Initialization and boot scripts

One of the biggest historical differences between BSD and some other Unices is its init system, `init(8)`. It is the first application being run by the kernel, and reads its configuration from the file `/etc/rc.conf` which, amongst other things, handles the (de)activation of services (usually by `servicename=[YES|NO]`, e.g., `sshd=YES`). There are no so-called *runlevels* as in Linux and other systems which use Unix System V-style initialization.

In the `/etc/rc.d` directory, there are the rc(8)-scripts, equivalent to Linux' `/etc/init.d` directory. Out of these, init(8) builds a list of dependencies, as well as the order in which the services will be started.

Starting/stopping is as it is in Linux, just with the different path. So, `/etc/rc.d/sshd start` for examples starts your sshd(8). Analogous to `start`, there are options like `stop`, `status`, and `restart`.

If a service is not activated in your rc.conf(5) file, you have to prepend the action by a `one`. So, if you for example don't have `sshd=YES` in the file but want to start sshd(8) once, you can do that with `/etc/rc.d/sshd onestart`.

Networking

Network configuration in NetBSD is done through the use of `rc.conf(5)` as well, with the keyword `ifconfig_IFNAME`, where IFNAME is the interface name. `defaultroute` sets your default gateway to its value, and `hostname` specifies the hostname. Additional IP addresses can be specified with `ifaliases_IFNAME`. To get a thorough description, see the `rc.conf(5)` manpage.

To configure and manipulate your network manually, you will mostly use `ifconfig(8)`, `route(8)`, and `netstat(1)`. The former is very similar to its Linux counterpart, but integrates some of the functionality of the iwconfig command as well.

Hard disks / block devices

Unix was originally programmed for platforms different from Intel's x86, so it traditionally had a *disk label* instead of a Master Boot Record (MBR). NetBSD still adhers to these roots, and some platforms (e.g., SPARC) still use this disk label natively. Because of this tradition, several differences when addressing block devices and partitions exist.

Partitioning

Though often you want to use `gpt(8)` for larger disks, you still often encounter `disklabel(8)` being used on a disk.

You edit the disk label of a hard disk with the tool `disklabel(8)`. If the architecture you're on supports booting from it (as is the case with, e.g., SPARC), you can directly write the disk label to the disk; if it doesn't, as is the case with for example x86, you need to write the disk label into a native partition of your platform, in most cases a MBR.

In this case, you create an MBR partition first (called a *slice*), and inside the slice, you create a disk label. Such a disk label contains up to 16 partitions, being enumerated with *a*–*p*. Partition *c* and, possibly, *d* have special meanings: *c* is used for addressing the slice the disk label lies in or, in case of a non-embedded disk label, the whole disk; in the former case, *d* is used to address the whole disk.

Raw devices

NetBSD distinguishes the abstraction layer you use the devices with, depending on the device nodes you use. Each block device (in the following, say `/dev/sd0`) is either raw as a character device (with DMA) addressable, then specified with an *r* in front (e.g., `/dev/rsd0`), or as a block device, then with its usual name.

Thus, you should always use the block device for work like mounting; for operations not taking the file system into account, for example creating file systems with `newfs(8)`, you should use the raw device.

NetBSD for Newcomers

Nowadays there are only few people who start using Unix by using a shell. Most people install their Unix-like operating system, with GNU/Linux being the most widespread one, with a graphical installer.

This introductionary text is made for people who already use GNU/Linux and know about its internals and the principles of the system. It provides a few tips and tricks on how to switch to NetBSD, and what you have to consider when switching.

Note: in the following text, *Linux* actually means *GNU/Linux*, except where just the kernel is meant.

Package management

When installing additional software on NetBSD, you have two possibilities: either you install pre-compiled (i.e., binary) packages from a repository, or you compile them yourself. With both methods the same result can be achieved; pre-compiled packages are just built in a standard way pkgsrc would do it for you when no package options are modified.

Additionally to this flyer, a short look at the according chapters in The NetBSD Guide is advisable. There is also *The pkgsrc Guide*, explaining the details of pkgsrc package management.

pkgsrc

The system for managing source packages in NetBSD is called `pkgsrc`. It originated from FreeBSD's *ports*, but forked a long time ago, and nowadays is a (nearly) operating system- and archtecture-independent package management system, usable on many different Unices – even as an unprivileged (i.e., non-root) user.

Although both paths can be customized, usually, the pkgsrc sources are kept in `/usr/pkgsrc`, while packages are installed into `/usr/pkg`. In the installation path, a structure close to the one in the `/` root directory is present: there is `bin/`, `sbin/`, `etc/`, etc. pkgsrc usually doesn't install any files outside its installation path; thus, you have to configure all additional software in `/usr/pkg/etc`. If you installed a service, you can either copy its rc(8)-script from `/usr/pkg/share/examples/rc.d` to `/etc/rc.d`, or modify your rc(8)-configuration to have this path as its `rc_directory` as well.

pkgsrc is separated into categories like *chat*, *mail*, *www*, etc., most of which are self-explanatory. To install a package, enter its directory (e.g., `misc/bsdstats`), then issue `make install`. This will compile and install the package.

To search for packages based on their names or other properties, you can use the `pkgtools/pkgfind` package. Each package also contains a `DESCR` file, containing a longer description of it.

After installing a package, you may clean up files that were used during the compilation of it; issue `make clean clean-depends` for that. If you ever forget this (or just want to clean all files which belonged to package compile processes) you can also run this commands from the pkgsrc root directory or from a category directory, cleaning all subdirectories from your current point. (Alternatively, use the `pkgtools/pkgclean` package.)

Binary packages

Actual package management (as pkgsrc is actually the name of the underlying framework and accompanying source files) consists of several tools, most notably `pkg_add(1)` and `pkg_delete(1)` for installation and deinstallation of packages, `pkg_info(1)` for listing installed packages and compilation, installation and dependency information thereof, and `pkg_admin(1)` for housekeeping of the package database (defaultly kept in `/var/db/pkg`), checking for security vulnerabilities in packages (using the separately downloadable `pkg-vulnerabilities` file), and integrity checking of installed packages' files.

mk.conf(5)

pkgsrc (as well as the base system, by the way) is configured by the `mk.conf(5)` file. This file does not exist by default, so you have to create it. Besides options generic to pkgsrc itself, per-package compile options may also be set; to see which options a package supports, issue `make show-options` from within a package's directory.

Other tools

There are several other tools simplifying package management, most of which can be found in the `pkgtools` category; browse around there to find out more useful ones than the ones already mentioned.

Monitoring

Monitoring tools

NetBSD has several very powerful monitoring tools in its base distribution, including but not limited to the following:

- `sysstat(1)` – shows an overview of all possible data. You can also rotate between some of them, or only show specific subsystems
- `iostat(8)` – shows statistics about I/O of hard disks, terminals, and CPU
- `netstat(1)` – shows networking-related information like listening ports, routing tables
- `vmstat(1)` – shows information about virtual memory usage
- `fstat(1)` – shows information about open files
- `sockstat(1)` – shows information about open sockets

Though you can call all tools without parameters or options, their manpages show their usage options, used to customize the monitoring.

Periodic mails

The standard crontab(5) file from NetBSD has three entries: `daily`, `weekly`, and `monthly`, all of which are being called as often as their name says. These scripts send mails to the superuser, and log about memory and disk usage, system warnings, security issues, login accounting, etc. You can configure them over their files `daily.conf(5)`, `weekly.conf(5)`, and `monthly.conf(5)` respectively, to tell which checks they shall or shall not do, and what to show or not.

It is advised you always take a look at these mails, whether or not you have other monitoring software running: these mails might contain important information about for example hard disk failures in RAID systems, critical security holes, etc.