04

Nomeação de arquivos

Bem- vindo a mais um curso preparatório de certificação do *LPI Essentials*.

Nesse curso vamos falar bastante sobre arquivos e diretórios. Desde, quais são nomes válidos e quais são inválidos, o que os nomes significam, seja de um arquivo, seja de um diretório e o que o nome implica para o sistema de arquivos. Veremos quais são comandos comuns na manipulação de diretórios e arquivos. Quais são as diferenças de trabalhar com caminhos relativos e caminhos absolutos e abordaremos, ainda, diversos comandos tanto sobre criação quanto de manipulação e remoção de arquivos e diretórios.

Veremos tudo isso nesse curso.

Nomes de arquivos e diretórios

Vamos começar a falar sobre os diretórios e a listagem de arquivos, ou seja, como trabalhar com listagem de arquivos. Primeiramente, é cobrado o que são arquivos e diretórios.

Diretórios contêm diversos arquivos e outros diretórios. Já vimos em nosso terminal que o diretório raiz, inicial, o diretório `root` não é o usuário `root`, o "superusuario", e sim é o diretório `root`. São duas coisas distintas. O diretório raiz é o diretório `/`.

O que tem nele?

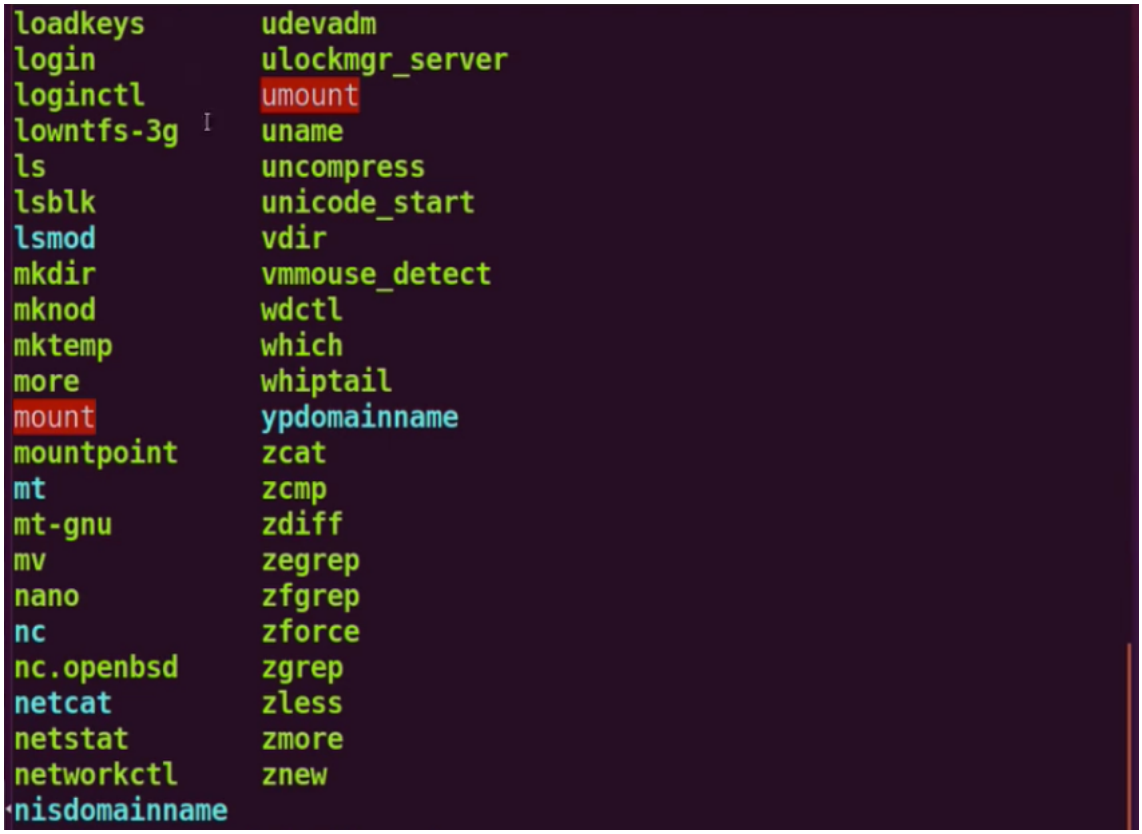
Nele encontramos diversas coisas. Vamos digitar `ls /` e observar o que temos:

```
guilherme@ubuntudesktop:~$ ls /
bin    dev   initrd.img  lost+found  opt   run   sys  var
boot   etc   lib         media       proc  sbin  tmp  vmlinuz
cdrom  home  lib64       mnt         root  srv   usr
```

Temos nesse diretório diversas coisas e como o nosso *Ubuntu* já está configurado para fazer um `ls` mais bonitinho devido ao `--color=auto` ele já nos mostra o os diretório, diferenciando-os em azul, como o `bin`, o `dev`, o `lib64` e etc.

Lembre-se que podemos observar como isso já está configurado se dermos `type ls`. Teremos como resposta `ls is aliased to 'ls --color=auto'`.

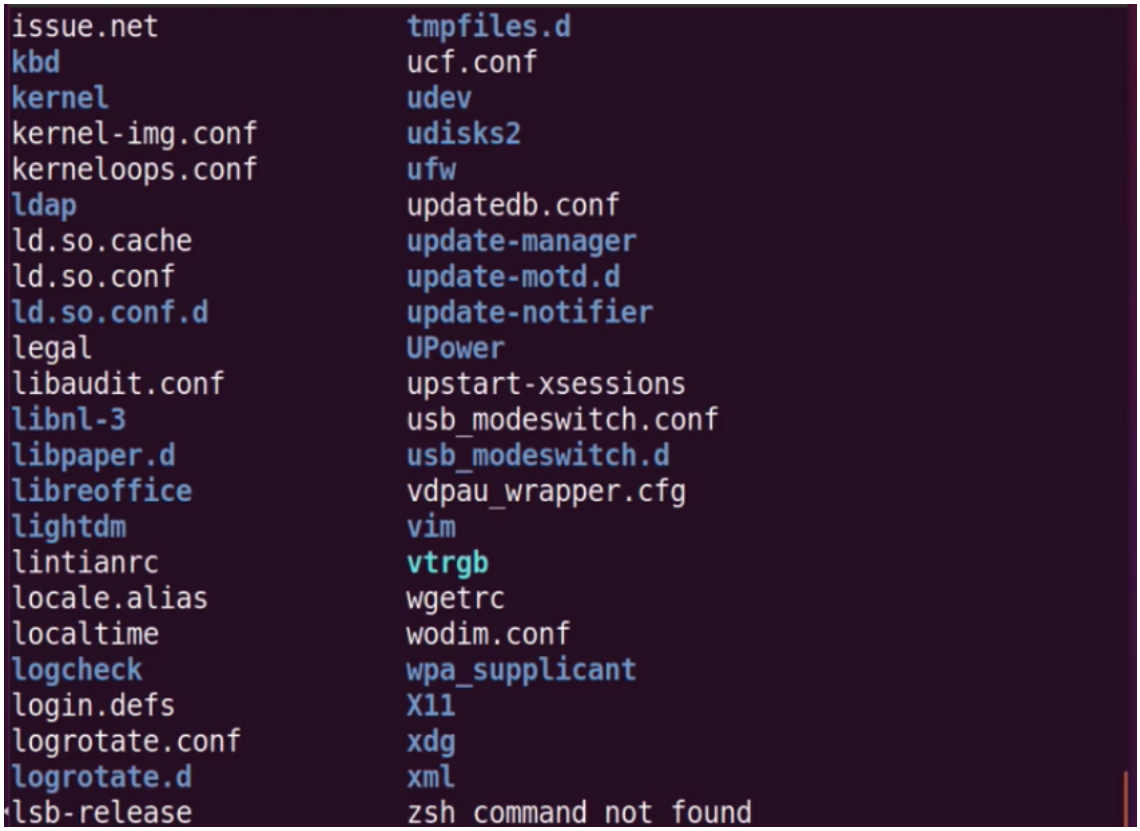
O `ls` nos mostra diversas características e o que estamos fazendo nesse instante é olhando com mais calma o que temos em nosso diretório raiz, o `/`. Dentro do nosso diretório raiz temos, por exemplo, o diretório `bin`. Vamos observar o que temos nesse diretório digitando `ls /bin`:



Dentro dele temos diversos programas que são executáveis. Vamos observar, novamente, o diretório raíz, digitando `ls /` :

```
> ls /
bin         dev     initrd.img      lost+found      opt         run         sys     var
boot        etc     lib             media           proc        sbin        tmp     vmlinuz
cdrom       home    lib64           mnt             root        srv         usr
```

Dentro dele temos o diretório `etc` . Vamos analisar o que temos nesse `etc` digitando `ls /etc` :



Dentro dele, podemos observar que temos diversos outros diretórios. Por exemplo, o `/etc/udev`. Vejamos o que ele nos mostra se digitarmos `ls /etc/udev`:

```
ls /etc/udev
hwdb.d  rules.d udev.conf
```

Podemos ver mais outro diretório do `etc` que é o `/udisks2`. Vamos digitar `ls /etc/udisks2` e nele não encontraremos nada.

Fizemos esse caminho para demostrar uma coisa! Repare que começamos com um diretório raiz, que é o diretório `/` e dentro dele temos diversos outros diretórios e dentro de cada diretório temos diversos arquivos. Inclusive, passamos por vários desses diretórios a medida em que fomos explorando o *Linux*.

Falaremos, mais adiante, sobre alguns diretórios mais especiais, mas nesse momento queremos entender como funcionam diretórios e arquivos.

Para avançarmos sobre a questão dos diretórios e arquivos, primeiro, temos que falar sobre nomes válidos para esses diretórios e arquivos.

O diretório raiz se chame `/`, portanto, não faz sentido termos um arquivo chamado "/" ou um diretório dentro do diretório atual que também seja nomeado de "/". Como poderíamos dizer, por exemplo, no caso de termos mais de um diretório chamado "/", que queremos algo relacionado ao diretório "/", que não é o diretório raiz? Como ele saberia que estamos falando de um e não de outro?

O diretório `/` é solto, é um carácter específico, ele é especial na nossa representação de caminhos de arquivos, de `file paths`. Ele é um carácter que indica a raiz de nossos diretórios e por esse motivo não podemos criar um arquivo que tenha em seu nome `/`, independente de ser seguido de mais alguma coisa ou não. Tampouco podemos criar um arquivo que se chama alguma coisa e que seja seguido de uma `/` no meio.

Vamos tentar criar um arquivo?

Vamos abrir o *gedit* e vamos escrever em um novo arquivo "Meu arquivo" e vamos tentar salvá-lo em `/home/guilherme` com o nome de "arquivo/saida.txt". Ao selecionarmos a opção para criar esse arquivo aparece uma caixa que nos diz: "The folder contents could not be displayed". Ou seja, ele nos diz, "Me desculpa, mas não conseguimos gravar isso que você está querendo".

Isso acontece porque `/home/guilherme/arquivo` não existe. O que aconteceu? Ele interpretou a `/` como sendo um separador de diretórios. E a barra é justamente isso, sua função é ser um demarcador de diretório, a `/` começa, por padrão, indicando a raiz, mas se ela estiver no meio do caminho de um `path`, ele indica diretório.

Por isso, simplesmente, não podemos utilizar a barra como um carácter válido para o nome de um arquivo.

Então, que outros nomes podemos usar?

Vamos avançar e analisar quais são os nomes possíveis de serem utilizados. Vamos voltar ao arquivo que estávamos tentando salvar. Podemos tentar nomear esse arquivo de "guilherme.txt"? Pode! Não tem problema nenhum. Agora, já podemos salvar esse arquivo e voltar ao Terminal que ele deve estar no `ls`. Vamos digitar `ls` e procurar ele:

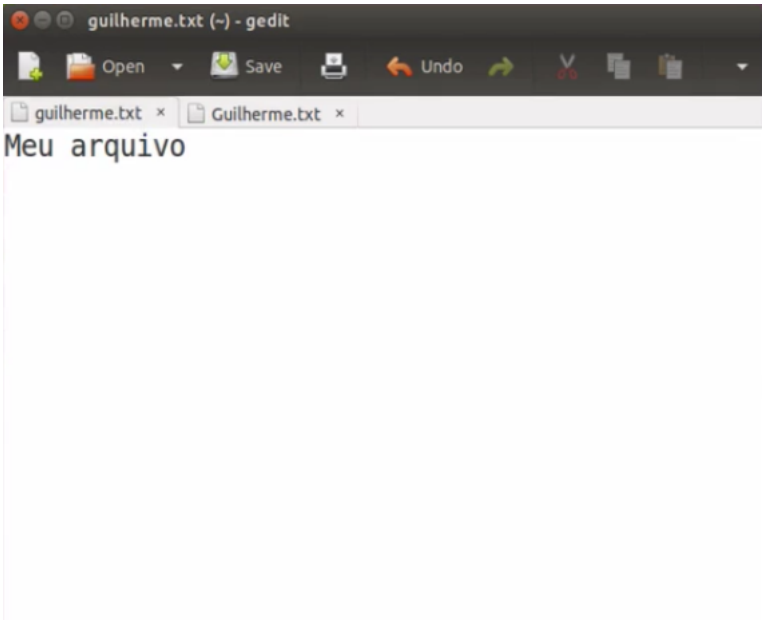
```
> ls
arquivos.zip          falha~                mostra_idade        sucesso~
Desktop               guilherme.txt         mostra_idade~       temp
Documents             help                  Music               Templates
Downloads             log completo.txt      Pictures            Videos
examples.desktop      log completo.txt~     Public              zip
falha                 loja                  sucesso
```

O `guilherme.txt` está no nosso `ls`. Inclusive, se quisermos ler esse arquivo usando o `cat guilherme.txt` teremos a comprovação de que tudo funciona bem, pois teremos como resposta "Meu arquivo", ou seja, que ele é um arquivo de texto!

```
> cat guilherme.txt
Meu arquivo
```

Que outros nomes são válidos para esse arquivo?

Podemos chamar ele de `Guilherme.txt` com letra maiúscula no início? Vamos tentar fazer isso, voltamos ao nosso *gedit* e criamos esse novo arquivo, cujo nome será "Guilherme.txt". Salvamos isso e acrescentamos nele o texto "Meu segundo arquivo".



Temos, portanto, dois arquivos, repare na figura acima. Um com o nome de "guilherme.txt" com letra minúscula e o outro "Guilherme.txt" com letra maiúscula. Se dermos um `cat` em `Guilherme.txt` teremos o seguinte:

```
> cat Guilherme.txt
Meu segundo arquivo
```

E se dermos `cat` em `guilhermes.txt` teremos o seu respectivo arquivo também:

```
> cat guilherme.txt
Meu arquivo
```

Repare que o *Linux*, diferente de alguns outros sistemas operacionais, por exemplo o *Windows*, permite o uso de letras maiúsculas e minúsculas e estas são capazes de diferenciar coisas.

Por isso, para evitar problemas, o que não fazemos é criar dois arquivos com um mesmo nome e cuja diferença seja apenas o uso de letras maiúsculas e minúsculas, não importa se essa diferença esta no primeiro carácter ou no décimo quinto, isso acaba tornando difícil saber qual dos dois se quer abrir. Por mais que esses dois arquivos sejam distintos na prática a recomendação é: Não use nomes que são equivalentes e cuja única diferença encontra-se no maiúsculo ou minúsculo de alguma letra. Se começarmos a utilizar isso pode ocorrer de nos perdermos e ficarmos sem saber com qual dois dois queremos trabalhar.

E, pior que isso, se você deseja copiar esses arquivos para o *Windows*, ou outro sistema operacional, que não é *case sensitive*, isto é, que não consegue diferenciar entre letras maiúsculas e minusculas, você não conseguirá fazer isso. Pois, os dois arquivos, diante desse outro sistema operacional, possuem o mesmo nome. Portanto, se usarmos os mesmo nomes diferenciando apenas na letras, simplesmente, não conseguiremos copiar os arquivos para o *Windowns*, pois eles possuem o mesmo nome nesse sistema. O que iria acontecer é que um subscreveria o outro.

Na prática, o *Linux* é *case sensitive* para o nome dos arquivos e diretórios. O que ocorre é que ele é sensível ao *case*, ao maiúsculo e minúsculo. Mas, na prática não queremos criar dois arquivos com o mesmo nome e que contenham apenas essa diferença.

O que fazemos?

Seguimos um padrão, diretórios, em geral, na maior parte das vezes, usam letras em minúsculo. Diretórios onde o usuário final cria, por exemplo, alguém que grava um arquivo de áudio ou cria arquivos de texto, é permitido que ele crie o que quiser, nesse caso, pode ser tanto usando maiúsculo quanto minúsculo.

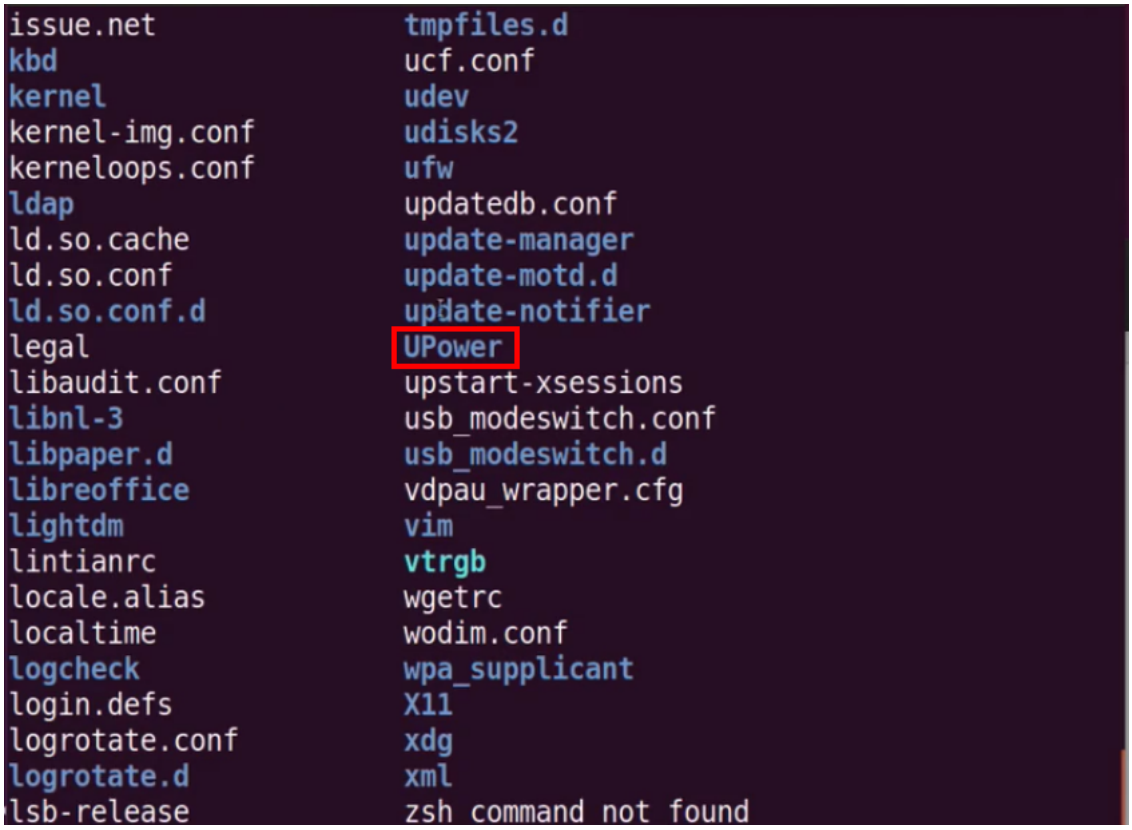
Tanto é verdade que se dermos um `ls` no diretório base do nosso usuário, na `home` do usuário, veremos que temos diversas coisas:

```
> ls
arquivos.zip            falha~                mostra_idade        sucesso~
Desktop                 guilherme.txt         mostra_idade~       temp
Documents               help                  Music               Templates
Downloads               log completo.txt      Pictures            Videos
examples.desktop        log completo.txt~     Public              zip
falha                   loja                  sucesso
```

E que o `Desktop` , o `Pictures` , `Public` , `Templates` , `Videos` , `Documents` , `Downloads` , `Music` , ou seja, todos os diretórios que já vieram com o usuário possuem, por padrão, em maiúsculo a primeira letra e quem faz isso por nós é o *Ubuntu*. Mas, repare que todos os outros que podemos encontrar no `ls /` que é o raiz, possuem letras minúsculas:

```
> ls
bin         dev     initrd.img      lost+found      opt         run         sys         var
boot        etc     lib             media           proc        sbin        tmp         vmlinuz
cdrom       home    lib64           mnt             root        srv         usr
```

Podemos, inclusive, entrar em algum desses diretórios, por exemplo, o `ls /etc` e poderemos observar que teremos tudo em minúsculo. Observe:



Você pode reparar que no meio disso temos um `UPower` . Por algum motivo ele foge desse padrão que vinha sendo usado nos diretórios de um sistema operacional. Mas, podemos perceber através de uma olhada rápida que o padrão é sempre que

possível utilizar letra minúscula no sistema operacional. E mais importante do que isso para nós é saber que ele é sensível ao *case*, então, se temos um `ls` e queremos ler o arquivo `mostra_idade` teremos que digitar tudo em minúsculo:

```
> ls mostra_idade
echo ${FULL NAME} voce tem ${AGE} anos.
job=Instrutor
```

Se tentarmos digitar isso em maiúsculo `cat MOSTRA_IDADE` estaremos nos referindo, na verdade, a outro arquivo. E, justamente, porque existe uma diferenciação, é que pudemos criar dois arquivos distintos, o `Guilherme.txt` e o `guilherme.txt` .

Então, já sabemos que o *Linux* é sensível a maiúsculo e minúsculo, o que é extremamente importante para nós. Já sabemos, também, que a `/` não pode ser utilizada como um nome válido para um arquivo. Se estamos em nosso diretório atual e falamos que queremos pegar o arquivo que se chama `Documents/log10.txt` teremos o seguinte:

```
> cat Documents/log-2016-05-10.txt
log 10
```

Repare que ele nos mostra o arquivo.

E o que essa barra representa?

Representa que, o que temos a esquerda da barra é um diretório, então, observe, se o barra fosse válida como nome de um arquivo, esse `Documents/log-2016-05-10.txt` seria o que? Um diretório nome de arquivo ou um nome de arquivo? Ficaria ambíguo! Para evitar essa ambiguidade o *Linux* não permite que o caracter `/` seja válido como nome de arquivo e temos.

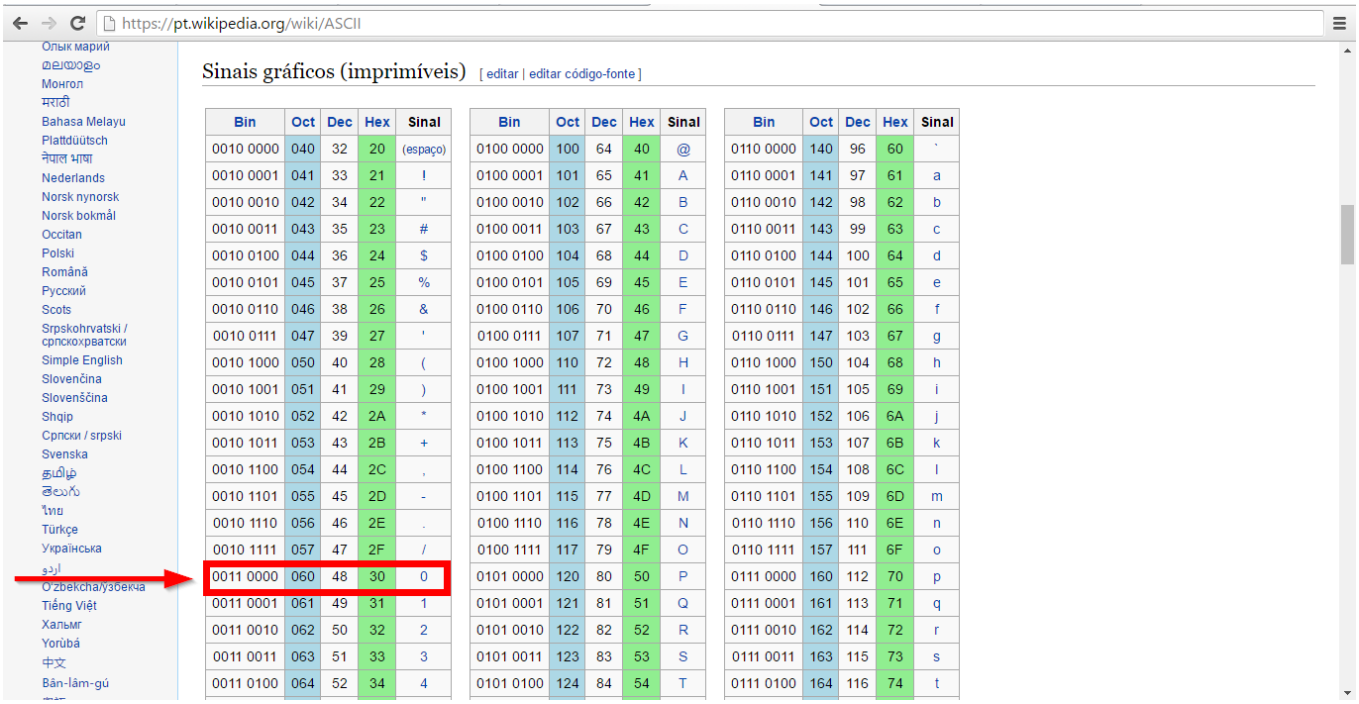
Temos, ainda, mais um caso especial que não é aceito como caracter válido como nome de arquivo.

Quem é esse caracter?

É algo que possui o código "0". Mas, o arquivo com nome "0" podemos criar. Vamos no *gedit* e escrevemos nesse novo arquivo apenas "0" e salvamos como "0". Conseguimos salvar isso. Vamos voltar ao Terminal, se dermos um `ls` encontraremos o arquivo chamado "0". Observe:

```
> ls
0                   falha~              loja                sucesso~
arquivos.zip        guilherme.txt       mostra_idade~       temp
Desktop             Guilherme.txt       mostra_idade~       Templates
Documents           Guilherme.txt~      Music               Videos
Downloads           help                Pictures            zip
examples.desktop    log  completo.txt   Public
falha               log  completo.txt   sucesso
```

Com isso, percebemos que um arquivo de nome "0" é possível de ser criado, entretanto, um arquivo com código 0, não pode ser criado. O "0" é um algorismo, se observarmos a tabela *ASCII* no site https://pt.wikipedia.org/wiki/ASCII (https://pt.wikipedia.org/wiki/ASCII), e formos na tabela e buscarmos os números, encontraremos o "0". Veremos que, segundo essa tabela, ele possui o valor 42. Portanto, o número 1, 2, 3 e etc são números válidos como nomes de arquivos. Podemos utilizar esse números como nomes e criá-los que funciona.



https://pt.wikipedia.org/wiki/ASCII

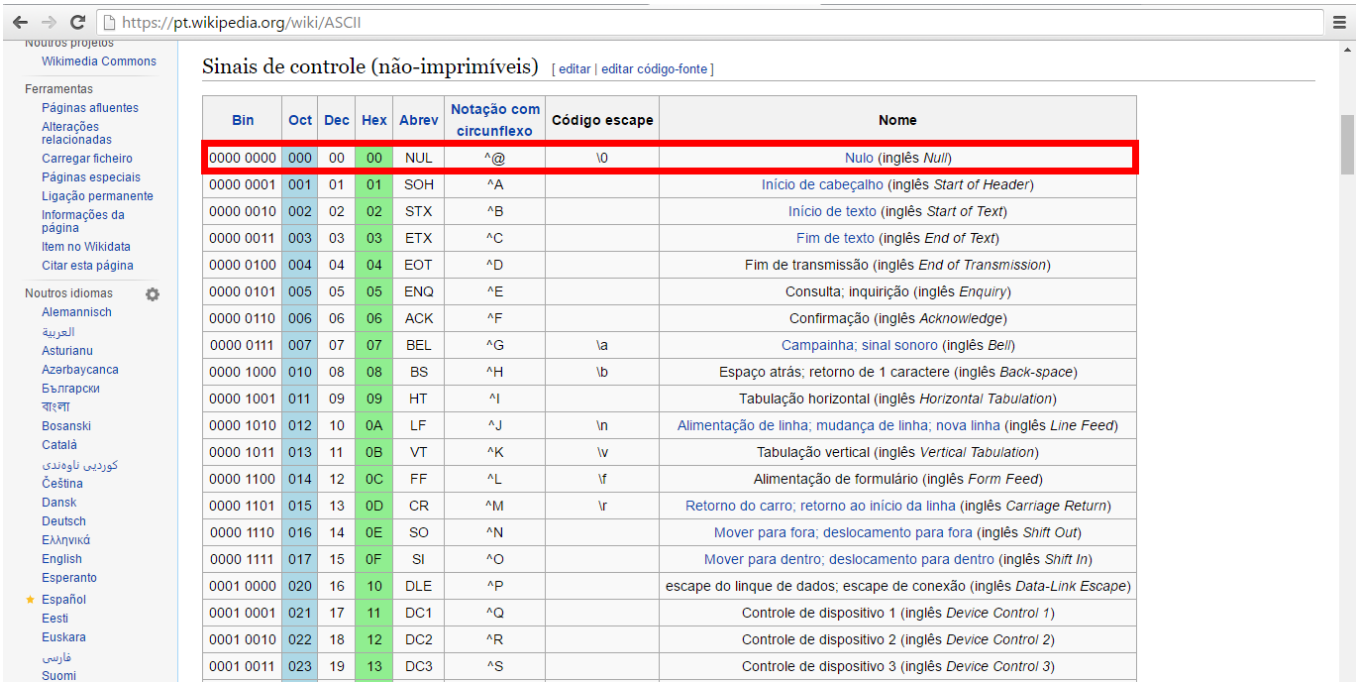
Sinais gráficos (imprimíveis) [editar | editar código-fonte]

Bin	Oct	Dec	Hex	Sinal
0010 0000	040	32	20	(espaço)
0010 0001	041	33	21	!
0010 0010	042	34	22	"
0010 0011	043	35	23	#
0010 0100	044	36	24	$
0010 0101	045	37	25	%
0010 0110	046	38	26	&
0010 0111	047	39	27	'
0010 1000	050	40	28	(
0010 1001	051	41	29	)
0010 1010	052	42	2A	*
0010 1011	053	43	2B	+
0010 1100	054	44	2C	,
0010 1101	055	45	2D	-
0010 1110	056	46	2E	.
0010 1111	057	47	2F	/
0011 0000	060	48	30	0
0011 0001	061	49	31	1
0011 0010	062	50	32	2
0011 0011	063	51	33	3
0011 0100	064	52	34	4

Bin	Oct	Dec	Hex	Sinal
0100 0000	100	64	40	@
0100 0001	101	65	41	A
0100 0010	102	66	42	B
0100 0011	103	67	43	C
0100 0100	104	68	44	D
0100 0101	105	69	45	E
0100 0110	106	70	46	F
0100 0111	107	71	47	G
0100 1000	110	72	48	H
0100 1001	111	73	49	I
0100 1010	112	74	4A	J
0100 1011	113	75	4B	K
0100 1100	114	76	4C	L
0100 1101	115	77	4D	M
0100 1110	116	78	4E	N
0100 1111	117	79	4F	O
0101 0000	120	80	50	P
0101 0001	121	81	51	Q
0101 0010	122	82	52	R
0101 0011	123	83	53	S
0101 0100	124	84	54	T

Bin	Oct	Dec	Hex	Sinal
0110 0000	140	96	60	`
0110 0001	141	97	61	a
0110 0010	142	98	62	b
0110 0011	143	99	63	c
0110 0100	144	100	64	d
0110 0101	145	101	65	e
0110 0110	146	102	66	f
0110 0111	147	103	67	g
0110 1000	150	104	68	h
0110 1001	151	105	69	i
0110 1010	152	106	6A	j
0110 1011	153	107	6B	k
0110 1100	154	108	6C	l
0110 1101	155	109	6D	m
0110 1110	156	110	6E	n
0110 1111	157	111	6F	o
0111 0000	160	112	70	p
0111 0001	161	113	71	q
0111 0010	162	114	72	r
0111 0011	163	115	73	s
0111 0100	164	116	74	t

Agora, vamos reparar no número "00", ele é que não é válido. Repare que ele não é um carácter válido, na verdade ele é o código "0" da tabela *ASCII* e esse código "0" é que não funciona.



https://pt.wikipedia.org/wiki/ASCII

Sinais de controle (não-imprimíveis) [editar | editar código-fonte]

Bin	Oct	Dec	Hex	Abrev	Notação com circunflexo	Código escape	Nome
0000 0000	000	00	00	NUL	^@	\0	Nulo (inglês *Null*)
0000 0001	001	01	01	SOH	^A		Início de cabeçalho (inglês *Start of Header*)
0000 0010	002	02	02	STX	^B		Início de texto (inglês *Start of Text*)
0000 0011	003	03	03	ETX	^C		Fim de texto (inglês *End of Text*)
0000 0100	004	04	04	EOT	^D		Fim de transmissão (inglês *End of Transmission*)
0000 0101	005	05	05	ENQ	^E		Consulta; inquirição (inglês *Enquiry*)
0000 0110	006	06	06	ACK	^F		Confirmação (inglês *Acknowledge*)
0000 0111	007	07	07	BEL	^G	\a	Campainha; sinal sonoro (inglês *Bell*)
0000 1000	010	08	08	BS	^H	\b	Espaço atrás; retorno de 1 caractere (inglês *Back-space*)
0000 1001	011	09	09	HT	^I		Tabulação horizontal (inglês *Horizontal Tabulation*)
0000 1010	012	10	0A	LF	^J	\n	Alimentação de linha; mudança de linha; nova linha (inglês *Line Feed*)
0000 1011	013	11	0B	VT	^K	\v	Tabulação vertical (inglês *Vertical Tabulation*)
0000 1100	014	12	0C	FF	^L	\f	Alimentação de formulário (inglês *Form Feed*)
0000 1101	015	13	0D	CR	^M	\r	Retorno do carro; retorno ao início da linha (inglês *Carriage Return*)
0000 1110	016	14	0E	SO	^N		Mover para fora; deslocamento para fora (inglês *Shift Out*)
0000 1111	017	15	0F	SI	^O		Mover para dentro; deslocamento para dentro (inglês *Shift In*)
0001 0000	020	16	10	DLE	^P		escape do linque de dados; escape de conexão (inglês *Data-Link Escape*)
0001 0001	021	17	11	DC1	^Q		Controle de dispositivo 1 (inglês *Device Control 1*)
0001 0010	022	18	12	DC2	^R		Controle de dispositivo 2 (inglês *Device Control 2*)
0001 0011	023	19	13	DC3	^S		Controle de dispositivo 3 (inglês *Device Control 3*)

Ele é utilizado para representação de finalização de *Strings*, em geral ele representa o nulo, o nada, o final. Então, se estivéssemos utilizando ele estaríamos dizendo que nosso arquivo é um "nada" e não podemos ter como nome de um arquivo um "nada", portanto, esse carácter não é válido, independente da sua posição no nome do arquivo.

Então, sabemos que o resto é válido, significa, então, que podemos criar o nome de um arquivo com `til` ? Nós inclusive já temos arquivos com `til` , toda vez que salvamos um novo arquivo ele faz um backup automático disso e temos o nome do arquivo acompanhado com um til. Vamos salvar aquele arquivo zero após escrevermos no *gedit* a palavra "bonitinho" e ver o que acontece se dermos um `ls` :

```
> ls
0                falha                log completo.txt~      sucesso
0~               falha~               loja                   sucesso~
```

```
arquivos.zip        guilherme.txt          mostra_idade          temp
Desktop             Guilherme.txt          mostra_idade~         Templates
Documents           Guilherme.txt~         Music                 Videos
Downloads           help                   Pictures              zip
examples.desktop    log completo.txt       Public
```

Perceba que agora temos o `0~` , portanto, o til é válido sim. E o til com a letra "a", isso pode ser válido também? É sim!

Outros diversos caracteres podem ser válidos, mas tudo depende do sistema operacional que estamos utilizando, do *unicode* desse sistema, de como ele vai trabalhar para codificar o carácter no nome de um arquivo, ou seja, em *bytes*. O sistema pode estar usando o *tf8*, o *tf16*, *iso88591*, qualquer uma das tabelas de *unicode*, qualquer um dos diversos padrões que são utilizados pelo *Linux* . E, dependendo, o sistema operacional pode suportar alguns caracteres e outros não.

Mas, não foi mencionado que ele suporta tudo? Sim, na verdade ele suporta, mas depende do *Unicode* que estiver sendo utilizado. Na prática, o que fazemos é nos limitar nos caracteres da tabela *ASCII* da parte "USA" e isso entre aspas. Na prática podemos utilizar a parte usável da tabela *ASCII*. A parte usável começa no zero e segue adiante.

Portanto, usa-se bastante do zero até o número 9 e do "A" até o "Z" maiúsculo e do "a" até o "z" minúsculo, inclusive o carácter ponto, ".". Usamos bastante ele o ponto, o *underline*, o til e mais alguns caracteres da tabela *ASCII*, mas, repare que o melhor é fugirmos dos acentos. Pois, não vamos salvar arquivos com o tipo de nome "A~", com acentos da língua portuguesa, com acentos da língua alemã, com caracteres coreanos, pois eles não estão na tabela *ASCII*. Assim, a chance deles simplesmente não serem suportados em outras maquinas é muito grande. Imagine, você tem um programa e aquilo que você queria fazer não vai funcionar só por causa dos caracteres que não rodam em outras máquinas.

Portanto, como foi mencionado, anteriormente, por mais que o *Linux* suporte diversos tipos de caracteres exceto o carácter que tem *code* "0", o nulo, e a barra simples, "/", o resto depende do *unicode* . Por mais que o *Linux* suporte isso, na prática, usamos um conjunto pequeno da tabela *ASCII* para fugir de possíveis problemas de compatibilidade entre sistemas, pois, queremos evitar o risco de criar um nome de um arquivo que pode ser completamente inválido no *Windows* e quando copiamos algo e enviamos para alguém que utilize outro sistema, ele pode não conseguir abrir o arquivo que enviamos, justamente, pois seu nome é inválido.

O importante de lembrar para a prova é que o nome dos aquivos não pode ter barra simples, carácter nulo e buscar evitar ao máximo caracteres que fogem do *ASCII* tradicional. Portanto, evitamos os acentos da língua portuguesa, da língua alemã ou caracteres de outros idiomas, pois eles não existem no padrão *ASCII* e podem existir máquinas onde esse arquivo não funcione como deveria.

Mais sobre nomes de arquivos e diretórios

Mencionamos anteriormente que o "." é algo válido para nomear arquivos. Você pode estar se perguntando se seria válido criar um arquivo chamado "guilherme.txt.txt", ou seja, seria possível criar um arquivo com duas extensões? Vamos testar isso? Primeiro, vamos testar com apenas uma extensão, vamos abrir o *gedit* e tentamos salvar um arquivo, primeiro, com o nome "guilherme.silveira.txt". Após criarmos o arquivos voltamos ao Terminal e damos um `ls` :

```
> ls
0                   guilherme.txt          Pictures
0~                  Guilherme.txt          Public
arquivos.zip        Guilherme.txt          sucesso
Desktop             help                   sucesso~
Documents           log completo.txt       Templates
Downloads           log completo.txt~      Templates
examples.desktop    loja                   Videos
```

```
falha                   mostra_idade              zip
falha~                  mostra_idade~
guilherme.silveira.txt  Music
```

Podemos verificar que o arquivo `guilherme.silveira.txt` está em nossa lista! Você pode estar pensando que na verdade isso que acabamos de criar não é uma extensão, são vários caracteres. Vamos testar novamente?

Vamos abrir o *gedit* mais uma vez e vamos criar outro arquivo que dessa vez chamaremos de `guilherme.jpg.txt` , com duas extensões e salvamos esse arquivo. Se dermos um novo `ls` veremos que esse arquivo estará em nossa lista:

Ou seja, podemos criar arquivos usando "." em seus nomes que não teremos problema nenhum. O caracter do ponto, na verdade, é um caracter qualquer no nome de um arquivo, só que, diferentemente de outros sistemas operacionais, em geral, o *Windows* interpreta o ponto como um delimitador entre um nome de arquivo e o seu sufixo, então, temos, por exemplo, no caso de "guilherme.txt" o nome "guilherme", um ponto, e o sufixo "txt". No *Linux* não temos essa questão, então, o nome do arquivo continua sendo tudo, "guilherme.silveira.txt", pois não existe esse conceito de extensão na hora de lidarmos com esses arquivos. Assim, para o sistema operacional e para o sistema de arquivos o nome de arquivos é "guilherme.silveira.txt", portanto, não existe o conceito de extensão nesse nível.

Então, se formos trabalhar, por exemplo, com a execução de um *script*, no caso, o `mostra_idade` , poderemos executá-lo. Mas, antes disso, vamos lembrar dele digitando `cat mostra_idade` e teremos o seguinte:

```
> cat mostra_idade
echo ${FULL_NAME} voce tem ${AGE} anos
job=Instruto
export ADDRESS='Rua Vergueiro, 3185'
```

Ele lê o `FULL_NAME` e o `AGE` , então, vamos executar o `FULL_NAME=Guilherme AGE=34` e vamos chamar o `mostra_idade` .

```
> FULL_NAME=Guilherme AGE=34 bash mostra_idade
Guilherme voce tem 34 anos.
```

E ele possui uma extensão para ser um programa? Não. Poderíamos recriá-lo com uma extensão `sh` de *shell*? O `sh` referindo-se a ser uma abreviação de *shell*? Podemos!

Então, vamos pegar esse arquivo e vamos move-lo arquivo. Para isso utilizaremos o `mv` para mover o `mostra_idade` para `mostra_idade.sh` :

```
> mv mostra_idade mostra_idade.sh
```

Com isso estamos, entre aspas, colocando uma "extensão". Por que entre aspas? Pois não existe o conceito de extensão nesse nível que estamos falando.

Vamos executar novamente o `mostra_idade.sh` :

```
> FULL_NAME=Guilherme AGE=34 bash mostra_idade.sh
Guilherme voce tem 34 anos.
```

Como podemos perceber, executamos e não tem problema, ele continua sendo um arquivo cujo nome é `mostra_idade.sh` . e se quisermos colocar outro extensão nesse arquivo? Podemos, basta, para fazer isso digitar `mv mostra_idade.jpg` . Mas, `jpg` ? Esse é apenas o nome do arquivo, `jpg` é seu nome, ele não possui extensão, a extensão não necessariamente fala o que esse arquivo é no *Linux*, diferente do que ocorre no sistema operacional do *Windows*. Então, se quisermos executar isso, repetimos o processo alterando o nome:

```
> mv mostra_idade mostra_idade.jpg
> FULL_NAME=Guilherme AGE=34 bash mostra_idade.jpg
Guilherme voce tem 34 anos.
```

Observe que ele executa! Vamos voltar atrás, vamos renomeá-lo de simplesmente `mostra_idade` . para fazermos isso digitamos `mv mostra_idade.jpg mostra_idade` uma vez que não temos a necessidade de indicar se ele é um `jpg` , um *script*, um *bash* ou um `txt` deixaremos apenas como `mostra_idade` no caso.

```
> mv mostra_idade.jpg mostra_idade
```

Repare que as "extensões" que criamos utilizando um ponto, por padrão, são usadas para sabermos o que aquele arquivo é. Se é uma imagem, um vídeo, um `txt` , um *script* e etc . Mas, essas "extensões" que criamos no nome do arquivo somos nós que inventamos porque queremos, ou seja, pois temos essa vontade e usamos elas, mas isso, no caso do *Linux*.

Vamos refletir, seria muito estranho salvar uma imagem nomeando ela de "minha foto", por isso, salvamos ela chamando-a de "mfoto.jpg", para que os programas meio que saibam que aquele arquivo é uma imagem, ainda que, ele só venha a saber que é uma imagem, apenas quando abra ela é. É nesse instante que se sabe que é uma imagem. O mesmo acontece com o *script*, pode ser que aquele arquivo seja um *script*, mas apenas ao abrir esse arquivo é que pode se ter certeza do que é. Só quando é aberto é que realmente sabemos qual a natureza do arquivo. Apenas pelo nome de um arquivo não temos como ter certeza do que é, de que aquele arquivo é de um determinado tipo. Isso ocorre pois, a extensão não faz parte do sistema operacional e mesmo que fizesse, ainda, poderíamos enganá-lo.

Repare que o ponto, portanto, é um carácter válido no nome dos arquivos, caminhos e diretórios e podemos usá-lo livremente, assim como, estamos utilizando até esse momento, sem nos preocuparmos com o conceito de "extensão". Mas, note que alguns programas podem nos restringir. Por exemplo, se queremos compilar um programa em "C", podemos usar o `gcc` . Vamos escrever `gcc` e dar um "Enter" nisso e visualizar o que ele nos traz:

```
> gcc
gcc: fatal error: no input files
compilation terminated
```

Ele fala que não tem nenhum arquivo de entrada, vamos, então, criar um arquivo bem simples de entrada. Abrimos, novamente o *gedit* e nele escrevemos:

```
#include <stdio.h>

int main()  {
        printf("Bem vindo");
        return 0;
}
```

E salvamos esse arquivo como "meu programa" e agora, voltando ao Terminal, digitamos `gcc meuprograma` e teremos o seguinte:

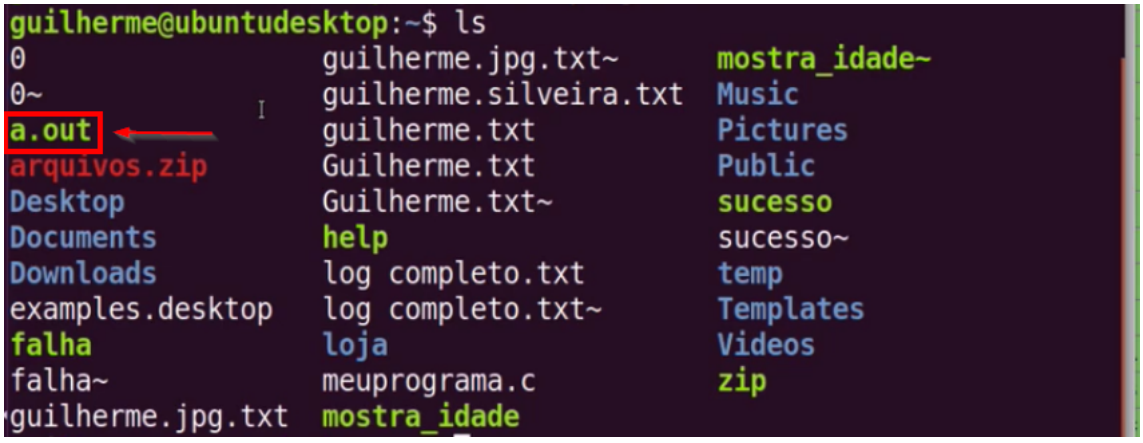
```
> gcc meuprograma
meuprograma: file not recognized: File format not recognized
collect2: error: ld returned 1 exit status
```

E quando digitamos isso para que ele rode algo no `gcc` ele simplesmente nos responde que não sabe do que estamos falando. E por que ele não sabe? O `gcc` recebe como argumento diversos tipos de arquivos e alguns destes arquivos são do tipo de texto, fonte, outros são do tipo objetos, já pré-compilados, o "1.o", por exemplo. Como ele não sabe se esse arquivo é um arquivo fonte ou uma biblioteca pré-compilada ou algo do gênero, ele não tem o que fazer. Mesmo que o sistema não nos obrigue a colocar extensões, um programa, ou seja, um outro nível, no caso, o programa `gcc` está nos obrigando a falar qual é o tipo de arquivo.

Por isso vamos dar um `mv` e renomear o arquivo:

```
> mv meu programa meuprograma.c
```

E agora sim, conseguimos rodar o `gcc meuprograma.c` . No nosso caso, por padrão, o `gcc` cria o arquivo `a.out` . Vejamos isso digitamos `ls` :



E podemos executar o `a.out` . Ao executarmos ele o que aparece é a mensagem de "Bem vindo":

```
> /home/guilherme/a.out
Bem vindo
```

Mas, nosso foco não é aprender programação "C" nesse curso. O que queríamos mostrar é que em outro nível, não no do sistema operacional, nem no dos arquivos, mas sim, no nível das aplicações que estão rodando é que temos exigências de especificação dos arquivos. Elas podem querer fazer algo com as extensões, detectar que tipos de arquivos são esses. Por isso

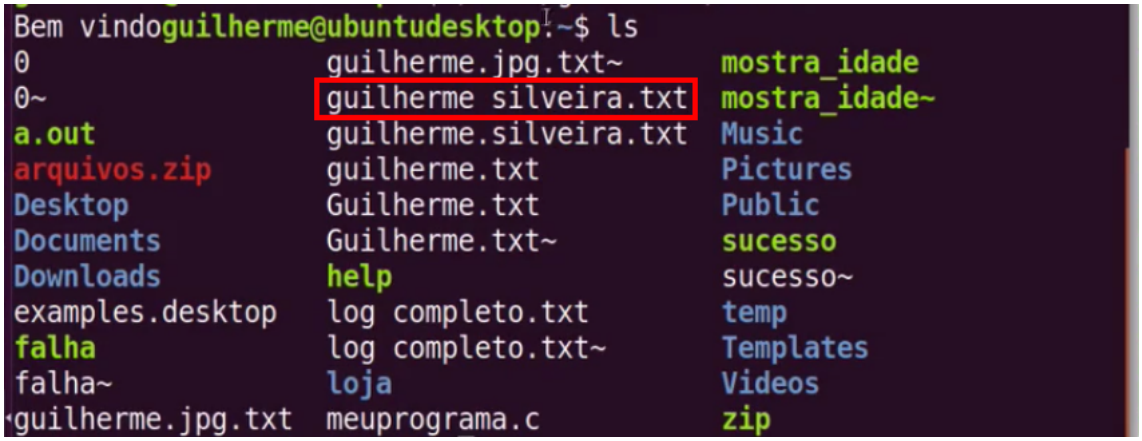
elas tem interesse que tenhamos uma extensão, um ponto, onde gostaria que disséssemos algo para essas aplicações. Nesse caso, o `a.out` diz para o `gcc` que esse arquivo é do tipo fonte.

Sabendo tudo isso sobre os arquivos, sobre o carácter ponto, sobre não usar a barra e que devemos evitar alguns caracteres que podem ocasionar em problemas futuros em outros sistemas operacionais.

E quais são os caracteres que realmente podemos usar de boa, sabendo que ele vai conversar bem com o *Windows*?

Na prática , podemos voltar a tabela do site para inclusive visualizar melhor, que podemos utilizar, mesmo, do "A" maiúsculo ao "Z" maiúsculo e do "a" minúsculo ao "z" minúsculo. Os números do "0" até o "9" e o ponto ".", o mais "+", o menos "-" e o *underline* "_". São esses os caracteres que, normalmente, utilizamos nos nomes de arquivos. E, espaço pode? Pode!

Vamos criar um arquivo que contenha um espaço no nome. Voltamos ao *gedit* e nomeamos esse arquivo de "guilherme silveira.txt" e salvamos esse arquivo. Se dermos um `ls` em nosso terminal veremos o arquivo:



Se quisermos ver o conteúdo desse arquivo? Podemos digitar `cat guilherme silveira.txt` , mas aqui teremos um problema, repare no que ele nos traz:

```
> cat guilherme silveira.txt
cat: guilherme: No such file or directory
cat: silveira.txt: No such file or directory
```

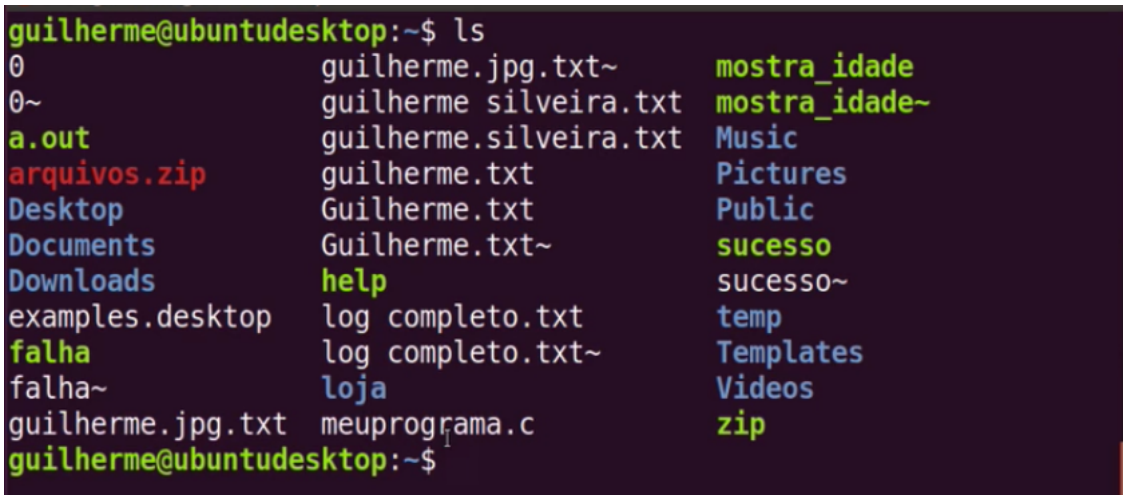
Lembra-se do *scape*? O espaço no *shell* é um carácter especial, então, temos que dar uma `\` para escapar. Teremos que digitar `cat guilherme\ silveira.txt` e como ele não possui conteúdo nenhum, não mostrará nada. Então, mesmo o *Linux* que suporta alguns caracteres especiais esses mesmos caracteres são especiais para o *shell*, então, teremos que ficar escapando disso.

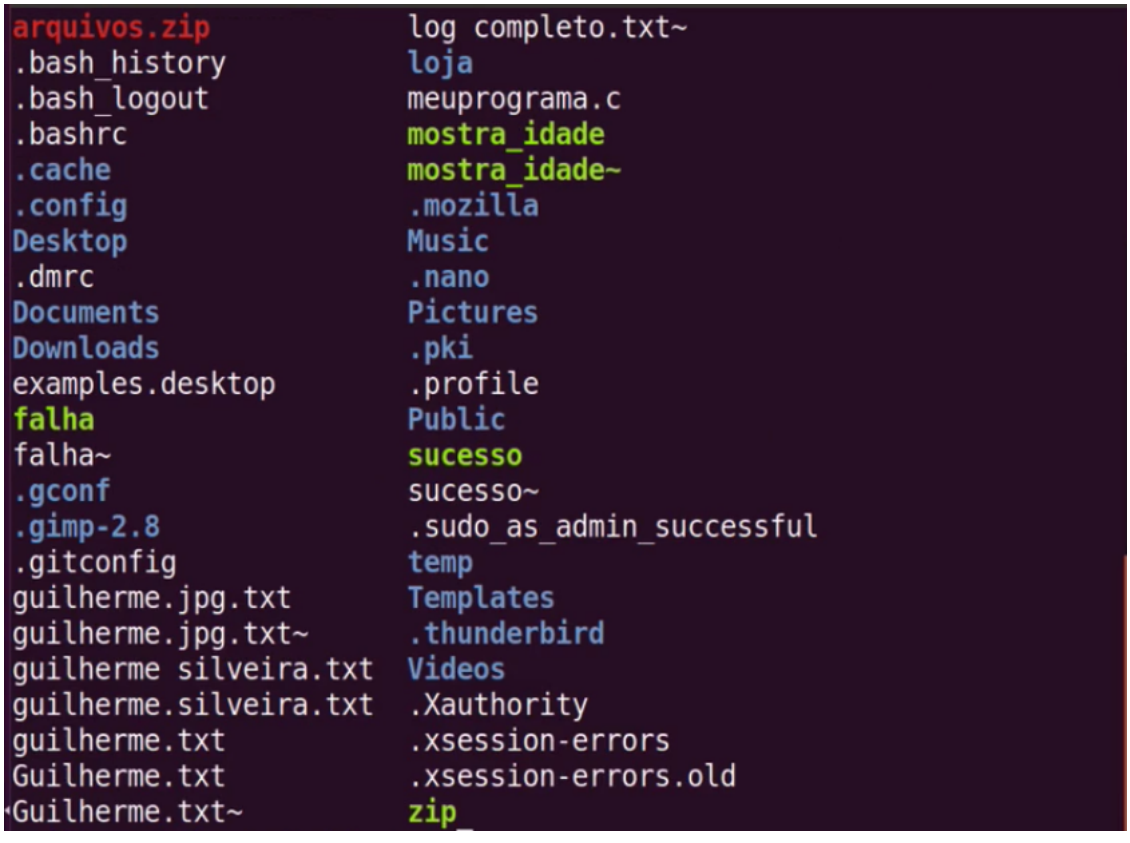
Claro que se for algum usuário final, por exemplo, sua prima que está salvando algum arquivo, ela não estará usando o terminal e isso não terá problema, mas se você estiver administrando as máquinas e criando programas que trabalham de um canto para outro você começa a correr certos riscos, a medida que utilizamos o terminal, de criar uma confusão. Então o espaço, o asterisco, os colchetes, o hífen, eles são todos caracteres especiais para o *shell*. O que faremos é evitar o uso dessas carcterísticas em nossos arquivos, então, em geral, utilizaremos do "A" maiúsculo ao "Z" maiúsculo, do "a" minusculo ao "z" minusculo, do "0" até o "9" e "+", "-", "." e "_". Esses caracteres, inclusive, acabam aparecendo bastante.

E o nome do arquivo pode ser de qualquer tamanho? O nome do arquivo não possui um tamanho limitado padrão, a maioria dos sistemas operacionais podem ter como limitação de 8 até 14 caracteres. Podemos criar o nome do arquivo com quantos caracteres quisermos, mas o que fazemos, na prática, é evitar criar um nome gigantesco pelo simples fato de ser difícil de trabalhar com ele. Por isso, evitamos nomes muito grandes. Tentamos usar nomes curtinhos e que além disso sejam, de preferência, claros.

Além dessa questão do tamanho do nome do arquivo, que podemos usar qualquer tamanho, temos que tomar um certo cuidado em relação a outro aspecto. Lembra-se que em geral, utilizaremos do "A" maiúsculo ao "Z" maiúsculo, do "a" minúsculo ao "z" minúsculo, do "0" até o "9" e "+", "-", "." e "*", são caracteres no nome do arquivo como inteiro. Agora, para evitar problemas, não usamos o "+", "-", "." e "* no começo de um arquivo. Fazemos isso para que, justamente, problemas com o *Windows* sejam evitados.

Usamos o "." no ínico do arquivo em um caso bastante específico, são os arquivos chamados de arquivos invisíveis. São aqueles arquivos que quando damos um `ls` não vemos nenhum arquivo começando com um ".". Usamos o "." no início do arquivo em um caso bastante específico, são os arquivos chamados de arquivos invisíveis. São aqueles arquivos que quando damos um `ls` não vemos nenhum arquivo começando com um ".". A primeira imagem abaixo é do `ls` e a segunda é do `ls -a`:





Temos, por exemplo, um arquivo chamado `.profile` e podemos ver esse arquivo digitando `cat .profile` e ele será mostrado para nós:

```
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_l
ogin
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

Assim, os arquivos que começam com "." são especiais, são arquivos denominados invisíveis. E esses arquivos podem ser criados, sem acarretarem em problemas, agora, arquivos que começam "+", "-" e "_" podem causar confusão em outros sistemas operacionais e por isso, evitaremos criar arquivos que começam com esses caracteres. Colocamos um "." só quando é para ser um arquivo invisível que, por padrão, apenas com o `ls` não aparece. Usaremos do "A" maiúsculo ao "Z" maiúsculo, do "a" minúsculo ao "z" minúsculo e do "0" até o "9" quando queremos que esse arquivo possam ser utilizados em outros sistemas operacionais. Nesse caso poderemos mandar o arquivo para outra pessoa e isso será aberto e funcionará perfeitamente, sem problemas.

Se utilizarmos o "-" para iniciar o nome de um arquivo pode ser que em outro sistema operacional ele não funcione. E se começarmos um arquivo com o "+" acontecerá a mesma coisa, assim como o "ç", que pode vir a funcionar ou não dependendo do *Unicode*. E, importante de lembrar, a "/" não poderá ser utilizado pois ela é um indicador de diretório que antecessor.

Repare, temos que tomar cuidado com uma série de questões, podemos utilizar "." em nomes de arquivos no início se queremos que ele seja invisível. Já, no *Windows*, é utilizado um atributo para indicar se ele visível ou não e no *Linux* o funcionamento é distinto, o nome do arquivo é que indica se ele é visível ou não. Simplesmente, não existe um atributo para indicar isso. O que existe é que, se o nome do arquivo que se começar com um ponto ele é invisível. Com isso, vimos diversas características de como criar um arquivo, com que nomes podemos criar. Podemos usar quase tudo, com exceção do código nulo , o "0", ou a barra simples, "/". Mas, por questão de compatibilidade entre vários *Linux*, evitamos usar coisas de fora da tabela *ASCII*, daqueles que comentamos, do "A" maiúsculo ao "Z" maiúsculo, do "a" minúsculo ao "z" minúsculo, do "0" até o "9" e o "+", "-", "_". Sendo que, para manter compatibilidade, o primeiro carácter só usaremos "A" maiúsculo ao "Z" maiúsculo, do "a" minúsculo ao "z" minúsculo e o "." para indicar o arquivo invisível e mesmo nesse último caso corremos o risco de um outro sistema não conseguir ler esse arquivo, por exemplo, pode acontecer que o *Windows* leia isso como uma extensão ou com um arquivo sem nome e bom, isso pode nos dar problemas.

Então, quando queremos trabalhar com arquivos que serão enviados para outros sistemas operacionais ou, que queremos que funcionem em outro sistema temos que tomar esses cuidados. E na prática, nós tomamos usando de "a" minúsculo a "z" minúsculo, de "A" maiúsculo a "Z" maiúsculo e de "0" até "9", no começo podemos usar o "." quando queremos que ele seja

invisível e no meio do nome de um arquivo podem aparecer "+", "-", "_" e ".". Portanto, quando queremos manter a compatibilidade do arquivo e estar enviando para outros sistemas usaremos esse padrão.