PACC web site maintenance

Nick Dokos

December 6, 2004

Reviewer List

Rob Adams
Jill Lewis
Merrith Sabo-Jones

PACC Web Team

Contents

1 Introduction

This document describes the structure of the PACC web site. It is intended for members of the Web team. It is assumed that the reader knows the basics of HTML and, roughly, what the interactions between a browser and a web server are. Familiarity with CGI processing and a scripting language (Perl or Python) will be helpful, but not essential.

Some references to material describing these things are provided in the bibliography.

2 Structure

The web site *www.pacc-ucc.org* is served by a machine called `sand.ebeep.org`, that is administered by Rob Adams. Even though Rob has moved away, he still graciously provides both the computing resources and much time for the church web site. This machine is currently running a version of Linux, Debian "Sarge", with an Apache HTTP server, version 1.3.31. It has an IP address of 69.44.59.84.

2.1 HTTP server

The Apache server allows the use of virtual hosts, so one machine can actually serve multiple web sites, each with its own configuration. For example, the machine `sand.ebeep.org` hosts all of the church web sites: *www.pacc-ucc.org*, *deacons.pacc-ucc.org*, *ce.pacc-ucc.org*, *music.pacc-ucc.org*, *choir.pacc-ucc.org*, *exec.pacc-ucc.org*, *lg.pacc-ucc.org*, *youth.pacc-ucc.org* and *lists.pacc-ucc.org*.

These are set up through */etc/apache/httpd.conf*, the Apache server configuration file. Here is a small portion of this file that describes the main church web site[1]:

```
<VirtualHost 69.44.59.84:80>
   User "#1023"
   Group "#1015"
   ServerName pacc-ucc.org
   ServerAlias www.pacc-ucc.org
   DocumentRoot /home/pacc/public_html
   ErrorLog /home/pacc/logs/error_log
   CustomLog /home/pacc/logs/access_log common
   ScriptAlias /cgi-bin/ /home/pacc/cgi-bin/
   <Directory /home/pacc/public_html>
      Options FollowSymLinks IncludesNOEXEC Indexes
      AllowOverride All
   </Directory>
   Alias /images/ /home/pacc/public_html/images/
</VirtualHost>
```

So when a browser asks for `http://www.pacc-ucc.org`, some internet magic happens,the request arrives at machine 69.44.59.84, port 80 (the port on which the HTTP server is listening) and the match on the name `www.pacc-ucc.org` ensures that the portion of the config file shown above is the active portion for this request. There are a couple of things of interest here.

First, note the *DocumentRoot* directive: this tells the server that the document files are to be found in the directory */home/pacc/public_html*. This is where all the files with the church web content reside. Note also the location of the log files: access requests and errors are logged to these log files, so if something is wrong, these are the first files to check.

As a security measure, the server will *not* execute arbitrary programs: only programs that live in a directory named by a *ScriptAlias* directive are allowed (presumably after careful testing and vetting by competent security authorities - i.e. Rob). The *Alias* directive allows the directory that contains images to live somewhere else on the disk, instead of being under the document root directory - this is not taken advantage of here – the images directory *is* under the document root directory – so the directive is not strictly necessary, but it allows flexibility, in case we want to move it in the future.

Finally the *Directory* directive allows us to extend or override some of the global configuration, through a local configuration file, residing in the directory named in the directive and usually called *.htaccess*. There is some magic that happens through this file that is described in section 3.

The definitive reference on all of this stuff is the Apache documentation[1].

[1]Only Rob can modify this file, so this is only to illustrate how things are structured.

2.2 Directories and files

The *old* directory structure was (almost) flat: all the files and the programs were in the top level directory. The new structure has two levels: the top level directory contains the ***index.htm*** file[2] and a bunch of subdirectories. This directory structure reflects the structure of the menu that you see on the left hand side of the page, when you view it through your browser:

- *pacc:* The "About PACC" items.
- *staff*: The "Staff" items.
- *events*: The "Events" items.
- *groups*: The "Groups" items.
- *utils*: The rest: search, other links and contacts.

In addition, I created some subdirectories for topics that don't appear in the menu, but there were multiple files associated with them, so they tend to clutter the directory:

- *music*: Choir, concert series, concert schedules, organ.
- *outreach*: Outreach activities.
- *sarah*: Sarah's Circle and book list.
- *spire*: One issue of the Spire – should we resurrect it?

And then, there are the tools that glue everything together:

- *index.htm*: The top-level page.
- *cgi-bin*: ***frameit.cgi***, ***mailto.cgi*** and ***FormMail.pl***: see section 3 for the details.
- *images*: GIFs and JPGs of people, places and page backgrounds.
- *stats*: Stats collection.

The hope is that this structure will make it easier to find what needs to be changed.

There are three points that are worth mentioning. In the old structure, most links were very simple:

```
<a href="file.html">Link to file</a>
```

In the new structure, the safest thing to do is to always use an "up-and-over" scheme:

```
<a href="../dir/file.html">Link to file</a>
```

[2]Note the convention of using *.htm* as the suffix for HTML files.

I have done this for all of the files that needed it. There are some files that are only pointed to by files in the same directory, so I did not bother changing them. However, it is always safe to use the "up-and-over" scheme, even for file in the same directory. The only file that should not use the scheme is the top-level *index.htm*.

Second, references to images and to programs are now absolute, rather than relative.[3] Programs are always in the *cgi-bin* directory and images are always in the *images* directory, so references to them become

```
<img ... src="/images/image.jpg" ...>
<a href="/cgi-bin/prog.cgi?...">Run a program</a>
```

Third, some files are referred to by two different files that ended up in different directories. For example, *concertseries.htm* fits naturally in the music subdirectory, but it is part of the events menu, so I added a symbolic link in the events subdirectory, pointing to the "real" file in the music subdirectory. If you need to create a new file and it has to be in two places, this is the preferred method of doing it: put it where it naturally belongs and put a link to it in the other place.

2.3 The *other* PACC web sites

The other PACC web sites have document roots at */home/pacc/domains/<name of web site>*. There is not much to tell here, since they all have their own usage patterns. They all have the same top-level structure: a *public_html* directory containing the web content and a *log* directory.[4] There is a top-level *index.htm[l]* in each. After that, each site goes its own way.

Perhaps, the only thing to discuss here is that a couple of directories are password-protected. Since we may be called upon to reset passwords or create new password-protected directories, I describe the mechanism here.

XXX Is the following correct? XXX

The policy is set by adding a couple of directives to the *.htaccess* file. This file can be thought of as an extension of the global server configuration file that applies to a directory and all its subdirectories. Adding a few directives to this file will cause the server to ask for a user ID and password, before letting a remote browser access to the page(s) served by files in that directory (and any subdirectories). The directives are as follows (I believe all of the following are necessary except possibly for the AuthGroupFIle one, but it's a belief unfettered by considerations of connections to reality - all I know is that with all of these, it asks me for a user name and password):

```
AuthUserFile "/home/pacc/domains/domain.pacc-ucc.org/domain.passwd"
AuthAuthoritative On
AuthGroupFile /dev/null
```

[3]Of course, they are not really absolute path names: they are relative to the document root directory.

[4]The *cgi-bin* and *homes* directories do not seem to be used by anybody.

```
AuthName "Authorized Access"
AuthType Basic
require user mrdomain
```

This says that authorization is on, and that only users who give the correct user ID ("mrdomain") and password (stored in the file named in the *AuthUserFile* directive) will be allowed in. Btw, the password travels as clear text over the wire – it'll deter the casual passer-by, but not much more than that.

The password file is created using the command *`htpasswd`*. It asks you to type in a password twice and stores an encrypted form in the Usage is `htpasswd -c <passwdfile> <username>`; e.g. to create the password file above for the given user, say

```
htpasswd -c /.../domain.pacc-ucc.org/domain.passwd mrdomain
```

The *`-c`* option (re)creates the password file: without it, an entry is added to the file. You can of course put the password file wherever you want (as long as file permissions will allow), but it's better to keep it out of the document root tree. Read the man page for more information.

(see section 3 for add

3 Processing

There are three programs (written in Perl) that the HTTP server invokes on the browser's behalf. Two of them, *`mailto.cgi`* and *`FormMail.pl`* require explicit action by the remote user: clicking on a "Send" button in a form.[5]

The third one, *`frameit.cgi`*, is invoked automatically by the server. This magic happens through the *`.htaccess`* file (see section 2.1). Actually, there are two of them: one in the top-level directory and one in the *`cgi-bin`* directory. The top level one contains the following:

```
AddHandler frame-it .htm
Action frame-it /cgi-bin/frameit.cgi
```

The first line associates the "frame-it" action with any file whose name ends in *.htm*. The second define the "frame-it" action to be the invocation of the *`frameit.cgi`* program in the *`cgi-bin`* directory. This program adds the left-hand side menu, the vertical rule, the banner at the top and the comments, copyright and date notices at the bottom of each page.

3.1 The frameit program

You probably should be looking at the code while reading this section.

[5]BTW, these programs include e-mail addresses, so keep an eye out on them as well, if you are making changes to e-mail lists.

When the server invokes this program, it passes to it the pathname of the page that is to be served. The program opens two files: ***pacctemplate.htm*** [6] and the file containing the page. The template file contains some placeholders (for the title, the contents of the page and the date) and the ***frameit.cgi*** program simply substitutes the appropriate values in place of the placeholders. It writes the result to its standard output.[7] The only cases where the framing does *not* happen is for any file named ***index.htm*** and for any file that contains the word "deacons" in its name.[8] This is what keeps the top-level page from being decorated.It also means that if you ever create another file called ***index.htm*** (perhaps in a subdirectory), it too will *not* be decorated!

3.2 The mailto program

The ***mailto.cgi*** program is more complicated. The idea is to not embed e-mail addresses in web pages, because robots can get to them and use them for spam, virus e-mail and other such shady purposes. So the mail links in all the pages have been replaced by invocations of the ***mailto.cgi*** program. This program presents a form to the remote browser, allowing the user to select the recipient of the e-mail from a predefined list and to type the text of the email into a text box. When the user clicks on the "Send" button, an e-mail message is constructed with the real e-mail address of the recipient and the text that the user typed in the text box as the e-mail message text. It is then sent normally.

There seem to be some bugs here. For example, the ***mailto.cgi*** program is sent arguments but it seems to ignore them.[9]

In any case, the mailto.cgi program consructs a page that contains a POST form associated with the ***FormMail.pl*** program, and adds the various GUI elements: a table with several rows: a row with a label and a popup menu (although it looks like a rolldown menu in my browser), rows with a label and a text box for the sender's name, e-mail address and the subject of the message and a row with a label and a text area for the message itself; and a "submit" button with the label "Send". When the user presses the "submit" button, the form is processed by the ***FormMail.pl*** program.

This program contains a mapping table, ***recipient_alias***, that maps the aliases that were presented to the remote user to the real e-mail addressses corresponding to those aliases. Note that these addresses are not part of any page, so they cannot be harvested by Webbots. It constructs the mail message by substituting the correct values for placeholders in a template. In this case, the template is part of the program itself, instead of being stored in a separate

[6]The template file is also in the ***cgi-bin*** directory. It should be thought of as part of the frameit program, so this is the proper place for it.

[7]Which the server has cleverly arranged to pipe the stuff to the remote browser. This is part of the CGI spec.

[8]Why is that?

[9]For example, when the "e-mail the web team" link at the bottom of each page is clicked, the recipient is specified as "web" and the subject is specified as "Web Page Request", but the values don't make into the form.

file. It then pipes the message to `sendmail` which sends it on its way.[10]

This is enough for simple modification purposes, but there are a few things that might be worth doing. One would be to fix, if possible, the bug mentioned above, so that the parameters are used to fill in the default values of a couple of the fields in the form. The second is to document how to debug the `mailto.cgi` program: there is a “magic” parameter that adds the web team addresses to the normal target, so that the mail is sent to the web team as well. Mañana.

4 How do we get there?

Now that we know what the files look like, let’s discuss how to change them. In this section, I describe a couple of methods that can be used to modify web content. I have a strong preference for one of them, which will become obvious in the course of the description. Let’s start by enumerating various methods and discussing pros and cons.

The simplest method is to login to the server and edit the file directly. The next simplest is to get a copy of the file on your machine, edit it there and then copy it back. Then there are various web authoring tools that hide the copying from you.

The disadvantage of all of these is that any change is immediately visible. In other words, any mistake can bring down (part of) the site. The obvious solution is to make changes to a test web site and then copy files to the real web site after testing. But that’ still error prone and we can do better.

Security is another problem: we want to be able to modify the web pages in as secure a manner as possible. One way to do this is to provide a secure way to login to the server and modify the pages locally. SSH provides a way to do this. But, given a choice, most people would like to work locally on their machine and see the changes reflected on the web site instantly.

The final problem is source control. What happens if you make a change and you find it does not work: you want to take back the change and revert to the previous state.

There is an open-source tool, called “Subversion” (abbreviated svn),[2] that can help with these problems. It’s a client-server system: the server hosts a *repository* that holds the project’s files. Clients, running on remote machines, can check out a snapshot of the repository and make local modifications. A client can them *commit* the resulting changes, an action which updates the files on the server.

Svn can be tunnelled over ssh very easily, which pretty much takes care of the security aspects. For command-line people, the standard svn distribution includes a command-line client that will run on Linux, any Unix flavor or

[10]Since the file is around 1500 lines (about 300 empty lines, 550 lines of documentations and 650 lines of code), you might suspect that I have simplified the description and you would be right. But I can only manage bite-sized quantities of Perl, before my head starts hurting. And this program has the additional disadvantage of containing large swaths of HTML for the template elements, so you have to figure out the demarcation points. By now, it’s become a migraine.

MacOSX. But there is also a very nice Windows client, TortoiseSVN that is integrated tightly with Windows Explorer. This overlays icons on folders

I have prototyped all the steps below on machines at home, but I have not done anything with the server yet. As is usually the case, the description below will sound much more complicated than the actual situation. The best thing to do is read it and say "Huh?", then we get together and we do it and all becomes dazzlingly clear.

4.1 Installation

I asked Rob to install the SVN bits on the server. The next step is to set up a repository on the server that can be accessed over SSH by all of us.

Each of us needs to install an SVN client on his/her machine. I will be happy to help with that. If you are working on Linux, getting the appropriate package for your distribution is the best way to go. As a last resort, building from source can be done and it is very easy as well. On Windows, there is a command line client,[3] but I would recommend the TortoiseSVN client.[4] You can install both and see which one suits your work style best.

One nice thing about the Tortoise client is that it is tighly integrated with Windows Explorer (the only place you see Tortoise is in the context menus of files and directories - the ones you get by right-clicking). And when a file or directory is under source control, there is a visual indication of its state: a green check mark if it has not been modified, a red exclamation mark if it has been and has not been committed yet, etc.

4.2 Configuration

This step needs to be done once: when you check out the project's files for the first time.[11] There are two pieces of information that are needed: the URL of the repository and the authentication information (user ID and password) that SSH needs to create the tunnel between your machine and the server. For Tortoise, the tunnelling setup is accomplished as follows: open Windows Explorer, then click on File-¿TortoiseSVN-¿Setup. In the pop-up window, click on the Network tab, and in the Client field, either type or browse to the PLINK program that came with TortoiseSVN. The default location is

```
c:\Program Files\TortoiseSVN/bin/TortoisePlink.exe
```

[11]You might find yourself in a jam in the future: everything is foobar and rather than fiddling with a million bits and pieces trying to glue the whole thing back together, you just want to sweep everything out and start afresh. You can do that. The server does not keep any state on a client's behalf, so what the client does is its own business. The server learns about changes only when a client does a commit. So you can always delete everything on the client and do another checkout. Tortoise remembers the locations of repositories so you will not have to remember anything. The command line client does not but before you delete the project, every directory contains a subdirectory called *.svn*, which contains the metadata that svn uses. In particular, the top-level directory's *.svn/entries* file contains the URL of the repository.

Then add to the end of the line the authentication information as follows:

```
c:\Program Files\TortoiseSVN/bin/TortoisePlink.exe -l USER -pw PASS
```

where USER and PASS are placeholders for the real authentication information.[12] [This is TBD] The URL of the PACC web site repository is

```
svn+ssh://www.pacc-ucc.org/home/pacc/svn/web/pacc/trunk.
```

4.3 Working with svn

After installation, create a folder somewhere on your machine, change into it and then check out the repository. For Tortoise, just right-click in the folder and select Checkout from the pop-up menu. From the command line, say [The actual URL is TBD]

```
svn checkout svn+ssh://www.pacc-ucc.org/home/pacc/svn/web/pacc/trunk
```

The machine will churn for a while, but at the end you will have on your machine a copy of the whole repository. You can then edit a file with whatever tool you use to edit HTML files. Once done, you can commit the changed file(s): with Tortoise, right-click on the top-level directory and choose Commit from the pop-up menu; from the command line say *`svn commit`*. That's it.

Fire up your browser and visit http://test.pacc-ucc.org and admire your masterpiece.

The thing is that since each of us makes changes and commits *our* changes, how are we ever going to see the changes that other people make in our own working directory? The answer is that we don't, unless we do an update. For Tortoise, go to the top-level folder and righ-click, then select "Update" from the popup menu; for the command line client, say *`svn update`*. Svn will figure out what has changed and will pull the changed stuff from the server. What if you forger to do that? You will find out when you try to commit changes in the future: *`svn`* will complain that some files on the server are newer than the old, stale copies in your working directory. So you do an update, and then commit. But there might be a problem: what if you have changed a file and somebody else has also changed and committed before you? In this case, when you do the update, svn will merge the other person's changes into your changed copy, so when you commit, *both* sets of changes are going to be in the repository. Generally, svn will be able to merge the changes with no problem, but occasionally there might be a conflict. Here is an example: somebody notices a typo on one of the web pages and notifies the web team. One member gets the email immediately, corrects the typo in her own working copy and does a submit. I read the email later that night and correct my working copy, except it's very late and I make a different typo on that same word. Now, when I try

[12]That means that the password is stored unencrypted on disk, something that makes security-conscious people bristle. There are ways to make it more secure, but as long as the machine you are using for this is at home and not accessible to other people, it should be OK. Don't do this on your laptop!

to commit, svn will tell me that my copy is out of date, so I do the update. But when svn tries to merge the two sets of changes, it throws up its hands: the same word has to change in two incompatible ways. This is a conflict: svn will throw a hissy fit and tell me to fix it manually. Presumably, at that point I will be awake enough to choose the correct alternative between the two. I can then commit my copy (perhaps I've made other changes as well that need to be saved in the repository) and everything will go swimmingly.

One thing that I have not discussed is moving files or directories from one place to another, or deleting files or directories (creating a file or a directory is no problem). [This is TBD - if you need to do it, read the documentation first!]

4.4 Testing

[TBD: The following is not implemented yet. For now, it has to be done by hand.] If you were paying attention earlier, I said that the URL to go to, in order to admire your masterpiece was `http://test.pacc-ucc.org`. The real web site on the other hand is at `http://www.pacc-ucc.org`. There are indeed two web sites: the test one is server directly from the repository, so after you commit some changes, the test site reflects those changes immediately. Not so for the real web site. That is served out of a different directory, which we don't directly touch. Instead, every 24 hours, at 4am, a daemon wakes up, finds out what has been modified on the test site directory and copies it to the real web site directory. Presumably, you will have changed everything to your satisfaction on the test site before then, so this will be safe.

What happens if a change needs to be propagated to the real web site *right now*? [TBD]

References

[1] Apache Documentation. `http://www.apache.org`.

[2] Subversion web site. `http://subersion.tigris.org`.

[3] Subversion packages. `http://subversion.tigris.org/project_packages.html`.

[4] TortoiseSVN web site: download and documentation. `http://tortoisesvn.tigris.org/`.