JUMPSTORM

Jumpstorm wants to help you concentrate on the task you have to do. No more time has to be wasted for Magento setup. It even could setup your Magento automatically, so it can even be used for demo or test systems.

Its flexible architecture allows you to extend its functionality as you need.

Installation

Clone from Github:

```
git clone --recursive git@github.com:netresearch/jumpstorm.git

cd jumpstorm
```

Hint:

We recommend to symlink the executable `jumpstorm` in `/usr/bin` or at least to create an alias in your `~/.profile`:

```
echo 'alias jumpstorm="/path/to/jumpstorm/executable"' >> ~/.profile
source ~/.profile
```

After that you can leave the folder `jumpstorm` and use command `jumpstorm` instead of `./jumpstorm` in the following tutorial.

Usage

Type on command line:

```
./jumpstorm
```

and you will get a list of available commands and options.

Currently supported commands are

command	description
`magento`	Install Magento
`extensions`	Install extensions
`unittesting`	Install framework for unittests and prepare test database
`plugins`	Run plugins

The first thing you should do, is creating the configuration file, your installation should be based on. We provided a sample configuration file `ini/sample.jumpstorm.ini`, *Jumpstorm* will use `ini/jumpstorm.ini` by default, but you could specify a different configuration file by using option `-c` (`--config`):

```
./jumpstorm magento -c /path/to/my/ini
```

Every command needs the `[common]` section of the configuration file, so you should fill in the correct values of the Magento target path and your database settings.

Magento

As you propably expected, this command will install Magento.

Let's have a look at the `[magento]` section of your configuration file:

Jumpstorm needs to know where to find Magento, you will have to specify that in option `source`. [1] If you decide for Git as source, you might specify a `branch`.

During installation Magento will need to know its base url and it will create an admin account. You should set its data in your configuration file.

If you want *Jumpstorm* to install Magento sample data, you should provide its source [1] (and its branch, if source is Git).

When you finished configuration, you will get a fresh Magento installation after running

```
./jumpstorm magento -c /path/to/my/ini
```

Please note, that option `-c /path/to/my/ini` is optional.

Extensions

Jumpstorm was developed for testing, supporting, and developing extensions. So let's have a look into extension installation configuration.

In section `[extensions]` you can provide a list of extensions to be installed automatically. For every extension you have to provide a source [1] and you could provide a branch, if you use Git as source.

All configured extension will be installed by executing:

```
./jumpstorm extensions -c /path/to/my/ini
```

Please note, that option `-c /path/to/my/ini` is optional.

Unittesting

We are big fans of test driven development. So unit testing is essential. In our sample configuration file, EcomDev_PHPUnit is used as default testing framework. We recommend to use this, so just copy this section to your configuration file (you could set its branch to `dev`, if you want to use its latest features and bugfixes).

Now just run the following command and start testing and developing:

```
./jumpstorm unittesting -c /path/to/my/ini
```

Please note, that option `-c /path/to/my/ini` is optional.

Plugins

In some cases you may want to have a little different setup, e.g. some special users, products, or settings. To achieve that, *Jumpstorm* is extendible. The sample configuration file already provides a `[plugins]` section. Every plugin mentioned there will be executed by running

```
./jumpstorm plugins -c /path/to/my/ini
```

Please note, that option `-c /path/to/my/ini` is optional.

How to write your own plugins?

Plugins follow a simple structure. They are placed in `plugins` directory. Each plugin is a directory with at least one php file inside, which contains a php class. This directory, the php file and php class must have the same name (with uppercase first letter), which is used as key in configuration file. Plugins can be disabled by either not mentioning them in configuration or by setting its configuration value to `enabled = 0`.

Plugin configuration can be achieved by dot syntax:

```
PluginA.enabled = 0         => plugin will be skipped

PluginB = someValue         => plugin will be active and will have that single configuration value "someValue"

PluginC.foo = 0             => plugin will be active and will have configuration ['foo' => 0, 'bar' => 'foobar']
PluginC.bar = foobar
```

Plugin configuration might get quite extensive. Therefore a separate ini file can be provided within the plugin directory. Again, it follows the same naming conventions as mentioned above.

The plugin's main php class must implement Netresearch\PluginInterface.

Upcoming features

Modman support

As we use modman for all our extensions, we will implement *Jumpstorm* to support this as soon as possible. Currently we only copy (or clone) the extensions to the `.modman` directory, but deployment is not yet done via *modman*.

Interactive mode

If you use *Jumpstorm* for a bunch of different projects on and on, you become tired of changing the config file. So it would be nice to specify some settings to be confirmed (and corrected) during execution of *Jumpstorm*. This mode will be suppressed by the built-in option `-n (--no-interaction)`.

1(1, 2, 3) *Jumpstorm* is well-tested to support Git, but other sources like file system and ssh are supported, too.