Tópicos Especiais em Fundamentos da Computação
Shell Scripting

Sistema de Arquivos

Andrei Rimsa Álvares
andrei@cefetmg.br

Sumário

- Sistema de arquivos
- Trabalhando com diretórios
- Arquivos e diretórios padrões
- Permissões
- Mais comandos

SISTEMA DE ARQUIVOS

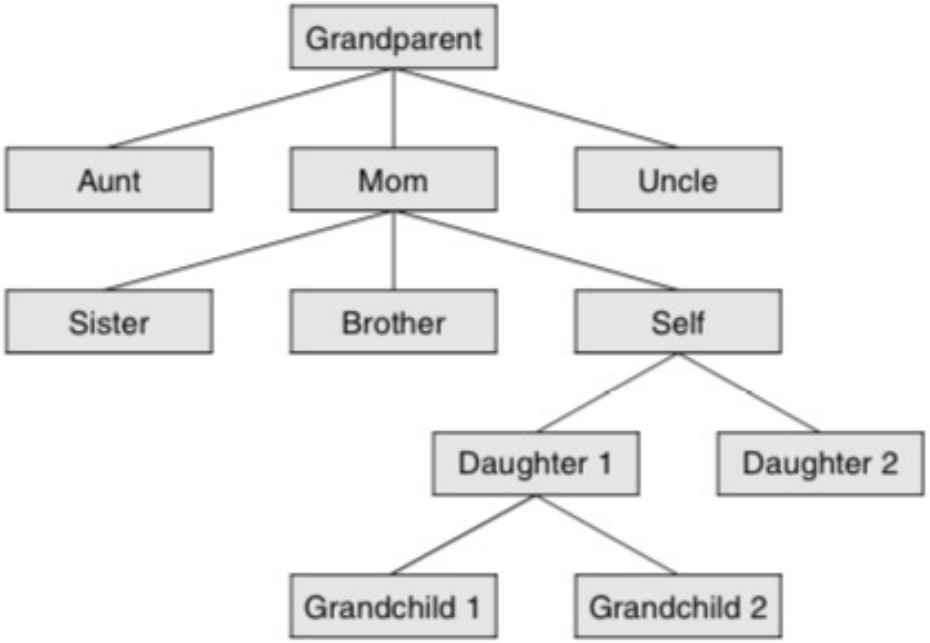
Shell Scripting

Sistema de Arquivos

- Um **sistema de arquivos** é um conjunto de estruturas de dados que tipicamente residem em parte de um disco e armazena diretório de arquivos

- Sistemas de arquivos guardam dados de usuários e do sistema que são a base do trabalho do usuário no sistema e da própria existência do sistema

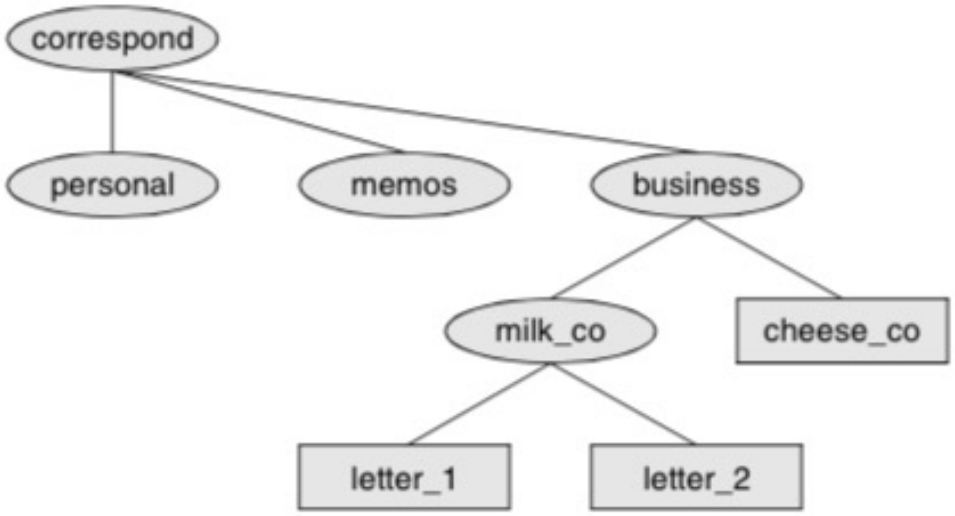
Estrutura Hierárquica



Árvore Genealógica

- Uma estrutura hierárquica é uma árvore que tem, normalmente, um formato de pirâmide
- Por exemplo, a da imagem: um casal tem um filho que por sua vez pode ter mais filhos...
- O sistema de arquivos usa uma estrutura similar, com um conjunto de arquivos conectados
  - Permite encontrar arquivos rapidamente

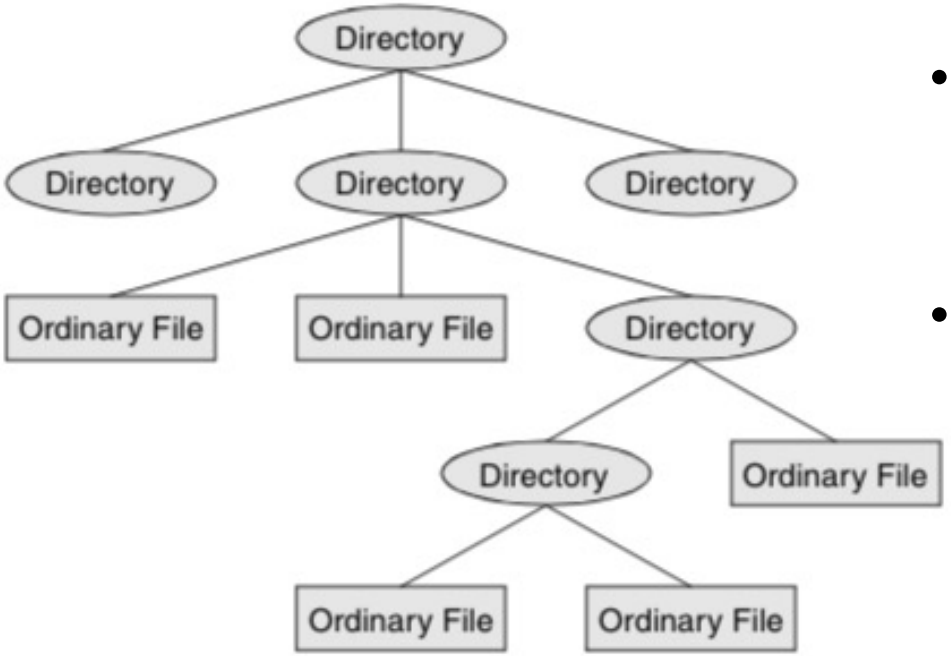
Sistema de Arquivo Hierárquico



- Um usuário começa em um diretório e pode adicionar mais diretórios nele em qualquer nível
  - Ao criar múltiplos níveis de subdiretórios, pode-se expandir a estrutura conforme sua necessidade
- Por exemplo, a imagem mostra um subdiretório de uma secretaria, chamado *correspond*

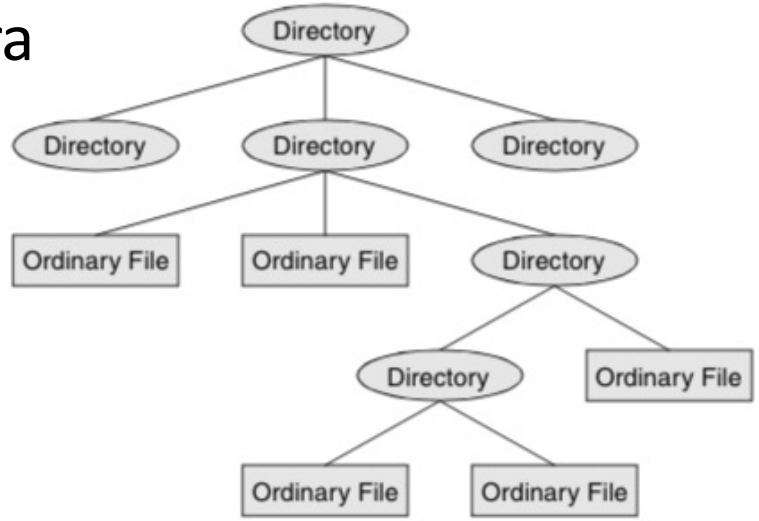
Diretórios e Arquivos Ordinais

- Como em uma hierarquia familiar, o sistema de arquivos é uma árvore com um diretório como raiz que *cresce* para baixo, com caminhos conectando diretórios e arquivos
  - No final de cada caminho tem um **arquivo ordinário**, um **arquivo de diretório** ou um **arquivo especial** (de sistema)



- **Arquivos ordinários** ou **especiais** não podem gerar novos caminhos
- **Arquivos de diretórios** (diretórios ou pastas) são pontos onde novos caminhos podem ser formados

Nomenclatura



- Ao se referir a uma árvore
  - **Acima**: em direção à raiz
  - **Abaixo**: afastado da raiz
- Diretórios diretamente conectados por um caminho são chamados
  - **Pais**: próximos da raiz
  - **Filhos**: afastados da raiz
- Um **caminho de arquivo** (*pathname*) é uma série de nomes que criam um traço do caminho passando pelos ramos de um arquivo a outro

Nomes de Arquivos

- Todo arquivo possui um **nome** (*nome de arquivo*)
  - O tamanho máximo do nome do arquivo varia com o tipo de sistema de arquivo, mas tipicamente aceita até `255` caracteres

- Embora seja possível usar praticamente quaisquer caracteres em um nome de arquivo, deve-se evitar confusões usando somente
  - Letras maiúsculas (`A-Z`), minúsculas (`a-z`), números (`0-9`), sublinhado (`_`), ponto (`.`), vírgula (`,`)

Cuidado: dois arquivos em um **mesmo diretório** não podem ter o mesmo nome

Arquivos em diferentes diretórios podem ter o mesmo nome

Dica: escolha nomes que signifiquem alguma coisa para o usuário

Espaços em Branco

- Embora seja possível usar espaços em branco em nomes de arquivo, normalmente isso não é uma boa ideia
  - Como esse é um caractere especial, é preciso colocar entre aspas ou escapar com um outro caractere especial

- Use pontos ou sublinhados ao invés de espaços sempre que possível
  - Ex.: `joe.05.04.26`, `new_stuff`

- Para arquivos que incluem espaço no nome deve-se usar uma das seguintes alternativas

```
$ lpr my\ file
```

```
$ lpr "my file"
```

O Diretório Raiz (/)

- O diretório raiz da hierarquia do sistema de arquivo não possui propriamente um nome
  - É referenciado como o diretório raiz
  - É representado por uma barra (/) sozinha no lado esquerdo de um caminho de arquivo

Caminho Absoluto

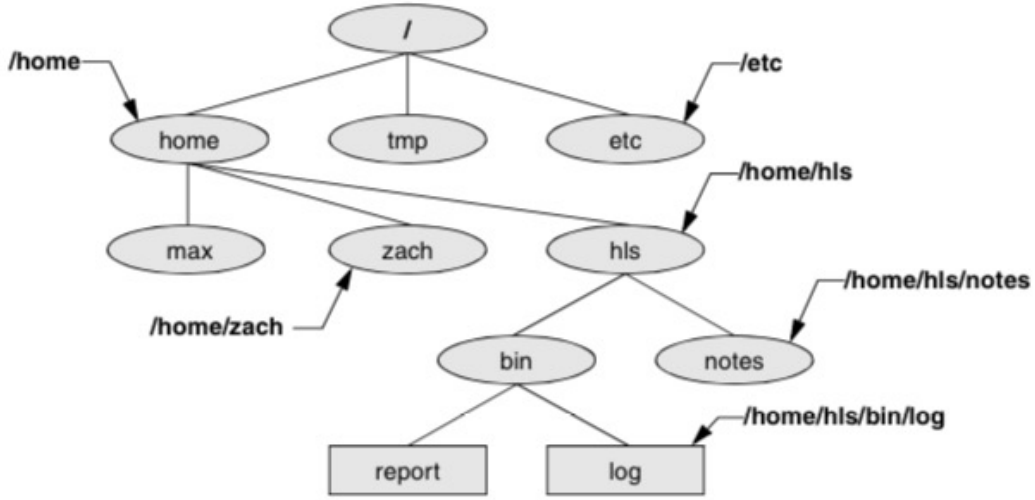
- Um **caminho absoluto** começa com uma barra (/) que representa o diretório raiz
  - A barra é seguida pelo nome do arquivo localizado no diretório raiz
  - O caminho absoluto continua, traçando um caminho através de vários diretórios intermediários até determinado arquivo

- Os arquivos podem ser encadeados juntos no caminho, em que cada diretório é seguido de barra (/)

- Tipicamente um caminho absoluto para um diretório não inclui a barra no final, embora esse formato pode ser usado para enfatizar que o caminho é um diretório
  - Ex.: `/home/zach/`

Caminho Absoluto

- Exemplo de caminhos de arquivos absolutos de diretórios e arquivos ordinários como parte de uma hierarquia de diretórios



- Usando um caminho absoluto, pode-se listar ou trabalhar com qualquer arquivo no sistema (se tiver permissões), independente do diretório de trabalho atual

```
$ pwd
/home/sam
$ ls /usr/bin
7z
...
```

Til (~) no Caminho

- Uma outra forma de caminho absoluto é usar um til seguido de uma barra (`~/`) no começo do caminho do arquivo para expandir para o diretório pessoal do usuário
  - Por exemplo, pode-se usar esse atalho para visualizar o arquivo de inicialização `.bashrc`, independente do diretório de trabalho atual

```
$ less ~/.bashrc
...
```

- Um til seguido de um nome de usuário no começo do caminho expande para o diretório pessoal desse usuário
  - Por exemplo, pode-se examinar o arquivo `.bashrc` do usuário `sam` (se tiver permissão para isso)

```
$ less ~sam/.bashrc
..
```

Caminho Relativo

- Um **caminho de arquivo relativo** traça um caminho a partir do diretório de trabalho atual até um arquivo; ou seja, o caminho é relativo ao diretório de trabalho

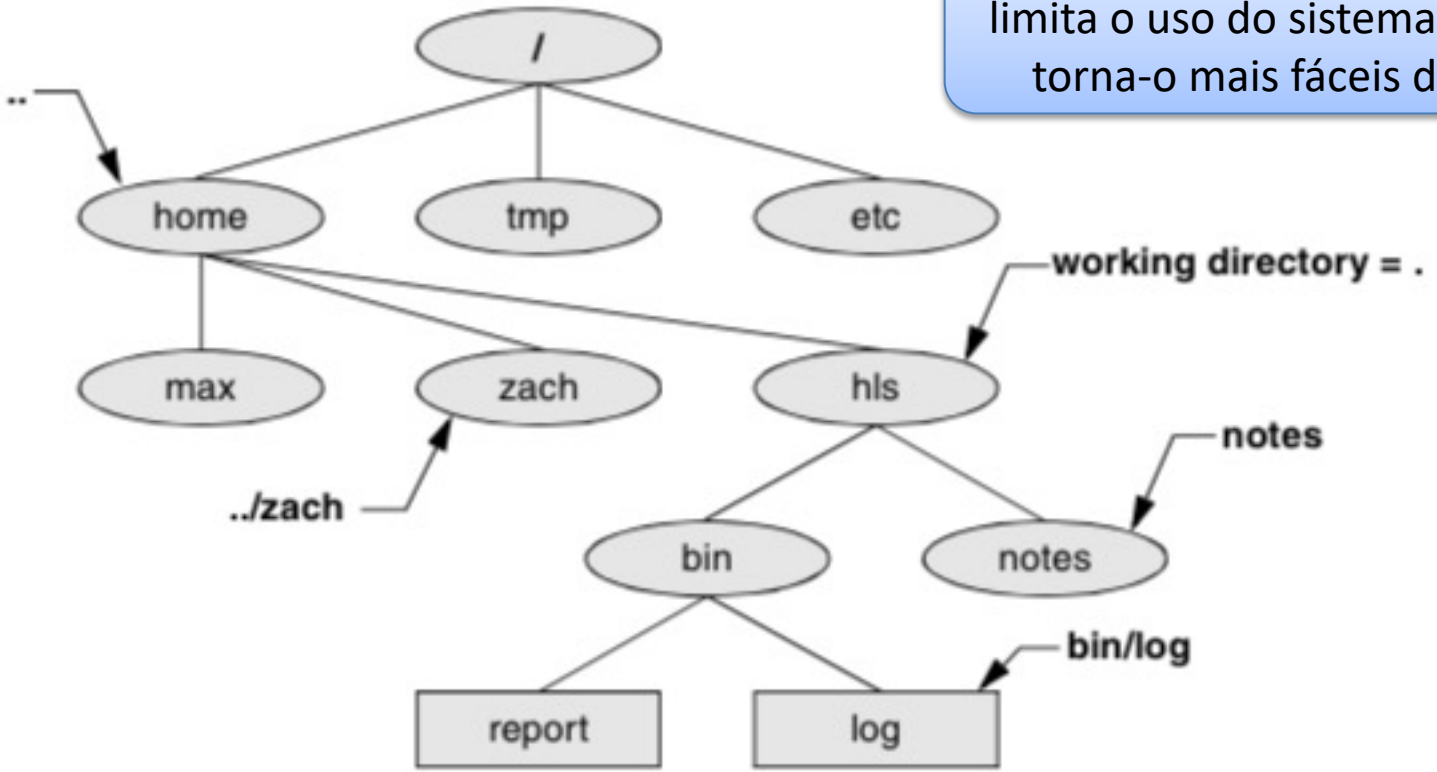
- Qualquer diretório que não começa do diretório raiz (/) ou por um til (~) é um caminho relativo

- Assim como caminhos absolutos, caminhos relativos podem ter um caminho com vários diretórios

Caminho Relativo

- Digitar um longo caminho pode ser tedioso e aumentar a chance de errar; pode-se usar o diretório de trabalho atual para reduzir a necessidade de usar longos caminhos

A escolha do diretório atual não limita o uso do sistema, apenas torna-o mais fáceis de usar



TRABALHANDO COM DIRETÓRIOS

Shell Scripting

Trabalhando com Diretórios

- Operações que podem ser feitas com diretórios
  - Criar diretórios (`mkdir`)
  - Alterar diretórios de trabalho atual (`cd`)
  - Remover diretórios (`rmdir`)
  - Mover (`mv`) ou copiar (`cp`) arquivos
  - Mover (`mv`) diretórios

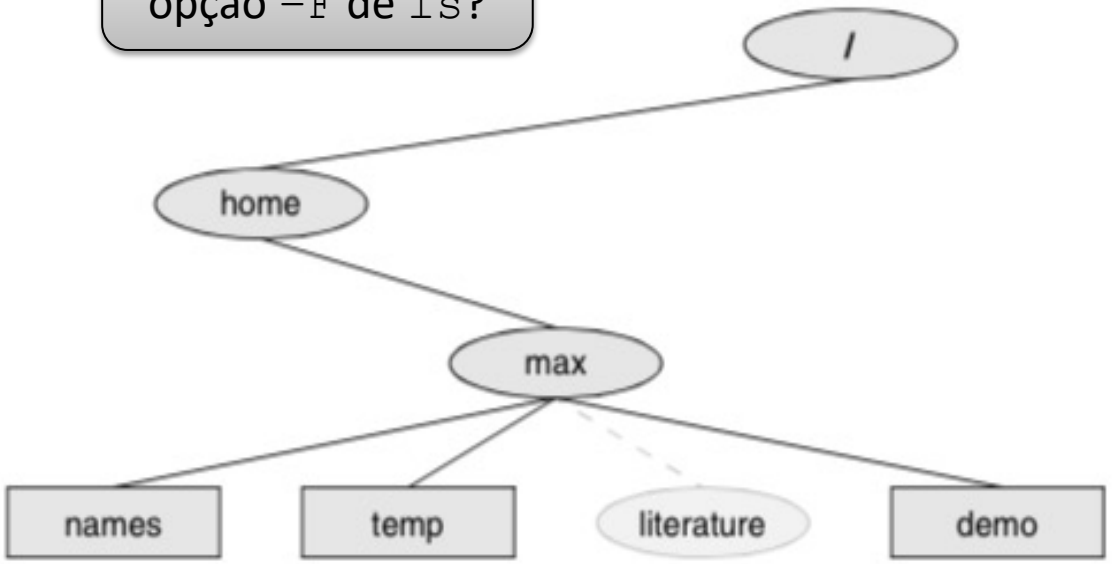
Criar Diretórios (`mkdir`)

- A ferramenta `mkdir` (*make directory*) cria um novo diretório, tal que o argumento é o caminho do novo diretório

- O exemplo a seguir desenvolve a estrutura de diretórios da figura

```
$ pwd
/home/max
$ ls
demo names temp
$ mkdir literature
$ ls -F
demo literature/
names temp
$ ls literature
$
```

Para que serve a opção `-F` de `ls`?



Criar Diretórios (`mkdir`)

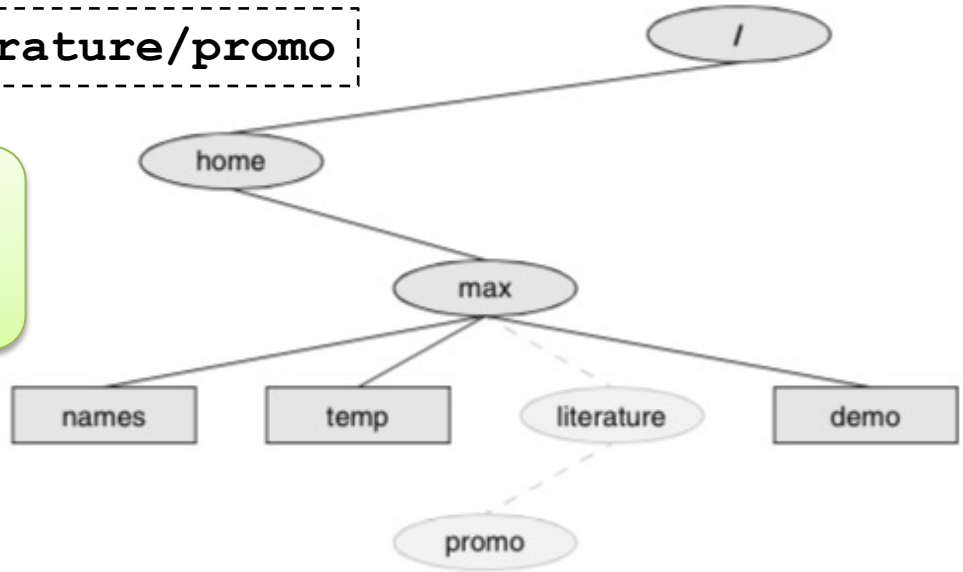
- Para criar o diretório `promo` pode-se usar
  - Caminho relativo

```
$ pwd
/home/max
$ mkdir literature/promo
```

  - Caminho absoluto

```
$ mkdir /home/max/literature/promo
```

Dica: pode-se usar o comando a seguir para criar os dois diretórios simultaneamente:
`$ mkdir –p literature/promo`



Alterar Diretórios de Trabalho Atual (`cd`)

- A ferramenta `cd` (*change directory*) faz com que outro diretório seja o diretório de trabalho

- Exemplo alterando o diretório de trabalho atual (`/home/max`), verificado pelo comando `pwd`
  - Usando caminho relativo

```
$ cd literature
$ pwd
/home/max/literature
```

  - Usando caminho absoluto

```
$ cd /home/max/literature
$ pwd
/home/max/literature
```

Diretórios `.` (ponto) e `..` (dois pontos)

- A ferramenta `mkdir` coloca automaticamente duas entradas em cada diretório criado
  - Um ponto único (`.`): sinônimo do diretório de trabalho atual
  - Um ponto duplo (`..`): sinônimo do diretório antecessor ao atual

- Os três exemplos a seguir mostram o uso do diretório anterior
  - Listar o conteúdo do diretório anterior
  - Copiar o arquivo `memoA` do diretório atual para o diretório anterior
  - Listar o conteúdo do diretório anterior novamente

```
$ pwd
/home/max/literature
$ ls ..
demo literature names temp
$ cp memoA ..
$ ls ..
demo literature memoA names temp
```

Remover Diretórios (`rmdir`)

- A ferramenta `rmdir` (*remove directory*) remove um diretório
  - Mas não se pode remover o diretório de trabalho `ou` diretórios que contenham arquivos (exceto as entradas `.` e ..)
  - Se precisar remover um diretório que contém arquivos, remova-os primeiro usando o comando `rm` para depois remover o diretório

- Não é preciso (nem é possível) remover as entradas `.` e `..`, `rmdir` os remove automaticamente

- Por exemplo, para remover o diretório `promo`, pode-se usar

```
$ rmdir /home/max/literature/promo
```

Dica: `rm` possui a opção `-r` que recursivamente remove arquivos (incluindo diretório) e também o próprio diretório

Mover Arquivos (mv)

- A ferramenta `mv` (*move*) é usada para mover arquivos de um diretório para outro (alterar o caminho do arquivo) como também para alterar nomes de arquivos simples (renomear)

- Quando usado para mover um ou mais arquivos para um novo diretório, têm-se a seguinte sintaxe

```
mv existing-file-list directory
```

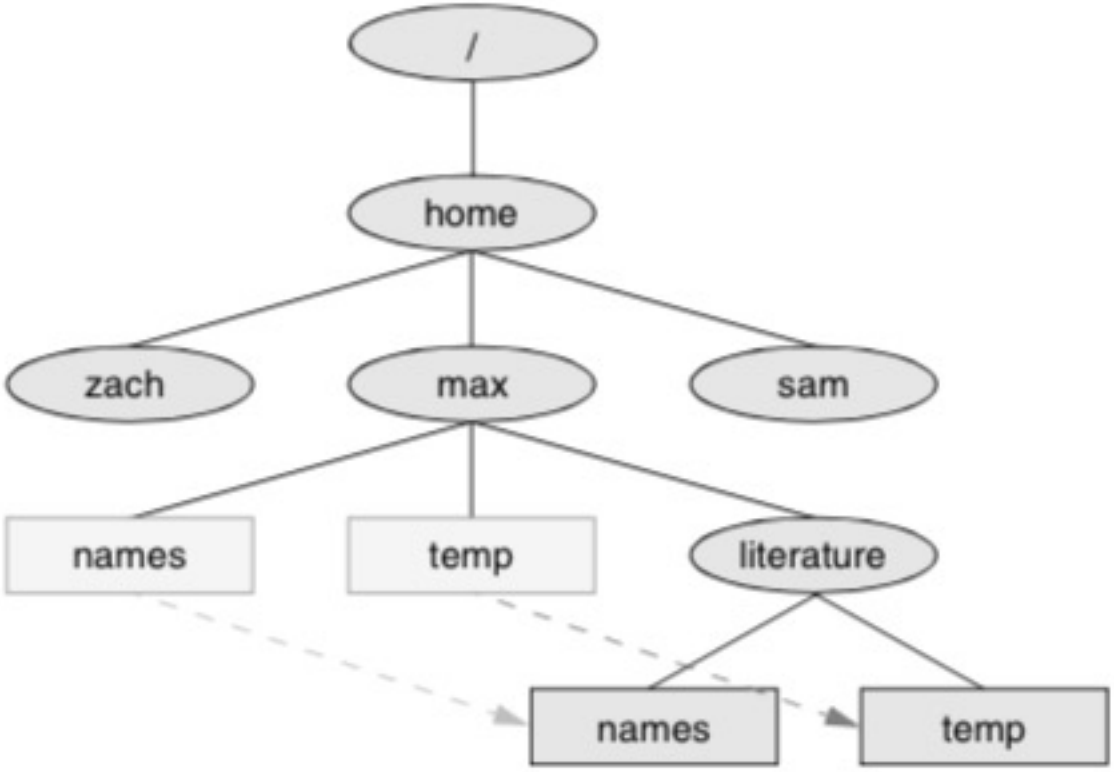
- Por exemplo, se o diretório de trabalho for `/home/max`, pode-se usar o seguinte comando para mover os arquivos `names` e `temp` para o diretório `literature`

```
$ mv names temp literature
```

O comando `cp` funciona de forma similar ao `mv`, mas copia ao invés de mover

Mover Arquivos (`mv`)

- O comando anterior altera o caminho absoluto de `/home/max/names` e `/home/max/temp` para `/home/max/literature/names` e `/home/max/literature/temp`, respectivamente



Mover Diretórios (`mv`)

- Assim como `mv` é capaz de mover arquivos ordinários de um diretório para outro, `mv` também pode ser usado para mover diretórios

- A sintaxe é similar a anterior, exceto que deve-se especificar um ou mais diretórios, não arquivos ordinários, para mover

```
mv existing-directory-list new-directory
```

  - Se o novo diretório (`new-directory`) não existir, a lista de diretórios (`existing-directory-list`) deve conter apenas um diretório que é renomeado para o novo nome (`new-directory`)

Cuidado: para se copiar o conteúdo de um diretório, deve-se usar o comando `cp` com a opção `-r` (recursiva)

Criar *Links* (`ln`)

- Um *link* é um ponteiro para um arquivo
  - Cada vez que se cria um arquivo adiciona-se um ponteiro em um diretório que associa o nome do arquivo com seu lugar no disco

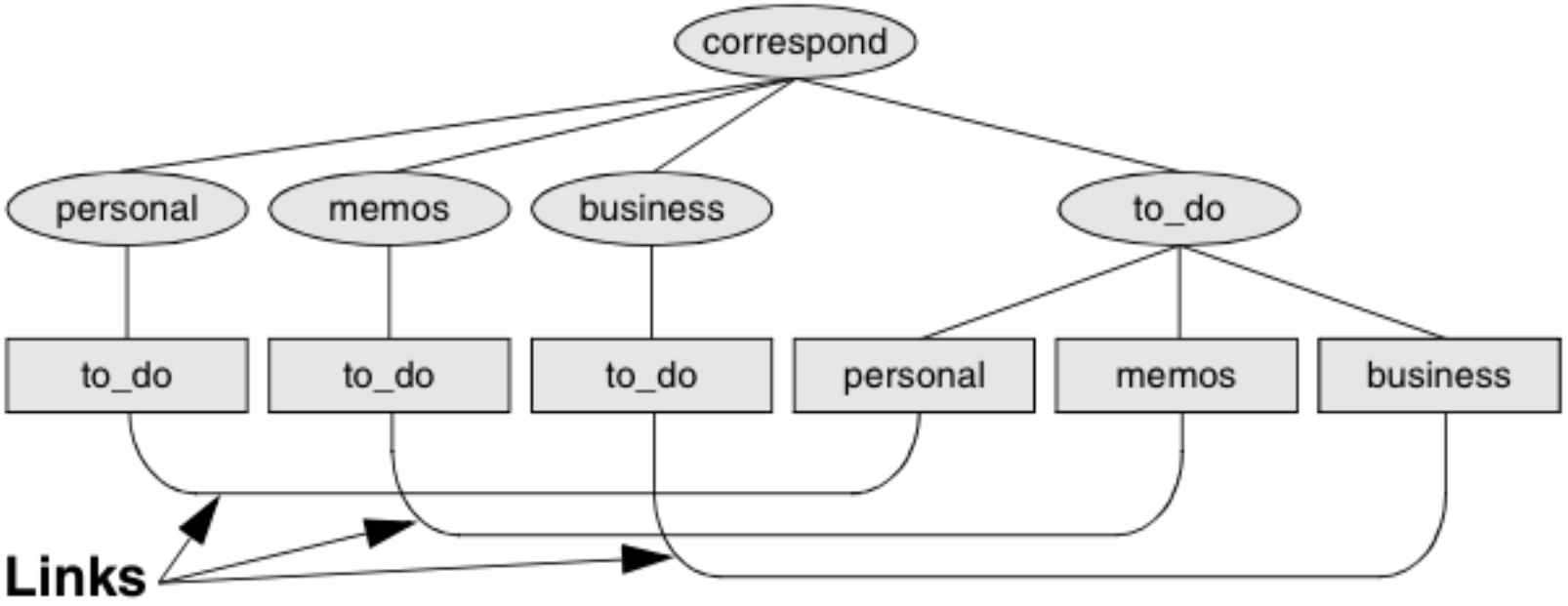
- Compartilhar arquivos pode ser útil quando duas ou mais pessoas estão trabalhando em um mesmo projeto e precisam compartilhar algumas informações
  - Pode-se facilitar o acesso a seus arquivos por outros usuários usando ***links***

Pode ser que precise de permissões nos arquivos e diretórios para que isso seja possível

Criar *Links* (`ln`)

- *Links* podem ser úteis até para um mesmo usuário com uma grande hierarquia de diretórios

- Por exemplo



Criar *Links* (`ln`)

- Existem dois tipos de *links*

Hard links

Symbolic (soft) links

Hard Links

- Um ***hard link*** para um arquivo aparece como outro arquivo
  - Se o arquivo estiver no mesmo diretório que o *link*, o *link* deve ter um outro nome de arquivo (dois arquivos no mesmo diretório não podem ter o mesmo nome)

- Restrição: pode-se apenas criar *hard links* para arquivos no mesmo sistema de arquivo que armazena o arquivo

- A ferramenta `ln` (*link*) cria um *hard link* (se usado sem a opção `-s` ou `--symbolic`) para um arquivo existente usando a seguinte sintaxe

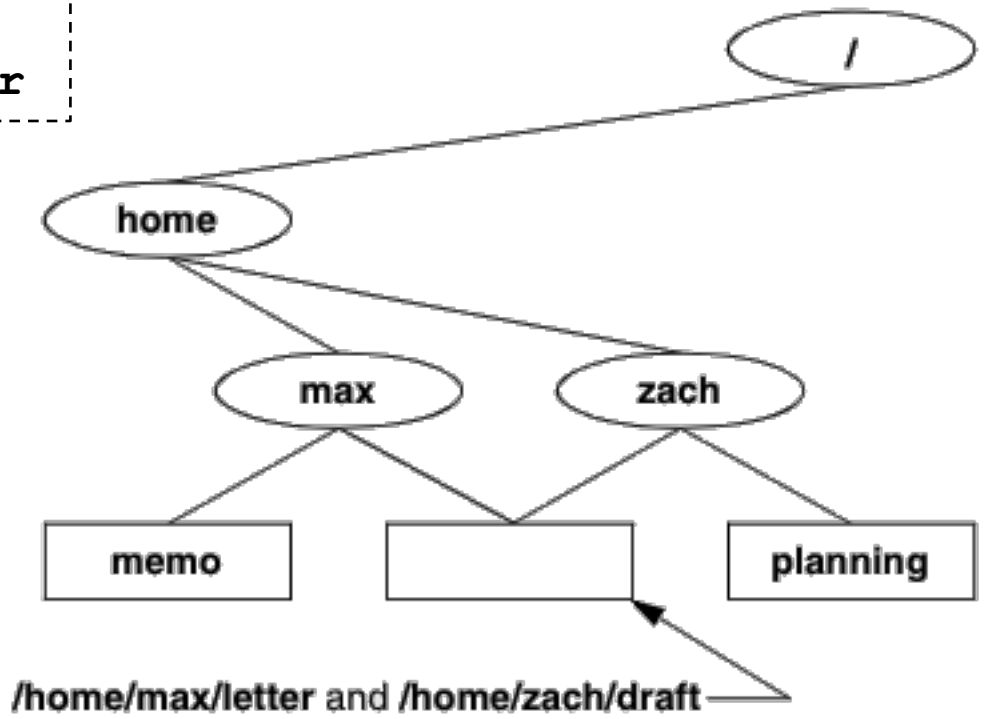
```
ln existing-file new-link
```

Hard Links

- Por exemplo, o comando a seguir cria um *hard link* chamado `letter` no diretório pessoal de `max` para um arquivo já existente `draft` no diretório pessoal de `zach`

```
$ pwd
/home/zach
$ ln draft /home/max/letter
```

Cuidado: *hard link* (`ln`) é diferente de copiar arquivo (`cp`)



Symbolic Links

- Além de *hard links*, sistemas Unix suportam ***links* simbólicos** (também chamados de *soft links* ou *symlinks*)
  - Um ***hard link*** é um ponteiro para um arquivo (a entrada no diretório aponta para um *inode*)
  - Um ***symbolic link*** é um ponteiro indireto para um arquivo (a entrada no diretório contém um caminho de arquivo do arquivo apontado)

- *Symbolic links* foram desenvolvidos por causa das limitações de *hard links*
  - Não se pode criar um *hard link* para um diretório, mas pode-se criar um *symbolic link*

Symbolic Links

- Como cada sistema de arquivos mantém as informações de controle separadas para seus arquivos
  - Não é possível criar *hard links* entre arquivos de diferentes sistemas de arquivos
  - Mas é possível criar um *link* simbólico, já que podem apontar para qualquer arquivo, independentemente de onde está localizado na estrutura de arquivos

Symbolic Links

- A maior vantagem de *links* simbólicos é que eles podem apontar para arquivos inexistentes; tal habilidade é útil quando se tem um *link* para um arquivo que é periodicamente removido e recriado
  - Um *hard link* continua apontando para o arquivo "removido" (que na verdade é mantido vivo mesmo após a criação do novo arquivo)
  - Um *link* simbólico sempre aponta para o novo arquivo criado e não interfere na remoção do arquivo antigo

Symbolic Links

- Embora sejam mais gerais que *hard links*, *links* simbólicos possuem desvantagens; *links* simbólicos não possuem o mesmo status de *hard links*
  - Quando um arquivo possui múltiplos *hard links* é análogo a uma pessoa com múltiplos nomes legais
  - Quando um arquivo possui múltiplos *links* simbólicos é análogo a ter vários apelidos; uma pessoa pode ter vários apelidos, mas possuem menos importância que nomes legais

Symbolic Links

- A ferramenta `ln` com a opção `-s` (ou `--symbolic`) cria um *symbolic link*

```
ln -s existing-file new-link
```

- Por exemplo

```
$ ln -s /home/max/sum /tmp/s3
$ ls -l /home/max/sum /tmp/s3
-rw-rw-r-- 1 max max 38 Jun 12 09:51 /home/max/sum
lrwxrwxrwx 1 max max 14 Jun 12 09:52 /tmp/s3 -> /home/max/sum
$ cat /tmp/s3
this is sum
```

Repare que os tamanhos e última data de modificação são diferentes para os arquivos; não mantêm informações de status do arquivo

Symbolic Links

- *Symbolic links* são literais e não tem ciência de diretórios
  - Um *link* que aponta para um caminho relativo, incluindo um nome de arquivo simples, assume o caminho relativo do diretório onde o *link* foi criado (não do diretório de onde é o *link*)

- Por exemplo, o link aponta para um arquivo chamado sum no diretório `/tmp` que não existe

Dica: dê preferência para links com caminho completo

```
$ pwd
/home/max
$ ln -s sum /tmp/s4
$ ls -l sum /tmp/s4
lrwxrwxrwx 1 max max 3 Jun 12 10:13 /tmp/s4 -> sum
-rw-rw-r-- 1 max max38 Jun 12 09:51 sum
$ cat /tmp/s4
cat: /tmp/s4: No such file or directory
```

ARQUIVOS E DIRETÓRIOS PADRÕES

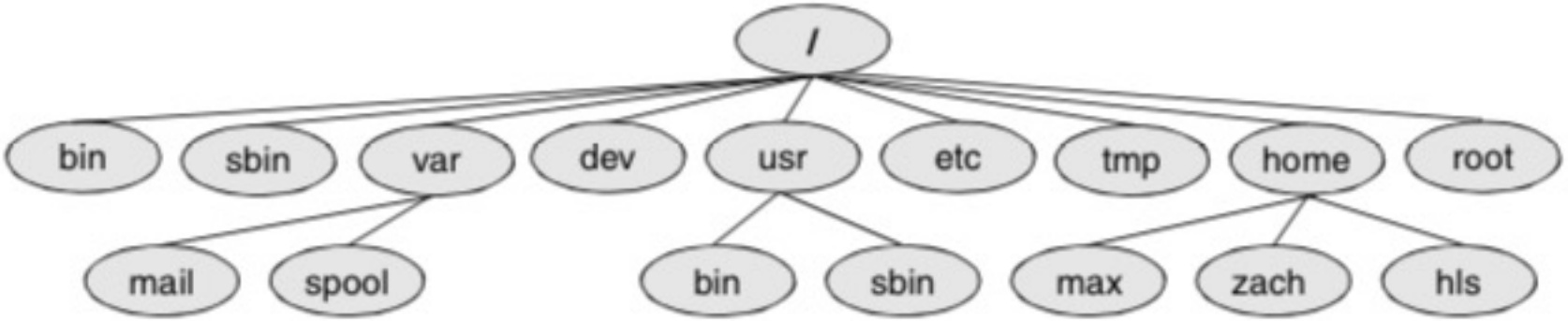
Shell Scripting

Padronização

- Arquivos em um sistema Linux não ficavam originalmente localizados nos lugares padrões na hierarquia de diretórios tornando difícil a documentação e manutenção do sistema

> Nessa época era praticamente impossível para lançar um pacote que compilaria e executaria em todos os sistemas Linux

- A figura a seguir mostra a localização de importantes diretórios e arquivos especificados pela **FHS** (*Linux Filesystem Hierarchy Standard*)



Estrutura

- `/` (**raiz**): o diretório raiz, presente na estrutura de todo sistema de arquivos Linux, ancestral de todos os arquivos no sistema de arquivos

- `/bin` (**comandos binários essenciais**): armazena os arquivos necessários para inicializar o sistema e executá-lo

- `/boot` (**arquivos estáticos do *boot loader***): contém todos os arquivos necessários para dar boot no sistema

- `/dev` (**Arquivos de dispositivos**): contém todos os arquivos que representam dispositivos periféricos, como drivers de disco, terminais e impressoras; atualmente `udev` provê um diretório de dispositivos dinâmicos que configuraram `/dev` somente com os dispositivos presentes no sistema (ao invés de todos os possíveis)

Estrutura

- `/etc` (**Arquivos de configurações locais do sistema**): mantém arquivos de administração, configuração e outros de sistema; um dos mais importantes é `/etc/passwd` que contém uma lista de todos os usuários que possuem permissão de usar o sistema

- `/etc/opt` (**Arquivos de configuração para pacotes opcionais mantidos em** `/opt`)

- `/etc/X11` (**Arquivos de configuração para o sistema de janelas X**)

- `/home` (**Diretórios pessoais de usuários**): cada diretório pessoal do usuário é tipicamente um de muitos dos diretório no `/home`; por exemplo, o caminho absoluto para o usuário `zach` normalmente é `/home/zach`

Estrutura

- `/lib` (**Bibliotecas compartilhadas**)
- `/lib/modules` (**Módulos de kernel carregáveis**)
- `/mnt` (**Pontos de montagem para montar sistemas de arquivos temporários**)
- `/opt` (**Pacotes de software opcionais**)
- `/proc` (**Sistema de arquivo virtual com informações do kernel e processos**)

Estrutura

- `/sbin` **(Binários de sistema essenciais):** utilitários usados para administração do sistema são mantidos em `/sbin` (binários usados no processo de boot) e `/usr/sbin` (binários depois que o sistema está carregado e executando)

- `/sys` (**Pseudo sistema de arquivos para dispositivos**)

- `/tmp` (**Arquivos temporários**)

- `/usr` (**Segunda maior hierarquia de diretórios**): tradicionalmente inclui subdiretórios que contém informações usadas pelo sistema; não são modificados frequentemente e podem ser compartilhados entre vários sistemas

Estrutura

- `/usr/bin` (**Maioria dos comandos de usuários**): contém ferramentas padrões, binários que não são necessários no modo de usuário único e recuperação
- `/usr/games` (**Jogos e programas educacionais**)
- `/usr/include` (**Arquivos de cabeçalhos usados por programas C**)
- `/usr/lib` (**Bibliotecas**)
- `/usr/local` (**Hierarquia local**): armazena arquivos e diretórios de importância local, como `bin`, `games`, `include`, `lib`, `sbin`, `share` e `src`

Estrutura

- `/usr/sbin` (**Binários de administração de sistema não-vitais**)
- `/usr/share` (**Dados independentes de arquitetura**): subdiretórios incluem `dict`, `doc`, `games`, `info`, `locale`, `man`, `misc`, `terminfo` e `zoneinfo`
- `/usr/share/doc` **(Documentação)**
- `/usr/share/info` **(Diretório principal do sistema GNU `info`)**
- `/usr/share/man` **(Manuais online)**

Estrutura

- `/usr/src` (**Código-fonte do kernel**)

- `/var` (**Dados variáveis**): arquivos com conteúdo que variam durante a execução do sistema, exemplos comuns são arquivos temporários, logs de sistema, de *spool* e caixa de e-mail do usuário; subdiretórios incluem `cache`, `lib`, `lock`, `log`, `mail`, `opt`, `run`, `spool` e `tmp`

- `/var/log` (**Arquivos de logs**): contém `lastlog` (registro do último *login* de cada usuário), `messages` (mensagem de sistema do `syslogd`) e `wtmp` (registro de todos os *login/logout*), entre outros

- `/var/spool` (**Dados de aplicação *spool***): contém `anacron`, `at`, `cron`, `lpd`, `main`, `mqueue`, `samba` e outros diretórios

PERMISSÕES

Multiuser

- Sistemas operacionais UNIX não são somente *multitasking*, mas também *multiuser*; ou seja, mais de uma pessoa pode usar o computador ao mesmo tempo

- Mesmo que o computador tenha somente um teclado e um monitor, ainda sim é possível ter mais de um usuário simultaneamente se estiver conectado a uma rede ou internet
  - Via `ssh` (*secure shell*)
  - Executar aplicações gráficas em um display remoto

- Essa capacidade é profundamente embutida no design do sistema operacional se considerar o ambiente onde foi criado
  - Antes de computadores serem pessoais eles eram grandes, caros e centralizados; por exemplos computadores de universidades que conectavam vários campus em um mainframe

Proteção

- Para fazer isso prático, um método deve ser criado para proteger usuários uns dos outros
  - As ações de um usuário não poderiam travar o sistema, nem um usuário poderia afetar os arquivos pertencentes a outro usuário

- Será visto as partes essenciais desse sistema de segurança e introduzir os seguintes comandos
  - `id`: mostrar informações de identidade do usuário
  - `chmod`: alterar os modos de um arquivo
  - `umask`: definir as permissões de arquivos padrão

Não iremos ver alguns comandos administrativos como: `su`, `sudo`, `chown`, `chgrp`, `passwd`

Donos, Membros de Grupos e Todo Mundo

- Ao explorar um sistema, pode-se deparar com os seguintes problemas ao tentar examinar um arquivo como `/etc/shadow`

```
$ file /etc/shadow
/etc/shadow: regular file, no read permission
$ less /etc/shadow
/etc/shadow: Permission denied
```

  - Isso acontece porque, como usuário regular, não se tem permissão para ler esse arquivo (que contém as senhas dos usuários)

Donos, Membros de Grupos e Todo Mundo

- No modelo de segurança de Unix, um usuário pode

  1) Ser **dono** de arquivos e diretórios; nesse caso, o usuário tem controle sobre seu acesso
  2) Pertencer a um **grupo** (que pode ter mais usuários) com acesso dado pelo dono aos arquivos e diretórios
  3) Ter acesso dado pelo dono a arquivos e diretórios para **todos** os outros usuários

Donos, Membros de Grupos e Todo Mundo

- Quando uma conta de usuário é criada é atribuído
  - Um número chamado de **identificador do usuário** (*UID*), mapeado para um nome de usuário
  - Um **identificador de grupo** primário (*GID*) e possíveis **grupos adicionais**

- Por exemplo, pode-se usar o comando `id` para obter informações sobre sua identidade

```
$ id
uid=1000(me) gid=1000(me) groups=4(adm),20(dialout),
24(cdrom),25(floppy),29(audio),30(dip),44(video)
46(plugdev),108(lpadmin),114(admin),1000(me)
```

Donos, Membros de Grupos e Todo Mundo

- Como muitas coisas no Linux, a gerencia dessas informações é feitas por arquivos de texto simples
  - Contas de usuários são definidas no arquivo `/etc/passwd`, contém nome de *login*, *uid*, *gid*, nome real, diretório pessoal e a shell de *login*
  - Senhas são mantidas no arquivo `/etc/shadow`
  - Grupos são definidos no arquivo `/etc/group`

Embora alguns sistemas Unix adicionam usuários a um grupo comum (como `users`), sistemas modernos os adicionam em um grupo de mesmo nome do usuário com um único usuário

Donos, Membros de Grupos e Todo Mundo

- Direitos de acesso à arquivos e diretórios são definidos em termos de acesso de **leitura**, **escrita** e **execução**

- A saída do comando `ls` (com opção `-l`) indica como isso é implementado

Dica: para criar um arquivo vazio pode-se usar o comando `touch`

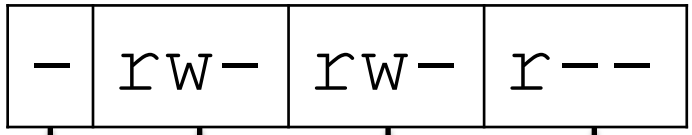
```
$ > foo.txt
$ ls -l foo.txt
-rw-rw-r-- 1 me me 0 2012-03-06 14:52 foo.txt
```

  - Os dez primeiros caracteres da listagem são atributos de arquivos
    - O primeiro caractere é o **tipo do arquivo**
    - Os nove caracteres restantes são o **modo do arquivo** e representam as permissões de leitura, escrita e execução para o dono do arquivo, grupo e todo o mundo

Atributos

- Quebrando os atributos do arquivo em partes

Quando definidos, os modos `r`, `w` e `x` e têm certos efeitos em arquivos e diretórios



1) Tipo do arquivo
2) Permissões do dono
3) Permissões do grupo
4) Permissões de todos

Atributos: Tipo do Arquivo

- A tabela a seguir detalha os tipos de arquivos

Atributo	Tipo do Arquivo
`-`	Um arquivo regular
`d`	Um diretório
`l`	Um link simbólico - repare que o resto dos atributos é sempre `rwxrwxrwx` que são valores fictícios
`c`	Um arquivo de caractere especial - dispositivos que lidam com dados como um fluxo de bytes
`b`	Um arquivo de bloco especial - dispositivos que lidam com dados como blocos, como discos rígidos ou drive de CD-ROM

Atributos: Permissões

- Para arquivos
  - *read* (`r`): permite que um arquivo seja aberto e lido
  - *write* (`w`): Permite que um arquivo seja escrito ou truncado; mas não permite renomear ou remover arquivos (isso depende das permissões do diretório)
  - *execute* (`x`): Permite que um arquivo seja tratado como um programa executável; programas escritos em linguagens de scripts devem também ser legíveis para que sejam executados

Atributos: Permissões

- Para diretórios
  - *read* (`r`): permite que o conteúdo de um diretório seja listado se o atributo de execução também estiver definido
  - *write* (`w`): permite que arquivos dentro de um diretório sejam criados, removidos e renomeados se o atributo de execução também estiver definido
  - *execute* (`x`): permite entrar em um diretório (ex.: `cd dir`)

Exemplos

- Qual o significado de cada um dos atributos a seguir?

Atributos
-rwx------
-rw-------
-rw-r--r--
-rwxr-xr-x
-rw-rw----
lrwxrwxrwx
drwxrwx---
drwxr-x---

Alterar Modo do Arquivo (`chmod`)

- O comando `chmod` é usado para alterar o modo (permissão) de um arquivo ou diretório
  - Somente o dono do arquivo ou superusuário podem alterar o modo de um arquivo ou diretório

- `chmod` suporta duas formas distintas para especificar mudanças de modo, são elas
  - Representação **octal**
  - Representação **simbólica**

Representação Octal

- Com a notação octal usa-se números na base octal para definir o padrão de permissão desejado
  - Como cada digito em octal representa três dígitos em binário, esse mapeamento casa perfeitamente com o esquema usado para armazenar o modo do arquivo

Octal	Binário	Modo
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Usar três dígitos octais permite definir o modo para o dono, grupo e todos

Representação Octal

- Ao passar o argumento `600` para o `chmod` é possível definir a permissão de leitura e escrita para o dono enquanto se remove todas as permissões para o grupo e todos

```
$ > foo.txt
$ ls -l foo.txt
-rw-rw-r-- 1 me me 0 2012-03-06 14:52 foo.txt
$ chmod 600 foo.txt
$ ls -l foo.txt
-rw------- 1 me me 0 2012-03-06 14:52 foo.txt
```

Dica: Uma forma mais fácil de usar é decorar somente os modos de **leitura** (`4`), **escrita** (`2`) e **execução** (`1`) e somá-los para criar as combinações desejadas

Representação Simbólica

- `chmod` também suporta a notação simbólica para especificar modos de arquivos

- A notação simbólica é dividida em três partes
  1) Quem a mudança irá afetar
  2) Qual operação será realizada
  3) Qual permissão será definida

Representação Simbólica

1) Quem a mudança irá afetar
   - Uma combinação de caracteres `u`, `g`, `o` e `a` são usados, conforme a tabela a seguir

Símbolo	Significado
`u`	`u` de usuário, mas significa dono
`g`	`g` de grupo para o dono do grupo
`o`	`o` de outros, mas significa todos
`a`	`a` de todos (*all*); combinação de `u`, `g` e `o`

Se não for especificado, assume-se `a` (todos)

Representação Simbólica

2) Qual operação será realizada
   - As operações podem ser
     - `+`: indica que uma permissão será adicionada
     - `-`: indica que uma operação será retirada
     - `=`: indica que somente as permissões especificadas serão aplicadas, todas as outras serão removidas

3) Qual permissão será definida
   - Permissões são especificadas usando as letras
     - `r`: leitura
     - `w`: escrita
     - `x`: execução

Representação Simbólica

Qual das duas notações usar?

- Exemplos de permissões usando a notação simbólica

Notação	Significado
`u+x`	Adicionar permissão de execução para o dono
`u-x`	Remover permissão de execução do dono
`+x`	Adicionar permissão de execução para o dono, grupo e todos; equivalente a `a+x`
`o-rw`	Remover a permissão de leitura e escrita de todos, exceto do dono e do grupo
`go=rw`	Definir a permissão de leitura e escrita para o grupo e todos, exceto o dono; se o grupo ou todos tivessem a permissão de execução essa seria removida
`u+x,go=rx`	Múltiplas especificações podem ser separadas por vírgulas

Definir Permissões Padrões (`umask`)

- O comando `umask` controla as permissões padrões dadas a um arquivo no momento que é criado
- Usa a notação octal para expressar uma máscara de bits a serem removidos dos atributos de modo do arquivo
- Por exemplo, por padrão a máscara é `0022`

```
$ rm -f foo.txt
$ umask 0022
$ > foo.txt
$ ls -l foo.txt
-rw-r--r-- 1 me me 0 2012-03-06 14:53 foo.txt
```

Definir Permissões Padrões (`umask`)

- Um outro exemplo, mas usando a máscara `0000` (efetivamente desligando-a); agora todos tem permissão de escrita

```
$ rm -f foo.txt
$ umask 0000
$ > foo.txt
$ ls -l foo.txt
-rw-rw-rw- 1 me me 0 2012-03-06 14:58 foo.txt
```

Mas como que funciona isso?

Definir Permissões Padrões (umask)

- Para a máscara `0022`

Modo original	`---`	`rw-`	`rw-`	`rw-`
Máscara	`000`	`000`	`010`	`010`
Resultado	`---`	`rw-`	`r--`	`r--`

Onde o bit é um (`1`), desabilita a permissão

Dica: experimente com outras máscaras, por exemplo sete (`7`)

Qual(is) operação(es) de bits são executadas nesta operação?

Permissões Especiais

- Normalmente, expressões octais são expressada com três dígitos octais, contudo é tecnicamente mais correto expressar com quatro

    1) ***SUID*** (*set user ID*): octal `4000`
    2) ***SGID*** (*set group ID):* octal `2000`
    3) ***Sticky bit:*** octal `1000`

Permissões Especiais

- ***SUID*** (*set user ID*)
  - Quando um arquivo com *setuid* é executado, o processo resultante irá assumir o identificador efetivo do dono do arquivo

`chmod u+s` *`program`* `-rwsr-xr-x`

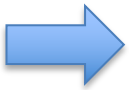
Permissões Especiais

- ***SGID*** (*set group ID*)
  - Quando um arquivo com *setgid* é executado, o processo resultante irá assumir o identificador efetivo do grupo do arquivo
  - Se for um diretório novos arquivos criados nele receberão o grupo do diretório ao invés do grupo do usuário que o criou

`chmod g+s` *`dir`* `drwxrwsr-x`

Permissões Especiais

- ***Sticky bit***
  - Para arquivos: permite que um executável não seja paginado (*swapped*); essa permissão é ignorada atualmente para arquivos
  - Para diretórios: previne que um usuário remova ou renomeie um arquivo (a não ser que ele seja o dono do diretório, dono do arquivo ou o superusuário)

`chmod +t` *`dir`*  `drwxrwxrwt`

MAIS COMANDOS

Shell Scripting

Comandos

- Mais alguns comandos envolvendo sistema de arquivos

Comando	Propósito
`basename`	Obter o nome do arquivo de um caminho de arquivo
`dirname`	Obter o diretório onde está o arquivo em um caminho de arquivo
`gzip/gunzip`	Comprimir/descomprimir arquivos
`tar`	Criar/extrair uma coleção de arquivos
`split`	Dividir grandes arquivos em múltiplos pedaços
`shred`	Apagar arquivos de forma segura
`cksum`	Mostrar o *checksum* (soma de verificação) de um arquivo
`md5sum`	Mostrar o *hash* MD5 de um arquivo
`dd`	Criar cópias brutas de dados de dispositivos
`lsof`	Listar arquivos abertos
`fuser`	Mostrar informações sobre arquivos abertos

basename

- **Propósito:** obter o nome do arquivo de um caminho de arquivo
- **Sintaxe:** `basename [filepath]`
  - Executar o comando `basename` em um caminho de arquivo imprime o nome do arquivo

```
$ basename /etc/hosts
hosts
```

  - Se executado sobre um diretório, mostra o nome do diretório no caminho

```
$ basename /home/max
max
```

E se o caminho não existir?

dirname

- **Propósito:** obter o diretório do arquivo de um caminho de arquivo
- **Sintaxe:** `dirname [filepath]`
  - Executar o comando `dirname` em um caminho de arquivo imprime o nome do diretório desse arquivo

```
$ dirname /etc/hosts
/etc
```

  - Se executado sobre um diretório, mostra o nome base do diretório no caminho

```
$ dirname /home/max
/home
```

gzip

- **Propósito:** comprimir arquivos
- **Sintaxe:** `gzip [options] [file]`
  - `gzip` é uma ferramenta de compressão simples usada para reduzir o tamanho de um grande arquivo para um arquivo `.gz`

```
$ ls -lh BigFile
-rw-r--r-- 1 nick nick 3.0M 2010-05-20 14:02 BigFile
$ gzip BigFile
$ ls -lh BigFile.gz
-rw-r--r-- 1 nick nick 433K 2010-05-20 14:02 BigFile.gz
```

Dica: existem outros compactadores, como `compress` (`.z` ou `.Z`), `bzip` (`.bz`) e `bzip2` (`.bz2`)

gunzip

- **Propósito:** descomprimir arquivos
- **Sintaxe:** `gunzip [options] [file]`
  - `gunzip` (ou `gzip -d`) descompacta um arquivo compactado com a ferramenta `gzip`

```
$ gunzip BigFile.gz
$ ls -lh BigFile
-rw-r--r-- 1 nick nick 3.0M 2010-05-20 14:02 BigFile
```

Dica: para os outros compactadores, existem os descompactadores relacionados, como `uncompress` (`.z` ou `.Z`), `bunzip` (`.bz`) e `bunzip2` (`.bz2`)

gzip/gunzip

- Usos comuns

Comando	Propósito
`gzip [file]`	Comprimir o arquivo especificado
`gzip --fast [file]`	Comprimir o arquivo usando o método mais rápido
`gzip --best [file]`	Comprimir o arquivo usando o maior nível de compressão
`gzip -tv [archive]`	Testar o arquivo especificado por erros
`gzip -l [archive]`	Mostrar informações sobre o arquivo especificado
`gunzip [archive]`	Descomprimir o arquivo especificado
`gzip -d [archive]`	Descomprimir o arquivo especificado

tar

- **Propósito:** criar/extrair uma coleção de arquivos
- **Sintaxe:** `tar [options] [output] [input]`
  - `tar` é muito utilizado para criar backups, usando as opções `-cvf`

```
$ sudo tar -cvf backup.tar /etc/*
/etc/acpi/
/etc/acpi/stopbtn.sh
/etc/acpi/videobtn.sh
...
```

  - Pode-se usar a opção `-x` para extrair todo o conteúdo ou apenas um arquivo

```
$ sudo tar -xvf backup.tar
etc/hosts
etc/hosts
...
```

`tar` remove a barra inicial dos caminhos, usando o caminho relativo durante a extração

tar

- Usos comuns

Comando	Propósito
`tar -cvf [file] [item]`	Fazer um backup dos arquivos especificados
`tar -czvf [file] [item]`	Comprimir o arquivo com `gzip`
`tar -cjvf [file] [item]`	Comprimir o arquivo com `bzip2`
`tar -xvf [file]`	Extrair todos os arquivos da coleção na pasta atual
`tar -xvf [file] -C [dir]`	Extrair todos os arquivos da coleção para o diretório `dir`
`tar -xvf [file] [item]`	Extrair arquivos especificados da coleção
`tar -zxvf [file]`	Extrair os arquivos da coleção compactada com `gzip`
`tar -jxvf [file]`	Extrair os arquivos da coleção compactada com `bzip2`
`tar -tf [file]`	Listar todos os arquivos da coleção

split

Como restaurar o arquivo juntando as partes?

- **Propósito:** dividir grandes arquivos em múltiplos pedaços
- **Sintaxe:** `split [options] [file] [output]`
  - Nesse exemplo, o comando `split` divide um arquivo em vários pedaços de tamanho 100MB (opção `-b 100M`), em que cada pedaço recebe uma extensão incremental

```
$ ls -l ubuntu.iso
-rw-r--r-- 1 nick nick 671686656 2009-10-27 12:07 ubuntu-9.10.iso
$ split -d -b 100M ubuntu-9.10.iso ubuntu.iso.
$ ls -lh ubuntu*
-rw-r--r-- 1 nick nick 641M 2009-10-27 12:07 ubuntu-9.10.iso
-rw-r--r-- 1 nick nick 100M 2010-04-11 11:44 ubuntu.iso.00
-rw-r--r-- 1 nick nick 100M 2010-04-11 11:44 ubuntu.iso.01
-rw-r--r-- 1 nick nick 100M 2010-04-11 11:44 ubuntu.iso.02
-rw-r--r-- 1 nick nick 100M 2010-04-11 11:44 ubuntu.iso.03
-rw-r--r-- 1 nick nick 100M 2010-04-11 11:44 ubuntu.iso.04
-rw-r--r-- 1 nick nick 100M 2010-04-11 11:44 ubuntu.iso.05
-rw-r--r-- 1 nick nick  41M 2010-04-11 11:44 ubuntu.iso.06
```

split

- Usos comuns

Comando	Propósito
`split -b [size] [file] [output]`	Dividir um arquivo em múltiplos pedaços
`split -d -b [size] [file] [output]`	Usar um sufixo numérico

shred

- **Propósito:** apagar arquivos de forma segura
- **Sintaxe:** `shred [options] [file]`
  - O comando `shred` sobreescreve um arquivo de forma segura (e opcionalmente o deleta com a opção `-u`)

```
$ shred -u SecretPlans.txt
$ ls -l SecretPlans.txt
ls: cannot access SecretPlans.txt: No such file or directory
```

  - Por padrão, o comando `shred` sobreescreve com 3 passos de dados aleatórios, mas pode-se usar a opção `-n` com outro valor

```
$ shred -n 5 -v SecretPlans.txt
shred: SecretPlans.txt: pass 1/5 (random)...
shred: SecretPlans.txt: pass 2/5 (ffffff)...
shred: SecretPlans.txt: pass 3/5 (000000)...
shred: SecretPlans.txt: pass 4/5 (333333)...
shred: SecretPlans.txt: pass 5/5 (random)...
```

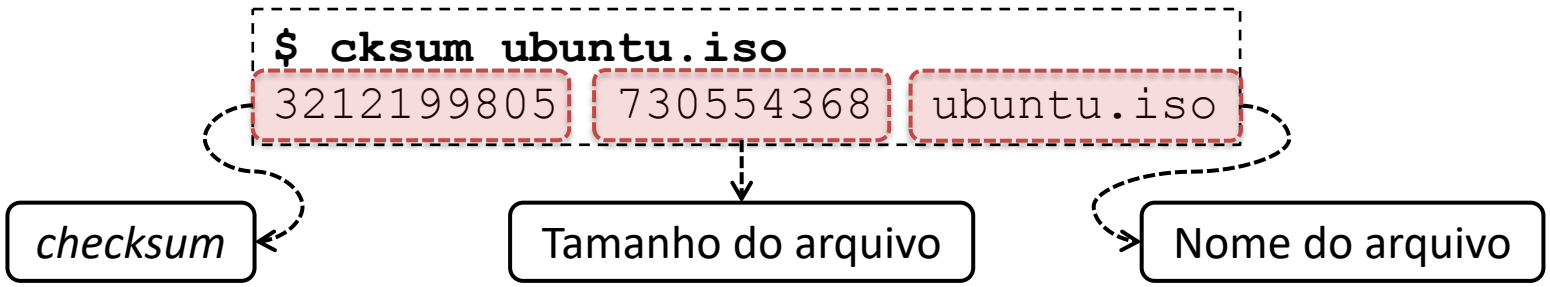
shred

- Usos comuns

Comando	Propósito
`shred [file]`	Despedaçar o arquivo especificado
`shred -u [file]`	Despedaçar e remover o arquivo especificado
`shred -zu [file]`	Tentar esconder evidencias que o arquivos foi despedaçado
`shred -v [file]`	Mostrar o progresso de cada passo
`shred -n [num] [file]`	Executar o número especificado de passes de sobrecarga

cksum

- **Propósito:** mostrar o *checksum* (soma de verificação) de um arquivo
- **Sintaxe:** `cksum [options] [file]`
  - O comando `cksum` é tipicamente utilizado para verificar a integridade de arquivos (principalmente transferidos pela rede)
  - o *checksum* resultante pode ser comparado com o *checksum* do arquivo original para garantir que não contém erros



Dica: pode-se analisar múltiplos arquivos passando-os como parâmetros

md5sum

- **Propósito:** mostrar o *hash MD5* de um arquivo
- **Sintaxe:** `md5sum [options] [file]`
  - O comando `md5sum` computa a *soma MD5* (também referenciada como *hash*) do arquivo especificado

```
$ md5sum ubuntu.iso
cace6ea9dde8dc158174e345aabe3fae ubuntu.iso
```

  - *Hashs MD5* são equivalente a impressões digitais e são improváveis de gerar duplicatas controladas como `cksum` (mas em alguns casos é possível)

Em sistemas BSD (inclusive Mac) é comum encontrar a ferramenta `md5` para o mesmo propósito

Dica: o comando `sha256sum` é similar ao `md5sum`, mas usa o algoritmo *SHA-256* para geração do *hash*

dd

- **Propósito:** criar cópias brutas de dados de dispositivos
- **Sintaxe:** `dd if=[source] of=[target] [options]`
  - O comando `dd` realiza uma cópia de dados bruta (bit a bit) de dispositivos de dados; por exemplo, copiar todo o conteúdo de um disco para outro

```
$ sudo dd if=/dev/sdb of=/dev/sdc
```

  - Ou pode ser usado para extrair a imagem de um CD-ROM

```
$ sudo dd if=/dev/cdrom of=/tmp/image.iso
```

Cuidado: `dd` é também conhecido como **destruidor de dados**, porque pode ser muito destrutivo se usado incorretamente

lsof

- **Propósito:** listar arquivos abertos
- **Sintaxe:** `lsof [options] [name]`
  - Executar o comando `lsof` sem argumentos irá mostrar todos os arquivos abertos no sistema; especificar o nome do arquivo aberto irá mostrar informações de quem o está usando

```
$ sudo lsof /etc/hosts
COMMAND   PID     USER   FD   TYPE   DEVICE   SIZE    NODE    NAME
tail      12793   nick   3r   REG      8,1     256  1777676   /etc/hosts
```

  - A opção `-u` permite ver todos os arquivos aberto de um determinado usuário

```
$ sudo lsof -u nick
COMMAND   PID     USER   FD   TYPE   DEVICE   SIZE    NODE    NAME
sshd      17899   nick   cwd  DIR      8,1    4096       2    /
tail      12793   nick   3r   REG      8,1     256  1777676   /etc/hosts
...
```

lsof

- Usos comuns

Comando	Propósito
`lsof`	Listar todos os arquivos abertos
`lsof [file]`	Listar informações sobre um arquivo específico
`lsof -u [username]`	Listar arquivos abertos por um determinado usuário
`lsof -p [pid]`	Listar arquivos abertos por um determinado número de processo (*PID*)
`lsof -c [process name]`	Listar todos os arquivos aberto pelo processo de nome especificado
`lsof -i`	Listar todos as portas e *sockets* de rede abertos

fuser

- **Propósito:** mostrar informações sobre arquivos abertos
- **Sintaxe:** `fuser [options] [directory/file]`
  - `fuser` é útil para identificar o usuário ou o programa que está usando um arquivo

```
$ fuser -v /home/nick/ShoppingList.txt
28528c(nick)
                     USER     PID  ACCESS  COMMAND
ShoppingList.txt:    nick   14044  ..c..   tail
```

  - A opção `-ki` pode ser usada para terminar um processo atualmente usando um arquivo especificado

```
$ fuser -ki /home/nick/ShoppingList.txt
/home/nick/ShoppingList.txt:     14044
Kill process 14044 ? (y/N) y
```

fuser

- Usos comuns

Comando	Propósito
`fuser [file]`	Mostrar processos usando o arquivo especificado
`fuser -v [file]`	Mostrar informações detalhadas sobre o arquivo em uso
`fuser -ki [file]`	Matar todos os processos que estão usando determinado arquivo

ISSO É TUDO, PESSOAL!