

# OSCAR CLI: a Command Line Interface for OSCAR

Benoît des Ligneris<sup>1</sup>  
Fernando Laudaes Camargos<sup>2</sup>

<sup>1</sup> Révolution Linux Inc.  
145, rue Sauvé - bureau 15-16  
Sherbrooke (Quebec)  
J1L1L6 CANADA

[benoit.des.ligneris@revolutionlinux.com](mailto:benoit.des.ligneris@revolutionlinux.com)

<sup>2</sup> Federal University of Santa Catarina (UFSC)

Servidão Dias, 292  
Florianópolis, SC  
8830-330 Brasil

[laudaes@inf.ufsc.br](mailto:laudaes@inf.ufsc.br)

## Abstract

This article discusses the implementation of a Command Line Interface (CLI) for OSCAR. The structure of the CLI as well as the different commands required to reproduce the Graphical User Interface (GUI) will be presented.

**Keywords:** CLI, Command Line Interface

## Introduction

At the time of this writing OSCAR functionalities are only available via the OSCAR Wizard Graphical Interface (GUI): this is the only way to proceed with the installation of OSCAR. While it is easy to use, and offers an intuitive and friendly installation process it also has certain limits :

- to reproduce a given OSCAR installation, the steps must be followed explicitly. For large clusters with multiple servers, this can be a major problem;
- the GUI is not scriptable per se (one has to use an external tool);
- testing OSCAR is a lengthy process that is difficult to automate (because the GUI needs

some human interaction to function properly);

- an X-windowing system has to be installed on the server.

This paper defines the organization of a Command Line Interface (CLI) for OSCAR. The CLI should provide roughly the same level of functionality as the mainline GUI and is not there to replace the GUI : a GUI is clearly more user-friendly than a CLI and performs certain classes of actions much better (selection, display of large lists, etc).

Trying to work through the CLI for all GUI actions limits the GUI and also makes the CLI much more complex. Indeed, the objective of the

CLI is not to replace the actual Wizard (GUI), but instead to have the GUI call the CLI. This will be especially interesting once an automated, and tested system will be in place for OSCAR. In the meantime, our proposal is that the GUI, and CLI should be alternatives and not dependent layers: this way the CLI could be connected to the same API as the GUI connects to, but neither interface is limited by, or capable of overwrite the other.

The idea of developing a command line interface for OSCAR arose when we had to face the task of building a large cluster of more than 1024 nodes. While the main GUI simplifies the installation of a generic cluster, based on the OSCAR supported architecture (one server, and several nodes), it does not support more complex architectures. Moreover, the architecture we planned for the large cluster is hierarchical with a "master OSCAR server" and several sub-servers that will each control between 48 and 96 nodes

In order to achieve this goal, building a CLI will provide exceptional flexibility for OSCAR clusters. It will then be possible to support any kind of cluster-architecture. One bonus of the CLI is to ease the development of an automated test suite, which will in turn, save OSCAR developers a lot of development time.

reproduce the OSCAR installation and produce a script file that will allow a complete replay of the installation, except for the MAC address collection. This solution can be particularly attractive for cluster vendors as it will automate the installation, and customization, of any cluster installation. It will be possible to edit the cluster install script, as well as the key information (namely the node MAC addresses) provided by the system integrator. This will allow a cluster to be installed in a matter of minutes, according to the customer specifications, without human intervention.

### CLI structure

At this time, the CLI structure has been created in order to mimic the actual steps of the OSCAR wizard. All functions, with the exception of one, should be easy to implement, especially those functions that affect the database (read or write).

Indeed, OSCAR uses, almost exclusively, ODA (the Oscar Database) as well as an interface for accessing the existing database (read, write, update, delete).

As the CLI requires direct access to the OSCAR libraries, which are programmed in Perl, this is the language of choice for the implementation.

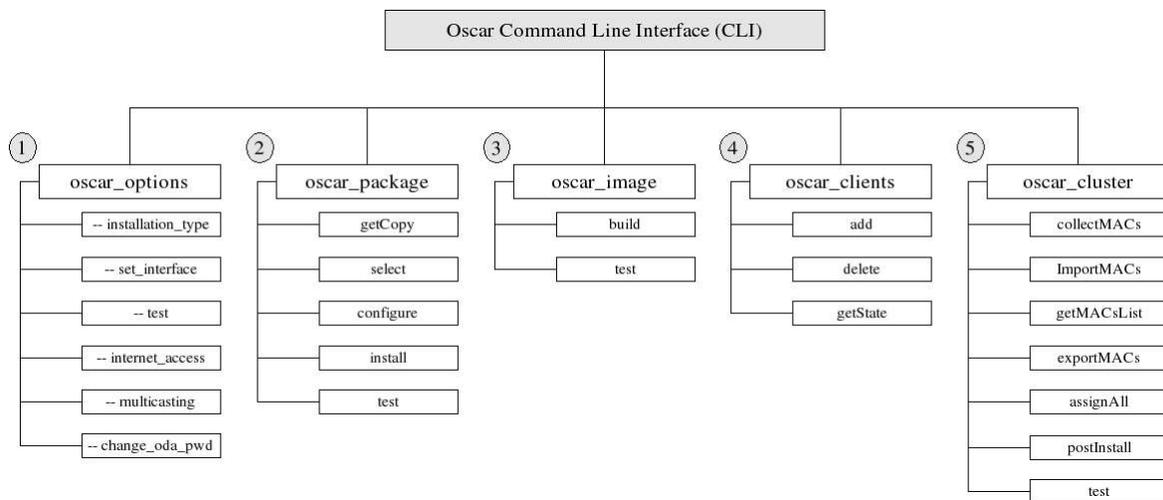


Figure 1: structure of the OSCAR CLI commands

At some point, we would like to be able to

## OSCAR global options : oscar\_option

Before attempting to install OSCAR, there is a set of options that should be configured. As this step is optional, there must be a 'default' configuration (when required). The command name is "oscar\_option"

### **--installation\_type (default : disk)**

Defines which type of OSCAR installation will be used (meaning: 'disk' or 'diskless' (Thin-OSCAR)).

Ex: 'oscar\_option --intallation\_type diskless'.

### **--set\_interface (default : eth1)**

Defines which server adapter (interface) will be used to connect to the cluster network.

Ex: 'oscar\_option --set\_interface eth1'.

### **--internet\_access (default : yes)**

Enables or disables the cluster's internet access (i.e.: if the cluster will or will not have "contact" with the "outside world"). This issue is particularly important when deciding if the server will be able to 'download' additional packages (from an outside repository).

Ex: 'oscar\_option --internet\_acces yes'.

### **--multicasting (default : no)**

Enable or disable multicasting.

Ex: 'oscar\_option --multicasting yes'.

### **--change\_oda\_pwd**

Modifies the OSCAR DATABASE administrator's password.

### **--test**

Performs a series of test scripts that ensures that the choosen global options are OK ("checks the system to see if it is "OSCAR-ready" by checking file locations, rpm versions, etc").

Ex: 'oscar\_option --test'.

## OSCAR Package : oscar\_package command

Performs the copy/download, selection, installation, configuration, and testing of the OSCAR (core and additional) packages.

### **getCopy**

The command '*getCopy*' retrieves a copy of additional packages. This command depends entirely on whether or not the Internet is enabled. If the Internet is enabled, *getCopy*, by using the 'OSCAR Package Downloader (OPD)', establish a connection with a repository and download the desired packages from there. If there is no Internet connection, the only way to get additional packages is to copy them from a local directory into the local Opder package repository (/var/lib/oscar/packages/).

**command:** 'oscar\_package getCopy [options]'

### **--local (default : /var/lib/oscar/packages)**

Specifies the location where the additional packages can be found (it can be either a local directory, or a URL for a specialized OSCAR package repository), and prints a list with the available package's information.

Ex: 'oscar\_package getCopy --local /tmp/packages'

### **--pkname**

When '*--pkname*' is the only option chosen, (besides the '*--local*' option), the copy/download of the package will be performed.

Ex: 'oscar\_package getCopy --pkname clumon'

### **--pkname --xmlinfo**

When the option '*--pkname*' is entered with the "*--xmlinfo*" option, the package information will be printed. (This can be any XML tag of the config.xml definition of the package.)

## **select**

The command '*select*' chooses which additional packages will be installed (by default, all packages directly included in OSCAR (the core packages) are selected and installed). The additional copied/downloaded packages must be manually selected for installation.

**command:** 'oscar\_package select [options]'

### **--table**

Gets the list (table) of the available packages for immediate selection (including package name, class and location/version).

Ex: 'oscar\_package select --table'.

### **--pkname**

As with the command 'getCopy', if this option is entered "alone", the package is selected for installation.

Ex: 'oscar\_package select --pkname clumon'.

### **--pkname --xmlinfo**

When the option '--pkname' is entered with the "--xmlinfo" option, the package information will be printed. (This can be any XML tag of the config.xml definition of the package.)

Ex: 'oscar\_package select --pkname clumon --requires'.

A complementary, but important option, that helps in the automatization of the process, is:

### **--all**

Select all packages (including the packages previously copied/downloaded)

Ex: 'oscar\_package select --all'.

(PS: there is no need for a '*default*' select option, as all of the core packages of OSCAR are automatically selected for installation)

## **configure**

Some packages in OSCAR have configuration options. The command '*configure*' allows for the configuration of these packages.

**command:** 'oscar\_package configure [options]'

### **--table**

Prints a list of packages that can be configured.

## **install**

Performs the installation of selected packages in the OSCAR server.

**command:** 'oscar\_package install'.

## **test**

Performs a test with a package to assure that the installation was successful.

**command:** 'oscar\_package test [option]'

Options:

### **--table**

Print a list (table) of the installed packages.

Ex: 'oscar\_package test --table'.

### **--pkname**

Specifies which package you want to test.

Ex: 'oscar\_package test --pkname clumon'.

## **OSCAR Image commands : oscar\_image**

Builds and tests the future client's image, which is constructed from the OSCAR core, and additional (selected) packages, plus the operational system files (like RPMs).

## **build**

Builds the image, that will be "network copied" to the node (uses the 'mkssiimage' command).

**command:** 'oscar\_image build [options]'.

It is possible to either customize the default image, by altering the following options, or simply build the default image (by omitting the options).

Options:

**--seeDefault**

Print the default options.

Ex: 'oscar\_image build --seeDefault'.

**--imageName**

Specifies the name of the current image customization, by which you will identify the image (you can create several different image customizations);

**--packageFile**

Specifies the full address of the text file containing a list of packages that should be installed (usually present in '/opt/oscar/oscarsamples');

**--packageDirectory**

Specifies the directory which contains the packages that will be used to build the image (usually '/tftpboot/rpm');

**--diskPartFile**

Specifies a text file that contains the disk partition table for the image;

**--ipAssignMet**

Specifies the IP assignment method, that can be either 'static', 'dhcp' or 'replicant';

**--postInstAct**

Specifies which will be the client's post install action, this can be either 'beep', 'reboot' or 'shutdown';

Ex: 'oscar\_image build --imagename oscar1 --packageFile /opt/oscar/oscarsamples/mandrake-

```
9.2-i386.rpm1ist --packageDirectory /tftpboot/rpm
--diskPartFile /
opt/oscar/oscarsamples/sample.disk.ide --
ipAssignMet static --postInstAct beep;.
```

**delete**

Delete an existing OSCAR image.

**--imageName**

The name of the image, used to identify it.

Ex: 'oscar\_image delete --imageName oscar1'.

OSCAR Node commands : oscar\_node

Takes care of tclient management (include, remove, test).

**add**

Defines which clients will be installed. It can be used either for the first installation to establish the main set of nodes, as well as later to add new clients to the selected image group.

**command:** 'oscar\_node add [options]'.

Options:

**--seeDefault**

Print the default values

**--imageName**

Specifies which image (previously built) to use.

**--domainName**

Specifies the client's IP domain name;

**--baseName**

Specifies the first part of the client's name and hostname (this name can not contain an underscore character);

**--numberOfHosts**

Specifies the number of hosts. This field does not have a default value and must be filled. The value must be greater than zero.

### **--startNumber**

Specifies the number of the first node;

### **--padding**

Specifies the number of digits to pad the node name with;

### **--startIP**

Specifies the network subnet mask;

### **--defaultGateway**

Specifies the default route to send all packets;

Ex: 'oscar\_node install --imageName oscarTest1 --numberOfHosts 2'.

### **delete**

Deletes a client from your running OSCAR cluster.

**command:** 'oscar\_node delete [options]'.

Options:

### **--clientID**

Specifies the ID of the client that you want to remove from your cluster (the ID is formed by the base name plus the client's index number [which depends on the pad digits]).

Ex: 'oscar\_node delete --clientID oscarNode1'.

### **--all**

Deletes all clients from the cluster.

Ex: 'oscar\_node delete --all'.

### **getState**

Get the current state of a determined client. By default, use the ping test.

**command:** 'oscar\_node getState [option]'.

### **--clientID**

Node name

### **--protocol**

Which protocol to use (ping, SSH, openPBS, etc.)

### **-- clientFileListst**

Name of a file that contains one node name per line. Will return a list clientName, value

Ex: 'oscar\_node getState --clientID oscarNode1 --protocol SSH'.

OSCAR global cluster commands

Collect the MAC addresses and assign specific IP addresses to the Clients, complete the installation of the cluster and test it.

### **collectMACs**

This command is very difficult to reproduce in a CLI. The idea is to collect as many MAC addresses as the number of nodes, and then stop the process.

**command:** 'oscar\_cluster collectMACs'

### **importMACs**

Import a list of MAC addresses (previously collected) from a file.

**command:** 'oscar\_cluster importMACs [file]'.

Ex: 'oscar\_cluster importMACs /mnt/floppy/listA'.

### **getMACsList**

Print the list of the collected (or “imported”) MACs.

**command:** 'oscar\_cluster getMACsList'.

### **exportMACs**

Save the list containing the collected MAC addresses in a file, for “re-use”.

**command:** 'oscar\_cluster exportMACs [file]'.

Ex: 'oscar\_cluster exportMACs /tmp/listOfMACs'

### **assignAll**

Assign all of the collected MACs to all of the "free" nodes.

**command:** 'oscar\_cluster assignAll'

## **postInstall**

Execute scripts that complete the installation of the cluster (must be executed only after all the nodes have completed their installation, rebooted from the hard disk and have their ethernet adaptors up).

**command:** 'oscar\_cluster postInstall'

## **test**

Test the cluster to check if the OSCAR installation is working.(oscar\_package performs a node-level test; oscar\_cluster test performs a cluster-level test. The Cluster-level test can be done only if the package-level test is successful).

## **Conclusion**

The main objective of this article is to present the specifications for a command line interface (CLI) for OSCAR in order that the OSCAR community may comment the article.

The rationale for developing a CLI are numerous : adaptation of OSCAR to any arbitrary cluster topologies, scalability of OSCAR, ease the work

of OSCAR cluster integrators, remove the necessity to have an X windowing system on the master node, ensure the exact image of any OSCAR installation, and finally, allow an easy implementation of an automated regression testing suite for OSCAR.

The proposal for the development of the CLI is not disruptive for the GUI as both will, at first, be independant. The development of the CLI will ensure a strict separation of functionality and interface as all the functionality code will be moved into the OSCAR libraries. This is the best known method for software development.

The long-term objective is to remove code duplication in order that the GUI makes use of the CLI instead of calling the Perl library directly. This will allow for separate testing of the GUI, and of the functionalities (CLI). Once this is achieved, each OSCAR installation will produce a script of OSCAR CLI commands that will allow the reproduction of the exact same cluster installation (very useful for cluster integrators and for the reproductability of bugs).