Tutorial para uso de LaTeX

Rui Lopes

novembro de 2016

0.1 Introdução

Uma das tarefas mais importantes e valorizadas na realização de projetos e outras ações de devOps (ações de colaboração entre *developers* e gestores de operação) é a criação de pequenos **scripts** - pequenos excertos de código executável por um interpretador - numa linguagem própria - a **Bash**. Esta linguagem de programação é a nativa de grande parte dos terminais que existem nos sistemas operativos da família UNIX, tal como as distribuições Linux ou o sistema operativo *macOS*, da Apple.

Capítulo 1

Uma Primeira Aproximação à Linguagem Bash

Na sua forma mais simples, um *script* não é nada mais, nada menos, que um conjunto de comandos bash guardados num ficheiro de texto com capacidades executáveis. Consideremos, para este efeito, o seguinte *script*, que cria um diretório, um ficheiro dentro deste e mostra a árvore de diretórios desde a raiz.

```
1 mkdir directoryA
2 touch file1
3 mv file1 directoryA
4 tree
```

Ao longo deste documento iremos abordar mais detalhes tanto a nível de comandos da bash, como de *shell scripting*, como um conjunto de comandos bash ou outras opções de abordagem.

1.1 Primeiros comandos bash

Não tendo nunca trabalhado sobre uma consola bash, é importante ter em conta alguns dos seus comandos mais simples, fáceis e úteis para a manipulação de ficheiros. Falamos de **ficheiros** porque estes são a unidade básica de informação no que toca à manipulação desta por parte de uma consola bash e do sistema operativo UNIX. Assim sendo, uma das primeiras coisas que devemos aprender como fazer é criar, editar, copiar, mover e apagar um ficheiro. Mais, e como nem todos os ficheiros são iguais entre si, dentro desta família de sistemas operativos, também devemos conseguir efetuar estas mesmas operações sobre um tipo específico - o **diretório**.
Antes de avançarmos com estes comandos é importante referir como obter o estado atual (*feedback*) da própria consola bash. Para obtermos tal resposta devemos procurar pelo **prompt**. O *prompt* é um símbolo que pretende identificar o local e o tempo em que se pode escrever um comando da bash. Geralmente este identifica-se pelo carater $ - que iremos usar ao longo deste documento -, embora possa assumir diferentes formas, consoante a personalização que cada um faz da mesma, sobre uma variável de sistema (assunto o qual iremos debater mais à frente).

1.1.1 Criar um ficheiro - comando `touch`

Existem várias formas de criar um ficheiro. Entre encaminhar um conjunto de carateres para um local (que virá a produzir um ficheiro), como iremos abordar mais à frente, a criar um ficheiro por si, existe um variado leque de opções para a criação de ficheiros. Por agora, iremos abordar o uso do comando `touch` para esse efeito. Note-se, não obstante, que este comando apenas serve para criar ficheiros regulares, isto é, ficheiros que não correspondem a diretórios, atalhos ou blocos de

dados/carateres.
Consideremos que estamos num diretório vazio e pretendemos criar um ficheiro denominado `shakespeare.txt` vazio. Para o fazer precisamos então de executar o comando `touch` seguido do nomes do ficheiro que pretendemos criar. Veja-se o seguinte código:

```
touch shakespeare.txt
```

A execução deste comando ocorre sem erros quando consegue criar um ficheiro com o nome especificado no diretório atual. Caso ocorra algum erro ao longo do processo, o mais provável é não haver permissão para escrever no diretório atual. As permissões de ficheiros são um assunto a abordar mais à frente.

1.1.2 Criar um diretório - comando `mkdir`

Os diretórios são ficheiros do sistema operativo UNIX um pouco especiais. Anexos a estes estão conjuntos de outros ficheiros com vários tipos, entre os quais ficheiros regulares, outros diretórios, atalhos, entre outros... Assim sendo, é importante saber como criar este tipo de ficheiro. Para tal, existe o comando `mkdir` - proveniente de *make directory*.
Para o nosso exemplo de aplicação, tentemos assim criar uma pasta chamada `authors`, pelo que teremos de executar o comando `mkdir` juntamente do nome do diretório que pretendemos criar, como podemos ver de seguida:

```
mkdir authors
```

A execução deste comando ocorre sem erros quando consegue criar um diretório com o nome especificado no diretório atual. Caso ocorra algum erro ao longo do processo, o mais provável é não haver permissão para escrever no diretório atual. As permissões de ficheiros são um assunto a abordar mais à frente.

1.1.3 Copiar um ficheiro ou diretório - comando `cp`

Sabendo já como criar um ficheiro regular e um diretório, então é agora importante saber como copiar um ficheiro e, assim, criar um duplicado deste. Para este tipo de ação usamos o comando `cp`, que recebe, no mínimo, dois argumentos. Na verdade, o comando de cópia permite que se liste um conjunto de ficheiros para copiar, sendo que o último item listado é sempre o local e o nome de destino da cópia. Por outras palavras, podemos resumir a sintaxe do comando de cópia de duas formas:

```
cp ficheiro1 [ficheiro2] [...] [ficheiroN] diretório_de_destino
```

No caso acima temos que um conjunto de ficheiros são copiados para o diretório denominado de `diretório_de_destino`. Não obstante, podemos ter o seguinte caso:

```
cp ficheiro_de_origem ficheiro_de_destino
```

Neste caso temos então que o ficheiro com o nome `ficheiro_de_origem` será copiado, sendo criado um ficheiro equivalente, mas com o nome `ficheiro_de_destino`.
Testando os dois casos temos o seguinte:

```
cp shakespeare.txt dickens.txt
cp shakespeare.txt dickens.txt authors
```

No primeiro caso temos que o ficheiro `shakespeare.txt` será copiado para o ficheiro novo com o nome de `dickens.txt`. No segundo caso de aplicação, onde aplicamos o primeiro padrão

que vimos para este comando, temos que os ficheiros `shakespeare.txt` e `dickens.txt` serão ambos copiados para dois novos ficheiros com o mesmo nome, agora dentro do diretório já criado com o nome `authors`. No fim deste processo devemos ter algo como o seguinte:

```
authors
  | -- dickens.txt
  | -- shakespeare.txt
dickens.txt
shakespeare.txt
```

Continuando de subssecção em subssecção iremos modificar esta árvore de ficheiros.
A execução deste comando ocorre sem erros quando consegue copiar um ou mais ficheiros com o nome especificado (ou para o diretório especificado). Caso ocorra algum erro ao longo do processo, o mais provável é não haver permissão para escrever no diretório atual. As permissões de ficheiros são um assunto a abordar mais à frente. Note-se também que este comando não permite a cópia de diretórios sem a utilização de uma opção especial (a não ser que estes estejam vazios). Para copiar um diretório com conteúdo deverá executar o comando `cp` com a opção `-r` de "recursivo": este método permite que o sistema operativo possa consultar todos os ficheiros que residem dentro desse diretório e que os possa copiar um a um, mantendo a mesma organização.

1.1.4 Editar um ficheiro através de um editor de texto

Por esta altura já sabemos criar e copiar ficheiros e diretórios. Mas os ficheiros que temos até agora são meros ficheiros de texto que estão vazios e não possuem qualquer tipo de significado, a não ser o título que lhes demos. Para lhes acrescentar algum conteúdo necessitamos daquilo a que se chama um **editor de texto**. Não havendo a possibilidade de integrar processadores de texto como o Microsoft Word ou um Pages numa consola UNIX-*like*, temos então que arranjar uma ferramenta que nos forneça uma forma fácil e rápida de podermos modificar os nossos ficheiros.
Um exemplo de editor de texto mais convencional é o `nano`. Este editor de texto é muito simples e usa uma interface controlável através de combinações de teclas com a tecla Ctrl. Para abrir o ficheiro `shakespeare.txt` com o nano apenas temos de executar o seguinte:

```
nano shakespeare.txt
```

Neste documento não iremos abordar como trabalhar com o `nano`, mas antes com um editor muito mais completo, de seu nome `vim`. Para abrir ficheiros com o `vim` o processo é precisamente o mesmo, mas a interação com a própria interface deste editor de texto não é propriamente trivial, pelo que merece destaque numa das unidades deste documento.
Consideremos então que já temos algum conteúdo instalado dentro de cada um dos ficheiros com extensão `txt` - poderá inserir texto arbitrário dentro destes ficheiros, se assim o entender.

1.1.5 Mover um ficheiro ou diretório - comando `mv`

Olhando para a nossa árvore atual de ficheiros podíamos organizar melhor a forma como estruturamos os ficheiros e diretórios. Sabemos que ambos Shakespeare e Dickens são autores, mas até seria interessante criar pastas com os nomes deles e nelas colocar obras e informações acerca dos mesmos. Para isso necessitamos de saber criar diretórios, ficheiros e movê-los, tal como mudar o nome deles, eventualmente. Estas duas últimas tarefas (mover e mudar o nome) são consideradas objeto do comando `mv` que recebe dois ou mais argumentos, de forma algo semelhante ao que acontece com o comando `cp`.

```
mv ficheiro1 [ficheiro2] [...] [ficheiroN] diretório_de_destino
```

No caso acima temos que um conjunto de ficheiros são enviados para o diretório denominado de `diretório_de_destino`. Não obstante, podemos ter o seguinte caso:

```
cp ficheiro_de_origem local_de_destino
```

Neste caso temos então que o ficheiro com o nome `ficheiro_de_origem` será movidopara o local com o nome `ficheiro_de_destino`.
Não obstante, este comando não permite apenas a movimentação de ficheiros dentro do disco, mas também a mudança de nome dos mesmos, isto é, sendo que o comando, tal como foi expresso acima, também poderá ser interpretado da seguinte forma:

```
cp nome_de_ficheiro_original nome_de_ficheiro_novo
```

Testando os dois casos temos o seguinte:

```
mkdir authors/dickens
mkdir authors/shakespeare
mv authors/dickens.txt authors/dickens/
mv authors/shakespeare.txt authors/shakespeare
mv authors/dickens/dickens.txt authors/dickens/biography.txt
mv authors/shakespeare/shakespeare.txt authors/shakespeare/biography.
    txt
```

No *script* acima temos então que primeiro criamos pastas com o nome de ambos os autores Dickens e Shakespeare dentro do diretório `authors`. Tendo sido criados tais diretórios, então agora é a vez de movermos os ficheiros `txt` que residem na pasta `authors` para dentro do diretório do autor respetivo. No fim, alteramos os nomes desses ficheiros para `biography.txt`. A estrutura final de ficheiros deverá ser algo do género da representação abaixo:

```
 authors
   | -- dickens
   |      | -- biography.txt
   | -- shakespeare
   |      | -- biography.txt
 dickens.txt
 shakespeare.txt
```

A execução deste comando ocorre sem erros quando consegue mover um ou mais ficheiros com o nome especificado (ou para o diretório especificado). Caso ocorra algum erro ao longo do processo, o mais provável é não haver permissão para escrever no diretório indicado. As permissões de ficheiros são um assunto a abordar mais à frente.

1.1.6 Eliminar um ficheiro - comando `rm`

Claro está que, de acordo com a última organização que aplicámos na nossa estrutura arbórea de ficheiros, temos dois ficheiros `txt` que estão a mais na raiz - o ficheiro `dickens.txt` e o ficheiro `shakespeare.txt`. Para isso precisamos de saber como os eliminar.
Assim sendo, por outras palavras, necessitamos de executar o comando `rm` de forma a consolidar esta nossa pretensão. Este comando é muito simples, recebendo apenas como argumento a lista de ficheiros que se pretendem eliminar. Como tal, para eliminar os ficheiros `txt` no diretório raiz (da nossa árvore, não do sistema operativo), apenas precisamos de executar o seguinte comando:

```
rm dickens.txt shakespeare.txt
```

Através do uso de **expressões regulares** - assunto o qual iremos abordar com mais detalhe mais à frente - podemos inclusive diminuir o detalhe explícito do comando, referindo que pretendemos eliminar, no diretório atual, todos os ficheiros que possuam um conjunto de carateres quaisquer ao início, mas que terminem com `.txt`, o que se faz representar pela expressão regular `*.txt`.

Em algumas consolas, inclusive, ao escrever tal expressão regular seguida da tecla TAB, é listada automaticamente todos os ficheiros a que tal comando corresponde, mas de modo explícito, sendo que no nosso caso tal deverá ser exatamente igual ao comando anterior.

```
rm *.txt
```

Caso o utilizador que execute este comando não tenha permissões para ler ou escrever sobre este ficheiro, então a sua eliminação não será possível. Um outro erro frequente e possível é a tentativa de eliminação de um diretório, o qual só poderá ser feito, sem opções extra, com o comando explicado de seguida.

1.1.7 Eliminar um diretório - comando `rmdir`

Na nossa estrutura de árvore de ficheiros não temos nenhuma pasta que pretendamos eliminar, mas podemos considerar que criamos uma chamada `temp` e que a queremos eliminar agora. Para o fazer, e dado que estamos perante um novo tipo de ficheiros, será necessário invocar o comando `rmdir`. Vejamos o seguinte exemplo:

```
mkdir temp
rmdir temp
```

No código acima criamos a pasta `temp` para a podermos apagar com o comando `rmdir`. Note-se, não obstante, que a pasta que criámos foi apagada enquanto vazia, sendo que não é possível apagar diretórios não vazios com este comando. Para tal, e como estamos a referir mais do que um tipo de ficheiros no sistema operativo UNIX, então devemos usar o comando `rm` para apagar diretórios com conteúdo, através de uma sua opção recursiva e forçosa - mais propriamente a `-r` e a `-f`. Considerando novamente o diretório `temp` criado (e agora com ficheiros nele incluídos), para o apagar, necessitamos de executar o seguinte comando:

```
mkdir temp
touch fileA fileB fileC fileD
mv file* temp
rm -rf temp
```

Possíveis erros que possam ser criados poderão ocorrer simplesmente à conta de permissões que o utilizador atual não possui.

1.1.8 Ver o conteúdo de um ficheiro - comando `cat`

Tendo editado anteriormente pelo menos um dos ficheiros de biografia dos autores, é-nos possível visualizar o seu conteúdo através do comando `cat`. O nome deste comando não provém da palavra *cat* em inglês (gato, em português), mas antes de uma abreviatura para *concatenation* (em português, concatenação), que significa junção de várias partes (anexação). Isto tem este nome porque ao executar este comando iremos enviar para o nosso terminal um determinado ficheiro carater a carater, concatenados.
Assim sendo, para executarmos a leitura do ficheiro de biografia de Dickens, podemos executar o seguinte comando:

```
cat authors/dickens/biography.txt
```

O resutado da execução acima é a impressão de todos os seus carateres no próprio terminal. Mais à frente iremos ver como contornar este possível obstáculo de imprimir exclusivamente para o terminal, através de outras ferramentas intrínsecas à *bash*.

```
Charles John Huffam Dickens (7 February 1812 - 9 June 1870) was an English writer and
    social critic. He created some of the world's best-known fictional characters and
    has his regards by many as the greatest novelist of the Victorian era. His works
    enjoyed unprecedented popularity during his lifetime, and by the twentieth century
    critics and scholars had recognised him as a literary genius. His novels and short
    stories enjoy lasting popularity...
```

A execução deste comando poderá suscitar erros a nível de permissão, caso o utilizador não possua direitos de leitura deste.

1.1.9 Listar o conteúdo de um diretório - comando `ls` e `tree`

Até agora temos vindo sempre a apresentar uma tarefa com um ficheiro regular e a mesma com diretórios. Assim, e dado que anteriormente falámos de como ver o conteúdo de ficheiros de texto, agora podemos investigar como listar o conteúdo de um diretório.
Para esta função precisamos de conhecer o comando `ls`, de *list*. Executando este comando dentro de um diretório `dir` é então possível ver todos os ficheiros que nele residem, entre ficheiros regulares, diretórios e outros. Se executarmos o comando `ls` dentro do nosso diretório raiz podemos ver o seguinte:

```
$ ls
authors
```

Nas consolas *bash* os comandos não são necessariamente tão lineares como temos vindo a apresentar, sendo que eles possuem (se assim pretendermos ou precisarmos), opções de execução. Dentro do comando `ls` existem três opções que nos serão relevantes, entre elas a `-a`, `-l` e a `-r`.
A opção `-a` permite que se vejam todos (*all*) os ficheiros que estão contidos no diretório atual. Isto por vezes torna-se necessário porque, por convenção, existem ficheiros ocultos cujo nome começam por um ponto `.`. Para vermos todos os ficheiros usamos então a opção `-a`, executando `ls -a`.
Para listarmos os vários ficheiros e alguns detalhes acerca dos mesmos podemos executar o comando com a opção `-l`, o que revela todo um conjunto de informações acerca dos ficheiros. Mais precisamente, são apresentadas as seguintes informações pela ordem precisa: permissões, número de ligações físicas (*hard links* - abordaremos este assunto mais à frente), dono do ficheiro, grupo dono do ficheiro, tamanho do ficheiro, data da última modificação e nome. Vejamos um exemplo:

```
$ ls -l
drwxr-xr-x   2 apontamentos  staff    68B Jan 1 12:00 authors
```

1.1.10 Ver informações acerca de comandos - comando `man`

1.1.11 Identificar um comando por um propósito - comando `apropos`

1.2 Sha-Bang, para não haver dúvidas

1.3 Introdução à *Bash Scripting*

1.3.1 Criação de ficheiros de scripting (extensão `sh`)

1.3.2 Portabilidade de scripts e Permissões

Capítulo 2

Sessões em Consola (*Time-Sharing*) e Processos

2.1 Enquadramento histórico da consola UNIX

2.2 Manipulação de Processos e Envio de Sinais

2.2.1 Conceito de processo

2.2.2 Visualização de processos - comando `ps`

2.2.3 Hierarquia de processos

2.2.4 Conceito de sinal

2.2.5 Envio de sinais a identificadores de procesos - comando `kill`

2.2.6 Envio de sinais a processos nomeados - comando `killall`

2.2.7 Ciclo de vida dos processos em UNIX

2.2.8 Redirecionamento de saída de processos para entrada de outros - *pipe*

2.2.9 Conceito de *threads* e sua manipulação

2.2.10 Interfaces de gestão de processos na máquina - comando `htop`

Capítulo 3

Manipulação de Dados e Informação em Bash

3.1 Filtragem de Informação em *streams* de Dados

3.1.1 Utilização do *pipe* para filtragem de dados

3.1.2 Expressões Regulares como filtro para dados - comando `grep` e `egrep`

3.1.3 Corte de dados para seleção de informação a apresentar - comando `cut`

3.2 Manipulação direta de dados com indexação direta

3.2.1 Utilização da ferramenta `awk`

Sintaxe básica do `awk`

Ciclo de vida da ferramenta

Manipulação de variáveis de sistema

Operadores e Arrays

Controlo de fluxo e ciclos

Criação e utlilzação de funções

Redirecionamento e melhoramento de saídas de dados

3.2.2 Manipulação de conjuntos de dados com a ferramenta `sed`

Sintaxe básica do `sed`

Ciclo de vida da ferramenta

Manipulação de ciclos e *branches*

***Buffer* e âmbito de padrões**

Carateres especiais

Receitas úteis

Capítulo 4

Editar Ficheiros de Texto com `vim`

Conteúdo

Acrónimos