Laboratório 1.4: Sistema de Arquivos - Comandos, Links Físicos e Simbólicos

Introdução

Neste Laboratório, continuaremos a explorar o sistema de arquivos do Linux, mas focando em alguns comandos usados para encontrar arquivos e executáveis, ver o status do disco, trabalhar com links físicos e simbólicos e, por fim, entender melhor os diretórios "`.`" e "`..`".

1. Descobrindo a Localização dos Comandos

No final do laboratório anterior, mostramos o comando `who` que, normalmente, não fica no `/bin`. Para saber qual o caminho completo de um comando, use o comando `whereis`:

```
$ whereis who
```

```
/usr/bin/who /usr/share/man/man1/who.1.gz
```

Note que ele encontra outros arquivos (e.g., manuais) relacionados ao comando pesquisado.

Use o comando `whereis` e diga abaixo o caminho completo do comando `ls`.

```
ls: /usr/bin/ls /usr/share/man/man1/ls.1.gz
```

Qual o caminho completo do comando `hexdump`?

```
hexdump: /usr/bin/hexdump /usr/share/man/man1/hexdump.1.gz
```

Qual o caminho completo do comando `grep`?

```
grep: /usr/bin/grep /usr/share/man/man1/grep.1.gz
```

Qual o caminho completo do comando `ifconfig`?

```
ifconfig: /usr/sbin/ifconfig /usr/share/man/man8/ifconfig.8.gz
```

Qual o caminho completo do comando `whereis`?

```
whereis: /usr/bin/whereis /usr/share/man/man1/whereis.1.gz
```

2. Buscando Arquivos no Sistema

Existem algumas formas de se buscar um arquivo no computador. Provavelmente, usando um gerenciador de arquivos gráfico possa ser mais fácil. Entretanto, como estamos focando na linha de comando, usaremos o comando `find` para encontrar arquivos. Este comando possui várias opções, mas mostraremos a seguir as formas mais comuns dele.

Executado sem opções, o `find` lista todos os arquivos, incluindo de subdiretórios, a partir do caminho especificado. Por exemplo:

```
$ find /etc
```

O comando acima, irá listar todos os arquivos do diretório `/etc` e de todos os seus subdiretórios. É uma quantidade grande de arquivos, como você pode ver no comando abaixo:

```
$ find /etc | wc -l
```

Note que o comando anterior deve ter gerado algumas linhas de erros, pois alguns diretórios são inacessíveis por usuários normais. Vamos mudar o comando para ignorar esses erros:

```
$ find /etc 2> /dev/null | wc -l
```

```
1584
```

No comando anterior, estamos enviando a saída de erro do `find` para o `/dev/null`, efetivamente ignorando tais erros, e o restante (saída padrão) está sendo redirecionado para o comando `wc` que, com a opção `-l` conta a quantidade de linhas.

Continuando com o `find`, a principal opção dele é a `-name`, que filtra os arquivos por nome. Por exemplo, para mostrar todos os arquivos do diretório `/usr/include` cujo nome tenha a palavra `stdio`, execute:

```
$ find /usr/include -name *stdio*
```

```
/usr/include/stdio.h
/usr/include/stdio_ext.h
/usr/include/x86_64-linux-gnu/bits/stdio2.h
/usr/include/x86_64-linux-gnu/bits/stdio.h
/usr/include/x86_64-linux-gnu/bits/stdio_lim.h
```

Sendo um pouco mais prático (e sincero), é comum esquecermos todas as opções do `find` e simplesmente combinarmos ele com o comando `grep`:

```
$ find /usr/include | grep stdio
```

Note como a saída foi a mesma e, melhor, as palavras encontradas foram marcadas em vermelho. O que o comando anterior faz é listar todos os arquivos do `/usr/include` e usar o `grep` para filtar as linhas mostrando apenas as que queremos.

Se você quiser pesquisar uma palavra em todo o sistema, basta usar o `find` passando o diretório raiz (`/`) como parâmetro. Obviamente, o comando irá demorar um pouco, pois absolutamente todos os arquivos do sistema serão listados.

3. Espaço em Disco Consumido por um Diretório

O comando `df` (*disk free*), mostrado anteriormente, exibe o espaço total livre nas diversas partições do sistema. Mas, uma vez que identificamos que o sistema está ficando com pouco espaço, como descobrimos os diretórios que estão consumindo mais?

O comando `du` (*disk usage*), mostra o espaço consumido por diretório. Se você executá-lo sem opções, ele irá mostrar o consumo de todos os diretórios e subdiretórios a partir do atual, o que acaba não sendo tão útil. Portanto, a opção mais comum dele é a `-s` que sumariza o consumo total do diretório atual incluindo todos os subdiretórios:

```
$ cd /usr
$ du -s
```

```
4663340
```

Entretanto, como você deve ter visto, o resultado é em bytes, o que pode ser ruim de ser lido por humanos. A opção `-h` resolve isso:

```
$ du -sh
```

```
4.5G
```

Para ver o consumo por subdiretório, execute:

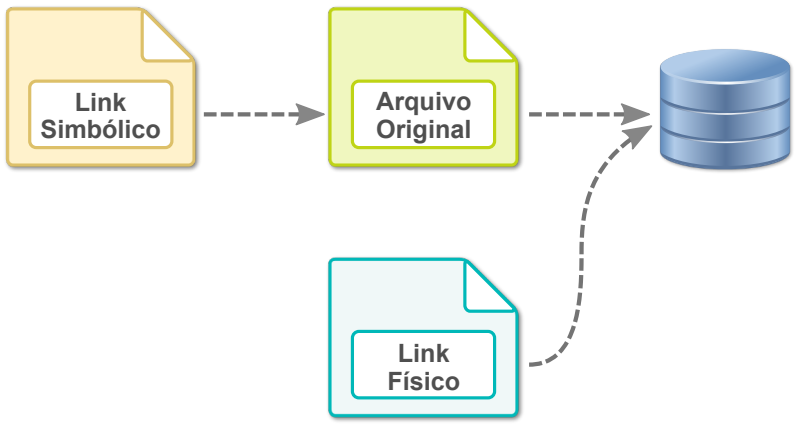
```
$ du -sh *
```

```
110M    bin
40K     games
12M     include
1.4G    lib
4.0K    lib32
4.0K    lib64
588K    libexec
4.0K    libx32
52K     local
21M     sbin
```

Você pode executar o comando anterior no diretório raiz para ver os diretórios que mais consomem espaço em disco em todo o sistema. Entretanto, esse comando deve demorar. Mas sinta-se livre para fazer isso! Não esqueça de redirecionar a saída de erro para o `/dev/null`, pois haverão erros de permissões de acesso a alguns diretórios.

4. Links Físicos e Simbólicos

Alguns arquivos são, na verdade, *links* (referências, ligações) para outros arquivos. Existem dois tipos de links: simbólicos (mais comuns) e físicos (do inglês, *hard links*). O link simbólico é apenas um redirecionamento para o nome original do arquivo. Já o link físico é um pouco mais "baixo nível". Ele aponta diretamente para os dados originais no disco. É como se tivéssemos dois arquivos apontando fisicamente para o mesmo conteúdo no disco. A figura a seguir ilustra esses dois tipos de links:



Estes links são criados através do comando `ln`. Para entender melhor, vamos ver um exemplo prático:

```
$ cd                                    # Volta para o diretório Home
(~)
$ echo "Mensagem no arquivo original" > ArquivoOriginal.txt
$ ls -lh ArquivoOriginal.txt
```

```
-rw-r--r-- 1 marcos marcos 29 May  6 23:22 ArquivoOriginal.txt
```

Link Simbólico

No comando acima, criamos um arquivo chamado `ArquivoOriginal.txt`. Vamos agora criar um link simbólico para ele:

```
$ ln -s ArquivoOriginal.txt ArquivoLinkSimbolico.txt
$ ls -lh ArquivoLinkSimbolico.txt
```

```
lrwxrwxrwx 1 marcos marcos 19 May  6 23:22
ArquivoLinkSimbolico.txt -> ArquivoOriginal.txt
```

Note como o comando `ls` indica que o arquivo `ArquivoLinkSimbolico.txt` é, na verdade, um link simbólico para `ArquivoOriginal.txt`. Veja o que acontece se pedirmos para ver o conteúdo do link simbólico:

```
$ cat ArquivoLinkSimbolico.txt
```

```
Mensagem no arquivo original
```

Como você pode ver o conteúdo é o mesmo. O que aconteceu é que o Linux "encaminha" a solicitação de acesso para o arquivo original, identificado no link. Portanto, tem-se apenas um único arquivo original ocupando espaço no disco. Você pode, inclusive, modificar o conteúdo do link simbólico e o arquivo original que será realmente modificado:

```
$ echo "Mensagem concatenada no link simbólico" >>
ArquivoLinkSimbolico.txt
$ cat ArquivoOriginal.txt
```

```
Mensagem no arquivo original
Mensagem concatenada no link simbólico
```

Veja agora o que acontece se você apagar o arquivo original:

```
$ rm ArquivoOriginal.txt
$ cat ArquivoLinkSimbolico.txt
```

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

Note que o próprio comando `ls` identifica que o link não é mais válido:

```
$ ls -lh ArquivoLinkSimbolico.txt
```

Link Físico - *Hard Link*

Conforme mencionado, fiferentemente do link simbólico que é apenas um redirecionamento para o nome original do arquivo, o link físico é um pouco mais "baixo nível". Ele aponta diretamente para os dados originais no disco. É como se tivéssemos dois arquivos apontando fisicamente para o mesmo conteúdo no disco. Vamos ver um exemplo:

```
$ echo "Mensagem no arquivo original 2" > ArquivoOriginal2.txt
$ ls -lh ArquivoOriginal2.txt
```

```
-rw-r--r-- 1 marcos marcos 31 May  6 23:23 ArquivoOriginal2.txt
```

Gostaria de chamar a atenção para o segundo campo retornado pelo comando `ls`, que atualmente contém o número `1` (um). Este campo indica a quantidade de links físicos para o arquivo. Como o mesmo acabou de ser criado, ele só possui um. Criaremos um link físico para o arquivo:

```
$ ln ArquivoOriginal2.txt ArquivoLinkFisico.txt
$ ls -lh ArquivoOriginal2.txt ArquivoLinkFisico.txt
```

```
-rw-r--r-- 2 marcos marcos 31 May  6 23:23 ArquivoLinkFisico.txt
-rw-r--r-- 2 marcos marcos 31 May  6 23:23 ArquivoOriginal2.txt
```

Note agora que o segundo campo do `ls` indica dois links físicos para o arquivo. Neste momento, não se tem nem mais como saber qual o arquivo original. Os dois são, basicamente, o mesmo arquivo. Veja o que acontece se mudarmos o conteúdo de um dos arquivos:

```
$ echo "Mensagem concatenada no link físico" >> ArquivoLinkFisico.txt
$ ls -lh ArquivoOriginal2.txt ArquivoLinkFisico.txt
```

```
-rw-r--r-- 2 marcos marcos 68 May  6 23:24 ArquivoLinkFisico.txt
-rw-r--r-- 2 marcos marcos 68 May  6 23:24 ArquivoOriginal2.txt
```

Note como os dois arquivos mudaram de tamanho e os dois arquivos tiveram a data de modificação atualizados. Entretanto, apesar de não parecer, o espaço total em disco usado pelos arquivos é apenas o tamanho de um dos dois, ou seja, o tamanho usado não é duplicado.

Por fim, veja o que acontece se apagarmos um dos arquivos:

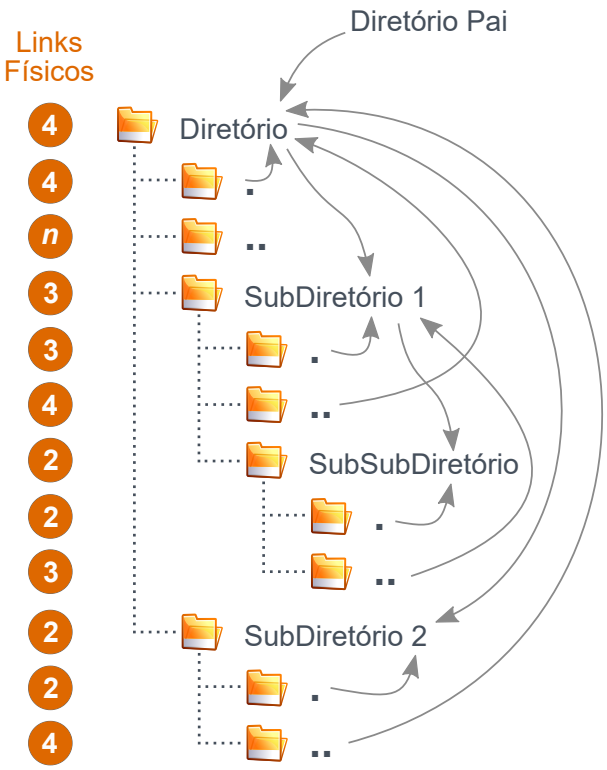
```
$ rm ArquivoOriginal2.txt
$ ls -lh ArquivoLinkFisico.txt
$ cat ArquivoLinkFisico.txt
```

```
-rw-rw-r-- 1 masp masp 105 May 10 00:19 ArquivoLinkFisico.txt
Mensagem no arquivo original 2
Mensagem concatenada no link físico
Mensagem concatenada no link físico
```

Note como a quantidade de links físicos no comando `ls` reduziu para 1 (um). Note também que o acesso aos dados através do segundo arquivo `ArquivoLinkFisico.txt` continua válido. Ou seja, um arquivo no Linux só é removido de verdade quando todos os links físicos para ele são removidos.

5. Os Diretórios "." e ".." são Links Físicos

Vamos aproveitar a oportunidade para voltar aos diretórios "`.`" e "`..`". Conforme mencionamos, eles apontam para o diretório atual e para o anterior, respectivamente. Podemos agora falar que eles são links físicos para os diretórios. A imagem a seguir ilustra os links físicos criados por estes dois diretórios especiais:



Para ver isso na prática, execute o comando `ls` abaixo e observe os links físicos dos diretórios:

```
$ ls -alh | grep "\." | head -2
```

```
drwxr-xr-x  3 masp masp 4.0K May  7 03:27 .
drwxr-xr-x 56 root root 4.0K May  4 23:46 ..
```

Note como eles possuem vários links físicos. Em geral, tem-se um link físico para cada subdiretório dentro do diretório. O motivo disso é que cada subdiretório terá, dentro dele, o link físico "`..`" apontando para o diretório pai. Veja o que acontece se criarmos um novo subdiretório:

```
$ mkdir NovoSubdiretorio
$ ls -alh | grep "\." | head -2
```

```
drwxr-xr-x  4 masp masp 4.0K May  7 03:27 .
drwxr-xr-x 56 root root 4.0K May  4 23:46 ..
```

Note como a quantidade de links físicos para o diretório "`.`" (atual) aumentou. Veja agora a quantidade de links físicos para o subdiretório recém-criado:

```
$ ls -lh | grep NovoSubdiretorio
```

```
drwxrwxr-x 2 masp masp 4.0K May  7 03:27 NovoSubdiretorio
```

Note que um diretório é criado já com dois links físicos: o do diretório pai e o do diretório "`.`" dentro dele.

Por fim, analise a quantidade de links físicos dos diretórios "`.`" e "`..`" dentro do `NovoSubdiretorio`:

```
$ ls -alh NovoSubdiretorio/
```

```
total 8.0K
drwxrwxr-x 2 masp masp 4.0K May  7 03:27 .
drwxr-xr-x 4 masp masp 4.0K May  7 03:27 ..
```

I have no special talent. I am only passionately curious. — Albert Einstein

π