BSD vs. Linux
A short comparison

One of the most frequently asked questions is the one how BSD is different from Linux. We try to answer that question in short. This shall not be used as a flaimebait. We admit prefer using BSD, but that is personal based on experience.

We will not go too much into technical details, we want to explain commonalities and differences. This flyer is mainly based on the work of Matthew D. Fuller, Greg Lehey and Dru Lavigne, please see the link section at the end.

What is Unix?

Unix is an operating system originally developed in the late 60's at Bell Labs by Ken Thompson and Dennis Ritchie. Maybe Unix is the single most influential operating system in modern computing. Every general-purpose computing device you'll find, and a lot of specific-purpose computing devices, will be using ideas, concepts and often code from something in the Unix family tree.

When we use the word 'Unix', then, we far more often mean the general form, than the specific OS that carries the name UNIX™. The general form means "Any operating system which, in design and execution and interface and general taste, is substantially similar to the Unix system." That means all the BSDs, Linux, Solaris, AIX, HP/UX, and a cast of hundreds or thousands of others.

BSD Myths

There are all sorts of myths, objections and "common knowledge" about BSD. Here are a few of them:

Hardware

"BSD doesn't support common hardware." Does Linux support hardware that BSD doesn't? Probably. Does it matter? Only if you have that hardware. Does Windows support hardware Linux doesn't? For that matter, MacOS probably supports hardware that none of the rest do. BSD supports most common hardware you'd stick in a server, workstation or desktop. There are gaps, but the gaps change from release to release, just like every other system. All BSDs have there hardware lists, you can easily check them before you try to install a specific BSD or you can use a Live CD like FreeSBIE; OliveBSD or NewBIE.

Program Availability

"But Linux has more programs than BSD!" Simply not true. If any given application is written reasonably portably, 98% or better of it will compile right off on any POSIX-compliant system. FreeBSD and Debian have the largest collections of Open Source software, compare them on your own. All the BSDs have a Linux emulation layer that provides binary compatibility. It won't always do the trick but it works surprisingly often.

Popularity

"But Linux is more popular." So? Windows is even more popular. Go use that. Usually the popularity argument really means things like "It's easier to find support", or it ties into the program-availability issue above. But there are plenty of places and people providing commercial support for BSD, and the community on the various mailing lists and newsgroups is large and knowledgeable.

Usability

"BSD is hard to use, more advanced, more complicated, less user-friendly." It's *different* from Windows or Linux, sure, but so what? In many ways, we find it a *lot* more logical and easy to figure out. That's where all the effort about consistency and cleanliness pays off. BSD isn't any harder, it just requires thinking a little differently.

Elitism

"BSD users are a bunch of elitist self-centered rude snobs." Sorry, BSD people are just ordinary women and men like anybody else. They like Unix, BSD and Beastie.

Why should you use BSD?

Should I use BSD or Linux? This is a very difficult question to answer. Here are some guidelines:

1. It Just Works. When you install the system, a set group of pieces are there.

2. "If it ain't broke, don't fix it": If you already use an open source OS, and you are happy with it, there is probably no good reason to change.

3. BSD systems can have notably higher performance than Linux. But this is not across the board. In many cases, there is little or no difference in performance. In some cases, Linux may perform better than BSD.

4. In general, BSD systems have a better reputation for reliability as a result of the more mature code base.

5. The BSD license with less restrictions may be more attractive than the GPL.

6. BSD can execute most Linux binaries, while Linux can not execute BSD binaries.

Important websites and links:
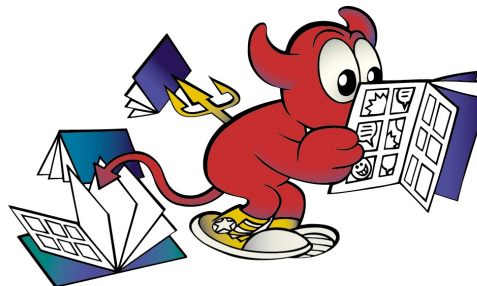
All BSDs have excellent documentation available, see:

http://www.FreeBSD.org/

http://www.NetBSD.org/

http://www.OpenBSD.org/

http://www.DragonFlyBSD.org/

http://www.over-yonder.net/~fullermd/rants/bsd4linux/bsd4linux1.php

http://www.FreeBSD.org/doc/en_US.ISO8859-1/articles/linux-comparison/

http://sites.inka.de/mips/unix/bsdlinux.html

http://www.FreeBSD.org/doc/en_US.ISO8859-1/articles/explaining-bsd/index.html

What is Linux?

Linux also means several things. It's a kernel, originally written by Linus Torvalds. Linux is also the term for a family of operating systems. We don't care about people discussing how "Linux isn't really an operating system, it's just a kernel", or "It should be called 'GNU/Linux'", or similar topics. When we say "Linux" we mean Red Hat, Slackware, Debian, gentoo and hundreds of other distros, based around a Linux kernel with substantially similar userlands, mostly based on various GNU tools.

What is BSD?

BSD stands for "Berkeley Software Distribution". Originally, it was developed at the University of California, Berkeley (CSRG). All new code was released under the BSD license, which basically translates to "Do whatever the hell you want with the code, just give us credit for writing it". Later the 386BSD project started and made it run on the Intel i386 platform. As the 386BSD project wound down, two main groups formed: FreeBSD and NetBSD. In 1995 OpenBSD split from NetBSD and in 2003 DragonFly BSD split from FreeBSD.

When we say "BSD" we mean the general BSD flavor and approach to systems. We have 4 main free BSDs:

- **FreeBSD** though it's expanded to a number of other platforms its primary goal is to be as robust and efficient as possible both for server and desktop roles Intel and AMD platforms.
- **NetBSD** is aimed at running on as many platforms as possible. Its goal is to be the most portable OS on the planet.
- **OpenBSD** is focused primarily on security and related topics. Tight integration of security and auditability and cryptography and related issues are its primary goal.
- **DragonFly BSD** focus lies on providing a SMP-capable infrastructure that is easy to understand, maintain and develop for.

All of those goals are fungible. *Every* BSD cares about security. *Every* BSD cares about performance. Massive chunks of code are shared among the group. Many developers work on more than one system. The more general the point, the more likely it is to be the same across the group. Philosophically, all the BSDs are very similar, in contrast to the Linux methodology. And anyway, the philosophy is what this flyer is primarily about.

The base system

The concept of the "base system" maybe is difficult to understand because the whole idea just doesn't even exist in the Linux world.

Linux, from the start, was just a kernel and a kernel by itself isn't very useful. You need all the userland utilities to make it work. Linux has always been a conglomerate; a kernel from here, a ls from there, vim, perl, gzip, tar, and a bundle of others. Linux has never had any sort of separation between what is the "base system" and what is "addon utilities". The *entire system* is "addon utilities".

By contrast, BSD has *always* had a centralized development model. BSD doesn't use GNU ls or GNU libc, it uses BSDs ls and BSDs libc, which are direct descendents of the ls and libc that were in the CSRG-distributed BSD releases. The system as a whole is one piece, not a bunch of little pieces. Now, X isn't a part of a FreeBSD base system. It's an addon package and therefor X apps like xterm, KDE, GNOME, Mozilla etc. obviously can't be part of the base system either. The primary difference is where they're developed. NetBSD and OpenBSD *do* have an X implementation in the base, because of the way they integrate it with their console driver. They both use heavily modified, very custom versions, so it's not feasible to keep it as a separate package.

The entire base system is developed together. To be sure, there're parts of the base system like sendmail, BIND, ssh and such, which are in fact individual packages which are developed elsewhere. There are even some GNU packages like GCC etc., which will be immediately recognizable to any Linux user. But these are treated specially, in that versions are imported into the tree, then molded to fit the rest of the system. In fact, many of them used to be BSD-only; BIND and sendmail were originally developed as part of BSD, and only later became available separately.

The primary reason an externally-maintained package becomes imported into and tracked in the base system is that it is, in some way, basic enough to the functioning of the system that it's easiest to have it there by default. GCC and the binutils are part of the base system because... well, they're required to *build* the base system. GNOME, KDE etc. are *not* part of the base system, and likely never will be, because they're not required to get the system up and deployed, won't be used on the majority of systems etc.

The base just needs to provide the tools to get the system running, and allow you to update it and install other packages. Then you install what you need, for the specific role this system is intended for. Each BSD defines its own base, NetBSD and OpenBSD for example have much broader criteria for determining what to include in the base system (Apache is in OpenBSD base) than FreeBSD.

Ports/pkgsrc versus rpm/packages

Then, there's the second category; those programs which are add-on packages. In the BSD world, this is usually called the "ports system" (FreeBSD and OpenBSD) or "pkgsrc" (NetBSD and DragonFly BSD).

Traditionally, when you wanted to run a package on your system, the first thing you had to do was to compile it. And often before you could compile it, you'd have to fiddle with it. Your system would require different header files. Sometimes, you'd even need to rewrite parts of it from scratch, because of basic assumption that didn't hold on your system. Or, in other words, you'd have to "port" it to your OS, and/or to your specific system. The basic intent of the ports system is to do all that "porting" stuff for you. That it also automates building and installing, and provides packaging services (for things like 'uninstall') isn't as well reflected in the name.

Most Linux users install binary packages, and most BSD users install by building from source. Partly, that's a result of the tools; the ports system is designed around the concept of building from source, with the ability to make and install binary packages being something of an afterthought, while Linux packaging like rpm and apt and such are designed around the concept of installing a binary package, with building from source as an afterthought. Now, there are advantages to pre-compiled binaries; mostly time (as in much less), and usually it'll take a lot less space to install a pre-compiled package, than it would to compile the package. There are also advantages to building from source, like avoiding all sorts of library versioning ugliness. You can install binary packages on Linux or BSD; you can build from source on both.

Things break, of course. Maybe a dependency will have its main site disappear, so nobody can download the source file. Maybe a new version of some third program will break a program, which will keep other things that depend on it from working. It doesn't solve *all* the problems. But the incidence of "I want A, which requires B, which I can't find" is a lot less than it is with such essentially decentralized systems like rpms turn out to be.