A FreeBSD 11 Desktop How-to

Contents

FreeBSD is a fast, secure, modern Unix-like operating system with a fantastic community, great documentation, and powerful technologies like ZFS and LLVM. It's my operating system of choice for everything from my i7-2600k desktop to my home router to my ARM plug computer jukebox. Though famed for its uptime in the datacenter the same OS is just as suited to desktop or laptop computing with a little work.

Why use FreeBSD? Maybe I'm just getting old, but it's nice to use an operating system that didn't spawn a billion-dollar anti-malware industry through frequent security failings, where you can choose the interface you like and reasonably expect it to stay that way instead of being forced into the design fad du jour, where you don't have to argue about the init system being replaced two times in the same decade, and whose key organizations don't collectively kowtow to Microsoft when convenient. I've used many operating systems and have yet to find one more consistent and cohesive yet as well-supported as the BSD family, and FreeBSD is the one with the biggest community and most available drivers for things like graphics cards. "FreeBSD on the server, Linux on the desktop" is an oft-seen sentiment among some FreeBSD enthusiasts, and it's sort of understandable considering the conservative out-of-the-box FreeBSD installation. Despite that, FreeBSD is just a few settings away from being an easy, powerful Desktop OS rivaling Linux, complete with the same software ecosystem available through the Ports collection.

Unlike Linux where everything including the kernel is a package, FreeBSD is developed in a single source tree and released on a set schedule – twice a year – as a complete operating system on top of which you can install third-party software. The Release Engineering page tracks the release history and schedule. Two major branches see releases in parallel, and major branches tend to live for two years (four minor

versions) after their x.0 release. FreeBSD 11.x is currently the newest release branch with 10.x in maintenance mode and major development happening on 12.x.

This guide attempts to show users with various hardware configurations the best way to configure a usable modern workstation running FreeBSD based on my own experience with Emi, my FreeBSD workstation. There are projects such as TrueOS and GhostBSD that can give you a good out-of-the-box desktop FreeBSD experience, but I find them disconnected from the underlying operating systems because they dump you into KDE or XFCE and attempt to hide as much of FreeBSD as possible behind graphical configuration. That's not a bad thing, and I'm glad those projects exist, but this guide gets there in the other direction. You will install FreeBSD, learn how it works, and configure it into a great desktop. Compare this guide to something like Linux From Scratch or a Gentoo installation and you'll see what a breeze this really is. Let's do it!

New Installations

The Getting FreeBSD page has links to ISOs for the six tier 1 architectures. This guide focuses on amd64/i386 PCs though is broadly applicable to them all. The ISOs are available in `bootonly`, CD, DVD, and `memstick`. I usually grab the nearest >1GB USB drive and dd the newest `memstick` image to it. The larger DVD images are available complete with Ports distribution files for those doing fully-offline installations.

Boot the chosen installation media via whatever means your computer can (EFI boot menu, BIOS setting, fallback) and get ready to install. FreeBSD will boot to bsdinstall and offer to Install, load an interactive rescue shell, or just boot normally off the installation disk. Choose `Install`, choose the keyboard mapping appropriate for your computer, and enter a hostname for your machine.

When asked to choose system components I recommend selecting all of them. `doc` is useful to have locally if your Internet connection isn't working, and system `src` is needed for some Ports to build and install.

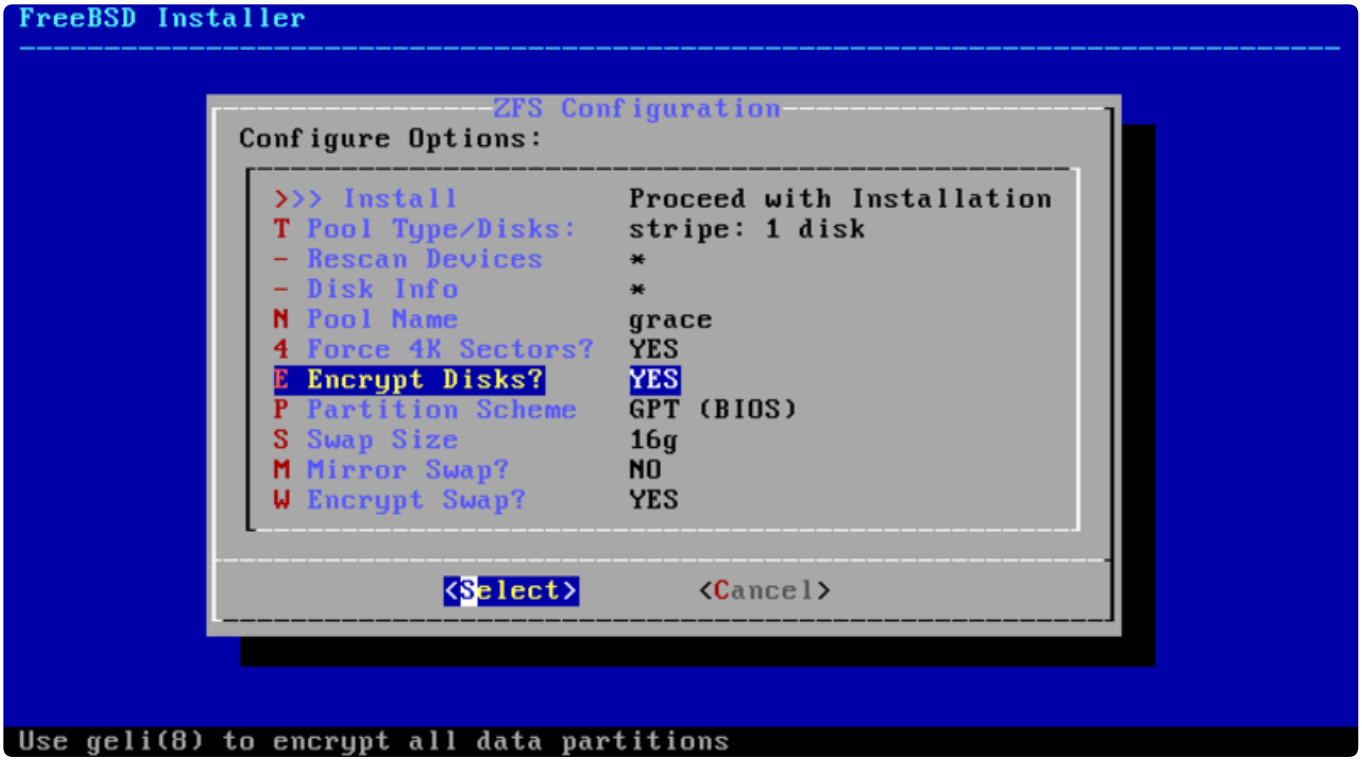
Disks

On the next screen, Partitioning, select either the `Auto (ZFS)` option or the `Auto (UFS)` option. UFS is the traditional Unix file system and is usable on any machine. It

is fragile in the case of power loss or crashes unless journaled. ZFS, on the other hand, is both a volume (pool) manager and a great filesystem. I strongly recommend ZFS for modern computers due to its resilience and rich feature set that makes it very practical for desktop use. It checksums your data constantly to ensure integrity and prevent silent corruption on-disk, and its copy-on-write model never overwrites blocks, eliminating the RAID-5 write hole. It supports snapshots, allowing you to snapshot a filesystem at an arbitrary point in time and roll back to it at will, like Apple's Time Machine, and snapshots can be sent seamlessly across a network for incremental backups. It supports SSD cache devices to speed up reads and writes of pools backed by magnetic hard drives. It can deduplicate files, reducing the on-disk space for files that are significantly identical at the cost of lots of RAM. Achieving these features makes ZFS very memory-hungry. Plan to have 1GiB of physical memory for every 1TB of space in a zpool and much more if deduplication is used. ZFS can be tuned for tighter memory limitations, but very limited systems should use UFS instead. Most modern machines should not need to tune ZFS at all. It will use the memory available to it but also respond to memory pressure when other processes need RAM.

Assuming you choose ZFS, set up your zpool. The pool can be a single disk (still called `stripe` but just striped with itself), a mirror, or any combination of disks in RAID-Z. Name the pool something. I usually name boot pools after the hostname of the machine and then data pools by function.

The other ZFS options are dictated by your hardware.



If you have an Advanced Format hard disk (any made in the last few years) or an SSD you should force 4k sectors.

If your computer has a recent Intel or AMD CPU supporting AES instruction set there is very little downside to encrypting your pool. I recommend it for any pool that doesn't need to automatically mount at boot, i.e. I use encryption on my workstation but not on my FreeBSD router. Encrypting multiple physical devices with the same key will only require the passphrase once. You will be prompted for the passphrase while the kernel has loaded and is detecting hardware. On my system the passphrase prompt usually gets buried under my USB devices as the kernel enumerates them, so if you find yourself stuck there at boot hit a few keys and Enter to make the passphrase prompt reappear.

Use GUID Partition Table (GPT) if your computer uses EFI. PCs with BIOS most likely need to use a legacy MBR. GPT is a requirement to use disks over 2TB in size because MBR can address a maximum of 2^{32} x 512 bytes, just larger than 2TB.

The amount of swap space you use if any is dictated by the amount of memory in your computer and the loads you plan to place on it. Conventional wisdom says to use a swap size double the amount of physical memory in the machine, but I find that to apply less and less when you get up into double-digit gigabytes of RAM. My computer has 16GiB or physical memory and 8GiB of swap space defined. On some systems I don't touch swap at all, but I recommend having at least some. You can enlarge it later.

Finish

Once your disks are set up the installer will copy files and prompt you for the base configuration of things such root password, time zone, and network options. It will ask you to add at least one non-root user. When creating your personal user account be sure to invite it to the `wheel` and `operator` user groups. `wheel` membership is necessary to gain root privileges for administration tasks, and you will assign `operator` device permissions later in this guide. When prompted you can remove the install disk and reboot into your new system!

Upgrading

Are you running a previous version already? The upgrade process is covered in full in Chapter 24 of the Handbook, but assuming you are running the stock `GENERIC` kernel the process is very simple using `freebsd-update`. Check my upgrade notes section for specific version instructions.

First, fetch the new system using `freebsd-update install` by specifying the `-r` argument. Without `-r` it will just fetch security and errata updates for your current minor version. You can upgrade to a new minor version in your current major or to a new major version entirely. If you're upgrading to a new major version, go to the `.0` release first. Don't upgrade from, say, `8.4-RELEASE` to `9.3-RELEASE`, but to `9.0-RELEASE` first and then to `9.3`.

```
1. freebsd-update upgrade -r 11.0-RELEASE
```

`upgrade` is interactive and will ask you to confirm the system components it thinks you have installed. Once it fetches the updated system files you can begin the installation process.

```
1. freebsd-update install
```

This will install the new kernel but not any non-kernel OS components like userland executables. Reboot via `shutdown -r now`, `reboot`, or a swift kick to the power switch. When the system comes up, log in as root and install the new userland by re-running the install command.

```
1. freebsd-update install
```

At this point your OS itself is ready to go but your packages need to be updated to run on the new major version via your preferred method, such as `pkg-static upgrade -f` for binary packages or `portupgrade, or portmaster -af`` if you prefer to build ports. It is alternatively possible to maintain ABI compatibility with an older major version of FreeBSD by installing a compatibility library package such as misc/compat9x, but you shouldn't unless you need it for a particular binary that isn't available as source to build for the new version.

Once that's done you can run `freebsd-update install` one last time to chean out the shared libraries from the previous version. Reboot once more to your final updated system.

First Boot

Log in as root with the password you configured in the installer. Congratulations! You are now a FreeBSD user!

The FreeBSD base system is a fully-featured operating system but as you can see does not contain a graphical environment or any third-party software like your typical Linux distribution. Before installing any of that you should configure your new system to be a better desktop. FreeBSD's roots are in academia and the datacenter, so its default configuration is very conservative. The desktop or laptop computer you are most likely using is vastly more powerful than some of the configurations that will run FreeBSD, so there is room to grow without being unreasonable.

You will need a text editor to edit configuration files. The base system ships with vi (not vim!) but for most users I would recommend ee. It's part of the base system and is a simple but fully-featured editor like nano from the Linux world. If you aren't happy with `ee` there are plenty of great editors you can install from Ports, like editors/vim, but let's continue.

If you aren't entirely comfortable with editing config files you can re-access the graphical configuration screen from the installer by running bsdconfig as root, but I don't think you'd be reading this page if that's the case :)

UTF-8

The `LANG=xx_YY.ZZZZ` environment variable sets the system locale to language code `xx`, country code `YY`, and character encoding `ZZZZ`. Language and country code affect default application language, number formatting, date and time formatting, string collation, currency settings, and more.

By enabling a locale using UTF-8 character encoding, the system can understand and display each of the 1112064 characters in the Unicode character set, instead of just US ASCII as is default with `LANG=C`.

Check `locale -a | grep UTF-8` for a list of every available UTF-8 locale on your computer. As an American anglophone, I use `en_US.UTF-8`.

Edit the login class capability database in /etc/login.conf to add a default character set and locale. Login shells will inherit the environment variables defined here in the `default` class or in a narrower class if it matches one.

/etc/login.conf

```
1. --- login.conf.default  2012-01-02
   17:08:05.804291477 -0500
2. +++ login.conf  2012-01-02 17:08:16.996213774
   -0500
3. @@ -44,7 +44,9 @@
4.         \:pseudoterminals=unlimited:\
5.         \:priority=0:\
6.         \:ignoretime@:\
7. -       \:umask=022:
8. +       \:umask=022:\
9. +       \:charset=UTF-8:\
10. +      \:lang=en_US.UTF-8:
```

Rebuild the login database with `cap_mkdb /etc/login.conf` after making changes.

You may have to specify the new locale elsewhere (like `/etc/profile`) for non login shell uses such as GDM and other login managers.

/etc/profile

```
1. LANG=en_US.UTF-8;   export LANG
2. CHARSET=UTF-8;  export CHARSET
```

You can read more in the Using Localization chapter of the Handbook. Check your work by running locale on your next login.

```
locale
```

```
1. LANG=en_US.UTF-8
```

```
2. LC_CTYPE="en_US.UTF-8"
3. LC_COLLATE="en_US.UTF-8"
4. LC_TIME="en_US.UTF-8"
5. LC_NUMERIC="en_US.UTF-8"
6. LC_MONETARY="en_US.UTF-8"
7. LC_MESSAGES="en_US.UTF-8"
8. LC_ALL=en_US.UTF-8
```

Tuning and drivers

Change a few sysctl variables to enhance the experience of FreeBSD on the desktop, including expanding the amount of shared memory, tuning the process scheduler for desktop use, and increasing the limit of simultaneously-open files to something sensible.

/etc/sysctl.conf

```
1. # Enhance shared memory X11 interface
2. kern.ipc.shmmax=67108864
3. kern.ipc.shmall=32768
4.
5. # Enhance desktop responsiveness under high CPU
   use (200/224)
6. kern.sched.preempt_thresh=224
7.
8. # Bump up maximum number of open files
9. kern.maxfiles=200000
10.
11. # Disable PC Speaker
12. hw.syscons.bell=0
13.
14. # Shared memory for Chromium
15. kern.ipc.shm_allow_removed=1
```

Some knobs can only be set at boot by the loader by setting them in `/boot/loader.conf`. This is also where we define kernel modules to load at boot.

/boot/loader.conf

```
# Devil worship in loader logo
loader_logo="beastie"

# Boot-time kernel tuning
kern.ipc.shmseg=1024
kern.ipc.shmmni=1024
kern.maxproc=100000

# Load MMC/SD card-reader support
mmc_load="YES"
mmcsd_load="YES"
sdhci_load="YES"

# Access ATAPI devices through the CAM subsystem
atapicam_load="YES"

# Filesystems in Userspace
fuse_load="YES"

# Intel Core thermal sensors
coretemp_load="YES"

# AMD K8, K10, K11 thermal sensors
amdtemp_load="YES"

# In-memory filesystems
tmpfs_load="YES"

# Asynchronous I/O
aio_load="YES"

# Handle Unicode on removable media
libiconv_load="YES"
libmchain_load="YES"
cd9660_iconv_load="YES"
msdosfs_iconv_load="YES"
```

Finally, enable everything else.

/etc/rc.conf

```
moused_enable="YES"

# powerd: hiadaptive speed while on AC power,
adaptive while on battery power
powerd_enable="YES"
powerd_flags="-a hiadaptive -b adaptive"

# Enable BlueTooth
hcsecd_enable="YES"
sdpd_enable="YES"

# Synchronize system time
ntpd_enable="YES"
# Let ntpd make time jumps larger than 1000sec
ntpd_flags="-g"
```

Enable remote access via SSH if you plan to use it. Otherwise, there's no need to expose your system.

/etc/rc.conf

```
# Remote logins
sshd_enable="YES"
```

Mounts

The procfs and fdescfs virtual filesystems are not a default part of BSD but are frequently required for compatibility with programs and environments written with Linux in mind, such as GNOME/MATE and KDE. The FreeBSD equivalent is sysctl, but you can mount `/proc` too if you plan to use software requiring it.

Some special filesystems like `fdescfs` must be mounted `late` on ZFS-rooted systems since the location of their mountpoint won't exist until late in the boot process.

/etc/fstab

```
1. proc    /proc   procfs  rw      0       0
2. fdesc   /dev/fd fdescfs rw,auto,late    0       0
```

Toggle the sysctl that lets users mount disks.

/etc/sysctl.conf

```
1. # Allow users to mount disks
2. vfs.usermount=1
```

If you neglected to add your personal user account to the `wheel` and `operator` groups at creation, now is a good time to do so. `wheel` membership lets you use su to become root, and `operator` membership is required for device permissions in this configuration. In this example my user is `okeeblow`. Substitute it for yours.

```
1. pw usermod okeeblow -G wheel
2. pw usermod okeeblow -G operator
```

Device Permissions

Relax default permissions on the device filesystem to allow normal users access to a variety of disks and input/output devices.

Permissions for devices existing at boot time are set in devfs.conf. Each line defines a full device path and octal permission value.

/etc/devfs.conf

```
1. # Allow all users to access optical media
2. perm    /dev/acd0       0666
3. perm    /dev/acd1       0666
4. perm    /dev/cd0        0666
5. perm    /dev/cd1        0666
6.
7. # Allow all USB Devices to be mounted
```

```
perm    /dev/da0        0666
perm    /dev/da1        0666
perm    /dev/da2        0666
perm    /dev/da3        0666
perm    /dev/da4        0666
perm    /dev/da5        0666

# Misc other devices
perm    /dev/pass0      0666
perm    /dev/xpt0       0666
perm    /dev/uscanner0  0666
perm    /dev/video0     0666
perm    /dev/tuner0     0666
perm    /dev/dvb/adapter0/demux0    0666
perm    /dev/dvb/adapter0/dvr       0666
perm    /dev/dvb/adapter0/frontend0 0666
```

For devices that may be connected post-boot, we add an entry to a devfs.rules ruleset. Rulesets must have a unique name and number, and their rules are composed of a path or quoted path glob and octal permission value.

/etc/devfs.rules

```
[devfsrules_common=7]
add path 'ad[0-9]\*'            mode 666
add path 'ada[0-9]\*'   mode 666
add path 'da[0-9]\*'            mode 666
add path 'acd[0-9]\*'   mode 666
add path 'cd[0-9]\*'            mode 666
add path 'mmcsd[0-9]\*' mode 666
add path 'pass[0-9]\*'  mode 666
add path 'xpt[0-9]\*'   mode 666
add path 'ugen[0-9]\*'  mode 666
add path 'usbctl'               mode 666
add path 'usb/\*'               mode 666
add path 'lpt[0-9]\*'   mode 666
add path 'ulpt[0-9]\*'  mode 666
add path 'unlpt[0-9]\*' mode 666
```

```
16. add path 'fd[0-9]\*'            mode 666
17. add path 'uscan[0-9]\*' mode 666
18. add path 'video[0-9]\*' mode 666
19. add path 'tuner[0-9]*'  mode 666
20. add path 'dvb/\*'               mode 666
21. add path 'cx88*' mode 0660
22. add path 'cx23885*' mode 0660 # CX23885-family
    stream configuration device
23. add path 'iicdev*' mode 0660
24. add path 'uvisor[0-9]*' mode 0660
```

Enable our new ruleset.

/etc/rc.conf

```
1. devfs_system_ruleset="devfsrules_common"
```

Sound

Enable sound support at boot in loader.conf, and load it immediately with `kldload snd_driver`.

```
1. echo 'snd_driver_load="YES"' >> /boot/loader.conf
```

Then, `cat /dev/sndstat` to see your available devices.

cat /dev/sndstat

```
1. FreeBSD Audio Driver (newpcm: 64bit
   2009061500/amd64)
2. Installed devices:
3. pcm0: <HDA NVidia (Unknown) PCM #0 DisplayPort>
   (play)
4. pcm1: <HDA NVidia (Unknown) PCM #0 DisplayPort>
   (play)
5. pcm2: <HDA NVidia (Unknown) PCM #0 DisplayPort>
   (play)
```

```
6. pcm3: <HDA NVidia (Unknown) PCM #0 DisplayPort>
   (play)
7. pcm4: <HDA Realtek ALC892 PCM #0 Analog>
   (play/rec)
8. pcm5: <HDA Realtek ALC892 PCM #1 Analog>
   (play/rec)
9. pcm6: <HDA Realtek ALC892 PCM #2 Digital> (play)
10. pcm7: <HDA Realtek ALC892 PCM #3 Digital> (play)
11. pcm8: <USB audio> (play) default
12. pcm9: <USB audio> (rec)
```

The `hw.snd.default_unit` sysctl variable controls the default audio output. I want to use the S/PDIF output of my onboard Realtek audio, `pcm6`, so I set `hw.snd.default_unit` to `6`.

Enabling the `hw.snd.default_auto` boolean will automatically assign `hw.snd.default_unit` to newly-attached devices.

/etc/sysctl.conf

```
1. # S/PDIF out on my MSI board
2. hw.snd.default_unit=6
3.
4. # Don't automatically use new sound devices
5. hw.snd.default_auto=0
```

Networking

If you didn't enable networking during the install process now is a good time to do so. Here as an example is my computer, `emi`, a desktop with a wired network. I have a Realtek interface, `re0`. The name of your interface may vary based on the driver it uses. Most drivers are built into the `GENERIC` kernel, so your interface should be visible by running `ifconfig`. Common drivers you may see include if_em for Intel PRO interfaces, if_re for Realtek interfaces, and if_en for Midway interfaces. Read more about network configuration in the Handbook to learn about other possible configurations.

Wired

You can use DHCP and SLAAC auto-discovery on most home networks:

/etc/rc.conf

```
1. hostname="emi.aloe.cooltrainer.org"
2.
3. # Enable DHCP for re0 and don't let dhclient block
4. background_dhclient="YES"
5. ifconfig_re0="DHCP"
6. ifconfig_re0_ipv6="inet6 accept_rtadv"
```

You can bring up the interface and get a DHCP address immediately by issuing `ifconfig re0 up` and `dhclient re0`, again substituting the name of your own interface.

Alternatively, you can supply static network addresses for your computer and default router:

/etc/rc.conf

```
1. hostname="emi.aloe.cooltrainer.org"
2.
3. ifconfig_re0="inet 172.16.0.40 netmask 255.240.0.0
   broadcast 172.31.255.255"
4. defaultrouter="172.16.0.1"
5.
6. ifconfig_re0_ipv6="inet6 2001:370:10f5:806::40
   prefixlen 64"
7. ipv6_defaultrouter="2001:370:10f5:806::1"
```

Wireless

For WiFi configuration, see the wireless networking section of the Handbook. I sometimes tether my desktop to my Android phone using a run B/G USB interface. It's as simple as defining a new virtual `wlan` interface on `run0`, configuring

wpa_supplicant for the WPA pre-shared key, and specifying the SSID and encryption standard (WPA).

/etc/rc.conf

```
1. wlans_run0="wlan0"
2. ifconfig_wlan0="ssid Doubleshot WPA DHCP"
```

/etc/wpa_supplicant.conf

```
1. network={
2.         ssid="Doubleshot"
3.         psk="pantsupantsu"
4. }
```

IPv6

My configuration examples cover both IPv4 and IPv6 because I have a dual-stacked network. Depending on your network you may prefer to enable and prefer IPv6 like me, enable it but prefer IPv4, or not enable it at all. This can be done in rc.conf:

/etc/rc.conf

```
1. ipv6_activate_all_interfaces="YES"
2. ip6addrctl_policy="ipv6_prefer"
```

/etc/sysctl.conf

```
1. # Accept IPv6 router advertisements
2. net.inet6.ip6.accept_rtadv=1
```

Firewall

You should run a firewall. Windows, OS X, and many Linux distributions ship with a default firewall ruleset. FreeBSD does not, because there is no one-size-fits-all firewall configuration, but it does include one of the best software firewalls in the world, PF, courtesy of the OpenBSD project.

Configuring a firewall can be a very complex topic. There are entire books on the matter. Shown here is the ruleset from my computer. It has rules for a single network interface defined at the top of the file in the `ext_if` macro. Change it to the name of your computer's interface as seen in `ifconfig`. The macros on the next few lines define the TCP and UDP ports on which this ruleset will allow incoming connections. My computer runs an SSH server on the default port, runs www/subsonic on port 443 (HTTPS), and has control ports for net/mosh in the 60000 range. You can open ports for additional services by defining them in those macros. The named services like `ssh` are defined in /etc/services.

/etc/pf.conf

```
# The name of our network interface as seen in `ifconfig`
ext_if="re0"

# Macros to define the set of TCP and UDP ports to open.
# Add additional ports or ranges separated by commas.
# UDP 60000-60010 is mosh control http://mosh.mit.edu/
tcp_services = "{ssh, https}"
udp_services = "{60000:60010}"

# If you block all ICMP requests you will break things like path MTU
# discovery. These macros define allowed ICMP types. The additional
# ICMPv6 types are for neighbor discovery (RFC 4861)
icmp_types = "{echoreq, unreach}"
icmp6_types="{echoreq, unreach, 133, 134, 135, 136, 137}"

# Modulate the initial sequence number of TCP packets.
# Broken operating systems sometimes don't randomize this number,
```

```
# making it guessable.
tcp_state="flags S/SA keep state"
udp_state="keep state"

# Don't send rejections. Just drop.
set block-policy drop

# Exempt the loopback interface to prevent services utilizing the
# local loop from being blocked accidentally.
set skip on lo0

# all incoming traffic on external interface is normalized and fragmented
# packets are reassembled.
scrub in on $ext_if all fragment reassemble

# set a default deny policy.
block in log all

# This is a desktop so be permissive in allowing outgoing connections.
pass out quick modulate state

# Enable antispoofing on the external interface
antispoof for $ext_if inet
antispoof for $ext_if inet6

# block packets that fail a reverse path check. we look up the routing
# table, check to make sure that the outbound is the same as the source
# it came in on. if not, it is probably source address spoofed.
block in from urpf-failed to any

# drop broadcast requests quietly.
```

```
49. block in quick on $ext_if from any to
    255.255.255.255
50.
51. # Allow the services defined in the macros at the
    top of the file
52. pass in on $ext_if inet proto tcp from any to any
    port $tcp_services $tcp_state
53. pass in on $ext_if inet6 proto tcp from any to any
    port $tcp_services $tcp_state
54.
55. pass in on $ext_if inet proto udp from any to any
    port $udp_services $udp_state
56. pass in on $ext_if inet6 proto udp from any to any
    port $udp_services $udp_state
57.
58. # Allow ICMP
59. pass inet proto icmp all icmp-type $icmp_types
    keep state
60. pass inet6 proto icmp6 all icmp6-type $icmp6_types
    keep state
```

Enable the firewall in rc.conf and start it now.

```
1. echo 'pf_enable="YES"' >> /etc/rc.conf
2. service pf start
```

After making changes to your ruleset you can check them for validity and load the new rules with pfctl. It will abort if your ruleset contains an error unlike `service pf restart` which will stop, fail to start due to the error, and leave you locked out of SSH. Ask me how I know this will happen.

```
1. pfctl -f /etc/pf.conf
```

Installing Software

FreeBSD is historically famous for its Ports Collection, a directory skeleton of `Makefile`s and patches describing how to programmatically build packages of third-party software for your computer. Each port contains the metadata for that piece of software including the filename of the source archive, sha256 hash of the files, what other software dependencies it requires, what compile-time options are available, what files it installs, and any patches necessary to work around non-portable code or fix issues that can't be upstreamed to the projects themselves. Every port has a maintainer whose job it is to keep the port up to date and respond to issues if they arise with newer versions of the operating system. For example, check out the ports I maintain! When you `make` a port in the Ports Collection it downloads the upstream source archive, patches it, configures it, builds a customized binary package, and uses FreeBSD's underlying binary package manager to install it.

Third-party software installed through the package manager ends up in `/usr/local` where it mirrors the hierarchy of `/usr`. It might seem confusing to have them in two places, but it gives a fairly clean separation of the base system from the packages. For example, `/usr/local/bin` is where you will find `firefox` after installing www/firefox, but `/usr/bin` is where you will find `ee` or `sed`. As you have experienced, configuration for the operating system is done in `/etc`. Configuration files for your ports will usually be in `/usr/local/etc`. Check the hier manual for the full layout.

As of FreeBSD 10 there is a new binary package manager, known as `pkgng` or just pkg. It replaces the old suite of `pkg_` tools such as `pkg_add` and `pkg_delete` and contains many advanced features that have been missing from FreeBSD for years. Compared to the old package manager, pkgng supports safe upgrades (meaning it saves a copy of the previous-version package to roll back in case of failure), multiple repositories, package staging before install, a more modern binary package format, a more robust sqlite-based package registration database, and most importantly remote binary package upgrades. Thanks to pkgng it is now possible to add multiple remote binary package repositories and get updates from them without relying on building Ports at all.

The FreeBSD project provide binary packages built from the Ports tree using the default options. For example, installing a binary package of Firefox with `pkg install firefox` or `pkg install www/firefox` is equivalent to doing `portsnap fetch extract && cd /usr/local/www/firefox && make install`. I will stick to the convention of `pkg install` on this page but the

names will be identical if you prefer to build customized packages from Ports. It is possible to mix and match binary packages with your own custom packages built from Ports using pkg's `lock` and `unlock`, but that gets more advanced and more annoying than is appropriate for this guide. I build my ports on one of my computers using poudriere then use that computer as a binary package repository for the other FreeBSD computers on my network. A new user with just one computer should stick to `pkg install` from the default package server until they are more familiar with the OS.

In the past I recommended building Ports over the default binary package server due to some strange default port options and the lack of package coverage for certain software like KDE. These days however I recommend using binary packages by default unless you find a particular port option you absolutely must have. Even then, consider using poudriere to create a local package server instead of building and installing ports the traditional way. It's more work initially to set up but saves you from the hassle of updating shared libraries while trying to use your computer. The old fashioned way will replace `libfoo.so.4` with `libfoo.so.5` and only then start rebuilding ports that depend on it. Meanwhile your currently-installed software will be unusable due to the missing `libfoo.so.4`. Packages avoid this issue and alert you upfront to any build failures. Probably 99% of users these days will be fine with the packages provided by the FreeBSD project.

No matter the route you choose, you should begin by updating your pkg repository or your Ports tree. `pkg update` will fetch an updated index from every `PACKAGESITE` defined in that environment variable or in `pkg.conf`. For Ports, `portsnap fetch extract` will retrieve a new full ports tree. You should read pkg for the list of commands supported by `pkg` and read ports for the list of available `make` targets in Ports, but in general you will use `pkg install <portname>` (or `pkg install <category>/<portname>`) or `cd /usr/ports/<category>/<portname> && make install` to install software.

If you choose to build your own packages from Ports it will be because you want to customize options, so use `make config-recursive` in any port's directory to set these port options in advance so they don't continually interrupt the build. Read over the options as they are presented, but generally don't toggle an option if you don't know what it does. The defaults are default for a reason! Once you've chosen all the options, run `make config-recursive` again, since it's likely for a dependency enabled the first time to have options of its own. When no further port

options are displayed run `make install` to compile and install your custom package.

The first things I usually install on a new system are sysutils/tmux and shells/zsh because I prefer it to the venerable default tcsh. You might be more comfortable with shells/bash. Once installed, changing your user's default shell is as simple as `chsh -s /usr/local/bin/zsh`.

Build Settings

Some third-party software options can only be set at compile time. Here are a few you should consider for desktop use if you decide to build your own customized packages.

The Qt toolkit has some options that can be set via any combination of the following knobs. If you change the `QT4_OPTIONS` after Qt is installed you will need to rebuild devel/qt4-corelib and x11-toolkits/qt4-gui.

- QGtkStyle is a selectable theme engine that lets Qt applications integrate more closely with GTK+ environments and can be enabled with the `QGTKSTYLE` Qt4 option. You should enable this option if you plan to use a GTK+-based desktop environment like MATE or XFCE. Once built you can select the GTK+ visual style in `qt4-qtconfig`.
- CUPS is the standard printing engine on Free Unix-like systems. Support for it in Qt can be enabled with the `CUPS` Qt4 option.
- Network Audio System is a network transparent client/server audio transport system and can be enabled in Qt applications by setting the `NAS` Qt4 option.

```
1. echo "QT4_OPTIONS=      CUPS QGTKSTYLE NAS" >>
   /etc/make.conf
```

Staying Up To Date

Before updating your Ports tree or pkg catalogue on an already-installed computer I suggest checking out UPDATING. This file – also found at `/usr/ports/UPDATING` if you have an extracted Ports tree – tracks breaking changes in packages and the steps necessary to to fix them. It will tell you when configuration file syntax has

changed, how to fix dependencies when ports split or merge, when default versions of packages change, and more.

You can update your already-extracted Ports tree to the newest revision with `portsnap fetch update`. There are two common tools for programmatically upgrading your ports, ports-mgmt/portupgrade and ports-mgmt/portmaster. I prefer portupgrade because portmaster stubbornly aborts the entire upgrade on any error, but you might like it. The common portmaster usage is `portmaster -a` to upgrade installed ports. I usually update with `portupgrade -rac`, with `-r -a` to recursively update all installed ports and `-c` to preemptively show new port options so they won't interrupt the build.

If you are using binary packages you can update the package repository catalog with `pkg update` and upgrade your installed packages with `pkg upgrade`.

The Handbook has a more detailed chapter on using Ports that includes upgrading and other tasks.

Going Graphical

I'm sure you're eager to get out of the text-only console. To do so you need to install the X.org distribution of the X Window System from x11/xorg with `pkg install xorg`.

X.org by default uses FreeBSD's devd for hardware detection and D-Bus for interprocess communication. HAL used to be the default hardware detection mechanism but is in the process of deprecation, so you may not need it unless you plan to use a desktop environment that still depends on it. Check the dependencies for sysutils/hal for a list of software still requiring it.

```
/etc/rc.conf

1. hald_enable="YES"
2. dbus_enable="YES"
```

X.org on its own is just the display server. It needs stuff to display. You can install applications, but you also need a window manager to handle displaying and interacting with those applications in the way you'd expect from any modern OS. By

default X comes with the Tab Window Manager which is probably not software you would enjoy using in the 21st century.

Install fonts

X.org doesn't include many attractive typefaces by default. Luckily, there are plenty available in Ports.

Here are a few I use, including many from non-Roman languages for better Unicode coverage. On a new system I usually install something like:

```
1. pkg install chinese/arphicttf chinese/font-std
   hebrew/culmus hebrew/elmar-fonts japanese/font-ipa
   japanese/font-ipa-uigothic japanese/font-ipaex
   japanese/font-kochi japanese/font-migmix
   japanese/font-migu japanese/font-mona-ipa
   japanese/font-motoya-al japanese/font-mplus-ipa
   japanese/font-sazanami japanese/font-shinonome
   japanese/font-takao japanese/font-ume
   japanese/font-vlgothic x11-fonts/hanazono-fonts-
   ttf japanese/font-mikachan korean/aleefonts-ttf
   korean/nanumfonts-ttf korean/unfonts-core x11-
   fonts/anonymous-pro x11-fonts/artwiz-aleczapka
   x11-fonts/dejavu x11-fonts/inconsolata-ttf x11-
   fonts/terminus-font x11-fonts/cantarell-fonts x11-
   fonts/droid-fonts-ttf x11-fonts/doulos x11-
   fonts/ubuntu-font x11-fonts/isabella x11-
   fonts/ecofont x11-fonts/junicode x11-fonts/khmeros
   x11-fonts/padauk x11-fonts/stix-fonts x11-
   fonts/charis x11-fonts/urwfonts-ttf russian/koi8r-
   ps x11-fonts/geminifonts x11-fonts/cyr-rfx x11-
   fonts/paratype x11-fonts/gentium-plus
```

This includes:

- Arphicttf and font-std for Chinese coverage.
- Culmus and El-Mar for Hebrew language coverage.

- IPA, IPA UIGothic, IPAex, Kochi, MigMix, Migu, Mona, MOTOYA, M+, Sazanami, Shinonome, Takao, UmeFont, VLGothic, Hanazono Mincho, and Mika-chan for Japanese language coverage.
- A-Lee fonts, Nanum, and Un fonts for Korean language coverage.
- Anonymous Pro, artwiz-aleczapka, DejaVu, Inconsolata, and Terminus for terminals and editors.
- Cantarell, Droid, Doulos SIL, and Ubuntu for general Roman alphabet language coverage.
- Telugu for Telugu language.
- Isabella, Ecofont, and Junicode as novelty typefaces.
- KhmerOS for Khmer language coverage.
- Padauk for Myanmar language coverage.
- STIX and Computer Modern for technical and mathematic symbols.
- Charis, URW, KOI8, Geminifonts, CYR-RFX, and ParaType for Cyrillic and Eastern European language coverage.

x11-fonts/webfonts is a special case. It includes the Microsoft Core Fonts for the Web such as Andale and Verdana. If you own a valid Microsoft Windows license you can get Tahoma as well by adding to `/etc/make.conf` and building a custom package.

/etc/make.conf

```
# Enable Tahoma in x11-fonts/webfonts if you have a Windows license
# You show me yours and I'll show you mine ;)
.if ${.CURDIR:M*/x11-fonts/webfonts}
WITH_MSWINDOWS_LICENSE=yes
.endif
```

Now that those fonts are installed you need to tell Xorg about them using a `FontPath` in your `/etc/X11/xorg.conf` so they can be used. Unfortunately the font paths in Ports are not entirely standardized. Most of them end up in `/usr/local/lib/X11/fonts`, some end up in `/usr/local/share/fonts/`, and some in just `/usr/local/share/`. Here's the relevant section from my `xorg.conf`. Yours should look similar.

In the past it was necessary to tell X about your installed fonts using `FontPath` directives in `xorg.conf`, but these days most software will detect them

automatically using fontconfig. You probably won't run into a case where you will need to, but you can manually inform the X server of fonts like this:

```
Section "Files"
    FontPath "/usr/local/lib/X11/fonts/Liberation/"
    FontPath "/usr/local/lib/X11/fonts/anonymous-pro/"
    FontPath "/usr/local/share/font-mona-ipa/"
EndSection
```

You can install other invididual fonts later for a single user by copying them inside the `~/.fonts` directory under your home directory.

You can re-scan the configured `FontPath`s and automatically-detected fonts by running `fc-cache -vf` and can list all installed fonts with `fc-list` to check that they all appear. For example, I have a fairly large local font repository in my `~/.fonts/` directory and have 2313 fonts available on my system as of this writing according to `fc-list | wc -l`.

Starting X

The shell script `.xinitrc` in your home directory is the script that is executed for the lifecycle of your X session. `.xinitrc` is used to run startup applications and then run your window manager. When you finally end your X session your window manager will exit, `.xinitrc` will return, and the X server will stop.

Here's a simple example `.xinitrc`:

~/.xinitrc

```
xscreensaver -no-splash &
xdg-user-dirs-update &
redshift &
compton &
exec wmaker
```

In my example `.xinitrc` I use Window Maker and start x11/xscreensaver, xdg-user-dirs, accessibility/redshift, and the lightweight compositor x11-wm/compton.

Starting your graphical session after logging in on a TTY (the command-line login prompt) is the traditional way and is as easy as running the command `startx` after loggin in as your user. Don't run X as root.

I prefer to use a graphical login manager. Some desktop environments include their own, like KDE's KDM. On my computer I use the environment-agnostic x11/slim. A graphical login manager, besides being nicer to look at, also protects you from leaving a logged-in user session (or even worse, a root session!) on a TTY that could be used by someone walking up to your computer.

You can configure the login manager to start at boot by adding either `kdm_enable="YES"` or `slim_enable="YES"` to `/etc/rc.conf`, but I would recommend not doing that until you are confident everything is set up correctly. It will be more difficult to fix a config file if your computer automatically starts to a broken X server. Test a plain old `startx` first, then test your login manager with `service slim onestart` or `service kdm onestart` before enabling them for every boot.

You can configure the `sessiondir` directive in the SLiM configuration file to define the path to desktop entries installed by your ports, usually `/usr/local/share/xsessions`. Use `exec $1` instead of a particular window manager's executable name in your `.xinitrc` to run the command from SLiM passed in as an argument.

```
/usr/local/etc/slim.conf

1. sessiondir /usr/local/share/xsessions/
```

```
1. echo 'slim_enable="YES"' >> /etc/rc.conf
2. service slim start
```

```
~/.xinitrc

1. exec $1
```

You can switch session (window manager) with the *F1* key on the SLiM login screen.

The X session will use one of the virtual ttys. Once started, you can switch to a different virtual TTY with *CTRL-ATL-F#*. You can switch back to the graphical TTY with *CTRL-ATL-F9*. You may need to create a desktop entry for your window manager of choice if the port maintainer doesn't ship one. Here's an example, again for Window Maker.

```
/usr/local/share/xsessions/wmaker.desktop

[Desktop Entry]
Encoding=UTF-8
Name=Window Maker
Exec=/usr/local/bin/wmaker
Comment=This session logs you into Window Maker
Type=Application
```

Automatic configuration

Versions of X.org since 1.2 (X11R7.2) in 2007 have supported autoconfiguration of display devices. HAL automatically detects input devices, and fontconfig automatically detects fonts, so you may never need `/etc/X11/xorg.conf` at all!

Once you've set up your `.xinitrc` to use your window manager of choice, just give `startx` a try. Hopefully it will work flawlessly. If it does feel free to skip the sections on manual configuration. That should only be necessary if you need to force a specific video driver, toggle a specific option, or just hate yourself and feel like learning one of the worst configuration file formats of all time.

Manual configuration

This section explains how you can manually configure X for systems using an AMD Radeon and the `radeon` driver, an Intel grahpics chip with the `intel` driver, virtualized graphics cards like emulators/virtualbox-ose-additions's `vboxvideo`, or any other generic framebuffer device supported by the default `vesa` driver. If autodetection works you shouldn't have to do this.

As root, run `X -configure`. It will spit out a new X.org configuration file in `/root/xorg.conf.new` based on your detected hardware. Copy this file to `/etc/X11/xorg.conf`, then pull it up in a text editor for a few modifications.

Add an "Extensions" section and enable the Composite extension.

```
1. Section "Extensions"
2.         Option  "Composite" "Enable"
3. EndSection
```

Add one line to the `ServerLayout` section to enable AIGLX.

```
1. Section "ServerLayout"
2.         Identifier     "X.org Configured"
3.         Screen      0  "Screen0" 0 0
4.         InputDevice    "Mouse0" "CorePointer"
5.         InputDevice    "Keyboard0" "CoreKeyboard"
6.         Option         "AIGLX" "true"
```

```
7. EndSection
```

I usually enable the EXA `AccelMethod` and `DRI` by adding their respective lines to the `Device` section. If you have a Radeon card ensure your `Driver` is configured as `radeon`, not `radeonhd`! radeonhd is an older, Novell-sponsored, defunct driver for Radeon HD hardware, but `-configure` likes to pick it by default if it's installed. You should use `radeon` instead. Otherwise `X -configure` should pick the best driver.

```
1. Section "Device"
2.         Option     "AccelMethod" "EXA"
3.         Option     "DRI" "true"
4.         Identifier  "Card0"
5.         Driver      "radeon"
6.         VendorName  "Advanced Micro Devices [AMD]
   nee ATI"
7.         BoardName   "RV770 [Radeon HD 4850]"
8.         BusID       "PCI:1:0:0"
9. EndSection
```

Enable the `freetype`, `bitmap`, and `type1` X font modules by adding them to the `Module` section. According to the manual, "[t]he `extmod`, `dbe`, `dri`, `dri2`, `glx`, and `record` extension modules are loaded automatically if they are present", but I like to go for the explicit configuration and define them anyway.

```
1. Section "Module"
2.     Load           "dbe"
3.     Load           "dri"
4.     Load           "dri2"
5.     Load           "extmod"
6.     Load           "record"
7.     Load           "freetype"
8.     Load           "bitmap"
9.     Load           "type1"
10.    Load           "glx"
11. EndSection
```

Manual Configuration with NVIDIA

Skip this section if you don't use an NVIDIA graphics card or if automatic configuration works for you.

The binary x11/nvidia-driver is the only proprietary software on my system. As much as I'd prefer a free and open solution, I've found that neither Nouveau nor the free Radeon or Intel driver compare to the speed and feature support of Nvidia's official driver.

Install the driver itself, x11/nvidia-settings, and x11/nvidia-xconfig.

To load the nvidia kernel module at boot, enable it in `/boot/loader.conf`.

```
1. echo 'nvidia-modeset_load="YES"' >>
   /boot/loader.conf
```

Run `nvidia-xconfig` to get a base xorg.conf in `/etc/X11/xorg.conf`. Pull it up in your favourite text editor and add the `Module` section to enable the `freetype2`, `glx`, `type1` extensions.

```
   /etc/X11/xorg.conf

1. Section "Module"
2.     Load           "freetype"
3.     Load           "bitmap"
4.     Load           "type1"
5.     Load           "glx"
6. EndSection
```

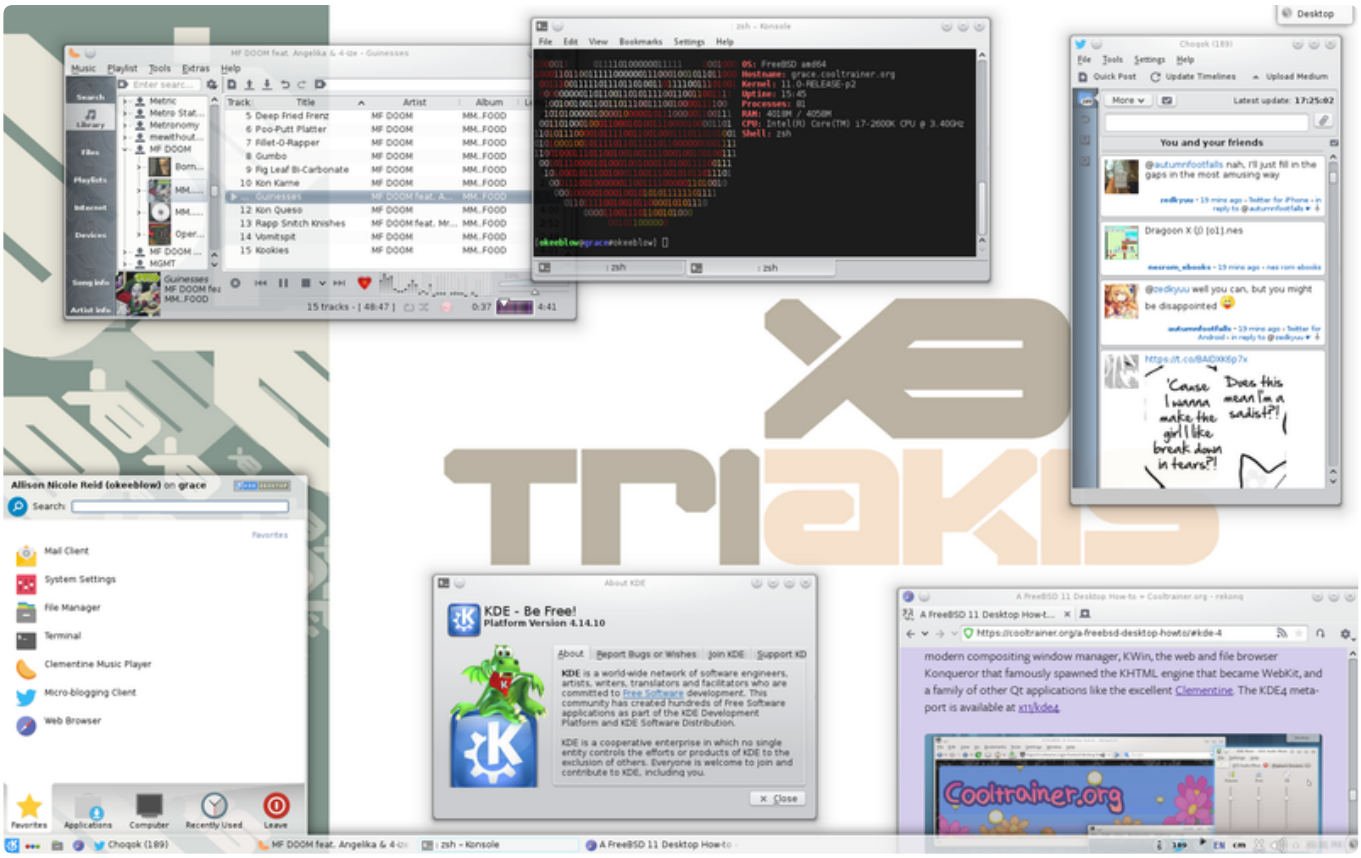
Later on, once you're booted into the graphical environment, you can use `nvidia-settings` to configure TwinView and any other settings.

Desktop Environments

The best part of the X Window System ecosystem is the variety of environments available. You might prefer the configurability of KDE, the sparse cleanliness of MATE, something minimal like Window Maker, or even tiling like i3. This isn't xwinman, but I've included screenshots and descriptions for some of the most popular environments to give you an idea what's out there.

KDE 4

KDE is the largest and most fully-featured Free Software desktop environment, based on the Qt toolkit. Its configuration options are dizzyingly numerous and it has excellent support for modern technologies like high-DPI displays. It has a modern compositing window manager, KWin, the web and file browser Konqueror that famously spawned the KHTML engine that became WebKit, and a family of other Qt applications like the excellent Clementine. The KDE4 meta-port is available at x11/kde4.



You can start KDE with `exec startkde` in your `.xinitrc`, but KDE also includes its own graphical login manager, KDM which you can optionally enable in `rc.conf` and start as a service.

```
1. echo 'kdm4_enable="YES"' >> /etc/rc.conf
```

```
2. service kdm4 start
```

MATE née GNOME 2

GNOME 2 was a venerable desktop environment made famous as the Ubuntu default around 2008, and MATE is the community fork of GNOME 2 after the GNOME team lost their collective minds.

MATE users should configure PolicyKit to allow normal users to mount removable media automatically.

```
/usr/local/etc/PolicyKit/PolicyKit.conf

1. <config version="0.1">
2.         <match
   action="org.freedesktop.hal.storage.mount-
   removable">
3.                 <return result="yes"/>
4.         </match>
5.         <match
   action="org.freedesktop.hal.storage.mount-fixed">
```

```
            <return result="yes"/>
        </match>
        <match user="root">
            <return result="yes"/>
        </match>
        <define_admin_auth group="wheel"/>
</config>
```

MATE doesn't include a GDM alternative, so start it with `exec mate-session` in your `.xinitrc` using either `startx` or SLiM.

Compiz is a popular alternative window manager with MATE and GNOME 2 users. It gives you those fancy wobbly windows, 3d cubes, and all kinds of flashy stuff. You can install Compiz-Fusion from x11-wm/compiz-fusion. Be sure to disable the obsolete and unmaintained window decorator Emerald, a leftover from the Beryl project, when prompted on the port configuration screen. With Emerald disabled, compiz will default to `gtk-window-decorator` and will take on your normal GTK theme appearance but with more transparency and garish animation.

Once installed, open Settings > Preferences > CompizConfig Settings Manager. You'll probably want to enable the following plugins at minimum:

- General: Gnome Compatibility
- Desktop: Desktop Cube, Rotate Cube
- Effects: Animations, Window Decoration, Wobbly Windows
- Window Management: Application Switcher, Move Window, Place Windows, Resize Window

Open the "Run" box with Alt+F2 and execute `compiz-manager`. `compiz-manager` is a script for detecting and using the proper compiz options for your

video hardware. Your screen will flash while Compiz and gtk-window-decorator initialize and replace Marco, MATE's default window manager.

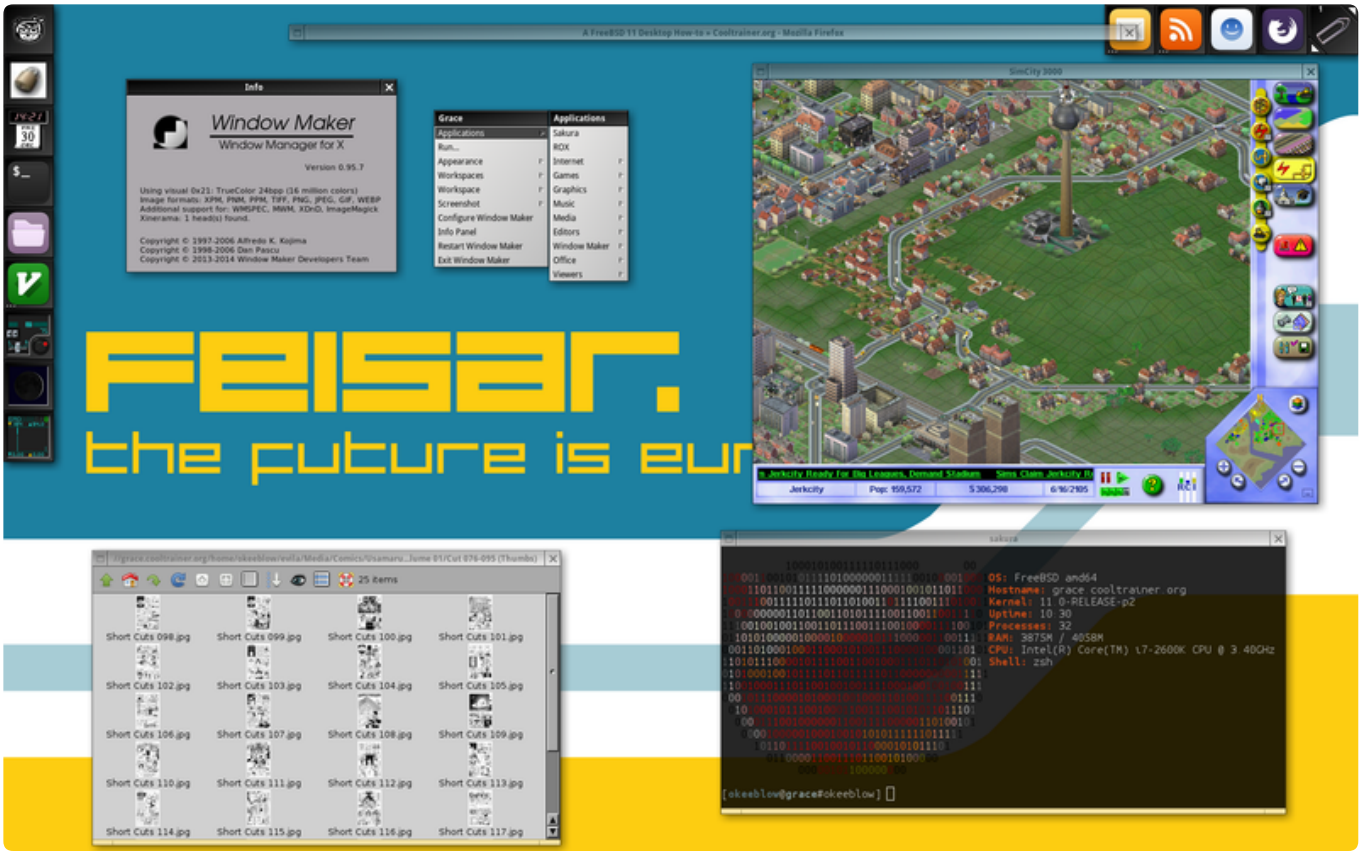
If your windows are missing titlebars, double-check you've enabled "Window Decoration" in `ccsm`. If all seems well, add `compiz-manager` as a new startup application (in Settings > Preferences > Startup Applications), then change MATE's window manager preference in DConf:

```
1. gsettings set org.mate.session.required-components
   windowmanager compiz
```

To switch back, use the same command with the argument `marco`.

Window Maker

It isn't as popular or well-known as the others here, but Window Maker is my favorite and longest-used window manager. It is based on the look and feel of the NEXTSTEP operating system, the OS that became Rhapsody and then Mac OS X and iOS. Unlike the extremely limited one-dimensional Mac OS X dock, Window Maker offers a main dock as well as a "clip" dock that is unique to each virtual desktop. Docks can hold normal launchers and "dockapps", small self-contained dock accessories.

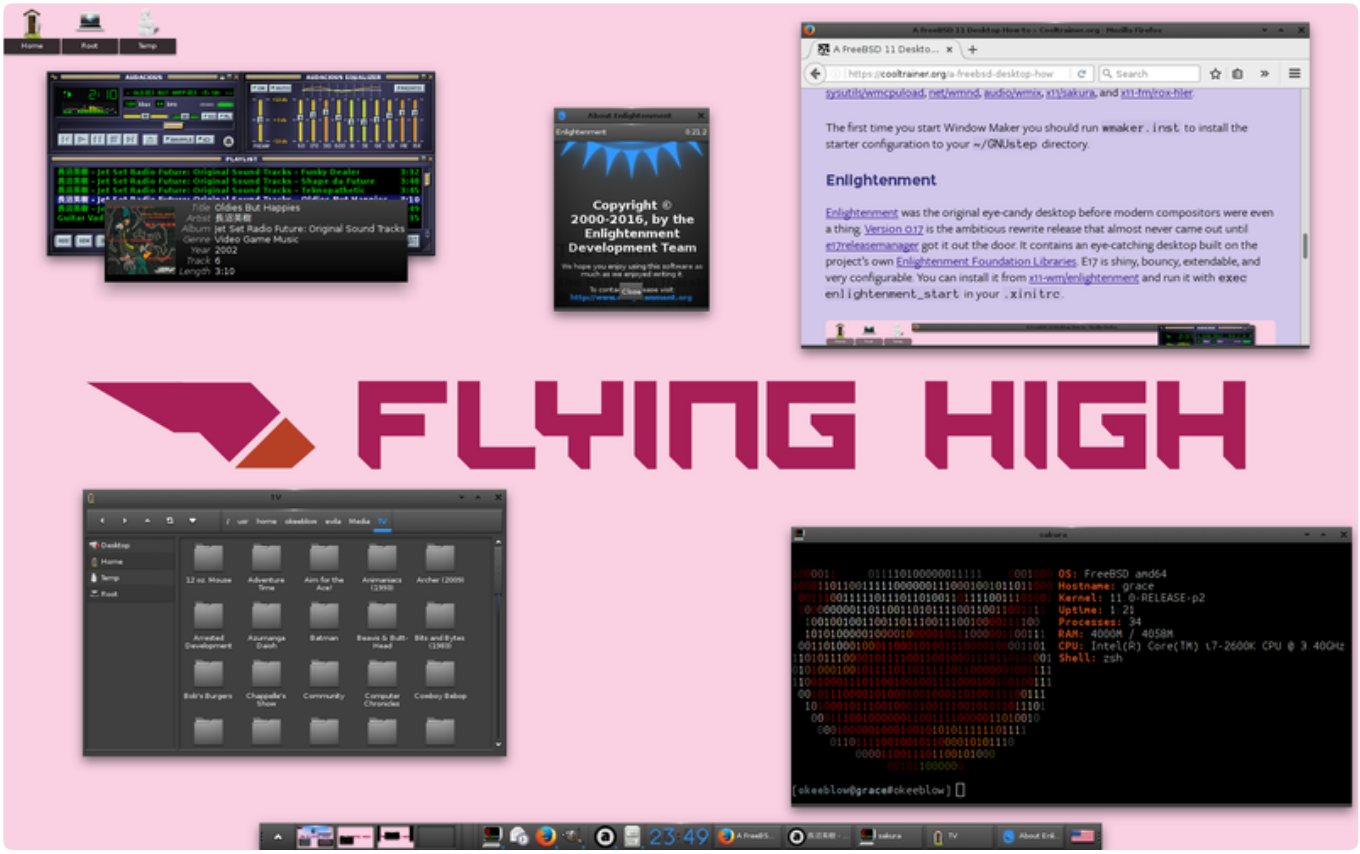


You can install Window Maker from x11-wm/windowmaker and start it with `exec wmaker` in your `.xinitrc`. In addition, I also show x11-clocks/wmclock, sysutils/wmcpuload, net/wmnd, audio/wmix, x11/sakura, and x11-fm/rox-filer.

The first time you start Window Maker you should run `wmaker.inst` to install the starter configuration to your `~/GNUstep` directory.

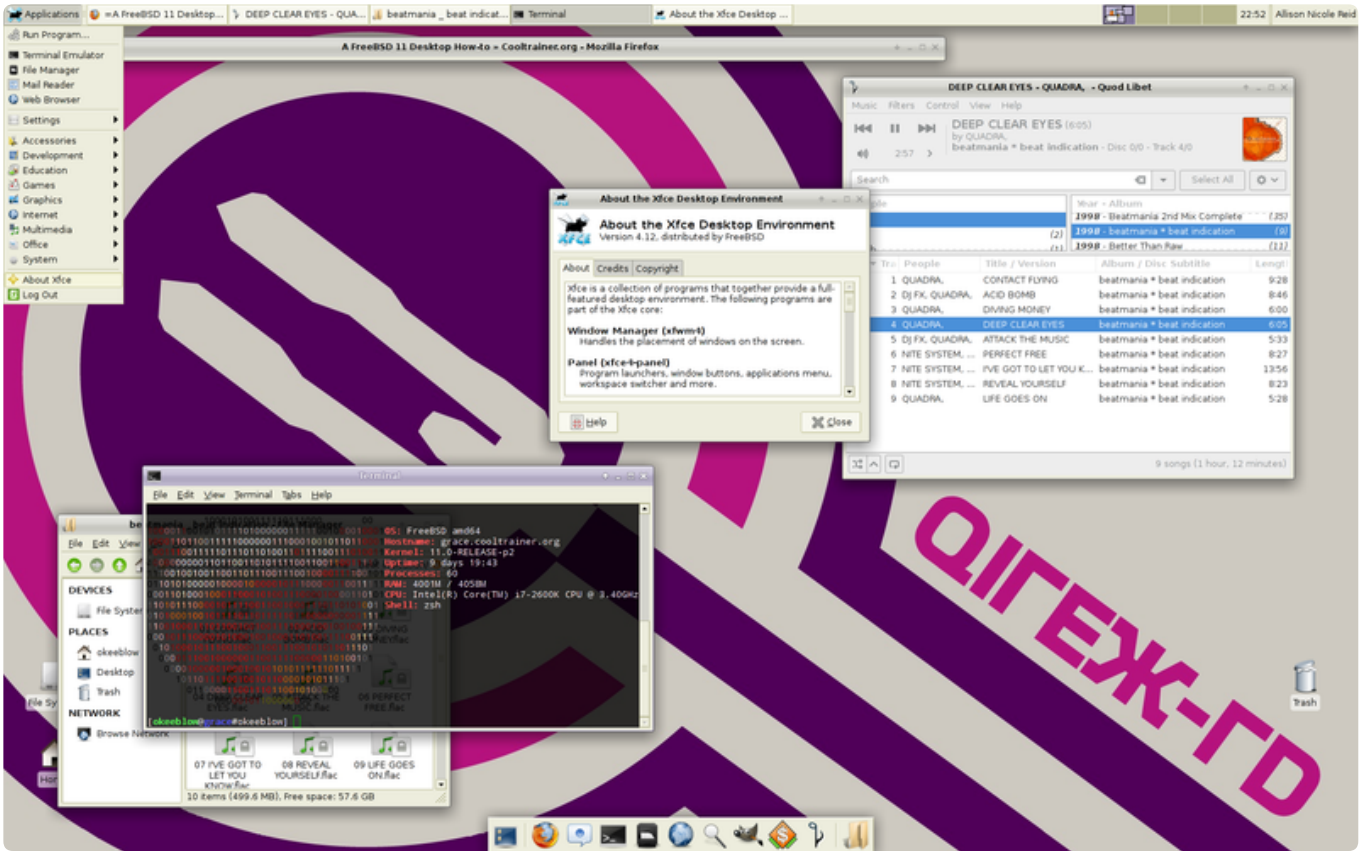
Enlightenment

Enlightenment was the original eye-candy desktop before modern compositors were even a thing. Version 0.17 was the ambitious rewrite release that almost never came out until e17releasemanager got it out the door. Since then it has followed a steady update schedule and is no longer vaporware. It contains an eye-catching desktop built on the project's own Enlightenment Foundation Libraries. E17 is shiny, bouncy, extendable, and very configurable. You can install it from x11-wm/enlightenment and run it with `exec enlightenment_start` in your `.xinitrc`.



XFCE

XFCE descends, like KDE, from the design of the once-proprietary Common Desktop Environment. As of XFCE 4.0, however, the desktop has become more of a GNOME-lite, the "other" GTK+ desktop environment. I don't have much to say about it, but it is a very functional and lightweight desktop with panels, a window manager, a great file manager (Thunar), and some other lightweight applications like a terminal.

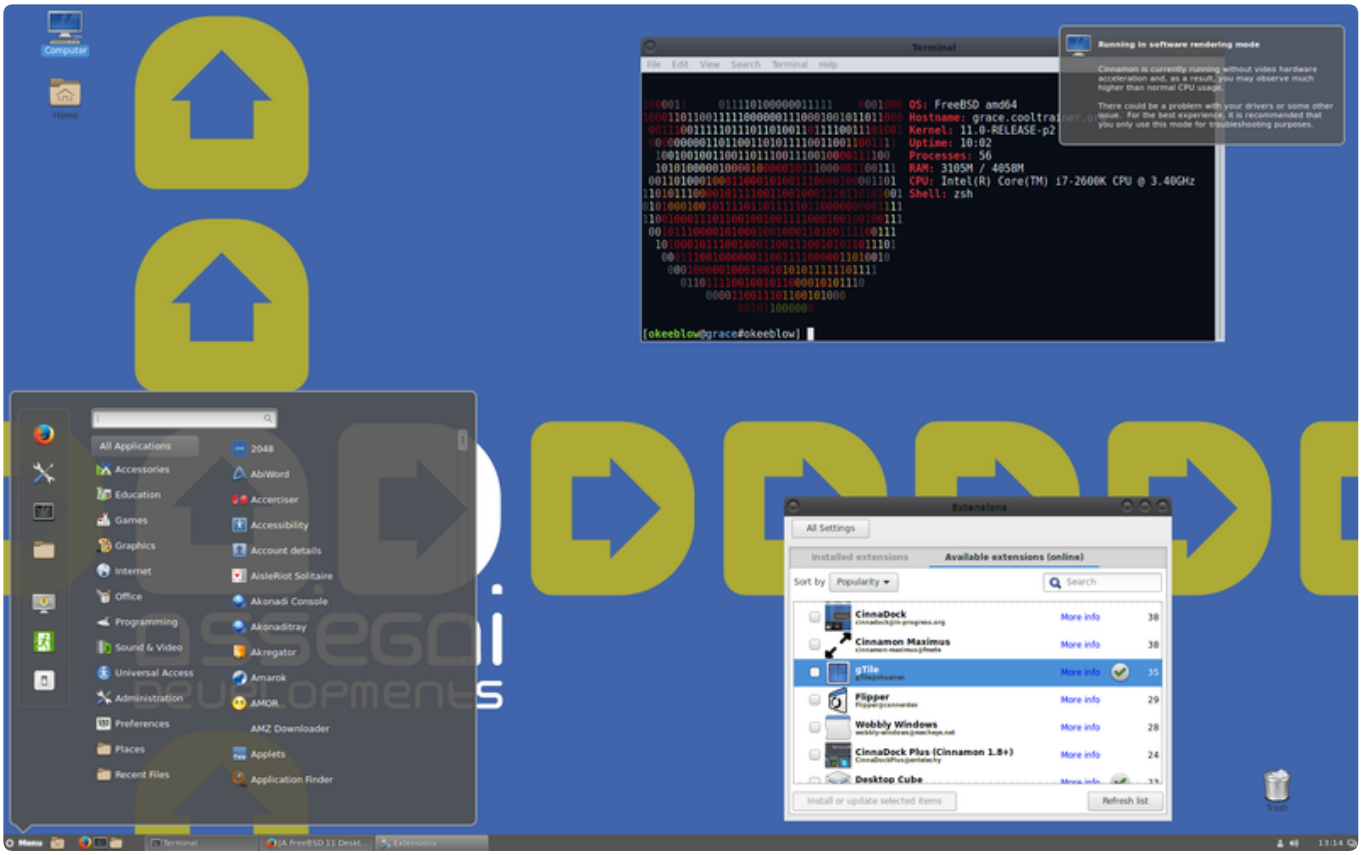


You can start XFCE with `exec startxfce4` in your `.xinitrc`.

Cinnamon

Cinnamon is a GTK 3 desktop environment from the Linux Mint project. It began as a fork of the GNOME 3 Shell into a more traditional panels and menus UI since many were dissatisfied with the drastic redesign of a beloved environment. It's come into its own as a modern DE and offers everything you would expect from GNOME 2 or MATE with a cohesive feel and forks of several of the GNOME core apps.

You can install Cinnamon from x11/cinnamon and start it with `exec cinnamon-session` in your `.xinitrc`. There is also a fallback software-rendering mode that can be started with `exec cinnamon-session-cinnamon2d` instead.



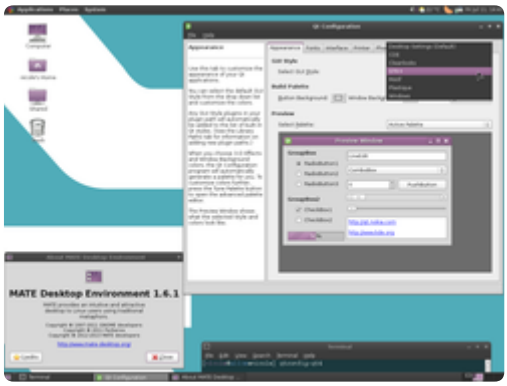
GNOME 3

GNOME 3, available on Linux since 3.0 in the spring of 2011, is finally available in the official FreeBSD tree as of November 2014. The three and a half year delay is thanks to the upstream GNOME project's years-long fight against any operating system that doesn't have a penguin for a mascot. It took several years and a vastly waning userbase, then suddenly they care about portability again. Sure, okay. Either way, it's here and you can install it from x11/gnome3.

We weren't missing much in the delay, since the GNOME team tossed out everything great about their once-ubiquitous DE and turned it into a shiny but unconfigurable iOS imitator where basic features and options are either not available at all, buried inside a settings registry more reminiscent of Windows 98 than BSD, or relegated to extensions that will break with every new minor version thanks to the lack of any stable extensions API and whose very existence are opposed by many of the main project contributors. Take a look at the GNOME Shell Extensions page with me and be amused that you need an extension to use a theme, categorize the Applications menu, remove the otherwise-omnipresent accessibility menu from the status bar, or even power off your computer without knowing about the magic Alt-button toggle.

If you want a great GTK-based desktop environment maintained by a team that doesn't hate you, check out MATE, XFCE, or Cinnamon. All three are excellent. I guess I can thank the GNOME project's self-destruction for getting me back into Window Maker.

Theming



Finding Clearlooks too drab and blue? You can find a world of themes and icons for MATE over on GNOME-Look, for KDE at KDE-Look, for XFCE at XFCE-Look, for E17 at E17-Stuff, and for several lightweight window managers at Box-Look. There are several attractive and usable themes buried among the OS X Aqua clones, Vista Aero clones, and black-as-my-soul darkness-fests that are standard on any theming website.

You can change theme settings for Qt4 applications with `qt4-qtconfig` and for KDE Qt applications (like Clementine) in the KDE System Settings. GTK+ is selectable in many window managers' appearance preferences, and you can also install x11-themes/lxappearance for a light GTK theme switcher.

Extras and Miscellany

Printing

CUPS is the standard for printing on Free Unix-like systems and can be installed from Ports along with any needed filters. Install the CUPS meta-port at print/cups. Install HPLIP in print/hplip for HP printer drivers (and my Brother HL-2170W, for some reason). Install the Foomatic filter collection in print/foomatic-filters and its database and engine in print/foomatic-db and print/foomatic-db-engine, respectively. I find the CUPS-PDF virtual printer in print/cups-pdf very useful as well.

Enable CUPS once installed:

/etc/rc.conf

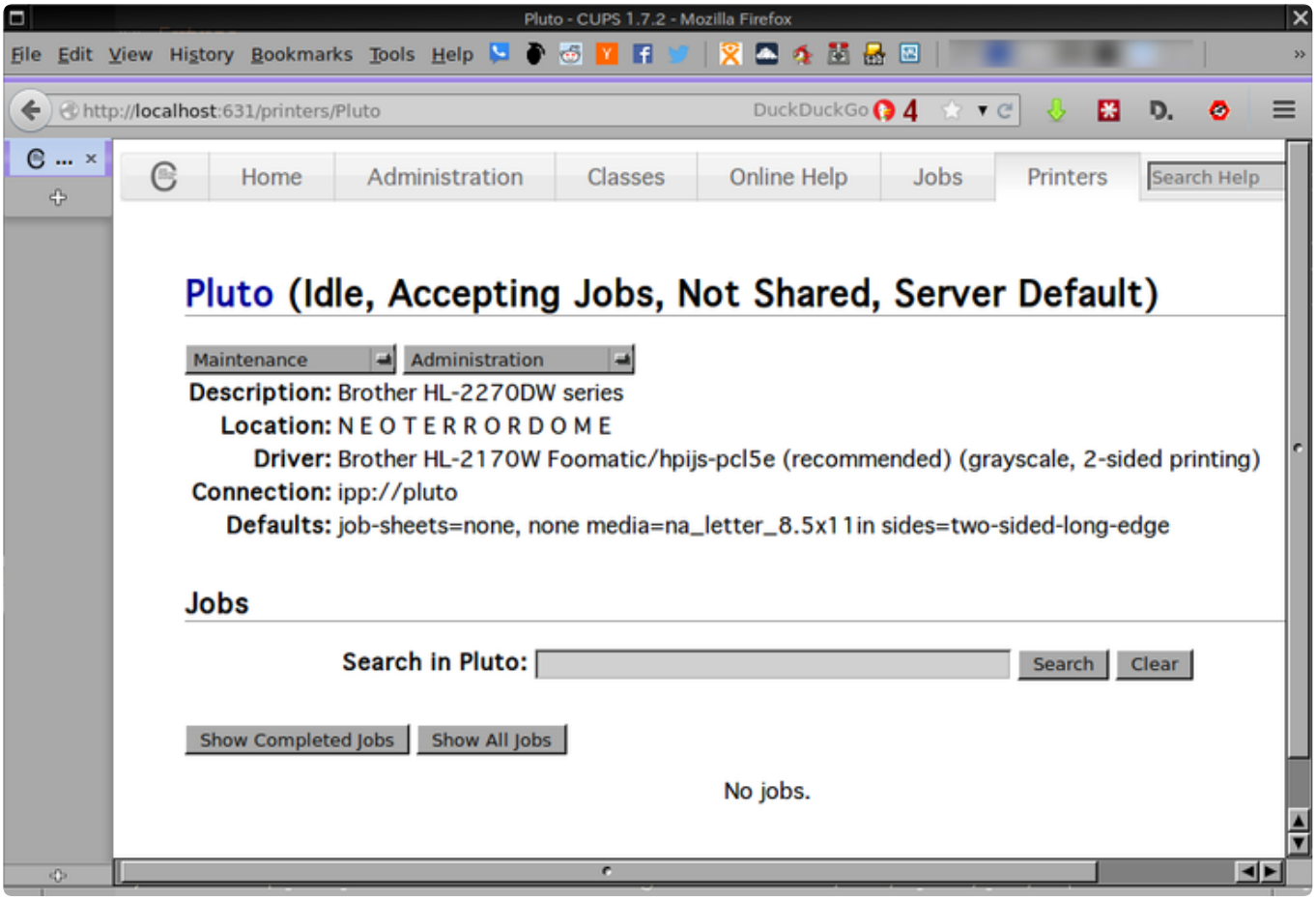
```
# Disable line printer daemon since we have CUPS
lpd_enable="NO"

# Enable CUPS
cupsd_enable="YES"
```

Add local users to the `cups` group if you want them to be able to print.

```
pw usermod root -G cups
pw usermod okeeblow -G cups
```

Start the CUPS service with `service cupsd start` and you should be able to access its web configuration UI at http://localhost:631/ in your web browser. It may prompt you for your root password to write the config files in `/usr/local/etc/cups`. Most full desktop environments include a GUI to control CUPS and add printers, but the web interface is available in any of them.



The web interfaces' built-in documentation can help you configure different models of printer, specifically the Network Protocols Supported by CUPS and Common Network Printer URIs sections.

S.M.R.T.

sysutils/smartmontools installs `smartd` and `smartctl`, a daemon and utility for checking the S.M.A.R.T. status of your local disks.

Enable the sample `smartd.conf`. It contains one directive, `DEVICESCAN`, that causes smartd to scan all attached drives.

```
1. cp /usr/local/etc/smartd.conf.sample
   /usr/local/etc/smartd.conf
2. echo 'smartd_enable="YES"' >> /etc/rc.conf
3. service smartd start
```

You can check the S.M.A.R.T. status of a drive directly with `smartctl` as root. The `-H` flag will show basic pass or fail health status, and the `-a` flag will show everything.

```
smartctl -H /dev/ada0

1. smartctl 6.2 2014-02-18 r3874 [FreeBSD 10.0-
   RELEASE-p2 amd64] (local build)
2. Copyright (C) 2002-13, Bruce Allen, Christian
   Franke, www.smartmontools.org
3.
4. === START OF READ SMART DATA SECTION ===
5. SMART overall-health self-assessment test result:
   PASSED
```

Java

FreeBSD has several available Java providers, including OpenJDK and ~~Sun's~~ Oracle's JDK. I recommend the newest OpenJDK for most people. It's far easier to install than the binary Oracle JRE which requires logging in to a web page, agreeing to the license, and manually downloading the distfile for Ports. At the time of this writing the newest OpenJDK is java/openjdk8. OpenJDK 6 and 7 are available as well if you have software that doesn't run on Java 8.

If you need a Java browser plugin you can install java/icedtea-web once a Java provider is available.

Webcams and DVB

Most USB webcams and many DVB tuners are supported by multimedia/webcamd, and webcamd depends on the userland character device driver in multimedia/cuse4bsd-kmod. Install them, then enable them in `rc.conf` and `loader.conf`.

```
/etc/rc.conf

1. # Webcam daemon
2. webcamd_enable="YES"
```

```
/boot/loader.conf

1. # Userland character device driver for webcams
2. cuse4bsd_load="YES"
```

You can use your camera device with pwcview available in multimedia/pwcview or with Cheese in multimedia/cheese. Cheese provides a nice interface similar to Apple's Photobooth on OS X, but it has a heavy GNOME library dependency some may not want on their systems.

IBus

IBus is a modern IME for Unix-like systems, allowing one to input CJK languages. Install the main IME from textproc/ibus as well as QT application support from textproc/ibus-qt.

You'll need one or more input methods once the IME itself is installed. Ports of interest:

- chinese/ibus-chewing - Chewing engine for IBus
- chinese/ibus-pinyin - The PinYin input method
- japanese/ibus-anthy - Anthy engine for IBus
- japanese/ibus-mozc - Mozc engine for IBus
- japanese/ibus-skk - SKK engine for IBus
- korean/ibus-hangul - Hangul engine for IBus
- textproc/ibus-kmfl - KMFL IMEngine for IBus framework
- textproc/ibus-m17n - The m17n IMEngine for IBus framework
- textproc/ibus-table - Table based IM framework for IBus

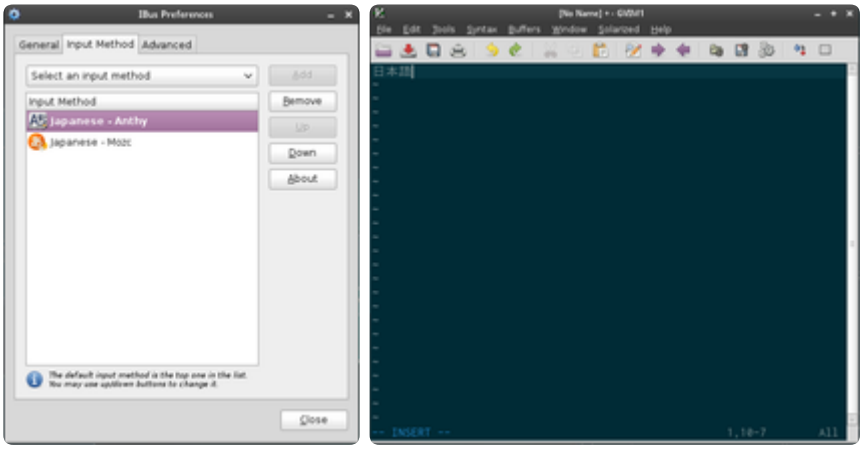
Heavyweight desktop environments like GNOME or KDE will let you configure the input method graphically. In GNOME 2 and MATE, for example, you can open the IBus preferences from the Settings > Preferences menu. KDE/Qt users can enable it as the default IME in `qtconfig-qt4`.

Lightweight window manager users like me can start it in `.xinitrc`:

~/.xinitrc

```
1. export XMODIFIERS="@im=ibus"
2. export GTK_IM_MODULE="ibus"
3. export QT_IM_MODULE="ibus"
4. exec ibus-daemon -d -x &
```

Now run any GTK or QT application, press your keyboard shortcut to switch input methods, and test it out.



Linuxulator

FreeBSD's Linuxulator allows it to run Linux application binaries using system call translation. Desktop users will find it useful for running the handful of proprietary but necessary programs that are available for Linux but not for FreeBSD, such as Adobe Flash Player.

Install the Linux base distribution from Ports. As I write this the default base distribution is emulators/linux_base-c6, based on CentOS 6, replacing the old Fedora 10 based `linux_base-f10`. A newer CentOS 7 emulators/linux_base-c7 is also available and will become the default at some point in the future.

Once your chosen `linux_base` installed, tell your system to load the `linux` kernel module at boot.

```
1. echo 'linux_load="YES"' >> /boot/loader.conf
2. kldload linux
```

Mount the linprocfs virtual filesystems for compatibility with GNOME and other programs requiring them.

```
   /etc/fstab

1. linproc   /compat/linux/proc   linprocfs,auto,late
   rw   0  0
```

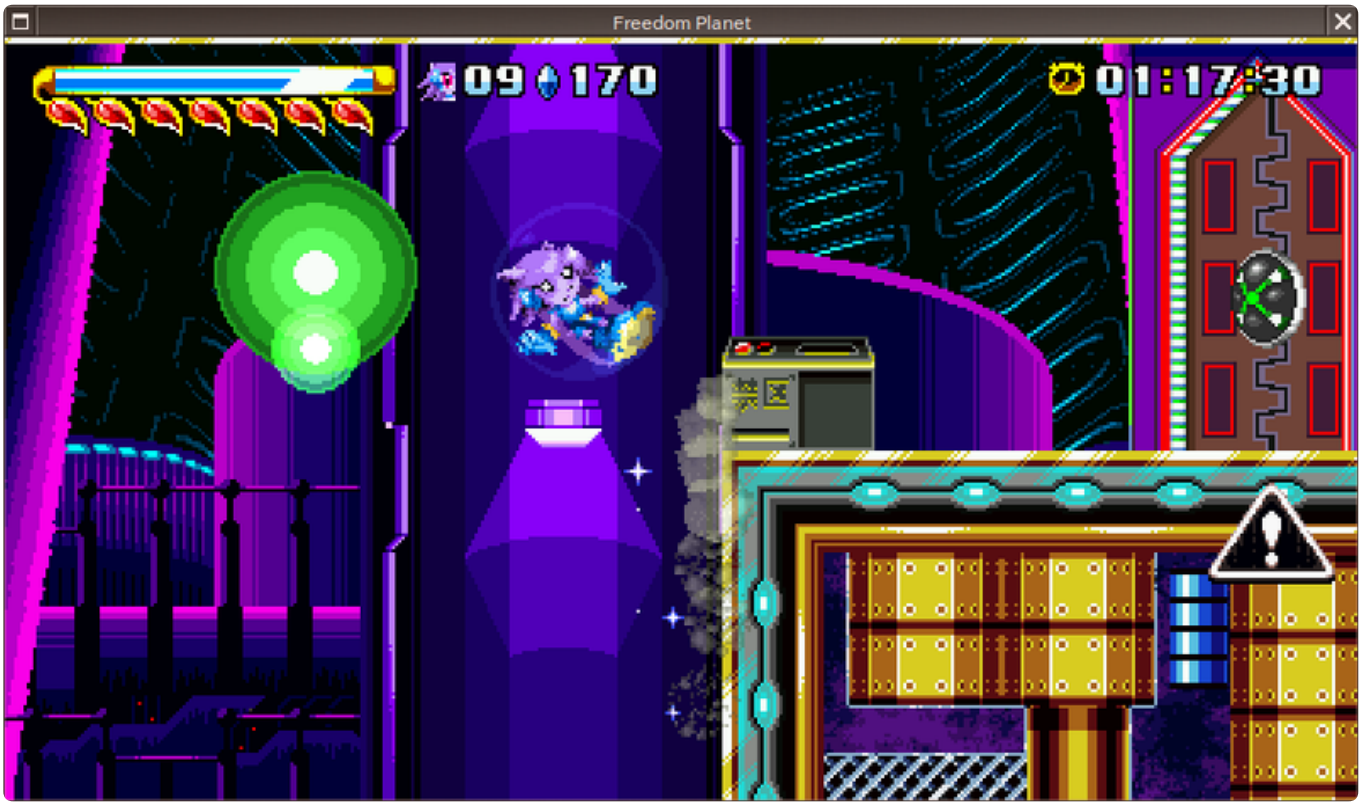
Besides proprietary garbageware like Flash Player I also use Linuxulator along with the Loki compatibility library package to run Loki Software's Linux ports of some of the best PC games ever made, like Simcity 3000 Unlimited.

Wine

Wine is a free implementation of the Win32 API capable of running real Windows applications on Unix-like systems. It's available from emulators/wine emulators/wine-devel, or emulators/wine-staging, containing the latest stable, unstable, and staging versions respectively. The optional emulators/wine-gecko is an `mshtml.dll` replacement that will allow Windows programs to embed web pages using the Mozilla engine. The optional emulators/wine-mono will let Wine run Windows programs written in versions 1.x or 2.0 of the .NET Framework without using the proprietary .NET runtime. You can also install emulators/winetricks, a script containing Wine installation recipes for popular software, and you may find a Wine GUI such as emulators/swine useful for maintaining separate Wine prefixes.

Installing 64-bit Wine on an `amd64` FreeBSD system normally precludes you from running 32-bit Windows software, a.k.a. most of the software you probably care about. As a workaround, `i386` Wine packages are also available. You can install them from emulators/i386-wine, emulators/i386-wine-devel, or emulators/i386-wine-staging.

If you plan to use Wine to run browser plugins, use the `staging`-patched version of wine in emulators/wine-staging or emulators/i386-wine-staging.



Wine is very impressively-compatible these days. I use it to run a lot of games from Humble Store and GOG so I don't have to fire up a Windows computer or VM. Wine is so good it is used to create the Linux versions of many of these titles, such as Freedom Planet, and in that case it's a lot easier to run the same executable in FreeBSD Wine than to try and get the Linux Wine binaries running via Linuxulator.

If every `wine` command fails with `ELF interpreter /libexec/ld-elf.so.1 not found` your 64-bit system is missing the 32-bit libraries necessary for Wine. You'll need to install them. From the releases FTP, grab the `lib32.txz` matching your version of the OS and extract it either as root or with sudo to the root of your filesystem to install. The archive contains a full directory hierarchy so all the files will end up in the right place.

```
1. fetch
   ftp://ftp.freebsd.org/pub/FreeBSD/releases/amd64/1
   1.0-RELEASE/lib32.txz
2. tar xfp lib32.txz -C /
```

Browser Plugins

> *NOTE on NPAPI Deprecation:*
> *Firefox and Chromium are deprecating support for NPAPI plugins, so everything in this section will stop working at that time.*
> *Chromium removed support as of version 32. Firefox 52 is the last version with support and will be available as www/firefox-esr until the release of ESR 60.*

Adobe Flash Player is not released for FreeBSD but is usable on FreeBSD i386 and amd64 through your choice of two wrappers.

The 32-bit Linux version of the Flash plugin can be installed via www/linux-c6-flashplugin24 or www/linux-c7-flashplugin24, executed through Linuxulator, and adapted to 32-bit or 64-bit native browsers with www/nspluginwrapper.

Adobe dropped support for the NPAPI Linux version of Flash with version 11.2 back in 2012. They left development up to Google who deprecated NPAPI and only released a PPAPI version for many years as part of their proprietary Chrome browser bundle, something that isn't even available for FreeBSD. Adobe, in a surprising move, resumed Linux NPAPI Flash plugin development in 2016.

To use nspluginwrapper, make sure `linprocfs` is mounted and execute `nspluginwrapper -v -a -i` as your normal user to locate and enable the Linux Flash plugin. `-a` automatically finds available plugins and `-i` installs them. It's important to remember it makes a copy of the Flash library in your home directory when you do this, so every time you update the version of Flash installed through Ports you need to remove your local cory with `nspluginwrapper -v -a -r` and install the new one with `nspluginwrapper -v -a -i`. Adobe promised security updates for the NPAPI Flash Player through 2017, and if history is any indication you're going to need them.

A newer and arguably better solution is emulators/pipelight, a Wine-based wrapper for Windows browsers plugins. It supports the very newest version of Windows

Flash, freeing us from the limitations of Adobe's Linux development, and also supports Shockwave, Silverlight, Unity, Widevine, and more. It requires a special patched version of Wine, emulators/wine-staging or emulators/i386-wine-staging. Once installed, activate plugins as root.

```
1. pipelight-plugin --update
2. pipelight-plugin --create-mozilla-plugins
3. pipelight-plugin --enable flash
```

Silverlight is available as well and can allow you to watch DRMed NetFlix content via your browser on FreeBSD. I have no experience using it since I don't buy DRMed streaming media subscriptions, but it can be installed the same way.

```
1. pipelight-plugin --enable silverlight5.1
```

Heed www/pipelight's pkg-message warning about Silverlight DRM if you have a ZFS-based system:

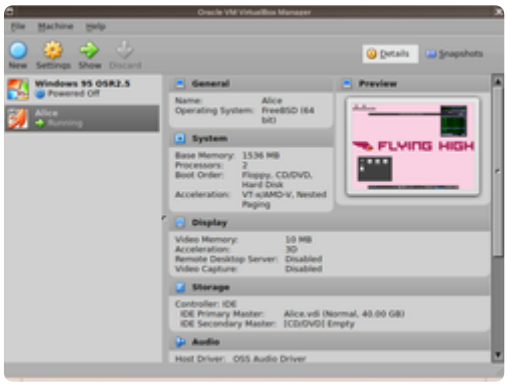
> *For users running with ZFS on root, watching DRM protected content requires extensive xattr support. If you run into issues with DRM failing, you can use the "pipelight-mkufs" command to create a UFS formatted ZVOL mounted on your users ~/.wine-pipelight directory.*

Other plugins are enabled the same way, and the list of available plugins can be seen in `pipelight-plugin --help`.

Virtualization

It's pretty common to virtualize another operating system on your computer, possibly to run a proprietary program or access a proprietary office groupware system. Whatever the reason, it's easy to accomplish on FreeBSD.

FreeBSD 10 has a new built-in hypervisor known as bhyve capable of running FreeBSD natively and other operating systems via sysutils/grub2-bhyve. Originally bhyve supported only a serial console but as of FreeBSD 11 also supports graphics.



Alternatively, VirtualBox open source edition is available at emulators/virtualbox-ose. Build it with Guest Additions enabled for the best experience virtualizing Windows and Linux.

To use VirtualBox, configure `loader.conf` to load the VirtualBox kernel module and configure `rc.conf` to start VirtualBox bridged networking.

```
/boot/loader.conf
```

```
1. vboxdrv_load="YES"
```

```
/etc/rc.conf
```

```
1. vboxnet_enable="YES"
```

```
/etc/devfs.conf
```

```
1. # Allow VirtualBox network access
2. own     vboxnetctl      root:vboxusers
3. perm    vboxnetctl      0660
```

```
1. pw usermod okeeblow -G vboxusers
```

Skype

Skype is bad, proprietary software that doesn't value your freedom. Use audio/mumble or net-im/ekiga instead if you can. If you still wish to use Skype, make sure you have Linuxulator enabled and install net-im/skype.

Version 2.1 of Skype's Linux client dropped support for OSS, found in FreeBSD as the default sound API. Thanks to this, Skype 2.0 persisted for years as the version in Ports. With the introduction of an ALSA compatibility shim in FreeBSD 8.3 and 9.0 and audio/linux-c6-alsa-plugins-oss, we can use the newer ALSA-only Skype 2.1 client.

The ALSA client, unlike the old OSS client, requires some explicit configuration to use our sound devices. They must be defined manually in `/compat/linux/etc/alsa/pcm/pcm-oss.conf`, the configuration file of alsa-plugins-oss.

In this example, I enable `pcm6/dsp6` for audio output, and `pcm8/dsp8`, a USB webcam, as a microphone source.

`/compat/linux/etc/alsa/pcm/pcm-oss.conf`

```
pcm.oss8 {
        type oss
        device /dev/dsp8
        hint {
                description "Open Sound System - Webcam"
        }
}

ctl.oss8 {
        type oss
        device /dev/mixer8
        hint {
                description "Open Sound System - Webcam"
        }
}

pcm.oss6 {
        type oss
        device /dev/dsp6
        hint {
                description "Open Sound System - S/PDIF"
        }
}

ctl.oss6 {
        type oss
```

```
        device /dev/mixer6
        hint {
                description "Open Sound System - S/PDIF"
        }
}
```

There is an even newer Skype client available as net-im/skype4, but that version is not usable on FreeBSD 10.x due to missing syscalls in that branch's Linuxulator. It will work via Pulseaudio in FreeBSD 11 and later.

For Skype 4.x, load the Video4Linux2 wrapper module:

```
kldload linux_v4l2wrapper
echo 'linux_v4l2wrapper_load="YES"' >> /boot/loader.conf
```

The microphone volume can be controlled by invoking the mixer of your chosen recording device. Let's raise the microphone volume now from 0% to 75%.

```
mixer -f /dev/mixer8 mic 75
```

ISO-8601 and other locales

This is personal preference, but I also set my `LC_TIME` environment variable to the `en_DK` faux-locale for ISO-8601 date formats instead of the ridiculous American standard.

Quick, what date is 6/5/12? Oh, it's 2012-06-05, of course.

The locale isn't included with the FreeBSD base system as it is in many Linux distributions, but it's available from Ivan Voras' blog.

```
tar -C /usr/share/locale -zxf
/path/to/your/en_DK.UTF-8.tgz
```

Enable it in the login database and `/etc/profile`.

```
/etc/login.conf
```

```
@@ -46,7 +46,8 @@
        \:ignoretime@:\
        \:umask=022:\
        \:charset=UTF-8:\
-       \:lang=en_US.UTF-8:
+       \:lang=en_US.UTF-8:\
+       \:setenv=LC_TIME=en_DK.UTF-8:
```

```
cap_mkdb /etc/login.conf
```

```
/etc/profile
```

```
LC_TIME=en_DK.UTF-8; export LC_TIME
```

Upgrade Notes

This guide assumes you will track the newest `STABLE` branch, upgrading to new stable branches at the initial `.0` release. That's not a thing I would ever recommend with OS X, but I haven't been burned by a new major FreeBSD version yet. This section notes things you need to know to keep your system in top shape when upgrading.

9.x to 10.0

If you are updating from 9.x to 10.0 or higher, run `pkg2ng` after rebooting into the new OS. This conversion script will convert the list of installed packages from the format used by the old `pkg_` tools to the format used by pkgng, the new binary package manager. If you neglect this step the OS will think you have no packages installed and your life will become very confusing.

10.0 to 10.1

`WITH_NEW_XORG` is no longer a thing. The old version is gone.

vt is a new console driver designed to replace syscons. It offers Unicode and graphics support using kernel modesetting. This is necessary to support UEFI. A loader variable `kern.vty` can select between `vt` and `sc`.

```
/boot/loader.conf

1. kern.vty=vt
```

It will default to graphics mode but can be configured to use a text mode instead if necessary.

```
/boot/loader.conf

1. hw.vga.textmode=1
```

History

2017-03-23

- Cover FreeBSD 11
- Recommend packages over ports
- Re-arrange build and Xorg sections
- Suggest Xorg autoconfiguration
- nvidia -> nvidia-modeset
- New screenshots
- Stop mentioning GNOME 2, because it's gone
- wine-compholio -> wine-staging
- linux_base-f10 is gone and linux_base-c7 is added
- NPAPI deprecation and Flash 24

2014-12-26

- Whoops, I missed that GNOME 3 and Cinnamon finally made it to the official tree!
- CentOS 6 is the default Linux base.

- HAL is mostly deprecated throughout the Free desktop world. Now that X.org no longer uses it by default I stopped recommending its use.

2014-11-29

- Cover changes in 10.1 like installer changes, UEFI support, and vt.
- Upgrade notes moved to their own section.
- Cover CentOS 6 Linuxulator.
- Cover Pipelight for browser plugins.

2014-07-27

- SLiM wants `sessiondir` now, not enumerated `sessions`.
- Update and expand Wine section
- TeXLive is now the default TeX provider, so no need to specify it over teTeX.

2014-07-17

- Section re-organization
- I stopped using or recommending GNOME
- Cover firewalling
- Add a whole desktop environments section.
- Cover pkgng
- Cover installation instead of linking to a ZFS installation guide now that bsdinstall supports it
- Stop recommending `blf` in `login.conf` because the default is `sha512` now instead of `md5`
- `KDE4_PREFIX` is `${LOCALBASE}` by default now
- `WITHOUT_NOUVEAU` is gone
- Use TeX Live
- FUSE is built-in now
- Add additional device paths for permissions
- Add virtualization section
- Wine packages are available in Ports now

2012-06-06

- Re-arrange sections
- Java
- S.M.A.R.T.
- Network time
- WiFi configuration example
- Compiz-Fusion

- IBus
- Table of contents!
- Radeon
- Theming
- ISO-8601

2012-01-02

- Initial article

```
Always seeking to survive and flourish.
--EOF--
```