MoinMoin - Report on Install Guides Validity

Ricardo de Freitas Gesuatto <ricardo.fang@gmail.com>

ABOUT THIS PAPER:

This report is based on MoinMoin's issue #73 at Google's Highly Open Participation Contest, which seeks to analyze if the project's installation documents makes sense or not on the current development status of the project and the open source world in general. For testing this issue, a platform is chosen for installing MoinMoin according to the documentation. The problems and results are listed step by step in this paper.

For standardizing the text on this report, normal text typed in Times font family are the steps and comments itself, while `monospaced text` represents blocks of code (mostly output from bash).

OPERATING SYSTEM AND HARDWARE:

The hardware used for the tests in this paper is basically an Intel Pentium 4 2.26GHz (no hyperthreading) with 512Mb RAM, on an ASUS P4V8X-X motherboard. Although the first IDE hard disk (hda) has 80Gb, the second IDE hard disk (hdb, 10Gb) was used during the tests. The PC also counts with a NVidia Geforce 4000MX 128Mb, but this is irrelevant for the tests.

Based on availability, the operating system chosen was Fedora 8, a community-driven Linux based operating system led by Red Hat. Fedora is not really made with servers in mind, but for standing between stability and cutting edge, it becomes a viable platform for testing the compatibility with MoinMoin and documentation validity.

Installing Fedora 8 on a LVM partitioning scheme went almost flawlessly. Some issues in 3D rendering and PulseAudio were found, but this paper is not intended to report problems in the operating system itself. These issues were left for fixing in a more appropriate time as server usage is not affected in general.

Some information about the kernel and processor can be found below:

```
[fang@fang ~]$ uname -a
Linux fang.moonrupt.com 2.6.23.1-49.fc8 #1 SMP Thu Nov 8 21:41:26 EST 2007
i686 i686 i386 GNU/Linux

[fang@fang ~]$ head -n 8 /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 15
model           : 2
model name      : Intel(R) Pentium(R) 4 CPU 2.26GHz
stepping        : 5
cpu MHz         : 2266.753
cache size      : 512 KB
```

SETUP AND DOCUMENTATION FOLLOWED:

According to the issue on which this paper is based, the web server used for testing the documentation must be different than the builtin server shipped by MoinMoin. For this report, a simulation of a high performance server scenario will be set up. MoinMoin will be run on a Lighttpd web server via FastCGI. A caching reverse proxy, namely Varnish, will be set on the top for commodity, caching typically static content.

The documentation followed step by step for this setup can be found below:

< http://moinmo.in/MoinDev/MercurialGuide >
< http://moinmo.in/HelpOnInstalling >
< http://moinmo.in/HelpOnInstalling/BasicInstallation >
< >
< http://moinmo.in/HelpOnConfiguration >
< http://moinmo.in/HelpOnInstalling/FastCgi >

For consistency on testing the documentation, important packages will be compiled from source, unless specified otherwise with a reason. The most recent release of every package will be used when compiling from source. The moin/1.6 branch of MoinMoin will be downloaded via Mercurial.

FIRST STEPS - MERCURIAL

Most recent Linux distributions comes with Python by default. Unless you install only the very base system, there is always something depending on Python thanks to it popularity. The first step is to check whether we have a working Python installation:

```
[fang@fang ~]$ python -V
Python 2.5.1
```

It works and is up to date. Now, we must install `python-devel` for later usage on compiling python based packages. It's necessary to use Fedora's own package management system in this case, as it is a development package:

```
[root@fang ~]# yum -y install python-devel
(output snipped)
```

Two packages are needed for future dependencies: `asciidoc` and `xmlto`. The first one is rather easy to install by hand, but `xmlto` has too many dependencies, many of which i find unnecessary, turning manual installation a pain... It is wiser to, once again, use Fedora's package management system, and not reinventing the wheel on this system:

```
[root@fang ~]# yum -y install asciidoc xmlto
(output snipped again)
```

Finally, we are ready to install Mercurial. Yet Mercurial is not directly related to the objective of this paper, we will use it later for getting MoinMoin itself from the repositories. So, it doesn't hurt to set it up already. Download Mercurial source code from it's project official page (http://www.selenic.com/mercurial/wiki/). The snapshot from the stable branch was downloaded and extracted:

```
[fang@fang Download]$ tar zxf mercurial-stable-snapshot.tar.gz
[fang@fang Download]$ cd mercurial-*/
```

Now, compile and install it. On this system, Python is under `/usr`, so adding `PREFIX=/usr` actually makes sense for saving some work on exporting paths later:

```
[fang@fang mercurial-d2831a5d5947]$ make all PREFIX=/usr
[fang@fang mercurial-d2831a5d5947]$ su -c "make install" PREFIX=/usr
```

THE FOCUS - MOINMOIN

The previous section set up the base for building another parts of the desired setup. It is now time to work on the main focus of this paper now: MoinMoin.

Before getting its source code, it is necessary to set up an username into ~/`.hgrc` like shown below so it's possible to identify yourself and any changes done on the repository. Setting real information is a good practice of “Internet Etiquette”.

```
[ui]
username=Ricardo de Freitas Gesuatto <ricardo.fang@gmail.com>
```

And now, download the source from MoinMoin's moin/1.6 branch using `hg` :

```
[fang@fang ~]$ hg clone http://hg.moinmo.in/moin/1.6 moin-1.6
requesting all changes
adding changesets
adding manifests
adding file changes
added 2398 changesets with 10753 changes to 2502 files
1773 files updated, 0 files merged, 0 files removed, 0 files unresolved
[fang@fang ~]$
```

This will pull the entire repository, and takes a while to finish, but subsequent updates can be done by `cd`-ing to the cloned source tree and running a `hg pull -u` as long as you do not delete it.

Anyway, continuing the process... Now cd to the source tree and install MoinMoin as recommended by it's documentation:

```
[fang@fang ~]$ cd moin-1.6/
[fang@fang moin-1.6]$ su -c "python setup.py install --prefix='/usr/local'
--record=install.log"
```

Unless `python-devel` is not installed (which was done in the “First Steps” section), this should work without problems. If no issues were found, as in this paper, then proceed to test the MoinMoin installation. The documentation shows a simple way to test it, simply try importing MoinMoin in `python` :

```
[fang@fang moin-1.6]$ python
Python 2.5.1 (r251:54863, Oct 30 2007, 13:54:11)
[GCC 4.1.2 20070925 (Red Hat 4.1.2-33)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import MoinMoin
>>>
```

Nothing was displayed in response to the “import” command, meaning no error occurred and it was successful. If instead it returns “`ImportError: No module named MoinMoin`”, set `sys.path` to the path of MoinMoin's Python with the prefix used for installation:

```
>>> import sys
>>> sys.path.insert(0, '/usr/local/lib/python2.5/site-packages')
>>> import MoinMoin
>>>
```

Success! The next step is to create a wiki instance. This consists on copying several files and directories from MoinMoin to a new directory. Each wiki instance is independent from the others. This can be done by hand, but there is a script for doing this automatically. It can be obtained from MoinMoin's Wiki (http://tinyurl.com/3xyrxm – the original URL is huge).

The script `createinstance.sh` can help a lot on this task, but it needs some tweaking first. First, edit the `SHARE` variable to the path of MoinMoin shared files. `USER` and `GROUP` can also be edited if the server is planed to run as another user and group. For this paper, the following was edited in the script:

```
# path of MoinMoin shared files
SHARE=/usr/local/share/moin

[...]

# should be nice
USER=www-srv
GROUP=www-srv
```

Then, make the script executable and run it as root. It takes a single parameter: the desired location of the wiki instance, either as a relative or absolute path:

```
[root@fang Download]# chmod +x createinstance.sh
[root@fang Download]# ./createinstance.sh /srv/www/wiki
cp: cannot stat `/usr/local/share/moin/underlay': No such file or directory
Done.
```

One directory, `underlay`, could not be copied. After some searching, this directory was found as a .tar.bz2 archive in the share directory... And so the wiki instance is deleted, and the steps above are repeated after extracting the archive:

```
[root@fang Download]# cd /usr/local/share/moin/
[root@fang moin]# tar xjf underlay.tar.bz2
```

(back to the script directory)

```
[root@fang Download]# ./createinstance.sh /srv/www/wiki
Done.
[root@fang Download]#
```

The wiki instance was created with success. It is a good idea to take a look in `wikiconfig.py`. Most settings will suffice for a small to medium wiki, but tweaking is still necessary. It is a good idea, for example, to set up a name (`sitename`), logo (`logo_string`) and main page (`page_front_page`) for the wiki, as well as setting yourself as superuser (`superuser` and `acl_rights_before`).

SOME MORE STRUCTURE WORK

Before setting the web server, some more structure work is still required. First, copy the FastCGI script from MoinMoin's share directory to the wiki instance. It actually needs to be edited, so it is easier to have a copy for each wiki instance:

```
[root@fang ~]# cp /usr/local/share/moin/server/moin.fcg /srv/www/wiki/
[root@fang ~]# chown www-srv www-srv /srv/www/wiki/moin.fcg
```

After copying, set it's permissions accordingly, so it can be executed from the web server configuration. Edit this script, setting the paths of the `MoinMoin` package itself and `wikiconfig.py`:

```
# Path to MoinMoin package, needed if you installed with --prefix=PREFIX
# or if you did not use setup.py.
sys.path.insert(0, '/usr/local/lib/python2.5/site-packages')

# Path of the directory where wikiconfig.py is located.
# YOU NEED TO CHANGE THIS TO MATCH YOUR SETUP.
sys.path.insert(0, '/srv/www/wiki')
```

For the web server, two directories will be created at the wiki instance. The first one, `htdocs`, will serve as a document root, needed for running a web server, while still allowing the server to host a non-wiki page:

```
[root@fang fang]# cd /srv/www/wiki/
[root@fang wiki]# mkdir htdocs
[root@fang wiki]# chown -R www-srv:www-srv htdocs/
```

And the second one, `moin_static160`, can be more confusing: it is actually the `htodcs` located inside MoinMoin's share directory. Copying it as `moin_static160` inside our previously created `htdocs` is not an advantage: it is a loss of modularity. However, two scenarios are possible: if changing it's default contents, or developing new themes etc, is a possibility, then copying it to your wiki instance can be useful to keep the defaults intact:

```
[root@fang wiki]# cp -R /usr/local/share/moin/htdocs/ moin_static160/
[root@fang wiki]# chown -R www-srv:www-srv moin_static160/
[root@fang wiki]# chmod -R 770 moin_static160/
```

If the above will be kept intact, then do not copy the directory. It will be taken care of later, by using an `alias`.

THE WEB SERVER

After setting up every single package needed for our server setup, it's time to set up the web server itself. For the high performance scenario in this paper, Lighttpd was chosen for it's remarkable speed, thanks to a threaded, event-driven architecture.

After downloading Lighttpd's latest release source tarball from it's official page (http://www.lighttpd.net/), extract it and `cd` to the extracted directory:

```
[fang@fang Download]$ tar xjf lighttpd-1.4.18.tar.bz2  (if you downloaded the .tar.bz2)
[fang@fang Download]$ tar zxf lighttpd-1.4.18.tar.gz (if you downloaded the .tar.gz)
[fang@fang Download]$ cd lighttpd-1.4.18/
```

Now, configure it according to your needs. In this paper, IPv6 was disabled as this feature is not available in Brasil, at least for not for non-server customers. For optional fam/gamin support, which reduces the number of stat() calls, you may need to install the development package, named `gamin-devel` in this case. The same applies for OpenSSL (`openssl-devel`) and bzip2 (`bzip2-devel`) support.

```
 [fang@fang lighttpd-1.4.18]$ ./configure --prefix=/usr/local --disable-
ipv6 --with-bzip2 --with-fam --with-openssl
```

If nothing goes wrong for missing dependencies or development packages, it's ready to be compiled and installed:

```
[fang@fang lighttpd-1.4.18]$ make
[fang@fang lighttpd-1.4.18]$ su -c "make install"
```

The disadvantage of installing from the source it that system scripts such as init / rc.d scripts must be created manually. Creating one is really simple, it is even possible to pick an existing one as a basis and edit it according to the program's options and location. After creating the file, running `chkconfig --add` was necessary on this system to get it working.

Finally, create a configuration file for Lighttpd suitable for your needs. This usually varies from server to server, as the scenario and the sysadmin are never the same. But for running MoinMoin via FastCGI, some specific details are required in the configuration file.

First, check if mod_alias and mod_fastcgi are lodaded by your configuration:

```
server.modules          = (
                        #"mod_rewrite",
                        #"mod_redirect",
                        "mod_status",
                        "mod_alias",
                        "mod_fastcgi",
                        "mod_access",
                        "mod_accesslog"  )
```

Then, bind the server on a defined IP and port like shown in the example below. This is useful for setting the reverse proxy later.

```
server.port                     = 8080
server.bind                     = "192.168.0.133"
```

Now, define the document root and any alias needed. The alias links a special document root to the contents of a separate directory. Depending on which scenario described in the section above was chosen, one directory will be aliased, but the end result is the same for the web server:

```
server.document-root            = "/srv/www/wiki/htdocs"

# Scenario 1 – Separate static directory
alias.url = ( "/moin_static160" => "/srv/www/wiki/moin_static160" )

# Scenario 2 – Shared static directory (suggested by the documentation):
alias.url = ( "/moin_static160" => "/usr/local/share/moin/htdocs/" )
```

Lastly, create a FastCGI server instance in the configuration:

```
fastcgi.server  = (  "/wiki" =>
                        (( "docroot"   => "/",
                        "min-procs" => 2,
                        "max-procs" => 12,
                        "bin-path"  => "/srv/www/wiki/moin.fcg",
                        "host"      => "127.0.0.1",
                        "port"      => 2200,
                        "check-local" => "disable",
                        "broken-scriptfilename" => "enable"
                        ))
        )
```

This way, Lighttpd will spawn multiple MoinMoin processes by itself, assigning a port for each process starting from the specified one. Note that `min-procs` is deprecated, yet it is used in most configurations, including this one. Depending on the expected traffic, a higher value may be set for `max-procs`.

The web server itself is now ready to run! Start it and, using any browser, access the wiki page, in this case http://192.168.0.133:8080/wiki/ . If everything was done correctly, it should work without problems.

CACHING STATIC CONTENT

Configuring the setup could end up here, but it is possible to cache some static content from the wiki using a reverse proxy and save resources. This is optional, as it brings a problem: the web server will see all requests coming from a unique IP. Varnish, a high performance, caching reverse proxy, was chosen for this setup.

Varnish is very simple and lightweight. It can be downloaded from it's project official page (http://varnish.projects.linpro.no/). Extract and compile it. Compiling it is a very straightforward process if GNU autotools are present:

```
[fang@fang Download]$ tar zxf varnish-1.1.1.tar.gz
[fang@fang Download]$ cd varnish-1.1.1/
[fang@fang Download]$ ./autogen.sh
[fang@fang Download]$ ./configure --disable-debugging-symbols --enable-
dependency-tracking
[fang@fang Download]$ make
[fang@fang Download]$ su -c "make install"
```

Again, a `init.d` script must be created by hand. This time, the configuration script is not the only detail needed on the script: set up a working directory for Varnish with the `-n dir` option. For this paper and setup, it was set as `/var/varnish` .

Configuring Varnish is simple, but almost every action can be edited in the configuration. A nice base configuration can be found at iPerSec (http://tinyurl.com/2l8axc – original link is big), and some concept about caching static content can be found at Varnish's VCL Examples (`http://tinyurl.com/2sk67m`). Independent of general configuration, set up the backend according to the web server address:

```
backend default {
        set backend.host = "192.168.0.133";
        set backend.port = "8080";
        }
```

And Varnish is ready to run! It listens by default at port 80. Simply open a browser and try accessing the wiki via it: `http://192.168.0.133/wiki/` in this case.

CONCLUSION

MoinMoin is up and running by following available documentation, so it does make sense in general. No major problems were found, showing the success and quality of documentation even on a not-so-stable system.

However, two points in documentation were a bit confusing. When using `createinstance.sh`, some problems were found due to the nonexistence of an underlay directory, as shown during the paper. Some search was done and it was found as a tarball, effectively solving the problem. So it was mostly an error from the writer of this paper...

The script itself does the job on setting up the wiki instance, but wont help any further on setting up the structure for the web server. Anyway, by following the permissions set up by it on the rest of the necessary structure, any sysadmin can guarantee the security of the server.

Documentation about FastCGI apparently made no sense in a first look as many ways of doing the same thing are shown sequentially. There is too much information, but it was a question of focusing on one and following it with care. Informations are actually right and the server configuration worked.

As time allows it, following the results and conclusions of this paper to install MoinMoin on a more suitable system, with stability and updated packages in mind, is planned by the author of this paper.