Proposal

Deploying the Earthquake Hazards Program website using Subversion

Problem Statement

The Earthquake Hazards Program (EHP) website is currently developed following a very ad-hoc process. Developers often *develop and test* on the master web server, or worse, on the production servers themselves. Versioning and backups used during the development process often include the creation of "*.bak" files or a Tape ARchive (TAR) file of the directories to be changed. These files are generally forgotten about and this leaves the EHP document root directory tree littered with old files and confusion about why all these "*.bak" files exist and if they are still needed. As the EHP website evolves from a grassroots effort into a structured project involving additional developers the current development process is no longer feasible.

Prior to this writing the EHP development team has established a web-based bug tracking/feature request system using Trac. Additionally, the EHP System Administrator (SA) has also already set up a subversion (SVN) repository. All that remains is to integrate these tools into a robust development process.

Solution

(1) Content on the EHP website falls into four basic categories: general web content, web application source code, cached content, and web application data. Each of these types of content should be physically separated on the servers with the former two types of content under source-code control in the SVN repository.

(2) Trac must become the entry point for all development requests. This ranges from tasks as small as, "...can you fix a typo on this page...", to tasks as large as, "...can you develop an application to do X...".

(3) Initial development of any request is done on the development web server which has a working copy of the repository trunk. After developing and testing in the development environment, the change can be deployed. Deployment depends on the type of content being deployed (see: *Implementation Details*).

(4) At regular intervals, changes made to the repository trunk are pushed "live" to the public. General web content as well as individual web applications may be deployed independently of each other.

Justification

(1) The EHP web naturally fits these four distinct categories. Cached content is essentially a static HTML version of the PHP equivalent web page. We cache content because serving static HTML is much faster and lightweight for our servers. This is very important when we experience a flash crowd following a significant earthquake. Because we want our web content version controlled for integrity, we would like to place the PHP pages into SVN, however there is no need to place the cached copies in SVN. In order to ease the versioning of PHP and non-versioning of HTML, these two types of content need be physically separated on the servers. We also must consider the relationship between web application source code and its relative content. There is a clear need for versioning the source code, however the real-time nature of the application data necessarily prohibits the use of SVN for its versioning. Again, to accommodate this inherit difference, these two types of content are physically separated on the web servers.

(2) While it may seem trivial to use the Trac ticketing system for minor tasks, it is important to do so for the sake of historical reference and accountability. As we develop new content it is important to know who developed the content and why they did so (this applies equally to changes in existing content). If something is broke on a page, who do we contact to get it fixed? Using Trac we can quickly see who requested the content be developed, who developed it, and quickly resolve the problem.

(3) Using a development server gives the development team an isolated environment to test new content and features before deploying them to the web. It also aids in the use of version controlling our content by using a well-designed deployment process. (See: *Implementation Details*) The isolated environment and version control add to the integrity and security of the EHP website as a whole.

(4) Regular releases of our content are important in order to ensure the web site does not become "stale" for visitors. However releasing new versions of large web applications at the same pace as general web content is not feasible. Therefore it is important to be able to release web content independently of a web application. Additionally, each web application is independent of each other and should also be able to be released independently of each other. The release cycles of general web content will happen at regular intervals, and the release of each web application is dependent upon the development team for that application.

Implementation Details

<u>Phase 1: Physically Split Content on Server</u>

First we create the directory hierarchy for the content (see below). Next, we will delete any locally cached versions of the HTML content. Then the general web content will be placed in the vhosts/*hostname*/htdocs directory and newly cached versions of the PHP pages will be created in the vhosts/*hostname*/cache directory. Finally each web application will be placed into its own apps/*appname* directory.

Once all the directories have been created and populated, Apache will need to be reconfigured to work with the new structure. Part of this reconfiguration will be to separate the httpd.conf into application configurations and vhost configurations.

apps	
appname	Application instance name
conf	
httpd.conf	Application apache configuration (optional)
htdocs	Application front-end source code (optional)
lib	Application back-end source code
data	Application data. Possibly shared between multiple instances of a single application, or possibly different applications.
vhosts	
hostname	Domain name
cache	Cached content
conf	
httpd.conf	Virtual host apache configuration
htdocs	Virtual host general web content

This phase will first be tested and completed on the development server. After thorough testing these changes are pushed to the public server. Once the change is pushed to the public server this phase is complete and will not be rolled back. For this reason it is imperative to perform thorough testing before pushing to the public server.

Phase 2: Import Content into Subversion Repository

The EHP project will have a single SVN repository and set up multiple sub-projects within it.

ehp	
www	Web content
hostname	
branches	
tags	
trunk	
conf	
httpd.conf	
htdocs	
apps	
appname	Application name
branches	
tags	
trunk	
conf	
httpd.conf	
htdocs	
lib	

The repository structure reflects the physical content structure in as much as possible. Each *hostname* and *appname* becomes its own self-contained project. They may be tagged and released independently of each other. Once the repository is set up, content will be imported from the web servers. Content from the public web server will be imported into the trunk and content from the development server will be imported into a development branch.

Phase 3: Deploy Website

Immediately following the initial checkin, the *www* trunk will be tagged as stable (since its content is currently being served on the web). After this we will merge relevant changes from the development branch that are considered ready for production into the trunk and tag this new trunk as release candidate 0.0.1a. This release candidate will be deployed into a quality assurance testing server. During testing we may iteratively tag additional development branch data into the subsequent release candidates (b, c, d, ...) until we are satisfied with the release. At this point the release will be tagged for released and deployed on the public web server as version 0.0.1. Finally, the remaining code in the development branch will be merged with the trunk on a case-by-case basis.

Release Cycles

After initial deployment of the EHP web site through SVN the EHP development team will continue to make periodic releases of each project in the repository. When a project is ready for a release it will follow the steps below:

I. Tag the current project trunk as a release candidate.
   A. Feature development in this tag is stopped.
   B. Release notes are compiled from each project developer, the Trac ticketing system, and the SVN commit log history.
   C. Committing source code to this tag may only be done after consulting project development team and release manager.
      1. Changes may only be done for bug fixes found in testing (see below).
      2. Widespread approval is needed to ensure new bugs are not introduced.
      3. Any changes must be added to the release notes.

II. Tag is deployed into the quality assurance testing web server.
   A. A working copy of the tag is checked out into the project's directory on the testing web server.
   B. Developers test each change and any possible side-effects of the change.
      1. Unit testing suite may be run if applicable.
      2. Bugs are reported through the Trac ticketing system.

3. Changes are made on the test server's working copy, committed to both the project tag and trunk.

C. The working copy is exported and effectively removed from version control.
   1. This deletes the ".svn" files used exclusively by SVN that we do not want on the web site.
   2. Changes can **NOT** be made to the release candidate at this point.

III. Release is made to the master web server.
   A. Exported content is compressed and transferred to the master web server.
   B. Compressed file is extracted into web server project directory.
   C. Apache server *may* need to be restarted if the project introduced new configuration file.

IV. Release is made to the public web servers.
   A. rsync_ehp *projectdir*
   B. Each public web server *may* need to be restarted if project introduced new configuration file.

Critical and Time-Dependent Modifications

Due to the real-time/time-dependent nature of much of the EHP website content, some fixes cannot wait for the full deployment process as defined above. In this case, the fix *may* be made directly on the master web server and pushed to the public servers. Because the master web server's content is not under version control, this fix must be manually replicated into the project trunk and any current project release candidate tags.

Changes should only be made in this way when the change is of a critical or time-dependent nature, the risk of making the untested change greatly outweighs the benefit, and the change has a high likelihood of fixing the problem. While these changes may be initially reported via email/instant message to expedite the process, the change should also then be reported into the Trac ticketing system for historical reasons.