

Using PTXdist on Mac OS

based on PTXdist 2012.04

Bernhard Walle*

2012-04-20

*bernhard@bwalle.de

Contents

1. Motivation	3
2. Basics	4
2.1. Getting OpenSource Software on Mac OS	4
2.2. Preparing the Hard Disk	5
2.2.1. Creating and Using a <i>Sparse Bundle</i>	6
3. Installing PTXdist	7
3.1. Requirements	7
3.1.1. Mac OS	7
3.1.2. Host compiler	7
3.2. GNU Tools	7
3.3. Installation	8
3.4. First setup	9
4. Building an OSELAS.Toolchain	11
5. Building an Embedded Linux project	12
5.1. Using OSELAS.BSP-Pengutronix-Generic	12
5.1.1. Getting the BSP	12
5.1.2. Building FSF GCC	13
5.1.3. Building Qemu	13
5.1.4. Running the System	13
5.2. Using real Hardware	14
6. Limitations	15
6.1. Building UBI or JFFS2 images	15
6.2. Linux kernel	15
6.3. Bootloader	15
7. Using a Mac for Embedded Linux Development	16
7.1. Using a serial console	16
7.2. Exporting directories via NFS	16
7.3. Mounting <i>ext2</i> partitions	18
A. About this Document	19
B. Change History	20
Bibliography	21

1. Motivation

PTXdist (<http://www.ptxdist.org>) is a great way to build Embedded Linux systems. It downloads all required components, configures them for cross-compilation and finally builds a target image and/or target packages. In addition, it provides an easy way to build a cross-toolchain for most common processors. Read [2] for a description how to use *PTXdist*.

While a Linux based host is normally the platform of choice for Embedded Linux development, it is not absolutely *necessary* to use Linux for this. In commercial environments, it's quite common to use Microsoft Windows. Commercial Linux distributions such as *MontaVista Linux* provide Windows IDEs. Under the hood, Cygwin is used for much of the work.

Since Mac OS is a BSD-based Unix, there's no need for such an "emulation layer". However, since BSD is not GNU, and Mac OS doesn't use the ELF binary format but Mach-O, things are not as easy as it may seem.

The reader may ask why I use Mac OS to build Linux systems. Even on a Mac computer, Linux can be installed for dual booting. However, I don't like dual boot. You have to install, maintain and configure everything twice, it takes a lot of time and even disk space. The more comfortable approach would be to use some virtualization and install Linux there, which is what I do in parallel. So the answer just may be just "because I can".

But before starting with the manual I would like to thank the Pengutronix team for providing *PTXdist* to the community and keeping it up-to-date. Especially I would like to thank MICHAEL OLBRICH for reviewing and accepting my patches and JÜRGEN BEISERT for the good documentation and for useful comments to this document. Also I want to thank ANDREAS BIEßMANN who developed some of the Mac OS-related patches especially for the toolchain and who gave valuable feedback on this document.

2. Basics

This guide should make it a bit easier if you decide to use *PTXdist* on Mac OS the first time. You should read [1] and [2] before reading this documentation. I would also recommend you to install and use *PTXdist* on Linux first and get started with it. Using Linux in VirtualBox¹ is probably the easiest method if you don't have a second computer that runs Linux.

Mac OS has a fancy GUI and you probably never ever need to use the command line if you do not want. However, *PTXdist* is based on the command line, you need a Terminal to use it. You can use *Terminal.app* from Apple, but I would recommend *iTerm 2*² as terminal emulator.

2.1. Getting OpenSource Software on Mac OS

While Mac OS comes with quite a lot (compared to Windows) command line tools, this is not sufficient to do Linux development. The set of commands available can be compared to a minimal installation of a Linux distribution.

Instead of doing everything manually, people already built systems which makes it easy to install additional (Unix-based) software on Mac OS:

- **MacPorts** is based on the BSD ports system. It installs everything from source and has quite a huge number of packages³. It can be downloaded from <http://www.macports.org> and you need to install Xcode from Apple including the command line utilities before. Its documentation [3] contains everything you need to get started.
- **Fink** (<http://www.finkproject.org>) uses the package management tools and the package format from Debian. Contrary to MacPorts, it has both binaries and can be built from source (but as far as I know, the binaries are less up-to-date compared the source packages).
- **Homebrew** (<http://mxcl.github.com/homebrew/>) is an alternative to *MacPorts*.

As you may have guessed, I personally use *MacPorts*. When it matters, the documentation focusses on *MacPorts* for simplicity. However, *PTXdist* is known to be work also with the tools installed with *Fink*. In fact, it's not required to use any of them. Just install the required tools manually.

¹<http://www.virtualbox.org>

²<http://www.iterm2.com>

³sometimes there are even packages that Linux distributions like openSUSE don't have in their repositories

2.2. Preparing the Hard Disk

Unfortunately, the default file system of Mac OS is *case insensitive* (but *case preserving*). So if a file Foo exists and you create a file foo, then instead of creating a new file, the already existing Foo is used.

As *PTXdist* is a Linux build system, it assumes a case sensitive file system. Luckily, the Mac OS file system *HFS+* can also be used in a case sensitive mode. Whenever you create a file system with *Disk Utility*, you can choose the case sensitive variant. So there are various ways to solve the problem with the case insensitive file system:

1. Re-install Mac OS on a case sensitive HFS+ file system.
This option is not recommended as some applications assume a case insensitive file system and will not run on a case sensitive file system. A famous example are the Adobe applications.
2. Use a network file system (NFS in that case).
This is just slow and not recommended at all.
3. Use an external disk and format it using a case sensitive file system.⁴
If you want to use *PTXdist* only on this external disk, everything is fine with that solution. But note that the speed of external disks (except maybe ones connected with *Thunderbolt*⁵) is normally lower compared to internal disks.
4. Use a case sensitive data partition on your internal disk.
As Mac OS partitions can be resized online, this works quite well. Only if you have installed Windows or Linux on your Mac using *BootCamp*, you may experience problems. But if Mac OS is the only operating system, this works fine.
5. Use a disk image, a so-called *sparse bundle*.
This is just a file with a file system inside it. In Linux terms, the file is then “loopback mounted”. Section 2.2.1 describes this approach.

It's not required to install *PTXdist* itself in a case sensitive partition. Only the projects that are built need to be located there. When building a *PTXdist* project, it automatically checks if the file system is case insensitive and rejects the build in that case⁶

☞ You may want to disable *Spotlight* for this image as it probably doesn't contain anything useful for indexing and it only makes your computer slower. To do so, execute `mdutil -i off /Volumes/Development` on the attached volume. Just replace *Development* with the real name of the volume.

To disable indexing only for some directories, you can add it to the list of excluded directories in the System Settings → Spotlight → Privacy tab.

⁴For simplicity I would recommend HFS+.

⁵[http://en.wikipedia.org/wiki/Thunderbolt_\(interface\)](http://en.wikipedia.org/wiki/Thunderbolt_(interface))

⁶The check is performed in the `check_dirs` function in the *ptxdist* main program.

2.2.1. Creating and Using a *Sparse Bundle*

You can create and maintain disk images using the *Disk Utility* which comes with Mac OS. However, in my opinion it's much easier on the command line using *hdiutil*(1). In order to create a new disk image, just execute

```
% hdiutil create -size 100g \  
  -fs "Case-sensitive Journaled HFS+" \  
  -volname "Development" \  
  devel.sparsebundle
```

Of course you can choose another size than 100 Gigabytes as I did in the example above. As the volume name is used as mount point in */Volumes*, it's recommended to not use any whitespace inside.

After the image is created, you can actually mount it using

```
% hdiutil attach devel.sparsebundle
```

Now you should have an empty file system inside */Volume/Development* which you can use to build Embedded Linux Systems. To detach from it, i. e. to unmount it, use

```
% hdiutil detach /Volumes/Development
```

After detaching, you can even resize the image using the *hdiutil resize* command.

3. Installing PTXdists

3.1. Requirements

3.1.1. Mac OS

The documentation focusses on Mac OS 10.7 (Lion).

3.1.2. Host compiler

PTXdists requires a working host compiler. Apple bundles the compiler with the IDE. The guide has been tested with following version of LLVM-GCC which comes with Xcode 4.3.2.

```
% gcc --version
i686-apple-darwin11-llvm-gcc-4.2 (GCC) 4.2.1 (Based on Apple Inc. build 5658) \
    (LLVM build 2336.9.00)
Copyright (C) 2007 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Normally Xcode is installed via the *AppStore*. Alternatively, you can download Xcode (including older versions) from the *Apple Developer Website* (<https://developer.apple.com/downloads/index.action>). You have to register there (an Apple ID is not enough), but it's free.

To get the command line tools, you need to start *Xcode*. You can cancel the startup dialog. Then select in the menu *Xcode* → *Preferences* → *Downloads* → *Components* → *Command Line Tools* → *Install*. Figure 3.1 shows the dialog to install the tools.

3.2. GNU Tools

PTXdists consists of a lot of shell scrips and Makefiles. Making them “portable” which means to use only the minimal subset of commands and options specified by POSIX and maybe the *Single Unix Specification* would be a nightmare. Instead, we just install the GNU variants. At a minimum, you need to install the tools mentioned in table 3.1.

Even if you put `/opt/local/bin` (that's the place where *MacPorts* installs its binaries) in front of your `$PATH`, *MacPorts* doesn't overwrite existing tools. So for example even if *GNU date* is present, the command `date` still points to the BSD variant from Apple. The GNU variant is available as `gdate`. In addition, they are provided without the prefix in `/opt/local/libexec/gnubin`.

3. Installing *PTXdist*

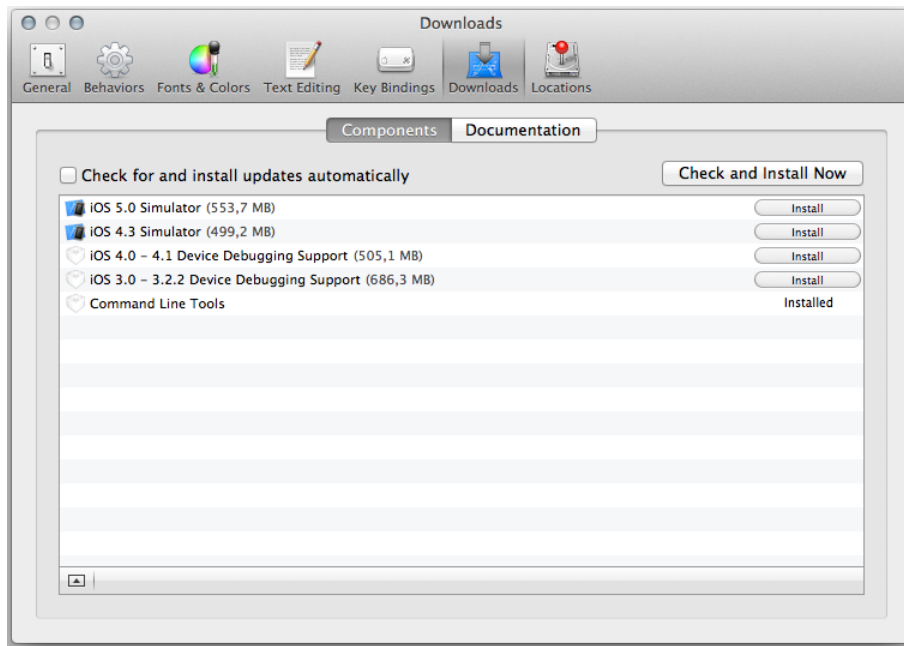


Figure 3.1.: Installation of Xcode commandline tools

3.3. Installation

Grab a copy of *PTXdist* at <http://www.ptxdist.org/software/ptxdist/download/>. You can also use the GIT tree, but then you need install the Autotools on the host first — they are not required when using the tarball because that one ships with generated files.

First, extract the tarball and change into the newly created directory:

```
% tar xvf ptxdist-2012.04.0.tar.bz2
% cd ptxdist-2012.04.0/
```

After that, configure it:

```
% PATH=/opt/local/libexec/gnubin:$PATH \
./configure --prefix=/opt/ptxdist
```

The prefix `/opt/ptxdist` is just a proposal, you can use any other directory of your choice, but it's recommended to keep it separate from the system (`/usr`) and from *MacPorts* (`/opt/local`).

You might have noticed the `PATH` setting: We use the non-prefixed GNU tools. But we don't need to export the `PATH` because *PTXdist* remembers the location of the tools it has found in the the configuration step: It puts symlinks in the binary directory of the installation. We will see later.

Now just build the small parts of *PTXdist* that need to be built:

```
% make
```

Finally, install the binaries, scripts and recipes:

3. Installing PTXdist

Name	Description	MacPorts / Fink
GNU awk	The GNU awk utility.	gawk
GNU Coreutils	Standard programs like <i>uname(1)</i> .	coreutils
GNU tar	The standard tape archiver with more options.	gnutar / tar
GNU Findutils	<i>find(1)</i> , <i>locate(1)</i> and <i>xargs(1)</i> .	findutils
GNU sed	Command-line stream editor <i>sed(1)</i> .	gsed / sed
GNU wget	Downloads files from the web via HTTP or FTP.	wget
xz	<i>xz(1)</i> and <i>unxz(1)</i> compression and uncompression tools.	xz
quilt (optional)	Tool to make working with the patches easier.	quilt
dialog (optional)	Needs to be installed to use <i>ptxdist</i> menu.	dialog

Table 3.1.: GNU tools required to be installed to use *PTXdist*

```
% sudo make install
```

Now let's look in `/opt/ptxdist/lib/ptxdist-2012.04.0/bin`:

```
% ls -l /opt/ptxdist/lib/ptxdist-2012.04.0/bin
lrwxr-xr-x 1 root admin 19 6 Apr 19:42 awk@ -> /opt/local/bin/gawk
lrwxr-xr-x 1 root admin 31 6 Apr 19:42 chmod@ -> /opt/local/libexec/gnubin/chmod
lrwxr-xr-x 1 root admin 31 6 Apr 19:42 chown@ -> /opt/local/libexec/gnubin/chown
lrwxr-xr-x 1 root admin 28 6 Apr 19:42 cp@ -> /opt/local/libexec/gnubin/cp
lrwxr-xr-x 1 bwall admin 30 7 Apr 09:00 find@ -> /opt/local/libexec/gnubin/find
lrwxr-xr-x 1 root admin 33 6 Apr 19:42 install@ -> /opt/local/libexec/gnubin/install
lrwxr-xr-x 1 root admin 32 6 Apr 19:42 md5sum@ -> /opt/local/libexec/gnubin/md5sum
lrwxr-xr-x 1 root admin 31 6 Apr 19:42 mkdir@ -> /opt/local/libexec/gnubin/mkdir
lrwxr-xr-x 1 root admin 31 6 Apr 19:42 mknod@ -> /opt/local/libexec/gnubin/mknod
lrwxr-xr-x 1 root admin 28 6 Apr 19:42 mv@ -> /opt/local/libexec/gnubin/mv
-rwxrwxr-x 1 root admin 57707 6 Apr 19:40 ptxdist*
lrwxr-xr-x 1 root admin 24 6 Apr 19:42 python@ -> /opt/local/bin/python2.7
lrwxr-xr-x 1 root admin 34 6 Apr 19:42 readlink@ -> /opt/local/libexec/gnubin/readlink
lrwxr-xr-x 1 root admin 28 6 Apr 19:42 rm@ -> /opt/local/libexec/gnubin/rm
lrwxr-xr-x 1 root admin 31 6 Apr 19:42 rmdir@ -> /opt/local/libexec/gnubin/rmdir
lrwxr-xr-x 1 root admin 29 6 Apr 19:42 sed@ -> /opt/local/libexec/gnubin/sed
lrwxr-xr-x 1 root admin 30 6 Apr 19:42 stat@ -> /opt/local/libexec/gnubin/stat
lrwxr-xr-x 1 root admin 29 6 Apr 19:42 tar@ -> /opt/local/libexec/gnubin/tar
lrwxr-xr-x 1 root admin 31 6 Apr 19:42 xargs@ -> /opt/local/libexec/gnubin/xargs
```

As you can see, there are a lot of symbolic links in this directory. *PTXdist* sets the `$PATH` to include that directory in the front just after startup and so the “right” tools are used without modification of the system's `$PATH`. Latter would have a lot of side effects, so it should be avoided.

Since we installed the software to `/opt/ptxdist`, we create a symlink to have it in the search path:

```
% sudo ln -s /opt/ptxdist/bin/ptxdist /usr/local/bin
```

3.4. First setup

Just after installing *PTXdist*, configure it with

```
ptxdist setup
```

Now a setup screen as shown in figure 3.2 should appear.

PTXdist documentation [1] explains the various options.

3. Installing PTXdist

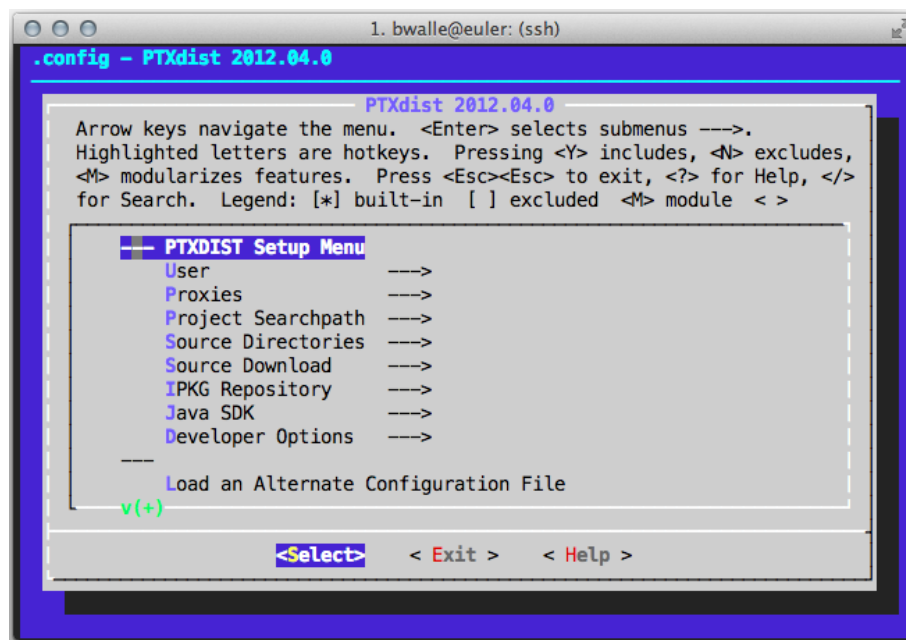


Figure 3.2.: Setup screen of *PTXdist*

4. Building an OSELAS.Toolchain

While *PTXdist* can be used with most (cross) toolchains, the easiest is to use OSELAS.Toolchain (http://www.pengutronix.de/oselas/toolchain/index_en.html).

As the current GIT version of the toolchain contains some unreleased fixes, I recommend using that one¹:

```
% git clone git://git.pengutronix.de/git/OSELAS.Toolchain.git
% cd OSELAS.Toolchain
```

In our example we build the toolchain for armv4 to be used with Qemu, but nothing differs if you select another one.

```
% ptxdist select ptxconfigs/arm-v4t-*.ptxconfig
```

Now the recommended way would be to use the version of *PTXdist* for which the current project was built. However, *PTXdist* 2011.11 doesn't contain the fixes required for Mac OS. So as a exception, we migrate the project:

```
% ptxdist migrate
```

You can run `git diff` to see what that command changed. If everything went right, only some version strings should have changed. Now we can build the toolchain which takes some hours, even on up-to-date hardware:

```
% ulimit -n 5000
% ptxdist go
```

The call to *ulimit(1)* is required at least on my Mac because otherwise I get the error “too many open files” when building glibc, even when I build with `--j-intern=1` which disables parallelization. The error results in the failure to build codepage conversion objects.

Since the toolchain will be installed into `/opt`, `sudo` asks you to enter your password.

After the toolchain build has finished, you should have `/opt/OSELAS.Toolchain-2011.11.0/arm-v4t-linux-gnueabi/gcc-4.6.2-glibc-2.14.1-binutils-2.21.1a-kernel-2.6.39-sanitized/bin/arm-v4t-linux-gnueabi-gcc` and some other files in that directory such as `arm-v4t-linux-gnueabi-gdb` for remote debugging.

To avoid accidentally overwriting some files of the toolchain, remove write permission with

```
% sudo chown -R root:wheel /opt/OSELAS.Toolchain-2011.11.0
```

¹Since “current GIT” is a moving target, you can use the hash 7f195586c0170ba6ec71c01ab73d465b94d6dbbd.

5. Building an Embedded Linux project

5.1. Using OSELAS.BSP-Pengutronix-Generic

The OSELAS.BSP-Pengutronix-Generic BSP from Pengutronix is a good start since it can be used to try out *PTXdist* without real hardware¹. In addition, it doesn't need parts of *PTXdist* that currently don't work on Mac OS, see section 6.

5.1.1. Getting the BSP

Since the last release is from November 2011, I created a public GIT repository which contains a more up-to-date version of the BSP. Apart from the project migration to the new version, only minor fixes were made and the documentation [1] still fully applies.

To retrieve my forked version of the BSP, use

```
% git clone https://bitbucket.org/bwalle/oselas.bsp-pengutronix-generic.git
% git checkout -b macos origin/macos
```

Select the arm variant:

```
% ptxdist platform configs/arm-qemu-2011.11.0/platformconfig
```

and build the full image with

```
% ptxdist images
```

After that, the directory platform-versatilepb/images should contain following files:

kernelimage the Linux kernel

root.ext2 the ext2 root file system

hd.img a disk image that contains a partition table in addition to the file system

¹except from the computer on which *PTXdist* runs, of course

5.1.2. Building FSF GCC

Before building Qemu we have to download another toolchain. Mac OS ships with LLVM GCC (<http://www.llvm.org>). However, to successfully *run* Qemu we need FSF² GCC which is basically the “real” GCC as you know it from Linux. One of the reasons why Apple switched from FSF GCC to LLVM is licensing. Apple wants to avoid GPL in version 3.

If you have an older Mac OS installation that has been updated from time to time, you may have already an FSF GCC available as gcc-4.2. Otherwise, *MacPorts* makes it easy to install the last FSF GCC that Apple provided:

```
% sudo port install apple-gcc42
```

5.1.3. Building Qemu

Unfortunately, the *MacPorts* version of Qemu for ARM doesn’t work, at least it didn’t in my tests. However, the current³ GIT version does. So let’s just build our own Qemu.

Now retrieve the Qemu GIT tree with:

```
% git clone git://git.qemu.org/qemu.git
% cd qemu
```

We need one patch that is not (yet) mainline to apply. The patch is from *MacPorts*, but for convenience I added it to the BSP git repo. To apply it, just use

```
% git clone git://git.qemu.org/qemu.git
% patch -p0 -i \
  <path-to-bsp>/configs/arm-qemu-2011.11.0/patches ...
  .../patch-cocoa-uint16-redefined.diff
```

Configure and build Qemu⁴:

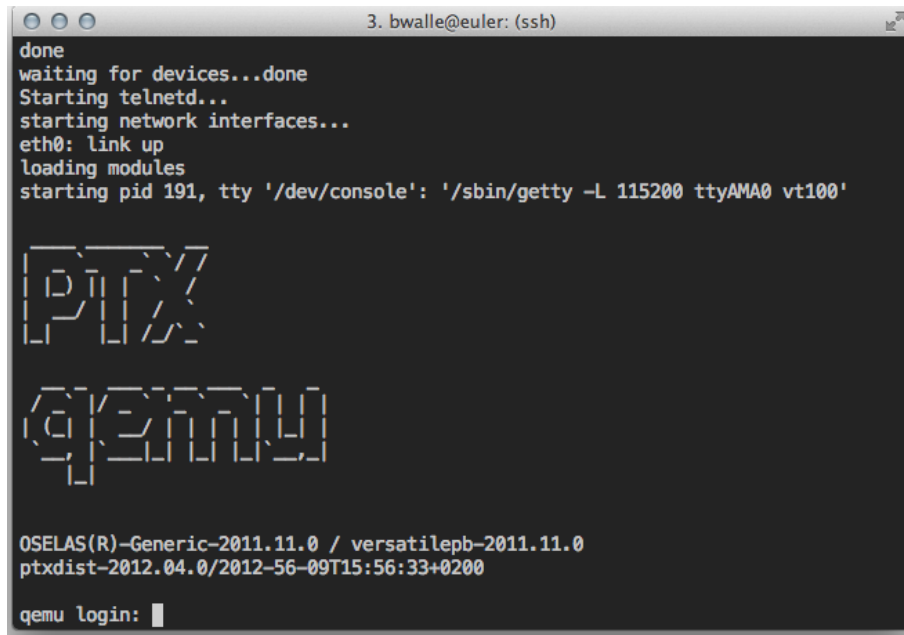
```
% ./configure --cc=gcc-apple-4.2 --host-cc=gcc-apple-4.2 \
  --target-list=arm-softmmu
% make -j$[2 * $(sysctl -n hw.ncpu)]
```

After a while, the file `./arm-softmmu/qemu-system-arm` should be ready.

5.1.4. Running the System

Change back to the BSP directory and execute

```
PATH=../qemu/arm-softmmu/:$PATH configs/arm-qemu-2011.11.0/run
```



```
3. bwallo@euler: (ssh)
done
waiting for devices...done
Starting telnetd...
starting network interfaces...
eth0: link up
loading modules
starting pid 191, tty '/dev/console': '/sbin/getty -L 115200 ttyAMA0 vt100'

PTXdist
qemu

OSELAS(R)-Generic-2011.11.0 / versatilepb-2011.11.0
ptxdist-2012.04.0/2012-56-09T15:56:33+0200
qemu login: █
```

Figure 5.1.: Qemu running an ARM kernel

If you have another directory layout, adjust the `$PATH` setting. After a while, you should see the login prompt as shown in figure 5.1. You should be able to login as root without password.

Now open another Terminal window and execute `telnet localhost 4444`. Even there you should be able to login as root. The Qemu network stack has been setup to forward the port 4444 the host to port 23 of the emulated system. Just look in the file `configs/arm-qemu-2011.11.0/run` to see how this was made.

5.2. Using real Hardware

Embedded Linux development is boring without hardware toys. You should be able to use *any* PTXdist-based BSP with Mac OS as long as you don't need something that is listed in section 6.

If you have a *Beagleboard* (<http://beagleboard.org/>) or a *BeagleBone* (<http://beagleboard.org/bone>), my BSP at <https://bitbucket.org/bwallo/ptxdist-arm-boards> may be a good start.

²Free Software Foundation

³7914cb3c738a03a5d5f7cb32c3768bc62eb1e944

⁴You may replace `gcc-apple-4.2` by `gcc-4.2` if you didn't use *MacPorts* to build that GCC

6. Limitations

This section should list what is currently known to not work.

6.1. Building UBI or JFFS2 images

Currently the *host-mtd-utils* don't compile. I already have something that works ready at <https://bitbucket.org/bwalle/mtd-utils>, but it's not yet integrated.

6.2. Linux kernel

Not every Linux kernel version can be compiled without patches. However, kernel 3.4 at least on ARM (tested OMAP3, Marvell Kirkwood and the “versatilepb” platform used in section 5.1) works.

Some patches you may need¹:

- `scripts/Kbuild.include`: Fix portability problem of `"echo -e"` (875de98623fa2b29f0cb19915fe3292ab6daa1cb)
- ARM: 7184/1: fix `$(CROSS_COMPILE)` prefix missing from `size` invocation (1ec332a3756a22405d2fbd5352e3afab556cb205)

6.3. Bootloader

As far as I know, Barebox (<http://www.barebox.org>) currently doesn't build while U-Boot (<http://www.denx.de/wiki/U-Boot>) works. Some U-Boot developers occasionally run Mac OS themselves.

¹all GIT hashes are for [git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git](http://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git)

7. Using a Mac for Embedded Linux Development

This section doesn't have anything to do with *PTXdist* in particular. Instead, it collects some hints that makes Linux development easier with a Mac.

7.1. Using a serial console

Macs don't have a RS-232 port, but PC Laptops also don't have one. Most USB-to-RS-232 adapters also work on Mac OS. The interfaces are named `/dev/tty.name`, for example `/dev/tty.usbserial-TIV90V1YA` or just `/dev/tty.usbserial`.

Most console-based Terminal programs can be also used on Mac OS. *screen*(1) is even shipped by default, but I prefer *picocom*(1) which can be installed using *MacPorts* and used like

```
picocom -b 115200 /dev/tty.usbserial-TIV90V1YA
```

To quit it, use C-a C-q. To send a break character which is important for using *Sysrq* [4], use C-a C-\ followed by the key that should sent to the kernel. If you can't remember, use h for *help*.

7.2. Exporting directories via NFS

For application development exporting the root file system on NFS rather than copying it to the target again and again, improves the turnaround speed a lot. Using Mac OS is surprisingly easy.

Just create a file `/etc/exports` (if it doesn't already exist) containing a line such as

```
/Volumes/Daten/devel -maproot=0 -network 192.168.0.0 -mask 255.255.255.0
```

The option `-maproot=0` is basically the same as `no_root_squash` on Linux. You can even map other users, see the manpage *exports*(5) for details.

Creating the file — at least if you get it initially right — should be sufficient. The NFS server automatically detects creation or modification of that file. However, one just can use `nfsd status` to check the status of the NFS server and `nfsd checkexports` to check the syntax after editing `/etc/exports`. Read *nfsd*(8) for more details.

Like on Linux, the *showmount*(8) command shows what you're exporting:

7. Using a Mac for Embedded Linux Development

```
% sudo showmount -e
Exports list on localhost:
/Volumes/Daten/devel          192.168.0.0
```

On the target side everything stays the same.

7.3. Mounting *ext2* partitions

When working with root file systems on SD cards it's quite useful if you cannot only copy images but also look at the files. Especially as most Macs have a builtin SD card reader.

While Mac OS has no kernel driver for *ext2*, like Linux Mac OS supports file systems in userspace. Just install *ext2fuse* with

```
% sudo port install ext2fuse
```

and then you're able to mount an *ext2* or *ext3* partition using

```
% diskutil list # find out the name of the disk  
% ext2fuse /dev/disk1s2 ~/mnt
```

The important thing is that *ext2fuse* has to be executed as the same user as the one that works with the file. So the mountpoint must be accessible as the user — that's why I'm not using a directory below /Volumes.

A. About this Document

This document has been typeset with \LaTeX . The sources of the document can be downloaded at <https://bitbucket.org/bwalle/ptxdist-macos-doc>.

The canonical location for the PDF is http://bwalle.de/docs/ptxdist_mac.pdf. Please send any feedback to me via email bernhard@bwalle.de.

B. Change History

Date	Modification
2012-04-21	Add hint to disable <i>Spotlight</i> indexing.
2012-04-20	Ad description about case insensitive file system. Add <i>dialog</i> as optional dependency.

Bibliography

- [1] **How to become a PTXdist Guru,**
<http://www.ptxdist.org/software/ptxdist/appnotes/OSELAS.BSP-Pengutronix-Generic-arm-Quickstart.pdf>
- [2] **Installing PTXdist,**
http://www.ptxdist.org/software/ptxdist/appnotes/AppNote_InstallingPtxdist.pdf
- [3] **MacPorts Guide,**
MARK DULING, DR. MICHAEL A MAIBAUM, WILL BARTON,
<http://guide.macports.org/>
- [4] **Linux Magic System Request Key Hacks**
<http://kernel.org/doc/Documentation/sysrq.txt>