Chapter 1

Pharo Zero Conf

with the participation of:
Camillo Bruni

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 command line. Yes we do and we want the best of both worlds! Since Pharo 2.0, Pharo supports a way to define and handle command line argument. We will present that in this chapter.

We really wanted it to free our mind of retaining arbitrary information. A zero configuration is then a script that automatically download everything you need to get started.

In this chapter we will show how to get the zeroConf scripts family for Pharo as well as how you can pass argument to the environment from the command-line.

1.1 Getting the latest VM and the Image

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

```
wget http://files.pharo.org/script/ciPharo20PharoVM.sh
```

Note that files.pharo.org is an alias for pharo.gforge.inria.fr/ci. So we suggest to use wget instead of curl because curl does not deal well with redirections.

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

you also to have a look at the script help.

```
bash ./ciPharo20PharoVM.sh --help
```

What the help mentions is that the script will download the current vm and put it into the vm folder, vm.sh a script to launch the system, vm-ui.sh a script to launch the image in ui mode, the image and its associated changes.

```
This script will download the latest Pharo 2.0 image and VM

Result in the current directory:
   vm          VM directory
   vm.sh          Script forwarding to the VM in vm
   vm-ui.sh          Script running the VM interactively with a UI
   Pharo.image          The latest pharo image
   Pharo.changes          The corresponding pharo changes
```

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

```
wget http://files.pharo.org/script/ciPharo20PharoVM.sh | bash
```

If you do not like the log of web, use --quiet -O.

```
wget --quiet -O - http://files.pharo.org/script/ciPharo20PharoVM.sh | 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 latest VM only

You can also use different scripts. For example ciPharoVM.sh only downloads the latest vm.

```
wget http://files.pharo.org/script/ciPharoVM.sh | bash
```

Again as any script you can always check its help message.

```
This script will download the stable pharo VM

Result in the current directory:
   vm          Directory containing the VM
   vm.sh          Script forwarding to the VM inside the vm directory running headlessly
   vm-ui.sh          Script running the VM interactively with a UI
```

Figure 1.1 shows the list of scripts available that you can get at http://pharo.gforge.inria.fr/ci/script/.

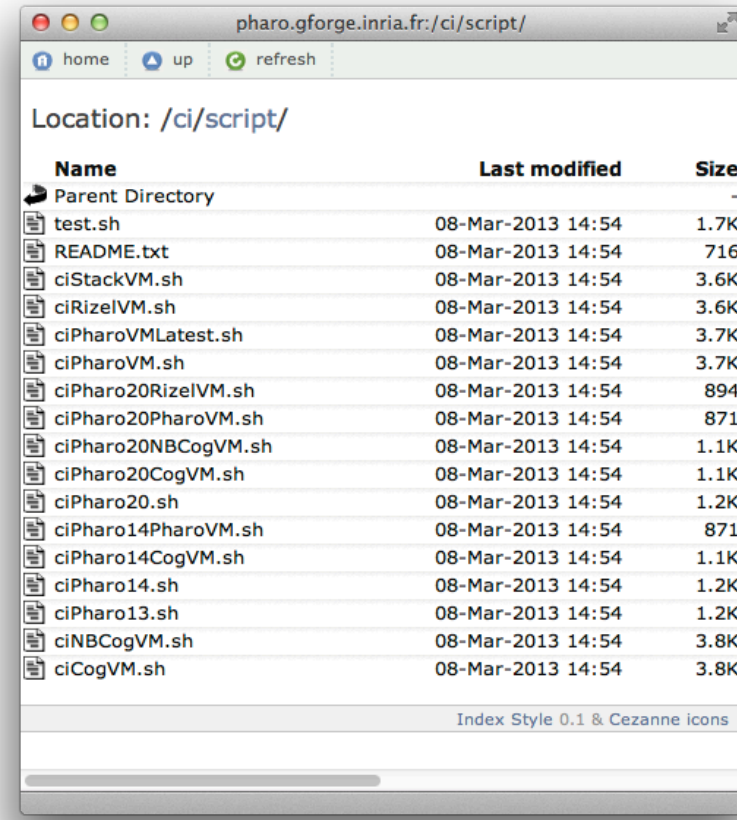

Figure 1.1: All the scripts are available at http://pharo.gforge.inria.fr/ci/script/ .

1.3 The scripts

Here is a typical script. We list it here for archival purpose and the definition may change in the future but you should get the idea.

ciPharo20PharoVM.sh.

```
#!/bin/bash

# stop the script if a single command fails
set -e

# ARHUMENT HANDLING ===========================================================

if { [ "$1" = "-h" ] || [ "$1" = "--help" ]; }; then
    echo "This script will download the latest Pharo 2.0 image and VM

Result in the current directory:
    vm              VM directory
    vm.sh           Script forwarding to the VM in vm
    Pharo.image     The latest pharo image
    Pharo.changes   The corresponding pharo changes"
    exit 0
elif [ $# -gt 0 ]; then
    echo "--help is the only argument allowed"
    exit 1
fi

# FETCH DATA ==================================================================
wget --quiet -qO - http://pharo.gforge.inria.fr/ci/script/ciPharoVM.sh | bash
wget --quiet -qO - http://pharo.gforge.inria.fr/ci/script/ciPharo20.sh | bash
```

Now that you get ready to launch Pharo nearly everywhere, let us look at the way you can always script the image from the command-line.

1.4 Handling command line options

We have now a brand new and nice way to handle command line arguments. It is self documented and really easy to extend. Let us have a look at how the command line is handled. As usual we will start to show you how to find your way alone.

How to find our way

Again we love and value self documentation so just use --help to get an explanation.

```
./vm.sh 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.
```

System version and handler list

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

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

```
./vm.sh 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
```

List of available handlers. The command line option --list is interesting since it give you the list of the *current* option handlers. This is list depends on the handlers that are currently loaded in the system. In particular it means that you can simply add an handler to handler your specific situation and wishes.

The following list shows that handlers available in the system we used when writing this chapter.

```
./vm.sh 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
```

Now you probably wonder how certain option should be used. Indeed it does not look like using the config option which handles the loading of Metacello configurations is crystal clear, isn't. But as you guessed handler are also self described. Let us have a look at this one.

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

```
./vm.sh Pharo.image config --help
```

Note that this is help is the one of the associated handler not the 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
  $PharoVM My.image config

  # list all configurations of a repository
  $PharoVM My.image config $MC_REPOS_URL

  # list all the available versions of a confgurtation
  $PharoVM My.image config $MC_REPOS_URL ConfigurationOfFoo

  # install the stable version
  $PharoVM My.image config $MC_REPOS_URL ConfigurationOfFoo --install

  #install a specific version '1.5'
  $PharoVM My.image config $MC_REPOS_URL ConfigurationOfFoo --install=1.5

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

1.5 Anatomy of a handler

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

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

```
./vm.sh 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'
```

Then we 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.

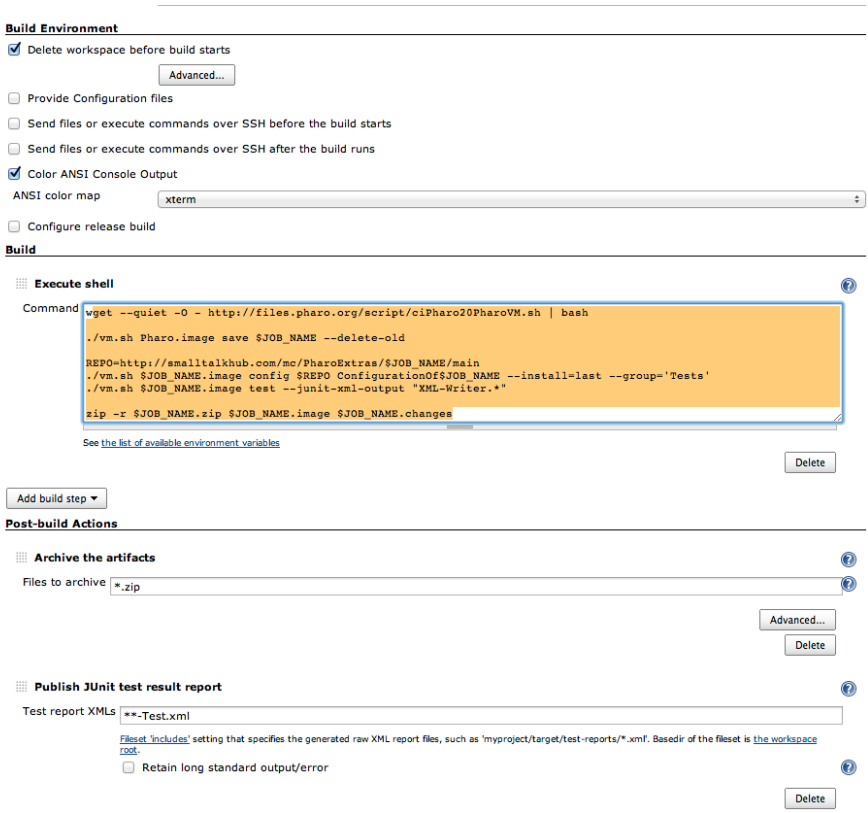

Figure 1.2: XMLWriter ZeroConf scripts.

1.6 Using ZeroConf script with Jenkins

Now that we have such scripts and the possibility to specify option, we can write jenkins scripts which are relying as less as possible on bash.

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

```
get --quiet -O - http://files.pharo.org/script/ciPharo20PharoVM.sh | bash

./vm.sh Pharo.image save $JOB_NAME --delete-old

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

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

1.7 Conclusion

As conclusion, you have now tools to quickly create UIs and to reuse them. Since the reuse is really a strong value for specs, keep in mind that your tools can be reused. So do not forget to provide a proper API for it.