Bash скриптове

1. Въведение

Всички UNIX и Linux системи позволяват писането на програми които да се изпълняват (интерпретират) от терминала. Тези програми могат да са на различни езици, включително и на езици от по-високо ниво като Perl, Ruby, Rexx, C, Java и др. Най-често използваните обвивки са следните:

- Bash (Bourne Again Shell)
- sh (Bourne Shell – това е най-старата от всички)
- csh (C Shell)
- tcsh (Tenex C shell)
- ksh (Korn Shell)
- zsh (Z shell)
- ash (Almquist shell)
- rsh (Remote shell)

Нека изпълним от терминала следните редове един след друг:

```
a=vision
b=”tele$a”
echo␣$b
```

Това ще изведе стринга `television`. Чрез знака `$` се извлича стойността на дадена променлива.
За удобство със символа ␣ ще означим интервала, тъй като неговото произволно използване може да бъде интерпретирано погрешно. Например:

x=10

ще означава, че на променливата x се присвоява целочислената стойност 10, докато

x= ␣10

се разбира, че bash трябва да интерпретира командата „x=“ с аргумент 10, а в някои случаи, че x присвоява стринга „␣10“, което няма да е коректно при използването му в някой брояч.

Нека се върнем към предния пример. Вместо да пишем програмата в терминала е по удобно да се изпълни от файл. Може да ги запишем във файл с името script, чрез подходящ текстов редактор:

```
#!/bin/bash
a=vision
b=”tele$a”
echo␣$b
```

Всички редове които започват със знака # означават коментари, докато редовете с #! означават директива. За този случай директивата показва, че следващите редове от файла ще се изпълняват от програмата bash. За да направим файла изпълним е необходимо да дигнем правото за изпълнение:

```
chmod␣u+x␣script
```

Този текстов файл се нарича скрипт за обвивката и може да се изпълни директно от терминала, стига да се намираме в същата директория, чрез:

```
./script
```

Файла може да се изпълни и чрез:

```
sh␣script
```

Ако искаме програмата Hello world ще използваме следните редове:

```
#!/bin/bash
echo␣"Hello, World"
```

==========

2. Променливи

Променливите в shell скриптовете не са строго типизирани. Типа се определя при инициализация или изрично при деклариране. Имената на променливите е прието да се пишат без използването на служебни символи.

За присвояване на стойност на променлива се използва оператора за присвояване (знака за равенство (=)). Пример:

```
x=10
```

Отново да обърнем внимание, че не трябва да се използват интервали около оператора за присвояване. Ако сложим интервал след него (x=␣10), bash ще се опита да изпълни команда "x=" с аргумент 10. Извличането на стойност на променлива става чрез знака за долар ($). Стойностите на променливите могат да бъдат произволни символи, но ако се използват символи, които се използват от обвивката, възникват проблеми. Тези символи са: интервал, точка (.), знак за долар ($), по-голямо (>) и по-малко (<), (|), (&), (*), ({) и (}). Тези знаци могат да се използват като стойност на променлива ако са поставени в двойни или единични кавички или с обратно наклонени черти (escape). С двойни кавички може да се използват всички запазени знаци без знака за долар ($). За да се включи в стойността на променливата може да се използва обратно наклонена черта преди него (\$) или да оградите

стойността на променливата с единични кавички. Следващия пример илюстрира всичко описано до сега.

```
#!/bin/bash
num1=10
num2=20
msg1="$num1 < $num2 & $num2 > $num1" # Тук ще се покажат всички
#знаци без знака за долар,
# който използваме за да се обърнем към стойността на
# променливата.
echo $msg1

msg2="\$100 > \$10" # Тук използваме обратно наклонена черта за да
#покажем знака за долар
echo $msg2

msg3='Here we can use all of these symbols: ".", ">", "<", "|", "$",
etc'
# по този начин всички знаци ще се покажат на екрана, единственият
#символ, който не може да се използва е единична кавичка (')

echo $msg3

# При всички примери по-горе не е нужно стрингът да се слага в
#променлива.

echo ""\$msg1" is not needed to print '$num1 < $num2 & $num2 >
$num1'"
```

Изпълняваме скрипта от файла с
```
./script
```

Ако искаме да присвоим резултата от дадена команда на променлива, трябва да се огради командата в обратно (ляво) наклонени апострофи.

Примерно:

```
s=’ls␣/home/s503’
echo␣$s
```

ще изведе просто стринга заграден в единичните кавички, но ако сложим стринга в обратно наклонени апострофи:

```
s=`ls␣/home/s503`
echo␣$s
```

тогава резултата от командата ls ще се присвои на променливата. В случая единичните кавички могат да се използват като друго име на командата:

```
s='ls␣/home/s503'
$s
```

Тук липсва `echo` и извикването само на `s` ще бъде резултата от командата `ls␣/home/s503`.
Ако искаме да използваме резултата от дадена команда в стринг е необходимо да се загради командата в скоби и да се постави знака за долар преди тях `($(ls))`.

```
#!/bin/bash
echo "The date is $(date)"
```

Ако се изпълни друг скрипт от текущо изпълняващият се, текущият скрипт спира изпълнението си и предава контрола на другия. След изпълнението му се продължава първия скрипт. При този случай всички променливи, дефинирани в първия скрипт не могат да се използват във втория. Но ако те се експортират, променливите от първия те ще могат да се използват във втория. Това става с командата `export`.

```
#!/bin/bash
# Това е първият файл.
var=100
export var
./script-a
# EOF
```

```
#!/bin/bash
# Вторият файл.
echo "The value of var is $var"
# EOF
```

Изпълнението на скрипта (обърнете внимание, че вторият файл трябва да е `script-a`, ако използвате друго име, сменете името на файла в първия скрипт).
Стартираме, чрез:

```
./script
```

Друг начин за деклариране на променливи е командата `declare`. Синтаксисът на тази команда е:

```
declare -тип име-на-променливата
```

Типовете променливи са:

```
-r   - readonly
-i   - integer (цяло число)
-a   - array (масив)
-x   - export
```

Пример:

```
#!/bin/bash
declare -i var # декларираме променлива от тип integer
var=100
echo $var
declare -r var2=123.456 # декларираме readonly променлива
echo $var2
var2=121.343 # опитваме се да променим стойността на var2
echo $var2   # стойността на var2 ще е все още 123.456
```

Изпълнението на скрипта:

```
./script
```

Повече информация можете да видите в bash manual pages (man bash).

==========

3. Shell позиционни променливи

Когато се изпълнява дадена команда от терминала, тя може да включва аргументи, които се записват след нея, разделени с интервал. Тъй като командата е програма, тези аргументи се достъпват от нея. Наричат се позиционни променливи или аргументи на скрипта. Стойностите са от 1 до 9 и могат да се използват чрез `$1, $2, … ,$9`.

Например за командата:

```
command␣arg1␣arg2␣arg3
```

позиционните променливи ще имат съответните стойности:

`$0` – е името на командата или command

`$1 – arg1`

`$2 – arg2`

`$3 – arg3`

Освен позиционните променливи има и специални променливи, които се използват за аргументите на скрипта за обвивката:

`$*` - всички аргументи от командния ред

`$@` - всички аргументи от командния ред поотделно

`$#` - броят на аргументите от командния ред

Разликата между `$*` и `$@` се състои в това, че при `S*` аргументите се вземат заедно като един стринг, докато при `$@` те са поотделно и могат да се ползват като масив.

Нека разгледаме следния пример, чрез който да илюстрираме използването на позиционните променливи:

```
#!/bin/bash
echo "The first argument is $1, the second $2"
echo "All arguments you entered: $*"
echo "There are $# arguments"
```

И да изпълним скрипта чрез следните аргументи от терминала:

./script␣arg1␣arg2␣arg3␣arg4

Да разгледаме още един пример. Да се напише скрипт който преброява файловете в дадена директория, посочена като аргумент.

```
#!/bin/bash
ls $1 | wc –w
```

съответно скрипта трябва да стартираме по следния начин:

```
./script /etc
```

Този пример не пълен, тъй като е коректно да се провери дали подадения аргумент е директория. Ако не сме подали директория, командата ls ще върне съобщение за грешка и след това (по конвейр), командата `wc –w` ще преброи думите в него.

В началото на тази точка споменахме, че позиционните променливи с които можем да работим са от 0 до 9. Какво се получава когато техният брой е по-голям. Стойности на позиционните променливи можем да зададем чрез командата set. Например

```
set `ls  /etc`
```

ще бъдат зададени толкова позиционни променливи колкото резултата върне командата `ls`, и съответно за директорията `/etc` техния брой ще е много повече от 9. В този случай се използва преместване на ляво чрез командата shift за да можем да изчетем всички променливи. Да обърнем внимание, че тук се използват отново обратно наклонени апострофи.

```
#!/bin/bash
set `ls /etc`  #задаваме стойности на позиционните променливи
#в случая техния брой се определя от броя на файловете и
#директориите в /etc
echo $1 $2 $3  #извеждаме първите 3
echo $#        #извеждаме техния брой
echo $*  #извеждаме всички
echo "SHIFT"
shift 2  #изместваме на ляво с две позиции
#отново извеждаме
echo $1 $2 $3
echo $#
echo $*
```

Изпълняваме с

```
./script
```

Ще забележим, че след изместването стойността на `$1` е предишната на `$3`, `$2` на `$4` и т.н. Старите стойности за `$1` и `$2` няма да се пазят и броя на всички `($#)` ще е с две по-малък.

4. Аритметични операции

За въвеждане от клавиатурата се използва командата `read`.

```
#!/bin/bash
echo -n "Enter a string: "
read str
echo "String you entered: $str"
```

За изчисление на аритметични операции се използват командите `let` и `expr`.

Чрез `let` могат да се сравняват две стойности на променливи, да се извършват аритметични операции, както и да се използват за управляващи конструкции на цикли. Синтаксисът на командата е следния:

```
let value1 operator value2
```

Възможните оператори, които могат да се използват са:

```
+     - събиране
-     - изваждане
*     - умножение
/     - деление
%     - остатък при деление
```

```
>     - по-голямо
<     - по-малко
>=    - по-голямо или равно
<=    - по-малко или равно
==    - равно
!=    - различно
&     - логическо И (AND)
|     - логическо ИЛИ (OR)
!     - логическо НЕ (NOT)
```

Тази команда има и втори запис, като се използват двойни скоби:

```
$ (( value1 operator value2 ))
```

Когато променливите са от един и същи тип, аритметичните операции могат да се използват директно без командата `let` (или без скобите).

Пример:

```
#!/bin/bash
echo -n "Enter the first number: "
read var1
echo -n "Enter the second: "
read var2
declare -i var3

echo ----------
echo "$var1 + $var2 = $(( $var1+$var2 ))" # за да се изчисли израза
#в двойните скоби, трябва да се сложи знака $
let res=$var1*var2
echo "$var1 * $var2 = $res"
var3=100
var3=$var3+10 # var3 е декларирана като integer и не е нужно да се
#използва let
echo "$var3"
```

Командата `expr` има подобно действие като `let`. Нейния синтаксис е следния:

```
expr value1 ‘operator’ value2
```

Допустими са следните операции: `+` (събиране), `-` (изваждане), `*` (умножение), `/` (деление), `%` (модул), като не е задължително да се слагат в единични кавички. Оператора за умножение трябва да се използва с наклонена черта `\*`. По същия начин се използват и скобите за по-сложни изрази `\(` и `\)`. Възможно е да се съставят и логически изрази, като се използват знаците: `=`, `!=`, `<`, `>`, `<=` и `>=`.
Примери:

```
expr␣5␣+␣2
expr␣5␣\*␣3
expr␣8␣/␣2
expr␣5␣\*␣\(␣2␣+␣3␣\)
expr␣7␣\!=␣5
expr␣3␣\<␣5
```

Командата `expr` може да се използва с някои функции, например за стрингове:

```
expr␣length “Some string”
expr␣index␣”Some string”␣”m”
```

Характерното за командата `expr` е, че тя извежда резултата в терминала, затова ако искаме да го присвоим на променлива трябва да се използват обратно наклонените апострофи:

```
a=5
b=3
i=`expr␣$a␣+␣$b`
echo␣$i
```

5. Условен оператор

Първото нещо което трябва да се разгледа това е какво представлява едно условие. Ако използваме оператора if (който го има в повечето езици) и го комбинираме с друга команда като условие, не е ясно как ще сработи. Например:

if ls

винаги ще е вярно, тъй като дори да не съществува директория, командата ls ще върне стринг за грешка, но exit кода (по късно ще бъде разгледан) ще е 0, което означава, че тя е завършила правилно. За тази цел, не само за оператора if, но и за други оператори като цикли, за условие се използва командата test. Тя има специфичен синтаксис и е команда която изследва. Командата test проверява различни условия и връща тяхната логическа стойност: 0(true), 1(false), >1(error) като exit код. Командата има вида:

```
test израз
```

или втория вариант:

```
[␣израз␣]
```

където *израз* може да включва променливи и константи, разделени със знаци за операции.

Има три класа знаци: знаци за отношения, знаци за логически операции и знаци за файлове.

Примерът

test␣-f␣fname

проверява дали посоченото име fname е име на файл, с -d за директория и други. Ако използваме съкратения синтаксис ще изглежда така:

[␣-f␣fname␣]

Командата test за отношение между две променливи се използва по следния начин:

```
test value1 -option value2
```

```
test string operator string
```

или чрез съкратения запис:

```
[␣value1 -option value2␣]
```

```
[␣string operator string␣]
```

Обръщаме внимание, че при сравнение на стрингове се използва оператор, а не опция. Когато се сравняват променливи трябва да се взима тяхната стойност (чрез използването на знака $):

[␣$a␣-gt␣5␣]

Ще изброим само част от използваните операции за test:

=	еднакви низове
!=	различни низове
-z	нулева дължина на низ
-n	ненулева дължина на низ
-eq	равно (за цели числа)
-ne	неравно (за цели числа)
-gt	> (за цели числа)
-lt	< (за цели числа)
-ge	>= (за цели числа)
-le	<= (за цели числа)
-a	логическо И
-o	логическо ИЛИ
!	логическо НЕ
-f	обикновен файл
-d	директория
-w	файл с разширение за запис
-r	файл с разширение за четене
-x	файл с разширение за изпълнение

Останалите опции могат да се проверят чрез командата man в терминала:

```
man test
```

В началото на този точка споменахме за exit кода. При изпълнение програмите често могат да настъпят грешки. Всеки път, когато една програма завърши, се връща код за завършване

наречен exit код. По подразбиране стойността 0 означава коректно завършване, а при грешка кода е различен от 0. Този код може да се провери, чрез стойността на специална променлива $?, която съдържа кода на завършване на последната команда. Тази променлива също може да се използва за проверка на резултата от командата test.

```
a=10
b="asdfg"
[ $a -eq 10 ]
echo $?   # exit code = 0

[ $b = "aaaaaa" ]
echo $? # exit code = 1
```

Условения оператор if има следния синтаксис:

```
if  условие
      then
            оператор1
      else
            оператор2
fi
```

Оператора if проверява exit кода на дадена команда, но най-често се използва командата test за условие. В секциите then и else може да се съдържат повече от един оператори.

Когато в else има следващ оператор if се използва ключовата дума elif:

```
if  условие
      then
            оператор1
      elif условие
            then
                  оператор2
            else
                  оператор3
fi
```

Може да запишем няколко оператора на един ред като използваме знака (;)

```
if  условие; then
      оператор1
      else
            оператор2
fi
```

Примери:

==========

Да се напише скрипт който проверява дали подаденото му име е име на файл или на директория:

```
#!/bin/bash
if␣[␣-f␣$1␣]
     then
          echo “$1 is a file”
     elif␣[␣-d␣$1␣]
          then
               echo “$1 is a dir”
          else
               echo “$1 is not a file or dir.”
fi
```

Стартираме скрипта с подходящ параметър:

```
./script  /etc

./script  /etc/passwd
```

==========

```
#!/bin/bash
echo -n "Enter a string: "
read str1
echo -n "Enter a string: "
read str2
echo -n "Enter a number: "
read num1

if [ $str1 == "asdfg" ]; then
     echo "str1 = asdfg "
elif [ $str1 == "asdfg" ] && [ $str2 == "asdfg" ]; then
     echo "str1 and str2 = asdfg"
     else
          echo "str1 and str2 != asdfg"
fi

if [ -f "/etc/passwd" ]; then
     cat /etc/passwd
fi
```

```
if [ $num1 -eq 10 ]; then
      echo "num1 = 10"
elif [ $num1 -gt 100 ]; then
      echo "num1 > 100"
else
      echo "?!?"
fi
```

======

Тук се проверява резултата от дадена команда (в случая whoami) дали е равен на определен стринг. Обръщаме внимание, че се използват обратно наклонени апострофи (`).

```
if  [ `whoami` = “root” ]
      then  echo  “Hello, admin”
      elif [ `whoami` = “s503” ]
            then echo  “Hello, s503”
      else   echo “Hello, guest in  `pwd` ”
fi
```

====

Освен създаването на условия, чрез командата test е възможно използването на условни или аритметични изрази. Те могат да заместят условията за операторите if, while и until. Условните изрази се затварят в двойни квадратни скоби, като преди и след тях има празни интервали, например (за оператора if):

```
if [[␣условен израз␣]]
```

Условните изрази използват синтаксиса на условия чрез test, но те включват и логическите операции:

`&&` - И

`||` -ИЛИ

`!` -Отрицание

Пример:

```
if [[  $x  -gt  3   &&   &x  -lt   42  ]]
```

ще има стойност true , ако стойността на променливата x е между 3 и 42.

За да използваме аритеметични изрази като условия, трябва да ги заградим с двойни кръгли скоби:

```
if ((i>10))
```

6. Оператори за цикли

Използват се циклите: **while**, **until**, for (**for-in**). Операторите while и until проверяват exit кода на дадена команда, но както и при оператора if се използва резултата от test като условие. Цикъла for-in се използва за обхождане на списък от стойности (или масив).

Цикъл while – изпълни докато условието е 0 (true):

```
while условие
      do
            оператори (или команди)
      done
```

Пример – въвеждаме числа, докато са по-големи от 5:

```
#!/bin/bash
i=10
while␣[␣$i␣-gt␣5␣]
      do
            read i
            echo "You enter:  " $i
      done
```

Пример – извежда числата от 1 до 9

```
#!/bin/bash
i=1
while [␣$i␣-lt␣10␣]
      do
            echo –n "… $i"
            i=`expr $i + 1`
      done
echo " "
```

Цикъл until – изпълни докато условието е 1 (false):

```
until условие
      do
            оператори (или команди)
      done
```

Пример – въвеждаме стринг, докато не въведем стринга “no”:

```
#!/bin/bash
answer=yes
until [ $answer == "no" ]; do
      echo -n "Enter a string: "
      read str1
      echo "You entered: $str1"
      echo -n "Do you want to continue? "
      read answer
done
```

Цикъл for-in

```
for управляваща-променлива  in  списък-от-стойности
      do
            оператори (или команди)
      done
```

Примери:

```
#!/bin/bash
for  k  in 1 2 3 4 5 6 7 8 9
      do
            if  [  $k  -eq   3  ];  then continue;  fi
            if  [  $k  -eq   7  ]; then break;  fi
      echo -n  ". . . $k"
      done;  echo " "
```

======

```
#!/bin/bash
for archive  in   ~/*.tar
     do
          echo $archive
     done
```

==========

Може да използваме цикъла for да обходим позиционните променливи:

```
#!/bin/bash
for  x in  $1 $2 $3 $4
     do
          echo $x
     done
```

Но това може да се замести със служебната променлива $@ която взима всички позиционни променливи поотделно:

```
#!/bin/bash
for x in $@
     do
          echo $x
     done
```

Примера може да се стартира:

```
./script  arg1 arg2 arg3 arg4
```

==========

Да разгледаме още един пример, при който се извеждат всички файлове от дадена директория:

```
#!/bin/bash
set  `ls $1`
for a in $@
     do
          echo $a
     done
```

При този пример се използва командата set, чрез която се задават стойности за позиционните променливи. За да стартираме примера е необходимо да посочим аргумента $1, който е някоя директория:

```
./script   /home/s503

===============
```

Ако е необходимо да запишем резултата от изпълнението на даден цикъл във файл, може да използваме операторите > и >>.

> - изтрива предишното съдържание

>> - долепя в края

```
#!/bin/bash
nums="1 2 3 4 5 6"
for num in $nums
do
  echo $num
done > fnums
```

Или в комбинация с конвеир. В случая след приключване на цикъла се сортира резултата и се записва във файл:

```
#!/bin/bash
strings="asd  dse rdft  wert gty"
for str in $ strings
     do
          echo $str
     done | sort > fwords
```

Файловете можем да прочетем с:

```
cat fnums
cat fwords
```

7. Генериране на случайни числа

За целта се използва променливата на обвивката $RANDOM. Всеки път когато се използва се връща псевдослучайно число между 0 и 32767. Можем да променим интервала чрез аритметичните операции: %, +, -, *, /.
За да гарантираме, че променливата $RANDOM ще генерира различни последователности всеки път, трябва да я инициализираме с различни стойности. Например може да използваме стойността на date с опции +%s, която ще върне текущия брой секунди, изминали от началото на UNIX епохата (започваща в 00:00:00 ч. Гринуичко време на 1 януари 1970г.) .
Пример за скрипт който връща случайно число между 1 и 20:

```
RANDOM=`date  +%s`
let NUM=($RANDOM  %  20  +  1)
echo $NUM
```

8. Масиви

Масивите съхраняват всяка стойност в уникален, отделен, индексиран компонент, наречен елемент. Елементите се означават с индекси автоматично, като се започне от 0. Масивите в bash могат да се създават по три начина: чрез инициализиране, чрез оператора declare и чрез задаване на стойности в скоби.
Обръщението към конкретен елемент на масив се извършва чрез името на променливата и индексния номер на елемента заграден в квадратни скоби „[]“. За да се видят самите стойности, съхранени в елементите на даден масив е необходимо да се деадресират. За целта се използват фигурни скоби “{}“.

```
#!/bin/bash
array[1]=a1
array[5]=abc
array[10]=123
echo "array[1] = ${array[1]}"
echo "array[5] = ${array[5]}"
echo "array[10] = ${array[10]}"
```

Създаване на масив, чрез присвояване на стойности зададени в скоби:

```
array=(␣a␣4␣wer␣35␣t7␣)
```

В този случай
array[0] = a
array[1] = 4
array[2] = wer
array[3] = 35
array[4] = t7

Може да създадем масив и чрез командата `declare` и опция `-a`:

```
declare -a array
echo -n "Enter some numbers separated by space: "
read -a array # с опция –a въвеждаме елементите на масив, като те са
#разделени с интервал
elements=${#array[@]} #elements ще съдържа броя елементи на масива
# ${array[@]} съдържа елементите на масива поотделно. Може да се
#използва за цикъл for-in  например:
```

```
for i in ${array[@]}; do
    echo $i
done
#извеждане с цикъл while
i=0
while [ $i -lt $elements ]; do
  echo ${array[$i]}
  let "i = $i + 1"
done
echo " "
```

Примери за използване на масиви в изрази:

```
array=(2 3 4 5)
let array[2]=7
((array[3]=9))
((array[4]=array[2]+3))
array[5]=`expr ${array[0]} + ${array[1]}`
```

9. Оператор case за условни разклонения

Пример – избор на извеждане:

```
#!/bin/bash
echo -n "Enter an option (l, s or al): "
read opt
case $opt in
l)
     ls -l;;
s)
     ls -s;;
al)
     ls -al;;
*) # other cases
     ls;;
esac
```

Пример – избор на цвят:

```
#!/bin/bash
clear
echo “Change terminal font color”
echo “--------------”
```

```
echo –en “[1]Red\n  [2]Green\n [3]Brown\n”
echo –en “[4]Blue\n [5]Purple\n [6]Cyan\n”
echo “[x] Exit”
echo –n “Enter: ”
read   char
case  $char in
1)  echo  -e  “\e[033;31m”;;
2)  echo  -e  “\e[033;32m”;;
3)  echo  -e  “\e[033;33m”;;
4)  echo  -e  “\e[033;34m”;;
5)  echo  -e  “\e[033;35m”;;
6)  echo  -e  “\e[033;36m”;;
7)  echo  -e  “\e[033;37m”;;
esac
```

10. Прихващане на сигнали с trap

Понякога е необходимо да се прихващат сигнали към даден скрипт. Това може да стане, чрез trap. Командата trap има следния синтаксис:

trap команда сигнал

и указва коя команда (или функция) да се изпълни при дадения сигнал. Сигнала се задава чрез име или номер.

Пример:

```
#!/bin/bash
trap sorry INT
sorry()
{
     echo “I am sorry. I cannot do that.”
     sleep 3
}
for i  in 10  9  8  7  6  5  4  3  2  1  0;  do
     echo $i  seconds remaining
     sleep  1
done
```

При стартиране на примера на всяка секунда се отброяват последователно числата от 10 до 0. Ако се опитаме да прекъснем изпълнението чрез комбинацията CTRL + C , чрез trap се извиква функцията sorry. Вместо с името на сигнала може да се използва неговия номер. За сигнала INT (Interrupt) номера е 2 и в горния пример можехме да използваме реда :

```
trap sorry  2
```

Сигнала с номер 9 е kill и той не може да бъде прихващан от trap.

За информация останалите номера на сигнали можем да проверим с:

```
man 7 signal
```

11. Функции

```
#!/bin/bash

func() {
  echo "Hello"
}

info() {
  echo "host uptime: $( uptime )"
  echo "date: $( date )"
}
func   # извикване
info   # извикване
```

Сортировка sleep sort:

```
#!/bin/bash
f() {
      sleep “$1”
      echo “$1”
}
while   [ -n “$1” ]
      do
            f  “$1” &
            shift
      done
wait
```

Стартираме с параметри - числа които искаме да се сортират. Тази сортировка разбира се не претендира за ефективност. При нея се използва оператора sleep, който изчаква толкова секунди колкото е стойността, и затова не е удачно да въвеждаме големи стойности:

```
./sleepsort   3 1 4 1 5 9
```