Bundler v0.5 Documentation

Prepared by the West Quad Computing Group
October, 2008

1 Overview

In the past, all development and computational activities took place on the (former) Roth lab cluster head-node, llama.med.harvard.edu (or on the cluster compute nodes). In addition, this machine also hosted the web site http://llama.med.harvard.edu. While this was simple from an architectural standpoint, it also meant that any disruption of llama due to a runaway process, for example, also disrupted the web site. Since several of the applications on http://llama.med.harvard.edu are highly visible to the outside world, this situation was deemed unacceptable.

It was decided to purchase a separate server on which to run the web site. This server would be isolated from the day-to-day computational work of the head node, and, also the head node, would not mount any directories from the file server dogma. Thus, even if the file-server goes down, the web site will continue to run. Since the URL http://llama.med.harvard.edu has been published far and wide, it was decided to keep this name for the web host and decide on a new name for the cluster head node, which is now called dalai. Actually, the naming situation has become a bit more complicated. In order to make it clear when one is speaking of a web site, versus the server itself, further names were introduced. Thus, now the hostname of the server running the production web site is called mandala.med.harvard.edu, while the production web site itself is referred to as llama.med.harvard.edu. Likewise, the hostname of the head node is dalai.med.harvard.edu, and there are two web sites, dev.shinay.med.harvard.edu and shinay.med.harvard.edu, running on dalai. The purpose of these two web sites is detailed below.

With the introduction of two separate hosts, comes added complexity. The host dalai is designated for development work, while mandala is designated to (primarily) provide web services. Thus, it is desirable that web applications will be developed and tested on dalai, and moved to mandala when mature enough. Due to the added complexity of maintaining the multiple websites necessary to support this model, Bundler was introduced.

Bundler is a system designed to ease web site management. With Bundler in place, the develop-test-publish cycle becomes more efficient for the developer. The building block upon which Bundler operates, is (as the name implies) the *bundle*, and once a bundle has been set up, a developer can for the most part ignore the details of setup of the websites and instead focus on the development.

This document provides an overview of the Roth lab web site structure, and then describes the architecture and use of the Bundler system. In section 2 we present an overview of the Roth lab web site organization. Section 3 details the structure of a bundle, the Bundler scripts and their usage, including a typical workflow. Section 4 is a reference for the Bundler scripts.

2 Roth Lab Web Site Organization

There are two conceptual components to the Roth lab web infrastructure: physical and logical. The physical component consists of the two servers with hostnames dalai.med.harvard.edu and mandala.med.harvard.edu. The logical component of the web infrastructure consists of the organization (namespace) of the three web sites: http://dev.shinay.med.harvard.edu, http://shinay.med.harvard.edu and http://llama.med.harvard.edu. The directory organization of the files for the web sites, the configurations for the web servers and databases, and so on, involves both the physical and logical components.

We consider first the physical setup. The server dalai.med.harvard.edu, which we refer to as the *development* host, is for developing and testing of code (it is also the cluster head node, and is otherwise used for many activities unrelated to web development, but we only focus here on web-related aspects of the systems). Dalai also runs an Apache web server and PostgreSQL database server, and mounts directories from the fileserver, dogma, including user home directories (where users' personal web site files are located). The server mandala.med.harvard.edu, which we refer to as the *production* web host, also runs Apache and PostgreSQL, but is not (generally) used as a login host. Mandala also differs from dalai, in that no directories from dogma are mounted, and that port 80/tcp (the http port) is open through the HMS firewall, and so is accessible from outside of HMS. A primary reason for having a separate production host is to isolate important, high-visibility applications from development activities and day-to-day work, as well as to distribute the computational load.

The logical setup is closely tied to the physical setup, but there are some important distinctions.

The development site (http://dev.shinay.med.harvard.edu), hosted on dalai, is the web site on which to rapid prototyping and unit testing is done, and is subject to the most disruption. This is the place to explore new ideas without fear of affecting critical code.

The (integration) testing site (http://shinay.med.harvard.edu), also hosted on dalai, is for integration testing of code that is destined for the production web site. Unlike the development web site, this web site mirrors the public web site, so that if code runs here, it should (hopefully) run on the production web site. This is generally the last stop for code before being pushed to the production server, to ensure it will run properly in the production environment.

The public web site (http://llama.med.harvard.edu) is hosted on mandala. Only code that is reasonably well tested should be put here, as this is the site that the outside world sees. Mature code that is likely to receive high usage and/or has been referenced in publications should (likely) go here.

Conceptually, there is a “fourth” web site. This web site exists for applications, one-off projects, code to be shared with collaborators and immature code for publications still under review, for example. Such applications are accessible through the http://llama.med.harvard.edu namespace, but will be physically hosted on dalai. The purpose is to present a unified URL space to the outside world, while still being able to control where applications actually run. The namespace mapping is achieved via the mod_proxy module of Apache. See figure 1. Related to this, all personal web sites (those of the form http://llama.med.harvard.edu/~username) reside on dalai, in the users' home directories, and are proxied from mandala. Thus, users do not need to push their personal web site files to mandala. Note that hosting a new, publicly accessible web site from dalai (other than a personal web site) requires a configuration change on mandala, and so a support request must be submitted.

3 Bundler Overview

In order to manage this somewhat complicated collection of web sites, we have developed *Bundler*, a collection of Python scripts intended to make the deployment of code between the three web sites relatively straight-forward. Before we discuss the details of Bundler, we present the requirements that are to be addressed by such a system. The system should, as much as is reasonable, hide the physical component of the web site infrastructure, so that the developer can focus instead on the logical component. This includes offering the developer control over the installation path of files, on a project-by-project basis, but in a way that, once configured, should “just work.” The system should also provide a straight-forward mechanism whereby a web project can be moved between development, test

and public web sites with a minimum of effort by the developer. Finally, it should provide a method to manage multiple code revisions, including providing the ability to roll back changes, should that be necessary.

To consider why a scripted solution is useful, consider some possible alternatives. One could copy the relevant files for one's project, by hand, as needed, between web site directories. The advantage of that technique is that it gives the developer fine-grained control of file placement. However, this technique is tedious, and is prone to error. Additionally, should a bug be discovered after the new files are installed, it can be hard to determine which files are affected and need to be rolled back, since everything would have to be tracked manually. Another approach would be to use Makefiles for each web project. Certainly Makefiles have been around for a long time, and are still the most common way under Unix and Unix-like operating systems, such as Linux, to compile and install code that is distributed in source form. On the other hand, Makefile syntax is somewhat of an arcane art, and does take some time to master. Additionally, both the manual copying and Makefile-based techniques do nothing to address the version control requirement.

To address some of these shortcomings, Bundler was developed to work in conjunction with a version control system (VCS). The scripts will allow one to install files into the various web sites using relatively simple commands. The use of a VCS ensures that if a rollback becomes necessary, it is easily accomplished. We have chosen to use the VCS Subversion, in part due to its ability to store both text and binary documents.

In designing Bundler, an abstraction was introduced to ease the maintenance of the web sites. We define a *bundle* to be simply a collection of files that together comprise some logical component of the web site. The actual distinction of which files belong in which bundle is entirely arbitrary. For example, a bundle called ‘SoftwareProject’ might contain all of the perl scripts, html files and configuration files needed to run SoftwareProject. SoftwareProject could also depend on files provided by another bundle, such as a library bundle, for example. Most of the Bundler scripts take a bundle name as an argument, and so operate on the entire collection of files contained in a bundle. A detailed description of the structure of a bundle is presented below.

All bundles are stored in a Subversion repository. Subversion is a VCS that stores the version history of a file in a database, along with an optional comment that is associated to each version, which can detail the changes made for that version. Thus, it is possible to 'go back in time' (in a conceptually similar way to Mac OS X Leopard's Time Machine) to see the version of a file that existed last week, last month or last year, assuming of course that changes are regularly checked in to the repository! For the complete documentation of Subversion, please refer to the Subversion Red Book at http://svnbook.red-bean.com/en/1.4/svn-book.html. The use of Subversion also ensures that, even if a program installed on mandala causes unexpected problems, it is a simple matter to quickly roll back to a previous version.

3.a Web Site Directories

The development, test and production web sites are comprised of distinct directories which do not overlap. For example, the development web site html files are located under /www/dev.shinay.med.harvard.edu/docroot, and the test web site html files are under /www/shinay.med.harvard.edu/docroot. The point of this organization is of course that changes made to files on one web site shouldn't affect the other web sites. See the appendix for the standard directory locations.

3.b Staging Areas

When a bundle is installed, it actually resides in a *staging area,* which lies outside of the web site directories. There are separate staging areas for the development, test and production web sites. Directories from each web site are then symbolically linked from the actual web site directories mentioned above, to these staging areas. Bundler provides scripts to install bundles into the staging areas and set up the symbolic links to the staging area from the web site directories. Figure 2 gives a brief overview of this setup, and figure 3 illustrates the steps that take place when a bundle is installed.

A staging area provides flexibility when working with the web site files. For example, the location of the web site directories are fixed by the configuration of Apache, but the staging areas can be moved around on the fly, and, in particular, individual bundles can be moved around. This is particularly useful during development. Rather than continually make changes to a bundle, checking your edits in to the Subversion repository, then installing the bundle into the staging area, you can install a bundle into a directory in your home directory. The symbolic links from the web site will then point directly to your version of the bundle, and changes made take immediate effect. See figure 4. Note that this only works for the development web site, since the production webserver is on a separate host where your home directory is not available. However, this does not create a problem, as you should not be doing development work on the production server! When you used EditBundle.py to start editing a bundle, the symbolic links are set up automatically from the development web site (http://dev.shinay.med.harvard.edu), assuming you have configured your bundle correctly.

3.c Bundle Organization

Just about every web site document or script should belong to some bundle. Notable exceptions are run-time created temporary, log files or data files that are modified through access to the web site itself, such as wiki data files. Bundles provide a way to operate on logical chunks of the web site, without having to worry about checking out or installing individual files. Most of the command scripts in Bundler operate on bundles. Once the work has been done to set up a bundle, working with bundles becomes much easier, using the Bundler scripts.

A bundle is simply a directory, which we refer to as the *root directory* of the bundle, that contains all of the files and directories associated with that logical unit of the web site. The name of the root directory is the name of the bundle itself. For example, the Publications bundle contains all the files contained in the docroot/publications and docroot/supplements directories. The bundle is stored in a Subversion repository, and each bundle is a separate Subversion project. A Subversion project has a specific structure by convention. Below the project root directory, it contains a branches, tags and trunk

directory, and the trunk directory is where the working copy files are located. The branches and tags directories have specific uses, but for now we will ignore them. See the Subversion Red Book and the diagram below for details. When you check out a bundle using the EditBundle.py script, the project is actually checked out from its trunk directory. Thus, if you typed 'EditBundle.py MyBundle', your working directory would contain a directory called MyBundle, and MyBundle would contain the contents of svn://<BUNDLE_REPOSITORY>/MyBundle/trunk.

The trunk of each bundle in Subversion is organized as a collection of subdirectories that determine the destination of the files they contain. For example, the bundle SoftwareProject might be organized as such:

```
SoftwareProject |
                |-branches
                |
                |-tags
                |
                |-trunk    -|
                           |
                           |- admin
                           |
                           |- DOCROOT
                           |
                           |- ICONS
                           |
                           |- CGI
```

where for example, DOCROOT contains files destined for the web site document root, as defined by the Apache configuration item DocumentRoot, while ICONS might map to /www/shinay.med.harvard.edu/icons (for the TEST web site), and so on. The translation from bundle paths to install directory paths is done by the install script, InstallBundle.py, on the fly. The names of the directories shown in all caps above are completely arbitrary, and are configurable through the user-editable configuration file 'translate_rules', located in the admin directory, which must exist for each bundle.

When a bundle is installed, it is placed in a staging area, as mentioned above, and symbolic links are created from web directories to each of the upper-case directories under the bundle trunk directory.[1] Considering the example bundle SoftwareProject shown above, the symbolic links created when installing the bundle into the production server might be

```
/www/llama.med.harvard.edu/docroot/SoftwareProject  →
                        /usr/local/rothlab/Staging-PROD/SoftwareProject/DOCROOT
/www/llama.med.harvard.edu/icons/SoftwareProject    →
                        /usr/local/rothlab/Staging-PROD/SoftwareProject/ICONS
/www/llama.med.harvard.edu/cgi/SoftwareProject      →
                        /usr/local/rothlab/Staging-PROD/SoftwareProject/CGI
```

so that, for example, when a user enters http://llama.med.harvard.edu/SoftwareProject/index.html in their browser, the file actually served is /usr/local/rothlab/Staging-PROD/SoftwareProject/index.html. The rules to control the creation of the symbolic links are contained in the file translate_rules, located in the admin directory, which is located directly under the bundle root directory.

The “admin” Directory

The admin directory contains the file “translate_rules” which provides a map between the bundle directories and the web site directories. When a new bundle is created, using either EditBundle.py or NewBundle.py, there are by default two additional files called post_staging_setup.template and

1 In fact, the directories under trunk do not have to be capitalized – this is merely a convention.

post_install.template placed in the admin directory. These are, as their names imply, template scripts that can be used as a guide to write a script which is run during the installation of a bundle. The former is run just after the bundle has been updated in the staging area, and the second is run just after the symbolic links between the web site directories and the bundle are set up. Both translate_rules and the scripts are described in more detail below.

translate_rules

The file translate_rules must exist in order for a bundle to be successfully installed. When a bundle is first created (for example, with EditBundle.py or NewBundle.py), a default translate_rules file is created, but with its entries commented out. The format of the translate_rules file is as follows:

```
[LABEL]
SVN_DIR:<SOURCE_PATH>[+]
[SVN_DIR2:<SOURCE_PATH2>[+]]
```

where `LABEL` is one of DEV, TEST or PROD, for the development, test and production web sites, respectively, `SVN_DIR` is the name of a directory under bundle trunk directory (the full path of this directory will be the target for the symlink) and `SOURCE_PATH` is the symbolic link source (that is, the directory where the symbolic link is created). The optional '+' at the end of the source path controls how the symbolic link is created. By default, if a source path is specified without a '+' at the end, then a symbolic link with the name of the bundle is created in the directory SOURCE_PATH which points to SVN_DIR. If, on the other hand, a '+' is included, then the last element in SOURCE_PATH is used as the name of the symbolic link. Some examples will make this more clear. (For the examples below, assume the name of the bundle in question is called 'MyBundle' and the staging areas are all located under /usr/local/rothlab/).

Example 1:

```
[DEV]
CGI:/www/dev.shinay.med.harvard.edu/cgi
```

Result: The symbolic link /www/dev.shinay.med.harvard.edu/cgi/MyBundle is created, and points to /usr/local/rothlab/Staging-DEV/MyBundle/CGI/

Example 2:

```
[PROD]
DOCROOT/myfile.html:/www/llama.med.harvard.edu/docroot/myfile.html+
DOCROOT:/www/llama.med.harvard.edu/docroot/MyDir/+
```

Result: Two symbolic links are created in /www/llama.med.harvard.edu/docroot, myfile.html, which points to the file /usr/local/rothlab/Staging-PROD/MyBundle/DOCROOT/myfile.html, and MyDir, which points to the directory /usr/local/rothlab/Staging-PROD/MyBundle/DOCROOT.

Since the directories /www/dev.shinay.med.harvard.edu, /www/shinay.med.harvard.edu and /www/llama.med.harvard.edu are used so often as the root of the source directories, there is a shortcut

that can be used for them: $WWW.

Example 3:

```
[DEV]
DOCROOT:$WWW/docroot
CGI:$WWW/cgi

[PROD]
DOCROOT:$WWW/docroot
CGI:$WWW/cgi
```

Result: In this case, note that both the `[DEV]` and `[PROD]` blocks are exactly the same. However, in the first case, symbolic links are created in the directories /www/dev.shinay.med.harvard.edu/docroot and /www/dev.shinay.med.harvard.edu/cgi, while in the second, the directories are /www/llama.med.harvard.edu/docroot and /www/llama.med.harvard.edu/cgi. The directory that is substituted for `$WWW` in each block of the DEV, TEST and PROD blocks is specified in Configuration.py.

The ***post_staging_setup*** *and* ***post_install*** *scripts*

It is sometimes necessary or desirable to execute a script during or immediately after installing a bundle. For example, your bundle may expect certain directories to exist, or a server process may need to be started, and these tasks can be accomplished from within a script. Template scripts are included in the bundle admin directory when a bundle is created. They are called post_staging_setup.template, and post_install.template. To use a script, copy it to a file with the same name, but without the .template suffix, edit it, then add it to the Subversion directory and check it in. Additionally, you must ensure that the 'svn:executable' property is set on the file. This ensures that the script is marked as executable when it is checked out of the Subversion repository.

For example, to use the post_install script:

```
cd MyBundle/admin/
cp post_install.template post_install        Copy the template file
vi post_install                               Make your edits
svn add post_install                          Put the file under version control
svn propset svn:exectable '*' post_install    Set the file as being executable
svn ci post_install                           Check the file in to Subversion
```

Both scripts are called with arguments. The post_staging_setup script is called with two arguments: the target destination (either DEV or TEST or PROD) and the full path of the bundle in the staging area. The post_install script is called with the three arguments, the first two arguments are identical to the arguments of post_staging_setup, and the final argument is the full path to a plain text file containing the symbolic links created during the install, one per line.

3.d Initial Setup

Before you can begin to use Bundler, there are a few housekeeping chores that need to be done.

1. Obtain a Subversion account and obtain membership in appropriate Unix groups.
   If you do not already have a Subversion account on dalai, you will need to request one to be set up. In addition, you will need to be a member of 'www' and 'wwwrun' groups. To check if you have this group membership, type 'groups' and check for them in the output. If you need a Subversion account and/or need to be added to the www and wwwrun groups, submit a support request to WQCG (http://wqcg.med.harvard.edu/support). If you are having a Subversion account set up, make sure you obtain the password, as you will need it in step two.
2. Once step one is complete, you should be able to run the script ListBundle.py on dalai (it should be in your path). If your Subversion account was just set up, you will be prompted for a password. Type in the password you received in step one. After you have authenticated once, your credentials should be cached and you will not be prompted again.
3. If your Subversion account was set up in step one, you will also need to run ListBundle.py on mandala. Ssh to mandala and run ListBundle.py again. Enter your password from step one when prompted.

At this point, you should have full access to the Bundler system!

3.e Script Overview

Here is a list of the scripts that can be used to maintain the web site, and a brief description of their function. (They are documented more fully below).

BundlerMeta.py	Prints out the configuration options defined in Configuration.py.
Configuration.py	Contains the default configurations for the Bundler installation, some of which can be overridden through command-line switches, and some of which are only configurable by your system administrator.
SetupStaging.py	Used for the initial setup of the staging area(s). This usually only needs to be run once, unless the staging area is redefined in Configuration.py.
UpdateStaging.py	Initializes or updates a bundle in a staging area directory. Is called directly from InstallBundle.py, so does not usually need to be called directly.
EditBundle.py	A helper script which can be used to check out a bundle that is already in SVN, or to create a new bundle and check it out in one step. Also sets up the symbolic links between the DEV site and the checked-out bundle, so that changes may be immediately tested.
ListBundle.py	A helper script to list the files contained in bundle's trunk directory in SVN. If a bundle name is provided as an argument, it will instead show a list of files in the bundle.
NewBundle.py	A helper script to set up a Bundle skeleton structure in the SVN repository, without checking it out.
InstallBundle.py	Install a bundle (that has been checked in to the Subversion repository) by checking the bundle out to the appropriate staging area, and creating the symbolic links between the web site directories and the staging area.
UninstallBundle.py	As the name implies, does the opposite of InstallBundle.py: removes the symbolic links for the bundle and removes the bundle from the staging area. Note, the bundle will still exist in the Subversion repository.

3.f Typical workflow

A typical workflow for a bundle might be as follows:

1. Check out bundle from the Subversion repository and edit.

   cd ~/MyProjects
   EditBundle.py MyBundle[2]
   cd MyBundle
   <Edit Away>

   Directory ~/MyProjects/MyBundle is created, and files in MyBundle/trunk are checked out to it. If the bundle doesn't exist in SVN, you will be prompted if you want to create a new bundle. Answer yes and a bundle skeleton is created in SVN, and then checked out as above. If the bundle already exists in SVN (and assuming that translate_rules has been set up for the DEV web site), then symbolic links are created automatically from the DEV web site directories to the bundle. Thus, you can make changes and immediately test them by going to http://dev.shinay.med.harvard.edu.

   Files can then be edited, new files added (with 'svn add', unneeded files are deleted with 'svn rm', etc.)

2. Check in appropriate files to SVN.

   svn ci
   (or svn ci <filename>, or svn ci <subdir of MyBundle>) is done. The updates are checked in to SVN in the normal fashion.

3. Install the bundle using the InstallBundle.py script.

   InstallBundle.py TEST MyBundle

   The bundle MyBundle in the TEST staging area is then checked out or updated via UpdateStaging.py, and the resulting files are symbolically linked from the TEST web site install directories.

2 You can also use the Subversion commands directly (svn co), instead of using EditBundle.py, but in that case of course, no symbolic links are created between the web site directories and your checked-out bundle.

4 Script Reference and Usage

Note: You can obtain a brief usage summary of any (runnable) script by passing the option -h (or –help).

Name: BundlerMeta.py

Usage: BundlerMeta.py

Description: Shows the configurable options set in Configuraton.py. Can be particularly useful for the

Options: (None)

Examples: *(None)*

Name: Configuration.py

Usage: Configuraton.py

Description: This module contains the configurable options of the Bundler system. Edit it to set options such as the default location of the Subversion repository and the full paths to the staging areas. If run, will show the current configuration. (BundlerMeta.py is actually just a symbolic link to this script.)

Options: (None)

Examples: (None)

Name: EditBundle.py

Usage: EditBundle.py BUNDLE_NAME

Description: Check out bundle from repository for editing. The bundle is checked out from its trunk directory, and is stored in the bundle working directory BUNDLE_NAME in the current working directory. In addition, assuming the bundle translate_rules have been set for the DEV web site, symbolic links are created from the DEV web site to the bundle working directory, so that changes made to the bundle files can be immediately seen by going to the DEV web site url.

If on the other hand the bundle does not exist in the Subversion repository, the user is prompted if they wish to create a new bundle. If they answer in the affirmative, a bundle skeleton is created in the repository, and the bundle is checked out. Note that by default all entries in translate_rules are commented out, and so no symbolic links are created between the DEV web site and the bundle working directory.

Options: -h, --help show this help message and exit

Examples:

(For an already existing bundle)
cd ~/MyProjectsDir
EditBundle.py SoftwareProject

Results in the creation of ~/MyProjectsDir/SoftwareProject, containing
all files from SVN:/SoftwareProject/trunk/

(For a new bundle)
cd ~/MyProjectsDir
EditBundle.py MyNewBundle
The bundle blah was not found in the subversion repository.
Do you want me to create it? Y

Results in the creation of a skeleton bundle directory called MyNewBundle in the repository.
The skeleton bundle is then checked out to ~/MyProjectsDir/MyNewBundle.

Name: InstallBundle.py

Usage: InstallBundle.py [options] DEV|TEST|PROD BUNDLE_NAME

Description: Installs the files from a bundle (that has been checked in to SVN) to the appropriate install directories, as determined by the first argument. This script actually updates the specified bundle (via 'svn up' or 'svn co') in the staging area specified in Configuration.py. It then 'installs' said files by creating symbolic links from the install directories to the staging area, as specified in the 'translate_rules' file in the bundle's admin directory.

Options:

-h, --help
show this help message and exit
-n, --dry-run
Only show what would be done, don't actually create the symbolic links from the apache directories to the staging area. NOTE: The staging area will still be updated from the repository, so do not check in any changes if you are not ready for them to be propagated to the web site.

Example:

InstallBundle.py DEV MyBundle
Install the bundle MyBundle to the DEV web site.

InstallBundle -n TEST MyBundle
Check out (or update) the bundle MyBundle in the TEST staging

area, then show which symbolic links would be created, but don't actually create the links.

Name: ListBundle.py

Usage: ListBundle.py [BUNDLE_NAME] [[BUNDLE_NAME]...]

Description: Dump a list of bundle names from repository (when no arguments specified). Otherwise, dump list of files from SVN trunk of bundle(s).

Options:

-h, --help
show this help message and exit

Examples:

ListBundle.py
Show a list of bundle names in SVN repository.
ListBundle.py MyBundle
Show list of files contained in bundle MyBundle

Name: NewBundle.py

Usage: NewBundle.py BUNDLE_NAME

Description: Set up a template bundle BUNDLE_NAME in the SVN repository. A bundle has the following structure:

```
BundleName--|
            |-branches
            |
            |-tags
            |
            |-trunk-|
                    |
                    |- admin
                    |
                    |- CGI
                    |
                    |- DOCROOT
```

Note that the skeleton is set up with some optional directories in all caps, but that the directory trunk/admin is required. See the main documentation for more details. Note that the module is only set up in the Subversion repository with this command. You must use the EditBundle.py command to actually check out the bundle to start editing.

Options:

-h, --help

show help message and exit

Examples:

NewBundle.py MyBundle

Creates default structure outlined above for “MyBundle” in the SVN repository.

Name: SetupStaging.py

Usage: SetupStaging.py [options] DEV|TEST|PROD

Description: Set up the development, test or production staging areas.

Options:

-h, --help

show this help message and exit

Example:

SetupStaging.py DEV

Set up the development staging area defined in Configuration.py.

Name: UpdateStaging.py

Usage: UpdateStaging.py [options] DEV|TEST|PROD BUNDLE_NAME

Description: Update bundle BUNDLE_NAME from SVN to development, test or production staging area. Note that this module is called from InstallBundle.py, and so rarely has to be called directly.

Options:

-h, --help Show this help message and exit

-s STAGING_AREA, --staging_area=STAGING_AREA

use STAGING_AREA instead of the default staging area.

Name: UninstallBundle.py

Usage: UninstallBundle.py [options] DEV|TEST|PROD BUNDLE_NAME

Description: Uninstalls the files from a bundle by removing the associated symbolic links and purging the bundle directory from the appropriate staging area. Does not remove the bundle from the repository.

Options:

-h, --help

show this help message and exit

-n, --dry-run

Only show what would be done, don't actually remove anything.

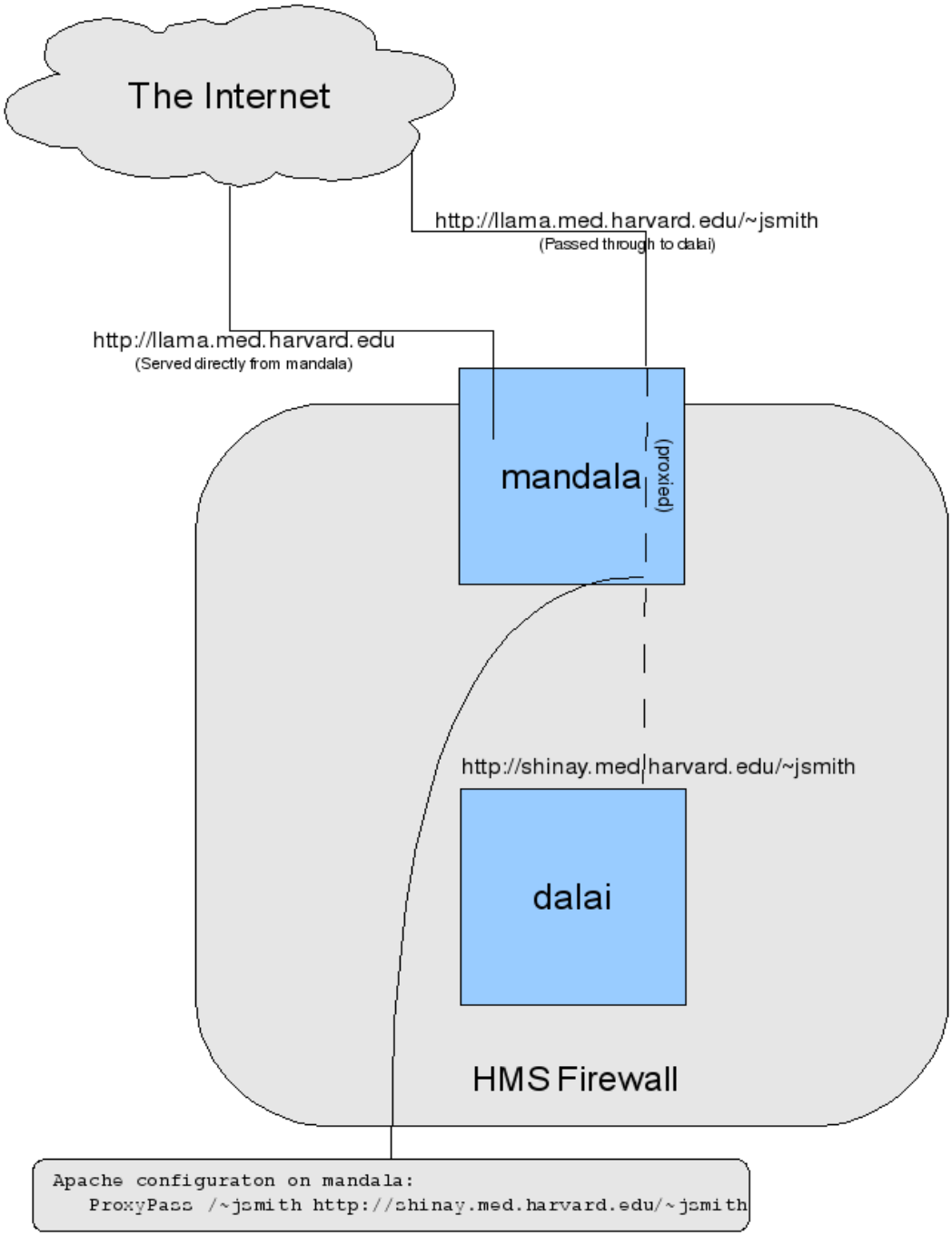
Example:

UninstallBundle.py DEV MyBundle

Remove the bundle MyBundle from the DEV web site.

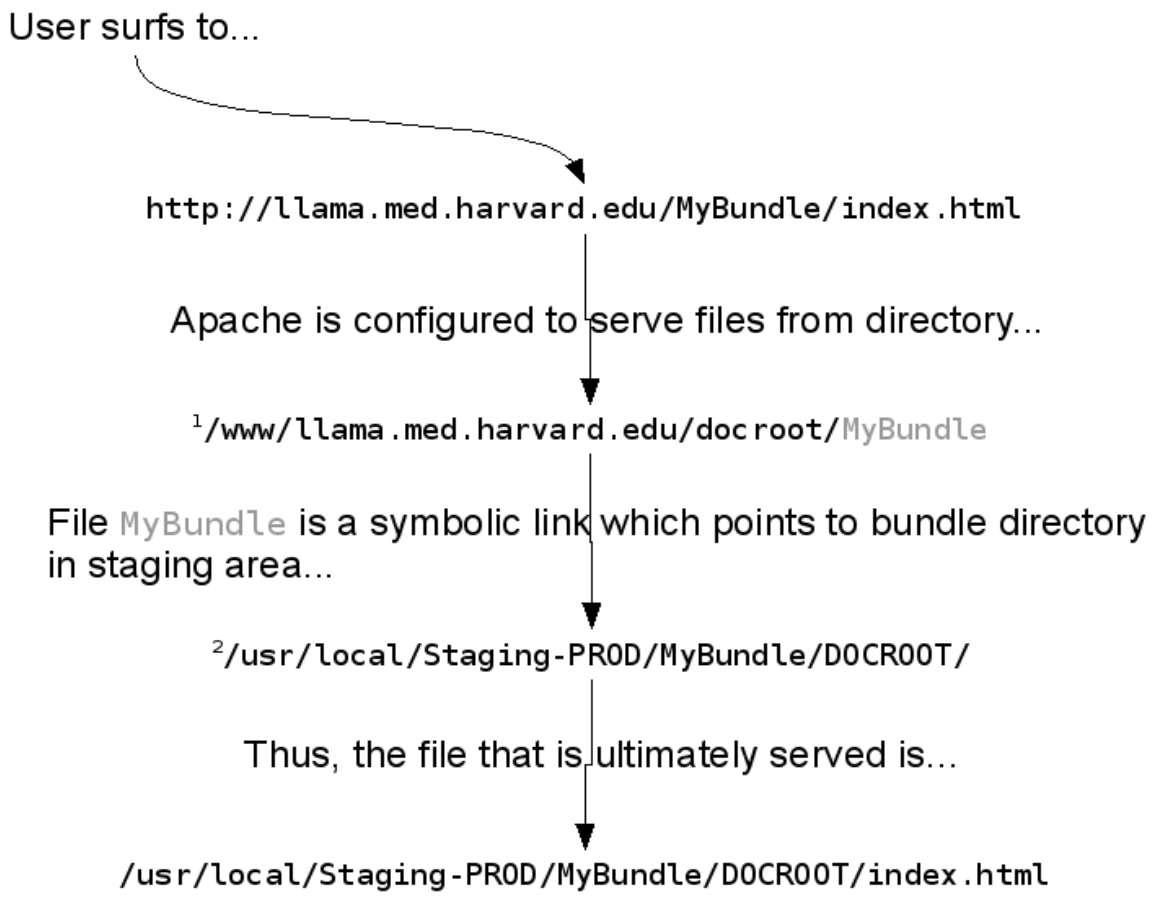
InstallBundle -n TEST MyBundle

Show what would be done in a



Personal web sites live on dalai, but are proxied from mandala, so they are available to the outside world. (Normally, web sites on dalai are not open to outside of HMS, due to the HMS firewall). Other directories can also be proxied in this fashion.

Figure 1



[1]As defined in mandala:/etc/apache2/sites-enabled/000-llama

[2]As specified in Bundler configuration file /usr/local/rothlab/Bundler/Configuration.py and bundle's translate_rules file.

Figure 2

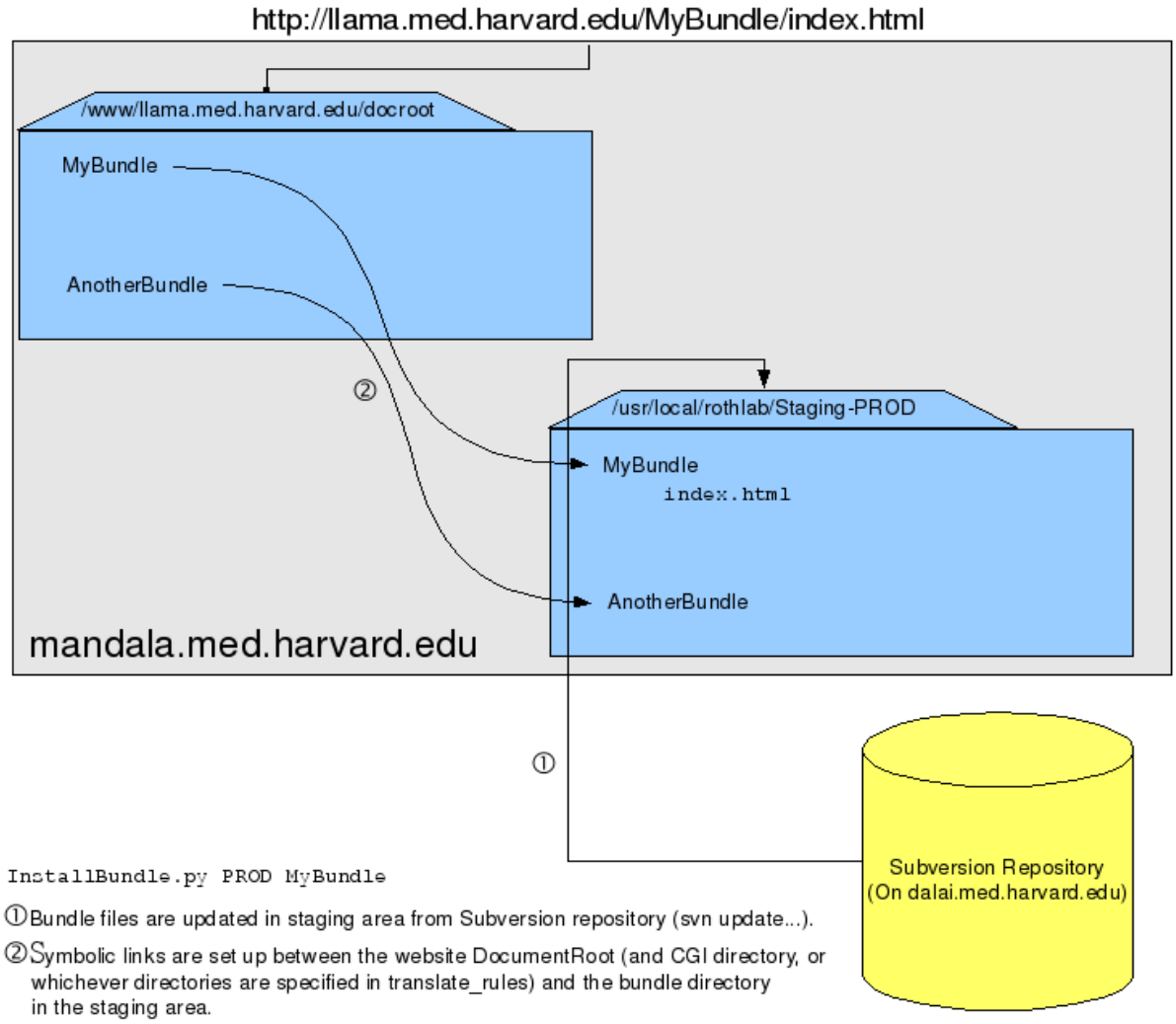
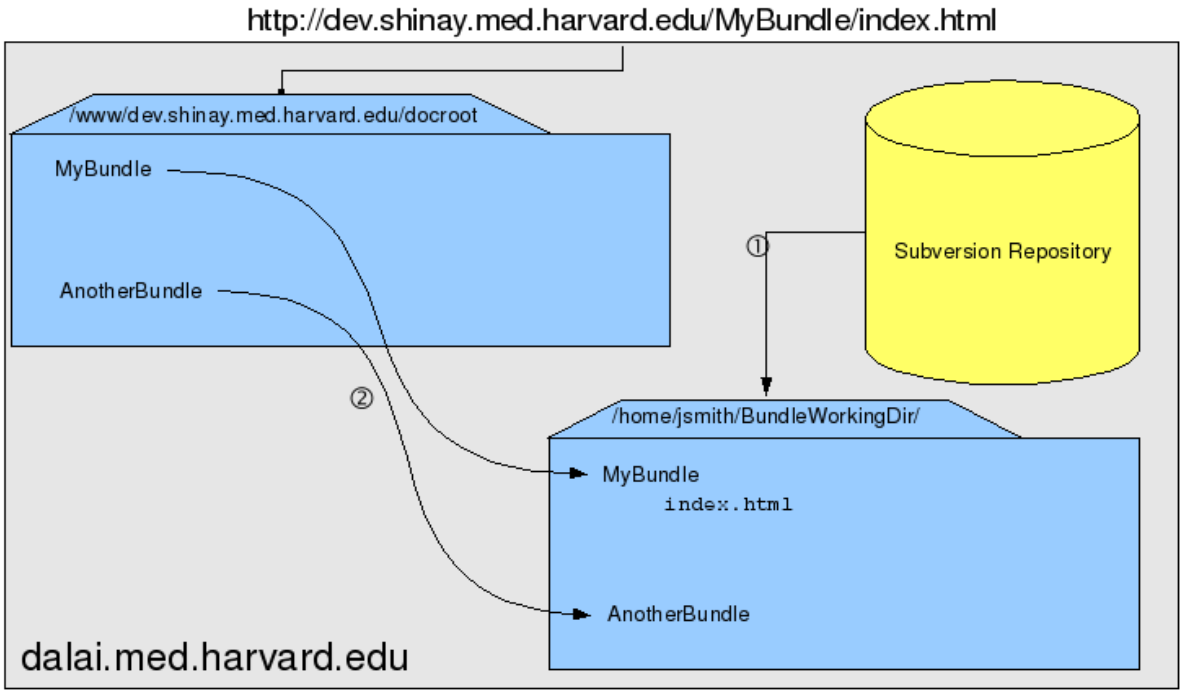

http://llama.med.harvard.edu/MyBundle/index.html
/www/llama.med.harvard.edu/docroot
MyBundle
AnotherBundle
②
/usr/local/rothlab/Staging-PROD
MyBundle
index.html
AnotherBundle
mandala.med.harvard.edu
①
Subversion Repository
(On dalai.med.harvard.edu)
InstallBundle.py PROD MyBundle
①Bundle files are updated in staging area from Subversion repository (svn update...).
②Symbolic links are set up between the website DocumentRoot (and CGI directory, or whichever directories are specified in translate_rules) and the bundle directory in the staging area.

Figure 3



`EditBundle.py MyBundle`

①Bundle files are checked out under user jsmith's home directory (svn co ...)..

②Symbolic links are set up between the website DocumentRoot (and CGI directory, or whichever directories are specified in translate_rules) and the bundle directory that was just checked out.

Figure 4

5 Appendix – Standard Directory Locations

Development website: http://dev.shinay.med.harvard.edu on dalai.med.harvard.edu:

URL Suffix[3]	Directory	Usage
docroot	/www/dev.shinay.med.harvard.edu/docroot	Html files
icons	/www/dev.shinay.med.harvard.edu/icons	Apache icons
cgi	/www/dev.shinay.med.harvard.edu/cgi	Scripts

Other directories:

Directory	Usage
/usr/local/rothlab/Bundler	Execution path for Bundler scripts.
/usr/local/rothlab/Staging-DEV	Staging area for development website.

Integration testing website: http://shinay.med.harvard.edu on dalai.med.harvard.edu:

URL Suffix	Directory	Usage
docroot	/www/shinay.med.harvard.edu/docroot	Html files
icons	/www/shinay.med.harvard.edu/icons	Apache icons
cgi	/www/shinay.med.harvard.edu/cgi	Scripts

Other directories:

Directory	Usage
/usr/local/rothlab/Bundler	Execution path for Bundler scripts.
/usr/local/rothlab/Staging-TEST	Staging area for development website.

Production website: http://llama.med.harvard.edu on mandala.med.harvard.edu:

URL Suffix	Directory	Usage
docroot	/www/llama.med.harvard.edu/docroot	Html files
icons	/www/llama.med.harvard.edu/icons	Apache icons
cgi	/www/llama.med.harvard.edu/cgi	Scripts

Other directories:

Directory	Usage
/usr/local/rothlab/Bundler	Execution path for Bundler scripts.
/usr/local/rothlab/Staging-PROD	Staging area for development website.

3 Appended to the end of the base URL.