news@UK

The Newsletter of UKUUG, the UK's Unix and Open Systems Users Group

Published electronically at `http://www.ukuug.org/newsletter/`

Volume 19, Number 3 **ISSN 0965-9412** **September 2010**

Contents

From the Secretariat

Jane Morrison

UKUUG continues to work with Josette Garcia at O'Reilly to organise tutorials.

Unfortunately the 'Understanding Regular Expressions' Tutorial (by Damian Conway) that was scheduled for 11th August had to be cancelled due to a death in the tutor's family. We hope to be able to reschedule this for later in the year or early 2011.

We are currently working to organise a tutorial by Allison Randal on 'Packaging for Debian and Ubuntu'. This will probably be held in November in London and as soon as we have firm details we shall announce the event to members via email.

The AGM this year will be held on Thursday 23rd September at the Imperial Hotel, Russell Square, London starting at 6:00 p.m. The agenda and other documents were sent to all members on 18th August. We hope you will be able to attend. Full details can be found on the UKUUG web site.

OpenTech 2010 will be held on Saturday 11th September and this looks like being another very successful day organised by Sam Smith.

Looking ahead to Saturday 16th October we are organising our first ever UKUUG un-conference. There are further details elsewhere in this Newsletter.

The Spring 2011 event (Tutorial and Conference) will be held from 22nd–24th March in Leeds. The call for papers is enclosed with this issue.

During the quiet month of August I have been having a tidy up of the office and stationery cupboard and have come across our stock of UKUUG polo shirts. These stylish polo shirts contain the UKUUG logo embroidered in silver onto a black shirt. See the UKUUG web site for a picture. Small, large and extra large sizes are available, at a price to members of only £15 including VAT and postage. Please contact the UKUUG office by phone (01763 273475) to order your shirt today!
The next Newsletter will be the December issue and the copy date is: 19th November. As usual any material be sent for inclusion can be sent to: `newsletter@ukuug.org`.

Chairman's Report

Paul Waring

Recent and Upcoming Conferences

We are planning to run our first one-day unconference in the autumn in Birmingham. This event will take place on a Saturday so that delegates can travel to the venue and back in one day and without having to book time off work. More details can be found within this issue and on the UKUUG website, and I would like to encourage people to attend and get involved in an event which will be more relaxed and informal than our usual conferences.

Preparations for next year's Spring conference are also underway, and the call for papers is included with this newsletter, as well as being available on the UKUUG website. I hope you will be able to join us for another enjoyable three day tutorial and conference programme.

Get Involved

UKUUG exists to serve its members and we are always on the lookout for people who are keen to get involved in any capacity, whether that be through volunteering to help with organising events, writing newsletter articles, or entirely new activities which haven't been tried before.
We have a vacancy on Council for someone who would like to get involved on a regular basis, and we are also keen to get members involved in running next year's Spring conference. If you would like to help out in any capacity please do get in touch via `office@ukuug.org`.

FLOSS UK 2010

The FLOSS UK 2010 unconference will be UKUUG's first unconference, and will be held on Saturday, 16th October 2010 at the Birmingham and Midland Institute in the centre of Birmingham.

Various groups are joining us in this event, including local Linux User Groups, the Birminham Perlmongers, the West Midlands PHP group, the PyCon UK Society and the West Midlands Ruby Group.

A wiki has been set up at `http://unconference2010.flossuk.org/` with more details.

This should be an exiting and lively event: we look forward to seeing you there.

SAGE-WISE donation to UKUUG

Paul Waring

Jonathan Crompton, the former Treasurer of SAGE-WISE, has been in touch with us and has kindly donated the residue of SAGE-WISE funds to UKUUG, following the winding-up of that organisation.

The correspondence (reproduced below) speaks for itself. We are both grateful for and touched by this gesture, and the donation will be used in the manner suggested.

> Hi Lindsay
>
> I have been given your name as a contact for UKUUG.
>
> I was one of the founder members of SAGE-WISE (The Systems Administrators' Guild – Wales. Ireland, Scotland & England).
>
> The organisation thrived in its early days, not least due to the huge contribution from Pete Humble.
>
> After Pete's untimely death in November 2001, we found some new blood, but the group eventually ran out of steam and disbanded.
>
> We have a small balance in our bank account (around £250) and the original constitution stated that in the event of SAGE-WISE closing, the funds were to be donated to a like-minded organisation.
>
> We have thought long and hard about how the funds could be put to best use, and have decided that we would like them to be used to sponsor or subsidise a student to attend a UKUUG event, in memory of Pete Humble.
>
> Could I ask you whether UKUUG would be prepared to accept a donation on this basis?
>
> Kind regards,
>
> Jonathan Crompton
>
> (Former) Treasurer, SAGE-WISE.

In memory of Pete Humble

Dear Jane,

Further to our correspondence with Howard Thomson, the SAGE-WISE club would like to make a donation of £264.41 to UKUUG Limited. This money is to be used to sponsor or subsidise a student to attend a UKUUG event, in memory of Pete Humble.

Pete Humble died on Sunday 25th November 2001 after a brief illness. Pete was present at the very first meeting at the Rising Sun, in London, where the seeds of SAGE-WISE (The Systems Administrators' Guild of Wales, Ireland, Scotland and England) were sewn. His commitment as the SAGE-WISE Secretary was second to none. Anyone attending the early London meetings, will have known Pete's friendly face. Pete was a mine of information – not just the techie stuff; his interests were broad, and he always had time to listen to (and entertain) others. In committee meetings, Pete exercised great humour whilst keeping us on the straight and narrow. His untimely death was a great loss to SAGE-WISE and to his friends and family.

Jonathan Crompton

Treasurer, SAGE-WISE

SCO -v- World Saga Update

John Collins

Now this is not the end. It is not even the beginning of the end. but it is, perhaps, the end of the beginning. – Winston Churchill (1942)

It is strange to write this after 7 years of meaningless litigation. But it is probably true.

To Recap – the story so far

Back in March 2003, SCO, as Caldera then renamed itself, to aid confusion with the Santa Cruz Operation (which got renamed Tarantella and then got taken over by Sun), sued IBM. It was a long and complicated lawsuit which alleged that Linux was an illegal copy of UNIX, infringing on SCO's copyrights, for which IBM was largely responsible. No meaningful evidence was presented, or ever has been presented, to support this notion, but somehow the US Federal legal system doesn't seem too worried about this. IBM has been forced to spend literally millions over the past 7 years defending itself and producing responses to SCO's endless quest for evidence to support its case. A filing from SCO, dismissed by the court, tried to claim that the lack of evidence was because IBM had destroyed it. The case has been stretched out to a large extent by the fact that SCO will not accept, and appeals to the fullest, every single court decision, however trivial, which goes against them.

One of the oddest parts of the copyright claims by SCO is the notion that the copyrights are "viral" and work backwards in time. For example, a major claim by SCO relates to the JFS filing system, which IBM wrote for OS/2, then ported to Linux, finally porting it to AIX, IBM's UNIX implementation. In SCO's logic, the port to AIX made it their copyright and the port to Linux a breach of that copyright even though it was done earlier. Some of the other documents SCO has produced over the years in support of its claim have the same backward logic – files dated before the files they were supposed to have been copied from.

Enter Novell

During 2003 Novell pointed out that it had never sold the UNIX copyrights to Santa Cruz (for them to sell on to Caldera/SCO) and moreover had the right to direct SCO to waive any claims against IBM. IBM had paid Novell a large fee for an "irrevocable" licence, and for SCO to say that it wasn't "irrevocable" without intervention by Novell would have left Novell liable for a huge sum to IBM.

SCO responded to this in early 2004 by suing Novell for "Slander of Title". This is a claim that Novell had falsely made statements maliciously and knowing them to be false that they and not SCO owned the relevant UNIX copyrights. The case rapidly hinged on the meaning of the 1995 "Asset Purchase Agreement" (APA) between Novell and Santa Cruz, whereby Novell transferred the UNIX business. However at the time Santa Cruz could not buy the copyrights as well, so Novell retained these, a fact explicitly set out in the APA. There was however a rather vaguely-worded amendment to the APA signed a year or so later referring to Novell transferring whatever copyrights are needed to run the UNIX business (which was never defined and could have been the null set).

The importance of this is quite far-reaching in that if Novell and not SCO own the copyrights to UNIX, over which SCO were suing IBM, SCO's case against IBM almost completely collapses, as do the other cases SCO had got embroiled in in the meantime.

Other cases

As well as suing IBM and Novell, SCO sued two seemingly randomly-picked Linux users, a car parts company called Autozone and Daimler-Chrysler. Red Hat also sued SCO over various disparaging remarks SCO had made about them. Moreover both IBM and Novell counter-sued SCO.

Most of these matters got put on hold pending the key IBM and Novell cases, apart from the Daimler-Chrysler case, which got rapidly dismissed by a no-nonsense judge in Michigan.

After creating reams of paper, countless court hearings and decisions, almost all of which went entirely against SCO, the IBM case itself got put on ice in 2007 pending a decision in the Novell case.

The 2007 Novell decision

In August 2007, Judge Kimball at the US Federal Court in Salt Lake City, Utah issued a far-reaching decision on "Summary Judgement" in the Novell case. (Summary Judgement is where the judge considers the case so cut and dried that no full trial – which in the US is always heard by a jury in civil as well as criminal cases – is required to decide it).

He decided that the US Copyright Act requires any transfer of copyright to be explicit. The original APA clearly excluded the copyrights from the transaction and the amendment was too vague to qualify.

After issuing the decision, Judge Kimball (who was also handling the SCO-v-IBM case) ordered IBM and SCO to report on the effect of the Novell decision on that case. Both prepared reports, and even SCO had to admit that the case against IBM was almost completely gutted by the decision.

Chapter 11

A month after the Novell decision SCO went into "Chapter 11". This almost unique provision in US bankruptcy law lets a company in difficulties put all pending claims against it on hold whilst it attempts to reorganise itself. As both the Novell case and the IBM case involved counter-suits against SCO, then these were suspended. The other cases involving SCO were in turn suspended awaiting Novell and IBM.

Chapter 11 seems to be a licence to print money for all the lawyers and accountants who administer it whilst sacking nearly all the staff of the affected company. One of the casualties was Darl McBride, the CEO of SCO whose gung-ho approach to litigation had caused all SCO's difficulties in the first place.

SCO is now being run by a “trustee” appointed by the bankruptcy court, former Judge Edward Cahn, although most people don’t seem able to detect much difference between his approach and Darl McBride’s, apart from an early settlement of the Autozone case on undisclosed terms.

SCO appeals

The US Federal legal system has three tiers (each state has its own legal system and court hierarchy as well but inter-state and important cases are usually tried by Federal Courts).

The bottom tier is the “district court” where the actual trials take place, in this case Salt Lake City, Utah.

The middle tier is the relevant “Circuit Court of Appeals”. There are 12 such courts, 11 of which cover geographical areas and the remaining one (“The Federal Circuit”) considers appeals on a restricted range of subjects, notably patents. The “10th Circuit Court of Appeals” covers 6 states including Utah.

The top tier is the US Supreme Court.

At the end of a case in the district court, either party can appeal to the relevant circuit Court of Appeals. It would, of course, have been a huge surprise if SCO hadn’t done so.

To appeal further from the Court of Appeals to the Supreme Court, the permission of the Supreme Court has to be granted, which it does only in a tiny fraction of cases considered important or precedent-setting, or perhaps where different Circuit Courts have come to conflicting decisions on the same issue.

SCO appealed, to no one’s surprise, to the 10th Circuit Court of Appeals. However to much surprise and consternation the court reversed Judge Kimball’s decision. It didn’t say SCO owned the copyrights. However it did say that a jury would have to decide it not a judge on summary judgement.

The jury trial and second appeal

SCO had got the jury trial it said it wanted in the Novell case. This went ahead in March 2010 over 3 weeks. After less than a day’s deliberation the jury delivered their verdict. This was the same as Judge Kimball had decided nearly 3 years before; Novell not SCO owned the copyrights to UNIX. SCO reacted in simulated stunned surprise.

What was noticeable during the trial was that Judge Stewart (who had taken over the case from Judge Kimball) seemed to bend over backwards to decide every point in favour of SCO and against Novell concerning which evidence to allow and so forth, to the point that people were accusing him of bias. Many observers also commented that if despite all that, Novell still won, it would make it very hard for SCO to appeal.

Hard or not, SCO nevertheless have filed an appeal and the arguments for the appeal have yet to be disclosed at the time of writing. Most legal commentators have said in any case SCO will have a much harder time this way round having argued so hard for a jury trial before and now wanting the outcome of that trial to be set aside because they don’t like it.

Whatever else, this will be the source of yet further delay. Unless the court decides to dismiss the appeal without a hearing (which is possible but unlikely) the appeal probably won’t be decided before Christmas.

What next?

SCO recently made a rather half-hearted attempt to revive bits of the IBM case which they said (IBM disagreed) weren’t affected by the Novell decision. It looks unlikely that the judge now hearing that case will want to carve out bits of the case rather than hear it as one complete entity when ready which can only be when the Novell decision has been appealed as far as it can.

If SCO win yet another trial by another jury in the Novell case, the whole thing will add the best part of 2 years to the timeline.

If they lose the appeal, they will almost certainly try to appeal again to the Supreme Court, which will almost certainly fail in that the Supreme Court will refuse to hear it, but will waste lots of time and money (arguably the whole point of the exercise).

The IBM case will then be revived. Bits will be decided against SCO as they don't have the copyrights to argue about. However there are bits, especially IBM's claims against SCO, which will need to be argued about. SCO will undoubtedly appeal all the bits of that they don't like. Even at a best case scenario this doesn't look like ending much before the 10-year anniversary in March 2013.

The only possibility is that the trustee, former judge Cahn, will admit defeat and pull the plug before then. However this would be on punishing terms from IBM, who have spent so much defending themselves against SCO's claims and will undoubtedly be after blood. Even if SCO are bankrupt, their lawyers could be the next target if they can be shown to have acted unprofessionally. It is hard to see that IBM will lightly let the matter rest.

In the meantime, "Son of SCO" in the form of Oracle's suit against Google over Oracle's software patents, has already risen to take its place. Those who enjoy taking an interest in ludicrous lawsuits (which thankfully aren't costing them anything!) will have no shortage of material to satisfy their cravings for a considerable time.

Programming with Technological Ritual and Alchemy

Alva Couch

I have taught programming – intensive courses to college students for over 20 years, and I cannot help but notice that the nature of programming has drastically changed in recent years. I teach a generation of students unfettered by my compulsion to understand and, as a substitute, fettered by their own compulsion to do and experience. Unlike the simple languages I employed to learn programming, my students employ complex frameworks, exploit primitive crowdsourcing to come up with solutions, and engage in a shamanistic ritual of dance and pattern to produce software that works, but whose complete function they do not and cannot fully understand. So? As they might say, "What's wrong with that?"

Technological Shamanism

I call the practice of creating software systems from ritual *technological shamanism.* Programming via socially derived ritual makes a surprising amount of sense and is often productive. But to understand its nature, we need to carefully draw some important distinctions between the concepts of "pattern", "example", and "ritual".

A *design pattern* is a proven and reusable approach to solving a specific kind of programming problem [1]. It is called a "pattern" because it specifies the nature of a programming solution without the details. The original description of patterns expressed a pattern as an object-oriented program that operates on objects that implement specified interfaces. These objects represent details the programmer must supply. For example, a "sorting" pattern works on objects that implement a comparison interface. One uses a pattern by implementing the interfaces it requires.

More recently, "design patterns" have acquired a general popular meaning outside the strictures of object-oriented programming, as reusable program fragments with some details missing, in which the method whereby a programmer fills in details is well documented. A pattern is a code fragment that has been proven to work through widespread application, and where its applicability, proven uses, and limits are carefully documented [2]. There are several Internet repositories in which one can search for patterns to solve various problems.

Patterns are a powerful way of archiving programming knowledge in an accessible form. One key part of a pattern – independent of the multiple ways that one can be described – is a statement of its limits, i.e., "how far you can stretch it without breaking it". Applying a pattern – in principle – substitutes for having the knowledge to create it. Using patterns creates code that is, in many cases, better than code written from scratch and based upon full knowledge of system function.

However, the contemporary programmer is not always fortunate enough to have true design patterns in hand. Documenting a pattern is a labour-intensive task. Thus, programmers turn to Internet sources containing "examples" that substitute for patterns, in the sense that the programmer can twist an example to accomplish a goal and/or combine examples toward a new purpose. But *examples are not patterns.* There is no documentation of what can be changed. There is no documentation as to applicability, scope, or limits; there is no claim of broad applicability. There is not even the implicit claim that an example has been rigorously tested.

But an example that has proven useful in a broad variety of contexts – even in a limited way – is not quite an example anymore. It is not yet a pattern and lacks many kinds of documentation. But it is a "ritual" that "usually works". There is some value in distinguishing between "examples", as untried, versus "rituals", as partly validated.

One might compare patterns, examples, and rituals with pharmaceuticals, folk medicines, and folk remedies. A pharmaceutical – like a pattern – has well-documented effects. A folk medicine, like a programming example, might create some effects when, e.g., made into tea. A folk remedy – like a programming ritual – is contrived from a folk medicine through some experience of success, but *without complete understanding of its effects.* In other words, a programming "ritual" is somewhat like a new drug without FDA approval. A ritual "just works", for some partly understood reason.

One weakness of using rituals is that finding new rituals requires some mix of guesswork, experimentation, and/or deep knowledge. Modifications to existing rituals – however minor – are *not* guaranteed to work, any more than modifications of gourmet recipes are; only the master chef knows what can be substituted.

As an example of this, I assigned my students to write a simple document search program in Hadoop, thinking that this was a straightforward program. Not! We got caught in the netherworld between Hadoop 0.19 and Hadoop 0.20, in which the syntax changed enough in 0.20 to invalidate all of the 0.19 tutorials. The tutorial examples said nothing about how to propagate the search terms to the mapper processes. Worse, the *only* candidate variable that we knew enables that kind of propagation had a type that was marked as deprecated! Through some educated guesswork and experimentation, we found out how to do it, though others before us did not fare as well, and we found one Internet lament that text search – which Hadoop was designed to do well – was impractical in Hadoop!

How did we succeed? Well, it is difficult to admit it, but we succeeded by locating a shamanistic ritual with a closely related outcome, searched the Web for related rituals, and then guessed what to do and verified the guess through experimentation, *thus creating our own personalised ritual.* I call this "ritual" and not "pattern", because in the process of making the program work, *we did not obtain a comprehensive idea of why it works, or its limits!*

A little play with modern Web frameworks such as Ruby on Rails, Symfony, and Cake will demonstrate why modern programmers think this way: one cannot deviate from predefined rituals without courting disaster and/or inconceivable behaviour. Frameworks have reached the complexity point where documenting their complete function to a programmer is impractical, or perhaps I should call it "unempowering". The total workings of modern programming frameworks are not that useful to know for someone using them. So we resort to shamanism and employ rituals that the creators of the frameworks kindly provide for us, or we guess and, from experimentation, create our own rituals to taste.

Engaging in ritual rather than understanding is not just exhibited by programmers. System administrators often crowdsource their configurations of software, for desktops or servers, based upon what rituals are found to work by others. The job of the system administrator is *not* to understand but, rather, just to repair the problem. Time pressure often limits personal exploration, so that successful repairs by others are important to know. Often, the quickest approach to troubleshooting is to mine "rituals that work" from the Internet. The mailing lists are full of these simple – but generally effective – uses of crowdsourcing.

From Shamanism to Alchemy

By this time, the reader may think I am advocating a return to barbarism, throwing away the knowledge of the past. I am instead pointing out that we are *already* barbaric, in the sense that our technology has already vastly surpassed our understanding. How empowering is it, for example, to take apart a cell phone? The physical structure of a cell phone is not empowering to someone looking to understand its function, which is mostly hidden.

And the barbarians are winning, because they can produce programs faster than the civilised programmers!

The modern technological shaman, like the primitive shaman, trusts his or her senses and engages in a kind of science. The difference between primitive shamanism and technological shamanism lies in what the shaman's senses include. The technological shaman has the observational powers of the Internet-connected world available and can crowdsource a solution to a mystifying problem simply by querying the proper mailing lists. The appropriate response to "Doctor, it hurts if I do this" is usually "Don't do that; do this"; a non-working ritual is countered with a working ritual, but without a satisfying explanation of why one ritual does not work while the other does.

Like a folk remedy, a modern ritual gains validity through direct observation of when it does and doesn't work. Thus its validity grows with application and observation, including observation of what requirements it does not meet. One severe downside is that there is no such thing as a "secure ritual"; a reasonably complete security analysis would transform it into a pattern!

Crowdsourced solutions are laced with shamanistic rituals that might do nothing, but were part of an initial pattern. I had a student who always put the statement "`X=X`" into his Matlab program before using a variable "`X`". This was ritual rather than knowledge (the statement does nothing); but it was extremely difficult to convince him – even with objective evidence to the contrary – that the statement was not needed. This shamanistic ritual reminded me of the rituals of aboriginal tribes who feed wooden birds because it seems to help the crops. Why do it? Because it might help and does not hurt!

One thing that greatly influenced my thinking on social ritual in technology was Jim Waldo's Project Darkstar at Sun Microsystems. Darkstar attempted to analyse the requirements for interactive role-playing games [3]. To me, the most surprising finding from Darkstar is that young people do not approach interactive RPGs as adults do; bugs are "features", workarounds are "rituals", and software quality is defined in terms of not losing long-term state (although losing short-term state is acceptable). In other words, if you engage in a ritual, the game should not erase your character (a matter of weeks of development), but erasing your character's development for the past day is relatively acceptable! The quality of the RPG system is relative to how it reacts to ritual and whether its reactions to ritual are appropriately bounded. Some Web frameworks could learn from this definition of quality!

So, what is my advice to young students of programming? I do not advise them to program as a "civilised" adult like myself; I am less facile than they are! I do not advise them to reject shamanism and ritual; technological ritual is a basic part of modern survival. I do tell them to develop their own observational skills to a high art, so that they can track a "personal alchemy" that describes *how their rituals interact*. The result of crowdsourcing this "personal alchemy" is a shared "technological alchemy" describing how rituals can be combined to achieve new goals. This social emergence of order – and not the traditional practice of reading and understanding the source code – is already the key to productive programming in the new world of large-scale frameworks.

From Alchemy to Chemistry

It is with some poetic justice that I – having shown no aptitude for undergraduate chemistry – am now put in the position of advocating its value! In the same way that alchemy became chemistry, we need a new "chemistry of programming" that describes how to combine "elements" (our rituals) into "molecules" (more involved rituals) that achieve our ends.

By chemistry, I am not referring to the precise rules of component-based programming or object-oriented programming but, instead, to the fuzzily defined rules by which rituals are combined into new rituals, without full knowledge of the structure behind the rituals. “Programming chemistry” is a matter of classifying what rituals are safe to combine, what combinations are questionable, and what combinations are likely to explode!

Am I advocating throwing away detailed knowledge? Certainly not. But this kind of knowledge – while valuable to the developers of a framework – is not empowering to the average programmer of a framework. Or, rather, it is as empowering as knowing the physics of electrons is to the chemical engineer. This is not a matter of throwing away knowledge but, instead, packaging it in a much more digestible form, in terms of what programmers should and should not do.

Generations

The difference between me and my students is quite generational. I was taught to be compulsive about knowing, in the sense that I will not stop peeling away layers of a thing until I know everything about how it works. This compulsion was empowering for me but is not empowering for my students. Or, it would be better to say, it is about as empowering for them as taking apart their cell phones to see how they work! To my students, I am somewhat of a dinosaur, and to me, my students are the new shamans of technology, engaging in dance and ritual to produce working code.

But my students are not lacking in ambition; they engage in their own unique flavour of lifelong learning. They learn the rituals that work, and the alchemy between rituals: their own descriptions of how to transmute base software components into “gold”. But, somehow, it all works, and often it does result in “gold”.

References

[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, illustrated edition, 1994.

[2] See, e.g., Chapter 12 of Roger Pressman, *Software Engineering, a Practitioners’ Approach*, 7th edition, McGraw-Hill, 2009.

[3] Project Darkstar: `http://www.projectdarkstar.com`.

*Originally published in **;login: The USENIX Magazine,**, volume 35, number 3, June 2010, (Berkeley, CA: USENIX Association, 2010). Reprinted by kind permission.*

HPC for Free

Andy Thomas

In the second and concluding part of this article, enough information will be given for you to set up a FreeBSD network boot server, configure the client PCs, work around some issues with making ssh host keys and crontabs persist across client PC reboots and set up scheduled reboots from FreeBSD into Windows and vice versa.

First, you need to set up the network boot server. Technically, just about any computer able to support NFS and TFTP with sufficient disk space to accommodate the boot image can be used for this – it need not run FreeBSD and it doesn’t even have to be based on x86 or x86_64 architecture. But it is so much easier to build, write scripts, develop additional software, maintain and make run-time changes if the network boot server is running the same architecture and version of FreeBSD as the network boot image the clients will be booting and only a masochist will choose to do this any other way. So we will keep things simple – if your client PCs are 32-bit, choose a 32-bit PC or server as the boot server and if you have x86_64 PCs, use a 64-bit machine instead; in this article, we will assume all your PCs are 64-bit.

Installing FreeBSD onto your boot server will not be described in any detail here as it is already well documented elsewhere – for a good start, look on the FreeBSD website (**`http://www.freebsd.org`**). Just make sure before you start that you have sufficient disk space in `/usr/local` for the netboot image. This will typically be 1-2 GB and you can't go wrong if you opt for the automatic default FreeBSD disk layout and partitioning during installation, as this will allocate most of the available disk space to the `/usr` partition anyway. If you plan to keep your client PCs' user home directories on a different server mounted via NFS, then a 10 GB disk is all that is required for the boot server but if you want the home directories to reside on the same server, then you'll need to use a larger or a secondary disk.

If you have an Internet network connection available, the quickest way to install FreeBSD on the boot server is to download the FreeBSD 8.1 'bootonly' ISO image from the FreeBSD website, burn it to a CD and then boot from the CD. This will start a minimal FreeBSD followed by the installer program which guides you through language and keyboard selection and then disk selection, creating the slices and partitions before going on to selecting the packages you want to install. Keeping things simple, opt for the default FreeBSD disk layout and partitioning scheme (unless you have a real need for a custom set-up) by pressing `A` followed by `Q` in the Fdisk Partition Editor menu and pressing these same keys again in the FreeBSD Disklabel Editor.

Next, choose the developer software selection including kernel sources and after you have negotiated the documentation language menu that follows, you will be invited to install the FreeBSD ports collection – it is highly recommended you answer 'yes' as this installs the skeleton framework of the ports directory hierarchy including makefiles, package descriptions, etc ready for when you want to add extra packages and doesn't take up much more than 400 MB of disk space. After a final 'last chance' screen warning you that all data on the hard disk will be lost if you decide to continue, the install starts and you may be pleasantly surprised at how quickly it proceeds compared with many Linux distributions.

Once the installation has completed the system will reboot and you'll (hopefully) be able to log in as root and go on to build the netboot image. Before you do this, you need to look at the PCs you will be using as your diskless clients and find out what NIC (network interface controller) hardware they use as you'll need to make sure your netboot kernel will support the NICs in all your client PCs. The generic FreeBSD kernel supports a huge range of network cards so it's very unlikely you'll need to add a driver unless your PCs are either very new or very unusual.

If your PCs are already running Windows you can do this by right-clicking on 'My Computer', and right-click on 'Properties' from the drop-down dialogue box and then select 'System Properties'. Now click on the 'Hardware' tab and then click on the 'Device Manager' button – you'll see a tree of hardware items so click on the '+' symbol next to 'Network Adapters' to expand this and reveal the NIC type and model number. Alternatively, boot the PC from a FreeBSD CD (a bootonly CD will do) and after selecting your country and keyboard in the installation menu, select the 'Fixit' option followed by 'Shell' – this will start a simple shell on the virtual console which you can reach by pressing `Alt-F4`. Giving the `ifconfig` command will list the network interfaces FreeBSD has found. Make a note of the NIC types you need to support as you will need this information when you customise your kernel later.

Creating the netboot image and custom kernel for your client PCs

Now that you know which NIC(s) your netboot kernel needs to support, you can trim down the generic kernel to make it smaller and load faster by removing support for NICs you definitely will be not be using in your client PCs. This is not an essential step and you could use the generic FreeBSD kernel if you want to but building a new kernel is so easy you might as well do this:

- `cd` to `/sys` (this is a symlink to `/usr/src/sys`) and find the machine architecture you're building the new kernel for – in our example, we are using 64-bit PCs so select `amd64` (it does not matter that your PCs might be using Intel processors).
- now change to the kernel configuration directory under `amd64` with `cd amd64/conf` and make a copy of the `GENERIC` kernel configuration file. You can call this file anything you like

but note that the configuration filename will appear in the output of `uname -a` when you boot your new kernel so choosing something meaningful here is a good idea; traditionally, FreeBSD config filenames are in uppercase so for this example we'll call it `KERN_01`.

- now edit your new config file with your favourite editor...What? Your favourite editor isn't installed, I hear you cry? No problem, just `cd` to `/usr/ports/editors` (you did install the ports collection as I recommended, didn't you?), select the editor you want, `cd` into it and type `make` followed by `make install`. For example:

```
cd /usr/ports/editors
ls
cd vim
make        # (and go and have a cup of tea or three)
make install
```

- when you are finally in your config file, you'll find a section containing a lot of lines beginning with the word 'device'. Here you can disable support for devices you know you'll not be using in your PCs by commenting out that line with a '#' (pound or hash) character; this will include rather a lot of NICs and quite a lot of other devices as well (for example SCSI tape changers and FC RAID controllers are unlikely to be attached to a normal desktop PC) so you can comment all these out as well.
- now save the file ready for the kernel build

You are now ready to build the netboot image followed by building the kernel and then installing both into the final location it will be served from by the tftp server. In our example, the netboot image will live in `/usr/local/diskless/pxeroot` so we first need to create this directory and then use the commands `make buildworld` and `makebuildkernel KERNCONF=<YOUR_KERNEL>` where `<YOUR_KERNEL>` is the name of your custom kernel configuration file to create a complete image before installing it into its final target directory:

```
mkdir -p /usr/local/diskless/pxeroot
cd /usr/src
make buildworld
make buildkernel KERNCONF=KERN_01
make DESTDIR=/usr/local/diskless/pxeroot installworld
make DESTDIR=/usr/local/diskless/pxeroot installkernel
make DESTDIR=/usr/local/diskless/pxeroot distribution
```

This will leave our new netboot image in `/usr/local/diskless/pxeroot` and the next step is to give it a boot loader – FreeBSD helpfully installs a PXE loader by default in `/boot/pxeboot` so you can use this to boot your client PCs too. We will copy this to a subdirectory called `pxeboot` under the root of your diskless installation:

```
mkdir /usr/local/diskless/pxeboot
cp /boot/pxeboot /usr/local/diskless/pxeboot
```

This makes it easier to add the optional Pxelinux boot control later if you wish.

Finally, one last task – your netboot image is complete and ready to be served by the TFTP server (which we will cover in a moment) but we have not created any user accounts in the image; you'll be able to boot it but you won't be able to log into any of the client PCs until you have done this. As well as the root account, you almost certainly created a user account for yourself during the FreeBSD install and you can add more accounts now with the `adduser` utility; when you are ready, duplicate the boot server account structure to your netboot image:

```
cp /etc/*passwd /usr/local/diskless/pxeroot/etc
cp /etc/*pwd.db /usr/local/diskless/pxeroot/etc
```

Setting up the NFS, TFTP and DHCP servers

In this article we will assume your boot server will be handling DHCP for the client PCs as well as serving the boot image via TFTP, that there is no other DHCP server on your network, that the client network addresses will be from the range 192.168.1.0/24 and that your boot server will be providing all the NFS filesystems mounted by the client PCs, including user home directories. If your network environment is different – for example, your organisation has a central DHCP server or your client PCs are served by a DHCP server on your local subnet – then you will need to talk to the administrator(s) of these servers to arrange for the DHCP configuration to be modified so that a PC requesting a DHCP lease is pointed at your netboot server's TFTP server once it has obtained the lease, as described below. Similarly, if you need to mount remote NFS shares from other servers, typically for centralised user home directories, you will need to liaise with their sysadmin(s) to allow your PCs to do this.

Secondly, a bit about the architecture we have chosen for this diskless scheme; the `/usr` filesystem in the netboot image is fairly basic and doesn't contain any ports or additional packages – there's just enough for the system to boot to a login prompt. But we export the netboot server's `/usr` filesystem to the client PCs and mount it on the client's `/usr` mount point, overlaying the existing minimal `/usr` filesystem of the netboot image. This might seem strange at first but it has some advantages – `/usr` is typically quite large and there may also be a lot of additional packages, the ports collection software, and so on in `/usr/local`; if this was part of the netboot image's `/usr` filesystem, it would occupy a lot of memory on the client PC that could otherwise be put to better use. Also, we can add new packages, additional software, scripts, etc to the `/usr` and `/usr/local` filesystems on the netboot server and they will immediately appear on all the client PCs as well!

Finally, in the default FreeBSD installation, `/home` is a symlink to `/usr/home` and on the client PCs, `/home` will also point to this location. So the netboot `/usr` filesystem can also contain the user home directories and the whole lot – `/usr`, `/usr/local` and `/usr/home` – can be exported to the clients as a single NFS share.

(Note: to keep things simple at this stage and to aid troubleshooting, initially we will not mount the netboot server's `/usr` filesystem on the client PCs until after we have successfully booted a client PC from the server).

Let's get on – first we set up and test the NFS server – NFS is installed by default with FreeBSD but not enabled. So we need to configure our NFS server so any PC in the 192.168.1.0/24 subnet can mount `/usr` from your netboot server; with your favourite editor, create the file `/etc/exports` (which will not already exist in a fresh installation) and add the line:

```
/usr -alldirs -ro -maproot=0:    -network 192.168.1  -mask 255.255.255.0
```

Now you need to enable and start both `nfsd` and `rpcbind` by editing `/etc/rc.conf` and adding the two lines:

```
nfs_server_enable="YES"
rpcbind_enable="YES"
```

and then after saving this file, start the NFS daemon with:

```
/etc/rc.d/nfsd start
```

You can confirm your NFS server is indeed exporting the netboot `/usr` filesystem with:

```
showmount -e localhost
```

You should see output like:

```
Exports list on localhost:
/usr                192.168.1.0
```

Next we configure and test the TFTP server – like `nfsd`, `tftpd` is part of the default FreeBSD install and lives in `/usr/libexec/tftpd` but it is not enabled by default for security reasons (neither is the `inetd` daemon) so you will need to edit your `/etc/inetd.conf` file to enable the `tftp` service by simply removing the '`#`' hash (pound, for those of you in the US) character from the start of the line that reads:

```
#tftp   dgram   udp   wait  root  /usr/libexec/tftpd    tftpd -l -s /tftpboot
```

Save your `inetd.conf` file and to make sure `inetd` starts whenever you boot your netboot server, add the following line to your `/etc/rc.conf`:

```
inetd_enable="YES"
```

and you can start `inetd` right away with

```
/etc/rc.d/inetd start
```

Now we can test the `tftp` server by linking `/tftpboot` (it doesn't actually exist yet) to a directory containing recognisable files, for example to your `/etc` directory:

```
ln -s /etc /tftpboot
```

and then using your netboot server's own `tftp` client to connect and retrieve the hosts file from `/etc`:

```
cd /tmp     # (you don't want to risk overwriting /etc/hosts !)
tftp localhost
tftp> get hosts
Received 1127 bytes in 0.0 seconds
tftp> quit
```

You should find a copy of `/etc/hosts` in `/tmp`, proving that your `tftp` server is working. If you have access to another FreeBSD, UNIX or Linux system, you can optionally check that you can use a `tftp` client on these to retrieve files from your netboot server. Finally we will create the symlink `/tftpboot` to point to the diskless FreeBSD boot loader in `/usr/local/diskless/pxeboot` (later, once everything is working, we can optionally install Pxelinux so that you can control what the PCs boot up into from the relative comfort of your netboot server):

```
rm /tftpboot    # (delete the symlink we used earlier for testing)
ln -s /usr/local/diskless/pxeboot /tftpboot
```

With the NFS and TFTP servers set up and tested we can move on to the DHCP server – this will not have been installed on your netboot server during the initial FreeBSD installation so you need to install it now from the ports collection:

```
cd /usr/usr/ports/net/isc-dhcp41-server
make
make install
```

Now we need to configure it by editing `/usr/local/etc/dhcpd.conf`; assume you are using the 192.168.1.0/24 RFC 1918 private address range with the following settings:

- 100 addresses starting at 192.168.1.10 reserved for your client PCs
- your router (network gateway) address is 192.168.1.1
- your netboot server has the IP address 192.168.1.2
- you are using the OpenDNS resolvers at 208.67.222.222 and 208.67.220.220

(the OpenDNS resolvers are shown here for simplicity as anyone can use these but you may get quicker lookups if you use your organisation's or ISP's DNS resolvers so substitute their IP addresses for the OpenDNS addresses in your dhcpd configuration file). The sample DHCP server configuration file contains a wealth of useful information and is quite large, so I'd suggest renaming this and creating your own copy from it with the following simple configuration:

```
# dhcpd.conf
#
# Configuration file for ISC dhcpd
#
default-lease-time 600;
```

```
max-lease-time 7200;
option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option routers 192.168.1.1;
option domain-name-servers 208.67.222.222, 208.67.220.220;

subnet 192.168.1.0 netmask 255.255.255.0 {
range 192.168.1.10 192.168.1.109;
}

group { # diskless clients
next-server 192.168.1.2;
filename "pxeboot/pxeboot";
option root-path "192.168.1.2:/usr/local/diskless/pxeroot";
}

ddns-update-style ad-hoc;
```

This simple configuration will allow any PC set by its BIOS to attempt a network boot to boot from your netboot server but you will probably want more control over which PCs join your HPC and to assign the same IP address to each PC every time you start the cluster. To do this you will need to know the MAC (Ethernet) address of each PC and then add a host declaration for each inside your group declaration like this, assuming you want your PCs to have the hostnames `pc-1`, `pc-2` and so on:

```
group { # diskless clients
    next-server 192.168.1.2;
    filename "pxeboot/pxeboot";
    option root-path "192.168.1.2:/usr/local/diskless/pxeroot";

    host pc-1 {
        hardware ethernet 00:17:31:db:9b:8d;
        fixed-address 192.168.1.10;
    }

    host pc-2 {
        hardware ethernet 00:e0:18:fe:62:e6;
        fixed-address 192.168.1.11;
    }
    <add more host declarations here>
}
```

Once you have saved your `dhcpd.conf` file, you can start the DHCP server with the command:

```
/usr/local/etc/rc.d/isc-dhcpd start
```

First boot

Now go to one of your PCs and do whatever you have to do in the machine's BIOS or set-up utility to set it to boot first from the network via PXE – PCs vary greatly in how this function is described depending on the PC's make and BIOS type and this will not be covered here. Once you have done this, save your BIOS or set-up settings and do a cold boot (your PC may do this for you as soon as you exit the BIOS set-up). Again, the messages you will see on your PC will vary according to the PC, BIOS and version of PXE installed but generally you will see messages to the effect that the PC is contacting the DHCP server followed shortly afterwards by the FreeBSD bootstrap loader message. A message about the BTX loader will flash by and then you should be presented with the default boot menu. If you do nothing, the PC will boot FreeBSD after 10 seconds – this will all happen very quickly so try not to blink otherwise you may miss a possible warning message.

All being well, you should now be able to log into the PC as root and you should also be able to log in as one of the users whose accounts you set up earlier although you will get a message about that user's home directory not being found – we will fix this in the next boot. Congratulations!

Fixing a few rough edges and adding enhancements

The basic system is now working but to make it more usable as a HPC cluster, we need to do a few more things:

- mount the netboot server's `/usr` filesystem on the client PCs
- fix the ssh hosts keys; at each boot, the client PCs will generate a new set of RSA and DSA private/public host key pairs. This is a nuisance if you have a part-time HPC where the client PCs have to frequently shut down and be rebooted back into some other local operating system from their hard disks (such as Windows) as HPC users will often be using their own ssh private/public keypairs for unattended scripted logins. These automated logins will fail at every client PC boot as the user will be prompted to take action on the changed ssh host keys, being warned of a possible man-in-the-middle attack! (Note that we cannot save these 3 ssh keypairs in the netboot image as each client will require its own set linked to its hostname.)
- you might want to set up some cron jobs on the clients – for example, to reboot systems out of FreeBSD and back into Windows early each morning. But as the `/var` filesystem is in MFS (memory file system) this is volatile. So we need to restore the crontabs from a backup file from a disk-based filesystem
- finally, you might want to install Pxelinux so you can have more control over how the PC clients boot

Let's fix these one by one;

Mount the netboot server's /usr filesystem

Mounting the server's `/usr` filesystem on the client PCs can be done by adding the following line to the netboot image's `/etc/fstab` on the server, i.e. `/usr/local/diskless/etc/fstab`:

```
192.168.1.2:/usr      /usr      nfs     ro      0 0
```

and save the file. Next time a client PC boots from the server, it will overlay its own `/usr` filesystem with the server's.

Fix the ssh host keys

Next, the ssh host keys. This is actually a bit of a kludge and when I get a bit of free time, I'll try and work out a more elegant solution! Basically it involves booting all the client PCs in your cluster, logging into each and then saving the 6 ssh keypair files listed below into a `tar.gz` backup in a subdirectory on the netboot server named after the client's hostname:

```
/etc/sshd/ssh_host_key
/etc/sshd/ssh_host_key.pub
/etc/sshd/ssh_host_dsa_key
/etc/sshd/ssh_host_dsa_key.pub
/etc/sshd/ssh_host_rsa_key
/etc/sshd/ssh_host_rsa_key.pub
```

For example, to save the keys for a host called `pc-32`, we would first create somewhere to store the backup in the netboot image:

```
mkdir -p /usr/local/diskless/pxeroot/root/ssh-keys-backup/pc-32
```

and then after logging into `pc-32` as root, archive the current keys and copy them back to the boot server:

```
cd /etc/ssh
find ./ -name 'ssh_host_*' > /tmp/ssh-host-keys.list
tar czf /tmp/sshd-keys.tgz -T /tmp/ssh-host-keys.list
scp /tmp/sshd-keys.tgz 192.168.1.2:/usr/local/diskless/pxeroot/root/ssh-keys-backup/pc-32
```

This has to be repeated for every client PC in your cluster but once you have done it, you will not have to do this again (unless the hostname or MAC address changes). Now we have a permanent set of keys, we simply need to restore these at client boot time and restart sshd by adding this to your `/etc/rc.local` script (create this script if it does not already exist):

```
#!/bin/sh
# rc.local
# script to start things after the system has booted but before the login
# prompt appears
# now we need to restore the ssh keys from the permanent backup under /root
# to /etc/ssh then restart sshd

HOST=`/bin/hostname | /usr/bin/cut -d'.' -f1`
tar xzp -C /etc/ssh -f /root/ssh-keys-backup/${HOST}/sshd-keys.tgz
/etc/rc.d/sshd restart
```

Restoring crontabs to the client PCs

The client crontabs can be backed up and then restored at boot time in a similar fashion but it's much easier as the cron tabs are not host-specific so the same tabs will work for all the client PCs. Simply create somewhere in the server's netboot image to save them:

```
mkdir -p /usr/local/diskless/pxeroot/root/crontab-backup
```

and then log into a client PC as root, set up your cron jobs in the usual way with `crontab -e` and then save the crontab files back into the netboot image on the server:

```
scp /var/cron/tabs/root 192.168.1.2:/usr/local/diskless/pxeroot/root/crontab-backup
```

Adding these lines to your `/usr/local/diskless/pxeroot/etc/rc.local` will restore the crontabs on each PC after it has booted and then restart cron:

```
# restore crontabs from a backup
if [ -d /var/cron/tabs ]; then
    cp /root/crontab-backup/* /var/cron/tabs
fi
/etc/rc.d/cron restart
```

Control client PC booting with Pxelinux

The final enhancement is to install Pxelinux which will allow you to select whether the client PCs boot into FreeBSD over the network or from their local hard drives. Despite the name, Pxelinux is not a Linux utility, it is written in 32-bit x86 assembler and will execute on any PC before any operating system has been selected or booted. First download the latest Syslinux from the project pages on **http://syslinux.zytor.com/** to your netboot server and unpack the distribution; you'll find a subdirectory called `core` and within that, a file called `pxelinux.0`. This is a pre-compiled binary and you need to copy this to a subdirectory where you serve your netboot image from and then create a further folder below this called `pxelinux.cfg` for the default configuration file you will be creating.

For example, assuming you've unpacked Syslinux 4.02 into `/tmp`:

```
mkdir -p /usr/local/diskless/pxelinux/pxelinux.cfg
cp /tmp/syslinux-4.02/core/pxelinux.0 /usr/local/diskless/pxelinux
```

Pxelinux will look for the configuration file `./pxelinux.cfg/default` relative to the location of `pxelinux.0`. In our setup this is `/usr/local/diskless/pxelinux/pxelinux.cfg`.

A simple config file that will wait 10 seconds and then boot whatever operating system is installed on the PC client's local hard disk is shown below:

```
# Perform a local boot by default
default local
```

```
# Boot automatically after 1 seconds
timeout 10

label local
    localboot 0
```

but there are many other options so read the Syslinux and Pxelinux documentation if you want to get the most out of this facility. Now, remember the `/tftpboot` symlink we created pointing to `/usr/local/diskless/pxeboot` when we set up our `tftpd` server? Well, if we deleted this and instead created the link:

```
ls -s /usr/local/diskless/pxelinux /tftpboot
```

then when a client PC boots from your server, it will boot from its own hard drive (Windows or Linux, etc). It's easy to change this link from a script and that script could be executed by a cron job, paving the way to a fully automated switch of operating system between FreeBSD and, say, Windows at scheduled times.

For instance, supposing the client PCs are all running FreeBSD and you want them to reboot back into Windows at 8am each morning; a cron job running on your netboot server can switch this link to point to `/usr/local/diskless/pxelinux` sometime before 8am while a cron job running on the PC client can reboot the system at 8am using the command:

```
/sbin/shutdown -r now
```

When the client PC comes back up and boots from your netboot server again, it will execute Pxelinux instead which will hand control over to the PC's local hard disk and boot it into Windows.

Shutting down from Windows and rebooting into FreeBSD

Finally, for completeness some information on scheduling Windows PCs to shut down and then boot into FreeBSD from a netboot server; rebooting in Windows is done with the `shutdown.exe` utility which is supplied with Windows XP while if you're still running Windows 2000 or even NT 4.0, then you'll need to install this utility from the Windows 2000 Resource kit CD-ROM (there never was an official `shutdown.exe` for NT 4 but the Windows 2000 version does work with this version of Windows). The Windows equivalent of UNIX cron is called 'at' and you need to make sure that the Task Scheduler service is running for this utility to work.

Rather then bloating this article any more than it already is, Microsoft actually has some excellent documentation on using `shutdown.exe` at **http://support.microsoft.com/?kbid=317371** while the use of at is described in detail at **http://support.microsoft.com/kb/313565** so this information will not be repeated here. Both refer to Windows 2000 but are equally applicable to Windows XP and although I have not tested these for myself, it apparently applies with minor differences to PCs running Windows Vista and Windows 7 as well.

Conclusion

Hopefully you will find enough information here to get started on using FreeBSD and ordinary PCs to create large diskless HPC clusters for little or no cost. Quite a lot more could be written on the topic but condensing 7 years experience of doing this as a production service for the Maths Department of Imperial College London into two newsletter articles isn't easy. Development of Imperial's SCAN (SuperComputer At Night) clusters, described in the first part of this article, is ongoing and the Torque (formerly OpenPBS) cluster job queuing system developed originally by NASA has recently been implemented cluster-wide along with the Maui scheduler.

Finally my thanks go to Dr Dan Moore of the Maths Department for allowing me to use all of his PC teaching clusters for this project and for having faith in the idea from the beginning and also, to my friend Dr Gunnar Pruessner, now a lecturer in the department, for all his help and encouragement, for testing and using the clusters and for all those Staropramens in the Queens Arms.

Network Flow Analysis

Michael Lucas

No Starch Press
ISBN: 978-1-59327-203-6
224pp.
£ 31.49
Published: June 2010

reviewed by Andy Thomas

Michael Lucas is arguably the best writer on network and systems topics. Full stop.

That deserved a paragraph all to itself. I was interested in reviewing this book as I was dimly aware of network flow analysis having seen it demonstrated on a PC about 15 years ago but always assumed it required horrendously expensive commercial software such as HP Openview, various CiscoWorks tools, etc. And a Windows PC, which I don't have. So the offer of reviewing a book on open source network monitoring and flow analysis was hard to resist but after I had read a couple of chapters there were even more compelling reasons; the book is easy to read, it engages the reader, explanations are thorough without being wordy and verbose or, worse still, condescending towards a reader who may not be an expert on the topic. There was something about the author's style that I both liked and at the same time seemed familiar to me and then I glanced at the rear cover and noticed the author wrote 'Absolute BSD'. It all clicked into place, my well-thumbed copy of Absolute BSD is one of the best systems books I have.

At 204 pages it's not a thick book but with the usual high text to white space ratio you come to expect from a book from the No Starch stable, it is packed with information, starting with a look at the various monitoring tools, the evolution of Cisco's NetFlow flow information system before moving on to discuss the common flow collectors cflowd, flowd and flow-tools. Settling on flow-tools, building this from source, installing and configuring this popular suite is described in detail. Flow sensors come next, both sensors implemented by manufacturers in network hardware and software sensors like softflowd.

With your flow analysis system set up we're ready to start using it to capture and view flows in chapter 3. At this point you realise you have collected masses and masses of data and it's hard to see to see the network traffic for all the packets but then chapter 4 comes to the rescue with timely help and advice on filtering and suddenly, the network picture becomes much clearer. But we need to do something with the filtered data we now have so that we can first see what is going on in our network, deduce the possible problems lurking in it and last, but not least for some of us, produce impressive-looking reports for the benefit of the management; chapter 5 introduces us to flow-report and its bewildering array of options while chapter 6 looks at FlowScan, the problems with installing the Cflow.pm Perl module upon which it depends and rounds off with a discussion of CUFlow.

A plethora of graphical and visualisation tools follow in chapter 7 – FlowViewer, FlowTracker and FlowGrapher and their pros and cons are explained clearly in detail leaving the reader in no doubt which is the best for his or her needs. Now a surprise – chapter 8 contains probably the best tutorial I have seen on using gnuplot and this really is an unexpected bonus! gnuplot is used to produce plotted graphs from captured data passed through the various utilities that make up flow-tools.

Finally, chapter 9 begins with an introduction to the newer flowd which supports the latest Netflow 9 data format used mainly for IPv6 and sFlow, which is the data export format used by HP and Extreme, among other network hardware vendors. Conversion of both flowd and sFlow data to Netflow format is discussed and this, the last chapter in the book, rounds off with a number of real-world case studies of how flow-tools has been used to track down persistent network problems.

Every chapter contains full details on how to build, install and configure each piece of software along with many examples of command line usage (some of them very long pipes). Little black circles with inverse numbers are used to annotate command line parameters and connect them to the surrounding explanatory text which is an unusual but very effective idea that works.

This book is highly recommended for those who really want to know what is going on in their networks, where the traffic is coming from and where it is going to (or not, as the case may be). I really enjoyed reviewing this book and it is still open on my desk today. In fact, I ought to confess that on the first day after I'd read chapter 3 during my daily commute, I rushed home quite excited about the possibility of using open source network analysis tools on my networks. I stayed up until dawn the next day installing and using flow-tools both at home and on other networks I'm responsible for. For the first time in years, I felt I'd found something that I can actually PLAY with while at the same time learning a lot of useful new things about networking.

And at last, I have found a real use for that monster Cisco Catalyst 5505 switch I inherited a few years ago and which has sat under a desk at home ever since. It's been powered on and much use has been made of it while working through the book, particularly the hardware flow sensors section. Borrowing SuSE's catchphrase, I had a lot of fun and that can't be bad for a jaded old systems guy like me. Thank you, Michael!

DocBook 5: The Definitive Guide

Norman Walsh and Richard L Hamilton

O'Reilly Media
ISBN: 978-0-596-80502-9
560pp.
£ 38.50
Published: May 2010

reviewed by Paul Waring

DocBook has been around for nearly two decades, and is used for documentation in a number of open source projects, as well as The Linux Documentation Project (`http://tldp.org`). Unfortunately the previous edition of this book has been out of print for some time, but the release of DocBook 5.0 in late 2009 has prompted a new and updated edition.

The first chapter provides a brief introduction to DocBook, although most of the text focuses on the differences between 5.0 and earlier versions, which is of minimal interest if you are learning DocBook for the first time. One thing which the book makes clear up front is that SGML and Docbook DTDs 4.x and earlier are not covered from here on. This is a good way to start fresh without worrying about bogging readers down with legacy information, though it does mean that the book is not the right choice for anyone who has to deal with old SGML documents.

Moving on to the second chapter, the authors outline the basics of creating a DocBook document. All the standard areas are covered, including how to create indexes and bibliographies, although the detail is somewhat thin and you are expected to use the reference section to fill in the gaps.

The next chapter covers validating documents, though at only five pages you could easily miss it. The same applies to chapter four, which manages to spend less than four pages on stylesheets, at the end of which I was none the wiser as to how to incorporate one into my document. Chapter five manages a bit better, containing a useful guide on how to customise DocBook by adding and removing elements.

After only seventy pages of tutorial text, the rest of the book is devoted to a reference guide which describes every element in DocBook 5.0. I was somewhat disappointed by this as I had expected more tutorial information and examples, but this was partially made up for by how comprehensive and detailed the reference is. Finally, the book is rounded off with plenty of links to online resources, including schemas, W3C documentation and more. Whilst you could probably find most of this through a quick search online, having it all collated in one place is certainly useful.

Overall, this is an excellent reference guide to DocBook. However, it is written more in the style of the O'Reilly "In a Nutshell" series, so if you are looking for plenty of tips and examples before jumping into a detailed reference section, this is probably not the right book for you.

Java: The Good Parts

Jim Waldo
O'Reilly Media
ISBN: 978-0-596-80373-5
192pp.
£ 22.99
Published: April 2010

reviewed by Mike Fowler

Though I know it's bad to judge a book by its cover, I presumed this book would be another O'Reilly-couple-hundred-paged tome. Imagine my surprise when such a thin book arrived. Surely Java has more good parts than can be contained in these pages?

Actually this is a book about the core Java language itself, not all the thousands of frameworks that have popped up in the last few years. The author himself oversaw the development of some of these good parts giving him a very unique perspective. Armed with an insider's view, the author explains to us why and how some of the design decisions that were made.

For each of the nine good parts the author has chosen we are treated to a small tutorial teaching us the basics. Interspersed with development anecdotes and some interesting history, the author thoroughly explain why each good part is truly good.

Of particular note is the discourse on generics. I've long known that generics are largely syntactic that perhaps don't live up to their potential. Ways of defeating the generics system are well documented. The author explains that modifying the JVM to accommodate generics was not an option, amongst other design constraints. In this new light I have new found respect for the way generics work.

Sadly at times the book seems disjointed. Topics were often suddenly introduced only to disappear after a paragraph, reappearing two or three paragraphs later. This interrupts the flow somewhat, sometimes causing you to re-read a section in order for it to make sense. To me it almost seems as if some late re-ordering of paragraphs occurred without any thorough proof reading.

This book was an enjoyable and educational read. For me, it satisfies the author's goals of reminding me of some of the good parts and indeed introduced me to some new ideas. I expect this book to be of particular interest to Java practitioners who fell their Java is a touch rusty and experienced developers lost in the details of all the modern frameworks.

Cocoa and Objective-C

O'Reilly Media
ISBN: 978-0-596-80479-4
416pp.
£ 26.99
Published: April 2010

reviewed by Mike Smith

This book tries to cover a great deal of ground. I think a little too much, for its 380-odd pages of content. The title focuses on Apple's development language and framework, but after an initial short chapter on installing Xcode we actually veer off into covering basic (and slightly more advanced) C programming. (Even dropping to the command line to run gcc; not using Xcode at all!) So perhaps those 80 or so pages are unnecessary.

This means that in the remaining 300 or so pages we have to cover the Objective-C language, key Classes and a few subjects that the author has chosen to go into more detail (predominantly graphics orientated). There is a chapter on MVC (Model, View, Controller) that may be use for those that don't know how to set up Xcode applications; tying together the code and the views with Interface Builder. (Though I'd also recommend you look at the CS193P course videos in iTunes U for this.)

There's an out-of-place discussion on refactoring that came in near the end, nestled between two graphics topics, and a section on handling mouse and keyboard events – little more than listing out the methods used to handle those events. This didn't really flow for me.

It depends on your starting point but this is not at all what I was expecting. The strap line of the book is "Foundations of Mac, iPhone, and iPad Programming"; iPhone and iPad are also mentioned in the introduction and on the back cover – "Build solid applications for Mac OS X, iPhone and iPad". But that's it. There's no discussion on what the practical differences are when developing for these platforms (setting target device in Xcode, using the simulator, the additional classes you might want to make use of – WebKit, Core Location, interaction with the hardware, device orientation, dealing with multi-touch etc). To be fair, the strap line is true – it's the foundations below everything else, but there should be no implication that you will be able to build iPhone apps after reading this book.

So I can't really recommend this book, or at least need to reset your expectations (if they were like mine) based on the title and overview.

It's worth pointing out that there's a similarly titled O'Reilly book, *Learning Cocoa and Objective-C*, which is at least 8 years old. I'm sure a lot will have changed in the intervening period (not least Xcode used to be called Project Builder back then) so don't confuse that book with this one. But perhaps neither will be what you're looking for.

As I didn't really get on with this book, I didn't work through the coding examples as I normally do, so can't vouch for their accuracy.

I'm looking forward to Xcode 4 and all of its magical loveliness. I know many people prefer Eclipse, but it will be good to have Interface Builder completely integrated instead of a separate app, and the current clunky way of setting up and linking outlets.

Hackers

Steven Levy
O'Reilly Media
ISBN: 978-1-449-38839-3
528pp.
£ 16.99
Published: May 2010

reviewed by Roger Whittaker

This book was first published in 1984, and this new version is billed as a "25th anniversary" edition (maybe a year late?), and has a specially additional written afterword by the author.

Levy's book was highly influential in generating awareness in the wider culture of a particular mentality and outlook which he discovered in various individuals and groups that he identified as (in the words of the subtitle) "heroes of the computer revolution". By identifying a "Hacker Culture" and, through observation of its members, defining or at least attempting to summarise a "Hacker Ethic", Levy helped to open the eyes of the wider society to the character and motivations of some of the people who were in the process of changing it by advancing technology.

The Hacker Ethic that Levy defines right at the start of the book consists of the following statement, that he distilled from the attitudes of those people he was writing about, particularly in the first section of the book. The general principles of this Hacker Ethic are *Sharing*, *Openness*, *Decentralisation*, *Free access to computers* and *World Improvement*, and expressed more specifically:

- Access to computers – and anything that might teach you something about the way the world works – should be unlimited and total. Always yield to the hands-on imperative
- All Information should be Free
- Mistrust Authority – promote decentralisation
- Hackers should be judged by their hacking, not bogus criteria such as degrees, age, race or position
- You can create art and beauty on a computer
- Computers can change your life for the better

The book is in four sections, covering different periods of history. The first, mostly about the characters who inhabited the MIT AI Lab in the 1960s, is probably the most enlightening and entertaining. Even if you have never read the book before, some of the anecdotes here will be familiar, as they have been widely retold, often with Levy's book as the unacknowledged source. What he brings out most clearly in this section is how an obsessive desire to understand how things work led to an explosion of brilliant inventiveness among a group of young people whose habits of thought, work, eating, sleeping (and, yes, sometimes personal hygiene) were unconventional in the extreme. To these people, understanding how systems worked and making them do interesting things was the only motivation. Every locked door (both metaphorically and literally) was a challenge. Physical locks were defeated by locksmithing abilities of a very high order (reminiscent of the amateur safecracking work at Los Alamos some 20 years before by that other great eccentric obsessive truth seeker Feynman), while artificial restrictions of any kind were circumvented by clever hacks that left those who imposed them looking foolish. A wise management at the Lab understood the kind of people the hackers were, turned a blind eye to all but the most outrageous infractions of the rules and in return some highly creative and innovative work got done in a fraction of the time that would have been needed in a top-down managed corporate environment.

The second section of the book tells the story of the hardware hackers of the seventies: the people who realised that it was now a realistic possibility to put together some reasonably readily accessible components and make your own micro-computer. The Home Brew Computer Club in California was where these efforts were incubated, with the Apple II among the eventual results. Levy traces the way in which the home computer industry developed, and, in a theme that is repeated later in the book, sees a slow tragedy unfolding as the hacker ethic was overcome by commercialism. People who had started producing microcomputers commercially stopped attending the Home Brew Computer Club because if they did they would have had to give away their new trade secrets. But the machines that became affordable at this time for people to have in their homes offered a new and broader group of people the opportunity to hack away, try to understand how these systems worked, and, in many cases, to write programs not just in the BASIC that was provided with the machine but also in assembly language.

In the third section, Levy looks at some of the people who, almost by accident, got rich by writing and selling software and games for home users to run on these new microcomputers: the Apple II, the Atari, the Commodore and others. He mentions Bill Gates' famous Open Letter to Hobbyists, seeing it as the opening shot in the battle between commercial interests and those who could not understand why information should not be free. He looks in some detail at how one particular games startup gradually put the Hacker Ethic behind it as a result of the profit motive, copyright questions and the need to become at least minimally organised in order to operate in a competitive commercial environment, where some of the hardware companies now had their own software divisions and were out to crush independent competition.

The fourth and last section of the original book is entitled "The Last of the True Hackers" and contains a portrait of Richard Stallman as he was in 1983 at Cambridge, Massachusetts. Describing himself as the "last survivor of a dead culture", Stallman is portrayed by Levy at this period as a sad but defiant figure. The GNU project is mentioned, but at the time it was in its infancy.

In general the message of the book is a rather pessimistic one: there were once giants in the land; people with extraordinary skills and an extraordinary culture; people motivated only by the desire to understand and use the system, but their skills were tamed and their culture crushed by those whose motivations were profit, control and secrecy.

But with hindsight we know that the story is not such a simple one, and so does Levy. There are two “afterwords”, one written ten years after the first publication, and one this year to go with this new edition.

The *Ten Years After* afterword is fairly short, but is interesting to read for Levy’s discussion of the word “hacker” itself. He tells us that in 1984 he was almost forced by his publisher to change the name of the book on the grounds that no-one would know what a hacker was. But the book’s publication popularised the word, with the unintended consequence that it was widely taken up by the media and mostly applied to those with destructive intentions. He regrets this, but thinks it was inevitable. He singles out a CNN report in 1988 that made a big media scare out of the “Hacker Menace”. But at the same time he comments on the effect of the growth of the Internet and the way in which it allowed hacker culture to spread, as well as ‘cyberpunk’ and related phenomena that made hacker culture cool again.

The final *Afterword: 2010* starts with a quote from Bill Gates meeting Levy again and reflecting on the passage of time – “When we did the microprocessor revolution there was nobody old, *nobody*. . . . It’s weird how old this industry has become.” Gates gives Levy a lecture on “intellectual property”. But the author also revisits Stallman, and despite making the inevitable comments on his personal unhappiness and inflexibility, Levy acknowledges the achievements of the GNU project and today’s Free Software and Open Source hackers, and he discusses Linux, but disappointingly briefly. In a way, I found the 2010 afterword the most disappointing part of the book, because there is so much more to say, and despite all the conflicts and challenges, the hacker ethic is still very much with us, and indeed has triumphed in a way that the author could not forsee when he wrote the original book in 1984.

If you have not read this book, it is certainly to be recommended. It is a pity that Levy did not attempt to bring it really up to date, but that would have required a whole new volume, telling many new and wonderful stories, particularly of the history of the World Wide Web, and the Linux world that began in 1991. But others (including Glyn Moody, Russell Pavlicek and Clay Shirky) have written those stories elsewhere, and I urge you to read their books too.

Contributors

John Collins is a UKUUG Council member and runs Xi Software Ltd which specialises in System Management tools for UNIX and Linux systems. Xi Software is based in Welwyn Garden City and has been in existence for about 25 years.

Alva Couch is an associate professor of computer science at Tufts University, where he and his students study the theory and practice of network and system administration. He served as program chair of LISA ’02 and was a recipient of the 2003 SAGE Outstanding Achievement Award for contributions to the theory of system administration. He currently serves as Secretary of the USENIX Board of Directors.

Mike Fowler is a Senior Software Engineer at Dotted Eyes. He is a firm open-source advocate and has helped companies move from Microsoft based solutions to complete Linux/PostgreSQL stacks. He is a major contributor to YAWL, an open source BPM/Workflow system. When not melded with a keyboard, he spends his time playing as many different stringed instruments as he can.

Jane Morrison is Company Secretary and Administrator for UKUUG, and manages the UKUUG office at the Manor House in Buntingford. She has been involved with UKUUG administration since 1987. In addition to UKUUG, Jane is Company Secretary for a trade association (Fibreoptic Industry Association) that she also runs from the Manor House office.

Mike Smith works in the Chief Technology Office of a major European listed outsourcing company, setting technical strategy and working with hardware and software vendors to bring innovative solutions to its clients. He has 20 years experience in the industry, including mid-range technical support roles and has experience with AIX, Dynix/ptx, HP-UX, Irix, Reliant UNIX, Solaris and of course Linux.

Andy Thomas is a UNIX/Linux systems administrator working for Dijit New Media and for Imperial College London and as a freelancer. Having started with Linux when it first appeared in the early 1990's, he now enjoys working with a variety of UNIX and Linux distributions and has a particular interest in high availability systems and parallel compute clusters.

Paul Waring is chairman of UKUUG and is currently the Technical Manager for an insurance intermediary.

Roger Whittaker works for Novell Technical Services at Bracknell supporting major Linux accounts in the UK. He is also the UKUUG Newsletter Editor, and co-author of three successive versions of a SUSE book published by Wiley.

Contacts

Paul Waring
UKUUG Chairman
Manchester

John M Collins
Council member
Welwyn Garden City

Phil Hands
Council member
London

Holger Kraus
Council member
Leicester

Niall Mansfield
Council member
Cambridge

John Pinner
Council member
Sutton Coldfield

Howard Thomson
Treasurer; Council member
Ashford, Middlesex

Jane Morrison
UKUUG Secretariat
PO Box 37
Buntingford
Herts
SG9 9UQ
Tel: 01763 273475
Fax: 01763 273255
office@ukuug.org

Roger Whittaker
Newsletter Editor
London

Alain Williams
UKUUG System Administrator
Watford

Sam Smith
Events and Website
Manchester