

---

# Lago Documentation

*Release 0.34.0*

**David Caro**

February 07, 2017



<b>1</b>	<b>Lago Introduction</b>	<b>1</b>
<b>2</b>	<b>Getting started</b>	<b>3</b>
2.1	Installing Lago . . . . .	3
2.2	Getting started with some Lago Examples! . . . . .	4
2.3	Configuration . . . . .	4
2.4	Debugging Lago Environment . . . . .	6
<b>3</b>	<b>Developing</b>	<b>11</b>
3.1	CI Process . . . . .	11
3.2	Environment setup . . . . .	12
3.3	Getting started developing . . . . .	14
<b>4</b>	<b>Contents</b>	<b>17</b>
4.1	lago package . . . . .	17
4.2	lago_template_repo package . . . . .	60
4.3	ovirtlago package . . . . .	61
<b>5</b>	<b>Releases</b>	<b>67</b>
5.1	Release process . . . . .	67
<b>6</b>	<b>Changelog</b>	<b>71</b>
<b>7</b>	<b>Indices and tables</b>	<b>73</b>
	<b>Python Module Index</b>	<b>75</b>



---

## Lago Introduction

---

Lago is an add-hoc virtual framework which helps you build virtualized environments on your server or laptop for various use cases.

It currently utilizes ‘libvirt’ for creating VMs, but we are working on adding more providers such as ‘containers’.



---

## Getting started

---

### 2.1 Installing Lago

You'll notice that some of the actions you need to do to run Lago are currently manual, but we are working to add them as part of the standard Python packaging for Lago which is in progress.

#### 2.1.1 Setting up yum repos

Currently only RPM installation is available but we are working on adding support for Ubuntu and Debian soon.

Add the following repos to a lago.repo file in your /etc/yum.repos.d/ dir:

For Fedora:

```
[lago]
baseurl=http://resources.ovirt.org/repos/lago/stable/0.0/rpm/fc$releasever
name=Lago
enabled=1
gpgcheck=0
```

For EL distros (such as CentOS, RHEL, etc.), make sure you have epel-release and centos-release-qemu-ev repositories installed and enabled, and:

```
[lago]
baseurl=http://resources.ovirt.org/repos/lago/stable/0.0/rpm/el$releasever
name=Lago
enabled=1
gpgcheck=0
```

#### 2.1.2 Installing the packages

Once you have them, install the following packages:

```
$ yum install python-lago lago
```

This will install all the needed packages to get you up and running with Lago.

#### 2.1.3 Configuring Libvirt

Make sure libvirt is configured to run:

```
$ systemctl enable libvirtd
$ systemctl start libvirtd
```

## 2.1.4 User permissions setup

Running lago requires certain permissions, so the user running it should be part of certain groups.

Add yourself to lago and qemu groups:

```
$ usermod -a -G lago USERNAME
$ usermod -a -G qemu USERNAME
```

It is also advised to add qemu user to your group (to be able to store VM files in home directory):

```
$ usermod -a -G USERNAME qemu
```

For the group changes to take place, you'll need to re-login to the shell. Make sure running *id* returns all the aforementioned groups.

Make sure that the qemu user has execution rights to the dir where you will be creating the prefixes, you can try it out with:

```
$ sudo -u qemu ls /path/to/the/destination/dir
```

If it can't access it, make sure that all the dirs in the path have your user or qemu groups and execution rights for the group, or execution rights for other (highly recommended to use the group instead, if the dir did not have execution rights for others already)

It's very common for the user home directory to not have group execution rights, to make sure you can just run:

```
$ chmod g+x $HOME
```

And, just to be sure, let's refresh libvirtd service to ensure that it refreshes its permissions and picks up any newly created users:

```
$ sudo service libvirtd restart
```

## 2.2 Getting started with some Lago Examples!

Get Lago up & running in no time using one of the available examples

Important: make sure you followed the installation step before to have Lago installed.

### 2.2.1 Available Examples

- Simple Jenkins server + slaves: Jenkins\_Example
- Advanced oVirt example (using nested virtualization): oVirt\_Example

## 2.3 Configuration

The recommend method to override the configuration file is by letting lago auto-generate them:



```
$ mkdir -p $HOME/.config/lago
$ lago generate-config > $HOME/.config/lago/lago.conf
```

This will dump the current configuration to `$HOME/.config/lago/lago.conf`, and you may edit it to change any parameters. Take into account you should probably comment out parameters you don't want to change when editing the file. Also, all parameters in the configuration files can be overridden by passing command line arguments or with environment variables, as described below.

### 2.3.1 lago.conf format

Lago runs without a configuration file by default, for reference-purposes, when lago is installed from the official packages(RPM or DEB), a commented-out version of `lago.conf`(INI format) is installed at `/etc/lago/lago.conf`.

In `lago.conf` global parameters are found under the `[lago]` section. All other sections usually map to subcommands(i.e. `lago init` command would be under `[init]` section).

*Example:*

```
$ lago generate-config
> [lago]
> # log level to use
> loglevel = info
> logdepth = 3
> ....
> [init]
> # location to store repos
> template_repos = /var/lib/lago/repos
> ...
```

### 2.3.2 lago.conf look-up

Lago attempts to look `lago.conf` in the following order:

1. `/etc/lago/lago.conf`
2. According to [XDG standards](#) , which are by default:
  - `/etc/xdg/lago/lago.conf`
  - `/home/$USER/.config/lago/lago.conf`
3. Any environment variables.
4. CLI passed arguments.

If more than one file exists, all files are merged, with the last occurrence of any parameter found used.

### 2.3.3 Overriding parameters with environment variables

To differentiate between the root section in the configuration file, lago uses the following format to look for environment variables:

```
'LAGO_GLOBAL_VAR' -> variable in [lago] section
'LAGO__SUBCOMMAND__PARAM_1' -> variable in [subcommand] section
```

Example: changing the `template_store` which `init` subcommand uses to store templates:

```
# check current value:
$ lago generate-config | grep -A4 "init"
> [init]
> # location to store repos
> template_repos = /var/lib/lago/repos
> # location to store temp
> template_store = /var/lib/lago/store

$ export LAGO__INIT__TEMPLATE_STORE=/var/tmp
$ lago generate-config | grep -A4 "init"
> [init]
> # location to store repos
> template_repos = /var/lib/lago/repos
> # location to store temp
> template_store = /var/tmp
```

## 2.4 Debugging Lago Environment

Now that you've run any of the examples, you probably eager to dive into the Lago environment and checkout the created VMs and resources.

This document will give you a taste of how to do it, but if you're interested In a deeper view, please checkout the full Lago tutorial (TBD).

We'll use the oVirt example for explaining how to debug the environment created.

### 2.4.1 Debugging the Lago environment created by oVirt system tests

As the above script has become a bit complicated, and it's not (yet) part of Lago itself, this section will do the same as the script, but step by step with Lago only command to give you a better idea of what you have to do in a usual project.

So, let's get back to the root of the ovirt-system-tests repo, and cd into the basic\_suite\_4.0 dir:

```
cd ovirt-system-tests/basic_suite_4.0
```

Let's take a look to what is in there:

```
$ tree
.
-- control.sh
-- deploy-scripts
|   -- add_local_repo.sh
|   -- bz_1195882_libvirt_workaround.sh
|   -- setup_container_host.sh
|   -- setup_engine.sh
|   -- setup_host.sh
|   -- setup_storage_iscsi.sh
|   -- setup_storage_nfs.sh
-- engine-answer-file.conf
-- init.json.in
-- reposync-config.repo
-- template-repo.json
-- test-scenarios
    -- 001_initialize_engine.py
    -- 002_bootstrap.py
```

```
-- 003_create_clean_snapshot.py
-- 004_basic_sanity.py
```

We can ignore the *control.sh* script, as it's used by the *run\_suite.sh* and we don't care about that in this readme.

### init.json.in: The heart of lago, virt configurations

This *init.json.in* file, is where we will describe all the virtual elements of our test environment, usually, vms and networks.

In this case, as the file is shared between suites, it's actually a template and we will have to change the *@SUITE@* string inside it by the path to the current suite:

```
$ suite_path=$PWD
$ sed -e "s/@SUITE@/$suite_path/g" init.json.in > init.json
```

Now we have a full *init.json* file :), but we have to talk about another file before being able to create the prefix:

Note that lago supports json and yaml formats for that file.

### template-repo.json: Sources for templates

This file contains information about the available disk templates and repositories to get them from, we can use it as it is, but if you are in Red Hat office in Israel, you might want to use the Red Hat internal mirrors there, for that use the *common/template-repos/office.json* file instead, see next for the full command line.

**NOTE:** You can use any other template repo if you specify your own json file there

### Initializing the prefix

Now we have seen all the files needed to initialize our test prefix (aka, the directory that will contain our env). To do so we have to run this:

```
$ lagocli init \
  --template-repo-path=template-repo.json \
  deployment-basic_suite_4.0 \
  init.json
```

Remember that if you are in the Red Hat office, you might want to use the repo mirror that's hosted there, if so, run this command instead:

```
$ lagocli init \
  --template-repo-path=common/template-repos/office.json \
  deployment-basic_suite_4.0 \
  init.json
```

This will create the *deployment-basic\_suite\_4.0* directory and populate it with all the disks defined in the *init.json* file, and some other info (network info, uuid... not relevant now).

This will take a while the first time, but the next time it will use locally cached images and will take only a few seconds!

### If you are using run\_suite.sh

To use an alternate repository template file when running *run\_suite.sh*, you'll have to edit it for now, search for the *init* command invocation and modify it there, at the time of writing this, if you want to use the Red Hat Israel office mirror, you have to change this:

```
38 env_init () {
39     $CLI init \
40         $PREFIX \
41         $SUITE/init.json \
42         --template-repo-path $SUITE/template-repo.json
43 }
```

by:

```
env_init () {
    $CLI init \
        $PREFIX \
        $SUITE/init.json \
        --template-repo-path common/template-repos/office.json
}
```

### reposync-config.repo: yum repositories to make available to the vms

This file contains a valid yum repos definition, it's the list of all the yum repos that will be enabled on the vms to pull from. If you want to use any custom repos just add the yum repo entry of your choice there and it will be make accessible to the vms.

The internal repository is built from one or several 'sources', there are 2 types of sources:

- External RPM repositories:

A yum .repo file can be passed to the verb, and all the included repositories will be downloaded using 'reposync' and added to the internal repo.

This is used by the *ovirt reposetup* verb. To prefetch and generate the local repo, we have to run it:

```
$ lagocli ovirt reposetup --reposync-yum-config="reposync-config.repo"
```

This might take a while the first time too, as it has to fetch a few rpms from a few repos, next time it will also use a cache to speed things up considerably.

**NOTE:** From now on, all the *lagocli* command will be run inside the prefix, so cd to it:

```
$ cd deployment-basic_suite_4.0
```

### Bring up the virtual resources

We are ready to start powering up vms!

```
# make sure you are in the prefix
$ pwd
/path/to/ovirt-system-tests/deployment-basic_suite_4.0
$ lagocli start
```

This starts all resources (VMs, bridges), at any time, you can use the *stop* verb to stop all active resources.

### Run oVirt initial setup scripts

Once all of our vms and network are up and running, we have to run any setup scripts that will configure oVirt in the machines, as we already described in the *init.json* what scripts should be executed, the only thing left is to trigger it:

```
$ lagocli ovirt deploy
```

This should be relatively fast, around a minute or two, for everything to get installed and configured

## Running the tests

Okok, so now we have our environment ready for the tests!! \o/

Lets get it on, remember that they should be executed in order:

```
$ lagocli ovirt runtest 001_initialize_engine.py
...
$ lagocli ovirt runtest 002_bootstrap.py
...
$ lagocli ovirt runtest 003_create_clean_snapshot.py
...
$ lagocli ovirt runtest 004_basic_sanity.py
...
```

This tests run a simple test suite on the environment:

- Create a new DC and cluster
- Deploy all the hosts
- Add storage domains
- Import templates

The tests are written in python and interact with the environment using the python SDK.

## Collect the logs

So now we want to collect all the logs from the vms, to troubleshoot and debug if needed (or just to see if they show what we expect). To do so, you can just:

```
$ lagocli ovirt collect \
  --output "test_logs"
```

We can run that command anytime, you can run it in between the tests also, specifying different output directories if you want to see the logs during the process or compare later with the logs once the tests finish.

You can see all the logs now in the dir we specified:

```
$ tree test_logs
test_logs/
-- engine
|   -- _var_log_ovirt-engine
|       -- boot.log
|       -- console.log
|       -- dump
|       -- engine.log
|       -- host-deploy
|       -- notifier
|       -- ovirt-image-uploader
|       -- ovirt-iso-uploader
|       -- server.log
|       -- setup
|           -- ovirt-engine-setup-20151029122052-7g9q2k.log
```

```
-- host0
|   -- _var_log_vdsm
|       -- backup
|       -- connectivity.log
|       -- mom.log
|       -- supervdsm.log
|       -- upgrade.log
|       -- vdsmd.log
-- host1
|   -- _var_log_vdsm
|       -- backup
|       -- connectivity.log
|       -- mom.log
|       -- supervdsm.log
|       -- upgrade.log
|       -- vdsmd.log
-- host2
|   -- _var_log_vdsm
|       -- backup
|       -- connectivity.log
|       -- mom.log
|       -- supervdsm.log
|       -- upgrade.log
|       -- vdsmd.log
-- host3
|   -- _var_log_vdsm
|       -- backup
|       -- connectivity.log
|       -- mom.log
|       -- supervdsm.log
|       -- upgrade.log
|       -- vdsmd.log
-- storage-iscsi
-- storage-nfs
```

## Cleaning up

As before, once you have finished playing with the prefix, you will want to clean it up (remember to play around!), to do so just:

```
$ lagocli cleanup
```

## 2.4.2 FAQ

1. How do I know if the `run_suite.sh` is stuck or still running?

Sometimes the script is downloading very big files which might seem to someone as the script is stuck. One hacky way of making sure the script is still working is to check the size and content of the store dir:

```
$ ls -la /var/lib/lago/store
```

This will show any templates being downloaded and file size changes.

---

## Developing

---

### 3.1 CI Process

Here is described the usual workflow of going through the CI process from starting a new branch to getting it merged and released in the [unstable repo](#).

#### 3.1.1 Starting a branch

First of all, when starting to work on a new feature or fix, you have to start a new branch (in your fork if you don't have push rights to the main repo). Make sure that your branch is up to date with the project's master:

```
git checkout -b my_fancy_feature
# in case that origin is already lago-project/lago
git reset --hard origin/master
```

Then, once you can just start working, doing commits to that branch, and pushing to the remote from time to time as a backup.

Once you are ready to run the ci tests, you can create a pull request to master branch, if you have [hub](#) installed you can do so from command line, if not use the ui:

```
$ hub pull-request
```

That will automatically trigger a test run on ci, you'll see the status of the run in the pull request page. At that point, you can keep working on your branch, probably just rebasing on master regularly and maybe amending/squashing commits so they are logically meaningful.

#### 3.1.2 A clean commit history

An example of not good pull request history:

- Added right\_now parameter to virt.VM.start function
- Merged master into my\_fancy\_feature
- Added tests for the new parameter case
- Renamed right\_now parameter to sudo\_right\_now
- Merged master into my\_fancy\_feature
- Adapted test to the rename

This history can be greatly improved if you squashed a few commits:

- Added `sudo_right_now` parameter to `virt.VM.start` function
- Added tests for the new parameter case
- Merged master into `my_fancy_feature`
- Merged master into `my_fancy_feature`

And even more if instead of merging master, you just rebased:

- Added `sudo_right_now` parameter to `virt.VM.start` function
- Added tests for the new parameter case

That looks like a meaningful history :)

### 3.1.3 Rerunning the tests

While working on your branch, you might want to rerun the tests at some point, to do so, you just have to add a new comment to the pull request with one of the following as content:

- `ci test please`
- `ci :+1:`
- `ci :thumbsup:`

### 3.1.4 Asking for reviews

If at any point, you see that you are not getting reviews, please add the label ‘needs review’ to flag that pull request as ready for review.

### 3.1.5 Getting the pull request merged

Once the pull request has been reviewed and passes all the tests, an admin can start the merge process by adding a comment with one of the following as content:

- `ci merge please`
- `ci :shipit:`

That will trigger the merge pipeline, that will run the tests on the merge commit and deploy the artifacts to the [unstable repo](#) on success.

## 3.2 Environment setup

Here are some guidelines on how to set up your development of the lago project.

### 3.2.1 Requirements

You’ll need some extra packages to get started with the code for lago, assuming you are running Fedora:

```
> sudo dnf install git mock libvirt-daemon qemu-kvm autotools
```



And you'll need also a few Python libs, which you can install from the repos or use venv or similar, for the sake of this example we will use the repos ones:

```
> sudo dnf install python-flake8 python-nose python-dulwich yapf
```

Yapf is not available on older Fedoras or CentOS, you can get it from the [official yapf repo](#) or try on [copr](#).

Now you are ready to get the code:

```
> git clone git@github.com:lago-project/lago.git
```

From now on all the commands will be based from the root of the cloned repo:

```
> cd lago
```

### 3.2.2 Style formatting

We will accept only patches that pass pep8 and that are formatted with yapf. More specifically, only patches that pass the local tests:

```
> make check-local
```

It's recommended that you setup your editor to check automatically for pep8 issues. For the yapf formatting, if you don't want to forget about it, you can install the pre-commit git hook that comes with the project code:

```
> ln -s scripts/pre-commit.style .git/pre-commit
```

Now each time that you run `git commit` it will automatically reformat the code you changed with yapf so you don't have any issues when submitting a patch.

### 3.2.3 Testing your changes

Once you do some changes, you should make sure they pass the checks, there's no need to run on each edition but before submitting a patch for review you should do it.

You can run them on your local machine, but the tests themselves will install packages and do some changes to the os, so it's really recommended that you use a vm, or as we do on the CI server, use mock chroots. If you don't want to setup mock, skip the next section.

Hopefully in a close future we can use lago for that ;)

#### Setting up mock\_runner.sh with mock (fedora)

For now we are using a script developed by the *oVirt* devels to generate chroots and run tests inside them, it's not packaged yet, so we must get the code itself:

```
> cd ..
> git clone git://gerrit.ovirt.org/jenkins
```

As an alternative, you can just download the script and install them in your `$PATH`:

```
> wget https://gerrit.ovirt.org/gitweb?p=jenkins.git;a=blob_plain;f=mock_configs/mock_runner.sh;hb=r
```

We will need some extra packages:

```
> sudo dnf install mock
```

And, if not running as root (you shouldn't!) you have to add your user to the newly created mock group, and make sure the current session is in that group:

```
> sudo usermod -a -G mock $USER
> newgrp mock
> id # check that mock is listed
```

### Running the tests inside mock

Now we have all the setup we needed, so we can go back to the lago repo and run the tests, the first time you run them, it will take a while to download all the required packages and install them in the chroot, but on consecutive runs it will reuse all the cached chroots.

The *mock\_runner.sh* script allows us to test also different distributions, any that is supported by mock, for example, to run the tests for fedora 23 you can run:

```
> ../jenkins/mock_runner.sh -p fc23
```

That will run all the *check-patch.sh* (the *-p* option) tests inside a chroot, with a minimal fedora 23 installation. It will leave any logs under the *logs* directory and any generated artifacts under *exported-artifacts*.

## 3.3 Getting started developing

Everyone is welcome to send patches to lago, but we know that not everybody knows everything, so here's a reference list of technologies and methodologies that lago uses for reference.

### 3.3.1 Python!

Lago is written in python 2.7 (for now), so you should get yourself used to basic-to-medium python constructs and technics like:

- Basic python: Built-in types, flow control, pythonisms (import this)
- Object oriented programming (OOP) in python: Magic methods, class inheritance

Some useful resources:

- Base docs: <https://docs.python.org/2.7/>
- Built-in types: <https://docs.python.org/2.7/library/stdtypes.html>
- About classes: <https://docs.python.org/2.7/reference/datamodel.html#new-style-and-classic-classes>
- The Zen of Python:

```
> python -c "import this"

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
```

```

Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

```

### 3.3.2 Bash

Even though there is not much bash code, the functional tests and some support scripts use it, so better to get some basics on it. We will try to follow the same standards for it than the [oVirt project](#) has.

### 3.3.3 Libvirt + qemu/kvm

As we are using intensively libvirt and qemu/kvm, it's a good idea to get yourself familiar with the main commands and services:

- libvirt: <http://libvirt.org>
- virsh client: <http://libvirt.org/virshcmdref.html>
- qemu (qemu-img is useful to deal with vm disk images): <https://en.wikibooks.org/wiki/QEMU/Images>

Also, there's a library and a set of tools from the [libguestfs](#) project that we use to prepare templates and are very useful when debugging, make sure you play at least with virt-builder, virt-customize, virt-sparsify and guestmount.

### 3.3.4 Git + Github

We use git as code version system, and we host it on Github right now, so if you are not familiar with any of those tools, you should get started with them, specially you should be able to:

- Clone a repo from github
- Fork a repo from github
- Create/delete/move to branches (git checkout)
- Move to different points in git history (git reset)
- Create/delete tags (git tag)
- See the history (git log)
- Create/amend commits (git commit)
- Retrieve changes from the upstream repository (git fetch)
- Apply your changes on top of the retrieved ones (git rebase)
- Apply your changes as a merge commit (git merge)
- Squash/reorder existing commits (git rebase --interactive)

- Send your changes to the upstream (git push)
- Create a pull request

You can always go to [the git docs](#) though there is a lot of good literature on it too.

### 3.3.5 Unit tests with py.test

Lately we decided to use [py.test](#) for the unit tests, and all the current unit tests were migrated to it. We encourage adding unit tests to any pull requests you send.

### 3.3.6 Functional tests with bats

For the functional tests, we decided to use [bats framework](#). It's completely written in bash, and if you are modifying or adding any functionality, you should add/modify those tests accordingly. It has a couple of custom constructs, so take a look to the [bats docs](#) while reading/writing tests.

### 3.3.7 Packaging

Our preferred distribution vector is through packages. Right now we are only building for rpm-based system, so right now you can just take a peek on [how to build rpms](#). Keep in mind also that we try to move as much of the packaging logic as possible to the [python packaging system](#) itself too, worth getting used to it too.

### 3.3.8 Where to go next

You can continue setting up your environment and try running the examples in the readme to get used to lago. Once you get familiar with it, you can pick any of the [existing issues](#) and send a pull request to fix it, so you get used to the ci process we use to get stuff developed flawlessly and quickly, welcome!

## 4.1 lago package

### 4.1.1 Subpackages

#### **lago.plugins package**

**exception** `lago.plugins.NoSuchPluginError`

Bases: `lago.plugins.PluginError`

`lago.plugins.PLUGIN_ENTRY_POINTS = {'vm': 'lago.plugins.vm', 'vm-service': 'lago.plugins.vm_service', 'vm-provider': 'lago.plugins.vm_provider'}`

Map of plugin type string -> setuptools entry point

**class** `lago.plugins.Plugin`

Bases: `object`

Base class for all the plugins

**exception** `lago.plugins.PluginError`

Bases: `exceptions.Exception`

`lago.plugins.load_plugins(namespace, instantiate=True)`

Loads all the plugins for the given namespace

#### **Parameters**

- **namespace** (*str*) – Namespace string, as in the setuptools entry\_points
- **instantiate** (*bool*) – If true, will instantiate the plugins too

**Returns** Returns the list of loaded plugins

**Return type** dict of str, object

#### **Submodules**

#### **lago.plugins.cli module**

About CLIPlugins

A CLIPlugin is a subcommand of the lagocli command, it's ment to group actions together in a logical sense, for example grouping all the actions done to templates.

To create a new subcommand for `testenvcli` you just have to subclass the `CLIPlugin` abstract class and declare it in the `setuptools` as an `entry_point`, see this module's `setup.py/setup.cfg` for an example:

```
class NoopCLIplugin(CLIPlugin):
    init_args = {
        'help': 'dummy help string',
    }

    def populate_parser(self, parser):
        parser.addArgument('--dummy-flag', action='store_true')

    def do_run(self, args):
        if args.dummy_flag:
            print "Dummy flag passed to noop subcommand!"
        else:
            print "Dummy flag not passed to noop subcommand!"
```

You can also use decorators instead, an equivalent is:

```
@cli_plugin_add_argument('--dummy-flag', action='store_true')
@cli_plugin(help='dummy help string')
def my_fancy_plugin_func(dummy_flag, **kwargs):
    if dummy_flag:
        print "Dummy flag passed to noop subcommand!"
    else:
        print "Dummy flag not passed to noop subcommand!"
```

Or:

```
@cli_plugin_add_argument('--dummy-flag', action='store_true')
def my_fancy_plugin_func(dummy_flag, **kwargs):
    "dummy help string"
    if dummy_flag:
        print "Dummy flag passed to noop subcommand!"
    else:
        print "Dummy flag not passed to noop subcommand!"
```

Then you will need to add an `entry_points` section in your `setup.py` like:

```
setup(
    ...
    entry_points={
        'lago.plugins.cli': [
            'noop=noop_module:my_fancy_plugin_func',
        ],
    }
    ...
)
```

Or in your `setup.cfg` like:

```
[entry_points]
lago.plugins.cli =
    noop=noop_module:my_fancy_plugin_func
```

Any of those will add a new subcommand to the `lagocli` command that can be run as:

```
$ lagocli noop
Dummy flag not passed to noop subcommand!
```

TODO: Allow per-plugin namespacing to get rid of the `**kwargs` parameter

```
class lago.plugins.cli.CLIPlugin
```

Bases: *lago.plugins.Plugin*

**\_abc\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache\_version** = 25

**\_abc\_registry** = <\_weakrefset.WeakSet object>

**do\_run** (*args*)

Execute any actions given the arguments

**Parameters** *args* (*Namespace*) – with the arguments

**Returns** None

**init\_args**

Dictionary with the argument to initialize the cli parser (for example, the help argument)

**populate\_parser** (*parser*)

Add any required arguments to the parser

**Parameters** *parser* (*ArgumentParser*) – parser to add the arguments to

**Returns** None

```
class lago.plugins.cli.CLIPluginFuncWrapper (do_run=None, init_args=None)
```

Bases: *lago.plugins.cli.CLIPlugin*

Special class to handle decorated cli plugins, take into account that the decorated functions have some limitations on what arguments can they define actually, if you need something complicated, used the abstract class *CLIPlugin* instead.

Keep in mind that right now the decorated function must use *\*\*kwargs* as param, as it will be passed all the members of the parser, not just whatever it defined

**\_\_call\_\_** (*\*args*, *\*\*kwargs*)

Keep the original function interface, so it can be used elsewhere

**\_abc\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache\_version** = 25

**\_abc\_registry** = <\_weakrefset.WeakSet object>

**add\_argument** (*\*argument\_args*, *\*\*argument\_kwargs*)

**do\_run** (*args*)

**init\_args**

**populate\_parser** (*parser*)

**set\_help** (*help=None*)

**set\_init\_args** (*init\_args*)

```
lago.plugins.cli.cli_plugin (func=None, **kwargs)
```

Decorator that wraps the given function in a *CLIPlugin*

**Parameters**

- **func** (*callable*) – function/class to decorate

- **\*\*kwargs** – Any other arg to use when initializing the parser (like help, or prefix\_chars)

**Returns** cli plugin that handles that method

**Return type** *CLIPlugin*

### Notes

It can be used as a decorator or as a decorator generator, if used as a decorator generator don't pass any parameters

### Examples

```
>>> @cli_plugin
... def test(**kwargs):
...     print 'test'
...
>>> print test.__class__
<class 'cli.CLIPluginFuncWrapper'>
```

```
>>> @cli_plugin()
... def test(**kwargs):
...     print 'test'
>>> print test.__class__
<class 'cli.CLIPluginFuncWrapper'>
```

```
>>> @cli_plugin(help='dummy help')
... def test(**kwargs):
...     print 'test'
>>> print test.__class__
<class 'cli.CLIPluginFuncWrapper'>
>>> print test.init_args['help']
'dummy help'
```

`lago.plugins.cli.cli_plugin_add_argument(*args, **kwargs)`

Decorator generator that adds an argument to the cli plugin based on the decorated function

### Parameters

- **\*args** – Any args to be passed to `argparse.ArgumentParser.add_argument()`
- **\*\*kwargs** – Any keyword args to be passed to `argparse.ArgumentParser.add_argument()`

**Returns** Decorator that builds or extends the cliplugin for the decorated function, adding the given argument definition

**Return type** function

### Examples

```
>>> @cli_plugin_add_argument('-m', '--mogambo', action='store_true')
... def test(**kwargs):
...     print 'test'
...
>>> print test.__class__
```



```
<class 'cli.CLIPluginFuncWrapper'>
>>> print test._parser_args
[('-', '--mogambo'), {'action': 'store_true'}]]
```

```
>>> @cli_plugin_add_argument('-', '--mogambo', action='store_true')
... @cli_plugin_add_argument('-b', '--bogabmo', action='store_false')
... @cli_plugin
... def test(**kwargs):
...     print 'test'
...
>>> print test.__class__
<class 'cli.CLIPluginFuncWrapper'>
>>> print test._parser_args
[('-', '--bogabmo'), {'action': 'store_false'}],
[('-', '--mogambo'), {'action': 'store_true'}]]
```

`lago.plugins.cli.cli_plugin_add_help` (*help*)

Decorator generator that adds the cli help to the cli plugin based on the decorated function

**Parameters** `help` (*str*) – help string for the cli plugin

**Returns** Decorator that builds or extends the cliplugin for the decorated function, setting the given help

**Return type** function

### Examples

```
>>> @cli_plugin_add_help('my help string')
... def test(**kwargs):
...     print 'test'
...
>>> print test.__class__
<class 'cli.CLIPluginFuncWrapper'>
>>> print test.help
my help string
```

```
>>> @cli_plugin_add_help('my help string')
... @cli_plugin()
... def test(**kwargs):
...     print 'test'
>>> print test.__class__
<class 'cli.CLIPluginFuncWrapper'>
>>> print test.help
my help string
```

## `lago.plugins.output` module

### About OutFormatPlugins

An OutFormatPlugin is used to format the output of the commands that extract information from the prefixes, like status.

`class lago.plugins.output.DefaultOutFormatPlugin`

Bases: `lago.plugins.output.OutFormatPlugin`

`_abc_cache = <_weakrefset.WeakSet object>`

```
_abc_negative_cache = <_weakrefset.WeakSet object>
_abc_negative_cache_version = 25
_abc_registry = <_weakrefset.WeakSet object>
format (info_obj, indent='')
indent_unit = ''
```

**class** `lago.plugins.output.FlatOutFormatPlugin`  
Bases: `lago.plugins.output.OutFormatPlugin`

```
_abc_cache = <_weakrefset.WeakSet object>
_abc_negative_cache = <_weakrefset.WeakSet object>
_abc_negative_cache_version = 25
_abc_registry = <_weakrefset.WeakSet object>
format (info_dict, delimiter='/')
```

This formatter will take a data structure that represent a tree and will print all the paths from the root to the leaves

in our case it will print each value and the keys that needed to get to it, for example:

**vm0:** net: lago memory: 1024

will be output as:

vm0/net/lago vm0/memory/1024

**Args:** `info_dict` (dict): information to reformat `delimiter` (str): a delimiter for the path components

**Returns:** str: String representing the formatted info

**class** `lago.plugins.output.JSONOutFormatPlugin`  
Bases: `lago.plugins.output.OutFormatPlugin`

```
_abc_cache = <_weakrefset.WeakSet object>
_abc_negative_cache = <_weakrefset.WeakSet object>
_abc_negative_cache_version = 25
_abc_registry = <_weakrefset.WeakSet object>
format (info_dict)
```

**class** `lago.plugins.output.OutFormatPlugin`  
Bases: `lago.plugins.Plugin`

```
_abc_cache = <_weakrefset.WeakSet object>
_abc_negative_cache = <_weakrefset.WeakSet object>
_abc_negative_cache_version = 25
_abc_registry = <_weakrefset.WeakSet object>
format (info_dict)
```

Execute any actions given the arguments

**Parameters** `info_dict` (*dict*) – information to reformat

**Returns** String representing the formatted info

Return type `str`

```
class lago.plugins.output.YAMLOutFormatPlugin
    Bases: lago.plugins.output.OutFormatPlugin
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 25
    _abc_registry = <_weakrefset.WeakSet object>
    format (info_dict)
```

#### `lago.plugins.service module`

**Service Plugin** This plugins are used in order to manage services in the vms

```
class lago.plugins.service.ServicePlugin (vm, name)
    Bases: lago.plugins.Plugin
    BIN_PATH
        Path to the binary used to manage services in the vm, will be checked for existence when trying to decide
        if the service is supported on the VM (see func:is_supported).
        Returns Full path to the binary inside the domain
        Return type str
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 25
    _abc_registry = <_weakrefset.WeakSet object>
    _request_start ()
        Low level implementation of the service start request, used by the func:start method
        Returns True if the service succeeded to start, False otherwise
        Return type bool
    _request_stop ()
        Low level implementation of the service stop request, used by the func:stop method
        Returns True if the service succeeded to stop, False otherwise
        Return type bool
    alive ()
    exists ()
    classmethod is_supported (vm)
    start ()
    state ()
        Check the current status of the service
        Returns Which state the service is at right now
        Return type ServiceState
```

```
stop ()
```

**class** `lago.plugins.service.ServiceState`  
Bases: `sphinx.ext.autodoc.Enum`

**ACTIVE** = 2

**INACTIVE** = 1

**MISSING** = 0

This state corresponds to a service that is not available in the domain

### `lago.plugins.vm` module

**VM Plugins** There are two VM-related plugin extension points, there's the VM Type Plugin, that allows you to modify at a higher level the inner workings of the VM class (domain concept in the initfile). The other plugin extension point, the [VM Provider Plugin], that allows you to create an alternative implementation of the provisioning details for the VM, for example, using a remote libvirt instance or similar.

```
exception lago.plugins.vm.ExtractPathError
    Bases: lago.plugins.vm.VMError
```

```
exception lago.plugins.vm.ExtractPathNoPathError
    Bases: lago.plugins.vm.VMError
```

```
exception lago.plugins.vm.VMError
    Bases: exceptions.Exception
```

```
class lago.plugins.vm.VMPlugin (env, spec)
    Bases: lago.plugins.Plugin
```

```
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 25
    _abc_registry = <_weakrefset.WeakSet object>
    _artifact_paths ()
    _detect_service_provider ()
    _get_service_provider ()
        NOTE: Can be reduced to just one get call once we remove support for the service_class spec entry
        Returns class for the loaded provider for that vm_spec None: if no provider was specified in the
            vm_spec
        Return type class
    _get_vm_provider ()
    classmethod _normalize_spec (spec)
    _scp (*args, **kws)
    _template_metadata ()
    alive ()
    all_ips ()
    bootstrap (*args, **kwargs)
        Thin method that just uses the provider
```

**collect\_artifacts** (*host\_path*, *ignore\_nopath*)  
**copy\_from** (*remote\_path*, *local\_path*, *recursive=True*, *propagate\_fail=True*)  
**copy\_to** (*local\_path*, *remote\_path*, *recursive=True*)  
**create\_snapshot** (*name*, *\*args*, *\*\*kwargs*)  
 Thin method that just uses the provider  
**defined** (*\*args*, *\*\*kwargs*)  
 Thin method that just uses the provider  
**distro** ()  
**extract\_paths** (*paths*, *\*args*, *\*\*kwargs*)  
 Thin method that just uses the provider  
**guest\_agent** ()  
**has\_guest\_agent** ()  
**interactive\_console** (*\*args*, *\*\*kwargs*)  
 Thin method that just uses the provider  
**interactive\_ssh** (*\*args*, *\*\*kwargs*)  
**ip** ()  
**iscsi\_name** ()  
**metadata**  
**name** ()  
**nets** ()  
**nics** ()  
**revert\_snapshot** (*name*, *\*args*, *\*\*kwargs*)  
 Thin method that just uses the provider  
**root\_password** ()  
**save** (*path=None*)  
**service** (*\*args*, *\*\*kwargs*)  
**ssh** (*command*, *data=None*, *show\_output=True*, *propagate\_fail=True*, *tries=None*)  
**ssh\_reachable** (*\*args*, *\*\*kwargs*)  
 Check if the VM is reachable with ssh

#### Parameters

- **tries** (*int*) – Number of tries to try connecting to the host
- **propagate\_fail** (*bool*) – If set to true, this event will appear
- **the log and fail the outter stage. Otherwise, it will be** (*in*) –
- **discarded.** –

**Returns** True if the VM is reachable.

**Return type** *bool*

**ssh\_script** (*path*, *show\_output=True*)

**start** (\*args, \*\*kwargs)  
Thin method that just uses the provider

**state** (\*args, \*\*kwargs)  
Thin method that just uses the provider

**stop** (\*args, \*\*kwargs)  
Thin method that just uses the provider

**wait\_for\_ssh**()

**class** `lago.plugins.vm.VMProviderPlugin`(vm)  
Bases: `lago.plugins.Plugin`

If you want to use a custom provider for you VMs (say, ovirt for example), you have to inherit from this class, and then define the 'default\_vm\_provider' in your config to be your plugin, or explicitly specify it on each domain definition in the initfile with 'vm-provider' key

You will have to override at least all the abstractmethods in order to write a provider plugin, even if they are just running *pass*.

**\_extract\_paths\_scp**(paths, ignore\_nopath)

**bootstrap** (\*args, \*\*kwargs)  
Does any actions needed to get the domain ready to be used, ran on prefix init.

**Returns** None

**create\_snapshot**(name, \*args, \*\*kwargs)  
Take any actions needed to create a snapshot

**Parameters** **name** (*str*) – Name for the snapshot, will be used as key to retrieve it later

**Returns** None

**defined** (\*args, \*\*kwargs)  
Return if the domain is defined (libvirt concept), currently used only by the libvirt provider, put here to allow backwards compatibility.

**Returns** True if the domain is already defined (libvirt concept)

**Return type** `bool`

**extract\_paths**(paths, ignore\_nopath)  
Extract the given paths from the domain

**Parameters**

- **paths** (*list of str*) – paths to extract
- **ignore\_nopath** (*boolean*) – if True will ignore none existing paths.

**Returns** None

**Raises**

- `ExtractPathNoPathError` – if a none existing path was found on the VM, and `ignore_nopath` is True.
- `ExtractPathError` – on all other failures.

**interactive\_console**()  
Run an interactive console

**Returns** result of the interactive execution

**Return type** `lago.utils.CommandStatus`

**revert\_snapshot** (*name*, \**args*, \*\**kwargs*)

Take any actions needed to revert/restore a snapshot

**Parameters** **name** (*str*) – Name for the snapshot, same that was set on creation

**Returns** None

**start** (\**args*, \*\**kwargs*)

Start a domain

**Returns** None

**state** (\**args*, \*\**kwargs*)

Return the current state of the domain

**Returns** Small description of the current domain state

**Return type** *str*

**stop** (\**args*, \*\**kwargs*)

Stop a domain

**Returns** None

`lago.plugins.vm._check_alive` (*func*)

`lago.plugins.vm._resolve_service_class` (*class\_name*, *service\_providers*)

**NOTE:** This must be removed once the `service_class` spec entry is fully deprecated

Retrieves a service plugin class from the class name instead of the provider name

**Parameters**

- **class\_name** (*str*) – Class name of the service plugin to retrieve
- **service\_providers** (*dict*) – `provider_name->provider_class` of the loaded service providers

**Returns** Class of the plugin that matches that name

**Return type** *class*

**Raises** `lago.plugins.NoSuchPluginError` – if there was no service plugin that matched the search

## 4.1.2 Submodules

### 4.1.3 lago.brctl module

`lago.brctl._brctl` (*command*, \**args*)

`lago.brctl._set_link` (*name*, *state*)

`lago.brctl.create` (*name*, *stp=True*)

`lago.brctl.destroy` (*name*)

`lago.brctl.exists` (*name*)

#### 4.1.4 lago.cmd module

```
lago.cmd.check_deps()
lago.cmd.check_group_membership()
lago.cmd.create_parser(cli_plugins, out_plugins)
lago.cmd.main()
```

#### 4.1.5 lago.config module

```
class lago.config.ConfigLoad(root_section='lago')
    Bases: object
```

Merges configuration parameters from 3 different sources: 1. Environment variables 2. config files in .INI format 3. argparse.ArgumentParser

The assumed order (but not necessary) order of calls is: load() - load from config files and environment variables update\_parser(parser) - update from the declared argparse parser update\_args(args) - update from passed arguments to the parser

```
__getitem__(key)
```

Get a variable from the default section, good for fail-fast if key does not exist.

**Parameters** **key** (*str*) – key

**Returns** config variable

**Return type** *str*

```
get(*args)
```

Get a variable from the default section :param \*args: dict.get() args

**Returns** config variable

**Return type** *str*

```
get_ini(defaults_only=False, incl_unset=False)
```

Return the config dictionary in INI format :param defaults\_only: if set, will ignore arguments set by the CLI.

**Returns** string of the config file in INI format

**Return type** *str*

```
get_section(*args)
```

get a section dictionary Args:

**Returns** section config dictionary

**Return type** *dict*

```
load()
```

Load all configuration from INI format files and ENV, always preferring the last read. Order of loading is: 1) Custom paths as defined in constants.CONFS\_PATH 2) XDG standard paths 3) Environment variables

**Returns** dict of section configuration dicts

**Return type** *dict*

```
update_args(args)
```

Update config dictionary with parsed args, as resolved by argparse. Only root positional arguments that already exist will be overridden.



**Parameters** `args` (*namespace*) – args parsed by argparse

**update\_parser** (*parser*)

Update config dictionary with declared arguments in an `argparse.parser`. New variables will be created, and existing ones overridden.

**Parameters** `parser` (*`argparse.ArgumentParser`*) – parser to read variables from

`lago.config._get_configs_path()`

Get a list of possible configuration files, from the following sources: 1. All files that exists in `constants.CONFS_PATH`. 2. All XDG standard config files for “lago.conf”, in reversed order of importance.

**Returns** `list` – list of files

**Return type** `str`

`lago.config.get_env_dict(root_section)`

Read all Lago variables from the environment. The lookup format is: `LAGO_VARNAME` - will land into ‘lago’ section `LAGO__SECTION1__VARNAME` - will land into ‘section1’ section, notice the double ‘\_\_’. `LAGO__LONG_SECTION_NAME__VARNAME` - will land into ‘long\_section\_name’

**Returns** `dict` of section configuration dicts

**Return type** `dict`

## Examples

```
>>> os.environ['LAGO_GLOBAL_VAR'] = 'global'
>>> os.environ['LAGO__INIT__REPO_PATH'] = '/tmp/store'
>>>
>>> config.get_env_dict()
{'init': {'repo_path': '/tmp/store'}, 'lago': {'global_var': 'global'}}
```

## 4.1.6 lago.constants module

`lago.constants.CONFS_PATH = ['/etc/lago/lago.conf']`

`CONFS_PATH` - default path to first look for configuration files.

`lago.constants.LIBEXEC_DIR = '/usr/libexec/lago/'`

`LIBEXEC_DIR` -

## 4.1.7 lago.dirlock module

`lago.dirlock._lock_path(path)`

`lago.dirlock.lock(path, excl, key_path)`

Waits until the given directory can be locked

**Parameters**

- **path** (*str*) – Path of the directory to lock
- **excl** (*bool*) – If the lock should be exclusive
- **key\_path** (*str*) – path to the file that contains the uid to use when locking

**Returns** `None`

`lago.dirlock.trylock(path, excl, key_path)`

Tries once to get a lock to the given dir

**Parameters**

- **path** (*str*) – path to the directory to lock
- **excl** (*bool*) – If the lock should be exclusive
- **key\_path** (*str*) – path to the file that contains the uid to use when locking

**Returns** True if it did get a lock, False otherwise

**Return type** `bool`

`lago.dirlock.unlock(path, key_path)`

Removes the lock of the uid in the given key file

**Parameters**

- **path** (*str*) – Path of the directory to lock
- **key\_path** (*str*) – path to the file that contains the uid to remove the lock of

**Returns** None

## 4.1.8 lago.libvirt\_utils module

Utilities to help deal with the libvirt python bindings

`lago.libvirt_utils.DOMAIN_STATES = {<class 'sphinx.ext.autodoc.VIR_DOMAIN_CRASHED'>: 'crashed', <class 'sp`

Mapping of domain statuses values to human readable strings

**class** `lago.libvirt_utils.Domain`

Bases: `object`

Class to namespace libvirt domain related helpers

**static** `resolve_state(state_number)`

Get a nice description from a domain state number

**Parameters** **state\_number** (*list of int*) – State number as returned by `libvirt.virDomain.state()`

**Returns** small human readable description of the domain state, unknown if the state is not in the known list

**Return type** `str`

`lago.libvirt_utils.LIBVIRT_CONNECTIONS = {}`

Singleton with the cached opened libvirt connections

`lago.libvirt_utils.auth_callback(credentials, user_data)`

`lago.libvirt_utils.get_libvirt_connection(name, libvirt_url='qemu://system')`

## 4.1.9 lago.log\_utils module

This module defines the special logging tools that lago uses

`lago.log_utils.ALWAYS_SHOW_REG = <sre.SRE_Pattern object>`

Regex that will match the above template

`lago.log_utils.ALWAYS_SHOW_TRIGGER_MSG = 'force-show:%s'`

Message template that will always show the message

**class** `lago.log_utils.ColorFormatter` (*fmt=None, datefmt=None*)

Bases: `logging.Formatter`

Formatter to add colors to log records

**CRITICAL** = `'\x1b[31m'`

**CYAN** = `'\x1b[36m'`

**DEBUG** = `''`

**DEFAULT** = `'\x1b[0m'`

**ERROR** = `'\x1b[31m'`

**GREEN** = `'\x1b[32m'`

**INFO** = `'\x1b[36m'`

**NONE** = `''`

**RED** = `'\x1b[31m'`

**WARNING** = `'\x1b[33m'`

**WHITE** = `'\x1b[37m'`

**YELLOW** = `'\x1b[33m'`

**classmethod** `colored` (*color, message*)

Small function to wrap a string around a color

**Parameters**

- **color** (*str*) – name of the color to wrap the string with, must be one of the class properties
- **message** (*str*) – String to wrap with the color

**Returns** the colored string

**Return type** `str`

**format** (*record*)

Adds colors to a log record and formats it with the default

**Parameters** **record** (*logging.LogRecord*) – log record to format

**Returns** The colored and formatted record string

**Return type** `str`

**class** `lago.log_utils.ContextLock`

Bases: `object`

Context manager to thread lock a block of code

`lago.log_utils.END_TASK_MSG = 'Success'`

Message to be shown when a task is ended

`lago.log_utils.END_TASK_REG = <_sre.SRE_Pattern object>`

Regex that will match the above template

`lago.log_utils.END_TASK_TRIGGER_MSG = 'end task %s'`

Message template that will trigger a task end

```
class lago.log_utils.LogTask(task, logger=<module 'logging' from
                             'usr/lib/python2.7/logging/__init__.pyc'>,
                             level='info', propagate_fail=True, uuid=None)
```

Bases: `object`

Context manager for a log task

### Example

```
>>> with LogTask('mytask'):
...     pass
```

```
lago.log_utils.START_TASK_MSG = ''
    Message to be shown when a task is started
```

```
lago.log_utils.START_TASK_REG = <_sre.SRE_Pattern object>
    Regex that will match the above template
```

```
lago.log_utils.START_TASK_TRIGGER_MSG = 'start task%s'
    Message template that will trigger a task
```

```
class lago.log_utils.Task(name, *args, **kwargs)
    Bases: collections.deque
```

Small wrapper around deque to add the failed status and name to a task

**name**  
*str*  
name for this task

**failed**  
*bool*  
If this task has failed or not (if there was any error log shown during it's execution)

**force\_show**  
*bool*  
If set, will show any log records generated inside this task even if it's out of nested depth limit

**elapsed\_time()**

```
class lago.log_utils.TaskHandler(initial_depth=0, task_tree_depth=-1, buffer_size=2000,
                                  dump_level=40, level=0, formatter=<class
                                  'lago.log_utils.ColorFormatter'>)
```

Bases: `logging.StreamHandler`

This log handler will use the concept of tasks, to hide logs, and will show all the logs for the current task if there's a logged error while running that task.

It will hide any logs that belong to nested tasks that have more than `task_tree_depth` parent levels, and for the ones that are above that level, it will show only the logs that have a loglevel above `level`.

You can force showing a log record immediately if you use the `log_always()` function bypassing all the filters.

If there's a log record with log level higher than `dump_level` it will be considered a failure, and all the logs for the current task that have a log level above `level` will be shown no matter at which depth the task belongs to. Also, all the parent tasks will be tagged as error.

**formatter***logging.LogFormatter*

formatter to use

**initial\_depth***int*

Initial depth to account for, in case this handler was created in a subtask

**tasks\_by\_thread (dict of str**

OrderedDict of str: Task): List of thread names, and their currently open tasks with their latest log records

**dump\_level***int*

log level from which to consider a log record as error

**buffer\_size***int*

Size of the log record deque for each task, the bigger, the more records it can show in case of error but the more memory it will use

**task\_tree\_depth***int*

number of the nested level to show start/end task logs for, if -1 will show always

**level***int*

Log level to show logs from if the depth limit is not reached

**main\_failed***bool*

used to flag from a child thread that the main should fail any current task

**\_tasks\_lock***ContextLock*

Lock for the tasks\_by\_thread dict

**\_main\_thread\_lock***ContextLock*

Lock for the main\_failed bool

**TASK\_INDICATORS = ['@', '#', '\*', '-', '~']**

List of chars to show as task prefix, to ease distinguishing them

**am\_i\_main\_thread****Returns** if the current thread is the main thread**Return type** *bool***close\_children\_tasks (parent\_task\_name)**

Closes all the children tasks that were open

**Parameters** *parent\_task\_name* (*str*) – Name of the parent task**Returns** *None***cur\_depth\_level**

**Returns** depth level for the current task

**Return type** `int`

**cur\_task**

**Returns** the current active task

**Return type** `str`

**cur\_thread**

**Returns** Name of the current thread

**Return type** `str`

**emit** (*record*)

Handle the given record, this is the entry point from the python logging facility

**Params:** record (logging.LogRecord): log record to handle

**Returns** None

**get\_task\_indicator** (*task\_level=None*)

**Parameters** **task\_level** (*int or None*) – task depth level to get the indicator for, if None, will use the current tasks depth

**Returns** char to prepend to the task logs to indicate it's level

**Return type** `str`

**get\_tasks** (*thread\_name*)

**Parameters** **thread\_name** (*str*) – name of the thread to get the tasks for

**Returns** list of task names and log records for each for the given thread

**Return type** OrderedDict of str, Task

**handle\_closed\_task** (*task\_name, record*)

Do everything needed when a task is closed

**Params:** task\_name (str): name of the task that is finishing record (logging.LogRecord): log record with all the info

**Returns** None

**handle\_error** ()

Handles an error log record that should be shown

**Returns** None

**handle\_new\_task** (*task\_name, record*)

Do everything needed when a task is starting

**Params:** task\_name (str): name of the task that is starting record (logging.LogRecord): log record with all the info

**Returns** None

**mark\_main\_tasks\_as\_failed** ()

Flags to the main thread that all it's tasks sholud fail

**Returns** None

**mark\_parent\_tasks\_as\_failed** (*task\_name*, *flush\_logs=False*)

Marks all the parent tasks as failed

**Parameters**

- **task\_name** (*str*) – Name of the child task
- **flush\_logs** (*bool*) – If True will discard all the logs form parent tasks

**Returns** None

**pretty\_emit** (*record*, *is\_header=False*, *task\_level=None*)

Wrapper around the `logging.StreamHandler` emit method to add some decoration stuff to the message

**Parameters**

- **record** (*logging.LogRecord*) – log record to emit
- **is\_header** (*bool*) – if this record is a header, usually, a start or end task message
- **task\_level** (*int*) – If passed, will take that as the current nested task level instead of calculating it from the current tasks

**Returns** None

**should\_show\_by\_depth** (*cur\_level=None*)

**Parameters** **cur\_level** (*int*) – depth level to take into account

**Returns** True if the given depth level should show messages (not taking into account the log level)

**Return type** `bool`

**should\_show\_by\_level** (*record\_level*, *base\_level=None*)

**Parameters**

- **record\_level** (*int*) – log level of the record to check
- **base\_level** (*int or None*) – log level to check against, will use the object's `dump_level` if None is passed

**Returns** True if the given log record should be shown according to the log level

**Return type** `bool`

**tasks**

**Returns** list of task names and log records for each for the current thread

**Return type** `OrderedDict` of `str`, `Task`

`lago.log_utils.end_log_task` (*task*, *logger=<module 'logging' from 'usr/lib/python2.7/logging/\_\_init\_\_.pyc'>*, *level='info'*)

Ends a log task

**Parameters**

- **task** (*str*) – name of the log task to end
- **logger** (*logging.Logger*) – logger to use
- **level** (*str*) – log level to use

**Returns** None

`lago.log_utils.hide_paramiko_logs` ()

`lago.log_utils.hide_stevedore_logs()`

Hides the logs of stevedore, this function was added in order to support older versions of stevedore

We are using the NullHandler in order to get rid from 'No handlers could be found for logger...' msg

**Returns** None

`lago.log_utils.log_always(message)`

Wraps the given message with a tag that will make it be always logged by the task logger

**Parameters** `message` (*str*) – message to wrap with the tag

**Returns** tagged message that will get it shown immediately by the task logger

**Return type** *str*

`lago.log_utils.log_task(task, logger=<module 'logging' from  
'usr/lib/python2.7/logging/__init__.pyc'>, level='info',  
gate_fail=True, uuid=None)`

Parameterized decorator to wrap a function in a log task

### Example

```
>>> @log_task('mytask')
... def do_something():
...     pass
```

`lago.log_utils.setup_prefix_logging(logdir)`

Sets up a file logger that will create a log in the given logdir (usually a lago prefix)

**Parameters** `logdir` (*str*) – path to create the log into, will be created if it does not exist

**Returns** None

`lago.log_utils.start_log_task(task, logger=<module 'logging' from  
'usr/lib/python2.7/logging/__init__.pyc'>, level='info')`

Starts a log task

**Parameters**

- **task** (*str*) – name of the log task to start
- **logger** (*logging.Logger*) – logger to use
- **level** (*str*) – log level to use

**Returns** None

## 4.1.10 lago.paths module

`class lago.paths.Paths(prefix)`

Bases: *object*

**images** (*\*path*)

**logs** ()

**metadata** ()

**prefix\_lagofile** ()

This file represents a prefix that's initialized

**prefixed** (*\*args*)



```

scripts(*args)
ssh_id_rsa()
ssh_id_rsa_pub()
uuid()
virt(*path)

```

#### 4.1.11 lago.prefix module

**class** `lago.prefix.Prefix(prefix)`

Bases: `object`

A prefix is a directory that will contain all the data needed to setup the environment.

**`__prefix`**

*str*

Path to the directory of this prefix

**`__paths`**

*lago.path.Paths*

Path handler class

**`__virt_env`**

*lago.virt.VirtEnv*

Lazily loaded virtual env handler

**`__metadata`**

*dict*

Lazily loaded metadata

**`VIRT_ENV_CLASS`**

alias of `VirtEnv`

**`__add_nic_to_mapping`**(*net, dom, nic*)

Populates the given net spec mapping entry with the nicks of the given domain

**Parameters**

- **net** (*dict*) – Network spec to populate
- **dom** (*dict*) – libvirt domain specification
- **nic** (*str*) – Name of the interface to add to the net mapping from the domain

**Returns** `None`

**`__allocate_ips_to_nics`**(*conf*)

For all the nics of all the domains in the conf that have dynamic ip, allocate one and addit to the network mapping

**Parameters** **conf** (*dict*) – Configuration spec to extract the domains from

**Returns** `None`

**`__allocate_subnets`**(*conf*)

Allocate all the subnets needed by the given configuration spec

**Parameters** **conf** (*dict*) – Configuration spec where to get the nets definitions from

**Returns** `tuple` – allocated subnets and modified conf

**Return type** `list, dict`

**static** `_check_predefined_subnets` (*conf*)

Checks if all of the nets defined in the config are inside the allowed range, throws exception if not

**Parameters** `conf` (*dict*) – Configuration spec where to get the nets definitions from

**Returns** `None`

**Raises** `RuntimeError` – If there are any subnets out of the allowed range

**\_config\_net\_topology** (*conf*)

Initialize and populate all the network related elements, like reserving ips and populating network specs of the given configuration spec

**Parameters** `conf` (*dict*) – Configuration spec to initialize

**Returns** `None`

**\_copy\_delpoy\_scripts** (*scripts*)

Copy the given deploy scripts to the scripts dir in the prefix

**Parameters** `scripts` (*list of str*) – list of paths of the scripts to copy to the prefix

**Returns** `list` with the paths to the copied scripts, with a prefixed with `$LAGO_PREFIX_PATH` so the full path is not hardcoded

**Return type** `list of str`

**\_copy\_deploy\_scripts\_for\_hosts** (*domains*)

Copy the deploy scripts for all the domains into the prefix scripts dir

**Parameters** `domains` (*dict*) – spec with the domains info as when loaded from the initfile

**Returns** `None`

**\_create\_disk** (*name, spec, template\_repo=None, template\_store=None*)

Creates a disk with the given name from the given repo or store

**Parameters**

- **name** (*str*) – Name of the domain to create the disk for
- **spec** (*dict*) – Specification of the disk to create
- **template\_repo** (*TemplateRepository or None*) – template repo instance to use
- **template\_store** (*TemplateStore or None*) – template store instance to use

**Returns** `Tuple` – Path to the disk and disk metadata

**Return type** `str, dict`

**Raises** `RuntimeError` – If the type of the disk is not supported or failed to create the disk

**\_create\_disks** (*domain\_name, disks\_specs, template\_repo, template\_store*)

**\_create\_ssh\_keys** ()

Generate a pair of ssh keys for this prefix

**Returns** `None`

**Raises** `RuntimeError` – if it fails to create the keys

**`_create_virt_env()`**

Create a new virt env from this prefix

**Returns** virt env created from this prefix

**Return type** *lago.virt.VirtEnv*

**`_deploy_host(host)`**

**`static _generate_disk_name(host_name, disk_name, disk_format)`**

**`_generate_disk_path(disk_name)`**

**`_get_metadata()`**

Retrieve the metadata info for this prefix

**Returns** metadata info

**Return type** *dict*

**`_get_scripts(host_metadata)`**

Temporary method to retrieve the host scripts

**TODO:** remove once the “ovirt-scripts” option gets deprecated

**Parameters** **`host_metadata`** (*dict*) – host metadata to retrieve the scripts for

**Returns** deploy scripts for the host, empty if none found

**Return type** *list*

**`_handle_empty_disk(host_name, disk_spec)`**

**`_handle_file_disk(disk_spec)`**

**`_handle_lago_template(disk_path, template_spec, template_store, template_repo)`**

**`_handle_ova_image(domain_spec)`**

**`_handle_qcow_template(disk_path, template_spec)`**

**`_handle_template(host_name, template_spec, template_store=None, template_repo=None)`**

**`static _init_net_specs(conf)`**

Given a configuration specification, initializes all the net definitions in it so they can be used comfortably

**Parameters** **`conf`** (*dict*) – Configuration specification

**Returns** the adapted new conf

**Return type** *dict*

**`_ova_to_spec(filename)`**

Retrieve the given ova and makes a template of it. Creates a disk from network provided ova. Calculates the needed memory from the ovf. The disk will be cached in the template repo

**Parameters** **`filename`** (*str*) – the url to retrieve the data from

**TODO:**

- Add hash checking against the server for faster download and latest version
- Add config script running on host - other place
- Add cloud init support - by using cdroms in other place
- Handle cpu in some way - some other place need to pick it up

- Handle the memory units properly - we just assume MegaBytes

**Returns** list with the disk specification int: VM memory, None if none defined int: Number of virtual cpus, None if none defined

**Return type** list of dict

**Raises**

- `RuntimeError` – If the ova format is not supported
- `TypeError` – If the memory units in the ova are not supported (currently only ‘MegaBytes’)

**`_prepare_domain_image`** (*domain\_spec, prototypes, template\_repo, template\_store*)

**`_prepare_domains_images`** (*conf, template\_repo, template\_store*)

**`_register_preallocated_ips`** (*conf*)

Parse all the domains in the given conf and preallocate all their ips into the networks mappings, raising exception on duplicated ips or ips out of the allowed ranges

**See also:**

*lago.subnet\_lease*

**Parameters** *conf* (*dict*) – Configuration spec to parse

**Returns** None

**Raises** `RuntimeError` – if there are any duplicated ips or any ip out of the allowed range

**`_retrieve_disk_url`** (*disk\_url, disk\_dst\_path=None*)

**`static _run_qemu`** (*qemu\_cmd, disk\_path*)

**`_save_metadata`** ()

Write this prefix metadata to disk

**Returns** None

**`_set_scripts`** (*host\_metadata, scripts*)

Temporary method to set the host scripts

**TODO:** remove once the “ovirt-scripts” option gets deprecated

**Parameters** *host\_metadata* (*dict*) – host metadata to set scripts in

**Returns** the updated metadata

**Return type** *dict*

**`_use_prototype`** (*spec, prototypes*)

Populates the given spec with the values of it’s declared prototype

**Parameters**

- *spec* (*dict*) – spec to update
- *prototypes* (*dict*) – Configuration spec containing the prototypes

**Returns** updated spec

**Return type** *dict*

**cleanup** (\*args, \*\*kwargs)

Stops any running entities in the prefix and uninitializes it, usually you want to do this if you are going to remove the prefix afterwards

**Returns** None

**collect\_artifacts** (\*args, \*\*kwargs)

**create\_snapshots** (name)

Creates one snapshot on all the domains with the given name

**Parameters** **name** (*str*) – Name of the snapshots to create

**Returns** None

**deploy** (\*args, \*\*kwargs)

**destroy** ()

Destroy this prefix, running any cleanups and removing any files inside it.

**fetch\_url** (url)

Retrieves the given url to the prefix

**Parameters** **url** (*str*) – Url to retrieve

**Returns** path to the downloaded file

**Return type** *str*

**get\_nets** ()

Retrieve info on all the nets from all the domains

**Returns** **dict of str->list** – dictionary with net\_name -> net list

**Return type** *str*

**get\_snapshots** ()

Retrieve info on all the snapshots from all the domains

**Returns** list(str): dictionary with vm\_name -> snapshot list

**Return type** dict of str

**get\_vms** ()

Retrieve info on all the vms

**Returns** **dict of str->list** – dictionary with vm\_name -> vm list

**Return type** *str*

**initialize** (\*args, \*\*kwargs)

Initialize this prefix, this includes creating the destination path, and creating the uuid for the prefix, for any other actions see [Prefix.virt\\_conf\(\)](#)

Will safely roll back if any of those steps fail

**Returns** None

**Raises** `RuntimeError` – If it fails to create the prefix dir

**classmethod** **is\_prefix** (path)

Check if a path is a valid prefix

**Parameters** **path** (*str*) – path to be checked

**Returns** True if the given path is a prefix

**Return type** *bool*

**classmethod** `resolve_prefix_path` (*start\_path=None*)

Look for an existing prefix in the given path, in a path/.lago dir, or in a .lago dir under any of it's parent directories

**Parameters** `start_path` (*str*) – path to start the search from, if None passed, it will use the current dir

**Returns** path to the found prefix

**Return type** *str*

**Raises** `RuntimeError` – if no prefix was found

**revert\_snapshots** (*name*)

Revert all the snapshots with the given name from all the domains

**Parameters** `name` (*str*) – Name of the snapshots to revert

**Returns** None

**save** ()

Save this prefix to persistent storage

**Returns** None

**start** (*vm\_names=None*)

Start this prefix

**Parameters** `vm_names` (*list of str*) – List of the vms to start

**Returns** None

**stop** (*vm\_names=None*)

Stop this prefix

**Parameters** `vm_names` (*list of str*) – List of the vms to stop

**Returns** None

**virt\_conf** (*conf, template\_repo=None, template\_store=None, do\_bootstrap=True*)

Initializes all the virt infrastructure of the prefix, creating the domains disks, doing any network leases and creating all the virt related files and dirs inside this prefix.

**Parameters**

- `conf` (*dict*) – Configuration spec
- `template_repo` (*TemplateRepository*) – template repository instance
- `template_store` (*TemplateStore*) – template store instance

**Returns** None

**virt\_conf\_from\_stream** (*conf\_fd, template\_repo=None, template\_store=None, do\_bootstrap=True*)

Initializes all the virt infrastructure of the prefix, creating the domains disks, doing any network leases and creating all the virt related files and dirs inside this prefix.

**Parameters**

- `conf_fd` (*File*) – File like object to read the config from
- `template_repo` (*TemplateRepository*) – template repository instance
- `template_store` (*TemplateStore*) – template store instance

**Returns** None

**virt\_env**

Getter for this instance's virt env, creates it if needed

**Returns** virt env instance used by this prefix

**Return type** *lago.virt.VirtEnv*

`lago.prefix._create_ip(subnet, index)`

Given a subnet or an ip and an index returns the ip with that lower index from the subnet (255.255.255.0 mask only subnets)

**Parameters**

- **subnet** (*str*) – Strign containing the three first elements of the decimal representation of a subnet (X.Y.Z) or a full ip (X.Y.Z.A)
- **index** (*int or str*) – Last element of a decimal ip representation, for example, 123 for the ip 1.2.3.123

**Returns** The dotted decimal representation of the ip

**Return type** *str*

`lago.prefix._ip_in_subnet(subnet, ip)`

Checks if an ip is included in a subnet.

---

**Note:** only 255.255.255.0 masks allowed

---

**Parameters**

- **subnet** (*str*) – Strign containing the three first elements of the decimal representation of a subnet (X.Y.Z) or a full ip (X.Y.Z.A)
- **ip** (*str or int*) – Decimal ip representation

**Returns** True if ip is in subnet, False otherwise

**Return type** *bool*

## 4.1.12 lago.service module

`class lago.service.SysVInitService(vm, name)`

Bases: *lago.plugins.service.ServicePlugin*

**BIN\_PATH** = '/sbin/service'

**\_abc\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache\_version** = 25

**\_abc\_registry** = <\_weakrefset.WeakSet object>

**\_request\_start** ()

**\_request\_stop** ()

**state** ()

`class lago.service.SystemdContainerService(vm, name)`

Bases: *lago.plugins.service.ServicePlugin*

```
BIN_PATH = '/usr/bin/docker'
HOST_BIN_PATH = '/usr/bin/systemctl'
_abc_cache = <_weakrefset.WeakSet object>
_abc_negative_cache = <_weakrefset.WeakSet object>
_abc_negative_cache_version = 25
_abc_registry = <_weakrefset.WeakSet object>
_request_start()
_request_stop()
state()

class lago.service.SystemdService(vm, name)
    Bases: lago.plugins.service.ServicePlugin
    BIN_PATH = '/usr/bin/systemctl'
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 25
    _abc_registry = <_weakrefset.WeakSet object>
    _request_start()
    _request_stop()
    state()
```

### 4.1.13 lago.ssh module

```
lago.ssh._gen_ssh_command_id()
lago.ssh._get_cipher(self, name, key, iv)
    Workaround for BZ: https://bugzilla.redhat.com/show\_bug.cgi?id=1419312 This monkeypatching was tested to
    resolve the issue with the combination of: paramiko-1.16.1 python2-crypto-2.6.1-13.el7
lago.ssh.drain_ssh_channel(chan, stdin=None, stdout=<open file '<stdout>', mode 'w'>,
                           stderr=<open file '<stderr>', mode 'w'>)
lago.ssh.get_ssh_client(ip_addr, ssh_key=None, host_name=None, ssh_tries=None, propa-
                        gate_fail=True, username='root', password='123456')
lago.ssh.interactive_ssh(ip_addr, command=None, host_name=None, ssh_key=None, user-
                        name='root', password='123456')
lago.ssh.interactive_ssh_channel(chan, command=None, stdin=<open file '<stdin>', mode
                                'r'>)
lago.ssh.ssh(ip_addr, command, host_name=None, data=None, show_output=True, propa-
             gate_fail=True, tries=None, ssh_key=None, username='root', password='123456')
lago.ssh.ssh_script(ip_addr, path, host_name=None, show_output=True, ssh_key=None, user-
                   name='root', password='123456')
lago.ssh.wait_for_ssh(ip_addr, host_name=None, connect_timeout=600, ssh_key=None, user-
                     name='root', password='123456')
```



#### 4.1.14 lago.subnet\_lease module

Module that handles the leases for the subnets of the virtual network interfaces.

---

**Note:** Currently only /24 ranges are handled, and all of them under the 192.168.MIN\_SUBNET to 192.168.MAX\_SUBNET ranges

---

The leases are stored under LEASE\_DIR as json files with the form:

```
[
    "/path/to/prefix/uuid/file",
    "uuid_hash",
]
```

Where the *uuid\_hash* is the 32 char uuid of the prefix (the contents of the uuid file at the time of doing the lease)

`lago.subnet_lease.MAX_SUBNET = 209`

Upper range for the allowed subnets

`lago.subnet_lease.MIN_SUBNET = 200`

Lower range for the allowed subnets

`lago.subnet_lease._acquire(*args, **kwargs)`

Lease a free network for the given uuid path

**Parameters** `uuid_path (str)` – Path to the uuid file of a `lago.Prefix`

**Returns** the third element of the dotted ip of the leased network or `None` if no lease was available

**Return type** `int` or `None`

`lago.subnet_lease._lease_owned(path, current_uuid_path)`

Checks if the given lease is owned by the prefix whose uuid is in the given path

---

**Note:** The prefix must be also in the same path it was when it took the lease

---

##### Parameters

- `path (str)` – Path to the lease
- `current_uuid_path (str)` – Path to the uuid to check ownership of

**Returns** `True` if the given lease is owned by the prefix, `False` otherwise

**Return type** `bool`

`lago.subnet_lease._lease_valid(path)`

Checks if the given lease still has a prefix that owns it

**Parameters** `path (str)` – Path to the lease

**Returns** `True` if the uuid path in the lease still exists and is the same as the one in the lease

**Return type** `bool`

`lago.subnet_lease._locked(func)`

Decorator that will make sure that you have the exclusive lock for the leases

`lago.subnet_lease._release(*args, **kwargs)`

Free the lease of the given subnet index

**Parameters** `index` (*int*) – Third element of a dotted ip representation of the subnet, for example, for 1.2.3.4 it would be 3

**Returns** None

`lago.subnet_lease._take_lease` (*path*, *uuid\_path*)

Persist to the given leases path the prefix uuid that's in the uuid path passed

**Parameters**

- `path` (*str*) – Path to the leases file
- `uuid_path` (*str*) – Path to the prefix uuid

**Returns** None

`lago.subnet_lease._validate_lease_dir_present` (*func*)

Decorator that will ensure that the lease dir exists, creating it if necessary

`lago.subnet_lease.acquire` (*uuid\_path*)

Lease a free network for the given uuid path

**Parameters** `uuid_path` (*str*) – Path to the uuid file of a `lago.Prefix`

**Returns** the dotted ip of the gateway for the leased net

**Return type** `str`

`lago.subnet_lease.is_leasable_subnet` (*subnet*)

Checks if a given subnet is inside the defined provisionable range

**Parameters** `subnet` (*str*) – Subnet or ip in dotted decimal format

**Returns** True if subnet is inside the range, False otherwise

**Return type** `bool`

`lago.subnet_lease.release` (*subnet*)

Free the lease of the given subnet

**Parameters** `subnet` (*str*) – dotted ip or network to free the lease of

**Returns** None

### 4.1.15 lago.sysprep module

`lago.sysprep._config_net_interface` (*iface*, *\*\*kwargs*)

`lago.sysprep._upload_file` (*local\_path*, *remote\_path*)

`lago.sysprep._write_file` (*path*, *content*)

`lago.sysprep.add_ssh_key` (*key*, *with\_restorecon\_fix=False*)

`lago.sysprep.config_net_interface_dhcp` (*iface*, *hwaddr*)

`lago.sysprep.edit` (*filename*, *expression*)

`lago.sysprep.set_hostname` (*hostname*)

`lago.sysprep.set_iscsi_initiator_name` (*name*)

`lago.sysprep.set_root_password` (*password*)

`lago.sysprep.set_selinux_mode` (*mode*)

`lago.sysprep.sysprep` (*disk*, *mods*, *backend='direct'*)

### 4.1.16 lago.templates module

This module contains any disk template related classes and functions, including the repository store manager classes and template providers, some useful definitions:

- **Template repositories:** Repository where to fetch templates from, as an http server
- **Template store:** Local store to cache templates
- **Template:** Unitialized disk image to use as base for other disk images
- **Template version:** Specific version of a template, to allow getting updates without having to change the template name everywhere

```
class lago.templates.FileSystemTemplateProvider(root)
    Handles file type templates, that is, getting a disk template from the host's filesystem

    _prefixed (*path)
        Join all the given paths prefixed with this provider's base root path

        Parameters *path (str) – sections of the path to join, passed as positional arguments

        Returns Joined paths prepended with the provider root path

        Return type str

    download_image (handle, dest)
        Copies over the handl to the destination

        Parameters

        • handle (str) – path to copy over

        • dest (str) – path to copy to

        Returns None

    get_hash (handle)
        Returns the associated hash for the given handle, the hash file must exist (handle + ' .hash').

        Parameters handle (str) – Path to the template to get the hash from

        Returns Hash for the given handle

        Return type str

    get_metadata (handle)
        Returns the associated metadata info for the given handle, the metadata file must exist (handle + ' .metadata').

        Parameters handle (str) – Path to the template to get the metadata from

        Returns Metadata for the given handle

        Return type dict

class lago.templates.HttpTemplateProvider(baseurl)
    This provider allows the usage of http urls for templates

    download_image (handle, dest)
        Downloads the image from the http server

        Parameters

        • handle (str) – url from the self.baseurl to the remote template

        • dest (str) – Path to store the downloaded url to, must be a file path
```

**Returns** None

**static** `extract_if_needed` (*path*)

**get\_hash** (*handle*)

Get the associated hash for the given handle, the hash file must exist (`handle + '.hash'`).

**Parameters** **handle** (*str*) – Path to the template to get the hash from

**Returns** Hash for the given handle

**Return type** *str*

**get\_metadata** (*handle*)

Returns the associated metadata info for the given handle, the metadata file must exist (`handle + '.metadata'`). If the given handle has an `.xz` extension, it will get removed when calculating the handle metadata path

**Parameters** **handle** (*str*) – Path to the template to get the metadata from

**Returns** Metadata for the given handle

**Return type** *dict*

**open\_url** (*url*, *suffix=''*, *dest=None*)

Opens the given url, trying the compressed version first. The compressed version url is generated adding the `.xz` extension to the `url` and adding the given suffix **after** that `.xz` extension. If `dest` passed, it will download the data to that path if able

**Parameters**

- **url** (*str*) – relative url from the `self.baseurl` to retrieve
- **suffix** (*str*) – optional suffix to append to the url after adding the compressed extension to the path
- **dest** (*str or None*) – Path to save the data to

**Returns** response object to read from (lazy read), closed if no `dest` passed

**Return type** `urllib.addinfourl`

**Raises** `RuntimeError` – if the url gave http error when retrieving it

**class** `lago.templates.Template` (*name*, *versions*)

Disk image template class

**name**

*str*

Name of this template

**\_versions** (*dict* (*str*

`TemplateVersion`)): versions for this template

**get\_latest\_version** ()

Retrieves the latest version for this template, the latest being the one with the newest timestamp

**Returns** `TemplateVersion`

**get\_version** (*ver\_name=None*)

Get the given version for this template, or the latest

**Parameters** **ver\_name** (*str or None*) – Version to retrieve, None for the latest

**Returns** The version matching the given name or the latest one

**Return type** *TemplateVersion*

**class** `lago.templates.TemplateRepository` (*dom*)

A template repository is a single source for templates, that uses different providers to actually retrieve them. That means for example that the ‘ovirt’ template repository, could support the ‘http’ and a theoretical ‘gluster’ template providers.

**\_\_dom**

*dict*

Specification of the template

**\_\_providers**

*dict*

Providers instances for any source in the spec

**\_\_get\_provider** (*spec*)

Get the provider for the given template spec

**Parameters** **spec** (*dict*) – Template spec

**Returns** A provider instance for that spec

**Return type** `HttpTemplateProvider` or `FileSystemTemplateProvider`

**classmethod** **from\_url** (*path*)

Instantiate a *TemplateRepository* instance from the data in a file or url

**Parameters** **path** (*str*) – Path or url to the json file to load

**Returns** A new instance

**Return type** *TemplateRepository*

**get\_by\_name** (*name*)

Retrieve a template by it’s name

**Parameters** **name** (*str*) – Name of the template to retrieve

**Raises** `KeyError` – if no template is found

**name**

Getter for the template repo name

**Returns** the name of this template repo

**Return type** *str*

**class** `lago.templates.TemplateStore` (*path*)

Local cache to store templates

The store uses various files to keep track of the templates cached, access and versions. An example template store looks like:

```
$ tree /var/lib/lago/store/
/var/lib/lago/store/
-- in_office_repo:centos6_engine:v2.tmp
-- in_office_repo:centos7_engine:v5.tmp
-- in_office_repo:fedora22_host:v2.tmp
-- phx_repo:centos6_engine:v2
-- phx_repo:centos6_engine:v2.hash
-- phx_repo:centos6_engine:v2.metadata
-- phx_repo:centos6_engine:v2.users
-- phx_repo:centos7_engine:v4.tmp
```

```
-- phx_repo:centos7_host:v4.tmp
-- phx_repo:storage-nfs:v1.tmp
```

There you can see the files:

- **\*.tmp** Temporary file created while downloading the template from the repository (depends on the provider)
- **\${repo\_name}:\${template\_name}:\${template\_version}** This file is the actual disk image template
- **\*.hash** Cached hash for the template disk image
- **\*.metadata** Metadata for this template image in json format, usually this includes the *distro* and *root-password*

**\_\_contains\_\_** (*temp\_ver*)

Checks if a given version is in this store

**Parameters** **temp\_ver** (*TemplateVersion*) – Version to look for

**Returns** `True` if the version is in this store

**Return type** `bool`

**\_init\_users** (*temp\_ver*)

Initializes the user access registry

**Parameters** **temp\_ver** (*TemplateVersion*) – template version to update registry for

**Returns** `None`

**\_prefixed** (*\*path*)

Join the given paths and prepend this stores path

**Parameters** **\*path** (*str*) – list of paths to join, as positional arguments

**Returns** all the paths joined and prepended with the store path

**Return type** `str`

**download** (*temp\_ver, store\_metadata=True*)

Retrieve the given template version

**Parameters**

- **temp\_ver** (*TemplateVersion*) – template version to retrieve
- **store\_metadata** (*bool*) – If set to `False`, will not refresh the local metadata with the retrieved one

**Returns** `None`

**get\_path** (*temp\_ver*)

Get the path of the given version in this store

**Parameters** **TemplateVersion** (*temp\_ver*) – version to look for

**Returns** The path to the template version inside the store

**Return type** `str`

**Raises** `RuntimeError` – if the template is not in the store

**get\_stored\_hash** (*temp\_ver*)

Retrieves the hash for the given template version from the store

**Parameters** **temp\_ver** (*TemplateVersion*) – template version to retrieve the hash for

**Returns** hash of the given template version

**Return type** `str`

**get\_stored\_metadata** (*temp\_ver*)

Retrieves the metadata for the given template version from the store

**Parameters** **temp\_ver** (`TemplateVersion`) – template version to retrieve the metadata for

**Returns** the metadata of the given template version

**Return type** `dict`

**mark\_used** (*temp\_ver*, *key\_path*)

Adds or updates the user entry in the user access log for the given template version

**Parameters**

- **temp\_ver** (`TemplateVersion`) – template version to add the entry for
- **key\_path** (`str`) – Path to the prefix uuid file to set the mark for

**class** `lago.templates.TemplateVersion` (*name*, *source*, *handle*, *timestamp*)

Each template can have multiple versions, each of those is actually a different disk template file representation, under the same base name.

**download** (*destination*)

Retrieves this template to the destination file

**Parameters** **destination** (`str`) – file path to write this template to

**Returns** `None`

**get\_hash** ()

Returns the associated hash for this template version

**Returns** Hash for this version

**Return type** `str`

**get\_metadata** ()

Returns the associated metadata info for this template version

**Returns** Metadata for this version

**Return type** `dict`

**timestamp** ()

Getter for the timestamp

`lago.templates._locked` (*func*)

Decorator that ensures that the decorated function has the lock of the repo while running, meant to decorate only bound functions for classes that have *lock\_path* method.

`lago.templates.find_repo_by_name` (*name*, *repo\_dir=None*)

Searches the given repo name inside the repo\_dir (will use the config value 'template\_repos' if no repo\_dir passed), will rise an exception if not found

**Parameters**

- **name** (`str`) – Name of the repo to search
- **repo\_dir** (`str`) – Directory where to search the repo

**Returns** path to the repo

**Return type** `str`

**Raises** `RuntimeError` – if not found

#### 4.1.17 lago.utils module

**class** `lago.utils.CommandStatus`  
Bases: `lago.utils.CommandStatus`

**class** `lago.utils.EggTimer` (*timeout*)

**elapsed** ()

**class** `lago.utils.ExceptionTimer` (*timeout*)  
Bases: `object`

**start** ()

**stop** ()

**class** `lago.utils.LockFile` (*path*, *timeout=None*, *\*\*kwargs*)  
Bases: `object`

Context manager that creates a lock around a directory, with optional timeout in the acquire operation

##### Parameters

- **path** (*str*) – path to the dir to lock
- **timeout** (*int*) – timeout in seconds to wait while acquiring the lock
- **\*\*kwargs** (*dict*) – Any other param to pass to `lockfile.LockFile`

**\_\_enter\_\_** ()  
Start the lock with timeout if needed in the acquire operation

**Raises** `TimerException` – if the timeout is reached before acquiring the lock

**class** `lago.utils.RollbackContext` (*\*args*)  
Bases: `object`

A context manager for recording and playing rollback. The first exception will be remembered and re-raised after rollback

Sample usage: > with RollbackContext() as rollback: > step1() > rollback.prependDefer(lambda: undo step1) > def undoStep2(arg): pass > step2() > rollback.prependDefer(undoStep2, arg)

More examples see tests/utilsTests.py @ vdsms code

**\_\_exit\_\_** (*exc\_type*, *exc\_value*, *traceback*)  
If this function doesn't return True (or raises a different exception), python re-raises the original exception once this function is finished.

**clear** ()

**defer** (*func*, *\*args*, *\*\*kwargs*)

**prependDefer** (*func*, *\*args*, *\*\*kwargs*)

**exception** `lago.utils.TimerException`  
Bases: `exceptions.Exception`

Exception to throw when a timeout is reached

**class** `lago.utils.VectorThread` (*targets*)



**join\_all** (*raise\_exceptions=True*)

**start\_all** ()

lago.utils.\_**CommandStatus**

alias of *CommandStatus*

lago.utils.\_**add\_subparser\_to\_cp** (*cp, section, actions, incl\_unset*)

lago.utils.\_**ret\_via\_queue** (*func, queue*)

lago.utils.\_**run\_command** (*command, input\_data=None, stdin=None, out\_pipe=-1, err\_pipe=-1, env=None, uuid=None, \*\*kwargs*)

Runs a command

#### Parameters

- **command** (*list of str*) – args of the command to execute, including the command itself as command[0] as ['ls', '-l']
- **input\_data** (*str*) – If passed, will feed that data to the subprocess through stdin
- **out\_pipe** (*int or file*) – File descriptor as passed to :ref:subprocess.Popen to use as stdout
- **stdin** (*int or file*) – File descriptor as passed to :ref:subprocess.Popen to use as stdin
- **err\_pipe** (*int or file*) – File descriptor as passed to :ref:subprocess.Popen to use as stderr
- **of str** (*env(dict) – str*): If set, will use the given dict as env for the subprocess
- **uuid** (*uuid*) – If set the command will be logged with the given uuid converted to string, otherwise, a uuid v4 will be generated.
- **\*\*kwargs** – Any other keyword args passed will be passed to the :ref:subprocess.Popen call

**Returns** result of the interactive execution

**Return type** *lago.utils.CommandStatus*

lago.utils.**add\_timestamp\_suffix** (*base\_string*)

lago.utils.**argparse\_to\_ini** (*parser, root\_section='lago', incl\_unset=False*)

lago.utils.**deepcopy** (*original\_obj*)

Creates a deep copy of an object with no crossed referenced lists or dicts, useful when loading from yaml as anchors generate those cross-referenced dicts and lists

**Parameters** **original\_obj** (*object*) – Object to deep copy

**Returns** deep copy of the object

**Return type** *object*

lago.utils.**func\_vector** (*target, args\_sequence*)

lago.utils.**in\_prefix** (*prefix\_class, workdir\_class*)

lago.utils.**invoke\_in\_parallel** (*func, \*args\_sequences*)

lago.utils.**ipv4\_to\_mac** (*ip*)

lago.utils.**json\_dump** (*obj, f*)

`lago.utils.load_virt_stream(virt_fd)`

Loads the given conf stream into a dict, trying different formats if needed

**Parameters** `virt_fd` (*str*) – file like object with the virt config to load

**Returns** Loaded virt config

**Return type** `dict`

`lago.utils.read_nonblocking(file_descriptor)`

`lago.utils.rotate_dir(base_dir)`

`lago.utils.run_command(command, input_data=None, out_pipe=-1, err_pipe=-1, env=None, **kwargs)`

Runs a command non-interactively

**Parameters**

- **command** (*list of str*) – args of the command to execute, including the command itself as `command[0]` as `['ls', '-l']`
- **input\_data** (*str*) – If passed, will feed that data to the subprocess through stdin
- **out\_pipe** (*int or file*) – File descriptor as passed to `:ref:subprocess.Popen` to use as stdout
- **err\_pipe** (*int or file*) – File descriptor as passed to `:ref:subprocess.Popen` to use as stderr
- **of str** (*env(dict)* – str): If set, will use the given dict as env for the subprocess
- **\*\*kwargs** – Any other keyword args passed will be passed to the `:ref:subprocess.Popen` call

**Returns** result of the interactive execution

**Return type** `lago.utils.CommandStatus`

`lago.utils.run_interactive_command(command, env=None, **kwargs)`

Runs a command interactively, reusing the current stdin, stdout and stderr

**Parameters**

- **command** (*list of str*) – args of the command to execute, including the command itself as `command[0]` as `['ls', '-l']`
- **of str** (*env(dict)* – str): If set, will use the given dict as env for the subprocess
- **\*\*kwargs** – Any other keyword args passed will be passed to the `:ref:subprocess.Popen` call

**Returns** result of the interactive execution

**Return type** `lago.utils.CommandStatus`

`lago.utils.service_is_enabled(name)`

`lago.utils.with_logging(func)`

## 4.1.18 lago.virt module

`class lago.virt.BridgeNetwork(env, spec)`

Bases: `lago.virt.Network`

`_libvirt_xml()`

```

    start()
    stop()
class lago.virt.NATNetwork(env, spec)
    Bases: lago.virt.Network
    _libvirt_xml()
class lago.virt.Network(env, spec)
    Bases: object
    _libvirt_name()
    _libvirt_xml()
    add_mapping(name, ip, save=True)
    add_mappings(mappings)
    alive()
    gw()
    is_management()
    mapping()
    name()
    resolve(name)
    save()
    start(attempts=5, timeout=2)
        Start the network, will check if the network is active attempts times, waiting timeout between each attempt.

```

#### Parameters

- **attempts** (*int*) – number of attempts to check the network is active
- **timeout** (*int*) – timeout for each attempt

#### Returns

#### Raises

- `RuntimeError` – if network creation failed, or failed to verify it is
- `active.`

```

    stop()
class lago.virt.VirtEnv(prefix, vm_specs, net_specs)
    Bases: object
    Env properties: * prefix * vms * net
        • libvirt_con
    _CPU_FAMILIES = {'Westmere': 'Intel Westmere Family', 'Opteron_G2': 'AMD Opteron G2', 'Penryn': 'Intel Penryn'}
    _compatible_cpu_and_family = None
    _create_net(net_spec)
    _create_vm(vm_spec)
    bootstrap()

```

**create\_snapshots** (\*args, \*\*kwargs)

**classmethod from\_prefix** (prefix)

**get\_compatible\_cpu\_and\_family** ()

**get\_cpu\_model** ()

**get\_net** (name=None)

**get\_nets** ()

**get\_snapshots** (domains=None)

Get the list of snapshots for each domain

**Parameters**

- **domanins** (list of str) – list of the domains to get the snapshots
- **all will be returned if none or empty list passed** (for,) –

**Returns** dict of str -> list – with the domain names and the list of snapshots for each

**Return type** str

**get\_vm** (name)

**get\_vms** ()

**prefixed\_name** (unprefixed\_name, max\_length=0)

Returns a uuid pefixed identifier

**Parameters**

- **unprefixed\_name** (str) – Name to add a prefix to
- **max\_length** (int) – maximum length of the resultant prefixed name, will adapt the given name and the length of the uuid ot fit it

**Returns** prefixed identifier for the given unprefixed name

**Return type** str

**revert\_snapshots** (\*args, \*\*kwargs)

**save** (\*args, \*\*kwargs)

**start** (vm\_names=None)

**stop** (vm\_names=None)

**virt\_path** (\*args)

lago.virt.\_gen\_ssh\_command\_id()

lago.virt.\_guestfs\_copy\_path(g, guest\_path, host\_path)

lago.virt.\_path\_to\_xml(basename)

### 4.1.19 lago.vm module

**class** lago.vm.DefaultVM(env, spec)

Bases: *lago.plugins.vm.VMPlugin*

**\_abc\_cache** = <\_weakrefset.WeakSet object>

**\_abc\_negative\_cache** = <\_weakrefset.WeakSet object>

```

    _abc_negative_cache_version = 25
    _abc_registry = <_weakrefset.WeakSet object>
class lago.vm.LocalLibvirtVMPProvider (vm)
    Bases: lago.plugins.vm.VMPProviderPlugin

    _create_dead_snapshot (name)
    _create_live_snapshot (name)
    _extract_paths_gfs (paths, ignore_nopath)
    _libvirt_name ()
    _libvirt_xml ()
    _reclaim_disk (path)
    _reclaim_disks ()
    bootstrap ()
    cpu_model
        Return the VM CPU model for domain XML generation

        Returns cpu model

        Return type str
    create_snapshot (name)
    defined ()
    extract_paths (paths, ignore_nopath)
        Extract the given paths from the domain

        Attempt to extract all files defined in paths with the method defined in extract_paths(), if it fails,
        will try extracting the files with libguestfs.

        Parameters
            • paths (list of str) – paths to extract
            • ignore_nopath (boolean) – if True will ignore none existing paths.

        Returns None

        Raises
            • ExtractPathNoPathError – if a none existing path was found on the VM, and ignore_nopath is True.
            • ExtractPathError – on all other failures.
    interactive_console (*args, **kwargs)
        Opens an interactive console

        Returns result of the virsh command execution

        Return type lago.utils.CommandStatus
    revert_snapshot (name)
    start ()
    state ()
        Return a small description of the current status of the domain

```

**Returns** small description of the domain status, ‘down’ if it’s not defined at all.

**Return type** `str`

```
stop ()

class lago.vm.SSHVMPProvider (vm)
    Bases: lago.plugins.vm.VMPProviderPlugin

    bootstrap (*args, **kwargs)

    create_snapshot (name, *args, **kwargs)

    defined (*args, **kwargs)

    revert_snapshot (name, *args, **kwargs)

    start (*args, **kwargs)

    state (*args, **kwargs)

    stop (*args, **kwargs)

lago.vm._check_defined (func)

lago.vm._guestfs_copy_path (guestfs_conn, guest_path, host_path)

lago.vm._path_to_xml (basename)
```

## 4.1.20 lago.workdir module

A workdir is the base directory where lago will store all the files it needs and that are unique (not shared between workdirs).

It’s basic structure is a directory with one soft link and multiple directories, one per prefix. Where the link points to the default prefix to use.

```
exception lago.workdir.MalformedWorkdir
    Bases: lago.workdir.WorkdirError

exception lago.workdir.PrefixAlreadyExists
    Bases: lago.workdir.WorkdirError

exception lago.workdir.PrefixNotFound
    Bases: lago.workdir.WorkdirError

class lago.workdir.Workdir (path, prefix_class=<class ‘lago.prefix.Prefix’>)
    Bases: object
```

This class represents a base workdir, where you can store multiple prefixes

**Properties:** `path(str)`: Path to the workdir prefixes(dict of str->self.prefix\_class): dict with the prefixes in the workdir, by name `current(str)`: Name of the current prefix `prefix_class(type)`: Class to use when creating prefixes

`_set_current (new_current)`  
Change the current default prefix, for internal usage

**Parameters** `new_current (str)` – Name of the new current prefix, it must already exist

**Returns** None

**Raises** `PrefixNotFound` – if the given prefix name does not exist in the workdir

`_update_current ()`  
Makes sure that a current is set

**add\_prefix** (*workdir*, \*args, \*\*kwargs)

Adds a new prefix to the workdir.

**Parameters**

- **name** (*str*) – Name of the new prefix to add
- **\*args** – args to pass along to the prefix constructor
- **\*kwargs** – kwargs to pass along to the prefix constructor

**Returns** The newly created prefix

**Raises** *PrefixAlreadyExists* – if the prefix name already exists in the workdir

**destroy** (*workdir*, \*args, \*\*kwargs)

Destroy all the given prefixes and remove any left files if no more prefixes are left

**Parameters**

- **prefix\_names** (*list of str*) – list of prefix names to destroy, if None
- **passed** (*default*) –

**get\_prefix** (*workdir*, \*args, \*\*kwargs)

Retrieve a prefix, resolving the current one if needed

**Parameters** **name** (*str*) – name of the prefix to retrieve, or current to get the current one

**Returns** instance of the prefix with the given name

**Return type** self.prefix\_class

**initialize** (*prefix\_name='default'*, \*args, \*\*kwargs)

Initializes a workdir by adding a new prefix to the workdir.

**Parameters**

- **prefix\_name** (*str*) – Name of the new prefix to add
- **\*args** – args to pass along to the prefix constructor
- **\*kwargs** – kwargs to pass along to the prefix constructor

**Returns** The newly created prefix

**Raises** *PrefixAlreadyExists* – if the prefix name already exists in the workdir

**classmethod is\_workdir** (*path*)

Check if the given path is a workdir

**Parameters** **path** (*str*) – Path to check

**Returns** True if the given path is a workdir

**Return type** bool

**join** (\*args)

Gets a joined path prefixed with the workdir path

**Parameters** **\*args** (*str*) – path sections to join

**Returns** Joined path prefixed with the workdir path

**Return type** str

**load** ()

Loads the prefixes that are available in the workdir

**Returns** None

**Raises** *MalformedWorkdir* – if the wordir is malformed

**classmethod** `resolve_workdir_path` (*start\_path*='.')

Look for an existing workdir in the given path, in a path/.lago dir, or in a .lago dir under any of it's parent directories

**Parameters** `start_path` (*str*) – path to start the search from, if None passed, it will use the current dir

**Returns** path to the found prefix

**Return type** *str*

**Raises** *RuntimeError* – if no prefix was found

**set\_current** (*workdir*, \**args*, \*\**kwargs*)

Change the current default prefix

**Parameters** `new_current` (*str*) – Name of the new current prefix, it must already exist

**Returns** None

**Raises** *PrefixNotFound* – if the given prefix name does not exist in the workdir

**exception** `lago.workdir.WorkdirError`

Bases: `exceptions.RuntimeError`

Base exception for workdir errors, catch this one to catch any workdir error

`lago.workdir.workdir_loaded` (*func*)

Decorator to make sure that the workdir is loaded when calling the decorated function

## 4.2 lago\_template\_repo package

**class** `lago_template_repo.TemplateRepoCLI`

Bases: `lago.plugins.cli.CLIPugin`

`_abc_cache` = <\_weakrefset.WeakSet object>

`_abc_negative_cache` = <\_weakrefset.WeakSet object>

`_abc_negative_cache_version` = 25

`_abc_registry` = <\_weakrefset.WeakSet object>

`do_run` (*args*)

`init_args` = {'help': 'Utility for system testing template management'}

`populate_parser` (*parser*)

**class** `lago_template_repo.Verbs`

`ADD` = 'add'

`UPDATE` = 'update'

`lago_template_repo.do_add` (*args*)

`lago_template_repo.do_update` (*args*)



## 4.3 ovirtlago package

```
class ovirtlago.OvirtPrefix(*args, **kwargs)
    Bases: lago.prefix.Prefix

    VIRT_ENV_CLASS
        alias of OvirtVirtEnv

    _create_rpm_repository(dists, repos_path, repo_names, repoman_config, cus-
                           tom_sources=None, projects_list=None)

    _create_virt_env()

    deploy(*args, **kwargs)

    prepare_repo(*args, **kwargs)

    run_test(*args, **kwargs)

    serve(*args, **kwargs)

class ovirtlago.OvirtWorkdir(*args, **kwargs)
    Bases: lago.workdir.Workdir
```

### 4.3.1 Submodules

### 4.3.2 ovirtlago.cmd module

```
class ovirtlago.cmd.OvirtCLI
    Bases: lago.plugins.cli.CLIPlugin

    _abc_cache = <_weakrefset.WeakSet object>

    _abc_negative_cache = <_weakrefset.WeakSet object>

    _abc_negative_cache_version = 25

    _abc_registry = <_weakrefset.WeakSet object>

    do_run(args)

    init_args = {'help': 'oVirt related actions'}

    populate_parser(parser)

ovirtlago.cmd._populate_parser(cli_plugins, parser)
```

### 4.3.3 ovirtlago.constants module

### 4.3.4 ovirtlago.paths module

```
class ovirtlago.paths.OvirtPaths(prefix)
    Bases: lago.paths.Paths

    build_dir(*path)

    internal_repo(*path)

    test_logs(*args)
```

### 4.3.5 ovirtlago.reposetup module

**exception** `ovirtlago.reposetup.RepositoryError`

Bases: `exceptions.Exception`

**exception** `ovirtlago.reposetup.RepositoryMergeError`

Bases: `ovirtlago.reposetup.RepositoryError`

`ovirtlago.reposetup._fix_reposync_issues` (*reposync\_out*, *repo\_path*)

**Fix for the issue described at::** [https://bugzilla.redhat.com/show\\_bug.cgi?id=1399235](https://bugzilla.redhat.com/show_bug.cgi?id=1399235)  
[https://bugzilla.redhat.com/show\\_bug.cgi?id=1332441](https://bugzilla.redhat.com/show_bug.cgi?id=1332441)

`ovirtlago.reposetup.merge` (*output\_dir*, *sources*, *repoman\_config*=None)

Run repoman on *sources*, creating a new RPM repository in *output\_dir*

#### Parameters

- **output\_dir** (*str*) – Path to create new repository
- **sources** (*list of str*) – repoman sources
- **repoman\_config** (*str*) – repoman configuration file, if not passed it will use default repoman configurations, equivalent to:

```
[main]
on_empty_source=warn
[store.RPMStore]
on_wrong_distro=copy_to_all
with_srcrpms=false
with_sources=false
```

#### Raises

- `RepositoryMergeError` – If repoman command failed.
- `IOError` – If *repoman\_config* is passed but does not exists.

#### Returns

None

`ovirtlago.reposetup.sync_rpm_repository` (*repo\_path*, *yum\_config*, *repos*)

`ovirtlago.reposetup.with_repo_server` (*func*)

### 4.3.6 ovirtlago.testlib module

**class** `ovirtlago.testlib.LogCollectorPlugin` (*prefix*)

Bases: `nose.plugins.base.Plugin`

**\_addFault** (*test*, *err*)

**addError** (*test*, *err*)

**addFailure** (*test*, *err*)

**configure** (*options*, *conf*)

**name** = 'log-collector-plugin'

**options** (*parser*, *env*=None)

```

class ovirtlago.testlib.TaskLogNosePlugin(*args, **kwargs)
    Bases: nose.plugins.base.Plugin

    addError(test, err)

    configure(options, conf)

    name = 'tasklog-plugin'

    options(parser, env)

    score = 10000

    startTest(test)

    stopTest(test)

ovirtlago.testlib._instance_of_any(obj, cls_list)

ovirtlago.testlib._vms_capable(vms, caps)

ovirtlago.testlib.assert_equals_within(func, value, timeout, allowed_exceptions=None)

ovirtlago.testlib.assert_equals_within_long(func, value, allowed_exceptions=None)

ovirtlago.testlib.assert_equals_within_short(func, value, allowed_exceptions=None)

ovirtlago.testlib.assert_true_within(func, timeout, allowed_exceptions=None)

ovirtlago.testlib.assert_true_within_long(func, allowed_exceptions=None)

ovirtlago.testlib.assert_true_within_short(func, allowed_exceptions=None)

ovirtlago.testlib.engine_capability(caps)

ovirtlago.testlib.get_prefixed_name(entity_name)

ovirtlago.testlib.get_test_prefix()

ovirtlago.testlib.host_capability(caps)

ovirtlago.testlib.test_sequence_gen(test_list)

ovirtlago.testlib.with_ovirt_api(func)

ovirtlago.testlib.with_ovirt_api4(func)

ovirtlago.testlib.with_ovirt_prefix(func)

```

### 4.3.7 ovirtlago.utils module

ovirtlago.utils.\_create\_http\_server(listen\_ip, listen\_port, root\_dir)  
 Starts an http server with an improved request handler

#### Parameters

- **listen\_ip** (*str*) – Ip to listen on
- **port** (*int*) – Port to register on
- **root\_dir** (*str*) – path to the directory to serve

**Returns** instance of the http server, already running on a thread

**Return type** `BaseHTTPServer`

ovirtlago.utils.generate\_request\_handler(root\_dir)  
 Factory for `_BetterHTTPRequestHandler` classes

**Parameters** `root_dir` (*path*) – Path to the dir to serve

**Returns** A ready to be used improved http request handler

**Return type** `_BetterHTTPRequestHandler`

`ovirtlago.utils.repo_server_context(*args, **kws)`

Context manager that starts an http server that serves the given prefix's yum repository. Will listen on `constants.REPO_SERVER_PORT` and on the first network defined in the previx virt config

**Parameters** `prefix` (`ovirtlago.OvirtPrefix`) – prefix to start the server for

**Returns** None

### 4.3.8 ovirtlago.virt module

`class ovirtlago.virt.EngineVM(*args, **kwargs)`

Bases: `lago.vm.DefaultVM`

`_abc_cache = <_weakrefset.WeakSet object>`

`_abc_negative_cache = <_weakrefset.WeakSet object>`

`_abc_negative_cache_version = 25`

`_abc_registry = <_weakrefset.WeakSet object>`

`_artifact_paths()`

`_create_api(api_ver)`

`_get_api(api_ver)`

`add_iso(path)`

`engine_setup(config=None)`

`get_api(api_ver=3)`

`get_api_v3()`

`get_api_v4()`

`stop()`

`class ovirtlago.virt.HEHostVM(env, spec)`

Bases: `ovirtlago.virt.HostVM`

`_abc_cache = <_weakrefset.WeakSet object>`

`_abc_negative_cache = <_weakrefset.WeakSet object>`

`_abc_negative_cache_version = 25`

`_abc_registry = <_weakrefset.WeakSet object>`

`_artifact_paths()`

`class ovirtlago.virt.HostVM(env, spec)`

Bases: `lago.vm.DefaultVM`

`_abc_cache = <_weakrefset.WeakSet object>`

`_abc_negative_cache = <_weakrefset.WeakSet object>`

`_abc_negative_cache_version = 25`

`_abc_registry = <_weakrefset.WeakSet object>`

```
    _artifact_paths ()

class ovirtlago.virt.NodeVM(env, spec)
    Bases: lago.vm.DefaultVM

    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 25
    _abc_registry = <_weakrefset.WeakSet object>
    _artifact_paths ()
    wait_for_ssh ()

class ovirtlago.virt.OvirtVirtEnv(prefix, vm_specs, net_spec)
    Bases: lago.virt.VirtEnv

    _create_vm (vm_spec)
    engine_vm ()
    get_ovirt_cpu_family ()
    host_vms ()
```



---

## Releases

---

### 5.1 Release process

#### 5.1.1 Versioning

For Iago we use a similar approach to semantic versioning, that is:

```
X.Y.Z
```

For example:

```
0.1.0
1.2.123
2.0.0
2.0.1
```

Where:

- Z changes for each patch (number of patches since X.Y tag)
- Y changes from time to time, with milestones (arbitrary bump), only for backwards compatible changes
- X changes if it's a non-backwards compatible change or arbitrarily (we don't like Y getting too high, or big milestone reached, ...)

The source tree has tags with the X.Y versions, that's where the packaging process gets them from.

On each X or Y change a new tag is created.

For now we have only one branch (master) and we will try to keep it that way as long as possible, if at some point we have to support old versions, then we will create a branch for each X version in the form:

```
vX
```

For example:

```
v0
v1
```

There's a helper script to resolve the current version, based on the last tag and the compatibility breaking commits since then, to get the version for the current repo run:

```
$ scripts/version_manager.py . version
```

### 5.1.2 RPM Versioning

The rpm versions differ from the generic version in that they have the `-1` suffix, where the `-1` is the release for that rpm (usually will never change, only when repackaging without any code change, something that is not so easy for us but if there's any external packagers is helpful for them)

### 5.1.3 Repository layout

Tree schema of the repository:

```
lago
-- stable <-- subdirs for each major version to avoid accidental
| |             non-backwards compatible upgrade
| |
| | -- 0.0 <-- Contains any 0.* release for lago
| |   -- ChangeLog_0.0.txt
| |   -- rpm
| |     -- el6
| |     -- el7
| |     -- fc22
| |     -- fc23
| |   -- sources
| -- 1.0
|   -- ChangeLog_1.0.txt
|   -- rpm
|     -- el6
|     -- el7
|     -- fc22
|     -- fc23
|   -- sources
| -- 2.0
|   -- ChangeLog_2.0.txt
|   -- rpm
|     -- el6
|     -- el7
|     -- fc22
|     -- fc23
|   -- sources
-- unstable <-- Multiple subdirs are needed only if branching
  -- 0.0 <-- Contains 0.* builds that might or might not have
  | |             been released
  | -- latest <--- keeps the latest build from merged, static
  | |             url
  | -- snapshot-lago_0.0_pipeline_1
  | -- snapshot-lago_0.0_pipeline_2
  | |             ^ contains the rpms created on the pipeline build
  | |             number 2 for the 0.0 version, this is needed to
  | |             ease the automated testing of the rpms
  | |
  | -- ... <-- this is cleaned up from time to time to avoid
  | |             using too much space
  -- 1.0
  | -- latest
  | -- snapshot-lago_1.0_pipeline_1
  | -- snapshot-lago_pipeline_2
  | -- ...
  -- 2.0
```



```
-- latest
-- snapshot-lago_2.0_pipeline_1
-- snapshot-lago_2.0_pipeline_2
-- ...
```

### 5.1.4 Promotion of artifacts to stable, aka. releasing

The goal is to have an automated set of tests, that check in depth lago, and run them in a timely fashion, and if passed, deploy to stable. As right now that's not yet possible, so for now the tests and deploy is done manually.

The promotion of the artifacts is done in these cases:

- New major version bump ( $X+1.0$ , for example  $1.0 \rightarrow 2.0$ )
- New minor version bump ( $X.Y+1$ , for example  $1.1 \rightarrow 1.2$ )
- If it passed certain time since the last  $X$  or  $Y$  version bumps ( $X.Y.Z+n$ , for example  $1.0.1 \rightarrow 1.0.2$ )
- If there are blocking/important bugfixes ( $X.Y.Z+n$ )
- If there are important new features ( $X.Y+1$  or  $X.Y.Z+n$ )

### 5.1.5 How to mark a major version

Whenever there's a commit that breaks the backwards compatibility, you should add to it the pseudo-header:

```
Sem-Ver: api-breaking
```

And that will force a major version bump for any package built from it, that is done so in the moment when you submit the commit in Gerrit, the packages that are build from it have the correct version.

After that, make sure that you tag that commit too, so it will be easy to look for it in the future.

### 5.1.6 The release procedure on the maintainer side

1. Select the snapshot repo you want to release
2. **Test the rpms, for now we only have the tests from projects that use it:**
  - Run all the [ovirt tests](#) on it, make sure it does not break anything, if there are issues -> [open bug](#)
  - **Run [vdsm functional tests](#), make sure it does not break anything, if** there are issues -> [open bug](#)
3. **On non-major version bump  $X.Y+1$  or  $X.Y.Z+n$** 
  - [Create a changelog](#) since the base of the tag and keep it aside
4. **On Major version bump  $X+1.0$** 
  - [Create a changelog](#) since the previous  $.0$  tag ( $X.0$ ) and keep it aside
5. Deploy the rpms from snapshot to dest repo and copy the `ChangeLog` from the tarball to `ChangeLog_X.0.txt` in the base of the `stable/X.0/` dir
6. Send email to [lago-devel](#) with the announcement and the changelog since the previous tag that you kept aside, feel free to change the body to your liking:

Subject: [day-month-year] New lago release - X.Y.Z

Hi everyone! There's a new lago release with version X.Y.Z ready for you to upgrade!

Here are the changes:

<CHANGELOG HERE>

Enjoy!

---

## Changelog

---

Here you can find the full changelog for this version



---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



## I

- [lago](#), 17
- [lago.brctl](#), 27
- [lago.cmd](#), 28
- [lago.config](#), 28
- [lago.constants](#), 29
- [lago.dirlock](#), 29
- [lago.libvirt\\_utils](#), 30
- [lago.log\\_utils](#), 30
- [lago.paths](#), 36
- [lago.plugins](#), 17
- [lago.plugins.cli](#), 17
- [lago.plugins.output](#), 21
- [lago.plugins.service](#), 23
- [lago.plugins.vm](#), 24
- [lago.prefix](#), 37
- [lago.service](#), 43
- [lago.ssh](#), 44
- [lago.subnet\\_lease](#), 45
- [lago.sysprep](#), 46
- [lago.templates](#), 47
- [lago.utils](#), 52
- [lago.virt](#), 54
- [lago.vm](#), 56
- [lago.workdir](#), 58
- [lago\\_template\\_repo](#), 60

## O

- [ovirtlago](#), 61
- [ovirtlago.cmd](#), 61
- [ovirtlago.constants](#), 61
- [ovirtlago.paths](#), 61
- [ovirtlago.reposetup](#), 62
- [ovirtlago.testlib](#), 62
- [ovirtlago.utils](#), 63
- [ovirtlago.virt](#), 64





## Symbols

- `_CPU_FAMILIES` (lago.virt.VirtEnv attribute), 55
- `_CommandStatus` (in module lago.utils), 53
- `__call__()` (lago.plugins.cli.CLIPluginFuncWrapper method), 19
- `__contains__()` (lago.templates.TemplateStore method), 50
- `__enter__()` (lago.utils.LockFile method), 52
- `__exit__()` (lago.utils.RollbackContext method), 52
- `__getitem__()` (lago.config.ConfigLoad method), 28
- `_abc_cache` (lago.plugins.cli.CLIPlugin attribute), 19
- `_abc_cache` (lago.plugins.cli.CLIPluginFuncWrapper attribute), 19
- `_abc_cache` (lago.plugins.output.DefaultOutFormatPlugin attribute), 21
- `_abc_cache` (lago.plugins.output.FlatOutFormatPlugin attribute), 22
- `_abc_cache` (lago.plugins.output.JSONOutFormatPlugin attribute), 22
- `_abc_cache` (lago.plugins.output.OutFormatPlugin attribute), 22
- `_abc_cache` (lago.plugins.output.YAMLOutFormatPlugin attribute), 23
- `_abc_cache` (lago.plugins.service.ServicePlugin attribute), 23
- `_abc_cache` (lago.plugins.vm.VMPlugin attribute), 24
- `_abc_cache` (lago.service.SysVInitService attribute), 43
- `_abc_cache` (lago.service.SystemdContainerService attribute), 44
- `_abc_cache` (lago.service.SystemdService attribute), 44
- `_abc_cache` (lago.vm.DefaultVM attribute), 56
- `_abc_cache` (lago\_template\_repo.TemplateRepoCLI attribute), 60
- `_abc_cache` (ovirtlago.cmd.OvirtCLI attribute), 61
- `_abc_cache` (ovirtlago.virt.EngineVM attribute), 64
- `_abc_cache` (ovirtlago.virt.HEHostVM attribute), 64
- `_abc_cache` (ovirtlago.virt.HostVM attribute), 64
- `_abc_cache` (ovirtlago.virt.NodeVM attribute), 65
- `_abc_negative_cache` (lago.plugins.cli.CLIPlugin attribute), 19
- `_abc_negative_cache` (lago.plugins.cli.CLIPluginFuncWrapper attribute), 19
- `_abc_negative_cache` (lago.plugins.output.DefaultOutFormatPlugin attribute), 21
- `_abc_negative_cache` (lago.plugins.output.FlatOutFormatPlugin attribute), 22
- `_abc_negative_cache` (lago.plugins.output.JSONOutFormatPlugin attribute), 22
- `_abc_negative_cache` (lago.plugins.output.OutFormatPlugin attribute), 22
- `_abc_negative_cache` (lago.plugins.output.YAMLOutFormatPlugin attribute), 23
- `_abc_negative_cache` (lago.plugins.service.ServicePlugin attribute), 23
- `_abc_negative_cache` (lago.plugins.vm.VMPlugin attribute), 24
- `_abc_negative_cache` (lago.service.SysVInitService attribute), 43
- `_abc_negative_cache` (lago.service.SystemdContainerService attribute), 44
- `_abc_negative_cache` (lago.service.SystemdService attribute), 44
- `_abc_negative_cache` (lago.vm.DefaultVM attribute), 56
- `_abc_negative_cache` (lago\_template\_repo.TemplateRepoCLI attribute), 60
- `_abc_negative_cache` (ovirtlago.cmd.OvirtCLI attribute), 61
- `_abc_negative_cache` (ovirtlago.virt.EngineVM attribute), 64
- `_abc_negative_cache` (ovirtlago.virt.HEHostVM attribute), 64
- `_abc_negative_cache` (ovirtlago.virt.HostVM attribute), 64
- `_abc_negative_cache` (ovirtlago.virt.NodeVM attribute), 65
- `_abc_negative_cache_version` (lago.plugins.cli.CLIPlugin attribute), 19
- `_abc_negative_cache_version` (lago.plugins.cli.CLIPluginFuncWrapper attribute), 19

[\\_abc\\_negative\\_cache\\_version](#) (lago.plugins.output.DefaultOutFormatPlugin attribute), 22  
[\\_abc\\_negative\\_cache\\_version](#) (lago.plugins.output.FlatOutFormatPlugin attribute), 22  
[\\_abc\\_negative\\_cache\\_version](#) (lago.plugins.output.JSONOutFormatPlugin attribute), 22  
[\\_abc\\_negative\\_cache\\_version](#) (lago.plugins.output.OutFormatPlugin attribute), 22  
[\\_abc\\_negative\\_cache\\_version](#) (lago.plugins.output.YAMLOutFormatPlugin attribute), 23  
[\\_abc\\_negative\\_cache\\_version](#) (lago.plugins.service.ServicePlugin attribute), 23  
[\\_abc\\_negative\\_cache\\_version](#) (lago.plugins.vm.VMPlugin attribute), 24  
[\\_abc\\_negative\\_cache\\_version](#) (lago.service.SysVInitService attribute), 43  
[\\_abc\\_negative\\_cache\\_version](#) (lago.service.SystemdContainerService attribute), 44  
[\\_abc\\_negative\\_cache\\_version](#) (lago.service.SystemdService attribute), 44  
[\\_abc\\_negative\\_cache\\_version](#) (lago.vm.DefaultVM attribute), 56  
[\\_abc\\_negative\\_cache\\_version](#) (lago\_template\_repo.TemplateRepoCLI attribute), 60  
[\\_abc\\_negative\\_cache\\_version](#) (ovirtlago.cmd.OvirtCLI attribute), 61  
[\\_abc\\_negative\\_cache\\_version](#) (ovirtlago.virt.EngineVM attribute), 64  
[\\_abc\\_negative\\_cache\\_version](#) (ovirtlago.virt.HEHostVM attribute), 64  
[\\_abc\\_negative\\_cache\\_version](#) (ovirtlago.virt.HostVM attribute), 64  
[\\_abc\\_negative\\_cache\\_version](#) (ovirtlago.virt.NodeVM attribute), 65  
[\\_abc\\_registry](#) (lago.plugins.cli.CLIPugin attribute), 19  
[\\_abc\\_registry](#) (lago.plugins.cli.CLIPuginFuncWrapper attribute), 19  
[\\_abc\\_registry](#) (lago.plugins.output.DefaultOutFormatPlugin attribute), 22  
[\\_abc\\_registry](#) (lago.plugins.output.FlatOutFormatPlugin attribute), 22  
[\\_abc\\_registry](#) (lago.plugins.output.JSONOutFormatPlugin attribute), 22  
[\\_abc\\_registry](#) (lago.plugins.output.OutFormatPlugin attribute), 22  
[\\_abc\\_registry](#) (lago.plugins.output.YAMLOutFormatPlugin attribute), 23  
[\\_abc\\_registry](#) (lago.plugins.service.ServicePlugin attribute), 23  
[\\_abc\\_registry](#) (lago.plugins.vm.VMPlugin attribute), 24  
[\\_abc\\_registry](#) (lago.service.SysVInitService attribute), 43  
[\\_abc\\_registry](#) (lago.service.SystemdContainerService attribute), 44  
[\\_abc\\_registry](#) (lago.service.SystemdService attribute), 44  
[\\_abc\\_registry](#) (lago.vm.DefaultVM attribute), 57  
[\\_abc\\_registry](#) (lago\_template\_repo.TemplateRepoCLI attribute), 60  
[\\_abc\\_registry](#) (ovirtlago.cmd.OvirtCLI attribute), 61  
[\\_abc\\_registry](#) (ovirtlago.virt.EngineVM attribute), 64  
[\\_abc\\_registry](#) (ovirtlago.virt.HEHostVM attribute), 64  
[\\_abc\\_registry](#) (ovirtlago.virt.HostVM attribute), 64  
[\\_abc\\_registry](#) (ovirtlago.virt.NodeVM attribute), 65  
[\\_acquire\(\)](#) (in module lago.subnet\_lease), 45  
[\\_addFault\(\)](#) (ovirtlago.testlib.LogCollectorPlugin method), 62  
[\\_add\\_nic\\_to\\_mapping\(\)](#) (lago.prefix.Prefix method), 37  
[\\_add\\_subparser\\_to\\_cp\(\)](#) (in module lago.utils), 53  
[\\_allocate\\_ips\\_to\\_nics\(\)](#) (lago.prefix.Prefix method), 37  
[\\_allocate\\_subnets\(\)](#) (lago.prefix.Prefix method), 37  
[\\_artifact\\_paths\(\)](#) (lago.plugins.vm.VMPlugin method), 24  
[\\_artifact\\_paths\(\)](#) (ovirtlago.virt.EngineVM method), 64  
[\\_artifact\\_paths\(\)](#) (ovirtlago.virt.HEHostVM method), 64  
[\\_artifact\\_paths\(\)](#) (ovirtlago.virt.HostVM method), 65  
[\\_artifact\\_paths\(\)](#) (ovirtlago.virt.NodeVM method), 65  
[\\_brctl\(\)](#) (in module lago.brctl), 27  
[\\_check\\_alive\(\)](#) (in module lago.plugins.vm), 27  
[\\_check\\_defined\(\)](#) (in module lago.vm), 58  
[\\_check\\_predefined\\_subnets\(\)](#) (lago.prefix.Prefix static method), 38  
[\\_compatible\\_cpu\\_and\\_family](#) (lago.virt.VirtEnv attribute), 55  
[\\_config\\_net\\_interface\(\)](#) (in module lago.sysprep), 46  
[\\_config\\_net\\_topology\(\)](#) (lago.prefix.Prefix method), 38  
[\\_copy\\_delpoy\\_scripts\(\)](#) (lago.prefix.Prefix method), 38  
[\\_copy\\_deploy\\_scripts\\_for\\_hosts\(\)](#) (lago.prefix.Prefix method), 38  
[\\_create\\_api\(\)](#) (ovirtlago.virt.EngineVM method), 64  
[\\_create\\_dead\\_snapshot\(\)](#) (lago.vm.LocalLibvirtVMProvider method), 57  
[\\_create\\_disk\(\)](#) (lago.prefix.Prefix method), 38  
[\\_create\\_disks\(\)](#) (lago.prefix.Prefix method), 38  
[\\_create\\_http\\_server\(\)](#) (in module ovirtlago.utils), 63  
[\\_create\\_ip\(\)](#) (in module lago.prefix), 43  
[\\_create\\_live\\_snapshot\(\)](#) (lago.vm.LocalLibvirtVMProvider method), 57  
[\\_create\\_net\(\)](#) (lago.virt.VirtEnv method), 55

\_create\_rpm\_repository() (ovirtlago.OvirtPrefix method), 61  
 \_create\_ssh\_keys() (lago.prefix.Prefix method), 38  
 \_create\_virt\_env() (lago.prefix.Prefix method), 38  
 \_create\_virt\_env() (ovirtlago.OvirtPrefix method), 61  
 \_create\_vm() (lago.virt.VirtEnv method), 55  
 \_create\_vm() (ovirtlago.virt.OvirtVirtEnv method), 65  
 \_deploy\_host() (lago.prefix.Prefix method), 39  
 \_detect\_service\_provider() (lago.plugins.vm.VMPlugin method), 24  
 \_dom (lago.templates.TemplateRepository attribute), 49  
 \_extract\_paths\_gfs() (lago.vm.LocalLibvirtVMProvider method), 57  
 \_extract\_paths\_scp() (lago.plugins.vm.VMProviderPlugin method), 26  
 \_fix\_reposync\_issues() (in module ovirtlago.reposetup), 62  
 \_gen\_ssh\_command\_id() (in module lago.ssh), 44  
 \_gen\_ssh\_command\_id() (in module lago.virt), 56  
 \_generate\_disk\_name() (lago.prefix.Prefix static method), 39  
 \_generate\_disk\_path() (lago.prefix.Prefix method), 39  
 \_get\_api() (ovirtlago.virt.EngineVM method), 64  
 \_get\_cipher() (in module lago.ssh), 44  
 \_get\_configs\_path() (in module lago.config), 29  
 \_get\_metadata() (lago.prefix.Prefix method), 39  
 \_get\_provider() (lago.templates.TemplateRepository method), 49  
 \_get\_scripts() (lago.prefix.Prefix method), 39  
 \_get\_service\_provider() (lago.plugins.vm.VMPlugin method), 24  
 \_get\_vm\_provider() (lago.plugins.vm.VMPlugin method), 24  
 \_guestfs\_copy\_path() (in module lago.virt), 56  
 \_guestfs\_copy\_path() (in module lago.vm), 58  
 \_handle\_empty\_disk() (lago.prefix.Prefix method), 39  
 \_handle\_file\_disk() (lago.prefix.Prefix method), 39  
 \_handle\_lago\_template() (lago.prefix.Prefix method), 39  
 \_handle\_ova\_image() (lago.prefix.Prefix method), 39  
 \_handle\_qcow\_template() (lago.prefix.Prefix method), 39  
 \_handle\_template() (lago.prefix.Prefix method), 39  
 \_init\_net\_specs() (lago.prefix.Prefix static method), 39  
 \_init\_users() (lago.templates.TemplateStore method), 50  
 \_instance\_of\_any() (in module ovirtlago.testlib), 63  
 \_ip\_in\_subnet() (in module lago.prefix), 43  
 \_lease\_owned() (in module lago.subnet\_lease), 45  
 \_lease\_valid() (in module lago.subnet\_lease), 45  
 \_libvirt\_name() (lago.virt.Network method), 55  
 \_libvirt\_name() (lago.vm.LocalLibvirtVMProvider method), 57  
 \_libvirt\_xml() (lago.virt.BridgeNetwork method), 54  
 \_libvirt\_xml() (lago.virt.NATNetwork method), 55  
 \_libvirt\_xml() (lago.virt.Network method), 55  
 \_libvirt\_xml() (lago.vm.LocalLibvirtVMProvider method), 57  
 \_lock\_path() (in module lago.dirlock), 29  
 \_locked() (in module lago.subnet\_lease), 45  
 \_locked() (in module lago.templates), 51  
 \_main\_thread\_lock (lago.log\_utils.TaskHandler attribute), 33  
 \_metadata (lago.prefix.Prefix attribute), 37  
 \_normalize\_spec() (lago.plugins.vm.VMPlugin class method), 24  
 \_ova\_to\_spec() (lago.prefix.Prefix method), 39  
 \_path\_to\_xml() (in module lago.virt), 56  
 \_path\_to\_xml() (in module lago.vm), 58  
 \_paths (lago.prefix.Prefix attribute), 37  
 \_populate\_parser() (in module ovirtlago.cmd), 61  
 \_prefix (lago.prefix.Prefix attribute), 37  
 \_prefixed() (lago.templates.FileSystemTemplateProvider method), 47  
 \_prefixed() (lago.templates.TemplateStore method), 50  
 \_prepare\_domain\_image() (lago.prefix.Prefix method), 40  
 \_prepare\_domains\_images() (lago.prefix.Prefix method), 40  
 \_providers (lago.templates.TemplateRepository attribute), 49  
 \_reclaim\_disk() (lago.vm.LocalLibvirtVMProvider method), 57  
 \_reclaim\_disks() (lago.vm.LocalLibvirtVMProvider method), 57  
 \_register\_preallocated\_ips() (lago.prefix.Prefix method), 40  
 \_release() (in module lago.subnet\_lease), 45  
 \_request\_start() (lago.plugins.service.ServicePlugin method), 23  
 \_request\_start() (lago.service.SysVInitService method), 43  
 \_request\_start() (lago.service.SystemdContainerService method), 44  
 \_request\_start() (lago.service.SystemdService method), 44  
 \_request\_stop() (lago.plugins.service.ServicePlugin method), 23  
 \_request\_stop() (lago.service.SysVInitService method), 43  
 \_request\_stop() (lago.service.SystemdContainerService method), 44  
 \_request\_stop() (lago.service.SystemdService method), 44  
 \_resolve\_service\_class() (in module lago.plugins.vm), 27  
 \_ret\_via\_queue() (in module lago.utils), 53  
 \_retrieve\_disk\_url() (lago.prefix.Prefix method), 40  
 \_run\_command() (in module lago.utils), 53  
 \_run\_qemu() (lago.prefix.Prefix static method), 40  
 \_save\_metadata() (lago.prefix.Prefix method), 40

`_scp()` (lago.plugins.vm.VMPlugin method), 24  
`_set_current()` (lago.workdir.Workdir method), 58  
`_set_link()` (in module lago.brctl), 27  
`_set_scripts()` (lago.prefix.Prefix method), 40  
`_take_lease()` (in module lago.subnet\_lease), 46  
`_tasks_lock` (lago.log\_utils.TaskHandler attribute), 33  
`_template_metadata()` (lago.plugins.vm.VMPlugin method), 24  
`_update_current()` (lago.workdir.Workdir method), 58  
`_upload_file()` (in module lago.sysprep), 46  
`_use_prototype()` (lago.prefix.Prefix method), 40  
`_validate_lease_dir_present()` (in module lago.subnet\_lease), 46  
`_virt_env` (lago.prefix.Prefix attribute), 37  
`_vms_capable()` (in module ovirtlago.testlib), 63  
`_write_file()` (in module lago.sysprep), 46

## A

`acquire()` (in module lago.subnet\_lease), 46  
`ACTIVE` (lago.plugins.service.ServiceState attribute), 24  
`ADD` (lago\_template\_repo.Verbs attribute), 60  
`add_argument()` (lago.plugins.cli.CLIPuginFuncWrapper method), 19  
`add_iso()` (ovirtlago.virt.EngineVM method), 64  
`add_mapping()` (lago.virt.Network method), 55  
`add_mappings()` (lago.virt.Network method), 55  
`add_prefix()` (lago.workdir.Workdir method), 58  
`add_ssh_key()` (in module lago.sysprep), 46  
`add_timestamp_suffix()` (in module lago.utils), 53  
`addError()` (ovirtlago.testlib.LogCollectorPlugin method), 62  
`addError()` (ovirtlago.testlib.TaskLogNosePlugin method), 63  
`addFailure()` (ovirtlago.testlib.LogCollectorPlugin method), 62  
`alive()` (lago.plugins.service.ServicePlugin method), 23  
`alive()` (lago.plugins.vm.VMPlugin method), 24  
`alive()` (lago.virt.Network method), 55  
`all_ips()` (lago.plugins.vm.VMPlugin method), 24  
`ALWAYS_SHOW_REG` (in module lago.log\_utils), 30  
`ALWAYS_SHOW_TRIGGER_MSG` (in module lago.log\_utils), 30  
`am_i_main_thread` (lago.log\_utils.TaskHandler attribute), 33  
`argparse_to_ini()` (in module lago.utils), 53  
`assert_equals_within()` (in module ovirtlago.testlib), 63  
`assert_equals_within_long()` (in module ovirtlago.testlib), 63  
`assert_equals_within_short()` (in module ovirtlago.testlib), 63  
`assert_true_within()` (in module ovirtlago.testlib), 63  
`assert_true_within_long()` (in module ovirtlago.testlib), 63

`assert_true_within_short()` (in module ovirtlago.testlib), 63  
`auth_callback()` (in module lago.libvirt\_utils), 30

## B

`BIN_PATH` (lago.plugins.service.ServicePlugin attribute), 23  
`BIN_PATH` (lago.service.SystemdContainerService attribute), 43  
`BIN_PATH` (lago.service.SystemdService attribute), 44  
`BIN_PATH` (lago.service.SysVInitService attribute), 43  
`bootstrap()` (lago.plugins.vm.VMPlugin method), 24  
`bootstrap()` (lago.plugins.vm.VMProviderPlugin method), 26  
`bootstrap()` (lago.virt.VirtEnv method), 55  
`bootstrap()` (lago.vm.LocalLibvirtVMProvider method), 57  
`bootstrap()` (lago.vm.SSHVMProvider method), 58  
`BridgeNetwork` (class in lago.virt), 54  
`buffer_size` (lago.log\_utils.TaskHandler attribute), 33  
`build_dir()` (ovirtlago.paths.OvirtPaths method), 61

## C

`check_deps()` (in module lago.cmd), 28  
`check_group_membership()` (in module lago.cmd), 28  
`cleanup()` (lago.prefix.Prefix method), 40  
`clear()` (lago.utils.RollbackContext method), 52  
`cli_plugin()` (in module lago.plugins.cli), 19  
`cli_plugin_add_argument()` (in module lago.plugins.cli), 20  
`cli_plugin_add_help()` (in module lago.plugins.cli), 21  
`CLIPugin` (class in lago.plugins.cli), 18  
`CLIPuginFuncWrapper` (class in lago.plugins.cli), 19  
`close_children_tasks()` (lago.log\_utils.TaskHandler method), 33  
`collect_artifacts()` (lago.plugins.vm.VMPlugin method), 24  
`collect_artifacts()` (lago.prefix.Prefix method), 41  
`colored()` (lago.log\_utils.ColorFormatter class method), 31  
`ColorFormatter` (class in lago.log\_utils), 31  
`CommandStatus` (class in lago.utils), 52  
`config_net_interface_dhcp()` (in module lago.sysprep), 46  
`ConfigLoad` (class in lago.config), 28  
`configure()` (ovirtlago.testlib.LogCollectorPlugin method), 62  
`configure()` (ovirtlago.testlib.TaskLogNosePlugin method), 63  
`CONFS_PATH` (in module lago.constants), 29  
`ContextLock` (class in lago.log\_utils), 31  
`copy_from()` (lago.plugins.vm.VMPlugin method), 25  
`copy_to()` (lago.plugins.vm.VMPlugin method), 25  
`cpu_model` (lago.vm.LocalLibvirtVMProvider attribute), 57

create() (in module lago.brctl), 27  
 create\_parser() (in module lago.cmd), 28  
 create\_snapshot() (lago.plugins.vm.VMPlugin method), 25  
 create\_snapshot() (lago.plugins.vm.VMProviderPlugin method), 26  
 create\_snapshot() (lago.vm.LocalLibvirtVMProvider method), 57  
 create\_snapshot() (lago.vm.SSHVMProvider method), 58  
 create\_snapshots() (lago.prefix.Prefix method), 41  
 create\_snapshots() (lago.virt.VirtEnv method), 55  
 CRITICAL (lago.log\_utils.ColorFormatter attribute), 31  
 cur\_depth\_level (lago.log\_utils.TaskHandler attribute), 33  
 cur\_task (lago.log\_utils.TaskHandler attribute), 34  
 cur\_thread (lago.log\_utils.TaskHandler attribute), 34  
 CYAN (lago.log\_utils.ColorFormatter attribute), 31

## D

DEBUG (lago.log\_utils.ColorFormatter attribute), 31  
 deepcopy() (in module lago.utils), 53  
 DEFAULT (lago.log\_utils.ColorFormatter attribute), 31  
 DefaultOutFormatPlugin (class in lago.plugins.output), 21  
 DefaultVM (class in lago.vm), 56  
 defer() (lago.utils.RollbackContext method), 52  
 defined() (lago.plugins.vm.VMPlugin method), 25  
 defined() (lago.plugins.vm.VMProviderPlugin method), 26  
 defined() (lago.vm.LocalLibvirtVMProvider method), 57  
 defined() (lago.vm.SSHVMProvider method), 58  
 deploy() (lago.prefix.Prefix method), 41  
 deploy() (ovirtlago.OvirtPrefix method), 61  
 destroy() (in module lago.brctl), 27  
 destroy() (lago.prefix.Prefix method), 41  
 destroy() (lago.workdir.Workdir method), 59  
 distro() (lago.plugins.vm.VMPlugin method), 25  
 do\_add() (in module lago\_template\_repo), 60  
 do\_run() (lago.plugins.cli.CLIPlugin method), 19  
 do\_run() (lago.plugins.cli.CLIPluginFuncWrapper method), 19  
 do\_run() (lago\_template\_repo.TemplateRepoCLI method), 60  
 do\_run() (ovirtlago.cmd.OvirtCLI method), 61  
 do\_update() (in module lago\_template\_repo), 60  
 Domain (class in lago.libvirt\_utils), 30  
 DOMAIN\_STATES (in module lago.libvirt\_utils), 30  
 download() (lago.templates.TemplateStore method), 50  
 download() (lago.templates.TemplateVersion method), 51  
 download\_image() (lago.templates.FileSystemTemplateProvider method), 47  
 download\_image() (lago.templates.HttpTemplateProvider method), 47  
 drain\_ssh\_channel() (in module lago.ssh), 44

dump\_level (lago.log\_utils.TaskHandler attribute), 33

## E

edit() (in module lago.sysprep), 46  
 EggTimer (class in lago.utils), 52  
 elapsed() (lago.utils.EggTimer method), 52  
 elapsed\_time() (lago.log\_utils.Task method), 32  
 emit() (lago.log\_utils.TaskHandler method), 34  
 end\_log\_task() (in module lago.log\_utils), 35  
 END\_TASK\_MSG (in module lago.log\_utils), 31  
 END\_TASK\_REG (in module lago.log\_utils), 31  
 END\_TASK\_TRIGGER\_MSG (in module lago.log\_utils), 31  
 engine\_capability() (in module ovirtlago.testlib), 63  
 engine\_setup() (ovirtlago.virt.EngineVM method), 64  
 engine\_vm() (ovirtlago.virt.OvirtVirtEnv method), 65  
 EngineVM (class in ovirtlago.virt), 64  
 ERROR (lago.log\_utils.ColorFormatter attribute), 31  
 ExceptionTimer (class in lago.utils), 52  
 exists() (in module lago.brctl), 27  
 exists() (lago.plugins.service.ServicePlugin method), 23  
 extract\_if\_needed() (lago.templates.HttpTemplateProvider static method), 48  
 extract\_paths() (lago.plugins.vm.VMPlugin method), 25  
 extract\_paths() (lago.plugins.vm.VMProviderPlugin method), 26  
 extract\_paths() (lago.vm.LocalLibvirtVMProvider method), 57  
 ExtractPathError, 24  
 ExtractPathNoPathError, 24

## F

failed (lago.log\_utils.Task attribute), 32  
 fetch\_url() (lago.prefix.Prefix method), 41  
 FileSystemTemplateProvider (class in lago.templates), 47  
 find\_repo\_by\_name() (in module lago.templates), 51  
 FlatOutFormatPlugin (class in lago.plugins.output), 22  
 force\_show (lago.log\_utils.Task attribute), 32  
 format() (lago.log\_utils.ColorFormatter method), 31  
 format() (lago.plugins.output.DefaultOutFormatPlugin method), 22  
 format() (lago.plugins.output.FlatOutFormatPlugin method), 22  
 format() (lago.plugins.output.JSONOutFormatPlugin method), 22  
 format() (lago.plugins.output.OutFormatPlugin method), 22  
 format() (lago.plugins.output.YAMLOutFormatPlugin method), 23  
 formatter (lago.log\_utils.TaskHandler attribute), 32  
 from\_prefix() (lago.virt.VirtEnv class method), 56  
 from\_url() (lago.templates.TemplateRepository class method), 49  
 func\_vector() (in module lago.utils), 53



## G

`generate_request_handler()` (in module `ovirtlago.utils`), 63  
`get()` (`lago.config.ConfigLoad` method), 28  
`get_api()` (`ovirtlago.virt.EngineVM` method), 64  
`get_api_v3()` (`ovirtlago.virt.EngineVM` method), 64  
`get_api_v4()` (`ovirtlago.virt.EngineVM` method), 64  
`get_by_name()` (`lago.templates.TemplateRepository` method), 49  
`get_compatible_cpu_and_family()` (`lago.virt.VirtEnv` method), 56  
`get_cpu_model()` (`lago.virt.VirtEnv` method), 56  
`get_env_dict()` (in module `lago.config`), 29  
`get_hash()` (`lago.templates.FileSystemTemplateProvider` method), 47  
`get_hash()` (`lago.templates.HttpTemplateProvider` method), 48  
`get_hash()` (`lago.templates.TemplateVersion` method), 51  
`get_ini()` (`lago.config.ConfigLoad` method), 28  
`get_latest_version()` (`lago.templates.Template` method), 48  
`get_libvirt_connection()` (in module `lago.libvirt_utils`), 30  
`get_metadata()` (`lago.templates.FileSystemTemplateProvider` method), 47  
`get_metadata()` (`lago.templates.HttpTemplateProvider` method), 48  
`get_metadata()` (`lago.templates.TemplateVersion` method), 51  
`get_net()` (`lago.virt.VirtEnv` method), 56  
`get_nets()` (`lago.prefix.Prefix` method), 41  
`get_nets()` (`lago.virt.VirtEnv` method), 56  
`get_ovirt_cpu_family()` (`ovirtlago.virt.OvirtVirtEnv` method), 65  
`get_path()` (`lago.templates.TemplateStore` method), 50  
`get_prefix()` (`lago.workdir.Workdir` method), 59  
`get_prefixed_name()` (in module `ovirtlago.testlib`), 63  
`get_section()` (`lago.config.ConfigLoad` method), 28  
`get_snapshots()` (`lago.prefix.Prefix` method), 41  
`get_snapshots()` (`lago.virt.VirtEnv` method), 56  
`get_ssh_client()` (in module `lago.ssh`), 44  
`get_stored_hash()` (`lago.templates.TemplateStore` method), 50  
`get_stored_metadata()` (`lago.templates.TemplateStore` method), 51  
`get_task_indicator()` (`lago.log_utils.TaskHandler` method), 34  
`get_tasks()` (`lago.log_utils.TaskHandler` method), 34  
`get_test_prefix()` (in module `ovirtlago.testlib`), 63  
`get_version()` (`lago.templates.Template` method), 48  
`get_vm()` (`lago.virt.VirtEnv` method), 56  
`get_vms()` (`lago.prefix.Prefix` method), 41  
`get_vms()` (`lago.virt.VirtEnv` method), 56  
`GREEN` (`lago.log_utils.ColorFormatter` attribute), 31  
`guest_agent()` (`lago.plugins.vm.VMPlugin` method), 25  
`gw()` (`lago.virt.Network` method), 55

## H

`handle_closed_task()` (`lago.log_utils.TaskHandler` method), 34  
`handle_error()` (`lago.log_utils.TaskHandler` method), 34  
`handle_new_task()` (`lago.log_utils.TaskHandler` method), 34  
`has_guest_agent()` (`lago.plugins.vm.VMPlugin` method), 25  
`HEHostVM` (class in `ovirtlago.virt`), 64  
`hide_paramiko_logs()` (in module `lago.log_utils`), 35  
`hide_stevedore_logs()` (in module `lago.log_utils`), 35  
`HOST_BIN_PATH` (`lago.service.SystemdContainerService` attribute), 44  
`host_capability()` (in module `ovirtlago.testlib`), 63  
`host_vms()` (`ovirtlago.virt.OvirtVirtEnv` method), 65  
`HostVM` (class in `ovirtlago.virt`), 64  
`HttpTemplateProvider` (class in `lago.templates`), 47

## I

`images()` (`lago.paths.Paths` method), 36  
`in_prefix()` (in module `lago.utils`), 53  
`INACTIVE` (`lago.plugins.service.ServiceState` attribute), 24  
`indent_unit` (`lago.plugins.output.DefaultOutFormatPlugin` attribute), 22  
`INFO` (`lago.log_utils.ColorFormatter` attribute), 31  
`init_args` (`lago.plugins.cli.CLIPugin` attribute), 19  
`init_args` (`lago.plugins.cli.CLIPuginFuncWrapper` attribute), 19  
`init_args` (`lago_template_repo.TemplateRepoCLI` attribute), 60  
`init_args` (`ovirtlago.cmd.OvirtCLI` attribute), 61  
`initial_depth` (`lago.log_utils.TaskHandler` attribute), 33  
`initialize()` (`lago.prefix.Prefix` method), 41  
`initialize()` (`lago.workdir.Workdir` method), 59  
`interactive_console()` (`lago.plugins.vm.VMPlugin` method), 25  
`interactive_console()` (`lago.plugins.vm.VMProviderPlugin` method), 26  
`interactive_console()` (`lago.vm.LocalLibvirtVMProvider` method), 57  
`interactive_ssh()` (in module `lago.ssh`), 44  
`interactive_ssh()` (`lago.plugins.vm.VMPlugin` method), 25  
`interactive_ssh_channel()` (in module `lago.ssh`), 44  
`internal_repo()` (`ovirtlago.paths.OvirtPaths` method), 61  
`invoke_in_parallel()` (in module `lago.utils`), 53  
`ip()` (`lago.plugins.vm.VMPlugin` method), 25  
`ipv4_to_mac()` (in module `lago.utils`), 53  
`is_leasable_subnet()` (in module `lago.subnet_lease`), 46  
`is_management()` (`lago.virt.Network` method), 55  
`is_prefix()` (`lago.prefix.Prefix` class method), 41  
`is_supported()` (`lago.plugins.service.ServicePlugin` class method), 23

is\_workdir() (lago.workdir.Workdir class method), 59  
 iscsi\_name() (lago.plugins.vm.VMPlugin method), 25

## J

join() (lago.workdir.Workdir method), 39  
 join\_all() (lago.utils.VectorThread method), 52  
 json\_dump() (in module lago.utils), 53  
 JSONOutFormatPlugin (class in lago.plugins.output), 22

## L

lago (module), 17  
 lago.brctl (module), 27  
 lago.cmd (module), 28  
 lago.config (module), 28  
 lago.constants (module), 29  
 lago.dirlock (module), 29  
 lago.libvirt\_utils (module), 30  
 lago.log\_utils (module), 30  
 lago.paths (module), 36  
 lago.plugins (module), 17  
 lago.plugins.cli (module), 17  
 lago.plugins.output (module), 21  
 lago.plugins.service (module), 23  
 lago.plugins.vm (module), 24  
 lago.prefix (module), 37  
 lago.service (module), 43  
 lago.ssh (module), 44  
 lago.subnet\_lease (module), 45  
 lago.sysprep (module), 46  
 lago.templates (module), 47  
 lago.utils (module), 52  
 lago.virt (module), 54  
 lago.vm (module), 56  
 lago.workdir (module), 58  
 lago\_template\_repo (module), 60  
 level (lago.log\_utils.TaskHandler attribute), 33  
 LIBEXEC\_DIR (in module lago.constants), 29  
 LIBVIRT\_CONNECTIONS (in module lago.libvirt\_utils), 30  
 load() (lago.config.ConfigLoad method), 28  
 load() (lago.workdir.Workdir method), 59  
 load\_plugins() (in module lago.plugins), 17  
 load\_virt\_stream() (in module lago.utils), 53  
 LocalLibvirtVMProvider (class in lago.vm), 57  
 lock() (in module lago.dirlock), 29  
 LockFile (class in lago.utils), 52  
 log\_always() (in module lago.log\_utils), 36  
 log\_task() (in module lago.log\_utils), 36  
 LogCollectorPlugin (class in ovirtlago.testlib), 62  
 logs() (lago.paths.Paths method), 36  
 LogTask (class in lago.log\_utils), 31

## M

main() (in module lago.cmd), 28

main\_failed (lago.log\_utils.TaskHandler attribute), 33  
 MalformedWorkdir, 58  
 mapping() (lago.virt.Network method), 55  
 mark\_main\_tasks\_as\_failed()  
     (lago.log\_utils.TaskHandler method), 34  
 mark\_parent\_tasks\_as\_failed()  
     (lago.log\_utils.TaskHandler method), 34  
 mark\_used() (lago.templates.TemplateStore method), 51  
 MAX\_SUBNET (in module lago.subnet\_lease), 45  
 merge() (in module ovirtlago.reposetup), 62  
 metadata (lago.plugins.vm.VMPlugin attribute), 25  
 metadata() (lago.paths.Paths method), 36  
 MIN\_SUBNET (in module lago.subnet\_lease), 45  
 MISSING (lago.plugins.service.ServiceState attribute), 24

## N

name (lago.log\_utils.Task attribute), 32  
 name (lago.templates.Template attribute), 48  
 name (lago.templates.TemplateRepository attribute), 49  
 name (ovirtlago.testlib.LogCollectorPlugin attribute), 62  
 name (ovirtlago.testlib.TaskLogNosePlugin attribute), 63  
 name() (lago.plugins.vm.VMPlugin method), 25  
 name() (lago.virt.Network method), 55  
 NATNetwork (class in lago.virt), 55  
 nets() (lago.plugins.vm.VMPlugin method), 25  
 Network (class in lago.virt), 55  
 nics() (lago.plugins.vm.VMPlugin method), 25  
 NodeVM (class in ovirtlago.virt), 65  
 NONE (lago.log\_utils.ColorFormatter attribute), 31  
 NoSuchPluginError, 17

## O

open\_url() (lago.templates.HttpTemplateProvider method), 48  
 options() (ovirtlago.testlib.LogCollectorPlugin method), 62  
 options() (ovirtlago.testlib.TaskLogNosePlugin method), 63  
 OutFormatPlugin (class in lago.plugins.output), 22  
 OvirtCLI (class in ovirtlago.cmd), 61  
 ovirtlago (module), 61  
 ovirtlago.cmd (module), 61  
 ovirtlago.constants (module), 61  
 ovirtlago.paths (module), 61  
 ovirtlago.reposetup (module), 62  
 ovirtlago.testlib (module), 62  
 ovirtlago.utils (module), 63  
 ovirtlago.virt (module), 64  
 OvirtPaths (class in ovirtlago.paths), 61  
 OvirtPrefix (class in ovirtlago), 61  
 OvirtVirtEnv (class in ovirtlago.virt), 65  
 OvirtWorkdir (class in ovirtlago), 61

## P

Paths (class in lago.paths), 36  
Plugin (class in lago.plugins), 17  
PLUGIN\_ENTRY\_POINTS (in module lago.plugins), 17  
PluginError, 17  
populate\_parser() (lago.plugins.cli.CLIPugin method), 19  
populate\_parser() (lago.plugins.cli.CLIPuginFuncWrapper method), 19  
populate\_parser() (lago\_template\_repo.TemplateRepoCLI method), 60  
populate\_parser() (ovirtlago.cmd.OvirtCLI method), 61  
Prefix (class in lago.prefix), 37  
prefix\_lagofile() (lago.paths.Paths method), 36  
PrefixAlreadyExists, 58  
prefixed() (lago.paths.Paths method), 36  
prefixed\_name() (lago.virt.VirtEnv method), 56  
PrefixNotFound, 58  
prepare\_repo() (ovirtlago.OvirtPrefix method), 61  
prependDefer() (lago.utils.RollbackContext method), 52  
pretty\_emit() (lago.log\_utils.TaskHandler method), 35

## R

read\_nonblocking() (in module lago.utils), 54  
RED (lago.log\_utils.ColorFormatter attribute), 31  
release() (in module lago.subnet\_lease), 46  
repo\_server\_context() (in module ovirtlago.utils), 64  
RepositoryError, 62  
RepositoryMergeError, 62  
resolve() (lago.virt.Network method), 55  
resolve\_prefix\_path() (lago.prefix.Prefix class method), 41  
resolve\_state() (lago.libvirt\_utils.Domain static method), 30  
resolve\_workdir\_path() (lago.workdir.Workdir class method), 60  
revert\_snapshot() (lago.plugins.vm.VMPlugin method), 25  
revert\_snapshot() (lago.plugins.vm.VMProviderPlugin method), 26  
revert\_snapshot() (lago.vm.LocalLibvirtVMProvider method), 57  
revert\_snapshot() (lago.vm.SSHVMProvider method), 58  
revert\_snapshots() (lago.prefix.Prefix method), 42  
revert\_snapshots() (lago.virt.VirtEnv method), 56  
RollbackContext (class in lago.utils), 52  
root\_password() (lago.plugins.vm.VMPlugin method), 25  
rotate\_dir() (in module lago.utils), 54  
run\_command() (in module lago.utils), 54  
run\_interactive\_command() (in module lago.utils), 54  
run\_test() (ovirtlago.OvirtPrefix method), 61

## S

save() (lago.plugins.vm.VMPlugin method), 25

save() (lago.prefix.Prefix method), 42  
save() (lago.virt.Network method), 55  
save() (lago.virt.VirtEnv method), 56  
score (ovirtlago.testlib.TaskLogNosePlugin attribute), 63  
scripts() (lago.paths.Paths method), 36  
serve() (ovirtlago.OvirtPrefix method), 61  
service() (lago.plugins.vm.VMPlugin method), 25  
service\_is\_enabled() (in module lago.utils), 54  
ServicePlugin (class in lago.plugins.service), 23  
ServiceState (class in lago.plugins.service), 24  
set\_current() (lago.workdir.Workdir method), 60  
set\_help() (lago.plugins.cli.CLIPuginFuncWrapper method), 19  
set\_hostname() (in module lago.sysprep), 46  
set\_init\_args() (lago.plugins.cli.CLIPuginFuncWrapper method), 19  
set\_iscsi\_initiator\_name() (in module lago.sysprep), 46  
set\_root\_password() (in module lago.sysprep), 46  
set\_selinux\_mode() (in module lago.sysprep), 46  
setup\_prefix\_logging() (in module lago.log\_utils), 36  
should\_show\_by\_depth() (lago.log\_utils.TaskHandler method), 35  
should\_show\_by\_level() (lago.log\_utils.TaskHandler method), 35  
ssh() (in module lago.ssh), 44  
ssh() (lago.plugins.vm.VMPlugin method), 25  
ssh\_id\_rsa() (lago.paths.Paths method), 37  
ssh\_id\_rsa\_pub() (lago.paths.Paths method), 37  
ssh\_reachable() (lago.plugins.vm.VMPlugin method), 25  
ssh\_script() (in module lago.ssh), 44  
ssh\_script() (lago.plugins.vm.VMPlugin method), 25  
SSHVMProvider (class in lago.vm), 58  
start() (lago.plugins.service.ServicePlugin method), 23  
start() (lago.plugins.vm.VMPlugin method), 25  
start() (lago.plugins.vm.VMProviderPlugin method), 27  
start() (lago.prefix.Prefix method), 42  
start() (lago.utils.ExceptionTimer method), 52  
start() (lago.virt.BridgeNetwork method), 54  
start() (lago.virt.Network method), 55  
start() (lago.virt.VirtEnv method), 56  
start() (lago.vm.LocalLibvirtVMProvider method), 57  
start() (lago.vm.SSHVMProvider method), 58  
start\_all() (lago.utils.VectorThread method), 53  
start\_log\_task() (in module lago.log\_utils), 36  
START\_TASK\_MSG (in module lago.log\_utils), 32  
START\_TASK\_REG (in module lago.log\_utils), 32  
START\_TASK\_TRIGGER\_MSG (in module lago.log\_utils), 32  
startTest() (ovirtlago.testlib.TaskLogNosePlugin method), 63  
state() (lago.plugins.service.ServicePlugin method), 23  
state() (lago.plugins.vm.VMPlugin method), 26  
state() (lago.plugins.vm.VMProviderPlugin method), 27



state() (lago.service.SystemdContainerService method), 44

state() (lago.service.SystemdService method), 44

state() (lago.service.SysVInitService method), 43

state() (lago.vm.LocalLibvirtVMProvider method), 57

state() (lago.vm.SSHVMProvider method), 58

stop() (lago.plugins.service.ServicePlugin method), 23

stop() (lago.plugins.vm.VMPlugin method), 26

stop() (lago.plugins.vm.VMProviderPlugin method), 27

stop() (lago.prefix.Prefix method), 42

stop() (lago.utils.ExceptionTimer method), 52

stop() (lago.virt.BridgeNetwork method), 55

stop() (lago.virt.Network method), 55

stop() (lago.virt.VirtEnv method), 56

stop() (lago.vm.LocalLibvirtVMProvider method), 58

stop() (lago.vm.SSHVMProvider method), 58

stop() (ovirtlago.virt.EngineVM method), 64

stopTest() (ovirtlago.testlib.TaskLogNosePlugin method), 63

sync\_rpm\_repository() (in module ovirtlago.reposetup), 62

sysprep() (in module lago.sysprep), 46

SystemdContainerService (class in lago.service), 43

SystemdService (class in lago.service), 44

SysVInitService (class in lago.service), 43

## T

Task (class in lago.log\_utils), 32

TASK\_INDICATORS (lago.log\_utils.TaskHandler attribute), 33

task\_tree\_depth (lago.log\_utils.TaskHandler attribute), 33

TaskHandler (class in lago.log\_utils), 32

TaskLogNosePlugin (class in ovirtlago.testlib), 62

tasks (lago.log\_utils.TaskHandler attribute), 35

Template (class in lago.templates), 48

TemplateRepoCLI (class in lago\_template\_repo), 60

TemplateRepository (class in lago.templates), 49

TemplateStore (class in lago.templates), 49

TemplateVersion (class in lago.templates), 51

test\_logs() (ovirtlago.paths.OvirtPaths method), 61

test\_sequence\_gen() (in module ovirtlago.testlib), 63

TimerException, 52

timestamp() (lago.templates.TemplateVersion method), 51

trylock() (in module lago.dirlock), 29

## U

unlock() (in module lago.dirlock), 30

UPDATE (lago\_template\_repo.Verbs attribute), 60

update\_args() (lago.config.ConfigLoad method), 28

update\_parser() (lago.config.ConfigLoad method), 29

uuid() (lago.paths.Paths method), 37

## V

VectorThread (class in lago.utils), 52

Verbs (class in lago\_template\_repo), 60

virt() (lago.paths.Paths method), 37

virt\_conf() (lago.prefix.Prefix method), 42

virt\_conf\_from\_stream() (lago.prefix.Prefix method), 42

virt\_env (lago.prefix.Prefix attribute), 42

VIRT\_ENV\_CLASS (lago.prefix.Prefix attribute), 37

VIRT\_ENV\_CLASS (ovirtlago.OvirtPrefix attribute), 61

virt\_path() (lago.virt.VirtEnv method), 56

VirtEnv (class in lago.virt), 55

VMError, 24

VMPlugin (class in lago.plugins.vm), 24

VMProviderPlugin (class in lago.plugins.vm), 26

## W

wait\_for\_ssh() (in module lago.ssh), 44

wait\_for\_ssh() (lago.plugins.vm.VMPlugin method), 26

wait\_for\_ssh() (ovirtlago.virt.NodeVM method), 65

WARNING (lago.log\_utils.ColorFormatter attribute), 31

WHITE (lago.log\_utils.ColorFormatter attribute), 31

with\_logging() (in module lago.utils), 54

with\_ovirt\_api() (in module ovirtlago.testlib), 63

with\_ovirt\_api4() (in module ovirtlago.testlib), 63

with\_ovirt\_prefix() (in module ovirtlago.testlib), 63

with\_repo\_server() (in module ovirtlago.reposetup), 62

Workdir (class in lago.workdir), 58

workdir\_loaded() (in module lago.workdir), 60

WorkdirError, 60

## Y

YAMLOutFormatPlugin (class in lago.plugins.output), 23

YELLOW (lago.log\_utils.ColorFormatter attribute), 31