Scripts to automatically create a Debian rootfs image and/or a complete, bootable SD-card.

README.md

Gnublin_Debian

Scripts to automatically create a Debian rootfs image and/or a complete, bootable SD-card.

Author: Ingmar Klein (ingmar.klein@hs-augsburg.de) Created in scope of the "Embedded Linux" lecture, held by Professor Hubert Hoegl, at the University of Applied Sciences Augsburg, 2012

README: Gnublin Debian scripts

This Readme should explain how to use the debian build-scripts. It also explains the settings and their meaning.

Overview:

1. Host System Requirements

You need a machine with at least Debian Squeeze, or Ubuntu 10.04. . Default installs! If you did a minimal install of something similar, you will probably have to install a few extra packages, before being able to run the script. Enough disk space for the build process (~2GB free space) and maybe 512MB RAM. Internet connection is mandatory for this script (on the host machine, not on the gnublin). No internet connection equals no build process! You will need **root priviliges**! Either through the sudo package and the corresponding sudoers file, or by running the script as root.

2. Files

- **build_debian_system.sh** : Main script file, running the functions defined in the "build_functions.sh". This is what you run in shell.
- **build_funtions.sh** : As already mentioned, this includes the main build and some helper funtions that are used in the "build_debian_system.sh"
- **general_settings.sh**: File for all settings and configurations. Here you define what the output of the scripts will look like.

3. Settings

ATTENTION:
You must (!) edit the *"general_settings.sh"* to fit your host system and your needs, before running the main script!

Most of the settings are either self-explanatory, or already commented in the file itself. I'll now list the settings that you absolutely must change/edit/check:

- **host_os**: Either Debian or Ubuntu
- **output_dir_base**: This sets the local path for all the files generated by the script
- **nameserver_addr**: Without the correct setting for your network environment, the build process will most likely fail.
- **use_udev**: For the 8MB Gnublin, say "no" here, whereas the 32MB version of Gnublin is fine running udev
- bootloader_bin_path : This setting is the path of the bootlaoder binary. It belongs to the setting below (bootlaoder_bin_name) and DOES NOT NEED to be edited. But of course you can do so.
- **bootloader_bin_name**: This is the name of the bootloader binary to download. 'apex_8MB.bin' for the 8MB Gnublin and 'apex_32MB.bin' for the 32MB version! Make sure to set the right one for your board.

All other settings are optional and CAN, but don't necessarily need to be changed!

Note: The variable ***"additional_packages"*** is just a list of packages that should be installed. So, browse the debian package list (http://packages.debian.org) and add the packages that you want to the list. These will then be included in the resulting rootfs, if nothing goes wrong.

4. Running the script

Running the script in full mode, utilizing all functions, is as easy as doing the following:

- Make sure that you downloaded/cloned all 3 script files (or the complete package) via git and edited the "general_settings.sh".
- Run `chmod +x build_debian_system.sh` if needed, to make the file executable.
- Run the script itself by typing `sudo ./build_debian_system.sh` in a terminal window (NOT in a virtual console!). Of course you can also run the script as root directly, not using sudo.
- Then follow the instructions on the screen! If nothing goes wrong, no user-input should be required until choosing the SD-Card device is necessary (if you did set the option to create one!).
- The output should be in form of a compressed archive file (.tar.gz or .tar.bz2) containing the rootfs and kernel. The filename will be like set in "general_settings.sh".
- If you set the option to build a SD-Card, you will be asked to specify the correct device name for the SD-card, which then will be subsequently partitioned, formatted and filled with bootloader, rootfs and kernel.

5. Running the script with call parameters (special functions)

The script now accepts two main call parameters. These are **"build"** and **"install"**.

Descriptions:

build : This only runs the script's functions to build a new rootfs including kernel, but does not create a bootable SD-card. It is meant for those creating rootfs-archives for others to use, with known-to-work configurations.

install : Opposite of "build"! It will just download a prebuilt rootfs-archive + bootloader and will create a bootable SD-card using these files. This is meant for people not familiar with the build specifics who just want to create a SD-card with known-to-work rootfs configurations. The "install" parameter, however REQUIRES a valid second parameter, to work properly. This can either be "default", which then uses "default_rootfs_package_path" and "default_rootfs_package_name" from the "general_settings.sh" configuration file. Or you can pass the complete path+filename of a compressed (.tar.gz or .tar.bz2) rootfs-archive of your choice on to the script. The archive can be located online (for example 'http://www.tester.com/rootfs.tar.bz2') or on your local storage (for example '/home/tester/rootfs.tar.bz2'). The script should be able to handle both cases just fine.

Usage examples:

`sudo ./build_debian_system.sh --build` will run only the scripts build funtions to create a rootfs-archive according to your settings.

`sudo ./build_debian_system.sh --install default` will create a bootable SD-card containing the default-rootfs and bootloader specified in the "general_settings.sh"

`sudo ./build_debian_system.sh --install http://www.tester.com/rootfs.tar.bz2` will download the named rootfs-archive and create a bootable SD-card with it (bootloader as set per "general_settings.sh")

`sudo ./build_debian_system.sh --install http://www.tester.com/rootfs.tar.bz2 --bootloader http://www.tester.com/apex_32MB.bin` will download the named rootfs-archive and create a bootable SD-card with the specified bootloader

Any wrong parameter leads to an error message and a short help text, explaining the valid usage options.

6. Log-Files and further information

There are a number of log files that get created at run time. The main log file is located in the "output_dir"-folder. This log file includes all main points in the script. Some detail information gets logged elsewhere, but that can be seen in the script code.

Another important file is the **installed_packages.txt** in the root folder of the compressed archive. As the name suggests, this file contains a list (created by running dpkg -l) of all packages, included in that very rootfs.

7. Logging In

The system gets built with a standard root account with password. The initial login data is the following:

User: **root**
Password: **root**

If you want to change the root password, just run the command "passwd root" and follow the instructions. If you want to add a normal user account, run "adduser 'username' ", obviously with the name you want your user to have.

ATTENTION:

There is a Bug that causes the system to ask you to change the password on each system start, if the system time wasn't set when invoking passwd! Make sure that the system time is set to something else than 01/01/1970, if you intend to change the password!

HAVE FUN!