05

Caminhos absolutos, caminhos relativos, e o ls

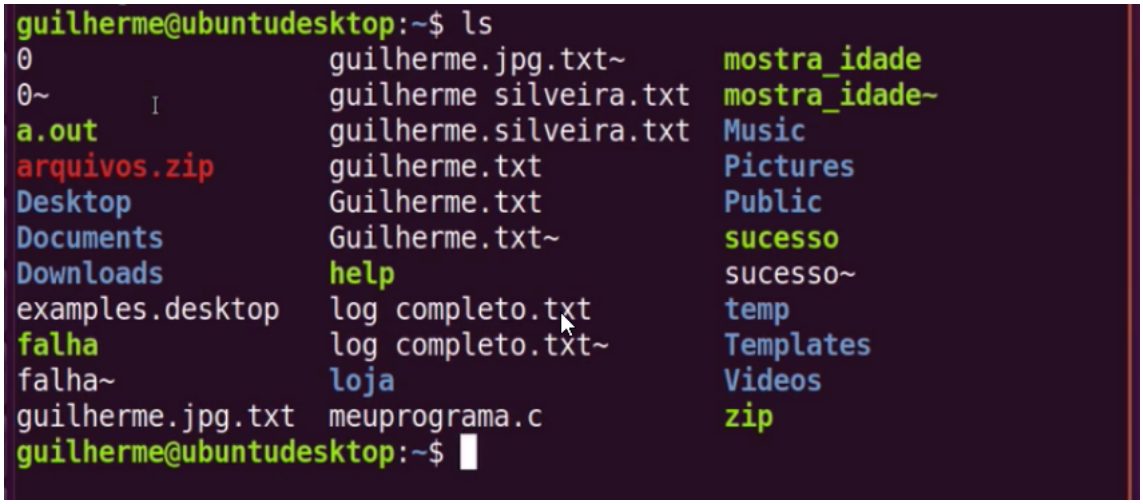
Caminhos absolutos e relativos. Atalhos para diretórios e a home

Vamos continuar falando sobre arquivos e diretórios, já falamos sobre nomes possíveis e válidos para diretórios. Lembrando que já falamos que o carácter barra, "/", não é algo válido para nomes de arquivos. O "/" representa uma separação de diretórios e o barra puro representa o diretório raiz, que chamamos `root`, que não é o "superusuario" `root`. Não possui relação o "superusuario" `root` com a raiz, o diretório "/".

Vamos verificar o diretório atual, podemos visualiza-lo a partir do `pwd` que é o `/home/guilherme`.

```
> pwd
/home/guilherme
```

Vejamos os diversos arquivos e diretórios que temos no `ls`:



Como podemos observar o que o pwd nos diz?

Em `/home/guilherme`, a `/` indica o diretório raiz e dentro dele estamos no `home` e dentro do `home` estamos no `guilherme`. Então, temo que este é o caminho para o diretório atual, ou seja, p `/home/guilherme`. Por isso, se queremos executar um programa que está nesse diretório, por exemplo, o `cat sucesso` ele mostrará que estamos "no sucesso":

```
> cat sucesso
echo Estou no sucesso
exit 0
```

Como fazemos para executar o `sucesso`? Podemos, simplesmente falar o caminho inteiro do programa que queremos executar, e para isso digitamos `/home/guilherme/sucesso` e executaremos o programa.

```
> /home/guilherme/sucesso
Estou no sucesso
```

O que estamos falando ao digitar o caminho inteiro é que o programa deve ser buscado no diretório `/home/guilherme` e que ele se chame "sucesso". Se apenas escrevemos sucesso, onde ele seria buscado?

```
> sucesso
sucesso: command not found
```

Ele buscaria "sucesso" no `path`, isto é, na variável de ambiente e, evidentemente, ele não encontrará nada no `path`. Vamos observar a variável de ambiente `path` e para tanto digitaremos `echo $PATH` e teremos o seguinte:

```
> echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

A partir do que ele nos traz concluímos que não existe o diretório `/home/guilherme` no `path`. Então, se quisermos executar esse comando temos que falar explicitamente todo o caminho, absoluto. desde a raiz até a coisa que desejamos. Então se por exemplo, temos o diretório loja, vamos mudar para esse diretório digitando `cd loja` e vamos dar um `ls` para ver os arquivos que temos nesse diretório. Constataremos que é apenas um, o `bemvindo.html`. Observe:

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

Vamos, agora, retornar ao nosso diretório anterior, digitando `cd ..` e observaremos o arquivo `bemvindo.html`. Como podemos fazer para ver o conteúdo desse arquivo? Vamos digitar:

```
cat /home/guilherme/loja/bemvindo.html
```

E damos um "Enter". Teremos o seguinte:

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

Então, esse é o caminho do `path` absoluto para esse arquivo: `/home/guilherme/loja/bemvindo.html`. Repare que já estamos no diretório `/home/guilherme`. Basta para confirmar isso digitar `pwd` que ele nos contestará `/home/guilherme`, então, podemos apenas escrever o caminho relativo ao diretório atual, isto é, basta falarmos `cat loja/bemvindo.html` e teremos a mesma resposta:

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

Ele nos mostra o `bemvindo.html`, pois, chegamos até ele indo por um caminho relativo. Então, o caminho, o `$PATH` pode ser absoluto, se começamos com a `/` ou se começarmos sem a `/` ele é relativo, e relativo ao diretório atual.

Assim, se começarmos usando o caracter da barra ele é absoluto, como no caso do `/home/guilherme/loja/bemvindo.html` e se não começarmos usando este carácter é relativo, como em `loja/bemvindo.html` .

Por isso que se entrarmos no diretório loja, digitando `cd loja` e logo em seguida dermos um `cat bemvindo.html` , estaremos utilizando um caminho relativo ao diretório atual e ele nos mostrará o que estamos buscando. Observe:

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

Se voltarmos mais um diretório digitando `cd ..` estaremos novamente no diretório `/home/guilherme` e se quisermos ver aquele mesmo arquivo podemos digitar para isso `cat guilherme/loja/bemvindo.html` :

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

Esse caminho que acabamos de digitar é o meio relativo de chegar ao diretório atual.

E se quisermos ver o caminho absoluto? Basta digitar `cat /home/guilherme/loja/bemvindo.html` e teremos o seguinte:

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

Ele nos mostrará o que pedimos!

De novo, os `PATHS` que utilizamos, os caminhos, podem ser ou absolutos se iniciam com uma barra ou podem ser relativos se estão sem esse carácter. Portanto, é muito fácil dizer se um `PATH` do qual estamos falando é algo absoluto ou relativo, basta olhar se ele possui ou não uma barra.

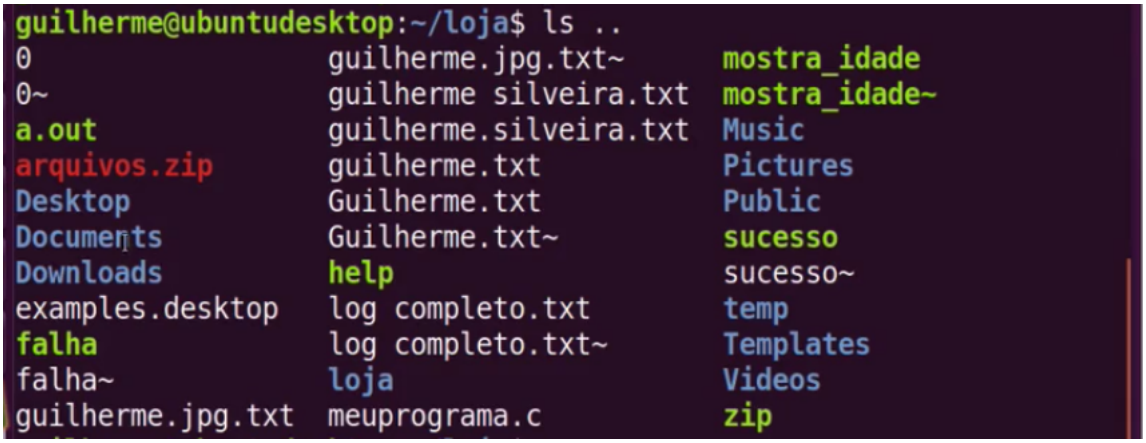
O caminho relativo é interessante pois ele torna as coisas mais curtas e não apenas isso, através do relativo conseguimos executar *scripts* e programas relativos ao diretório onde estamos, podemos executar algo em um diretório, outra vez em outro diretório, uma próxima vez, ainda, em outro e assim por diante. Mas, talvez não queiramos isso, talvez queiramos um lugar absoluto e por isso, utilizamos a barra.

Lembrando que, se usarmos o relativo, ele será relativo ao diretório atual. E se usarmos o absoluto será em um único lugar, será absoluto quando começa com a barra e no diretório raiz.

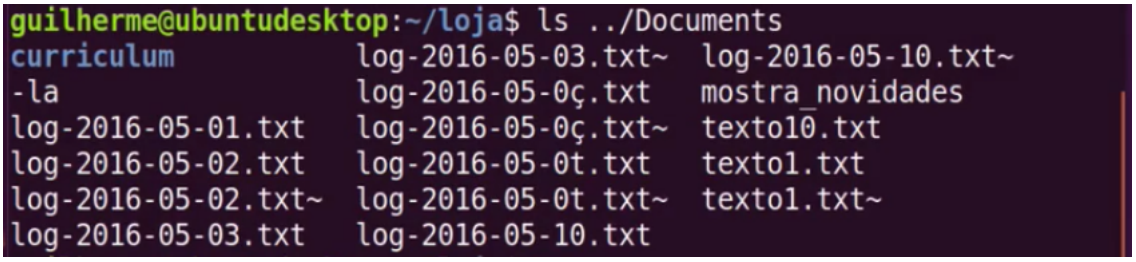
Além disso, existem dois outros casos que acabam aparecendo aos poucos e que são importantes para nós. Se entrarmos no `cd loja` e queremos voltar ao diretório `/home/guilherme` , podemos digitar `cd /home/guilherme/` e voltamos ao diretório anterior. Mas essa situação pode ser um pouco complexa, ter sempre que saber qual é o diretório anterior para poder voltarmos, bom... é complicado! O que podemos fazer é utilizar um atalho para o diretório anterior. O diretório anterior se chama "ponto ponto", isto é, `..` e este é o atalho para o diretório anterior. Assim, se digitarmos `cd loja` estamos nos referindo ao diretório `loja` e se digitarmos `cd ..` é sobre o diretório anterior.

Repare que se estamos no diretório `home` e se digitarmos `cd ../etc` ele entrará no diretório `etc`. E se quisermos voltar ao diretório `/home/guilherme`, podemos digitar `cd ../home/guilherme`.

Se entrarmos no `cd loja` e dermos um `ls ..` veremos a lista do diretório anterior:



Repare que temos como diretórios anteriores o `loja` e o `Documents`. Agora, vamos pegar tudo o que tinha no diretório anterior e vamos entrar no diretório `Documents`, então, estamos no diretório atual e voltaremos no diretório anterior e entramos no `Documents`. Para isso digitaremos `ls .. /Documents` e então, podemos ver o que temos no `Documents`:



Repare o poder de usar caminhos relativos! Ele nos dá o poder, baseado no diretório atual, de voltarmos ao diretório anterior e entrarmos de novo, ou simplesmente sair entrando em diretórios, esse é o poder que o relativo nos dá. E o absoluto nos concede o poder de, independentemente, de onde estamos, acessar alguma coisa da maneira que queremos e explicitamente. Temos, portanto, o poder do absoluto e o poder do relativo. Para o `absoluto` temos `..` como um atalho para o diretório anterior. E se estivermos no diretório `/home` e falarmos `cd /home/guilherme/loja`, ele entrará no `loja`. E se quisermos voltar ao `home` digitaremos `cd /home` e se quisermos entrar no diretório `loja` e para isso voltarmos no diretório anterior e entrarmos no `Documents` digitaremos:

```
~/home$ cd /home/guilherme/loja../Documents
~/Documents$
```

E isso funciona! O `..` não precisa ser em um começo de um diretório ou de um caminho relativo, ele pode ser no meio de um caminho relativo. E até parece um pouco estranho isso que acabamos de fazer. Escrevemos `loja` para na verdade voltarmos no diretório , realmente, parece um pouco estranho ter feito isso. Mas as vezes podemos ter uma variável que se chame `LOJA` e ela pode ser igual a `/home/guilherme/loja`, por exemplo:

```
> LOJA=/home/guilherme/loja
```

Vamos supor que queremos entrar no diretório `home` e depois entrar no diretório `loja` e, por fim, voltar para o `Documents`. E se escrevermos `cd $LOJA` ele entrará no diretório `loja`.

```
> cd $LOJA
~/loja
```

Vamos voltar ao `cd /home` para verificar uma coisa, e se escrevermos `cd $LOJA/../Documents`, o que ele faz? Ele vai para o diretório `LOJA` e depois volta para o diretório `Documents`:

```
~/home$ $LOJA/../Documents
~/Documents$
```

Ele entrou no `Documents`. Perceba que nesse contexto o `..` faz sentido. Pois, não sabemos qual diretório essa variável está nos apontando, mas usamos ela por diversos motivos. Utilizar a variável `..` pode ser útil.

No caso do `/home/guilherme/loja../Documents` pode ser pouco útil, mas, o "ponto ponto" costuma ser utilizado, com uma variável que é muitas vezes mais útil. O `..` não precisa, necessariamente, ser utilizado no começo, ele pode ser usado também no meio de um caminho e podemos usar também junto com variáveis pois elas serão, primeiro, interpretadas pelo *shell* antes de executarem-se os comandos.

Existe, ainda, outro atalho para que gostaríamos de mostrar.

Vamos entrar no diretório padrão o `/home/guilherme` e nesse diretório temos um *script*, o "sucesso". Como faziamos para executá-lo? Digitávamos `/home/guilherme/sucesso` e tínhamos:

```
> /home/guilherme/sucesso
Estou no sucesso
```

Repare que começamos com barra, isto é, começamos com um caminho absoluto. E se quiséssemos começar com um relativo? Vamos ir para o `home` e lá digitamos `guilherme /sucesso`:

```
/home$ guilherme/sucesso
Estou no sucesso
```

Então, ele também está sendo executado quando utilizamos o caminho relativo. Ele também está funcionando!

Agora, entramos no diretório `guilherme`, digitando `cd guilherme` e tentamos executar `sucesso`:

```
/home$ cd guilherme
sucesso
sucesso: command not found
```

Vimos que não funciona!

Mas, lembre-se já vimos isso diversas vezes, executar pelo nome do diretório atual não funciona, porque se olharmos a variável `$PATH` não fala para buscarmos no diretório atual. Vamos digitar `echo $PATH` e observar:

```
> echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

Ela fala, simplesmente, para buscarmos nesses diretórios que especifica. Se digitarmos apenas um nome que é de um *script* ou de um programa, ele não procura no diretório atual, só procura nesses que foram especificado acima e não encontra nada. Se escrevermos de maneira absoluta, `/home/guilherme/sucesso` estaremos dizendo exatamente onde está o que buscamos para executar.

```
> /home/guilherme/sucesso
Estou no sucesso
```

Se dermos um `cd ..` e digitarmos de maneira relativa o caminho, como `guilherme/sucesso` estamos dizendo para o *shell* que é para executar isso nesse caminho específico. Ele encontra, pois, não procura no `PATH`:

```
/home$ guilherme/sucesso
Estou no sucesso
```

Agora, se entrarmos no diretório, dando um `cd guilherme/` e digitarmos apenas `sucesso`, não estamos dizendo absoluto, pois não iniciou com o caracter da barra, nem estamos dizendo relativo, pois não colocamos uma barra no meio. Estamos dizendo, simplesmente, "esse é o nome" e "busque no PATH". Ele buscará por "sucesso" e não encontrará nada e teremos `sucesso: command not found`. E como fazemos para referenciar relativamente, assim como temos o `..` para nos referirmos ao diretório anterior, como fazemos para falar do diretório atual?

O diretório atual é, simplesmente ".", ou seja, um ponto. E ao utilizarmos este ponto, em `./sucesso` estaremos falando para executar no diretório atual o "sucesso". Observe:

```
>./sucesso
Estou no sucesso
```

E se estamos no diretório `loja` e queremos executar algo no diretório anterior? Basta digitar `../sucesso`:

```
> cd loja
~/loja$ ../sucesso
Estou no sucesso
```

Com isso estamos dizendo, execute no diretório anterior o `sucesso`. Pegue o `PATH` relativo e execute o sucesso.

E, se estamos no diretório atual e digitamos `./sucesso` ele executará também:

```
~/loja$ cd ..
./sucesso
Estou no sucesso
```

Assim, vimos que `..` é um atalho para o diretório anterior e somente um `.` é atalho para o diretório atual.

Poderíamos utilizar isso de uma maneira distinta? Por exemplo `cd/home/guilherme/./loja`?

Podemos usar! Mas, como vimos antes, parece ser bem inútil se utilizarmos dessa maneira o `.` e mesmo acompanhado de uma variável parece não ser muito funcional. Antes, o `..` pelo menos voltava um, agora o `.` é a mesma coisa que não falar

nada. Nessa situação mesmo que usassemos a variável, não teríamos uma grande vantagem, mas se quisessemos fazer uso disso poderíamos, como em `cd/home/guilherme/./loja` . Assim, como usamos o `..` em outras situações.

Muitas vezes nos referimos ao `..` como dois pontos, mas temos que tomar cuidado para que não seja confundido com ":". Nessa aula especificamos nos referimos como "ponto ponto" quando queremos nos referir ao diretório anterior mas, repare que muitas vezes e, incluisive, nas próximas aulas e no mundo real, encontraremos as pessoas se referindo como "dois pontos". Assim, falaremos `cd` , "dois pontos" para nos referirmos a `cd ..` , mas não confunda com `:` .

Vamos ver em que diretório estamos após digitarmos o `cd ..` :

```
~/loja$ cd ..
> pwd
/home/guilherme
```

Repare que se estamos no diretório `/home/guilherme` ele não possui nenhuma referência para dizer que estamos nesse dirretório. Temos indicações de que estamos no diretório loja através do `~/loja$` assim como também temos uma referência do `home` que é `/home$` . Por que no `/home/guilherme` ele não traz nada que marque que estamos nesse diretório?

Lembra-se que quando vimos as variáveis tínhamos uma que se chamava `$HOME` que é a pasta do diretório base do usuário que é o `/home/guilherme` . O diretório base do usuário possui `/home` e é seguido do nome do usuário e o login do usuário. Esse é o padrão do *Linux*, `/ , home , / , nome do usuário` , exceto para o usuário `root` . O diretório `home` para o usuário `root` é o `/root` . Vamos digitar `cd /root` e ver o que ele nos traz:

```
> cd /root
bash: cd: root: Permission /root
```

Nós, como usuários normais, não temos como ter acesso ao `root` , veremos isso quando abordarmos a questão de permissão.

Então, quais são as duas regras que temos que lembrar de diretório base? O diretório base do usuário geral é `/home/nome do usuário` e o do usuário `root` é apenas `/root` .

Quando estamos em um diretório qualquer e desejo voltar para a `home` basta digitar `cd` e "Enter". Quando damos um "Enter" no `cd` vazio ele vai para o diretório de nossa `home` que é o `/home/guilherme` ou para o diretório `/root` ou para qualquer diretório que for configurado na sua `home` . Para ir nesse diretório basta digitar `cd` e darmos um "Enter".

Então, se estivermos no diretório `loja` e quisermos executar um arquivo que está no diretório `home` sabemos que para executar algo que está no diretório anterior basta digitar `..` e se quisermos algo que esteja no atual, digitamos o `./` , temos, respectivamente, o que é relativo ao diretório anterior e relativo ao diretório atual. E, agora, se quisermos falar, relativo a `home` , como falamos isso? É só, na verdade, não falar nada, basta digitar `cd` e dar um "Enter".

Mas se digitarmos `/sucesso` ? Não estaremos falando nada. Não estaremos falando para executar algo que está na raiz, deixar isso vazio não pode, usamos o `./sucesso` para falar no diretório atual ou `../sucesso` para nos referirmos ao diretório anterior.

Faltou um símbolo para dizermos na "minha raiz". Então, como o *shell* vai traduzir algo para se referir a `raiz` ? Usamos o til, `~` . Se digitarmos `~/sucesso` estaremos nos referindo a algo que estiver na raiz, o `~` , portanto, significa relativo a minha raiz, a minha `home` , a base. Com isso, estaremos dizendo que é relativo a base que queremos executar o sucesso. Observe:

```
~/loja$ ~/sucesso
Estou no sucesso
```

E ele executará!

E se estamos no diretório `home` e queremos entrar no diretório `/home/guilherme/loja` ? Podemos digitar `cd ~/loja` e como isso estaremos dizendo através do `~` que dentro da `home` queremos entrar em `loja` e então, ele entra. Se dermos um `pwd` poderemos visualizar isso:

```
/home$ cd ~/loja
~/loja$ pwd
/home/guilherme/loja
```

Repare, é por isso que o nosso atalho mostra o til antes de `loja` . O `~` é uma referência a `home` .

Retomado: O que vimos até agora?

Conseguimos trabalhar com diretórios e arquivos de maneira absoluta, para uma representação absoluta começamos utilizando uma barra, `/` , ou para uma representação relativa, basta dizemos de maneira "literal" o que queremos, ou seja, sem usarmos a barra. O absoluto nos permite dizer as coisas de uma maneira absoluta, isto é, teremos a certeza de que ele encontrará a partir do absoluto. E o relativo é referente ao diretório atual, seja lá onde estivermos.

Em nosso caso, exploramos diversas características do relativo, vimos que se quisermos executar algo relativo ao atual podemos colocar o nome do diretório, uma barra, o nome do diretório, barra, o nome de outro diretório e assim por diante e referenciaremos o que está dentro disso.

Mas, se quisermos executar algo que está no diretório atual, precisamos de algo para representar o diretório atual e utilizamos, para isso, o `.` . Por exemplo `./sucesso` , com isso estamos dizendo que é para executar, no diretório atual, o que tem o nome "sucesso".

Se digitarmos `cd /home/guilherme/loja` , a barra indica algo dentro, isto é, isso vai ser lido como, "no diretório home, queremos entrar no diretório guilherme e nele queremos entrar no diretório loja".

Então, para dizer "diretório atual" usamos o `.` e para "alguma coisa aí dentro" usamos a `/` seguido do nome do que queremos.

Para nos referirmos ao diretório anterior usamos o ponto ponto, o `..` , ou, por vício de linguagem falamos "dois pontos".

E se quisermos referenciar o diretório `home` , a base do diretório do usuário? Usaremos o `~` e ele executará na `home` , por exemplo `~/sucesso` .

Então, `home` , é o diretório base, o `/home/guilherme` , que é `/home/` acompanhado do nome do usuário e isso pode ser reconfigurado, trabalhado e isso, especificamente, não cai na prova.

O que precisávamos saber é como trabalhar com caminhos absolutos e relativos e esses atalhos, o `.` , `..` e o `~` e um cuidado muito importante que é quando queremos nos referir ao diretório atual digitamos `./` .

Buscando ajuda no dia a dia

Vamos observar um pouco mais a fundo um comando que utilizamos até o momento. É isso que gostaríamos que fosse feito desse momento em diante, veremos diversos comandos que utilizamos no dia a dia. A medida que formos utilizando esses comandos no cotidiano a recomendação é que vocês se detenham a analisar os seus respectivos manuais e também que comecem a buscar informações no `help` dos comandos.

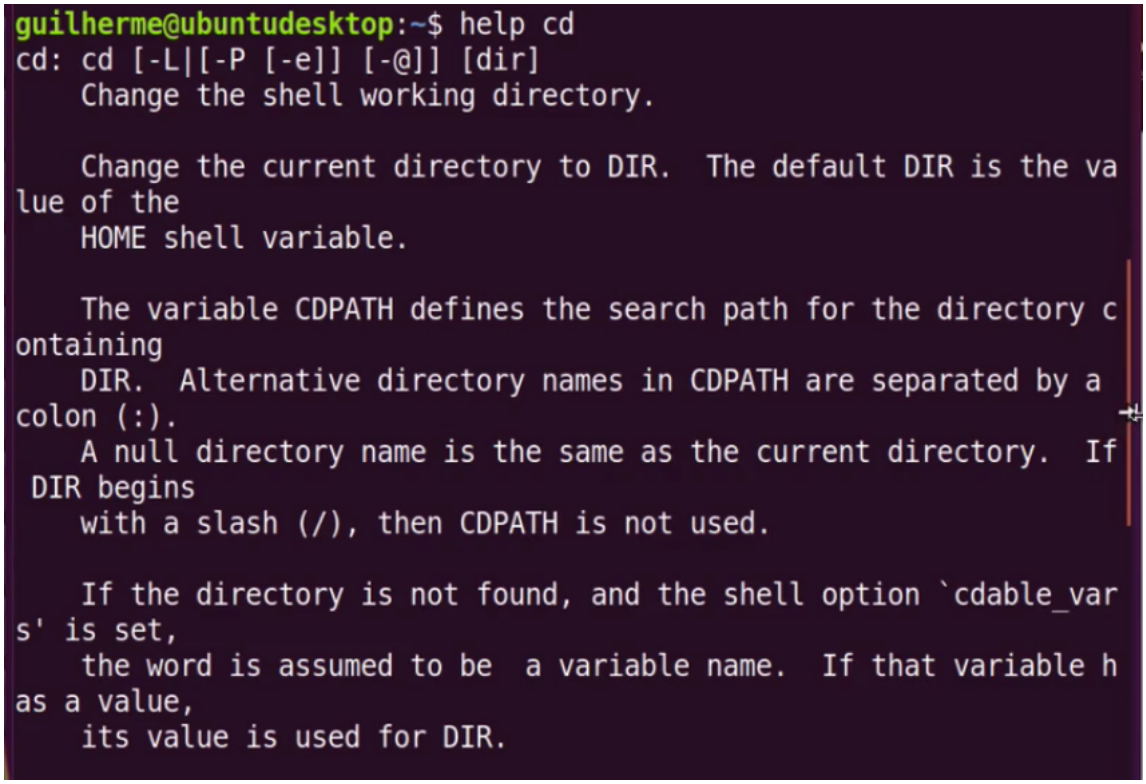
Vamos observar, primeiramente, um comando que já utilizamos bastante, o `cd` .

Dando um `pwd` podemos verificar que estamos no diretório atual, o `/home/guilherme` . E como queremos entrar em outro diretório usaremos o `cd` , daremos um espaço e acrescentaremos o local onde queremos ir, por exemplo `cd loja` e iremos ao diretório `loja` .

Vimos também que o `cd` comporta os dois pontos, `..` . Vimos também que o `cd` é um `shell builin` e para comprovar isso basta digitarmos `type cd` :

```
> type cd
cd is a shell builtin
```

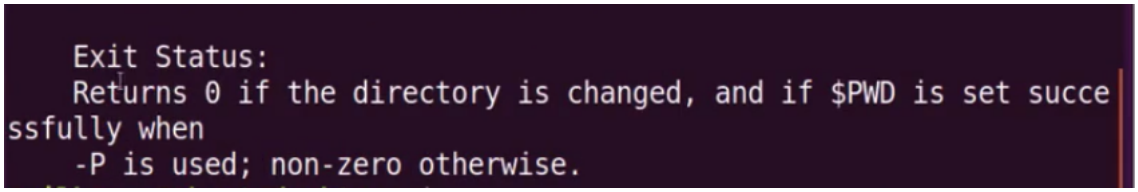
E se dermos um `help cd` veremos que ele mostra a ajuda do `cd` .



Nesse momento estamos interessados em observar dois pontos, o primeiro é que o `help` nos fornece diversas informações e tipos de opções que podemos passar para o `cd` . Podemos observar que o `help` nos informa que o diretório é algo opcional junto com o `cd` , pois se não passarmos nada ele acaba entendendo que é para ir para a `home` , conforme vimos anteriormente.

Ele explica também como é o funcionamento de entrar em um diretório ou em outro, menciona, inclusive, que podemos utilizar uma variável de ambiente para isso. O `help` nos explica diversas coisas, inclusive, de uma maneira ainda mais complexa do que apenas mencionar o absoluto e o relativo. Em nosso caso, não estamos muito preocupados em nos ater aos diversos detalhes que essas documentações trazem, mas, o que queremos sublinhar é que é importante lembrarmos de olhar as documentações. Vamos observar, rapidamente, por exemplo, o *status* de saída. Temos a informação de que o retorno será

igual a "0" se foi um sucesso e se a variável `$PWD` foi configurada adequadamente, portanto, nesse caso ele devolverá "0", caso contrário, teremos um outro código de saída que indica algum erro.



E se fizermos `cd /home` e digitarmos o código de saída `echo $?` veremos que essa informação é confirmada, pois, teremos que o código de saída será "0":

```
> cd /home
/home$ echo $?
0
```

Agora, vamos voltar usando o `cd .` E se fizermos um `cd /home/udhasdhiaus`, ou seja, nos referirmos a algo que não existe? Se pedirmos o código de saída teremos uma reposta que é diferente de "0", no caso, "1":

```
> cd /home/udhasdhiaus
bash: cd:/home/udhasdhiaus: No such file or directory
> echo $?
1
```

Repare que esse tipo de informação conseguimos aprender se utilizamos e lemos o `help`. As vezes essas informações nem mesmo são encontradas nas documentações, as vezes, elas estão em outros locais. E essas informações não vão significar que sempre, todos os comandos em todos os *Linux* irão seguir esse padrão.

Um exemplo disso e que é muito comum é, por exemplo, estamos trabalhando no diretório `loja` e agora queremos ir no diretório `/etc` e usamos o `cd /etc` para ir nesse diretório e após trabalharmos nesse diretório temos o interesse de voltarmos ao diretório `/home/guilherme/loja`. Podemos nem lembrar qual era o diretório que estávamos antes, o último diretório que utilizamos. Para isso digitaremos `cd -`. O hífen é utilizado para voltarmos ao último diretório que estávamos utilizando. Observe:

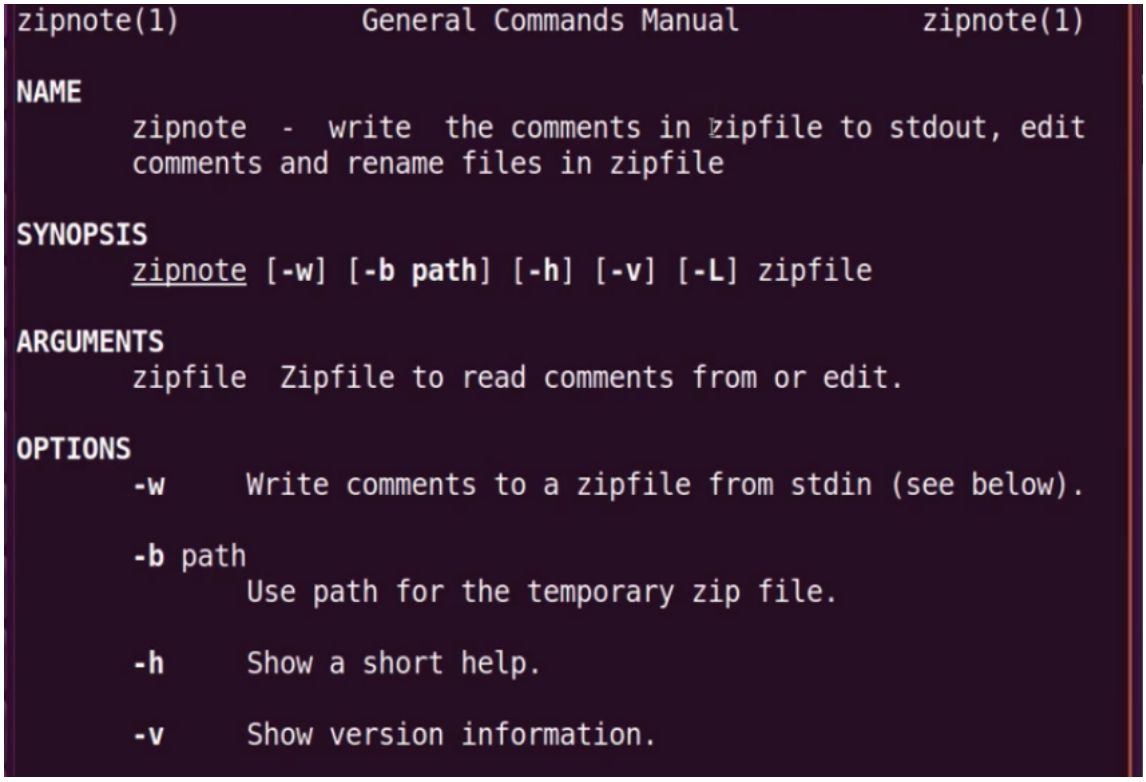
```
~/loja$ cd /etc
/etc$ cd -
/home/guilherme/loja
~/loja$
```

Repare que se dermos um `cd -` não voltaremos para o penúltimo diretório, voltaremos para o último que estávamos, o `/etc`. E se agora estamos nesse diretório `/etc` e queremos voltar no último voltaremos usando o `cd -` e retornaremos ao `loja`. E, uma vez no `loja`, usando novamente o `cd -` voltaremos ao `etc` e assim por diante:

```
~/loja$ cd -
/etc
/etc$ cd -
/home/guilherme/loja
~/loja$ cd -
/etc
```

Repare que essa informação específica não encontraremos no `help cd`. Isso não significa que todos os `cds` de todos os *Linux* irão suportar o hífen. Não podemos falar isso com toda a certeza, pois não temos isso documentado, mas na prática, eles acabam suportando essa ação de trocar o diretório atual pelo último diretório utilizado.

Um dos pontos mais importantes que gostaríamos de passar aqui é que ler a documentação dos comandos utilizados é muito interessante, leia o `help` e também o `man` das informações dos comandos. Leia uma parte a cada vez que utilizar um comando, um pouco a cada dia. Por exemplo, suponhamos que é a primeira vez que você utilizará o `zip`, aproveite para, nesse momento, dar um `man zip` e ler um pedacinho das informações que ele contêm. Você pode verificar, por exemplo, que o `zip` tem o `zipcloack`, o `zipnote` e o `zipslip` e em uma próxima vez que utilizar esse comando você já pode aproveitar e ler informações acerca desses três. Assim, na próxima vez que você rodar o `zip` lembre de pedir o `man zipnote` e observar o que ele nos traz:



No caso ele adiciona os comentários que estão no `zipfile`, na saída, para lermos os comentários que estão dentro do `zip`

E utilizando esse método, cada vez que você for lendo a respeito dos comandos você vai aprendendo um pouco mais acerca deles. Não é que o `zipnote` vá cair necessariamente na prova, mas é simplesmente dizer: podemos aprender no dia a dia a medida que utilizamos comandos e programas.

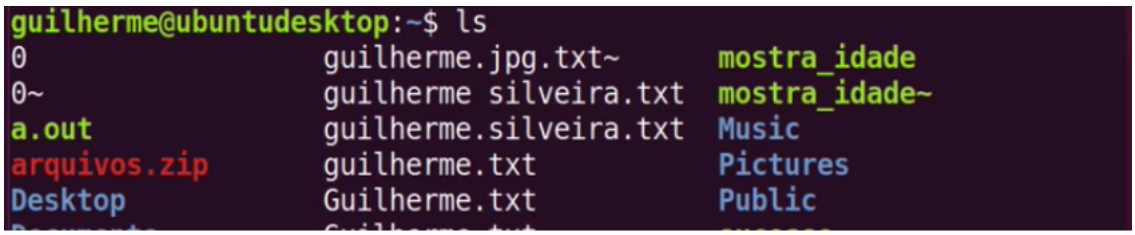
Com isso, falamos sobre diretórios e arquivos, mencionamos o diretório raiz e também os caminhos, os `paths`, absolutos e relativos, falamos do `.` e do `..`, do `cd`, da `home` e o `~`. Perpassamos também pelos arquivos invisíveis e mencionamos isso quando falamos sobre os nomes de arquivos que são válidos ou inválidos, o que é aceito para nomear arquivos e diretórios e o que não é.

Falta, ainda, falarmos mais acerca do `ls` e dos arquivos e diretórios que estão dentro dele. Vimos isso acontecendo aos poucos e seguiremos avançando nesse tema e na importância do `ls`.

Opções do ls

Vamos observar o `ls`, analisar as opções comuns do `ls` e abordar mais profundamente sobre diretórios e arquivos escondidos.

Primeiramente, se queremos listar arquivos usaremos o comando `ls` :



O `ls` já traz diversas informações nas coisas que colore para nós. É comum que cada distribuição já venha com algum tipo de atalho que configure alguma opção como essa, em nosso caso, as coisas aparecem coloridas. Como fazemos para garantir que estamos utilizando o `ls` padrão?

Queremos utilizar um `ls` que seja padrão para termos a certeza de que as opções que passarmos durante esse curso são exatamente iguais as que deverão existir em qualquer outro *Linux* padrão. Como podemos fazer isso? Vamos dar um `type ls` e ver o que ele nos contesta:

```
> type ls
ls is aliased to `ls --color=auto`
```

Percebemos que o `ls` está como um `aliased` do `ls --color=auto` . Vamos dar um `type -a ls` para verificarmos todas as variações do `ls` . Podemos observar que ele nos mostra dois `ls` :

```
> type -a ls
ls is aliased to 'ls --color=auto'
ls is /bin/ls
```

Vamos executar o `/bin/ls` que é o `ls` em sua forma pura. O `ls --color=auto` , é um `aliased` . Observemos o que aparece ao executarmos o `/bin/ls` :

O `/bin/ls` lista o que temos no diretório atual de uma maneira tabular. Ele começa a lista na primeira coluna e coloca o máximo possível de nomes até ir para a próxima coluna, sendo esta do mesmo tamanho que a anterior. Quantas linhas ele

usa depende do número de arquivos e diretórios, perceba que ele tenta organizar um máximo de colunas possíveis. Nesse caso que pudemos visualizar ele coloca até três colunas. Ele divide o número de itens que temos nesse diretório por três. Na verdade o número de colunas não nos importa, na prática, estamos vendo a listagem dos nomes de arquivos, diretórios e o que quer que seja de uma maneira tabular. Esse é `/bin/ls` padrão, mas o que acontece é que vários desses itens que temos nesse `ls` são diretórios. Como sabemos qual é diretório, script, arquivo normal, programa que podemos executar, um atalho, um link para alguém que está de fora do diretório, ou alguma outra coisa diferente dessas já citadas? Como sabemos o que é o que dentre o que está listado?

É nesse caso que entraram as cores, quando os monitores com cores surgiram é que compreendeu-se que podemos usar as cores para indicar as diferenças, se é uma coisa ou se é outra. Lembra-se quando demo um `type ls` , ele nos disse que o `ls is aliased to 'ls --color=auto'` , ou seja, ele nos mostrou qual é a opção para que as coisas apareçam com cores. Podemos, então, digitar `/bin/ls -- color=auto` e pediremos que sejam mostradas as cores. O `auto` vêm de automático e então tudo é colorido automaticamente:

Agora sim, temos o resultado que aparecerá em outras distribuições, esse padrão de usar as cores é recorrente e a maior parte das distribuições adotará. Não significa que é um padrão obrigatório para todos, mas é algo que é bastante comum.

Se o arquivo é normal, teremos a cor branca, se é um diretório sua cor será azul, se é um executável será verde e o vermelho é um padrão que o `ls` do *Ubuntu* adota na hora de indicar um pacote que pode ser descompactado. Vamos dar um `ls` em outros diretórios para visualizarmos esses caras acontecendo?

Retomando: O azul indica um diretório, o vermelho um pacote que pode ser descompactado, o verde um arquivo executável e o branco um arquivo normal.

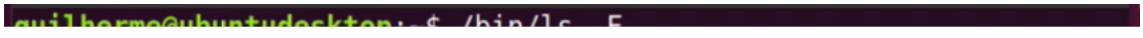
Vamos usar o `--color=auto` em outros diretórios também? Vamos usar isso no `/bin` :

```
> /bin/ls --color=auto /bin
```

É, justamente, no `/bin` que temos o `ls` , vejamos como fica:

Temos o verde que indica algo executável e temos uma nova cor, o azul claro que por padrão é um link para alguma coisa que está em outro canto do nosso sistema. Abordaremos o tema dos `links` , como criá-los e outros aspectos, mais adiante. Para nós, é importante percebermos pelo menos quatro cores, o azul escuro para diretório, o branco para arquivos normais, o verde para executáveis e o azul claro para links. Outras cores, como por exemplo, o vermelho que indica um pacote, não é um padrão interessante, pois, nem sempre o usaremos. É, principalmente, mais interessante nos atermos nas quatro cores, o branco que indica os arquivos normais, aqueles que estão soltos, os *plain file*, o verde que indica os *executable files*, o azul escuro que indica os *directory* e o azul claro que indica os *links*. Isso funciona muito bem, mas se estivermos em um monitor colorido.

E se você estiver em um monitor preto e branco? Você executará o `/bin --color=auto` e não vai entender o que estará acontecendo. Então, não só temos a opção de colocar cor como também podemos pedir ao `ls` para mostrar visualmente se o arquivo é um arquivo normal, executável, link ou diretório. Temos outras maneiras de ter essa diferenciação, também de maneira visual, mas não através de cor. Para isso, utilizaremos o `-F` maiúsculo. Digitaremos `/bin/ls -F` e repare no que ele nos trará:



Temos tudo normal, mas com um adendo, um asterisco foi agregado em alguns arquivos que são executáveis, aqueles que antes estavam em verde, temos também uma barra "/" que indica o que são diretórios e nos arquivos normais não existe

nenhuma marcação.

Vamos ver o que diferencia os atalhos digitando `/bin/ls -F /bin` . Vejamos:

Podemos observar que os executáveis são acompanhados de um asterisco, os atalhos, os links, estão com um arroba, o barra indica um diretório e o "nada" indica um arquivo normal. Esses são os quatro casos que são interessantes que lembremos.

O `-F` traz isso para nós. E se chamarmos o `-F` e ao mesmo tempo junto com o `--color=auto` ? Teremos o seguinte:

```
> /bin/ls -F --color=auto
```

Ele nos mostrará o seguinte:

Ele nos mostra o colorido e o carácter extra como sufixo, mostrando se é executável, se é diretório, se é um arquivo normal ou nada. Podemos, portanto, combinarmos essas duas opções, o `-F` com o `--color=auto` .

Da mesma maneira que vimos o `--color=auto` e o `-F` maiúsculo para que ele mostre que tipo de coisas são aquilo que vemos na listagem, temos, ainda, outra marcação que podemos utilizar quando queremos visualizar apenas os diretórios. Então, podemos digitar `/bin/ls` e se queremos apenas que os diretórios sejam marcados podemos digitar apenas um `-p` minúsculo e ele só marcara os diretórios:

Repare que ele não marcou nada nos executáveis e não vai marcar nada nos links. Só os diretórios terão uma barra, `/`.

Então, vimos o `/bin/ls` com o `-p` minúsculo, com o `-F` maiúsculo. O `-p` adiciona apenas a barra e o `-F` adiciona o asterisco, a barra, o arroba ou nada. O `--color=auto` deixa a lista colorida e bonitinha.

E o que acontece quando estamos executando o `ls` e ele está colorido e, na verdade, não queremos que ele tenha cor. Vamos no `man ls` e vamos buscar o `./color`:

Lembra-se que utilizamos o `auto`? Podemos usar também o `never`, o `auto` e o `always`. Vamos observar o `never`, digitando `ls --color=never` e com isso estamos dizendo que não queremos observar cor nenhuma.

Removendo e movendo arquivos com rm e mv

O que mais podemos fazer com o `ls`?

Utilizamos ele para listar arquivos e diretórios no diretório atual. E se quisermos visualizar os arquivos que são invisíveis? Comentamos que de acordo com o nome de um arquivo, se o primeiro carácter for um ponto, ele é considerado invisível. Isso quer dizer que, por padrão, o `ls` e outros programas não deveriam mostrá-lo. A não ser que, o usuário explicitamente fale que deseja ver coisas invisíveis.

Esse é o que significa do ponto (`.`) na frente de um nome de um arquivo. Como prefixo no nome de um arquiv, o ponto, dá a dica, tanto para o `ls` quanto para outros programas, que esse arquivo é considerado invisível e que não deve ser visualizado, por padrão. Repare que se estamos no diretório da `home` e se viermos no navegador de *files* do *Ubuntu*, não teremos nenhum arquivo que inicia com um `.`.

Mas, será que não temos mesmo?

Como fazemos para mostrar todas as coisas que temos do diretório da *home*? Inclusive arquivos, diretórios e links que iniciam com um `ponto`? Digitamos `ls -a` e ele mostra tudo:

Podemos observar o `g.conf`, o `gimp-2.8`, o `.cache` e assim por diante, então, temos todos esses arquivos, necessariamente, na `home` também. Quando damos um `ls` normal, não conseguimos visualizar todos eles. Por isso, quando abrimos o file tão pouco somos capazes de visualizar os aquivos iniciados com `.`:

Para enxergarmos os arquivos invisíveis, teríamos que ir no "Menu" e pedir *Show Hidden Files*, isto é, mostrar os arquivos escondidos, que são os arquivos invisíveis. Agora, teremos todos os arquivos:

Lembrando, o `.` é uma dica e o `ls` segue essa dica de não mostrar o que é invisível. O nosso *explorer*, a `home` do *Ubuntu* também segue essa dica, assim como os nossos programas também podem seguir essa dica. Se o arquivo começa com um `.` ele não é para ser considerado para visualização do usuário final, pois ele é um arquivo interno dos programas, um diretório interno de configuração e outros tipos de coisa que não são interessantes de serem compartilhados com os usuários. Para isso é que serve o `.` no começo dos nomes. Quando queremos visualizar tudo, `-a` serve para nos ajudar a enxergar tudo. Uma outra maneira de poder enxergar os arquivos invisíveis é digitando `ls --all`, o *all* significa tudo em inglês e é a maneira completa de dizer `ls -a`. O `ls --all` mostra tudo o que temos, arquivos, links e diretórios, inclusive os invisíveis.

Então, podemos usar o `-a` ou o `--all` que veremos tudo.

Existem outras opções que podemos utilizar para trabalhar com o `ls`? Vamos dar um `ls` e verificar o que aparece:

Vemos que existe uma ordem das coisas, elas estão distribuídas em ordem alfabética até a letra "z", onde, os número aparecem antes das letras e as letras em minúsculo e maiúsculo aparecem juntas. As vezes, podemos querer inverter essa ordem lexográfica. Para modificar a ordenação podemos digitar `ls --reverse` que estaremos pedindo: "Porfavor, reverta a ordem" e ele trará do "z" até o "a":

As vezes queremos procurar algo em um diretório que possui muitas informações e o que queremos está muito no final ou muito no começo, para facilitar essa busca basta usarmos o `--reverse`. O atalho do `--reverse` é o `-r` minúsculo. Temos que tomar bastante cuidado com o `-r` minúsculo, pois em muitos programas isso quer dizer recursivo, mas, no `ls` o significado não é esse, quer dizer: "reverta a ordem de ordenação que estamos utilizando para listar coisas". Vimos o `-a`, o `--color=auto`, o `--color=never`, o `-p`, o `-F` e, por fim, o `-r`, de reverse. Vamos ver, ainda, outras coisas!

Falamos que por padrãos as coisas estão sendo ordenadas a partir de uma ordem *lexográfica*, a partir do nome dos arquivos. E se quisermos ordenar de outra maneira?

Por exemplo, temos diversos arquivos e queremos apagar apenas os arquivos de maior tamanho nesse diretório atual. Então, queremos que a ordem seja pelo tamanho dos arquivos, para tanto, dizemos para o `ls` ordenar `sort` pelo `size`, isto é, pelo tamanho. Digitaremos o seguinte: `ls --sort=size` e teremos o seguinte:

Veremos que a ordem foi rearranjada e agora ela é feita a partir do tamanho do que temos. Primeiro temos o `examples.desktop` até chegar ao `sucesso~`. Essa é a ordem por tamanho dos arquivos. Mas, você pode estar se perguntando, se o `--sort=size` ordenou por tamanho, como podemos verificar se isso é de fato uma verdade? Onde está o tamanho dos arquivos? Então, imagine que ao em vez de fazer uma simples ordenação tabela, gostariamos que fosse possível ver em uma lista onde podemos confirmar o tamanho dos arquivos. Para verificar mais informações em uma lista longa podemos dizer ao `ls` para formatar a sua saída em um formato longo. Assim, escreveremos `ls --format=long` e teremos o seguinte:

Ele nos mostra todos com o nome e o tamanho. Por que ele mostrou em uma lista longa já colorida? Pois, estamos utilizando o `ls` padrão que possui o atalho de cor e se quisermos forçar, usando para tanto o `/bin/ls --format=long` teremos todas as

informações em preto e branco:

Repare que ele traz uma listagem longa que traz o tamanho dos arquivos e diretórios, isto é, o quanto um item em específico está ocupando em termos de espaço. Repare que no caso de um diretório não aparece ele e mais o seu conteúdo, temos apenas a representação do diretório. Não é o diretório com tudo o que está dentro dele, não é o acumulado do diretório, é apenas a representação do diretório. É só informado que aquele diretório existe e não o peso do seu conteúdo, isso já é outra história. Então, cuidado! Além disso ele fala outras informações, sobre o acesso aos arquivos e sobre execução. Uma informação muito importante que aparaece é se ele é um diretório, nesse caso, o primeiro caracter de todos é usado para indicar, com a letra "d", se é um diretório. O primeiro caracter é um indicador especial. Se é um diretório possui a letra `d`, se é um aqruivo normal possui o `-` e se ele é um link, possui um `l`

Então, `l` é link, hífen é arquivo normal e `d` é diretório. E não só isso, ele não nos mostrou apenas esse caracter, ele mostrou também as permissões, tópico que abordaremos mais adiante. Ele mostra também detalhes de quem é o grupo dono e o usuário dono, outro tema que falaremos no tópico de permissões. Mas, no que estamos interessados, nesse instante, é na parte que menciona o tamanho do arquivo.

Vamos, mais uma vez, pedir para ele ordenar o `/bin/ls` conforme o tamanho, `/bin/ls --format=long reverse --sort=size /bin`, veremos que primeiro estão os maiores e depois os menores tamnhos. Mas, você pode estar querendo saber apenas os sobre os maiores, por isso, já passamos pelo `reverse`. Vamos digitar `/bin/ls --format=long --reverse --sort=size`, onde o `--format=long` significa formatação longa, o `reserve` muda a ordem de ordenação e `--sort=size` dita que a ordem é por tamanho. Assim, teremos uma ordenação por tamanho, mas que começará do menor para o maior. Teremos o seguinte:

Quando usamos o `size` ele faz do maior para o menor e como pedimos para que a ordem fosse feita ao contrário, ele fez do menor para o maior. Conforme podemos verificar, onde o último item da lista possui mais ou menos 2 megas.

O `--sort` e utilizado para ordenar as coisas.

Podemos ordenar por outros campos? Sim! Podemos usar diversos outros campos, por exemplo, podemos ordenar pelo tempo, isto é, quando foi a última modificação do arquivo, para isso usamos o `time`:

```
> /bin/ls --format=long reverse --sort=time /bin
```

E teremos o seguinte:

Repare que ele está em ordem reversa, então, temos as modificações mais antigas até as mais recentes. Por padrão, não damos o `reverse` usamos apenas o `sort` pelo `time`, sendo:

```
bin/ls --format=long --sort=time /bin
```

A ordenação, por padrão, no `time` é do mais recente ao mais antigo, no `size` do tamanho maior para o menor, e normalmente do nome da letra mais simples, do "a" até o "z". E temos um último padrão que é a extensão, para isso usaremos `--sort=extension`. Lembra-se que já falamos sobre extensão? No caso, `extension` é um artefato fingido, não é de verdade, é simplesmente um ponto, `.` seguido de alguma coisa, faz parte do nome de um arquivo. Vamos pedir, então, que as coisas

sejam ordenadas por extensão. Para isso digitaremos `/bin/ls --=long --sort=extension` e ele ordenará, no diretório atual, por ordem de extensão. Teremos o seguinte:

Ele nos mostra o diretório atual ordenado segundo a extensão, podemos perceber que os `txt` estão organizados todos juntos. Se olharmos, mais em cima, teremos os arquivos sem extensão. Então a ordem foi realizada pela `extensão` . Ele agrupou aqueles que possuem uma mesma extensão e ordenou por isso.

Temos a mesma coisa nesse caso, temos o `-r` , o `reverse` que serve para ordenar ao contrário as extensões. Vamos testar isso digitando `/bin/ls --=long --sort=extension -r` e teremos o seguinte:

Agora, as extensões estão ordenadas ao contrário!

Com isso, falamos de ordenação `--sort` podemos usar como sendo igual a `size` , tamanho, a `time` , que refere-se ahora da última modificação do arquivo e `extension` que é a extensão. Por padrão o `size` é do maior para o menor, o `time` é do mais recente para o mais antigo, o `extension` é de sem extensão para com extensão em ordem alfabética e se usarmos o `-r` , o `reverse` , estaremos revertendo essa ordem.

O `ls -l` é o que traz uma informação longa para nós:

Aqui, o que é interessante de presentamos atenção é o carácter que inicia, que nos indica se é um diretório, um link ou algo normal, também é importante repararmos no tamanho, quantos bytes são ocupados por cada um dos itens, o que não é recursivo no caso do diretório. E a data da última modificação, cuidado, essa data da última modificação pode seguir o padrão do seu sistema operacional, isto é, de estiver em inglês, português, francês e etc Lembra-se quando falamos da instalação?

Usaremos um padrão, em inglês, pois, a prova será em `inglês` . Então, use em inglês, pois a prova será nesse idioma e o padrão que veremos será o americano que é o mês, onde as três primeiras letras estarão escritas em inglês, a data e a hora. Em nosso exemplo a hora está em formato de 24 horas, não é am e pm. Esse é o padrão que ele nos mostra no `ls -l` .

Tem mais um detalhe que gostaríamos de explorar no `ls` , quando estamos vendo os arquivos com o `ls` estamos vendo tudo o que temos no diretório atual. E se quisermos ver tudo o que temos no diretório atual, só que queremos entrar nos diretórios filhos, queremos entrar no diretório `desktop` e ver o que tem lá dentro, queremos entrar no diretório `documents` e ver o que temos lá dentro. Queremos entrar nos diretórios filhos recursivamente e ver que temos dentro. Então, digitamos `ls --recursive` e ele nos mostra o que temos:

Ele nos mostra o diretório atual e os diretórios filhos. Ele entra no diretório `Desktop` , no `Documents` , no `Downloads` , no `loja` , no `Music` e assim por diante. Ele entra nos diretórios e executa, para nós, o `ls` .

Então para usar o `recursive` abreviado não usamos o `-r` , usamos o `-R` maiúsculo para ver o recursivo. Então, qual a diferença entre o `ls*` e o `ls -R` ?

Vamos ter cuidado nesse instante. O `ls -R` mostra que no diretório atual temos um diretório chamado `Music` e nele não temos nada e no diretório `Documents/curriculum` temos o `curriculum-guilherme-silveira.odt` .

E o `ls*` ? Lembra-se do que faz o asterisco? O asterisco é interpretado pelo *shell* com o nome de tudo o que temos, então, ao digitar `ls*` estamos dizendo tudo o que tem no `ls` . É como se estivéssemos digitando `ls 0 0~a.out arquivos.zip e etc e` executando tudo isso.

Qual é a diferença entre esses dois? O `ls -R` entra no diretório como, por exemplo, o `Documents` , mostra o diretório, entra no subdiretório `/curriculum` e nos mostra esse diretório:

E se fizermos um `ls*` ? Ele só entra em um nível de diretório, ele não entra, por exemplo no `curriculum` :

É como se tivessemos feito um `ls Documents` e apenas com isso ele não entrará no `curriculum` para nós.

Percebeu a diferença entre o `ls*` e o `ls -R` ?

O `-R` significa ser recursivo, justamente, um dos pontos que a prova cobra, o *recursive listings*. O que ele faz é entrar em uma listagem recursiva, que busca o `ls` de uma maneira recursiva, entrando nos diretórios filhos. O `ls *` não faz isso, pois, quem enterpretou o carácter asterisco foi o *shell*. Interpretou e trocou por todos os nomes de diretórios e coisas que existem no diretório atual. Foi isso que ele fez nesse caso.

Vamos observar dois últimos detalhes que queremos que sejam vistos. O primeiro é, se temos um diretório onde queremos dar `ls` , por exemplo, o diretório `Documents` e queremos ver mais detalhes desse diretório `documents` o que fazemos? Podemos digitar `ls -l Documents` e teremos o conteúdo de dentro do diretório `Documents` .

Então, se quisermos fazer um `ls` para ver as informações do diretório `Documents` temos que fazer um `ls -l` no diretório atual, só que ele traz diversas coisas e teremos que buscar o `Documents` em diversas coisas e isso acaba sendo muito chato.

O `ls` tem uma opção que se chama `-d` minúsculo e quando usamos essa opção o que estamos dizendo é que queremos analisar um diretório específico. Podemos, então, digitar `ls -d -l Documents` . O `-d` serve para dizer que queremos analisar um determinado diretório e não que queremos entrar em um diretório e mostrar o que tem dentro dele. Vejamos o que aparece quando digitamos `ls -d -l Documents` :

```
> ls -d -l Documents
drwxr-xr-x 3 guilherme guilherme 4096 Mar 14 10:02
```

Usando o `-d` falamos que queremos analisar um determinado diretório.

E se quisermos ver uma permissão de uso do `ls` ? Podemos digitar `ls /etc` e teremos o seguinte:

Não é bem o que queríamos. Podemos fazer `ls -l /etc` e teremos o seguinte:

E não é, tão pouco, o que queríamos. Vamos tentar uma última coisa, digitar `ls -ld /etc` e teremos:

```
> ls -ld /etc
drwxr-xr-x 135 root root 12288 Mar 10 09:20 /etc
```

Agora, ele traz para nós as informações sobre o `/etc`. Então, acabamos usando o `ld` minúsculo abastante quando queremos saber informações sobre um diretório específico.

Por fim, um sistema de arquivos do *Linux* utiliza por padrão uma estrutura chamada `inode`. Cada arquivo é representado por um ou mais nós que trazem a informação base dos arquivos que estão ligados que por sua vez estão ligados a outros nós que estão ligados a outros nós que trazem dados de arquivos, diretórios e etc e assim por diante. Então, é um pouco uma árvore que vai se interligando com todos esses nós.

Temos, ainda, uma opção do `ls` que é o `ls -i` que mostra a informação desses nós. Vejamos o que ele nos mostra quando digitamos isso:

Repare que o `0`, o primeiro item da lista, aponta para o nó `474722`. O `0~` está apontando para o nó `474620`. Cada arquivo, cada diretório, aponta para um nó diferente do sistema de arquivos e esse nós estão indicados por esses números. Acabamos usando esses nós quando estamos lidando com tarefas mais avançadas de recuperação de dados, quando temos algum problema e queremos entender o que está acontecendo com a estrutura de arquivos do nosso disco. Aí utilizamos a estrutura de `ls -i` para ver quais são os nós, isto é, o *inode* desses itens que estamos vendo. Temos também o `ls --inode`, podemos falar a mesma coisa de uma maneira mais explícita, usando o `--inode` que é a mesma coisa que `-i`.

Com isso, vimos diversas opções do `ls`. O `ls` possui ainda, mais um monte de opções, para verificarmos isso basta digitarmos `man ls`. Não se preocupe em decorar todas, lembre-se da dica passada! A medida que você for utilizando o `ls` observe e leia o seu manual, vai aprendendo uma ou outra.

Mas, existem algumas opções que são muito importantes de serem aprendidas, é interessante que saibamos sobre elas para que estejamos aptos para responder questões na hora da prova.

Por exemplo, o `ls -F` indica, visualmente, através de caracteres o que cada coisa que está na lista é. Um arquivo normal não possui nenhum carácter, um arquivo executável possui um asterisco, um diretório é marcado pela barra e um link possui um arroba.

O `ls --color=auto` coloca uma cor para diferenciar os itens listados, branco, se for normal, verde se for executável, azul escuro se for diretório e azul claro se for um link.

Se digitarmos `ls -p` só estaremos demarcando o diretório através do uso de uma barra, `/`, aliás, no `ls -F` acontece a mesma coisa segue uma `/` se é um diretório.

E o `ls -l` mostra uma listagem longa, é o `--format=long`. Lembra-se da listagem longa? O primeiro carácter mostra para nós se é um carácter normal ou executável, se um diretório inicia com a letra "d" e se é um link inicia com "l".

Vimos também o `ls -a` ou `ls --all` que mostra também os arquivos que são considerados invisiveis, aqueles que iniciam com um ponto, `. .`

O `ls --inode` ou `ls -i` que mostra o nó que estamos referenciando através de um nome de arquivo, diretório ou link.

Vimos o `ls --color=never` para sumir com as cores.

Vimoso `ls --reverse` para ordenarmos ao contrário, então, no caso de termos uma ordem alfabética que nos mostra da letra "a" ao "z" se usarmos o `reverse` ele passará a nos mostrar do "z" ao "a". Poderíamos mudar também utilizando o `-r` minúsculo. Atenção o `-r` minúsculo refere-se a ordenação de maior para menor e vice e versa.

Podemos usar, também, o `ls --sort=size` que ordena os itens por tamanho, do maior para o menor, o `--sort=time` que ordena do mais recente ao mais antigo e `--sort=extension` que mostra primeiro os sem extensões, depois do "a" até o "z". Podemos usar todos eles junto com o `-r` que mostra os itens ordenados ao contrário.

O `ls --sort` também possui algumas abreviações, então, se queremos ordenar as coisas por tamanho podemos digitar `ls --sort=size` e `ls -S` que elas serão ordenadas por tamanho. No caso de `--sort=time` podemos escrever `ls -t` , minúsculo e,se queremos usar o `--sort=extension` podemos usar `ls -X` maiúsculo.

Por fim, falamos um caso muito importante o `ls --recursive` que entra recursivamente nos diretórios e nos filhos dos diretórios e nos filhos destes e que abreviado é `ls -R` maiúsculo.

Lembre-se, as opções podem ser combinadas, por exemplo, ao em vez de escrever `ls -l -a` , podemos digitar `ls -la` , mas isso apenas quando for uma opção dupla, caso contrário, ela deve vir separada, por exemplo `ls -la --sort=time` . Podemos trabalhar dessa maneira.

O diretório que queremos listar ou as coisas que queremos listar, vêm depois das opções, por exemplo: `ls -la --sort=time /bin` . Podemos colocar o `/bin` antes de tudo? Lembre-se os comandos do *Linux* seguem um padrão "nome do comando, opções e parâmetros". Porém, o `ls` aceita essa execução, `ls /bin -la --sort=time` . O `ls` aceita as opções, depois dos parâmetros. Vai depender de cada comando e programa.

A dica é, use o `ls` bastante, no seu dia a dia e use também o seu `man` para aprender, na medida do possível, o máximo de coisas sobre o `ls` . E lembre-se, s casos que vimos aqui são importantíssimos.

E com isso, passamos por diretórios e arquivos, arquivos e diretórios invisíveis, o diretório home e a variável de ambiente `$HOME` , caminhos absolutos e relativos e os atalhos, `.` , `..` , `~` , que é nosso diretório `home` , `cd` , o diretório raiz (`/`) , `pwd` que é o diretório atual, listagem recursiva através do `ls -R` ou através do `ls --recursive` e diversas opções comuns do `ls` .