Installing Scratchbox

Valtteri Rahkonen

valtteri.rahkonen@movial.fi

Installing Scratchbox

by Valtteri Rahkonen

Revision history

Version:	Author:	Description:
2005-03-09	Savola	Added umask instruction
2004-05-03	Rahkonen	Updated sbrsh README and environment variables
2004-04-26	Rahkonen	Fixed target creting, added requirements to introduction and added getting help
2004-04-19	Rahkonen	Added Debian devkit testing section
2004-04-15	Rahkonen	Layout and grammar fixes
2004-04-13	Rahkonen	New layout and added references
2004-04-07	Rahkonen	Initial version

Table of Contents

Chapter 1. Introduction

Scratchbox is a cross-compilation toolkit designed to make embedded Linux application development easier. It provides a full set of tools to integrate and cross-compile an entire Linux distribution. Scratchbox supports cross-compiling for ARM and PowerPC targets. Especially Debian is supported, but Scratchbox has also been used to cross-compile e.g. Slackware for ARM.

Scratchbox provides a sandboxed build environment which offers a controlled set of tools and utilities needed for cross-compilation. It is an environment in which it is easy to assure that the intended versions of libraries, headers and other similiar files are used during the build. Most of the higher level software built using GNU Autotools do not cross-compile well in their as-is form, Scratchbox solves this problem by allowing the small test programs (used by the configure script to test for availability of features in the environment) to run transparently either using an emulator or through Sbrsh protocol between Scratchbox and actual target device. In practice software configuration and building using Scratchbox is quite identical to how it's done for the desktop.

Scratchbox has been designed to allow multiple application developers work simultaneously on a single host machine. Each developer has his private user account, and all configuration is developer-specific.

This document describes Scratchbox install procedure and how Scratchbox can be used to cross-compile applications to ARM target platform.

1.1. Hardware requirements for host

Requiremenst for using Scratchbox are:

- Linux distribution installed on the host.
  - Debian GNU/Linux is recommended.
- x86 platform.
  - For multiple users using Scratchbox on the same host a dual CPU machine or HyperThreading capable processor is recommended.
- 512 MB of memory.
  - For multiple users using Scratchbox on the same host atleast 1 GB is recommended.
- 3 GB of hard disc space when using rootstrap.
  - For multiple users using Scrathbox on the same host atleast 1.5 GB more is required for each additional user.
- Working networking environment.

- Working NFS environment.
- Your kernel has the /proc/sys/fs/binfmt_misc/ feature which is required by Scratchbox QEMU emulation or Sbrsh functionality. It is on by default on most Linux distributions (not on RedHat Enterprise Linux 3).

1.2. Hardware requirements for ARM device

When using QEMU specific target platform is not needed. However, if Sbrsh is used with ARM target device following pre-requirements have to be met:

- A Linux ARM device with networking. An iPAQ is recommended as it has known stable Linux support. Handhels.org has information about Linux support for iPAQs [4]. Scratchbox reference target device is Compaq iPAQ H3600 series PDA. Scratchbox has been tested and verified to work with the following devices:
  - Compaq iPAQ H3630
  - Compaq iPAQ H3870
- Working networking environment (USB cradle or PCMCIA sleeve with an ethernet card).
- You have a suitable ARM distribution installed on the ARM device. The recommended Linux distribution is Familiar 0.7.2 with GPE2. See Familiar on for more information and installation instructions [5].
- Network between your host machine and the ARM device has been setup and works correctly, see: Support distribution installation instructions or How to setup iPAQ networking in Familiar .
- You've created an ARM target for the Scratchbox as described in Section 2.4. You can create new target for Sbrsh if you like to keep QEMU target. However switching between QEMU emulation and Sbrsh is simple.

Chapter 2. Installing Scratchbox

2.1. Scratchbox packages

Following packages are required:

- scratchbox-core - development environment, tools and host-gcc compiler
- scratchbox-libs - libraries required by tools and compilers

Optional packages:

- scratchbox-doctools - document generation tools
- scratchbox-devkit-debian - environment and tools for building Debian packages
- scratchbox-devicetools - prebuilt binaries for target devices

One or more:

- scratchbox-toolchain-arm-glibc
- scratchbox-toolchain-arm-uclibc
- scratchbox-toolchain-i386-uclibc
- scratchbox-toolchain-i686-glibc
- scratchbox-toolchain-powerpc-glibc

Scratchbox packages are available for Debian and systems that use RPM packaging format. For systems that do not support these formats Scratchbox is also available as tarball.

2.2. Installing Scratchbox on host

2.2.1. Installing Scratchbox on Debian GNU/Linux

You will need root privileges for this part of the Scratchbox installation.

1. Add this line to the /etc/apt/sources.list file:

   ```
   deb http://scratchbox.org/debian ./
   ```

2. Update the package list with command:

   ```
   # apt-get update
   ```

3. Install packages:

   ```
   # apt-get install <package names>
   ```

   If you want to install everything, simply install the ’scratchbox’ package.

   > **Note:** If ’/’ directory does not contain enough space for Scratchbox it can be installed to some other partion by creating a symbolic link from ’/scratchbox’ to desired place with command: **ln -s /opt/sb /scratchbox**.

4. After downloading Scratchbox will be unpacked to ’/scratchbox’ directory and the installation procedure will ask you some questions about the group and user accounts. Default group to Scratchbox users is ’sbox’. Group can be renamed but default should be fine unless you have LDAP or similar network authentication scheme. If network authentication is used using an existing group is recommended. Users that will be using Scratchbox should be selected from a user list that install offers. Install script will automatically include users to sbox group, create user directories under ’/scratchbox/users directory and mount several directories (/dev, /proc, /tmp) under user directory. Users can be added later with command:

   ```
   # dpkg-reconfigure scratchbox-core
   ```

This is all that needs to be done for basic Scratchbox functionality. To be able to cross-compile more complex software you need to enable the Sbrsh feature (see Chapter 3).

2.2.2. Installing Scratchbox on other Linux distributions

You need root privileges for this part of the Scratchbox installation.

1. Obtain necessary packages from Scratchbox download area [2].
2. Ensure that your umask setting allows normal users to read and execute the extracted files.

   ```
   # umask 022
   ```

3. Uncompress tar-balls to the / directory (for each tarball):

   ```
   # tar zxf <package> -C /
   ```

> **Note:** If '/' directory does not contain enough space for Scratchbox it can be installed to some other partion by creating a symbolic link from '/scratchbox' to desired place with command: **ln -s /opt/sb /scratchbox**.

4. After extraction configure Scratchbox with following command:

   ```
   # /scratchbox/run_me_first.sh
   ```

   Answer questions (defaults should be fine). This creates 'sbox' user group and setup Scratchbox.

5. Add users to the Scratchbox with command:

   ```
   # /scratchbox/sbin/sbox_adduser username
   ```

   This adds the user to 'sbox' user group, creates the Scratchbox user directory under '/scratcbox/users' directory and mounts several directories (/dev, /proc, /tmp) under user directory.

2.3. Starting Scratchbox environment

1. If you were logged into the Scratchbox machine before you were added as a Scratchbox user, you may need to re-login to your machine, so that you get 'sbox' group privileges needed for running Scratchbox. You can check this by running the following command:

   ```
   $ groups
   ```

   If it prints out sbox group name, you're ready to run the Scratchbox.

2. Start Scratchbox with command:

   ```
   $ /scratchbox/login
   ```

> **Note:** If you have installed Scratchbox from Debian packages or RPM's you can use command **/usr/bin/scratchbox** to start Scratchbox session.

When you run this for the first time it copies the terminfo terminal capability database to your home directory inside Scratchbox and does some checks and initializations before starting the Scratchbox sandbox.

Note: Because Scratchbox is fully self-contained sandbox, it doesn't inherit anything from the outside system except for the environment variables. So, now you can copy your configuration files from your normal home directory into your home directory inside the Scratchbox (/scratchbox/users/$USER/$HOME/): ~/.bashrc (Scratchbox uses bash shell so if you use bash on host its configuration can be imported from host), ~/.bash_profile, ~/.inputrc, ~/.cvspass etc.

2.4. Creating cross-compilation target for ARM

Cross-compilation target can be created with the following commands:

1. New target can be created with command:

   ```
   [sbox-HOST: ~] > sbox-config --create-target
   ```

   First it will ask target name that will be used identifying target:

   ```
   Enter target name: MYTARGET
   ```

   Here MYTARGET was used as a target name. After targets name has been entered setup will ask wich compiler will be used to compile programs:

   ```
   Available compilers:
       0) arm-gcc-3.3.2-uclibc-snapshot-20040229
       1) arm-linux-gcc-3.3_3.3.2ds5-glibc-2.3.2.ds1
       2) host-gcc
       3) i386-gcc-3.3.2-uclibc-snapshot-20040229
       4) i686-linux-gcc-3.3_3.3.2ds5-glibc-2.3.2.ds1
       5) powerpc-linux-gcc-3.3_3.3.2ds5-glibc-2.3.2.ds1

   Enter compiler number: 1
   ```

   Here gcc compiler version 3.3 for ARM was selected. After compiler selection CPU-transparency method is selected:

   ```
   Available CPU-transparency methods:
       sbrsh
       qemu-arm
       qemu-ppc
   ```

```
Enter method name (qemu-arm):
```

Default method ’qemu-arm’ is sufficient for this example target and it can be selected by pressing enter. As a last question setup will ask which devkits will be used. Here we will select default option ’none’ by pressing enter:

```
Available devkits:
    debian

Enter list of devkit names (none):
```

2. Newly created target can be selected with command:

   ```
   [sbox-HOST: ~]> sbox-config --select-target=MYTARGET
   ```

3. Newly created target compiler needs to be configured with following command:

   ```
   [sbox-MYTARGET: ~] > sbox-config --copy-clibrary
   ```

4. Libfakeroot enviroment can be configured with following command:

   ```
   [sbox-MYTARGET: ~] > sbox-config --copy-libfakeroot
   ```

   Note: Configuring libfakeroot is optional if you do not need fakeroot or you are planning to use rootstrap.

   Note: Targets can be selected inside Scratchbox command **sbox-config --select-target=TARGETNAME**. User may have multiple targets for different c-libraries or different target platforms.

Scratchbox is now ready for cross-compilation with QEMU emulator. QEMU fits for simple use but Sbrsh with an actual target device should be used for more advanced requirements (see Chapter 3).

2.5. Testing installation

Scratchbox is now ready for cross-compiling and it can be tested with a simple command line program. Our example program uses GNU Autotools for detecting compilation environment. Scrathbox example program can be compiled with following steps:

1. Untar Scratchbox test programs source from '/scratchbox/packages' directory in user's home directory.

   ```
   [sbox-MYTARGET: ~] > tar zxf /scratchbox/packages/hello-world.tar.gz
   ```

   This will create directory 'hello-world' that contains test program sources to users home directory.

2. Change directory to directory scratchbox-tests/hello-world.

   ```
   [sbox-MYTARGET: ~] > cd hello-world
   ```

3. Configure example and generate makefile with following command:

   ```
   [sbox-MYTARGET: ~/hello-world] > ./autogen.sh
   ```

   Autogen.sh runs necessary Autotools programs and executes configure script.

4. Our example is now ready for cross-compilation. It can be compiled with following command:

   ```
   [sbox-MYTARGET: ~/hello-world] > make
   ```

5. This make command should generate ARM target executable named 'hello'. This can be verified with command:

   ```
   [sbox-MYTARGET: ~/hello-world] > file hello
   ```

   Output should state that hello is executable for ARM:

   ```
   hello: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux
   2.0.0, dynamically linked (uses shared libs), not stripped
   ```

6. Program can be run with following command:

   ```
   [sbox-MYTARGET: ~/hello-world] > ./hello
   ```

   Program should print text ’Hello World!’ on console.

If support for more complicated programs (programs that uses higher level libraries GUI programs for example) is needed libraries that they use should be installed to Scratchbox. One option is to compile them from sources and another is to use prebuild packages. For Scratchbox there is a Debian based rootstrap that contains prebuilt ARM libraries and all required include files (see Section 4.3).

Chapter 3. Setting up Sbrsh

Sbrsh runs the configure scripts test programs on a networked device with the same CPU as the cross-compilation target device, in a way that is transparent to the configuration system. Some build systems also benefit from it when they attempt to execute a target-binary which is used to generate certain data files. Running programs on actual target device is more reliable than emulating specific target device because emulator might not support all required features.

3.1. Setting up NFS environment

Sbrsh works by sharing your home (build) and target directories in your Scratchbox with the ARM device. This is done using NFS, so you have to setup your machine to export the required directories with NFS. NFS setup has to be done as root.

Note: NFS is inherently insecure so you should run it only on trusted networks.

1. On Debian ’nfs-common’, ’nfs-kernel-server’ and ’portmap’ packages are needed. On RedHat you need ’nfs-utils’ and ’portmap’ packages.
2. Add NFS exports for Scratchbox users home and ARM-target directories to the ’/etc/exports ’ file on the host machine The export lines should look something like this:

   ```
   /scratchbox/users/$USER/targets/MYTARGET <IP>(rw,all_squash,anonuid=<UID>,
   anongid=<GROUPS>)

   /scratchbox/users/$USER/home <IP>(rw,all_squash,anonuid=<UID>,anongid=<GROUPS>)
   ```

   Where <IP> is replaced with the IP-address of your ARM device, <UID> with user’s user-id and <GROUPS> with the default group id for that user (last two values you can see by echoing those variables from shell as that user).

3. Now NFS server can be started. In Debian NFS is started with command:

   ```
   # /etc/init.d/nfs-kernel-server start
   ```

   And in RedHat NFS server:

   ```
   # service nfs start
   ```

This is all that is needed for ARM device to mount necessary directories from Scratchbox. More information on NFS is in [6]. However following commands might be useful for NFS:

- If you make changes to the /etc/exports file, you need to tell the NFS server to reread the /etc/exports file:

  ```
  # /usr/sbin/exportfs
  ```

  Alternatively NFS server can be restarted. In Debian NFS is restarted with command:

  ```
  # /etc/init.d/nfs-kernel-server restart
  ```

  In RedHat NFS is restarted with command:

  ```
  # service nfs restart
  ```

- If you don't want to export filesystems anymore, run this:

  ```
  # /usr/sbin/exportfs -uav
  ```

  Or NFS could be stopped. In Debian NFS is stopped with command:

  ```
  # /etc/init.d/nfs-kernel-server stop
  ```

  And in Redhat with command:

  ```
  # service nfs stop
  ```

3.2. Configuring Sbrsh

Sbrsh needs to be configurated in both Scratchbox and ARM target device (Scratchbox side needs to know the target location and target device needs to know the mount points locations). Configuration can be done with following steps:

1. Copy sbrshd to target device.

2. The daemon configuration file '/home/$USER/.sbrshd' in target device lists all known client IPs and passwords. Each user has his own .sbrshd file in his home directory. # is a comment character. The layout is:

   ```
   <IP> <PASSWORD>
   ```

   Where <IP> is Scratchbox host IP-address and <PASSWORD> is same password in both sbrshd daemon configuration (in target device) and in sbrsh client configuration (in Scratchbox). Example configuration:

   ```
   1.2.3.4 killr0y
   ```

3. Run sbrshd in target device with following command:

   ```
   # sbrshd
   ```

   More info about options and sbrshd usage is available in [3].

4. The client configuration file ('/home/$USER/.sbrsh' inside Scratchbox) lists all known targets. The first line of a target block must not contain whitespaces before the name of the target. The subsequent lines must be indented. A '#' character is a comment. The layout of the first line:

   ```
   <target> [username@]<ip>[:port] <password>
   ```

   The subsequent lines define the mounts needed by the target (type is either 'nfs' or 'bind'):

   ```
   <type> <share/path> <point> [nfs options]
   ```

   Here is an example configuration:

   ```
   MYTARGET john@172.16.6.76 killr0y
     nfs   1.2.3.4:/scratchbox/users/john/targets/MYTARGET / rw,nolock,noac
     nfs   1.2.3.4:/scratchbox/users/john/home /home rw,nolock,noac
     bind  /dev      /dev
     bind  /dev/pts  /dev/pts
     bind  /proc     /proc
     bind  /tmp      /tmp
   ```

5. After sbrsh is configured it can be activated by modifying the SBOX_CPUTRANSPARENCY_METHOD environment variable from qemu-arm to sbrsh with following command:

```
export SBOX_CPUTRANSPARENCY_METHOD=sbrsh
```

Re-enabling QEMU emulation can be done with command:

```
export SBOX_CPUTRANSPARENCY_METHOD=qemu-arm.
```

Note: These export commands are not permanent because they will be cleared when Scratchbox session is restarted. Thus this is feasible only for testing purposes. Instead of using these export commands new target should be created for sbrsh (see Section 2.4).

3.3. Testing Sbrsh

Sbrsh can be tested by repeating same steps that were done in testing installation section (see Section 2.5). This time only difference is that program is executed on target platform instead of being emulated with QEMU emulator.

Chapter 4. Setting up Debian environment

This section describes how to set up Debian environment inside Scratchbox. Debian devkit offers tools for creating Debian packages and Debian rootstrap offers necessary prebuild ARM libraries for obtaining necessary development files.

4.1. Installing Debian devkit

Installing Debian devkit needs to be done as root to obtain necessary privileges to write '/' directory. If Scratchbox is installed on Debian GNU/Linux system then Debian devkit can be installed with following command:

```
# apt-get install scratchbox-devkit-debian
```

Otherwise you need to obtain Debian devkit package and untar it under '/'. Package can be obtained from [2].

After devkit is installed a target that uses Debian devkit must be created as in Section 2.4 and by entering 'debian' to devkit list.

Building Debian packages require root privileges or working fakeroot environment. In Scratchbox fakeroot environment can be configured with command:

```
[sbox-MYTARGET: ~] > sbox-config --copy-libfakeroot
```

4.2. Testing Debian devkit

Debian devkit is tested by building some Debian packages. In this Section we will build and install ncurses package. Ncurses package is easy to build because it does not have building dependencies other than working compiler and C library (provided by toolchain). Packages that require some other libraries or packages to build can be build after installing Debian rootstrap (see Section 4.3).

1. Select the previously created target that was created with Debian devkit (here we assume that MYTARGET was created with Debian devkit):

   ```
   [sbox-HOST: ~] > sbox-config -st MYTARGET
   ```

2. Update the package repository:

   ```
   [sbox-MYTARGET: ~] > fakeroot apt-get update
   ```

3. Get the ncurses source package:

   ```
   [sbox-MYTARGET: ~] > apt-get source ncurses
   ```

4. Change directory to ncurses source directory:

   ```
   [sbox-MYTARGET: ~] > cd ncurses-5.3.20030719
   ```

5. Ncurses package cannot be build straight away. Reason for this is that if you are not using rootstrap dpkg package is not present at the package database. One of ncurses packages (ncurses-bin) checks that dpkg is installed in its preinstall script.

   This can be avoided by executing the following command before running dpkg-buildpackage:

   ```
   [sbox-MYTARGET: ~/ncurses-5.3.20030719] > sed -i ’s/^dpkg --assert-support-predepends$//
   debian/ncurses-bin.preinst
   ```

6. Build the ncurses binary packages for the target architecture:

   ```
   [sbox-MYTARGET: ~/ncurses-5.3.20030719] > dpkg-buildpackage -b -d -rfakeroot
   ```

7. Change directory back to home directory (builded Debian packages are located there):

   ```
   [sbox-MYTARGET: ~/ncurses-5.3.20030719] > cd ..
   ```

8. Install the libncurses and ncurses-bin packages that were created:

```
[sbox-MYTARGET: ~] > fakeroot dpkg -i libncurses5_5.3.20030719-4_arm.deb
[sbox-MYTARGET: ~] > fakeroot dpkg -i ncurses-bin_5.3.20030719-4_arm.deb
```

9. Check that /usr/bin/tic was installed and that it is an ARM binary

```
[sbox-MYTARGET: ~] > file /usr/bin/tic
/usr/bin/tic: ELF 32-bit LSB executable, ARM, version 1 (ARM), for
GNU/Linux 2.0.0, dynamically linked (uses shared libs), stripped
```

10. Run the ’tic -V’ command and check that it was executed via Sbrsh:

```
[sbox-MYTARGET: ~] > tic -V
ncurses 5.3.20030719
[sbox-MYTARGET: ~] > tail /tmp/cputransp_$USER.log
```

4.3. Installing Debian rootstrap

Rootstrap contains prebuilt ARM libraries and development files. It provides one way of installing the required development libraries. It gives you a ’clean’ target installation within Scratchbox. To install a rootstrap follow the instructions below:

1. Copy or download the rootstrap tarball to ’/scratchbox/packages/’ (this needs to be done as root because normal users do not have permissions to write on ’/scratchbox/packages/’ directory)
2. Login to Scratchbox

```
$ /scratchbox/login
```

3. Select the MYTARGET target (if not already selected) with following command:

```
[sbox-HOST: ~] > sbox-config -st MYTARGET
```

4. Extract the rootstrap into the MYTARGET target with:

```
[sbox-MYTARGET: ~] > sbox-config --extract-rootstrap
rootstrap_<version>.tar.gz
```

5. To check updates and upgrading in rootstrap is a simple operation. Rootstrap can be updated with commands:

```
[sbox-MYTARGET: ~] > fakeroot apt-get update
[sbox-MYTARGET: ~] > fakeroot apt-get dist-upgrade
```

Note: It is very important that build environment libraries and include files match those on the target device. Any discrepancies can lead to errors that are very hard to track down.

4.4. Building a GUI application

After rootstrap is installed or required libraries are installed to Scratchbox by other means more complicated programs can be cross-compiled inside Scratchbox environment. Our example program is simple ’Hello World!’ dialog that uses the GTK+ toolkit. It can be build using the following commands:

1. Untar sources for Scratchbox test program hello-world-gtk (from ’/scratchbox/packages’ directory).

   ```
   [sbox-MYTARGET: ~/] > tar zxf /scratchbox/packages/hello-world-gtk.tar.gz
   ```

2. Change directory to directory hello-world-gtk.
3. Configure program and create makefile script with following command:

   ```
   [sbox-MYTARGET: ~/hello-world-gtk] > ./autogen.sh
   ```

   Autogen script will execute necessary autotool commands and it also runs configure script automatically.

4. Our example is now ready for cross-compilation. It can be compiled with following command:

   ```
   [sbox-MYTARGET: ~/hello-world-gtk] > make
   ```

5. This make command should generate ARM target executable named ’hello-gtk’. This can be verified with command:

   ```
   [sbox-MYTARGET: ~/hello-world-gtk] > file hello-gtk
   ```

   Output should state that hello-gtk is a executable for ARM:

   ```
   hello: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux
   ```

```
2.0.0, dynamically linked (uses shared libs), not stripped
```

6. Before program can be executed the DISPLAY environment variable should be set to point desired X-server with following command:

   ```
   [sbox-MYTARGET: ~/hello-world-gtk] > export
   DISPLAY=ipaddress:displaynumber
   ```

   Where ipaddress is address to desired host and displaynumber is the host's display (display is typically 0). Host's X-server should be configured so that connections from outside are allowed. This can be done with command:

   ```
   # xhost +
   ```

   > **Note:** Allowing remote programs to use host's display is considered to be unsafe. This should be used only in trusted networks.

7. Program can be run with following command:

   ```
   [sbox-MYTARGET: ~/hello-world-gtk] > ./hello-gtk
   ```

   Program should start a dialog that shows 'Hello World!' text and contains a 'Close' button.

Chapter 5. Scratchbox maintenance

5.1. Starting Scratchbox system

If Scratchbox was installed from Debian or RPM packages correct init script should have been installed and Scratchbox should start normally when the system is restarted. However, if you installed Scratchbox from tarballs then rebooting your machine will clear away all the mounts that Scratchbox has done and CPU transparency registration for binfmt_misc. So, to get your Scratchbox working again after reboot, you have to run as root the command:

```
# /scratchbox/sbin/sbox_ctl start
```

This registers the CPU transparency client and mounts directories inside the sandbox. Alternatively you can add 'sbox_ctl' command as an init script to your /etc/init.d/ runlevel directories.

5.2. Upgrading Scratchbox

If Scratchbox is installed on Debian system it can be easily upgraded with following commands:

```
# apt-get update
# apt-get dist-upgrade
```

Note: Debian will replace toolchains with newer ones. If compiler name inside toolchain has changed users may need to create new target with a new compiler (see Section 2.4).

If Scratchbox is installed on other system than Debian you need to obtain newer packages and untar them in '/'.

Note: Upgrading Scratchbox from tarballs will not replace older toolchains and they are still being used if user has created target with them. If a new toolchain is needed users should create a new target (see Section 2.4).

5.3. Uninstalling Scratchbox

Scratchbox uninstalling needs root privileges so you need to be root when uninstalling.

5.3.1. Uninstalling Scratchbox on Debian GNU/Linux

If Scratchbox was installed from Debian packages all Scratchbox packages can be removed from the system with following command:

```
# apt-get remove scratchbox-libs scratchbox-devicetools
```

Note: All packages except scratchbox-devicetools depend on scratchbox-libs so removing those two should be enough.

Now the /scratchbox directory should be empty except for the user's directory. If you want to remove the user directories, double-check that there are no active mounts under /scratchbox (or if it's a symlink, the path it points to) with the 'mount' command. After that you can just write:

```
# rm -rf /scratchbox/users
```

5.3.2. Uninstalling Scratchbox on other Linux Distributions

If Scratchbox was installed from tarballs removing it is somewhat more complicated because Scratchbox mounts some directories from host system to user's sandbox environment. As a result of this scratchbox can't be removed simply using 'rm -r'. You would then remove also files from directories that are mounted under user directories (e.g. /tmp dir contains X11 socket, so you would need to re-login to X).

There's a utility called '/scratchbox/sbin/sbox_umount_all' which you have to use before removing the Scratchbox or any of it's user directories. You can use the 'mount' command to check if something is still mounted under the /scratchbox directory. After doing the unmounts, you can remove the Scratchbox directory /scratchbox with command:

```
rm -r /scratchbox
```

Note: This command removes user directories also. It is recommended that you copy any valuable data from user’s home directories before uninstalling the Scratchbox.

Chapter 6. Additional information

More information and support can be obtained through following channels:

- Scratchbox website [1].
- Scratchbox IRC-channel on ircnet: #scratchbox.
- Scratchbox mailing list (see[1] for more information).
- Commercial support is provided by Movial [7].

References

[1] *Scratchbox website (http://www.scratchbox.org/)* .

[2] *Scratchbox Download Area (http://www.scratchbox.org/index.html?id=4)* .

[3] *Scratchbox Devicetools document (http://www.scratchbox.org/docs/devicetools/index.html)* , Timo Savola.

[4] *How to Run Linux on iPAQ Handhelds (http://www.handhelds.org/handhelds-faq/handhelds-faq.html)* , Jamey Hicks.

[5] *the Familiar Project (http://familiar.handhelds.org/)* .

[6] *Linux NFS-HOWTO (http://www.tldp.org/HOWTO/NFS-HOWTO/index.html)* , Tavis Barr, Nicolai Langfeldt, Seth Vidal, Tom McNeal.

[7] *Movial website (http://www.movial.fi/en/products/Scratchbox/)* .

Appendix A. Scratchbox environment variables