Instituto Superior de Engenharia

Politécnico de Coimbra

Sistemas Operativos

CTeSP Tecnologias e Programação de Sistemas de Informação (Cantanhede)

Professor: João Leal

joao.leal@isec.pt

UNIgreen The green European University

PRR Plano de Recuperação e Resiliência

REPÚBLICA PORTUGUESA

Financiado pela União Europeia NextGenerationEU

Linha de comando (*Shell*)

- Existem vários *shells* diferentes no Linux; estes são apenas alguns exemplos:
  - Bourne-again shell (Bash)
  - C shell (csh ou tcsh, a versão aprimorada do csh)
  - Korn shell (ksh)
  - Z shell (zsh)

- No Linux, o mais comum é o ***shell Bash***.

Prompt

- Ao usar um shell interativo, o utilizador insere comandos no chamado prompt.

- Para cada distribuição Linux, o prompt padrão tem uma aparência um pouco diferente, mas ele geralmente segue esta estrutura:

```
username@hostname diretório_atual tipo_de_shell
```

Prompt

- No Ubuntu ou Debian GNU/Linux, o prompt de um utilizador comum provavelmente será assim:

```
joao@mycomputer:~$
```

- O prompt do superutilizador terá a seguinte aparência:

```
root@mycomputer:~#
```

Prompt

- No CentOS ou Red Hat Linux, o prompt para um utilizador comum será semelhante a este:

```
[joao@mycomputer ~]$
```

- E o prompt do superutilizador será assim:

```
[root@mycomputer ~]#
```

Prompt

- **`username`**

  Nome do utilizador que está a utilizar o shell

- **`hostname`**

  Nome da máquina "hospedeira" na qual o shell é executado. Também existe um comando hostname, que permite exibir ou definir o nome de "hospedeiro" do sistema.

Prompt

- **`current_directory`**

  O diretório em que o shell está atualmente. Um ~ indica que o shell está no diretório Home do utilizador atual.

- **`shell_type`**
  - *$ indica que o shell está a ser executado por um utilizador comum.* **(Vamos utilizar o $ como prompt)**
  - *# indica que o shell está a ser executado pelo super utilizador root.*

Estrutura da linha de comando

- A maioria dos comandos na linha de comando segue a mesma estrutura básica:

```
comando [opção(ões)/parâmetro(s)...] [argumento(s)...]
```

Estrutura da linha de comando

- *Exemplo:*

```
$ ls -l /home
```

- **Comando**

  Programa que o utilizador pretende executar `ls`, no exemplo acima.

Nota: A maioria dos comandos exibe uma visão geral dos comandos disponíveis quando executados com o parâmetro --help.

Estrutura da linha de comando


```
$ ls -l /home
```

- **Opção(ões)/Parâmetro(s)**

  Um "botão" que modifica o comportamento do comando de alguma forma, como `-l`. Podemos ter acesso às opções na forma curta ou longa. Por exemplo, `-l` é idêntico a `--format = long`. Também é possível combinar múltiplas opções e, no caso do formato abreviado, as letras geralmente podem ser digitadas juntas. Por exemplo,:

```
$ ls -al
$ ls -a -l
$ ls --all --format=long
```

Estrutura da linha de comando


```
$ ls -l /home
```

- **Argumento(s)**

  Dados adicionais exigidos pelo programa, como um nome de arquivo ou caminho, como `/home` no exemplo acima.

Nota: A única parte obrigatória desta estrutura é o próprio comando. Em geral, todos os outros elementos são opcionais, mas um programa pode exigir que determinadas opções, parâmetros ou argumentos sejam especificados.

Tipos de comportamento dos comandos

- O shell aceita **dois tipos de comandos**: Internos e Externos.

- **Internos:** Comandos que fazem parte do próprio shell e não são programas separados. Existem cerca de 30 desses comandos. O principal objetivo é **executar tarefas dentro do shell** (por `exemplo, cd, set, export`).

Tipos de comportamento dos comandos

- **Externos: Comandos em arquivos individuais**, que geralmente são programas binários ou scripts. Quando um comando externo ao shell é executado, o shell usa a variável PATH para procurar um arquivo executável com o mesmo nome que o comando. Os utilizadores também podem criar os seus próprios comandos externos.

Tipos de comportamento dos comandos

- O comando type mostra a que tipo pertence um comando específico:

$ type echo

echo is a shell builtin

$ type man

man is /usr/bin/man

Citação

- Os shells oferecem um recurso chamado citação, que encapsula dados (por exemplo, precisamos usar espaços, caracteres especiais e variáveis) usando diversos tipos de aspas (" ", ' '). No Bash, há três tipos de citações:
  - Aspas duplas
  - Aspas simples
  - Caracteres de escape

Aspas Duplas

- As aspas duplas dizem ao shell para considerar o texto entre as aspas ("...") como caracteres regulares.

- Todos os caracteres especiais perdem o significado, exceto **$ (cifrão), \ (barra invertida)** e **` (crase)**.

- Significa que as variáveis, substituições de comando e funções aritméticas ainda podem ser usadas.

Aspas Duplas

- Por exemplo, a substituição da variável $USER não é afetada pelas aspas duplas:

$ echo Eu sou $USER

Eu sou joao

$ echo “Eu sou $USER"

Eu sou joao

Aspas Duplas

- Um caractere de espaço, por outro lado, perde seu significado como separador de argumentos:

```
$ touch new file
$ ls -l
-rw-rw-r-- 1 joao students 0 May 8 21:18 file
-rw-rw-r-- 1 joao students 0 May 8 21:18 new
$ touch "new file"
$ ls -l
-rw-rw-r-- 1 joao students 0 May 8 21:19 new file
```

Aspas Duplas

- O comando **touch** cria dois arquivos individuais; o comando interpreta as duas sequências como argumentos individuais.

- No segundo exemplo, o comando interpreta as duas sequências como um argumento e, portanto, cria apenas um arquivo.

- No entanto, recomenda-se evitar o caractere de espaço nos nomes de arquivos. Em vez disso, é preferível usar um **sublinhado (_) ou um ponto (.)**.

Aspas Simples

- As aspas simples não têm as exceções das aspas duplas. Elas revogam qualquer significado especial de cada caractere.

```
$ echo Eu sou $USER
```

Eu sou joao

- Ao aplicar aspas simples, obtemos um resultado diferente:

```
$ echo ‘Eu sou $USER’
```

Eu sou $USER

- O comando agora exibe a sequência exata, sem substituir a variável.

Caracteres de *escape*

- Podemos usar caracteres de escape para remover os significados especiais dos caracteres do Bash.

```
$ echo $USER
```

Joao

- Vemos que, por padrão, o conteúdo da variável é exibido no terminal. No entanto, se colocarmos um caractere de barra invertida (\) antes do cifrão, o significado especial do cifrão será cancelado.

Caracteres de *escape*

- Por sua vez, isso não permitirá que o Bash expanda o valor da variável para o nome de utilizador da pessoa que está a executar o comando, mas, em vez disso, interpretará o nome da variável literalmente:

```
$ echo \$USER
```

$USER

- O funcionamento do caractere de escape é diferente, pois **instrui o Bash a ignorar qualquer significado especial do caractere que ele precede**.

Exercício 1

- Divida as linhas abaixo nos componentes de comando, opção(ões)/parâmetro(s) e argumento(s):

  - `cat -n /etc/passwd`
  - `ls -l /etc`
  - `ls -l -a`
  - `cd /home/user`

Exercício 1

- `cat -n /etc/passwd`

Comando	`cat`
Opção	`-n`
Argumento	`/etc/passwd`

(...)

Exercício 2

- Indique de que tipo são os seguintes comandos:
  - `pwd`
  - `mv`
  - `cd`
  - `cat`
  - `exit`

Exercício 2

- **`pwd`**

  Embutido no Shell

(...)

Exercício 3

- Resolva os seguintes comandos que usam citações:
  - `echo "$HOME é a minha pasta principal"`

    `echo /home/user é a minha pasta principal`
  - `touch "$USER“`
  - `touch 'touch'`

Variáveis

- As variáveis são espaços de armazenamento para dados, como texto ou números. Uma vez definido, podemos aceder ao valor de uma variável posteriormente.

- As variáveis têm um nome, o que permite aceder a uma variável específica mesmo quando seu conteúdo é alterado.

- Na maioria dos shells do Linux, existem dois tipos de variáveis: Variáveis locais e Variáveis de ambiente.

Variáveis locais

- Variáveis que estão disponíveis apenas para o processo atual do shell.

- Se criarmos uma variável local e, em seguida, iniciarmos outro programa nessa shell, não poderemos aceder mais a essa variável nesse programa.

- Como **não são herdadas por subprocessos**, essas variáveis são chamadas variáveis locais.

Variáveis de ambiente

- Estão disponíveis quer numa sessão de shell específica quer em subprocessos gerados a partir dessa sessão de shell.

- Podem ser usadas para transmitir dados de configuração para os comandos executados.

- Como os programas podem aceder a essas variáveis, elas são chamadas variáveis de ambiente.

Variáveis de ambiente

- A maioria das variáveis de ambiente aparece em letras maiúsculas (por exemplo, PATH, DATE, USER).

- Um conjunto de variáveis de ambiente padrão fornece, por exemplo, informações sobre o diretório inicial ou o tipo de terminal do utilizador.

- Em certos casos, referimo-nos ao conjunto completo de todas as variáveis de ambiente como ambiente.

Manipulação de Variáveis

- Como administrador do sistema, vamos necessitar de criar, modificar ou remover variáveis locais e de ambiente.

Trabalhar com variáveis locais

- Podemos configurar uma variável local usando o operador **= (igual)**. Uma atribuição simples criará uma variável local:

```
$ greeting=ola
```

NOTA: Não coloque espaços antes ou depois do operador =.

- É possível exibir qualquer variável usando o comando **echo**. O comando geralmente exibe o texto na seção de argumentos:

```
$ echo ola
ola
```

Trabalhar com variáveis locais

- Podemos configurar uma variável local usando o operador **=** **(igual)**. Uma atribuição simples criará uma variável local:

```
$ greeting=ola
```

NOTA: Não coloque espaços antes ou depois do operador =.

- É possível exibir qualquer variável usando o comando **echo**. O comando geralmente exibe o texto na seção de argumentos:

```
$ echo ola
```

ola

Trabalhar com variáveis locais

- Para aceder ao valor da variável, teremos que usar **$ (cifrão)** na frente do nome da variável.

```
$ echo $greeting
```

ola

- A variável foi criada. Agora abrindo outro shell se tentarmos exibir o conteúdo da variável criada.

```
$ echo $greeting
```

- Nada é exibido.As variáveis existem apenas num shell específico.

Trabalhar com variáveis locais

- Para verificar se a variável é de facto uma variável local, tentemos gerar um novo processo e vamos conferir se esse processo é capaz de aceder à variável. Iniciamos outro shell e executamos o comando **echo** (o novo shell abre num novo processo, ele não herdará as variáveis locais do processo pai):

```
$ echo $greeting mundo
```

ola mundo

```
$ bash -c 'echo $greeting mundo'
```

mundo

Nota: Lembre-se de usar aspas simples no exemplo.

Trabalhar com variáveis locais

- Para remover uma variável, usamos o comando **unset**:

```
$ echo $greeting
```

ola

```
$ unset greeting
$ echo $greeting
```

Nota: O unset requer o nome da variável como argumento. Portanto, não é possível adicionar $ ao nome, já que isso resolveria a variável e passaria o valor da variável para unset em vez do nome da variável.

Trabalhar com variáveis globais

- Para disponibilizar uma variável local para subprocessos, podemos transformá-la em variável de ambiente. Usamos para isso o comando **export**. Quando ele é invocado junto ao nome da variável, essa variável é adicionada ao ambiente do shell:

```
$ greeting=ola
$ export greeting
```

Nota: Aqui também não se deve usar $ ao executar export, já que queremos transmitir o nome da variável e não seu conteúdo.

Trabalhar com variáveis globais

- Uma maneira mais fácil de criar a variável de ambiente é combinar os dois métodos anteriores, atribuindo o valor da variável na parte de argumento do comando.

```
$ export greeting=ola
```

Trabalhar com variáveis globais

- Vamos conferir novamente se a variável está acessível aos subprocessos:

```
$ export greeting=ola
$ echo $greeting mundo
```

ola mundo

```
$ bash -c 'echo $greeting mundo'
```

ola mundo

Trabalhar com variáveis globais

- Outra maneira de usar variáveis de ambiente é colocá-las na frente dos comandos. (Vamos testar essa possibilidade com a variável de ambiente TZ, que contém o fuso horário - variável usada pelo comando date para determinar qual hora do fuso horário)

```
$ TZ=EST date
```

Fri 23 May 10:07:35 EST 2025

```
$ TZ=GMT date
```

Fri 23 May 10:07:35 GMT 2025

Nota: Para exibir todas as variáveis de ambiente, use o comando env.

A variável PATH

- A variável PATH é uma das variáveis de ambiente mais importantes de um sistema Linux.

- Armazena uma lista de diretórios, separados por dois pontos, que contêm programas executáveis que funcionam como comandos do shell do Linux.

```
$ echo $PATH
```

/home/user/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games

A variável PATH

- Para acrescentar um novo diretório à variável, usamos o sinal de dois pontos (:).

```
$ PATH=$PATH:new_directory
```

Exemplo:

```
$ echo $PATH
```

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

```
$ PATH=$PATH:/home/user/bin
$ echo $PATH
```

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/user/bin

A variável PATH

- Como vimos, $PATH é usado no novo valor atribuído a PATH. Essa variável é resolvida durante a execução do comando e garante que o conteúdo original da variável seja preservado. Também podemos usar outras variáveis na atribuição:

```
$ mybin=/opt/bin
$ PATH=$PATH:$mybin
$ echo $PATH
```

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/user/bin:/opt/bin

A variável PATH

- A variável PATH deve ser usada com cuidado, pois é crucial para o trabalho na linha de comando.

- Vamos considerar a seguinte variável PATH:

```
$ echo $PATH
```

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

A variável PATH

- Para descobrir como o shell chama um comando específico, **which** pode ser executado com o nome do comando como argumento. Podemos, por exemplo, tentar descobrir onde o nano está armazenado:

```
$ which nano
```

/usr/bin/nano

A variável PATH

- Neste exemplo, o executável nano está localizado no diretório /usr/bin. Vamos remover o diretório da variável e verificar se o comando ainda funciona:

```
$PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games
$ echo $PATH
```

/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games

- Vamos procurar pelo comando nano novamente:

```
$ which nano
```

which: no nano in (/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games)

A variável PATH

- O comando não foi encontrado, e portanto não foi executado. A mensagem de erro também explica o motivo pelo qual o comando não foi encontrado e em quais locais foi procurado. Vamos readicionar os diretórios e executar o comando novamente.

```
$ PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
$ which nano
```

/usr/bin/nano

Nota: agora nosso comando voltou a funcionar.

Exercícios

4. Crie uma variável local number.

```
$ number=10
```

5. Crie uma variável de ambiente ORDER usando um dos dois métodos anteriores.

```
$ export ORDER=desc
```

6. Exiba o nome das variáveis e seu conteúdo.

```
$ echo number

(…)
```

Exercícios

7. Crie uma variável local **nr_files** e atribua o número de linhas encontrado no arquivo **/etc/passwd**. **Dica: pesquise sobre o comando wc e substituição de comandos, e não se esqueça das aspas.**

8. Crie uma variável de ambiente ME. Atribua a ela o valor de variável USER.

9. Inclua o valor da variável HOME a ME, usando o delimitador :. Exiba o conteúdo da variável ME.

Exercícios

10. Usando o exemplo anterior, crie uma variável chamada today e atribua a data de um dos fusos horários.

11. Crie outra variável chamada today1 e atribua a ela a data do sistema


Questions?