BSD MAGAZINE

FOR NOVICE AND ADVANCED USERS

NANOBSD AND ALIX

INSIDE

VOL.4 NO.6
ISSUE 06/2011(23)
1898-9144

10 Gig On Board

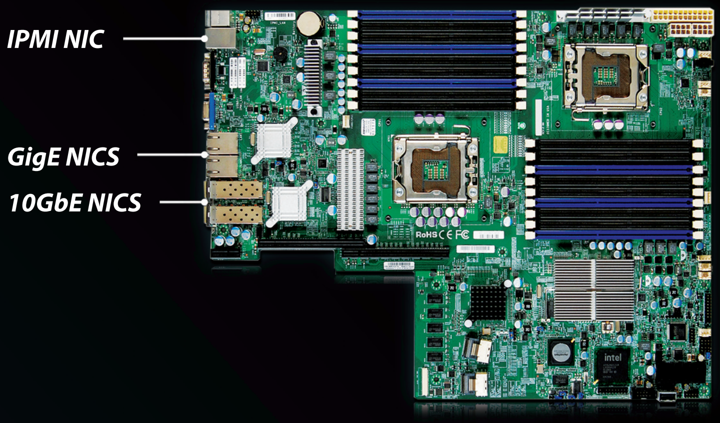
On-Board 10 Gigabit Ethernet Adapters leave your existing PCI-E slots available for other expansion devices.

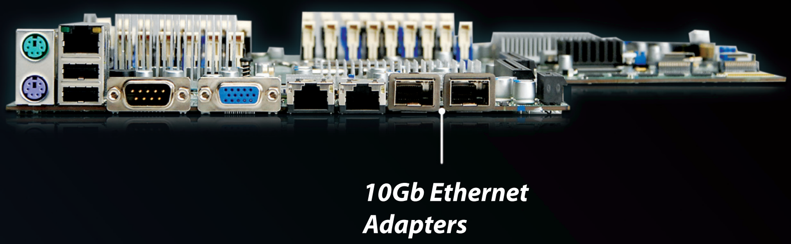
$ 30% cost savings/port over equivalent Dual-Port 10 GB PCI Express add-on card solution

Blazing Fast, Embedded 10Gb Ethernet

10G Rackmount Servers in the iX-Neutron server line feature the Intel® Xeon® Processor 5600/5500 Series, and come with 10GbE networking integrated onto the motherboard. This eliminates the need to purchase an additional expansion card, and leaves the existing PCI-E slots available for other expansion devices, such as RAID controllers, video cards, and SAS controllers.

For more information on the iX-1204-10G, or to request a quote, visit: **http://www.iXsystems.com/neutron**





KEY FEATURES:

- Supports Dual 64-Bit Six-Core, Quad-Core or Dual-Core, Intel® Xeon® Processor 5600/5500 Series
- 1U Form Factor with 4 Hot-Swap SAS/SATA 3.5" Drive Bays
- Intel® 5520 chipset with QuickPath Interconnect (QPI)
- Up to 192GB DDR3 1333/1066/800 SDRAM ECC Registered Memory (18 DIMM Slots)
- 2 (x8) PCI-E 2.0 slots + 1 (x4) PCI-E 2.0 (in x8 slot -Low-Profile - 5.5" depth)
- Dual Port Intel® 82599EB 10 Gigabit SFP+ - Dual Port Intel® 82576 Gigabit Ethernet Controller
- Matrox G200eW Graphics
- Remote Management - IPMI 2.0 + IP-KVM with Dedicated LAN
- Slim DVD
- 700W/750W Redundant AC-DC 93%+ High-Efficiency Power Supply

iX systems®

Call iXsystems toll free or visit our website today!
1-855-GREP-4-IX | www.iXsystems.com

intel® Xeon® inside™
Powerful. Intelligent.

Here it is!

We are back and present you with the June issue of BSD magazine. As you may have noticed, this month's issue is a little bit thicker than usual and we hope you enjoy all the content!
We get started with Ivan Rambius Ivanov and his „Introduction to OpenSSL: Command line Tool" article followed by the What's New section where you will find a short summary of the BSDCan 2011 conference written by Josh Paetzel, and Dru Lavigne who will introduce us to the recently released FreeNAS 8.0.
In the Developers Corner we will meet Antoine Jacoutot describing an OpenBSD success story, Justin C. Sherrill sharing news about the DragonflyBSD project, and Kris Moore who shows us how to install FreeBSD with the PC-BSD installer – PC-SYSINSTALL.
In this months How-Tos Rob Somerville introduces us to GIS in what seems to be a promise of another series of his. Then you will find Sufyan bin Uzayr's article teaching how to deploy a Cloud OS on BSD, and the third part of the Mutt series written by Michael Hernandez.
Also in How-Tos you will find a cover story – „NanoBSD and ALIX" written by Erwin Kooi.
In the Let's Talk section we have articles from Girish Venkatachalam and Mohammed Farrag, as well as a first of BSD Certification articles written by Jim Brown.
In the end we prepared an Interview with Rafał Jaworowski from Semihalf – an embedded hardware and software developing company which makes great use of BSD in their work.

I wish you have a good read and feedback is always welcome!
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:
Ivan „Rambius" Ivanov, Josh Paetzel, Dru Lavigne, Antoine Jacoutot, Justin C. Sherrill, Kris Moore, Rob Somerville, Sufyan bin Uzayr, Erwin Kooi, Michael Hernandez, Girish Venkatachalam, Mohammed Farrag, Jim Brown.

Proofreaders:
Sander Reiche, Christopher J. Umina

Top betatester:
Josmar Calin de Pierri, Daniel Cialdella Converti

Special Thanks:
Denise Ebery, Matt Olander, Rafał Jaworowski

Art Director:
Ireneusz Pogroszewski

DTP:
Ireneusz Pogroszewski

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

CEO:
Ewa Dudzic
ewa.dudzic@software.com.pl

Production Director:
Andrzej Kuca
andrzej.kuca@software.com.pl

Executive Ad Consultant:
Ewa Dudzic
ewa.dudzic@software.com.pl

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*

Introduction to OpenSSL: Command-line Tool

The article describes the command-line utility of openssl. It is a tool that supports encryption and decryption, message digests, key generation and exchange and ssl channel manipulations.

What you will learn…

- basic concepts in cryptography
- Encrypting / Decrypting files with OpenSSL
- Signing and verifying documents

What you should know…

- basic shell scripting

The OpenSSL project is an open-source implementation of SSL and TLS protocols and a general-purpose cryptographic library. Its command-line tool is a client of the library and is widely known as it is used to create SSL keys and certificates for web servers. It supports many other cryptographic operations which makes it versatile, but also complicated with more than 100 subcommands and a myriad of options.

OpenSSL comes preinstalled in all base systems of BSDs. In case the preinstalled version is outdated you can install a newer one as a port or package. FreeBSD and NetBSD have it under security/openssl in the ports and pkgsrc trees.

To verify your installation of openssl execute the command shown on Listing 1.

Basic Terminology

Cryptography is the practice and study of hiding information. It studies the schemes of converting some original intelligible data into some unreadable data. The schemes usually take as input a *plaintext*, feed it to an *encryption algorithm* along with some *keys* and produce a *ciphertext*. A *decryption algorithm* reverts the ciphertext to the plaintext.

The secrecy of the scheme must depend only on the secrecy of the keys and not on the secrecy of the algorithm.

An attack on a cryptographic system is an attempt to compute the plaintext or the keys of the system. The attacker is assumed to know the encryption and the decryption algorithms and to possess a considerable number of ciphertexts.

The attacks can be ciphertext-only, known-plaintext, chosen-plaintext and chosen-ciphertext. The *ciphertext-only* one is the *default* – the attacker is supposed to eavesdrop on the communications and capture the ciphertext – otherwise with no possibility of eavesdropping, there would be no need to encrypt the data in the first place. In the *known-plaintext* attack the attacker knows both some plaintexts and their corresponding ciphertexts. The *chosen-plaintext* one is similar, but the attacker knows the ciphertexts of the plaintexts she chooses. In the *chosen-ciphertext* attack the attacker knows the ciphertexts of chosen plaintexts and the plaintexts of chosen ciphertexts. Finally, there is also the *brute-force* attack in which the attacker tries all possible keys in order to find the one used in the scheme.

If one key is used for encryption and decryption, we speak of *symmetric encryption*. If one key is used for encryption and another for decryption, we speak of *asymmetric encryption*.

In case of symmetric encryption the key, called *symmetric* or *secret* or *shared*, must be exchanged between the two parties before the communication begins. In asymmetric encryption schemes each party has *a pair of keys* – one *private* and one *public* key. The private key must be kept secret and should not be shared with anyone, including any other participant in the communication. The public key can be distributed freely.

Symmetric Ciphers

OpenSSL supports a wide variety of symmetric ciphers. We can obtain the full list of the subcommands that do symmetric encryption with the command from Listing 2.

The OpenSSL command that encrypts and decrypts data is `enc`. By default it encrypts the input and when it is given its `-d` option it decrypts it. The cipher is given as another option and can be any name shown by *list-cipher-commands*.

The symmetric ciphers can be of two types: block or *stream*.

The stream ciphers generate a *keystream* of random data from a seed and usually XORs it with the plaintexts. RC4 is a popular stream cipher and Listing 3 shows how to encrypt with it. We decrypt by passing the `-d` option of the command `enc`, see Listing 4.

The block ciphers work on fixed-sized blocks of input. When we want to encrypt a plaintext P that is longer than that size we split it into blocks $P_1P_2...P_n$. If the data's length is not a multiple of the block's length, we may need to pad the last chunk P_n with additional bytes in order to form a block and then we encrypt the P_i's to ciphertext blocks $C_1C_2...C_n$.

The block ciphers can operate in several *modes* depending on how they relate the different plaintext and ciphertext blocks. The modes can be *Electronic Codebook* (ECB), *Cipher Block Chaining* (CBC), *Cipher Feedback* (CFB) and *Output Feedback* (OFB) and *Counter* (CTR). We will quickly describe ECB and CBC.

ECB simply encrypts each block of plaintext separately:

$$C_i = E(K, P_i), \text{ for } i = 1, ..., n,$$

where K is the shared key. This mode is vulnerable to dictionary attack because if two plaintext blocks are equal the corresponding ciphertext blocks are also equal and that reveals at least some information about the message.

The CBC mode fixes this vulnerability of ECB by XORing a plaintext block with the previous ciphertext block. For the first plaintext block it uses an initialization vector IV:

$$C_0 = IV$$
$$C_i = E(K, P_i \oplus C_{i-1}) \text{ for } i = 1, ..., n$$

The initialization vector can be chosen in several ways including a randomly generated one for each communication session. The IV has to be known to the decrypting side so the the ciphertext is $C_0C_1...C_n$ instead of $C_1...C_n$ that is, it is one block longer that the plaintext.

Two popular block ciphers are DES and AES. The commands that do DES encryption are `des-cbc` for CBC mode, `des-cfb` for CFB mode, `des-ecb` for ECB mode and

Listing 1. *Showing OpenSSL Version*

```
$ openssl version
OpenSSL 1.0.0d 8 Feb 2011
```

Listing 2. *List Supported Ciphers*

```
$ openssl list-cipher-commands
```

Listing 3. *RC4 Encryption*

```
#!/bin/sh

echo "Test is a message" > message.txt openssl enc
                    -rc4 -salt -in message.txt -out
                    message.rc4 \
-pass pass:testpass
```

Listing 4. *RC4 Decryption*

```
#!/bin/sh
openssl enc -d -rc4 -in message.rc4 -pass pass:testpass
```

Listing 5. *DEC ECB Encryption*

```
#!/bin/sh
echo "This is a message" > message.txt openssl enc -des-
                    ecb -salt -in message.txt -out \
message.desecb -pass pass:testpass
```

Listing 6. *DEC ECB Decryption*

```
#!/bin/sh
openssl enc -des-ecb -d -in message.descbc -pass \
pass:testpass
```

Listing 7. *DEC CBC Encryption*

```
openssl enc -des-ecb -salt -in message.txt -out \
message.descbc -pass pass:testpass
```

`des-ofb` for OFB mode and des is an alias for `des-cbc`. We encrypt and decrypt with any of these ciphers in a similar way as with rc4 in Listings 5-8.

Initially we were speaking about symmetric keys but the above commands contain no keys – only passphrases in `-pass` option. OpenSSL can take passphrases from the command line as shown, from a file, from an environment variable, from a file descriptor or from stdin. Once it has a passphrase it can derive a stronger key from it by salting and stretching techniques (we will explain them in the next section). The option `-salt` enables salting and it should always be used unless compatibility with previous versions of OpenSSL is required; in that case `-nosalt` disables salting. We can see the generated salt and key by providing the option `-p`, see Listing 9-10.

The command `enc` has a unified interface to all ciphers and modes, but Listings 9 and 10 show a difference in the internal workings of ECB and CBC modes – the usage of an initialization vector for the latter one. As we mentioned the IV is XORed with the first block.

It is possible to generate the key and encrypt or decrypt separately. The option `-P` prints the key and the IV without encrypting or decryption and they can be used later. Listing 11 contains an example using DES CBC.

The first command prints the generated key, salt and IV. We extract them from its output and we pass them to the next commands using the options `-K` for the key and `-iv` for the initialization vector. Thus if we have a key and a IV we don't need to provide a passphrase. The block size and the key length of the DES algorithm are 64 bits. Because of the short key the algorithm is considered insecure nowadays and several replacement algorithms were proposed. One of them is AES. Its block size is 128 bits and the key length can be 128, 192 or 256. Similarly to DES it can work in CBC or ECB mode and the options of the command `enc` that produce AES are `aes-128-cbc`, `aes-128-ecb`, `aes-192-cbc`, `aes-192-ecb`, `aes-256-cbc`, `aes-256-ecb`, where the numbers in the names are the length of the keys. Here are examples of AES in which we print the generated keys to get a notion of their lengths: see Listing 12-15.

TripleDES is another algorithm that is based on DES. It applies DES three times to each block thus increasing its strength. It first encrypts the input with one key, then decrypts it with another, then encrypts with a third:

$$C = E(D(E(P,\ K_1),K_2),K_3)$$

If the first and the third key are equal, we speak of Two-key TripleDES, otherwise we speak of Three-key TripleDES. The options of the `enc` command that produce TripleDES are `des-ede` and `des-ede3`. The reader can readily con- struct examples that use this cipher in different modes following the listings above and the general format of `enc`.

Message Digests

Message digests take a variable-sized input and produce fixed-size output, called a hash or a digest. It is computationally difficult to restore the original message from the hash or to find a collision – that is another message with the same hash. These algorithms also follow some statistical

Listing 8. *DES CBC Decryption*

```
#!/bin/sh openssl enc -des-ecb -d -in message.descbc -pass \
pass:testpass
```

Listing 9. *Encrypt and Print Key and Salt with DES ECB*

```
$ openssl enc -p -des-ecb -salt -in message.txt -out \
message.descbc -pass pass:testpass
                   salt=903B88D249DEC0B4
                   key=83277DDED6E54786
```

Listing 10. *Encrypt and Print Key Salt and IV with DES CBC*

```
$ openssl enc -p -des-cbc -salt -in message.txt -out \
message.descbc -pass pass:testpass
                   salt=633978D05BFD58BF
                   key=50266891D5AB8100 iv
                   =030447613BBE6A9F
```

Listing 11. *Generate Key and IV*

```
#!/bin/sh
echo "This is a message" > message.txt
s='openssl enc -P -des-cbc -salt -passpass:testpass'
key='echo $s | sed -n -e "s/salt=.* key=\(.*\) iv=.*/\1/p"'
iv='echo $s | sed -n -e "s/salt=.*key=.* iv=\(.*\)/\1/p"'
openssl enc -des-cbc -K $key -iv $iv -in message.txt -out \
message.desebc
openssl enc -des-cbc -K $key -iv $iv -in message.desebc -d
```

Listing 12. *Encrypt with AES with 128-bit Key in CBC*

```
$ echo "This is a message" > message.txt
$ openssl enc -p -aes-128-cbc -in message.txt -out \
message.aes128cbc -pass pass:testpass
                   salt=6A2CB7D200C545E7 key=39F2F2FE
                   E1F6B114FF0015B0A89CDF6C
iv =BAE04A6528DBE50C68BC6892F507B2A1
```

requirements as a one-bit change in the input should change on average at least half of the bits of the output.

OpenSSL supports several digest algorithms. The pseudocommand `list-message-digest-commands` on Listing 16 prints the supported digest commands.

Message digests can be used as checksums. People often distribute files over the Internet along with their digests. When one downloads a file and its digest, she computes the digest once again and compares it with the original digest. If they do not coincide, the file must have been corrupted during the download. This scenario works only when the errors are caused by *unintentional noise* in the channel. It does not work when an attacker is actively modifying the file and possibly its digest during the download.

We will develop simple scripts that will generate and verify files'digests. The script that calculates the hashes is fairly simple. It calls openssl dgst with the supplied algorithm and forwards its ouput to a file, see Listing 17.

The verification script has to parse the file with the digests. Each line of this file has the following format:

```
ALG(filename) = digest
```

We will extract the algorithm, the file name and the digest from each line and then re-calculate it and compare it, see Listing 18.

Listing 13. *Decrypt with AES with 128-bit Key in CBC*

```
$ openssl enc -aes-128-cbc -in message.aes128cbc -pass \
pass:testpass -d
This is a message
```

Listing 14. *Encrypt with AES with 192-bit Key in CBC*

```
$ echo "This is a message" > message.txt
$ openssl enc -p -aes-192-cbc -in message.txt -out \
message.aes192cbc -pass pass:testpass salt=29554552A5C915E6
key=60D6DF60C2A9F698900CEA0EE8B6FCFB12F18C39AF98AC6D
iv =3BCDDDF88523C9B0A3E060A8070BA133
```

Listing 15. *Decrypt with AES with 192-bit Key in CBC*

```
$ openssl enc -aes-192-cbc -in message.aes192cbc -pass \
pass:testpass -d
This is a message
```

Listing 16. *List Supported Message Digests*

```
$ openssl list-message-digest-commands
md4 md5 mdc2 rmd160 sha sha1
```

We use sed to replace the different field delimiters with spaces and cut to extract the fields. When we have the filename we calculate its digest and we compare it with the one in the checksums file.

Cryptographic hashes are also used for password storage. We keep the passwords'digests instead of the plaintext. When users log in we hash the password they input and compare it with the digest in the storage. If they match, the user has given the correct password. If an attacker steals the passwords database it will be difficult for her to restore the

Listing 17. *Digests Calculation on Files*

```
#!/bin/sh
alg='md5'checksums_file='checksums'
while getopts a:c: o
do
case "$o" in
a) alg="$OPTARG";;
c) checksums_file="$OPTARG";; *) exit 2;;
esac
done
shift
$(($OPTIND - 1))
if [ $# -eq 0 ]
then
echo "Provide at least one file to digest" >&2
exit
1 fi
openssl dgst -$alg $* > $checksums_file
```

Listing 18. *Digests Verification of File*

```
while read l
do
pl='echo $l | sed -e 's/\(.*\)(\(.*\)) *= * \(.*\)/\1 \2 \3/'
a='echo $pl | cut -f 1 -d " " | tr "[:upper:]" "[:
                    lower:]"
'f='echo $pl | cut -f 2 -d " "
'd='echo $pl | cut -f 3 -d " "`
cmd="openssl dgst -$a $f"
nl='exec $cmd'pnl='echo $nl | sed -e 's/\(.*\)(\(.*\))
                    *= * \(.*\)/\1 \2 \3/'u8216
 nd='echo $pnl | cut -f 3 -d " "
'if [ "$d" = "$nd" ]
then
echo "Digests for $f match" else
echo"Digests for $f do not match" >&2
fi
done < $digest_file
```

passwords from their digests. However, some information can leak in this scenario. If two users are using one and the same password, their hashes will be the same. The attacker can also attempt to brute force the passwords, so to slow her down we can use salting. A salt is just a random string that is not secret. We append the salt to the password thus stretching it and we hash the whole string. When we have to verify a user's password we take the salt, combine it with the password and generate the hash of the combination and compare it with the hash in the database. The salting eliminates the problem of two users with one and the same password – the salts are different and thus the hashes of the salt-password combinations are different. It also slows down the attacker when brute-forcing passwords. The implementations of salting can differ from this simplistic explanation – they may combine the password and the salt in a different way and they can repeat the process several times.

OpenSSL has a `passwd` command that can generate password hashes with salts. We will demonstrate it by generating a hash as it would appear in BSDs'passwd files. We have a test user called `testuser` with password `testpass` in `/etc/master.passwd` and its hash is shown on Listing 19.

The contents between the first and the second `$` signs means that the hash is generated using MD5, the one between the second and third is the salt and the one after the third is the digest. Let's feed the salt and the original password to openssl `passwd` command. The two hashes in Listing 19 and 20 coincide.

Listing 19. *Testuser Password MD5 Digest*

```
$ sudo cat /etc/master.passwd | grep testuser | cut -d ":" \ -f 2
$1$NjLOxTjY$b9cDDbFUnidKfdOp/V.CH/
```

Listing 20. *Password Hashing*

```
$ openssl passwd -1 -salt NjLOxTjY testpass
$1$NjLOxTjY$b9cDDbFUnidKfdOp/V.CH/
```

Listing 21. *Bob Generates Private Key*

```
$ openssl genrsa -out bob_privkey.pem -passout pass:testpass\
-des3 2048
```

Listing 22. *Bob Generates Public Key*

```
$ openssl rsa -in bob_privkey.pem -passin pass:testpass \
-pubout -out bob_pubkey.pem
```

Asymmetric Ciphers

One of the problems with the symmetric ciphers is the growing number of keys – if n people desire to communicate securely with each other the total number of user keys generates needed a is `n(n-1)/2`. Asymmetric keys – cryptography one public and one solves that problem. Each private. The private is kept secret, the public can be distributed freely. The two keys are related via some mathematical properties but it is computationally difficult to find the private one from the public.

Let's say that Bob has generated his pair and PR_{Bob} is his public key and PU_{Bob} is his public one. Charlie wants to send him a message and he obtains Bob's public key and encrypts the message with it $C = E(M, PU_{Bob})$. Bob receives the cipher text C and descrypts with his private key $M = D(C, PR_{Bob})$. The mathematical properties between the two keys ensure that the holder of the private key can decrypt the message, and if the key is kept secret and not stolen that would be Bob and Bob only.

Here is an illustration of this scheme using the RSA public key algorithm. Bob generates his private key using `genrsa` commands, see Listing 21.

The `out` option specfies where the private key will be stored; if omitted the key will be written on stdout. The `passout` option provides the source of the passphrase that will be used to derive a secret key to symmetrically encrypt the private key; if omitted it will not be encypted. As we said, it is important to protect the private key – encrypting it will at least slow down the attackers if they obtain it somehow. The option `des3` means the key will be encrypted with Triple DES. The final number argument, 2048, is the length of the key modules in bits; generally, the longer the modulus, the better. Its values can be 512, 1024, 2048 or 4096 and at least 1024 is recommended.

Listing 23. *Charlie Encrypts Message with Bob's Public Key*

```
#!/bin/sh echo "Message from Charlie to Bob" > \
message-from-charlie.txt
openssl rsautl -encrypt -pubin -inkey bob_pubkey.pem -in \
message-from-charlie.txt -out message-from-charlie.bin
```

Listing 24. *Bob Decrypts Message with his Private Key*

```
#!/bin/sh
openssl rsautl -decrypt -inkey bob_privkey.pem -passin
                  \ pass:testpass -in message-from-
                  charlie.bin -out \ message-from-
                  charlie-dec.txt
cat message-from-charlie-dec.txt
```

Next Bob generates the public key for the public key for the private in Listing 22. The option `pubout` means that the output will be the public key; if it is not provided the private key will outputted. The option `out`, again, specifies where the public key will be stored. Bob now has his public key and he gives it to Charlie.

Once Charlie has Bob's public key he encrypts his message with `rsautl` command, see Listing 23.

When Bob receives the message, he decrypts it with his private key: see Listing 24.

If Bob wishes to send a secure response to Charlie he can encrypt it in the same way with Charlie's public key.

A private key can be used as an identification or as an authentication. Let's consider that Bob wants to prove that a document comes from him and not from someone else. In this case he can sign the document with his private key. Then everyone else can verify the signature with Bob's public key, see Listing 25.

When Charlie receives the message he can verify that it comes from Bob, see Listing 26.

We should note that the asymmetric operations are computationally intensive. For performance reasons in the case of signing the hash of the data is signed, rather than the original data itself. The above examples are only illustrative and `rsautl` should not be used for signing a large amount of input. We can use the command `dgst` to sign the hashes and to verify the signature, see Listing 27.

The command generates the SHA1 digest as specified by `-sha1` option – any digest command names can be used instead of it. Then it signs the hash using the private key from `-sign` option and outputs in the file given in `-out` option. The final argument is the file whose signature is created. Bob can then send the signature and the message to Charlie.

Charlie verifies the signature with the `dgst` command as well, see Listing 28. The option `-verify` holds the public key that will be used to verify the signature. The -signature option contains the file with the signed hash and the last argument is the message. If the verification succeeds the message is shown.

The RSA algorithm is convenient because it supports both signing and encryption. The DSA algorithm, on the other hand provides only signing and Diffie-Hellman –

Listing 25. *Bob Signs Message with his Private Key*

```
#!/bin/sh echo "Signed message from Bob to Chrarlie" > \
message-from-bob.txt openssl rsautl -sign -inkey
                    bob_privkey.pem -in \
message-from-bob.txt -out message-from-bob.bin
```

Listing 26. *Charlie Verifies Message with Bob's Public Key*

```
#!/bin/sh openssl rsautl -verify -pubin -inkey bob_
                    pubkey.pem -in \
message-from-bob.bin -out message-from-bob-verified.txt
if [ "$?" -eq "0" ]
then
echo "Verification successful"
cat message-from-bob-verified.txt else
echo "Verification failed"
fi
```

Listing 27. *Bob Signs the Hash of the Message*

```
#!/bin/sh
echo "Signed message from Bob to Chrarlie" > \
message-from-bob.txt openssl dgst -sha1 -sign bob_
                    privkey.pem -passin \
pass:testpass -out message-from-bob.sha1 \ message-
                    from-bob.txt
```

Listing 28. *Charlie Veri?es the Hash of the Message*

```
#!/bin/sh
openssl dgst -sha1 -verify bob_pubkey.pem -signature \
message-from-bob.sha1 message-from-bob.txt
if [ "$?" -eq "0" ]
then
cat message-from-bob.txt
fi
```

Listing 29. *Bob Generates DSAP arameters and Keys*

```
#!/bin/sh
openssl dsaparam -out bob_dsaparam.pem 1024
openssl gendsa -out bob_dsaprivkey.pem bob_dsaparam.pem
openssl dsa-in bob _dsaprivkey.pem -out bob_
                    dsaprivkey.pem \
-des3 -passout pass:testpass
openssl dsa -in bob_dsaprivkey.pem -pubout -out \
bob_dsapubkey.pem -passin pass:testpass
```

Listing 30. *Bob Signs the Message with DSA Private Key*

```
#!/bin/sh
echo "Signed message from Bob to Charlie"> \
message-from-bob.txt
openssl dgst -dss1 -sign bob_dsaprivkey.pem -passin \
pass:testpass -out message-from-bob.sha1 \
message-from-bob.txt
```

only *key agreement*. These two algorithms require some parameters to be generated before their usage and that makes them more complex than RSA. However, they are popular as well and they were not patented while RSA was until 2000. Here is an example of DSA usage. Bob generates his DSA parameters and keys in Listing 29.

Bob first generates the DSA parameters with `dsaparam` and stores them in the file given in `-out` option and the length of their prime and generator is 1024. Next, he creates a DSA private key out of the parameters with `gendsa` commmand. That command can encrypt the private key, but it only accepts the passphrase interactively and it makes it difficult to automate the creating of encrypted DSA private keys. The next command, `dsa`, somehow alleviates this issue – it takes the unecrypted key and encrypts and its `-passout` option accepts the passphrase in the usual for OpenSSL way. The name of the symmetric cipher it uses is given as another option, in our case `-des3`. Finally, Bob generates the public key from the private key.

Once we have the DSA private key we can sign a message with the command `dgst` in the same as we did with the RSA private key, see Listing 30.

We verify the signature in the same way as we did with the RSA public key, see Listing 31.

The `-dss1` option means SHA1 but in this case OpenSSL behaves oddly and insists on naming the digest algorithm in this way.

We will round our treatment with mentioning the Diffie-Hellman algorithm used for key agreement. We said that the public key algorithms are computationally intensive. The performance of the symmetric ones is better but they require the two parties to have a shared secret key. Key agreement means that the parties can securely exchange that secret key over an insecure channel. After they do it they can symmetrically encrypt, send and symmetrically decrypt their messages with the shared key thus securing their communication. The Diffie-Hellman algorithms requires pregenerated parameters similar to DSA. We can generate them with `dhparam` command, see Listing 32.

It generates a new set of DH parameters with a 1024-bit long prime and 2 as a generator and stores them in the file given in out option. 5 can be used instead of 2 as a generator.

Listing 31. *Charlie Verifies the Message with Bob's DSA Public Key*

```
#!/bin/sh
openssl dgst -dss1 -verify bob_dsapubkey.pem -
                    signature \
message-from-bob.sha1 message-from-bob.txt
if [ "$?" -eq "0" ]
then
cat message-from-bob.txt
fi
```

Listing 32. *Generating DH Params*

```
#!/bin/sh
openssl dhparam -out dhparam.pem -2 1024
```

Pseudorandom Numbers

Pseudorandom numbers are widely used in cryptography. As we said the salts are random strings. One way to generate a large prime number is to randomly generate large numbers and test them for primality. The command rand can generate random bytes:

```
$ openssl rand -hex 4
2aa5dc61
$ openssl rand -hex 4
6f7c3504
$ openssl rand -base64 4
kMuVow==
$ openssl rand -base64 8
/C3fMqBavOs=
```

The number argument specifies the number of bytes that will be generated, and the -hex and -base64 means that they will be encoded with hex or base64 formats. If these options are omitted the bytes will be in binary.

Final Words

We provided a lot of command line examples, but we skipped some topics as well – for example, `s_client` and `s_server` commands that implement a generic SSL client and a SSL server. OpenSSL also supports some hardware accelerators and if such is installed it can be used with the `-engine` option. The command `speed` allows us to compare the performance of the different algorithms in general and the performance of the aforementioned hardware accelarators in particular.

If the reader is tempted to learn more, the book *Network Security with OpenSSL* by O'Reilly is a good resource about OpenSSL, although it mainly discusses the API. The official documentation is provided in the man pages as usual and on the web site.

IVAN "RAMBIUS"IVANOV

Ivan „Rambius" Ivanov is a Bulgarian software developer currently based in New York and a member of NYC BSD User Group.

BSDCan 2011

By Josh Paetzel

BSDCan 2011 is over, once again I'm impressed by the turnout and by the conference itself. I get the opportunity to visit many conferences, tradeshows, and other types of events every year, and while I feel that many of them are of marginal value BSDCan is a conference that I always look forward to attending, and always come away from it having gained something of value.

BSDCan is a well organized conference that appeals to developers and administrators of BSD UNIX systems. The conference opened this year with a talk on the history on UNIX by Peter Salus, who has served as the Executive Director of USENIX, the VP of the Free Software Foundation, and has authored books such as *A Quarter Century of UNIX*. While many of the presentations at BSDCan are given by strongly technical individuals, several of them are given by people with very polished presentations and public speaking skills, and the keynote address given by Peter definitely fell in the latter category. Other noteable presentations over the two day span of the conference included *FreeBSD going IPv6 Only* by Bjoern Zeeb, *BhyVE: A native BSD Hypervisor* by Neel Natu, and *Sendmail: History and Design* by Eric Allman. Many of the presentations were very interesting, and unfortunately due to the length of the conference and the number of presentations there were several times when painful decisions had to be made as to which presentation to attend.

The conference proper was proceeded by two days of tutorials, as well as a FreeBSD developers summit. The DevSummit included the yearly bikeshed discussions on topics like *which SCM system should the project use*, as well as some more productive discussions, such as what features to include in the upcoming FreeBSD 9.0-RELEASE. A new approach was tried this year with a Vendor Summit, in which vendors could talk directly with core developers about issues that are pertinent in their worlds. It was refreshing to see that a significant chunk of this time was spent discussing strategies for integrating code from the vendors into FreeBSD.

The BSDA certification exam was given this year during the conference. I was told there was a single individual who took the exam, despite my threats of physical violence with several of the attendees. Clearly I'll need to work on making sure I'm taken more seriously in the future.

All in all, a fantastic time was had by all. A time to learn, a time to trade knowledge and experience, a time to catch up with people over pints at The Royal Oak.

Introducing FreeNAS™ 8.0

On May 2, 2011 the much anticipated redesign of FreeNAS was released. This article introduces FreeNAS 8.0, describing the reasons for its redesign, the current and upcoming features, the graphical administrative interface, and where you can find additional information.

FreeNAS is an embedded open source network attached storage (NAS) system based on FreeBSD. FreeNAS was first released by Olivier Cochard-Labbé in 2005 and was originally derived from m0n0wall (*http://www.monowall.org/*), an embedded firewall based on FreeBSD and released under the BSD license. Due to its ease of use and rich feature set, FreeNAS quickly became a popular open source storage solution and attracted a large and varied userbase.

As features continued to be added over time, the monolithic design of FreeNAS became problematic. There are many different uses for a storage system and by not having a modular design that allowed the user to install just the features they needed, FreeNAS was suffering from feature bloat. It was becoming obvious that a complete redesign was needed in order for FreeNAS to remain a viable NAS solution.

In December, 2009 Olivier announced that FreeNAS 0.7 would be placed in maintenance-only mode as he no longer had the time to devote to further FreeNAS development. Volker Theile, a FreeNAS developer who develops on Debian in his day job, announced that he would fork FreeNAS and his redesign would be based on Debian Linux and released under the terms of the GPLv3 license. The resulting project became OpenMediaVault (*http://blog.openmediavault.org/*). Many FreeNAS users were not pleased about the change of license and the loss of kernel-based ZFS support due to GPL incompatibilities with the CDDL license.

iXsystems (*http://www.ixsystems.com*), a provider of FreeBSD-based hardware solutions and professional support, took the initiative to redesign a version of FreeNAS that remained FreeBSD based and BSD licensed. FreeNAS would be rewritten from scratch using a modular design that would support plugins. This would allow FreeNAS to have a small footprint that was easy to support. It would also allow developers to create and contribute plugins for niche features, allowing FreeNAS's usage cases to grow with users' needs. Work on the new design began in 2010 and FreeNAS 8.0 was released on May 2, 2011.

Differences Between FreeNAS 0.7 and FreeNAS 8.0

Besides the redesign from a monolithic to a modular system, there are many differences between the FreeNAS 0.7 maintenance version and the new FreeNAS 8.0. The most significant changes are as follows:

- FreeNAS is now based on NanoBSD (*http://www.freebsd.org/doc/en_US.ISO8859-1/articles/nanobsd/article.html*), an embedded version of FreeBSD. This means that the operating itself has a small footprint and is easily modifiable through the use of scripts.
- The operating system is now separate from the storage volumes. This means that FreeNAS should be installed on a flash card or USB thumb drive rather than a hard disk as the operating system will *take over* all of the space on the installation device, regardless of its size. This separation offers an advantage: when you upgrade, a copy of the previous operating system and configuration database is saved, allowing you to easily revert back to a previous release or configuration should there be a problem with the upgrade.
- The minimum disk requirement has increased from 128 MB to 1 GB. This is to provide sufficient room to hold a copy of the previous operating system and configuration.

- The goal of the 8.0 release was to design a modular operating system that contains the core features needed by a NAS. Future releases will introduce plugins that provide additional features that are not required by all FreeNAS users. This means that some features–such as UPnP, iTunes/DAAP, and BitTorrent–are no longer in the core of the operating system. Current FreeNAS 0.7 users who require those features should wait for the next version of FreeNAS 8 before they consider upgrading.
- Configuration changes are now stored in a database. This means that you should not make any configuration changes using FreeBSD commands; for example, you should not change the root password using the passwd command or install FreeBSD packages using the `pkg_add` command. FreeBSD commands may still work, but their results will not be written to the configuration database and all such changes will not survive a reboot. All configuration changes should be made using either the GUI administrative interface or the command line console. If you need to add additional software to your FreeNAS system, you should wait for 8.1 (which will support additional software via plugins) and remain on 0.7 for now.
- The graphical administrative interface has been rewritten in Django and is now tree based rather than drop-down menu based. Figures 1 and 2 provide examples of the graphical interfaces for each version.

A comparision chart of feature differences is available from *http://freenas.org/comparison/item/freenas-comparison*.

FreeNAS 8.0 Graphical Interface

The FreeNAS 8.0 graphical administrative interface, seen in Figure 2, is designed to make it easy to configure and monitor a FreeNAS system. This section provides an overview of the options that are available within the tree menu.

NOTE

The 8.0 FreeNAS Guide (*http://doc.freenas.org*) contains screenshots for every screen in the administrative tool, tables that summarize every configurable option, and descriptions for using each component of the administrative interface.

Account

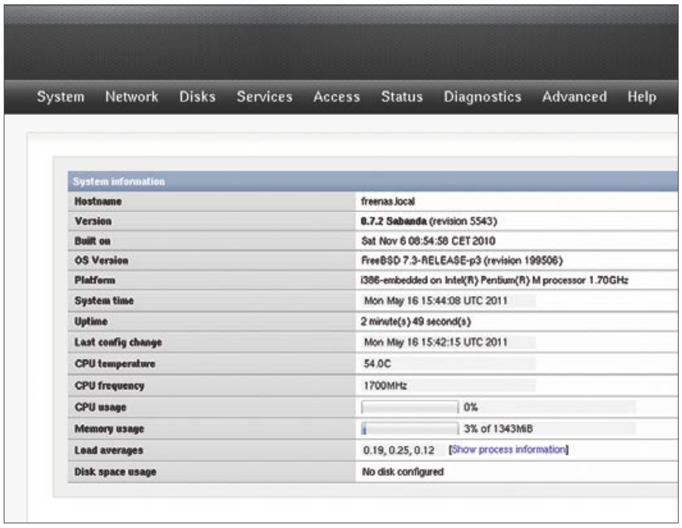
Account used to change the administrative username and password. It can also be used to add users and groups to the FreeNAS system.

System

The Reports tab provides hourly/daily/weekly/monthly/yearly graphs of CPU and RAM usage, system load, swap utilization, and running prcoesses.

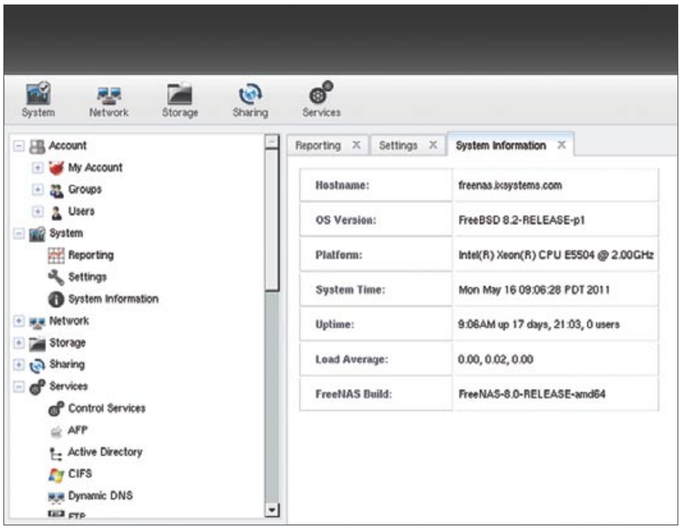
The Settings tab allows you to select HTTP or HTTPs access to the FreeNAS administrative interface, set the localization and time zone, set the NTP server addresses, configure miscellaneous settings such as the MOTD, set the email settings for notifications, upload the SSL key, perform upgrades, and reset the configuration to the initial defaults.

The System Information tab shows information such as the FreeNAS version, processor, hostname, and uptime.



System information	
Hostname	freenas.local
Version	**0.7.2 Sabanda** (revision 5543)
Built on	Sat Nov 6 08:54:58 CET 2010
OS Version	FreeBSD 7.3-RELEASE-p3 (revision 199506)
Platform	i386-embedded on Intel(R) Pentium(R) M processor 1.70GHz
System time	Mon May 16 15:44:08 UTC 2011
Uptime	2 minute(s) 49 second(s)
Last config change	Mon May 16 15:42:15 UTC 2011
CPU temperature	54.0C
CPU frequency	1700MHz
CPU usage	0%
Memory usage	3% of 1343MiB
Load averages	0.19, 0.25, 0.12 [Show process information]
Disk space usage	No disk configured

Figure 1. *FreeNAS 0.7 Graphical Administrative Interface*



Hostname:	freenas.ixsystems.com
OS Version:	FreeBSD 8.2-RELEASE-p1
Platform:	Intel(R) Xeon(R) CPU E5504 @ 2.00GHz
System Time:	Mon May 16 09:06:28 PDT 2011
Uptime:	9:06AM up 17 days, 21:03, 0 users
Load Average:	0.00, 0.02, 0.00
FreeNAS Build:	FreeNAS-8.0-RELEASE-amd64

Figure 2. *FreeNAS 8.0 Graphical Administrative Interface*

Network

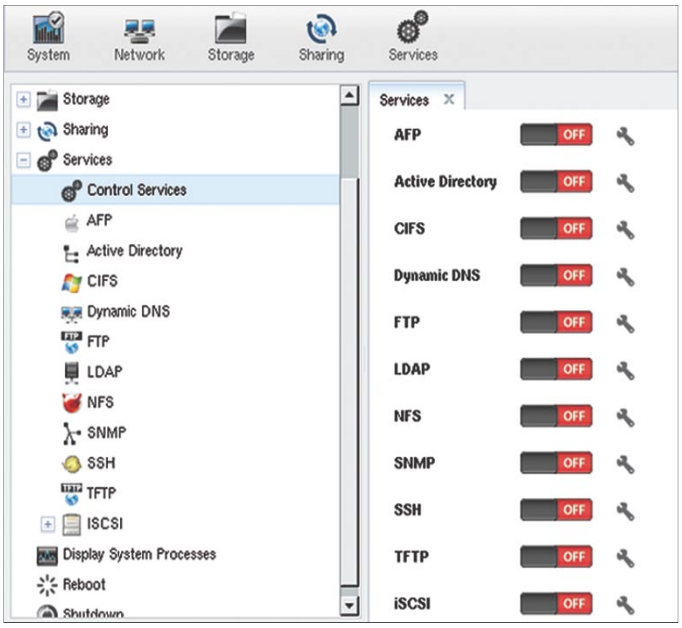
This section is used to view and configure network settings. It allows you to add interfaces, configure them for both IPv4 and IPv6 and to add interface options such as the MTU. The Link Aggregations tab allows you to configure lagg(4) devices which are used to provide link aggregation and link failover using the failover, FEC (Cisco EtherChannel), LACP (IEEE 802.3ad), loadbalance, or roundrobin protocols. The Static Routes tab allows you to view and configure static routes. The VLAN tab allows you to assign a parent interface and VLAN tag ID to a virtual vlan(4) interface.

Storage

The Periodic Snapshot Tasks allows you to schedule the creation of ZFS dataset snapshots, where a ZFS dataset is a section of a volume (e.g. a directory) that can be shared. A snapshot is a copy of a dataset at a specific period in time and can be used to restore or rollback to a saved dataset. The Replication Tasks tab can be used to replicate a ZFS volume to another FreeNAS system over SSH. The Volumes tab allows you to create UFS or ZFS volumes, ZFS datasets, and to import existing UFS, NTFS, or DOS volumes.

Sharing

This section of the administrative interface allows you to create shares to be accessed by Mac, Unix (e.g. Linux or BSD), or Windows clients. The tabs in this section allow you to create and view Apple, UNIX, and Windows shares, as well as configure permissions and common settings



AFP	OFF
Active Directory	OFF
CIFS	OFF
Dynamic DNS	OFF
FTP	OFF
LDAP	OFF
NFS	OFF
SNMP	OFF
SSH	OFF
TFTP	OFF
iSCSI	OFF

Figure 3. *Control Services Screen*

applicable to each type of share. The shares themselves are managed by services: AFP for Apple Shares, NFS for UNIX shares, and CIFS for Windows shares. Additionally, share authentication can be provided by external servers running LDAP or Active Directory. This means that after you create your shares, you should configure and start the applicable share and authentication service(s) in the Services section.

Services

The tabs in this section allow you to configure the following services:

- *AFP:* The Apple Filing Protocol provides file services to Mac computers. This service must be configured and started if you have created Apple shares.
- *Active Directory:* AD provides sharing resources to Windows computers. If your Windows computers are members of an AD domain and you have created Microsoft shares, you will need to configure and start this service. Note that you will need to add a DNS record for the FreeNAS system on the Windows server. FreeNAS 8.0 supports Windows 2000 or 2003 domains; if your AD server is running Windows 2008, make sure that its forest level is set to 2003.
- *CIFS:* The Common Internet File System offers file services within a Windows network. FreeNAS uses Samba to provide CIFS capability without the need for a Windows server in the network. Unix-like systems that provide a CIFS client can also connect to CIFS shares. If you have created Windows shares and do not have a Windows server running AD, you will need to configure and start this service.
- *Dynamic DNS:* DDNS is useful if your FreeNAS system is connected to an ISP that periodically changes the IP address of the system. With dynamic DNS, the system can automatically associate its current IP address with a domain name, allowing you to access the FreeNAS system even if the IP address changes. DDNS requires you to register with a DDNS service such as DynDNS.
- *FTP:* The File Transfer Protocol can be used by clients to browse and download data using their web browser or FTP client software. FTP is considered to be an insecure protocol and should not be used to transfer sensitive files.
- *LDAP:* The Lightweight Directory Access Protocol provides directory services for finding resources, such as users and their associated permissions, in a network. If your network contains an LDAP server,

you should configure and start this service on the FreeNAS system so that it can connect to the LDAP server in order to authenticate users before they access a share.

- *NFS:* The Network File System is a protocol for sharing files on a network. If you have created any UNIX shares, you will need to configure and enable this service.
- *SNMP:* The Simple Network Management Protocol is used to monitor network-attached devices for conditions that warrant administrative attention. Use this tab if you would like to configure bsnmpd(8) on your FreeNAS system.
- *SSH:* Secure Shell allows for files to be transferred securely over an encrypted network. If you configure your FreeNAS system as an SSH server, the computers in your network will need to run SSH client software in order to transfer files using SSH.
- *TFTP:* The Trivial File Transfer Protocol is a light-weight version of FTP usually used to transfer configuration or boot files between machines, such as routers, in a local network. An example where this service is useful is when you wish to store all of the images and configuration files for your network's devices on the FreeNAS system.
- *ISCSI:* This service allow FreeNAS to act like a *storage area network* (SAN) over an existing Ethernet network by exporting disks that iSCSI clients can attach to and mount. iSCSI provides an advantage in an environment that uses Windows shell programs; these programs tend to filter by *Network Location* but iSCSI mounts are not filtered. In order to setup iSCSI, you will need to use this tab to configure authorized users, authorized clients (initiators), device or file extents, iSCSI targets, and to map targets to extents.
- *Control Services:* This tab, shown in Figure 3, displays the running status of each service. By default, every service is disabled; this means that you have to remember to enable a service after you configure it. To do so, simply click the red OFF button next to the service and wait a few seconds until it changes to a blue ON. Should you wish to disable a running service, click its ON button and wait a few seconds for it to return to OFF. The wrench icon next to a service can be used to enter its configuration screen.

Display System Processes
This tab will open up a read-only terminal where you can view the output of the top command.

Reboot/Shutdown/Log Out
Reboot/Shutdown/Log Out: menu items are provided for each of these functions.

Help
Click this icon to see a listing of the FreeNAS help resources. These are described in more detail in the next section.

Finding Additional Information
Several community resources are available to assist you should you have a question regarding FreeNAS 8.0:

Website
The FreeNAS website is located at *http://www.freenas.org*. It contains news and announcements, information about FreeNAS, links to FreeNAS social media sites, documentation, and support resources.

Blog
The official FreeNAS blog is located at *http://blog.freenas.org/*. Subscribe to its feed to keep up-to-date with what is happening with FreeNAS.

Documentation
The FreeNAS Guide (*http://doc.freenas.org*) is the first place to look if you need to know how to configure your FreeNAS system. As new features are added to FreeNAS, they are documented in the Guide. Over time, the Guide will mature into the definitive FreeNAS resource. The Guide is written using a wiki and contributions that address knowledge gaps are welcome. Simply create a wiki login and submit your content; it will be reviewed by editors for technical accuracy and readability.

Mailing Lists
Several FreeNAS mailing lists are available which allow users and developers to ask and answer questions related to the topic of the mailing list. To post an email to a list, you will need to subscribe to it first. Each mailing list is archived, allowing you to browse for information by date, thread name, or author. The following mailing lists are available:

- *announce:* This is a low-volume, read-only list where major milestones, such as new releases, are announced. You can subscribe to the list and access the archives from *https://lists.sourceforge.net/lists/listinfo/freenas-announce*.
- *commit:* This is a read-only list. As code changes in the FreeNAS repository, the commit message is automatically sent to this list. You can subscribe

to the list and access the archives from *https://lists.sourceforge.net/lists/listinfo/freenas-commit*.

- *devel:* FreeNAS developers are subscribed to this list. Technical questions about the current FreeNAS release can be posted here. You can subscribe to the list and access the archives from *https://lists.sourceforge.net/lists/listinfo/freenas-devel*.
- *docs:* This list is for discussion regarding FreeNAS documentation. You can subscribe to the list and access the archives from *https://lists.sourceforge.net/lists/listinfo/freenas-docs*.
- *testing:* FreeNAS developers are subscribed to this list. Technical questions about the upcoming FreeNAS release and feedback on testing snapshots can be posted here. You can subscribe to the list and access the archives from *https://lists.sourceforge.net/lists/listinfo/freenas-testing*.
- *translations:* This list is for discussion regarding FreeNAS localization and translating FreeNAS documentation. You can subscribe to the list and access the archives from *https://lists.sourceforge.net/lists/listinfo/freenas-translations*.

IRC

There is a `#freenas` channel on IRC Freenode if you would like to try asking a question in *real time*. Depending upon your time zone and who happens to be watching the channel messages at the time, you may or may not get an answer right away. Be patient and check back every now and then as no one can answer your question if you disappear from the channel.

Forum

The FreeNAS 8 Forums (*http://forums.freenas.org*) contain information, tips and solutions which can be accessed from a web browser. If you are having problems with something on your FreeNAS system, try using the forum's search utility. You'll often find that someone else has posted a similar question and that other users have responded with a fix or a how-to. The information in the Forums has been categorized, allowing you to browse through posts on a specific topic. Registered users can also ask and answer questions on the Forums.

Bug Tracker

FreeNAS uses a trac database where you can view existing support tickets to see if your issue has already been reported or create new tickets for unreported issues (*http://support.freenas.org/*). You do not need to create a login account in order to view existing tickets, but you will need to use the Register link if you wish to submit a bug report.

Roadmap

The FreeNAS 8.1 Roadmap (*http://doc.freenas.org/index.php/Roadmap_for_8.1*) contains the list of features that are being worked on for the next FreeNAS release. It is a good resource to check if your favourite feature isn't currently implemented. Some of the features on the Roadmap include:

- migration utility from .7 to 8.x
- rsync over SSH
- SMART monitoring
- more detailed system information
- UPS management
- error reporting and user feedback
- encryption
- network bandwidth reporting
- web server
- 3rd party plugin system
- Unison configuration
- mount management through the graphical interface
- BitTorrent through plugin system
- UPnP/DAAP/DLNA through plugin system

Conclusion

FreeNAS 8.0 represents a significant milestone in the evolution of open source storage solutions. Future releases will be able to leverage its modular design through the use of plugins, making it a NAS suited for any type of storage need.

DRU LAVIGNE

Dru Lavigne is author of BSD Hacks, The Best of FreeBSD Basics, and The Definitive Guide to PC-BSD. As Director of Community Development for the PC-BSD Project, she leads the documentation team, assists new users, helps to find and fix bugs, and reaches out to the community to discover their needs. She is the former Managing Editor of the Open Source Business Resource, a free monthly publication covering open source and the commercialization of open source assets. She is founder and current Chair of the BSD Certification Group Inc., a non-profit organization with a mission to create the standard for certifying BSD system administrators, and serves on the Board of the FreeBSD Foundation.

Looking for help, tip or advice?
Want to share your knowledge with others?

Give us your opinion about the magazine's content and help us create the most useful source for you!

A Puffy In The Corporate Aquarium

Success story: OpenBSD as an Enterprise Desktop

While OpenBSD [2] is well known for its use in infrastructure services (MTA, DNS, firewall...) or appliances, this article will focus on a less known application: the use of OpenBSD as a *Joe User* Desktop.

What you will learn…

- Running unattended OpenBSD installations
- Managing an infrastructure with Puppet
- Replacing Windows with a BSD Desktop

What you should know…

- OpenBSD installation process
- Generic IT services
- Concepts behind the Puppet configuration manager

It will also cover how numerous installations of servers and desktops can be deployed and maintained using puppet [3] from a central location.

Background

For several years at M:tier [4] we have setup complete IT infrastructures running exclusively on OpenBSD, from the entry site firewall to the secretary's workstation.

Our clients are Fortune 500 companies (each operating in totally different and unrelated sectors) which means that:

- We are not setting up systems for small geek-friendly-only companies but for huge ones with a long IT history (some of them are present in more than one hundred countries worldwide)
- we have to comply with very large and complex technical and legal specifications (which is always a challenge when using a non-mainstream operating system)

When it makes sense, any local modifications or enhancements are made so that they can be merged back into OpenBSD.

Of course some developments are very specific to what we do and have no place in the OpenBSD project's CVS tree.

The Big Picture

We are currently managing over 600 users in several locations around the globe (and expecting a large increase in the upcoming months). As mentioned before, all of these locations are fully running under OpenBSD, that is:

- the firewalls PF (packet filter), IPSEC (VPN), CARP (for redundancy – with pfsync and sasyncd)...
- the infrastructure servers DNS, DHCP, TFTP, FTP, HTTP, NFS, LDAP, puppetmaster, Kerberos, Squid Proxy, CUPS print server...
- the desktops (workstations and laptops) The GNOME Desktop [5] and plethora of graphical applications.

95% of the services are redundant (including our puppet distribution setup). Shutting down one server has zero impact production services. Most of the time, we do not use the master-slave feature for a particular service (such as DNS) but rather a multi-master mode where puppet is responsible for distributing the shared files and CARP (the *Common Address Redundancy Protocol*) handles the fail-over at the network layer (e.g. both local DNS servers are masters; when the first one goes down or is unreachable, the second one takes over automatically).

Everything is monitored by Zabbix [6]: there is a Zabbix proxy running on each site that sends its data over an IPSEC VPN to an off-site central Zabbix server. Add to that

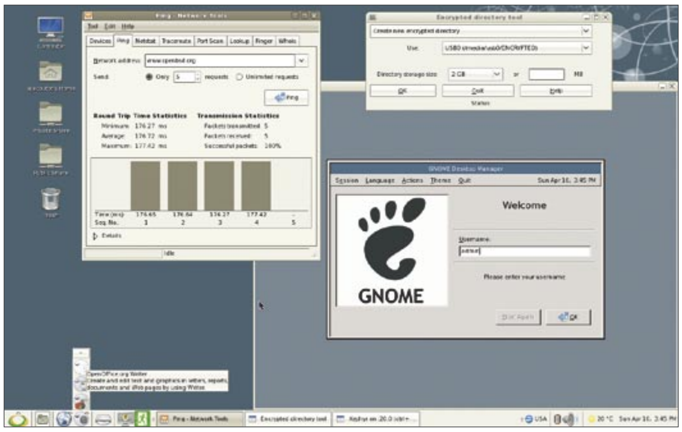


Figure 1. *A typical pre-configured OpenBSD/GNOME Desktop*

our Zabbix mobile client that runs on the iPhone, Android, Blackberry and Windows Mobile 6 and all site managers can see the status of their OpenBSD machines from anywhere.

Another aspect of our setup is that we use exclusively signed packages. This has nothing to do with security but as a solution provider we want to make sure the packages installed on the machines are all coming from us. This feature is available in any standard OpenBSD installation and an HOWTO is available in the OpenBSD FAQ: *http://www.openbsd.org/faq/faq15.html#PkgSig*.

Joe User meets OpenBSD

As a regular user, when the IT staff starts migrating your Windows XP workstation to Windows 7, it is very traumatic! But, you accept it because everyone else in the world is more or less doing the same. But imagine if you had to migrate from Windows XP to Mac OS X! Ok you can also accept it because, well you own an iPod like everyone else, so Apple is known to you.

But the real challenge comes when migrating users from a ten-plus year habit of using Windows and MS Office to an OpenBSD GNOME Desktop with Libre/Openoffice – all without impacting their daily work, aka production, aka company revenue.

The important aspect of such a migration even before considering the new system is obviously information and ... information. A user always feels somewhat punished when his computer work environment changes, so it is very important to explain why it is happening and how in the end, it will serve and benefit him or her. We have a tendency to be on the user's side here; people want to get work done and not learn how computers work.

I believe large Linux migrations face the same issue although people have heard of the operating system, which is not the case with OpenBSD. For most people the critical applications on their desktop are often MS Office and a couple of other editing tools (like Photoshop). The good thing is that several applications that they will use on OpenBSD are available on Windows as well (e.g. Libre/OpenOffice, The Gimp...) so users can get acquainted to these tools in advance and proactively check for issues.

On the technical aspect, our typical Joe User Desktop is made of the following parts:

- a pre-configured GNOME 2.32 environment for the user interface
- Libre/OpenOffice suite for enterprise editing
- OpenNX [7] for accessing Windows Remote Desktop or Terminal services (most companies use internally developed applications running on Windows only)
- CUPS and SANE for printing and scanning (non-networked scanners can be accessed over the network using the saned(8) daemon from the sane-backends package
- the rest of the graphical applications needed for daily office work (web browser, ftp client, multimedia player, picture editor...)

The Desktop configuration in itself is done using a script containing gconftool-2 commands (this script itself is being run by the puppet daemon). This is where the registry-like system that gconf uses can come in handy.

Everything that we provide is available in OpenBSD, except for handful of scripts to ease daily usage of the Desktop. After several years of using Windows, people are used to a certain behavior from their computer and we try to bring that to them (e.g. plugging a USB key will automatically mount it on the Desktop thanks to a hotplug(8) script and open a nautilus window with a right-click menu to unmount the device). We also wrote some graphical interfaces for commonly used tasks like encrypting a USB drive, restarting the network, setting up a PPP/PPPoE/WiFi connection, connecting to the office IPSEC VPN, changing the local and Kerberos passwords... most of them using simple scripting and zenity (a GNOME port of the dialog command).

Lastly, all users' home directories on the laptops are encrypted using standard OpenBSD tools. Note that it is the user's HOME that is encrypted, and not the entire `/home` partition which allows us to use a different pass-phrase for each user.

Everything a user needs is just a click away; he/she is by no means required to ever use the terminal.

Joe User meets tiny OpenBSD

In some situations, a full-blown Desktop is not required. For this particular case, we have developed an OpenBSD-based thin client solution. The thin client is a small

OpenBSD installation with only the base and `x*` sets that runs under a read-only `/` partition. Filesystems needed for normal operation are mounted as mfs in read-write mode (these are `/tmp`, `/home`, `/var/log` and `/var/db`). When the boot process is done, *X.org* and the thin client software (which is a fullscreen `gtk+2` based interface) get started directly from `/etc/ttys`. You can then browse the Internet or connect to a remote NX server. Depending on the configuration IPSEC may be used and you are provided with a button to start the VPN. IPSEC and LAN status are all indicated on the client itself.

The thinclient uses a central update server running puppetmaster on the M:tier side. On the client side, the puppet daemon remounts the root partition in ead-write mode, does its run then remounts it back to read-only.

Note that besides a modified /etc/rc (stripped down to speed up boot time and handle the read-only root partition and mfs filesystems) the thin client is running a generic OpenBSD installation.

Installation, maintenance and configuration

Here is some more detail on the technical side of managing all the different locations.

Installation

The installation process for desktops and servers is the same. At each customer site we have a redundant pxeboot server and puppetmaster that holds all the necessary pieces for an unattended installation.

Listing 1. *Implementation of a dot.plocal overlay directory for Puppet multi_source_template*

```
sources.each do |file|
    Puppet.debug("Looking for #{file} in
                #{environment}")
    if FileTest.exists?("#{file}")
if FileTest.exists?("#{file}.plocal")
    Puppet.info("Found #{file}.plocal in
                #{environment}")
    if Digest::MD5.hexdigest(File.read(file)) ==
                Digest::MD5.hexdigest(File.read("#
                {file}.plocal"))
      Puppet.info("#{file}.plocal is identical to
                #{file} in #{environment} removing
                #{file}.plocal")
      File.unlink("#{file}.plocal")
    else
      file = "#{file}.plocal"
    end
end
```

At all sites, we have a synchronisation job that pulls all the latest packages from our central repository. We are using a fully redundant puppetmaster setup, both servers sharing the same virtual IP over CARP.

Installing a machine becomes a matter of PXE booting on a modified bsd.rd kernel. This bsd.rd automatically partitions the hard drive and installs the required sets from the local FTP store according to configuration files held on the pxeboot server. The OpenBSD infrastructure allows easy modification of the bsd.rd installation kernel to fit anyone's need.

At the end of the installation, the ruby-puppet package is installed and puppetd is started. Puppet will then finish the installation using different manifests and recipes whether we are installing a laptop, workstation, server, etc... Puppet takes care of everything from installing the packages to creating daemon users and configuring the whole setup. It will also run the necessary commands required for proper operation of newly installed packages (*squid -z* to initialize the proxy cache, *aide -i -V0* to create a reference sum of the installation, *update-desktop-database* and *gtk-update-icon-cache* ... for the desktops etc).

Once the installation is finished, the machine automatically reboots and we use `rc.firsttime(8)` to do some interactive post-install administrative tasks (like setting up the root password, registering the machine into the Kerberos realm, etc...).

Due to the high connection rate to puppet (especially during the installation phase where dozen of machines are installed concurrently) we had to start using nginx with mongrel as a front-end for puppet (i.e *puppetmasterd –servertype=mongrel*).

Configuration deployment

For configuration purposes we exclusively use puppet. Amongst the obvious benefit of having a central location for configuration files, it also allows us to setup generic configuration files that will be shared amongst all customers but that we can still make specific by only changing some variables. Puppet variables are stored in LDAP, so it's easy for the local staff to make modifications using a graphical LDAP front-end.

The puppet recipes and manifests are actually contained in a package called mtier-puppet that is automatically installed on the puppetmaster that manages the local site. When a configuration change is requested, we edit the recipe, create a new package which then gets pulled by the requesting site and the puppetmaster does the rest. Obviously, most of the tasks are automated with sanity checks but an interactive manual approval is always needed before deploying a new configuration.

Being written in Ruby, puppet is easily extensible and can be made to do pretty much anything. We are exploiting this feature quite a bit.For instance we use a modified version of the available `multi_source_template` module which makes it possible to create template files with a `.plocal` extension. The .plocal files always get picked first by this function (in case we need a fast way to override the default templates on a machine or at a site).

This is our multi level walk-tree (using `/etc/fstab` for the example):

- `customer/site/machine/etc/fstab.plocal`
- `customer/site/machine/etc/fstab`
- `customer/site/generic/etc/fstab.plocal`
- `customer/site/generic/etc/fstab`
- `customer/generic/etc/fstab.plocal`
- `customer/generic/etc/fstab`
- `generic/etc/fstab.plocal`
- `generic/etc/fstab`

In addition if a .plocal change gets delegated to the original file, and the md5 sum of fstab.plocal equals fstab, the plocal file gets removed automatically.

Here is an extract of our `multi_source_template.rb` function to illustrate this feature (see Listing 1).

We also had to extend the support for OpenBSD packages in puppet like allowing a double `-` in a package name to allow for version-less FLAVORs (OpenBSD concept that is in some ways similar the USE flags in Gentoo Linux, OPTIONS in FreeBSD and *PKG_OPTIONS* in pkgsrc) or forcing the *update* and *updatedepends* flags every time a package gets installed or updated.

Another modification we made was the addition of a *defnode* facter variable during puppetd invocation. Since we have desktop and server machines enlisted in LDAP all of the nodes are getting pulled from there, but we have hundreds of workstations so there is no point in listing them all and we cannot use the default node because that is used to store default settings applicable to all machines. Running the puppet daemon as such allow us to specify the type of machine: `env FACTER_defnode=XXX puppetd...` (where XXX can be *laptop* for example) This feature is now standard in the OpenBSD package and the patch can be seen in the ports tree at: *http://www.openbsd.org/cgi-bin/cvsweb/ports/sysutils/ruby-puppet/patches/patch-lib_puppet_indirector_node_ldap_rb*.

BSD Certification

The BSD Certification Group Inc. (BSDCG) is a non-profit organization committed to creating and maintaining a global certification standard for system administration on BSD based operating systems.

WHAT CERTIFICATIONS ARE AVAILABLE?

BSDA: Entry-level certification suited for candidates with a general Unix background and at least six months of experience with BSD systems.

BSDP: Advanced certification for senior system administrators with at least three years of experience on BSD systems. Successful BSDP candidates are able to demonstrate strong to expert skills in BSD Unix system administration.

WHERE CAN I GET CERTIFIED?

We're pleased to announce that after 7 months of negotiations and the work required to make the exam available in a computer based format, that the BSDA exam is now available at several hundred testing centers around the world. Paper based BSDA exams cost $75 USD. Computer based BSDA exams cost $150 USD. The price of the BSDP exams are yet to be determined.

Payments are made through our registration website:
https://register.bsdcertification.org//register/payment

WHERE CAN I GET MORE INFORMATION?

More information and links to our mailing lists, LinkedIn groups, and Facebook group are available at our website:
http://www.bsdcertification.org

Registration for upcoming exam events is available at our registration website:
https://register.bsdcertification.org//register/get-a-bsdcg-id

On the 'Net

[1] Undeadly OpenBSD Journal *http://undeadly.org/cgi?action=article&sid=20110420080633*
[2] The OpenBSD project *http://www.openbsd.org/*
[3] Puppet Labs *http://www.puppetlabs.com/*
[4] M:tier Ltd *http://www.mtier.org/*
[5] The GNOME Project *http://www.gnome.org/*
[6] Zabbix Open Source Monitoring Solution *http://www.zabbix.com/*
[7] OpenNX (OSS drop-in replacement for Nomachine's nxclient) *http://sourceforge.net/projects/opennx/*
[8] The binpatchng framework *http://binpatchng.puffy-at-work.org/*
[9] Back In Time *http://backintime.le-web.org/*
[10]Bacula Open Source Network Backup Solution *http://www.bacula.org/*

Maintenance

Surprisingly, we are also using puppet for that! The puppet daemon running on all machines will automatically check for packages updates, new antivirus definitions... Antivirus is important because we are interacting with Windows users and the private and public shares that are available to the users over NFS on the LAN are also available via FTP (encapsulated into the IPSEC VPN) from the remote Windows Terminal server.

For base system updates, we use *binpatches* built using the binpatch-ng framework [8]. A binpatch is just a regular OpenBSD package containing the updated binaries that also creates a backup of the modifed files at `pkg_add(1)` time so that the update can be reverted in case something goes wrong. Binpatches are handled using the regular OpenBSD package tools. Backups are handled in two different ways. On the Desktops we use *backintime* [9] which takes an incremental snapshot every hour into a hidden private share on the NFS server. Backintime is really handy for users because it provides an easy to use graphical interface to restore files and directories from a particular point in time (in some ways similar to Apple Time Machine).

For enterprise compliant backup, we use bacula [10] to backup all the servers to a central point (that includes the NFS server, so in fact user data has a double backup). For bare metal recovery, all that is needed is to bootstrap a server from scratch then running bacula to restore everything. Critical files are of course saved off site as well. This dual setup allows for easy access to backup recovery mode while maintaining compliancy with corporate specifications.

Obviously not being physically on site, we need to have local system administrators. Almost none of them ever have heard of OpenBSD before. Most worked exclusively with Windows. To deal with that, we provide them with several days of theoretical and practical OpenBSD training. That part is very interesting, because we get questions and feedback that we would never get from people used to handling Unix systems.

In the end

It usually takes several months for a complete office to finally be accustomed to its new solution but in the end users are very happy with their work environment. It is a source of great pride for us when entering an office and seeing all these users running OpenBSD on their machine. While from a first glance it just looks like another office like we've all seen countless times, when actually paying attention at what is on the screen it is quite different!

While we are facing several social, legal and technological challenges, one of the most interesting ones is the use of OpenBSD on the desktop. Most people use OpenBSD as a server, router, firewall or embedded device but for a desktop it is pretty rarely considered (I'm not including OpenBSD enthusiasts that are the only ones running an OpenBSD workstation at their workplace). That means we are building our experience from scratch because there is not much feedback from anywhere else. From our point of view, with correct knowledge of the available tools, OpenBSD (and by extension, any BSD system) can perfectly fit into the world of the corporate Desktop.

I hope you enjoyed reading this article that I hope uncovered some ideas about how OpenBSD can be used in the enterprise world in a way that is not obviously expected.

This is a reworked and extended version of an original article published by the OpenBSD Journal.[1]

ANTOINE JACOUTOT

Antoine Jacoutot is an OpenBSD committer who lives in Paris, France. He is responsible for more than 300 packages, wrote the sysmerge(8) utility and is part the OpenBSD rc.d(8) framework development. He runs OpenBSD for pretty much everything.
The section describing our puppet setup and changes was written in collaboration with OpenBSD committer Robert Nagy (robert@openbsd.org).

DragonFly News

Google Summer of Code progress

As this article sees print, the first weeks of Google Summer of Code 2011 will be underway. Each of the 6 students involved have posted summaries of their intended work to the main DragonFly development list, *kernel@dragonflybsd.org*. See *http://www.dragonflybsd.org/docs/developer/gsoc2011/* for links to each of the summaries.

Compiler updates, and more

DragonFly has traditionally had two compilers included in the base system, with one considered *standard* and the other as either potentially the next candidate for the default compiler, or as backup. DragonFly recently moved from GCC 4.1 to GCC 4.4 as the base compiler. GCC 4.1 is still present, and can be used for building by setting CCVER. John Marino and Sascha Wildner have been taking this even farther. DragonFly now has binutils 2.20.1, gcc 4.4.6, the gold linker, and GDB 7.2, for instance, along with a host of other program upgrades like texinfo and diffutils, and the removal of ipfilter.

Chromium and LibreOffice support

Matthias Rampke has been working on Google's Chromium browser. Version 11 builds and run on DragonFly, at least for i386 and possibly for x86_64. The changes for DragonFly should soon appear in pkgsrc's wip/chromium package. Francois Tigeot has done the same for LibreOffice, the new fork from Sun/Oracle's OpenOffice software suite. It builds on DragonFly and can be found in wip/libreoffice.

Deduplication

HAMMER in DragonFly 2.10 supports deduplication. It needs to be explicitly enabled with a *dedup* line in the configuration for each Hammer pseudo-file system. It's possible to find out the potential space savings with deduplication by running *hammer dedup-simulate*.

The *dedup-simulate* command will complete quickly and provide a ratio number. This ratio is how well the bulk data on disk overlaps. A ratio of 1.00 means the exact same amount of space would be taken after deduplication as before. The higher the number over 1.00, the greater the deduplication rate. The work described so far is for a batch process that happens with normal overnight batch processing, along with other Hammer cleanup tasks. It is also possible to enable *live* deduplication with the sysctl vfs.hammer.live_dedup. Setting it to 1 will cache data on reads for deduplication, and 2 will enable write deduplication. This will cause the greatest speedup for operations like cp and cpdup, or when creating extensive directory hierarchies.

Multiprocessor changes

DragonFly (and other operating systems at various points) would historically force the user to choose a kernel that would support multiple processors. Uniprocessor kernels on multiprocessor hardware obviously don't perform to the machine's full potential, and multiprocessor kernels would not work on single-processor machines. Various kernel options like IOAPIC would not work in all configurations, either.

Sepherosa Ziehau has been working on support for multiprocessor kernels for any system, regardless of the number of processors on the system. This also allows IOAPIC to work, and the generic (2.11) DragonFly kernel builds with SMP support on by default.

Hammer design

Matthew Dillon has been thinking about how to reach true multi-master disk support in Hammer, and has posted a design document on how to reach this state.

He's calling it *HAMMER2*. *http://apollo.backplane.com/DFlyMisc/hammer2.txt*. This work, as described, retains most of the features of Hammer so far. In addition to the multiple master disks, it adds more features like writable historical snapsnots and quotas.

This project will take some time, though initial code for it should appear in DragonFly 2.11 soon.

JUSTIN C. SHERRILL

Justin Sherrill has been publishing the DragonFly BSD Digest since 2004, and is responsible for several other parts of DragonFly that aren't made out of code. He lives in the northeast United States and works over a thousand feet underground.

Installing FreeBSD with PC-SYSINSTALL

Several months ago, the PC-SYSINSTALL system (The installer for PC-BSD 8.0 and higher) was merged into FreeBSD CURRENT, in preparation for FreeBSD 9.0.

This installer is primarily a backend, but it can also be used stand-alone for multiple scripted or single installations of FreeBSD. In this article we will take a look at how to use PC-SYSINSTALL, as well as various configuration options to take advantage of FreeBSD features which currently cannot be used via the default installer.

Running PC-SYSINSTALL is relatively simple. When booting from a recent FreeBSD / PC-BSD 9-CURRENT snapshot, PC-SYSINSTALL is located in `/usr/share/pc-sysinstall`, and also available as the command *pc-sysinstall* from the command-line. Starting an installation is done by using the following syntax:

```
# pc-sysinstall -c <config file>
```

Once the above command is run, PC-SYSINSTALL will then read the supplied configuration file, setup the disk(s), and begin the installation. This configuration file is where all the magic happens, so lets take a closer look at it (Listing 1).

Above we can see a typical configuration script, which would setup a single-disk FreeBSD system. In the first block of settings we are defining some information about the type of install, as well as source file locations. In this case we are doing a fresh *FreeBSD* installation, from a tar archive on DVD. These settings normally won't change often, assuming you are doing installations from the same type of media. Next we have a few blocks of code dealing with localization, such as time-zones and keyboard layouts.

After this, we come to the disk configuration settings, which is what will most often be changed from system to system. In this example, we are starting by defining which disk is going to be used (`ad0`), that the entire disk will be converted into a single MBR partition (ad0s1) and no boot manager will be installed. The `partscheme= setting` can be toggled between *MBR/GPT*, and if GPT was selected, then the resulting disk would end up with `ad0p1`, `ad0p2`, etc.

With the initial disk options set, next we come to the partition configuration. In this example, we are creating four partitions:

- 1000MB partition, formatted UFS and mounted to `/boot`
- 2000MB SWAP partition
- 2000MB partition formatted UFS with Soft Updates Journaling and mounted to `/`
- 15000MB partition formatted UFS with Soft Updates Journaling and GELI encryption, using the passphrase *mypass* and mounted to `/usr`.

Listing 1. *Example of a complete PC-SYSINSTALL configuration file*

```
#Example pc-sysinstall configuration

installInteractive=no
installMode=fresh
installType=FreeBSD
packageType=tar
installMedium=dvd

# Timezone
timeZone=America/New_York
enableNTP=yes

# Keyboard Layout Options
localizeKeyModel=pc104
localizeKeyLayout=us

# Disk Setup for ad0
disk0=ad0
partition=ALL
bootManager=none
partscheme=MBR
commitDiskPart

# Partition Setup for ad0(ALL)
# All sizes are expressed in MB
# Avail FS Types, UFS, UFS+S, UFS+J, UFS+SUJ, ZFS, SWAP
# UFS.eli, UFS+S.eli, UFS+J.eli, UFS+SUJ.eli, ZFS.eli, SWAP.eli
disk0-part=UFS 1000 /boot
disk0-part=SWAP 2000 none
disk0-part=UFS+SUJ 2000 /
disk0-part=UFS+SUJ.eli 15000 /usr
encpass=mypass
commitDiskLabel

# Root Password
rootPass=mypass

# Users
userName=kris
userComment=Kris Moore
userPass=mypass
userShell=/bin/csh
userHome=/home/kris
userGroups=wheel,operator
autoLoginUser=kris
commitUser
```

Listing 2. *Disk configuration using ZFS*

```
 # Disk Setup for ad0

partition=ALL
bootManager=none
partscheme=MBR
commitDiskPart

# Partition Setup for ad0(ALL)
# All sizes are expressed in MB
# Avail FS Types, UFS, UFS+S, UFS+J, UFS+SUJ, ZFS, SWAP
# UFS.eli, UFS+S.eli, UFS+J.eli, UFS+SUJ.eli, ZFS.eli, SWAP.eli
disk0-part=ZFS 0 /,/usr,/var,/data (mirror: ad1)
commitDiskLabel
```

For systems with multiple disks, both code sections for `commitDiskPart` and `commitDiskLabel` can be repeated multiple times, by changing *disk0›disk1›disk2*, etc and supplying new partition configurations.

Last is the user configuration, with values that are fairly self-explanatory. In this case pc-sysinstall would set the root password to *mypass* and then add a single user *kris* with the specified settings. As with the disk configuration, the code block for `commitUser` could be repeated multiple times, allowing many users to be created at the same time.

This is all that is required in the building of a typical PC-SYSINSTALL configuration script. At this point, the configuration file would be saved to some location on disk, and the command `pc-sysinstall -c <file>` would be run, with no more user interaction necessary.

In addition to supporting disk features such as Journaling, Encryption and others, PC-SYSINSTALL also supports the ZFS file system. Implementing ZFS on disk requires slightly different syntax in order to accommodate the differences in how it is implemented. Lets take a closer look at how this is done in our configuration file (Listing 2).

In this example, the initial disk setup is the same as before, but the partition information is implemented slightly differently. For this disk drive, the configuration `disk0-part=` line instructs PC-SYSINSTALL to create a single zpool (automatically named tank0), use the all available MB (The 0 flag), and create the following ZFS mount points: `/`, `/usr`, `/var`, `/data`. In addition we are able to specify the flag `(mirror: ad1)`, which inserts the disk ad1 into the zpool, using mirroring mode. Options for raidz and others can also be used. Again with this configuration in hand, PC-SYSINSTALL would be able to take these values and handle the rest of the installation unattended.

We have taken a brief look at how a PC-SYSINSTALL configuration file is created and some of the functionality it offers. Since every install is scripted it is very easy to take a preexisting configuration file and tweak it to suite your needs. Common ways to do this are by using examples from `/usr/share/pc-sysinstall/examples/` or by using the PC-BSD GUI installer to generate a config file, which is saved in `/tmp/sys-install.cfg`.

Further Reading and Discussion

Creating automated installations with PC-SYSINSTALL: *http://wiki.pcbsd.org/index.php/Creating_an_Automated_Installation_with_pc-sysinstall.*

Examples and README from SVN: *http://svnweb.freebsd.org/base/head/usr.sbin/pc-sysinstall/examples/.*

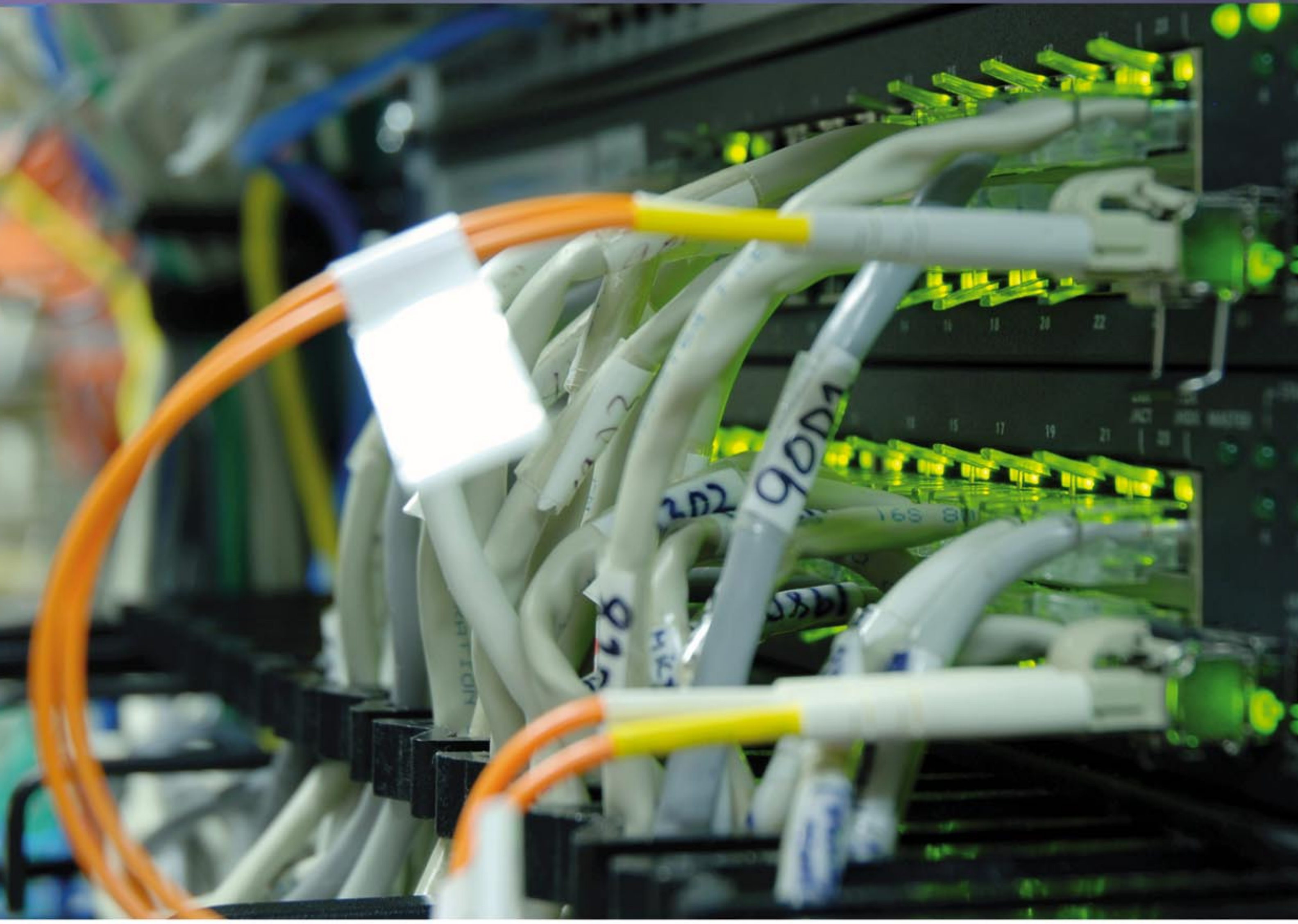
PC-BSD Developers Discussion: *http://lists.pcbsd.org/mailman/listinfo/dev.*

KRIS MOORE

Kris Moore is the founder and lead developer of PC-BSD. He lives with his wife and four children in East Tennessee (USA), and enjoys building custom PC's and gaming in his (limited) spare time. kris@pcbsd.org

EXOnetric

Reliable FreeBSD Jails and hosting at the heart of the UK Internet



Find out what we can do for you today...

Exonetric Consulting Ltd.
Tel. +44 (0) 870 787 9394
Fax. +44 (0) 870 787 9395

www.exonetric.com
info@exonetric.com

An introduction to GIS on FreeBSD

Geographic information systems (GIS) are rapidly gaining popularity both commercially and on the Internet, and used with location aware devices such as mobile phones can be a powerful tool for aiding productivity.

What you will learn…

- How to install and initially configure Geoserver and PostGIS

What you should know…

- BSD administration skills

The downside of GIS systems are their complexity and availability of datasets. Commercial systems are often prohibitively expensive, and licensing or organisational restrictions often prevent the sharing of GIS datasets, so GIS has traditionally been a very specialized area. Large government organisations often have a dedicated GIS department or officer, and when the author was asked to commission a new GIS server, the steep learning curve was quickly apparent especially as the terminology and the varied file formats are complex.

To fully exploit GIS, much time and patience are required but that said, for a capable systems administrator it is relatively straightforward to configure a basic server to serve mapping data. A number of custodians are now releasing their datasets under Open Source licensing, and while the quality and content may vary, this is a welcome move and will no doubt help spread the use of GIS systems where previously the idea would be unthinkable.

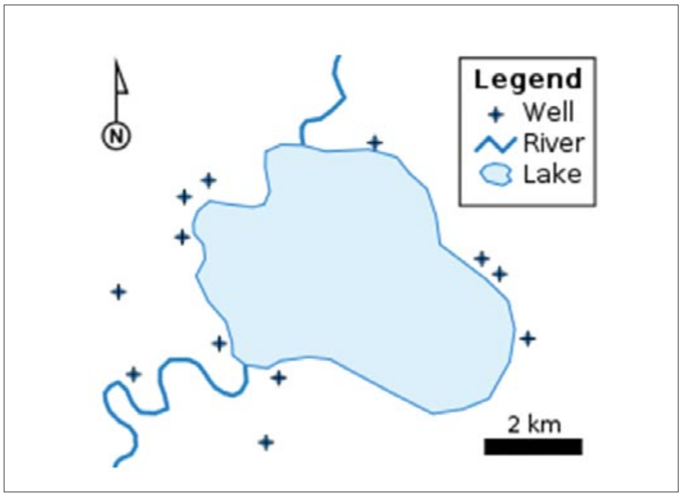


Figure 1. *A simple vector map, using each of the vector elements: points for wells, lines for rivers , and a polygon for the lake*

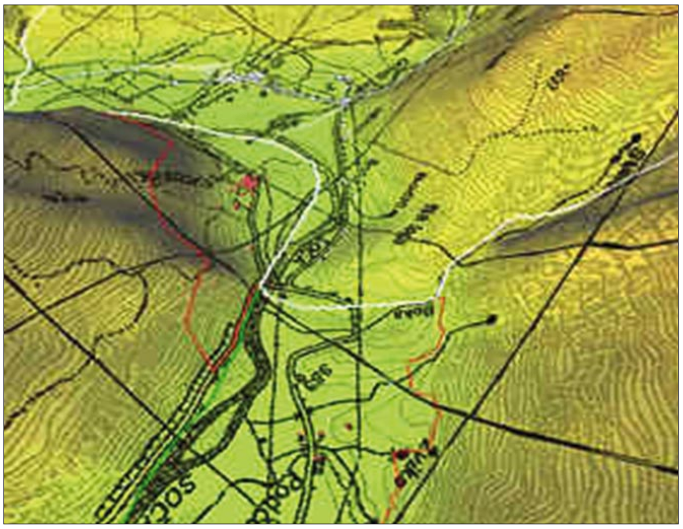


Figure 2. *A typical complex map tile, with elevation, map and vector data*

Basic concepts

To quote Wikipedia, *In the simplest terms, GIS is the merging of cartography, statistical analysis and database technology*. GIS data can be stored in many formats, but the basic concept of presentation is one of tiles and layers. Whereas a static map could be considered one tile (e.g. 640 x 800 pixels), the resolution of that map would be fixed, and if the user attempted to zoom out past the boundaries of the map this would be impossible. On a GIS or web-map system, multiple tiles cover a specific area with a view-port set at a particular resolution. As the user scrolls the map, further tiles come into view effectively stitched together by the mapping software. From the base map, additional data is presented via layers and styles. Additionally, a geographical node can point to data in a separate flat file or database e.g. to display population statistics. To speed up the process and to add additional functionality, a dedicated tile-cache may be used (for example to add watermarks). The tile data is stored in either vector or raster format, depending on the type of map that is to be displayed (Figures 1-3 Images courtesy of Wikipedia).

Data sources come in two flavours, Vector and Raster. Vector data formats include Shapefiles (including directories of shapefiles), PostGIS databases, External WFS layers and Java Properties files. Raster data includes ArcGrid, GeoTIFF, Gtopo30, ImageMosaic and WorldImage. One of the most popular formats are ESRI Shapefiles.

Requirements

In a production environment, 3 servers would be used, one for the mapping software, one for the database and

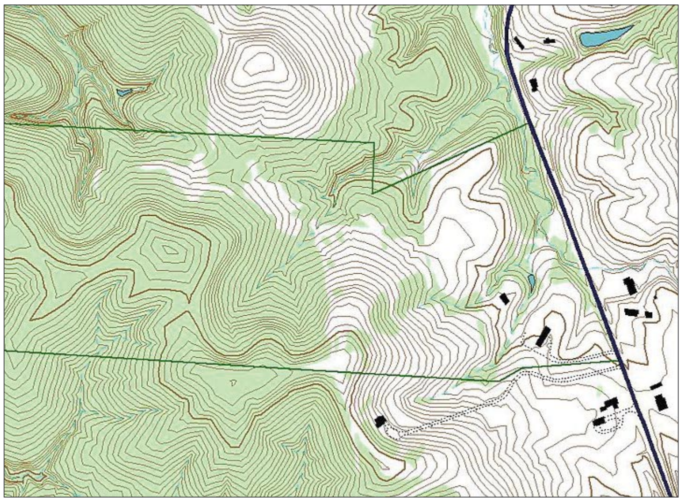
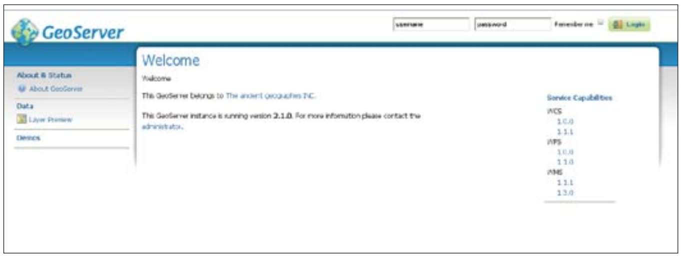


Figure 3. *Example of layers used in GIS work. This map is of an Athens County, Ohio property, and was made using ArcView GIS 3.3, by John Knouse*


GeoServer

username | password | Remember me | Login

About & Status
About GeoServer
Data
Layer Preview
Demos

Welcome

Welcome

This GeoServer belongs to The ancient geographes INC.

This GeoServer instance is running version **2.1.0**. For more information please contact the administrator.

Service Capabilities
WCS
1.0.0
1.1.1
WFS
1.0.0
1.1.0
WMS
1.1.1
1.3.0

Figure 4. *Geoserver login*

one for the tile cache. In this example we will dispense with the tile-cache.

The server will be configured with Tomcat, Oracle JDK, Postgresql and PostGIS, and Geoserver.

Tomcat 7 and Oracle Java JDK 1.6

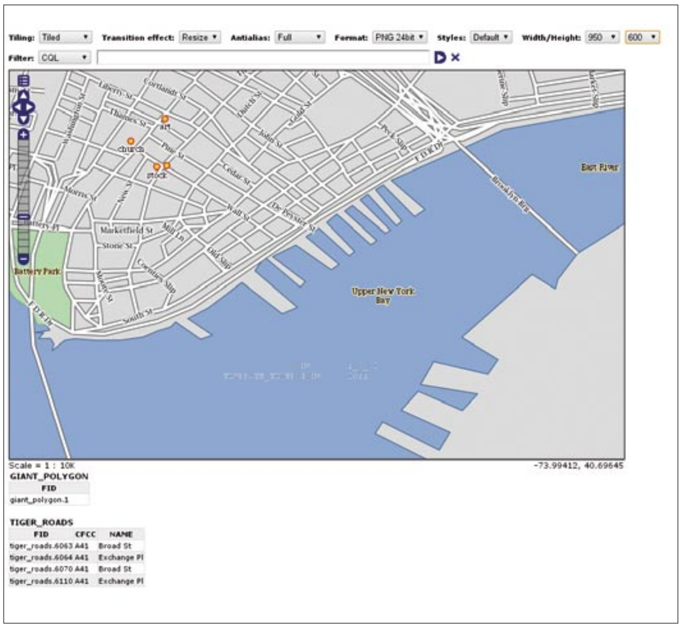
These provide the web-server and runtime JDK environment for Geoserver.

Geoserver

The GeoServer project is a full transactional Java (J2EE) implementation of the OpenGIS Consortiums *Web Feature Server* (WFS) specification. Additionally an OGC *Web Map Server* (WMS) and support for WCS (*Web Coverage Service*) and WMS Raster is provided.

PostGIS

PostGIS adds support for geographic objects to the PostgreSQL object-relational database. In effect, PostGIS *spatially enables* the PostgreSQL server, allowing it to be used as a backend spatial database for geographic



Tiling: Tiled   Transition effect: Resize   Antialias: Full   Format: PNG 24bit   Styles: Default   Width/Height: 950   600
Filter: CQL

Scale = 1 : 10K   -73.99412, 40.69645

GIANT_POLYGON

FID
giant_polygon.1

TIGER_ROADS

FID	CFCC	NAME
tiger_roads.6063	A41	Broad St
tiger_roads.6064	A41	Exchange Pl
tiger_roads.6070	A41	Broad St
tiger_roads.6110	A41	Exchange Pl

Figure 5. *Tiger:ny layer preview with option toolbar shown*

information systems, much like ESRI's SDE or Oracle's Spatial extension.

Installation

Proceed as per normal with a FreeBSD 8.2 installation with the ports tree installed. To install Tomcat 7, this will be compiled from source and additional files are to be downloaded as these are outside the FreeBSD licence. As a prerequisite, I have installed *Midnight Commander* (MC), unzip and Wget to allow quick browsing at the command line as as I installed FreeBSD as a minimal install.

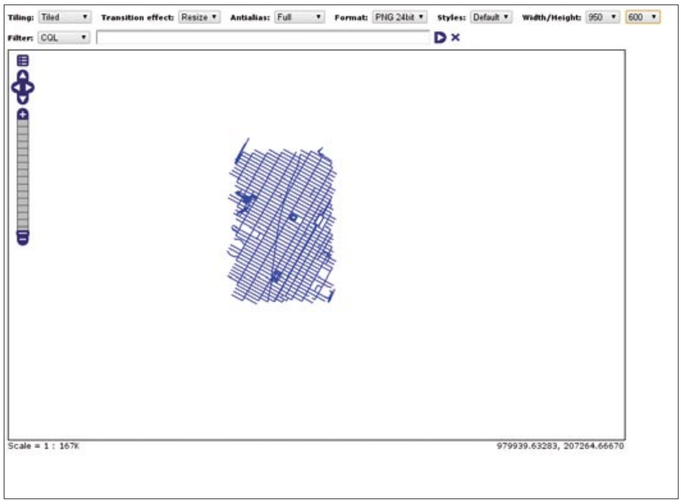
```
pkg_add -r mc rename wget unzip
```

Download the JDK patchset, the JAR files and TZUPDATE into `/usr/ports/distfiles`. The patchset and TZUPDATE need to be downloaded via the web page as licence agreements need to be confirmed, but the remaining JAR files can be retrieved using wget:

```
cd /usr/posts/distfiles
wget http://www.java.net/download/jdk6/6u3/promoted/b05/
jdk-6u3-fcs-src-b05-jrl-24_sep_2007.jar
wget http://www.java.net/download/jdk6/6u3/promoted/b05/
jdk-6u3-fcs-bin-b05-jrl-24_sep_2007.jar
wget http://www.java.net/download/jdk6/6u3/promoted/b05/
jdk-6u3-fcs-mozilla_headers-b05-unix-24_sep_2007.jar
```

You should now have all the files listed in Table 1 installed in `/usr/posts/distfiles`.

As the makefile for JDK6 refers to an older version of TZUPDATE, edit the `/usr/ports/java/diablo-jdk16/Makefile` and change the `TZUPDATE_VERSION` as follows:



Tiling: Tiled   Transition effect: Resize   Antialias: Full   Format: PNG 24bit   Styles: Default   Width/Height: 950 600
Filter: CQL

Scale = 1 : 167K   979939.63283, 207264.66670

Figure 6. *NYC roads*

```
TZUPDATE_VERSION= 1_3_38
TZUPDATE_TZVERSION= 2011e
```

Update the checksum for distinfo:

```
cd /usr/ports/distfiles
sha256 tzupdater-1_3_38-2011e.zip >> /usr/ports/java/
                    jdk16/distinfo
```

Install Tomcat 7:

```
cd /usr/ports/www/tomcat7
make install BATCH=YES
echo 'tomcat7_enable="YES"' >> /etc/rc.conf
/usr/local/etc/rc.d/tomcat7 onestart
```

Once installed, check that Tomcat is running by pointing your browser at port 8080 of the server.

Install Posgresql database, the client and PostGIS bindings:

```
pkg_add -r postgis
echo 'postgresql_enable="YES"' >> /etc/rc.conf
/usr/local/etc/rc.d/postgresql initdb
reboot
```

Table 1. *Additional downloads required for Tomcat install from ports*

Files to be added to /usr/posts/distfiles	tzupdater-1_3_38-2011e.zip bsd-jdk16-patches-4.tar.bz2 jdk-6u3-fcs-bin-b05-jrl-24_sep_2007.jar jdk-6u3-fcs-mozilla_headers-b05-unix-24_sep_2007.jar jdk-6u3-fcs-src-b05-jrl-24_sep_2007.jar diablo-caffe-freebsd7-i386-1.6.0_07-b02.tar.bz2
TZ Updater (Download via browser)	*http://www.oracle.com/technetwork/java/javase/downloads/tzupdater-1-3-38-download-354298.html*
JDK Patchset (Download via browser)	*http://www.eyesbeyond.com/freebsddom/java/jdk16.html*
Fiablo Caffe (Download via browser)	*http://www.FreeBSDFoundation.org/cgi-bin/download?download=diablo-caffe-freebsd7-i386-1.6.0_07-b02.tar.bz2*
JDK src, binaries etc.	*http://www.java.net/download/jdk6/6u3/promoted/b05/jdk-6u3-fcs-src-b05-jrl-24_sep_2007.jar*
	http://www.java.net/download/jdk6/6u3/promoted/b05/jdk-6u3-fcs-bin-b05-jrl-24_sep_2007.jar
	http://www.java.net/download/jdk6/6u3/promoted/b05/jdk-6u3-fcs-mozilla_headers-b05-unix-24_sep_2007.jar

Table 2. *Logins*

Application	Username	Password
Tomcat manager	admin	admin
Geoserver	admin	geoserver

Geoserver is installed by downloading the web archive (Version 2.1.0) from the Geoserver website and copying the extracted files into the webapps directory:

```
/usr/local/etc/rc.d/tomcat7 stop
mkdir /usr/local/apache-tomcat-7.0/webapps/geoserver-2.1.0
cd /usr/local/apache-tomcat-7.0/webapps/geoserver-2.1.0
wget http://downloads.sourceforge.net/geoserver/geoserver-
                    2.1.0-war.zip
unzip geoserver-2.1.0-war.zip geoserver.war
unzip geoserver.war
chown -R www:www *
/usr/local/etc/rc.d/tomcat7 onestart
```

The Geoserver zip and war files should now be moved to another location. Point you browser to *http://yourserver:8080/geoserver-2.1.0* and you should see the Geoserver login. Navigate to layer preview, select Tiger: ny Openlayers and drill down on the map and you will see the corresponding map data. (Figure 4-5).

Adding a shapefile

Copy and extract the shapefiles into the Geoserver tree:

```
cd /usr/local/apache-tomcat-7.0/webapps/geoserver-2.1.0/
                    data/data/
mkdir nyc_roads
cd nyc_roads
wget http://docs.geoserver.org/latest/en/user/_downloads/
                    nyc_roads.zip
unzip nyc_roads.zip
```

This should extract 4 files `nyc _ roads.shp`, `nyc _ roads.dbf`, `nyc _ roads.shx` and `nyc _ roads.prj` into the `nyc _ roads` directory.

Login to Geoserver (admin/geoserver) and create a new workspace called nyc_roads with the name nyc_roads and the URI *http://yourserver/nyc_roads*. Create a new Vector Datasource for the type ESRI Shapefile using the workspace `nyc_roads`. The shapefile to use is `nyc_roads.shp` in the data directory. When saved, you will be prompted to publish the new layer, which you should accept. You will then need to compute the Native/Lat/lon data for the bounding box, if you click on the link it will calculate all the parameters for you. Click on the publishing tab and ensure the default style is line. Save the layer and navigate to layer preview. Clicking on `nyc_roads` should present you with an Openlayers map of the new road Figure 6.

Summary

Setting up a GIS system in a production environment is not for the faint of heart. Depending on the size of the dataset, the geographical area covered, and the operation requirements, numerous servers are required. The current project that the author is working on, there will eventually be 5 servers for mapping a very small area of the UK (less than 50 square miles), and the resulting datasets will be in the order of many *Gigabytes – potentially > 100Gb*. This doesn't take into account meta-data, clustering or load balancing etc. which may be required at a later date, depending on the popularity of the new system.

The biggest problem is getting to grips with the many different standards, technologies, mathematics and *ad hoc* scripts that are used to manipulate and transform GIS data. This is where a dedicated GIS solution from a commercial vendor has advantages – all the hard work has been done for you. However, using Open Source software brings a major benefit (apart from cost) – you will truly learn what is happening *under the bonnet*.

Next article

In the next article, we will look at integrating Postgresql and adding other data formats.

References and further reading

- FreeGIS *http://www.freegis.org/search?q=wms&_ZopeId=99954837A45SCMZdhEQ*
- Geoserver *http://geoserver.org/display/GEOS/Welcome*
- PostGIS *http://postgis.refractions.net/*
- Geoserver Documentation *http://docs.geoserver.org/stable/en/user/*
- Web Mapping O'Reilly Books ISBN 978-0-596-00865-9

Recommended utilities installed via pkg_add

wget	unzip
mc	rename

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.

Exploring The Powers Of The Cloud

Deploying Eyeos On BSD

Ever thought of running things in the cloud? How about doing that from your own server, without any extra effort or cost? We take a look at eyeOS, a cloud OS, and as usual, we do so on BSD.

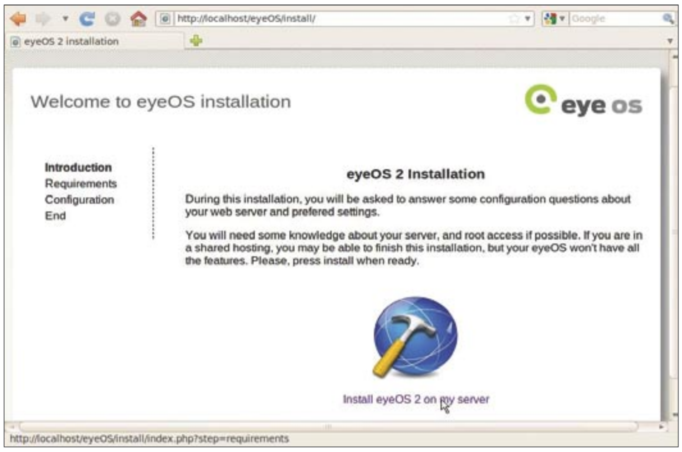
What you will learn…

- Deploying Cloud OS under BSD host

What you should know…

- Basic concepts about cloud computing and deployment, proficiency with the terminal, patience

Yes, I know you've been hearing everyone telling stories about Cloud Computing and how it can do wonders for you. Well, in this article too, we will not ignore the regular introductory part. Cloud refers to provisioning computing as a service, in which you pay for only the service and not the hardware and infrastructure. Naturally, it saves costs for an enterprise by a large margin. Be it Google Apps or even browser-based operating systems; in cloud computing, you let the cloud service provider deal with the infrastructural and hardware related costs. However, from a small or medium sized enterprise's perspective; self-deployment, though not so common, seems to be a better (and arguably cheaper) alternative in the long run. Why? Simply because for average enterprise cloud needs, deployment hardware is nothing out of the blue! The best thing about cloud operating systems is the fact that they can be used from anywhere on the planet, without any concern for synchronization of data across multiple machines. All you require is a computer with an internet connection. However, most of the cloud based operating systems target either the general traveller-type users (journalists, writers, coders, etc) or the high-end enterprises with diversified needs and staffs that span continents. I'm not that rich. Yet, there are a few cloud Operating Systems that cater to the needs of the mid-range enterprises as well and



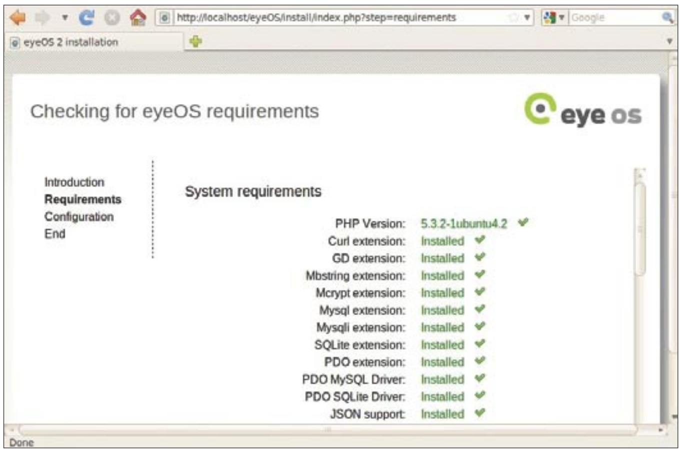
eyeOS 2 Installation

During this installation, you will be asked to answer some configuration questions about your web server and prefered settings.

You will need some knowledge about your server, and root access if possible. If you are in a shared hosting, you may be able to finish this installation, but your eyeOS won't have all the features. Please, press install when ready.

Install eyeOS 2 on my server
http://localhost/eyeOS/install/index.php?step=requirements

Figure 1. *The eyeOS Install Wizard*


http://localhost/eyeOS/install/index.php?step=requirements
eyeOS 2 installation
Checking for eyeOS requirements
eye os
Introduction
Requirements
Configuration
End

System requirements

PHP Version:	5.3.2-1ubuntu4.2 ✓
Curl extension:	Installed ✓
GD extension:	Installed ✓
Mbstring extension:	Installed ✓
Mcrypt extension:	Installed ✓
Mysql extension:	Installed ✓
Mysqli extension:	Installed ✓
SQLite extension:	Installed ✓
PDO extension:	Installed ✓
PDO MySQL Driver:	Installed ✓
PDO SQLite Driver:	Installed ✓
JSON support:	Installed ✓

Figure 2. *Checklist!*

we are going to explore one of them! Not just that. The real potential of cloud computing can be harnessed if you deploy it yourself. Fret not, though. Deploying a Cloud OS may be a complex procedure, but it definitely is not the case if the OS of choice is eyeOS. As we shall soon see, deployment of eyeOS can be done even by those with little hardware resources.

The (Not-so-usual) Details eyeOS – What and why?

To make a long story short, eyeOS is a cloud-based OS powered by Apache, MySQL and PHP.

Desktop

The eyeOS desktop looks like any other operating system that you would come across. It can be customized on the basis of themes, though the looks of Windows Aero are obviously not a possibility. At present, the eyeOS system supports translations in 30 languages.

Productivity

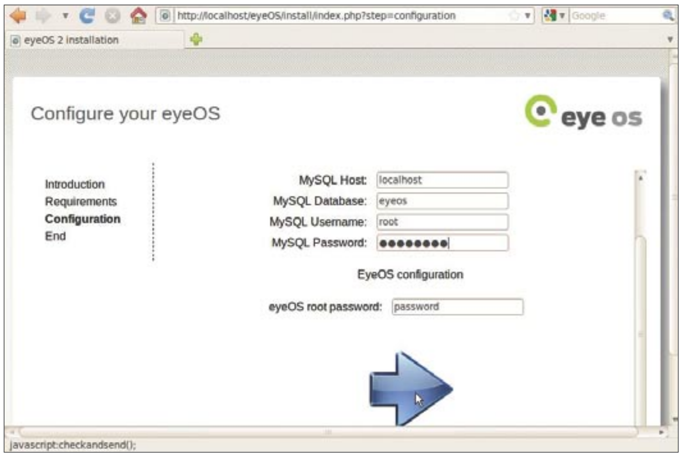
On the productivity front, eyeOS supports MS Office and Open Office documents, spreadsheets and presentations. There is also a *Personal Information Management* (PIM) system with basic support for calendars and contacts (import/export in vCard format). Clearly, the PIM is nothing to drool over.

System

Uploading/downloading files to the cloud is a breeze, and so is browsing pictures using the default system viewing application. Compression support for ZIP/TAR formats is impressive. Additional applications can be installed using the default package manager. Also, multiple instances of an application can run at the same time.

Network

eyeOS has a dedicated (proxy) FTP client, messaging client, RSS Feed Reader and Bulletin Board bundled by default.


http://localhost/eyeOS/install/index.php?step=configuration

eyeOS 2 installation

Configure your eyeOS

Introduction
Requirements
Configuration
End

MySQL Host: localhost
MySQL Database: eyeos
MySQL Username: root
MySQL Password: ●●●●●●●●

EyeOS configuration

eyeOS root password: password

javascript:checkandsend();

Figure 3. *Specifying Db Details And Root Db User*

Installation

Head to *www.eyeos.org/downloads* and download the file. There are two builds of eyeOS at the moment: v1.x and v2.x – choose whichever suits your purpose. In the Figures, we've used eyeOS 2.

In order to run eyeOS as your own server, you will not need anything extraordinarily powerful. The minimum hardware required would be a Pentium-class processor with 256 MB of RAM, 200+ MB free Hard Disk space. The disk space is likely to increase depending on the number of users your enterprise has. So it is quite obvious, that requirements for running eyeOS from your own server are met by almost every mid-sized enterprise.

On the software front, you'll need a BSD flavor (I employed Dragonfly, but you might consider something more robust like FreeBSD) Apache, PHP and MySQL. But before going any further, let's configure PHP and MySQL for eyeOS installation.

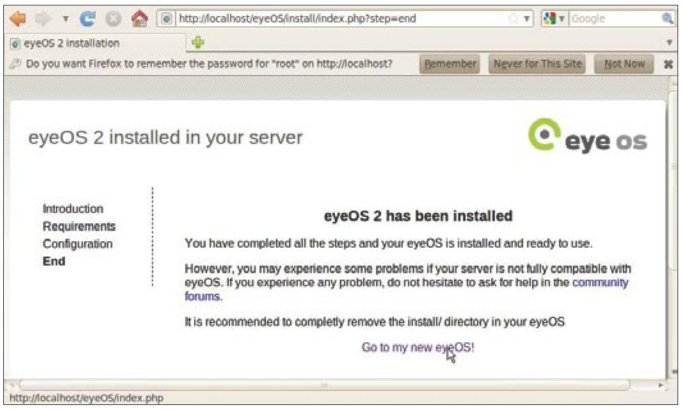
On the software front, you'll need Apache, PHP and MySQL on your OS of choice. But before going any further, let's configure PHP and MySQL for the eyeOS installation. The first step would be to create a database for the eyeOS installation. In the terminal, type the following command:

```
mysql  -u root
```

Enter your password (if any), and at the prompt, type:

```
CREATE DATABASE eyeos DEFAULT
CHARACTER SET utf8
COLLATE utf8_unicode_ci ;
```

Last, run FLUSH PRIVILEGES. That's it, MySQL is ready for action! Next, we need to modify the *php.ini* file (generally located at *php5/apache2/php.ini*) The values, as per the minimum requirements specified above, are as follows (you will probably need to edit the values on the basis of your machine specs):


http://localhost/eyeOS/install/index.php?step=end

eyeOS 2 installation

Do you want Firefox to remember the password for "root" on http://localhost?

eyeOS 2 installed in your server

Introduction
Requirements
Configuration
End

eyeOS 2 has been installed

You have completed all the steps and your eyeOS is installed and ready to use.

However, you may experience some problems if your server is not fully compatible with eyeOS. If you experience any problem, do not hesitate to ask for help in the community forums.

It is recommended to completly remove the install/ directory in your eyeOS

Go to my new eyeOS!

http://localhost/eyeOS/index.php

Figure 4. *Install Complete*

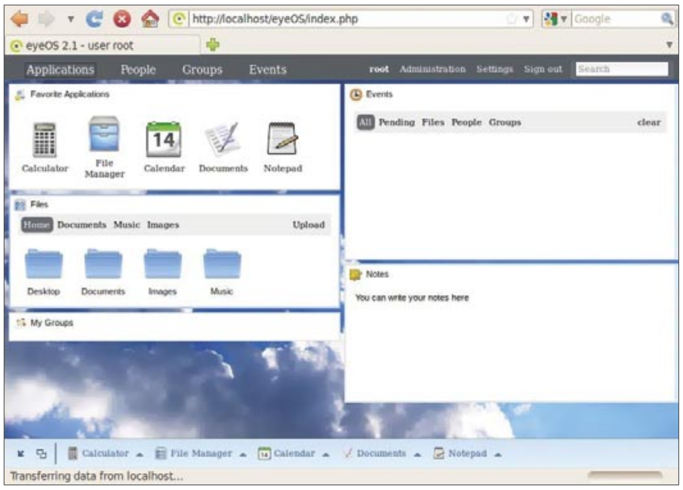


Figure 5. *The eyeOS Desktop*

```
memory_limit = 128M
display_errors = Off
post_max_size = 200M
upload_max_filesize = 100M
```

Note
If your system doesn't already have PHP-Curl, you'll need to install it. Now, *set the </www/>variable AllowOverride to All*. Last, restart Apache.

DEPLOYING eyeOS
Extract the downloaded tarball of eyeOS. Next, in a web browser, navigate to *http://127.0.0.1/eyeos/install* (sample IP, of course). If all goes well, you'll come up with something like Figure 1. Click the *Install eyeOS on my server* link. If there are any red items on the check list, they need to be resolved.

Once all items are green/orange, proceed to the next screen, wherein you'll need to enter details about your database configuration as well specify the password for

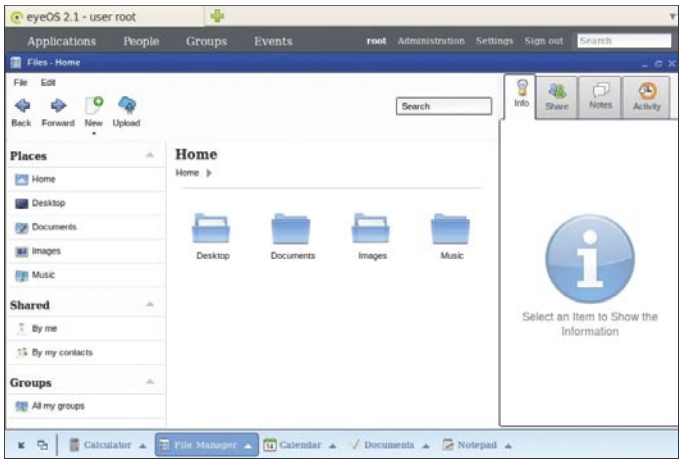


Figure 6. *The eyeOS File Manager*

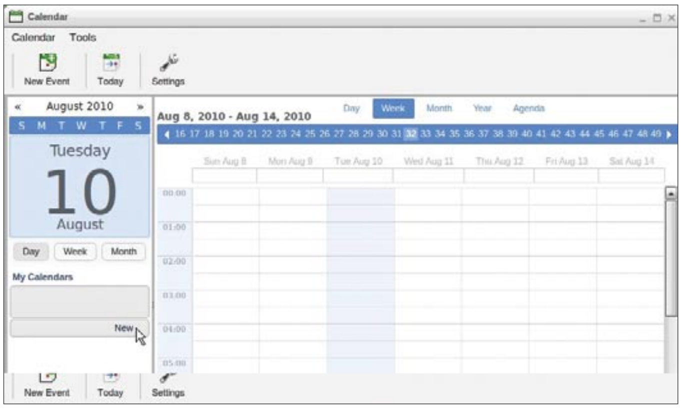


Figure 7. *eyeOS Calendar Pim*

the *root* user in the database. If need be, a new user can also be created using *PHPMyAdmin*.

That's it. You're done with the installation. You'll be greeted with a screen such as this: see Figure 4. With that done, eliminate the *install* directory from *www/eyeos* for security reasons. You can use the root account to create/delete users and perform other administrative and maintenance tasks.

What's Next?
Well, you've just setup a cloud-OS server. Once you kick it on, you will land on the nimble and easy-to-use eyeOS desktop. What this basically means is that you have a fully functional Operating System, bundled with an Office suite and other productivity features for enterprise purposes – all on your own regular piece of hardware, with the only cost of maintenance being that spent on your regular hardware.

eyeOS by default has a powerful File Manager that ensures effective management of your data.

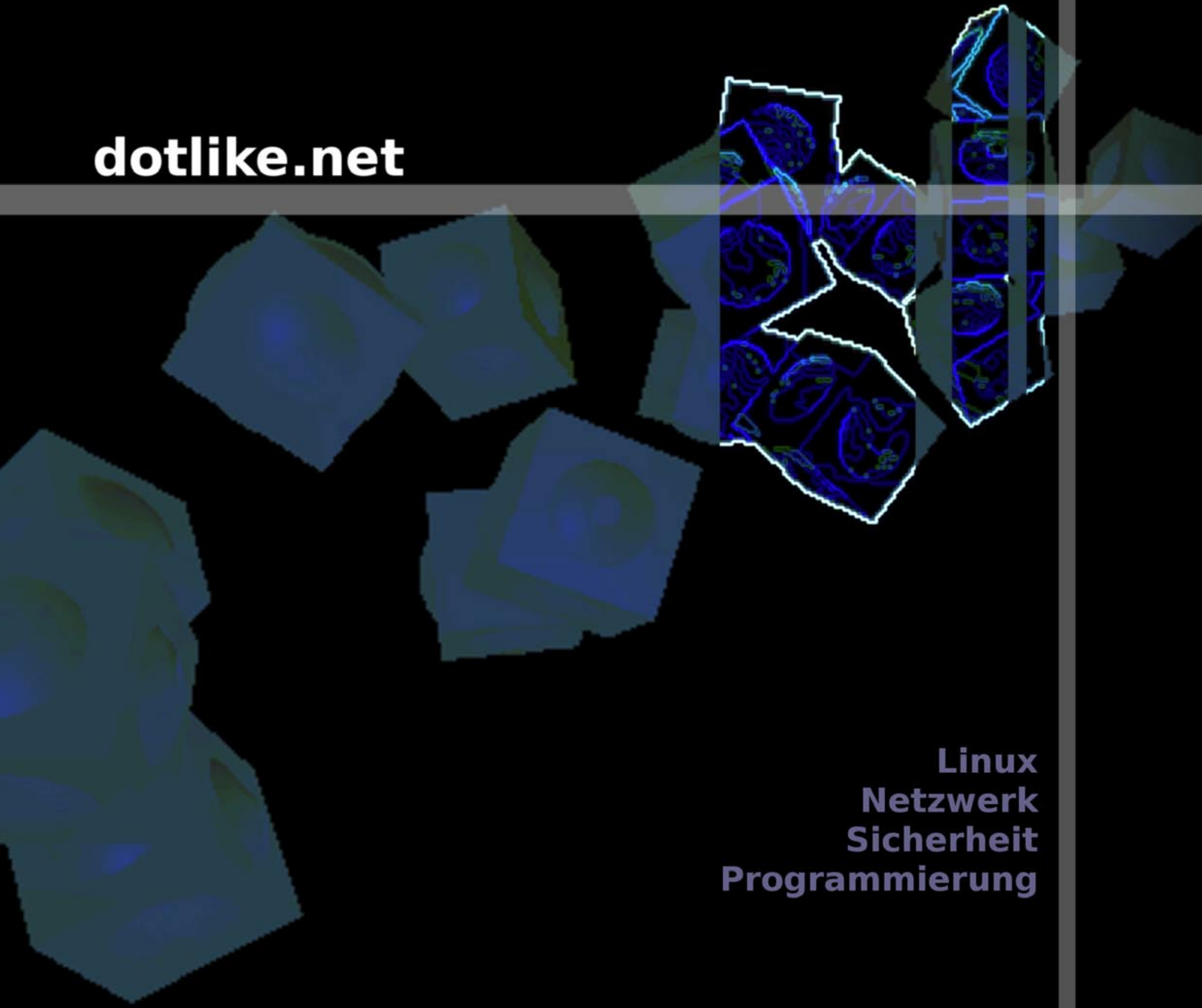
In addition, eyeOS productivity and Office suites cater especially to usage by small firms and medium sized enterprises. There are programs for performing complex data entry, taking notes and other related jobs installed by default. On that note, I leave you to play with eyeOS. Once you implement the extremely simple installation of eyeOS, gear up to see the miracles it can perform for your enterprise.

Note
eyeOS website: *www.eyeos.org.*

SUFYAN BIN UZAYR

Sufyan bin Uzayr is a 20-year old freelance writer, graphic artist, programmer and photographer based in India. He writes for several print magazines as well as technology blogs. His prime areas of interest include open source, mobile development, web CMS and vector art. He is also the Founder and Editor-in-Chief of http://www.bravenewworld.in He can be reached at http://www.sufyan.co.nr

dotlike.net



Linux
Netzwerk
Sicherheit
Programmierung

NanoBSD and ALIX

In the previous issue of BSD Magazine, Bill Harris described how to do a basic installation of FreeBSD on a PC-Engines ALIX board with a Compact Flash card. This is a great way to get started, but there are some risks to this approach.

What you will learn…

- The working of and working with NanoBSD
- The creation of NanoBSD for an embedded system

What you should know…

- Your way around a FreeBSD system
- Basic system administration
- How to compile FreeBSD from source

CF cards can be written to only a limited number of times so putting your `/var` and its logfiles on it, will quickly wear out the card. To adress this issue and others, Poul-Henning Kamp wrote a script called nanobsd.sh. NanoBSD is not a fork from FreeBSD, but an optimized build script for read-only media. This article gives an overview of NanoBSD in general and my setup in more detail.

The working of NanoBSD

When NanoBSD boots, `/` (and its subdirectories like /boot, `/root` and `/usr`) are mounted as a read-only file system, while `/etc` and `/var` are mounted as read-write file systems on memory disks. The content of these memory disks is lost when the power is lost or when the system reboots.

There is a partition on the NanoBSD CF card reserved for the persistent storage of the `/etc` configuration files. This partition is mounted early in the boot process as /cfg and the files are copied to the `/etc` memory disk. During normal operation the `/cfg` partition is not mounted to prevent accidental writes to the configuration files. A number of scripts are used to keep the `/cfg` partition up to date with changed configuration files in `/etc`.

The CF card contains three partitions in total. The p3 partition is used for the persistent storage of configuration files. The p1 and p2 partitions both contain a/file system. This is very convenient for a system upgrade and roll-back as we will see in a later section.

The building of NanoBSD

NanoBSD is built off-line, which means that the preparation, build and installation process has no impact on the build system or the live NanoBSD system. It will produce image files that can be put on the CF card in the NanoBSD system. There are a number of important files and directories.

nanobsd.sh

The script has been located in the FreeBSD source tree since FreeBSD 6.0 and can be found in the `/usr/src/tools/tools/nanobsd` directory. Running the script with no options will produce a disk image with a GENERIC kernel and a complete world. A nanobsd.conf configuration file can be used to tune the build process.

nanobsd.conf

This configuration file overrides defaults that are set in the `nanobsd.sh` script. These defaults include the name, the architecture, the world options and kernel configuration.

It is also possible to add custom script functions in order to tune the system even further.

Adding ports and files

Because the/file system is read-only, ports have to be added during the build process. All port files (distfiles) that reside in the `/usr/src/tools/tools/nanobsd/Pkg` directory are compiled and installed before the image file is created.

The distfiles must be of the same architecture as the target system and do not forget to install all dependencies as well! These dependencies can be found in the port description on the *www.FreeBSD.org/ports* page. Individual files that reside in the `/usr/src/tools/tools/nanobsd/Files` directory are copied before the image file is created.

Build process

With a source tree of the desired FreeBSD version in `/usr/src`, the commands

```
# cd /usr/src/tools/tools/nanobsd
# sh nanobsd.sh
```

will start the build process. The detailed output of the process is written to logfiles and only high-level progress status is written to the screen. All files are located under `/usr/obj/nanobsd.NANOBSD`. The most important files are

- `_.bw` (build world logfile),
- `_.bk` (build kernel logfile),
- `_.iw` (install world logfile),
- `_.ik` (install kernel logfile),
- `_.disk.full` (full disk image for the entire CF card)
- `_.disk.image` (partition image for one partition on an existing NanoBSD CF card)

If something did go wrong, the only indication, is the termination of the script before an image file is created. Reading the logfile of the last reported step will give more info on the exact reason for failing.

If parts of the build process have already been completed before the process failed, these parts can be reused in the new build by specifying command line options to nanobsd.sh:

`-n` – do not clean directories before building
`-k` – do not build kernel
`-w` – do not build world
`-b` – do not build anything

After the image file is created, it is transferred to the CF card using a (generic USB) card reader. CF cards are typically seen as ATA drives, so the device name will be something like ad1 or ad2. Running

```
# dd if=/usr/obj/nanobsd.NANOBSD/_.disk.full of=/dev/ad1 bs=64k
```

transfers the image file to the CF card. `/dev/ad1` is the CF card here. The created partition image can be mounted directly by running

```
# mdconfig -a -t vnode -f /usr/obj/nanobsd.NANOBSD/_
                  .disk.image -u 1
# mount -t ufs /dev/md1a /mnt
```

The `/mnt` directory now contains the root of the CF card's / partiton. Unmount it by running

```
# umount /mnt
# mdconfig -d -u 1
```

Upgrading / Updating NanoBSD

One of the features of NanoBSD is the offline upgrade and roll-back mechanism, allowing for upgrades to the base system with only seconds of downtime.

The nanobsd.sh script generates two image files. One full disk image and one partition image. In the previous steps, the full disk image was used to fill a CF card. On a running NanoBSD system, there is no need to remove the CF card to perform an upgrade.

The second partition can be upgraded while the system is running from the first partition. When the system is ready to be rebooted (in the maintenance window), booting from the second partition will start the upgraded system. Assuming the first partition is the active partition, run

```
# sh /root/upgradep2 < _.disk.image
```

on the NanoBSD system to upgrade the second partition. (`/root/upgradep1` will upgrade the first partition).

The system will be set to boot from this partition. When the system is ready to be rebooted, reboot.

```
# reboot
```

If the upgrade was unsuccessful, simply set the boot partition back to the first partition and reboot.

The system will revert to the not-yet upgraded partition.

```
# boot0cfg -v -s 1 ad0
# reboot
```

The active partition can also be selected using the [*F1*] and [*F2*] keys during startup.

NanoBSD for ALIX

The following chapters will show my build system, configuration files and caveats.

The hardware I used is the PC Engines ALIX 2D13 board. I had it lying around from my tests with `pfSense`. It has the following features:

Listing 1. *The complete nanobsd.conf*

```
NANO_NAME=ALIX    # directory will be /usr/obj/nanobsd.ALIX
NANO_DRIVE="ad0"  # target drive for the CF card is ATA
NANO_ARCH=i386    # architecture
NANO_KERNEL=ALIX  # kernel file
NANO_BOOTLOADER="boot/boot0"
NANO_BOOT0CFG="-o nopacket -s 1 -m 3"  # ALIX boot options
NANO_MEDIASIZE=1981728  # 1Gb Sandisk CF card
NANO_SECTS=63
NANO_HEADS=32

CONF_WORLD='
TARGET=i386
TARGET_ARCH=i386
TARGET_CPUTYPE=pentium-mmx
# WITHOUT_ options can be inserted here
# examples are WITHOUT_BLUETOOTH, WITHOUT_I4B and WITHOUT_PROFILE
'
# This function enables three tweaks for embedded systems
cust_embedded_setup() {
  # turn off ascii beastie as boot menu
  echo 'autoboot_delay="4"' >> ${NANO_WORLDDIR}/boot/loader.conf
  echo 'beastie_disable="YES"' >> ${NANO_WORLDDIR}/boot/loader.conf

  # turn on noatime for /cfg for more performance
  sed -i "" -e "/cfg/s/rw/rw,noatime/" ${NANO_WORLDDIR}/etc/fstab

  # No "message of the day" for me
  rm ${NANO_WORLDDIR}/etc/motd
  touch ${NANO_WORLDDIR}/etc/motd
}
customize_cmd cust_embedded_setup
# We only have a serial port for console
customize_cmd cust_comconsole
# We allow root to ssh directly
customize_cmd cust_allow_ssh_root
# Install files in /usr/src/tools/tools/nanobsd/Files
customize_cmd cust_install_files
# Install packages in /usr/src/tools/tools/nanobsd/Pkg
customize_cmd cust_pkg
```

- AMD Geode LX800 + glxsb hardware crypto
- 256 MB RAM
- 3 x VIA Rhine network interface (vr)
- IDE CF card slot (master)
- IDE 44-pin interface (slave)
- RTC / USB / I2C / Serial ports

An ideal board for a dedicated firewall/VPN appliance and so much more. My build system is a virtual machine on my laptop running FreeBSD 8.2 amd64 on scsi disks. This means we will have to cross compile, as the ALIX board has an i386 architecture and an IDE disk interface.

NanoBSD config

The most important configuration aspects for the ALIX platform are the target architecture (i386), the target drive (ad0) and the bootoptions (-o nopacket).

We also have to specify the size of the CF card in sectors. This can be tricky, as the values of the sectors and heads of the CF card are often reported incorrectly by the various system tools. We start with the number of blocks. Running

```
# diskinfo /dev/ad0
/dev/ad0    512    1014644736    1981728    967    64    32
```

will give a `NANO _ MEDIASIZE` of 1014644736 / 512 = 1981728 blocks. (diskinfo sees 967 cylinders, 64 sectors and 32 heads.) The safes way to fill the `NANO _ SECTS` and `NANO _ HEADS` is to put the CF card in the ALIX board and boot from it (see the booting section below). It will report

```
PC Engines ALIX.2 v0.99h
640 KB Base Memory
261120 KB Extended Memory

01F0 Master 044A CF 1GB
Phys C/H/S 1966/16/63 Log C/H/S 983/32/63
```

So the system thinks the card has 983 logical cylinders, 32 heads and 63 sectors. That's what we have to work with. The complete nanobsd.conf looks like this: see Listing 1.

The `cust_embedded_setup` function will turn off the beastie ascii art, set the boot delay to 2 seconds and remove the motd (settings I like on my headless platforms).

Kernel config

A GENERIC kernel works fine for ALIX boards, but we can tune the configuration for a leaner kernel with hardware crypto enabled. The processor is i586 (pentium-mmx) compatible

Special thanks

- Poul-Henning Kamp for giving us NanoBSD. He doesn't like personal pages, but this is his *real* one: *http://people.freebsd.org/~phk/*
- pfSense for getting me interested in NanoBSD and for providing all my firewall needs: *http://www.pfsense.org*
- Paul Schenkeveld for performing excellent work on the use of NanoBSD and extending the use to regular production servers with ZFS and Jails. *http://www.psconsult.nl/talks/NLLGG-BSDdag-Servers/*
- PC-Engines for their nifty boards: *http://www.pcengines.ch/*

```
cpu       I586_CPU
ident    ALIX
options  CPU_GEODE
```

The network interface is a VIA Rhine, so only the vr and miibus devices are needed.

```
device  miibus  # MII bus support
device  vr      # VIA Rhine, Rhine II
```

Memory disks are an essential part of NanoBSD.

```
device  md  # Memory „disks”
```

The Geode processor has a hardware crypto module, so we need to enable the glxsb, crypto and cryptodev devices.

```
device  crypto     # core crypto support
device  cryptodev  # /dev/crypto for access to h/w
device  glxsb      # AMD Geode LX Security Block
```

Booting for the first time

After creating the CF card an inserting it into the board, connect a serial cable and start a terminal program. I like to use the screen command for this

```
# screen /dev/tty.PL2303 38400
```

(Yes, I use a Prolific serial-to-usb converter here.)

The ALIX boards run 38400,8,n,1 out of the box, so the terminal program has to work at this baud rate. The first thing I do is set it to 9600 baud by hitting the `s` key during the memory test and pressing the `9` key for 9600 baud. Save the config and restart the terminal program

```
# screen /dev/tty.PL2303 9600
```

ALIX boots and we are presented with a choice. [*F1*] for FreeBSD or [*F2*] for FreeBSD. These are the two partitions on the CF card that both contain the / file system.

This is the moment to chose which partition to boot if an upgrade of a partition completely failed.

Configuring the live system

The configuration of NanoBSD is equal to configuring FreeBSD. You can revert to Bill Harris' article in the previous issue for a quick start.

Because the / file system is read-only and the /etc file system is a memory disk, it is very important to sync the configuration files in `/etc` to `/cfg` after changes. There is a number of scripts in the `/root` directory to help with this synchronisation.

```
change_password  change the root password and sync it to /cfg
save_cfg          sync changed files in /etc to /cfg
save_sshkeys     sync changed ssh keys in /etc/ssh to /cfg/ssh
updatep1          update the first partition with a new
                     partiton image
updatep2          update the second partition with a new
                     partition image
```

Paul Schenkeveld wrote an excellent sync script called cfgsync that will automatically sync all content of `/etc`, including subdirectories (see the special thanks section below).

Where do we go from here?

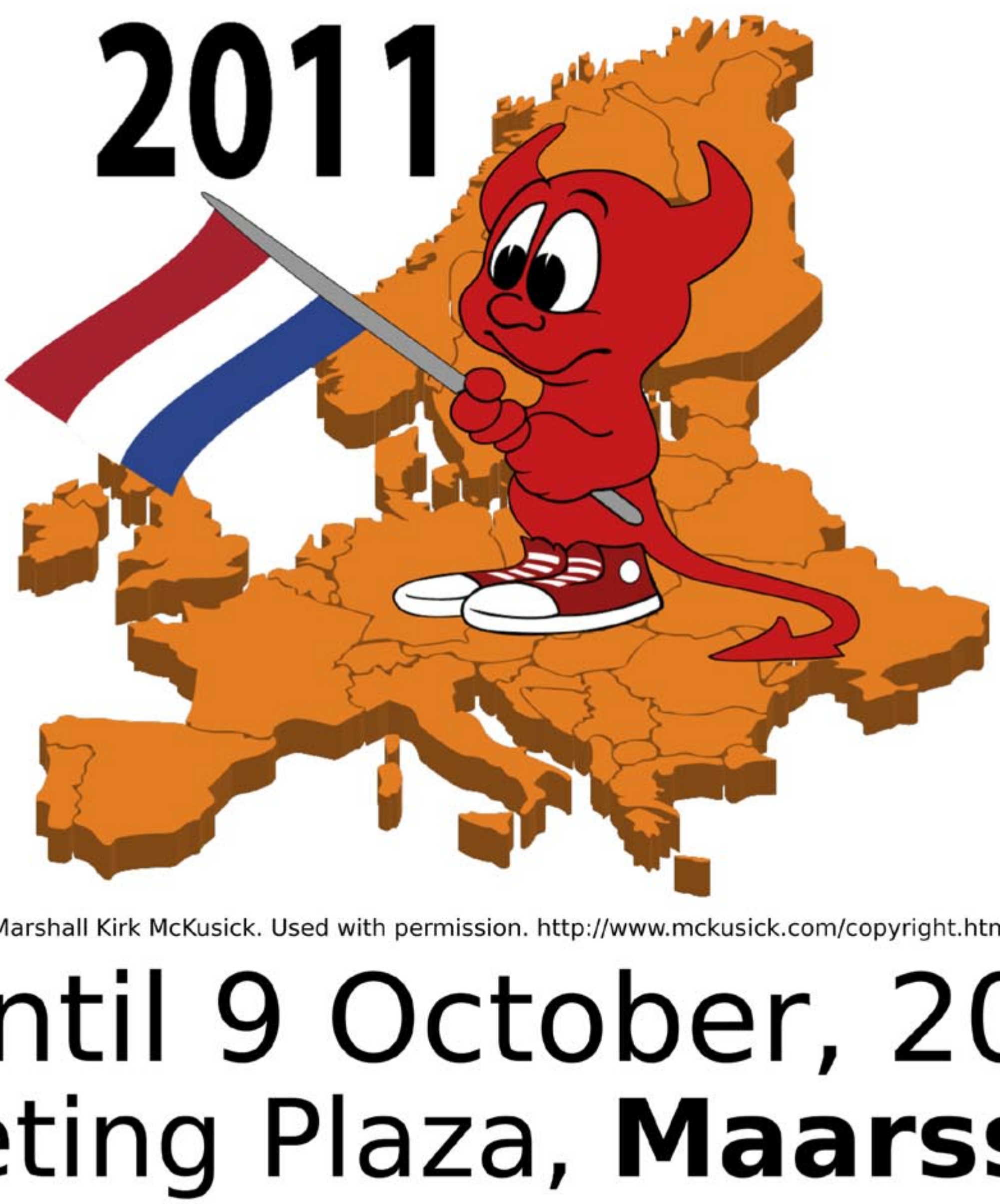
After playing with embedded systems, I wondered if it was possible to use this concept on my production servers as well. As it turns out, Paul Schenkeveld has had that same idea and extended it with ZFS (see BSD Magazine issue 02/2011) and Jails for a server with near-zero downtime, even for full systems or ports upgrades. He wrote a paper on it and gave talks at AsiaBSDcon2010 and the Dutch NLLGG in December 2010. If you are interested in NanoBSD, I strongly recommend reading his paper.

There are also a number of well-known projects that use NanoBSD as their base. Examples are the pfSense firewall (see BSD Magazine issue 02/2011) and the FreeNAS file server (again, see BSD Magazine issue 02/2011).

ERWIN KOOI

Erwin Kooi is an information security manager for a large grid operator. He started with FreeBSD 4.4 and is an avid fan ever since.

EuroBSDcon



The **Anniversary**

6 until 9 October, 2011

Meeting Plaza, **Maarssen**

Address: **Planetenbaan** 100
3606 AK Maarssen
The Netherlands

GPS: **N**52.12840, **E**5.0360

10th European BSD Conference

http://2011.eurobsdcon.org/

Mutt On OS X

Part III

When we last left our heroes (in April, 2011 issue of BSD Magazine), I had briefly discussed searching our Mac's address book as well as begin the process of setting up a complex, multi-account Mutt setup. In this article, I'll go a bit deeper into setting up Mutt to make the most of Gmail's features, as well as a way to handle attachments on your Mac.

What you will learn…

- How to handle attachments with Mutt on OS X
- How to gain deeper integration with Gmail in Mutt on OS X

What you should know…

- This article is a continuation of my Mutt on OS X articles, it's recommended you begin with the first article which appeared in BSD Magazine's 02/2011 issue)

Handling attachments with Mail.app or most GUI mail user agents is so easy that we never really need to think about it (until we have an attachment that our computer doesn't know what to do with.) With Mutt, things are a little trickier, as you might imagine, since Mutt is a text-based mail reader and a lot of incoming attachments are intended to be used with a GUI application. Let's say, for example, you are sent some photographs. In Mail.app, they are generally displayed inline, along with the body of the email. You could also save the files and then open them with any program that has the capability of showing you the images (preview, photoshop, firefox, etc...) Mutt has similar options available, i.e., you can save the attachment and then view it in whichever program you choose, or Mutt can open the attachment for you automatically (`Terminal.app` won't let you view images within your console, however there are console programs that can render images for you, such as zgv[1].)

In Mutt, saving attachments and opening them is as simple as pressing `v` while reading a mail message that contains attachments, and then choosing which attachment to save and pressing `s`. You can then open the attachment via Finder. If you'd like to open the attachment directly (in other words, by pressing `v` and then choosing the attachment and simply pressing return, thereby skipping the *save the file somewhere and open it with Finder* steps, you'll need to create a mailcap file. A mailcap file contains a list of MIME types along with what program to use to handle attachments of the corresponding type. If our computers spoke english, we could request, *if my mail application comes upon an attachment that is a jpeg image, please open it with my favorite photo viewer*, however, while we cannot make the request this way, our macs are based on Unix and so we can do even better: we can add something like the following to our .mailcap file:

```
image/jpeg; /Path/To/My/Favorite/PhotoViewer.app %s
```

In fact, it gets even better! OS X has a command-line program named *open*, which will open whatever file we ask, with the default application designated to open it that type of file. How convenient! To make it easier, Eric Gebhart has written a great shell script (aptly named `view_attachment.sh`) that gives us more control over when and how to open various attachments. For the most part, it simply acts as a wrapper for *open*, but it can be configured to take action in other ways as well. You can download a copy of `view_attachment.sh` from my site [2]. I found it via an amazing Mutt on OS X tutorial written by Vincent Danen [3] (HUGE thanks to Vincent!). Once you download `view_attachment.sh`, you should keep it somewhere it will be safe (I keep mine in my `.mutt` folder, i.e., `~/.mutt/view_attachment.sh`) and then you'll need to create a .mailcap file. A very basic `.mailcap` file could then contain:

```
image/jpeg; /Users/you/.mutt/view_attachment.sh %s
```

And that would be enough to tell mutt that if you ask it to open a jpeg image attachment that it should look for `view_attachment.sh` and let it take over from there. This would result in the OS X command *open* to be invoked, and unless you've changed the default jpeg viewer, the image will be opened with `Preview.app`. You can use your favorite text editor (vim?) to create your .mailcap file, and make your it as detailed as you like, or you could even download Vincent Danen's extensive mailcap file from his site (see resources.)

Opening HTML email in Mutt requires you to view the HTML attachment portion of the email the same as you would any other attachment. You can configure your `.mailcap` to open HTML files with `view_attachment.sh`, which would result in the file being opened with your default web browser. Vincent says, *it's gay to use safari for this*, and he recommends using elinks or another text-based browser. This is your setup, though – feel free to open your attachments with whatever you choose! I like to use my default browser (Firefox) because it generally allows me to see the HTML mail the way the sender expects. Of course, I'm not a huge fan of HTML mail in general, but that doesn't stop people from sending it to me anyway! If you'd like to read the fine manual regarding mailcap files, you can find all the juicy details in RFC 1524 [4].

Since we've been using Gmail as an example for all of the mail setup so far, I thought it would be worth mentioning that you can get quite deep Gmail integration in Mutt. For example, I've mentioned in a previous article that you can search your local Mac address book, but what if you want to use your Gmail contacts? For that you can use GooBook. As it says on the GooBook site [5], *The purpose of GooBook is to make it possible to use your Google Contacts from the command-line and from MUAs such as Mutt*. If this sounds like something you'd like, you can find all the info you need at the GooBook home page, along with instructions on how to integrate it with your existing Mutt setup.

Another thing you might miss from Gmail is the ability to flag messages as spam (or tell Gmail a message isn't spam). Interestingly (and conveniently!) it turns out that this is actually quite simple from within Mutt, because the action of marking a message as spam in Gmail is not a special script or piece of programming, it's actually as easy as moving a message into the spam folder. There's a *User Story* in the Mutt Wiki that explains how to set up a macro to do this for you – basically creating a macro that you can bind a key for that will either mark a message as spam (i.e., move it to the Spam folder) or mark it as ham / non-spam (i.e., move it to the Inbox). You can find a link to this *User Story* in the resources [6].

On the 'Net

- *http://www.svgalib.org/rus/zgv/* [1]
- *http://www.culmination.org/Mike/view_attachment.sh* [2]
- *http://linsec.ca/Using_mutt_on_OS_X* [3]
- *http://tools.ietf.org/html/rfc1524* [4]
- *http://pypi.python.org/pypi/goobook/1.4alpha4#about* [5]
- *http://wiki.mutt.org/?UserStory/GmailMultiIMAP* [6]
- *http://lifehacker.com/5574557/how-to-use-the-fast-and-powerful-mutt-email-client-with-gmail* [7]
- *http://www.techrepublic.com/blog/opensource/easy-gmail-reading-with-mutt/1737* [8]

You may have noticed that Google doesn't really want you to delete mail (they give you tons of space so you won't have to). When you're done with a message, it generally gets archived instead of deleted. Deleting unwanted mail is possible, however, and you can archive or delete mail via Mutt. To delete mail, you'd want to tell Mutt that your trash folder is the remote Gmail trash bin, and then delete the message, which would then move the message to Gmail's bin (as opposed to your local trash file). Remember, if you simply delete the message in Mutt, Gmail will keep an archive of the message. So how do you view those archived emails? Easy! You just need to view the *All Mail* folder. You could use Mutt's change folder commands, or you could create a macro that does it for you. I found info on this on Lifehacker and Tech Republic, see the resources for links to those entries if you're interested [7] [8].

If you had doubts about the power and flexibility of Mutt, I hope those have all been dispelled by now! Mutt can really do it all, even if it has to ask a few Unix utility friends for some help. As I said when I started this article series, some people just don't see the point of using a text-based mail reader when you're sitting at a Mac. If you don't like `Mail.app`, there's Thunderbird, or even Sparrow – surely you can find a GUI program that fits the bill? I'm not the only one who sees the benefit of staying in my Terminal, though! On that note, I'd like to mention that I had not planned on writing a part 3 of my Mutt on OS X articles but apparently I had left a reader hanging, hoping to hear more tips! So Richard, this article is for you! Thanks for reading, and I hope you've enjoyed this article and the rest of the magazine as well. Until next time...

MICHAEL HERNANDEZ

Mike is an IT consultant and web programmer. He lives in Brooklyn, New York, and he and his wife are celebrating their one year anniversary on February 14th. He also loves electronic dance music and commuting on his fixed gear bike, appropriately named Constance.

OpenBSD Networking

OpenBSD has an often mistaken image across the world that it stands for cryptography and crypto alone. Only for security applications OpenBSD is a good choice. This is what I used to think till I started looking at its IPsec stack in 2003.

What you will learn…

- Why should you try OpenBSD

What you should know…

- Basic understanding of UNIX and BSD world

A lot of water has flown under the bridge since then. Governments changed, I changed companies, the world of technology has seen a lot of developments but my love with OpenBSD that started then has not relented one bit. And interestingly this has nothing to do with crypto. It is perfectly true that what made me try OpenBSD was certainly crypto.

Networking

But to say that crypto is what OpenBSD is good at is like saying that sun is good at throwing light. Of course yes. Sun is primarily a light source. But there cannot be any life without sun. It is a source of heat, life giving photosynthesis cannot happen without the sun's rays.

OpenBSD's kernel code is the best I have seen. And I focused only on the networking parts since I was not, after all, writing a scheduler or a disk driver. I was writing IPsec. For porting IPsec, I only needed to know the kernel parts dealing with networking.

The general approach of cleanliness, beauty and grace found in OpenBSD is all pervasive; even in its third party packages known as ports.

Now we are getting to the point.

OpenBSD is all about getting your job done with the minimum effort.

Why do we choose an OS but to get our job done quickly? Sometimes the learning curve is steep and annoying but then that is part of the bargain. No use cribbing(??) about it. Some people like to suffer some pain to get a sweet fruit.

If you take the pains to master OpenBSD and in particular its networking capabilities then you can enrich your life in a lot of ways.

Most of my work nowadays revolve around systems administration though I have not had a job for more than 5 years. My products all require very advanced networking knowledge.

And guess what? You don't muck around with the kernel code to do high-end networking work.

You need to understand real life situations and know what is done by what. What is the userland's , the kernel's and the hardware's role?

Without this eye for technical detail you don't become a great systems admin or a great networking engineer.

And OpenBSD offers networking in its kernel (of course), base system toolset like ifconfig(8) and of course, its rich repertoire of third party packages called ports.

Some of the tools I use for sophisticated networking are written by the official developers themselves but they are ports rather than installed as part of the base OS. The developers could easily have licensed it to be part of the OS. The reason they did not do so was that the base should be of maximal use to people. We don't want bits that are optional to be mandatory.

Examples include greyscanner, pftop and gotthard. Regardless we are going to see in this article how to do certain cool things with OpenBSD.

Interestingly ultra powerful tools like OpenBSD's pf(4) or carp(4) are not meant to be used with other BSDs. If you are attracted to OpenBSD innovations then why use an inferior alternative?

Always bear in mind that an OS is a complete whole. Everything has to work in concert to provide a sophisticated feature like load balancing.

Now that we are talking about that, we have both load balancing and SNMP in OpenBSD base. Now we also have LDAP but I have not had time to play with it yet.

I have been focusing solely on spamd(8) for around 4 years now and I can find plenty of new ways for it to work in concert with the facilities provided by CARP clustering, pf(4) redirection and so on.

Each new day I am discovering a new application for solving hard networking problems. Real life situations are so much fun as opposed to laboratory situations.

For instance as I writing this article I am solving a problem for an ISP in Gujarat, India who is having trouble with his IPs getting blacklisted for being a spam source.

Initially I told the guy that I could not help this directly. Then after 40 days it occurred to me I could indeed do something about it.

And here I am, trying to deploy spamd in a manner that I never was exposed to before.

I admit that quite a few advanced tricks take a long time to master and get comfortable with. It is like a poet getting inspired to write poems or a music composer to create new tunes. Or even a Veena player learning how to play a new raga.

Inspiration comes from the subconscious and networking is no different. Once you gain the mastery that is so hard to come by in the beginning, later it becomes effortless and automatic.

In general spamd(8) can be deployed wherever port forwarding is possible. Initially the corporate firewall port-forwards SMTP traffic to SpamCheetah, followed by SpamCheetah port-forwarding mails to the company's mail server and the spam being forwarded to 127.0.0.1 port 8025.

This is a very long story cut short on how spamd(8) works.

But this is just one application. Perhaps the most common but by no means the only.

spamd(8) can be used to stop outgoing spam, it can also be deployed without any port forwarding. Port forwarding complicates matters since you have to muck around with the default gateway of the forwarded host.

And the destination IP address once rewritten, gets irretrievably lost.

This is not a great situation. Instead you can use the pf(4) route-to switch to route SMTP traffic without address rewrite. But this will work only within a network.

There is a lot of fine print that is not clearly mentioned in the man pages and it is also not a reasonable expectation. One has to learn by experience; there is really no other way.

Apropos of networking bridges and relayd(8) load balancing come to mind. OpenBSD bridging is equivalent to Linux bridges or Windows bridges in that it also does STP and acts as a managed switch.

But there are interesting possibilities when you combine bridging with firewalling or link failover.

The trunk(4) interface is a way to assign the same IP address to two physical links and you can bridge this link pair with a third link to provide transparent high availability of links.

Now relayd(8) helps us with high availability by doing a health check of third party servers which could be running some proprietary OS or service. The health check is done from the network; so it does not matter.

The other thing that gives us high availability is the CARP protocol which is a layer III protocol that uses multicast to fail over multiple nodes with the same virtual IP address.

As we can guess from all the above OpenBSD offers a garden variety of various networking options; which even the costliest commercial offerings cannot match, let alone exceed.

So why waste money on big brands where your time and knowledge can serve you better instead? For support?

Fret not. You can get OpenBSD support at a much lower cost; some of the developers will be glad to get paid a few extra dollars for solving your critical needs.

And before I conclude let us not forget that Internet core routing protocol BGP 4 (No Internet without that) has an implementation in OpenBSD by Henning Brauer and party. OSPF which is a necessary evil, is also included. So is RIP which is historical cruft.

I never got a chance to play with these things but this article is already long and we need to end somewhere.

GIRISH VENKATACHALAM

Girish has close to 15 years of UNIX experience and he loves OpenBSD more than he loves anything else in the technology world.

OMAP3 Full Support is Coming Soon in FreeBSD

The trend in the FreeBSD development is bringing FreeBSD for new sets of hardware. The OMAP™ 3 family of multimedia applications processors from TI introduces a new level of performance that enables laptop-like productivity and advanced entertainment in multimedia-enabled mobile devices.

OMAP3 products support a wide range of end equipments, from entry-level multimedia-enabled handsets to high-end Mobile Internet Devices (MIDs). OMAP3 has many advantages differentiate it from the other processors. Some of them are low power consumption, High speed I/O interfaces, Packaging, Module Availability, Multimedia processing, M-Shield Hardware Security and many more!. So, an urgent need led to OMAP3 porting. The boards we used for porting are BeagleBoard and BeagleBoardXM. One of the challenges of porting OMAP3 was the emulation process due to the fact that there is no software capable of emulating OMAP3 configured kernel on FreeBSD host. From my experience, the best emulator is qemu-maemo from Meego Tree which can run over Linux. A good start in the porting process was to work with Dockstar Machine since It shares the same ARM core (ARM926EJ-S) with OMAP1710, OMAP1610, OMAP1611, OMAP1612, OMAP-L137, OMAP-L138 devices. Sample configurations achieved by FreeBSD Kirkwood Team. They configured Dockstar with DOCKSTAR-COOLTRAINER identity, 128 MB physical memory, different serial ports, Networking, Ethernet, Wireless NIC Cards, USB wireless, Process Communication, NFS, OpenBSD Packet Filter, Swap Space and USB Audio. As a result, a team consisting of Warner Losh and me is working on Porting OMAP3 Completely. OMAP3 has partial support in FreeBSD by individual developers. One of those patches was developed by Mark Murray *http://people.freebsd.org/~markm/src.beagleboard.diff* and the complete support will be available soon. I spent much time trying to solve the omap3 emulation problem and I found qemu-maemo which supports two types of emulations. The first is running full system emulation (i.e. have a complete system with kernel and so on running inside the emulation) and the second is user mode emulation (i.e. have a single user mode process running under emulation like running a standalone ARM bash binary in your host system).

If the former, pass the parameter `--target-list=arm-softmmu` to qemu configure script. If the latter, pass the parameter `--target-list=arm-linux-user` or *arm-bsd-user* in case if your host is BSD and not Linux. At OMAP3 full system emulation, I should compose the kernel with u-boot for NetBSD which is similar to this one on FreeBSD. Thus to have successful kernel emulation, we should build the kernel which produces a kernel.bin binary file and a kernel ELF file. The process is a bit more complex, because u-boot expects a special header in front of ELF files. You will need the mkimage binary that can be installed from ports in devel/u-boot. To get it installed, type in the following commands:

```
myhost % cd /usr/ports/devel/u-boot
myhost % sudo make install
```

Once mkimage is installed, you can convert your kernel image file into an image that u-boot can load and run. Since u-boot doesn't support FreeBSD images, the following command instructs it to treat the kernel as a NetBSD image. Luckily, the behavior of u-boot in the case of a NetBSD image is generic enough so that things work anyway.

```
myhost % mkimage -A arm -O netbsd -T kernel -C none -a
80200000 -e 802000e0 -n „FreeBSD" -d path_to_kernel/kernel ukernel
```

You should obtain a ukernel file, that you can emulate on Ubuntu using qemu-maemo. Qemu-maemo can emulate multiple OMAP3 machines and we can retrieve a set of all machines using `-M ?` parameter. It supports beagle, beaglexm and n900 machines.

MOHAMMED FARRAG

Mohammed Farrag, FreeBSD Contributor, ArabBSD Project Manager and Google Technology User Group Administrator.

FreeBSD Mall

Your FreeBSD & PC-BSD Resource

www.FreeBSDMall.com

FreeBSD 8.2 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 8.2 CD	$39.95
FreeBSD 8.2 DVD	$39.95
FreeBSD 7.4 CDROM	$39.95
FreeBSD 7.4 DVD	$39.95

FreeBSD Subscriptions

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

FreeBSD Subscription , start with CD 8.2	$29.95
FreeBSD Subscription, start with DVD 8.2	$29.95
FreeBSD Subscription, CD 7.4	$29.95
FreeBSD Subscription, DVD 7.4	$29.95

PC-BSD 8.2 DVD (Hubble Edition)

PC-BSD 8.2 DVD	$29.95
PC-BSD Subscription	$19.95

BSD Magazine

BSD Magazine	$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 8.2	$79.95

The FreeBSD Bundle

Inside the Bundle, you'll find:
- FreeBSD Handbook, 3rd Edition, Users Guide
- FreeBSD Handbook, 3rd Edition, Admin Guide
- FreeBSD 8.2 4-disc set
- FreeBSD Toolkit DVD

★ Special: The FreeBSD CD Bundle	$99.95
★ Special: The FreeBSD DVD Bundle	$99.95

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!

NEW APPAREL!

What It Takes

Starting and Running an Open Source Certification Program, Part I

So you're all excited about your new software and its amazing capabilities to change the world. You truly believe that if only more people knew about it and were competent at using it, the world (or at least the information technology world) would be revolutionized.

You want it to succeed, but you know it won't until more people use it and are truly good at it. Suddenly, there's a flash of lightning…

Hey, that's it! I'll create a certification for it. I'll test people and if they are good at my stuff, it will change everything! you exclaim.

Sounds like a great idea, until you actually sit down to do it, and you realize you know absolutely nothing about creating a reliable certification program or accurately measuring people's knowledge, application skill, or performance. You remember some of those other certification tests you've taken- some of them truly pathetic- and wonder if there is any way to do it right. After all, if you're going to put effort into it, you want it to be an excellent certification program, just like your amazing software.

In this three part series of articles, we'll walk through what it takes to get your certification program off the ground. We'll look at the real-world experiences of the *BSD Certification Group* (BSDCG), how we started with just such an idea, and how we grew it into a solid certification program. We'll look at *People, Processes, and Technology*, and explain how important each element is in getting a certification program up and running. We'll explain what works for us, the decisions we've made, and the mistakes we've made. Hopefully, your certification program will be up and running in no time.

This article will focus on the *People* element- who you'll need, why you need them, and what they can do. In the next article, we'll focus on *Processes*- dealing with the business side, managing your content, defining and creating your certification test(s), and taking a look at governance and accreditation. The final article in this series will look at the *Technology* element- websites, surveys, collaboration, test construction and delivery, scoring, and metrics. We'll wrap it up with a look at the future of Open Source Certifications.

People

Every organization needs people to make it work. To build a reputable certification program, you can't do it all by yourself. You need people with several kinds of skills-

SMEs

Subject Matter Experts. These are the people that are knowledgeable in every aspect of your software system. They understand the technical details. They can clearly determine whether it's working correctly and they know how to fix it when it's not. When you get around to writing questions (called *items* in certification circles), these folks can contribute by writing and editing items. They can also help define the *Domains of Knowledge* of your certification program. These are the core topics that apply to understanding your system.

The BSDCG has a couple of dozen SMEs that generously donate their time to help generate and review items, and set expectations for the level of expertise for the certification exams. This group is a mixture of seasoned BSD users and younger folks interested in all flavors of BSD- FreeBSD, NetBSD, OpenBSD, DragonFlyBSD, PC-BSD, and a number of others. Having all the BSD systems represented by knowledgeable experts helps us keep our certification well-rounded. It also presents a challenge to keep the focus of the certification from drifting too heavily in one BSD direction. We are always eager to welcome new SMEs, whatever their favorite BSD may be.

General Technical

These are the people that can help you set up and run the various technology pieces that are crucial to your certification program.

There are system administrators, network engineers, website designers, and software developers. You'll need at least two servers (possibly many more) to host your website, registration system, technical content, mailing lists, etc.

The BSDCG uses several servers for our internal processes such as our website, registration system, mailing lists, wikis, etc. It takes a fair number of smart technical people to help run all these pieces.

There is software to install and continually manage. There are planned and unplanned outages for networks, servers, and applications. Without help from our technical team, the BSDCG would not be able to function at all.

Writers

The writers are those people that write the website, wikis, FAQs, brochures, handouts, and other explanatory material. If they have no access to the item bank (question database) they can write training materials and study guides for the certification. They can even write books and articles for profit as long as there is a clear separation of training and certification activities.

There are several writers within the BSDCG and a couple of them have wanted to pursue opportunities to publish training materials for profit. In order to accommodate these requests, the BSDCG worked with them to ensure that their training materials were based solely on publicly available materials such as our Exam Objectives documents.

This ensures that there is no conflict of interest with creating the examination itself, and it also ensures that the training materials they produce have no unfair advantage over anyone else's training materials.

There is a special category of writers that can be very important- the translators. Having a team of translators can be a big help to spread the word about your system and your certification program.

The BSDCG has a translation team who has been extremely helpful in getting our documents translated into other languages. But keeping the translation teams engaged can be tricky.

You have to deal with many people of different backgrounds whose knowledge of English may be marginal. Then you have the different time zones for people from around the globe. It can be very challenging to keep a translation team going. This is one area the BSDCG needs more help.

Media

Media savvy people such as journalists, bloggers, social media mavens, publicists, and public relations people can help you get the word out about your system and your certification program special events such as publication of your Exam Objectives documentation, translations, group meetings, new exams, and other news.

The BSDCG has several people who have experience in blogging and social media- setting up BSD groups on Facebook, Google, and Linked In, and publishing information on various blogs. But we lack experience in getting attention from mainstream tech media, and this is where a lot of *industry buzz* is generated and consumed. One of our current goals is to improve our profile and the visibility of BSD systems in the tech media.

Managers

Having people who can keep things on track, maintain lists of things to do, and help find and apply resources are crucial to getting a certification program up and running. It may seem strange, but they don't have to be technical gurus. In fact, they don't have to be knowledgeable about your software system at all. Yet these people are important because they provide the lubrication (some would say sandpaper) to get things organized and keep everything moving in the same direction.

Managing an Open Source project is different from managing other business and commercial activities. People will sign up or agree to get something done and then not do it. They get too busy with work or family to work on the project. In the business world you can compel people to cooperate with schedules and deadlines. But you can't do that with Open Source project people. People join the project because they want to help or contribute. Sometimes they have to realize that family and their `$DAY_JOB` comes first.

An Open Source project manager is a good communicator. They try to keep information flowing between all the various groups. They help find extra resources to backfill someone who, for whatever reason, can't find the time to get their piece of the project done.

There aren't any real managers in the BSDCG. There are several people with management experience, and lots of people who know how to run a project. But we don't have dedicated managers, mostly because everyone is busy. It takes time to manage an Open Source project well.

Advisory Board

An advisory board is a group of senior level IT people that you contact to see if they are interested in being loosely associated with your system and certification effort. From time to time, you'll want advice on various topics such as business direction, certification promotion, various technical and legal issues, and other matters of general interest. It's to your advantage to select a diverse group of informed, knowledgeable, and competent people. They can help you steer your group through some rough times. And be aware, every certification effort has rough sailing. You'll want to keep the advisory board up to date at regular intervals.

As you contact them, they may be wary of getting involved. You can put them at ease by explaining that they are primarily a sounding board- not developers, managers, or other hands-on workers. You value their advice, not their active participation.

The BSDCG does have an advisory board of senior Unix people. We've given them infrequent updates on our progress, and we've gotten some feedback. But we really should do a much better job of keeping them in the loop.

Psychometrician

Last but not least is the psychometrician. The psycho-what you say? The psychometrician is the person who knows the science behind measuring knowledge and performance. They have experience in setting up and running testing and certification programs. Theirs is a specialized knowledge of how to create certification exams that are accurate, repeatable, and reliable indicators of knowledge and performance competence. Usually they are PhD's in Education or Sociology, or a related discipline. And note- they cost money. A psychometrician is a specialized career professional, just like a doctor or a lawyer. Don't expect to find one who will donate their services for free.

The BSDCG was very fortunate in landing Dr. Sandra Dolan to be our psychometrician. Her previous psychometric engagements included the design of the Linux Professional Institute certification, and The American Osteopathic Association certification. She has helped our BSD Associate certification immensely. Without Dr. Dolan, we would have little more than some questions a few dozen people dreamed up in their heads. Instead, we have a solid psychometric foundation for our BSD Associate exam, and we are currently contracting with her for help on our BSD Professional exam.

With all those people, there is still one ingredient missing- passion. You have to be passionate about getting your software system and your certification program up and running. Communicating that passion and getting others excited is a key element in getting it off the ground. Once you have passionate, dedicated people who love working on the certification program and contributing to its success, you will be successful.

There is an interesting dilemma when you start an Open Source certification project. The very word *Open* means transparency and inclusion. It's the ability to be involved in any sort of capacity in the project.

As it turns out, there is a limit to *openness* in a certification project. If the objective is to measure knowledge in a structured way, the questions and answers being asked on the exam can't be truly open and out there for anyone to see. A certification program is one of the few Open Source projects that has to have a *Non-Disclosure Agreement* (NDA) as a part of the engagement process for those people that are going to be working with the item bank. It's an unfortunate necessity, but one that most people, if they think about it, will understand and accept.

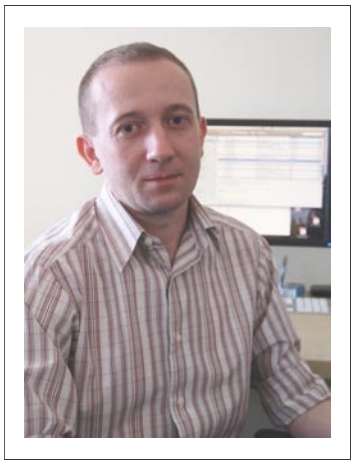
An NDA is a legal document. You'll want to get a lawyer to look it over. We'll talk more about the business and legal elements of your certification program in the next article.

For now, get that passion fired up and go find a bunch of people that feel just the way you do about getting your system and certification program going. And if you don't have a certification program going, come join the BSD Certification Group. We're waiting for you to help us change the world!

JIM BROWN

Jim Brown has worked in the computer industry with continuous Unix involvement in development or administration since the early 1980s. His experience includes applications, systems and database programming, in a variety of languages. One of the founders of the BSD Certification Group, he is helping to develop the BSD Professional certification. He currently lives in Northwest Arkansas, USA.

Interview with Rafał Jaworowski



Rafal Jaworowski is a co-founder of Semihalf, where he is leading the operating systems department.
With over 12 years of experience in the embedded systems field he has ported FreeBSD to various ARM and PowerPC systems, designed and developed device drivers and kernel infrastructure components, which are embedded in commercial products and installations.
He contributes to the FreeBSD Project as a src committer. He has earned a M.Sc. degree in Mathematics.

Could you introduce Semihalf to our readers?
We are an embedded systems company, with expertise in both hardware design and software development. On the hardware side we mostly work with higher-end system-on-chip devices based on architectures like ARM, MIPS, PowerPC. The software development is often related to open source code and technologies, in particular the BSD family of operating systems. We have experience with embedded development for FreeBSD, but also work on NetBSD as well as bootloaders like UEFI or U-Boot. Besides this, we know and work with Linux, so it is possible to have a broader view of various systems‘ strengths and weaknesses when considering an embedded operating system and applications for our clients. We are based in Kraków, Poland, but we cooperate with international customers.

While having a wide field of expertise, you put noticeably more emphasis on BSD. Why?
We like BSD because of the mature, quality code base and good licensing terms. We appreciate the scientific heritage and technical excellence as one of the key objectives. Since we share these views in our daily work, the BSD route is simply a natural direction. Another important factor is that people in the community are friendly and helpful. It just feels right to take part in this undertaking.

You are contributing to the FreeBSD project. Could you tell our readers what is your role?
I'm part of the FreeBSD committers team, which means I work together with other developers from around the world on improving and maintaining the system. Technically, I can submit code to the central source repository and, actually, much of the code developed at Semihalf was contributed this way after peer reviews. Besides pure development I also happen to mentor people on FreeBSD. In the past years I was shepherding two Google Summer of Code students on behalf of the project with embedded-related tasks. Personally I like to talk about what we do and whenever possible I'm trying to share this with a wider audience at the BSD conferences like BSDCan or AsiaBSDCon. Speaking of conferences, last year Semihalf was co-organizing the meetBSD 2010 technical conference in Kraków, which brought many BSD developers and users together, and was a successful event.

Do you have any plans for organising more conferences or partaking in any more this year?
This year we are not able to involve in co-organizing the event directly, but in general we are willing to support the conference and are looking forward to future editions. It's possible however, you'll see our developers speak at sister BSD conferences as we plan to submit papers and talks about some of the work we've done.

Let's get back to Semihalf. Could you tell us something about your customers?
One group of our customers are vendors of the embedded processors, for whom we provide platform-level software i.e. bootloader and operating system. Code from such developments can eventually be published for example in the official FreeBSD repository.

What was the most difficult and challenging implementation you've done so far? Could you give us some details?

Among the most memorable projects for me personally was developing FreeBSD support for an embedded PowerPC family of chips (the so called Book-E specification). This project's challenge came from the fact that at the time we started to work on this, there was no FreeBSD support whatsoever for this kind of machine. Since the Book-E is in some aspects so much different from the traditional PowerPC (which was supported back then), it is considered a separate architecture variant with specific MMU approach and other low-level differences. We therefore had to implement from scratch the low-level virtual memory management layer (pmap) and then all major peripheral drivers.

I have also completed a follow up project to this one, which introduced SMP support for dual-core versions of these chips. This also was kind of a fresh start so it brought many surprises during development.

But these are just part of a bigger picture: our team has accomplished a number of interesting projects, like porting to new embedded ARM systems, developing a complete NAND Flash framework (including a filesystem) and similar.

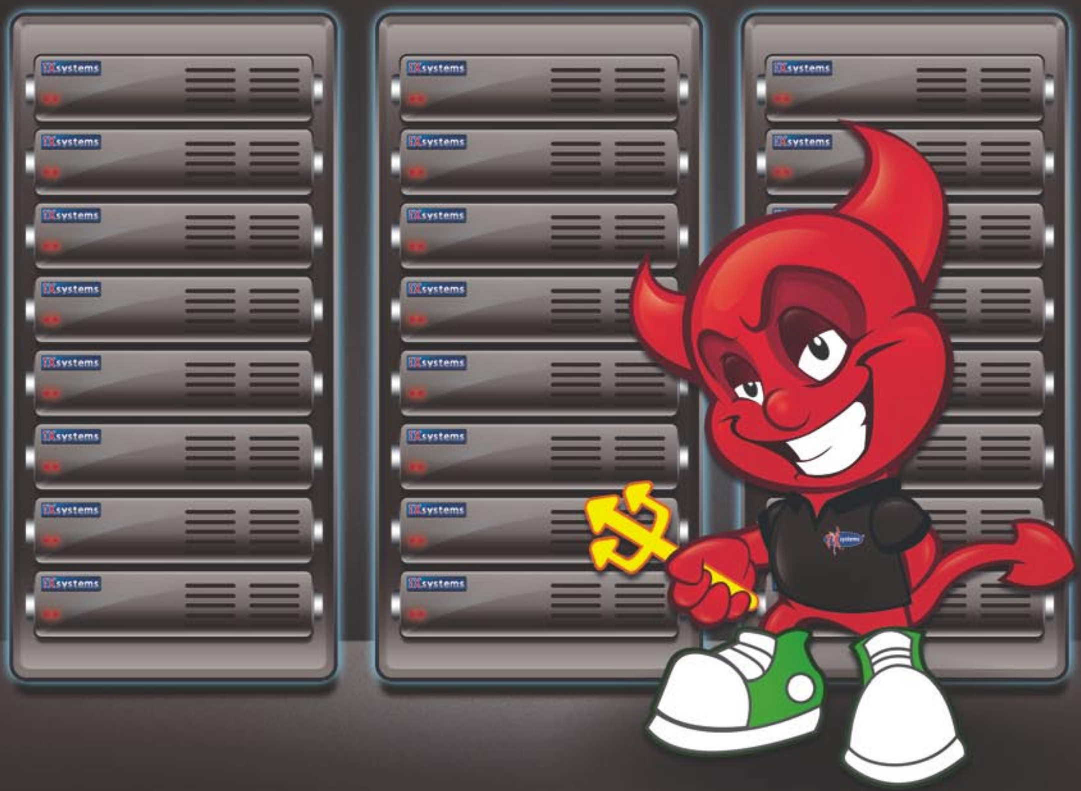
Future plans for Semihalf?

Plans related to the BSD world include porting FreeBSD and NetBSD to the latest and upcoming PowerPC embedded processors. We are also working on better support for ARMv6 and v7 which will cover for SMP operation of FreeBSD on multicore ARM systems. Other developments will include support for more contemporary embedded platforms and system-on-chip devices. There are going to be more projects, but these are some major highlights, so stay tuned. Some details about our latest development efforts can be found on the web page *http://www.semihalf.com*.

What future do you see for BSD systems? In general and in the embedded field?

There's plenty of things still to be done in the embedded BSD space, so from this perspective the future seems exciting as we can only grow. However for this to happen we should encourage more people to get involved and work on the embedded development of BSD systems, which I'm trying to do also through this interview.

What has your server vendor done for BSD lately? Probably, not much.



Work with a vendor that supports the operating system you love!

iX is the corporate sponsor of the PC-BSD® Project, a major corporate donor to the FreeBSD Foundation, and leads the FreeNAS™ development team -- all while employing some of the most brilliant minds in the FreeBSD® community. For BSD hardware and software expertise, look no further.

1-855-GREP-4-IX

http://www.iXsystems.com/community

iX systems®