MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

Adobe® PDF Magazine Version

INTRODUCTION TO WEBDAV

INSIDE

DRUPAL ON FREEBSD

BACKUP YOUR LAPTOP FROM ANYWHERE

AUTHENTICATING NAT WITH AUTHPF

XMODMAPON THE WAY TO WRITING HIEROGLYPHS QUICKLY

FREEBSD BINARY UPGRADE

MANAGING SOFTWARE WITH NETBSD'S PKGSRC PACKAGING SYSTEM

VOL.3 NO.11
ISSUE 11/2010(17)
1898-9144

iX-Triton TwinBlade Servers: The Easy-to-Manage, Greener Way to Serve

› AFFORDABLE › ECONOMICAL › SAVINGS

The new Triton TwinBlade Server is the most technologically advanced blade server system in the industry, and the ideal solution for power-efficiency, density, and ease of management.

The Triton TwinBlade Server supports up to 120 DP servers with 240 Intel® Xeon® 5600/5500 series processors per 42U rack, achieving an unmatched 0.35U per DP node. Up to two 4x QDR (40 Gbps) Infiniband switches, 10GbE switches or pass-through modules give the TwinBlade the bandwidth to support the most demanding applications.

With N+1 redundant, high efficiency (94%) 2500W power supplies, the TwinBlade is the Greenest, most energy-efficient blade server in the industry. The energy saved by the iX-Triton TwinBlade Server will keep the environment cleaner and greener, while leaving the green in your bank account.

Server management is also simple with the Triton Twin Blade Server. Remote access is available through SOL (Serial Over Lan), KVM, and KVM over IP technologies. A separate controller processor allows all of the Triton's remote management and monitoring to function regardless of system failures, offering true Lights Out Management.

Using the Triton's management system, administrators can remotely control TwinBlades, power supplies, cooling fans, and networking switches. Users may control the power remotely to reboot and reset the Triton TwinBlade Center and individual Twin Blades, and may also monitor temperatures, power status, fan speeds, and voltage.

For more information on the **iX-Triton TwinBlade**, or to request a quote, visit:

http://www.iXsystems.com/tritontwinblade

20 Server Compute Nodes in 7U of Rack Space

The iX-TB4X2 chassis holds 10 TwinBlade servers and each TwinBlade supports two nodes. This gives the iX-TB4X2 chassis the ability to house 20 nodes in 7U of rack space. The powerful Triton TwinBlade achieves 0.35U per dual-processor node, and is twice as dense as the previous generation of dual-processor blades.

A fully-loaded iX-Triton TwinBlade supports 40 Intel® Xeon® 5600/5500 series processors and up to 2.5 TB DDR 1333/1066/800MHz ECC Registered DIMM memory. In a 42U rack this translates into 120 nodes with 240 Intel® Xeon® 5600/5500 series processors and 15 TB DDR 1333/1066/800MHz ECC Registered DIMM memory.

- By replacing 1U servers with TwinBlade servers, the power savings of the iX-TB4X2 can reach more than $1000* per year, per server with reduced cooling costs added in.

- Replacing 1U rackmount servers with an iX-TB4X2 Twin Blade can reduce carbon dioxide emissions by over 5.5 metric tons.**

- The iX-Triton TwinBlade delivers the most energy-efficient blade server in the industry with four N+1 redundant, high efficiency (94%) 2500W power supplies.

** Electricity costs vary by location.*

*** According to Energy Information Agency (a statistical agency of the U.S. Department of Energy), saving one kilowatt hour of electricity reduces carbon dioxide emissions by 1.43 pounds.*

Key features:

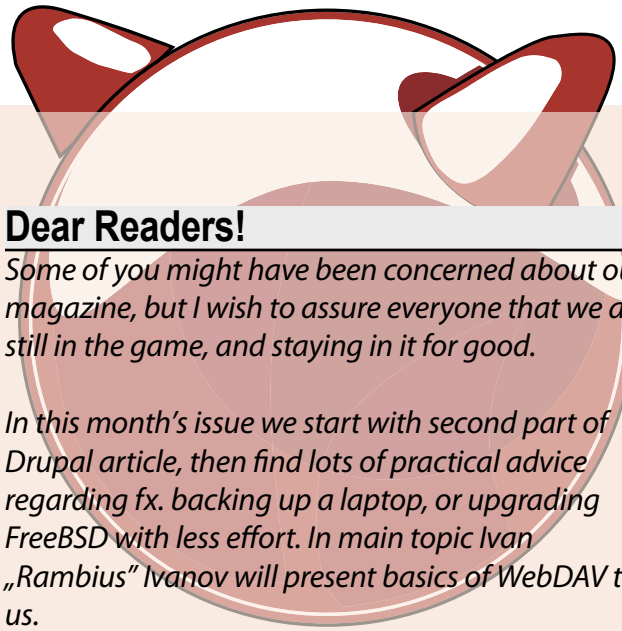
- Up to 10 dual-node TwinBlades in a 7U Chassis, 6 Chassis per 42U rack
- Remotely manage and monitor TwinBlades, power supplies, cooling fans, and networking switches
- Hardware Health Monitor
- Virtual Media Over Lan (Virtual USB, Floppy/CD, and Drive Redirection)
- Integrated IPMI 2.0 w/ remote KVM over LAN/IP
- Remote Power Control
- Supports one hot-plug management module providing remote KVM and IPMI 2.0 functionalities
- Up to four N+1 redundant, hot-swap 2500W power supplies
- Up to 16 cooling fans

Each of the TwinBlade's two nodes features:

- Intel® Xeon® processor 5600/5500 series, with QPI up to 6.4 GT/s
- Intel® 5500 Chipset
- Up to 128GB DDR3 1333/ 1066/ 800MHz ECC Registered DIMM / 32GB Unbuffered DIMM
- Intel® 82576 Dual-Port Gigabit Ethernet
- 2 x 2.5" Hot-Plug SATA Drive Trays
- Integrated Matrox G200eW Graphics
- Mellanox ConnectX QDR InfiniBand 40Gbps or 10GbE support (Optional)

Call iXsystems toll free or visit our website today!

+1-800-820-BSDi | www.iXsystems.com



Dear Readers!

Some of you might have been concerned about our magazine, but I wish to assure everyone that we are still in the game, and staying in it for good.

In this month's issue we start with second part of Drupal article, then find lots of practical advice regarding fx. backing up a laptop, or upgrading FreeBSD with less effort. In main topic Ivan „Rambius" Ivanov will present basics of WebDAV to us.

I hope you will find this issue to be both interesting and helpful. And as always – if you have any comments or inputs, or you think you could contribute to our magazine, please mail us.

I would also like to take this opportunity to wish you all a merry christmas and happy new year. And many presents of course. :)

Thank you!

Zbigniew Puchciński
Editor in Chief
zbigniew.puchcinski@software.com.pl

MAGAZINE BSD

Editor in Chief:
Zbigniew Puchciński
zbigniew.puchcinski@software.com.pl

Contributing:
Rob Somerville,Daniele Mazzocchio, Rashid N. Achilov, Joseba Mendez, Laura Michaels
Lukas Holt, Caryn Holt, Laura Michaels

Special thanks to:
Marko Milenovic, Worth Bishop and Mike Bybee

Art Director:
Ireneusz Pogroszewski

DTP:
Ireneusz Pogroszewski

Senior Consultant/Publisher:
Paweł Marciniak *pawel@software.com.pl*

National Sales Manager:
Ewa Łozowicka
ewa.lozowicka@software.com.pl

Marketing Director:
Ewa Łozowicka
ewa.lozowicka@software.com.pl

Executive Ad Consultant:
Karolina Lesińska
karolina.lesinska@bsdmag.org

Advertising Sales:
Zbigniew Puchciński
zbigniew.puchcinski@software.com.pl

Publisher :
Software Press Sp. z o.o. SK
ul. Bokserska 1, 02-682 Warszawa
Poland
worldwide publishing
tel: 1 917 338 36 31
www.bsdmag.org

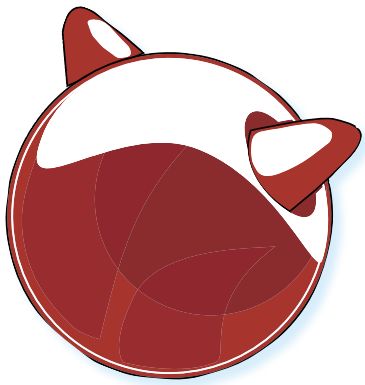
Software Press Sp z o.o. SK is looking for partners from all over the world. If you are interested in cooperation with us, please contact us via e-mail: *editors@bsdmag.org*

The editors use automatic DTP system AUPOS

Mathematical formulas created by Design Science MathType™.

Get Started

How To's



Drupal on FreeBSD

part 2

In the previous article in this series, we looked at installing the Drupal Content Management System. We are now going to look at configuration, templates, modules and adding content.

What you will learn…

- How to install and patch modules, add themes and edit content in Drupal

What you should know…

- Basic BSD system admin skills

Post install configuration

Log in to your website using the password supplied in the initial install and complete the CAPTCHA question if required. You will be presented with the administration page (see Figure 1 and 2). Visit *Home>Administer>Status reports* and see if any issues need fixing. In our example, Cron maintenance tasks, Image import, ImageMagick and APC need attention (Figure 3).

To fix cron

SSH onto the box and su to root, install wget and add a crontab to pull cron.php every hour:

```
ssh 192.168.0.117
su
pkg_add -r wget
crontab -e
```

Then add the following line (press `i` in the vi editor to add a line, `Esc : wq!` to save and exit):

```
15 * * * * /usr/local/bin/wget --quiet -O - http://
                    localhost/cron.php
```

This will run the Drupal cron.php script at 15 minutes past every hour.

To fix Image Import

```
mkdir /usr/local/www/drupal6/sites/all/images
chown www:www /usr/local/www/drupal6/sites/all/images
```

Then add `sites/all/images` to the import path field (Figure 4).

Figure 1. *Login*

Figure 2. *Initial Administrator page*

Figure 3. *Errors found when running status report*

To fix Imagemagick

```
pkg_add -r fpc-imagemagick
cp /usr/local/www/drupal6/sites/all/modules/image/
  image.imagemagick.inc   /usr/local/
www/drupal6/includes/
```

Select the image toolkit and change the path to `/usr/local/bin/convert`, and save the configuration.

To fix APC (Alternative PHP Cache)

```
pkg_add -r pecl-APC
cp /usr/local/etc/php.ini-production /usr/local/etc/php.ini
echo 'apc.enabled="1"' >> /usr/local/etc/php.ini
echo 'apc.shm_size="50"' >> /usr/local/etc/php.ini
apachectl restart
```

Some additional tuning

To enable clean URLs, add the following lines at the end of the `<directory>` section of `/usr/local/etc/apache22/Includes/drupal.conf`. This option makes Drupal emit *clean* URLs (i.e. without `?q=` in the URL).

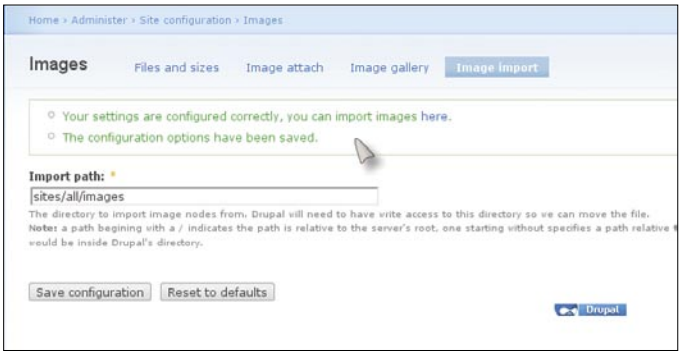


Figure 4. *Adding the image import path*

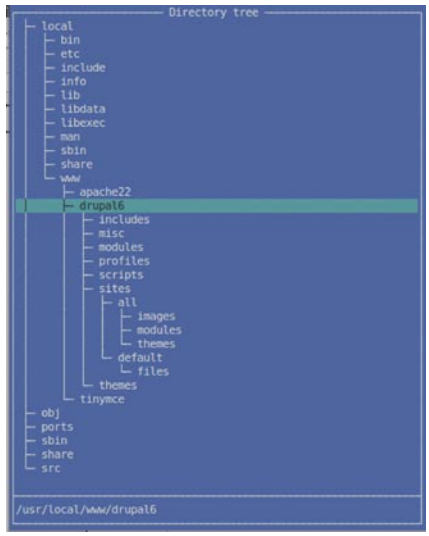


Figure 5. *The Drupal file structure*

```
    RewriteEngine on
    RewriteBase /
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule ^(.*)$ index.php?q=$1 [L,QSA]
```

Set your timezone in `php.ini`, under the `[DATE]` section:

```
[Date]
; Defines the default timezone used by the date functions
; http://php.net/date.timezone
date.timezone = „Europe/London”
```

Restart apache:

```
apachectl restart
```

Downloading and installing Modules and Themes

Installing/updating Modules

One of the major benefits of Drupal is the huge number of community contributed modules and themes that are

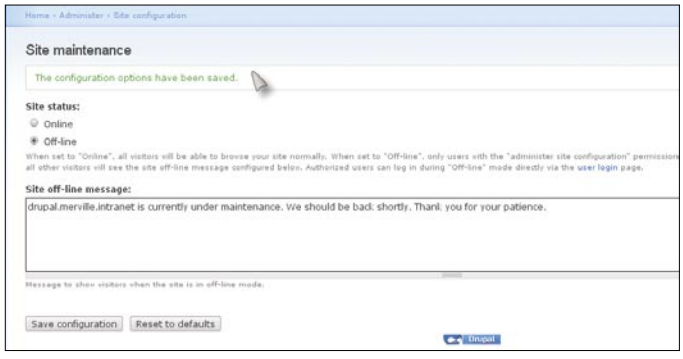


Figure 6. *Taking the site off-line*

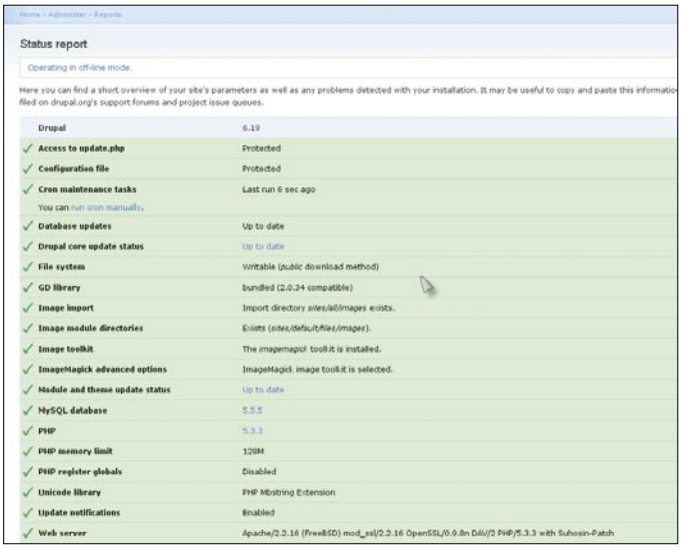


Figure 7. *A good status report*

available. While it is always advisable to check on the Drupal site as to the quality and life cycle of a module or theme, adding a theme or a module is just a matter of copying the tarball to the relevant directory (modules or templates) and extracting it. In this example we will update the Image module in the sites/all modules directory (see Figure 5 for a cut down overview of the Drupal file structure). *Ensure you run the update script after you update any modules*.

Drupal can be configured to automatically email the site admin when patches are required, visit *Home>Administer >Reports>Available updates*. See Figures 12-14 for an example of a Drupal installation that requires patching and the results after the update script has been run.

First, visit *Home>Administer>Site configuration>Site maintenance* and take the site offline (see Figure 6). In a root shell, perform the following:

Figure 8. *The site themed with Danland*

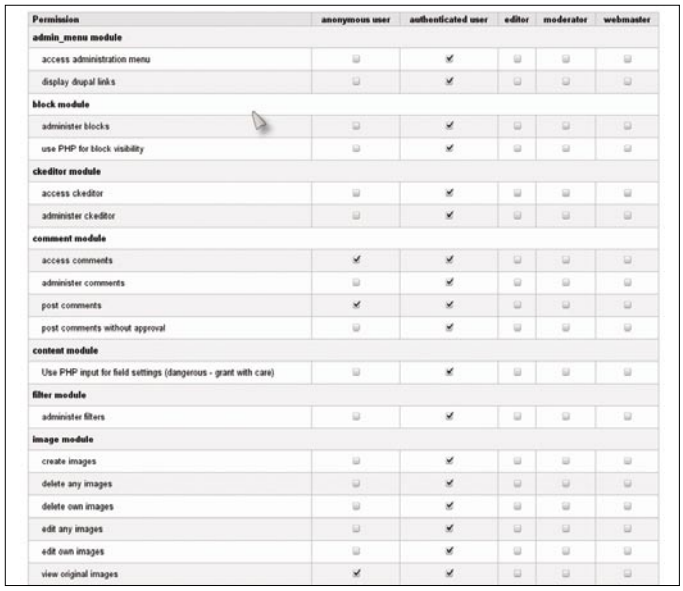


Figure 9. *Adding group permissions*

```
cd /usr/local/www/drupal6/sites/all/modules/
rm -fr image/
wget http://ftp.drupal.org/files/projects/image-6.x-1.0.tar.gz
tar -xvzf image-6.x-1.0.tar.gz
chown -R www:www image
```

N.B: By deleting the directory, you may be deleting additional libraries etc. This is particularly applicable to editors e.g. CKEditor. Overwriting the directory with the updated tarball may not be a good choice either, as cruft and old configuration files may remain to cause problems. If unsure, backup the directory to somewhere off the main Drupal tree to prevent the update script from failing which could prove fatal to your site.

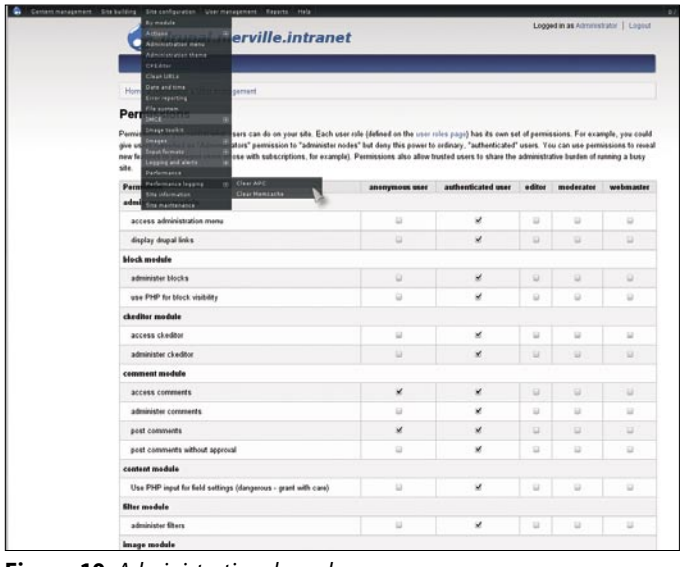


Figure 10. *Administration drop-down menu*

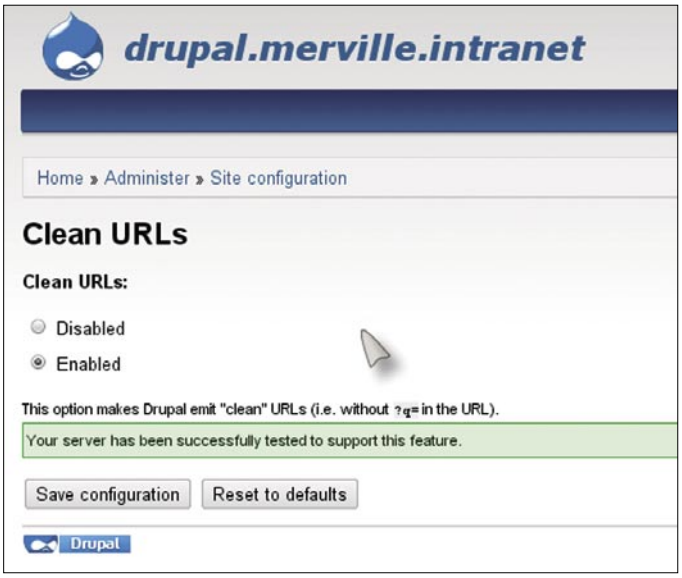


Figure 11. *Enabling clean URL's*

Visit the *Home>Administer>Status reports* page, and you will see that Drupal requests that you run the update script. Follow the instructions (On production systems it would be wise to backup the Drupal tree and the MySQL database). Provided the crontab has run successfully, you should be presented with no warnings or errors at the status report page (Figure 7). Return the site to on-line mode via *Home>Administer>Site configuration>Site maintenance*.

Be aware of module dependencies. For instance, if a module required the fictitious module `new_module` but it was not installed, you would need to proceed as follows:

```
cd /usr/local/www/drupal6/sites/all/modules/
wget http://ftp.drupal.org/files/projects/newmodule.tar.gz
```

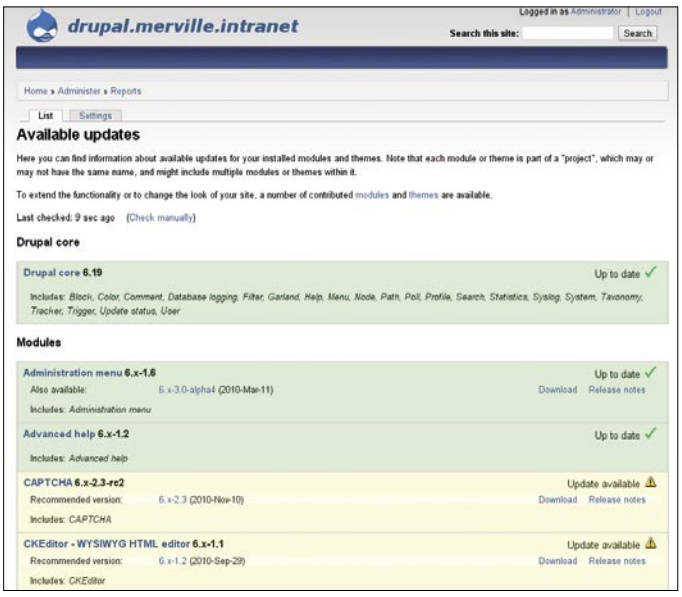


Figure 12. *Module update [1/3]*

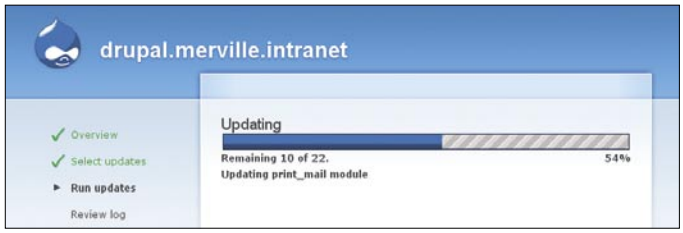


Figure 13. *Module update [2/3]*

```
tar -xvzf newmodule.tar.gz
chown -R www:www newmodule
```

Then enable the module as below.

Enabling and Configuring Modules

With over 3,000 modules available, the scope is endless for the web developer/webmaster to investigate and implement. In the previous article, we installed a few of the available modules from the FreeBSD ports collection. A list of some the most popular modules are listed in Table 2, and these will need to be configured from the *Home>Administer>Site building>Modules* page. When installed, most modules are disabled and you may have to enable various permissions and paths etc. Details on how to configure the editor follows in the next section. For the moment, you may find it useful to enable the Administration Menu and then save your changes which provides a drop-down menu at the top of the browser window (Figure 11).

Create a table for print module

```
mysql -udrupal -p\!1gH87i-LL34
use drupal6;
```

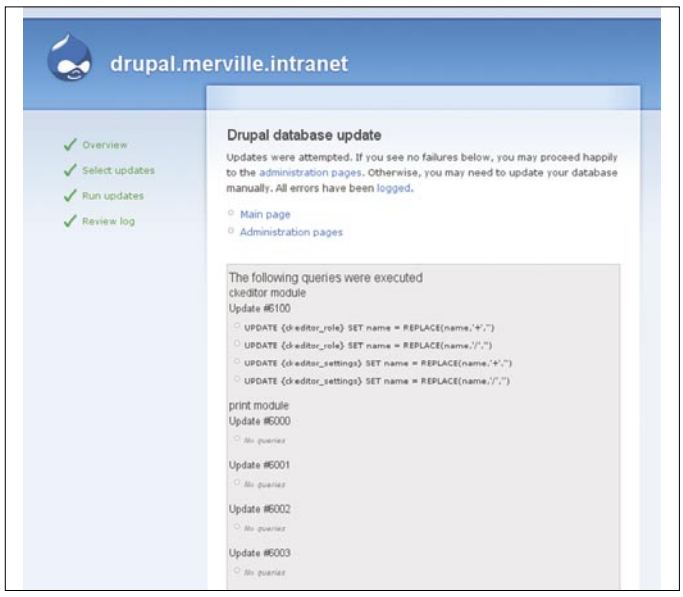


Figure 14. *Module update [3/3]*

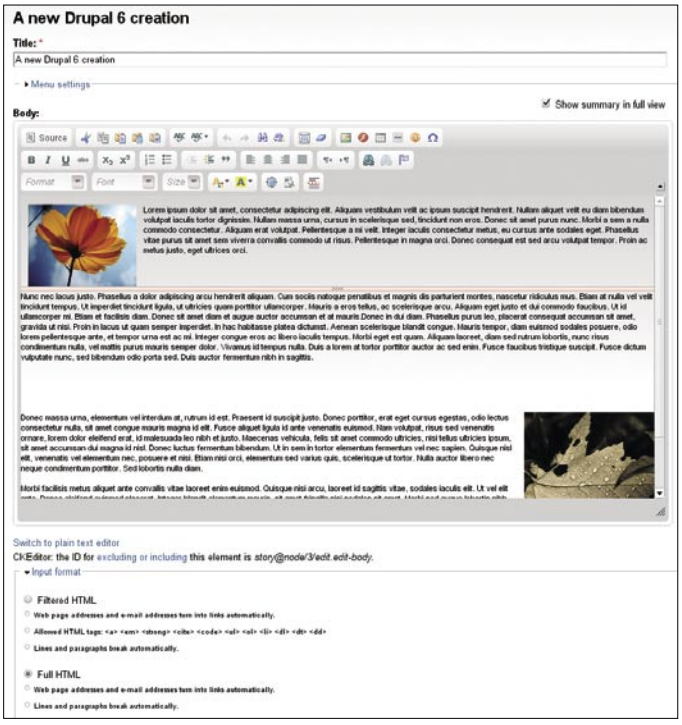


Figure 15. *Adding content. Full HTML should be used if you want to add images*

```
CREATE TABLE print_node_conf (nid int(10) unsigned NOT NULL,
 link tinyint(3) unsigned NOT NULL default '1', comments
tinyint(3) unsigned NOT NULL default '1',
url_list tinyint(3) unsigned NOT NULL default '1',
PRIMARY KEY (nid));
CREATE TABLE print_page_counter(path varchar(128) NOT NULL,
 totalcount bigint(20) unsigned NOT NULL default '0', timestamp
```

Figure 16. *Front page with teaser*

Figure 17. *Front page opened up – note the printer friendly icon and send to a friend*

```
int(10) unsigned NOT NULL default '0', PRIMARY KEY (path));
exit;
```

Installing Themes

For this example, I will download and deploy the CSS and XHTML compliant theme Danland which is released under the GPL licence by Dantesoft in Indonesia. In a root shell:

```
cd /usr/local/www/drupal6/sites/all/themes
wget http://ftp.drupal.org/files/projects/danland-6.x-
                    2.1.tar.gz
tar -xvzf danland-6.x-2.1.tar.gz
chown -R www:www danland
```

Navigate to *Home>Administer>Site building>Themes* in your browser and enable the Danland theme and set it to default. It is always worthwhile to check that themes / code etc. is compliant with W3C standards, and the front page of the newly themes site passes with flying colours (see Figure 7/8.)

Changing user permissions and roles

Drupal uses its own powerful permission system which needs to be configured if users other than the administrator, registered (authenticated) or non-registered (anonymous) are to be allowed access to modules. This is especially important in a corporate environment where there may be content approvers other than those who actually edit the content. Drupal also supports LDAP through various

Table 1. *Drupal resources*

Templates and themes	http://drupal.org/project/themes
Danetsoft	http://www.danetsoft.com/danland-drupal-theme
Drupal website	http://drupal.org
W3C HTML validation service	http://validator.w3.org/

Table 2. *Enabled Drupal modules*

* Denotes not in ports collection			
"SEO" Checklist	CCK (All modules)	Path	PHP filter
Poll	Profile	Search	Statistics
Trigger	Advanced help	CKEditor	IMCE
Lightbox2 *	Page Title	Path redirect	Pathauto
Token	Printer-friendly pages	Send by e-mail	CAPTCHA *
Views (All modules)	Administration menu *	Syslog	Tracker
Meta Tags (All modules)			

modules, but your mileage may vary. Unfortunately there is no PAM support available that I am aware of at present. Before we configure any further modules, we will set up 3 roles: Webmaster, Moderator and Editor. By configuring these roles (and later assigning them to the appropriate users at the time of account creation) we can fine tune the exact permissions they have when signed on to the site. For example we could allow the Webmaster to change themes, approve comments and add PHP code to the site, but only Moderators could approve comments.

Visit Administer/User management in your browser, add the above roles and then edit permissions as appropriate. To create a user, visit *Home>Administer>Users>Add users* where you can add your new users to the relevant group. See Figure 9 for permissions in action.

Note that as most modules are disabled for unauthenticated users, you will have to explicitly enable use by this class of user. For example, the printer friendly page and send to a friend will only appear in the content of the Administrator unless this module is enabled for anonymous users.

Installing the editor

If you have not done so already, enable CKEditor and IMCE in the modules section and save the configuration.

Now visit *Home>Administer>*Site *configuration>CKEditor* and edit the advanced profile. Change the toolbar to DrupalFull and the file browser type to IMCE and save the changes.

If you would prefer to use a different editor, visit the *Home>Administer>Site building>Modules* scroll down to User Interface enable wysiwyg and save the configuration. Click on *Home>Administer>By module* and visit the wysiwyg module. Once enabled, this module will allow you to use other editors (e.g. Yahoo YUI) as well as CKEditor.

If you get the following error:

```
  warning: Parameter 1 to theme_wysiwyg_profile_overview()
expected to be a reference,
     value given in /usr/local/www/drupal6/includes/
                  theme.inc on line 656.
```

please refer to the section Patching Drupal.

Adding content

Now that the editor and file uploads are in operation, we can add content. By default, Drupal provides 3 default content types, Image, Page and Story. A story is automatically promoted to the front page with an automatic teaser. Navigate to *Home>Create content* and create a Story, adding and uploading images via the editor dialogue – ensure you use Full HTML otherwise your images will not display. see Figures 15/17.

Patching Drupal

Occasionally, you may find you are advised to patch Drupal. Drupal supports a wide number of versions of PHP, but there seems to be an issue with wysiwyg under PHP 5.3 with the FreeBSD port. To rectify this, we will need to patch the module:

```
pkg_add -r patch
cd /usr/local/www/drupal6/sites/all/modules/wysiwyg/
wget http://drupal.org/files/issues/wysiwyg-php5.3.patch
patch -b .000 wysiwyg.admin.inc wysiwyg-php5.3.patch
chown www:www  wysiwyg.admin.inc
```

When the patch is tested, you can remove the .000 and .patch file.

ROB SOMERVILLE

Rob Somerville has been passionately involved with technology both as an amateur and professional since childhood. A passionate convert to *BSD, he stubbornly refuses to shave off his beard under any circumstances. Fortunately, his wife understands him (she was working as a System/36 operator when they first met). The technological passions of their daughter and numerous pets are still to be revealed.

An Absolute Beginner's

Guide To Using The Command Line Interface In Bsd

Life exists beyond Linux. There are other Open Source operating systems such as BSD and OpenSolaris. Introducing a Linux user to BSD is no big deal, considering the fact that most BSD versions now employ either KDE or GNOME. Read on as we bring explore the BSD command line.

What you will learn…

- to open/modify/work around with files using the Command Line, as well as perform most basic functions

What you should know…

- basic knowledge of UNIX terminology

A newcomer to BSD will probably find himself served well by the KDE/GNOME desktop environment that can run on BSD just as easily as on Linux. But BSD isn't just about the desktop; indeed, more so than Linux, it's aimed at the server market, with far less attention paid to desktop users than to system administrators who want a stable and speedy platform for running their high performance network services.

Even if you do not plan to use BSD for server purposes, the best interaction you can generally have with a BSD machine is through the textual command line. But for a newbie to the world of Unix, accustomed to computers as being launching points for web browsers and digital movies and 3D video games, it's a lot to swallow.

Fortunately, anyone whose experience with alternative operating systems includes any time spent dabbling in Linux can count himself fortunate if confronted by a BSD machine. This is because, as you can expect from any Unix, BSD works in much the same way that Linux does, particularly to a brand new user who wants to know what the basic commands are for getting around in the command-line interface.

Getting Work Done!

In a graphical operating system, the first thing you would do is double-click things on the desktop and start navigating around through the folders that are available. That impulse is no different in the *command line interface* (CLI); its just that the tools and techniques you use to do it are a bit different, and require you to memorize a few short commands and their obscure suffixes (flags or parameters) rather than learn a few moves to use with your mouse. The first command, the one that does the equivalent of showing you a window with a list of the files in it, is the `ls` command, which stands for *list*:

```
% ls
Birthday.mov
movies
shopping_list.txt
My DVD List.txt
Picture1.jpg
```

The textual output shown above consists of a line for the command you type, printed in bold, following the prompt set by your system (`%` in this case, but many systems use different prompts customized for their own users' purposes). This is followed by the output that appears after you press [*Enter*] or [*Return*].

Suppose, though, that you wanted to get a little bit more out of your file listing. File names are just text, after

all, and text is what a CLI has in spades. you ought to be able to coax a little more meaning out of this listing. Well, you're in luck: this is a perfect opportunity to demonstrate how parameters work.

```
% ls -F
Birthday.mov*
movies/
shopping_list.txt*
My DVD List.txt*
Picture1.jpg@
```

Adding the `-F` flag (after a space, don't forget the space!) modifies the output of `ls` by adding symbols that indicate what kind of file they are. They're not application-based icons like the ones you might be used to, distinguishing one file type from another; rather, BSD distinguishes objects much more broadly, by using a slash (`/`) to denote directories, the `@` sign for symbolic links (which you can think of as Unix versions of shortcuts or aliases), and asterisks (`*`) for executable files (programs). For example, in this sample output you can tell that movies is a folder, *Picture1.jpg* is really an alias for an image file somewhere else in the system, and *shopping_list.txt* is an executable text file, which you can launch like this:

```
% ./shopping_list.txt
```

The `./` prefix here tells the system that the specified file is in the current directory, denoted by a single dot (the parent directory is two dots, `..`).

In BSD, as in Linux and every other CLI-driven Unix, your command-line environment comes preloaded with several paths to directories in the system where executable programs might be found. These paths might include `/bin`, `/usr/sbin`, `/usr/local/bin`, and a few others. (The `ls` command, for example, lives in `/bin`).

Besides, most command line prompts display the current location or the active directory. But if yours does not, you can still find out your location using the `pwd` command (it stands for present working directory):

```
% pwd
/home/test
```

This tells me that I am in the `test` subdirectory of my home directory. I can move around using the `cd` (change directory) command, like this:

```
% cd /home/test2
```

This would move me into my home directory.

To change to the parent directory:

```
% cd ..
```

Or to move two levels up the tree:

```
% cd ../..
```

Or to return to my home directory using the shortcut of omitting the path parameter altogether:

```
% cd
```

Working With Files

Now that you know the basics of command-line navigation, you can put it to use by moving some files around. The first thing to do might be to create a text file. You can use the built-in ee editor for this:

```
% ee test.txt
```

When you've typed some text, press [*Escape*] to bring up the menu, then use the Up and Down arrows to select Leave editor. Choose Save changes in the next screen, and you'll exit the editor having created a new file. You can get a closer look at it now using the `ls` command in another new way:

```
% ls -l test.txt
-rw-r--r-- 1 sufyan users 561 Aug 31 2010 test.txt
```

Here, the `test.txt` argument makes `ls` show only the information on the specified file, and the `-l` flag makes it print its output in *long* form, meaning to show information on the file's permissions, ownership, last-modification date, and size (in bytes).

You can now move the file to some other location. This is done using the `mv` command (many core Unix commands, as you can see, are abbreviated to an almost comical degree – but for hardcore users who use these commands hundreds of times a day, the less typing they can get away with, the better):

```
% mv test.txt essays
```

This moves the file into the `essays` subdirectory. Just as with `cd`, you can also specify a destination such as the parent directory:

```
% mv test.txt ..
```

Or, if the file you want to move is somewhere other than your current directory, you can use a relative path to refer to it:

```
% mv ../test.txt essays
```

Or an absolute path:

```
% mv /home/sufyan/test.txt essays
```

What happens if you move a file into a directory where there's already a file with the same name as the one you're moving? Well, if the file's permissions allow it, the old file gets overwritten, without so much as a warning. This is one of the pitfalls – or, depending on how you see it, the blessings – of an austere CLI environment: it really knows how to get out of your way and let you do your work (or shoot yourself in the foot).

Indeed, the `mv` command has many potentially destructive uses. One of the most commonly surprising ones, to newcomers to Unix, is that the `mv` command is what you use to rename files.

Rather than having a dedicated command for *rename*, in Unix if you're changing a file's name, what you're really doing is *moving* the old file to a new name (which makes sense if you think of names as being trivial little identifiers that merely point to the really interesting stuff, the data):

```
% mv test.txt my_essay.txt
```

There is, however, a totally separate command for copying files: `cp`. This command works just like `mv`, except that it duplicates the original file and leaves it where it is:

```
% cp test.txt my_essay.txt
```

Finally, deleting a file is done using the `rm` command, which stands for remove:

```
% rm test.txt ?
```

Suppose you want to make a new directory to store this file and others. You can do that with the `mkdir` (make directory) command:

```
% mkdir essays/history ?
```

This command creates a subdirectory called `history` inside the `essays` directory. As with most of the other commands you've seen, the arguments can be bare filenames (to refer to a target in the current directory), absolute paths (beginning with the `/` directory and specifying each subsequent step down the tree), or relative paths.

Deleting a directory is a little trickier. The command for deleting a directory is `rmdir`, or remove directory; but it only works on a directory that's empty:

```
% rmdir essays
rmdir: essays: Directory not empty
```

Now, you can go through the contents of your directory and painstakingly delete every single file in it using the `rm` command (or, for more convenience and more risk of things going badly wrong, `rm*`, which matches all filenames in the current directory); or, if you're in a hurry or just want to get things over with, you can use the `rm -rf` command to delete a directory and everything inside it all at once:

```
% rm -rf essays
```

Note: Using `rm -rf` means you won't get any warnings or second chances this way (the `-rf` flag suppresses them); and you'd better be sure that everything inside all the subdirectories of the directory you're deleting is really safe to delete. Generally BSD has no Trash or Recycle Bin equivalent (excluding certain versions).

Viewing Files

In a GUI, you can do just about anything with a click or two of the mouse, be it viewing the contents of a text document or a picture file by double-clicking on it.

BSD, however, does not have this facility built into its architecture. While KDE and GNOME behave the same way as they do on Linux when it comes to file type associations and opener applications, in the CLI environment you have to use other means to see inside your files.

To show the contents of a plain text file, use the `cat` command:

```
% cat my_essay.txt
    This is a test.
Now is the time for all good men to come to.
```

This is great for short files; but what is your file runs into MB? There is a variety of other commands that let you manage these larger files much more efficiently, such as `more`:

```
% more /var/log/maillog ?
```

more is a program rather than a mere command, which lets you move through the contents of a text file page by page, using the space bar to move forward and the W key to move back. Skip to the end by pressing [*Shift*]+[>] or the beginning with [*Shift*]+[<]. You can search for a string of characters by typing / followed by the string, then pressing [*Return*] or [*Enter*]. Press [*Q*] to quit.

Working within the textual shell, particularly through a remote terminal connection (for example over SSH), you're really only going to be able to look at plain text files; images, movies, and even text documents that are stored in a binary format are going to be unviewable unless you're working within a graphical environment like KDE.

However, if you are using the console application of your GUI (such as KDE *Konsole* or GNOME *Terminal*, you can use the xv program to view pictures:

```
% xv Picture1.jpg ?
```

Movies can be viewed using MPlayer:

```
% mplayer Birthday.mov ?
```

Documents can be opened using OpenOffice.org. Indeed, if a program exists for Linux, chances are that it can be compiled and installed on BSD as well, if there isn't a binary package for it already. And if there isn't, BSD's Linux compatibility layer will allow you to run Linux programs natively.

Well, that sums up the description of BSD's command line capabilities. As with any Unix based OS, the real prowess of BSD lies in the CLI and thus, it is always useful to be literate with the console.

SUFYAN BIN UZAYR

Sufyan is a 19-year old freelance writer, graphic artist, programmer and photographer based in India. He writes for several print magazines as well as technology blogs. At present he is speculating the future of graphic art on UNIX platforms. He can be reached at http://www.sufyan.co.nr

Backup your laptop

from anywhere to your home server with openvpn, rsync and ssh

In this article I will explain how to setup a very simple backup system based on standard unix tools and two shell scripts to backup your laptop from anywhere if you have an internet connection available.

What you will learn…

- How to quickly setup a vpn with your laptop with PAM authentication.
- How to setup a client script to securely send your laptop data to a remote server.
- How to rotate backups on the server daily and monthly optimizing disk usage on the server side.

What you should know…

- How to use the FreeBSD ports management system to install openvpn.
- Basic SSH and rsync knowledge.

The system requires a remote server connected to the internet with even a fixed public IP address or a dynamic DNS service such as that provided by DynDNS. Also, we assume you have a Unix based operative system on your laptop.

Overview

Your laptop will connect automatically at startup to your home server via OpenVPN so that you will have direct connection with your backup server in a secure way.Backups are fired from a script on your laptop, so you can opt to run them manually or automatically via your laptop's Cron service. Once the client script is run any file changed since the last backup will be sent to the server with rsync (with optional compression, deletion of local deleted files, bandwidth limitation, files/directories exclusion, etc). On the server side, a script will run nightly and will rotate the backups so that you always will have a snapshot of your laptop for every day of the last month and the first backup of every month since you started to use this system, so, let's assume you have been using this system for a year running daily backups, then you will end up with 11 monthly backups (the first available of every past month) plus the last 30 days backups, a total of 41 snapshots of your files. On the server, rotation is done using hard links. This way, files that don't change aren't copied with every rotation. Rsync takes care of creating a new file on the current snapshot if the contents differ.

In this article I will setup my Macbook Pro (mlaptop) to backup my home directory to my FreeBSD home server (*hellbox.example.com*). My home server runs on my home LAN (network 192.168.1.0/24) and connects over my home adsl router to the internet. This router is setup to keep updated my dynamic IP address with a Dyndns domain (hellbox.example.com) and forwards port 1194/udp to my home server at 192.168.1.3. Also I have Tunnelblick (available at *http://code.google.com/p/tunnelblick/*) installed on my laptop to manage OpenVPN connections.

Preparation

Ensure you have already installed and configured SSH on your home server and check that an updated version of rsync is available on both systems (>3.0 is recommended). In freebsd you can install rsync via the Ports system, it is located at `/usr/ports/net/rsync`. In OSX you can get it via Macports (*sudo port install rsync*). Also, if you don't have a local unix account (different from root) in your home server, create it. We will use it to authenticate your vpn client and ease the setup.

Part 1 – Setting up openvpn on my home server

Openvpn installation was covered on the October issue, so we won't focus on its installation but in its configuration instead. Just install it via the Ports system (you can find

it in /usr/ports/security/openvpn20). Once installed, add the following lines to your /etc/rc.conf:

```
openvpn_enable="YES"
openvpn_configfile="/usr/local/etc/openvpn/server.conf"
```

Create the openvpn configuration file /usr/local/etc/openvpn/server.conf with the following content, read the commends for optional or particular settings: see Listing 1. Now, before starting openvpn one last step is required: Setup the server PKI certificates required by openvpn: see Listing 2.

Copy these three generated files to /usr/local/etc/openvpn/.

Now you should be able to start openvpn with:

```
/usr/local/etc/rc.d/openvpn start
```

Part 2 – Setting openvpn on the client

Copy the ca.crt to your laptop and create a new file named backupsopenvpn.conf with the following contents:

```
client
port 1194
proto udp
dev tun
remote hellbox.example.com
ca ca.crt
auth-user-pass
comp-lzo
```

Now, you can test it by running openvpn with backupsopenvpn.conf as the only argument or, in OSX, by placing it in ~/Library/openvpn and running it throught Tunnelblick GUI. You will be prompted for a username

Listing 1. *Openvpn server configuration file*

```
# Uncomment the following line if you want the process to listen just on one network interface.
#local 192.168.1.3
port 1194
proto udp
dev tun
ca ca.crt
key hellbox.key
cert hellbox.crt
dh dh1024.pem
server 10.0.0.0 255.255.255.0
ifconfig-pool-persist ipp.txt
# The following line makes my home network available to my laptop over the vpn
push "route 192.168.1.0 255.255.255.0"
user nobody
group nobody
persist-key
persist-tun
status openvpn-status.log
verb 3
# The following 3 lines will allow us to authenticate with the vpn through a local
# unix account on my home server - PKI for backup clients is not needed.
plugin /usr/local/lib/openvpn-auth-pam.so common-auth
client-cert-not-required
username-as-common-name
comp-lzo
link-mtu 1542
keepalive 10 60
# Uncomment the following line if you already have another vpn on your laptop (I.e: Work vpn).
#mssfix 1400
```

Listing 2. *Openvpn required certificates generation*

```
openssl req -nodes -new -x509 -keyout ca.key -out ca.crt
openssl req -nodes -new -x509 -keyout ca.key -out
                    ca.crt
Generating a 1024 bit RSA private key
.....++++++
.....................++++++
writing new private key to 'ca.key'
-----
You are about to be asked to enter information that
                    will be incorporated
into your certificate request.
What you are about to enter is what is called a
                    Distinguished Name or a DN.
There are quite a few fields but you can leave some
                    blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Madrid
Locality Name (eg, city) []:Madrid
Organization Name (eg, company) [Internet Widgits Pty
                    Ltd]:Example Corp.

Common Name (eg, YOUR name) []:hellbox
Email Address []:me@example.com

openssl req -nodes -new -keyout hellbox.key -out
                    hellbox.csr
Generating a 1024 bit RSA private key
.....................++++++
..................++++++
writing new private key to 'hellbox.key'
-----
You are about to be asked to enter information that
                    will be incorporated
into your certificate request.
What you are about to enter is what is called a
                    Distinguished Name or a DN.
There are quite a few fields but you can leave some
                    blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Madrid
Locality Name (eg, city) []:Madrid
Organization Name (eg, company) [Internet Widgits Pty
                    Ltd]:Example Corp
Organizational Unit Name (eg, section) []:IT
Common Name (eg, YOUR name) []:hellbox
Email Address []:me@example.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
openssl dhparam -out dh1024.pem 1024
Generating DH parameters, 1024 bit long safe prime,
                    generator 2
This is going to take a long time
.+.........................................++*++*
                    ++*
```

Listing 3. *SSH Private/Public keypair generation*

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/matias/
                    .ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/matias/
                    .ssh/id_rsa.
Your public key has been saved in /Users/matias/.ssh/
                    id_rsa.pub.
The key fingerprint is:
20:f2:99:6e:63:7e:b5:b4:65:79:5f:5a:a3:69:b6:99
                    root@laptop.example.org
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|  . . .      .  .|
|   o + .    o .o+|
|        S  . .++*|

|    . +  o +     |
|     + .. o      |
|      ..         |
```

and a password. Use the local unix account credentials you have on your home server (please, don't use the root account!). Note that tunnelblick may also ask you for your local (laptop) admin account password, don't get confused by the pop ups.

Part 3 – Setup ssh on both sides

This step is required so that the laptop script, when connecting, doesn't hang waiting for you to enter your home server username password.

First, if you don't have done it already (check if you have two files named `id_rsa` and `id_rsa.pub` under `~/.ssh/`), generate a ssh key pair in your laptop. Use default settings and blank password when prompted: see Listing 3.

Copy the new generated file `/Users/matias/.ssh/id_rsa.pub` contents into your server `/home/youraccount/.ssh/authorized_keys`. And finally test that you can connect from your laptop to your server without being asked for your password with:

```
$ ssh youraccount@192.168.1.3
```

If you got successfully to this step, you have to know that you have passed the difficult part.

Part 3 – Setting up the laptop script

Create a script in your laptop's home directory with the following contents: see Listing 4.

You may want to create the file pointed by the variable `BACKUP_EXCLUDES` where you can put a list of directories and files you want to exclude from backups, mine contains something like this:

```
Library/Caches
Downloads
iso
backup.log
.Trashes
```

Listing 4. *Laptop backup script*

```
#!/usr/bin/env bash

BACKUP_ROOT=/Users/matias
BACKUP_EXCLUDES=/Users/matias/backup-excludes.txt
BACKUP_LOG=/Users/matias/backup.log
BACKUP_TARGET="youraccount@192.168.1.3:data/backup/"

EXTRA_OPTS=""
# Uncomment next line for bandwidht limitation to 100KBPS
#EXTRA_OPTS="$EXTRA_OPTS --bwlimit=100"

# Uncomment next line for data compression to save bandwidth (costs a bit more cpu)
#EXTRA_OPTS="$EXTRA_OPTS -z"

# This piece of code checks if there is connectivity with the server before actually running the backup
pid='ps -Ac | egrep -i rsync | awk '{print $1}';'
if [ ! -z "$pid" ]
then
    exit 1
fi


/sbin/ping -oqQt3 192.168.1.3 2>&1 > /dev/null || ( echo "Unable to contact server on 'date'" > $BACKUP_LOG ; exit 1
                    )

rsync $EXTRA_OPTS -ae "/usr/bin/ssh  -o PasswordAuthentication=no -o UserKnownHostsFile=/dev/null -o StrictHostKeyCh
                   ecking=no" --partial --exclude-from $BACKUP_EXCLUDES  --delete $BACKUP_ROOT $BACKUP_TARGET 2>&1 >
                   $BACKUP_LOG && echo "Backup completed on 'date'" >> $BACKUP_LOG
```

Listing 5. *Server backup script*

```
#!/usr/local/bin/bash
BACKUP_STORAGE_ROOT=/home/matias/backups
version='date +%Y%m%d'
thisyear='date +%Y'

cp -al $BACKUP_STORAGE_ROOT/backup "$BACKUP_STORAGE_ROOT/data/backup.$version"
((lastmonth='date +%m'-1));
if ((lastmonth<1));
then
((lastmonth+=12));
fi

then
lastmonth="0$lastmonth";
fi
oldbackups='ls -d $BACKUP_STORAGE_ROOT/data/backup.*|grep $thisyear$lastmonth|tail -n +2'

echo $oldbackups
rm -rf $oldbackups
```

Listing 6. *Adding a crontab job*

```
And add it to youraccount's crontab to run nightly:
# crontab -e

Add a line like the following:
0 0 * * * /home/youraccount/backup-rotate.sh
```

```
.fsevents*
.Spotlight*
```

Now, run the script from the command line to test it and check if there is any error. Finally, add a cron entry to run it daily:

```
$ crontab -e
```

Add a line like the following:

```
0 */6 * * *  /Users/msurdi/backup.sh 2>&1 > /dev/null
```

That will run the backup every 6 hours. Adapt it to your needs. For me it is enough to run it daily.

Part 4 – Server side backup rotation

Create a script named `backup-rotate.sh` into your account's home directory on your server with the following contents: see Listing 5. Congratulations! You are done!

Conclusion

There are many backup tools out there, even my laptop operative system has an integrated backup mechanism (*Time Machine*) but none of the ones I tried satisfied my backup needs. In the particular case of Time Machine (that seemed the best option to me at first) the main issue I found was that as my home directory is encrypted with Filevault it won't be backed up if I'm logged in. So, after all, I ended developing this system that's been working perfectly for over two years and saved my data and me in more than one opportunity. Hope this works for you as it did for me.

MATIAS SURDI

Matias Surdi has been working as a unix systems administrator since his first job, currently working as the IT Lead of a large Spanish Social Network is a passionate of unix based server operative systems, networks management and systems tools development. In his spare time he loves to ride his Triumph motorbike across the beautiful Spanish landscapes and do sports.

FreeBSD Mall

Your FreeBSD & PC-BSD Resource

www.FreeBSDMall.com

FreeBSD 8.1 Jewel Case CD/DVD

Set contains:

- **Disc 1**: Installation Boot (i386)
- **Disc 2**: LiveFS (i386)
- **Disc 3**: Essential Packages (i386)
- **Disc 4**: Essential Packages (i386)

FreeBSD 8.1 CD $39.95
FreeBSD 8.1 DVD $39.95
FreeBSD 7.3 CDROM $39.95
FreeBSD 7.3 DVD $39.95

FreeBSD Subscriptions

Save time and $$$ by subscribing to regular updates of FreeBSD!

FreeBSD Subscription , start with CD 8.1 ..... $29.95
FreeBSD Subscription, start with DVD 8.1 ..... $29.95
FreeBSD Subscription, CD 7.3 ..... $29.95
FreeBSD Subscription, DVD 7.3 ..... $29.95

PC-BSD 8 DVD (Hubble Edition)

PC-BSD 8 DVD $29.95
PC-BSD Subscription $19.95

BSD Magazine

BSD Magazine $11.99
BSD Magazine Subscription $11.99

The FreeBSD Handbook

The FreeBSD Handbook, Volume 1 (User Guide) $39.95
The FreeBSD Handbook, Volume 2 (Admin Guide) $39.95
☆ Special: The FreeBSD Handbook, Volume 2 (Both Volumes) $59.95
☆ Special: The FreeBSD Handbook, Both Volumes, & FreeBSD 8.1 ..... $79.95

The FreeBSD Bundle

Inside the Bundle, you'll find:

- FreeBSD Handbook, 3rd Edition, Users Guide
- FreeBSD Handbook, 3rd Edition, Admin Guide
- FreeBSD 8.1 4-disc set
- FreeBSD Toolkit DVD

☆ Special: The FreeBSD CD Bundle $89.95
☆ Special: The FreeBSD DVD Bundle $89.95

The FreeBSD Toolkit DVD $39.95

FreeBSD Mousepad $10.00

FreeBSD Caps $20.00

PC-BSD Caps $20.00

For *MORE* FreeBSD & PC-BSD items, visit our website at FreeBSDMall.com!

CALL 925.240.6652 Ask about our software bundles!

t-shirts
$18-$21.99

Authenticating NAT with authpf

NAT (Network Address Translation) is a way to (hide) several hosts behind a gateway. It operates on IP packets (layer 3) by rewriting source address of each one. It should be noticed that another mechanism called PAT (Port Address Translation) operates on TCP/UDP ports (layer 4).

What you will learn…

- Authenticate outgoing NAT
- Authpf
- Linking Authpf to LDAP

What you should know…

- Basic knowledge of LDAP
- What is NAT
- Make an SSH connexion

There are two kinds of NAT: unidirectional (simple NAT) and bidirectional (BiNAT). The simple NAT allow hosts from internal network to talk with foreign peers. The gateway maintains an association table to enable replies from foreigners reaching the internal host that initiated connexion. The BiNAT maintains a 1:1 for each internal host translated making it possible for foreigners to initiate connexions on translation IP address.

NAT Issues

IP has been designed as a host-to-host protocol. It means that the source IP of a packet should be the (*real*) IP of the host that initiated the packet. If host is behind a NAT, source IP is the translation address. For some protocols (ie Kerberos, IPSec etc.), it is a major issue. Moreover, from the IT security officer NAT may stand for Non Authenticated Transmission because multiple hosts share the same IP. It makes it difficult to find out which host behind NAT had a bad behavior with remote resources (scan, talking to a botnet ...). Last, on many gateways that performed NAT, rules are defined for IP address (host, range or network) and not for a user. It means that for roaming NAT user, each potential network used must be translated. In addition, rules are permanently active in gateway (even if nobody needs NAT, it is performed). As a result, an important rule set must be maintained covering each potential NAT use whereas a restricted subset of people really needs NAT (the rank and file of humanity is happy with a squid proxy).

Authpf

Authpf is a shell shipped with *BSD firewall PF (*Packet Filter*). It makes it possible to trigger actions on the firewall rule set at SSH connexion on the gateway. Authpf is mainly used as a topnotch port knocking tool (or, more hype, SPA – *Single Packet Authentication*) to open a way to perform any kind of remote maintenance by identified users with minimum client tools (a simple SSH client). A way to use it to (partially) authenticate NAT with a LDAP directory will be discussed.

Setting up the gateway

Basic PF setup

Supplied configuration has been performed on OpenBSD 4.6. PF is already compiled in generic kernels. It only needs to be enabled through `rc.conf.local`:

```
pf=YES
pf_rules=/etc/pf.conf.local
```

The directive `pf_rules` specify an alternate configuration file to standard `pf.conf`. I like to do it this way to prevent loss of `pf.conf` at upgrade. PF anchors for authpf must be placed

in the rule set. Anchors can be considered as subsets of tables and rules (NAT, RDR, BINAT and filtering) which can be dynamically loaded through pfctl. In `pf.conf.local`:

```
nat-anchor „authpf/*”
rdr-anchor „authpf/*”
binat-anchor „authpf/*”

anchor „authpf/*”
```

Anchors to load authpf rules are added. The sequence is important (nat, rdr, binat, rules) pay attention to put the anchor in adequate section (for example, nat-anchor in NAT section of your `pf.conf.local`). If you misunderstand this recommendation, check manpage of `pf.conf` to see howto organize sections of `pf.conf`. Those anchors are used to launch specific sub rule set per user.

Users management

In many organizations, a LDAP is used to store each user. A very minimum entry for a user contains an uid and a userPassword hashed with SSHA. Each user must be able to perform a simple bind against the directory service. In our configuration, we would like to link authenticated NAT with our LDAP.

OpenBSD can handle LDAP in two ways: with a NIS-like operating mode called ypldap (it means that yp* operations are performed against an LDAP backend instead of pushing maps to each hosts). Ypldap requires users to be stored as PosixAccount wich is not convenient for our needs. The second way is to only use `login_ldap` to perform simple bind with accounts defined on the gateway. The first method suits well for large organization which have posixAccount defined for each user planning to make massive use of NAT. The second is more flexible. We will discuss the second. In `/etc/login.conf`, a new authentication method `ldap-authpf` is defined:

```
ldap-authpf:\
        :auth=-ldap:\
        :x-ldap-server=myldap.mydomain:\
        :x-ldap-basedn=ou=People,dc=mydomain:\
        :x-ldap-binddn=cn=toto,ou=user4bind,dc=mydomain:\
        :x-ldap-bindpw=azerty:\
        :x-ldap-filter=(&(objectClass=inetOrgPerson)(uid=%u)):\
        :welcome=/etc/motd.authpf:\
        :shell=/usr/sbin/authpf:\
        :tc=default:
```

It enables a LDAP server myldap.mydomain accessible trough a binddn (`cn=toto, ou=user4bind, dc=mydomain`) with azerty as password (who really cares about security?). A filter is supplied to select only inetOrgPerson (a very minimal subset of attributes commonly used when you store people in LDAP) on uid attribute. Shell is forced at `/usr/sbin/authpf` value. Now, users must be created with useradd command:

```
useradd -L ldap-authpf -d /var/empty -s /usr/sbin/authpf
                    -r 1500..3000 -g =uid zaza
```

This command `create` user zaza with a non existent home (`/var/empty`), authpf shell and login method ldap-authpf (the one previously created in `login.conf`). A UID between 1500 and 3000 is affected as well as a dedicated group (or UPG: *User private Group* – see RedHat documentation for details).

Per user ruleset configuration

Specific rules for user must be added in `/etc/authpf`. As an example `/etc/authpf/users/zaza/authpf.rules` contains:

```
int_if=bge0

nat pass log on $int_if from $user_ip to any -> XXX.XXX.XXX.XXX
```

This simple rule adds a NAT translation rule when zaza connects trough SSH on the gateway. `$user_ip` is a variable maintained by authpf which contains the source IP that performed SSH connexion. It means that as long as SSH connexion is opened, source address of each packet originating from `$user_ip` and going through the gateway is rewrited to `XXX.XXX.XXX.XXX`. Logs of SSH connections are stored in authlog file.

It should be highlighted that an IP alias for `XXX.XXX.XXX.XXX` must be defined on outgoing interface to enable packet reception from remote to internal client. In rc.local:

```
ifconfig bge1 inet alias XXX.XXX.XXX.XXX netmask 255.255.255.0
```

Conclusion

We addressed the issue of roaming users thanks to `$user_ip` variable. We also addressed the problem of permanent NAT access (NAT is only active during SSH session). Finally, we also increased quality of network auditing access process. NAT access is logged for a user and not for an IP. Indeed, this solution is not perfect because NAT always operates trusting a source IP (and not a user) and it's very easy to spoof an IP on the same LAN segment.

NICOLAS GRENECHE

Nicolas Greneche (nicolas.greneche@univ-orleans.fr) – [Free|Open]BSD addicted Technical blog (in french): http://blog.garnett.fr

Xmodmap

on the way to writing hieroglyphs quickly

This article is about how to make your own Xmodmap map – a definition for a keyboard layout in Linux/Unix.

What you will learn…

- You will learn how to write exotic characters with Xmodmap

What you should know…

- Xmodmap works in any Unix-like environment

Presented is a sample Xmodmap keyboard map with four keyboard layouts that users can toggle with by Caps Lock: 1) Standard English keyboard; 2) IAST keyboard layout for transliteration of Sanskrit; 3) a layout to be defined by users; 4) keyboard layout for Devanagari.

XKB and Xmodmap

First, I must say that Linux (Unix) uses two approaches to configure the keyboard layout (they are both independent of each other): XKB and Xmodmap. XKB is an extension of X, many people say that it is better, but too robust and perhaps less understandable by beginners. Xmodmap is one of the oldest ways how to configure your keyboard layouts – a little easier approach, especially good for experimentation, but no only that.

XKB, too, allows a number of easy solutions, but if we look at hundreds of Linux distributions and Window Managers including BSD, Xmodmap solution basically works the same way everywhere – whether on PC-BSD, SuSE Linux, Debian Linux, Slackware Linux, etc. The same can be said about XKB, too, but only from the system perspective. In the comfortable environment like GNOME/KDE, things are always a little easier than in other WM's. This is not a problem for experienced users, but for beginners it may sometime be a difficult task to find a way how to type some special – that is, very exotic characters.

But let us forget all these technical discussions on how the above two approaches differ from each other – for now it is important to know that Xmodmap does not require any additional editing of `/etc/X11/xorg.conf` file; the user will just load the Xmodmap map by running the command: `xmodmap /home/use /xmodmap.map`. It will work on every X server. The only condition to successfully type any special character on your Linux/Unix box is to use the UTF encoding.

The Xmodmap utility is a command (xmodmap) that will load the xmodmap map (a text file with definitions for keyboard layouts). You can thus assign a number of characters to your keyboard keys and redesign your keyboard layout the way that is suitable for you. After you restart your X, you will be again working with the XKB approach, which is the default setup for Linux/Unix today. If you do not restart your X, you may reload another Xmodmap map.

Xmodmap

Xmodmap approach is perhaps a little easier than XKB – especially when talking about Unix in global, because if you would like to learn Devanagari (Indian script used for writing), all you need is to have the .Xmodmap file in your home directory (the *dot* means that the file is a system file

and that it will be loaded automatically) with the following contents only: `keysym k = U0915 k`

Linux (X) should load the .Xmodmap configuration file automatically and you, after pressing the letter `k`, should see the character क. `U0915 k` means that when you press a key with this definition (the letter `k` on the English keyboard), you will see the sign क (its UTF code is U0915) and after applying the shift key you will see our Latin character `K`. If you want a capital character of any available choice, write it capitalized into the Xmodmap map: U0915 K.

Do you want to remap a letter (or more) on your keyboard? Any code you put into the .Xmodmap file will be available for you after your X Window system starts, or even after you load your Xmodmap map manually – it can be additionally loaded by running the following command from the Terminal Window in your X environment: xmodmap `/home/user/xmodmap.map`

Transliteration – what is it and why is it used?

It is the conversion of one script (a writing system) from one writing system to another one, which can also render the phonetics; the term transcription is also used. In our cultural environment one writing system is mostly the Latin alphabet (romanization), when scholars (in scientific disciplines such as lexicography, etc.) use one system of transliteration, for example, Romaji – romanization of Japanese, or IAST for Sanskrit (*International Alphabet of Sanskrit Transliteration*).

For systems of transcription such as IAST, which also renders the phonetics, we must use special characters such as a ā Ā ī Ṛ ṇ Ṇ ṣ Ṣ ṁ, so our standard Latin alphabet is not anymore sufficient.

It is also important to say that Sanskrit as a language is different from Hindi and both languages can be written into the Devanagari script. In India, there are many languages, and in addition to Hindi or Konkani, for example, the Devanagari script is also used in Nepal. For the reader to understand this – Sanskrit as a language can also be written in Tibetan scripts, the Tamil script, or even in the Siddham script, which the Shingon Buddhist school uses in Japan.

DEVANAGARI LETTER A	अ	Option+0905
DEVANAGARI LETTER AA	आ	Option+0906
DEVANAGARI LETTER I	इ	Option+0907
DEVANAGARI LETTER II	ई	Option+0908
DEVANAGARI LETTER U	उ	Option+0909
DEVANAGARI LETTER UU	ऊ	Option+090A
DEVANAGARI LETTER VOCALIC R	ऋ	Option+090B
DEVANAGARI LETTER VOCALIC L	ऌ	Option+090C
DEVANAGARI LETTER CANDRA E	ऍ	Option+090D
DEVANAGARI LETTER SHORT E	ऎ	Option+090E

Figure 1. *UTF codes for Devanagari*

Researchers often work with transliteration (as well as with tools that automate the conversion process to Latin and vice versa) for various reasons. It is more convenient for people of a different cultural background – especially for Europeans and Americans. And if you are not familiar with some exotic languages, you can at least read them, so in case someone starts a research project aimed at mapping similar words between Indian and Slavic languages, for example, no one can deny the significance of transliteration and no time needs to be wasted by learning the complex and difficult *hieroglyphic* letters.

Keyboard map for IAST transliteration and the Devanagari script

On the basis of this Xmodmap map any researcher or student can build his or her own keyboard layout. This Xmodmap map provides four keyboard layouts and you may toggle them by Caps Lock: 1) Standard (English) Keyboard (after you load the Xmodmap map with the command: xmodmap `/home/user/xmodmap.map`); 2) keyboard for IAST transliteration – Sanskrit; 3) user-defined keyboard only with `p p` strings to be later changed; 4) Devanagari – keyboard layout with the script to write Sanskrit and other Indian languages including Hindi.

If you press the Caps Lock key while working with the last keyboard (Devanagari) layout, you will return to the first (Standard English) keyboard.

This Xmodmap map requires UTF codes

However, if you need other languages such as Chinese, Telugu, Urdu, etc. – either to use the original writing

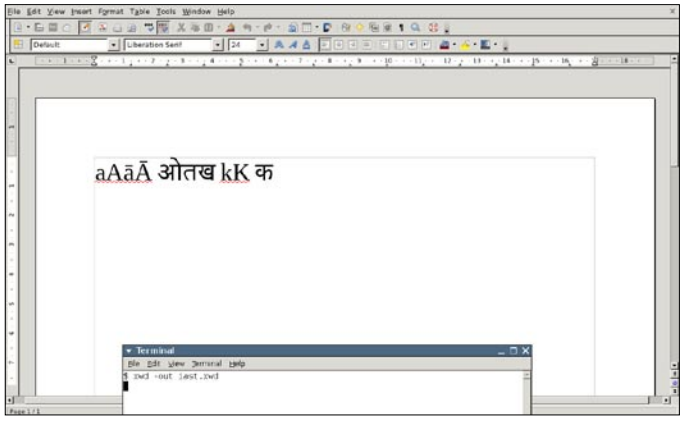


Figure 2. *Run xmodmap xmodmap.map and learn how to write in any language*

system of such languages, or their transliterated form (in Latin), you should know something about UTF codes (that you must write into this Xmodmap map).

To type a variety of special characters, you need to have their UTF codes – use search engines, as they provide very rich resources. You can also alternatively look into the file `/usr/include/X11/keysymdef.h` in Linux, or (`/usr/local/include/X11/keysymdef.h` in FreeBSD). The Latin letter a (with a macron – a horizontal line above it) is defined in `keysymdef.h` as follows:

```
# define XK_amacron 0x03e0 / * U 0101 LATIN SMALL LETTER A
```

`U101` is the Unicode code for the character `ā`.

If the keyboard map is not loaded automatically, or even in case one instance of it has already been loaded, you may load any other Xmodmap map and use any special characters defined in it: `xmodmap /home/user/xmodmap.map2`

The previously loaded keyboard layout will be deactivated and you will work with the keyboard layout map you have just loaded.

The Xmodmap map presented by this article allows 8 letters to be assigned to one keyboard key. As it allows toggling between four keyboards, I will explain one line of the definitions of keys in it:

```
keycode 0x1F = i I U012B U012A p p U0917 U0918
```

`keycode 0x1F` is a key that has standardly the letter `i` `I` assigned to it on our English keyboard. You may also use xkeycaps (xev, too) to find out which letter is assigned to which key – xkeycaps is a program

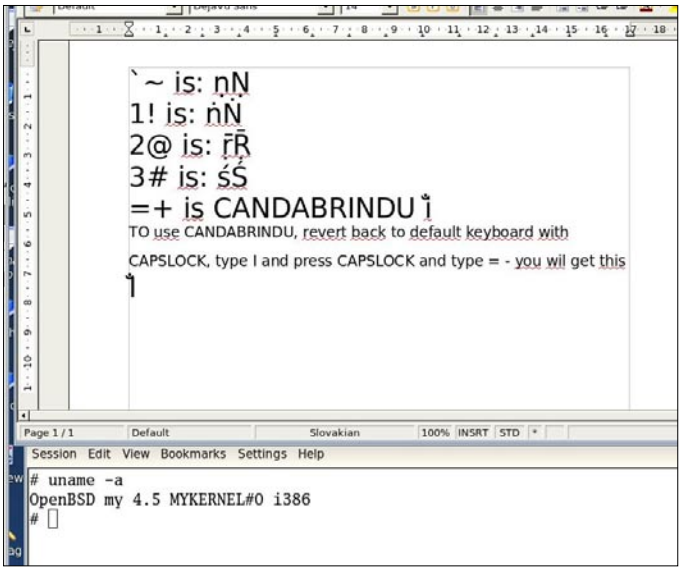


Figure 3. *Working with OpenOffice in OpenBSD becomes a joy*

that tells you everything about how characters are assigned to your (or any) keyboard. Thus, if you load this Xmodmap map, you will get the standard English keyboard layout at first. After pressing Caps Lock one time, you will toggle the keyboard to IAST transliteration – you will see the Latin small letter `ī` (with macron) when you press the letter `i`; after pressing the Shift key, you will see the Latin capitalized letter I (with macron). This also pertains to the letters like a, m, t, so try this out – `ā Ā ṣ Ṣ ṁ`, etc. Pressing Caps Lock again will toggle you to a user-defined keyboard layout that assigns `p p` characters to keys (not to all of them). The last time you press Caps Lock will allow you to write in the Devanagari script – after pressing the key `i`, you will see the sign `ः`.

Any available UTF codes you use in this Xmodmap map will appear on your screen in a good text editor (tested in gedit and *OpenOffice.org*) when you type them; however, some very special characters also require special fonts to be installed, as, of course, hardly any distributor of Linux expects that users want to write in Egyptian hieroglyphs.

Some "techie" information related to Xmodmap

This map was tested on my Debian, OpenBSD, Ubuntu, and FreeBSD box. I must say that in GNOME/KDE the third keyboard map (`p p`) may not work probably due to the reason that the GNOME/KDE desktop has its own system for localization. But after I uninstalled the Slovak Keyboard (*GNOME>Preferences>Keyboard*), all four keyboard layouts worked properly with this Xmodmap map.

In the worst case you need to press the Caps Lock key only twice (one time for the romanization of Sanskrit and second time for the Devanagari script); if you press it the third time, you are back in the first – English keyboard.

Download the Xmodmap map with four keyboard layouts

LINK

Some notes how to use the keyboard layout with IAST transliteration

IAST is the International Alphabet for Sanskrit Transliteration, but it can also be used for transliteration of Devanagari, thus also languages such as Hindi and many others.

This Xmodmap map is a sample map and its primary purpose is to let users redefine their keyboards according to their needs. The keyboard map for IAST

(after loading the map, you press the Caps Lock only once) corresponds to the English keyboard (which is the first keyboard layout) and it will allow you to type the following special (Latin) letters (*First* is the English keyboard layout, which is activated as soon as this Xmodmap map is loaded; for it to work, the Caps Lock key does not need to be pressed):

```
First: aAdDrRtTuUiIsShHlLnN
IAST: āĀḍḌḍ̦Ḍ̦ṛṚṛ̥Ṛ̥ṭṬṭ̦Ṭ̦ūŪīĪṣṢṣ̱Ṣ̱ḥḤḥ̱Ḥ̱ḷḶñÑ
```

As the letter `n N` has several more alternatives (ṅ Ṅ ñ Ñ), the other instances of the letter `N` can be typed by using the apostrophe/tilde key and then the keys that start with and follow the number 1 on the standard keyboard layout:

```
Apostrophe / tilde (`~) corresponds to sign  ṇ Ṇ
Number 1 corresponds to: ṅ Ṅ
Number 2 corresponds to: ṉ Ṉ
Number 3 corresponds to:  ś Ś
Number 4 corresponds to: ḷ̦ Ḷ̦
```

Users can type the Candrabindu sign in the IAST keyboard layout by pressing the key 0x15 on their keyboard (where the English keyboard has the equal sign combined with the plus sign): ̐.

Some lines in this Xmodmap map, such as the `keycode 0x15 = equal plus U0310 plus`, which contain only four items, can be extended to eight items.

Even though we work with the Latin alphabet, systems of transliteration vary. But users will be able to make their own keyboard layout and thus also any system of romanization used for transliteration of languages.

JURAJ SIPOS

Juraj lives in Slovakia, where he works in a library (in an educational institute). He has been writing and selling computer articles for over ten years. He wrote an xmodmap howto (www.faqs.org/docs/Linux-mini/Intkeyb.html) and in addition to computers he is also interested in spirituality, but not really the guru side of things, but more-so freedom and self-actualization. His website says more: www.freebsd.nfo.sk

FreeBSD Binary Upgrade

(with packages)

After We install FreeBSD system, we have fresh packages and up to date base system, but as new RELEASE appears its good to update to get new features and bugfixes.

What you will learn...

- After reading/completing this HOWTO You will know how to upgrade both the FreeBSD's base system and packages using binary packages without needless compilation process in an easy and mostly automated way.

What you should know...

- You should have knowledge about upgrading/updating FreeBSD's base system (check man freebsd-update), adding/removing packages (check man pkg_info/man pkg_add/man pkg_delete).

After FreeBSD system installation, we have fresh packages and up to date base system, but as next FreeBSD RELEASE appears its good to update to get new features and bugfixes. This process is often connected with lots of compiling and *wasted* time, especially for Ports infrastructe since FreeBSD's base system can be easyly updated binary way with `freebsd-update`. This guide shows how to update all installed packages with base system without needless, resources and time consuming process, using the fact, that with every new release we have also prebuilt binary packages ready to install.

Currently both legacy and main branches of FreeBSD will get release updates soon, 7.4-RELEASE and 8.2-RELEASE to be precise, so You will be *preparred* for upgrade as new version arrive ;)

Configuration Backup

No matter how simple the upgrade will be, according to Murphy Laws (*http://en.wikipedia.org/wiki/Murphy_law*), *if anything can go wrong, it will go wrong*, so just in case lets backup the current configuration with simple `tar(1)` to make archive of current `/etc` and `/usr/local/etc` directories.

```
# tar -czf /root/ETC.tar.gz /etc /usr/local/etc
```

Base System Upgrade

Upgrade if the base system is relatively easy, its about to type 4 commands, read and understand on screen comunicates, lets assume for example that we want to update do 8.1-RELEASE version.

```
# freebsd-update upgrade -r 8.1-RELEASE
# freebsd-update install
# shutdown -r now
# freebsd-update install
```

While doing the base system procedure we will be asked for installed *datasets* and about modifications in config files, You have backuped up Your configuration on step Configuration Backup. (You did right?) so You can safely say *y* here.

Packages Upgrade

If it goes to packages, we will use the fact, that all packages are build to newest version along with newest *-RELEASE version. Instructions below include gathering currently installed packages, wiping all of them and the adding them again, but from newer RELEASE build.

```
# pkg_info -qoa | sort > /root/pkg_list.OLD
# pkg_delete -a -f
# rm -r -f /boot/modules /usr/local /var/db/pkg
# while read PKG; do pkg_add -r $( basename ${PKG} ); done
                    < /root/pkg_list.OLD
```

At this point all old packages have been removed, and latest pacakges have been installed on their place, it works

automatically, unless port nam has changed, like some time with VirtualBox, when its port name changed from `emulators/virtualbox` into `emulators/virtualbox-ose`, so the only risk here, is that some packages will not be added automatically.

Searching for Missed Packages

Now we will get list of just installed aplications, and compare it to the list of old packages before update, its quick and easy way to add (eventually) missing packages.

```
# pkg_info -qoa | sort > /root/pkg_list.NEW
# diff /root/pkg_list.* | egrep „^(<|>)”
```

This will give You the list like that one below, the list of ports/packages that You need to check, what has changed, maybe there are no packages for that port (like `lame` because of patents).

```
< devel/bison
> graphics/cairo
< devel/cmake
> security/clusterssh
< textproc/docbook-410
< x11/ecore-x11
```

Cleanup

```
# rm -r -f /root/pkg_list.*
# rm -r -f /root/ETC.tar.gz
```

If we had made some bigger changes in the configuration, we may keep the old configuration, which is archived in the `/root/ETC.tar.gz` file.

Voila! You system is now up to date, both the FreeBSD's base system and packages.

SŁAWOMIR WOJTCZAK (VERMADEN)

Slawomir Wojtczak (vermaden) is just another busy sysadmin, that tries to use FreeBSD as desktop/workstation to overtake/seize many complex mechanisms like IBM TSM, Oracle databases and various UNIX/Linux variants that run these services. Deep K.I.S.S principle follower. Very interested in UNIX technologies (even these proprietary), but often towards BSD solutions, active forum troll at many BSD, UNIX and Linux forums. Feel free to spam him at: vermaden@gmx.com

Introduction to WebDAV

WebDAV is an extension of the HTTP protocol that performs remote Web content magagement, thus turning the Web into writable media.

What you will learn…
- The basic working of WebDAV

What you should know…
- Python programming; Apache administration

What is WebDAV

WebDAV stands for *Web-based Distributed Authoring and Versioning*. Authoring means creating, updating and managing content located on a Web server; distributed means that a resource can have several authors; and versioning means that different revisions of a resource can exist. Simply said, it is a protocol for publishing documents to a Web server. It is officially defined by RFC 4918 obsoleting RFC 2518. Versioning is excluded from RFC 2518 and RFC 4918 and is defined by RFC 3253. We will not discuss it in this article.

WebDAV is an extension of HTTP and the same *media* that is used to read Web content is used to write that content. All existing HTTP infrastructure can be reused

Listing 1. *Initial httpd configuration*

```
ServerRoot "/opt/local/apache2"
Listen 80

User www
Group www

LoadModule autoindex_module modules/mod_autoindex.so
LoadModule dir_module modules/mod_dir.so
LoadModule log_config_module modules/mod_log_config.so
LoadModule mime_module modules/mod_mime.so

DocumentRoot "/opt/local/apache2/htdocs"

DefaultType text/plain
TypesConfig conf/mime.types
DirectoryIndex index.html

ErrorLog "logs/error.log"
LogLevel warn
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog "logs/access.log" common

<Directory "/opt/local/apache2/htdocs">
    Options Indexes FollowSymLinks
    AllowOverride None
</Directory>

<VirtualHost localhost:80>
  DocumentRoot "/opt/local/apache2/htdocs"
</VirtualHost>
```

Listing 2. *Basic authentication of the DAV directory*

```
<Directory "/opt/local/apache2/htdocs/davtest">
    DAV On

    AuthName "DAV dir"
    AuthType Basic
    AuthUserFile /opt/local/apache2/conf/davusers

    <LimitExcept GET>
        Require davuser
    </LimitExcept>
</Directory>
```

Listing 3. *OPTIONS request and response*

```
#!/usr/bin/env python

import base64
import getpass
import httplib
import logging
import logging.config
import sys
import urlparse


logging.config.fileConfig('logging.config')
logger = logging.getLogger(sys.argv[0])


def send_request(conn, urlcomps, user=None,
                 password=None):
    headers = {'Host': urlcomps.hostname,
               'User-Agent': sys.argv[0]}
    if user and password:
        auth = base64.b64encode("%s:%s" % (user,
                 password))
        headers['Authorization'] = 'Basic %s' % auth
    logger.debug('Request headers: %s' % headers)
    conn.request('OPTIONS', urlcomps.path, None,
                 headers)
    response = conn.getresponse()
    return response


def handle_response(response):
    logger.debug('Response status: %s %s' %
                 (response.status, response.reason))
    logger.debug('Response headers: %s',
                 response.getheaders())
    logger.debug('Response body: %s', response.read())
    if response.status != httplib.OK:
        logger.error('Status line: %d %s' %
                     (response.status,
                     response.reason))
        sys.exit(2)

    davHeader = response.getheader('DAV')
    if davHeader:
        logger.info('DAV header: %s' % davHeader)
    allowed = response.getheader('Allow')
    logger.info('Allowed methods: %s' % allowed)


def main():
    from optparse import OptionParser

    usage = '%prog [options] <url>'
    parser = OptionParser(usage = usage)
    parser.add_option('-u', '--user', dest='user',


    (options, args) = parser.parse_args()
    if len(args) != 1:
        logger.error('Please provide a url')
        sys.exit(1)
    url = args[0];

    password = None
    if options.user:
        password = getpass.getpass('Password: ')

    urlcomps = urlparse.urlparse(url)
    host = urlcomps.hostname
    port = urlcomps.port

    logger.info('Sending OPTIONS request to %s' % url)
    conn = httplib.HTTPConnection(host, port)
    response = send_request(conn, urlcomps,
                            user=options.user,
                    password=password)
    handle_response(response)
    conn.close()


if __name__ == '__main__':
    main()
```

and no additional protocols are required to update a site. Let us briefly compare it with ftp and scp as tools that can transfer files to a web server. Almost every platform has them or can have them at no price – the cost of obtaining and installing is not an argument favoring WebDAV. They need to be configured, however. Imagine the resource is located at *http://myserver/mysite/*. It is not easy to conclude from the url the address of the ftp or sshd server, its port and the location of the site on the filesystem. The ftp or ssh access should also be protected and it is well outside of the realm of the web server where the content lives. In the case of WebDAV knowing the url of the resource is enough to update the resource. The Web server should protect the write-access to the resource, but it requires only configuration of the Web server software.

Listing 4. *DAV and Allowed Headers of WebDAV resource*

```
$ ./davoptions.py -u davuser http://localhost/davtest/
Password:
Sending OPTIONS request to http://localhost/davtest/
DAV header: 1,2, <http://apache.org/dav/propset/fs/1>
Allowed methods: OPTIONS,GET,HEAD,POST,DELETE,TRACE,
    PROPFIND,PROPPATCH,COPY,MOVE,LOCK,UNLOCK
```

Listing 5. *PUT request*

```
def send_request(conn, urlcomps, user, password,
                 filename,
                 contenttype=None,
                 locktoken=None):
    headers = {'Host' : urlcomps.hostname,
               'User-Agent' : sys.argv[0]}
    auth = base64.b64encode('%s:%s' % (user,
                    password))
    headers['Authorization'] = 'Basic %s' % auth
    if locktoken:
        headers['If'] = '<%s> (<%s>)' %
                    (urlcomps.geturl(), locktoken)

    f = open(filename, 'r')
    body = f.read()
    f.close()
    logger.debug('Request headers: %s' % headers)
    logger.debug('Request body: %s' % body)

    conn.request('PUT', urlcomps.path, body, headers)
    response = conn.getresponse()
    return response
```

WebDAV provides an ability to lock a resource thus preventing authors from overwriting their changes. It can also assign metadata to content – for example who is the owner of a file, who made the last change, etc.

WebDAV adds some new HTTP methods, headers and status codes and redefines some existing ones.

How WebDAV extends HTTP

WebDAV modifies the `OPTIONS` method to list the WebDAV capabilities of the server.

The original HTTP protocol defines `PUT` and `DELETE` methods – a `PUT` request creates a resource and a `DELETE` one removes it. WebDAV modifies PUT and DELETE to support locking – if an author locks a resource, then depending on the type of lock no other author can change it. It changes `DELETE` to support removal of collections or directories.

Listing 6. *PUT response*

```
def handle_response(response):
    if response.status == httplib.CREATED:
        logger.info('The resource was created')
    elif response.status == httplib.NO_CONTENT:
        logger.info('The resource was replaced')
    else:
        logger.error('Failed to put the resource;
                    status %s' %
                     response.status)
```

Listing 7. *MKCOL request*

```
def send_request(conn, urlcomps, user, password,
                 locktoken=None):
    headers = {'Host' : urlcomps.hostname,
               'User-Agent' : sys.argv[0]}
    auth = base64.b64encode('%s:%s' % (user,
                    password))
    headers['Authorization'] = 'Basic %s' % auth
    if locktoken:
        headers['If'] = '<%s>' % locktoken
    body = None
    logger.debug('Request headers: %s' % headers)
    logger.debug('Request body: %s' % body)

    conn.request('MKCOL', urlcomps.path, body,
                    headers)
    response = conn.getresponse()
    return response
```

The new methods defined by WebDAV are:

- **MKCOL** Creates a collection (directory) on the server;
- **COPY** Copies a resource or a collection of resources to a given destination;
- **MOVE** Moves a resource or a collection of resources to a given destination;
- **LOCK** Locks a resource or multiple resources. Locking is used to alleviate the problem of overwriting a previous update, although it does not completely eliminate it;
- **UNLOCK** Unlocks a previously locked resource.
- **PROPPATCH** Sets one or more properties on one or more resources, i.e. it associates metadata to resources;
- **PROPFIND** Retrieves the properties (metadata) of a resource; can be used to retrieve directory listings.

The new headers are:

- **DAV** Describes the WebDAV support of the server as levels of compliance; used in the response of an `OPTIONS` request;
- **Depth** Controls the level of hierarchy to which a request sent to a collection will be applied. A value of *0* means that only the collection itself is affected; *1* means that the collection and its files but not the subcollections are affected; if infinity – the collection, its files and its subcollections recursively are affected. Some Web servers can refuse requests with *Depth: infinity* for performance reasons;
- **Timeout** Specifies a desired timeout of a lock in a `LOCK` request; the server may not honor it and can return a different timeout in the response;
- **Destination** Specifies the destination to which a resource will be copied or moved in a `COPY` or `MOVE` request;
- **Overwrite** Specifies if the destination should be overwritten in a `COPY` or `MOVE` request.

Listing 8. *MKCOL response*

```
def handle_response(response):
    if response.status == httplib.CREATED:
        logger.info('Created the collection')
    elif response.status == httplib.METHOD_NOT_ALLOWED:
        logger.error('The resource already exists')
    elif response.status == httplib.CONFLICT:
        logger.error('The parent collection does not exist')
    elif response.status == httplib.MULTI_STATUS:
        parse_xml_body(data)
    else:
        logger.error('Failed to created the collection; status %s' %
                     response.status)

def parse_xml_body(data):
    dom = parseString(data)
    multistatusel = dom.documentElement
    responses = multistatusel.getElementsByTagNameNS('DAV:', 'response')
    for responseel in responses:
        hrefel = responseel.getElementsByTagNameNS('DAV:', 'href')[0]
        href = hrefel.firstChild.data
        statusel = responseel.getElementsByTagNameNS('DAV:', 'status')[0]
        status = statusel.firstChild.data
        descel = responseel.getElementsByTagNameNS('DAV:', 'responsedescription')[0]
        desc = descel.firstChild.data
        logger.info('Status for %s: %s - %s' % (href, status, desc))
```

WebDAV introduces the following status codes:

- **207 Multi-Status** Used in responses requests that act on multiple resources and may have different statuses for each resource;
- **422 Unprocessable Entity** Can be returned in a response for any method, when the server does not understand the XML body sent in the request; can happen when the XML body is well-formed but semantically incorrect;
- **423 Locked** When returned as a response of a `LOCK` request, it means that the resource is already locked; when returned by another request, it means that the destination is locked and the user has not provided the lock token for it;
- **424 Failed** Dependency Means that the method cannot be executed, because the requested action depends on another action that fails;
- **507 Insufficient Storage** Used in `PROPPATCH`, `COPY`, `MKCOL`, when there is not enough free space to create a property, copy a resource or make a collection.

Listing 9. *LOCK request*

```
def send_request(conn, urlcomps, user, password,
                 timeout=120, depth='infinity',
                 locktoken='None'):
    headers = {'Host' : urlcomps.hostname,
               'User-Agent' : sys.argv[0],
               'Timeout' : 120,
               'Depth':depth}
    auth = base64.b64encode('%s:%s' % (user, password))
    headers['Authorization'] = 'Basic %s' % auth
    body = None
    if locktoken:
        headers['If'] = '<%s>' % locktoken
    else:
        body = construct_body(user)
    logger.debug('Request headers: %s' % headers)
    logger.debug('Request body: %s' % body)

    conn.request('LOCK', urlcomps.path, body, headers)
    response = conn.getresponse()
    return response

def construct_body(user):
    dom = getDOMImplementation()
    bodydoc = dom.createDocument('DAV:', 'D:lockinfo', None)
    lockinfoel = bodydoc.documentElement
    lockinfoel.setAttribute('xmlns:D', 'DAV:')
    lockscopeel = bodydoc.createElementNS('DAV:', 'D:lockscope')
    lockinfoel.appendChild(lockscopeel)
    exclusiveel = bodydoc.createElementNS('DAV:', 'D:exclusive')
    lockscopeel.appendChild(exclusiveel)
    locktypeel = bodydoc.createElementNS('DAV:', 'D:locktype')
    lockinfoel.appendChild(locktypeel)
    writeel = bodydoc.createElementNS('DAV:', 'D:write')
    locktypeel.appendChild(writeel)
    ownerel = bodydoc.createElementNS('DAV:', 'D:owner')
    lockinfoel.appendChild(ownerel)
    ownertext = bodydoc.createTextNode(user)
    ownerel.appendChild(ownertext)
    body = bodydoc.toxml('UTF-8')
    return body
```

A WebDAV demo

We will illustrate how to set up a DAV server and how to interact with it with a DAV client. The WebDAV support in the web servers varies – Apache Httpd with `mod_dav` has full support, Microsoft IIS as well. Nginx has partial support – it implements only `PUT`, `MKCOL`, `DELETE`, `COPY` and `MOVE`. Lighttpd's lock support is experimental. Most modern operating systems provide built-in WebDAV clients. We will use Apache Httpd in this article and cadaver as a client.

Apache Setup

We start with the following simple configuration file and we gradually add DAV support in it (see Listing 1). The modules `mod_dav` and `mod_dav_so` provide the `DAV` support and we load them:

```
LoadModule dav_fs_module modules/mod_dav_fs.so
LoadModule dir_module modules/mod_dir.so
```

We provide the location of the database where resource locks are stored:

```
DAVLockDB /opt/local/var/run/httpd/davlockdb
```

We specify the minimum timeout of the locks:

```
DAVMinTimeout 240
```

We also need to enable DAV on some directory:

```
<Directory "/opt/local/apache2/htdocs/davtest">
    DAV On
</Directory>
```

It means the url *http://localhost/davtest/* can accept DAV requests and they modify the contents of `/opt/local/apache2/htdocs/davtest`. A web sever's process will perform those modifications so it should have write permissions to that directory:

```
# mkdir -m 770 /opt/local/apache2/htdocs/davtest
# chown www:www /opt/local/apache2/htdocs/davtest
```

Same applies for the lock database:

```
# mkdir -m 770 /opt/local/var/run/httpd
# chown www:www /opt/local/var/run/httpd
```

Listing 10. *Locking a resource*

```
$ ./davlock.py -u davuser http://localhost/davtest/
                    davtest.txt
Password:
Locking http://localhost/davtest/davtest.txt with timeout 120
and depth infinity Locked with locktoken
"opaquelocktoken:ce018a78-a1c4-4d11-8e47-a1b04584836e",
depth infinity and timeout Infinite
```

Listing 11. *Failed PUT and MKCOL requests on a locked collection*

```
$ ./davlock.py -u davuser -d infinity http://localhost/
                    davtest/
Password:
Locking http://localhost/davtest/ with timeout 120 and
                    depth infinity
Locked with locktoken "opaquelocktoken:fd2b0845-74a3-
                    4f09-85a3-65fea74d097a",
depth infinity and timeout Infinite
$ ./davput.py -u davuser -f davtest.txt \
    http://localhost/davtest/davtest.txt
Password:
Failed to put the resource; status 423 Locked
$ ./davmkcol.py -u davuser http://localhost/davtest/dir/
Password:
Status for /davtest/dir: HTTP/1.1 424 Failed Dependency
Status for /davtest: HTTP/1.1 423 Locked
$ ./davput.py -u davuser -f davtest.txt \
   -l opaquelocktoken:fd2b0845-74a3-4f09-85a3-
                    65fea74d097a \
   http://localhost/davtest/davtest.txt
Password:
Replaced the resource
```

A Test Run with Cadaver

We now put a file to the WebDAV directory using the interactive client `cadaver`:

```
$ echo "dav test" > davtest.txt
$ cadaver
dav:!> open http://localhost/davtest/
dav:/davtest/> put davtest.txt
dav:/davtest/> rm davtest.txt
Uploading davtest.txt to `/davtest/davtest.txt'
```

The file is put to *http://localhost/davtest/davtest.txt* and it can be opened in a browser. The command `open` sends an `OPTIONS` request to `/davtest` to check the server DAV capabilities. If the server supports WebDAV, the client then sends `PROPFIND` to retrieve the contents of the collection. The command `put` sends a `PUT` request whose body is the contents of `davtest.txt` and `rm` triggers a `DELETE` request. The `cadaver` session proves that Apache setup works.

Authentication of the WebDAV operations

The WebDAV requests must be authenticated because they modify or destroy the contents of the web server. We implement basic authentication.

Listing 12. *UNLOCK request and response*

```
def send_request(conn, urlcomps, user, password,
                 locktoken):
    host = urlcomps.hostname
    port = urlcomps.port
    path = urlcomps.path
    headers = {'Host' : host, 'User-Agent' : sys.argv[0],
               'If' : '<%s>' % locktoken}
    auth = base64.b64encode('%s:%s' % (user, password))
    headers['Authorization'] = 'Basic %s' % auth
    logger.debug('Request headers: %s' % headers)

    conn.request('UNLOCK', path, None, headers)
    response = conn.getresponse()
    return response

def handle_response(response):
    if response.status == httplib.NO_CONTENT:
        logger.info('Unlocked')
    else:
        logger.error('Response status: %s %s' %
                     (response.status,
                     response.reason))
```

Listing 13. *Lock-Modify-Unlock scenario*

```
$ ./davlock.py -u davuser http://localhost/davtest/
                    davtest.txt
$ ./davput.py -u davuser -f davtest.txt \
  -l "opaquelocktoken:c173aa69-8db8-46bd-b1f4-
                    611f4afc978b" \
  http://localhost/davtest/davtest.txt
$ ./davunlock.py -u davuser
  -l opaquelocktoken:c173aa69-8db8-46bd-b1f4-611f4afc978b
                    \
  http://localhost/davtest/davtest.txt
```

Listing 14. *PROPFIND request*

```
def send_request(conn, urlcomps, user, password,
                 depth='0',
                 properties=[]):
    headers = {'Host' : urlcomps.hostname,
               'User-Agent' : sys.argv[0],
               'Depth':depth}
    auth = base64.b64encode('%s:%s' % (user, password))
    headers['Authorization'] = 'Basic %s' % auth
    body = construct_body(properties)

    logger.debug('Request headers: %s' % headers)
    logger.debug('Request body: %s' % body)

    conn.request('PROPFIND', urlcomps.path, body, headers)
    response = conn.getresponse()
    return response

def construct_body(props):
    if len(props) == 0:
        return None

    dom = getDOMImplementation()
    bodydoc = dom.createDocument('DAV:', 'D:propfind',
                    None)
    propfindel = bodydoc.documentElement
    propfindel.setAttribute('xmlns:D', 'DAV:')
    propel = bodydoc.createElementNS('DAV:', 'D:prop')
    propfindel.appendChild(propel)
    for prop in props:
        el = bodydoc.createElement(prop)
        propel.appendChild(el)
    body = bodydoc.toxml('UTF-8')
    return body
```

We load the modules that provide basic authentication with credentials stored in a file:

```
LoadModule auth_basic_module modules/mod_auth_basic.so
LoadModule authn_file_module modules/mod_authn_file.so
LoadModule authz_user_module modules/mod_authz_user.so
```

We create a file with a test user's password. The file is `davusers` in the Apache `conf/` directory; the user's name is *davuser* with password *davtest*.

```
# htpasswd -b -c davusers davuser davtest
```

Listing 2 shows the configuration of the DAV-enabled directory with basic authentication.

When we open *http://localhost/davtest/* in `cadaver` it asks for a username and a password – because it sends an `OPTIONS` request and it is protected.

The basic authentication is not a great defense as the passwords are transferred almost in plain text, but it can give decent protection if used over SSL. SSL itself can authenticate the client to the server if the client provides a certificate, but it requires some PKI infrastructure setup. We leave these SSL configurations as an exercise to the reader with the hint the client certificate authentication is done with Apache's `SSLVerifyClient` directive.

WebDAV Methods in Detail

We will now describe the WebDAV requests and responses. Instead of capturing the communication between a client and a server we will programmatically construct the requests, send them to the server and parse the responses. We will use Python.

Options

A client sends an `OPTIONS` request to find out the `DAV` capabilities of the server. The latter describes them with some response's headers like `DAV`. Here is a complete program that sends an `OPTIONS` request: see Listing 3.

In `main()` we parse the command line options and arguments where `-u` contains the username and the url is the first argument after the options. If the user option is provided we prompt for a password. We extract the host and the port for the url and open a `HTTPConnection`. In `send_request` we populate a dictionary with headers. For the `Authorization` header we encode with Base64 the username and the password. Then we send the request using `request` function of the connection object. The function takes the path of the url, the body of the request and the headers. In the case of `OPTIONS` the body is empty, that is `None`. Next we obtain the response from the connection and we return

Listing 15. *PROPFIND response*

```
def handle_response(response):
    logger.debug('Response status: %s %s' %
                 (response.status, response.reason))
    logger.debug('Response headers: %s', response.getheaders())
    data = response.read()
    logger.debug('Response body: %s', data)

    if response.status == httplib.MULTI_STATUS:
        parse_xml_body(data)
    else:
        logger.error('Response status: %s %s' %
                     (response.status, response.reason))

def parse_xml_body(data):
    dom = parseString(data)
    multistatusel = dom.documentElement
    responses = multistatusel.getElementsByTagNameNS('DAV:', 'response')
    for responseel in responses:
        hrefel = responseel.getElementsByTagNameNS('DAV:', 'href')[0]
        href = hrefel.firstChild.data
        logger.info('Properties for %s' % href)
        propstatels = responseel.getElementsByTagNameNS('DAV:', 'propstat')
        for propstatel in propstatels:
            statusel = propstatel.getElementsByTagNameNS('DAV:', 'status')[0]
            status = statusel.firstChild.data
            propel = propstatel.getElementsByTagNameNS('DAV:', 'prop')[0]
            props = propel.childNodes
            if status.find(str(httplib.OK)) > -1:
                for prop in props:
                    if prop.nodeType == Node.ELEMENT_NODE:
                        if prop.hasChildNodes():
                            r = []
                            for node in prop.childNodes:
                                s = node.toxml(dom.encoding).strip()
                                r.append(s)
                            logger.info('%s = %s' % (prop.nodeName, ''.join(r)))
                        else:
                            logger.info('%s' % prop.nodeName)
            elif status.find(str(httplib.NOT_FOUND)) > -1:
                for prop in props:
                    if prop.nodeType == Node.ELEMENT_NODE:
                        logger.warn('Property %s not found' % prop.nodeName)
            else:
                for prop in props:
                    if prop.nodeType == Node.ELEMENT_NODE:
                        logger.error('%s %s', prop.nodeName, status)
```

it. The function `handle_response` parses the response. In our case we check for the presence of the `DAV` header that indicates WebDAV compliance. There are three classes of compliance numbered 1, 2, and 3. Class 2 means that the resource supports locking. The exact definitions are given in RFC 4918 and RFC 2518. We also show the `Allow` header and it contains the WebDAV methods.

Listing 4 shows these headers for the dav-enabled resource *http://localhost/davtest/*. For a non-dav resource, the `DAV` header is missing and the allowed methods header does not contain the `DAV` methods:

```
$ ./davoptions.py http://localhost/
Sending OPTIONS request to http://localhost/
Allowed methods: GET,HEAD,POST,OPTIONS,TRACE
```

For all other examples `main()` is similar – it opens the connection, calls `send_request` and `handle_response` and closes the connection. We will show only `send_request` and `handle_response` methods for the other examples. We also log the headers and the body for each request and each response.

PUT

PUT is used to upload a non-collection, that is a non-directory resource. Its behavior is undefined when applied to an existing collections. If we want to create a new collection we use `MKCOL`. Listing 5 shows how to construct a PUT request.

The body of the `PUT` request becomes the contents of the server resource. If it does not exists, it will be created; if it exists it will be replaced. In our case we read the body from a file. We also provide an optional lock token. If the resource is not locked we do not need it, but if it is locked we have to provide a valid token. We may also supply a content type in the request headers. If the we do not, the server may choose a content type and assign it to the resource.

The successful responses for PUT are 201 and 204 and `handle_response` in Listing 6 checks for them.

If *http://localhost/davtest/davtest.txt* does not exist we can create it with a `PUT` request:

```
$ echo 'test dav' > davtest.txt
$ ./davput.py -u davuser -f davtest.txt \
    http://localhost/davtest/davtest.txt
Password:
Created the resource
```

We can modify it with another PUT request:

```
$ echo 'changed' >> davtest.txt
$ ./davput.py -u davuser -f davtest.txt \
    http://localhost/davtest/davtest.txt
Password:
Replaced the resource
```

Using `PUT` we can upload not-only *static* content, but also scripts like PHP or Python.

Listing 16. *PROPFIND response and the properties it contains*

```
$ ./davpropfind.py -u davuser http://localhost/davtest/davtest.txt
Password:
Properties for /davtest/davtest.txt
lp1:resourcetype
lp1:creationdate = 2010-12-04T05:37:01Z
lp1:getcontentlength = 17
lp1:getlastmodified = Sat, 04 Dec 2010 05:37:01 GMT
lp1:getetag = "2c3022-11-4968f0b4e1940"
lp2:executable = F
D:supportedlock = <D:lockentry>
<D:lockscope><D:exclusive/></D:lockscope>
<D:locktype><D:write/></D:locktype>
</D:lockentry><D:lockentry>
<D:lockscope><D:shared/></D:lockscope>
<D:locktype><D:write/></D:locktype>
</D:lockentry>
D:lockdiscovery
D:getcontenttype = text/plain
```

MKCOL

An `MKCOL` request creates a collection resource at the given URI. It may contain a body, but the behavior in that case is undefined. It may optionally present a lock token, if the parent collection is locked (we will describe locking later). Listing 7 shows how to construct an `MKCOL` request.

The response returns the status code for the operation. The successful one is 201 Created. 405 Method Not Allowed means that the directory already exists – the method is allowed only for non-existing collections; 409 Conflict means that a parent collection does not exist; 423 Locked means that a parent collection is locked and the client has not provided the lock token; 207 Multistatus means that the request was related to several resource and they returned different status codes in the XML body of the response: see Listing 8. If *http://localhost/davtest/dir1/* does not exist we create it with:

```
./davmkcol.py -u davuser http://localhost/davtest/dir1/
Password:
Created the collection
```

A second attempt to recreate it results with an error:

```
$ ./davmkcol.py -u davuser http://localhost/davtest/dir1/
Password:
The resource already exists
```

An attempt to create a collection whose parent collection does not exist is also an error:

```
$ ./davmkcol.py -u davuser http://localhost/
                    davtest/bogus/dir/
Password:
The parent collection does not exist
```

Lock

`LOCK` is used to lock a resource. There are two types of locks – exclusive and shared. The exclusive one allows access only to the holder of the lock token. A shared one does not restrict access but simply informs that someone else may be working on this resource. We will demonstrate exclusive locks. When a client locks a resource, the server returns a lock token to it. Any requests that modify the resource must provide the lock token to succeed. If the client wants to extend the lock of the resource, it sends another `LOCK` request to it with the lock token returned from the previous one.

The bodies of the `LOCK` request and the `LOCK` response can be XML documents. Listing 9 shows how to use `xml.dom.minidom` to generate the request's XML body.

Several headers are important for this request. The `Timeout` header contains the client's desired period of the lock; after that period the lock will expire even if the resource is not explicitly unlocked. The server may not honor this header. The `Depth` header is ignored if the resource to be locked is not a collection. If it is a collection the `Depth` header specifies how *deep* will be the lock: if Infinity the collection and its children including the subcollections's children will be locked; if 0, only the collection without its children will be protected by the lock.

The response comes with a Timeout header that contains the lock's period assigned by the server. The body of the response is in XML and one of its nodes contains the lock token. The server will require the lock token in the `If` header of any subsequent requests that modify that resource.

We will illustrate all that headers and XML machinery with a couple of examples.Let us assume that *http://localhost/davtest/davtest.txt* exists and we want to lock it in order to update it: see Listing 10.

The resource is locked and the server returns its lock token. Any attempt to modify it without the lock token will fail:

Listing 17. *PROPPATCH request*

```
def send_request(conn, urlcomps, user, password,
                 toset={}, toremove=[]):
    headers = {'Host' : urlcomps.hostname,
               'User-Agent' : sys.argv[0]}
    auth = base64.b64encode('%s:%s' % (user, password))
    headers['Authorization'] = 'Basic %s' % auth
    body = construct_body(toset, toremove)
    logger.debug('Request headers: %s' % headers)
    logger.debug('Request body: %s' % body)
    conn.request('PROPPATCH', urlcomps.path, body, headers)
    response = conn.getresponse()
    return response
def construct_body(toset, toremove):
    dom = getDOMImplementation()
    bodydoc = dom.createDocument('DAV:', 'D:propertyupdate', None)
    propertyupdateel = bodydoc.documentElement
    propertyupdateel.setAttribute('xmlns:D', 'DAV:')
    if len(toset) > 0:
        setel = bodydoc.createElementNS('DAV:', 'D:set')
        propertyupdateel.appendChild(setel)
        propel = bodydoc.createElementNS('DAV:', 'D:prop')
        setel.appendChild(propel)
        for key in toset:
            keyel = bodydoc.createElement(key)
            valueel = bodydoc.createTextNode(toset[key])
            keyel.appendChild(valueel)
            propel.appendChild(keyel)
    if len(toremove) > 0:
        removeel = bodydoc.createElementNS('DAV:', 'D:remove')
        propertyupdateel.appendChild(removeel)
        propel = bodydoc.createElementNS('DAV:', 'D:prop')
        removeel.appendChild(propel)
        for key in toremove:
            keyel = bodydoc.createElement(key)
            propel.appendChild(keyel)
    body = bodydoc.toxml('UTF-8')
    return body
```

```
$ ./davput.py -u davuser -f davtest.txt \
    http://localhost/davtest/davtest.txt
Password:
Failed to put the resource; status 423 Locked
```

Now let's provide the lock token:

```
$ ./davput.py -u davuser \
    -l opaquelocktoken:7e083e8c-0fb8-4010-8571-b3ce86b1b4cb
    -f davtest.txt http://localhost/davtest/davtest.txt
Password:
Replaced the resource
```

This showed how to lock a non-collection resource. If we lock a collection with depth infinity we can't add new members to it without the locktoken: see Listng 11. Finally, we can refresh the lock on a resource, if we send a `LOCK` request with the current lock in the `If` header.

UNLOCK

The `UNLOCK` method unlocks a locked resource. It provides the lock token in its `If` header. 204 No Content is the successful status code for the response: see Listing 12.

We can now implement a lock-modify-unlock scenario: see Listing 13.

Propfind

The `PROPFIND` method retrieves the resources's metadata. It consists of properties with names and values. The method can retrieve all properties with their values, a specific set of

Listing 18. *PROPPATCH response*

```
def handle_response(response):
    data = response.read()
    if response.status == httplib.MULTI_STATUS:
        parse_xml_body(data)
    else:
        logger.error('Response status: %s %s' %
                     (response.status, response.reason))
def parse_xml_body(data):
    dom = parseString(data)
    multistatusel = dom.documentElement
    responses = multistatusel.getElementsByTagNameNS('DAV:', 'response')
    for responseel in responses:
        hrefel = responseel.getElementsByTagNameNS('DAV:', 'href')[0]
        href = hrefel.firstChild.data
        logger.info('Properties for %s' % href)
        propstatels = responseel.getElementsByTagNameNS('DAV:', 'propstat')
        for propstatel in propstatels:
            statusel = propstatel.getElementsByTagNameNS('DAV:', 'status')[0]
            status = statusel.firstChild.data
            propel = propstatel.getElementsByTagNameNS('DAV:', 'prop')[0]
            props = propel.childNodes
            if status.find(str(httplib.OK)) > -1:
                for prop in props:
                    if prop.nodeType == Node.ELEMENT_NODE:
                        logger.info('Operation on %s succeeded' % prop.nodeName)
            else:
                descriptionel = propstatel.getElementsByTagNameNS(
                    'DAV:', 'responsedescription')[0]
                description = descriptionel.firstChild.data.strip()
                for prop in props:
                    if prop.nodeType == Node.ELEMENT_NODE:
                        logger.error('%s %s' % (prop.nodeName, description))
```

properties or only the properties' names (see Listing 14). If we send a PROPFIND request with an empty body, the server returns all the properties. If we send an XML body with names of the properties, it returns only them. In both case we parse the body of the response in order to obtain them. The response's status is 207 Multistatus, because some properties may be fetched and some not (see Listing 15).

WebDAV itself uses properties for its internal working. For example, it uses resourcetype to describe if a resource is a collection or not, executable if the resource is executable; supportedlock to describe what locks are supported: see Listing 16.

The resourcetype is empty, so the resource is not a collection. The collections give that property a value:

```
$ ./davpropfind.py -u davuser http://localhost/davtest/
Password:
lp1:resourcetype = <D:collection/>
```

Some of those properties can be changed and some cannnot be changed. For example, we cannot modify resourcetype. We can modify executable and this has an interesting application.

Apache Httpd has an option XBitHack and when it is turned on, the Server Side Includes in a file are expanded if it is executable. Setting the executable property is one way to set the x bit on a file.

The PROPFIND method support the Depth header with values 0, 1 and Infinity.

A request with depth 0 retrieves the collection's properties without its members' ones, depth 1 – the collection's properties and its children's but without the properties of the subcollections, depth infinity recursively retrieves them for the whole tree:

```
$ ./davpropfind.py -u davuser -d 0 http://localhost/davtest/
$ ./davpropfind.py -u davuser -d 1 http://localhost/davtest/
$ ./davpropfind.py -u davuser -d Infinity http://localhost/davtest/
```

Some servers may reject PROPFIND request with depth infinity, because the operation can be computationally heavy. In the case of Apache, we can allow such request with the DavDepthInfinity directive.

Proppatch

The PROPPATCH method sets or remove properties from resources. Its XML body contains the properties and their values it will set, and the properties' names it will delete: see Listing 17.

The response's status code can be 207 Multistatus and its XML body contains the results from the operations. If there is an error for specific properties it provides a description: see Listing 18.

We set a single property and retrieve it:

```
$ ./davproppatch.py -u davuser -s prop=value http://
                    localhost/davtest/
$ ./davpropfind.py -u davuser -p prop htp://localhost/davtest/
```

Listing 19. *Setting deleting and listing properties*

```
$ ./davproppatch.py -u davuser -s prop=value http:
                    //localhost/davtest/
$ ./davproppatch.py -u davuser -s prop1=value1 -s
                    prop2=value2 -r prop \
    http://localhost/davtest/
Password:
Properties for /davtest
Operation on prop1 succeeded
Operation on prop2 succeeded
Operation on prop succeeded
$ ./davpropfind.py -u davuser -p prop1 -p prop2 -p prop \
    htp://localhost/davtest/
Password:
Properties for /davtest/
prop1 = value1
prop2 = value2
Property prop not found
```

Listing 20. *COPY/MOVE request*

```
def send_request(conn, srccomps, user, password, dest,
                 overwrite='F',
                 depth='0'):
    headers = {'Host' : srccomps.hostname,
               'User-Agent' : sys.argv[0],
               'Overwrite' : overwrite,
               'Destination' : dest,
               'Depth' : depth}
    auth = base64.b64encode('%s:%s' % (user, password))
    headers['Authorization'] = 'Basic %s' % auth
    body = None
    logger.debug('Request headers: %s' % headers)
    logger.debug('Request body: %s' % body)

    conn.request('MOVE', srccomps.path, body, headers)
    response = conn.getresponse()
    return response
```

Listing 21. *COPY/MOVE response*

```
def handle_response(response):
    data = response.read()
    if response.status == httplib.CREATED:
        logger.info('The source was copied to the destination')
    elif response.status == httplib.NO_CONTENT:
        logger.info('The source overwrote the destination')
    elif response.status == httplib.MULTI_STATUS:
        parse_xml_body(data)
    else:
        logger.error('%s %s' %
                     (response.status, response.reason))
def parse_xml_data(data):
    dom = parseString(data)
    multistatusel = dom.documentElement
    responses = multistatusel.getElementsByTagNameNS('DAV:', 'response')
    for responseel in responses:
        hrefel = responseel.getElementsByTagNameNS('DAV:', 'href')[0]
        href = hrefel.firstChild.data
        statusel = responseel.getElementsByTagNameNS('DAV:', 'status')[0]
        status = statusel.firstChild.data
        if (status.find(str(httplib.OK)) > -1) or
             (status.find(str(httplib.CREATED)) > -1):
           logger.info('Operation on %s succeeded', href)
        else:
            errorel = responseel.getElementsByTagNameNS('DAV:', 'error')[0]
            error = errorel.firstChild.data
            logger.error('%s %s %s', href, status, error)
```

```
Password:
Properties for /davtest/
prop = value
```

Now we will delete that property and set two new in one request: see Listing 19.

The method `PROPPATCH` can be used to assign ownership or authorship:

```
$ ./davproppatch.py -u davuser -s author=davuser
                    http://localhost/davtest/
$ ./davpropfind.py -u davuser -p author http://localhost/davtest/
Password:
Properties for /davtest/
author = davuser
```

The values of the properties can be custom XML documents. To prevent name clashing you should use xml namespaces in this case.

COPY and MOVE

The methods `COPY` and `MOVE` copy or move a resource. Their requests are relatively simple. They accept the `Depth` header if they act on a collection, the `Overwrite` header that specifies whether to overwrite the destination if it exists; the `Destination` header specifies the new location of the resource. The Request-URI is the source. The methods send no body: see Listing 20.

The `COPY` request is the same except the line:

```
    conn.request('COPY', srccomps.path,
body, headers)
```

The successful response status codes are 201 Created and 204 No Content. 207 Multistatus is also possible if the request acted on a collection and some of the operations succeed and some fail: see Listing 21.

This concludes the overview of the WebDAV methods. It covered the basic operations, but omits some details as the scopes of the locks and lock discovery properties.

Interoperability

Our experience shows that the dav clients and servers integrate well between each other. Both parties sometimes have to work around some idiosyncracies of the other side. As a whole the protocol is well-defined and it is easy to comply with it. Of course, different webdav products are at different level of maturity so you should evaluate them before production use. There is a test suite called `litmus` that checks if a server is compatible with the WebDAV RFCs.

IVAN „RAMBIUS" IVANOV

Ivan „Rambius" Ivanov is a Bulgarian software developer currently based in New York and a member of NYC BSD User Group. He has implememented a WebDAV-based software repository with dependenc management and deployment tracking.

iXsystems Releases FreeNAS 8 Beta

Community feedback sought before final release

San Jose, CA (PRWEB) November 19, 2010

FreeNAS–an Open Source Storage Platform based on FreeBSD–supports sharing across Windows, Apple, and UNIX-like systems. Both home and business users can share media across an entire network. Other features include FTP, data replication, iSCSI, and support for Apple's Time Machine. The current FreeNAS Beta release is now available for testing and can be downloaded from http://sourceforge.net/projects/freenas/files/. The project welcomes community feedback to improve the product for final release.

FreeNAS can be installed and configured with ease using its new web interface. Users can store, access, and manage data because the data resides on the user's private network. iXsystems has replaced the existing FreeNAS interface (based on m0n0wall) with the Python Django framework, whose modular nature easily integrates third-party software within the FreeNAS interface. Ajax technology provided by the Dojo toolkit enhances the user experience by providing real-time feedback.

„The new FreeNAS makes it easier to upgrade, back up, or restore, with the system taking care of all the details. FreeNAS makes running a NAS box simple and easy," says Warner Losh, Director of FreeBSD Development at iXsystems.

FreeNAS 8 Beta includes ZFS, which supports high storage capacities and integrates file systems and volume management into a single piece of software. One benefit of ZFS is System Snapshots, in which numerous snapshots of data are taken and used to restore lost or deleted files. Future versions of FreeNAS will include other ZFS benefits such as De-Duplication, which conserves disk space by sending a pointer to the location of the original file for any replicated data.

The base of the system has been upgraded to FreeBSD 8.1-RELEASE. Asynchronous IO has been enabled in Samba 3.5 to increase performance over previous versions. FreeNAS now uses FreeBSD's rc.d configuration and stores all system states in a sqlite3 database. The build process takes advantage of NanoBSD, a trimmed down version of FreeBSD with added enhancements. In previous versions, FreeNAS used many non-standard components that inhibited updates to FreeBSD. The rewrite ensures that FreeNAS can quickly and easily stay up to date with FreeBSD.

About The FreeNAS Project:

FreeNAS is an embedded open source NAS (Network-Attached Storage) distribution based on FreeBSD that enables safe and reliable data storage while also conserving disk space. FreeNAS was originally created by Olivier Cochard-Labbe in October of 2005. In January of 2010 its development was taken over by iXsystems, Inc. FreeNAS 8 is currently in the beta phase and project developers are looking to the community for feedback and assistance with bug testing before the final release.

About iXsystems, Inc.:

iXsystems is the all-around FreeBSD company that builds FreeBSD-certified servers and storage solutions, runs the FreeBSD Mall, and is the corporate sponsor of the PC-BSD Project. iXsystems is an employee-owned and-operated, open source-centric, customer focused organization, dedicated to providing the highest quality built-to-order enterprise rackmount server solutions, pre-configured server appliances, and scalable storage solutions to our customers around the globe.

Managing software

with NetBSD's pkgsrc packaging system

pkgsrc is NetBSD's package management system. But it supports far more than just NetBSD. At last count, over 11 distinct operating systems were supported by pkgsrc.

What you will learn…

- How to boot strap pkgsrc
- How configure and build packages using pkgsrc
- upgrading packages using pkgsrc

What you should know…

- Basic UNIX commands – tar, your shell, ftp
- How to download files from the internet

Supported operating systems include Solaris, HP-UX, Linux (in all it's incarnations), FreeBSD, MacOS, OpenBSD, NetBSD, AIX, IRIX, OSF/1, QNX, and DragonFlyBSD. NetBSD and DragonFlyBSD use it as their *native* package management systems.

As a Systems/Site administrator, I've personally used pkgsrc on NetBSD, HP-UX (where I did some of the porting/development of the pkgsrc support for HP-UX), Linux (Debian) and FreeBSD. pkgsrc is the packaging system of preference because of it's uniform dependency handling, and it's uniform interface for package installation and management. It is easy to boot strap on a non-NetBSD host (and non-NetBSD also includes NetBSD versions prior to NetBSD 4.0). pkgsrc provides mechanisms for building packages from source (the primary interface) as well as building binary distributions. pkgsrc also includes mechanisms for controlling the features desired and used by individual packages, as well as insuring a tightly controlled dependency tree. (no discovering some randomly installed piece of software accidently became a runtime dependency.)

Why choose pkgsrc over an operating system's provided package management system? pkgsrc tries to follow the latest releases of supported software, making them available in a timely fashion. It also provides tools to be able to upgrade the software once installed in a relatively painless fashion. (however, no upgrade between major versions of software is going to be entirely painless, sadly.)

pkgsrc also supports building as/for an individual user, within a users home directory. It is a wonderful way to experiment with pkgsrc on a system where you don't have super-user access. pkgsrc can build its packages as an unprivileged user, needing only super-user access for final installation onto the system (if not installing into the users home directory.)

pkgsrc provides quarterly stable releases, intended for production use, to minimize the packaging churn on stable production systems. The stable releases are named after their tags in the pkgsrc CVS repository. They are named in the fashion of `pkgsrc-<year>Q<quarter>`, where `<quarter>` is the number of the quarter just ended at the time of release. Only critical fixes and changes get brought into the stable release.

So, after all the promotion, lets get to using pkgsrc for package management! We'll start off by down loading the current stable pkgsrc tar-ball (pkgsrc-2010Q3 at the time of writing), boot strapping it on a non-NetBSD host (FreeBSD), and then proceed to building and installing packages. Off we go!

Downloading the pkgsrc tar archive

Lets start by downloading the current tar archive for the chosen pkgsrc stable branch. The URL for this archive is: *ftp://ftp.pkgsrc.org/pub/pkgsrc/pkgsrc-2010Q3/pkgsrc.tar.gz* (and it is available via http as well, my habit is to use ftp out of preference.)

After a few minutes, you should have the pkgsrc tar-ball in your current working directory. (of course, cvs can also be used to grab the stable branch, just use the tag name in the `cvs` command line. The repository is at `anoncvs.netbsd.org:/cvsroot`, the module is pkgsrc.)

Once you've downloaded the pkgsrc tar-ball, time to extract. Pick your favorite work directory, and extract the tar-ball using the standard command:

```
tar xzf <path to>/pkgsrc.tar.gz
```

If your tar doesn't understand the `z` option, (as might be the case on AIX, HP-UX or Solaris), you'll have to use gzcat to uncompress the archive, and then pipe that into tar. (hopefully, you know how to do that.)

Bootstrapping

Once you've got the tar archive extracted, you need to boot strap the pkgsrc package management tools. They are make, `nbftp`, `pkg_add`, `pkg_info`, `pkg_admin`, and `pkg_delete`. Other tools may be built as part of bootstrapping if suitable tools don't already exist on your system.

Of course, if you're running a recent, supported version of NetBSD, this boot strapping is not required.

The only tools required of the host system are a working C compiler (supporting ANSI C) and a working, basic make(1) program. To start bootstrapping, change directory to top level pkgsrc directory. Within you'll find the subdirectories shown in Listing 1.

Change directories into the *bootstrap* directory, and you will see the files/directories shown in Listing 2.

Reading the general README file, and the README file for your operating system can be very helpful. I would strongly suggest taking a break and reading them now, even though I'm going to give you the step by step process in this article.

In the instance of our FreeBSD installation, we're going to use it as a replacement for the FreeBSD packaging system, so we'll need to be root to do the boot strap. We'll also want to remove/rename the system provided packaging tools, to reduce the possibility of conflict with the pkgsrc provided tools. We'll start by renaming the system tools. To do so, we'll need to become super-user, and execute the following commands:

```
# mv /usr/sbin/pkg_add /usr/sbin/pkg_add-freebsd
# mv /usr/sbin/pkg_create /usr/sbin/pkg_create-freebsd
# mv /usr/sbin/pkg_delete /usr/sbin/pkg_delete-freebsd
v# mv /usr/sbin/pkg_info /usr/sbin/pkg_info-freebsd
# mv /usr/sbin/pkg_updating /usr/sbin/pkg_updating-freebsd
# mv /usr/sbin/pkg_version /usr/sbin/pkg_version-freebsd
```

and optionally remove the execute bits from the programs:

```
# chmod a-x /usr/sbin/pkg_*
```

For our FreeBSD bootstrap, we're going to use the following command line. The `-pkgdbdir` needs to be set,

Listing 1. *Top level pkgsrc directory*

```
CVS/          chat/         emulators/    lang/         net/          shells/
Makefile      comms/        filesystems/  licenses/     news/         sysutils/
README        converters/   finance/      mail/         packages/     templates/
archivers/    cross/        fonts/        math/         parallel/     textproc/
audio/        databases/    games/        mbone/        pkglocate*    time/
benchmarks/   devel/        geography/    meta-pkgs/    pkgtools/     wm/
biology/      distfiles/    graphics/     misc/         print/        www/
bootstrap/    doc/          ham/          mk/           regress/      x11/
cad/          editors/      inputmethod/  multimedia/   security/
```

Listing 2. *Contents of the pkgsrc/bootstrap directory*

```
CVS/                README.Haiku        README.MacOSX       cleanup*
README              README.IRIX         README.OSF1         macpkg.pmproj.in
README.AIX          README.IRIX5.3      README.OpenBSD      testbootstrap*
README.FreeBSD      README.Interix      README.Solaris
README.HPUX         README.Linux        bootstrap*
```

since the FreeBSD packaging tools want to use `/var/db/pkg` for their packaging database.

```
# ./bootstrap --pkgdbdir /usr/pkg/db
```

The bootstrap command will proceed to build the bmake, and the needed support programs to provide the environment that pkgsrc expects.

mk.conf

At the end of the build, you should have seen a message about `/usr/pkg/etc/mk.conf` containing various settings for building pkgsrc packages. `mk.conf` is used to define the list of licenses you accept (the standard open source licenses are accepted by default, but some software packages have more restrictive licenses), options for packages, and other items of interest.

The two most interesting items are `ACCEPTABLE_LICENSES`, which defines the allowed licenses, and `PKG_DEFAULT_OPTIONS`, which defines the list of options to apply to all packages that support them.

`mk.conf` can also define per package options using the `PKG_OPTIONS.<pkgname>` syntax. Individual packages may have more options than the standard list.

Other options include `X11_TYPE` (set to either *native* or *modular*, aka from pkgsrc), `USE_DESTDIR=yes`, which causes

Listing 3. *List of installed packages after installing sudo*

```
fbsd-> pkg_info
bootstrap-mk-files-20090807nb2 *.mk files for the bootstrap bmake utility
bmake-20100808      Portable (autoconf) version of NetBSD 'make' utility
pkg_install-20100915 Package management and administration tools for pkgsrc
digest-20080510     Message digest wrapper utility
tnftp-20070806      The enhanced FTP client in NetBSD
f2c-20090411nb5     Fortran to C compiler including a script to emulate f77
libtool-base-2.2.6bnb4 Generic shared library support script (the script itself)
sudo-1.7.4p4nb1     Allow others to run commands as root
fbsd->
```

Listing 4. *pkg_info(1) output for bmake*

```
fbsd-> pkg_info bmake
Information for bmake-20100808:

*** PACKAGE MAY NOT BE DELETED ***
Comment:
Portable (autoconf) version of NetBSD 'make' utility

Requires:
bootstrap-mk-files-[0-9]*

Description:
bmake is a portable version of NetBSD's make(1) utility,
conveniently packaged using a configure script, for other environments
which may lack NetBSD's libraries, regular expression code, etc.

Homepage:
http://www.crufty.net/help/sjg/bmake.html

*** PACKAGE MAY NOT BE DELETED ***

fbsd->
```

packages to build and package into subdirectories of the build tree, rather than directly into the system directories. Useful entries are WORKOBJDIR, DISTDIR, and PACKAGES, which tell the pkgsrc build infrastructure where to build software, where to stash the distribution archives downloaded from the Internet, and where to put the final packaged packages, respectively. WORKOBJDIR is usually within the package directory, DISTDIR is ./distfiles within the pkgsrc top level directory, and PACKAGES is ./packages within the pkgsrc top level directory.

Building our first package

Now that we've got the software bootstrapped, we can move to building packages. The first package I would recommend building is security/sudo. It is quick and simple. Use the following command:

```
bmake update
```

Well, it will once you add /usr/pkg/bin to your path. (yes, I forgot to do that while writing this.)

Our freshly installed bmake will proceed to download the sudo sources, build and install any dependencies (in this case, pkgtools/digest), build sudo, and install sudo. Once sudo is installed, you'll want to configure it using visudo. Make sure to list your non-privileged user id! Having installed and configured sudo, you can now add an additional feature to mk.conf, allowing the building of packages unprivileged, but using sudo to do the installation phase. The variable is SU_CMD, and you want to set it like this:

```
SU_CMD=sudo -u root sh -c
```

Otherwise, when building as an unprivileged user, you'll be prompted for the root password to install.

The Package Database

Now that we've bootstrapped, and installed sudo, lets take a look at see what the package database looks like. Use the following command:

```
pkg_info
```

And you should get something like Listing 3.

For more details about any package, you use pkg_info(1) followed by the package name (versions aren't required), and it will tell you a bit about the package. For example, asking about bmake will return Listing 4.

Listing 5. *package options for net/wget*

```
fbsd-> bmake show-options
Any of the following general options may be selected:
        idn      Internationalized Domain Names (IDN) support.
        inet6    Enable support for IPv6.
        ssl      Enable SSL support.

These options are enabled by default:
        idn ssl

These options are currently enabled:
        idn inet6 ssl

You can select which build options to use by setting PKG_DEFAULT_OPTIONS
or PKG_OPTIONS.wget.
fbsd->
```

Listing 6. *Example pkg_chk -uq output*

```
fbsd-> pkg_chk -uq
net/wget - wget-1.12nb1 < wget-1.12nb2
*** Unable to read PKGCHK_CONF
'/usr/home/eric/work/pkgsrc/pkgchk_update-fbsd.cirr.com..conf'
fbsd->
```

Building a package with options

Now, we've built a basic package, lets move onto something a bit more complex, a package with options. To determine if a package has options, look for the file options.mk in the pkgsrc package directory. Nearly all packages that support options have them defined in a file by that name.

Common, pkgsrc-wide, options are inet6, ssl, cups, etc. The global options are defined, with descriptions, in mk/defaults/options.description in the pkgsrc tree.

For my own systems, I like to use the following options as default:

```
PKG_DEFAULT_OPTIONS=sasl tcpwrappers inet6
 -hal cups -dbus ssl
```

Prefixing an option with a dash (-) disables the option when it defaults to on for a package. Myself, I dislike the baggage that dbus and hald add to the system, so I have them disabled by default.

I also like to make sure packages that can use SASL do so, and those that can use IPv6 and SSL also do so.

A package that uses options, and that shouldn't be too controversial, is net/wget. It has options of `idn`, `ssl`, and `inet6`. It defaults to using `idn` and `ssl`.

To see what options will be used by net/wget, change to the net/wget directory, and enter the following command:

```
bmake show-options
```

You'll get a response as seen in Listing 5.

If a different set of options is desired, edit `/usr/pkg/etc/mk.conf`, and add an entry for wget, such as:

```
PKG_OPTIONS.wget+= -idn -inet6
```

to remove the `idn` and `inet6` options. `FYI: PKG_OPTIONS` are always added too using the `+=` syntax in `mk.conf`. They can also be modified on the command line by using the following:

```
bmake PKG_OPTIONS.wget='-idn -inet6' update
```

Building wget demonstrates the nature of pkgsrc's dependency build system. wget has the following direct and indirect dependencies:

```
archivers/pax
devel/gettext-tools
devel/gettext-lib
converters/libiconv
lang/perl
```

They were all automatically downloaded, built and installed to support the build and installation of wget.

More Advanced Administration

So far, we've bootstrapped the pkgsrc system, and installed a few packages. That's easy, and nearly every packaging system handles basic build and installation well.

The hard part is upgrading packages once they've been installed. There are a few tools that are provided as part of pkgsrc that make upgrading packages easier. They are `pkgtools/pkg_chk` and `pkgtools/pkg_rolling-replace`.

`pkgtools/pkg_chk` provides two major pieces of functionality. The first is to build a predefined set of packages from a configuration file. The configuration file is formatted such that one file can be defined for all the systems at a site, including per-system specializations.

The second piece of functionality is upgrading the installed set of packages based on the extracted pkgsrc tree.

`pkgtools/pkg_rolling-replace` provides outstanding functionality for rebuilding changed packages, with a minimum window when the package being upgraded isn't installed. `pkg_rolling-replace` can also be used to make sure any packages depending on the replaced package are rebuilt, which is especially important in the case of API/ABI changes (although the package maintainers attempt to indicate changes in the API to the using packages by a revision increment of all the using packages.)

`pkgtools/pkg_chk` can be used to list the packages that may be out of date on the running system, verses the pkgsrc source tree.

```
pkg_chk -uq
```

will return something like Listing 6 identifying the out of date packages.

`pkg_chk -u` will rebuild the out of date packages by picking one of the out of date packages, un-installing it and the packages it depends upon, and then building/installing the updated package, and building/installing the packages that depend upon it.

`pkg_rolling-replace -u` determines the list of packages to be updated by calling `pkg_chk -uq` to generate the list, and then builds each of the out of date, installs it, and then rebuilds all packages that depend upon it. It does it in such a way to minimize the time any package is unavailable on a running system.

Wrapping up

Hopefully this article has provided a good overview of pkgsrc, and how to use it to build from sources, on one of the 11 supported operating systems. Our use of it on FreeBSD, in place of FreeBSD's native packaging system demonstrates some of the flexibility available with pkgsrc.

```
pkgsrc, it's not just for NetBSD!
```

ERIC SCHNOEBELEN

Eric Schnoebelen is a 25 year veteran of the UNIX wars, using both System V and BSD derived systems. He's spent more than 20 years working with and contributing to various open source projects, such as NetBSD, sendmail, tcsh, and jabberd2. He operates a UNIX consultancy, and a small, NetBSD powered ISP. His preferred OS is NetBSD, which he has running on Alpha, UltraSPARC, SPARC, amd64 and i386.


Looking for help, tip or advice?
Want to share your knowledge with others?
BSD MAGAZINE
Give us your opinion about the magazine's content
and help us create the most useful source for you!

Allocating Dynamic Memory with Confidence

Embedded software applications face many challenges that are not present on desktop computers. A device with a dedicated function is not generally regarded as a computer, even if a significant part of it is software.

What you will learn…

- Why dynamic memory allocation is necessary but dangerous.
- Strategies to avoid memory fragmentation.
- How to determine worst-case memory consumption.

What you should know…

- Application development concepts.
- Memory management without garbage collection.

Users will put up with occasional slowdowns and crashes on a desktop computer, but devices are held to a higher standard, especially when they are part of a mission-critical system. Memory allocation is an important factor for providing the necessary performance and reliability on an embedded device.

On a general-purpose computer, well-designed applications allocate memory on-demand, so that each application only uses as much memory as it needs at any given time. If an application needs a large amount of memory, the user is expected to stop using other applications until it is finished.

Embedded devices are typically designed to perform a fixed set of tasks. The user may not even realize that there are anything like applications running on a device. Devices that do not support virtual memory will simply fail when memory is full. Even an unexpected drop in performance can be frustrating, and in some cases dangerous. For that reason, well-designed applications on embedded systems often preallocate memory so that performance is consistent and failure is prevented.

However, for complex applications it is not always possible to predict all memory requirements in advance. Instead, the application can be analyzed to determine its worst-case memory consumption and allocate a buffer of that size in advance. Such analysis can be difficult, especially when starting from scratch.

Storing, organizing, and sharing data makes up a large part of the memory requirements for an application. A device application can use an embedded database library to manage memory more effectively, by both imposing bounds on memory usage and analyzing worst-case behavior in a consistent way. The database library can handle all the details of reading, writing, indexing, and locking data within a predictable footprint, so that the application's own memory requirements are greatly reduced.

Designing for Predictable Memory Usage

Reliable embedded devices depend on predictable behavior. For memory allocation, this requires knowing how much memory an application will need in the worst case, and then finding ways to reduce that amount. To do this, an application developer needs to follow a good memory allocation strategy, measure memory consumption under a variety of representative configurations, and analyze the results.

Total memory consumption includes not only the memory requested by the application, but also the overhead of the dynamic memory allocator itself. Some allocators are more susceptible to fragmentation than others, so it is important to know what kind of allocator the application is using. Most operating systems use a general-purpose allocator that performs well

on average, but that may badly fragment memory at unexpected times. On such platforms, a bounded allocator can be used in each application to limit allocation overhead.

Memory Allocation Strategy

A useful strategy to avoid memory fragmentation is two-phase allocation. Under this strategy, large and long-term object are allocated first so that they are guaranteed a place in memory. Small and short-lived objects are allocated in the second phase because they are less likely to fail even if memory is fragmented. In this way, there is little risk that an allocation will fail merely because no contiguous region of memory is large enough.

Both the application code itself and any libraries that allocate memory should apply this strategy. Otherwise, the worst-case behavior of the application cannot be predicted accurately. Even a bounded allocator cannot provide any guarantees if an embedded library only imposes soft limits on its allocation behavior.

Statistics Collection and Analysis

When measuring memory allocation behavior, the most important statistics to collect are the largest amount of memory allocated at any one time and the size of the single largest allocation, including allocator overhead. Other statistics may also be valuable for certain memory allocators.

The amount of memory used by an application usually depends on how it is configured and how it is used. Statistics should be collected for several different configurations that represent all of the extreme memory use cases. The application should also be divided into discrete operations that can be tested individually, so that results can be calculated without simulating all possible combinations.

By knowing an application's total memory consumption, it is possible to allocate a large enough memory pool when an application is started to satisfy all allocation requests for the life of the application. Provided that operations run sequentially, one by one, the memory consumption is defined as the largest consumption of any individual operation. If operations could overlap, the maximum memory consumption is defined as a sum of all the operations that could be run concurrently.

Managing Memory Effectively with ITTIA DB

ITTIA DB SQL is an embedded database library that is specifically designed for devices and embedded systems. For example, memory allocation in ITTIA DB SQL follows the two-phase principle, so that memory requirements are consistent and predictable.

ITTIA DB SQL also includes a built-in allocator that can be enabled to restrict all database allocations to preallocated segments of memory. The built-in memory allocator has proven limits on memory fragmentation overhead, and provides statistics so that worst-case behavior can be measured for each database-driven application.

Other statistics can also be collected, such as the number of database resource handles opened by the application and the number of locks used to provide safe, efficient shared access. These provide additional insight into application behavior, which can be used to reduce the memory footprint.

Conclusion

Memory allocation behavior can have a significant impact on the performance and reliability of an embedded device. Extreme measures such allocating all memory statically at compile-time are extremely restrictive, and not necessary if developers are willing to apply some analysis. For software libraries where the worst-case behavior is not clearly defined, applications can run out of memory unexpectedly even with a bounded memory allocator. An embedded database that provides robust memory management features, like ITTIA DB SQL, can be used to limit and analyze the most dynamic allocations in a device application.

ITTIA DEVELOPMENT TEAM

Support

Because sometimes, you just need some.

The iXsystems Professional Services Team is comprised of veteran FreeBSD® developers, administrators, and project committers.

Our expert staff provides unparalleled levels of support for FreeBSD®, PC-BSD® and FreeNAS™ ranging from problem resolution to consultation to custom development. For more information on our Professional Services Offering, visit our website at http://www.iXsystems.com/BSDsupport and complete the inquiry form.

iXsystems®

Call iXsystems toll free or visit our website today!

+1-800-820-BSDi | www.iXsystems.com