SO 2020 TP1

62362

André Martins

1. Executable programs or shell commands
2. System calls (functions provided by the kernel)
3. Library calls (functions within program libraries)
4. Special files (usually found in /dev)
5. File formats and conventions eg /etc/passwd
6. Games
7. Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
8. System administration commands (usually only for root)
9. Kernel routines [Non standard]

2.1 - Primeira parte

2.1.1 - Alguns comandos

- man
- info
- id
- whoami
- who
- tty
- finger
- last
- date

- time
- passwd
- echo
- mesg
- mail
- su
- bc
- cal
- clear
- ps
- jobs
- Ctrl z
- bg
- fg
- pwd
- ls
- cd
- find
- locate
- mkdir
- rmdir
- chmod
- cat
- more
- less
- cut
- vi
- head
- tail
- ln
- cp
- mv
- rm
- umask
- file
- wc
- split
- grep

- sort
- diff
- compress
- uncompress
- chgrp
- chown
- df
- du
- mount
- unmount

TODO: outros comandos importantes:

- seq
- timeout
- touch
- wget

man

> interface para a referência dos manuais dos pacotes instalados

Usage:

```
man [opções] page
```

Opções:

| flag | descrição | |---- |---- | | -a | mostra todas as entradas disponíveis no manual para *page* | | -k | mostra as descrições e páginas do manual para *page* onde esta é um regex| | -f | Mostra apenas as descrições das páginas do manual referenciadas por *page* |

info

> Read documentation in Info format. Apresenta mais informação que a página respectiva do manual

Usage:

```
$ info [opções] [ficheiroAdicional] page
```

Opções:

| flag | descrição | |---- |---- | | -o | coloca em um *ficheiroAdicional* especificado a informação presente no manual para *page*| | -f | mostra o ficheiro *./page* | (não consegui colocar esta ultima opção a funcionar com *$info -f ./asd.info*)

id

> Mostra a identidade do utilizador atual ($USER) Pode ser utilizado sem opções

Usage:

```
$ id [opções]
```

Opções:

flag	descrição
-u	mostra o id do utilizador
-g	mostra o id do grupo efetivo do utilizador
-G	mostra os ids de todos os grupos a que o utilizador pertence
-G	mostra apenas o id do utilizador
-n	mostra o nome invés do id. Necessita de -u, -g ou -G

whoami

> mostra o id do utilizador mesmo comando que id -un

Usage:

```
$ whoami
```

Opções:

Sem opções relevantes

who

> Mostra o utilizador que está logado Não consegui colocar este comando a funcionar

Usage:

```
$ who [opções]
```

Opções:

flag	descrição
-b	timestamp do ultimo inicio do sistema
-q	mostra todos os utilizadores e o número de utilizadores atualmente logados

tty

> Mostra o utilizador que está logado Não consegui colocar este comando a funcionar

Usage:

```
$ tty [opções]
```

Opções:

flag	descrição
-b	timestamp do ultimo inicio do sistema
-q	mostra todos os utilizadores e o número de utilizadores atualmente logados

finger

> Mostra informação acerca dos utilizadores do sistema Também pode ser usado para verificar informação de utilizadores remotamente através de "user@host" ou

> apenas "@host" Se um utilizador for especificado comporta-se como -l Senão comporta-se como -s

Usage:

```
$ finger [opções] [user]
```

Opções:

flag	descrição
-s	Mostra nomes login e real, nome da consola, hora de login e IP
-l	mostra a informação presente em -s assim como o diretório do utilizador, shell de login e outras informações como telefone e mail (estas últimas não foi possível testar)

last

> Mostra informação presente no log de logins

Usage:

```
$ last [opções] [user]
```

Opções:

flag	descrição
-n	apenas mostra as N últimas entradas. N passado como parâmetro
-a	coloca o IP do utilizador na última coluna
-F	mostra não só a hora de login como também a hora de logout
-p	mostra os utilizadores atualmente logados. Necessita de um argumento time que poderá ser now, today, yesterday, +5min, -5days, etc

date

>

> Mostra a data e horas atuais do sistema. Também pode ser usado para modificar estas

Usage:

```
$ date [opções] [+format]
```

Opções:

flag	descrição
-I	Mostra apenas a data
-r	Mostra a data de modificação do ficheiro/pasta cujo caminho e nome são passados como parâmetro
-s	Modifica a data e hora para o valor passado por parâmetro

Formatos:

%T - mesmo que %H:%M:%S %F - mesmo que %Y-%m-%d %G - número da semana do ano %q - trimestre (quarter) %u - dia da semana (1 é segunda) %Z - timezone

time

> Este comando executa programas e mostra informação sobre os recursos utilizados por esse programa durante a sua execução. Usualmente, invés de mostrar a informação na consola, informa-se um ficheiro no qual deverá ser escrita a informação. Excelente para verificar informação de um processo como, por exemplo, número de swaps, shared text size, %CPU, entre outros. **PROCURAR O EQUIVALENTE PRA WINDOWS**

Usage:

```
$ time [opções] Comando [argumentos]
```

Opções:

flag	descrição
-o	Escrever o output para um ficheiro cujo caminho é passado por parâmetro

flag	descrição
-a	Igual à opção -o porém faz append invés de overwrite
-s	Modifica a data e hora para o valor passado por parâmetro
-v	Escreve em formato verbose, informação mais legível

passwd

> Permite mudar a password de um utilizador

Usage:

```
$ passwd [opções] [user]
```

Opções:

flag	descrição
-S	Mostra informação geral sobre a password do utilizador
-a	Usado em conjunto com -s, mostra a informação sobre a password de todos os utilizadores **(necessita de sudo)**
-l	Bloqueia a password para alteração do utilizador informado **(necessita de sudo)**
-u	Desbloqueia a password do utilizador informado
-e	Imediatamente expira a password de um utilizador forçando o mesmo a alterá-la no proximo login **(necessita de sudo)**

echo

> Imprime as strings passadas como parâmetro separadas por um espaço

Usage:

```
$ echo [opções] [Strings]
```

Opções:

flag	descrição
-e	enable caracteres de escape
-n	não imprime o último \n

mesg

> Permite a expecificação na shell de login se os utilizadores podem receber mensagens no seu terminal relativamente ao output do standard error Se nenhum argumento é especificado o comando informa o utilizador n|y

Usage:

```
$ mesg [opções] [n|y]
```

Opções:

flag	descrição
-v	verbose

mail

> Permite envio e/ou leitura de mails entre contas

Usage:

```
$ mail [opções] destinatário1,destinatário2
```

Opções:

flag	descrição
-F	Ordena os emails por remetente
-p	mostra todo o mail na consola
-s	Informa que o parâmetro seguinte é o assunto do email

su

> Se username é especificado, permite que um utilizador aja como outro utilizador durante uma sessão Se username não é especificado, permite que um utilizador se transforme no superuser.

Usage:

```
$ su [opções] [comando|shell] [user]
```

Opções:

flag	descrição
-c	Permite executar o comando especificado como o user espeficicado
-l	Fornece um ambiente similar ao do utilizador especificado aparentando a que o mesmo estivesse logado
-s	Permite a invocação de uma shell especificada como o user especificado

bc

> Permite executar uma linguagem de cálculo aritmético

Usage:

```
$ bc [opções] [cálculo]
```

Opções:

flag	descrição
-i	Permite executar o comando iterativamente

cal

> Mostra um calendário com o dia atual em evidência

Usage:

```
$ cal [opções]
```

Opções:

flag	descrição
-m	Especifica um mês para visualização (pode ser inteiro ou uma string com o nome)
-y	Especifica um ano inteiro para visualização
-3	Mostra o mês anterior, o mês atual e o mês seguinte

clear

> Limpa a informação presente no terminal

Usage:

```
$ clear
```

ps

> Permite visualizar os processos em execução

Usage:

```
$ ps [opções]
```

Opções:

flag	descrição
-a	Visualizar todos os processos em execução identico a -e
-p	Visualizar o processo cujo PID é passado como parâmetro
-u	Visualizar os processos relacionados a um dado user cujo ID é passado como parâmetro
-j	Visualizar os processos no mesmo formato que os jobs

jobs

> Mostra os jobs em execução na shell

Usage:

```
$ jobs [opções] [jobID]
```

Opções:

flag	descrição
-r	Mostra apenas os jobs cujo estado é running
-n	Mostra apenas os jobs cujo estado foi alterado desde a última visualização
-s	Mostra apenas os jobs cujo estado é stopped
-l	Mostra também uma coluna com o PID de cada job

Ctrl z

> É efetuada uma interrupção e é passado um sinal SIGTSTP para o processo em execução o que o coloca em suspensão. Processos em suspensão não têm tempo de CPU e, como tal, ficam congelados assim como os recursos a eles atribuidos até lhes ser passado SIGKILL ou SIGCONT

bg

> Coloca o job corrente (ou o job com jobID passado como parâmetro) a executar em segundo plano (background)

Usage:

```
$ bg [jobID]
```

fg

> Coloca o job corrente (ou o job com jobID passado como parâmetro) a executar em primeiro plano (foreground)

Usage:

```
$ fg [jobID]
```

pwd

> Print working directory. Mostra o caminho desde a root até ao diretório corrente

Usage:

```
$ pwd
```

ls

> Lista os diretórios e ficheiros existentes no diretório corrente

Usage:

```
$ ls [opções]
```

Opções:

flag	descrição
-l	Formato longo, mostra diversos atributos de cada um dos conteúdos do diretório
-a	Não ignora ficheiros escondidos (iniciados com um ponto)
-s	Informa o tamanho de cada um dos conteúdos. O tamanho de um diretório é o tamanho da sua configuração não o tamanho de todos os seus conteúdos recursivamente
-las	União das três flags descritas anteriormente

cd

> Change directory, altera a working directory para a nova diretoria informada

Usage:

```
$ cd /path/to/dir
```

find

> Procura ficheiros ou diretórios numa dada diretoria que coincidam com a expressão passada como parâmetro

Usage:

```
$ find [search/file/like/*this*.*]
```

locate

Usage:

```
$
```

Opções:

flag	descrição
-	

mkdir

> Make directory, usado para criar um diretório

Usage:

```
$ mkdir /path/to/new/dir
```

rmdir

> Remove directory, usado para remover um diretório O diretório apenas é removido se se encontrar vazio

Usage:

```
$ rmdir /delete/this/dir
```

chmod

> Permite a alteração das permissões de leitura (r), escrita (w) e execução (x) de um determinado arquivo ou diretório informado. Se a flag -R não for usada a alteração das permissões apenas será aplicada ao diretório e n

Usage:

```
$chmod 752 fileOrDIr
```

Opções:

flag	descrição
-R	Executar o comando recursivamente no caso de uma diretoria para todo o seu conteúdo
-c	verbose mas só quando efetivamente há alteração
-v	verbose

cat

> Mostra o centeúdo de um ficheiro

Usage:

```
$ cat a.txt
```

more

> Usado em outputs grandes, permite uma melhor visualização dos mesmos.

Usage:

```
$ example.sh  | more
```

less

> Mesmas funcionalidades que more mas mais fácil de manusear

Usage:

```
$ cat a.txt | less
```

cut

vi

> :qa+return

head

> Mostra o inicio de um ficheiro

Usage:

```
$ head a.txt
```

Opções:

flag	descrição
-c	
-n	

tail

> O oposto de head

Usage:

```
$ tail a.txt
```

Opções:

flag	descrição
-c	

flag	descrição
-n	

ln

> oof

cp

> Copia um ficheiro para outro ainda que com um ficheiro existente.

Usage:

```
$ cp /origem /destino
```

mv

> O mesmo que cp porém apaga a origem

Usage:

```
$ mv /origem /destino
```

rm

> Remove um ficheiro ou diretoria

Usage:

```
$ rm /OLD
```

Opções:

flag	descrição
-i	Informa o utilizador de todos os ficheiros que serão apagados após execução do programa

flag	descrição
-l	No pior dos casos
-f	Nunca

umask

file

wc

> Diz quantos '\n' existem num ficheiro

split

grep

> Dado um input efetuar pesquisas sobre o conteúdo do mesmo

Usage:

```
$ cat a.txt | grep "texto"
```

sort

diff

compress

uncompress

chgrp

chown

df

du

mount

unmount

2.1.2.1 Variáveis e Operadores

Existem três tipos de variáveis – ou parâmetros – da shell: parâmetros chave, parâmetros posicionais e parâmetros especiais. Um parâmetros chave começa com uma letra ou '_'(underscore), ao que se pode seguir uma cadeia alfanumérica. A atribuição de valores a variáveis consegue-se com '=', como se exemplifica: valor=5. As variáveis posicionais estão associadas aos argumentos do script: a variável $0 contém o nome do script, sendo os demais argumentos atribuídos às variáveis $1, $2, . . . As variáveis ou parâmetros especiais são dotados de um significado específico. Indique o parâmetro especial que:

- representa o número de argumentos passados ao programa: $#
- referencia todos os parâmetros posicionais: $
- representa o nome do programa ou script a ser executado: $0
- representa o número do processo a ser executado: $$
- representa o número do último processo executado em segundo plano: $!
- representa o estado do último comando não executado em segundo plano: $?

O caracter '#' indica o início de um comentário. A sequência de caracteres '#!"seguida do caminho completo de um dos interpretadores de comandos indica qual o que deve ser usado para o script em causa. Apresente, em cada um dos seguintes casos, os operadores lógicos, | Operação| operador| |---- |---- | |AND | -a | |OR | -o | |NOT | !exp| relacionais, | Operação| operador| |---- |---- | |menor do que | -lt | |menor ou igual a | -le | |igual a | -eq| |maior do que | -gt| |diferente | -ne| de manipulação de cadeias de caracteres (strings), | Operação| operador| |---- |---- | |iguais| *str1 == str2*| |diferentes| *str1* != *str2* | |de comprimento maior que zero| -n *str*| |de comprimento igual a zero | -z *str*| de manipulação de ficheiros, | Operação| operador| |---- |---- | |é um directório| -d

ident| |é um ficheiro regular| -f *ident*| |é um ficheiro com permissão de leitura| -r *ident*| |é um ficheiro com permissão de escrita|-w *ident*| |é um ficheiro com permissão de execução|-x *ident*| |é um ficheiro com conteúdo| -s *ident*| |é um ficheiro do utilizador actual|| |é um ficheiro do grupo actual|| |é um pipe|| |mais recente|| |mais antigo||

2.1.2.2 Estruturas de controlo de execução em "bash"

1. Apresente a sintaxe da estrutura if e um exemplo da sua utilização.

sintaxe

```
if list; then
        list;
[elif list; then
        list;] …
[else
        list;]
fi
```

exemplo

2. Apresente a sintaxe da estrutura for e um exemplo da sua utilização.

sintaxe

```
for (( expr1 ; expr2 ; expr3 )) ; do
        list ;
done
```

exemplo

3. Apresente a sintaxe da estrutura case e um exemplo da sua utilização.

sintaxe

```
case word in [ [(] pattern [ | pattern ] ... )
        list ;; ] ...
esac
```

exemplo

4. Apresente a sintaxe da estrutura while e um exemplo da sua utilização.

sintaxe

```
while list; do
        list;
done
```

exemplo

5. Apresente a sintaxe da estrutura until e um exemplo da sua utilização.

sintaxe

```
until list; do
        list;
done
```

exemplo

```
#!/bin/bash
i=1
until [ $i -gt 20 ]
do
        echo $(( i++ ))
done
```

2.1.2.3 Exemplos

1. Faça um script em bash chamado exif1 que, usando a estrutura de controlo if, verifique se um determinado directório existe ou não.

```
#!/bin/bash
[[ $0 < "1" || ! -d $1 ]] &&
{ echo "O diretório não existe"; exit 0; } ||
echo "O diretório existe"
```

2. Faça um script em bash chamado exif2 que, usando a estrutura de controlo if, verifique se um determinado ficheiro existe e tem conteúdo. Caso tal não se verifique deve criar um ficheiro novo a partir do stdin até que ocorra um .

```
#!/bin/bash
[[ ! $# = 1 ]] && { echo "usage: $0 filename"; exit 0; }
[[ ! -f $1 ]] && { touch $1; echo "$1 criado em $(pwd)"; exit 0; }
[[ -s $1 ]] && { echo "$1 existe e tem conteúdo"; } || { echo "$1 existe mas não
```

3. Faça um script em bash chamado excase1 usando a estrutura de controlo case que, caso seja escolhida a opção 1 apresente a data do sistema, opção 2 apresente a data e hora do sistema, opção 3 saia, outra opção qualquer apresente a mensagem “opções válidas [1 2 3]”.

```
#!/bin/bash
while true ; do
        read opcao
        echo $opcao
        case $opcao in
                1)
                date +%x;
                ;;
                2)
                date +"%x %T";
                ;;
                3)
                exit 0;
                ;;
                *)
                echo opções válidas [1 2 3];
                ;;
        esac;
done
```

4. Faça um script em bash chamado exfor1 usando a estrutura de controlo for que entre por 2 segundos (faça man sleep) nos directórios /bin e /etc mostrando-os ou listando-os. Por fim volte ao directório inicial.

```
#!/bin/bash
(for path in /bin /etc
do
```

```
        cd $path
        echo $(pwd)
sleep 2
done)
```

5. Implemente num script em bash chamado exwhile1 um contador usando a estrutura de controlo while com valores entre 1 e 20.

```
#!/bin/bash
i=1
while [ $i -le 20 ]
do
        echo $(( i++ ))
done
```

6. Implemente num script em bash chamado exuntil1 um contador usando a estrutura de controlo until com valores entre 1 e 20.

```
#!/bin/bash
i=1
until [ $i -gt 20 ]
do
        echo $(( i++ ))
done
```

7. Crie um script em bash chamado soma que implemente uma função para a soma de dois inteiros. Acrescente a chamada à função e a validação do número de argumentos.

```
#!/bin/bash
add () {
        return $(( $1 + $2 ))
}
if [ $# -ne 2 ] ||
[[ ! $1 =~ ^\-?[0-9]+$ ||
! $2 =~ ^\-?[0-9]+$ ]]
then
{
        echo "Usage: $0 numeroInteiro numeroInteiro"; exit 0;
}
fi
add $1 $2
echo $?
```

2.2 Segunda Parte: Exercícios Propostos

1. Implemente num script em bash chamado exfor2 um contador usando a estrutura de controlo for com valores entre 1 e 20. Altere depois esse script para que sejam apresentados apenas os números ímpares e os limites inferior e superior do contador sejam pedidos ao utilizador. Inclua a validação dos valores recebidos do utilizador e, na presença de erros, a emissão das mensagens apropriadas.

```
#!/bin/bash
if [ $# -ne 2 ] || [[ ! $1 =~ ^\-?[0-9]+$  ||  ! $2 =~ ^\-?[0-9]+$ ]]
then
        echo "Apresenta os números ímpares desde limiteMínimo até limiteMáximo";
        echo "Usage: $0 limiteMínimo limiteMáximo" ; exit 0 ;
fi
for ((i=$1 ; $i <= $2 ; i++ )) ; do
        rem=$(( i % 2))
        [[ $rem -ne 0 ]] && { echo $i; }
done
```

2. Implemente uma função que calcule o dobro de um número. Acrescente as validações necessárias para verificar que a função é invocada com um argumento numérico.

```
#!/bin/bash
function dobro () {
        val=$1
        echo $(( $val * 2 ))
}

read num
while [[ ! $num =~ ^[0-9]+$ ]]
do
        echo "Insira um numero inteiro positivo";
        read num
done
dobra=$(dobro ${num})
echo $dobra
```

3. Implemente o cálculo do factorial de forma iterativa e recursiva.

```
#!/bin/bash
function fatorial () {
```

```
        local v=$1
        local acum=1
        for (( i=2 ; $i <= $1; i++)) ; do
                acum=$(( $acum * i ))
        done
        echo $acum
}
num=-1
while [[ ! $num =~ ^[0-9]+$ ]]
do
        echo "Insira um numero inteiro positivo";
read num
done

val=$(fatorial $num)
echo $val


#!/bin/bash
function fatorial () {
        local v=$1
        [[ $v -eq 0 || $v -eq 1 ]] && { echo 1; } || { echo $((v * $(fatorial $v
}
num=-1
while [[ ! $num =~ ^[0-9]+$ ]]
do
        echo "Insira um numero inteiro positivo";
read num
done

val=$(fatorial $num)
echo $val
```

4. Entre no servidor e crie um ficheiro na sua área fazendo last -f /var/log/wtmp > teste.wtmp Esse ficheiro contém o historial de logins e logouts do servidor. Pretende-se saber, para cada utilizador, quantas vezes entrou no sistema e o tempo total de utilização da máquina. Caso o utilizador ainda não tenha saído, conte o tempo até ao momento presente. Para esse fim, construa um script em bash.

Terceira Parte: Questões gerais e exercícios adicionais

1. O que é uma shell ?
2. Identifique três programas de shell.
3. Por omissão, há três ficheiros standard sempre abertos. Quais e qual o seu objectivo?
4. Quando é que se classifica um comando como simples, encadeado ou composto?
5. Se numa sessão quiser mudar de shell, como o pode fazer?
6. Sabendo que em '/usr/bin' coexistem programas executáveis e scripts, quantos são os scripts? E os executáveis? E que outro(s) tipo(s) de ficheiros existem?
7. Implemente um script em bash chamado testa_entradas.sh que avalie se a chamada foi feita com argumentos.

```
#!/bin/bash
[[ $# -le 0 ]] && {
        echo "sem argumentos";
        exit 0 ;
} ||
{
        msg="$# argumento" ;
        [[ $# -eq 1 ]] ||
        {
                msg2="s";
                msg="$msg$msg2"
        }
        echo $msg ;
}
```

8. Faça scripts em bash para: -- Listar os ficheiros com uma extensão definida pelo utilizador. Identificar os directórios cujo tamanho seja superior a uma dada dimensão, especificada pelo utilizador como argumento do script, e comprimi-los.

```
#!/bin/bash
[[ ! $# -eq 1 ]] && { echo "usage:$0 \"termo pesquisa\"" ; exit 0; }
ls | ( grep ".*$1" )
```

```
#!/bin/bash
```

-- Listar os ficheiros cuja data esteja compreendida num intervalo de tempo definido pelo utilizador.

-- Listar o login e nome dos utilizadores do sistema.

-- Listar os ficheiros que contenham uma dada expressão como, por exemplo “Sistemas Operativos”

```
#!/bin/bash
[[ ! $# -eq 1 ]] && { echo "usage:$0 \"termo pesquisa\"" ; exit 0; }
ls | ( grep "$1" )
```