

# 4

## Inputs/Outputs

---

In this chapter, you will see how Scilab integrates itself inside the computer environment. You will be given examples of simple commands executed in the console. These will help you get accustomed to using the console.

### 4.1. File system

The first concept that needs to be understood to use Scilab with ease is that of the current directory. At each moment, a directory of your file system gets associated to your work environment in Scilab. By default, this is the directory Scilab will search in to find auxiliary information during certain operations (e.g. opening files, writing to files). There exist several ways to interact with the file system:

- retrieve the name of the current directory through the `FILE` menu in the console or with the command `pwd`
- change the current directory through the `FILE` menu in the console or by using the commands `cd` or `chdir`
- create a new directory with `mkdir` and remove a directory with `rmdir`
- move, copy or delete directories straight from the console with the commands `copy-file`, `movefile` and `mdelete`

Paths for certain Scilab-specific directories are accessible from the console with the following commands:

- Scilab's installation directory can be retrieved with `SCI` (also see [Section 6.2, Installation](#)).
- The directory that stores user data (history, preferences, etc) is displayed with `SCIHOME`. This directory will change depending on the operating system used and the way the user accounts are managed. For example, the directory will be, in general:
  - `C:/Users/<User>/AppData/Roaming/Scilab/<Scilab-Version>` for Windows

- /home/<User>/.Scilab/<Scilab-Version> for Unix-type systems
- /Users/<User>/.Scilab/<Scilab-Version> for Mac OS
- A temporary directory is also assigned to each Scilab work session. It is created at the beginning of the session and destroyed at the end. Its path can be retrieved from the console with `TMPDIR`. Its name is of the form `SCI_TMP_*` and its location depends on the operating system.

Here are a few examples of directory operations that you can perform from the console:

```
-->path=pwd(); // current directory

-->cd SCI // go to the Scilab installation directory
ans =
D:\profil\Users\roux\AppData\Local\scilab-5.5.2

-->pwd // value of current directory
ans =
D:\profil\Users\roux\AppData\Local\scilab-5.5.2

-->cd contrib // go to the SCI/contrib/ directory
ans =
D:\profil\Users\roux\AppData\Local\scilab-5.5.2\contrib

-->cd '../' // "move up" to the SCI directory
ans =
D:\profil\Users\roux\AppData\Local\scilab-5.5.2

-->chdir('contrib') // go to the SCI/contrib/ directory
ans =
T

-->pwd // value of current directory
ans =
D:\profil\Users\roux\AppData\Local\scilab-5.5.2\contrib

-->chdir(TMPDIR) // go to the temporary directory
ans =
T

-->mkdir('test') // create the directory test/
ans =
1.

-->ls('te*') // list the elements starting with "te"
ans =
test

-->rmdir('test') // remove the test/ directory
ans =
1.
```

```

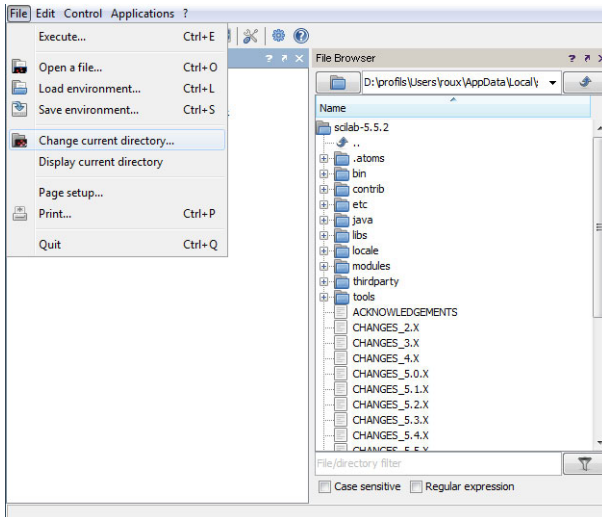
-->dir('te*') // this is the content of the current directory which is
empty []
ans =
[]

-->chdir(path) // return to the initial current directory
ans =
T

```

All these operations can also be performed via a graphics interface through the file browser (see [Figure 4.1](#)). This browser can be called from the APPLICATIONS menu from the console or with the command `filebrowser`.

**Figure 4.1 :** File browser



*Tip* > Scilab's launch shortcut can be customized to specify the startup directory. By default, this directory is usually the root directory of the user launching Scilab or the Scilab installation directory (SCI) for Windows.

## 4.2. System commands

It is possible to call system commands from Scilab. Depending on your operating system you may use the command `dos` or `unix`, but both commands work the same way! Four versions of the command `unix` process the result returned by the system in different ways:

- `unix_g` lets you redirect the output to a Scilab variable.

- `unix_w` redirects the output to the console.
- `unix_x` redirects the output to a popup window (see [Figure 4.2](#)).
- `unix_s` does not return anything.

Here are several examples of results displayed in the console:

```
-->unix('dir') // return code
ans =

    0.

-->unix_s('dir') // no output

-->unix_g('dir') // output to variable
ans =

! Volume in drive C has no label.
!
! Volume Serial Number is 3825-15B6
!
!
! Directory of C:\Program Files\scilab-5.5.0\contrib
!
!
!09/14/2014 02:05 PM <DIR> .
!
!09/14/2014 02:05 PM <DIR> ..
!
!04/11/2014 02:03 AM          119 loader.sce
!
!09/14/2014 02:05 PM <DIR>      toolbox_skeleton
!
!09/14/2014 02:05 PM <DIR>      xcoss_toolbox_skeleton
!
!
!           1 File(s)           119 bytes
!
!           4 Dir(s) 364,507,512,832 bytes free
!

-->unix_w('dir') // output to console
Volume in drive C has no label.
Volume Serial Number is 3825-15B6

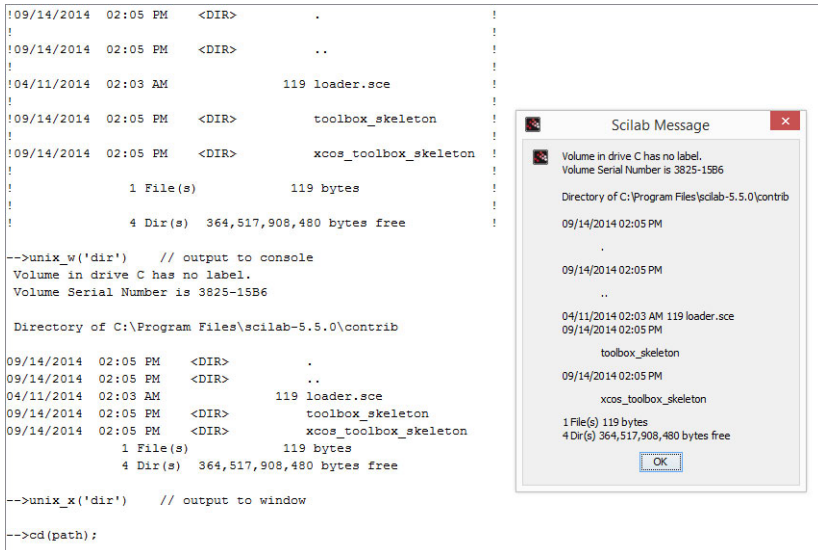
Directory of C:\Program Files\scilab-5.5.0\contrib

09/14/2014 02:05 PM <DIR> .
09/14/2014 02:05 PM <DIR> ..
04/11/2014 02:03 AM          119 loader.sce
09/14/2014 02:05 PM <DIR>      toolbox_skeleton
09/14/2014 02:05 PM <DIR>      xcoss_toolbox_skeleton
           1 File(s)           119 bytes
           4 Dir(s) 364,506,988,544 bytes free
```

```
-->unix_x('dir') // output to window
-->cd(path);
```

**Caution** > The *unix* command returns an integer code which depends on the given result.

**Figure 4.2** : Example of popup window returned by *unix\_x*



```
09/14/2014 02:05 PM <DIR>      .           !
!
!09/14/2014 02:05 PM <DIR>      ..          !
!
!04/11/2014 02:03 AM           119 loader.sce !
!
!09/14/2014 02:05 PM <DIR>      toolbox_skeleton !
!
!09/14/2014 02:05 PM <DIR>      xcoss_toolbox_skeleton !
!
!           1 File(s)           119 bytes      !
!
!           4 Dir(s) 364,517,908,480 bytes free !
!
-->unix_w('dir') // output to console
Volume in drive C has no label.
Volume Serial Number is 3825-15B6

Directory of C:\Program Files\scilab-5.5.0\contrib

09/14/2014 02:05 PM <DIR>      .           .
09/14/2014 02:05 PM <DIR>      ..          ..
04/11/2014 02:03 AM           119 loader.sce loader.sce
09/14/2014 02:05 PM <DIR>      toolbox_skeleton toolbox_skeleton
09/14/2014 02:05 PM <DIR>      xcoss_toolbox_skeleton xcoss_toolbox_skeleton
           1 File(s)           119 bytes
           4 Dir(s) 364,517,908,480 bytes free

-->unix_x('dir') // output to window
-->cd(path);
```

To retrieve information related to the operating system that's running Scilab, use the command *getos*. Similarly, the command *getversion* tells you which Scilab version is being used. More broadly, an operating system variable can be retrieved with the command *getenv*.

```
-->getversion() // Scilab version
ans =
scilab-5.5.2

-->getos() // os windows
ans =
Windows

-->getenv('TMP') // retrieve the environment variable TMP
ans =
D:\profil\Users\roux\AppData\Local\Temp
```

Moreover, you can interact with the clipboard by using `clipboard`.

```
-->clipboard("copy","test") // CTRL+C the text "test"
ans =
 []

-->clipboard("paste") // CTRL+V
ans =
 test
```

### 4.3. CPU dates and times

For certain applications, you may need to access information related to time. Different commands can be used to evaluate durations. You can:

- calculate the CPU time (= number of processor cycles) elapsed between two operations by using `timer`
- calculate the real elapsed time in milliseconds between two actions with the command `tic/toc`, which starts/stops the Scilab timer
- pause Scilab for a certain duration with `sleep(time in milliseconds)` or `xpause(time in microseconds)`
- perform real-time simulations with `realtimeinit` (which allows you to set the time unit) and `realtime`. The first call to `realtime` sets the current date origin and subsequent calls force Scilab to wait until a specified date has passed to carry on.

Test these different commands with the following examples:

```
-->// time with tic() and toc()

-->tic()

-->sleep(1000) // 1000ms=1second

-->toc()
ans =
 1.005

-->tic()

-->xpause(200000) // 200000micros=0.2 seconds

-->toc()
ans =
 0.206
```

```

-->// CPU time with timer

-->timer();

-->sleep(1000)    // 1000ms=1second

-->timer()
ans =
    0.0468003

-->timer();

-->xpause(1000000)    // 1000000micros=1 second

-->timer()
ans =
    0.0312002

-->//real time

-->realtimeinit(1)    // time unit of 1 second

-->realtime(0)    // sets current date to t=0

-->tic()

-->realtime(2)    // wait for date t=2

-->toc()    // the timer will display 2 secondes
ans =
    2.003

```

You can perform calculations using dates, but be careful, numerous formats can be used to manage dates and as many functions exist to manipulate them.

- `clock` retrieves the date as a vector with six parameters [`year`, `month`, `day`, `hour`, `minute`, `second`].
- `datenum` retrieves a date in the form of the number of days elapsed since January 1st of year zero.
- `getdate` retrieves a date as a timestamp (number of seconds elapsed since January 1st 1970) or as a vector of ten parameters [`year`, `month`, `week`, `Julian day`, `day of the week`, `day of the month`, `hour`, `minute`, `second`, `millisecond`] (more complex than the `clock` outputs!).

Once the dates are retrieved, they can be processed by using other commands:

- `datevec` converts a timestamp into a vector corresponding to the appropriate date for Scilab.
- `weekday` computes the day of the week corresponding to a timestamp.

- `eomday` computes the last day of a certain month for a given year.
- `etime` calculates the difference (in seconds) between two dates given by a vector of six parameters.

Finally, the following functions display dates or calendars:

- `date` retrieves a date as a character string.
- `calendar` displays a monthly or annual calendar.

*Caution* > For the function `weekday`, the first day of the week is Sunday, whereas for the function `calendar`, the first day of the week is Monday. If no arguments are specified, these functions return the values corresponding to the current date. Otherwise, they return results for the dates specified as parameters.

Here are a few examples of use of the previous functions:

```
-->calendar(1970,1) // Jan. 1970 calendar
ans =

ans(1)

Jan 1970

ans(2)

M      Tu      W      Th      F      Sat      Sun

ans(3)

0.      0.      0.      1.      2.      3.      4.
5.      6.      7.      8.      9.      10.     11.
12.     13.     14.     15.     16.     17.     18.
19.     20.     21.     22.     23.     24.     25.
26.     27.     28.     29.     30.     31.     0.
0.      0.      0.      0.      0.      0.      0.

-->eomday(2012,2) // last day of February 2012
ans =
29.

-->d1=[1970 1 1 0 0 0] // Scilab date format
d1 =
1970.    1.    1.    0.    0.    0.

-->t1=datenum(d1) // serial date number for date d1
t1 =
719529.

-->[N,S]=weekday(t1) // day of the week for date d1
S =
Thu
```



```

N =
  5.

-->date() // current date
ans =
  16-Jun-2015

-->d2=clock() // scilab vector for current date
d2 =
  2015.  6.  16.  21.  11.  18.000004

-->t2=datetime(d2) // serial date number of date d2
t2 =
  736131.88

-->datevec(t2) // date corresponding to t2
ans =
  2015.  6.  16.  21.  11.  18.000004

-->etime(d1,d2) // difference between dates d1 and d2
ans =
  - 1.434D+09

-->d=etime(d2,d1) // difference between dates d2 and d1
d =
  1.434D+09

-->getdate(d) // day occurring d seconds after Jan. 1st 1970
ans =
  2015.  6.  25.  167.  3.  16.  23.  11.  18.
  0.0000038

```

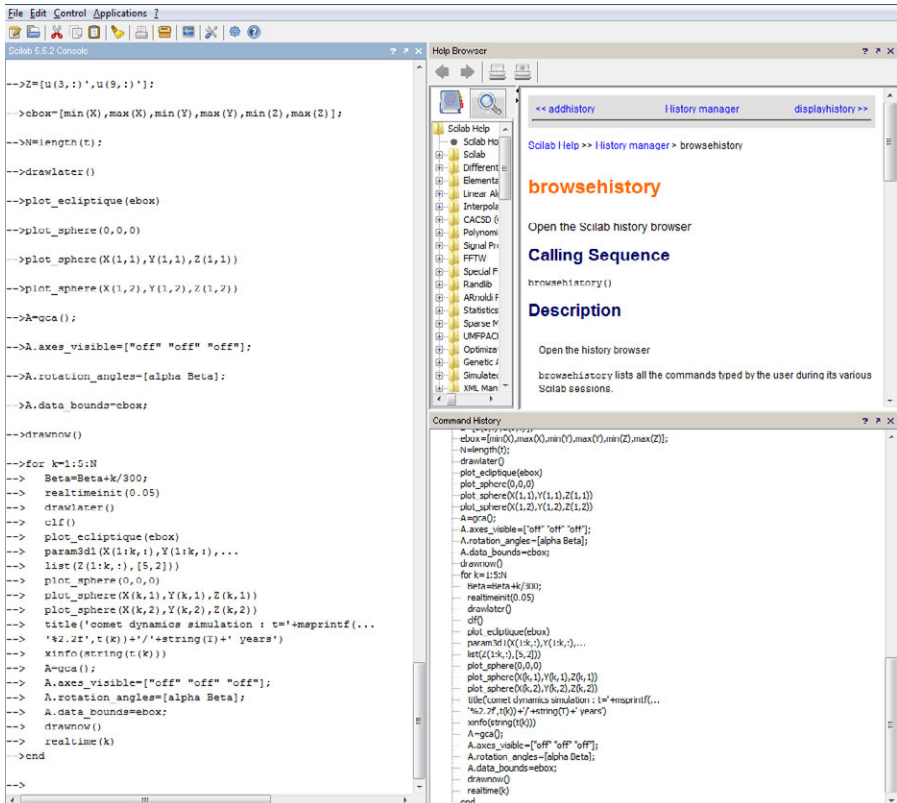
## 4.4. Command history

Scilab provides a command history browser which automatically records commands sent to the console. This capability helps speed up the script writing process. With the help of the following functions, you will be able to:

- clear the command history with `resethistory`
- save it to a file with `savehistory`, reload it in the Scilab environment with `loadhistory`
- modify entries in the history with `addhistory` and `removelinehistory`
- retrieve and store it in a variable with `gethistory`
- display the command history within the browser with `browserhistory` or within the console with `displayhistory`

You can also use the history browser (see [Figure 4.3](#)) if you prefer to use a graphical interface to perform the operations described above.

Figure 4.3 : The history browser



Finally, the function `diary` lets the user store anything displayed in the console inside a text file. Commands, along with their results, if applicable, are recorded between two calls to the function `diary`.

```

-->path=pwd(); // current directory
-->id=diary('scilab-base-diary.txt') // open the diary
id =
  2.
-->cd TMPDIR // temporary Scilab directory
ans =
  D:\profil\Users\roux\AppData\Local\Temp\SCI_TMP_6184_
-->resethistory() // erase history log

```

```

-->addhistory('ls')           // add a line to the history log

-->gethistory()               // retrieve the command history inside a
variable
ans =
!// -- 16/06/2015 21:11:16 -- // !
!                               !
!ls                             !

-->displayhistory()          // display history log
0 : // -- 16/06/2015 21:11:16 -- //
1 : ls

-->savehistory('essai.txt')  // save the history inside a file

-->dir('essai.txt')          // the file is created inside the current
directory
ans =
essai.txt

-->browsehistory()           // open the history browser

-->cd(path);                 // return to the initial directory

-->diary(id,'close')         // close the diary

```

**Caution** > When a Scilab program is launched from a saved file (like the ones shown in the chapter [Preview of Scilab](#)) by using graphical interface shortcuts (see the video in [Figure 3.5](#)), the commands executed, including the `exec` command that appears in the console, are left out of the history log.