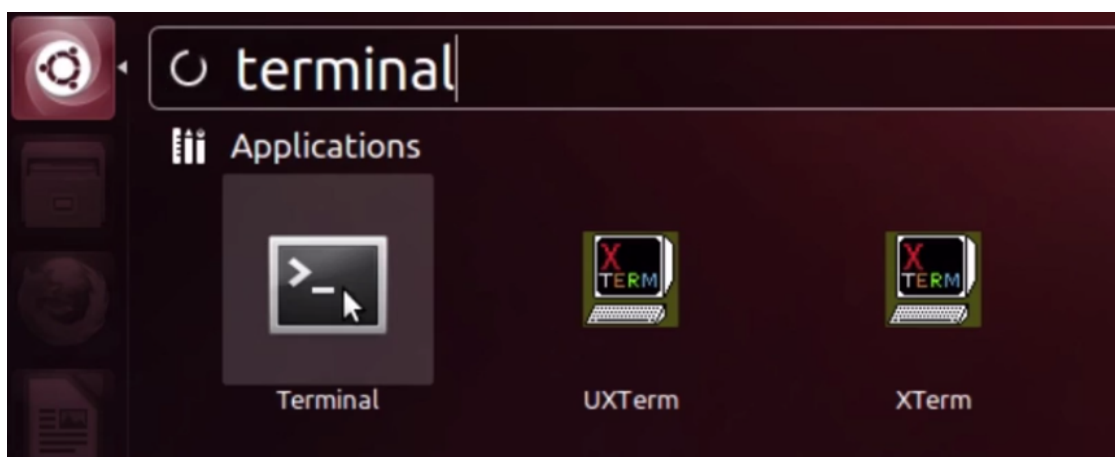




### Transcrição

Como já comentamos, ao longo curso veremos como utilizar o Terminal do Ubuntu para aprender os comandos e programas para tirar o melhor proveito desse Sistema Operacional no nosso dia-a-dia. O primeiro passo é abrirmos o Terminal pelo menu de início do Ubuntu:



Quando abrimos o Terminal, nós estamos em algum diretório do sistema operacional e para descobrirmos em qual estamos, digitamos o comando `pwd`. A saída do comando será algo como:

```
/home/guilherme
```

[COPIAR CÓDIGO](#)

Lembrando que o nome depois de `/home` mudará de acordo com o usuário atual logado no sistema operacional. O `/guilherme` é o diretório base para o usuário Guilherme. Agora se quisermos saber os arquivos e diretórios dentro deste diretório base, ou seja, a lista deles,

basta utilizarmos o comando `ls` . O resultado será algo como:

```
guilherme@ubuntu:~$ ls
Desktop  Downloads  Music      Public  Videos
Documents  examples.desktop  Pictures  Templates
guilherme@ubuntu:~$
```

Note que o Terminal do Ubuntu diferencia por meio de cores aquilo que é diretório e aquilo que é arquivo, outros terminais podem não fazer essa diferenciação.

Agora vamos criar um arquivo. Primeiramente vamos escolher um texto de exemplo que irá dentro desse arquivo. Para que o terminal imprima a mensagem "Bem vindo" podemos utilizar o comando `echo` , que irá imprimir esses dois argumentos ("Bem" e "vindo"):

```
echo Bem vindo
```

[COPIAR CÓDIGO](#)

O resultado será a mensagem impressa no terminal.

Enquanto digitamos comandos no terminal, uma espécie de histórico está sendo criada, se clicarmos no botão de seta para cima, voltamos ao comando anterior que foi executado. Usamos esse atalho para navegarmos pelos comandos, clicando mais vezes a seta pra cima, chegaremos a comandos digitados a mais tempo, a seta para baixo também funciona para voltar para os comandos mais atuais no histórico.

Vamos usá-lo para voltar ao `echo` e passarmos apenas um argumento, colocando aspas duplas na mensagem que queremos imprimir:

```
echo "Bem vindo"
```

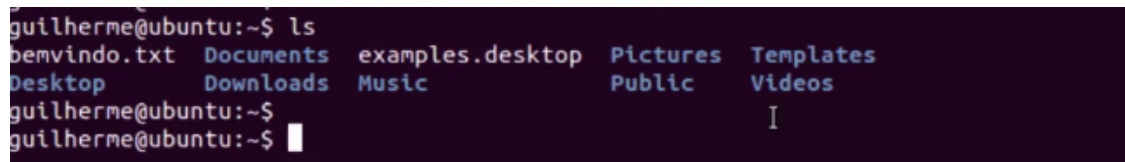
[COPIAR CÓDIGO](#)

Mas o que queremos é executar esse comando redirecionando sua saída para um arquivo, para isso, utilizamos o caractere `>` depois da mensagem seguido pelo nome do arquivo que queremos salvar a mensagem:

```
echo "Bem vindo" > bemvindo.txt
```

[COPIAR CÓDIGO](#)

O terminal já não imprime mais a mensagem, ela foi redirecionada para o arquivo, veja que se buscarmos novamente a lista de arquivos e diretórios usando o `ls`, teremos nosso arquivo `bemvindo.txt` listado:



```
guilherme@ubuntu:~$ ls
bemvindo.txt  Documents  examples.desktop  Pictures  Templates
Desktop       Downloads  Music             Public    Videos
guilherme@ubuntu:~$
guilherme@ubuntu:~$
```

Experimente abrir o explorador de arquivos e verificar os arquivos que criamos e listamos. Experimente também abrir os arquivos de texto criados em um editor de texto visual para garantir que as mensagens realmente estão nos arquivos.

Também podemos ler o conteúdo de arquivos no Terminal usando o comando `cat` :

```
cat bemvindo.txt
```

[COPIAR CÓDIGO](#)

A saída do comando `cat` será o texto presente dentro do arquivo `bemvindo.txt`.

Para todos os comandos envolvendo arquivos ou diretórios existe a possibilidade de escrevermos apenas uma parte do nome deles e buscar um específico utilizando a tecla **TAB**. Se houver apenas um arquivo com o início do nome digitado, o terminal o preenche automaticamente ao apertarmos a tecla.

Para limparmos a tela do terminal usamos o comando `clear`.

Já sabemos que com o comando `ls` conseguimos visualizar os arquivos e diretórios, porém o terminal não mostra muita informação sobre eles. Para isso podemos utilizar o comando `ls -l`, que listará seus tamanhos, datas de modificação e os tipos. Se for diretório, as informações começarão com a letra "d".

```
guilherme@ubuntu:~$ ls -l
total 48
-rw-rw-r-- 1 guilherme guilherme 10 Jun 11 15:24 bemvindo.txt
drwxr-xr-x 2 guilherme guilherme 4096 Jun 10 23:46 Desktop
drwxr-xr-x 2 guilherme guilherme 4096 Jun 10 18:18 Documents
drwxr-xr-x 2 guilherme guilherme 4096 Jun 10 23:46 Downloads
-rw-r--r-- 1 guilherme guilherme 8942 Jun 10 18:04 examples.desktop
drwxr-xr-x 2 guilherme guilherme 4096 Jun 10 18:18 Music
drwxr-xr-x 2 guilherme guilherme 4096 Jun 10 18:18 Pictures
drwxr-xr-x 2 guilherme guilherme 4096 Jun 10 18:18 Public
drwxr-xr-x 2 guilherme guilherme 4096 Jun 10 18:18 Templates
drwxr-xr-x 2 guilherme guilherme 4096 Jun 10 18:18 Videos
```

Note que além das informações que já comentamos antes sobre os detalhes dos arquivos e diretórios, são exibidas também os nomes do grupos e usuários a quem o arquivo ou diretório pertence. Veremos mais sobre essas informações mais adiante.

Contudo, esses não são todos os arquivos e diretórios que temos. No Linux existem alguns arquivos e diretórios invisíveis. Para listá-los usamos o comando `ls -la`. Note que arquivos e diretórios invisíveis no Linux são precedidos pelo caractere ponto(.) e há vários desses

diretórios, dentre eles os de cache e de configurações.

```
-rw-rw-r-- 1 guilherme guilherme 10 Jun 11 15:24 bemvindo.txt
drwx----- 14 guilherme guilherme 4096 Jun 10 22:02 .cache
drwx----- 14 guilherme guilherme 4096 Jun 11 08:56 .config
drwx----- 3 guilherme guilherme 4096 Jun 10 18:18 .dbus
drwxr-xr-x 2 guilherme guilherme 4096 Jun 10 23:46 Desktop
-rw-r--r-- 1 guilherme guilherme 25 Jun 10 23:45 .dmrc
drwxr-xr-x 2 guilherme guilherme 4096 Jun 10 18:18 Documents
drwxr-xr-x 2 guilherme guilherme 4096 Jun 10 23:46 Downloads
-rw-r--r-- 1 guilherme guilherme 8942 Jun 10 18:04 examples.desktop
drwx----- 3 guilherme guilherme 4096 Jun 10 23:45 .gconf
-rw----- 1 guilherme guilherme 636 Jun 10 23:45 .ICEauthority
drwxr-xr-x 3 guilherme guilherme 4096 Jun 10 18:18 .local
```

Para aprendermos sobre um determinado comando podemos utilizar o comando `man` + [comando sobre o qual queremos aprender]. Ele retornará toda a documentação do comando, seu manual. Podemos utilizar as setas para cima e para baixo para navegarmos nessa documentação e a tecla **q** para sairmos dela. Exemplo de uso do `man` :

```
man ls
```

COPIAR CÓDIGO

Vimos nesta primeira aula diversos comandos simples para começarmos a navegar mais profundamente com o terminal, dentre eles aprendemos:

- `pwd` : para descobrir o diretório atual.
- `ls` : para listar arquivos e diretórios, vimos as opções `-l` e `-la` , que listam além dos arquivos e diretórios ocultos, informações extras sobre cada um deles.
- `echo` para imprimir mensagens e o operador `>` para modificar o destino da mensagem.
- `clear` para limparmos o terminal.
- `man` para consultar o manual sobre determinado comando.
- setas para cima e para baixo para navegador no histórico de comandos do terminal.





### Transcrição

Como vimos na aula passada, o comando `ls` nos lista os arquivos e diretórios no diretório atual.

```
guilherme@ubuntu:~$ ls
bemvindo.txt  Documents  examples.desktop  Pictures  Templates
Desktop       Downloads  Music             Public    Videos
guilherme@ubuntu:~$
guilherme@ubuntu:~$
```

Vimos também que o comando `pwd` nos informa o diretório atual, algo como `/home/guilherme`, mas e se quiséssemos mudar de diretório? Para mudarmos de diretório, podemos utilizar o comando `cd` - *Change Directory*:

```
cd Desktop
```

[COPIAR CÓDIGO](#)

O comando `pwd` agora nos trará um resultado diferente. Algo como `/home/guilherme/Desktop`. Vamos exercitar a criação de arquivos novamente com o comando `echo`, lembra dele? Vamos criar um arquivo que lista algumas músicas. Por exemplo:

No Ubuntu, Fedora e talvez em outras distros Linux que suportam o nosso português (pt\_BR), a tradicional pasta "Desktop" foi renomeada para "Área de Trabalho" e, como os iniciados sabem, ela precisa ser acessada pelo terminal com o uso de barras invertidas ("") para designar os espaços, ou seja:

```
cd Área\ de\ Trabalho/
```

Os iniciantes muitas vezes esquecem ou não sabem disso, sem falar que eles podem vir a executar scripts que ainda usam a designação tradicional para essa pasta, ou seja, "Desktop".

```
echo "Faithless" > musicas-favoritas.txt
```

[COPIAR CÓDIGO](#)

Lembre-se de verificar a criação de arquivos e diretórios pelo explorador de arquivos. Só por questões de confirmação.

Criamos o arquivo `musicas-favoritas.txt` com o texto `Faithless` dentro do diretório `Desktop`. Vamos tentar colocar mais uma música dentro desse arquivo utilizando o comando `echo` da mesma maneira:

```
echo "REM" > musicas-favoritas.txt
```

[COPIAR CÓDIGO](#)

Se utilizarmos o comando `cat` para ver o conteúdo do arquivo, teremos uma surpresa: Apenas a música `REM` estará listada no arquivo. Isso por que o comando `echo` escreve no arquivo, criando o arquivo caso não exista. Caso exista, o conteúdo do arquivo será **sobrescrito**. Não queremos sobrescrever, queremos adicionar um texto, concatená-lo com o que já está inserido no arquivo. Para tal usamos `>>` em vez de apenas um `>`:



```
echo "Faithless" >> musicas-favoritas.txt
```

[COPIAR CÓDIGO](#)

Para confirmar a adição da segunda música, podemos ler o arquivo com o comando `cat` .

```
cat musicas-favoritas.txt
```

[COPIAR CÓDIGO](#)

E teremos como resultado a saída:

```
REM  
Faithless
```

[COPIAR CÓDIGO](#)

Até este ponto, criamos arquivos, listamos, entramos em um diretório, mas como voltamos para o diretório anterior? Para voltarmos para o diretório anterior usamos o comando `cd . . .`. Existem também o diretório `.` , que referência o diretório atual.

Podemos sempre confirmar o diretório atual usando o comando `pwd` . Utilize-o para confirmar a troca de diretórios.

Já sabemos como navegar entre diretórios. Vamos aprender agora a criar um diretório, para isso utilizamos o comando `mkdir` seguido do nome do diretório que queremos criar:

```
mkdir workspace
```

[COPIAR CÓDIGO](#)

Criamos o diretório `workspace` dentro do diretório

`/home/guilherme` . Perceba que os mesmos comandos utilizados anteriormente para criação e leitura de arquivos e diretórios também funcionam em subdiretórios. Experimente criar outros arquivos e subdiretórios dentro do `workspace` , como por exemplo: `projetos-java` e `projetos-php` .

```
echo "meu primeiro teste" > arquivo1.txt
echo "meu primeiro teste" > arquivo2.txt
echo "meu primeiro teste" > arquivo3.txt
```

```
mkdir projetos-java
mkdir projetos-php
```

COPIAR CÓDIGO

Navegue por esses subdiretórios usando o comand `cd` . Use o atalho do **TAB** para não precisar digitar o nome completo dos diretórios na navegação, mas note que o TAB não irá completar o nome do diretório totalmente, por que depois do traço, os nomes são diferentes, então você precisa digitar manualmente pelo menos as primeiras letras depois de `projetos-` para poder usar o TAB novamente.

Depois de entrar no diretório `projetos-java` , o comando `pwd` nos trará o seguinte retorno:

```
/home/guilherme/workspaces/projetos-java
```

COPIAR CÓDIGO

Agora, se estivéssemos dentro de um subdiretório que, por sua vez, estivesse dentro de outro (*como fizemos agora*), e quiséssemos ir para

o diretório base, como faríamos? O comando `cd`, sozinho, sem pontos, fará isso. Não importa onde estejamos, com o comando `cd` sempre voltamos para o diretório base. Que neste caso é:

```
/home/guilherme/
```

[COPIAR CÓDIGO](#)

O diretório base não é a raiz de tudo. Para visualizarmos a raiz do *HD* digitamos `ls /`. O `/` possui diversos outros diretórios úteis para o funcionamento do Linux. O diretório `home` por exemplo, é onde estão os diretórios de cada um dos usuários que utilizam a máquina.

Nesta aula aprendemos a criar diretórios, concatenar textos dentro de um arquivo e navegar entre diretórios.



### Transcrição

Aprendemos a criar um diretório com o comando `mkdir`, mas como podemos apagar um diretório? E um arquivo? Removemos um diretório usando o comando `rmdir` e para remover um arquivo utilizamos o comando `rm`, dentro de `workspace` por exemplo podemos fazer:

```
rmdir projetos-java  
rm arquivo3.txt
```

[COPIAR CÓDIGO](#)

Removemos/Apagamos o diretório `projetos-java` e o arquivo `arquivo3.txt`. Uma observação interessante é que o comando para remover diretórios só irá funcionar para aqueles diretórios que estiverem vazios. Ele não apaga os arquivos dentro do diretório automaticamente.

Dentro do diretório `workspace` atualmente há três arquivos e um diretório (*use o comando `ls` para listar os arquivos*):

```
arquivo10.txt arquivo1.txt  arquivo2.txt  projetos
```

[COPIAR CÓDIGO](#)

No `arquivo10.txt` temos a frase "Bem vindo" e nos outros dois a frase "meu primeiro teste". Para que o terminal imprima os textos de

todos os arquivos com um determinado nome, por exemplo, "arquivo...txt" podemos fazer:

```
cat arquivo?.txt
```

[COPIAR CÓDIGO](#)

O resultado será:

```
meu primeiro teste  
meu primeiro teste
```

[COPIAR CÓDIGO](#)

Também podemos usar o caractere `*` :

```
cat arquivo*.txt
```

[COPIAR CÓDIGO](#)

Mas o resultado será um pouco diferente do primeiro caso:

```
Bem vindo  
meu primeiro teste  
meu primeiro teste
```

[COPIAR CÓDIGO](#)

Esses caracteres de `?` e `*` são utilizados para que sejam buscados mais de um arquivo que tenham o mesmo nome base, porém existe uma pequena diferença: O `?` só encontra os arquivos que tenham apenas UM caractere diferente do nome base, enquanto o `*` busca quaisquer números de caracteres.

Por isso que o arquivo `arquivo10.txt` é encontrado no segundo

exemplo, mas não no primeiro. O `*` consegue lidar com o `10` depois do trecho `arquivo` no nome do arquivo, enquanto o `?` não consegue, pois são dois caracteres em `10`.

**Atenção:** se buscarmos o arquivo como `"*.txt"`, ou seja, com aspas, o terminal interpreta o asterisco como se o mesmo fosse parte do nome do arquivo:

```
cat "*.txt"
```

[COPIAR CÓDIGO](#)

O comando acima nos trará uma mensagem de erro como a seguinte:

```
cat: *.txt: No such file or directory
```

[COPIAR CÓDIGO](#)

Comentamos anteriormente que não podíamos apagar um diretório que possuía outros subdiretórios ou arquivos. Na verdade podemos, com o comando `rm -r` isto é possível. Ele remove **Recursivamente**, ou seja, apaga o diretório e tudo que está dentro dele. Podemos fazer por exemplo:

```
rm -r workspace/
```

[COPIAR CÓDIGO](#)

Dessa forma apagamos todo o diretório `workspace` e tudo que havia dentro dele. Muito cuidado na hora de utilizar este comando para não apagar diretórios importantes.





### Transcrição

Vamos criar novamente o diretório *workspace* - para não trabalharmos na raiz do *HD*:

```
mkdir workspace  
cd workspace/
```

[COPIAR CÓDIGO](#)

Vamos colocar uma mensagem em um arquivo qualquer dentro do diretório criado:

```
echo "bem vindo" > mensagem.txt
```

[COPIAR CÓDIGO](#)

Vamos copiar o texto do arquivo que acabamos de criar para um outro de nome "bemvindo.txt" com o comando `cp` :

```
cp mensagem.txt bemvindo.txt
```

[COPIAR CÓDIGO](#)

O texto agora está nos dois arquivos:

```
cat bemvindo.txt  
bem vindo  
cat mensagem.txt
```



bem vindo

COPIAR CÓDIGO

Podemos também mover o arquivo "mensagem.txt" para outro com o comando `mv` :

```
mv mensagem.txt bemvindo2.txt
```

```
ls  
bemvindo2.txt    bemvindo.txt
```

COPIAR CÓDIGO

Perceba que o arquivo mudou de nome.

Criemos agora mais dois diretórios:

```
mkdir projetos-java  
mkdir projetos-php
```

COPIAR CÓDIGO

Queremos mover o arquivo "bemvindo.txt" para dentro do diretório "projetos-java":

```
mv bemvindo.txt projetos-java/
```

COPIAR CÓDIGO

Se quiséssemos além de movê-lo, mudar seu nome, faríamos, por exemplo:

```
mv bemvindo.txt projetos-java/bemvindo-novo-nome.t
```

COPIAR CÓDIGO

Vamos verificar se ele foi realmente movido para o diretório que queríamos:

```
ls workspace/  
bemvindo2.txt    projetos-java    projetos-php
```

```
ls projetos-java/  
bemvindo.txt
```

[COPIAR CÓDIGO](#)

De fato, o arquivo "bemvindo.txt" saiu do *workspace* e foi para o *projetos-java*. Vamos copiar "bemvindo2.txt" com o nome "bemvindo.txt" para manter este nos dois diretórios:

```
cp bemvindo2.txt bemvindo.txt
```

```
ls workspace/  
bemvindo2.txt    bemvindo.txt    projetos-java    pro
```

[COPIAR CÓDIGO](#)

Se quisermos buscar os dois arquivos "bemvindos" dentro do diretório *workspace*, fazemos:

```
ls bemvindo*  
bemvindo2.txt    bemvindo.txt
```

[COPIAR CÓDIGO](#)

Perceba que quando usamos o comando `ls` para arquivos, o terminal retorna tais arquivos. Se fizermos o mesmo para diretórios, ele retorna o que estiver dentro deles. E, ainda, se fizermos `ls *`, o terminal retorna:

```
bemvindo2.txt    bemvindo.txt
```

```
projetos-java:  
bemvindo.txt
```

```
projetos-php:
```

[COPIAR CÓDIGO](#)

Ou seja, mostra tanto os diretórios dentro daquele em que estamos trabalhando e seus respectivos arquivos.

Assim como utilizamos o `-r` para conseguirmos apagar diretórios, do mesmo jeito fazemos para copiá-los, como por exemplo:

```
cp -r projetos-java projetos-c#
```

[COPIAR CÓDIGO](#)

Nesta aula aprendemos a mover, renomear e copiar arquivos e diretórios.



## Transcrição

Agora que já sabemos como trabalhar com arquivos e diretórios, copiá-los, movê-los, etc, vamos ver como compactá-los. Queremos, por exemplo, compactar o `workspace`, cujo conteúdo podemos conferir utilizando o comando `ls`:

```
bemvindo2.txt  bemvindo.txt  projetos-c#  projetos
```

[COPIAR CÓDIGO](#)

Primeiramente precisamos estar no diretório ao qual ele pertence com o comando `cd ..` para subir um diretório acima e depois usamos o comando `zip` informando o nome do arquivo que será gerado e o que o comando deve compactar:

```
zip work.zip workspace/
```

[COPIAR CÓDIGO](#)

A mensagem `adding: workspace/ (stored 0%)` será exibida. Lembre-se que `work.zip` é o nome que demos para o arquivo de compactação. Podemos verificar rapidamente o conteúdo do arquivo compactado utilizando o comando `unzip -l work.zip` (o comando após `unzip` é a letra L minúscula, e não o número 1) dessa forma conseguimos observar quais arquivos e diretórios foram compactados.

É importante observar que, da mesma forma que os comandos para apagar e copiar, o comando `zip` não é suficiente para compactar todos os arquivos e diretórios dentro de `workspace`. Se você conferiu o conteúdo do *zip*, percebeu que apenas o diretório `workspace` foi compactado, mas nenhum dos seus arquivos e subdiretórios estava presente no *zip* final. Para isso precisamos usar a ferramenta de recursividade aqui também, ou seja, o `-r` em conjunto com o comando `zip`:

```
zip -r work.zip workspace/
```

[COPIAR CÓDIGO](#)

Esse comando apresentará uma saída diferente, informando os arquivos encontrados e adicionados ao *zip* final.

```
updating: workspace/ (stored 0%)
  adding: workspace/bemvindo2.txt (stored 0%)
  adding: workspace/projetos-php/ (stored 0%)
  adding: workspace/projetos-java/ (stored 0%)
  adding: workspace/projetos-java/bemvindo.txt (st
  adding: workspace/bemvindo.txt (stored 0%)
  adding: workspace/projetos-c#/ (stored 0%)
  adding: workspace/projetos-c#/bemvindo.txt (stor
```

[COPIAR CÓDIGO](#)

Para descompactar o arquivo *zip* utilizamos o comando `unzip`, informando o nome do arquivo `work.zip`.

```
Archive: work.zip
  creating: workspace/
  extracting: workspace/bemvindo2.txt
  creating: workspace/projetos-php/
```

```
creating: workspace/projetos-java/  
extracting: workspace/projetos-java/bemvindo.txt  
extracting: workspace/bemvindo.txt  
creating: workspace/projetos-c#/  
extracting: workspace/projetos-c#/bemvindo.txt
```

[COPIAR CÓDIGO](#)

Descompactar os arquivos do *zip* no mesmo diretório irá sobrescrever os arquivos já existentes. Experimente remover o diretório `workspace` com o comando `rm` antes de descompactar o arquivo `work.zip`. Utilize também o comando `ls` para verificar o resultado de cada comando.

Perceba que tanto o *zip* quanto o *unzip* são comandos muito verborrágicos, ou seja, imprimem muita informação. O modificador `-q` permite que o retorno seja vazio (apesar de o resto funcionar perfeitamente):

```
unzip -q work.zip
```

[COPIAR CÓDIGO](#)

E para *zip*:

```
zip -rq work.zip workspace
```

[COPIAR CÓDIGO](#)

Podemos deixar juntos o `-q` e o comando de recursividade `-r` formando o `-rq`.

Nesta aula aprendemos os comandos para compactar e descompactar arquivos, além de esconder as informações que eles retornam, com o

comando -q .



### Transcrição

Aprendemos a compactar arquivos e diretórios utilizando a extensão *zip*, com o comando `zip`, que é uma extensão muito famosa no sistema operacional Windows. No Linux, uma outra extensão é mais convencional de se utilizar, a extensão ***tar***. Vamos, então, fazer o mesmo trabalho da aula passada mas utilizando o comando `tar`. Para compactar o diretório *workspace* fazemos:

```
tar -cz workspace > work.tar.gz
```

[COPIAR CÓDIGO](#)

O `tar` sozinho não serve para compactar arquivos. Na verdade o `tar` serve para empacotar vários diretórios e arquivos em um único arquivo, facilitando a transferência. Para compactar usaremos o `tar` combinado com o `zip`, o primeiro empacota e o segundo compacta.

O modificador `-cz` indica que o arquivo *tar* será criado (`-c`) e será compactado pelo *zip* (`-z`) usando o redirecionamento `>`. Uma observação interessante é que comando `tar` já é automaticamente recursivo.

Lembre-se de utilizar o comando `ls` para verificar o arquivo criado e de remover o diretório *workspace* antes de descompactar os arquivos com o comando `rm`.



Para descompactar o arquivo `.tar.gz` que criamos, fazemos:

```
tar -xz < work.tar.gz
```

[COPIAR CÓDIGO](#)

Perceba que houveram apenas duas diferenças em relação ao comando de compactação, a presença `-x` de "*extract*", para extrair os arquivos e a direção do redirecionamento `<` que agora em vez de indicar saída de dados, indica entrada de dados.

Mas não é muito bom ficarmos o tempo todo trabalhando com redirecionamento, é meio chato. Para isso o comando `tar` possui o modificador `-f` para podermos passar o nome do arquivo que queremos criar:

```
tar -czf work.tar.gz workspace/
```

[COPIAR CÓDIGO](#)

Note que inversão no nome do arquivo e diretório a ser compactado, com redirecionamento, o nome do arquivo final vem depois, agora ele está vindo antes. Agora para descompactar fazemos:

```
tar -xzf work.tar.gz
```

[COPIAR CÓDIGO](#)

Uma outra característica que faz o comando `tar` diferente do `zip` é que ele não é verborrágico por padrão. Consequentemente não precisamos dar comandos para que as informações de compactação não apareçam. Pelo contrário, para que as informações apareçam, usamos o modificador `-v` (de "*verbose*"):

```
tar -vxzf work.tar.gz
```

[COPIAR CÓDIGO](#)

A saída será semelhante a mostrada abaixo:

```
workspace/  
workspace/bemvindo2.txt  
workspace/projetos-php/  
workspace/projetos-java/  
workspace/projetos-java/bemvindo.txt  
workspace/bemvindo.txt  
workspace/projetos-c#/  
workspace/projetos-c#/bemvindo.txt
```

[COPIAR CÓDIGO](#)

Existem muitos modificadores para compactar e descompactar diretórios, tanto em `zip` quanto em `tar`. Por isso podemos recorrer sempre que necessário à documentação usando o comando `man`. Como por exemplo: `man tar` para sabermos sobre outras opções de modificadores disponíveis para o comando `tar`.



## Transcrição

Se entrarmos dentro do diretório `workspace` e observarmos os detalhes do arquivo `bemvindo.txt` com a ajuda do comando `ls -la` veremos a algo como seguinte saída para este arquivo:

```
[...]  
-rw-rw-r-- 1 user user 10 Jun 11 15:42 bemvindo.tx  
[...]
```

[COPIAR CÓDIGO](#)

Repare que a data da última atualização dele é de 11 de Junho às 15:42. Se quisermos apenas que essa data modifique-se para a data e hora atual, isto é, apenas encostar nesse arquivo sem modificá-lo, usamos o comando `touch` :

```
touch bemvindo.txt
```

[COPIAR CÓDIGO](#)

Após isso, se verificarmos seus detalhes novamente com o comando `ls -la` , veremos que a data e hora de modificação foram alterados para as mais atuais:

```
[...]  
-rw-rw-r-- 1 user user 10 Jun 11 22:15 bemvindo.tx  
[...]
```

O arquivo em si não altera-se, mas sua data de modificação sim. Para provar que realmente houve essa mudança, podemos ver o horário do sistema com o comando `date` e comparar com as informações do arquivo. A saída do comando `date` é semelhante a encontrada abaixo:

Tue Jun 11 22:15:33 BRT 2013

COPIAR CÓDIGO

De fato, acabamos de "modificar" o arquivo, **não seu conteúdo, mas sua data de última modificação.**



## Para saber mais: obtendo ajuda e formatando datas



100%

ATIVIDADES  
6 DE 11FÓRUM DO  
CURSOVOLTAR  
PARA  
DASHBOARD

Já vimos que podemos obter ajuda na documentação do Linux, as man pages, através do comando `man` . Uma outra forma de obter ajuda sobre a utilização do comando, de uma forma mais resumida, é utilizando o parâmetro `--help` suportado por alguns comandos, ou utilizando o comando `help` .

O `help` é um comando interno do interpretador `bash` , que, quando executado sem parâmetros, retorna uma lista com todos os demais comandos internos do shell `bash` . Quando executado com parâmetro, o comando `help` retorna a sintaxe de utilização e uma descrição do comando interno que estamos interessados.

```
$ help pwd  
$ date --help
```

[COPIAR CÓDIGO](#)

36.3k xp



Quando fazemos `date --help` , encontramos, dentre outras coisas, ajuda sobre como formatar a nossa data. Podemos imprimir, por exemplo, a data no formato `dia/mes/ano` :



```
$ date "+%d/%m/%Y"
```

23/01/2016

COPIAR CÓDIGO



100%

Aproveite para testar esses comandos no seu Linux!

ATIVIDADES  
6 DE 11

Sempre que precisar consultar os parâmetros que um comando suporta, você pode pedir ajuda através do `help` ou `--help`.

FÓRUM DO  
CURSO

VOLTAR  
PARA  
DASHBOARD



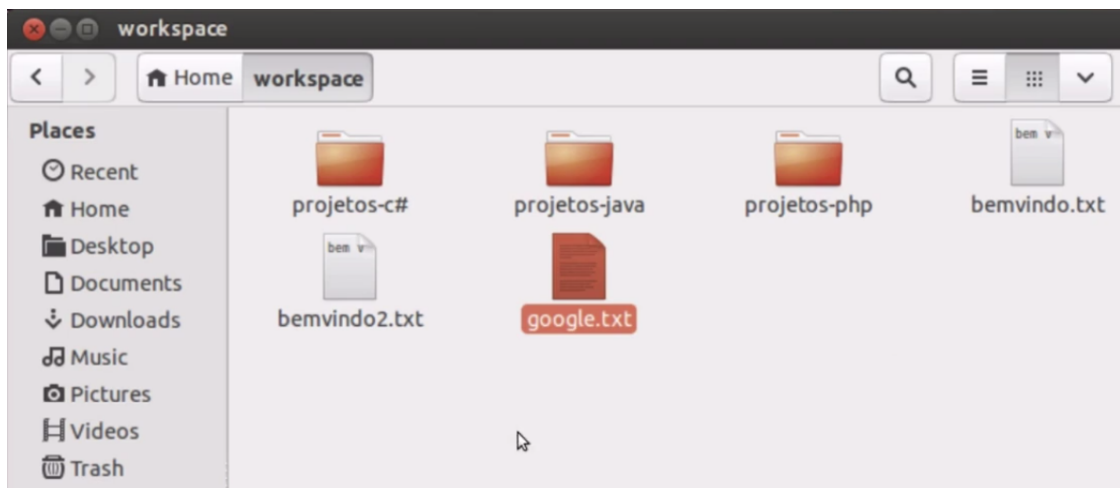
36.3k xp

a



## Transcrição

Continuando nossos estudos sobre manipulação de arquivos, vamos primeiramente, criar um da maneira visual, por meio do navegador de pastas um arquivo de texto chamado `google.txt` dentro do diretório `workspace` :



Vamos adicionar conteúdo nesse arquivo. Para isso entremos no site da [Wikipedia \(https://en.wikipedia.org\)](https://en.wikipedia.org) na página que fala sobre o [Google \(https://en.wikipedia.org/wiki/Google\)](https://en.wikipedia.org/wiki/Google) e vamos copiar uma parte considerável do texto para o arquivo.

No terminal, sabemos que podemos utilizar o comando `cat` para ler o arquivo criado, ele nos retornará todo o texto:

```
cat google.txt
```

[COPIAR CÓDIGO](#)

Porém é muito texto. Se quisermos ler apenas o começo do arquivo, usamos o comando `head`. Ele retorna apenas as dez primeiras linhas do arquivo:

```
head google.txt
```

[COPIAR CÓDIGO](#)

Para especificarmos a quantidade de linhas que queremos retornar, podemos usar a opção `-n` informando o número de linha queremos:

```
head -n 3 google.txt
```

[COPIAR CÓDIGO](#)

Dessa maneira o Terminal retorna apenas as três primeiras linhas do começo do arquivo. O inverso também é possível, ler as últimas linhas do arquivo. Para lermos as últimas linhas usamos o comando `tail`:

```
tail google.txt
```

[COPIAR CÓDIGO](#)

O `tail` por padrão retorna as dez últimas linhas, assim como o `head`, para alterar esse limite, podemos usar o `-n` assim como fizemos no comando anterior:

```
tail -n 3 google.txt
```

[COPIAR CÓDIGO](#)

Além do `head` e do `tail`, ainda temos o comando `less`, que nos permite abrir e navegar pelo texto do arquivo no Terminal utilizando as setas para cima e para baixo do teclado:



```
less google.txt
```

[COPIAR CÓDIGO](#)

E podemos fechar o arquivo utilizando a tecla `q` do teclado.



## Copiar e colar



100%

ATIVIDADES  
12 DE 13FÓRUM DO  
CURSOVOLTAR  
PARA  
DASHBOARD

Vamos agora utilizar o `vi` e abrir o arquivo `google.txt` :

```
vi google.txt
```

[COPIAR CÓDIGO](#)

Copie as três primeiras linhas do arquivo, salte para a linha 40 do arquivo e cole cinco vezes o conteúdo que foi copiado.



## Opinião do instrutor



Para copiar as três primeiras linhas utilizamos o comando `3yy` . Para ir até a linha de número 40 podemos executar o comando `40G` (Shift + g) . Por fim, para colar cinco vezes utilizamos o comando `5p` .



36.3k xp







### Transcrição

Na aula passada nós editamos um arquivo `.txt` via um editor de textos visual, vimos também como escrever em arquivos utilizando o comando `echo`, mas nesta aula veremos como editar arquivos de texto através do Terminal, usando um editor próprio dele. Este editor é o **VI**. Para abrir o arquivo `google.txt` no **VI** fazemos:

```
vi google.txt
```

[COPIAR CÓDIGO](#)

O texto do arquivo aparece para nós e podemos navegar por ele utilizando as setas do teclado. Para sairmos do modo de navegação e entrarmos no modo de *inclusão*, apertamos a tecla `i`, assim podemos inserir textos no arquivo exatamente no lugar onde pressionamos o `i`. Para voltarmos à navegação e sairmos do modo de inclusão, pressionamos a tecla `ESC`. Para salvarmos essas alterações usamos o comando `:w` e para sair do **VI**, digitamos `:q`.

Experimente editar algumas linhas no começo do arquivo, insira algumas linhas de texto, salve, saia do editor e para verificar as alterações, experimente utilizar o comando `head` que você aprendeu na aula anterior.

Vimos que para inserir textos na posição onde o cursor se encontra, pressionamos a tecla `i`, mas podemos inserir textos usando também

a tecla `a` , neste caso inclusão de texto será feita na posição seguinte de onde estiver o cursor.

Sabemos como inserir texto, mas e para excluir? Para excluir caracteres usamos a tecla `x` , ela funciona como o *delete* do teclado. Porém esse comando tem algo diferente: se digitarmos o número de caracteres que queremos apagar e logo depois a tecla `x` , serão apagados a quantidade digitada, é como se tivéssemos apertado o `x` várias vezes, algo como `11 x` , apagará 11 caracteres de texto. O `x` já ajuda bastante, mas caso queiramos excluir uma linha inteira, ele pode não ajudar muito, porém, para deletarmos uma linha inteira podemos simplesmente digitar a tecla `d` duas vezes (`dd`), digitar uma quantidade antes do `dd` funciona da mesma forma que no comando `x` .

Tente sair do editor sem salvar o arquivo. Dará erro, isso porque temos que salvar o arquivo para podermos sair. Podemos juntar os comandos de salvar e sair em apenas um comando: `:wq` . Se quisermos sair sem salvar, precisamos usar o comando `:q!` .

Vimos bastantes comandos do *VI* até aqui, mas todos eles são teclas de letras em minúsculo, isso porque letras em maiúsculas possuem um comportamento diferente, o `A` (`shift + a`) por exemplo, também é um comando para edição de texto, mas ao contrário da `i` e `a` , o `A` insere texto no final da linha atual.

Resumindo os comandos que vimos do *VI*, até agora temos:

- Setas: para navegação
- `i`: inclui (na posição atual)
- `a`: adiciona (na posição posterior)
- `Shift+A`: adiciona (fim da linha)

- x: remove caracteres ( $n^*$  *x* *remove*  $^*n$  caracteres)
- dd: remove uma linha ( $n^*$  *dd* *remove*  $^*n$  linhas)



## Transcrição

Continuando nossos estudos com o *VI*, aprenderemos mais sobre navegação, muito mais além das setas do teclado.

Se quisermos ir para a última linha do texto por exemplo, basta apertarmos `Shift + g`. Se quisermos pular para a linha *n*, apertamos o número correspondente e depois `Shift + g`. Então se, por exemplo, quisermos ir para a linha 5, fazemos `5` e depois `Shift + g`, para a primeira linha a combinação também é válida: `1` e depois `Shift + g`, sempre com `Shift` para que o `g` seja maiúsculo.

Se quisermos ir para o final da linha atual usamos a tecla `$`, ou seja, `Shift + 4` e para ir ao início da linha digitamos `0`.

Algo bem comum ligado à navegação de um arquivo é a procura de palavras. Para buscarmos palavras no texto usando o *VI*, digitamos `"/` + o texto que procuramos, algo como:

```
/California
```

[COPIAR CÓDIGO](#)

Se digitarmos isso, o cursor vai para a primeira ocorrência da palavra `California`. Se apertarmos a tecla `n`, ele irá para a próxima ocorrência e com `Shift + n`, podemos voltar para a anterior.







## **Transcrição**

Um recurso muito comum de ser utilizado nas edições de texto são o copiar e colar, os quais veremos como fazer no VI. Para copiarmos uma linha do texto apertamos a tecla `y` duas vezes e para colar o que foi copiado, basta usar a tecla `p`.

Também podemos usar a ideia de quantidade de linhas com os números para copiar e colar. Se quisermos copiar a linha atual e mais as duas linhas abaixo podemos fazer `3 y` - três linhas foram copiadas. O mesmo serve para o comando de colar: `3 p` cola três vezes o conteúdo copiado anteriormente.