2

Utilização do sistema UNIX/LINUX

(Bourne Again Shell - BASH)

Os objectivos deste módulo são os seguintes:

- Utilização dos comandos básicos do Unix;
- Utilização de ficheiros e directorias;
- Utilização de dispositivos de entrada/saída, incluindo o seu redireccionamento;
- Utilização de pipetas (pipes) na linha de comando.

Outros Documentos

http://www.di.ubi.pt/~crocker/prog3/docs-sisops.html

Utilização de comandos básicos em UNIX

1. Inquirição sobre a data e o tempo:
   $ date
   Mon Mar 11 19:15:08 GMT 2002

2. Inquirição sobre os utilizadores correntes da máquina:
   $ who
   alunos:~ crocker$ who
   admin console Jan 7 14:47
   david ttyp1 Feb 4 12:24
   crocker ttyp2 Mar 16 22:53 (2.82.2.138)

3. Inquirição sobre o estado do sistema, tempo que está a correr, numero de utilizadores ligado e dados sobre a carga de maquina
   $ uptime
   15:08:27 up 6 days, 5:13, 3 users, load average: 2.06, 2.02, 2.00

4. **echo** (eco) ou impressão no terminal
   $ echo segue-se um teste
   segue-se um teste

5. Informação acerca de tipo de sistema operativo e nome da maquina

   $ uname
   Linux

   $ hostname
   penhas.di.ubi.pt

6. Consulta sobre informações acerca do nome da maquina, arquitectura e sistema operativo
   Exemplo dum emulador de cygwin bash shell num computador pessoal Pentium com OS windows.
   $ uname –a

   Darwin alunos 8.11.0 Darwin Kernel Version 8.11.0: Wed Oct 10 18:26:00 PDT 2007; root:xnu-792.24.17~1/RELEASE_PPC Power Macintosh powerpc

7. **df** . Consulta informação sobre o espaço livre nos sistemas de ficheiros e respectivos pontos de montagem

   alunos:~ crocker$ df
   Filesystem 512-blocks Used Avail Capacity Mounted on
   /dev/disk2 160556800 92714168 67330632 58% /
   /dev/disk3s3 1470414768 694728008 775686760 47% /Volumes/alunos

8. Consulta ao manual on-line acerca do comando echo:

$ man echo

```
echo(1)                                                    echo(1)

NAME

  echo - Writes its arguments to standard output

SYNOPSIS

  echo [-n] [string...]

  [Digital]  The -n flag is valid only if the environment variable CMD_ENV is
  set to bsd.
                              Note
       The C shell has a built-in version of the echo command.  If
       you are using the c shell, and want to guarantee that you
       are using the command described here, you must specify the
       full path /usr/bin/echo.  See the csh(1) reference page for
       a description of the built-in command.

STANDARDS

  Interfaces documented on this reference page conform to industry standards as follows:
…
```

9. Consulta a informação on-line acerca dum comando em formato hipertexto

$ info gcc

Navegação com "selecção da sub- secção seguido por enter" e "< enter para traz)"

Sintaxe Padrão dum Comando

Comando [opções] [ficheiros]

Onde : Opções.. → -letra ou --letra ou

: ficherios → Ficheiros opcionais abertos para leitura e escrita pelo programa do comando.

Exemplos

ls [-ABCFGHLPRTWZabcdefghiklmnopqrstuwx1] [file ...]

sort [-cmus] [-t separator] [-o output-file] [-T tempdir] [-bdfiMnr] [file...]

sort {--help,--version}

Comandos de manipulação de ficheiros

Aqui são mencionados alguns comandos de manipulação de ficheiros:

1. Listagem dos ficheiros da directoria corrente:

```
$ ls
alunos:~ crocker$ ls
Desktop      Templates   cprogs      te
Documents    a.out       cprogs2     temp
```

2. Contagem do número de linhas, palavras e caracteres dum ficheiro :

```
$ wc /etc/passwd
    11      26     225 /etc/passwd
```

3. Cópia dum ficheiro para outro:

```
$ cp /etc/passwd passwords
```

4. Re-nomeação dum ficheiro:

```
$ mv passwords passes
```

5. Eliminação/Remoção dum ficheiro:

```
$ rm passes
```

6. Inquirição sobre a directoria corrente:

```
$ pwd
/usr/marta
```

7. Mudança de directoria:

```
$ ls
documentos
disciplina
$ cd disciplina
$ pwd
/usr/marta/disciplina
```

8. Criação de directoria:

```
$ cd ..
$ mkdir programas
$ ls
documentos
disciplina
programas
```

9. Eliminação/Remoção de directoria:

```
$ rmdir documentos
$ ls
disciplina
programas
```

Inquirição sobre o tipo de ficheiro via uma listagem longa

Há 3 tipos de ficheiros: ficheiros *ordinários (indicados por um -),* ficheiros directórios ou *directorias (indicados por um d) e* ficheiros *especiais (indicados por um s,l,p,b,c,..).* Os ficheiros ordinários contêm dados, texto, instruções de programa, etc. As diretorias contêm informação acerca de outros ficheiros. Os ficheiros especiais estão normalmente associados a entrada/saída (E/S), como por exemplo sockets (s), filas (p), dispositivos de bloco (b) e de carácter (c) e atalho/link (l).

```
$   cd programas
$   ls -l
drwxr-xr-x   2   marta   users       0 Mar    5   00:30 .
drwxr-xr-x   4   marta   users       0 Jun   23   2001 ..
-rw-r--r--   1   marta   users      92 Mar   6   09:10 funcsum.c
-rw-r--r--   1   marta   users     451 Mar   6   09:10 funcsum.o
-rw-r--r--   1   marta   users     184 Mar   5   00:51 mainsum.c
-rw-r--r--   1   marta   users     696 Mar   6   08:59 mainsum.o
```

10. Cópia de ficheiro duma directoria para outra:

```
$   cd ;  mkdir obj
$   cp programas/mainsum.o obj/main.o
$   ls obj
main.o
```

11. Transladação de ficheiros entre directorias (**mv**):

```
$   cd ; mv programas/*.o obj
$   ls –l obj
drwxr-xr-x   2   marta   users       0 Mar    5   00:30 .
drwxr-xr-x   4   marta   users       0 Jun   23   2001 ..
-rw-r--r--   1   marta   users     451 Mar   6   09:10 funcsum.o
-rw-r--r--   1   marta   users     184 Mar   5   00:51 main.o
-rw-r--r--   1   marta   users     696 Mar   6   08:59 mainsum.o
```

12. Ligação de ficheiros (hard links ln vs soft links ln -s) (**ln**):

```
$   cd obj
$   ln  funcsum.o  func.o
$   ls –l
drwxr-xr-x   2   marta   users       0 Mar    5   00:30 .
drwxr-xr-x   4   marta   users       0 Jun   23   2001 ..
-rw-r--r--   2   marta   users     451 Mar   6   09:10 funcsum.o
-rw-r--r--   2   marta   users     451 Mar   6   09:10 func.o
-rw-r--r--   1   marta   users     184 Mar   5   00:51 main.o
-rw-r--r--   1   marta   users     696 Mar   6   08:59 mainsum.o
```

13. Substituição de nomes e símbolos especiais (*,?,[],!) : * = cadeias de caracteres

```
$   cd ../programas
$   ls –l
drwxr-xr-x   2   marta   users       0 Mar    5   00:30 .
drwxr-xr-x   4   marta   users       0 Jun   23   2001 ..
-rw-r--r--   1   marta   users     451 Mar   6   09:10 funcsum.c
-rw-r--r--   1   marta   users     696 Mar   6   08:59 mainsum.c

$   cat *.c
int somatorio(int n)
{
  int soma =0;

  for (; n>= 1; n--)
    soma += n;
  return soma;
}

#include <stdio.h>
int somatorio(int);
```

```
int main()
{
  int n;
  printf("Enter a number: ");
  scanf("%d",&n);
  printf("Sum from 1 to %d = %d \n",n,somatorio(n));
  return 0;
}
```

```
$    cat f*.c
int somatorio(int n)
{
  int soma =0;
  for (; n>= 1; n--)
    soma += n;
  return soma;
}
```

```
$    cat *ain*.c
#include <stdio.h>
int somatorio(int);
int main()
{
  int n;
  printf("Enter a number: ");
  scanf("%d",&n);
  printf("Sum from 1 to %d = %d \n",n,somatorio(n));
  return 0;
}
```

```
$    ls -la f*.c
-rw-r--r--    1   marta    users        451 Mar  6   09:10 funcsum.c
```

14. Substituição de caracteres singulares em nomes de ficheiros (**?**):

```
$    touch  a
$    ls –l ?
drwxr-xr-x    1   marta    users        0 Mar     11    23:30 a
$    ls –l fun?sum.c
-rw-r--r--    1   marta    users        451 Mar  6   09:10 funcsum.c
$    ls –l fun??um.c
-rw-r--r--    1   marta    users        451 Mar  6   09:10 funcsum.c
$    ls –l ?????um.c
-rw-r--r--    1   marta    users        451 Mar  6   09:10 funcsum.c
-rw-r--r--    1   marta    users        696 Mar  6   08:59 mainsum.c
```

Outra maneira de estabelecer a concordância e substituição dum único carácter é através duma lista de caracteres possíveis entre parenteses rectos [ ]. Por exemplo, [abcdf] pretende fazer a concordância com um destes três caracteres: a, b ou c. Esta lista é equivalente a [a-f]. Do mesmo modo, [a-np-z]* especifica todos os ficheiros cujo primeiro carácter é qualquer letra minúscula do alfabeto excepto o minúsculo. O símbolo ! pode ser usado como negador dum carácter ou lista de caracteres. Assim, [!a-f] especifica qualquer letra minúscula excepto a, b, c, d, e, e ainda f.

```
$    ls –l [a-z]*[!0-9]
-rw-r--r--    1   marta    users        451 Mar  6   09:10 funcsum.c
-rw-r--r--    1   marta    users        696 Mar  6   08:59 mainsum.c
```

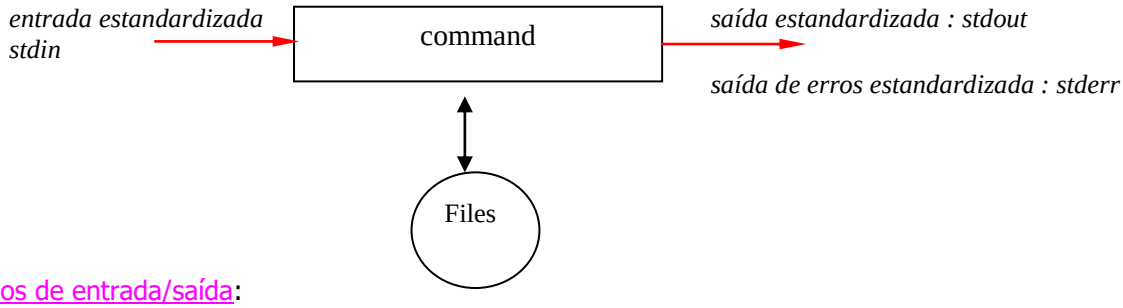
Note-se que o ficheiro a não aparece na listagem.

Re-direccionamento de Entrada/Saída estandardizada

Na maior parte dos comandos UNIX, o terminal serve para entrada e saída de dados.
Um comando efectua a leitura de dados a partir da entrada estandardizada, a qual é, por defeito, o teclado.
Do mesmo modo, por defeito, um comando faz a escrita de dados para o ecrã.

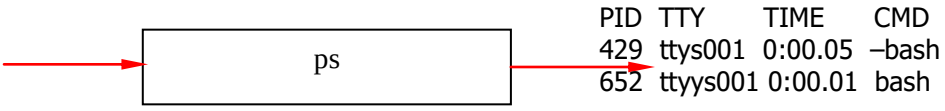
Sintaxe Padrão : command [-opçoes] [ficheiros]

Diagrama dum comando LINUX:

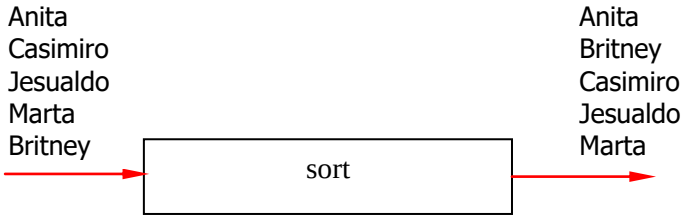


Exemplos de entrada/saída:

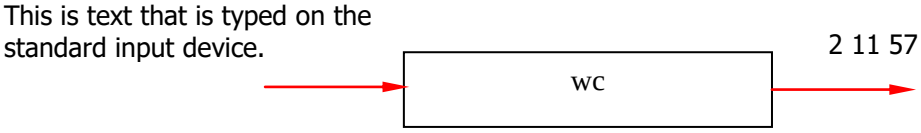
```
$   ps
PID  TTY      TIME     CMD
429  ttys001  0:00.05 –bash
652  ttyys001 0:00.01  bash
```




```
$   sort
Anita
Casimiro
Jesualdo
Marta
Britney
CTRL-d
Anita
Britney
Casimiro
Jesualdo
Marta
```




```
$   wc
This is text that is typed on the
standard input device.
CTRL-d
2  11  57
$
```



Re-direccionamento de Entrada/Saída estandardizada

Exemplos de redireccionamento (de saída):

1. Saída redireccionada para o ficheiro users (em modo de reescrita ou >):

```
$   ps -A > processos
$   cat processos
PIDTTY          TIMECMD
  69 ??         0:01.05/sbin/launchd
  82 ??         0:00.09 /System/Library/CoreServices/AirPortBase Station Agent.a
  86 ??         0:00.39/System/Library/CoreServices/Spotlight.app/Contents/MacOS
  87 ??         0:00.33/usr/sbin/UserEventAgent -l Aqua
  ....
$
```

2. Saída re-direccionada para os ficheiros shake e users (em modo de concatenação ou >>):

```
$   echo romeu sobe a varanda > shake
$   cat shake
romeu sobe a varanda
$   echo julieta escorrega e cai da varanda >> shake
$   cat shake
romeu sobe a varanda
julieta escorrega e cai da varanda
```

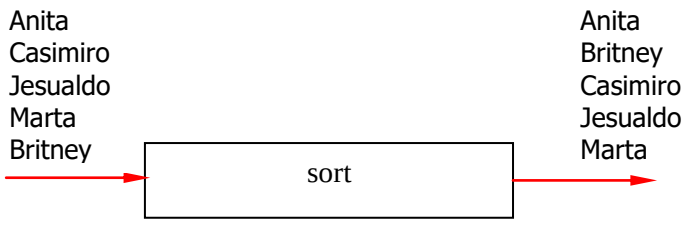
Exemplos de redireccionamento (de entrada):

3. Entrada re-direccionada a partir do ficheiro users (<):

4. Entrada re-direccionada a partir do ficheiro nomes (<):

```
$   pico nomes
    Anita
    Casimiro
    Jesualdo
    Marta
    Britney

$   cat< nomes
Anita
Casimiro
Jesualdo
Marta
Britney
$   sort< nomes
Anota
Britney
Casimirio
Jesualdo
Marta
```



Pipetas ou Tubos de Canalização! (Pipes)

Já vimos como funciona o redireccionamento da entrada e da saída estandardizadas.
Por exemplo, com a saída redireccionada para o ficheiro **users** (em modo de reescrita ou >), temos:

```
$ who > users
$ cat users
agomes    ttyp5    Mar 11 18:12
rebelo    ttyp6    Mar 11 18:15
mario     ttyp6    Mar 11 18:21
```

Do mesmo modo, com a entrada redireccionada a partir do ficheiro users, é possível saber quantos utilizadores correntes estão a usar a máquina:

```
$ wc -l < users
   3
```

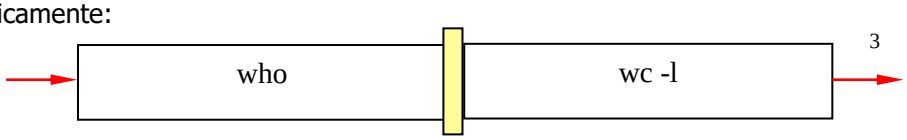
Temos, pois, uma sequência de comandos que permite saber quantos utilizadores estão a usar a máquina.

Mas, existe outra forma de saber quantos utilizadores estão activos sem usar o ficheiro intermédio **users** e onde os comandos são efectuados em simultâneo (Paralelamente ou concorrentemente). Para isso, basta fazer a junção dum comando com o seguinte de modo que a saída do primeiro será a entrada do segundo. Esta junção de comandos é conhecida por **pipeta** (ou pipe) e é feita usando o símbolo **|** entre os dois comandos.

Assim, para saber quantos utilizadores estão ligados à máquina, basta fazer o seguinte:

```
$ who | wc –l
   3
```

Diagramaticamente:



Exemplo Pipeta 2

Usando o comando curl para obter um "feed" de noticias e grep para pesquisar um a linha de interesse

```
$ curl https://www.rtp.pt/noticias/rss/desporto > tmp
$ grep benfica tmp
$ rm tmp
```

versus **$ curl https://www.rtp.pt/noticias/rss/desporto | grep benfica**

Modifique o comando em cima para enviar as estatísticas para o ficheiro stats.txt

Sumário : Redireccionamento de Outout

É possivel seperar os canais sdtout e stderr embora por defeito ambos os canais irão para o terminal.

```
// Redirect standard output to file ".txt". Will OVERWRITE output.txt if exists
java MyProgram > output.txt

// Append stdout to file, instead of overwriting
java MyProgram >> output.txt

// Redirect standard error to err.txt
java MyProgram 2> output.txt

// Redirect output to file, and then stderr to stdout. Both will write to file
java MyProgram > output.txt 2>&1

// In Linux, you can redirect all output to /dev/null if you don't want it at all
java MyProgram > /dev/null 2>&1
```

Sumário : Redireccionamento de Input

By default, standard input comes from the keyboard. You can instead pass a file as the input. It will pass along the file and the program won't know the difference that it came from a file instead of from the keyboard.

```
// Send file.txt to MyProgram as standard input
java MyProgram < input.txt
```

Another option is to use the output of another program as the input. This is called **piping** and is the same example we saw just a moment ago when redirecting output through a pipe, except output program is on the receiving end this time.

```
// Pass stdout of another process as stdin to your program.
cat data_file.txt | java MyProgram
```

Another Example

```
// Redirect standard output to the standard input of another program
java MyProgram | grep "keyword"
```

Filtros

Na terminologia UNIX, um filtro é qualquer programa que pode manipular entrada de informação a partir da entrada estandardizada, e , escrever os resultados para a saída estandardizada.
Assim, na seguinte pipeta de comandos, o comando wc é um filtro, mas o comando ls não é um filtro porque não tem entrada de dados a partir da entrada estandardizada:

```
$  ls | wc –l
      10
```

Outros Exemplos : Os commandos **grep, sed, awk, tr**

Outros comandos Úteis

touch : modifica a data de ultima modificação do ficheiro dado em parâmetro. O ficheiro será criado se esse não existir.

curl: transferir um url (https, http, ftp, scp)

wget: GNU Wget is a free utility for non-interactive download of files from the Web

Ref: http://pt.wikipedia.org/wiki/Encadeamento
Poderá experimentar o exemplo do corrector ortográfico descrito nesta referência.

Exit Status

O exit status dum comando é o valor devolvido pelo comando. Se o programa do comando termine duma maneira normal e correcta, chegando ao fim do "main" então por defeito o valor do retorno é zero. No entanto um programa poderá terminar em qualquer ponto do main() chamando return(valor) ou simplesmente chamando a função exit(valor) em qualquer pronto do programa.

O valor do retorno pode ser usado pelo processo que mandou executar o programa por vários motivos! No bash shell este valor poderá ser obtido e usado pois está guardado na variável do shell $?.

Exemplo 1

```
# O ficheiro status.c
#  exit status com valor de
#  zero, um ou dois conforme o valor da variavel x.
#include  <stdio.h>
#include <stdlib.h>
int main()
{
   int x;
   scanf("%d",&x)
   if (x<0)    exit (1);
   if (x>5)    exit (2);
   return (0);
}
```

```
import java.util.Scanner;
public class TestExit {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner sc = new Scanner (System.in) ;
        int x = sc.nextInt();
        if (x<0)     System.exit(1);
        if (x>5)     System.exit(2);
        System.exit(0);
    }
}
```

Execução do Exemplo 1

```
prompt$ cc -o status status.c
prompt$ ./status
  -1
prompt$ echo $?
```

```
prompt$  javac TextExit.java
prompt$  java -cp .  TextExit
```

```
1
prompt$ ./status
  3
prompt$ echo $?
0
prompt$ ./status
  7
prompt$ echo $?
2
```

Exemplo 2

O comando **ls** devolve zero caso seja *sucesso* e outro valor caso dum ficheiro não encontrado

```
prompt$ ls /etc/passwd
/etc/passwd
prompt$ echo $?
0
prompt$ ls /etc/passwdxxxxxxx
ls: /etc/passwdxxxxxxx: No such file or directory
prompt$ echo $?
1
```

Exemplo 3

Verificar que um utilizador está no ficheiro de passwords usando grep. O comando Grep devolve zero quando for encontrada a padrão pesquisado

```
prompt$ grep root /etc/passwd > /dev/null
prompt$ if [ $? -eq 0 ] ; then echo "root esta no ficheiro de passwords"; fi
root esta no ficheiro de passwords

prompt$ grep socrates /etc/passwd > /dev/null
prompt$ if [ $? -eq 0 ] ; then echo "socrates esta no ficheiro de passwords"; fi
prompt$ grep socrates /etc/passwd > /dev/null
prompt$ if [ $? -ne 0 ] ; then echo "socrates NÂO esta no ficheiro de passwords"; fi
socrates NÂO esta no ficheiro de passwords
```

-eq é o operador de "equals" (igualdade) equanto -ne é "not equals" (NÃO IGIAÇ)

REFS

https://www.tutorialspoint.com/linux_admin/basic_centos_linux_commands.htm
https://www.tutorialspoint.com/linux_admin/linux_admin_file_folder_management.htm
https://www.devdungeon.com/content/standard-input-output-and-error-java

Entrada/Saída estandardizada
Vamos ver como um exemplo dum programa parecido com o comando Linux cat

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;


/**
 *
 * @author Paul Crocker
 */
public class JavaCat {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws FileNotFoundException, IOException {
        // TODO code application logic here

        InputStream is =   System.in;              //new FileInputStream("in.txt");
        OutputStream os =  System.out;       //new FileOutputStream("out.txt");

        if ( 1 == args.length )
            is = new FileInputStream(args[0]);

        copiar(is,os);

        System.err.println("Finished");
    }


    public static void copiar(InputStream is, OutputStream os ) throws IOException{
            int c;
            while ( (c = is.read() ) != -1) {
               os.write(c);
            }
            is.close();
            os.close();
    }
}
```

- javac JavaCat.java
- java -classpath . JavaCat
- java -classpath . JavaCat JavaCat.java
- java -classpath . JavaCat < JavaCat.java
- cat JavaCat.java | java -classpath . JavaCat