Chapter 1

Zero Configuration Scripts and Command-Line Handlers

with the participation of:
Camillo Bruni *(camillobruni@gmail.com)*

Weren't you fed up not be able to install Pharo from a single command line or to pass it arguments? Using a nice debugger and an interactive environment development does not mean that Pharo developers do not value automatic scripts and love the command line. Yes we do and we want the best of both worlds! We really wanted it to free our mind of retaining arbitrary information. A zero configuration is a script that automatically downloads everything you need to get started. Since version 2.0, Pharo also supports a way to define and handle command line arguments.

This chapter shows how to get the zeroconf scripts for Pharo as well as how you can pass arguments to the environment from the command-line.

1.1 Getting the VM and the Image

First here is a way to download a zero configuration script to download the latest 2.0 Pharo image and vm.

```
wget get.pharo.org/20+vm
```

If you do not have `wget` installed you can use `curl -L` instead.

To execute the script that we just downloaded, you should change its permissions using chmod a+x or invoke it via bash as follows.

Configurations. There is a plethora of configurations available. The URL for each script can be easily built from an image version and a vm following the expression: get.pharo.org/$IMAGE+$VM

Possible values for $IMAGE are: 12 13 14 20 30 stable alpha

Possible values for $VM are: vm vmS vmLatest vmSLatest

Of course, one can just download an image as well get.pharo.org/$IMAGE or just the VM get.pharo.org/$VM

Looking at the help. Now let's have a look at the script help.

```
bash 20+vm --help
```

The help says that the 20+vm command downloads the current virtual machine and puts it into the pharo-vm folder. In addition, it creates several scripts: pharo to launch the system, pharo-ui a script to launch the image in UI mode. Finally, it also downloads the latest image and changes files.

```
This script downloads the latest Pharo 20 Image.
This script downloads the latest Pharo VM.

The following artifacts are created:
   Pharo.changes  A changes file for the Pharo Image
   Pharo.image    A Pharo image, to be opened with the Pharo VM
   pharo          Script to run the downloaded VM in headless mode
   pharo-ui       Script to run the downloaded VM in UI mode
   pharo-vm/      Directory containing the VM
```

Grabbing and executing it. If you just want to directly execute the script you can also do the following

```
wget -O - get.pharo.org/20+vm | bash
```

The option -O - will output the downloaded bash file to standard out, so we can pipe it to bash. If you do not like the log of web, use --quiet.

```
wget --quiet -O - get.pharo.org/20+vm | bash
```

Note for the believers in automated tasks. The scripts are fetched automatically from our Jenkins server (https://ci.inria.fr/pharo/job/Scripts-download/) from the gitorious server https://gitorious.org/pharo-build/pharo-build. Yes we believe in automated tasks that free our energy.

1.2 Getting the VM only

You can also use different scripts. For example get.pharo.org/vm only downloads the latest vm.

```
wget -O - get.pharo.org/vm | bash
```

As any script you can always check its help message.

```
This script downloads the latest Pharo VM.
The following artifacts are created:
  pharo      Script to run the downloaded VM in headless mode
  pharo-ui   Script to run the downloaded VM in UI mode
  pharo-vm/  Directory containing the VM
```

Figure 1.1 shows the list of scripts available that you can get at http://get.pharo.org.

1.3 Handling command line options

We have a brand new and nice way to handle command line arguments. It is self-documented and easily extendable. Let us have a look at how the command line is handled. As usual we will start by showing you how to find your way alone.

How to find our way

Because we highly value self-documentation, just use the --help option to get an explanation.

```
./pharo Pharo.image --help
```

It will produce the following output.

```
Usage: [<subcommand>] [--help] [--copyright] [--version] [--list]
  --help print this help message
  --copyright print the copyrights
  --version print the version for the image and the vm
  --list list a description of all active command line handlers
  <subcommand> a valid subcommand in --list

Documentation:
A DefaultCommandLineHandler handles default command line arguments and options.
The DefaultCommandLineHandler is activated before all other handlers.
It first checks if another handler is available. If so it will activate the found handler.
```

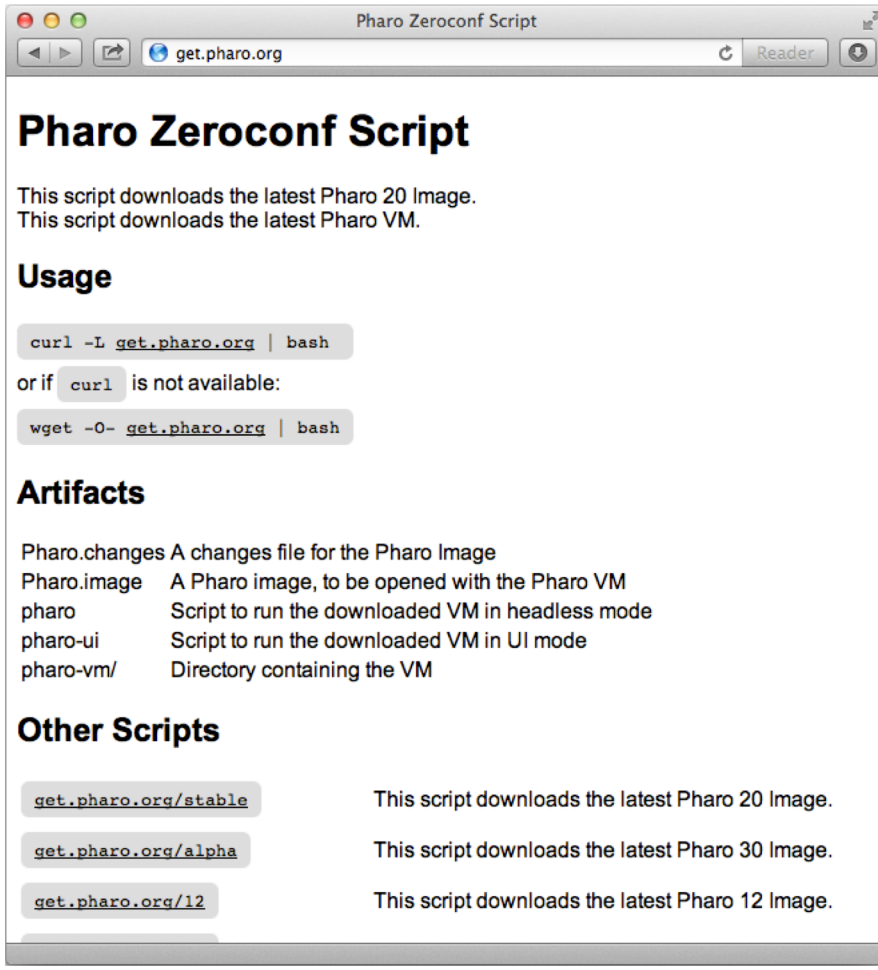

Figure 1.1: All the scripts are available at http://get.pharo.org.

System version and handler list

Two of the default options are important versions and list. Let us have a look at them now.

Getting system version. A typical and important command line option is --version. Please use it when you report bugs and deviant behavior.

```
./pharo Pharo.image --version
M:    NBCoInterpreter NativeBoost-CogPlugin-IgorStasenko.15 uuid: 44b6b681-38f1-
      4a9e-b6ee-8769b499576a Dec 18 2012
```

```
NBCogit NativeBoost-CogPlugin-IgorStasenko.15 uuid: 44b6b681-38f1-4a9e-b6ee-
    8769b499576a Dec 18 2012
git://gitorious.org/cogvm/blessed.git Commit: 452863
    bdfba2ba0b188e7b172e9bc597a2caa928 Date: 2012-12-07 16:49:46 +0100 By:
    Esteban Lorenzano <estebanlm@gmail.com> Jenkins build #5922
```

The --version argument gives the version of the virtual machine. If you wish to obtain the version of the image, then you need to open the image, use the World menu, and select About.

List of available handlers. The command line option --list lists of the *current* option handlers. This list depends on the handlers that are currently loaded in the system. In particular, it means that you can simply add a handler for your specific situation and wishes.

The following list shows the available handlers.

```
./pharo Pharo.image --list

Currently installed Command Line Handlers:
  st            Loads and executes .st source files
  Fuel          Loads fuel files
  config        Install and inspect Metacello Configurations from the command line
  save          Rename the image and changes file
  test          A command line test runner
  update        Load updates
  printVersion  Print image version
  eval          Directly evaluates passed in one line scripts
```

Loading Metacello Configuration. To get some explanation about the use of the config option, just request its associated help as follows:

```
./pharo Pharo.image config --help
```

Note that this help is the one of the associated handler, not one of the command line generic system.

```
Usage: config [--help] <repository url> [<configuration>] [--install[=<version>]] [--
    group=<group>] [--username=<username>] [--password=<password>]
  --help            show this help message
  <repository url>  A Monticello repository name
  <configuration>   A valid Metacello Configuration name
  <version>         A valid version for the given configuration
  <group>           A valid Metacello group name
  <username>        An optional username to access the configuration's repository
  <password>        An optional password to access the configuration's repository
```

```
Examples:
   # display this help message
   pharo Pharo.image config

   # list all configurations of a repository
   pharo Pharo.image config $MC_REPOS_URL

   # list all the available versions of a confgurtation
   pharo Pharo.image config $MC_REPOS_URL ConfigurationOfFoo

   # install the stable version
   pharo Pharo.image config $MC_REPOS_URL ConfigurationOfFoo --install

   #install a specific version '1.5'
   pharo Pharo.image config $MC_REPOS_URL ConfigurationOfFoo --install=1.5

   #install a specific version '1.5' and only a specific group 'Tests'
   pharo Pharo.image config $MC_REPOS_URL ConfigurationOfFoo --install=1.5 --
      group=Tests
```

1.4 Anatomy of a handler

As we mentioned, the command line mechanism is open and can be extended. We will look now how at the handler for the eval option is defined.

Evaluating Pharo Expressions. You can use the command line to evaluate expressions as follows: ./pharo Pharo.image eval '1+2'

```
./pharo Pharo.image eval --help
Usage: eval [--help] <smalltalk expression>
   --help    list this help message
   <smallltalk expression>  a valid Smalltalk expression which is evaluated and
                        the result is printed on stdout

Documentation:
A CommandLineHandler that reads a string from the command line, outputs the
      evaluated result and quits the image.

This handler either evaluates the arguments passed to the image:
   $PHARO_VM my.image eval  1 + 2

or it can read directly from stdin:

   echo "1+2" | $PHARO_VM my.image eval
```

Now the handler is defined as follows: First we define a subclass of CommandLineHandler. Here BasicCodeLoader is a subclass of CommandLineHandler and EvaluateCommandLineHandler is a subclass of BasicCodeLoader.

```
BasicCodeLoader subclass: #EvaluateCommandLineHandler
   instanceVariableNames: ''
   classVariableNames: ''
   poolDictionaries: ''
   category: 'System-CommandLine'
```

We then define the commandName on the class side as well as the method isResponsibleFor:.

```
EvaluateCommandLineHandler class>>commandName
   ^ 'eval'

EvaluateCommandLineHandler class>>isResponsibleFor: commandLineArguments
   "directly handle top-level -e and --evaluate options"
   commandLineArguments withFirstArgument: [ :arg|
      (#('-e' '--evaluate') includes: arg)
         ifTrue: [ ^ true ]].

   ^ commandLineArguments includesSubCommand: self commandName

EvaluateCommandLineHandler class>>description
   ^ 'Directly evaluates passed in one line scripts'
```

Then we define the method activate which will be executed when the option matches.

```
EvaluateCommandLineHandler>>activate
   self activateHelp.
   self arguments ifEmpty: [ ^ self evaluateStdIn ].
   self evaluateArguments.
   self quit.
```

In particular we define a class comment since this is this class comment that will be printed when the help is requested.

If you want your image saved at the end of an evaluation script, use the --save option just after eval.

1.5 Using ZeroConf script with Jenkins

Now that we have such scripts and the possibility to specify option, we can write Jenkins scripts which rely on BASH as least as possible.

For example here is the command that we use in Jenkins for the project XMLWriter (which is hosted on PharoExtras).

```
# Jenkins puts all the params after a / in the job name as well :(
export JOB_NAME=`dirname $JOB_NAME`

wget --quiet -O - get.pharo.org/$PHARO+$VM | bash

./pharo Pharo.image save $JOB_NAME --delete-old
./pharo $JOB_NAME.image --version > version.txt

REPO=http://smalltalkhub.com/mc/PharoExtras/$JOB_NAME/main
./pharo $JOB_NAME.image config $REPO ConfigurationOf$JOB_NAME --install=
      $VERSION --group='Tests'
./pharo $JOB_NAME.image test --junit-xml-output "XML-Writer-.*"

zip -r $JOB_NAME.zip $JOB_NAME.image $JOB_NAME.changes
```

1.6 Chapter summary

You can now really easily access to the latest version of Pharo and build scripts. In addition, the command-line handler opens new horizons to be used in shell scripts.