Development Concepts

From Matchi Wiki

Contents

Development

Coding Standards

- All indentation is 2 spaces
- Replace TAB with 2 SPACES on saving, since TAB (\t) characters in code are not desired
- Strip trailing spaces of lines on Save
- Logic bracing is the compact, '1TBS' One True Brace Style. See https://en.wikipedia.org/wiki/Indent_style
- Keep lines short so that logic is clear and changes between versions can easily be viewed
- Keep code compact and avoid pointless spaces
- Class names and function names are in Camel-Case.
- Defines and other immutables are in upper case
- Filenames, code and variables in lower case. Concatenate terms with underscores.
- Indicate global variables with a '$g_' prefix and arrays with a '$a_' prefix.
- Use Functions for code that can realistically be functionalized or needs to be reused in other contexts
- Use Classes and Object-Orientation where design complexity mandates it
- Groups of common functions are static public methods that are encapsulated in abstract classes

Development IDE

An Integrated Development Environment (IDE) that:

- Does language syntax highlighting
- Dynamically identifies basic coding errors such missing parenthesis, closing IFs, closing HTML tags, etc..

- Allows step through the code in a debugger, view and set variables, set breakpoints, etc...
- Readily integrates with Subversion or GIT
- Assists with a sensible indentation schema
- Supports PHP, Perl, BASH and SQL

Many IDEs are available. The currently-preferred IDE that covers all technologies is KOMODO from http://ActiveState.com. You can download a 21-day trial. Or use Zend, Eclipse, PHPStorm, NetBeans,

Remember to set the editor configuration to:

- TAB=2 SPACES
- Replace TAB with SPACES
- Strip trailing spaces on Save

This essential if you want the process of tracking code changes to remain as simple as possible!

Setting up your local development environment

Do a checkout from the Subversion repository to your laptop to develop on your code on from http://mapp01/svn/matchi/trunk/php/com_innovation.

Download the latest production backup from here: mapp01:/data/backups/application_YYYYMMDD.matchi.biz_mweb1_YYYYMMDD_backup.tar.bz2 (use the uncapped ADSL line as it is a 1.3GB file)

Point your database connection in your dev tool to the latest development database, which would normally reside on server mdb0 and is called database dev_YYYYMMDD. Use an SSH tunnel to connect to it, since the database service will not allow direct connections from IP addresses other than the PRODUCTION server (Login credentials: madman/XXXXXXX, or set up a P/P Key-pair).

Coding, Unit testing and Releasing

Code and Unit test on your local IDE.

Check your changes back into Subversion _after unit testing_ is successfully completed.

Deploy from Subversion to the test environment on the server for integration testing - http://test.matchi.biz

From there, any further changes made on TEST as a result of the testing are committed into Subversion.

When testing is completed, a deployment out of Subversion to Production can be done as described further below.

Notes on Subversion:

- Do not treat Subversion as a backup tool of your local development environment. You are responsible for your own backups on your BYOD.
- Only check code into Subversion that has a reasonable chance of working, i.e. code that has been unit-tested on your own development environment.
- Do not assume that you are the only person who makes code changes.
- Do an update from Subversion for smaller / incremental fixes.
- Do a new checkout from Subversion into a new development sandbox before starting a new major tranche of work.

Deploy a new component release

Scope

A new component release entails the upgrading and reconfiguration of the following:

- Component source code
- Database table schema changes
- Database data back-fill and data-alterations
- Graphics (if not already part of the source code deployment)
- Upgrading or installation of new third-party libraries (typically JavaScript libraries)
- System configuration changes

Some release are small and may entail just the deployment of a few files of source code, larger deployments need to be scripted and deployment-tested on test environments before being deployed on the production environment.

The preferred way to script larger deployments is to create a formal Joomla Installation package that installs through the Joomla Package Manager.

Preconditions to Deploying a new Release

A new software can not be installed in the production environment unless

- It is performed by an authorized person who is only able to execute the deployment, but to also quickly assess (with the help of testers if required) whether the deployment works, and to be in a position to actually perform a back-out operation
- The deployment steps are clear, and the steps have been tested in the case or a complex deployment
- There is an adequate back-out plan that can quickly be invoked to restore the system state if required
- All functional enhancements have been adequately unit-tested and were code-reviewed
- End-to-end tests where performed in a test environment

Deploying a new Third-Party Library

TODO

Deploying Database Changes

TODO

Deploying Source Code

Select the code release from Subversion

Decide which release out of the Subversion will be deployed

You can list the history of checkings, with the most recent commits at the top, with this command in an up-to-date sandbox

```
$ cd ~/svn/php/com_innovation
$ svn update
At revision 3020.
$ svn log | head -10
------------------------------------------------------------------------
r3020 | gerrit | 2015-11-28 15:56:24 +0000 (Sat, 28 Nov 2015) | 1 line
```

```
Recommendation Icon
------------------------------------------------------------------------
r3019 | madman | 2015-11-28 14:57:14 +0000 (Sat, 28 Nov 2015) | 1 line

Release 20151128 TEC-971
------------------------------------------------------------------------
r3018 | madman | 2015-11-28 14:55:14 +0000 (Sat, 28 Nov 2015) | 1 line
etc...
```

In this case, we decide that we want to deploy Release 3020, knowing that release has passed the testing phase.

Checking out the code from Subversion to the production area

The new code tree is checked out into a new directory adjacent to the current release, which when successfully checked out and ownership has been set, is symbolically linked to the diretory name that the system expects to find. Note the we are checking out code release 3020, and indicate that release number in the name of the directory to be created.

```
$ cd /var/www/vhosts/s16972616.onlinehome-server.info/matchi.biz/components
$ sudo svn co http://mapp01/svn/matchi/trunk/php/com_innovation com_innovation.3020 -r 3020
```

The files and directory `com_innovation.[new release number]` are still owned by user 'root' and the ownership needs to be set to what the Apache server expects, in this case it is '`plesk:psaserv`':

```
$ sudo chown -R plesk:psaserv com_innovation.3020
[sudo] password for madman:
```

Symbolically Linking to the new code directory

You should be able to see the current symbolic link pointing to the currenly-running release:

```
$ ls -al com_innovation
lrwxrwxrwx 1 plesk psaserv 23 Nov 22 14:36 com_innovation -> com_innovation.[active release number]
```

Break that symbolic link and establish the link to new directory. Also don't forget to set the correct ownership to the new-established link.

```
$ sudo rm com_innovation
$ sudo ln -s com_innovation.[active release number] com_innovation
$ sudo chown -R plesk:psaserv com_innovation
```

Do this in quick succession, script it, or do it in one line:

```
$ sudo rm com_innovation; sudo ln -s com_innovation.[active release number] com_innovation; sudo chown -R plesk:psaserv co
```

Functional Testing

The results of planned post-deployment tests determine the outcome to the "GO/NO GO" question. The planned post-deployment tests should indicate which test failures are severe enough to warrant a backing out. Other factors may also warrant a back-out, such as too many small failures, which usually indicate that biugger but as yet unseen problems are afoot.

TODO

Backing Out

A Backing-out process should normally be a series of simple steps, consisting of a selection of the following depending on the complexity of the deployment:

- Restore the symbolic link to that previous release
- Repoint the application's database to the previous database snapshot
- Restore the symbolic link to thrid-party libraries to the previous release

TODO

Continuous Integration

A continuous testing and integration environment will eventually be set.

Delivery of software into production is done only once at the end of the sprint. A sprint is normally 2 weeks long, give or take variances incurred due to holidays and other business pressures. A dedicated regression environment is created for each sprint, which will remain for 12 months before it is reused. 26 regression environments are prepared and are alphabetically named:

- archie
- beavis
- cartman
- daffy
- elmer
- flintstone
- garfield
- homer
- ironman
- jughead
- kenny
- linus
- mickey
- nemo
- olive
- popeye
- quasimodo
- richie
- snoopy
- tweety
- underdog
- veronica
- wendy
- xmen
- yogi
- zorro

Retrieved from "http://wiki.matchi.info/index.php?title=Development_Concepts&oldid=1457"

Category: Pages with syntax highlighting errors