05

Criação e remoção de arquivos

Usando o touch em arquivos e diretórios

Nós já falamos sobre a criação e remoção de diretórios. Vamos falar agora sobre arquivos? Primeiro, vamos começar falando sobre a criação de um arquivo. Se temos um diretório qualquer, por exemplo o `loja`, dentro deste podemos verificar através do `ls` que ele possui um único arquivo, o `bemvindo.html`:

```
> cd loja
~/loja$ ls
bemvindo.html
```

Gostaríamos de criar um novo arquivo que não existe ainda. Temos diversas maneiras de fazer isso, uma é simplesmente "tocar" em um arquivo que ainda não existe, para isso digitamos `touch` e o nome do arquivo que ainda não existe.

Por exemplo, se esse site é o site da minha loja poderíamos criar um outro arquivo que é o arquivo que lista os contatos do site, para tanto escrevemos `touch contato.html`. Quando damos um `touch`, um espaço, e um nome de um arquivo que ainda não existe, o `touch` cria esse arquivo para nós. Se verificarmos o que esse arquivo tem dentro, usando o `cat contato.html` veremos que não há nada dentro dele. O `touch` apenas tocou esse arquivo, mas não criou conteúdo para ele, ele está vazio.

Vamos observar o `ls -l` e analisar o que aparece na listagem longa:

```
> ls -l
total 4
-rw-rw-r-- 1 guilherme guilherme 38 Mar 10 09:44 bemvindo.html
-rw-rw-r-- 1 guilherme guilherme 0 Mar 22 10:32 contato. html
```

Vemos o horário que esse `contato.html` foi criado e podemos observar que o `bemvindo.html` foi criado a mais tempo. O `contato.html` não existia, então, o que o `touch` fez ao tocar nele foi criar o arquivo. O

Então, o `touch` com um arquivo que não existe cria esse arquivo. E o `touch` com um arquivo que já existe? O que será que ele é capaz de fazer?

Vamos verificar:

```
~/loja$ touch bemvindo.html
~/loja$ ls -l
total 4
-rw-rw-r-- 1 guilherme guilherme 38 Mar 10 09:44 bemvindo.html
-rw-rw-r-- 1 guilherme guilherme 0 Mar 22 10:32 contato. html
```

Ele altera o momento de última modificação para o atual instante, mas ele não altera o conteúdo, para observar se isso procede basta digitarmos `cat bemvindo.html` e teremos o seguinte:

```
> cat bemvindo.html
<html>
Bem vindo a nossa loja
</html>
```

Podemos observar que o `bemvindo.html` ainda possui o mesmo conteúdo de muito tempo atrás, porém, se olharmos a data de última modificação ela é a data de agora. Por que? Porque encostamos no arquivo, isto é, nós demos um `touch` nele.

As vezes desejamos mudar a data da última atualização de um arquivo para que certos programas percebam que esse arquivo foi modificado e façam alguma coisa. Um exemplo clássico disso é, por exemplo, temos um servidor *web* rodando e tocamos um arquivo que sinaliza para o servidor que ele tem que "startar", assim, não precisamos ficar mandando ele "reestartar" na hora que queremos . Nós simplesmente tocamos esse arquivo e esse arquivo é visualizado por um outro serviço e quando esse serviço observa que ele foi atualizado e assim que é possível o recebidor "reestarta". Não precisa, necessariamente, ser naquele instante obrigatoriamente. Esse é um dos tipos de serviço que podemos ter quando queremos tocar um arquivo. O outro seria um *backup*, queremos forçar um novo *backup* de um arquivo que já tinha sido "backupeada", então, atualizamos a data de modificação dele, assim o sistema operacional pode achar que ele necessita de um novo *backup*. Basta utilizar só essa variável para dizer que o arquivo foi modificado e, assim, ele vai achar que esse arquivo foi modificado e vai realizar um "backup" novo. Podemos usar outras técnicas para verificar se o arquivo foi modificado.

O arquivo possui como atributo dele, a data da última modificação e como fazemos para alterar essa data? Como já vimos, basta digitar `touch` e o nome de um arquivo que existe, que ele altera a data da atualização desse arquivo para o presente momento. E o `touch` seguido de um nome que não existe, cria esse arquivo e "seta" a data de atualização para a atual. E o `loja` ?

Vamos dar um ´ls -d loja´ e na sequência escreveremos `ls -dl loja` . Repare:

```
~/loja$ cd ..
> ls -d loja
loja
> ls -dl loja
drwxrwxr-x 3 guilherme 4096 Mar 22 10:32 loja
```

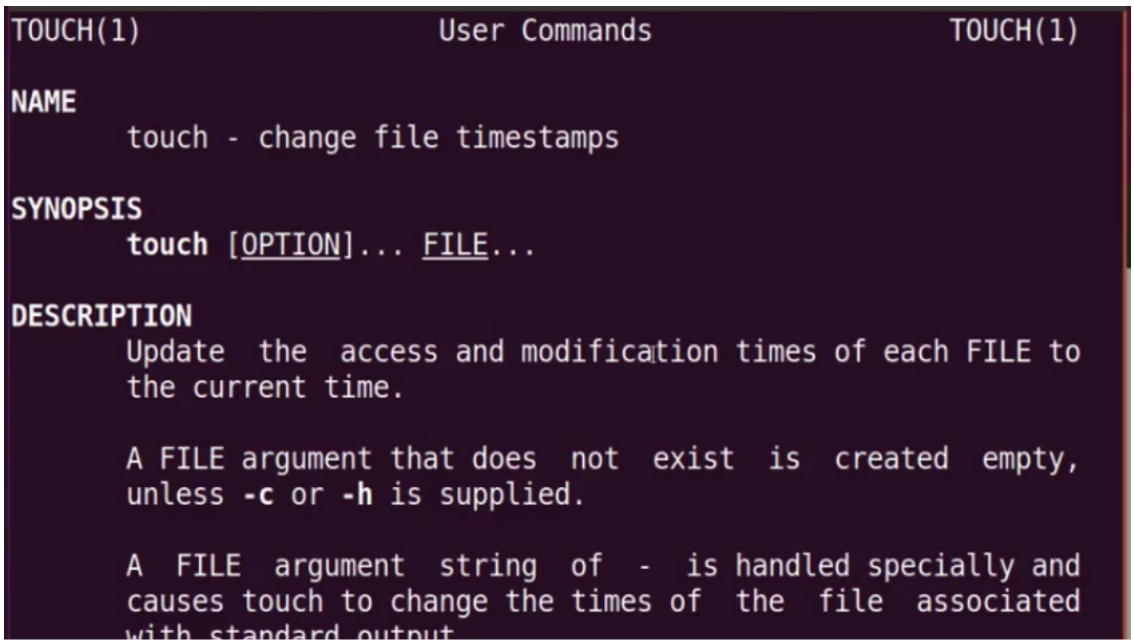
Temos a data de atualização, agora, bem recente, 10:32, no momento em que criamos o arquivo `contato.html` . Repare que a data da última atualização de um diretório não é a data de última atualização de um arquivo ou diretório dentro dele. Não! Por padrão é a data do último momento que o arquivo foi criado aí dentro, ou seja, não está funcionando da maneira mais "intuitiva" que seria aparecer a última atualização de qualquer coisa aí dentro e não necessariamente isso.

E se tentarmos dar um `touch` no diretório? O que ele fala para nós? Ele atualiza o horário do diretório, normal.

```
> ls -dl loja
drwxrwxr-x 3 guilherme 4096 Mar 22 10:35 loja
```

Então, o `touch` vai atualizar o campo do último acesso, da última modificação de um arquivo ou diretório. E se aquele nome não é de algo que existe, ele cria o arquivo.

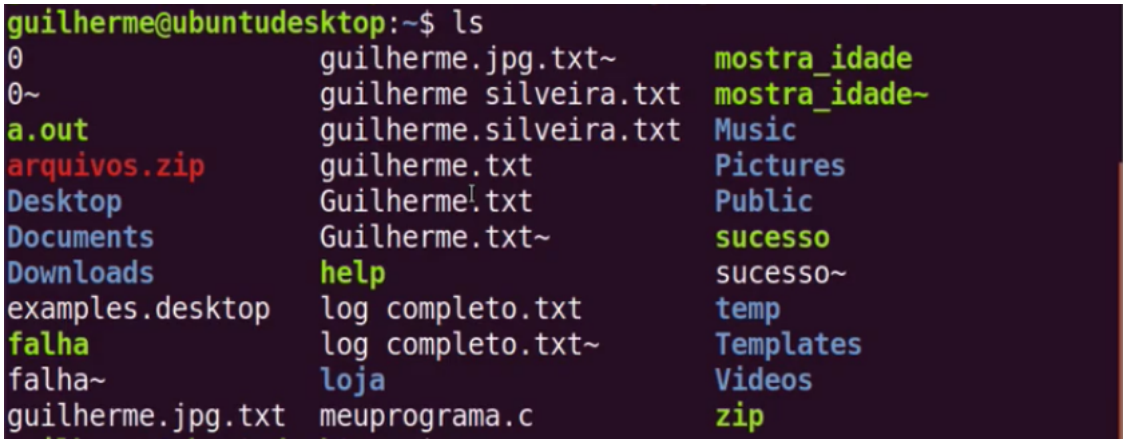
E se dermos um `man touch` veremos que ele fala que ao fazermos um `touch` em um arquivo, ele muda o momento de última modificação desse arquivo e do último acesso a ele.



Então, repare que um arquivo tem armazenado como atributo dele o momento de último acesso e última atualização dele.

Se passarmos a opção `-a` ele muda apenas o *acess time* tempo do último acesso. Se passarmos o `-m` ele muda apenas o tempo da última modificação, se passarmos o `-c` ou o `--no-create` estamos falando que se o arquivo não existe não é para criá-lo.

Vamos testar isso, vamos digitar `touch -c hehehehe` e passamos o nome de um arquivo que não existe. Ao digitar `ls` podemos verificar que o arquivo não foi criado :



Vemos que o arquivo não consta na lista! Se dermos de novo o comando `touch -c hehehehe` e vermos qual foi o retorno dele através do `echo` veremos que o retorno foi "0", ou seja, não deu erro:

```
> touch -c hehehe
echo $?
0
```

Porém, o arquivo não foi criado para nós. O `-c` ignora, não cria o arquivo.

Óbvio, essas as opções são as que você pode usar no dia a dia a medida que precisar, na prática, utilizamos o `touch` naturalmente, o `touch` é usado para criar um arquivo ou o `touch` é utilizado para mudar a data da última atualização, em geral do arquivo. Mas ele muda a data da última atualização e data de último acesso daquele arquivo ou diretório também. É isso que o `touch` faz. O `touch` é o primeiro dos quatro comandos que veremos a respeito de arquivos, veremos outros na sequência.

Copiando arquivos e diretórios com o cp

Uma segunda maneira de trabalhar com a criação de arquivos envolve copiar um arquivo para algum outro lugar. Vamos entrar no diretório da `loja` através do `cd` . Vamos ver os arquivos que temos usando o `ls` e verificamos que temos um arquivo `contato.html` e `bemvindo.html` . Vamos observar também o `-l` que contêm as especificidades desses arquivos, o `bemvindo.html` tem 38 bytes e o `contato.html` tem 0 bytes dentro:

```
> cd loja
~/loja$ ls
bemvindo.html contato.html
~/loja$ ls -l
total 4
-rw-rw-r-- 1 guilherme guilherme 38 Mar 22 10:33 bemvindo.html
-rw-rw-r-- 1 guilherme guilherme 0 Mar 22 10:32 contato.html
```

Se quisermos copiar o arquivo `bemvindo.html` basta darmos um `copy` , digitamos `cp` . Existem diversas maneiras de utilizar o `copy` , veremos algumas das principais. Novamente, o aconselhável é que a medida que vamos utilizando isso no dia a dia leia-se o `man` para que o aprendizado seja contínuo. O uso mais básico é falar que queremos copiar um arquivo para outro nome, por exemplo, "bemvindo" para a nomenclatura "bemvindo2", escreveremos `cp bemvindo.html bemvindo2.html` .Ele copiou todo esse conteúdo para um novo arquivo chamado `bemvindo2.html` . Se dermos um `ls` teremos os seguintes arquivos:

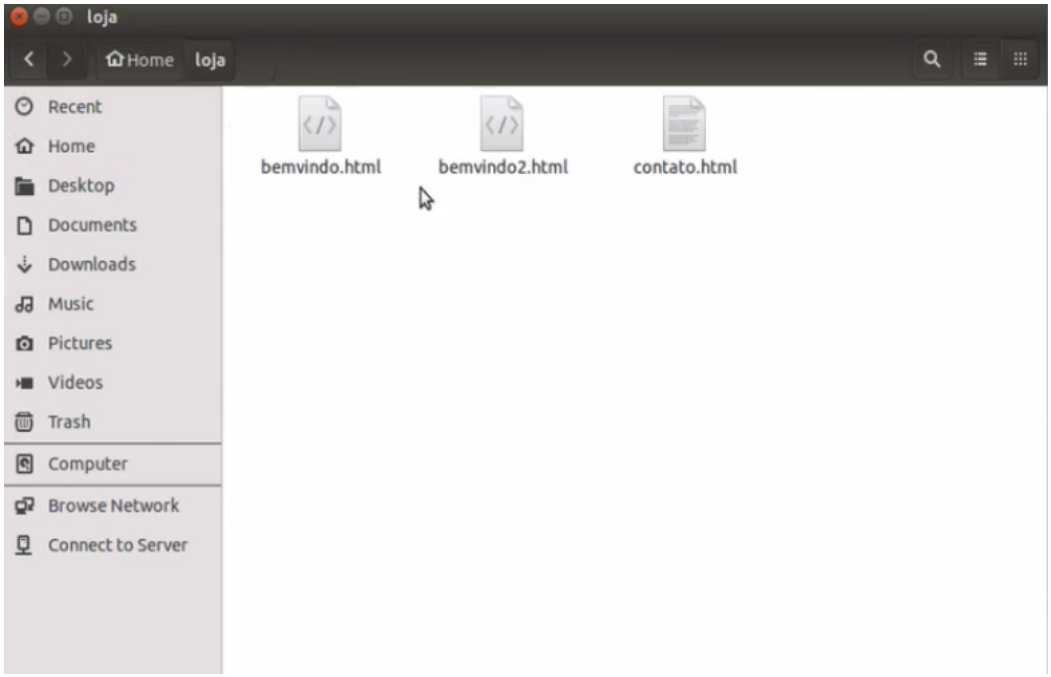
```
~/loja$ ls
bemvindo2.html        bemvindo.html        contato.html
```

E vejamos o que acontece se dermos um `ls -l` :

```
~/loja$ ls -l
total 8
-rw-rw-r-- 1 guilherme guilherme 38 Mar 22 10:39 bemvindo2.html
-rw-rw-r-- 1 guilherme guilherme 38 Mar 22 10:33 bemvindo.html
-rw-rw-r-- 1 guilherme guilherme 0 Mar 22 10:32 contato.html
```

Podemos verificar que os dois arquivos, inclusive, possuem o mesmo tamanho. Então, não é que os dois referenciem um único arquivo. Não temos um arquivo, onde o primeiro arquivo referencia o segundo com um link, um atalho. Não, eles são dois arquivos distintos.

Se abrirmos o navegador e abrirmos o diretório `loja` e clicarmos nele veremos que tem os três arquivos:



Se abrirmos o arquivo `bemvindo2.html` com o *gedit* veremos o que temos nesse arquivo:

```
<html>
Bem vindo a nossa loja
</html>
```

Vamos alterar esse arquivo acrescentando o número dois depois de "Bem vindo":

```
<html>
Bem vindo2 a nossa loja
</html>
```

Vamos salvar isso e vamos voltar ao Terminal e vamos ver o que temos no `ls` :

```
~/loja$ ls
bemvindo2.html        bemvindo2.html~        bemvindo.html        contato.html
```

Podemos observar os arquivos que temos. O "bemvindo2.html~" é o arquivo que o *gedit* cria de *backup*. Vamos olhar também o tamanho dos arquivos através do `ls -l` :

```
~/loja$ ls -l
total 8
-rw-rw-r-- 1 guilherme guilherme 39 Mar 22 10:40 bemvindo2.html
-rw-rw-r-- 1 guilherme guilherme 38 Mar 22 10:39 bemvindo2.html~
-rw-rw-r-- 1 guilherme guilherme 38 Mar 22 10:33 bemvindo.html
-rw-rw-r-- 1 guilherme guilherme 0 Mar 22 10:32 contato.html
```

Podemos ver que o arquivo "bemvindo2.html" tem o tamanho 39 e o "bemvindo2.html~" possui 38 de tamanho. Vamos dar um `cat bemvindo` no `bemvindo2.html` :

```
~/loja$ cat bemvindo2.html
<html>
Bem vindo2 a nossa loja
</html>
```

Está escrito nele "Bem vindo2 a nossa loja". Vamos agora ver o "bemvindo.html":

```
~/loja$ cat bemvindo.html
<html>
Bem vindo a nossa loja
</html>
```

Repare que são dois arquivos com conteúdos totalmente diferentes. Ou seja, não criamos um link, que é um assunto totalmente diferente e que veremos mais para sempre. Nesse instante estamos falando sobre arquivos, como copiar um arquivo completamente, como copiar o conteúdo de um arquivo, se é um arquivo gigante vai demorar para copiar porque ele vai ter que gastar um certo tempo copiando *byte* a *byte* desse arquivo. Essa é a maneira tradicional de copiar arquivos com o `copy` em um sistema de arquivos tradicional. Podemos ter um sistema remoto e virtual de copiar os arquivos e que funciona de uma maneira totalmente diferente, mas nesse caso seriam sistemas de arquivos especiais e que não são o padrão que utilizamos no dia a dia, pelo menos, na maior parte dos casos. Na maior parte dos casos usaremos o `copy`, onde estaremos copiando, localmente ou remotamente, *byte* a *byte*. Justamente, o que estamos fazendo nesse momento.

O que mais? Temos o seguinte conteúdo:

```
~/loja$ ls
bemvindo2.html        bemvindo2.html~        bemvindo.html        contato.html
```

Temos todos esses arquivos e gostaríamos de remover o arquivo extra o "bemvindo2.html~". Para remover um arquivo utilizamos o `rm` de remover, assim como tínhamos o `rmdir`. Então vamos digitar `rm bemvindo2.html~` e será possível remover esse arquivo.

E se utilizarmos o `rm` junto a um arquivo cujo nome não existe? Por exemplo, o `rm ahduahddaui`. Ele responderá que o arquivo não existe:

```
~/loja$ rm ahduahddaui
rm: cannot remove 'ahduahddaui': No such file or directory
```

Mas ele fala também em diretório, então, se tivermos um diretório vazio, por exemplo, o "imagens". Ele apagará esse diretório?

```
~/loja$ mkdir imagens
~/loja$ rm imagens
rm: cannot remove 'imagens': Is a directory
```

O `rm` não remove diretórios. Por mais que o `rm` fale que não existe arquivo ou diretório com esse nome ele falará, ao tentarmos remover um diretório, que ele não é capaz de fazer isso. O `rm` remove apenas arquivos e se usarmos ele solto, no caso do diretório usamos o `rmdir imagens`.

Então, estamos utilizando o *copy* para copiar arquivos e o `rm` para remover arquivos.

Que outras variações temos do `copy` ?

Imagine que temos uma segunda `loja` . Vamos criar a `loja2` utilizando o `mkdir loja2` . Na `loja` normal temos os arquivos `bemvindo2.html` , `bemvindo.html` e `contato.html` e na `loja2` não temos nada.

```
> mkdir loja2
ls loja
bemvindo2.html        bemvindo.html        contato.html
```

Vamos copiar o arquivo `bemvindo.html` usando o `cp` . Lembra-se qual o padrão do `copy` ? Comando, espaço e nome do arquivo, escrito de maneira absoluta ou relativa. E para onde vamos colocar isso com o nome novo, no caso `loja2/bemvindo` :

```
cp loja/bemvindo.html loja2/bemvindo.html
```

Dando um "Enter" nisso teremos copiado o arquivo `loja/bemvindo.html` para o nome `loja2/bemvindo.html` . Se dermos um `ls loja2` veremos que o arquivo que criamos está lá:

```
> ls loja2
bemvindo.html
```

Agora, se vamos copiar um arquivo, para outro diretório, com o mesmo nome, ou seja, mantendo o nome desse arquivo o `copy` pode ser utilizado de uma outra maneira, ao em vez de usarmos o `cp` dizendo "de um arquivo para outro arquivo", diremos "de um arquivo para este diretório", nesse caso, ele manterá o mesmo nome do arquivo. Vamos digitar, então, `cp loja/contato.html loja2` com isso estamos dizendo que queremos copiar o "contato.html" para o diretório "loja2". Repare que ele vai entrar no diretório "loja2", criar um arquivo chamado "contato.html" e colocar o mesmo conteúdo lá dentro.

Vamos testar mais uma vez? Vamos digitar `cp loja/bemvindo2.html loja2/` repare que agora usamos uma `/` no `loja2` , tanto faz estar com ou sem barra, se é uma referência para um diretório ele copia lá para dentro com o mesmo nome. Vamos observar o que aparece dando um `ls` em `loja2` :

```
> ls loja2
bemvindo2.html        bemvindo.html        contato.html
```

Se observamos o `ls -l loja2` teremos os arquivos com o mesmo tamanho que em `loja` :

```
ls -l loja2
total8
-rw-rw-r-- 1 guilherme guilherme 39 Mar 22 10:44 bemvindo2.html
-rw-rw-r-- 1 guilherme guilherme 38 Mar 22 10:43 bemvindo.html
-rw-rw-r-- 1 guilherme guilherme 0 Mar 22 10:44 contato.html
```

Agora repare o `ls - l loja` :

```
ls -l loja
total8
```

```
-rw-rw-r-- 1 guilherme guilherme 39 Mar 22 10:44 bemvindo2.html
-rw-rw-r-- 1 guilherme guilherme 38 Mar 22 10:43 bemvindo.html
-rw-rw-r-- 1 guilherme guilherme 0 Mar 22 10:44 contato.html
```

Os arquivos estão com o mesmo tamanho. Então, dois usos do `cp` . O primeiro é que quando o `cp` vai acompanhado do nome do arquivo original e nome do arquivo que queremos criar. O segundo é quando o `cp` vai acompanhado do nome do arquivo e diretório onde vamos colocar esse arquivo com o mesmo nome, mas é uma cópia desse arquivo.

Temos, ainda, um terceiro uso com `copy` , vamos ver esse uso.

Podemos querer criar uma `loja3` , para isso usamos `mkdir loja3` e nela queremos copiar os arquivos da "loja1", que são os seguintes: `loja/bemvindo.html` e o `loja/bemvindo2.html` , como queremos copiar esse dois arquivos teremos o seguinte

```
cp loja/bemvindo.html loja/bemvindo2.html
```

E se executarmos isso, o que ele irá fazer? Ele vai copiar o primeiro arquivo no segundo arquivo, ele subscreve o arquivo, assim, ele na verdade grava o primeiro nome no segundo.Então, o segundo arquivo vai ser copiado nesse segundo, vai subscrever se for o caso. Assim, no `copy` padrão ele subscreve. Se dermos um "Enter" nisso e dermos um `cd loja` e depois um `ls` e um `ls -l` veremos que deu tudo errado:

```
cd loja
~/loja$ ls
bemvindo2.html        bemvindo.html        contato.html
~/loja$ ls -l
total 8
-rw-rw-r-- 1 guilherme guilherme 39 Mar 22 10:44 bemvindo2.html
-rw-rw-r-- 1 guilherme guilherme 38 Mar 22 10:43 bemvindo.html
-rw-rw-r-- 1 guilherme guilherme 0 Mar 22 10:44 contato.html
```

Vamos abrir o *gedit* do arquivo `bemvindo2.html` para corrigir? Digitamos, para tanto, `gedit bemvindo2.html` . Vai abrir o *gedit* e vamos acrescentar o número 2 no que temos escrito nesse arquivo, ficaremos com:

```
<html>
Bem vindo 2 a nossa loja
</html>
```

Salvamos essa modificação e voltamos ao Terminal. Vamos remover o arquivo `bemvindo2.html~` usando `rm bemvindo2.html~` e vamos dar um `ls` e um `ls -l` . Teremos o seguinte:

```
~/loja$ rm bemvindo2.html~
~/loja$ ls
bemvindo2.html        bemvindo.html        contato.html
~/loja$ ls -l
-rw-rw-r-- 1 guilherme guilherme 40 Mar 22 10:44 bemvindo2.html
-rw-rw-r-- 1 guilherme guilherme 38 Mar 22 10:43 bemvindo.html
-rw-rw-r-- 1 guilherme guilherme 0 Mar 22 10:44 contato.html
```

Repare que agora temos todos os arquivos direitinhos! E o "bemvindo2.html" agora não tem mais 39 de tamanho, ele possui 40, pois, além do 2 que acrescentamos no seu arquivo, adicionamos também um espaço.

Temos que tomar cuidado! O `cp` subscreve, então, temos que tomar muito cuidado quando utilizamos o `copy`, quando copiamos um arquivo para outro arquivo. Tanto que muitas vezes o que fazemos é o `cp` e pedimos para ele nos perguntar antes o que devemos fazer, para isso, digitamos `cp -i loja/bemvindo.html loja/bemvindo2.html` e ele nos perguntará se queremos subscrever: "*overwrite 'loja/bemvindo2.html'?*". E como não queremos que nada seja subscrito, digitaremos apenas um `n`, de "no" para dizermos que não queremos que seja subscrito:

```
cp -i loja/bemvindo.html loja/bemvindo2.html
cp: overwrite 'loja/bemvindo2.html' ? n
```

E se quiséssemos que isso fosse subscrito poderíamos digitar `y` de "yes". O `-i` serve para falar "me pergunte", é uma opção interativa.

O que mais temos para ver?

Antes de chegarmos nesse ponto gostaríamos de ter copiado diversos arquivos, a nossa intenção era ter copiado o `loja/bemvindo.html` e o `loja/bemvindo2.html` para o diretório `loja3`. Bom, vamos voltar a isso, já temos o diretório `loja3` criado e queremos colocar os arquivos mencionados dentro desse diretório. Podemos fazer isso! Digitaremos o seguinte:

```
cp loja/bemvindo.html loja/bemvindo2.html loja3/
```

O terceiro uso do `cp` é falar `copy`, diversos arquivos para um diretório. Portanto, temos três maneiras de usar o `copy`. A primeira é copiar um arquivo para outro arquivo, a segunda é copiar de um arquivo para um diretório, com o mesmo nome e a terceira maneira é copiar diversos arquivos, fontes, para um diretório. Todas as três maneiras funcionam e não teremos problemas.

E se quiséssemos dizer para copiar diversas fontes para um único arquivo? Por exemplo, para o arquivo que está em "loja3", o `meu arquivo.txt`?

Se tentarmos fazer isso digitando:

```
cp loja/bemvindo.html loja/bemvindo2.html loja3/meuarquivo.txt
cp: target 'loja3/meuarquivo.txt' is not a directory
```

Ele nos responde que "loja3/meuarquivo.txt" não é um diretório. Ou seja, ele nos diz que se estamos tentando copiar vários arquivos temos que fazer isso direcionado a um diretório e como tentamos copiar para um arquivo ele nos diz que não é possível realizar esse procedimento pois, justamente, é um diretório e não um arquivo.

São três os usos que temos do `copy`, o primeiro é copiar "de algo para algo", onde, os dois são arquivos, o segundo é "de - para" onde algo é copiado de um arquivo para um diretório e o terceiro uso é copiar vários arquivos para um diretório.

Mas, repare que quando fizemos esse último `copy`, os dois arquivos para o diretório `loja3`, para isso escrevemos

```
cp loja/bemvindo.html loja/bemvindo2.html loja3/
```

Acabamos não vendo o que aconteceu. Poderíamos colocar, ainda, o `-i` que ele nos perguntaria, um a um sobre os arquivos que desejamos subscrever. Vamos digitar `y`, de "yes", para os dois quando ele nos perguntar isso:

```
cp -i loja/bemvindo.html loja/bemvindo2.html loja3/
```

```
cp: overwrite 'loja/bemvindo.html'? y
cp: overwrite 'loja/bemvindo.html2'? y
```

Mas e se fizemos o mesmo com o diretório `loja4` ? Vamos criar o diretório `loja4` digitando `mkdir loja4` e vamos copiar para a `loja4` . Se usarmos o `-i` ele nem pergunta nada:

```
> cp -i loja/bemvindo.html loja/bemvindo2.html loja4/
```

Ele não pergunta nada, pois esses arquivos não existiam, então, ele simplesmente copia!

Assim, nessa primeira vez ele nem perguntou porque não estava subscrevendo, o `-i` pergunta para nós apenas na hora de subscrever.

Vamos criar uma `loja5` , através do `mkdir loja5` e ao em vez de escrevermos o `-i` podemos escrever o `-v` de *verbose* e ele nós falará o que está acontecendo:

```
> mkdir loja5
> cp -v loja/bemvindo.html loja/bemvindo2.html loja5/
'loja/bemvindo.html' -> 'loja4/bemvindo.html'
'loja/bemvindo2.html' -> 'loja4/bemvindo2.html'
```

Ele nos fala o que está acontecendo, ele copia o `loja/bemvindo.html` para o `loja4/bemvindo.html` e também está copiando o `loja/bemvindo2.html` para o `loja4/bemvindo2.html` .

Então, o `-v` é bastante útil no dia a dia para entendermos o que está acontecendo. Mas, nem sempre queremos copiar tudo, por exemplo, aqui estamos fazendo um *backup* da minha `loja` para um diretório de *backup*. Vamos criar um diretório `mkdir lojabackup` e fazemos o mesmo comando, copiando todos os arquivos do diretório e digitaremos `loja/*` e `cp -v loja/* lojabackup/` . Usaremos, novamente, o `-v` para ver o que está acontecendo. Observemos:

```
> cp -v loja/* lojabackup/
'loja/bemvindo2.html' -> 'lojabackup/bemvindo2.html'
'loja/bemvindo.html' -> 'lojabackup/bemvindo.html'
'loja/contato.html' -> 'lojabackup/bemvindo.html'
```

Se rodarmos isso várias vezes ele copiará todas as vezes. O problema é que se o *backup* demora porque os arquivos são muito grandes, podemos querer apenas que ele copie os arquivos que mudaram. Então, vamos tentar algo, vamos pegar os arquivos abrindo o navegador, nós gostaríamos que ele pegue os arquivos originais e copie aqueles que são alterados, para isso, usamos a opção `-u` de *update*. Ele só copia os que foram atualizados, ou seja, os que forem novos desde a última atualização no diretório "lojabackup".

Se dermos um "Enter" depois de `cp -vu loja/* lojabackup` ele não copiára nada.

```
> cp -vu loja/* lojabackup
```

O que acontece se alterarmos algum dos arquivos? Vamos no navegador, clicamos no arquivo "bemvindo2" e abrimos no *gedit*. Vamos acrescentar no que está escrito apenas um `.` :

```
<html>
Bem vindo 2 a nossa loja.
</html>
`
```

Vamos, agora, copiar usando um `-u`, digitando `cp -vu loja/* lojabackup` e teremos o seguinte:

```
> cp -vu loja/* lojabackup
'loja/bemvindo2.html' -> 'lojabackup/bemvindo2.html'
'loja/bemvindo2.html' -> 'lojabackup/bemvindo2.html~' -> 'loja/bemvindo2.html' -> 'lojabackup/bemvir
```

E agora sim, ele fala que copiou o arquivo e copiou, inclusive, o de *backup* que não queríamos. Vamos apagar esse arquivo de *backup* digitando `rm loja/bemvindo2.html~`. Como ele sabe que um arquivo é mais recente do que outros? Através da data de atualização de um arquivo. Isso quer dizer que se dermos um `touch loja/bemvindo.html` ele vai ter uma data mais recente de atualização e acesso, então, quando dermos um `cp -vu loja/* lojabackup/`, ele copia, também, o arquivo que tocamos.

```
> cp -vu loja/* lojabackup/
'loja/bemvindo.html' -> 'lojabackup/bemvindo.html'
```

Lembra do que falamos do `touch`? Usamos ele para mudar a data de atualização e acesso daquele arquivo, então, o `-u` copia apenas o que é mais recente.

Copiando arquivos e diretórios com o cp

Que outras opções do `copy` podemos utilizar?

Temos outra possibilidade de uso que pode ser utilizada quando queremos copiar um arquivo de um lado para o outro, mas não queremos correr o risco de subscrever algo para o outro lado. Então, no caso, vamos copiar do diretório loja para o diretório `loja2`. Para isso, digitaremos `cp loja/* loja2`. Mas, como mencionamos, não queremos correr o risco de subscrever, qual a opção que podemos utilizar? Se usarmos o `-v`, ele acaba subscrevendo, podemos utilizar o `-i` que nos pergunta, um a um, se queremos subscrever. A pergunta "overwrite ?" responderemos: não. Para dizer "no" basta utilizarmos a letra "n". Vejamos:

```
> cp -i loja/* loja2
cp: overwrite 'loja2/bemvindo2.html'? n
cp: overwrite 'loja2/bemvindo.html'? n
cp: overwrite 'loja2/contato.html'? n
```

Entretanto, não é essa a opção que queremos utilizar. O que queremos é, na verdade, copiar. Se já existir, um arquivo com esse nome "loja2" podemos simplesmente fazer um *backup* desse arquivo e, para isso, usamos o `-b` de *backup*. O que ele faz é copiar os arquivos para o `loja2`. Assim, quando digitarmos `cp -b loja/* loja2` ele terá copiado o arquivo. Se entrarmos no diretório `cd loja2` e dermos um `ls` veremos a listagem dos arquivos:

```
cd loja2
~/loja2$ ls
bemvindo2.html        bemvindo.html        contato.html
bemvindo2.html~        bemvindo.html~        contato.html~
```

E veremos que foram criados os arquivos. Podemos verificar que temos dois tipos de arquivo, aqueles que possuem um til e que representam a versão antiga dos arquivos e os que estão sem o acento que representam a versão nova. O til no final, no último carácter, portanto, é um padrão para indicar que determinado arquivo é um *backup* de um arquivo que tínhamos. É uma convenção utilizada, o que não significa que todas as ferramentas utilizarão esse padrão.

O `cp -b` faz isso para nós, ele subscreve o arquivo de destino, só que antes disso ele acaba fazendo um *backup* da última versão. Se rodássemos o `cp -bv loja/* loja2` agora, o que ele faria? Note que vamos usar junto o `-v` para verificarmos o que está acontecendo:

```
> cp -bv loja/* loja2
'loja/bemvindo2.html' -> 'loja2/bemvindo2.html' (backup:'loja2/bemvindo2.html~')
'loja/bemvindo.html~')
'loja/contato.html' -> 'loja2/contato.html' (backup: 'loja2/contato.html~')
```

Observe que conseguimos verificar que, logo na primeira linha, ele nos diz que o `loja/bemvindo2.html` está sendo copiado e temos um *backup* dele, o `loja2/bemvindo2.html~'` . A mesma coisa acontece na outra linha, copiamos o `'loja2/contato.html'` e fizemos um *backup* dele.

Mas repare que agora o *backup* antigo foi subscrito. Toda vez que executarmos algo ele irá subscrever o *backup*. Mas, ele só fica com um *backup*, ou seja, só encontramos a última versão que foi *backupeada* e ela é marcada com um til. Podemos verificar isso através do `ls loja2` :

```
> ls loja2
bemvindo2.html        bemvindo.html        contato.html
bemvindo2.html~        bemvindo.html~        contato.html~
```

É comum quando temos ferramentas de otimização, por exemplo, o `copy` de um lugar para outro, usarmos o `-v` para sabermos exatamente o que está acontecendo e o `-b` para falarmos que se já existir algo com esse nome, gostaríamos que fosse feito um *backup* dele e que, portanto, ele não se perca. O `-b` é bem comum de ser utilizado e é muito importante no dia a dia.

Vimos diversas coisas de como copiar um arquivo de um lado para o outro. Vamos falar de como copiar um diretório, aliás, várias dessas cópias que fizemos eram para copiar um diretório inteiro. Como fazemos para copiar um diretório, por exemplo, `loja` inteiro para o diretório `loja20` ? Ao digitarmos isso teremos a seguinte resposta:

```
> cp loja loja20
cp: omitting directory 'loja'
```

Não funcionou. E por que? Porque o que estamos tentando copiar é um diretório e isso não é possível de ser feito. A grande questão é que o `copy` funciona das três maneiras que foram citadas anteriormente, a primeira é que ele recebe uma fonte, um arquivo e um destino que é um arquivo também, a segunda é que ele recebe uma fonte que é um arquivo e um destino que pode ser um diretório onde ele vai criar um arquivo de mesmo nome e a terceira opção é que ele aceita uma fonte que pode ter diversos nomes, diversas coisas e o destino que é o diretório.

Mas, podemos copiar um diretório, isto é, uma fonte que é um diretório, para isso usamos o `-r` , ou seja, queremos usar de modo recursivo. Seja o `-r` minúsculo ou `-R` maiúsculo, os dois são suportados no `copy` . Então, escrevendo:

```
cp -R loja20
```

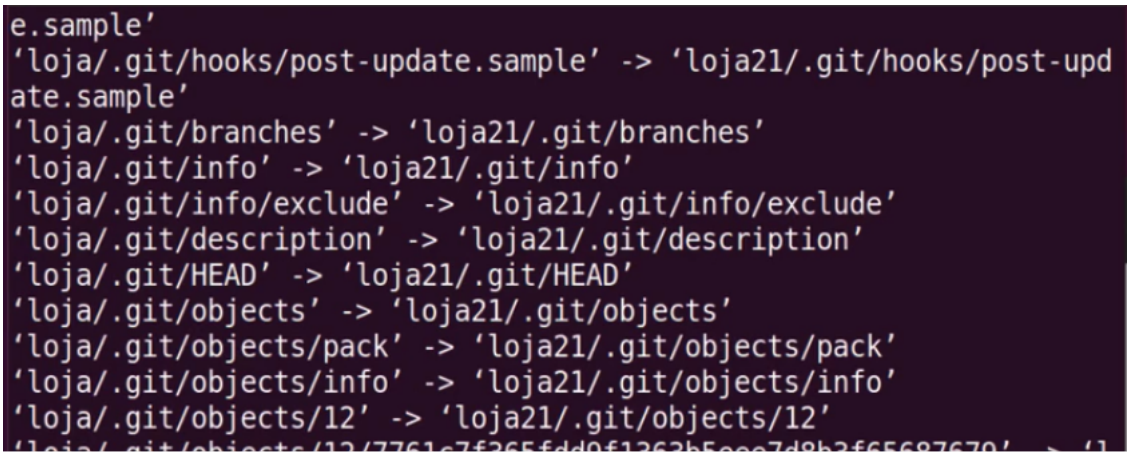
Estaremos falando que queremos que tudo seja copiado, recursivamente, para o diretório `loja20`. Dando um "Enter" podemos confirmar através do `ls loja20` que teremos todos os arquivos que queríamos seguinte:

```
> cp -R loja loja20
> ls loja20
bemvindo2.html          bemvindo.html      contato.html
```

O diretório `loja20`, agora existe e nele temos todos os arquivos. O legal de usar o `cp -r` é adicionar junto o `-v`. Vamos testar o que acontece escrevendo:

```
> cp -Rv loja loja 21
```

Dando um "Enter" nisso teremos o seguinte:



Ele copia e cria todos os arquivos que queríamos dentro. Repare que o diretório loja tinha diversos diretórios, inclusive, diretórios invisíveis, como o `.git` e se observarmos veremos que eles estão todos lá no diretório `loja21`. Portanto, quando usamos o `cp` junto ao recursivo, ele copia, inclusive, os arquivos invisíveis.

É, portanto, recursivo quando utilizamos o `-r` ou `-R`. O `-r` ou `-R` permite que nossas fontes sejam diretórios. Assim, quando digitamos `cp -Rv loja loja21` estamos querendo dizer para o diretório `loja` jogar tudo o que tem dentro do `loja21`. Temos que ter atenção para um aspecto, se falarmos para o `loja` para passar para o `loja21`, o que estamos fazendo é pegar o `loja` e copiar ele para um nome igual a `loja21` e dentro desse novo diretório serão criados os espaços para cada diretório de `loja`.

Se executarmos o `cp -Rv loja loja21` ele irá subscrever tudo:

Agora, mais um cuidado! Se acrescentarmos uma barra ao final desse comando, isto é, ao fim do `loja21`, teremos o seguinte:

```
> cp -Rv loja loja21
```

O que estamos fazendo é dizendo para copiar a fonte `loja` para dentro do diretório `loja21`. Então, dentro do diretório `loja21` ele cria o diretório `loja` e lá dentro ele copia tudo, mais uma vez. Vamos observar o `loja21` digitando `ls loja21`:

```
> ls loja21
bemvindo2.html        bemvindo.html        contato.html        loja
```

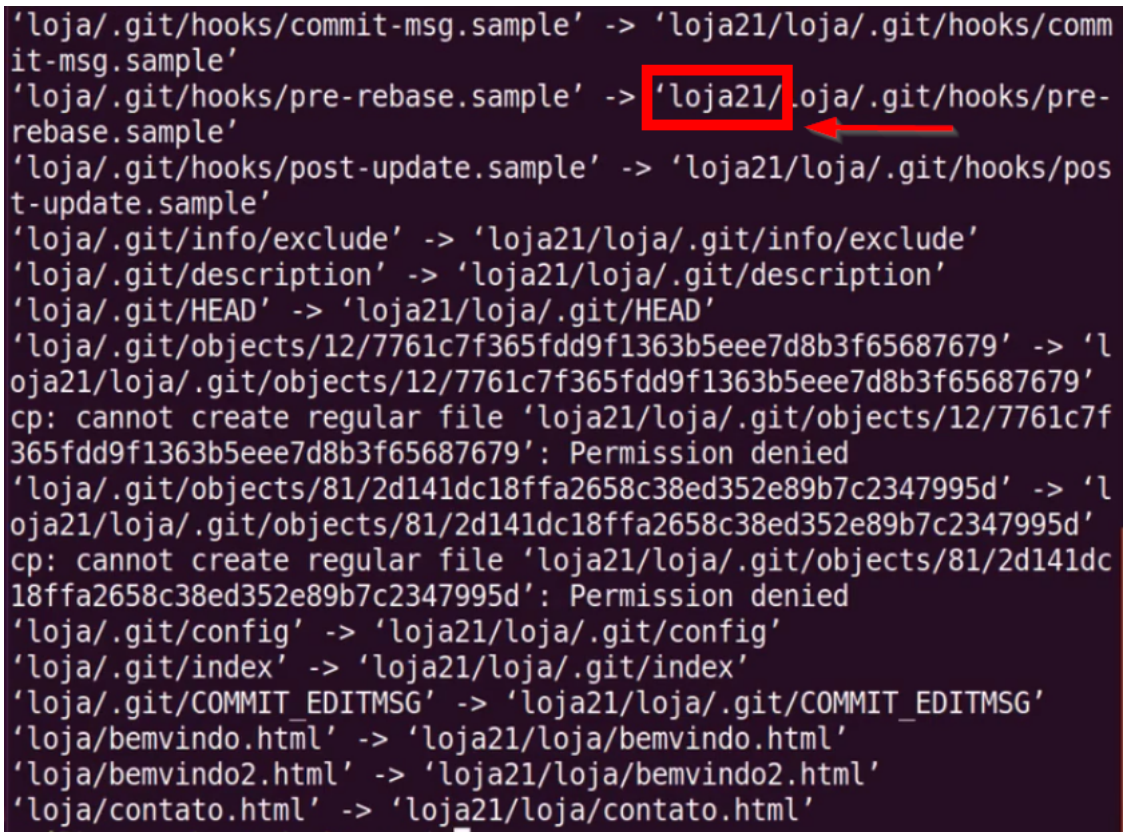
Podemos reparar que o diretório `loja21`, agora, tem o diretório `loja` e os demais arquivos. Se olharmos o `ls loja21/loja` também teremos:

```
> ls loja21
bemvindo2.html        bemvindo.html        contato.html
```

Então, muito cuidado com o uso do `copy`! Quando vamos dar um `copy`, por mais que possamos escrevamos um nome de um arquivo, pasta ou diretório, com ou sem barra, não trará diferença se isso já existir. Quando fazemos `cp -r`, significa copiar recursivamente, de uma fonte para uma pasta. Se digitássemos, por exemplo `cp -R fonte_pasta pasta` ele vai copiar essa pasta e criar uma outra pasta exatamente com esse nome, mas, ao colocarmos uma barra no final ele copia lá para dentro. Vamos observar o `copy`, novamente. Fizemos, anteriormente, da seguinte maneira `cp -Rv loja loja21` e o diretório `loja21` não existia. O que acontece se executarmos isso, agora que o diretório já existe? Dando um "Enter" após:

```
> cp -Rv loja loja21
```

Teremos o seguinte:

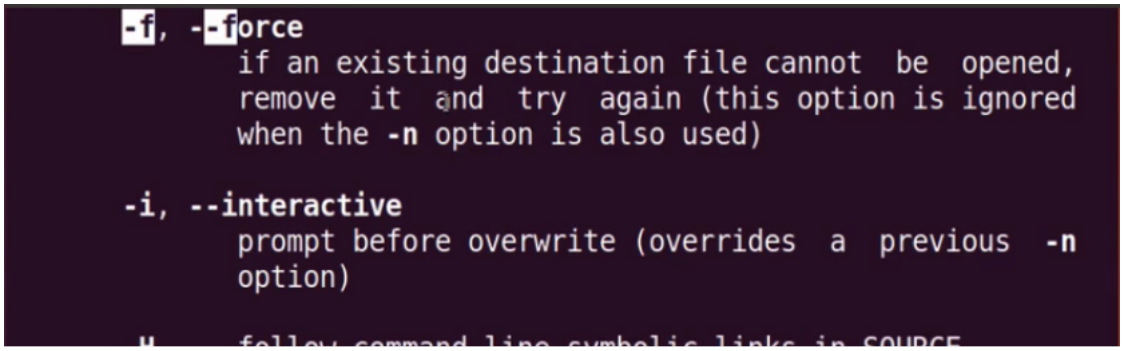


O que acontece é que dentro do diretório `loja21` é criado um diretório chamado `loja`. O que foi que ele fez? A primeira vez que executamos esse comando, tendo sido ele executado com ou sem uma barra, foi criado o `loja21` que copiou tudo o que tem dentro do `loja` para o `loja21`. No momento em que esse diretório já existe, é copiado tudo o que está no diretório `loja` para dentro do `loja21`, inclusive, o diretório `loja`.

Então, temos que tomar cuidado quando estamos copiando recursivamente um diretório, se o que estivermos colocando mais a direita é um diretório que não existe, ele acaba criando esse diretório e coloca os arquivos que estão dentro da fonte como arquivos. Se esse nome de arquivo que está mais a direita já existe, ele cria um diretório com o mesmo nome, por exemplo, `loja` e coloca nele o conteúdo. Esse é um tema extremamente delicado e com o qual temos que tomar muito cuidado. A medida que copiamos esses arquivos, da mesma maneira que temos o `-i`, que serve para dizer que gostaríamos, interativamente que os arquivos fossem subscritos, temos uma outra opção. Repare que quando usamos o `-r` ele copia e ponto final. Algumas coisas ele não consegue copiar por uma questão de permissão, algo que veremos mais adiante.

A outra opção é copiar recursivamente e forçar que não se pergunte se algo está sendo subscrito. Para isso, usamos o `-rf`. Dando um "Enter" em `cp -rf loja loja21` veremos que ele estará subscrevendo sem realizar nenhum questionamento.

Se dermos um `man cp` podemos observar o `-f` que é o `-force`. E o `-f` diz que se o destino não pode ser aberto, ele remove e tenta novamente. Assim, se o arquivo ou diretório de destino no qual estamos tentando criar alguma coisa, não tivermos acesso, não importa, podemos remover e criar ele de novo.



Ou seja, é razoavelmente perigoso esse `-f`, pois, ele vai remover e tentar criar de novo aquele arquivo. Imagine se tivéssemos outras coisas dentro dele, o que poderia acontecer?! Então, cuidado com o `-f`, em qualquer comando que executarmos usando ele temos que ter bastante atenção.

Já falamos sobre o `cp`, sobre o `touch` e `rm`. Falaremos sobre essas questões mais adiante.

Removendo e movendo arquivos com rm e mv

Já vimos o `cp` para copiar arquivos e diretórios e vimos que o `rm` remove arquivos. Vamos, agora, ver muito mais. Analisaremos, também, acerca do `mv`. O primeiro passo é como utilizamos o `mv` para mover arquivos ou diretórios. Estamos em nosso diretório raiz e podemos visualizar nossa lista de coisas através do `ls`:

Repare que temos o arquivo `meuprograma.c` e agora, o que queremos fazer é mover esse arquivo para dentro do diretório `loja`. Para fazermos isso, basta digitarmos `mv meuprograma.c loja/meuprograma.c`. Quando usamos o `mv` podemos

relacionar o seu uso com o do `cp`, portanto, o uso do `mv` é similar ao `copy`. O uso do `mv` é seguido do nome do arquivo fonte e o nome do arquivo de destino e dando um "Enter" nisso ele irá mover o que queremos.

Então, podemos digitar `mv meuprograma.c loja/meuprograma.c` e damos um "Enter". E, se na sequência quisermos analisar se o que fizemos deu certo, podemos perceber isso dando um `ls`. Verificaremos que não teremos mais o `meuprograma.c`, mas se dermos um `ls loja` veremos que ele estará lá:

```
> ls loja
bemvindo2.html        bemvindo.html        contato.html        meuprograma.c
```

Existem outras maneiras de mover, podemos fazer, então, `mv loja/meuprograma.c loja/programa.c`. Você pode estar se perguntando, mas mantemos o arquivo no mesmo diretório e só mudamos o nome disso? Sim! É assim que renomeamos arquivos, para renomear arquivos basta usar o `mv` seguido do nome do arquivo fonte e do nome do arquivo de destino. Então, estaremos ou movendo entre diretórios ou entre partições da máquina, nesse caso das repartição, para um hd, para um pen drive e etc. Nesse caso, realmente, mover será um fenômeno físico, onde as coisas estão passando de um lado para o outro ou podemos utilizar a segunda opção de uso do `mv` que é renomear. Para nós isso fica transparente, portanto, não estamos tão preocupados com isso. O `mv` serve para funcionar dessa maneira, alterar nome para nome, mas também funciona, por exemplo, se estamos no diretório `loja`, para mover para o diretório anterior. Nesse caso digitamos `mv programa.c ../`. Observe:

```
> cd loja
~/loja$ mv programa.c ../
```

Nesse caso passamos um diretório que é o anterior e ele move as coisas para um diretório anterior. É como se fosse um `copy`, mas o `mv` não deixa nada para trás. Vamos sair do `loja`, dando um `cd ..` e se dermos um `ls` veremos a lista de coisas que temos e, inclusive o arquivo `programa.c`

E no `loja` se dermos um `ls` não teremos mais o `programa.c`:

```
> ls
bemvindo2.html      bemvindo.html      contato.html
```

Então, usamos o `mv` junto ao nome de um arquivo e o nome de um diretório de destino, ou, `mv` junto ao nome de um arquivo e nome de um arquivo de destino.

Da mesma maneira que fizemos com o `copy`, se olharmos no manual do `mv`, digitando `man mv` veremos que temos, ainda, outras opções para trabalharmos. Temos o `-b` de `backup`, o `-f` para forçar, o `-i` para interagir. Então, temos diversas características, temos também o `-v` para *verbose*. O `mv`, portanto, é muito similar ao `cp`, ao `copy`.

O `mv`, consequentemente, é bastante utilizado para renomear arquivos e também para renomear diretórios. Imagine que temos a `loja` e queremos renomear ela para a `Loja` que começa com letra maiúscula. Se digitarmos `mv loja Loja` ele renomeara isso. Dando um `ls` podemos conferir isso:

E podemos voltar os nomes se quisermos, basta digitar `mv Loja loja`. Podemos conferir que eles voltaram dando, novamente um `ls`:

Então, esse é o `mv`, usado para mover arquivos e diretórios de um lugar para outro e usado, também, para renomear arquivos e diretórios no mesmo lugar onde eles estão. Podemos, portanto, mover um diretório ? Sim! Podemos pegar o diretório `loja5` e mover para dentro do diretório `backup`. Digitando:

```
> mv loja5 lojabackup
```

Quando fazemos isso qual é a reação? Pegamos esse diretório e colocamos ele lá dentro. Lembre que se fizermos `mv` em `mv loja4 lojax`, como esse segundo diretório não existe, o que o `mv` fará é apenas renomeá-lo, mas se tivéssemos escrito um diretório que já existe, por exemplo, o `mv loja5 lojabackup`, o primeiro estará sendo movido lá para dentro. Vamos olhar o `ls lojabackup`:

```
> ls lojabackup/
bemvindo2.html        bemvindo.thml        loja4
bemvindo2.html~        contato.html        loja5
```

Podemos perceber que o `loja4` e `loja5` estão lá dentro.

Esse é o `mv`, bastante útil no dia a dia. A dica é usar bastante esse comando para que você vá pegando sua utilização no dia a dia, para vermos o que ele faz e como ele faz e para nos acostumarmos com todas as possíveis variações. Usamos, portanto, o `mv`, um espaço, o nome de um arquivo e nome de arquivo, nesse caso estamos renomeando algo e `mv` nome de um arquivo e nome de um diretório, estamos movendo as coisas de lugar. Temos, portanto, diversos usos do `mv`.

Bom, já vimos, portanto, como mover as coisas e como fazemos para apagar agora?

Lembre-se, nós já utilizamos o `rmdir`, se digitarmos `rmdir loja2` e dermos um "Enter" ele falará que falha em remover `loja2` pois não é um diretório vazio:

```
> rmdir loja2
rmdir: failed to remove 'loja2': Directory not empty
```

Para conseguir apagar ele usando o `rmdir` temos que apagar tudo o que temos dentro do `loja2`. Vamos dar um `rm loja2/bemvindo.html loja2/bemvindo2.html` e podemo fazer o mesmo com todos os arquivos, podemos ir removendo, assim, diversos desses arquivos. O que vai demorar bastante ou, podemos dar um `rm loja2/*`. Mas, muito cuidado com o asterisco

quando estamos removendo as coisas. A sugestão é, sempre que você estiver utilizando um asterisco, no momento de remover algo, o melhor é dar um `echo` antes. Podemos digitar `echo loja2/*` :

```
> echo loja2/*
loja2/bemvindo2.html~loja2/bemvindo.html~loja2/contato.html loja2/contato.html~
```

O `echo` na verdade é para confirmar se realmente queremos apagar tudo o que temos dentro e após verificarmos isso podemos digitar `rm loja2/*` com mais segurança. E, com isso, teremos removido tudo o que está lá dentro. Agora, que removemos tudo o que está dentro do `loja2` podemos usar o `rmdir loja2` .

Vamos, novamente verificar o `ls` e ver o que criamos anteriormente para as explicações e o que ainda constam em nossa lista e que desejamos remover:

Poderíamos dar um `rm loja3/*` , mas ao em vez de escrever dessa maneira e termos que fazer um `echo` antes para verificar quais os arquivos que temos e se realmente queremos apagá-los, podemos escrever `rm -i loja3/*` que ele nos perguntará, arquivo a arquivo, se queremos, verdadeiramente, remove-los. No caso de querermos remover os arquivos específicos que ele nos pergunta, responderemos com um `y` de "yes". Observe:

```
> rm -i loja3/*
rm: remove regular file 'loja3/bemvindo2.html'? y
rm: remove regular file 'loja3/bemvindo.html'? y
```

E agora sim, podemos remover o diretório `loja3` , então, podemos deletar `rmdir loja3` . E se dermos um `ls` poderemos ver que ele não consta mais em nossa lista:

O que mais que queremos fazer? Podemos remover de uma vez só o `loja20` . Se dermos um `ls loja20` :

```
> ls loja20
bemvindo2.html        bemvindo.html        contato.html
```

Podemos remover tudo o que temos dentro do `loja20` usando o `rm -r loja20` . Quando entramos no recursivo ele encontra que temos um arquivo protegido da escrita e ele nos pergunta se temos a certeza de que queremos remover ele. Quando o recursivo fala que encontra um arquivo que está protegido para escrita, ou seja, é porque existem questões de permissão envolvidas. Ele, portanto, nos questiona se temos a certeza de se queremos remover o arquivo. Falaremos sobre permissão mais adiante, porém, em relação a remoção desse arquivo, não precisamos de permissão para escreve-lo, precisamos de permissão para escrever no diretório atual e se temos essa permissão para escrever no diretório atual, pois somos os donos desse diretório atual, podemos remover arquivos do diretório atual. O que não significa que possamos escrever nesse arquivo. Falaremos bastante sobre isso mais adiante.

Podemos não ter a permissão de escrita em um arquivo sem permissão, como esse que temos, porém, como somos os donos do diretório de onde esse arquivo está, podemos remover esse arquivo.

Então, são vários os cuidados que temos que tomar quando quisermos trabalhar com permissões. Falaremos desse tópico mais adiante. Podemos remover os arquivos digitando:

```
rm -r loja20
```

E responderemos com um `y`, de "yes", quando ele nos perguntar se queremos apagar algo.

Como podemos observar ele perguntará se queremos remover os dois arquivos. Poderíamos ter digitado, usando além do `-r`, também o `-i` e o `-v`, `rm -riv loja21`. Ele primeiro nos perguntará se queremos remover o diretório `loja21`, que responderemos "yes", usando o `y`. Ele nos perguntará se queremos apagar o arquivo regular `loja21/bemvindo2.html` e responderemos, novamente, `y`. Ele, ainda, perguntará se queremos entrar no diretório `loja21/.git`, nesse caso responderemos que não, e as outras duas perguntas responderemos "no", usando o `n`. Observemos:

```
> rm -riv loja21
rm: descend into directory 'loja21'? y
rm: remove regular file 'loja21/bemvindo2.html'? y
removed 'loja21/bemvindo2.html'
rm: remove regular file 'loja21/bemvindo2.html'? y
removed 'loja21/bemindo.html'
rm: descend into directory 'loja21/.git'? n
rm: remove regular empty file 'loja21/contato.html'? n
rm: descend into directory 'loja21/loja'? n
```

Utilizando essa opção é simples: entramos onde queremos e não entramos onde não queremos. Então, o `-v`, o *verbose* mostra o *verbose* do que foi removido. O `-i` pergunta se queremos entrar nos diretórios, se queremos apagar os arquivos e o `-r`, entra recursivamente. É bastante comum misturarmos todas essas opções junto ao `rm`. Precisamos tomar cuidado com o `rm`. Da mesma maneira que vimos o `-f` de `-force` que força para que não se perguntem as coisas, temos o `rm -f`, que força e sai apagando as coisas. Vamos observar o diretório `lojabackup` digitando `ls lojabackup/`:

```
> ls lojabackup/
bemvindo2.html        bemvindo.html        loja4
bemvindo2.html~        contato.html        loja5
```

Então, podemos usar o `rm -frv lojabackup` para forçar, isto é, sem perguntar nada e de maneira recursiva, o diretório `lojabackup` e tudo o que tem dentro. Como podemos observar, temos, ainda o `v` de verbose. Observemos:

O que quisemos dizer com isso é que o `rm -frv lojabackup` será forçado recursivamente, o que é extremamente perigoso. Mais perigoso ainda porque, copiamos o diretório `loja21` para o `lojabackup` digitando `cp -r lojabackup loja21` e vamos apagamos o `lojabackup`. Vamos observar o `ls`:

É comum as pessoas fazerem `rm -fr lojabackup/*` e isso funciona. Ele irá remover tudo o que tem no `lojabackup`, mas, não vai remover o `lojabackup`. O uso do asterisco, entretanto, é muito perigoso, pois, imagine que o que queríamos, na verdade, é apagar além de tudo o que tem dentro do `lojabackup` o próprio `lojabackup`. Poderíamos escrever `rm -fr lojabackup *`, se dermos um espaço, sem querer, entre o `lojabackup` e o `*` isso vai dar um grande problema! Ele vai não apenas irá remover o `lojabackup`, mas também todo o diretório atual, isso acontecerá por causa do *globbing*. Ou seja, usar o `-fr` pode ser muito perigoso.

Pior ainda, o que acontece se fizermos o `rm -fr lojabackup /` ? Onde temos um espaço entre `lojabackup` e `/` , bom, isso é um problemão, começaremos apagando do diretório `raiz` e tudo o que estiver no caminho, então, temos que tomar muito cuidado com essas duas opções juntas. Escrever `-fr` ou `-rf` dá na mesma, pois eles continuam a passar as informações de "forçar" e "recursivo". Os dois são a mesma coisa, o mesmo infernal perigo. Então, é aconselhável que quando utilizemos o `-fr` usemos junto o `-i` , o interativo e retiremos o `-f` , ou pelo menos, se for utilizar o `-fr` usemos também o `-v` de *verbose* para que você possa visualizar o que vai ser feito de errado e de certo. Preferível é, `-r` , de recursivo e `-i` de interativo para acompanhar o `rm` . Para verificarmos que aquilo que estamos removendo é, efetivamente, aquilo que queríamos.

Sempre que removermos algo, realizamos uma ação destrutiva, podemos estar subscrevendo ou apagando um arquivo que, na verdade não queremos. Por isso, é muitas vezes interessante utilizarmos o `-i` para tornar interativo, principalmente, se vamos utilizar o *globbing*.

Esses são os comando que a prova irá cobrar de nós nessa seção. Temos, portanto, o `mv` para mover as coisas, `cp` para copiar, `rm` para remover e `touch` para mudar a data de última atualização de um arquivo ou diretório. O `mkdir` serve para criar diretórios e o `rmdir` para remover eles. Com isso, completamos todo o tópico dois da prova.