FreeBSD From Scratch

Jens Schweikhardt

schweikh@FreeBSD.org

This article describes my efforts at **FreeBSD From Scratch**: a fully automated installation of a customized FreeBSD system compiled from source, including compilation of all your favorite ports and configured to match your idea of the perfect system. If you think `make world` is a wonderful concept, **FreeBSD From Scratch** extends it to `make evenmore`.

1 Introduction

Have you ever upgraded your system with `make world`? There is a problem if you have only one system on your disks. If the `installworld` fails partway through, you are left with a broken system that might not even boot any longer. Or maybe the `installworld` runs smoothly but the new kernel does not boot. Then it is time to reach for the Fixit CD and dig for those backups you have taken half a year ago.

I believe in the "wipe your disks when upgrading systems" paradigm. Wiping disks, or rather partitions, makes sure there is no old cruft left lying around, something which most upgrade procedures just do not care about. But wiping the partitions means you have to also recompile/reinstall all your ports and packages and then redo all your carefully crafted configuration tweaks. If you think that this task should be automated as well, read on.

2 Why would I (not) want FreeBSD From Scratch?

This is a legitimate question. We have **sysinstall** and the well known way to compile the kernel and the userland tools.

The problem with **sysinstall** is that it is severely limited in what, where and how it can install.

- It is normally used to install pre-built distribution sets and packages from some other source (CD, DVD, FTP). It cannot install the result of a `make buildworld`.

- It cannot install a second system under a directory in a running system.
- It cannot install in **Vinum** or **ZFS** partitions.
- It cannot compile ports, only install precompiled packages.
- It is hard to script or to make arbitrary post-installation changes.
- Last but not least, **sysinstall** is semi-officially at its End-Of-Life.

The well known way to build and install the world, as described in the Handbook (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/makeworld.html), by default replaces the existing system. Only the kernel and modules are saved. System binaries, headers and a lot of other files are overwritten; obsolete files are still present and can cause surprises. If the upgrade fails for any reason, it may be hard or even impossible to restore the previous state of the system.

FreeBSD From Scratch solves all these problems. The strategy is simple: use a running system to install a new system under an empty directory tree, while new partitions are mounted appropriately in that tree. Many config files can be copied to the appropriate place and mergemaster(8) can take care of those that cannot. Arbitrary post-configuration of the new system can be done from within the old system, up to the point where you can chroot to the new system. In other words, we go through three stages, where each stage consists of either running a shell script or invoking `make`:

1. `stage_1.sh`: Create a new bootable system under an empty directory and merge or copy as many files as are necessary. Then boot the new system.
2. `stage_2.sh`: Install desired ports.
3. `stage_3.mk`: Do post-configuration for software installed in previous stage.

Once you have used **FreeBSD From Scratch** to build a second system and found it works satisfactorily for a couple of weeks, you can then use it again to reinstall the original system. From now on, whenever you feel like an update is in order, you simply toggle the partitions you want to wipe and reinstall.

Maybe you have heard of or even tried Linux From Scratch (http://www.linuxfromscratch.org/), or LFS for short. LFS also describes how to build and install a system from scratch in empty partitions using a running system. The focus in LFS seems to be to show the role of each system component (such as kernel, compiler, devices, shell, terminal database, etc) and the details of each component's installation. **FreeBSD From Scratch** does not go into that much detail. My goal is to provide an automated and complete installation, not explaining all the gory details that go on under the hood when making the world. In case you want to explore FreeBSD at this level of detail, start looking at `/usr/src/Makefile` and follow the actions of a `make buildworld`.

There are also downsides in the approach taken by **FreeBSD From Scratch** that you should bear in mind.

- While compiling the ports during stage two the system can not be used for its usual duties. If you run a production server you have to consider the downtime caused by stage two. The ports compiled by `stage_2.conf.default` below require about 8 hours (of which 4 hours are due to **OpenOffice.org**) to build on a contemporary system. If you prefer to install packages instead of ports, you can significantly reduce the downtime to about 10 minutes.

3 Prerequisites

For going the **FreeBSD From Scratch** way, you need to have:

- A running FreeBSD system with sources and a ports tree.
- At least one unused partition where the new system will be installed.
- Experience with running mergemaster(8). Or at least no fear doing so.
- If you have no or only a slow link to the Internet: the distfiles for your favorite ports.
- Basic knowledge of shell scripting with the Bourne shell, sh(1).
- Finally, you should also be able to tell your boot loader how to boot the new system, either interactively, or by means of a config file.

4 Stage One: System Installation

The first version of this article used a single shell script for stage one where all your customization had to be done by editing the script. After valuable user feedback I have decided to separate the code and data in the scripts. This allows to have different configuration data sets to install different systems without changing any of the code scripts.

The code script for stage one is `stage_1.sh` and when run with exactly one argument, like

```
# ./stage_1.sh default
```

will read its configuration from `stage_1.conf.default` and write a log to `stage_1.log.default`.

Further below you find my `stage_1.conf.default`. You need to customize it in various places to match your idea of the "perfect system". I have tried to extensively comment the places you should adapt. The configuration script must provide four shell functions, `create_file_systems`, `create_etc_fstab`, `copy_files` and `all_remaining_customization` (in case it matters: this is also the sequence in which they will be called from `stage_1.sh`).

The points to ponder are:

- Partition layout.

  I do not subscribe to the idea of a single huge partition for the whole system. My systems generally have at least one partition for `/`, `/usr` and `/var` with `/tmp` symlinked to `/var/tmp`. In addition I share the file systems for `/home` (user homes), `/home/ncvs` (FreeBSD CVS repository replica), `/usr/ports` (the ports tree), `/src` (various checked out src trees) and `/share` (other shared data without the need for backups, like the news spool).
- Luxury items.

  What you want immediately after booting the new system and even before starting stage two. The reason for not simply chrooting to the new system during stage one and installing all my beloved ports is that in theory and in practice there are bootstrap and consistency issues: stage one has your old kernel running, but the chrooted environment consists of new binaries and headers. If the new binaries use a new system call, these binaries will die with `SIGSYS, Bad system call`, because the old kernel does not have that system call. I have seen other issues when I tried building `lang/perl5.8`.

Before you run `stage_1.sh` make sure you have completed the usual tasks in preparation for `make installworld installkernel`, like:

- configured your kernel config file
- successfully completed `make buildworld`

- successfully completed `make buildkernel KERNCONF=whatever`

When you run `stage_1.sh` for the first time, and the config files copied from your running system to the new system are not up-to-date with respect to what is under `/usr/src`, `mergemaster` will ask you how to proceed. I recommend merging the changes. If you get tired of going through the dialogues you can simply update the files on your *running* system once (Only if this is an option. You probably do not want to do this if one of your systems runs `-STABLE` and the other `-CURRENT`. The changes may be incompatible). Subsequent `mergemaster` invocations will detect that the RCS version IDs match those under `/usr/src` and skip the file.

The `stage_1.sh` script will stop at the first command that fails (returns a non-zero exit status) due to `set -e`, so you cannot overlook errors. It will also stop if you use an unset environment variable, probably due to a typo. You should correct any errors in your version of `stage_1.conf.default` before you go on.

In `stage_1.sh` we invoke `mergemaster`. Even if none of the files requires a merge, it will display and ask at the end

```
*** Comparison complete
*** Saving mtree database for future upgrades

Do you wish to delete what is left of /var/tmp/temproot.stage1? [no] no
```

Please answer `no` or just hit **Enter**. The reason is that `mergemaster` will have left a few zero sized files below `/var/tmp/temproot.stage1` which will be copied to the new system later (unless already there).

After that `mergemaster` will list the files it installed and ask if the new `login.conf` should be generated:

```
*** You chose the automatic install option for files that did not
    exist on your system.  The following were installed for you:
      /newroot/etc/defaults/rc.conf
      ...
      /newroot/COPYRIGHT

*** You installed a new aliases file into /newroot/etc/mail, but
    the newaliases command is limited to the directories configured
    in sendmail.cf.  Make sure to create your aliases database by
    hand when your sendmail configuration is done.

*** You installed a login.conf file, so make sure that you run
    '/usr/bin/cap_mkdb /newroot/etc/login.conf'
     to rebuild your login.conf database

    Would you like to run it now? y or n [n]
```

The answer does not matter since `stage_1.sh` will run cap_mkdb(1) for you in any case.

Here is the author's `stage_1.conf.default`, which you need to modify substantially. The comments give you enough information what to change.

```
# This file: stage_1.conf.default, sourced by stage_1.sh.
#
# $Id: stage_1.conf.default,v 1.2 2004/01/03 13:55:06 toor Exp toor $
# $FreeBSD: doc/en_US.ISO8859-1/articles/fbsd-from-scratch/stage_1.conf.default,v 1.4 2008/12/03 21:

# Root mount point where you create the new system. Because it is only
# used as a mount point, no space will be used on that file system as all
# files are of course written to the mounted file system(s).
DESTDIR="/newroot"

# Where your src tree is.
SRC="/usr/src"

# Where your obj is.
MAKEOBJDIRPREFIX="/usr/obj"

# Your kernel config name as from make buildkernel KERNCONF=...
KERNCONF="HAL9000"

# Your target architecture as used for make buildworld TARGET=...
# If you did not specify a TARGET when building world, it defaulted
# to the build architecture (run "uname -m" to find out if you are unsure).
TARGET="i386"  # amd64 arm i386 ia64 mips pc98 powerpc sparc64 sun4v

# Available time zones are those under /usr/share/zoneinfo.
TIMEZONE="Europe/Berlin"


#
# The create_file_systems function must create the mountpoints under
# DESTDIR, create the file systems, and then mount them under DESTDIR.
#
create_file_systems () {
  # The new root file system. Mandatory.
  # Change DEVICE names.
  DEVICE=/dev/daXYZs1a
  mkdir -m 755 -p ${DESTDIR}
  chown root:wheel ${DESTDIR}
  newfs -U ${DEVICE}
  mount -o noatime ${DEVICE} ${DESTDIR}

  # Additional file systems and initial mount points. Optional.
  DEVICE=/dev/daXYZs1e
  mkdir -m 755 -p ${DESTDIR}/var
  chown root:wheel ${DESTDIR}/var
  newfs -U ${DEVICE}
  mount -o noatime ${DEVICE} ${DESTDIR}/var

  DEVICE=/dev/daXYZs1e
  mkdir -m 755 -p ${DESTDIR}/usr
  chown root:wheel ${DESTDIR}/usr
  newfs -U ${DEVICE}
  mount -o noatime ${DEVICE} ${DESTDIR}/usr
}

#
# The create_etc_fstab function must create an fstab matching the
# file systems created in create_file_systems.
#
create_etc_fstab () {
  cat <<EOF >${DESTDIR}/etc/fstab
# Device         Mountpoint          FStype    Options              Dump Pass#
/dev/da0s1b      none                swap      sw                   0    0
/dev/da1s1b      none                swap      sw                   0    0
/dev/da2s2b      none                swap      sw                   0    0
/dev/da3s2b      none                swap      sw                   0    0
/dev/da0s1a      /                   ufs       rw,noatime           1    1
/dev/da0s1e      /var                ufs       rw,noatime           1    1
/dev/da2s1e      /usr                ufs       rw,noatime           1    1
/dev/vinum/Share /share              ufs       rw,noatime           0    2
/dev/vinum/home  /home               ufs       rw,noatime           0    2
/dev/vinum/ncvs  /home/ncvs          ufs       rw,noatime           0    2
/dev/vinum/ports /usr/ports          ufs       rw,noatime           0    2
/dev/ad1s1a      /flash              ufs       rw,noatime           0    0
/dev/ad0s1       /2k                 ntfs      ro,noauto            0    0
/dev/ad0s6       /linux              ext2fs    ro,noauto            0    0
#
/dev/cd0         /cdrom              cd9660    ro,noauto            0    0
/dev/cd1         /dvd                cd9660    ro,noauto            0    0
proc             /proc               procfs    rw                   0    0
linproc          /compat/linux/proc  linprocfs rw                   0    0
EOF
  chmod 644 ${DESTDIR}/etc/fstab
  chown root:wheel ${DESTDIR}/etc/fstab
}

#
# The copy_files function is used to copy files before mergemaster is run.
#
copy_files () {
  # Add or remove from this list at your discretion. Mostly mandatory.
  for f in \
    /.profile \
    /etc/devd.conf \
    /etc/devd.rules \
    /etc/exports \
    /etc/group \
    /etc/hosts \
    /etc/inetd.conf \
    /etc/ipfw.conf \
    /etc/make.conf \
    /etc/master.passwd \
    /etc/nsswitch.conf \
    /etc/ntp.conf \
    /etc/printcap \
    /etc/profile \
    /etc/rc.conf \
    /etc/resolv.conf \
    /etc/src.conf \
    /etc/sysctl.conf \
    /etc/ttys \
    /etc/mail/aliases \
    /etc/mail/aliases.db \
    /etc/mail/hal9000.mc \
    /etc/mail/service.switch \
    /etc/ssh/*key* \
    /etc/ssh/*_config \
    /etc/X11/xorg.conf \
    /var/cron/tabs/* \
    /root/.profile \
    /boot/*.bmp \
    /boot/loader.conf \
    /boot/device.hints ; do
    cp -p ${f} ${DESTDIR}${f}
  done
}

#
# Everything else you want to tune in the new system.
# NOTE: Do not install too many binaries here. With the old system running and
# the new binaries and headers installed you are likely to run into bootstrap
# problems. Ports should be compiled after you have booted in the new system.
#
all_remaining_customization () {
  # Without the compat symlink the linux_base files end up on the root fs:
  cd ${DESTDIR}
  mkdir -m 755 usr/compat; chown root:wheel usr/compat; ln -s usr/compat
  mkdir -m 755 usr/compat/linux;      chown root:wheel usr/compat/linux
  mkdir -m 555 usr/compat/linux/proc; chown root:wheel usr/compat/linux/proc
  mkdir -m 755 boot/grub;             chown root:wheel boot/grub
  mkdir -m 755 linux 2k;              chown root:wheel linux 2k
  mkdir -m 755 src;                   chown root:wheel src
  mkdir -m 755 share;                 chown root:wheel share
  mkdir -m 755 dvd cdrom flash;       chown root:wheel dvd cdrom flash
  mkdir -m 755 home;                  chown root:wheel home
  mkdir -m 755 usr/ports;             chown root:wheel usr/ports

  # Create the ntp and slip log files.
  touch ${DESTDIR}/var/log/ntp ${DESTDIR}/var/log/slip.log

  # Make /usr/src point to the right directory. Optional.
  # Note: some ports need part of the src tree, e.g. emulators/kqemu,
  # sysutils/lsof, sysutils/fusefs, ...
  cd ${DESTDIR}/usr
  if test "${SRC}" != /usr/src; then
    rmdir src; ln -s ${SRC}
  fi
  if test "${MAKEOBJDIRPREFIX}" != /usr/obj; then
    rmdir obj; ln -s ${MAKEOBJDIRPREFIX}
  fi

  # My personal preference is to symlink tmp -> var/tmp. Optional.
  cd ${DESTDIR}; rmdir tmp; ln -s var/tmp

  # Make spooldirs for the printers in my /etc/printcap.
  cd ${DESTDIR}/var/spool/output/lpd; mkdir -p as od ev te lp da
  touch ${DESTDIR}/var/log/lpd-errs

  # If you do not have /home on a shared partition, you may want to copy it:
  # mkdir -p ${DESTDIR}/home
  # cd /home; tar cf - . | (cd ${DESTDIR}/home; tar xpvf -)
}

# vim: tabstop=2:expandtab:shiftwidth=2:syntax=sh:
# EOF $RCSfile: stage_1.conf.default,v $
```

Download `stage_1.conf.default` (stage_1.conf.default).

Running this script installs a system that when booted provides:

- Inherited users and groups.
- Firewalled Internet connectivity over Ethernet.
- Correct time zone and NTP.
- Some more minor configuration, like `/etc/ttys` and `inetd`.

Other areas are prepared for configuration, but will not work until stage two is completed. For example we have copied files to configure printing and X11. Printing however is likely to need applications not found in the base system, like PostScript® utilities. X11 will not run before we have compiled the server, libraries and programs.

5 Stage Two: Ports Installation

> **Note:** It is also possible to install the (precompiled) packages at this stage, instead of compiling ports. In this case, `stage_2.sh` would be nothing more than a list of `pkg_add` commands. I trust you know how to write such a script. Here we concentrate on the more flexible and traditional way of using the ports.

The following `stage_2.sh` script is how I install my favorite ports. It can be run any number of times and will skip all ports that are already installed. It supports the *dryrun* option (`-n`) to just show what would be done. You run it like `stage_1.sh` with exactly one argument to denote a config file, e.g.

```
# ./stage_2.sh default
```

which will read the list of ports from `stage_2.conf.default`.

The list of ports consists of lines with two or more space separated words: the category and the port, optionally followed by an installation command that will compile and install the port (default: `make install BATCH=yes < /dev/null`). Empty lines and lines starting with # are ignored. Most of the time it suffices to only name category and port. A few ports however can be fine tuned by specifying `make` variables, e.g.:

```
www mozilla make WITHOUT_MAILNEWS=yes WITHOUT_CHATZILLA=yes install
```

In fact you can specify arbitrary shell commands, so you are not restricted to simple `make` invocations:

```
java jdk16           echo true > files/license.sh; make install BATCH=yes < /dev/null
print acroread8      yes accept | make install PAGER=ls
```

```
x11-fonts gnu-unifont make install && mkfontdir /usr/local/lib/X11/fonts/local
news inn-stable       CONFIGURE_ARGS="--enable-uucp-rnews --enable-setgid-inews" make install
```

The first two lines are examples how you can handle ports asking you to accept a licence. Note how the line for `news/inn-stable` is an example for a one-shot shell variable assignment to `CONFIGURE_ARGS`. The port `Makefile` will use this as an initial value and augment some other essential args. The difference to specifying a **make** variable on the command line with

```
news inn-stable make CONFIGURE_ARGS="--enable-uucp-rnews --enable-setgid-inews" install
```

is that the latter will override instead of augment. It depends on the particular port which method you want.

Be careful that your ports do not use an interactive install, i.e. they should not try to read from stdin other than what you explicitly give them on stdin. If they do, they will read the next line(s) from your list of ports in the here-document and get confused. If `stage_2.sh` mysteriously skips a port or stops processing, this is likely the reason.

Below is `stage_2.conf.default`. A log file named `LOGDIR/category+port` is created for each port it actually installs.

```
# vim: syntax=sh
# $Id: stage_2.conf.default,v 1.2 2004/03/06 12:50:30 toor Exp toor $
# $FreeBSD: doc/en_US.ISO8859-1/articles/fbsd-from-scratch/stage_2.conf.default,v 1.4 2008/12/03 21:
ports-mgmt portaudit
devel ccache
shells zsh
devel gettext
archivers unzip
archivers zip
security sudo
x11 xorg
x11-servers xorg-server
x11-fonts xorg-fonts-100dpi
x11-fonts xorg-fonts-75dpi
x11-fonts xorg-fonts-miscbitmaps
x11-fonts xorg-fonts-truetype
x11-fonts xorg-fonts-type1
x11-fonts gnu-unifont make install && mkfontdir /usr/local/lib/X11/fonts/local
x11-fonts urwfonts
x11-fonts webfonts
x11-toolkits open-motif
x11-wm ctwm
x11 wdm
security openssh-askpass
astro xplanet
astro xephem
editors vim
print ghostscript8
print psutils-a4
print a2ps-a4
print gv
print transfig
print teTeX
print cups-base
emulators linux_base-fc6
print acroread8 yes accept | make install PAGER=ls
java jdk16 echo true > files/license.sh; make install BATCH=yes < /dev/null
www apache22
www amaya
www firefox3
www checkbot
www p5-HTML-Parser
www validator
www mplayer-plugin
math p5-Math-Combinatorics
math p5-Bit-Vector
graphics evince
graphics xfig
graphics xv
graphics gphoto2
multimedia xawtv
lang expect
lang gawk
lang python
news tin
net freebsd-uucp
net cvsup-without-gui
net rsync
ftp wget
textproc ispell
german ispell-neu
german ispell-alt
textproc docproj
sysutils samefile
sysutils smartmontools
sysutils pstree
sysutils cdrtools
sysutils dvd+rw-tools
sysutils grub
sysutils lsof
devel subversion-freebsd
devel bcc
devel ddd
devel gindent
devel ctags
devel ElectricFence
devel strace
devel perltidy
mail procmail
mail metamail
mail mutt-devel
ports-mgmt portupgrade
news inn CONFIGURE_ARGS="--enable-uucp-rnews --enable-setgid-inews" make BATCH=yes install < /dev/nu
misc figlet-fonts
security gpa
mail spamoracle
```

```
textproc rman
multimedia mplayer
multimedia mplayer-fonts
multimedia acidrip
multimedia ogle
multimedia ogle-gui
audio pacpl
audio p5-CDDB_get
audio cowbell
shells bash
editors openoffice.org-3-RC
java eclipse
java netbeans
```

Download `stage_2.conf.default`.

6 Stage Three

You have installed your beloved ports during stage two. Some ports require a little bit of configuration. This is what stage three, the post-configuration is for. I could have integrated this post-configuration at the end of the `stage_2.sh` script. However, I think there is a conceptual difference between installing a port and modifying its out-of-the-box configuration that warrants a separate stage.

I have chosen to implement stage three as a `Makefile` because this allows easy selection of what you want to configure simply by running:

```
# make -f stage_3.mk target
```

As with `stage_2.sh` make sure you have `stage_3.mk` available after booting the new system, either by putting it on a shared partition or copying it somewhere on the new system.

7 Limitations

The automated installation of a port may prove difficult if it is interactive and does not support `make BATCH=YES install`. For a few ports the interaction is nothing more than typing `yes` when asked to accept some license. If the answer is read from standard input, simply pipe the appropriate answers to the installation command (e.g. `yes | make install`. For other ports you need to investigate where exactly the interactive command is located and deal with it appropriately. See the examples above for `print/acroread8` and `java/jdk16`.

You should also be aware of upgrade issues for config files. In general you do not know when and if the format or contents of a config file changes. A new group may be added to `/etc/group`, or `/etc/passwd` may gain another field. All of this has happened in the past. Simply copying a config file from the old to the new system may be enough most of the time, but in these cases it was not. If you update a system the canonical way (by overwriting the old files) you are expected to use `mergemaster` to deal with changes where you effectively want to merge your local config with potentially new items. Unfortunately, `mergemaster` is only available for base system files, not for anything installed by ports. Some third party software seems to be especially designed to keep me on my toes by changing the config file format every fortnight. To detect such silent changes, I keep a copy of the modified config files in the same place where I keep `stage_3.mk` and compare the result with a **make** rule, e.g. for **apache**'s `httpd.conf` in target `config_apache` with

```
@if ! cmp -s /usr/local/etc/apache2/httpd.conf httpd.conf; then \
    echo "ATTENTION: the httpd.conf has changed. Please examine if"; \
    echo "the modifications are still correct. Here is the diff:"; \
    diff -u /usr/local/etc/apache2/httpd.conf httpd.conf; \
fi
```

If the diff is innocuous I can make the message go away with `cp /usr/local/etc/apache2/httpd.conf httpd.conf`.

I have used **FreeBSD From Scratch** several times to update a `7-CURRENT` to `7-CURRENT` and `8-CURRENT` to `8-CURRENT`, i.e. I have never tried to install a `8-CURRENT` from a `7-CURRENT` system or vice versa. Due to the number of changes between different major release numbers I would expect this process to be a bit more involved. Using **FreeBSD From Scratch** for upgrades within the realm of a `STABLE` branch should work painlessly (although I have not yet tried it.)

8 The Files

Here are the three files you need beside the config files already shown above.

This is the `stage_1.sh` script, which you should not need to modify.

```
#!/bin/sh
#
# stage_1.sh - FreeBSD From Scratch, Stage 1: System Installation.
#              Usage: ./stage_1.sh profile
#              will read profile
#              and write ./stage_1.log.profile
#
# Author:      Jens Schweikhardt
# $Id: stage_1.sh,v 1.7 2004/01/03 13:50:41 toor Exp toor $
# $FreeBSD: doc/en_US.ISO8859-1/articles/fbsd-from-scratch/stage_1.sh,v 1.7 2008/12/11 19:48:21 schw

PATH=/bin:/usr/bin:/sbin:/usr/sbin

# Prerequisites:
#
# a) Successfully completed "make buildworld" and "make buildkernel"
# b) Unused partitions (at least one for the root fs, probably more for
#    the new /usr and /var, to your liking.)
# c) A customized profile file.

if test $# -ne 1; then
  echo "usage: stage_1.sh profile" 1>&2
  exit 1
fi

# ---------------------------------------------------------------------------- #
# Step 1: Create an empty directory tree below $DESTDIR.
# ---------------------------------------------------------------------------- #

step_one () {
  create_file_systems
  # Now create all the other directories. Mandatory.
  cd ${SRC}/etc; make distrib-dirs DESTDIR=${DESTDIR} TARGET=${TARGET}
}

# ---------------------------------------------------------------------------- #
# Step 2: Fill the empty /etc directory tree and put a few files in /.
# ---------------------------------------------------------------------------- #

step_two () {
  copy_files

  # Delete mergemaster's temproot, if any.
  TEMPROOT=/var/tmp/temproot.stage1
  if test -d ${TEMPROOT}; then
    chflags -R 0 ${TEMPROOT}
    rm -rf ${TEMPROOT}
  fi
  export MAKEDEVPATH="/bin:/sbin:/usr/bin"
  mergemaster -i -m ${SRC}/etc -t ${TEMPROOT} -D ${DESTDIR}
  cap_mkdb ${DESTDIR}/etc/login.conf
  pwd_mkdb -d ${DESTDIR}/etc -p ${DESTDIR}/etc/master.passwd

  # Mergemaster does not create empty files, e.g. in /var/log. Do so now,
  # but do not clobber files that may have been copied with copy_files.
  cd ${TEMPROOT}
  find . -type f | sed 's,^\./,,' |
  while read f; do
    if test -r ${DESTDIR}/${f}; then
      echo "${DESTDIR}/${f} already exists; not copied"
    else
      echo "Creating empty ${DESTDIR}/${f}"
      cp -p ${f} ${DESTDIR}/${f}
    fi
  done
  chflags -R 0 ${TEMPROOT}
  rm -rf ${TEMPROOT}
}

# ---------------------------------------------------------------------------- #
# Step 3: Install world.
# ---------------------------------------------------------------------------- #

step_three () {
  cd ${SRC}
  make installworld DESTDIR=${DESTDIR} TARGET=${TARGET}
}

# ---------------------------------------------------------------------------- #
# Step 4: Install kernel and modules.
# ---------------------------------------------------------------------------- #

step_four () {
  cd ${SRC}
  # The loader.conf and device.hints are required by the installkernel target.
  # If you have not copied them in Step 2, cp them as shown in the next 2 lines.
  #   cp sys/boot/forth/loader.conf ${DESTDIR}/boot/defaults
  #   cp sys/${TARGET}/conf/GENERIC.hints ${DESTDIR}/boot/device.hints
  make installkernel DESTDIR=${DESTDIR} KERNCONF=${KERNCONF} TARGET=${TARGET}
}

# ---------------------------------------------------------------------------- #
# Step 5: Install /etc/fstab and time zone info.
# ---------------------------------------------------------------------------- #

step_five () {
  create_etc_fstab

  # Setup time zone info; pretty much mandatory.
  cp ${DESTDIR}/usr/share/zoneinfo/${TIMEZONE} ${DESTDIR}/etc/localtime
  if test -r /etc/wall_cmos_clock; then
    cp -p /etc/wall_cmos_clock ${DESTDIR}/etc/wall_cmos_clock
  fi
}

# ---------------------------------------------------------------------------- #
# Step 6: All remaining customization.
# ---------------------------------------------------------------------------- #

step_six () {
  all_remaining_customization
}

do_steps () {
  echo "PROFILE=${PROFILE}"
  echo "TARGET=${TARGET}"
  echo "DESTDIR=${DESTDIR}"
  echo "SRC=${SRC}"
  echo "KERNCONF=${KERNCONF}"
  echo "TIMEZONE=${TIMEZONE}"
  echo "TYPE=${TYPE}"
  echo "REVISION=${REVISION}"
  echo "BRANCH=${BRANCH}"
  echo "RELDATE=${RELDATE}"
  step_one
  step_two
  step_three
  step_four
  step_five
  step_six
}

# ---------------------------------------------------------------------------- #
# The ball starts rolling here.
# ---------------------------------------------------------------------------- #

PROFILE="$1"
```

```
set -x -e -u # Stop for any error or use of an undefined variable.
. ${PROFILE}

# Determine a few variables from the sources that were used to make the
# world. The variables can be used to modify actions, e.g. depending on
# the system's version. The __FreeBSD_version numbers
# for RELDATE are documented in the Porter's Handbook,
# doc/en_US.ISO8859-1/books/porters-handbook/freebsd-versions.html.
# Scheme is:  <major><two digit minor><0 if release branch, otherwise 1>xx
# The result will be something like
#
#   TYPE="FreeBSD"
#   REVISION="8.0"
#   BRANCH="RC"      { "CURRENT", "STABLE", "RELEASE" }
#   RELDATE="800028"
#
eval $(awk '/^(TYPE|REVISION|BRANCH)=/' ${SRC}/sys/conf/newvers.sh)
RELDATE=$(awk '/^[ \t]*#[ \t]*define[ \t][ \t]*__FreeBSD_version[ \t]/ {
                print $3
              }' ${SRC}/sys/sys/param.h)

echo "=> Logging to stage_1.${PROFILE}.log"
do_steps 2>&1 | tee "stage_1.${PROFILE}.log"

# vim: tabstop=2:expandtab:shiftwidth=2:
# EOF $RCSfile: stage_1.sh,v $
```

Download `stage_1.sh`.

This is the `stage_2.sh` script. You may want to modify the variables at the beginning.

```
#!/bin/sh
#
# stage_2.sh - FreeBSD From Scratch, Stage 2: Ports Installation.
#              Usage: ./stage_2.sh [-hnp] configname
#
# Author:      Jens Schweikhardt
# $Id: stage_2.sh,v 1.5 2004/01/23 22:09:19 toor Exp toor $
# $FreeBSD: doc/en_US.ISO8859-1/articles/fbsd-from-scratch/stage_2.sh,v 1.5 2004/07/19 21:02:26 schw

DBDIR="/var/db/pkg"
PORTS="/usr/ports"
: ${PACKAGES:=${PORTS}/packages}
LOGDIR="/home/root/setup/ports.log"; mkdir -p ${LOGDIR}
PKG_PATH="/cdrom/packages/All:/dvd/packages/All"
PKG=

MYNAME="$(basename $0)"
usage () {
exec >&2
echo "usage: ${MYNAME} [-hnp] configname"
echo ""
echo "  Options:"
echo "  -h    Print this help text."
echo "  -n    Dryrun: just show what would be done."
echo "  -p    Install a precompiled package if one can be found."
echo ""
echo "  The config file (stage_2.conf.configname) is a list of"
echo "  ports to install with one entry per line. Each line"
echo "  consists of two or three space separated fields:"
echo "  category, port, and optionally a build command."
echo ""
exit 1
}

# Look for a package in these locations in sequence.
# Returns as soon as the first is found. Result on stdout.
#
#   ${PORTS}/${CATEGORY}/${NAME}
#   ${PACKAGES}/All
#   ${PACKAGES}/${CATEGORY}
#   ${PKG_PATH}
#
find_package () {
echo "${PORTS}/${CATEGORY}/${NAME}:${PACKAGES}/All:${PACKAGES}/${CATEGORY}:${PKG_PATH}" |
tr : '\n' |
while read d; do
test -d "${d}" || continue
PKG=$(ls ${d}/${PKGNAME}.* 2>/dev/null)
test $? -eq 0 && echo "${PKG}" && return
done
}

#
# Parse command line arguments.
#
args=`getopt hnp $*`
if test $? != 0; then
usage
fi
set -- $args
DRYRUN=
CHKPKG=
for i; do
case "$i" in
-n) DRYRUN="yes"; shift;;
-p) CHKPKG="yes"; shift;;
--) shift; break;;
*) usage;;
esac
done
if test $# -eq 1; then
DATAFILE="$1"
else
usage
fi
```

```
#
# Loop over the ports list.
#
while read CATEGORY NAME CMD; do
case "${CATEGORY}" in
\#*) continue;;
”) continue;;
esac
DIR="${PORTS}/${CATEGORY}/${NAME}"
if ! test -d "${DIR}"; then
echo "$DIR does not exist -- ignored"
continue
fi
cd ${DIR}
PKGNAME=‘make -V PKGNAME‘
if test -n "${CHKPKG}"; then
PKG=$(find_package)
else
PKG=""
fi
if test -d "${DBDIR}/${PKGNAME}"; then
echo "${CATEGORY}/${NAME} already installed as ${PKGNAME}"
continue
fi
LOG="${LOGDIR}/${CATEGORY}+${NAME}"
echo "===> Installing ${CATEGORY}/${NAME}; logging to ${LOG}"
test -n "${CMD}" || CMD="make install BATCH=yes < /dev/null"
if test -n "${DRYRUN}"; then
if test -n "${PKG}"; then
echo pkg_add -v ${PKG}
else
echo "${CMD}"
fi
continue
fi
date "++++ Started %v %T +++" > ${LOG}
STARTED=$(date +%s)
(
if test -n "${PKG}"; then
echo "Found package ${PKG}"
pkg_add -v ${PKG}
else
echo "CMD: ${CMD}"
make clean
eval "${CMD}"
make clean # Uncomment if diskspace is tight under ${PORTS}.
fi
) 2>&1 | tee -a ${LOG}
FINISHED=$(date +%s)
DURATION=$(dc -e "${FINISHED} ${STARTED} - p")
date "++++ Finished %v %T after ${DURATION} secs +++" >> ${LOG}
done < stage_2.conf.${DATAFILE}
```

```
# vim: tabstop=4:
# EOF $RCSfile: stage_2.sh,v $
```

Download `stage_2.sh`.

This is my `stage_3.mk` to give you an idea how to automate all reconfiguration.

```
# stage_3.mk - FreeBSD From Scratch, Stage 3: Ports Post-Configuration.
#              Usage: make -f stage_3.mk all     (configure everything)
#                or   make -f stage_3.mk target  (just configure target)
#
# Author:      Jens Schweikhardt
#
# It is a good idea to make sure any target can be made more than
# once without ill effect.
#
# $Id: stage_3.mk,v 1.8 2004/03/27 16:53:11 toor Exp toor $
# $FreeBSD: doc/en_US.ISO8859-1/articles/fbsd-from-scratch/stage_3.mk,v 1.5 2008/12/03 21:59:51 schw

.POSIX:

message:
@echo "Please use one of the following targets:"
@echo "config_apache"
@echo "config_cups"
@echo "config_firefox"
@echo "config_inn"
@echo "config_javaplugin"
@echo "config_openoffice"
@echo "config_sudo"
@echo "config_TeX"
@echo "config_tin"
@echo "config_wdm"
@echo "config_uucp"
@echo "all -- all of the above"

all: \
config_apache \
config_cups \
config_firefox \
config_inn \
config_javaplugin \
config_openoffice \
config_sudo \
config_TeX \
config_tin \
config_wdm \
config_uucp

APACHE = apache22
config_apache:
# 1. Modify httpd.conf.
perl -pi \
-e 's/^\s*ServerAdmin.*/ServerAdmin schweikh\@schweikhardt.net/;' \
-e 's/^#?ServerName .*/ServerName hal9000.schweikhardt.net:80/;' \
-e 's/^\s*Listen.*/Listen 127.0.0.1:80/;' \
-e 's/^\s*Deny from all/    Allow from 127.0.0.1/i;' \
-e 's,/usr/local/www/$(APACHE)/cgi-bin/,/home/opt/www/cgi-bin/,;' \
  /usr/local/etc/$(APACHE)/httpd.conf
cp w3c-validator.conf /usr/local/etc/$(APACHE)/Includes
# 2. Restore symlinks to web pages.
cd /usr/local/www/$(APACHE)/data && \
ln -fs /home/schweikh/prj/homepage schweikhardt.net && \
ln -fs /home/opt/www/test .
# 3. Restore W3C Validator config.
mkdir -p /etc/w3c
cp /usr/local/www/validator/htdocs/config/validator.conf.sample \
/etc/w3c/validator.conf
perl -pi \
-e 's/^Allow Private IPs.*/Allow Private IPs = yes/;' \
/etc/w3c/validator.conf
# Test if the httpd.conf has changed.
@if ! cmp -s /usr/local/etc/$(APACHE)/httpd.conf httpd.conf; then \
echo "ATTENTION: the httpd.conf has changed. Please examine if"; \
echo "the modifications are still correct. If so you can simply"; \
echo "cp /usr/local/etc/$(APACHE)/httpd.conf httpd.conf"; \
echo "to make this message go away. Here is the diff:"; \
diff -u /usr/local/etc/$(APACHE)/httpd.conf httpd.conf; \
fi
if test -f /var/run/httpd.pid; then \
/usr/local/etc/rc.d/$(APACHE) stop; \
/usr/local/etc/rc.d/$(APACHE) start; \
else \
/usr/local/etc/rc.d/$(APACHE) start; \
fi

# The original ppd file is from http://www.cups.org/ppd.php?L63+I0+T+Q2300
# = http://www.cups.org/ppd/hp/de/hpc2325s.ppd.gz
config_cups:
chmod 644 /usr/local/etc/cups/cupsd.conf
cp printers.conf /usr/local/etc/cups/printers.conf
cp LaserJet_2300d.ppd /usr/local/etc/cups/ppd/LaserJet_2300d.ppd

config_firefox:
# Make this group wheel writable to allow extensions being installed.
chmod -R g+w /usr/local/lib/firefox3/chrome

config_inn:
pw usermod -n news -d /usr/local/news -s /bin/sh
mkdir -p /share/news/spool/outgoing \
         /share/news/spool/incoming \
         /share/news/spool/articles \
         /share/news/spool/overview \
         /share/news/spool/tmp      \
         /share/news/db
chown -R news:news /share/news
# Give the news system its initial configuration.
```

```
cd /home/root/setup && \
if test ! -f /share/news/db/active; then \
echo "installing /share/news/db/active"; \
install -C -o news -g news -m 664 active /share/news/db; \
fi; \
if test ! -f /share/news/db/newsgroups; then \
echo "installing /share/news/db/newsgroups"; \
install -C -o news -g news -m 664 newsgroups /share/news/db; \
fi
# Configure storage method.
cd /home/root/setup &&    \
printf "%s\n%s\n%s\n%s\n" \
"method tradspool {"  \
"  newsgroups: *"     \
"  class: 0"          \
"}"                   \
>storage.conf &&          \
install -C -o news -g news -m 664 storage.conf /usr/local/news/etc
# Configure newsfeeds.
printf "%s\n%s\n" \
"ME:*::"      \
"shuttle/news2.shuttle.de:!junk,!control:B32768/512,Tf,Wfb:" \
>/usr/local/news/etc/newsfeeds
# Configure inn.conf.
perl -pi                                                        \
-e 's/^#*\s*(organization:\s*).*/$$1"An Open Pod Bay Door"/;'   \
-e 's/^#*\s*(pathhost:\s*).*/$$1hal9000.schweikhardt.net/;'     \
-e 's/^#*\s*(server:).*/$$1 localhost/;'                        \
-e 's/^#*\s*(domain:).*/$$1 schweikhardt.net/;'                 \
-e 's/^#*\s*(fromhost:).*/$$1 schweikhardt.net/;'               \
-e 's,^#*\s*(moderatormailer:).*,$$1 \%s\@moderators.isc.org,;' \
-e 's,^#*\s*(pathdb:\s*).*,$$1/share/news/db,;'                 \
-e 's,/usr/local/news/spool,/share/news/spool,;'                \
/usr/local/news/etc/inn.conf
# Create empty history, if none there.
# See post-install in /usr/ports/news/inn-stable/Makefile.
set -e; cd /share/news/db; \
if test ! -f history; then \
touch history; \
chmod 644 history; \
chown news:news history; \
su -fm news -c "/usr/local/news/bin/makedbz -i"; \
for s in dir hash index; do \
mv history.n.$${s} history.$${s}; \
done; \
fi
# Configure send-uucp.
echo shuttle:shuttle >/usr/local/news/etc/send-uucp.cf
# Satisfy inncheck:
set -e; cd /usr/local/news/etc; \
chown news:news *; \
chmod 640 control.ctl expire.ctl nntpsend.ctl readers.conf
/usr/local/news/bin/inncheck
```

```
# Test if the inn.conf has changed.
@if ! cmp -s /usr/local/news/etc/inn.conf inn.conf; then \
echo "ATTENTION: the inn.conf has changed. Please examine if"; \
echo "the modifications are still correct. If so you can simply"; \
echo "cp /usr/local/news/etc/inn.conf inn.conf"; \
echo "to make this message go away. Here is the diff:"; \
diff -u /usr/local/news/etc/inn.conf inn.conf; \
fi
if ! test -f /usr/local/news/run/innd.pid; then \
/usr/local/etc/rc.d/innd start; \
fi

config_javaplugin:
cd /usr/local/lib/firefox3/plugins && \
  ln -fs /usr/local/jdk1.6.0/jre/plugin/$$(uname -m)/ns7/libjavaplugin_oji.so

config_openoffice:
# Copy some truetype files so ooo can use them.
find /usr/local/openoffice.org* -type d -name truetype \
-exec echo cp *.ttf {} \; -exec cp *.ttf {} \;

config_sudo:
if ! grep -q schweikh /usr/local/etc/sudoers; then \
echo 'schweikh ALL = (ALL) NOPASSWD: ALL' >> /usr/local/etc/sudoers; \
fi
chmod 440 /usr/local/etc/sudoers

config_TeX:
# textproc/docproj advises: to typeset the FreeBSD Handbook with JadeTeX,
# change the following settings to the listed values:
perl -pi                                      \
-e 's/^% original texmf.cnf/% texmf.cnf/;'    \
-e 's/^(hash_extra\s*=\s*).*/$${1}60000/;'    \
-e 's/^(pool_size\s*=\s*).*/$${1}1000000/;'   \
-e 's/^(max_strings\s*=\s*).*/$${1}70000/;'   \
-e 's/^(save_size\s*=\s*).*/$${1}10000/;'     \
/usr/local/share/texmf/web2c/texmf.cnf
# Test if the texmf.cnf has changed.
@if ! cmp -s /usr/local/share/texmf/web2c/texmf.cnf texmf.cnf; then \
echo "ATTENTION: the texmf.cnf has changed. Please examine if"; \
echo "the modifications are still correct. If so you can simply"; \
echo "cp /usr/local/share/texmf/web2c/texmf.cnf texmf.cnf"; \
echo "to make this message go away. Here is the diff:"; \
diff -u /usr/local/share/texmf/web2c/texmf.cnf texmf.cnf; \
fi

config_tin:
# Point tin to our files.
printf "%s\n%s\n%s\n"                          \
"activefile=/share/news/db/active"         \
"newsgroupsfile=/share/news/db/newsgroups" \
"spooldir=/share/news/spool/articles"      \
>/usr/local/etc/tin.defaults
```

```
config_wdm:
cp daemon1-JS-1600x1200.jpg FreeBSD_small.png \
/usr/local/lib/X11/wdm/pixmaps
perl -pi \
-e 's,^(DisplayManager\*wdmBg:).*,\1 pixmap:/usr/local/lib/X11/wdm/pixmaps/daemon1-JS-1600x1200.jpg,
-e 's,^(DisplayManager\*wdmLogo:).*,\1 /usr/local/lib/X11/wdm/pixmaps/FreeBSD_small.png,;' \
-e 's,^(DisplayManager\*wdmWm:).*,\1 ctwm:icewm:xfce4:tvtwm,;' \
/usr/local/lib/X11/wdm/wdm-config
@if ! cmp -s /usr/local/lib/X11/wdm/wdm-config wdm-config; then \
echo "ATTENTION: the wdm-config has changed. Please examine if"; \
echo "the modifications are still correct. If so you can simply"; \
echo "cp /usr/local/lib/X11/wdm/wdm-config wdm-config"; \
echo "to make this message go away. Here is the diff:"; \
diff -u /usr/local/lib/X11/wdm/wdm-config wdm-config; \
fi

config_uucp:
cd /etc/mail && make install SENDMAIL_MC=/etc/mail/hal9000.mc
# Make the uucp user's shell the correct uucico, so su(1) works.
chpass -s /usr/local/libexec/uucp/uucico uucp
# UUCP expects to find /usr/bin/rnews.
cd /usr/bin && ln -fs ../local/news/bin/rnews .
# Actual UUCP configuration.
echo nodename js2015           > /usr/local/etc/uucp/config
echo shuttle js2015 `cat uucp` > /usr/local/etc/uucp/call
printf 'port tcp\ntype tcp\n'  > /usr/local/etc/uucp/port
printf "%s\n"                          \
"call-login    *"                  \
"call-password *"                  \
"time          any"                \
"system        shuttle"            \
"address       mail.s.shuttle.de"  \
"commands      rmail rnews"        \
"port          tcp"                \
>/usr/local/etc/uucp/sys
cd /usr/local/etc/uucp && chown uucp:uucp * && chmod o-rwx *
# Trigger uucico after booting.
mkdir -p /usr/local/etc/rc.d
cp uucp.sh /usr/local/etc/rc.d
# Rebuild the aliases.db.
cp aliases /etc/mail/aliases
newaliases

# vim: tabstop=4:
# EOF $RCSfile: stage_3.mk,v $
```

Download `stage_3.mk`.