



Introdução ao Ambiente *Linux*

Objetivos:

- Introdução ao *Linux*
- Interpretador de comandos
- Sistema de ficheiros
- Processos
- Utilizadores

2.1 Introdução

Quando o sistema *UNIX* foi concebido, os computadores eram controlados essencialmente através de *consolas* ou *terminais* de texto: dispositivos dotados de um teclado e de um ecrã onde se podia visualizar somente texto. A interação com o sistema fazia-se através da introdução de comandos escritos no teclado e da observação da resposta produzida no ecrã pelos programas executados.

Atualmente existem ambientes gráficos que correm sobre o *UNIX/Linux* e que permitem visualizar informação de texto e gráfica, e interagir por manipulação virtual de objetos gráficos recorrendo a um rato e ao teclado. É o caso do Sistema de Janelas **X**, ou simplesmente **X**¹, que está incluído em todas as distribuições usuais de *Linux*.

Apesar das novas formas de interação proporcionadas pelos ambientes gráficos, continua a ser possível e em certos casos preferível usar a interface de *linha de comandos* para muitas operações. No **X**, isto pode fazer-se usando um *emulador de terminal*, um programa que abre uma janela onde se podem introduzir comandos linha-a-linha e observar as respostas geradas tal como num terminal de texto à moda antiga.

¹<http://www.x.org>

2.1.1 Interfaces de texto e gráficas

Os sistemas *Linux* geralmente simulam um ambiente de trabalho formado por 6 interfaces de texto (consolas) e 2 interfaces gráficas. Na prática todas estas interfaces coexistem sobre os mesmos equipamentos de interface com os utilizadores: ecrã, teclado, rato, etc. Muito embora estas interfaces estejam sempre ativas, elas não estão sempre visíveis: só uma delas está visível em cada instante.

Na maior parte dos casos os sistemas *Linux* apresentam uma interface gráfica quando iniciam, ou após a terminação da sessão de um utilizador. A comutação entre interfaces faz-se através das seguintes combinações de teclas:

interface gráfica /consola \rightarrow **consola** : Ctrl + Alt + F_{*n*}, onde F_{*n*} é uma das teclas F1, F2, ..., F6. O valor de *n* indica o número da consola que se pretende usar (F1 para a consola 1, etc.).

consola \rightarrow **interface gráfica** : Alt + F7 (para a interface gráfica por omissão) ou Alt + F8 (para a interface gráfica secundária, normalmente inativa).

Após o arranque do *Linux*, cada consola apresenta uma interface muito simples de *login*, na qual são apresentadas algumas características do sistema (sistema operativo, nome da máquina, nome da interface (tty_{*n*}), e o que se pretende que o utilizador introduza: o *nome-de-utilizador* e a *senha*. Após um *login* bem sucedido, o utilizador pode continuar a trabalhar na linha de comandos que lhe é apresentada. Para terminar a sua sessão, o utilizador deverá fazer *logout* usando o comando **exit**. Após este comando, a consola voltará a apresentar a interface de *login* antes observada.

Exercício 2.1

Mude para uma consola de texto, faça *login* na mesma, execute o comando **whoami**, e execute em seguida o comando **exit**. Repita o processo de *login* e faça desta vez *logout* carregando na combinação de teclas Ctrl + D. Este conjunto de teclas é designado por **EOF** (*End-Of-File*).

Uma das consolas terá o ambiente gráfico ao qual pode voltar.

2.1.2 Interpretador de comandos

Sendo o *UNIX* um sistema operativo com apenas interface textual (Command Line Interface (CLI)) onde o utilizador escreve e executa comandos, o utilizador tem que conhecer os comandos do sistema operativo e para interagir com ele precisa de uma ferramenta de comunicação. Esta ferramenta é a *shell*.

A *shell* é um ambiente que permite ao utilizador interagir com o núcleo do sistema (*Kernel*) e desta forma aceder aos recursos do computador. Mas a *shell* além de ser um interpretador de comandos, também é uma linguagem de programação, com instruções condicionais e repetitivas e variáveis.

Com estas instruções, usando comandos *UNIX*, a comunicação entre processos (*pipe*) e os redirecionamentos de entrada e de saída de dados, é possível criar programas (*shell scripts*) de administração de sistema para automatizar tarefas, que é muito utilizado por programadores experientes e administradores de sistemas *UNIX*.

A primeira *shell* do *UNIX* (**sh**) foi inicialmente criada por Thompson e depois desenvolvida por Stephen Bourne, nos laboratórios AT&T. Ela é normalmente conhecida por Bourne shell e foi lançado em 1977 com o *UNIX* Versão 7. A *C shell* (**cs****h**) foi desenvolvido por Bill Joy na Universidade de Berkeley e é a *shell* mais utilizada nos sistemas BSD. Deriva originalmente da sexta edição do *UNIX*, ou seja da Thompson *shell*. Esta *shell* caracteriza-se por permitir uma maior interação do utilizador com o sistema, providenciando mecanismos que facilitam a trabalho por vezes repetitivo de desenvolvimento de software, tais como o mecanismo de história e o controlo de tarefas.

A Korn *shell* (**ksh**) foi desenvolvido por David Korn nos laboratórios AT&T. Ela fundiu o que havia de melhor nas *shells* já existentes, tendo no entanto mantido a compatibilidade com a Bourne *shell* e acrescentando ao mecanismo de história dois editores internos para corrigir comandos digitados anteriormente. Atualmente a *shell* mais usada, nomeadamente no sistema operativo *Linux*, é a *Bourne-Again Shell* (**bash**) que é uma evolução retro-compatível muito mais interativa da Bourne *shell*.

Portanto a maioria das distribuições *Linux* fornece uma CLI através do programa **bash**. Como não poderia deixar de ser, existem alternativas, tais como: **ksh**, **tcsh** (*shell* baseada e compatível com a **cs****h**), **zsh**, **dash** (*Debian Almquist shell*) ou **fish**. Todas elas fornecem uma CLI ao utilizador, mas possuem funcionalidades ligeiramente distintas.

Quando se utiliza um ambiente gráfico existem programas denominados por *emuladores de terminal*, responsáveis por na realidade apresentarem uma janela de interação com o utilizador, enviando teclas para a *Shell* e apresentando o conteúdo ao utilizador.

Existem vários emuladores de terminal, sendo que as distribuições fornecem sempre vários para escolha, tais como: **xterm**, **konsole**, **gnome-terminal** ou **lxterminal**.

Exercício 2.2

Usando o mecanismo de procura ou os menus do ambiente gráfico, procure um emulador de terminal e execute-o.

Procure um dos outros emuladores e execute-o.

Verifique que embora o texto apresentado seja semelhante, os emuladores apresentam aspectos ligeiramente diferentes e têm menus e funcionalidades de configuração diferentes. Descubra como pode mudar a cor de fundo e o tamanho de letra em cada um dos emuladores.

Execute o comando `echo $$SHELL` para verificar qual a *Shell* em utilização em cada terminal.

Ao executar o emulador de terminal, pode reparar que o interface é bastante simples, sempre apresentada apenas uma pequena informação tal representado na Figura 2.1.

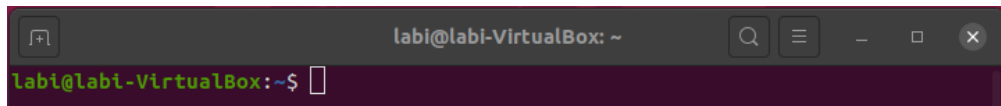


Figura 2.1: *Prompt* apresentado pela **bash**.

O formato utilizado neste *Prompt* é o de **utilizador @ nome da maquina : directorio actual \$**. Ou seja, neste caso em particular, utilizador **labi**, máquina **labi**, directório actual **~**. Mais à frente, na Secção 2.2 iremos ver o que representa o **~**.

Se escrever qualquer coisa aleatória, por exemplo **xpto**, seguido de **ENTER**, verá que a **bash** lhe indica uma situação de erro, tornando a pedir um comando válido.

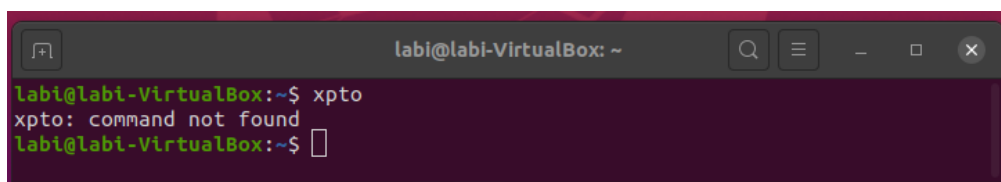


Figura 2.2: Indicação de erro pela **bash**.

Como pode notar, por muitos comandos errados que introduza, a **bash** irá sempre pedir de novo um comando, sem que isso traga algum prejuízo para o sistema.

Observe também que a resposta foi impressa imediatamente a seguir à linha do comando, de forma concisa, sem distrações nem grandes explicações. Este comportamento é usual em muitos comandos *UNIX* e é típico de um certo estilo defendido pelos criadores deste sistema. Simples, mas eficaz.

Exercício 2.3

Execute o comando **date** e observe o resultado.

Exercício 2.4

Execute o comando **cal** e observe o resultado. Descubra em que dia da semana nasceu, passando o mês e o ano como *argumentos* ao comando **cal**, por exemplo: **cal jan 1981**.

Formato dos comandos

Os comandos em *UNIX* têm sempre a forma:

```
comando argumento1 argumento2 ...
```

onde **comando** é o nome do programa a executar e os argumentos são cadeias de caracteres, que podem ser incluídas ou não, de acordo com a sintaxe esperada por esse programa.

Os argumentos podem ainda modificar o comportamento base do programa, designando-se nesse caso por opções (ou, por vezes, por *flags* em inglês, por pretenderem sinalizar algo de especial). Tipicamente as opções são indicadas de duas formas:

-x , onde *x* representa uma letra, maiúscula ou minúscula, ou um algarismo.

--nome-da-opção , onde *nome-da-opção* é um bloco de texto, sem espaços em branco, com a designação da opção.

Exercício 2.5

Execute o comando **date -u** para ver a hora atual no fuso horário UTC e observe o resultado. Execute o comando **date --utc** para obter o mesmo resultado.

Na linha de comandos é possível recuperar um comando indicado anteriormente usando as teclas de direção \uparrow e \downarrow . É possível depois editá-lo (alterá-lo) para produzir um novo comando (com argumentos diferentes, por exemplo). Outra funcionalidade muito útil é a possibilidade de o sistema completar automaticamente comandos ou argumentos parcialmente escritos usando a tecla **Tab**.

Um interpretador de comandos também mantém uma lista numerada com todos os comandos executados durante a sua execução. Esta lista pode ser consultada através do comando **history**. O resultado produzido por este comando é uma listagem, numerada, dos comandos executados pela ordem cronológica da sua execução. Os interpretadores de comandos permitem usar os números usados nessa listagem para recuperar (e executar novamente) os respetivos comandos, através do comando **!*n***, onde *n* é o número de ordem do comando que se pretende recuperar. A recuperação e execução rápida do comando exatamente anterior pode ser feita com o comando **!!**.

Exercício 2.6

Execute o comando **history** e observe o resultado.

Exercício 2.7

Execute o comando **!!** e observe o resultado. E se executar **!2**?

Exercício 2.8

Recupere um comando anterior usando as teclas de direção, edite-o e execute-o.

Exercício 2.9

Obtenha o número de ordem de um comando anterior e re-execute-o usando o comando que permite a indexação de um comando anterior (**!numero**).

Uma linha pode ser editada de forma elementar ou sofisticada. A forma elementar consiste em deslocar o cursor ao longo da linha, para trás ou para a frente, usando as teclas de direção \leftarrow e \rightarrow , adicionar novo texto à linha e apagar texto da linha, atrás do cursor, com a tecla **DELETE**.

Ajuda dos comandos

O que um comando faz, bem como a sintaxe e significado dos seus argumentos são decididos pelos seus programadores. Um utilizador médio tem de aprender a sintaxe e semântica de uma dezena ou duas de comandos e algumas das suas opções mais usuais, mas ninguém consegue memorizar os milhares de comandos disponíveis. Para uniformizar um pouco a sintaxe dos comandos e assim facilitar a aprendizagem, os programas seguem algumas convenções básicas:

1. Muitos programas aceitam opções de funcionamento quer no formato longo (`date --utc`), quer no formato curto, de uma letra só (`date -u`).
2. Várias opções no formato curto podem geralmente ser amalgamadas, e.g., `ls -l -a` equivale a `ls -la`.
3. Quase todos os comandos aceitam uma opção `--help`, por exemplo `date --help`, que serve apenas para mostrar um breve texto de ajuda.
4. Existe uma base de dados com manuais de utilização para todos os comandos disponíveis, acessível através do comando **man** seguido do nome do comando que se pretende consultar (e.g., `man date`).

Exercício 2.10

1. Considerando o comando **date** que vimos anteriormente, verifique o resultado de `date -u` e `date -utc`.
2. Aceda à ajuda do comando **date** através dos métodos descritos.
3. Aplique este método a outros comandos tais como **echo**, **true**, **false**.

Exercício 2.11

Verifique a utilidade do comando **apropos**. Pode usar como guia o parâmetro **successfully** e correlacionar o seu resultado com o comando **man**.

2.2 Sistema de Ficheiros

Os ficheiros e directórios são organizados seguindo o princípio de uma árvore com uma raíz (/) e directórios por baixo dessa raíz. Isto é semelhante ao utilizado no sistema operativo *Windows*, com a grande diferença que, enquanto o *Windows* considera que cada dispositivo (ex, Disco Rígido, Compact Disk (CD)) é uma unidade distinta, em *Linux* tal como na maioria dos restantes sistemas operativos, os diversos dispositivos encontram-se mapeados em directórios da mesma raíz. A Figura 2.3 mostra a estrutura do sistema de ficheiros típica de um qualquer sistema operativo *Linux*.

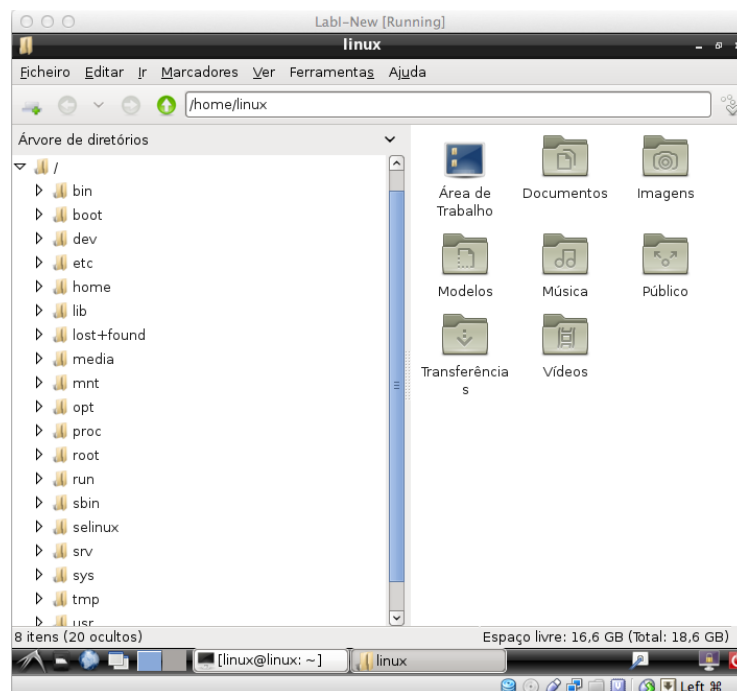


Figura 2.3: Estrutura do sistema de ficheiros *Linux*

Embora o *Linux* não obrigue a existência de uma estrutura específica, tipicamente o mesmo modelo é sempre utilizado. A Tabela 2.1 descreve o propósito de alguns destes directórios.

Cada utilizador possui um directório próprio nesta árvore, a partir do qual pode (e deve) criar e gerir toda a sua sub-árvore de directórios e ficheiros: é o chamado *directório do utilizador* ou *home directory*. Após a operação de *login* o sistema coloca-se nesse directório. Portanto neste momento deve ser esse o *directório atual* (*current directory*). Tipicamente este directório é representado pelo caractere `~`.

Tabela 2.1: Principais directórios típicos do *Linux*

Directório	Descrição
/	Raiz do sistema de ficheiros
/bin	Executáveis essenciais do sistema.
/boot	Contem o <i>kernel</i> para iniciar o sistema.
/etc	Configurações.
/home	Áreas dos utilizadores.
/mnt	Pontos de montagem temporários.
/media	Pontos de montagem de unidades como os CDs.
/lib	Bibliotecas essenciais e módulos do <i>kernel</i> .
/usr	Bibliotecas e aplicações tipicamente usadas por utilizadores.
/sbin	Programas tipicamente utilizados pelo super-utilizador.
/tmp	Ficheiros temporários.
/var	Ficheiros frequentemente modificados (bases de dados, impressões).

Para saber qual é o directório atual execute o comando **pwd**². Deve surgir um nome como indicado na Figura 2.4, que indica que está no directório **labi** que é um subdirectório de **home** que é um subdirectório direto da raiz **/**. Para listar o conteúdo do directório atual execute o comando **ls**³. Deve ver uma lista dos ficheiros (e subdirectórios) contidos no seu directório neste momento, tal como referido na Figura 2.4.

```

labi@labi-VirtualBox: ~$ pwd
/home/labi
labi@labi-VirtualBox:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
labi@labi-VirtualBox:~$

```

Figura 2.4: Resultado dos comandos **pwd** e **ls**

Dependendo da configuração do sistema, os nomes nesta listagem poderão aparecer com cores diferentes e/ou com uns caracteres especiais (**/**, **@**, *****) no final, que servem para indicar o tipo de ficheiro mas de facto não fazem parte do seu nome.

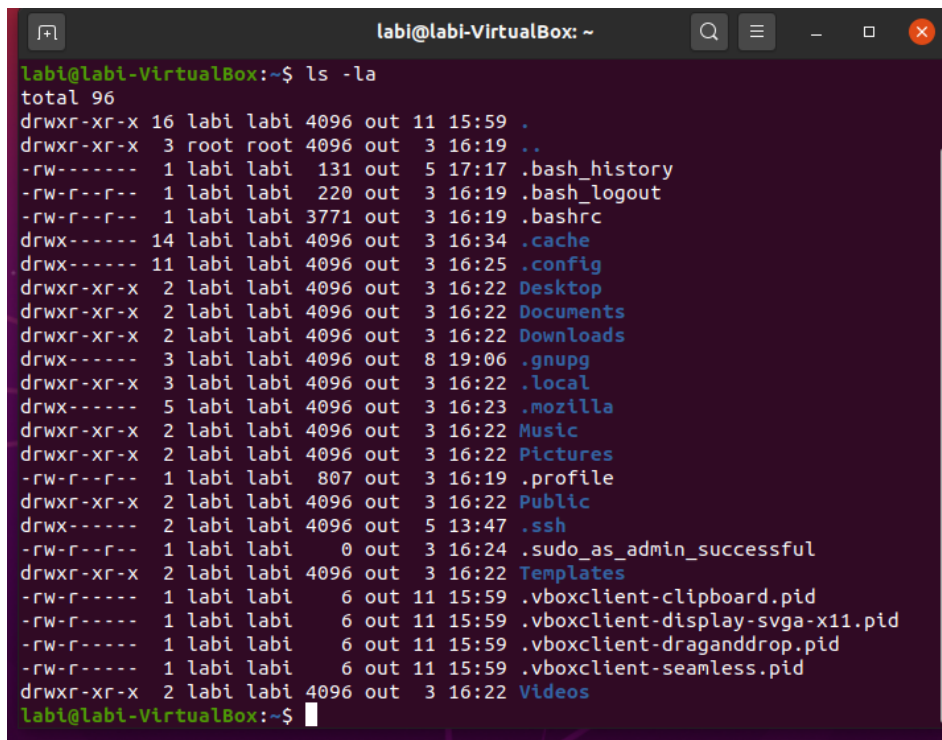
(Num ambiente gráfico a mesma informação está disponível numa representação mais visual. Se abrir a aplicação *Files* que aparece na barra vertical do lado esquerdo pode ver o conteúdo do seu directório pessoal *Home*.)

²Acrónimo de *print working directory*.

³Abreviatura de *list*.

Ficheiros cujos nomes começam por “.” não são listados por omissão, são ficheiros *escondidos*, usados geralmente para guardar informações de configuração de diversos programas. Para listar todos os ficheiros de um diretório, incluindo os escondidos, deve executar a variante `ls -a`⁴

Por vezes é necessário listar alguns atributos dos ficheiros para além do nome. Pode fazê-lo executando as variantes `ls -l` ou `ls -la`, sendo que o resultado deverá ser semelhante ao apresentado na Figura 2.5.



```
labi@labi-VirtualBox: ~  
labi@labi-VirtualBox:~$ ls -la  
total 96  
drwxr-xr-x 16 labi labi 4096 out 11 15:59 .  
drwxr-xr-x  3 root root 4096 out  3 16:19 ..  
-rw-r----- 1 labi labi  131 out  5 17:17 .bash_history  
-rw-r--r--  1 labi labi  220 out  3 16:19 .bash_logout  
-rw-r--r--  1 labi labi 3771 out  3 16:19 .bashrc  
drwx----- 14 labi labi 4096 out  3 16:34 .cache  
drwx----- 11 labi labi 4096 out  3 16:25 .config  
drwxr-xr-x  2 labi labi 4096 out  3 16:22 Desktop  
drwxr-xr-x  2 labi labi 4096 out  3 16:22 Documents  
drwxr-xr-x  2 labi labi 4096 out  3 16:22 Downloads  
drwx-----  3 labi labi 4096 out  8 19:06 .gnupg  
drwxr-xr-x  3 labi labi 4096 out  3 16:22 .local  
drwx-----  5 labi labi 4096 out  3 16:23 .mozilla  
drwxr-xr-x  2 labi labi 4096 out  3 16:22 Music  
drwxr-xr-x  2 labi labi 4096 out  3 16:22 Pictures  
-rw-r--r--  1 labi labi  807 out  3 16:19 .profile  
drwxr-xr-x  2 labi labi 4096 out  3 16:22 Public  
drwx-----  2 labi labi 4096 out  5 13:47 .ssh  
-rw-r--r--  1 labi labi    0 out  3 16:24 .sudo_as_admin_successful  
drwxr-xr-x  2 labi labi 4096 out  3 16:22 Templates  
-rw-r-----  1 labi labi    6 out 11 15:59 .vboxclient-clipboard.pid  
-rw-r-----  1 labi labi    6 out 11 15:59 .vboxclient-display-svgx11.pid  
-rw-r-----  1 labi labi    6 out 11 15:59 .vboxclient-draganddrop.pid  
-rw-r-----  1 labi labi    6 out 11 15:59 .vboxclient-seamless.pid  
drwxr-xr-x  2 labi labi 4096 out  3 16:22 Videos  
labi@labi-VirtualBox:~$
```

Figura 2.5: Listagem completa dos ficheiro na área pessoal

⁴A opção `-a` indica que se devem listar todos (*all*) os elementos do diretório.

Os principais atributos mostrados nestas listagens longas são:

Tipo de ficheiro identificado pelo primeiro carácter à esquerda, sendo **d** para diretório, - para ficheiro normal, **l** para *soft link*, etc.

Permissões representadas por 3 conjuntos de 3 caracteres. Indicam as permissões de leitura **r**, escrita **w** e execução/pesquisa **x** relativamente ao dono do ficheiro, aos outros elementos do mesmo grupo e aos restantes utilizadores da máquina.

Propriedade indica a que utilizador e a que grupo pertence o ficheiro.

Tamanho em número de bytes.

Data e hora da última modificação.

Nome do ficheiro.

Normalmente existe um *alias*⁵ **ll** equivalente ao comando **ls -l**.

Além do **ls** e variantes, existem outros comandos importantes para a observação e manipulação de diretórios, por exemplo:

cd — o diretório atual passa a ser o diretório do utilizador.

cd nome-do-dir — o diretório atual passa a ser o diretório **dir**.

mkdir nome-do-dir — cria um novo diretório chamado **dir**.

rmdir nome-do-dir — remove o diretório **dir**, desde que esteja vazio.

O argumento **dir** pode ser dado de uma forma absoluta ou relativa. Na forma absoluta, **dir** identifica o caminho (*path*) para o diretório pretendido a partir da raiz de todo o sistema de ficheiros; tem a forma **/subdir1/.../subdirN**. Na forma relativa, **dir** indica o caminho para o diretório pretendido a partir do diretório atual; tem a forma **subdir1/.../subdirN**.

Há dois nomes especiais para diretórios: “.” e “..” que representam respetivamente o diretório atual e o diretório pai, ou seja, o diretório ao qual o atual pertence.

⁵Um *alias* é um nome alternativo usado em representação de um determinado comando. São criados usando o comando interno **alias**.

Exercício 2.12

Execute os comandos seguintes e interprete os resultados:

```
cd /  
pwd  
cd /usr  
cd ~  
pwd  
cd /usr/sbin  
pwd  
cd -  
pwd
```

Exercício 2.13

Experimente utilizar o programa gráfico gestor de ficheiros para navegar pelos mesmos diretórios que no exercício anterior: `/`, `/usr`, `/usr/local/src`, etc.

Importante: Nos computadores das salas de aulas da UA, o subdiretório **arca** não é um diretório local do Computador Pessoal (PC) onde está a trabalhar; é na verdade a sua área privada de armazenamento no Arquivo Central de Dados (ARCA⁶), um servidor de ficheiros da Universidade de Aveiro. Esta área também é acessível a partir do ambiente Windows e através da Web. É neste diretório que deve gravar os ficheiros e diretórios que criar no decurso das aulas práticas. Os computadores das salas de aulas foram programados para apagarem o diretório de utilizador (e.g. `/homermt/a1245/`) sempre que são reiniciados. Só o conteúdo do subdiretório **arca** é salvaguardado. É portanto aí que deve colocar todo o seu trabalho. **Caso esteja a trabalhar num PC da sala de aulas faça os dois exercícios seguintes.**

Exercício 2.14

Experimente mudar o diretório atual para o seu subdiretório **arca**. Liste o seu conteúdo. Reconhece algum dos ficheiros?

Exercício 2.15

Crie, no diretório **arca**, um subdiretório chamado **labi** e, dentro desse, um diretório chamado **aula01**. Guarde neste diretório este guião.

⁶<https://arcaweb.ua.pt>

Manipulação de ficheiros

O *Linux* dispõe de diversos comandos de manipulação de ficheiros. Eis alguns:

cat *fic* — imprime no dispositivo de saída *standard* (por omissão o ecrã) o conteúdo do ficheiro ***fic***. O nome deste comando é uma abreviatura de *concatenate*, porque permite concatenar o conteúdo de vários ficheiros num só.

rm *fic* — remove (apaga) o ficheiro ***fic***.

mv *fic1 fic2* — muda o nome do ficheiro ***fic1*** para ***fic2***.

mv *fic dir* — move o ficheiro ***fic*** para dentro do directório ***dir***.

cp *fic1 fic2* — cria uma cópia do ficheiro ***fic1*** chamada ***fic2***.

cp *fic dir* — cria uma cópia do ficheiro ***fic*** dentro do directório ***dir***.

head *fic* — mostra as primeiras linhas do ficheiro (de texto) ***fic***.

tail *fic* — mostra as últimas linhas do ficheiro (de texto) ***fic***.

more *fic* — imprime no dispositivo de saída padrão (por omissão o ecrã), página a página, o conteúdo do ficheiro ***fic***.

less *fic* — comando similar ao ***more***, mas que permite uma navegação mais sofisticada para trás e para diante nas linhas apresentadas.

grep *padrão fic* — selecciona as linhas do ficheiro (de texto) ***fic*** que satisfazem o critério de seleção ***padrão***.

wc *fic* — conta o número de linhas, palavras e caracteres do ficheiro ***fic***. O nome deste comando é um acrónimo de *word count*

sort *fic* — ordena as linhas do ficheiro ***fic*** de acordo com um critério (alfanumérica crescente, por omissão).

find *dir -name fic* — procura um ficheiro com o nome ***fic*** a partir do directório ***dir***.

Todos estes comandos podem ser invocados usando argumentos opcionais que configuram o seu modo de funcionamento.

Exercício 2.16

Utilizando a linha de comandos, crie um directório chamado ***aula01*** no seu Ambiente de Trabalho e copie o ficheiro ***/etc/passwd*** para este directório. Imprima o seu conteúdo no ecrã.

Experimente outros comandos da lista acima.

2.3 Edição de ficheiros de texto

Nos computadores do laboratório temos instaladas as aplicações **vim** e **gedit** para manipulação de ficheiros de texto na consola e num ambiente gráfico, respectivamente. Na máquina virtual apenas está instalado a aplicação **vim**.

Um ficheiro de texto é um ficheiro constituído por octetos (bytes) que representam caracteres alfanuméricos, sinais de pontuação, espaços em branco e caracteres invisíveis de mudança de linha. Um dos editores de ficheiros de texto mais usados em sistemas UNIX é o **vim**, uma versão melhorada do **vi**. Apesar da sua antiguidade, continua a ser preferido por muitos utilizadores, particularmente programadores, pela sua eficiência, pelas suas funções avançadas e por correr numa consola de texto. Atualmente também existem versões gráficas, como o **gvim** e o **vim-qt**⁷ que é desenvolvido por membros da Universidade de Aveiro!

O **vim** é um editor com um modo de funcionamento único, porque permite usar o teclado completo como fonte de caracteres (para o texto que se pretende introduzir) e como fonte de comandos. Assim, se durante uma edição de um texto se carregar na tecla “a”, o resultado pode ser a introdução do carácter “a” no texto, no local onde se encontra o cursor, ou a execução do comando associado à tecla *a*, dependendo do modo de trabalho actual no editor vim.

O **vim** trabalha em dois modos, ditos de *comando* e de *inserção*. No modo de comando, todas as teclas representam comandos; no modo de inserção, todas as teclas representam algo que se pretende inserir no texto. A mudança entre estes dois modos faz-se com as seguintes teclas:

modo de comando → **modo de inserção** : através de uma de entre várias teclas que indicam a posição onde se pretende introduzir texto. Algumas das mais usadas:

- i** — inserir no local do cursor.
- o** — inserir uma linha abaixo da atual.
- O** — inserir uma linha acima da atual.

modo de inserção → **modo de comando** : através da tecla ESC (*escape*).

Para além destes dois modos, o **vi** possui ainda dois outros modos: editor (**ed**) e visual.

⁷<https://bitbucket.org/equalshraf/vim-qt/wiki/Home>

O modo **ed** é acionado quando no modo de comando se carrega na tecla “:”. Este modo serve para executar comandos relacionados com a salvaguarda do conteúdo do ficheiro e com o abandono da edição:

:q — terminar a edição atual sem salvaguardar o conteúdo do ficheiro.

:q! — terminar a edição atual sem salvaguardar o conteúdo do ficheiro, ignorando avisos caso tenha sido alterado.

:x — terminar a edição atual com a salvaguarda do conteúdo do ficheiro.

:w — salvaguardar o conteúdo do ficheiro.

:w fic — salvaguardar o conteúdo editado no ficheiro **fic**.

:e fic — abre o ficheiro **fic** para edição.

:r fic — insere o conteúdo do ficheiro **fic** abaixo do cursor.

O modo visual é o modo por omissão, ao qual o **vim** volta após se ter executado um comando no modo **ed**. Na versão gráfica do **vim** (**gvim**) pode fazer as operações do modo **ed** recorrendo aos menus da sua interface gráfica.

Caso não se sinta confortável com o **vim** pode usar outros editores mais convencionais, como o **gedit**, **kate**, **leafpad** ou outros. Porém, o **vim** tem a vantagem de se poder usar quase da mesma forma em ambientes gráficos e em consolas.

Todos estes editores possuem a função de realce de sintaxe, muito útil para programadores. Ou seja, sempre que editar um texto numa linguagem de programação ou de representação conhecida, o editor usa cores diferentes para realçar blocos sintáticos distintos, de modo a facilitar a sua leitura e análise.

Para programação, também existem ambientes integrados de desenvolvimento (IDE) que além de um editor também incluem ferramentas de compilação, execução e depuração. Um IDE simples e versátil é o **geany**.

Exercício 2.17

Edite um ficheiro e escreva os aspetos do sistema *Linux* que mais o surpreenderam, tanto positivamente como negativamente. Experimente fazê-lo com o **vim**, e exercite as diferentes formas de entrada em modo de inserção, bem como alguns comandos do modo **ed**.

2.3.1 Procura de texto

O modo de procura de texto do **vim** é o mesmo que se usa em vários outros comandos, como o **man**, o **more**, o **less**, etc.

A pesquisa de um bloco de texto num ficheiro é feita em modo comando com a tecla **/** (pesquisa para diante do cursor) ou **?** (pesquisa para trás do cursor).⁸ Após a escrita de um destes caracteres deve-se escrever o texto a procurar e terminar com a tecla **Enter** (ou **Return**), após o que a aplicação desloca o cursor até ao texto encontrado. Para encontrar a ocorrência seguinte, usa-se o comando **n**. Para inverter o sentido e encontrar a ocorrência anterior, usa-se o comando **N**.

Exercício 2.18

Edite o ficheiro anterior com o **vim** e realize pesquisas de texto no mesmo. Note que para o fazer terá de estar em modo de comando.

Exercício 2.19

Apresente o conteúdo do ficheiro anterior com o comando **less** e realize pesquisas de texto no mesmo.

Exercício 2.20

Execute o comando **man less** e realize pesquisas de texto do manual, tanto para diante como para trás.

Exercício 2.21

Leia a página de manual do comando **less** para descobrir como se consegue saltar para o início e para o fim do texto e experimente esses comandos. Tenha em consideração que a deslocação para um ponto do texto é referida através da expressão “Go to” no manual.

⁸Num teclado americano estes símbolos estão na mesma tecla física e num local muito conveniente.

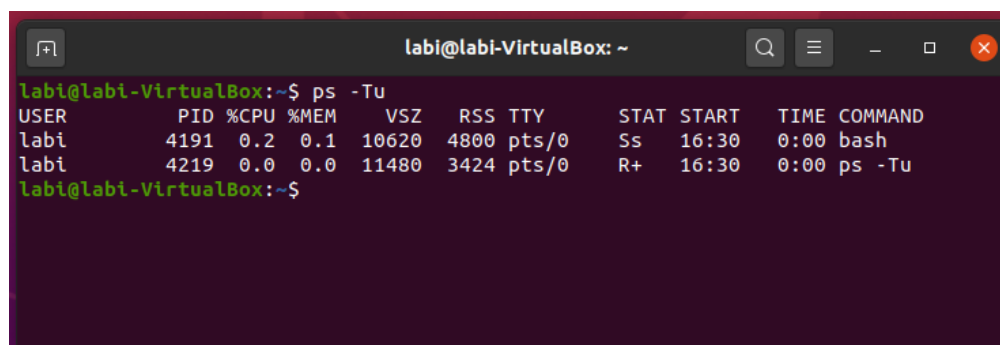
2.4 Processos

Em *Linux* existe uma máxima que diz que tudo é um ficheiro ou um processo. Até agora vimos como funcionam os ficheiros, mas ignorámos os processos.

Os processos não são nada mais do que os programas em execução na máquina. Todo o ambiente gráfico ou de texto que lhe é apresentado, assim como a *Shell* e muitos outros programas estão a executar simultaneamente. Um programa executado várias vezes dá origem a vários processos. No sistema, cada processo é identificado por um número chamado de Process Identifier (PID).

Pode utilizar o comando **ps** para verificar os processos da sua sessão, ou combinar com as opções **-ax** para ver que processos estão activos, independentemente do utilizador. Se adicionar a opção **-u** irá igualmente ver detalhes dos processos como por exemplo a memória utilizada.

O comando **ps Tu** irá mostrar os processos associados com o terminal e os seus detalhes. Como pode ver (Figura 2.6), é mostrado o utilizador, o PID, a quantidade de processador e memória utilizados, o seu estado, há quanto tempo foram iniciados e qual o seu nome.



```
labi@labi-VirtualBox: ~  
labi@labi-VirtualBox:~$ ps -Tu  
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND  
labi      4191  0.2  0.1  10620  4800 pts/0    Ss   16:30   0:00 bash  
labi      4219  0.0  0.0   11480  3424 pts/0    R+   16:30   0:00 ps -Tu  
labi@labi-VirtualBox:~$
```

Figura 2.6: Resultado da execução do comando **ps Tu**

Exercício 2.22

Compare o resultado de **ps**, **ps -a**, **ps -ax** e **ps -aux**.

Qual o processo que consome mais memória? Qual o processo que consome mais CPU? Quantos utilizadores têm processos activos?

Exercício 2.23

Repita o exercício anterior utilizando o comando **top**. Pode utilizar as teclas < e > para seleccionar qual a coluna escolhida para ordenar os processos. Use q para terminar.

Um aspecto importante dos processos é a sua gestão, nomeadamente a possibilidade de controlar o seu estado de execução e, eventualmente terminá-la de forma forçada.

Os comandos **kill** e **killall** permitem enviar sinais aos programas. A diferença entre estes comandos reside no facto de o primeiro (**kill**) enviar um sinal a um processo específico identificado pelo seu identificador:

```
kill [-sinal] identificador-do-processo
```

Enquanto o segundo envia um sinal a todos os processos com um determinado nome:

```
killall [-sinal] nome-do-processo
```

Se não se indicar o sinal, é enviado o sinal **SIGTERM**, que geralmente provoca a terminação do processo de forma relativamente segura. No entanto, em certas condições, um processo pode ficar num estado em que não reage ao sinal. Nesse caso, pode usar-se o sinal **SIGKILL** ou **9**, que não pode ser ignorado, mas corre-se maior risco de perder dados que não tenham sido gravados. Existem sinais com outras funcionalidades. Pode consultar uma lista com o comando **man 7 signal**.

Exercício 2.24

Inicie dois terminais. No primeiro inicie um processo **top**.

No segundo terminal, recorrendo aos comandos **ps** e **kill** ou **killall**, termine a execução do processo **top**.

2.5 Aspectos mais avançados de utilização da *shell*

2.5.1 Caracteres especiais para seleccionar grupos de ficheiros

Por vezes há a necessidade de seleccionar um determinado grupo de ficheiros, pelo que a *shell* providencia alguns caracteres (*wildcards*) com significado especial quando utilizado em nomes de ficheiros:

- `*` Iguala qualquer sequência com zero ou mais caracteres.
- `?` Iguala qualquer carácter na posição indicada.
- `[xyz]` Iguala qualquer carácter indicado entre `[e]`. São permitidas séries tais como `0-9` e `a-z`.

Exercício 2.25

Execute os comandos seguintes e interprete os resultados:

```
cd /bin
ls c*
ls ?ed
cd /dev
ls tty[0-5]
```

2.5.2 Descritores de ficheiros

Quando um comando é posto em execução, tem associado três ficheiros representados por números inteiros que se designam por descritores de ficheiros:

- O descritor 0 representa o ficheiro de entrada (**stdin** na linguagem C – **System.in** na linguagem Java) que está associado com o dispositivo convencional de entrada, que é o teclado.
- O descritor 1 representa o ficheiro de saída (**stdout** na linguagem C – **System.out** na linguagem Java) que está associado com o dispositivo convencional de saída, que é o monitor.
- O descritor 2 representa o ficheiro de saída de erro (**stderr** na linguagem C – **System.err** na linguagem Java), que também está associado com o dispositivo convencional de saída.

Os descritores de ficheiros servem para representar os respectivos ficheiros quando pretendemos fazer redirecionamento de dados dos comandos, como vamos ver nalguns exemplos seguintes.

2.5.3 Redirecionamento de entrada e de saída de dados de comandos

Existem comandos cuja entrada é recebida pelo teclado. A *shell* permite redireccionar a entrada de um comando do teclado para um ficheiro anteriormente criado, o que se designa por **redirecionamento de entrada**, utilizando o operador `<` da seguinte forma:

```
$ comando < ficheiro_de_entrada
```

Como esta propriedade é pouco útil não mostaremos qualquer exemplo da sua utilização. Muitas situações de redirecionamento de entrada podem ser sempre substituídas por encadeamento de comandos usando um *pipe*.

Normalmente o resultado de um comando é enviado para o monitor. No entanto, há a possibilidade de redireccioná-lo para um ficheiro para posterior análise, o que se designa por **redirecionamento de saída**, utilizando o operador `>` da seguinte forma:

```
$ comando > ficheiro_de_saída
```

Vejamos o exemplo que se apresenta na Figura 2.7. A invocação do comando **date** não apresentou qualquer resultado no monitor porque a saída do comando foi redireccionada para o ficheiro **saida.txt**. Se imprimirmos o conteúdo do ficheiro, usando o comando **cat**, veremos o resultado que deveria ter aparecido no terminal.

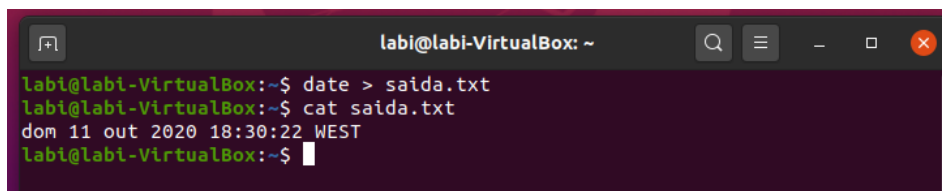
A screenshot of a terminal window titled 'labi@labi-VirtualBox: ~'. The terminal shows the following commands and output: 'labi@labi-VirtualBox:~\$ date > saida.txt', 'labi@labi-VirtualBox:~\$ cat saida.txt', and the output 'dom 11 out 2020 18:30:22 WEST'. The prompt 'labi@labi-VirtualBox:~\$' is visible at the bottom.

Figura 2.7: Exemplo de redirecionamento de saída

Mas, o redirecionamento de saída, para um ficheiro já existente, faz com que o seu conteúdo seja apagado. Para juntar o resultado de um comando a um ficheiro já existente, deve utilizar-se em alternativa o operador de redirecionamento de saída `>>`.

Exercício 2.26

Usando os comandos **ls** e **date** e o redirecionamento de saída apropriado crie um ficheiro (de nome **lsdate.txt**) com a data e a listagem do seu directório *Home*.

Existem comandos que, para além de produzirem um resultado, produzem também resultados que não são entendidos como saída, mas sim como indicador da ocorrência de situações de erro.

A implementação do **redirecionamento de saída de erro** depende da *shell* e da intenção do utilizador de juntar ou não a saída de erro com a saída. No caso da Bourne *shell*, se pretendermos armazenar a saída de erro num ficheiro separado, deve proceder-se da seguinte forma:

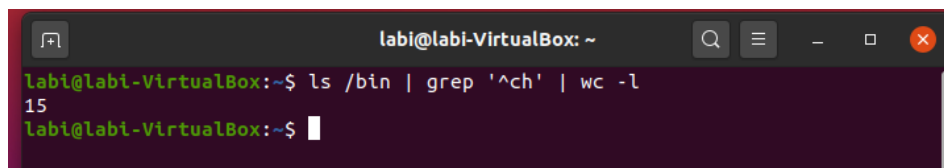
```
$ comando [ > ficheiro_de_saída ] 2 > ficheiro_de_saída_de_erro
```

2.5.4 Comunicação entre processos

A *shell* também permite a comunicação entre comandos, possibilitando encaminhar a saída de um comando para a entrada de outro comando. Este mecanismo designa-se por *pipe*. Para isso utiliza-se o operador `|` da seguinte forma:

```
$ comando1 | comando2
```

Vejamos o exemplo que se apresenta na Figura 2.8. O comando começa por listar todos os ficheiros do directório `/bin` que começam pelos caracteres **ch** – isto porque o padrão **ch** é precedido pelo carácter `^` – e imprime no terminal o seu número.

A screenshot of a terminal window titled 'labi@labi-VirtualBox: ~'. The terminal shows a command being executed: 'ls /bin | grep '^ch' | wc -l'. The output of the command is '15'. The prompt 'labi@labi-VirtualBox:~\$' is visible at the bottom of the terminal window.

```
labi@labi-VirtualBox:~$ ls /bin | grep '^ch' | wc -l
15
labi@labi-VirtualBox:~$
```

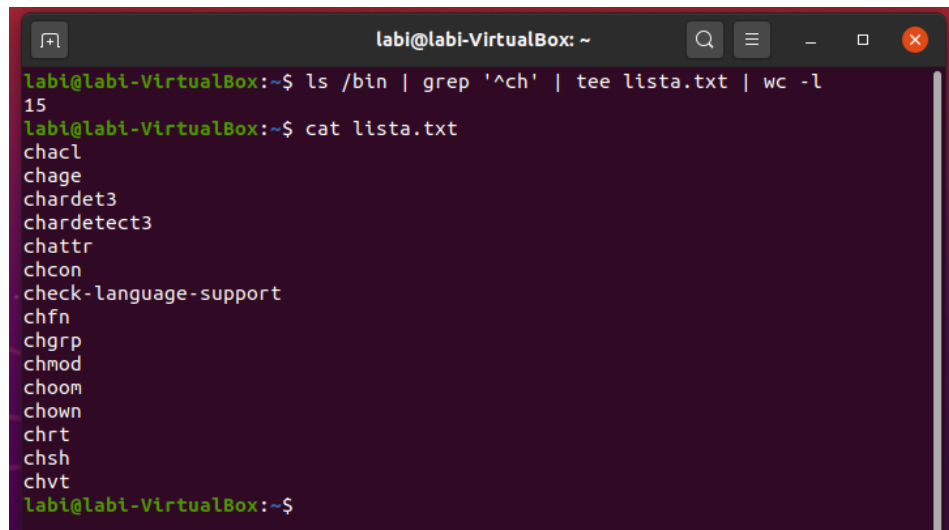
Figura 2.8: Exemplo de um *pipe*

Para obter dados no interior de um *pipe* podemos usar o comando **tee**, que faz por assim dizer uma amostragem sem provocar qualquer alteração ao seu normal funcionamento.

O exemplo que se apresenta na Figura 2.9 faz exactamente o mesmo que o exemplo anterior, mas armazena no ficheiro **lista.txt** os nomes dos comandos, que começam pelo padrão pretendido, que depois podem ser listados no terminal. Ou no caso do ficheiro ser muito extenso este pode ser analisado usando um editor de texto. Em alternativa ao armazenamento dos dados de saída num ficheiro, podemos enviá-los para o terminal.

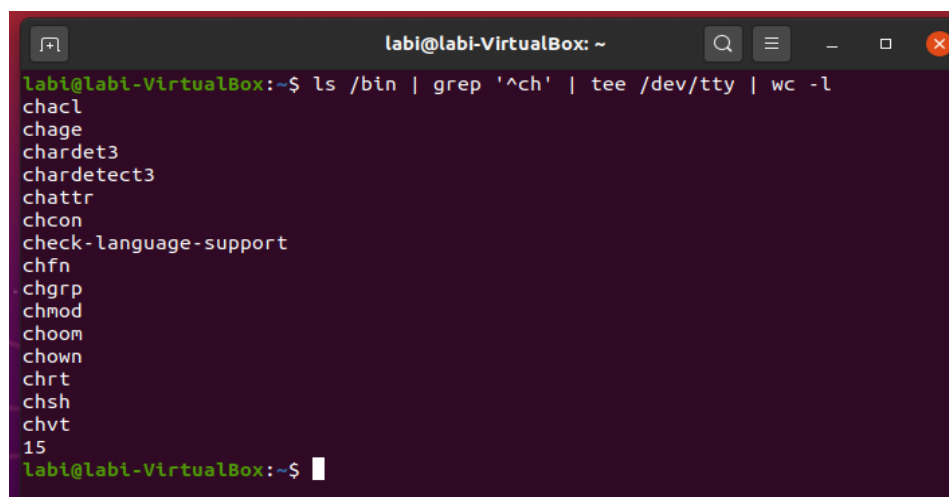
A Figura 2.10 apresenta a solução sendo que o ficheiro `/dev/tty` representa o terminal no qual o utilizador está a trabalhar.

No sistema operativo *UNIX/Linux* os dispositivos de entrada e de saída de dados são representados por ficheiros de texto e estão armazenados no directório `/dev`.



```
labi@labi-VirtualBox: ~  
labi@labi-VirtualBox:~$ ls /bin | grep '^ch' | tee lista.txt | wc -l  
15  
labi@labi-VirtualBox:~$ cat lista.txt  
chac1  
chage  
chardet3  
chardetect3  
chattr  
chcon  
check-language-support  
chfn  
chgrp  
chmod  
choom  
chown  
chrt  
chsh  
chvt  
labi@labi-VirtualBox:~$
```

Figura 2.9: Exemplo de um *pipe* com a utilização do *tee*



```
labi@labi-VirtualBox: ~  
labi@labi-VirtualBox:~$ ls /bin | grep '^ch' | tee /dev/tty | wc -l  
chac1  
chage  
chardet3  
chardetect3  
chattr  
chcon  
check-language-support  
chfn  
chgrp  
chmod  
choom  
chown  
chrt  
chsh  
chvt  
15  
labi@labi-VirtualBox:~$
```

Figura 2.10: Exemplo de um *pipe* com a utilização do *tee* e escrita no monitor

Exercício 2.27

Coloque um programa de Java no seu directório e com um *pipe* determine o número de vezes que o programa usa a instrução **if**.

Importante! Num *pipe*, o redirecionamento de entrada é feito no comando inicial e o redirecionamento de saída é feito no comando final da seguinte forma:

```
$ comando1 < ficheiro_de_entrada | ... | comandoN > ficheiro_de_saída
```

2.5.5 Substituição de um comando pela sua saída

A saída de um comando pode ser incorporada como argumento de entrada de outro comando, envolvendo a sua execução em sinais graves `...`. A Figura 2.11 mostra como se pode usar um *pipe* para incorporar o resultado do comando **wc -l** no comando **echo** de forma a imprimir no terminal uma mensagem de texto com o resultado pretendido.

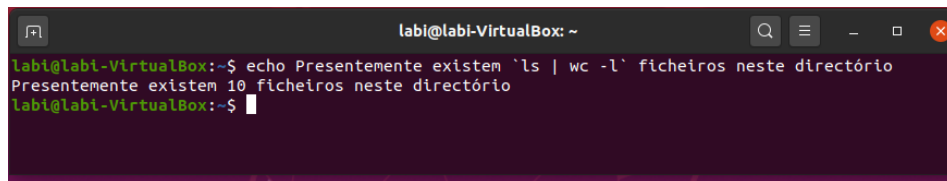


Figura 2.11: Exemplo de um *pipe* com substituição de um comando pela sua saída

2.5.6 Execução de programas

Quando a shell é utilizada no modo interactivo, o utilizador tem de esperar que um comando termine para poder executar outro comando, o que se designa por execução no “primeiro plano” (**foreground**). Este é o modo normal de execução de programas, tal como se usou em todos os exemplos anteriormente apresentados.

Mas para comandos de longa duração, existe a possibilidade de os mandar executar sem ter de se esperar pela sua finalização, podendo assim continuar a executar outros comandos encontrados que obedecem ao padrão indicado, o que se designa por execução nos “bastidores” (**background**), utilizando o operador **&** a finalizar o comando da seguinte forma:

```
$ comando &
```

Esta possibilidade é muito útil, por exemplo, quando usamos uma *shell Linux* onde não podemos executar vários terminais.

2.6 Administração básica

(Poderá saltar esta secção numa primeira leitura.)

Com certeza já reparou que muitos diretórios e ficheiros do sistema pertencem a um utilizador chamado **root**, e só ele tem permissão de os modificar. Isto impede que um erro de outro utilizador possa destruir o sistema, mas também implica que só o **root**, também chamado de *super-utilizador* ou *administrador* do sistema, consegue fazer certas tarefas como instalar software nos diretórios do sistema ou registar novos utilizadores, por exemplo.

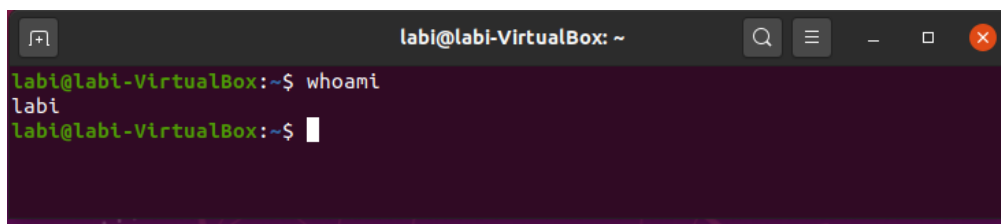
Nos sistemas modernos é usual permitir que um ou mais utilizadores do sistema possam usar o comando **sudo** para assumir temporariamente o papel de super-utilizador e assim realizar tarefas de administração. O primeiro utilizador criado numa instalação de *Linux* tem geralmente essa permissão.

2.6.1 Gestão de utilizadores

O sistema *Linux* assume uma gestão baseada em utilizadores e grupos. A utilização deste sistema permite que múltiplos utilizadores façam uso do sistema apenas após autenticação, respeitando a privacidade de cada utilizador, e evitando que um dado utilizador danifique os dados de um outro utilizador.

Este modelo foi generalizado de forma que além de utilizadores reais, como é o caso do utilizador **labi**, também há *utilizadores de sistema* que servem apenas para confinar aplicações a definições de segurança específicas. Por exemplo, um processo que serve páginas Web pertence a um certo utilizador de sistema para ter acesso aos ficheiros das páginas que serve, mas não aos ficheiros de qualquer outro utilizador.

Pode verificar qual o seu utilizador através do comando **whoami**, ou em mais detalhe através do comando **id**.⁹ O resultado deverá ser o demonstrado na Figura 2.12. Experimente o mesmo comando no computador da sala (fora da máquina virtual).

A terminal window titled 'labi@labi-VirtualBox: ~' with search, menu, and window control icons in the title bar. The terminal shows the command 'whoami' being executed, with the output 'labi' displayed on the next line. The prompt 'labi@labi-VirtualBox:~\$' is visible at the bottom.

```
labi@labi-VirtualBox:~$ whoami
labi
labi@labi-VirtualBox:~$
```

Figura 2.12: Resultado da execução do comando **whoami**

⁹O comando **id** irá mostrar o utilizador e todos os grupos aos quais ele pertence

Exercício 2.28

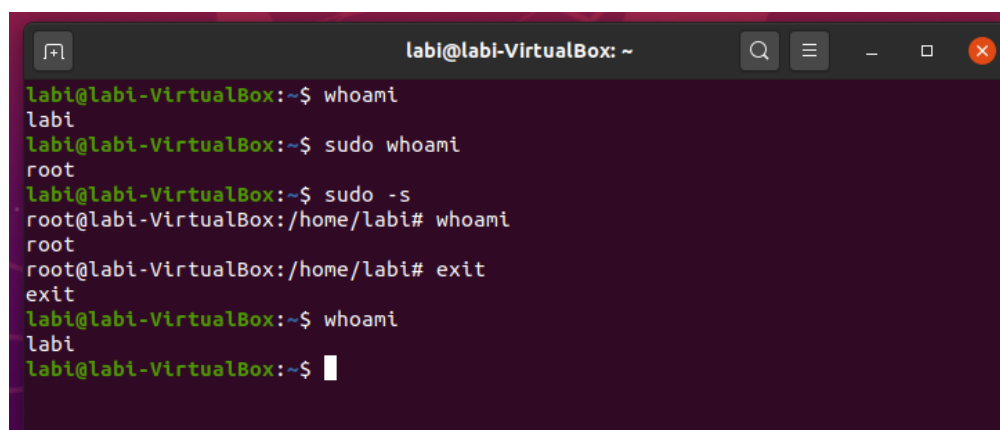
Utilizando os comandos **mkdir** e **rmdir** verifique se possui permissões para escrever nos seguintes locais: **/home/labi**, **/etc**, **/tmp**, **/bin**.

Na máquina virtual, o utilizador **labi** consegue assumir temporariamente as funções de super-utilizador (**root**) através do comando **sudo**.

Na primeira vez que um utilizador usa o comando **sudo**, é pedida a sua palavra passe para confirmação. Execuções seguintes na mesma sessão “lembram-se” da validação durante alguns minutos e não a pedem de novo.

A Figura 2.13 demonstra a utilização do comando **sudo** nas suas duas formas mais comuns. Na primeira forma, (**sudo whoami**), o comando **whoami** é executado como super-utilizador, mas o comando **whoami** seguinte é novamente executado pelo utilizador **labi**.

A segunda forma, **sudo -s**, cria uma nova *Shell* no contexto do super-utilizador (repare no prompt). Todos os comandos inseridos serão executados com os privilégios de **root**, até que se termine esta shell com **exit** ou **Ctrl-D**.



```
labi@labi-VirtualBox: ~  
labi@labi-VirtualBox:~$ whoami  
labi  
labi@labi-VirtualBox:~$ sudo whoami  
root  
labi@labi-VirtualBox:~$ sudo -s  
root@labi-VirtualBox:/home/labi# whoami  
root  
root@labi-VirtualBox:/home/labi# exit  
exit  
labi@labi-VirtualBox:~$ whoami  
labi  
labi@labi-VirtualBox:~$
```

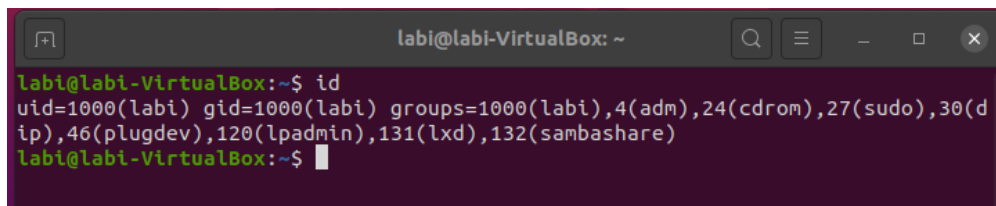
Figura 2.13: Resultado da execução do comando **sudo** com inspeção do utilizador em uso

Exercício 2.29

Verifique se possui permissões para visualizar (**cat**) o ficheiro **/etc/shadow**. Se não possuir, visualize o seu conteúdo recorrendo ao comando **sudo**.

Consegue perceber para que serve este ficheiro? Pode recorrer ao comando **man**.

Na verdade, os utilizadores não são tratados internamente pelo sistema operativo com os nomes que temos utilizado (ex., **labi**). Na realidade tanto os utilizadores como os grupos existentes são mapeados para números. São esses números que se vê no resultado do comando **id**, tal como representado na Figura 2.14.

A terminal window titled 'labi@labi-VirtualBox: ~' with search, menu, and window control icons. The command 'id' has been executed, showing the user's identity and group memberships. The output is: 'uid=1000(labi) gid=1000(labi) groups=1000(labi),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)'.

```
labi@labi-VirtualBox:~$ id
uid=1000(labi) gid=1000(labi) groups=1000(labi),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)
labi@labi-VirtualBox:~$
```

Figura 2.14: Resultado da execução do comando **id** com inspeção do utilizador **labi**

São de relevância dois valores apresentados:

uid — User Identifier, ou seja, o número que identifica o utilizador.

gid — Group Identifier, ou seja, o número do grupo principal do utilizador.

Estes valores são utilizados pelo sistema para controlar as permissões. Os nomes são utilizados para facilitar a gestão aos administradores (humanos).

Exercício 2.30

Verifique qual o **uid** e **gid** do utilizador **root**.

Exercício 2.31

Analise o resultado do comando **ls -la** sobre várias localizações, como por exemplo: **/etc**, **/tmp** e verifique a que utilizadores pertencem os ficheiros e quais as permissões.

2.7 Para aprofundar o tema

Exercício 2.32

Explore os restantes ficheiros e directórios nos directórios `/proc` e `/sys`.

Recorra ao comando `man` para obter informação sobre cada uma das entradas.

Glossário

CD	Compact Disk
CLI	Command Line Interface
PID	Process IDentifier
PC	Computador Pessoal