;
**adduser**: é linque simbólico para useradd, usado para incluir usuários no sistema local;
**usermod**: modifica a conta do usuário;
**passwd**: usado para alterar a senha;
**echo**: usado para enviar mensagens para a saída padrão;
**env**: altera o ambiente na execução de comando;
**printenv**: mostra o ambiente total ou parte deste;
**touch**: altera a data da última edição do arquivo. Se o arquivo não existe, é criado;
**df**: reporta uso do espaço no sistema de arquivo [file system disk space usage];
**du**: estima o uso de ocupação do espaço por arquivo [diretório, file space usage];
**free**: mostra a quantidade de memória usada e livre no sistema;
**uname**: mostra informações sobre o sistema;
**hostname**: mostra ou configura o nome da máquina ou domínio;
**id**: mostra o usuário real e efetivo, além dos grupos a que pertence o usuário;
**export:** usado no shell *sh* e descendentes para carregar ou modificar uma variável;
**top:** mostra, interativamente, uma lista de processos ordenados pelo consumo de recursos do computador.
**uniq:** reporta ou omite linhas duplicadas: "uniq -d" mostra apenas linhas duplicadas, "uniq -u" mostra apenas linhas únicas (não duplicadas).
**which**: mostra o caminho completo para os comandos. Exemplo: "which passwd".

# 6 - Pipes

Os pipes são implementações de comunicação entre processos. Na linha de comando, se for disparado mais de um executável de uma vez, vai gerar mais de um processo. E se esses executáveis estiverem ligados por um pipe "|", então a saída de um processo será a entrada do outro processo. Essa sequência de comunicação vai da esquerda para a direita.

Exemplo:
**ls /bin | more**

onde **ls /bin** lista o conteúdo do diretório **/bin**, e como essa lista não cabe nas dimensões do shell, então é usado na mesma linha de comando o executável **more** para apresentar essa listagem paginadamente.

Outro exemplo:
**cat /etc/passwd | grep bash | wc -l**

que determina o número de usuário cadastrados no sistema que têm o shell /bin/bash.

Outros exemplos:

**cat /etc/passwd | grep aluno** **"cat /etc/passwd | grep -n aluno" mostra o número da linha onde está a ocorrência**

**cat /etc/group | sort**

**cut -d: -f1 /etc/passwd | sort**

**cat /etc/passwd | grep -i juca | wc -l**

**ls -l /etc | grep "^d" | grep "a$"**

**cut -d -f3 /etc/passwd | sort -n**

# 7 - Redirecionamentos

Os redirecionamentos alteram a entrada ou saída padrão de um comando. Normalmente, a entrada padrão é o teclado, e a saída padrão é o monitor.

## 7.1 – Saída padrão

Por exemplo, o comando **ls /etc > /tmp/ls_etc.txt**

faz uma listagem do diretório **/etc** e envia a saída para o arquivo **/tmp/ls_etc.txt**. Se o arquivo não existe, é criado. No entando, se o arquivo existisse, o conteúdo anterior seria perdido.

Caso o arquivo existisse, para manter o conteúdo anterior e acrescentar o novo conteúdo no final do arquivo, o comando seria **ls /etc >> /tmp/ls_etc.txt**.

NOTA:
Quando se redireciona a saída padrão, esta vai para um arquivo e, portanto, não tem saída no shell [não vem saída para o monitor].

## 7.2 – Saída de erro

Caso o comando encontrasse algum erro, retornaria uma mensagem de erro. Por exemplo, tentar criar um diretório que já existe gera mensagem de erro. Essa mensagem de erro é chamada de saída de erro, e para ser redirecionada precisa incluir o número 2 antes do sinal de maior ">".

**mkdir /etc 2> /tmp/erro_do_comando.txt**

NOTA:
Se não usar essa notação a saída de erro não é redirecionada e vem para o monitor.

## 7.3 – Entrada padrão

A entrada padrão é redirecionada com o sinal "<" após o comando. Com isso, o comando irá ler a entrada de um arquivo e não mais do teclado.

Por exemplo, considere o executável que, ao ser disparado pergunta pelo nome e idade, depois finaliza informando o nome e a idade:

```
shell$ nome_idade.sh
        entre com o seu nome: juca
        entre com a sua idade: 286
Seu nome é juca, sua idade é 286 anos
```

Se esse comando fosse disparado com redirecionamento da entrada padrão, não esperaria pelas entradas de teclado.

```
shell$ nome_idade.sh < arq_entrada
        entre com o seu nome:
        entre com a sua idade:
Seu nome é juca, sua idade é 286 anos
```

No arquivo **arq_entrada** existem apenas duas linhas, que são as entradas pedidas pelo comando:

```
--- arq_entrada ---
juca
286
----------------------
```

Desse modo, existem 3 casos para os redirecionamentos:

- saída padrão **>** [o correto seria 1>, porém pode ser omitido o número 1]
- saída de erro **2>**
- entrada padrão **<**

A forma clássica usada no shell para redirecionar de uma única vez tanto a saída de erro quanto a saída padrão para um arquivo, é:

**comando > arquivo_saida 2>&1**

# 8 - Permissões do sistema de arquivo

As permissões do sistema de arquivo definem o acesso aos arquivos. Por acesso, nesse caso entendemos leitura, escrita e execução, entre outros. Para alterar esses acessos, existem três comandos: *chmod*, *chown* e *chgrp*.

Para entender o conceito de permissão de acesso ao sistema de arquivo, antes é necessário ter em mente que:

- Todo arquivo tem um dono. O dono é quem cria o arquivo;
- Todo arquivo pertence a um grupo de usuários. O conceito de grupo envolve usuários com atividades que requerem um conjunto semelhante de permissões de acesso a arquivos;
- Aqueles usuários que não são o próprio dono do arquivo ou não pertencem ao grupo do dono, são considerados "outros" do ponto de vista do acesso ao arquivo;
- Permissões são concessões no acesso a arquivos. A nível de sistema de arquivo, usa-se [por exemplo] os bits ***r*** (read – leitura), ***w*** (write – escrita) e ***x*** (execução ou acesso a diretório).

  Para as permissões, existem 3 campos distintos: o do dono, do grupo e outros.

  Por exemplo, o comando ***ls -l /home/aluno*** mostra:

```
shell$ ls -l /home/aluno
-rw-rw-r--    1  aluno  users    59939  Ago 18 21:13  arq.txt
drwxr-xr-x    2  aluno  users     4096  Mai 12 09:15  dir1
```

Olhando o resultado dessa listagem no formato longo, identificamos que o último campo de cada linha é o nome do arquivo, que já aparecia na listagem **ls** [sem formato longo]. Mas que agora apresenta mais atributos para cada arquivo.

Uma visão detalhada desses atributos mostra que:

| | | | | | | |
|---|---|---|---|---|---|---|
| -rw-rw-r--<br>drwxr-xr-x | 1<br>2 | aluno<br>aluno | users<br>users | 59939<br>4096 | Ago 18 21:13<br>Mai 12 09:15 | arq.txt<br>dir1 |
| permissões do sistema de arquivo | nº de linques para o arquivo | username (dono do arquivo) | groupname | tamanho do arquivo (bytes) | data da última edição no arquivo | nome do arquivo |

No entanto, o que foi chamado de permissões precisa ter descontado o primeiro caracter, que designa o tipo do arquivo. O caracter "**d**" (por exemplo, **d**rwxr-xr-x) designa que o tipo do arquivo é um **diretório**. Se iniciar com um traço "**-**" (por exemplo, **-**rw-rw-r--), o arquivo é do tipo **regular** ou ordinário, se iniciar com um caracter **"l"** (por exemplo, **l**rwxrwxrwx), trata-se de **link** (linque) simbólico.

Dos nove bits de permissão (por exemplo, rw-rw-r--), os três primeiros são permissões do dono, os três seguintes são do grupo e os três últimos dos outros. Não existe sobreposição dessas permissões. Por exemplo, as permissões dadas ao grupo não afetam as permissões de acesso para o dono do arquivo.

| | | |
|---|---|---|
| -**rw-**rw-r-- | -rw-**rw-**r-- | -rw-rw-**r--** |
| dono do arquivo | grupo do arquivo | outros |

# 8.1 – chmod

O comando ***chmod*** altera as permissões de acesso ao arquivo. Pode ser alterado no modo numérico [octal] ou simbólico.

Para poder alterar as permissões, o usuário precisa ser dono do arquivo ou então ser root.

### 8.1.1 - Modo numérico ou absoluto (octal)

O modo numérico usa a notação octal para traduzir os bits de permissão dos arquivos em números de 0 a 7. A tabela abaixo mostra a relação entre bits de permissão e números.

| rwx | binário | octal |
|---|---|---|
| --- | 000 | 0 |
| --x | 001 | 1 |
| -w- | 010 | 2 |
| -wx | 011 | 3 |
| r-- | 100 | 4 |
| r-x | 101 | 5 |
| rw- | 110 | 6 |
| rwx | 111 | 7 |

Convém notar que o bit ***r*** está associado ao octal **4**, ***w*** ao **2** e ***x*** ao **1**:

*r* => 4
*w* => 2
*x* => 1

e que os valores octais são cumulativos. Por exemplo, a permissão "**r-x**" é 4+1=**5**, "**rwx**" é 4+2+1=**7**.

Exemplos:

```
shell$ chmod  444  arq.txt
shell$ ls  -l  arq.txt
-r--r--r--     1  aluno  users     59939   Ago 18 21:13  arq.txt
shell$ chmod  246  arq.txt
shell$ ls  -l  arq.txt
--w-r--rw-     1  aluno  users     59939   Ago 18 21:13  arq.txt
shell$ chmod  750  dir1
shell$ ls  -ld  dir1
-rwxr-x---     1  aluno  users      4096   Mai 12 09:15  dir1
```

NOTA:
O bit ***x*** em arquivo regular é permissão de execução, já em diretório dá acesso ao seu conteúdo. Afinal, o diretório é estrutura organizacional e não pode ser executado.

### 8.1.2 - Modo Simbólico

No modo simbólico, a alteração de permissão de arquivos segue o esquema:

chmod [ugoa][ + – = ][rwxst]

onde:

- Os caracteres [ugoa] designam as permissões do dono [**u**, user], do grupo [**g**, group], dos outros [**o**, others] e de todos [**a**, all];
- Os sinais [ + – = ] definem a seleção para as permissões, onde o sinal [+] acrescenta a permissão às que já existem no arquivo, o sinal [–] subtrai das existentes e o sinal [=] remove todas as permissões que haviam no arquivo e impõem a que está sendo alterada para o(s) usuário(s);
- Os bits [**rwxst**] são os bits de permissão a serem alterados no arquivo.

Exemplos:

```
shell$ ls -l arq.txt
--w-r--rw-    1 aluno users    59939 Ago 18 21:13 arq.txt
shell$ chmod u+rx arq.txt
shell$ ls -l arq.txt
-rwxr--rw-    1 aluno users    59939 Ago 18 21:13 arq.txt
shell$ chmod g-r arq.txt
shell$ ls -l arq.txt
-rwx---rw-    1 aluno users    59939 Ago 18 21:13 arq.txt
shell$ chmod a+x arq.txt
shell$ ls -l arq.txt
-rwx--xrwx    1 aluno users    59939 Ago 18 21:13 arq.txt
shell$ chmod ug=r arq.txt
shell$ ls -l arq.txt
-r--r--rwx    1 aluno users    59939 Ago 18 21:13 arq.txt
shell$ ls -ld dir1
drwxr-x---    1 aluno users     4096 Mai 12 09:15 dir1
shell$ chmod a=rx dir1
shell$ ls -ld dir1
dr-xr-xr-x    1 aluno users     4096 Mai 12 09:15 dir1
```

### 8.1.3 - Bits *s* e *t*

Além dos bits de permissão ***r***, ***w*** e ***x***, também são frequentemente usados os bits ***s*** e ***t***.

#### 8.1.3.1 – Bit *s*

O bit ***s*** em arquivos regulares pode estar tanto no campo do dono do arquivo quanto do grupo. Por exemplo: **-rwsr-x--x**, **-rwxr-s--x**, **-rwsr-s--x**. Em arquivos regulares, o bit ***s*** é também um bit de execução.

Se o bit ***s*** aparecer no campo do **dono** do arquivo é **SUID** [Set User ID], se aparecer no

campo do **grupo** é **SGID** [Set Group ID]. Esse bit altera a identificação do usuário (ou do grupo) no processo gerado pela execução do arquivo executável.

Se for SUID, atribui a permissão do processo como idêntica a do dono do arquivo, independentemente de quem tenha executado.

Se for SGID, atribui a permissão do processo como idêntica a do grupo do arquivo, independentemente de quem tenha executado.

Por exemplo, o executável usado para alterar senha é o comando **passwd**, que tem as seguintes permissões:

```
shell$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 22520 Fev 26 2009 /usr/bin/passwd
shell$ ls -l /etc/shadow
-r-------- 1 root root 1324 Out 12 21:00 /etc/shadow
```

No caso, o dono do executável **passwd** é o root, porém este permite que qualquer usuário execute o comando, como pode ser visto pelo bit **x** no campo outros.

Porém, quando alguém executa o comando **passwd**, também precisa escrever a nova **hash** de senha no arquivo **/etc/shadow**, que não tem permissão de escrita para outros [por questões de segurança, obviamente].

Então, o bit ***s*** no campo do dono do executável **passwd** faz com que o processo originado por qualquer usuário tenha privilégio de acesso do dono do arquivo, isto é, de root, que consegue escrever no **/etc/shadow**. Desse modo, o usuário consegue alterar a sua própria senha.

Exemplo:

```
shell$ id
uid=1000(aluno) gid=1000(users) grupos=1000(users)
shell$ pwd
/tmp
shell$ cp  /bin/mkdir  .
shell$ ls  -l  mkdir
-rwxr-xr-x  1  aluno  users  43504  Abr  11 14:37  mkdir
shell$ chmod  4755  mkdir
shell$ ls  -l  mkdir
-rwsr-xr-x  1  aluno  users  43504  Abr  11 14:37  mkdir
```

NOTA:
O comando chmod, no modo octal, recebe um parâmetro com 4 dígitos. Porém, se o[s] primeiro[s] bit[s] for[em] 0 [zero], este[s] pode[m] ser omitido[s]. Exemplo: ***chmod 0123 arq.txt*** é idêntico a ***chmod 123 arq.txt***. Ou então, ***chmod 0023 arq.txt*** é idêntico a ***chmod 23 arq.txt***.

Para verificar o funcionamento desse bit será necessário outro usuário, que não pertença ao grupo de aluno. Neste caso, se ainda não existe adicionar o usuário juca. Para adicionar um novo usuário, precisa ser **root**:

```
shell# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
shell# /usr/sbin/useradd  -m  -s  /bin/bash  -c  "usuario juca"  -d  /home/juca  juca
shell# tail  -1  /etc/passwd
juca:x:1001:1001:usuario juca:/home/juca:/bin/bash
shell# passwd  juca
Mudando senha para o usuário juca.
Nova senha:
```

Agora, o usuário juca loga em outro shell [por exemplo, <CTRL><ALT>F3], e usa o executável em **/tmp/mkdir** para criar um novo diretório:

```
shell$ id
uid=1001(juca) gid=1001(juca) grupos=1001(juca)
shell$ cd /tmp
shell$ /tmp/mkdir juca
shell$ ls -ld /tmp/juca
drwxr-xr-x 2 aluno juca 4096 Abr 11 15:00 /tmp/juca
```

NOTA:
Repare que o diretório criado por juca pertence a **aluno** e não juca.

Analogamente ao exercício acima, se o dono do arquivo [aluno] mudar as permissões para ***-rwxr-sr-x*** com o comando **chmod 2755 /tmp/mkdir**, quando juca criar outro diretório este será do grupo de aluno [users].

Do mesmo modo, se aluno usar o comando **chmod 6755 /tmp/mkdir**, irá alterar as permissões do arquivo **/tmp/mkdir** para ***-rwsr-sr-x***. Nesse caso, se juca usar o comando /tmp/mkdir para criar outro diretório, este pertencerá a aluno e ao grupo users [que é o grupo de aluno].

8.1.3.2 – Bit *t*

O bit t em diretórios dá a permissão append-only [sticky bit], e é usado em diretórios públicos como o /tmp.

```
shell# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
shell# ls -ld /tmp
drwxrwxrwt 28 root root 4096 Abr 10 12:00 /tmp
shell# mkdir /tmp2
shell# ls -ld /tmp2
drwxrwxrwx 2 root root 4096 Abr 11 15:00 /tmp2
shell# chmod 1777 /tmp2
drwxrwxrwt 2 root root 4096 Abr 11 15:00 /tmp2
```

No exemplo abaixo, o usuário root irá criar um novo diretório público de nome **/tmp2**.

# 8.2 - chown

O comando **chown** é usado pelo root para alterar a posse do arquivo.
Por exemplo:

```
shell# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
shell# chown aluno /tmp/mkdir
shell# ls -l /tmp/mkdir
-rwsr-xr-x 1 aluno aluno 43504 Abr 11 14:37 mkdir
```

Além de alterar a posse, também poderia alterar o grupo primário. Por exemplo:

```
shell# touch /tmp/arq.txt
shell# chown aluno:users /tmp/arq.txt
shell# ls -l /tmp/arq.txt
-rw-rw-r-- 1 aluno users 0 Ago 15 15:32 /tmp/arq.txt
```

# 8.3 - chgrp

O comando **chgrp** é usado pelo root para alterar o grupo do arquivo.
Por exemplo:

```
shell# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
shell# chgrp aluno /tmp/mkdir
shell# ls -l /tmp/mkdir
-rwsr-xr-x 1 aluno aluno 43504 Abr 11 14:37 mkdir
```

# 9 - Umask

Umask é uma máscara usada na criação de arquivos. O valor que estiver configurado nessa máscara, que é individual, será usado como o modo de criação do arquivo.

Para saber qual a máscara, comandar:

```
shell$ umask
0002
```

Esse comando mostra que a máscara do usuário aluno é 0002. Então, se aluno criar um arquivo, as permissões serão **-rw-rw-r--**, que é octal 0664. Se aluno criar um diretório, as permissões serão **drwxrwxr-x**, que é octal 0775.

Esses valores de permissões na criação vêm da diferença binária entre 0666 para arquivos regulares e 0777 para diretórios.

Para calcular as permissões do arquivo criado, são usados os operadores bitwise NOT na máscara e posteriormente o bitwise AND binário entre 0666 e esse número, para o caso de arquivo regular. Se fosse diretório, seria 0777.

Exemplos de operações bitwise NOT e AND:

- NOT(0) = 1
- NOT(1) = 0
- NOT(110) = 001
- 1 AND 0 = 0
- 1 AND 1 = 1
- 0 AND 0 = 0
- 001 AND 111 = 001

Como exemplo, calcular as permissões de um arquivo regular criado com **umask 0002**. Primeiramente, converter a máscara e o número 0666 para binário:

0002 = 000 000 000 010
0666 = 000 110 110 110

Depois, aplicar o operador NOT na máscara:

NOT(000 000 000 010) = 111 111 111 101

Por fim, calcular o operador AND desses números binários:

111 111 111 101 AND
000 110 110 110
---
000 110 110 100 => 0664 [**-rw-rw-r--**]

Num outro exemplo, usando essa mesma máscara porém criando um diretório, é necessário primeiro converter 0777 para binário:

0777 = 000 111 111 111

Calcular o operador AND entre o NOT da máscara e 0777 binário:

111 111 111 101 AND
000 111 111 111
---
000 111 111 101 => 0775 [**drwxrwxr-x**]

Ainda como exemplo, alterar a máscara para 0027. Para isso, comandar umask e passar como parâmetro o novo valor:

```
shell$ umask  0027
hell$ umask
0027
```

Agora, calcular as permissões de arquivo regular e diretório criados com essa nova máscara.

Inicialmente, converte essa máscara para binário:

0027 = 000 000 010 111

Depois, aplicar o operador NOT nessa máscara:

NOT(000 000 010 111) = 111 111 101 000

Por fim, calcular o operador AND desses números binários para o caso de arquivo regular [0666]:

111 111 101 000 AND
000 110 110 110
---
000 110 100 000 => 0640 [**-rw-r-----**]

par o caso de criação de diretório, usar 0777 [000 111 111 111]:

111 111 101 000 AND
000 111 111 111
---
000 111 101 000 => 0750 [**drwxr-x---**]

Ainda como exemplo, alterar a máscara para 0022. Para isso, comandar umask e passar como parâmetro o novo valor:

```
shell$ umask  0022
hell$ umask
0022
```

Agora, calcular as permissões de arquivo regular e diretório criados com essa nova máscara.

Inicialmente, converte essa máscara para binário:

0022 = 000 000 010 010

Depois, aplicar o operador NOT nessa máscara:

NOT(000 000 010 010) = 111 111 101 101

Por fim, calcular o operador AND desses números binários para o caso de arquivo regular [0666]:

111 111 101 101 AND
000 110 110 110
---
000 110 100 100 => 0644 [**-rw-r--r--**]

par o caso de criação de diretório, usar 0777 [000 111 111 111]:

111 111 101 101 AND
000 111 111 111
---
000 111 101 101 => 0755 [**drwxr-xr-x**]

O comando *umask* também apresenta a máscara na forma simbólica:

```
shell$ umask  -S
u=rwx,g=rwx,o=rx
```

NOTA:

À primeira vista, passa a impressão que é simples obter a máscara dadas as permissões, fazendo o caminho inverso do que foi feito acima. Ou seja, definir as permissões desejadas e depois calcular a máscara a ser adotada. No entanto, esse processo inverso não costuma dar certo pois o caminho não é unívoco. Por exemplo, para **arquivos regulares** as máscaras 0027 e 0026 dão o mesmo resultado, que são permissões 0640 [**-rw-r-----**], já para diretórios a máscara 0027 dá permissões 0750 [**drwxr-x---**], mas a máscara 0026 dá permissões 0751 [**drwxr-x--x**]. Por isso, na prática se usa máscara 0027 e não 0026.

Mas se alguém quiser calcular a máscara dadas as permissões, deverá usar o operador XOR:

```
0640   000 110 100 000                    <=    permissões desejadas
                       XOR
0666   000 110 110 110                    <=    máscara para arquivo regular
       --------------------
       000 000 010 110      =>    máscara: 0026


0750   000 111 101 000                    <=    permissões desejadas
                       XOR
0777   000 111 111 111                    <=    máscara para diretório
       --------------------
       000 000 010 111      =>    máscara: 0027
```

# 10 - Scripts

Scripts são roteiros. O conteúdo do arquivo é texto a ser interpretado por algum interpretador. Se o interpretador for shell, será um **shell script**, se for o perl, será um perl script e assim por diante.

Como em Unix não existe o conceito de extensão de arquivo, que poderia ser usado para associar o interpretador ao arquivo script, então o interpretador precisa ser declarado na primeira linha do arquivo.

Para escrever ou editar os scripts, é necessário um editor de texto. No mundo Unix, o único editor efetivamente padronizado e presente em todos os sabores Unix é o **vi** [visual interface]. Embora o **vi** não seja amigável ao iniciante, vale a pena o esforço em aprender como usar esse editor.

## 10.1 – Editor vi/vim

É certo que existem muitos editores, alguns até mais práticos que o **vi**. Porém, não há a garantia de que ele esteja instalado na máquina específica a ser administrada.

O editor **vim** é o **vi** "improved" [aprimorado], e deve ser usado sempre que estiver disponível, pois é mais amigável que o vi. Quanto aos comandos, são idênticos.

A primeira noção que se precisa ter do vi/vim é que se trata de um editor de **2 modos**, que são INSERT [edição] ou COMMAND [comando]. Afinal, vi/vim usa apenas o teclado, então num caso teclar estará editando o texto de algum arquivo, no outro estará dando comandos [por exemplo, para salvar o texto].

Para entrar no modo inserção, basta pressionar a letra **i** [de insert]. A partir daí, qualquer tecla que for batida irá escrever no arquivo.

Para sair do modo inserção e entrar no modo comando, deve comandar a sequência **<ESC> <SHIFT> <:>**, que são as 3 teclas **ESC+SHIFT+:**, e posteriormente pressionar a tecla [ou teclas] com o comando. Nessa situação, a tecla **w** [write] salva o texto no arquivo, a tecla **q** [quit] abandona o editor de texto e volta para o shell.

Exemplo: para criar o arquivo teste.txt e escrever no seu interior, basta seguir a sequência:

1) no shell, comandar "**vi teste.txt**", que abre o editor e toma o shell:

```
shell$ id
uid=1000(aluno) gid=1000(users) grupos=1000(users)
shell$ vi  teste.txt
```

```
~
~
~
~
"teste.txt" [Novo arquivo]
```

2) Pressionar a tecla "**i**" para entrar no modo de inserção de texto;

```
~
~
~
~
-- INSERÇÃO --
```

3) Escrever o texto "teste de escrita vi";

```
teste de escrita vi
~
~
~
-- INSERÇÃO --
```

4) Sair do modo de inserção e entrar no modo comando com a sequência de teclado **<ESC> <SHIFT> <:>**

```
teste de escrita vi
~
~
~
:
```

5) pressionar as teclas **wq**, que vão salvar e sair do editor.

Os principais caracteres usados no modo comando são:

**x** deleta o carater onde está o cursor;
**dd** apaga a linha onde está o cursor;
**u** desfaz a edição;
**r** substitui o caracter na posição do cursor pelo que for teclado após o r.
**a** inicia no modo insert uma posição à direita da palavra sob o cursor.

# 10.2 – Scripts

Além da linha de comando, o administrador do sistema costuma usar scripts [roteiros] para automatizar tarefas rotineiras. Desse modo, ao invés de disparar a mesma sequência de comandos todo dia, basta escrever essa sequência num arquivo de texto, definir na primeira linha o nome do interpretador, colocar permissão para execução e disparar o script.

**Exemplo 1:**

```
----- teste.sh -------
#!/bin/bash
echo "Script teste"
----------------------
```

Dar permissão de execução:

```
shell$ chmod  755  teste.sh
```

Disparar o executável script:

```
shell$ ./teste.sh
Script teste
```

**Exemplo 2:**

```
--- nome_idade.sh ---------------------------------
#!/bin/bash
echo -n "Escreva o seu nome: "
read nome
echo -n "Escreva a sua idade: "
read idade
echo "Seu nome é $nome, sua idade é $idade"
---------------------------------------------------------
```

Dar permissão de execução:

```
shell$ chmod  755  nome_idade.sh
```

Conforme foi visto no capítulo 6 [Redirecionamentos], para a execução desse script pode-se informar as respostas via teclado ou então de algum arquivo.

Um variação é incluir na última linha desse script a entrada **sleep 600**, que vai deixar a aplicação em sleep por 600 segundos após o disparo [ou seja, não finaliza de imediato].

Com isso, o processo pode ser visto rodando com o comando **ps -ef**.

Este script também pode ser aproveitado para entender o conceito de processo em **background** com o comando abaixo:

```
shell$ nome_idade.sh  <  arq_entrada  >  arq_saida  &
[1] 10446
shell$ jobs
[1]+  Executando          ./nome_idade.sh < arq_entrada > arq_saida &
```

Para verificar se o processo está rodando, usar o comando ps:

```
shell$ ps -ef | grep nome_idade.sh
aluno   10446  2114  0  16:01  pts/1    00:00:00  /bin/bash  ./nome_idade.sh
aluno   10449  2114  0  16:02  pts/1    00:00:00  grep  nome_idade.sh
```

NOTA:
O processo em background não pode mais ser parado por <CTRL>c. Para pará-lo, é necessário o comando **kill** no PID [10446] do processo:

```
shell$ kill  10446
[1]+  Terminado              ./nome_idade.sh < arq_entrada > arq_saida
```

O exemplo 3, script **sleep.sh**, vai servir para mostrar outra maneira de enviar um processo para background.

**Exemplo 3:**

```
--- sleep.sh ---
#!/bin/bash
echo  "Ola!"
sleep  600
-----------------
```

```
shell$ ./sleep.sh  >  saida
```

A execução desse script dura 600 segundos [10 minutos], então é coveniente enviá-lo para **background** para liberar o shell. Isso é feito comandando **<CTRL>z**, que leva para background e deixa o processo stopped [parado].

```
shell$ ./sleep.sh  >  saida
^Z
[1]+  Parado                 ./sleep.sh
```

O comando jobs vai mostrar que o processo está parado em **background**:

```
shell$ jobs
[1]+  Parado              ./sleep.sh
```

Para reiniciar a execução em background, comandar **bg**:

```
shell$ bg  %1
[1]+ ./sleep.sh &
shell$ jobs
[1]+  Executando          ./sleep.sh &
```

onde **%1** é o primeiro job da lista de processos em **background**.

Para trazer o processo de volta para o shell [**foreground**], comandar **fg**:

```
shell$ fg  %1
./sleep.sh
^C
shell$
```

No exemplo 4 é apresentado um script simples de backup.

**Exemplo 4:**

```
--- backup.sh ----------------------------------------
#!/bin/bash
# backup diario de  /bin
if [ -s backup.tar.gz ]; then
        mv  -f  backup.tar.gz  backup.tar.gz.OLD
fi
tar  -cvf backup.tar  /bin
gzip  backup.tar
data_agora=`date`
string="Backup finalizado em  $data_agora"
echo  $string
-----------------------------------------------------
```

**Exemplo 5:**

```
--- loop.sh ----------
#!/bin/bash
echo  "Iniciando..."
while [ 2 -ge 1 ]; do
        echo  "."
        sleep 2
done
-----------------------------
```

No caso do script **loop.sh**, o usuário aluno o copia para o diretório **/tmp**, dá permissão de execução "**chmod 755 /tmp/loop.sh**" e dispara esse script:

```
shell$ id
uid=1000(aluno) gid=1000(users) grupos=1000(users)
shell$ /tmp/ loop.sh
```

Após o usuário aluno iniciar a execução do script, noutro terminal, o usuário root acompanha o uso da CPU com o comando **top**, e observa que o script não está consumindo muita CPU.

Posteriormente, o usuário aluno comenta a linha **sleep 2** e dispara o executável de novo:

```
--- loop.sh ----------
#!/bin/bash
echo  "Iniciando..."
while [ 2 -ge 1 ]; do
        echo  "."
        # sleep 2
done
-----------------------------
```

Noutro terminal, o usuário juca também o dispara. Simultaneamente, o root permanece monitorando o desempenho da CPU e percebe agora um uso muito intenso da CPU por parte desse script, que tem 2 processos rodando: um de aluno e o outro do juca. Cada processo consome cerca de 50% da CPU.

Para minizar o consumo, o root baixa a prioridade ao mínimo possível para um dos processos [o de juca, PID 13218] com o comando **renice** e volta a monitorar a CPU com o comando **top**:

```
shell# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
shell# renice +19 -p 13218
shell# top
```

Agora o root percebe que o processo de aluno [PID 21572], sozinho, passou a consumir cerca de 100% da CPU, então decide pará-lo com o comando **kill**:

```
shell# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
shell# kill 21572
```

Novamente o usuário aluno dispara o seu script, porém agora da forma correta, usando o comando **nice** para iniciar o esse processo com baixa prioridade:

```
shell$ id
uid=1000(aluno) gid=1000(users) grupos=1000(users)
shell$ nice -n 19 /tmp/ loop.sh
```

O root monitora com **top** e percebe que agora os dois processos voltam a consumir, cada um, cerca de 50% da CPU. Só que agora, devido à baixa prioridade desses processos, eles não atrapalham os demais, pois estão usando apenas os recursos ociosos do sistema. Nessa situação, quando outro usuário executar alguma aplicação, ele terá por *default* prioridade maior que os dois scripts que estão rodando.

**Exemplo 6:**

Neste exemplo, o script é Perl e tem uma sintaxe diferente do shell script. A ideia aqui é verificar o uso da memória.

```
--- mem.pl ----------------
#!/usr/bin/perl
$cont=0;
for(;;){
        $cont++;
        $vetor[$cont]=$cont*10000;
}
-----------------------------
```

O usuário aluno dispara esse script num terminal e o root monitora em outro, com os comandos **top** e **free**. O root observa que esse script rapidamente aloca toda a memória disponível.

# 11 – Comandos de rede

Os comandos apresentados neste capítulo são, de um modo geral universais e usados em diversos sistemas operacionais. No entanto, nesse caso o foco está no Linux e por causa disso alguns deles somente funcionam no Linux.

Para administrar a rede precisa ser root, então usar o comando *sudo* para ganhar um shell de root:

```
shell$ id
uid=1000(aluno) gid=1000(users) grupos=1000(users)
shell$ sudo su -
shell# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
shell#
```

## 11.1 – lspci

O comando *lspci* é usado para listar todos os dispositivos PCI da máquina. Como o interesse no momento é saber se tem uma interface de rede ethernet, então o comando pode ser usado filtrando a saída com *grep*, para reduzir o conteúdo de saída:

```
shell# lspci | grep -i ethernet
01:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168B PCI Express Gigabit Ethernet controller (rev 06)
```

Notar que com o grep foi usada a opção "**-i**", isso para tornar a busca insensível à caixa (case insensitive) e encontrar a palavra Ethernet (com E maiúsculo). Desse modo, descobrimos que tem uma interface de rede Realtek.

## 11.2 – ifconfig

O comando *ifconfig* é usado para listar (e também configurar) as configurações lógicas

na(s) interface(s) de rede. No exemplo abaixo, não foi passada nenhuma opção para o comando, portanto ele apenas listou o que está configurado.

```
shell# ifconfig
eth0      Link encap:Ethernet  Endereço de HW 00:21:97:80:7C:FF
          inet end.: 192.168.1.10  Bcast:192.168.1.255  Masc:255.255.255.0
          endereço inet6: fe80::221:97ff:fe80:7cff/64 Escopo:Link
          UP BROADCASTRUNNING MULTICAST  MTU:1500  Métrica:1
          RX packets:55602 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10620 errors:0 dropped:0 overruns:0 carrier:0
          colisões:0 txqueuelen:1000
          RX bytes:8883790 (8.4 MiB)  TX bytes:2304141 (2.1 MiB)
          IRQ:28 Endereço de E/S:0x6000

lo        Link encap:Loopback Local
          inet end.: 127.0.0.1  Masc:255.0.0.0
          endereço inet6: ::1/128 Escopo:Máquina
          UP LOOPBACKRUNNING  MTU:16436  Métrica:1
          RX packets:18 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
          colisões:0 txqueuelen:0
          RX bytes:1268 (1.2 KiB)  TX bytes:1268 (1.2 KiB)
```

Pode ser visto que existe a interface de rede eth0, que está configurada com o endereço IP 192.168.1.0 e máscara de rede 255.255.255.0. Normalmente, a primeira interface de rede é a eth0, a segunda é eth1, e assim sucessivamente.

# 11.3 – ping

O comando *ping* é usado para fazer diagnóstico de acesso em rede. Este comando recebe como opção ou parâmetro o endereço IP da máquina na qual se quer testar se tem acesso ou não. Havendo resposta (echo), indica que existe o acesso.

```
shell# ping  192.168.1.12
PING 192.168.1.12 (192.168.1.12) 56(84) bytes of data.
64 bytes from 192.168.1.12 (192.168.1.12): icmp_seq=1 ttl=64 time=0.057 ms
64 bytes from 192.168.1.12 (192.168.1.12): icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 192.168.1.12 (192.168.1.12): icmp_seq=3 ttl=64 time=0.055 ms
64 bytes from 192.168.1.12 (192.168.1.12): icmp_seq=4 ttl=64 time=0.048 ms
^C
--- 192.168.1.12 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3364ms
rtt min/avg/max/mdev = 0.048/0.052/0.057/0.004 ms
```

# 11.4 – route

O comando *route* mostra as configurações (e também altera) de rotas da máquina. No exemplo, o comando não está alterando nada, está apenas listando as rotas.

```
shell# route  -n
Tabela de Roteamento IP do Kernel
Destino         Roteador        MáscaraGen.     Opções Métrica  Ref   Uso  Iface
192.168.1.0     0.0.0.0         255.255.255.0   U      1        0     0    eth0
0.0.0.0         192.168.1.1     0.0.0.0         UG     0        0     0    eth0
```

A opção "**-n**" foi usada para não resolver nomes (uso de serviço DNS). Se fosse resolver nomes demoraria mais e não mostraria os endereços IP, e sim os nomes das máquinas.

No caso, a saída do comando *route* mostrou que existe uma rota *default* (padrão, *gateway*) para 192.1.68.1.1.

# 11.5 – nslookup

O comando *nslookup* é usado para fazer uma consulta (query) a um serviço de nomes. Esta consulta normalmente tem o intuito de resolver um nome e determinar o endereço IP.

```
shell# nslookup  www.jairo.pro.br
Server:         186.251.39.194
Address:        186.251.39.194#53

Non-authoritative answer:
Name:   www.jairo.pro.br
Address: 187.73.33.34
```

Neste exemplo, o serviço DNS que resolveu o endereço IP foi o server em 186.251.39.194. Desse modo, foi obtido o endereço IP de **www.jairo.pro.br**, que é 187.73.33.34.

E a aplicação cliente *nslookup* soube em qual serviço DNS fazer essa consulta pois está configurado o IP do servidor no arquivo /etc/resolv.conf:

```
shell# more  /etc/resolv.conf
nameserver  186.251.39.194
nameserver  186.251.39.195
```

# 11.6 – netstat

O comando *netstat* mostra conexões de rede, tabelas de roteamento e estatísticas de interfaces, entre outros.

```
Shell# netstat  -na  |  more
Conexões Internet Ativas (servidores e estabelecidas)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0  0.0.0.0:22            0.0.0.0:*               OUÇA
tcp        0      0  127.0.0.1:631         0.0.0.0:*               OUÇA
tcp        0      0  192.168.1.10:47311    64.233.163.19:80        ESTABELECIDA
tcp        0      0  192.168.1.10:47312    64.233.163.19:80        ESTABELECIDA
tcp        0      0  ::1:631               :::*                    OUÇA
udp        0      0  0.0.0.0:47061         0.0.0.0:*
udp        0      0  0.0.0.0:5353          0.0.0.0:*
udp        0      0  0.0.0.0:631           0.0.0.0:*
udp        0      0  0.0.0.0:68            0.0.0.0:*
Domain sockets UNIX ativos (servidores e estabelecidas)
--Mais--
```

Nesse comando, foi usado o "| more" pois a saída é muito longa.

O comando *netstat -nr* também mostra a tabela de rotas, idêntico ao que faz o comando "*route -n*":

```
shell# netstat -nr

Tabela de Roteamento IP do Kernel
Destino         Roteador        MáscaraGen.     Opções Métrica Ref  Uso Iface
192.168.1.0     0.0.0.0         255.255.255.0   U      1       0    0   eth0
0.0.0.0         192.168.1.1     0.0.0.0         UG     0       0    0   eth0
```

Um outro comando usado é o *traceroute*, que serve para determinar os saltos entre equipamentos para um pacote de dados ir de uma origem até um destino.

# 12 – Aplicações clientes de serviços

O acesso a serviços é feito com aplicações clientes. Por exemplo, para acessar um serviço web, usa-se um navegador da internet.

Este capítulo apresenta exemplos de uso das aplicações clientes wget, telnet, ftp, ssh, sftp e scp.

## 12.1 – wget

O cliente *wget* acessa o serviço web em linha de comando, e faz *dowload* de arquivos.

Por exemplo, para acessar o serviço web no endereço 192.168.1.10, o comando é:

```
shell# cd /tmp
shell# wget 192.168.1.10
--2013-06-27 20:29:57-- http://192.168.1.10/
Conectando-se a 192.168.1.10:80... conectado.
A requisição HTTP foi enviada, aguardando resposta... 200 OK
Tamanho: 726 [text/html]
Salvando em: “index.html”

100%[======================================>] 726         --.-K/s   em 0s

2013-06-27 20:29:57 (91,9 MB/s) - “index.html” salvo [726/726]
```

No caso, antes de dar o comando *wget* foi usado o comando *cd*, isto para baixar o arquivo index.html de 192.168.10 para o diretório local /tmp.

Caso haja um proxy no meio do caminho, que é o caso de acesso à internet a partir dos laboratórios acadêmicos da Uninove, precisa antes passar a instrução de usuário e senha para o wget, e isso é feito com o comando export:

```
shell# export http_proxy=http://RA:SENHA@186.251.39.196:3128
```

Onde:

RA: é o RA do aluno;

SENHA: é a senha de acesso do aluno;
186.251.39.196: é o IP do serviço proxy, que atende na porta **3128** (é um Squid).

# 12.2 – telnet

O comando *telnet* é usado para fazer um acesso a um serviço TELNET (Telecommunications Network Protocol), que basicamente oferece um acesso shell remoto, em linha de comando.

```
shell# telnet  192.168.1.10
Trying 192.168.1.10...
Connected to 192.168.1.10.
Escape character is '^]'.
Ubuntu 9.04
Lab10 login: aluno
Password:
Last login: Sun Sep 13 11:36:57 from 192.168.1.13
[aluno@192.168.1.10]$
```

# 12.3 – ftp

A aplicação cliente *ftp* é usada para transferir arquivos, com o uso de um serviço FTP (File Transfer Protocol).

```
shell# ftp  192.168.1.10
Connected to 192.168.1.10.
220 Lab10 FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.
Name (Lab10:aluno): aluno
331 Password required for aluno
Password:
230 User aluno logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Uma vez conectado no serviço FTP, usam-se comandos para transferir arquivos. Essa

transferência pode tanto ser enviar arquivos do cliente para o servidor quanto vice-versa.

# 12.4 – ssh

A aplicação cliente *ssh* também serve para fazer acesso shell remoto em linha de comando, semelhante ao telnet. A diferença é que ssh é cliente do serviço SSH (Secure Shell), que implementa criptografia de dados e portanto é um serviço seguro.

```
shell# ssh  aluno@192.168.1.10

The authenticity of host 'localhost (192.168.1.10)' can't be established.
RSA key fingerprint is 66:39:9d:7a:8f:4c:af:1b:40:c4:a7:35:42:15:3b:d6.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
aluno@192.168.1.10's password:
Last login: Sun Sep 13 11:37:30 2009 from 192.168.1.13
[aluno@192.168.1.10]$
```

# 12.5 – sftp

A aplicação cliente *sftp* é usada para transferir arquivos, portanto semelhante ao cliente ftp. A diferença é que *sftp* é cliente do serviço SSH, portanto implementa criptografia de dados.

```
shell# sftp  aluno@192.168.1.10
aluno@192.168.1.10's password:
sftp> ?
Available commands:
cd path                          Change remote directory to 'path'
lcd path                         Change local directory to 'path'
chgrp grp path                   Change group of file 'path' to 'grp'
chmod mode path                  Change permissions of file 'path' to 'mode'
chown own path                   Change owner of file 'path' to 'own'
df [path]                        Display statistics for current directory or filesystem containing 'path'
help                             Display this help text
get remote-path [local-path]     Download file
lls [ls-options [path]]          Display local directory listing
ln oldpath newpath               Symlink remote file
lmkdir path                      Create local directory
lpwd                             Print local working directory
ls [path]                        Display remote directory listing
lumask umask                     Set local umask to 'umask'
mkdir path                       Create remote directory
progress                         Toggle display of progress meter
put local-path [remote-path]     Upload file
pwd                              Display remote working directory
exit                             Quit sftp
quit                             Quit sftp
rename oldpath newpath           Rename remote file
rmdir path                       Remove remote directory
rm path                          Delete remote file
symlink oldpath newpath          Symlink remote file
version                          Show SFTP version
!command                         Execute 'command' in local shell
!                                Escape to local shell
?                                Synonym for help
sftp>
```

# 12.6 – scp

A aplicação cliente *scp* (**s**ecure **co**p**y**) é usada para transferir arquivos usando o protocolo SSH. Essa aplicação tem sintaxe de uso semelhante ao comando *cp*.

Por exemplo, para transferir o arquivo **/etc/passwd** do host remoto 192.168.1.10 para o diretório **/tmp** local (download do arquivo), comandar:

```
shell# scp  aluno@192.168.1.10:/etc/passwd  /tmp
aluno@192.168.1.10's password:
passwd                                        100% 2023     2.0KB/s   00:00
shell#
```

Para enviar o arquivo local **/etc/group** para o servidor remoto (upload), comandar:

```
shell# scp  /etc/group  aluno@192.168.1.10:/tmp
aluno@192.168.1.10's password:
group                                         100%  875     0.9KB/s   00:00
shell#
```

# 13 – Serviços em Unix

## 13.1 – Processo daemon

O processo de serviço rodando num sistema operacional membro da família Unix é chamado de **daemon**, de **d**isk **a**nd **e**xecution **mon**itor. É devido a isso que, de um modo geral, os processos de serviço tem nomes que terminam com a letra **d**. Por exemplo, httpd [serviço web], sshd [serviço ssh] e named [serviço de nomes, DNS].

Tipicamente, do ponto de vista do sistema operacional, um serviço é constituído de pelo menos três partes:

- o executável [que ao rodar dá origem ao processo daemon];
- o arquivo de configuração;
- o script de inicialização do serviço.

Um serviço pode ser local ou em rede. Se for local, somente atende localmente ao host, se for em rede atende aos clientes em rede. Como exemplo de serviço local temos o cron [abaixo], como exemplo de serviço em rede temos o serviço web [httpd].

## 13.2 – Serviço Cron

O serviço cron é um agendador de tarefas. Ou seja, esse serviço é usado para disparar executáveis em datas previamente agendadas, a partir de configuração de uma tabela chamada **crontab**. Essa tabela especifica o que deve ser executado e em qual mês, semana, dia, hora e minuto.

No diretório **/etc/init.d** ficam os scripts de inicialização de serviços. Por exemplo:

```
shell$ /etc/init.d
```

| | | | | | |
|---|---|---|---|---|---|
| *acpid* | *cpuspeed* | *firstboot* | *iptables* | *nfs* | *proftpd* |
| *atd* | ***cron*** | *named* | *nfslock* | *psacct* | *xinetd* |
| *auditd* | *cups* | *gpm* | *nscd* | *pure-ftpd* | *rsyslog* |
| *cups-config* | *killall* | *netfs* | *ntpd* | *rdisc* | *sshd* |
| *bluetooth* | *dc_client* | *halt* | *netplugd* | *ntpdate* | *sendmail* |
| *btseed* | *dc_server* | *httpd* | *network* | *pcscd* | *rpcbind* |
| *bttrack* | *dnsmasq* | *ip6tables* | *rpcgssd* | *smolt* | *wine* |

O arquivo **/etc/init.d/cron** é o script de inicialização do serviço cron:

```
shell$ file /etc/init.d/cron
/etc/init.d/crond: POSIX shell script text executable
```

O executável fica em /usr/sbin/crond:

```
shell$ file /usr/sbin/crond
/usr/sbin/crond: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.18, stripped
```

Este executável, ao ser disparado, dá origem ao processo **daemon**. Para saber se está rodando, usar o comando **ps**:

```
shell$ ps -ef | grep cron
root     1454     1  0 14:25 ?        00:00:00 cron
aluno    3661  2180  0 22:05 pts/4    00:00:00 grep cron
```

O arquivo de configuração está no **/etc**:

```
shell$ ls /etc | grep cron
cron.d
cron.daily
cron.deny
cron.hourly
cron.monthly
crontab
cron.weekly
```

Para parar/iniciar o serviço, usar o script de inicialização:

```
shell# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
shell# /etc/init.d/cron  stop
Parando o cron:                            [  OK  ]
shell# /etc/init.d/cron  start
Iniciando o cron:                          [  OK  ]
```

O comando **crontab** é usado tanto para listar o conteúdo da tabela cron [caso exista], quanto incluir uma nova entrada ou modificar uma entrada existente.

```
shell$ id
uid=1000(aluno) gid=1000(users) grupos=1000(users)
shell$ crontab  -l
no crontab for aluno
```

Onde a opção **-l** é para listar o conteúdo da crontab.

No caso, o usuário aluno não tem nenhuma tarefa agendada na crontab.

Nesse exemplo, aluno irá incluir uma entrada para disparar o comando **/bin/date** a cada 2 minutos.

Antes, deve alterar a variável de ambiente **EDITOR** para usar o editor **vi** nessa edição:

```
shell$ export  EDITOR=vi
shell$ echo  $EDITOR
vi
```

Agora, editar a crontab:

```
shell$ crontab  -e
```

onde a opção **-e** é para editar a crontab

Nesse caso, foi incluída a entrada:

```
=====================================================
*/2    *    *    *    *    /bin/date >> /tmp/date.txt
=====================================================
```

Os cinco campos de agendamento de horário, pela ordem, são:

1º - minutos [0-59];
2º - horas [0-23]
3º - dia do mês [1-31]
4º - mês [1-12]
5º - dia da semana [0-6, onde 0 é domingo e 6 sábado]

Se algum campo permanecer com o símbolo * [asterisco], será executado todas as vezes. Por exemplo, se for no campo da horas [2º], vai executar a toda hora.

Exemplos de crontab:

| | | | | | |
|---|---|---|---|---|---|
| **0** | * | * | * | * | => executa a toda hora cheia; |
| **15** | **2** | * | * | * | => diariamente, às 2:15; |
| **20** | **11** | **10** | * | * | => dia 10 de todo mês, às 11:20; |
| **30** | **23** | * | **6** | * | => apenas no mês de junho, todos os dias às 23:30; |
| **25** | **4** | * | * | **0** | => apenas aos domingos, às 4:25. |

# 13.3 – Serviço Samba

O Samba é composto de duas partes: cliente e servidor, e foram desenvolvidas na intenção de garantir a comunicação e compartilhamento de recursos em rede entre sistemas das famílias Windows e Unix.

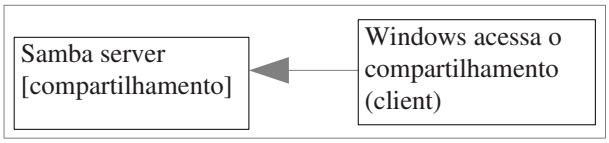
Se a rede for composta apenas por membros da família Windows (ou somente Unix), não existe problemas nessa comunicação, pois cada família tem nativamente seus próprios protocolos. Por exemplo, no Windows tem SMB/CIFS e no Unix tem NFS[19], entre outros.

Mas para resolver o problema de redes mistas Windows x Unix se faz necessário o Samba. Por exemplo, uma máquina Windows, nativamente, não tem porquê implementar também o suporte a sistema de arquivos NFS. Então a solução para esse problema, por mais paradoxal que possa parecer, é implementar no Unix um serviço e uma aplicação cliente que usem os mesmos protocolos SMB/CIFS e, dessa forma, garantir a comunicação em rede entre famílias Windows e Unix. Isto soa como o Unix "se disfarçar" de Windows para conseguir comunicação.

19 NFS: Network File System.

E com esse "disfarce", agora o cliente Windows acredita que o servidor Samba é outra máquina Windows de onde possa buscar recursos compartilhados em rede.

### 13.3.1 – Samba server




O serviço Samba pode ser disponibilizado em qualquer Unix e, uma vez configurado, permite ao cliente Windows buscar recursos neste Unix, sendo que para o Windows fica a impressão de que esse recurso está sendo disponibilizado numa máquina Windows.

Como o serviço Samba é usado basicamente para comunicação em rede entre Windows e Unix, para podermos configurar apropriadamente o serviço Samba precisamos primeiramente ter em vista o que seja a organização da rede Windows.

Quanto à organização, a rede Windows pode ser dividida basicamente em duas: *workgroup* e *domain*.

No caso do *workgroup*, trata-se de uma organização lógica apenas de visualização e agrupamento de computadores que vem desde o Windows 3.11.

A principal característica do *workgroup* é que cada máquina é uma unidade administrativa com suas próprias contas de usuários, senhas, etc, e por isso essa organização é um esquema ineficiente do ponto de vista administrativo para redes médias ou grandes.

Já o *domain* é um agrupamento lógico de servidores e estações de trabalho que compartilham informações comuns de segurança, contas de usuários e senhas. Dentro do domínio[20], o administrador cria uma conta de acesso para o usuário que poderá então efetuar "logon" a partir de qualquer estação que participe desse domínio.

A organização da rede Windows do qual o Samba participe tem impacto na autenticação do usuário, onde hora é local [workgroup], hora é no AD [domain].

Uma vez definido em qual organização de rede Windows irá trabalhar o serviço Samba, podemos passar à configuração.

Para descobrir se o serviço Samba está instalado, procurar pelo seu script de inicialização em **/etc/init.d**:

20 Desde o Windows 2000, o controlador de domínio é o AD [Active Directory].

```
shell# ls /etc/init.d | grep smb
/etc/init.d/smb
```

Se não houver saída no comando acima é indicativo de que o serviço **Samba** não está instalado. Nesse caso, instalar com o comando yum:

```
shell# yum install samba samba-client
```

Se fosse uma distribuição Linux baseada no Debian [por exemplo, Ubuntu], o comando para instalar seria apt-get:

```
shell# apt-get install samba samba-common smbclient
```

Uma vez instalado o serviço Samba, vamos criar um compartilhamento chamado TEMP, que compartilha o diretório local **/tmp**. Para isso, é necessário configurar o arquivo **/etc/samba/smb.conf** e incluir em *Share Definitions* o seguinte conteúdo:

========== arquivo **/etc/samba/smb.conf** ==========
[TEMP]
comment = "Compartilhamento do /tmp"
read only = no
browseable = yes
path = /tmp
==========

onde:

- **TEMP** é o nome do compartilhamento;
- **comment** é um comentário [é a mensagem que será vista na barra de título da janela no Windows];
- **read only = no** é para permitir escrita no /tmp;
- **path** é o caminho para o diretório físico que está sendo compartilhado no Unix;

Agora, então, iniciar o serviço Samba:

```
shell# /etc/init.d/smb start
Iniciando o samba:                         [  OK  ]
```

Depois de iniciar o serviço, verificar com o comando *ps* se o processo daemon está

rodando:

```
shell# ps -ef | grep smb
root      4568   1    0 16:02 ?     00:00:00 /usr/sbin/smbd -D
root      4570   4568 0 16:02 ?     00:00:00 /usr/sbin/smbd -D
```

Para checar as configurações do Samba server, usar o comando *testparm*:

```
shell# testparm
Load smb config files from /etc/samba/smb.conf
rlimit_max: rlimit_max (1024) below minimum Windows limit (16384)
Processing section "[homes]"
Processing section "[printers]"
Processing section "[TEMP]"
Loaded services file OK.
Server role: ROLE_STANDALONE
Press enter to see a dump of your service definitions

...
```

Precisa adicionar um usuário com senha ao serviço Samba. Para isso, usar o comando *smbpasswd*:

```
shell# smbpasswd -a aluno
New SMB password:
Retype new SMB password:
Added user aluno.
```

onde a opção "**-a**" é para adicionar um usuário que ainda não existe no serviço Samba. Se o usuário já existisse, para alterar a senha seria usado o mesmo comando, porém sem essa opção.

NOTA:
No caso de usar workgroup, verificar se o usuário existe localmente ao Windows, pois se não existir vai falhar o acesso.

Para testar o acesso, usar o Windows: ir para *Iniciar* -> *Executar* e comandar:

**\\192.168.1.10\TEMP**

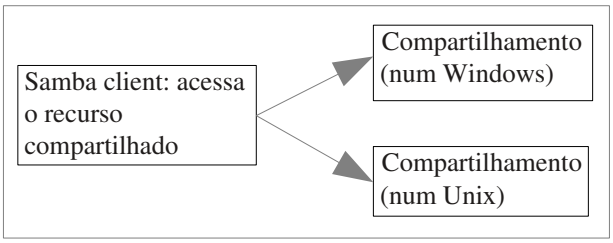
onde 192.168.1.10 é o IP do Samba server.

O Windows deverá acessar diretamente o compartilhamento, numa nova janela.

Nessa janela, copie algum conteúdo para esse compartilhamento, deposi volte para o Samba server e observe que esse conteúdo foi escrito no diretório **/tmp**.

### 13.3.2 – Samba client

O Samba client é uma aplicação para acessar, por exemplo, o recurso compartilhado pelo Windows. Também pode ser usado para acessar o recurso compartilhado pelo Samba server, num Unix.




Para testar o acesso com o Samba client, aproveitar o compartilhamento TEMP acima e usar o comando *smbclient*:

```
shell# smbclient  //192.168.1.10/TEMP  -U  aluno
Enter aluno's password:
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 3.4.7-0.50]
smb: \>
```

onde a opção "**-U**" é para informar que o usuário que vai autenticar no serviço Samba é aluno e não root, pois o shell é de root.

Convém notar que o prompt da aplicação *smbclient* é semelhante a do client FTP.

Para saber quais comandos podem ser passados pelo *smbclient*, comandar:

```
smb: \> ?
?              allinfo         altname         archive         blocksize
cancel         case_sensitive  cd              chmod           chown
close          del             dir             du              echo
exit           get             getfacl         hardlink        help
history        iosize          lcd             link            lock
lowercase      ls              l               mask            md
mget           mkdir           more            mput            newer
open           posix           posix_encrypt   posix_open      posix_mkdir
posix_rmdir    posix_unlink    print           prompt          put
pwd            q               queue           quit            rd
recurse        reget           rename          reput           rm
rmdir          showacls        setmode         stat            symlink
tar            tarmode         translate       unlock          volume
vuid           wdel            logon           listconnect     showconnect
..             !
smb: \>
```

NOTA:

Se estivéssemos usando organização da rede domain e não autenticação local [workgroup], o comando seria **smbclient //192.168.1.10/TEMP -U domain\\aluno**, onde o nome desse domínio é "domain".

Um outro cliente do serviço Samba é o *smbmount*, que permite montar num ponto de montagem (diretório vazio) o conteúdo compartilhado pelo Windows ou Samba server.

Um exemplo seria:

```
shell# smbmount //192.168.1.10/TEMP /mnt/samba -o username=aluno,password=uninove
```

onde 192.168.1.10 é o IP do Samba server, TEMP é o nome do compartilhamento e /mnt/samba é o ponto de montagem, no cliente.

Neste exemplo, a opção "**-o**" foi usada para passar um conjunto de opções, no caso usuário e senha. Se não tivessem sido passados usuário e senha, o serviço iria perguntar por eles. Repare que a construção acima pode muito bem ser usada num script para fazer essa montagem automaticamente, inclusive transparente para o usuário.

NOTA:

Num Linux Ubuntu, antes precisa instalar o pacote **smbfs**:

```
shell# apt-get  install  smbfs
```

Uma variação ao comando *smbmount* é usar diretamente o comando *mount*:

```
shell# mount  -t cifs  //192.168.1.10/TEMP  /mnt/samba  -o
username=aluno,password=uninove
```

onde a opção "**-t**" é para o comando *mount* montar como sistema de arquivo CIFS.

Para checar a montagem, usar o comando *df*. Para desmontar, usar *umount*.

# 14 – Gerenciamento de pacotes

## 14.1 – Gerenciamento de pacotes no Linux CentOS

Nas distribuições Linux, pacote é um arquivo contendo um conjunto de arquivos, que podem ser executáveis, configurações, documentos, bibliotecas, etc. A função do pacote é garantir uma maneira prática de instalar, remover e atualizar o sistema operacional e suas aplicações.

Usualmente quem cria e mantém esses pacotes é a própria distribuição Linux. Normalmente, a mídia de instalação do Linux contém pacotes a serem instalados.

NOTA: para instalar pacotes precisa ser root. O jeito simples de adquirir poderes de root é com o comando *sudo*. Por exemplo:

```
shell$ id
uid=1000(aluno) gid=1000(users) grupos=1000(users)
shell$ sudo su –
[sudo] password for aluno:
shell# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
```

Após enviar a senha de aluno, o usuário ganha o shell de root, como pode ser verificado com o símbolo do shell que agora é uma cerquilha "#".

CentOS significa *Community ENTerprise Operating System*. Na prática, a distribuição CentOS se alimenta do código fonte disponibilizado pela Red Hat: toda vez que a Red Hat lança uma nova versão, o seu código fonte é "reempacotado" com pequenas modificações para criar uma nova versão do CentOS. A lógica por trás dessa cópia é que o CentOS permite instalação e atualização do sistema e aplicações mesmo sem pagamento de licença de uso.

De um modo geral, os pacotes de instalação e atualização do Red Hat são os mesmos do CentOS, propositalmente mantidos compatíveis.

Os comandos para gerenciar pacotes no Red Hat são **rpm** e **yum**.

Como exemplo de utilização dos comandos *rpm* e *yum* será utilizado o pacote rpm

*nmap-5.51-1.i386.rpm*. Repare que o pacote é um arquivo.

### 14.1.1 - rpm

Para checar se o pacote já está instalado, comandar:

```
shell$ rpm -aq | grep nmap
```

se houvesse algum pacote instalado que contivesse a palavra *nmap*, seria mostrado.

Depois de baixar o arquivo *nmap-5.51-1.i386.rpm*, para saber quais arquivos esse pacote contém, comandar:

```
shell$ rpm -qlp nmap-5.51-1.i386.rpm
/usr/bin/ndiff
/usr/bin/nmap
/usr/share/doc/nmap-5.51
/usr/share/doc/nmap-5.51/COPYING
/usr/share/doc/nmap-5.51/README
/usr/share/doc/nmap-5.51/nmap.usage.txt
/usr/share/man/de/man1/nmap.1.gz
...
```

Para instalar um pacote rpm, comandar:

```
root# rpm -ivh nmap-5.51-1.i386.rpm
Preparando              ########################################### [100%]
   1:nmap               ########################################### [100%]
```

Para excluir um pacote rpm que já esteja instalado, comandar:

```
root# rpm -e nmap-5.51-1.i386
```

para atualizar um pacote que já esteja instalado, comandar:

```
root# rpm  -Uvh  nmap-5.51-1.i386
```

### 14.1.2 - yum

O *yum* faz o mesmo gerenciamento de pacotes que o comando *rpm*, o diferencial é que o *yum* busca pelo[s] pacote[s] ncessário[s] para a instalação em algum repositório de pacotes na internet. Mais interessante, alguns pacotes para serem instalados necessitam de pré-requisitos, e nesse caso o próprio *yum* faz os downloads e instalações sozinho. Se fosse o comando *rpm*, ele pararia a instalação e diria que está faltando algum pacote que é pré requisito para aquela instalação.

Para instalar algum pacote com o yum, comandar:

```
root# yum  install  nmap
Plugins carregados: refresh-packagekit
---> Pacote  nmap-5.51-1.i386.rpm  definido para ser atualizado
Tamanho total do download: 2.4 M
Correto? [s/N]:s
Baixando pacotes:
nmap-5.51-1.i386.rpm                                    | 2.4 MB     00:20
Executando o rpm_check_debug
  Instalando     : 2: nmap-5.51-1.i386.rpm                                  1/1

Instalados:
  nmap-5.51-1.i386.rpm
```

O *yum* também poderia ser usado para atualizar um pacote. Por exemplo:

```
root# yum  update  nmap
```

Mas para que o *yum* funcione a contento, precisa ser definido quais são os endereços de repositórios de pacotes na internet, que serão usados para download. Essa configuração está no arquivo "**/etc/yum.conf**". Normalmente, esse arquivo não precisa ser configurado, pois se for necessário adicionar um novo site repositório basta incluir um arquivo de configuração adicional no diretório "**/etc/yum.repos.d**".

Por exemplo, o arquivo "**/etc/yum.repos.d/CentOS-Base.repo**" contém, entre outras coisas, o seguinte conteúdo:

```
----- CentOS-Base.repo ------------------------------------------------------------------------
...
[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os
#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/
gpgcheck=1
gpgkey=http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-5
protect=1

#released updates
[updates]
name=CentOS-$releasever - Updates
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=updates
#baseurl=http://mirror.centos.org/centos/$releasever/updates/$basearch/
gpgcheck=1
gpgkey=http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-5
protect=1
...
-----------------------------------------------------------------------------------------------------
```

# 14.2 – Gerenciamento de pacotes no Linux Ubuntu

A distribuição Ubuntu é derivada do Debian, portanto o gerenciamento de pacotes, para instalação, é feito com o comando **apt-get**.

O comando **apt-get** lê o arquivo de configuração **/etc/apt/sources.list** para determinar os locais com repositório de pacotes para instalação.

Normalmente, a configuração em **sources.list** está correta, porém com frequência dá erro na instalação pois aparece na primeira linha uma configuração do tipo:

*deb cdrom:[Debian GNU/Linux 5.0.1 _Lenny_ - Official i386 DVD Binary-1 20090413-00:33]/ lenny contrib main*

indicando que **apt-get** está configurado para buscar pacotes na mídia de cd-rom, mas que não se encontra no drive no momento.

Se este for o caso, a solução então é editar o arquivo **sources.list** e comentar essa linha [colocar uma cerquilha "#" na frente da linha]:

*# deb cdrom:[Debian GNU/Linux 5.0.1 _Lenny_ - Official i386 DVD Binary-1 20090413-00:33]/ lenny contrib main*

Depois disso é necessário atualizar o database de repositórios com o comando:

```
shell# apt-get update
```

Agora, o comando para instalação de pacotes deverá funcionar.

Por exemplo, para instalar o pacote **nmap**, comandar:

```
shell# apt-get install nmap
```

O apt-get também pode ser usado para atualizar todo o sistema, nesse caso deve ser usada a opção *upgrade*:

```
shell# apt-get update
shell# apt-get upgrade
```

O *apt-get update* atualiza a lista de pacotes disponíveis, e *apt-get upgrade* atualiza todo o sistema.

O comando **apt-get** normalmente instala pacotes disponibilizados em repositórios na internet, por isso é muito prático.

Porém, pode ser o caso de não haver acesso à internet, então a aplicação poderia ser

instalada direta de um pacote "**.deb**" no CD-ROM.

Para usar o CD-ROM [ou DVD] como local de repositório de pacote, usar o comando apt-cdrom para declaração o CD-ROM como repositório:

```
shell# apt-cdrom add
```

Também poderia ser o caso de querer instalar a aplicação diretamente de um arquivo "**.deb**", nesse caso pode ser usado o comando **dpkg**:

```
shell# dpkg -i nmap-4.76.deb
```

onde **-i** é a opção para instalar a aplicação nmap do arquivo local nmap-4.76.deb. Para remover usar a opção **-r**.

Para saber se determinada aplicação está disponível para instalação via apt, usar o comando **apt-cache** para descobrir:

```
shell# apt-cache search nmap
```

que vai mostrar pacotes disponíveis que contêm o nome nmap.

O comando **apt-key** serve para gerenciar uma lista de chaves usadas pelo apt para autenticar pacotes. Pacotes autenticados por essas chaves são considerados confiáveis.

Existe também o **aptitude**, que é uma aplicação para instalar e remover pacotes.

# 14.3 – Gerenciamento de pacotes em sistemas em geral

O gerenciamento de pacotes usa diferentes comandos para os diferentes sistemas operacionais da família Unix.

No Solaris usa-se **pkgadd** e **pkgrm** para adicionar e remover pacotes. Para obter informações sobre pacotes instalados, é usado o comando **pkginfo**.

No AIX também tem o gerenciador de pacotes **rpm**, igual ao do Linux. Mas nesse sistema

é bastante comum usar o comando **smitty** [ou **smit**], que é a ferramenta padrão de administração/configuração do sistema.

Por exemplo, para instalar, entrar no diretório onde está o pacote e comandar:

```
shell# smitty install
```

que o comando vai oferecer uma seleção de *file sets* para instalar.

# 15 - Anexos

## 15.1 - EUID/EGID

A partir de um programa compilado em C, pode-se demonstrar o conceito de EUID e EGID.

O código do programa **uid-euid.c** é:

```
===uid-euid.c=============================================================
#include <stdio.h>
#include <unistd.h>

int main (void) {
        printf("uid: %d\neuid: %d\ngid: %d\negid: %d\n", getuid(), geteuid(), getgid(), getegid());
}
==========================================================================
```

A seguir, o usuário aluno compila no **/tmp** o programa uid-euid.c com o comando:

"**gcc -o uid-euid uid-euid.c**".

Isso vai gerar o executável **uid-euid**.

Na sequência, o usuário aluno, UID/GID 1000/1000, dá a permissão SUID com o comando:

**chmod 4755 /tmp/uid-euid**

Posteriormente, outro usuário (por exemplo juca, UID/GID 1001/1001), comanda o executável **/tmp/uid-euid**. O resultado no seu shell será:

```
----------------
uid: 1001
euid: 1000
gid: 1001
egid: 1001
----------------
```

O raciocínio feito para demonstrar o EUID é o mesmo para EGID. O usuário aluno comanda:

**chmod 2755 /tmp/uid-euid**

Após, o usuário juca comanda de novo **/tmp/uid-euid** e recebe como saída:

```
----------------
uid: 1001
euid: 1001
gid: 1001
egid: 1000
----------------
```

Agora o usuário aluno comanda:

**chmod 6755 /tmp/uid-euid**

O usuário juca agora recebe como saída:

```
----------------
uid: 1001
euid: 1000
gid: 1001
egid: 1000
----------------
```