# SISTEMAS EMBARCADOS

# Introdução ao Desenvolvimento com Linux

Autores:
**Edson Mintsu Hung**
**Evandro Leonardo Teixeira**
**Tiago Alves da Fonseca**

**Universidade de Brasília**

**Faculdade do Gama**

**Curso de Engenharia Eletrônica**

# SISTEMAS EMBARCADOS

***Conteúdo:***

1. Metodologia/Filosofia de desenvolvimento com o Linux/UNIX
2. Licenças de software livre (GPL e LGPL)
3. Comandos básicos do Linux
4. Organização da estrutura de diretórios
5. Obtendo informações sobre o sistema
6. Instalação de programas
7. Ferramentas de desenvolvimento em linguagem C
8. Bibliografia

# 1. Metodologia/Filosofia de desenvolvimento com Linux/UNIX

Apesar da programação em C ser bastante parecida independentemente das plataformas, é fato que os desenvolvedores Linux/UNIX possuem uma visão especial de programação e desenvolvimento de sistemas.

O sistema operacional UNIX encoraja um certo tipo de estilo de programação. Segue algumas características herdadas por programas e sistemas UNIX típicos. [1]

## Simplicidade

Muitas das mais úteis ferramentas Linux são muito simples e, como resultado, pequenos e fáceis de entender. KISS (Keep It Small ans Simple) é uma boa técnica a se aprender. Sistemas maiores e mais complexos são susceptíveis a bugs maiores e mais complexos e a depuração é uma tarefa que queremos evitar ao máximo.

## Foco

É muito interessante garantir que um programa execute uma tarefa com perfeição. Um programa resposável por uma gama de tarefas é difícil de usar e manter. Programas de propósito único são fáceis de melhorar à medida que melhores algoritmos e interfaces forem sendo desenvolvidos. No Linux, pequenas ferramentas são freqüentemente combinadas para realizar tarefas mais complexas quando as necessidades aparecem em vez de tentar antecipar as necessidades dos usuários em um programa grande e complexo.

## Componentes Reusáveis

Torne o núcleo da sua aplicação disponível como bibioteca. Bibliotecas bem documentadas com interfaces de programação simples mas flexíveis podem ajudar outros a desenvolver variações ou aplicar suas técnicas em novas áreas.

## Filtros

Muitas aplicações Linux podem ser usadas como filtros. Ou seja, elas tranformam a sua entrada e produzem saída. Como veremos, o Linux disponibiliza recursos que permitem que aplicações complexas sejam desenvolvidas por outros programas Linux combinando-os de maneiras novas e não usuais.

## Formatos de arquivos abertos

Os programas de sucesso do Linux usam arquivos de configuração e de dados que são codificados em texto ASCII puro. Se essa for uma opção para seu projeto, será uma escolha sensata acolhê-la. Isso permite aos usuários usar ferramentas padrões para mudar e buscar itens de configuração e desenvolver novas ferramentas que executem novas tarefas em arquivos de dados.

## Flexibilidade

Você não poderá antecipar exatamente como o seu usuário irá aplicar o seu programa. Tente ser o mais flexível o possível na sua programação. Tente evitar limites nos tamanhos dos campos ou no número de registros. Se você puder, escreva o programa para um ambiente em rede de forma que possa ser executado via rede bem como em uma máquina local. Nunca assuma que você sabe tudo que o usuário pretenderá fazer com o seu programa.

## 2. Licenças de software livre (LGPL e GPL)

Antes eventuais explicações sobre o que é o Linux, é preciso saber o que o motivou. As subseções a seguir apresentarão o conceito de software livre, as suas licenças e como o Linux se encaixa nesse panorama.

### O que é software livre?

Tirado de http://www.gnu.org/philosophy/free-sw.pt.html

Nós mantemos esta definição do Software Livre para mostrar claramente o que deve ser verdadeiro à respeito de um dado programa de software para que ele seja considerado software livre.

"Software Livre" é uma questão de liberdade, não de preço. Para entender o conceito, você deve pensar em "liberdade de expressão", não em "cerveja grátis".

"Software livre" se refere à liberdade dos usuários executarem, copiarem, distribuírem, estudarem, modificarem e aperfeiçoarem o software. Mais precisamente, ele se refere a quatro tipos de liberdade, para os usuários do software:

- A liberdade de executar o programa, para qualquer propósito (liberdade no. 0)
- A liberdade de estudar como o programa funciona, e adaptá-lo para as suas necessidades (liberdade no. 1). Acesso ao código-fonte é um pré-requisito para esta liberdade.
- A liberdade de redistribuir cópias de modo que você possa ajudar ao seu próximo (liberdade no. 2).
- A liberdade de aperfeiçoar o programa, e liberar os seus aperfeiçoamentos, de modo que toda a comunidade se beneficie (liberdade no. 3). Acesso ao código-fonte é um pré-requisito para esta liberdade.

Um programa é software livre se os usuários tem todas essas liberdades. Portanto, você deve ser livre para redistribuir cópias, seja com ou sem modificações, seja de graça ou cobrando uma taxa pela distribuição, para qualquer um em qualquer lugar. Ser livre para fazer essas coisas significa (entre outras coisas) que você não tem que pedir ou pagar pela permissão.

Você deve também ter a liberdade de fazer modificações e usá-las privativamente no seu trabalho ou lazer, sem nem mesmo mencionar que elas existem. Se você publicar as modificações, você não deve ser obrigado a avisar a ninguém em particular, ou de nenhum modo em especial.

A liberdade de utilizar um programa significa a liberdade para qualquer tipo de pessoa física ou jurídica utilizar o software em qualquer tipo de sistema computacional, para qualquer tipo de trabalho ou atividade, sem que seja necessário comunicar ao desenvolvedor ou a qualquer outra entidade em especial.

A liberdade de redistribuir cópias deve incluir formas binárias ou executáveis do programa, assim como o código-fonte, tanto para as versões originais quanto para as modificadas. Está ok se não for possível produzir uma forma binária ou executável (pois algumas linguagens de programação não suportam este recurso), mas deve ser concedida a liberdade de redistribuir essas formas caso seja desenvolvido um meio de cria-las.

De modo que a liberdade de fazer modificações, e de publicar versões aperfeiçoadas, tenha algum significado, deve-se ter acesso ao código-fonte do programa. Portanto, acesso ao código-fonte é uma condição necessária ao software livre.

Para que essas liberdades sejam reais, elas tem que ser irrevogáveis desde que você não faça nada errado; caso o desenvolvedor do software tenha o poder de revogar a licença, mesmo que você não tenha dado motivo, o software não é livre.

Entretanto, certos tipos de regras sobre a maneira de distribuir software livre são aceitáveis, quando elas não entram em conflito com as liberdades principais. Por exemplo, copyleft (apresentado de forma bem simples) é a regra de que, quando redistribuindo um programa, você não pode adicionar restrições para negar para outras pessoas as liberdades principais. Esta regra não entra em conflito com as liberdades; na verdade, ela as protege.

Portanto, você pode ter pago para receber cópias do software GNU, ou você pode ter obtido cópias sem nenhum custo. Mas independente de como você obteve a sua cópia, você sempre tem a liberdade de copiar e modificar o software, ou mesmo de vender cópias

"Software Livre" Não significa "não-comercial". Um programa livre deve estar disponível para uso comercial, desenvolvimento comercial, e distribuição comercial. O desenvolvimento comercial de software livre não é incomum; tais softwares livres comerciais são muito importantes.

Regras sobre como empacotar uma versão modificada são aceitáveis, se elas não acabam bloqueando a sua liberdade de liberar versões modificadas. Regras como "se você tornou o programa disponível deste modo, você também tem que torná-lo disponível deste outro modo" também podem ser aceitas, da mesma forma. (Note que tal regra ainda deixa para você a escolha de tornar o programa disponível ou não.) Também é aceitável uma licença que exija que, caso você tenha distribuído uma versão modificada e um desenvolvedor anterior peça por uma cópia dele, você deva enviar uma.

No projeto GNU, nós usamos *copyleft* para proteger estas liberdades legalmente para todos. Mas também existe software livre que não é *copyleft*. Nós acreditamos que hajam razões importantes pelas quais é melhor usar o *copyleft*, mas se o seu programa é free-software mas não é copyleft, nós ainda podemos utilizá-lo.

Às vezes regras de controle de exportação e sansões de comércio podem limitar a sua liberdade de distribuir cópias de programas internacionalmente. Desenvolvedores de software não tem o poder para eliminar ou sobrepor estas restrições, mas o que eles podem e devem fazer é se recusar a impô-las como condições para o uso dos seus programas. Deste modo, as restrições não afetam as atividades e as pessoas fora da jurisdição destes governos.

Quando falando sobre o software livre, é melhor evitar o uso de termos como "dado" ou "de graça", porque estes termos implicam que a questão é de preço, não de liberdade. Alguns temos comuns como "pirataria" englobam opiniões que nós esperamos você não irá endossar.

Finalmente, note que critérios como os estabelecidos nesta definição do software livre requerem cuidadosa deliberação quanto à sua interpretação. Para decidir se uma licença se qualifica como de software livre, nós a julgamos baseados nestes critérios para determinar se ela se segue o nosso espírito assim como as palavras exatas. Se uma licença inclui restrições impensadas, nós a rejeitamos, mesmo que nós não tenhamos antecipado a questão nestes critérios. Às vezes um requerimento de alguma licença levanta uma questão que requer excessiva deliberação, incluindo discussões com advogados, antes que nós possamos decidir se o requerimento é aceitável. Quando nós chegamos a uma conclusão sobre uma nova questão, nós frequentemente

atualizamos estes critérios para tornar mais fácil determinar porque certas licenças se qualificam ou não.

## O que é GPL?

**GNU General Public License** (Licença Pública Geral), **GNU GPL** ou simplesmente **GPL**, é a designação da licença para software livre idealizada por Richard Stallman no final da década de 1980, no âmbito do projeto GNU da Free Software Foundation.

A GPL é a licença com maior utilização por parte de projetos de software livre, em grande parte devido à sua adoção para o Linux.

Em termos gerais, a GPL baseia-se em 4 liberdades:

1. A liberdade de executar o programa, para qualquer propósito (liberdade nº 0)
2. A liberdade de estudar como o programa funciona e adaptá-lo para as suas necessidades (liberdade nº 1). O acesso ao código-fonte é um pré-requisito para esta liberdade.
3. A liberdade de redistribuir cópias de modo que você possa ajudar ao seu próximo (liberdade nº 2).
4. A liberdade de aperfeiçoar o programa, e liberar os seus aperfeiçoamentos, de modo que toda a comunidade beneficie deles (liberdade nº 3). O acesso ao código-fonte é um pré-requisito para esta liberdade.

Com a garantia destas liberdades, a GPL permite que os programas sejam distribuídos e reaproveitados, mantendo, porém, os direitos do autor por forma a não permitir que essa informação seja usada de uma maneira que limite as liberdades originais. A licença não permite, por exemplo, que o código seja apoderado por outra pessoa, ou que sejam impostos sobre ele restrições que impeçam que seja distribuído da mesma maneira que foi adquirido.

A GPL está redigida em inglês e atualmente nenhuma tradução é aceita como válida pela FSF, com o argumento de que há o risco de introdução de erros de tradução que poderiam deturpar o sentido da licença. Deste modo, qualquer tradução da GPL é não-oficial e meramente informativa, mantendo-se a obrigatoriedade de distribuir o texto oficial em inglês com os programas.

A licença GPL foi originalmente publicada em Janeiro de 1989. No entanto, passado pouco tempo, ficou claro que o texto da licença comportava vários problemas, pelo que em Junho de 1991 foi publicada a GPL versão 2, sendo ao mesmo tempo introduzida uma nova licença LGPL. Em 2005, Stallman anunciou que estava prepararando uma nova versão da licença em conjunto com Eben Moglen. Essa nova versão, foi chamada de GPLv3 e o primeiro esboço foi publicado em 16 de Janeiro de 2006.

## O que é LGPL?

A GNU Lesser General Public License (antes conhecida como GNU Library General Public License) é uma licença de software livre aprovada pela FSF escrita com o intuito de ser um meio-termo entre a GPL e licenças mais permissivas como a licença BSD e a licença MIT. Ela foi escrita em 1991 (e atualizada em 1999) por Richard Stallman e Eben Moglen.

A principal diferença entre a GPL e a LGPL é que esta permite ser ligada com programas que não sejam GPL ou LGPL, que podem ser software livre ou software proprietário, ou seja, permite o desenvolvimento de programas de código aberto que contenham módulos proprietários.

Na GPL "tradicional" todo o código do programa é aberto, isso atende bem à maioria dos projetos colaborativos. O problema é que muitas empresas possuem segredos a guardar, o que as impede de simplesmente abrir totalmente o código de seus programas.

A LGPL funciona melhor nestes casos, pois permite que você desenvolva programas "semi-abertos" onde parte do código do programa está disponível e o restante das funções é proporcionada por alguns binários que não possuem seu código aberto, estes binários linkados no programa no momento da compilação, de uma forma análoga às DLL's do Windows.

Imagine que você tenha uma empresa que desenvolve um processador de textos no estilo do Word. Depois de algumas noites sem dormir você desenvolve um módulo revolucionário que é capaz de abrir e salvar qualquer tipo de arquivo do Office sem falhas. Isso sem dúvida é um grande diferencial para o seu produto e por isso você resolve não abrir o código fonte, afinal você desenvolveu o software sozinho e pode decidir o que fazer com ele.

A LGPL funcionaria bem no seu caso, pois você poderia abrir o código do seu editor de textos para que outras pessoas possam contribuir com melhorias e correções, mas manter proprietário o módulo que lê arquivos do Office, o grande diferencial que faria as pessoas comprarem o editor ao invés de simplesmente instalar a partir do código fonte.

De fato, Stallman e a FSF às vezes defendem licenças até menos restritivas que a LGPL como questão de estratégia (para maximizar a liberdade de usuários). Um exemplo proeminente foi o endosso de Stallman do uso de uma licença BSD pelo projeto Vorbis para suas bibliotecas.

## O que é Linux?

O Linux é uma variante “free software” do Unix. Foi criado por um jovem universitário finlandês chamado Linus Torvalds, no inícios dos anos 1990, e de lá para cá tornou-se o segundo sistema operacional mais utilizado no mundo, atrás apenas do Microsoft Windows. É, hoje, o sistema operacional cuja base instalada mais cresce, principalmente no meio acadêmico e no ambiente corporativo, onde, devido a seu baixo custo (aliado ao alto desempenho e à confiabilidade típicos dos sistemas Unix-like), tem merecido cada vez mais a atenção dos departamentos de TI. [2]

O Linux está disponível sobre a forma de várias distribuições (Fedora, Mandriva, SUSE, Debian, Knoppix, Slackware etc,) cada qual com o seu próprio conjunto de utilitários.

## Onde temos Linux?

Bem, conscientemente ou não, você provavelmente esbarra com o Linux todos os dias. Quando você compra um livro na Amazon.com ou busca na Web com o Google, você usa o Linux. As animações vistas em Shrek 2 foram criadas por centenas de estações Linux e renderizadas por uma *server farm* de centenas de outros sistemas Linux.

## Entendendo o Linux

Linux, um sistema operacional completo, é uma espécie de clone livre do sistema operacional UNIX. Iniciar um computador com o Linux faz com que ele tome o controle da operação do PC e gerencie os seguintes recursos: [2]

- Processador: Pelo fato de o Linux poder rodar muitos processos de diferentes usuários ao mesmo tempo (até mesmo com múltiplas CPUs na mesma máquina), Linux tem que ser capaz de gerenciar esses processos. O *scheduler* do Linux ajusta as prioridades das tarefas e gerencia que processos rodam em quais CPUs (se múltiplos processadores estão presentes). O *scheduler* pode ser ajustado diferentemente para diferentes tipos de sistemas Linux. Quando ajustado devidamente, os processos mais importantes obtêm respostas mais rápidas do processador.
- Memória: O Linux tenta manter os processos com suas respectivas demandas de memória RAM sanadas e gerencia os processos que demandam muita memória sanando esse consumo excessivo por meio do uso do espaço de troca (*swap space*). O espaço de troca é uma área definida em seu disco rígido que é usada para gerenciar a demanda crescente de memória dos processos e dados. Quando a RAM está cheia, os processos são postos no espaço de troca. Quando o espaço de troca fica cheio (algo indesejável), novos processos não podem ser iniciados.
- Dispositivos: O Linux suporta milhares de dispositivos de *hardware* mas mantém o *kernel* em um tamanho gerenciável pela inclusão de um pequeno conjunto de drivers no *kernel* ativo. Usando módulos carregáveis, o *kernel* pode adicionar o suporte para outros *hardwares* sempre que necessário. Módulos podem ser inseridos e removidos sobre demanda, da mesma forma que os *hardware* são inseridos e removidos.
- Sistema de arquivos: Os sistemas de arquivos proporcionam a estrutura sobre a qual os arquivos são armazenados no disco rígido, CD, DVD, disco flexível ou outros meios de armazenamento. O Linux conhece diferentes tipos de sistemas de arquivos (como Linux ext3, reiserfs, VFAT e NTFS dos sistemas Windows) e sabe como gerenciá-los.
- Segurança: Como o UNIX, o Linux foi construído numa perspectiva de baixo para cima para permitir que múltiplos usuários acessem o sistema simultaneamente. Para proteger os recursos de cada usuário, a cada arquivo, diretório ou aplicação (programa) é associado um conjunto de permissões de leitura, escrita e execução que definem quem poderá acessá-lo. No sistema Linux padrão, o usuário root pode ter acesso ao sistema inteiro, alguns *logins* especiais têm acesso para controlar serviços particulares (como o Apache para serviços Web) e as permissões podem atingir usuários individualmente ou grupos.

O que foi descrito acima são componentes que constituem aquilo que se convencionou chamar de *kernel* do Linux. De fato, o *kernel* do Linux (que foi criado e ainda é mantido por Linus Torvalds) é o que dá ao Linux o seu nome. O *kernel* é o programa que inicia quando você liga o computador e gerencia os programas que você usa de maneira que eles possam se comunicar simples e eficientemente com o *hardware* do seu computador.

Outros componentes, como comandos administrativos e aplicações, são adicionados ao *kernel* a partir de outros projetos de *software* livre, tornando o Linux um sistema operacional completo. O projeto GNU, particularmente, contribuiu para muitos componentes que agora fazem parte do Linux. Esses projetos adicionaram outras características como:

- Interfaces gráficas com o usuário (GUIs): Consiste de um *framework* gráfico

(tipicamente o X Windows System), gerenciadores de janelas, painéis, ícones e menus. GUIs permitem que você use o Linux usando a combinação teclado e mouse em vez de simplesmente digitar comandos.

- Utilitários Administrativos: Incluem centenas (talvez milhares) de comandos e janelas gráficas que permitem executar tarefas como adicionar usuários, gerenciar discos, monitorar a rede, instalar *software* e tornar seguro e gerenciar o seu computador.
- Aplicações: Embora nenhuma distribuição Linux inclua todas as aplicações existentes, existem milhares de jogos, ferramentas de produtividade, navegadores da Web, bate-papo, reprodutores de mídia e outros programas disponíveis para Linux.
- Ferramentas de programação: Inclui ferramentas de programação para criar aplicações e bibliotecas para implementar interfaces especializadas.
- Servidores: Permite que você ofereça serviços a partir de seu computador Linux para outros computadores na mesma rede. Em outras palavras, enquanto o Linux inclui navegadores da Web para visualização de páginas, ele também pode ser o próprio computador que hospeda páginas da Internet. Servidores populares são o de Web, email, banco de dados, impressão, arquivo, DNS e DHCP.

## O que faz o Linux tão atrativo?

Se você nunca usou Linux anteriormente, é normal esperar que certas coisas sejam diferentes de outros sistemas operacionais. Abaixo segue uma breve lista de algumas características que fazem o Linux tão interessante.

- Não requer reinicialização do computador para instalações: O *uptime* (tempo de execução ininterrupto) é uma questão de orgulho e honra (lembre-se, Linux e outros sistemas UNIX são freqüentemente usados como servidores e espera-se que esses funcionem 24 horas por dia e sete dias por semana). Depois da instalação original, você pode instalar ou remover a maioria dos *software* sem precisar reiniciar seu computador.
- Inicializa e interrompe serviços sem interromper outros: Você pode inicializar e interromper serviços individualmente (como serviços de web, servidores de arquivo e de email) sem precisar reinicializar o computador ou mesmo interromper o trabalho de outros usuários ou desabilitar outras funcionalidades do computador. Em outras palavras, você não precisa reinicializar o computador cada vez que alguém “espirra”!
- Software Portável: Você pode mudar para outro Linux, UNIX ou BSD e usar o mesmo software. A maioria dos projetos de open source foram criados para rodar em qualquer sistema UNIX-like e muitos podem até mesmo rodar em sistemas Windows se necessário. E mesmo se isso não for possível para dado *software*, é provável que alguém da comunidade Linux esteja empenhado em fazê-lo.
- Download de programas: Se as aplicações que você precisa não foram entregues com a sua versão do Linux, você pode freqüentemente baixá-las e instalá-las em com um comando simples, usando ferramentas como yum e apt.
- Nenhuma configuração é oculta em códigos ou registros: Uma vez que o usuário fique familiarizado com o estilo do Linux, verá que (respeitando as permissões do computador) a maioria das configurações são feitas em arquivos de texto puro que são fáceis de localizar e modificar.

- Desktop maduro: O X Windows System (que proporciona o *framework* para sua estação de trabalho Linux) já existe há mais tempo que o Microsoft Windows. O ambientes de trabalho KDE e GNOME disponibilizam interfaces gráficas (janelas, menus, ícones, etc.) que competem com o Microsoft Windows. Os problemas de amigabilidade em relação ao usuário estão sendo sanados rapidamente.
- Liberdade: Linux, na sua forma mais básica, não possui uma agenda corporativa ou linha a seguir. Você é livre para escolher a distribuição de Linux mais conveniente, olhar o código fonte que compõe o sistema que está em execução, adicionar e remover qualquer *software* que você queira e fazer com que o computador aja da maneira que você deseja.

Algums aspectos do Linux podem parecer difíceis para novos usuários. Um aspecto é que ele foi feito para ser seguro por padrão, então faz-se necessário se acostumar a usar uma conta administrativa (**root**) para poder fazer as mudanças que afetam todo o sistema computacional. Isso pode parecer incoveniente numa primeira olhada, mas isso faz com que o computador seja mais seguro do que permitir que qualquer um faça o que quiser no mesmo.

## Encontrando informações no Linux

Cada nova distribuição do Linux vem com uma quantidade enorma de documentação. Você pode aprender tudo sobre o que será tratado nesse curso por meio da leitura da documentação que acompanha a sua distribuição Linux. A documentação nem sempre é bem organizada, portanto a parte mais complicada é encontrar o que você realmente precisa. Às vezes a documentação está desatualizada, portanto leia tudo com bastante cautela. Se o seu sistema se comporta de maneira diferente do que é mostrado nas man pages, pode ser que suas man pages estejam desatualizadas.

A maioria das distribuições do Linux incluem man pages para a maioria dos comandos padrões, chamadas de sistema e funções de bibliotecas padrão. As man pages são divididas em seções numeradas; para programadores, as mais importantes são:

(1) Comandos de usuário

(2) Chamadas de sistema

(3) Funções de bibliotecas padrão

(8) Comandos do sistema/administrativos

Os números indicam as seções das man pages. As man pages do Linux vem geralmente instaladas no sistema; use o comando man para acessá-las. Para olhar em uma página man, simplesmente chame man name, onde name é o nome do comando ou função. Em alguns casos, o mesmo nome ocorre em mais de uma seção; você pode especificar a seção pela indicação explícita do número da mesma antes de name. Por exemplo, se você digitar o seguinte, você terá a man page do comando sleep:

```
$ man sleep
```

Para ver o que a página da função de biblioteca mostra, use:

```
$ man 3 sleep
```

Cada man page inclui um sumário online do comando ou função. O comando whatis name mostra todas as man pages (de todas as seções) para um comando ou função com name. Se você não tiver certeza do comando ou função que você deseja, você pode fazer uma busca por palavra nas linha de sumário, usando man -k keyword.

Man pages condensam uma quantidade enorme de informações e deve ser o primeiro lugar a ser vasculhado na procura por informações. A man page para um comando descreve as opções do mesmo e seus argumentos, entradas e saídas, códigos de erro, configuração etc. A man page de uma chamada de sistema ou função de biblioteca descreve os parâmetros e valores de retorno, listas de códigos de erro, efeitos colaterais e especifica que include deve ser posto caso você deseje usar a função.

## 3. Comandos básicos do Linux

As interfaces gráficas de usuário, como o KDE e o Gnome, mesmo tendo evoluído muito nos últimos anos e contribuído inegavelmente para a popularização do Linux, não diminuíram a importância da linha de comando para os usuários deste sistema operacional. O Linux, como típico sistema Unix-like, oferece um vasto conjunto de comandos, conjunto que cresce a cada dia. Além disso, esse mesmo conjunto pode receber dinamicamente nossas próprias contribuições para a solução de problemas. [2]

As três principais razões para aprender como usar o shell são:

- Você saberá como usar qualquer Linux ou outro sistema operacional derivado do UNIX. Por exermplo, você pode logar em um servidor MySQL, um firewall ou roteador embarcado ou um iMAC e saber usar esses outros sistemas a partir do shell.
- Características fundamentais do shell permite com que você colha dados de entrada e direcione a saída dos dados entre os comandos do sistema Linux. Para economizar a digitação, você pode encontrar, editar e repetir os comandos do histórico do shell. Muitos usuários experientes prescindem da interface gráfica, fazendo a maioria das sua atividades pelo shell.
- Você pode condensar comandos em um arquivo usando programação estruturada como laços e estruturas de decisão para realizar operações complexas que exigiriam redigitações infindáveis. Programas que consistem de comandos que são armazenados e executados a partir de um arquivo são chamados de *shell* scripts. A maioria dos administradores de sistema Linux usam scripts para automatizar tarefas como backups, monitoramento dos arquivos de log ou checagem da sanidade do sistema.

O shell é um interpretador da linha de comando. Se você usou os sistemas operacionais da Microsoft, você pode ver que usar o shell no Linux é semelhante a – mas geralmente mais poderoso do que – o interpretador usado para rodar comandos no DOS. Você pode simplemente usar o Linux a partir da interface gráfica mas, à medida que você aprende o Linux, você certamente precisará usar o shell em algum momento para rastrear um problema. [2]

## Escolhendo o seu shell

Na maioria dos sistemas Linux, o shell padrão é o bash. Para descobrir qual o shell você está executando, simplesmente digite:

```
$echo $SHELL

/bin/bash
```

Nesse exemplo, 'shell é o bash. Há uma variedade de shells e você pode ativá-los digitando o nome do shell desejado na própria linha de comando (ksh, tcsh, csh, sh, bash etc.)

## Usando o bash

O bash inclui características que foram originalmente desenvolvidas pelos shells sh e ksh nos sistemas UNIX bem como algumas características do csh. O bash é considerado o shell padrão em qualquer sistema Linux que você esteja usando.

O bash pode rodar em vários modos de compatibilidade de maneira que ele se comporte como outro shell. Ele pode simular um shell Bourne (sh) ou um shell POSIX-compliant, por exemplo, permitindo com que o bash leia arquivos de configuração que são específicos ao esses shells e rode scripts desses mesmos shells com grande chances de sucesso.

## Explorando o shell

Uma vez que você tenha acesso ao shell, você pode começar digitando alguns comandos simples.

## Checando sua sessão de login

Quando você loga no Linux, o Linux vê você como se tivesse uma identidade particular, que inclui o seu nome de usuário, nome de grupo, ID de usuário e ID de grupo. Linux também rastreia a sua sessão de login. Ele sabe que logou, quanto tempo o usuário ficou inativo e de onde ele logou.

Para encontrar informação a respeito da sua identidade, use o comando id:

```
$id

uid=0(root)gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
```

Nesse, o nome do usuário root é representado por uma ID numérica de usuário (uid) 0. O grupo primário de root é chamado root, que tem uma gid (ID de grupo) de 0. Root também pertence a outros grupos chamados bin (gid 1), daemon (gid 3) etc. Esses nomes e números representam as permissões que root tem ao acessar os recursos de computador.

Você pode obter informações sobre a sessão atual de login por meio do comando who. No seguinte exemplo, a opção -u diz para apresentar mais informações sobre o tempo de inatividade e a ID do processo e -H solicita a impressão do header.

```
$who -uH

NAME    LINE        TIME        IDLE        PID COMMENT

root    :0          Jun  7 11:29   ?         4144

root    pts/0       Jun  7 11:29 04:05       4257
```

root pts/1 Jun 7 11:30 03:43 4393

root pts/2 Jun 7 14:17 . 26963

A saída do comando who mostra que o usuário root logou em pts/0 e sua sessão começou dia 7 de Junho, às 11:29.

## Checando diretórios e permissões

Associada à cada shell está a localização do sistema de arquivos Linux chamada de diretório atual. Cada usuário tem um diretório que é identificado como o diretório home do usuário. Quando você loga no Linux, você começa em seu diretório home como diretório atual.

Quando você pede para abrir ou salvar um arquivo, seu shell usa o diretório atual como o ponto de referência. Simplesmente apresente o nome do arquivo como quando você for salvá-lo e ele será posto no diretório atual. Alternativamente, você pode identificar um arquivo com relação ao diretório atual (caminho relativo) ou você pode ignorar o diretório atual e identificar o arquivo por um caminho absoluto que indique toda a hierarquia de diretórios que o localiza.

Para descobrir onde estamos, digita-se:

$ pwd

/usr/bin

Nesse exemplo, o diretório de trabalho é /usr/bin. Para encontrar o nome do diretório home, digite echo seguido pela variável $HOME:

$ echo $HOME

/root

Aqui, o diretório home é /root. Para voltar ao diretório home, apenas digite o comando cd sem argumentos:

$ cd

Para listar o conteúdo do diretório home, ou digita-se o caminho do diretório home ou use o comando ls sem um nome de diretório. Usando a opção -a para ls permite com que você veja os arquivos ocultos (arquivos que começam com ., dot files) bem como outros arquivos. Com a opção -l você verá uma lista detalhada de informações de cada arquivo.

```
ls -la

total 22532

drwxr-x---  61 root root    4096 Jun  7 14:55 .

drwxr-xr-x  27 root root    4096 Jun  7 11:20 ..

drwxr-xr-x   3 root root    4096 Aug 15  2005 .adobe

-rw-r--r--   1 root root 6231375 Apr  4 15:15 ALP.zip

drwxr-xr-x   7 root root    4096 May 12  2005 .amsn

drwx------   2 root root    4096 May  6 11:32 amsn_received

-rw-r--r--   1 root root    1342 Apr  4  2005 anaconda-ks.cfg

drwxr-xr-x  12 root root    4096 May 26 15:44 assis

drwxr-xr-x   2 root root    4096 Jun  2  2005 .assistant

-rw-r--r--   1 root root     157 May  2 18:54 .aumixrc

-rw-------   1 root root   17411 Jun  7 00:11 .bash_history

-rw-r--r--   1 root root      24 Sep 23  2004 .bash_logout

-rw-r--r--   1 root root     248 Mar  2 11:04 .bash_profile

-rw-r--r--   1 root root     176 Jun  2  2005 .bashrc

drwx------   2 root root    4096 Oct 19  2005 .camel_certs

-rw-r--r--   1 root root    4328 Apr  7 20:44 chardev.c

-rw-r--r--   1 root root 7091766 Mar 20 15:29 ColdFire_Overview.pdf

-rw-r--r--   1 root root    1580 Mar 22 21:24 Compiere.properties

drwxr-xr-x   4 root root    4096 Mar  2 11:25 .config

-rw-r--r--   1 root root     487 Jun  2  2005 .cshrc

-rw-------   1 root root       0 Oct 24  2005 .cvspass

-rw-r--r--   1 root root      71 Jun  7 11:29 .DCOPserver_yellow.hpgpds.ene.unb.br__0

-rwxr-xr-x   1 root root    5889 Feb 23 20:31 sample

drwxrwx---   3 root root    4096 Jun 15  2005 .sane

-rw-------   1 root root   20832 May 19  2005 sintaxe

drwx------   2 root root    4096 Oct 19  2005 .spamassassin

drwx------   2 root root    4096 May 24 20:14 .ssh

-rw-r--r--   1 root root      63 Apr  5  2005 .sversionrc

-rw-r--r--   1 root root     102 Sep 23  2004 .tcshrc

-rwxr-xr-x   1 root root    7780 Mar 14 16:08 teste

-rw-r--r--   1 root root    3019 Mar 14 16:08 teste.c

-rw-r--r--   1 root root    3014 Mar 14 16:08 teste.c~

-rwxr-xr-x   3 root root    4096 May  6 11:31 UnB

-rw-r--r--   1 root root     145 Feb 24 11:13 unnamed.pro
```

```
drwxr-xr-x   2 root root    4096 Apr 19  2005 vanessa
-rw-------   1 root root   10821 Jun  2 16:55 .viminfo
drwxr-xr-x   3 root root    4096 Oct 22  2005 .vlc
drwxr-xr-x   5 root root    4096 Jan 31 17:57 .xMule
```

## Checando a atividade do sistema

Além de ser um sistema operacional multiusuário, o Linux é também um sistema multitarefa. Multitarefa significa que muitos programas podem estar rodando ao mesmo tempo. Uma instância de um programa em execução é chamada de processo. O Linux disponibiliza ferramentas para listar os processos em execução, monitorar o uso do sistema e parar os processos sempre que necessário.

A aplicação mais comum para checar os processos em execução é o comando ps. Use-o para ver que programas estão em execução, os recursos consumidos e quem os está solicitando.

```
$ ps au
USER       PID $CPU $MEM    VSZ  RSS TTY      STAT START   TIME COMMAND
root       568  0.0  0.4   1440  464 ttyS0    Ss+  May09   0:00 /usr/sbin/gpm -m /dev/mouse -t ms
root       570  0.0  0.5   1392  476 tty1     Ss+  May09   0:00 /sbin/agetty 38400 tty1 linux
root     17580  1.8  1.7   2744 1648 pts/0    Ss   17:57   0:00 -bash
root     17593  0.0  0.8   2408  840 pts/0    R+   17:57   0:00 ps au
```

Nesse exemplo, a opção a pede para mostrar os processos de todos os usuários que estão associados ao terminal atual e a opção u pede que os nomes dos usuários sejam mostrados bem como outras  informações como quando o processo começou, o uso de memória e CPU. O conceito de terminal vem dos velhos tempos, quando as pessoas trabalhavam a partir de terminais de caracteres de forma que um terminal tipicamente representava uma pessoa em uma tela. Hoje é possível ter muitos terminais em uma tela abrindo múltiplas janelas de terminal.

Muitos processos rodam em um computador que não está associado a um terminal. Um sistema Linux tem muitos processos rodando em background. O sistema de processos de background gerencia tarefas como as atividades de log do sistema ou espera por dados que vêm via rede. Esses processos iniciam-se quando o sistema inicializa-se e executam constantemente até que o sistema seja desligado. Para navegar entre os processos de um sistema, adicione um pipe (|) e o comando less a ps aux

```
ps aux | less
```

Um pipe direciona a saída de um comando para a entrada de outro comando. Com a linha de comando acima, você pode navegar entre as páginas de saída do ps aux usando less por meio do caractere espaço; digitar q faz com que você saia do

comando.

## Saindo do shell

Para sair do shell quando você tiver terminado de trabalhar, digite exit ou Ctrl+D. Se você estiver saindo de seu login shell, digite logout.

## Usando o shell no Linux

Quando você digita um comando no Linux, você pode incluir outros caracteres que modificam ou adicionam comportamentos do programa. Você pode ajustar seqüências de comandos ou redirecionar a entrada e a saída de um comando por meio de caracteres de controle como pipes (|), :, maior que (>) e menor que (<).

Além do comando, há outros itens que você pode digitar numa linha de comando:

- Opções: A maioria dos comandos tem uma ou mais opções que você pode adicionar a fim de mudar o seu comportamento. Opções tipicamente consistem de uma letra simples precedidas por um traço. Você também pode combinar várias opções depois de um traço. Por exemplo, o comando ls -la lista o conteúdo do diretório atual. O -l pede uma lista detalhada de informações e o -a pede também os arquivos ocultos. Quando uma opção simples consiste de uma palavra, ela é precedida por --. Por exemplo, para usar a opção help na maioria dos comandos, entre - -help na linha de comando.
- Argumentos: Muitos comandos também aceitam argumentos depois de certas opções terem sido especificadas ou no fim da linha de comando. Um argumento é uma peça extra de informação, como o nome do arquivo, que pode ser usado pelo comando. Por exemplo, cat /etc/passwd mostra o conteúdo de /etc/passwd na sua tela. Nesse caso, /etc/passwd é o argumento.
- Variáveis de ambiente: O shell armazena informações que podem ser úteis à sessão do shell e que são chamadas de variáveis de ambiente. Exemplos de variáveis de ambiente são $SHELL, $PS1 (define o prompt do shell, $, #, etc) e $MAIL.
- Metacaracteres: Esses são caracteres que têm significado especial para o shell. Eles podem ser usados para direcionar a saída da linha de comando a um arquivo (>), tunelar a saída a outro comando (|) e rodar um comando em background.

A fim de evitar digitação, existem recursos do shell que armazenam os comandos que você deseja reusar, relembram comandos anteriores e editam comandos. Você pode criar aliases que permitem com que você digite um comando curto em vez de um comando longo. O shell armazena comandos anteriores na sua lista de histórico, permitindo que você reveja-os e reuse-os.

## Localizando comandos

Se você souber o diretório em que se encontra o comando que você deseja executar, uma maneira de executá-lo é digitar o caminho completo para o mesmo. Por exemplo, você roda o comando date a partir do diretório /bin digitando:

$ /bin/date

É claro, isso pode ser incoveniente, especialmente se o comando fica em um diretório de caminho longo. A melhor maneira é ter comando armazenados em locais bem determinados e adicionar esses locais à variável de ambiente PATH. PATH consiste na lista de diretórios que são checados seqüencialmente para os comandos que forem digitados. Para ver o valor de PATH:

# echo $PATH

/

usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/usr/games:/usr/lib/java/bi

n:/usr/lib/java/jre/bin

Os diretórios em PATH são separados por :. A maioria dos comandos que vem com o Linux ficam armazenados em /bin, /usr/bin ou /usr/local/bin. Se você for o usuário root, os diretórios contendo comandos administrativos estão incluídos em $PATH. Esses diretórios incluem /sbin e /usr/sbin.

A ordem mostrada em $PATH é importante. Os diretórios são checados da esquerda para a direita. Por exemplo, se houver um comando localizado tanto em /bin como em /usr/bin, o comando que está em /usr/bin é executado. Para executar o comando do diretório desejado, talvez faça-se necessário usar o caminho completo para o mesmo.

Nem todos os comandos que você executa estão localizados nos diretórios da variável $PATH. Alguns comandos fazem parte do shell. Outros comandos podem ser sobrescritos por meio da criação de aliases que definem comandos e opções que você deseje executar. Há também formas de definir uma função que consiste em uma série comandos armazenados. Aqui segue a ordem em que o shell busca por comandos digitados:

1. Aliases: Nomes configurados pelo comando alias representam um comando particular e um conjunto de opções. Geralmente os aliases permitem com que você defina um nome curto a um comando longo e complicado.
2. Palavras reservadas: Palavras que são reservadas pelo shell para uso especial. Muitas são palavras que você usaria num contexto de programação como do, while, case e else.
3. Função: Um conjunto de comandos que são executados juntos dentro de um shell.
4. Comandos embutidos: Um comando que é embutido no shell. Como resultado, não há representação do comando no sistema de arquivos. A maioria dos comandos mais comuns são comandos embutidos, como cd, echo (ecoa texto na tela), exit, fg (trás tarefas do background para o foreground), history (lista os comandos usados anteriormente), pwd, set (ajusta as opções do shell) e type (mostra a localização de um comando).

Para descobrir de onde um comando é executado, digite type. Por exemplo:

```
$ type bash

bash is /usr/bin/bash
```

Tente type com as seguintes palavras: which, case e return.

Se um comando residir em várias localizações, você pode usar a opção -a para descobrir as várias localizações do mesmo.

## Reexecutando comandos

É de certa forma frustrante depois de digitar um comando longo e complexo, descobrir que você cometeu erros de digitação. Felizmente, algumas das características do shell permite recuperar os comandos anteriores, editá-los ou completar uma comando digitado parcialmente.

O histórico do shell é uma lista dos comandos que você digitou anteriormente. Usando esse comando num shell bash, você poderá ver os comandos anteriores. Então, usando várias características do shell, você poderá recuperar comandos individuais a partir da lista e mudar os mesmos ao seu bel prazer.

## Linha de comando com completamento

Para evitar digitação, o shell bash oferece várias maneiras de completar comandos parcialmente digitados. Para tentar completar um valor, digite os primeiros caracteres e pressione Tab. Contextos interessantes:

- Variáveis de ambiente: Se o texto começa com um $, o shell completa o mesmo com uma variável de ambiente do shell.
- Nome de usuário: Se o texto começa com ~, o shell completa o texto com o nome do usuário.
- Comandos, alias ou funções: Se o texto começa com caracteres regulares, o shell tenta completar o texto com comandos, alias ou funções.
- Nome da máquina: Se o texto começa com @, o shell tenta completá-lo com o nome da máquina.

Ocorrerá situações em que há vários valores de completamento para uma string de caracteres que você digitou. Nesse caso, você pode checar os valores disponíveis pressionando Esc ou Tab (duas vezes).

## Recuperação da linha de comando

Depois de digitar o comando, o comando é salvo na lista de histórico do shell. Esse lista é armazenada num arquivo histórico que permite que qualquer comando possa ser chamado novamente. Depois de recuperar a linha, você poderá modificá-la.

Para ver a sua lista de histórico, use o comando history. Entre o comando sem opções ou seguido por um número do tamanho da lista dos comandos mais recentes. Por exemplo:

```
$ history 8
```

Um número precede cada comando na lista. Há várias maneiras de executar um comando diretamente da lista, incluindo:

- !n: executa o comando relativo ao número n. Substitua o n pelo número da linha do comando e o comando executará. Por exemplo:

```
$ !12
```

- !!: Executa o comando anterior. Executa a linha de comando anterior.

```
$ !!
```

- !?string?: executa o comando que contenha a string. Isso faz com que o comando mais recente que contenha a string particular de caracteres seja executa. Por exemplo, se você tiver executado o comando date anteriormente, para rodá-lo anteriormente, você pode tentar:

```
$ !?dat?
```

Em vez de rodar uma linha de comando do histórico diretamente, você pode recuperar uma linha particular e editar. Temos uma tabela com a seguintes teclas de atalho:

| **Atalho** | **Nome da função** | **Descrição** |
|---|---|---|
| Setas (cima/baixo) | Passo | Pressione as setas para passear entre os comandos da sua lista de histórico até chegar no comando desejado. |
| Crtl+R | Busca incremental reversa | Depois de pressionar essas teclas, entre uma string de busca para realizar a busca reversa. À medida que você digita, um casamento com a linha de comando provável é apresentado. |
| Crtl+S | Busca incremental direta | O mesmo que o anterior, mas de ordem direta na busca |
| Alt+P | Busca reversa | Depois de pressionar essas teclas, entre a string para fazer a busca reversa. Digite a string e |

| Atalho | Nome da função | Descrição |
|---|---|---|
| | | pressione Enter para ver o comando mais recente que inclui a string. |
| Alt+N | Busca direta | O mesmo que o anterior, mas de ordem direta na busca |
| Alt+< | Começo do histórico | Vai para o começo da lista do histórico |
| Alt+> | Fim do histórico | Vai para o fim da lista do histórico |

A lista de histórico é armazenada em .bash_history no diretório home do usuário. Até 1000 comandos são armazenados, por padrão.

## Conectando e expandindo comandos

Uma característica interessante do shell é a sua capacidade de redirecionar a entrada e a saída dos comandos para outros comandos e arquivos. Para permitir que comandos seja conectados, o shell usa metacaracteres.

## Tunelando comandos

O metacaracter pipe (|) conecta a saída de um comando à entrada do outro. Isso permite que você tenha um comando trabalhando em um conjunto de dados e que o próximo comando atue processando os resultados do primeiro. Aqui é apresentado um exemplo de linha de comando que inclui pipes:

```
$ cat /etc/passwd | sort | less
```

Esse comando lista o conteúdo de /etc/passwd e tunela a saída ao comando sort. O comando sort pega os usuários que começam em cada linha de /etc/passwd, ordena-os alfabeticamente e tunela para o comando less.

## Usando metacaracteres de redirecionamento de arquivo

Comandos recebem dados da entrada padrão e envia-os à saída padrão. Usando pipes, você pode direcionar a saída padrão de um comando para a entrada padrão do outro. Com arquivos, você pode usar menor que < e maior que > para direcionar dados de e para arquivos.

- < : Direciona o conteúdo de um arquivo a um comando;

- > : Direciona a saída de um comando a um arquivo, deletando o arquivo existente.
- >>: Direciona a saída de um comando a um arquivo, adicionando a saída do mesmo ao fim de um arquivo já existente.

## Comandos seqüenciais

Às vezes, é desejavel executar uma seqüência de comandos, com um comando sendo iniciado após a execução do outro. Você pode obter essa seqüência digitando os vários comandos desejados separados por ;.

## Comandos em background

Alguns comandos podem demorar consideravelmente até completarem a sua execução. Às vezes, você deseja não ter que esperar o comando terminar para continuar usando o shell. Nesses casos, você pode ter vários comandos em background usando o caractere &.

## Criando o seu ambiente shell.

Você pode ajustar o seu shell de forma que ele trabalhe mais eficientemente. Seu prompt pode te prover informações pertinentes sempre que você pressionar Enter. Você pode usar aliases para economizar digitação e ajustar variáveis de ambiente sempre que necessário. Para que essas mudanças fiquem ativas cada vez que você inicialize o shell, adicione essas configurações aos arquivos de configuração do shell.

## Configurando o seu shell

Vários arquivos de configuração ajustam o comportamento do shell. Alguns dos arquivos são executados para cada usuário e para cada shell enquanto outros são específicos ao usuário que criou o arquivo de configuração. Segue uma lista de arquivos que são de interesse:

| Arquivo | Descrição |
|---|---|
| /etc/profile | Ajusta as informações de ambiente de cada usuário. É executado quando você loga pela primeira vez. Disponibiliza valores de path bem como variáveis de ambientes úteis. Além disso, captura as informações de configuração de arquivos indicados no /etc/profile.d. |
| /etc/bashrc | Executa para cada usuário que roda o bash shell cada vez que uma sessão for |

| Arquivo | Descrição |
|---|---|
| | aberta. Diz qual é o prompt padrão e pode por um ou mais aliases. Valores são sobrescritos por informações de ~/.bashrc. |
| ~/.bash_profile | Usado sempre que cada usuário deseje configurar coisas específicas ao seu shell. É executado apenas uma vez, quando do início da sessão. Ajusta algumas variáveis de ambiente e executa o .bashrc do usuário. |
| ~/.bashrc | Contém informações que são específicas aos shells bash. É lido sempre que você inicia uma sessão. É o melhor local para adicionar variáveis de ambiente e aliases. |
| ~/.bash_logout | Executa sempre que o usuário finaliza uma sessão. Por padrão, apenas limpa a tela. |

Adicionando variáveis de ambientes

Você pode considerar adicionar variáveis ao .bashrc. Elas podem te auxiliar a trabalhar de forma mais eficiente.

- TMOUT: Ajusta quanto tempo o shell deve ficar inativo até que finalize. O valor é o número de segundos para os quais o shell não recebeu entrada. Isso é importante do ponto de vista de segurança caso você deixe o computador logado e ausente-se dele. De maneira a não finalizar a sua sessão enquanto você estiver trabalhando, use valores convenientes, como 1800 (30 minutos).
- PATH: Como descrito anteriormente, PATH indica os diretórios que devem ser procurados na execução de programas. Se for freqüente usar diretórios de comandos que não estão em PATH, pode ser interessante adicioná-los em PATH permanentemente. Para fazê-lo, basta adicionar os diretórios à variável PATH no arquivo .bashrc. Para adicionar um diretório chamado /getstuff/bin: PATH=$PATH:/getstuff/bin; export PATH. Esse exemplo inicialmente lê o conteúdo anterior de PATH, adiciona ao novo PATH além de adicionar o diretório desejado e exporta a nova variável PATH.
- WHATEVER: Você pode criar suas próprias variáveis de ambiente para proporcionar atalhos às suas tarefas. Escolha qualquer nome que não estiver sendo usado e associe um valor útil ao mesmo. Por exemplo, se você estiver trabalhando bastante com os arquivos do diretório /work/time/files/info/memos, você pode ajustar a seguinte variável: M=/work/time/files/info/memos; export M. Você pode mudar do diretório atual digitando cd $M. Você pode rodar um programa do diretório a partir do diretório atual digitando $M/roletarussa.

## Usando aliases

Ajustar aliases pode te salvar mais tempo do que ajustar variáveis de ambiente. Com aliases, você pode ter uma string de caracteres executando uma linha de comando complexa. Você pode adicionar e listar os aliases com o comando alias. Alguns exemplos:

```
alias p='pwd ; ls -CF'

alias rm='rm -i'
```

No primeiro exemplo, à letra p é associado a seqüência dos comandos pwd e ls -CF para imprimir o conteúdo do diretório localiza e a lista de seus conteúdos na forma de coluna. O segundo, executa o comando rm com a opção -i cada vez que você digitar rm.

Quando você estiver num shell, você pode checar quais aliases foram ajustados digitando o comando alias. Se você quiser remover um alias, digite unalias.

## Gerenciando processos em background e foreground

Você pode colocar um programa ativo em background de várias formas. Uma maneira é adicionar & no final da sua invocação. Outra forma é usar o comando at para rodar comandos não necessariamente conectado ao shell.

Para parar um programa e pô-lo em background, digite Ctrl+Z. Depois de ter interrompido o programa, você pode trazê-lo para foreground (comando fg) ou fazer ele rodar em backgroud (comando bg).

## Criando arquivos e diretórios

Como usuário Linux, a maioria dos arquivos que você salva e trabalha estarão provavelmente no diretório home. Aqui seguem alguns comandos usados para criar e usar arquivos e diretórios:

| Comando | Resultado |
|---|---|
| cd | muda de diretório |
| pwd | mostra o nome do diretório atual |
| mkdir | cria um diretório |
| chmod | muda as permissões de um arquivo ou diretório |
| ls | lista o conteúdo de um diretório |

## Movendo, copiando e deletando arquivos

Comandos para mover, copiar e deletar arquivos são de certa forma diretos. Para mudar a localização de um arquivo, use o comando mv. Para copiar um arquivo de uma posição para outra, use o comando cp. Para remover um arquivo, use o comando rm.

## Entendendo permissões de arquivos

Depois de trabalhar com o Linux por um tempo, é muito provável que você se depare com a mensagem: “Permissão negada”. Permissões associadas a arquivos e diretórios são destinadas a manter a privacidade dos arquivos dos usuários e a integridade do sistema, não permitindo que pessoas não autorizadas acessem conteúdos restritos.

São nove os bits associados às permissões de cada arquivo e que definem a política de acesso do proprietário do arquivo e de outros usuários do sistema. Os bits de permissão aparecem como rwxrwxrwx. Os primeiros três bits definem as permissões do proprietário do arquivo, os próximos três aplicam-se ao grupo associado ao arquivo e os outros três aplicam-se a todos os outros usuários. O r é de leitura, o w de escrita e o x de execução. Se um traço aparecer em vez de uma letra, significa que a não há permissão de leitura ou escrita ou execução.

Devido a diferenças entre arquivos e diretórios, as permissões de leitura, escrita e execução nos arquivos são diferentes das relativas aos diretórios.

| Permissão | Arquivo | Diretório |
|---|---|---|
| leitura | acessar o conteúdo do arquivo | ver quais são os arquivos e subdiretórios que ele contém |
| escrita | mudar o conteúdo do arquivo, movê-lo ou deletá-lo. | adiciona arquivos ou subdiretórios ao diretório |
| execução | executar o arquivo como um programa | Muda para o diretório de maneira que ele seja o atual, busca pelo diretório ou executa um comando a partir do diretório. |

Você pode verificar a permissão para qualquer arquivo ou diretório digitando o comando ls -ld.

```
$ls -ld modulo1 curriculum_vitae.rtf

-rw-r--r--  1 root root 57108 May 19 14:44 curriculum_vitae.rtf

drwxr-xr-x  2 root root  4096 Jun  8 15:13 modulo1
```

A primeira linha diz que o arquivo curriculum_vitae.rtf tem permissão de leitura e escrita para o proprietário. O grupo e os outros só podem ler o arquivo. A segunda linha diz que modulo1 é um diretório (d). O proprietário pode ler, escrever e executar; o grupo e os outros só podem ler e executar. Como resultado, o proprietário pode adicionar, mudar ou deletar arquivos arquivos que estão no diretório e todo mundo por ler o conteúdo do mesmo, entrar no diretório e listar o conteúdo do mesmo.

Se você é proprietário de um arquivo, você pode usar chmod para mudar a permissão dele sempre que desejar. Uma maneira de fazê-lo associa a cada permissão um número: r=4, w=2 e x=1; você pode usar o número total de cada conjunto para ajustar a permissão. Por exemplo, para fazer as permissões totalmente abertas para o usuário, ponha o primeiro número igual a 7 (4+2+1) e dê ao grupo e outros usuários apenas a permissão de leitura, ajustando o segundo e terceiro números em 4 (4+0+0), de forma que o número final é 744. Quaisquer combinações de permissões podem ir de 0 (nenhuma permissão) a 7 (controle total).

```
$chmod 744 curriculum_vitae.rtf
```

# 4. Organização dos diretórios

O que se espera encontrar em um sistema de arquivos?

- Processos;
- Dispositivos de *hardware*;
- Canais de comunicações entre processos;
- Segmentos de memória compartilhada.

Se o sistema for UNIX, a resposta é "todas as anteriores". E, é claro, você pode encontrar aí alguns arquivos. [3]

Embora o propósito básico de um sistema de arquivos seja representar e organizar os recursos de armazenamento do sistema, os programadores são ávidos por evitar reinventar a roda quando se trata do gerenciamento de outros tipos de objetos. Freqüentemente, tem-se provado que é natural e conveniente mapear esses objetos para o espaço de nomes do sistema de arquivos. Há algumas vantagens para essa unificação (uma interface de programação consistente, de acesso fácil a partir do *shell*) e algumas desvantagens (implementações Frankenstein do sistema de arquivos), mas, quer você aprove isso ou não, esse se tornou o estilo UNIX.

O sistema de arquivos pode ser pensado como se abrangisse quatro componentes principais:

- um espaço de nomes – uma maneira de nomear e organizar as coisas em uma hierarquia;
- uma API – um conjunto de chamadas de sistema para navegar e manipular nós;
- um modelo de segurança – um esquema para proteger, ocultar e compartilhar as coisas;
- uma implementação – o código que associa o modelo lógico a um disco real.

Sistemas modernos de arquivos UNIX definem uma interface abstrata no nível do *kernel* que acomoda diferentes *back-ends*. Algumas partes da árvore de arquivos são tratadas pela implementação tradicional baseada em disco; outras são agrupadas em campos por *drivers* separados dentro do *kernel*. Por exemplo, sistemas de arquivos NFS são tratados por um *driver* que encaminha as operações solicitadas para um servidor em um outro computador.

Infelizmente, os limites arquitetônicos não são claramente esboçados e há vários casos especiais. Por exemplo, arquivos de disposistivos fornecem uma maneira para que os programas se comuniquem com os *drivers* dentro do *kernel*. Eles não são realmente arquivos de dados, mas são tratados pelo *driver* básico do sistema de arquivos e suas características são armazenadas em disco. Talvez os detalhes pudessem ser relativamente diferentes se o sistema de arquivos fosse reimplementado à luz das experiências das últimas décadas. [manual de administração]

Começemos com algumas definições e sigamos pela padronização da hierarquia do sistema de arquivos Linux/UNIX, o que nos ajudará entender os nomes e funções dos diretórios, bastante diferentes dos usados no amplamente difunfido Windows.

## Nomes de caminho

O sistema de arquivos é apresentatdo como uma única hierarquia que inicia no diretório / e continua em um número arbitrário de subdiretórios. / também é chamado diretório-raiz.

A lista de diretórios que deve ser percorrida para localizar um arquivo em particular junto com seu nome de arquivo forma um "nome de caminho". Nomes de caminho podem ser absolutos (/tmp/foo) ou relativos (book3/filesystem). Nomes de caminho relativos são interpretados iniciando-se no diretório atual. Talvez você esteja acostumado a pensar no diretório atual como um recurso de *shell*, mas cada processo tem um diretório atual.

Os termos *arquivo*, *nome de arquivo*, *nome de caminho* e *caminho* são mais ou menos intercambiáveis. O *nome de arquivo* e o *caminho* podem ser utilizados tanto para caminhos absolutos como relativos; o *nome de caminho*, geralmente, sugere um caminho absoluto.

O sistema de arquivos pode ser arbitrariamente profundo. Mas cada componente de um nome de caminho deve ter um nome com não mais que 255 caracteres de extensão e um único caminho não pode conter mais que 1023 caracteres. Para acessar um arquivo com um nome de caminho mais longo que isso, você deve mudar (cd) para um diretório intermediário e utilizar um nome de caminho relativo.

Essencialmente, não há nenhuma restrição na nomeação dos arquivos e diretórios, exceto pelo fato de que os nome têm um limite de comprimento e não devem conter o caractere / nem o caracter nulo. Em particular, são permitidos espaços (mais ou menos, evite-os sempre que possível). Por causa da longa tradição do UNIX em separar os argumentos da linha de comando com um espaço em branco, o *software* legado tende a se confundir quando aparecem espaços dentro dos nomes de arquivo.

Hoje em dia, dada a quantidade de compartilhamento de arquivos entre tipos de sistemas diferentes, não é mais seguro assumir que todos os nomes de arquivos não conterão espaços. Mesmo se você não compartilhar arquivos em Macs e Pcs, há muitos usuários que têm o hábito de digitá-los. Quaisquer *scripts* que você escreva que lidem com o sistema de arquivos devem estar preparados para lidar com os espaços.

Em geral, nomes de arquivos grandes precisam apenas ser colocados entre aspas para manter suas partes juntas. Por exemplo, o comando:

```
$ more "Meu arquivo excelente.txt"
```

preservaria **Meu arquivo excelente.txt** como um único argumento para **more**.

## Padrão da Hierarquia do sistema de arquivos UNIX:

Propósito

Esse padrão possibilita:

- Ao *software*, prever a localização de arquivos e diretórios instalados e
- Aos usuários, predizer a localização de arquivos e diretórios instalados.

Isso é feito para:

Especificar os princípios guia para cada área do sistema de arquivos;

- Especificar os arquivos e os diretórios mínimos necessários;
- Enumerar as exceções aos princípios e
- Enumerar casos específicos onde já ocorreram conflitos históricos.

O documento FHS (File System Hierarchy Standard) é usado para:

- Diferentes desenvolvedores de *software* poderem criar aplicações que atendam a FHS e trabalhem com distribuições que atendem a FHS;
- Criadores de sistemas operacionais permitirem que seus sistema atendam a FHS e
- Usuários entenderem e seguirem a norma FHS em seu sistema.

O documento FHS tem um escopo limitado:

- A disposição local dos arquivos é um problema local de forma que a FHS não coage os administradores de sistemas;
- FHS indica problemas onde a disposição dos arquivos precisam ser coordenadas entre múltiplos indivíduos, como *sites* locais, distribuições, aplicações, documentação, etc.

O sistema de arquivos

O padrão assume que o sistema operacional que atenda ao padrão FHS para seu sistema de arquivos proporcione as mesmas características básicas de segurança encontradas na maioria dos sistemas de arquivos UNIX.

É possível definir dois tipos de arquivos: os compartilháveis vs os não-compartilháveis e os variáveis vs os estáticos. Em geral, arquivos que diferem em qualquer uma dessas duas classificações devem ser localizados em diferentes diretórios. Isso faz com que seja fácil armazenar arquivos com diferentes características de uso em diferentes sistemas de arquivos.

“Compartilháveis” são arquivos que podem ser armazenados em um host e usado por outros. “Não-compartilháveis” são arquivos que não são compartilháveis. Por exemplo, os arquivos no diretório **home** são compartilháveis enquanto dispositivos não o são.

“Estáticos” incluem arquivos binários, bibliotecas, arquivos de documentação e outros arquivos que não mudam sem a intervenção do administrador do sistema. “Variáveis” são arquivos que não são estáticos.

**O sistema de arquivos root**

Propósito

Os conteúdos do sistema de arquivos **root** devem ser destinados ao boot, recuperação ou reparo do sistema;

- Para inicializar um sistema, deve haver informações suficientes na partição root para montar outros sistemas de arquivos. Isso inclui utilitários, configurações, informações de *boot loader* e outros dados essenciais para a inicialização. /usr, /opt e /var são criados de maneira que possam ser localizados em outras partições ou

sistemas de arquivos.

- Para permitir a recuperação ou reparo de um sistema, ferramentas necessárias para que um administrador possa diagnosticar e reconstruir um sistema danificado devem estar presentes no sistema de arquivo root.
- Para restaurar um sistema, ferramentas necessárias para restaurar o sistema a partir de *backups* devem estar presentes no sistema de arquivos root.

Requisitos

Os seguintes diretórios, ou links simbólicos para diretórios, são exigidos em /:

| **Diretório** | **Descrição** |
|---|---|
| bin | Binários dos comandos essenciais |
| boot | Arquivos estáticos para o *boot loader* |
| dev | Arquivos de dispositivos |
| etc | Configurações específicas do computador |
| lib | Bibliotecas compartilhadas e módulos do *kernel* essenciais |
| media | Ponto de montagem para mídia removível |
| mnt | Ponto de montagem temporário para sistemas de arquivo |
| opt | Pacotes de aplicações adicionais |
| sbin | Binários esseciais do sistema |
| srv | Dados para os serviços disponibilizados pelo sistema |
| tmp | Arquivos temporários |
| usr | Hierarquia secundária |
| var | Dados variáveis |

Cada diretório listado acima é especificado em detalhe logo abaixo.

Opções específicas

Os seguintes diretórios, ou links simbólicos para diretórios, devem estar em / se o subsistema correspondente estiver instalado:

| Diretório | Descrição |
|---|---|
| home | Diretórios home dos usuários (opcional) |
| root | Diretório home do usuário root (opcional) |

**/bin: Binários de comandos essenciais ao usuário (para o uso de todos os usuários)**

Propósito

/bin contém comandos que podem ser usados tanto pelo administrador do sistema como pelos usuários, mas que são exigidos enquanto nenhum outro sistema de arquivos estiver montado (por exemplo, operação mono-usuário). Ele também pode conter comandos que são usados indiretamente por *script*.

Requisitos

Não podem ocorrer subdiretórios em /bin.

Os seguintes comandos, ou links simbólicos, são exigidos em /bin

| Comando | Descrição |
|---|---|
| cat | contatena os arquivos para o standard output |
| chgrp | muda o grupo proprietário |
| chmod | muda as permissões de acesso ao arquivo |
| chown | muda o usuário proprietário |
| cp | copia arquivos e diretórios |
| date | mostra e ajusta a hora e a data do sistema |
| dd | converte e copia um arquivo |
| df | mostra o uso dos discos do sistema |

| **Comando** | **Descrição** |
|---|---|
| dmesg | imprime ou controla o buffer de mensagens do *kernel* |
| echo | mostra linha de texto |
| false | não faz nada; falha |
| hostname | mostra ou ajusta o hostname |
| kill | envia sinais aos processos |
| ln | cria links entre os arquivos |
| login | inicia uma sessão do sistema |
| ls | lista o conteúdo do diretório |
| mkdir | cria diretórios |
| mknod | cria dispositivos especiais, sejam de bloco, sejam de caracteres |
| more | navega por um texto |
| mount | monta um sistema de arquivos |
| mv | move e renomeia arquivos |
| ps | relata os estados do processos |
| pwd | imprime o nome do diretório de trabalho atual |
| rm | remove arquivos e diretórios |
| rmdir | remove diretórios vazios |
| sed | editor de fluxo |
| sh | o *shell* de comandos Bourne |
| stty | muda e imprime os ajustes do terminal |
| su | muda a ID do usuário |
| sync | aplica as alteração aos sistemas de arquivos, esvaziando os *buffers* |

| Comando | Descrição |
|---|---|
| true | não faz nada; sucesso |
| umount | desmonta sistemas de arquivos |
| uname | imprime as informações do sistema |

Se o /bin/sh não é o *shell* de comandos Bourne, ele deve ser um link que aponta ao *shell* real.

Opções específicas

Os seguintes programas, ou links simbólicos para programas, deve estar em /bin se o correspondente sistema de arquivos estiver instalado:

| Comando | Descrição |
|---|---|
| csh | *shell* C |
| ed | editor |
| tar | programa de arquivamento |
| cpio | programa de arquivamento |
| gzip | compressor GNU |
| gunzip | descompressor GNU |
| zcat | descompressor GNU |
| netstat | ferramenta usada para gerar estatísticas de rede |
| ping | testa a rede usando o protocolo ICMP |

**/boot**

Propósito

Esse diretório contém o necessário para o processo de inicialização exceto arquivos de configuração não necessários à inicialização. Portanto, /boot armazena os dados que são usados antes de o *kernel* iniciar a execução em modo usuário. Pode incluir setores

de boot gravados e arquivos de mapas de setores.

Opções específicas

O *kernel* do sistema operacional deve estar localizado ou em / ou em /boot.

**/dev: Arquivos de dispositivos**

Propósito

O diretório /dev contém ou arquivos especiais de dispositivos.

Opções específicas

Se for possível criar manualmente dispositivos em /dev, /dev deve conter um comando chamado MAKEDEV, que pode criar dispositivos sempre que for necessário. Ele também pode conter um MAKEDEV.local para quaisquer dispositivos locais.

Se necessário, MAKEDEV deve prever a criação de eventuais dispositivos que possam ser encontrados no sistema, não apenas aqueles encontrados em implementações ou instalações particulares.

**/etc: Configuração específica do computador**

Propósito

A hierarquia /etc contém arquivos de configuração. Um arquivo de configuração é um arquivo local usado para controlar a operação de um programa; deve ser estático e não pode ser um binários executável.

Requisitos

Não devem ser encontrados binários em /etc.

Os seguintes diretórios, ou links simbólicos para diretórios, são exigidos em /etc:

| **Diretório** | **Descrição** |
|---|---|
| opt | configuração para opt |
| X11 | configuração para o sistema X Window (opcional) |
| sgml | configuração para SGML (opcional) |

| Diretório | Descrição |
|---|---|
| xml | configuração para XML (opcional) |

Opções específicas

Os seguintes diretórios, ou links simbólicos para diretórios, devem estar localizados em /etc se o subsistema correspondente estiver instalado:

| Diretório | Descrição |
|---|---|
| opt | configuração para /opt |

Os seguintes arquivos, ou links simbólicos para arquivos, devem estar em /etc se o correspondente subsistema estiver instalado:

| Arquivo | Descrição |
|---|---|
| csh.login | inicializalção para logins C *shell* |
| exports | lista de controle de acesso para sistema de arquivos NFS |
| fstab | informações estáticas sobre os sistemas de arquivos |
| ftpusers | lista de controle de acesso para o serviço de FTP |
| gateways | arquivo com lista de gateways para routed |
| gttydefs | velocidade e configurações de terminais usadas por getty |
| group | arquivo de grupos de usuário |
| host.conf | arquivo de configuração para a resolução de nomes |
| hosts | informações estáticas sobre nomes de host |
| hosts.allow | arquivo de controle de acesso para TCP wrappers |
| hosts.deny | arquivo de controle de acesso para TCP wrappers |

| Arquivo | Descrição |
|---|---|
| hosts.equiv | lista de hosts confiáveis para rlogin, rsh, rcp |
| hosts.lpd | lista de hosts confiáveis para lpd |
| inetd.conf | arquivo de configuração para inetd |
| inittab | arquivo de configuração para init |
| issue | arquivo de mensagem de pré-login e arquivo de identificação |
| ld.so.conf | lista de diretórios extras para busca por bibliotecas compartilhadas |
| motd | arquivo de mensagem de pós-login |
| mtab | informações dinâmicas sobre o sistema de arquivos |
| mtools.conf | arquivo de configuração para mtools |
| networks | informações estáticas sobre os nomes de rede |
| passwd | arquivo de passwd |
| printcap | banco de dados do serviço de impressão lpd |
| profile | arquivo de inicialização dos logins dos *shell* sh |
| protocols | listagem dos protocolos IP |
| resolv.conf | arquivo de configuração do resolvedor de nomes |
| rpc | listagem dos protocolos RPC |
| securetty | controle de acesso para o login root |
| services | nomes de portas associadas ao serviços de rede |
| shells | caminhos para shells válidos |
| syslog.conf | arquivo de configuração para syslogd |

**/home: diretório home dos usuários (opcional)**

Propósito

/home é um conceito padrão, mas é claramente um sistema de arquivos ajustado para cada situação. O ajuste dependerá das finalidades do computador. Portanto, nenhum programa deve confiar nessa localização.

Requisitos

Arquivos de configuração de cada usuário específicos para aplicações que são armazenadas no diretório home começam com o caracter '.' (um arquivo oculto). Se uma aplicação precisa criar mais que um arquivo oculto, ela deve colocá-los em um subdiretório com um nome que inicie com o caracter '.' (diretório oculto). Nesse caso, os arquivos de configuração não iniciam com o caracter '.'.

**/lib: Bibliotecas compartilhadas essenciais e módulos do kernel**

Propósito

O diretório /lib contém as imagens das bibliotecas compartilhadas necessária para inicializar o sistema e executar os comandos no sistema de arquivos root, por meio dos binários em /bin e /sbin.

Requisitos

Pelo menos um de cada um dos seguintes padrões de nome de arquivos são necessários (podem ser arquivos ou links simbólicos)

| **Arquivo** | **Descrição** |
|---|---|
| libc.so.* | A biblioteca dinâmica C (opcional) |
| ld* | O linker em tempo de execução (opcional) |

Opções Específicas

Os seguintes diretórios, ou links simbólicos para diretórios, devem aparecer em /lib se o subsistema correspondente estiver instalado.

| Diretório | Descrição |
|---|---|
| modules | Módulos do kernel carregáveis em tempo de execução (opcional) |

**/media: Ponto de montagem para mídia removível**

Propósito

Esse diretório contém subdiretórios que são usados como ponto de montagem para mídia removível como discos flexíveis, cdrom e pen-drives.

Opções específicas

Os seguintes diretórios, ou link simbólicos para diretórios, devem constar em /media, se o subsistema correspondente estiver instalado:

| Diretório | Descrição |
|---|---|
| floppy | Disco flexível (opcional) |
| cdrom | Leitor de cd (opcional) |
| cdrecorder | Gravador de cd (opcional) |
| zip | Zip drive (opcional) |

Em sistemas de arquivos onde mais de um dispositivo existe para um certo tipo de mídia, diretórios de montagem podem ser criados pela concatenação de um dígito ao nome do ponto de montagem disponível (tabela acima) começado com '0'.

**/mnt: Ponto de montagem para sistema de arquivos temporariamente montado**

Propósito

Esse diretório é disponibilizado de maneira que o administrador do sistema possa montar sistemas de arquivos temporariamente sempre que necessário. O conteúdo desse diretório é um problema local e não deve afetar o modo de execução de programas.

Esse diretório não pode ser usado por programas de instalação: um diretório temporário conveniente deve ser usado nesse caso.

**/opt: Pacotes adicionais de *software***

Propósito

/opt é criado para a instalação de pacotes adicionais de programas.

Um pacote a ser instalado em /opt deve por seus arquivos estáticos separadamente em /opt/<package>, onde <package> é o nome que descreve o pacote de *software*.

**/root: Diretório home do usuário root (opcional)**

Propósito

O diretório home da conta root pode ser determinado pelo desenvolvedor ou por preferências locais, mas essa é sua localização recomendada.

**/sbin: Binários do sistema**

Propósito

Programas usados por administradores de sistema (e outros comandos de execução permitida somente ao root) são armazenados em /sbin, /usr/sbin e /usr/local/sbin. /sbin contém binários essenciais para a inicialização, recuperação ou reparo do sistema além dos arquivos binários já disponíveis em /bin. Programas executados depois da montagem de /usr são postos em /usr/sbin. Programas de administração de sistemas localmente instalados devem ser postos em /usr/local/sbin.

Requisitos

Os seguintes comandos, ou links simbólicos para comandos, são exigidos em /sbin.

| Comando | Descrição |
|---|---|
| shutdown | Comando para desligar o sistema |

Opções específicas

Os seguintes arquivos, ou links simbólicos para arquivos, deve aparecer em /sbin se o subsistema correspondente estiver instalado:

| Comando | Descrição |
|---|---|
| fastboot | Reinicializa o sistema sem checar os discos |
| fasthalt | Para o sistema sem checar os discos |
| fdisk | Manipula a tabela de partições |
| fsck | Checa e repara sistema de arquivos |
| fsck.* | Checa e repara sistema de arquivos específico |
| getty | Programa getty |
| halt | Comando para parar o sistema |
| ifconfig | Configura interface de rede |
| init | Processo inicial |
| mkfs | Comando para criar um sistema de arquivos |
| mkfs.* | Comando para criar um sistema de arquivos específico |
| mkswap | Comando para ajustar uma área de swap |
| reboot | Comando para reinicializar o sistema |
| route | Tabela de roteamento IP |
| swapon | Habilita a área de swap |
| swapoff | Disabilita a área de swap |

| Comando | Descrição |
| --- | --- |
| update | Daemon para atualizar periodicamente o sistema de arquivos |

**/srv: Dados para serviços providos pelo sistema**

Propósito

/srv contém dados específicos ao serviços de rede que estão sendo providos pelo sistema.

**/tmp: Arquivos temporários**

Propósito

O diretório /tmp deve ser disponibilizado a programas que requeiram arquivos temporários

O programas não pode levar em consideração o fato de que eventuais arquivos permaneçam entre as várias chamadas do mesmo.

**/usr**

Propósito

O /usr é a segunda maior seção do sistema de arquivos. /usr é compartilhável e somente leitura. Isso significa que /usr deve ser compartilhável entre vários computadores que atendam a norma FHS e não pode permitir escrita no mesmo. Qualquer informação que for específica ao computador ou que varie com o tempo é armazenado em outro lugar.

Grandes pacotes de *software* não devem usar diretamente um subdiretório de /usr.

Requisitos

Os seguintes diretórios, ou links simbólicos para diretórios, são exigidos em /usr.

| **Diretório** | **Descrição** |
|---|---|
| bin | A maioria dos comandos de usuário |
| include | Hearders incluídos por programas C |
| lib | Bibliotecas |
| local | Hierarquia local |
| sbin | Binários de sistema não vitais |

| Diretório | Descrição |
|---|---|
| share | Dados independentes de arquitetura |

Opções específicas

| Diretório | Descrição |
|---|---|
| X11R6 | X Windows System, versão 11 release 6 |
| games | Jogos e binários educacionais |
| src | Código fonte |

**/var**

Propósito

/var contém arquivos de dados variáveis. Isso inclui diretórios spool e arquivos, dados administrativos e de log, arquivos transientes e temporários.

Algumas porções de /var não são compartilháveis entre os diferentes sistemas. Por exemplo, /var/log, /var/lock e /var/run. Outras porções podem ser compartilhadas, notavelmente /var/mail, /var/cache/man, /var/cache/fonts e /var/spool/news.

/var é especificado de forma a permitir montar o sistema de arquivos /usr somente leitura. Tudo que for proveniente de /usr e exigir escrita durante a operação do sistema deve estar em /var.

Se /var não puder ser posto em uma partição separada, é preferível mover /var da partição / e por na partição /usr. Entretanto, /var não deve ser linkada a /usr poque isso torna a separação entre /usr e /var mais difícil e sujeita o sistema a um conflito de nomes. Em vez disso, link /var a /usr/var.

Aplicações não devem adicionar diretórios no topo de /var. Esses diretórios deve ser adicionados somente se tiverem importância fundamental ao sistema.

Requisitos

Os seguintes diretórios, ou links simbólicos para diretórios, são exigidos em /var.

| Diretório | Descrição |
|---|---|
| cache | Cache de programas |
| lib | Informações de estado |

| Diretório | Descrição |
|---|---|
| local | Dados variáveis de /usr/local |
| lock | Arquivos lock |
| log | Arquivos de log e diretórios |
| opt | Dados variáveis de /opt |
| run | Dados importantes para processos em execução |
| spool | Spool de programas |
| tmp | Arquivos temporários preservados entre reinicializações |

Opções específicas

Os seguintes diretórios, ou links simbólicos para diretórios, deve estar em /var se o subsistema correspondente estiver instalado:

| Diretório | Descrição |
|---|---|
| account | Logs de contagem de processos |
| crash | Dumps de crash do sistema |
| games | Dados variáveis de jogos |
| mail | Arquivos de caixas de email de usuários |
| yp | Arquivos do banco de dados do NIS |

## Arquivos de Log

Uma das coisas que o Linux faz bem é documentar cada passo que ele dá. Isso é importante, especialmente quando se considera a infinidade de acontecimentos por trás da operação de um sistema operacional complexo. Às vezes, quando trabalha-se com uma nova ferramenta, ela não funciona adequadamente nem indica claramente o por quê da falha. Outras vezes, você quer monitorar o seu sistema de maneira e descobrir se alguém está tentando acessá-lo ilegalmente. Em ambos os casos, você pode usar arquivos de log para ajudar a rastrear o problema.

As principais ferramentas para logging de erros e mensagens de depuração são os daemons syslogd e klogd. O logging geral do sistema é feito pelo syslogd; Logging que é específico às atividades do kernel é feito por klogd. O logging é feito de acordo com as informações de /etc/syslog.conf. Mensagens são tipicamente direcionadas a arquivos de log que estão usualmente no diretório /var/log. Alguns arquivos de log são:

- boot.log: contém as mensagens dos serviços durante a inicialização do sistema;
- messages: contém informações gerais relacionadas à operação do sistema;
- secure: contém mensagens relacionadas à segurança, como atividades de login;
- Xfree86.0.log ou Xorg.0.log: dependendo do servidor X que você estiver usando, contém mensagens sobre as configurações da placa de vídeo, do mouse e do monitor.

## Arquivos de configuração

Você pode esperar encontrar muitos comandos, arquivos de configuração e arquivos de log nos mesmos lugares em um sistema de arquivos, não importa que distribuição Linux você esteja usando.

Arquivos de configuração são uma peça importante na administração de um sistema Linux (e, é claro, na sua operação). Quase tudo que você ajusta para um computador particular – contas de usuário, endereços de rede ou preferências de interface gráfica – são armazenados em arquivos texto. Esse procedimento têm vantagens e disvantagens.

As vantagens de arquivos texto é que é fácil lê-los e mudá-los. Qualquer editor de texto permitirá realizar as alterações. Por outro lado, entretanto, é que como você pode editar os arquivos de configuração, nenhuma checagem de erro é realizada em primeiro momento. É necessário executar o programa e verificar se o mesmo acusará erros por meio mensagens. Uma vírgula ou aspas no local errado pode fazer com que um sistema inteiro pare.

Os arquivos de configuração ficam confinados ao diretórios /etc (configurações para todo o sistema) e o próprio diretório home do usuário (configurações específicas do usuário).

A seguir são apresentadas descrições de diretórios ( e subdiretórios) que contêm arquivos de configuração úteis. Observar o conteúdo de um arquivo de configuração do Linux pode ensinar muito a respeito da administração do mesmo.

- $HOME: Todos os usuários armazenam informações em seu diretório home que direciona o comportamento das contas de login. A maioria dos arquivos de configuração em $HOME começam com um ponto (.) de maneira que ficam ocultos quando o usuário lista o conteúdo desse diretório. Eles são arquivos ponto (ou ocultos) que definem como o shell do usuário se comporta, a aparência do desktop e as opções usadas no editor de texto. Há até mesmo arquivos como .ssh/* e .rhosts que configuram as permissões de rede de cada usuário.
- /etc: Esse diretório contém as configurações mais básicas do Linux.
- /etc/cron*: Diretórios desse conjunto contém arquivos que definem como o aplicativo crond executa programas diariamente (cron.daily), por hora (cron.hourly), mensalmente (cron.monthly) ou semanalmente (cron.weekly).
- /etc/cups: Contém arquivos que são usados para configurar o serviço CUPS de

impressão.

- /etc/default: Contém arquivos que ajustam valores padrão para várias aplicações. Por exemplo, o arquivo para o comando useradd define o número do grupo padrão, diretório home, data de validade da senha, shell e diretórios esqueletos (/etc/skel) que são usado quando for criada uma nova conta.
- /etc/httpd: Contém uma variedade de arquivos usados para configurar o comportamento do servidor Apache.
- /etc/init.d: Contém cópias permanentes dos scripts de nível de execução do estilo System V. Esses scripts são freqüentemente linkados a arquivos nos diretórios /etc/rc?.d para que cada serviço associado a um script inicialize ou pare para um dado nível de execução. A ? é substituída pelo número do nível de execução (0 a 6).
- /etc/ppp: Contém vários arquivos de configuração usados para ajustar p PPP (Point-to-Point Protocol) de maneira a permitir que você se conecte à Internet usando uma linha discada.
- /etc/rc?.d: Há um diretório rc?.d para cada estado válido do sistema: rc0.d (estado de shutdown), rc1.d (estado mono-usuário), rc2.d (estado multi-usuário), rc3.d (estado multi-usuário mais rede), rc4.d (estado definido pelo usuário), rc5.d (estado multi-usuário, mais rede mais interface gráfica) e rc6.d (estado de reinicialização).
- /etc/security: Contém arquivos que ajustam um conjunto de condições de segurança para o computador. Esse arquivos fazem parte do pacote pam (pluggable authentication modules).
- /etc/skel: Quaisquer arquivos contidos nesse diretório são automaticamente copiados para o diretório home quando o usuário é adicionado ao sistema. Por padrão, esses arquivos são ocultos (arquivos .), como o .kde (um diretório para ajustar as configurações do KDE) e .bashrc (para ajuste de configurações do shell bash).
- /etc/sysconfig: Contém arquivos de configuração importantes que são criados e mantidos por vários serviços (incluindo iptables, samba e a maioria dos serviços de rede). Esse arquivos são críticos para as distribuições Linux que usam ferramentas administrativas com interface gráfica mas nem sempre são usados em todas as distribuições.
- /etc/xinetd.d: Contém um conjunto de arquivos, cada um definindo um serviço de rede que o daemon xinetd escuta em uma porta particular. Quando xinetd recebe um requisição de dado serviço, ele usa a informação nesses arquivos para responder à requisição.

| **Arquivo** | **Descrição** |
|---|---|
| aliases | Contém listas de distribuição usadas pelo serviço de email do Linux |
| bashrc | Padrões para todo o sistema para os usuário do shell bash |
| crontab | Ajusta o ambiente cron e o tempo de execução de tarefas automatizadas |
| csh.cshrc (ou cshrc) | Padrões para todo o sistema para os usuário do shell csh |

| Arquivo | Descrição |
|---|---|
| exports | Contém lista de diretórios locais cujo compartilhamento é disponível por meio do Network File System (NFS) |
| fstab | Identifica os dispositivos de armazenamento de dados de massa mais comuns (disco rígidos, disco flexível, cdrom etc.) e pontos de montagem para o sistema Linux. É usado por mout para escolher quais sistemas de arquivo o sistema deve montar na inicialização. |
| group | Identifica o nome dos grupos e suas Ids (GIDS) que estão definidos no sistema. As permissões de grupo no Linux são definidas pelo segundo dos atributos de permissões rwx (read, write, execute) binários associados a cada arquivo ou diretório. |
| gshadow | Contém shadow passwords para grupos. |
| host.conf | Ajusta as localizações em que nomes de domínio (por exemplo, redhat.com) são buscadas nas redes TCP/IP (como a Internet). Por padrão, o arquivo local hosts é buscado e então o sistema indaga resolvedores de nome em resolv.conf. |
| hosts | Contém endereços IP e nomes de hosts que você pode alcançar de seu computador. |
| hosts.allow | Lista de computadores remotos que são autorizados a usar certos serviços TCP/IP do computador local. |
| hosts.deny | Lista de computadores remotos que não são autorizados a usar certos serviços TCP/IP do computador local. |
| inittab | Contém informações que definem que programas iniciam ou param quando o Linux inicializa, desliga ou muda de nível de execução. É o arquivo de configuração mais básico para inicializar o Linux. |
| lilo.conf | Ajusta os parâmetros do carregador do Linux (Linux Loader). Em particular, lista informações sobre partições bootáveis em seu computador. |

| Arquivo | Descrição |
|---|---|
| modules.conf | Contém aliases e opções relacionadas aos módulos do kernel usados em seu computador. |
| mtab | Contém uma lista de sistema de arquivos que estão montados no momento. |
| mtools.conf | Contém ajustes para ferramentas do DOS no Linux |
| named.conf | Contém ajustes para DNS caso você esteja rodando o seu próprio servidor de DNS. |
| ntp.conf | Contém informações necessárias para executar o Network Time Protocol (NTP) |
| passwd | Armazena informações de conta para todos os usuários válidos para o sistema. Também contém outras informações, como o diretório home e o shell padrão. |
| profile | Ajusta o ambiente para todo o sistema e programas usados por todos os usuários. Esse arquivo é lido sempre que o usuário loga. |
| protocols | Ajusta os números e nomes de protocolos para uma variedade de serviços de Internet. |
| resolv.conf | Identifica as localizações dos servidores DNS que são usado pelo TCP/IP para traduzir os nomes host.domain para números de endereços IP. |
| services | Define serviços TCP/IP e as associações de portas para os mesmos. |
| shadow | Contém as senhas cifradas para usuários que estaõ definidos no arquivo passwd. |
| shells | Lista dos interpretadores de linha de comando que estão disponíveis no sistema bem como sua localização. |
| syslog.conf | Define quais são as mensagens de log a serem rastreadas pelo daemon syslogd e em que arquivos devem ser armazenadas. |
| xinetd.conf | Contém informações simples de configuração para o processo xinetd. |

## Dispositivos

Linux, como a maioria dos sistemas operacionais, interage com os dispositivos de hardware por meio de componentes modularizados de sotfware chamado device driver. Um device driver esconde as peculiaridades dos protocolos de comunicação do hardware do sistema operacional e permite o sistema interagir com o dispositivo por meio de uma interface padronizada.

No Linux, device drivers são parte do kernel e podem ser ou linkados estaticamente ao kernel ou carregados sobre demanda como módulos do kernel. Device drivers funcionam como parte do kernel e não são acessíveis diretamente aos processos do usuário. Entretanto, Linux proporciona um mecanismo pelo qual os processos podem se comunicar com um device driver – e através dele com os dispositivos de hardware – por meio de objetos semelhantes a arquivos. Esses objetos aparecem no sistema de arquivos e programas podem abri-los, lê-los e escrever nos mesmos como se eles fossem arquivos normais. Usando operações de I/O de baixo nível do Linux ou as operações de I/O da biblioteca C padrão, seus programas podem comunicar-se com os dispositivos de hardware através desses objetos que funcionam como arquivos.

Linux também disponibiliza vários objetos do tipo arquivo que permitem-nos comunicar-se diretamente com o kernel em vez dos device drivers. Esses não são ligados diretamente a dispositivos de hardware; em vez disso, ele disponibiliza vários tipos de comportamentos especializados que podem ser usados por programas.

### Tipos de Dispositivos

Arquivos de dispositivos não são arquivos comuns – eles não representam regiões de dados no disco de um sistema. Em vez disso, os dados lidos ou escritos em um arquivo de dispositivo comunicam-se diretamente com o correspondente device driver, e, daí, com o dispositivo. Arquivos de dispositivo podem ser de dois tipos:

- Um dispositivo de caracter representa o dispositivo de hardware que lê ou escreve um fluxo serial de bytes. Portas seriais e paralelas, drives de fita, dispositivos de terminais e placas de som são exemplos de dispositivos de caracter.
- Um dispositivo de bloco representa um dispositivo de hardware que lê ou escreve dados em tamanhos de blocos fixos. Diferentemente de um dispositivo de caracter, um dispositivo de bloco permite acesso aleatório aos dados armazenados em um dispositivo. Um disco rígido é um exemplo de dispositivo de bloco.

Programas típicos raramente irão usar dispositivos de blocos. Geralmente, faz-se uso de dispositivos de caracter.

### Números de dispositivo

O Linux identifica os dispositivos por meio de dois nomes: o major device number e o minor device number. O major number especifica a que driver um dispositivo corresponde. A correspondência entre os major numbers e os drivers é fixa e faz parte dos códigos fontes do kernel do Linux. O minor number distingue entre os dispositivos individuais ou componentes individuais controlados por um único driver.

Por exemplo, o dispositivo de major 3 corresponde ao controlador IDE primário do sistema. Um controlador IDE pode ter dois dispositivos (discos, fitas ou drives de CD-ROM) conectados a ele; o dispositivo master tem o minor 0 e o slave tem o minor 64.

Partições individuais do master são representadas pelos minors 1, 2, 3 ... As partições individuais no slave são representadas pelos minors 65, 66, 67 ...

Documentação a respeito dos números dos dispositivos podem ser encontradas em /usr/src/linux/Documentation/devices.txt.

## Diretório /dev

Por convenção, um sistema GNU/Linux inclui um diretório /dev contendo todas as entradas de dispositivos de caracter e bloco para os dispositivos que o Linux conhece. As entradas em /dev têm nomes padronizados correspondentes aos major e minor dos dispositivos.

Por exemplo, o master do controlador primário da IDE possui major e minor 3 e 0 e tem o nome padrão /dev/hda. Se o dispositivo suporta partições, a sua primeira partição tem o minor 1 e tem o nome padrão /dev/hda1. Por exemplo:

```
$ ls -l /dev/hda /dev/hda1

brw-rw---- 1 root disk 3, 0 May 5 1998 /dev/hda

brw-rw---- 1 root disk 3, 1 May 5 1998 /dev/hda1
```

## Dispositivos especiais

Linux possui vários dispositivos de caracter que não correspondem a dispositivos de hardware. Todos eles correspondem ao major 1, que está associado à memória do kernel em vez de um dispositivo de hardware.

**/dev/null**

Dispositivo nulo. Descarta qualquer dado escrito no mesmo. Se copiado, resulta em um arquivo vazio.

**/dev/zero**

Stream infinito de zeros.

**/dev/full**

Arquivo cheio.

**/dev/random** e **/dev/urandom**

Gerador de números aleatórios.

ioctl – Interface padrão.

A chamada de sistema ioctl é uma interface genérica usada para controlar os dispositivos de hardware. O primeiro argumento para ioctl é um descritor de arquivo, que deve ser o dispositivo que você deseja controlar. O segundo argumento é um código que indica a operação que você quer realizar. Várias operações estão disponíveis para dispositivos diferentes. Dependendo do código de operação solicitado, podem ser exigidos argumentos adicionais a ioctl.

Muitos dos códigos de operações estão listados na man de ioctl. Usar ioctl requer conhecimento do device driver correspondente ao hardware que você quer controlar.

Um exemplo prático:

```
#include <fcntl.h>
#include <linux/cdrom.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main (int argc, char* argv[])
{
/* Open a file descriptor to the device specified on the command line. */
int fd = open (argv[1], O_RDONLY);
/* Eject the CD-ROM. */
ioctl (fd, CDROMEJECT);
/* Close the file descriptor. */
close (fd);
return 0;
}
```

# 5. Obtendo informações sobre o sistema

Linux, por ser um sistema operacional aberto e de múltiplos propósitos, disponibiliza uma quantidade enorme de informações aos seus usuários, programadores e administradores. A seguir, mostraremos alguns locais de onde podem ser extraídas informações úteis quando se está desenvolvendo aplicações para o Linux.

## Utilidade de /proc

A maioria das entradas em /proc gera informações - das mais variadas - formatadas e legíveis por humanos, mas o formato é tão simples que pode ser facilmente interpretado por um programa. Por exemplo, /proc/cpuinfo contém informações sobre o processador (ou processadores). A saída é uma tabela de valores, um por linha, com a descrição do valor e : precedendo cada valor. Por exemplo:

```
$ cat /proc/cpuinfo

processor : 0

vendor_id : GenuineIntel

cpu family : 6

model : 5

model name : Pentium II (Deschutes)

stepping : 2

cpu MHz : 400.913520

cache size : 512 KB

fdiv_bug : no

hlt_bug : no

sep_bug : no

f00f_bug : no

coma_bug : no

fpu : yes

fpu_exception : yes

cpuid level : 2

wp : yes
```

flags : fpu vme de pse tsc msr pae mce cx8 apic sep

mtrr pge mca cmov pat pse36 mmx fxsr

bogomips : 399.77

Um programa interessante:

```
#include <stdio.h>
#include <string.h>
/* Returns the clock speed of the system's CPU in MHz, as reported by
/proc/cpuinfo. On a multiprocessor machine, returns the speed of
the first CPU. On error returns zero. */
float get_cpu_clock_speed ()
{
FILE* fp;
char buffer[1024];
size_t bytes_read;
char* match;
float clock_speed;
/* Read the entire contents of /proc/cpuinfo into the buffer. */
fp = fopen ("/proc/cpuinfo", "r");
bytes_read = fread (buffer, 1, sizeof (buffer), fp);
fclose (fp);
/* Bail if read failed or if buffer isn't big enough. */
if (bytes_read == 0 || bytes_read == sizeof (buffer))
return 0;
/* NUL-terminate the text. */
buffer[bytes_read] = '\0';
/* Locate the line that starts with "cpu MHz". */
match = strstr (buffer, "cpu MHz");
if (match == NULL)
return 0;
/* Parse the line to extract the clock speed. */
sscanf (match, "cpu MHz : $f", &clock_speed);
```

```
return clock_speed;

}

int main ()

{

printf (“CPU clock speed: $4.0f MHz\n”, get_cpu_clock_speed ());

return 0;

}
```

## /proc: Informações relativas a processos

O sistema de arquivos /proc contém uma entrada para cada processo que está sendo executado no Linux. O nome de cada diretório é a ID do processo correspondente. Esses diretórios aparecem e desaparecem dinamicamente à medida que processos começam e terminam durante a execução do sistema. Cada diretório possui várias entradas que disponibilizam informações sobre o processo em execução.

Cada diretório de processo contém as seguintes entradas:

- **cmdline** contém a lista de argumentos para o processo.
- **cwd** é um link simbólico que aponta para o diretório de trabalho do processo.
- **environ** contém o ambiente do processo.
- **exe** é um link simbólico que aponta para a imagem executável do processo.
- **fd** é o subdiretório que contém entradas para os descritores de arquivos abertos pelo processo.
- **maps** mostra as informações sobre arquivos mapeados no endereço do processo.
- **root** é um link simbólico para o diretório root desse processo.
- **stat** contém várias informações de estado e de estatísticas sobre o processo. Não formatado.
- **statm** contém informações sobre a memória usada pelo processo.
- **status** contém várias informações de estado e de estatísticas sobre o processo. Formatado.
- **cpu** aparece apenas em kernel para sistemas SMP.

## /proc: Informações do sistema

Especificamente, os seguintes arquivos que possuem informações relevantes sobre o sistema.

**/proc/cpuinfo**: informações sobre o processador.

**/proc/devices**: lista os dispositivos disponíveis no sistema, juntamente com o seus minor e major.

**/proc/pci**: lista os dispositivos anexados ao barramento PCI. Cartões PCI e AGP estão incluídos.

**/proc/version**: lista as informações de versão do kernel. Também indica como o kernel foi construído: quem o compilou, em que máquina, a data da compilação, a versão do compilador.

**/proc/sys/kernel/hostname**: indica o nome do computador.

**/proc/sys/kernel/domainname**: indica o domínio do computador.

**/proc/meminfo**: contém informações sobre o uso de memória do sistema. São apresentadas informações tanto para a memória física como para a swap.

**/proc/filesystems**: mostra os tipos de sistemas de arquivos conhecidos estaticamente pelo kernel.

**/proc/ide**: informações sobre as interfaces IDE:

**/proc/mounts**: sumário dos sistemas de arquivos montados

**/proc/uptime**: tempo de execução ininterrupta desde a última inicialização.

Um programa interessante:

```
#include <stdio.h>

/* Summarize a duration of time to standard output. TIME is the

amount of time, in seconds, and LABEL is a short descriptive label. */

void print_time (char* label, long time)

{

/* Conversion constants. */

const long minute = 60;

const long hour = minute * 60;

const long day = hour * 24;

/* Produce output. */

printf ("$s: $ld days, $ld:$02ld:$02ld\n", label, time / day,

(time $ day) / hour, (time $ hour) / minute, time $ minute);

}

int main ()

{

FILE* fp;
```

```
double uptime, idle_time;
/* Read the system uptime and accumulated idle time from /proc/uptime. */
fp = fopen (“/proc/uptime”, “r”);
fscanf (fp, “$lf $lf\n”, &uptime, &idle_time);
fclose (fp);
/* Summarize it. */
print_time (“uptime “, (long) uptime);
print_time (“idle time”, (long) idle_time);
return 0;
}
```

# 6. Instalação padrão de programas

Tirado de http://br-linux.org/tutoriais/001969.html

## Instalação de programas no GNU/Linux

*Por: Pedro Augusto de Oliveira Pereira - pogoslack@bol.com.br*

A instalação de aplicativos no GNU/Linux é um dos pontos que mais causam confusão nos iniciantes por ser um pouco diferente. Para fazer a instalação, nós temos 3 opções: o RPM (para distribuições baseadas na Red Hat como Fedora, Conectiva), a instalação através de código fonte (arquivos .tar.gz, .tar.bz2, etc), sendo que estes independem de distribuição e o apt-get, criado pela Debian. Vamos tratar cada uma delas separadamente para que se possa esclarecer as particularidades de cada uma.

## Instalação através de um tarball

Esse é um dos tipos de instalação mais populares. Aqui o desenvolvedor empacota os arquivos fonte do programa e os disponibiliza para download, junto com alguns scripts para facilitar a instalação no computador do usuário.

Como o desenvolvedor lhe envia os códigos-fonte dos arquivos, será necessário compilá- los para que eles funcionem no seu computador. Para conseguir realizar a compilação de qualquer programa no seu sistema, você deve ter os pacotes de desenvolvimento instalados no seu computador (pacotes como a glibc, automake, etc). Falando assim, pode parecer que instalar aplicativos dessa forma seja complicado. A compilação é um processo um tanto quanto padronizado, ou seja, quase sempre você vai precisar digitar os mesmos comandos para instalar qualquer programa. Cada um desses comandos serão explicados abaixo; lembre-se que todos eles devem ser executados dentro do diretório criado quando você descompactou o tarball do programa:

```
$./configure
```

O ./configure na verdade não é um comando, mas sim um shell script. Quando executado, ele verifica se seu sistema possui tudo o que é necessário para que o aplicativo que você está querendo instalar seja executado corretamente, sem nenhum problema. Além disso, ele também gera um arquivo chamado makefile. Este arquivo contém regras sobre a compilação do programa, sem este arquivo, o próximo comando não conseguirá executar pois não terá idéia do que será necessário fazer.

```
$ make
```

O make usa o makefile gerado pelo ./configure para realizar a compilação do programa.O makefile contém instruções para que o make compile tudo o que for necessário.

```
$ make install
```

Enquanto o make só compila o programa, o make install instala realmente o programa criando os diretórios necessários, colocando os binários no lugar certo, etc. Lembrando que o make install é o único dos 3 comandos que requer permissões de root para ser executado, já que este escreve em lugares em que só o root tem permissão.

Então, para instalar um programa:

```
# tar -xzvf programa.tar.gz

# cd dirdoprograma

# ./configure

# make

# make install
```

Geralmente, são esses os passos executados para que você instale um programa a partir do código fonte. A única coisa que pode variar é o nome do script (o primeiro "comando" digitado). Na maior parte das vezes é configure, mas o desenvolvedor pode colocar outro nome nele, como setup, por exemplo. Por isso, é muito recomendado que você leia o arquivo README e o arquivo INSTALL para ter instruções exatas de como proceder para instalar o aplicativo.

Depois de instalado, os binários do aplicativo vão estar em /usr/local/bin. Preste atenção a um detalhe muito importante: se seu aplicativo só funcionar em um ambiente específico (o KDE, por exemplo) os binários dele vão para outro diretório: /usr/local/kde/bin.

Portanto, nunca se esqueça de adicionar esses diretórios na variável PATH de todos os usuários do sistema, para que eles também possam executar o programa.

Você deve estar pensando: "Bom instalar é fácil, mas como eu desinstalo um programa?". A resposta: "É mais fácil ainda!".

Você simplesmente vai ter que voltar ao diretório criado pelo tarball quando este foi descompactado e digitar:

```
# make uninstall
```

## RPM

O RPM (Red Hat Package Manager) é um dos modos mais inteligentes e fáceis de se instalar, desinstalar, monitorar o que está instalado, atualizar programas, etc. Ele ajuda muito o usuário iniciante pois é muito intuitivo e fácil de se usar. Será explicado

aqui como instalar, desinstalar, atualizar e verificar se algum pacote está instalado no seu sistema.

## Instalando

*Os pacotes RPM são arquivos binários pré-compilados para uma certa distribuição. Por isso você sempre vai procurar por pacotes RPM feitos para a sua distribuição. Um ótimo lugar para se encontrar pacotes RPM é nos sites rpm.pbone.net. e www.rpmfind.net.*

O processo de instalação é bem fácil. Vou usar como exemplo de instalação o SIM, um clone do ICQ para GNU/Linux.

Com o pacote em mãos, vá ao console e digite:

```
# rpm -ivh sim-0.9.2-1.rh90.i386.rpm
```

Explicando as opções:

-i: instalar

-v: modo verbose. Com esta opção ativada, o rpm irá te falar o que está fazendo.

-h: mostra o progresso da instalação do pacote com caracteres #.

Opa! Tivemos uma mensagem de erro:

```
error: Failed dependencies: libsablot.so.0 is needed by sim-0.9.2-1.rh90 sablotron >= 1.0.1 is needed by sim-0.9.2.-1.rh90
```

Nesse caso o RPM está nos dizendo que para o SIM ser executado corretamente ele necessita do pacote sablotron com uma versão maior ou igual a 1.0.1. Portanto, simplesmente procuramos esse pacote (nesse caso é uma biblioteca) e o instalamos:

```
# rpm -ivh sablotron-1.0.1-1.rh90.i386.rpm
```

Depois de instalado, instalamos o pacote principal sim-0.9.2-1.rh90.i386.rpm:

```
# rpm -ivh sim-0.9.2-1.rh90.i386.rpm
```

Agora sim conseguimos instalar o SIM. Para executá-lo digite "sim" em um terminal

## Atualizando

Vamos supor que, depois de um tempo, é lançada uma nova versão do SIM e nós

queiramos atualizar o nosso programa. Para isso nós utilizaremos a opção -U, assim:

```
# rpm -U sim-x.x.x-2.rh90.i386.rpm
```

Assim ele vai atualizar o SIM automaticamente, sem ter que desinstalar a versão antiga e instalar a nova.

Desinstalando

Para você desinstalar um pacote RPM qualquer, basta usar a opção -e:

```
# rpm -e sim-x.x.x-2.rh90.i386.rpm
```

Verificando quais pacotes estão instalados

Para você pesquisar se um certo pacote está instalado ou não, você pode usar a opção -qa. Com essas opções o sistema RPM vai consultar toda a base de dados de pacotes RPM e lhe mostrar todos os pacotes RPM que estão instalados no seu micro. É uma boa idéia usar o grep para consultar um pacote específico:

```
# rpm -qa | grep pacote
```

Assim, se o pacote estiver instalado, você verá o nome do pacote como única saída do comando. Caso o RPM não ache um pacote com o nome especificado ele simplesmente irá ficar quieto, não indicará nada.

## APT-GET com suporte a RPM

Conforme você vai usando o sistema RPM você vai perceber que ele tem um sério problema. Muitas vezes você tem que instalar vários pacotes para conseguir fazer um programa funcionar (esses pacotes extras são chamados dependências).

A fim de evitar uma procura demorada por dependências sempre que se dejesar instalar uma aplicação, foi criado o sistema APT-GET.

Este sistema funciona como o RPM, mas um pouco mais melhorado. Com ele você pode instalar, desinstalar e atualizar pacotes no seu sistema. Mas a maior vantagem do APT sobre o RPM é que ele resolve problemas de dependências automaticamente. Assim, se você tentar instalar um pacote o APT já vai instalar todas as dependências dele automaticamente.

Para instalar o APT você vai precisar baixar o pacote RPM dele e instalá- lo como foi explicado no tópico anterior. Depois de instalá-lo, é hora de começar a usar! Digite "apt-get update" para que o apt atualize os pacotes necessários por ele. Lembre- se que neste passo você tem que estar conectado à Internet.

Depois que ele terminar o passo anterior, você já pode começar a usá- lo para instalar, desinstalar e atualizar seus pacotes.

Para instalar um pacote qualquer digite:

```
# apt-get install pacote
```

Com o comando acima ele vai instalar o pacote e qualquer dependência necessária.

Para que o apt cheque quais pacotes estão desatualizados no seu sistema e já os atualize digite:

```
# apt-get upgrade
```

Para remover algum pacote junto com suas dependências, use:

```
# apt-get remove pacote
```

Todas as fontes de onde o APT vai fazer download estão descritas no arquivo /etc/apt/sources.list. Se você quiser adicionar algum lugar de onde o APT deva fazer download de algum pacote, indique-o nesse arquivo. Uma fonte não precisa necessariamente ser um servidor FTP. Você pode adicionar os CD-ROMs da sua distribuição para poder instalar os pacotes contidos neles pelo APT. Você só vai precisar digitar o comando:

```
# apt-cdrom add
```

Se você não especificar onde está o seu drive, o APT vai usar as informações contidas no seu arquivo /etc/fstab.

# 7. Ferramentas de desenvolvimento em linguagem C

## Compilando com o GCC:

Um compilador torna um código legível a um ser humano em um objeto legível a um computador, que poderá ser executado. Os compiladores mais comuns do GNU/Linux fazem parte do GNU Compiler Collection, vulgo GCC. O GCC inclui compiladores C, C++, Java, Objective-C, Fortran e Chill.

Suponha que você tem um projeto como o apresentado abaixo com um código fonte em C++ (reciprocal.cpp) e um código fonte em C (main.c). Esses dois arquivos devem ser compilados e linkados para produzir um programa chamado reciprocal. Esse programa calculará o inverso de um inteiro. [4]

**main.c**

```
#include <stdio.h>

#include “reciprocal.hpp”

int main (int argc, char **argv)

{

int i;

i = atoi (argv[1]);

printf (“The reciprocal of %d is %g\n”, i, reciprocal (i));

return 0;

}
```

**reciprocal.cpp**

```
#include <cassert>

#include “reciprocal.hpp”

double reciprocal (int i) {

// I should be non-zero.

assert (i != 0);
```

```
return 1.0/i;

}
```

Há também um arquivo header chamado reciprocal.hpp.

**reciprocal.hpp**

```
#ifdef __cplusplus

extern “C” {

#endif

extern double reciprocal (int i);

#ifdef __cplusplus

}

#endif
```

O primeiro passo é tornar os códigos C e C++ em objetos.

## Compilando um único código fonte

O nome do compilador C é gcc. Para compilar um código fonte C, você usa a opção -c. Por exemplo, entrando a linha abaixo no prompt de comando compila o código fonte main.c

```
$ gcc -c main.c
```

O resultado é um arquivo objeto chamado main.o.

O compilador C++ é chamado g++. Sua operação é semelhante à do gcc; para compilar reciprocal.cpp:

```
$ g++ -c reciprocal.cpp
```

A opção -c diz ao g++ para compilar o programa em um arquivo objeto; sem isso, g++ tentaria linkar o programa para produzir um executável. Depois de digitar esse comando, você terá um arquivo objeto chamado reciprocal.o.

Você precisará de um par de opções para construir qualquer programa razoavelmente grande. A opção -I é usada para dizer ao GCC onde procurar por arquivos header. Por

padrão, o GCC olha no diretório atual e nos diretórios onde os headers para as bibliotecas padrão estão instalados. Se você precisar incluir headers de qualquer outra origem, a opção -I se faz útil. Por exemplo, suponha que seu projeto tem um diretório chamado src, para códigos fonte, e outro chamado include. Você pode compilar reciprocal.cpp indicando ao g++ que ele deve olhar no diretório ../include para encontrar reciprocal.hpp:

```
$ g++ -c -I ../include reciprocal.cpp
```

Às vezes, você desejará definir macros na linha de comando. Por exemplo, num código para entrega, você não desejaria overhead de checagem de declarações em reciprocal.cpp; isso é disponível apenas para ajudar a depurar o programa. É possível desligar a checagem definindo a macro NDEBUG. Você pode adicionar explicitamente um #define a reciprocal.cpp, mas isso exigirá a mudança do código fonte. É mais simples definir NDEBUG na linha de comando, como:

```
$ g++ -c -D NDEBUG reciprocal.cpp
```

Se você quiser definir algum valor particular a NDEBUG, você pode fazer o seguinte:

```
$ g++ -c -D NDEBUG=3 reciprocal.cpp
```

Se você estiver realmente produzindo código em larga escala, talvez seja interessante que o GCC otimize o código de maneira que ele execute de forma mais rápida. Você pode conseguir isso usando a opção -O3 na linha de comando. Por exemplo:

```
$ g++ -c -O3 reciprocal.cpp
```

Note que, compilando com otimização pode fazer o seu programa mais difícil de depurar e pode até mesmo ocultar alguns erros.

Você pode passar inúmeras opções para o gcc e o g++. A melhor maneira de obter uma lista completa de opções é usar a documentação online.

```
$ info gcc
```

## Linkando os objetos

Agora que já temos maic.c e reciprocal.cpp compilados, o próximo passo será linká-los. Você deve sempre usar o g++ para linkar programas que usem códigos em C++, mesmo que eles contenham códigos em C. Se o seu programa tiver apenas códigos em C, você pode usar o gcc. Devido ao fato de que o nosso programa têm códigos tanto em C como em C++, devemos usar o g++:

```
$ g++ -o reciprocal main.o reciprocal.o
```

A opção -o dá o nome do arquivo a ser gerado como saída do processo de linkagem. Agora, você pode rodar reciprocal:

```
$ ./reciprocal 7

The reciprocal of 7 is 0.142857
```

Como você pode perceber, g++ está linkado estaticamente à biblioteca padrão de C que contém a função printf. Se você precisar linkar outra biblioteca, você pode especificá-la com a opção -l. No Linux, os nomes de bibliotecas começam com lib. Por exemplo, a biblioteca PAM (Pluggable Authentication Module) é chamada de libpam.a. Para linkar com libpam.a, use um comando do tipo:

```
$ g++ -o reciprocal main.o reciprocal.o -lpam
```

O compilador adiciona o prefixo lib e o sufixo .a.

Da mesma forma que os headers, o linker procura por bibliotecas em lugares padrão, incluindo os diretórios /lib e /usr/lib que contêm as bibliotecas padrões do sistema. Se você quiser que o linker busque outros diretórios, use a opção -L, semelhante à opção -I. Você pode usar essa linha para instruir o compilador a procurar por bibliotecas em /usr/local/lib/pam antes de olhar em lugares padrão.

```
$ g++ -o reciprocal main.o reciprocal.o -L/usr/local/lib/pam -lpam
```

Embora você não tenha que usar a opção -I para fazer com que o preprocessador busque no diretório local, você deve usar a opção -L para fazer com que o linker busque no diretório local. Por exemplo, você poderia ter usado o seguinte comando para instruir o linker para procurar pela biblioteca test no diretório atual:

```
$ gcc -o app app.o -L. -ltest
```

## Automatizando o processo com o GNU Make

Se você está acostumado a programar para o sistema operacional Windows, é provável que você tenha usado IDEs (Integrated Development Environment) durante suas tarefas de desenvolvimento. Você pode adicionar códigos fonte ao seu projeto e pedir à IDE para contruir seu projeto automaticamente. Apesar de termos várias IDEs disponíveis no Linux, mostraremos como usar o GNU Make para recompilar automaticamente seu código, o que é amplamente utilizado pela maioria dos programadores de hoje.

A idéia básica por trás do make é simples. Você diz ao make quais são os alvos que você deseja construir e dá a ele regras explicando como construí-los. Você também pode especificar dependências que indicam como um alvo particular deve ser reconstruído.

Em nosso projeto inicial, existem dois alvos óbvios: reciprocal.o, main.o e o recriprocal. Você já deve ter em mente algumas regras para construir esses alvos na forma de linhas de comando. As dependências precisam um pouco mais de conhecimento. Claramente, reciprocal depende de recriprocal.o e main.o porque não é possível linkar completamente o programa até que tenhamos construído cada um desses objetos. Os objetos devem ser reconstruídos sempre que o código fonte correspondente for modificado. Há mais um detalhe que envolve reciprocal.hpp pois sempre que o modificarmos, a modificação deve provocar a reconstrução do objetos pois os objetos dependem desse header.

Além dos alvos óbvios, deve sempre existir um alvo clean. Esse alvo remove todos os objetos gerados e programas de maneira que você possa recomeçar do zero. A regra para esse alvo usa o comando rm para remover arquivos.

Você pode disponibilizar toda a informação necessária a make colocando-a em um arquivo chamado Makefile. Por exemplo:

```
reciprocal: main.o reciprocal.o

g++ $(CFLAGS) -o reciprocal main.o reciprocal.o

main.o: main.c reciprocal.hpp

gcc $(CFLAGS) -c main.c

reciprocal.o: reciprocal.cpp reciprocal.hpp

g++ $(CFLAGS) -c reciprocal.cpp

clean:

rm -f *.o reciprocal
```

Como você pode ver, os alvos são listados na esquerda, seguindos por : e então são indicadas as dependências. A regra para construir cada alvo aparece na próxima linha. A linha com a regra deve iniciar com o caracter <TAB>, caso contrária o make ficará confuso.

Se você remover os objetos que você tiver construído anteriormente, digite apenas:

```
$ make
```

na linha de comando e verá o seguinte:

```
$ make

gcc -c main.c
```

```
g++ -c reciprocal.cpp

g++ -o reciprocal main.o reciprocal.o
```

Você pode perceber que make automaticamente construiu todos os objetos e linkou-os. Se você mudar o main.c de alguma maneira e digitar make novamente, você verá o seguite:

```
$ make

gcc -c main.c

g++ -o reciprocal main.o reciprocal.o
```

Como você pode ver, o make sabe que deve reconstruir apenas main.o e relinká-lo ao programa mas não se importou em recompilar reciprocal.cpp porque nenhuma das suas dependências sofreram alterações.

O $(CFLAGS) é uma variável make. Você pode definir essa variável ou no Makefile ou na linha de comando. GNU make irá substituir o valor da variável quando executar a regra. Por exemplo, para recompilar com a otimização habilitada:

```
$ make clean

rm -f *.o reciprocal

$ make CFLAGS=-O2

gcc -O2 -c main.c

g++ -O2 -c reciprocal.cpp

g++ -O2 -o reciprocal main.o reciprocal.o
```

Note que a flag -O2 foi inserida no lugar de $(CFLAGS) nas regras.

Nessa seção, você viu como a maioria das capacidades básicas do make. Você poderá encontrar maiores informações na documentação sobre o make.

## Depurando com o GNU Debugger (GDB)

O debugger, ou depurador, é o programa que você usa para descobrir porque o seu programa não está se comportando da maneira esperada. E ele fará isso inúmeras vezes. O GNU Debugger (GDB) é um depurador usado pela maioria dos programadores Linux. Você pode usar o GDB para fazer um passo a passo de seu código, colocar breakpoints e examinar o valor de variáveis locais.

## Compilando com informações de Debugging

Para usar o GDB, você terá que compilar o seu código com informações de debugging. Para fazer isso, adicione -g na linha de comando. Se você estiver usando um Makefile como descrito anteriormente, você pode ajustar as CFLAGS adicionando -g e executar o make:

```
$ make CFLAGS=-g

gcc -g -c main.c

g++ -g -c reciprocal.cpp

g++ -g -o reciprocal main.o reciprocal.o
```

Quando você compila com -g, o compilador inclui informações extra nos arquivos objetos e nos executáveis. O depurador usa essa informação para descobrir a que endereços correspondem as linhas dos códigos fonte e para imprimir as variáveis locais etc.

## Usando o GDB

Você pode rodar o gdb dessa forma:

```
$ gdb reciprocal
```

Quando o gdb inicializa, você deve ver o prompt:

```
(gdb)
```

O primeiro passo é executar o programa dentro do depurador. Apenas digite run e os argumentos do programa.

```
(gdb) run

Starting program: reciprocal

Program received signal SIGSEGV, Segmentation fault.

__strtol_internal (nptr=0x0, endptr=0x0, base=10, group=0)

at strtol.c:287
```

287 strtol.c: No such file or directory.

(gdb)

O problema aqui é que não há checagem de exceções nó código de main. O programa espera um argumento, mas nesse caso o programa foi invocado sem argumentos. O SIGSEGV indica que ocorreu uma quebra. GDB sabe que a quebra aconteceu numa função chamada _strtol_internal. Essa função é da biblioteca padrão C e a seu fonte não foi instalado, o que justifica o "No such file or directory". Você pode ver a pilha usando o comando where:

(gdb) where

#0 __strtol_internal (nptr=0x0, endptr=0x0, base=10, group=0)

at strtol.c:287

#1 0x40096fb6 in atoi (nptr=0x0) at ../stdlib/stdlib.h:251

#2 0x804863e in main (argc=1, argv=0xbffff5e4) at main.c:8

Você pode ver que a função main chamou atoi com um ponteiro NULL, que gerou todos os problemas.

Você pode subir dois níveis na pilha até alcançar a função main usando o comando up:

(gdb) up 2

#2 0x804863e in main (argc=1, argv=0xbffff5e4) at main.c:8

8 i = atoi (argv[1]);

Note que o gdb é capaz de encontrar o fonte de main.c e pode mostrar o linha onde a chamada errônea de função ocorreu. Você pode ver o valor das variáveis usando o comando print:

(gdb) print argv[1]

$2 = 0x0

Isso confirma o problema da passagem de um ponteiro NULL para atoi.

Você pode inserir um breakpoint usando o comando break:

(gdb) break main

Breakpoint 1 at 0x804862e: file main.c, line 8.

Esse comando põe um breakpoint na primeira linha de main. Agora, tente rodar o programa com um argumento:

(gdb) run 7

Starting program: reciprocal 7

Breakpoint 1, main (argc=2, argv=0xbffff5e4) at main.c:8

8 i = atoi (argv[1]);

Você pode ver que o depurador parou no breakpoint.
Você pode continuar a execução usando o comando next:

(gdb) next

9 printf ("The reciprocal of $d is $g\n", i, reciprocal (i));

Se você quiser ver o que está acontecendo em reciprocal, use o comando step

(gdb) step

reciprocal (i=7) at reciprocal.cpp:6

6 assert (i != 0);

Você agora está no corpo da função reciprocal.

## Sistema de controle de versões: CVS – Concurrent Versions System

Tirado de http://twiki.im.ufba.br/bin/view/GAVRI/TutorialCVS

O CVS (*Concurrent Versions System*) permite que se trabalhe com diversas versões de arquivos organizados em um diretório e localizados local ou remotamente, mantendo-se suas versões antigas e os logs de quem e quando manipulou os arquivos.

A exemplo de outros softwares usados neste projeto, o CVS pode ser baixado gratuitamente e tem o seu código aberto.

Instalação e configuração do servidor CVS:

O CVS possui alguns métodos para acessar o servidor CVS e que devem ser configurados durante a instalação do CVS na máquina servidora.

O CVS pode ser baixado no seu sítio oficial citado acima.

Para pacotes Debian basta apenas executar o comando:

```
# apt-get install cvs
```

Assim basta seguir as telas de configuração para ter o pacote CVS instalado e (opcionalmente) com o servidor sendo executado. Você poderá a qualquer momento reconfigurar o CVS executando:

```
# dpkg-reconfigure cvs
```

Uma boa documentação de referência é encontrada no pacote **cvs-doc**.

Configurando métodos de acesso ao repositório

O CVS é uma aplicação cliente/servidor, possuindo diversas maneiras de fazer o acesso seu repositório. Estes métodos são os seguintes:

Método **local**:

Acessa o diretório do repositório diretamente no disco local. A vantagem deste método é que não é requerido nem nome nem senha para acesso (você precisa apenas ter permissões para acesso aos arquivos que deseja trabalhar) e também não é preciso nenhuma conexão de rede. Este método é ideal para trabalhar na máquina local ou com os arquivos administrativos do CVS existentes no diretório CVSROOT do repositório. É muito útil também para configurar outros métodos de acesso, como o pserver.

Para utilizar o método de acesso local, basta definir a variável CVSROOT da seguinte forma (assumindo que o repositório esteja instalado em /var/lib/cvs, padrão para o Debian):

```
$ export CVSROOT=/var/lib/cvs
```

ou

```
$ export CVSROOT=local:/var/lib/cvs
```

Depois disso, basta utilizar os comandos normais do cvs sem precisar se autenticar no sistema.

Método **ext**:

Este método de acesso lhe permite especificar um programa externo que será usado para fazer uma conexão remota com o servidor cvs.Este programa é definido na variável CVS_RSH e caso não ela seja especificada o padrão é rsh.

Este método requer que o usuário possua um login/senha no banco de dados de autenticação /etc/passwd do servidor de destino. Suas permissões de acesso ao CVS (leitura/gravação) serão as mesmas definidas neste arquivo.

Defina a variável CVSROOT da seguinte forma para utilizar este método de acesso:

```
$ export CVSROOT=:ext:<usuario>@<servidor>:caminho_do_repositorio

$ cvs login
```

O caminho padrão do repositório no Debian é /var/lib/cvs, mas nada impede que o usuário crie um repositório em qualquer outra pasta do seu interesse. Entretanto algumas funcionalidades podem ser perdidas com o procedimento de colocar o repositório em outras pastas. Por exemplo, existe um pacote chamado **cvsweb** que disponibiliza uma interface web para os repositórios da máquina. Caso alguém use outro local como /home/

O método de acesso "ext" usa uma conexão segura que pode ser por **ssh** (Secure Shell) ou **rsh** (Remote Shell). O uso mais freqüente do ext é para conexões seguras feitas via ssh, e a configuração é feita da seguinte forma:

```
$ export CVS_RSH=ssh

$ export CVSROOT=:ext:<usuario>@<servidor>:caminho_do_repositorio

$ cvs checkout
```

O acesso via ssh traz a vantagem de que as senhas trafegarão de forma segura via rede, não sendo facilmente capturadas por sniffers e outros programas de monitoração que possam estar instalados na rota entre você e o servidor.

O acesso de leitura/gravação do usuário, é definido de acordo com as permissões deste usuário no sistema. Uma maneira recomendada é definir um grupo que terá acesso a gravação no CVS e adicionar usuários que possam fazer gravação neste grupo para um determinado projeto ou repositório, como por exemplo:

```
# chown root:cvsadmin /var/lib/cvs

# chmod 2775 /var/lib/cvs
```

Desta forma todos os usuários terão acesso de leitura/escrita ao repositório local, o que pode não ser desejado. Usando o ext é recomendado a criação de mais grupos por

projeto para que usuários de outros projetos não deletem ou alterem arquivos dos projetos que não lhe envolvem.

Método **pserver** (password server):

Este é um método de acesso remoto que utiliza um banco de dados de usuários senhas para acesso ao repositório. A diferença em relação ao método de acesso ext é que o pserver roda através de um servidor próprio na porta 2401. O acesso dos usuários (leitura/gravação) no repositório pode ser feita tanto através do banco de dados de usuários do sistema (/etc/passwd) como através de um banco de dados separado por repositório.

A grande vantagem deste segundo método é que cada projeto poderá ter membros com acessos diferenciados; o membro x poderá ter acesso ao projeto sgml mas não ao projeto focalinux; ou o usuário y poderá ter acesso de gravação (para trabalhar no projeto focalinux) mas somente acesso de leitura ao projeto sgml.

Este é o método de acesso preferido para a criação de usuários anônimos (uma vez que o administrador de um servidor que hospede muitos projetos não vai querer abrir um acesso anônimo via ext para todos os projetos).

Também existe a vantagem que novos membros do projeto e tarefas administrativas são feitas por qualquer pessoa que possua acesso de gravação aos arquivos do repositório.

Para usar o CVS configurado pelo pserver basta na definição da variável CVSROOT colocar:

```
$ export CVSROOT=:pserver:<usuario>@<servidor>:caminho_do_respositorio

$ cvs login

$ cvs checkout

$ cvs ...
```

Perceba que pelo método **ext** não era preciso fazer "cvs login", mas pelo **pserver** é necessário, antes de qualquer outro comando CVS.

A configuração do servidor para este método pode ser selecionada executando o comando abaixo e selecionando SIM para a opção de pserver.

```
# dpkg-reconfigure cvs
```

Esta forma de acesso armazena os usuários em um banco de dados próprio, não requerendo a criação de contas locais no arquivo /etc/passwd. Para criar um servidor deste tipo siga os seguintes procedimentos:

1. Exporte a variável CVSROOT apontando para o repositório que deseja configurar o acesso. Como isto é uma configuração administrativa, assuma o método de acesso local sendo usada pelo usuário administrador do servidor "export CVSROOT=/var/lib/cvs".

2. Entre no diretório definido no CVSROOT e entre num subdiretório existente chamado CVSROOT, os arquivos de configuração do repositório se encontram lá.
3. Edite o arquivo **config** e mude a variável **SystemAuth** para **no**. Isto diz ao servidor pserver não usar os arquivos de autenticação do sistema, mas a invés disso usar seu banco de dados próprio.
4. Crie um arquivo passwd no diretório CVSROOT o formato deste arquivo é:

usuario:senha:usuario_local

Exemplo:

gleydsonm:K32dk1234k:cvsuser

anonymous::pooruser

Onde:

- usuario: Nome da conta de usuário que fará acesso ao CVS.
- senha: Senha que será usada pelo usuário. Ela deverá ser criptografada usando o algoritmo crypt. O comando mkpasswd senha pode ser usado para gerar a senha criptografada. Caso este campo seja deixado em branco, nenhuma senha de usuário será utilizada. O utilitário mkpasswd está presente no pacote whois na Debian.
- usuario_local: Usuário local que terá suas permissões mapeadas ao usuário do CVS. Como a conta de usuário do cvs não existe no sistema, é necessário que o sistema tenha uma maneira de saber que nível de acesso este usuário terá. Caso não crie este usuário ou ele seja inválido, você terá erros do tipo ": no such user" no momento que fizer o "cvs login".

Uma forma segura de se fazer isto, é criar uma conta de usuário **somente** com acesso aos arquivos do CVS, sem shell e senha. Isto permitirá mapear a UID/GID do usuário criado com o acesso do CVS sem comprometer a segurança do sistema de arquivos. Isto pode ser feito através do seguinte comando:

```
# adduser --disabled-password --disabled-login usuario
```

É necessário especificar um diretório home do usuário, pois o servidor cvs precisa ter acesso ao arquivo /home/

Certifique-se que o usuário local possui permissões de gravação no diretório do CVS, caso contrário ele não poderá fazer commits. Lembre-se que as permissões de leitura/gravação do usuário serão controladas através de arquivos do próprio pserver, mas também é necessária a permissão de gravação do usuário no repositório. Isto poderá ser feito através de grupos de sistema e garante uma dupla camada de segurança.

1. Para dar direito de leitura ao repositório, crie um arquivo chamado readers e adicione os nomes de usuários que terão acesso ao repositório (um por linha). O nome que deverá ser usado é o nome do usuário de CVS e não do sistema,

segundo o exemplo:

gleydsonm

anonymous

1. Para dar direito de gravação ao repositório, crie um arquivo chamado writers. Seu formato é idêntico ao arquivo readers.
2. Pronto, o acesso a CVS usando um banco de dados próprio está pronto! basta dar o commit nos arquivos, adicionar os arquivos readers, writers e passwd no repositório para o servidor de CVS para te-lo funcionando. Note que em versões mais novas do CVS, não é possível transferir o arquivo passwd via rede, então será necessário cria-lo manualmente dentro do repositório do servidor.

O método de acesso do CVS aos arquivos readers e writers é restritiva, portanto se um nome de usuário existir no arquivo readers e writers o que valerá será o menor nível de acesso. Vendo os exemplos acima, os usuários gleydsonm e anonymous terão somente acesso a leitura do repositório e outros, que não eles, se estiverem no arquivo writers terão acesso de leitura e gravação.

Alguns conceitos:

- **Projeto**: conjunto de fontes ou documentos, organizados de maneira hierárquica.
- **Repositório**: conjunto lógico de projetos.

Iniciando o uso do CVS

Crie o repositório

```
$ mkdir diretorio_do_projeto/

$ cd diretorio

$ cvs init
```

ou

```
$ cd diretorio_onde_projeto_vai_ficar

$ cvs init nome_projeto

$ ls nome_projeto
```

Exporte a variável de ambiente CVSROOT:

```
$ export CVSROOT=diretorio_do_projeto/
```

ou

```
$ export CVSROOT=<usuario>@<maquina>:diretorio_do_projeto/
```

Ou usa a opção -d do comando "cvs":

```
$ cvs -d [<usuario>@<maquina>:]diretorio_do_projeto/
```

## Importando os fontes

Caso os fontes do projeto já existam (antes de se criar o repositório CVS), importe seus fontes:
Entre no diretório onde se encontra seu projeto e digite

```
$ cvs import -m "Breve Descrição" nome_do_projeto desenvolvedor
```

Caso não seja utilizada a opção -m para descrever o projeto, o CVS abrirá um editor de texto para que isso seja feito.

## Pegando os arquivos (co = *checkout*)

```
$ cvs co diretorio_do_projeto
```

ou

```
$ cvs co nome_do_projeto
```

Isso criará o diretório com o nome do repositório localmente.

Se mais de um usuário forem trabalhar no mesmo projeto ao mesmo tempo, cada um deve rodar esse comando em um diretório distinto.

Se for um diretório num servidor remoto:

```
$ cvs -d usuario@maquina:diretorio_do_projeto co diretorio_do_projeto
```

## Pondo arquivos

Caso seja arquivo novo

```
$ cvs add arquivo_novo
$ cvs update
```

**Muito importante, ainda mais se tratando que mais de uma pessoa tá mexendo!**

```
$ cvs commit
```

Removendo arquivos

```
$ cvs remove arquivo
$ rm arquivo
$ cvs commit
```

Caso o projeto seja modificado

```
$ cvs update
$ cvs commit
```

Para manter o repositório atualizado. **Nunca esquecer** o *cvs update* e o *cvs commit* !!!

Checando alterações no projeto

Para acompanhar, durante o desenvolvimento, a versão do arquivo que está sofrendo alteração, o CVS dispõe de um *diff* . Assim, o comando:

```
$ cvs diff
```

verifica e mostra quais linhas de todos os arquivos do projeto sofreram alteração.

Para checar as alterações de um só arquivo:

```
$ cvs diff arquivo
```

Observação final

O CVS realiza um controle muito eficiente de versões e é capaz, inclusive, de realizar alterações no mesmo arquivo, caso duas pessoas editem o mesmo arquivo ao mesmo tempo.

Se houver conflito, entretanto, o sistema deixará a decisão para ser tomada pelos humanos.

Será dado um aviso que o mesmo ponto do arquivo foi alterado, com um log do que foi feito por cada usuário, e alguém vai ter que usar o dicernimento e escolher qual das duas alterações deverá ser efetivada.

## 8. Bibliografia

[1] R. Stone, N. Matthew, *Beginning Linux Programming*, 2nd Edition, Wrox, 2001

[2] C. Negus, *Linux Bible*, 2005 Edition, Wiley Publishing, 2005

[3] E. Nemeth, G. Snyder, S. Seebass, T. R. Hein, *Manual de Administração do Sistema UNIX*, 3a edição, Bookman, 2002

[4] M. Mitchell, J. Oldham, A. Samuel, *Advanced Linux Programming*, 1st Edition, New Riders Publishing, 2001