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 enviados 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`.

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

Conteúdo

Acrónimos