English • Deutsch • Español • Français • Italiano • ??? • Polski • Português • ??????? • Svenska • ???(????)? • ???(??)? •

Most of the time you don't really need to re-build DD-WRT to make your desired changes. Now, this process has been made easy with the Firmware Modification Kit. This kit gives the user the ability to make changes to a firmware image without recompiling the firmware sources. It works by extracting the firmware into its component parts, then extracting the squashfs-lzma based file system (other file systems can be supported with tweaks). The user can then make modifications to the extracted file system, then rebuild the firmware image.

Features include:

- add initialization scripts
- install new packages (remember, packages pre-installed are compressed much better)
- extend web UI
- remove un-needed packages
- mix-and-match packages from various DD-WRT variants

WARNING: Due to abuse by those re-branding DD-WRT and selling it, or pre-flashed routers with it on eBay, builds dated later than 08/04/2006 have some protections against re-branding the web UI.

Contents

Requirements

I've worked hard to make sure the requirements are low enough for anyone to use this kit. However, to reduce distribution costs and better represent the multi-platform capabilities of this kit, the binary tools in the kit do automatically build themselves the first time you run one of the scripts. Any machine set up to build anything C or C++ on the machine will probably be ready. The few requirements are all basic items that everyone has easy access to:

1. A compatible platform. The following are specifically supported (or specifically unsupported):
    1. LINUX
    2. OS X
    3. CYGWIN **alpha stage testing (!! built firmwares may not boot !!)**
2. GNU C (gcc)
3. GNU C++ (g++)
4. Standard C runtime library development
5. Standard C++ runtime library development
6. GNU make
7. TAR and GZIP (optional, used by ipkg_install.sh)

For OS X, installing the Mac OS X Developer Tools will supply the needed GNU C and C++ compilers and make utility.

To install/update most of them at once on a x32 Debian based distribution (I.E. Debian Lenny, Ubuntu, Linux Mint, etc), issue this command in a terminal:

```
sudo aptitude install gcc g++ binutils patch bzip2 flex bison make gettext unzip  zlib1g-dev l
```

Compatibility

Any cybertan style firmware image format that uses squashfs-lzma as a root file system should work fine. Firmware that uses regular squashfs or other file systems (i.e. cramfs) will work if the steps that extract the file system and rebuild the file system are tweaked. As is, it is compatible with the following:

Firmware	Notes
DD-WRT	
OpenWrt	probably not at the moment - should require older mksquashfs if using White Russian
FreeWrt	untested
Sveasoft Alchemy/Talismen	untested
HyperWrt	untested

I may extend default support to cramfs, regular squashfs, and older squashfs and squashfs-lzma versions if there is sufficient demand. Note that for firmwares that do have an easy to use build system and/or an image builder utility, i.e. OpenWrt, this kit is of less use.

Instructions

I've written two scripts that simplify the process considerably. Basically, it's a one-step process to extract the firmware, and a one-step process to rebuild the firmware.

if you haven't already, Install the required packages by issuing:

```
   sudo aptitude install gcc g++ binutils patch bzip2 flex bison make gettext unzip  zlib1g-dev l
```

grab the program from its SVN by issuing:

```
  mkdir firmware_mod_kit
  cd firmware_mod_kit
  svn checkout http://firmware-mod-kit.googlecode.com/svn/trunk/ firmware-mod-kit-read-only
  cd firmware-mod-kit-read-only/trunk/
```

(This is reported to be more up to date:
https://code.google.com/p/firmware-mod-kithttp://wiki.dd-wrt.com/wiki/Documentation)

Extracting the firmware (extract_firmware.sh)

Simply run extract_firmware.sh with the following parameters. You must run this tool from inside the directory it exists in.

```
$ ./extract_firmware.sh firmware.bin working_directory/
```

firmware.bin
: the firmware image you want to extract, i.e. DD-WRT.v23_MICRO_GENERIC.BIN.

working_directory
: the working directory you want to use. This is where the intermediate files and the extracted file system will be stored. Note that files in the working directory may be deleted!

It doesn't matter which firmware image you supply, i.e. for a WRT54G or an ASUS WL-500G. These images are usually all the same and differ only in the header format. The rebuilding process will create images for the various models.

An extraction log is created as *extract.log*.
dsf

Example:

```
$ ./extract_firmware.sh  dd-wrt.v24-11218_NEWD_mega  ./mega/
```

Modifying the firmware (manual and/or ipkg_install.sh)

Modifying the firmware is simple. The file system is stored in the working directory you supplied to extract_firmware.sh. Here are the subfolders of this directory:

- rootfs/
    - ◊ This is where the file system is. Make modifications here.
- image_parts/
    - ◊ This is where intermediate files are stored. If you need to replace the kernel (not at all recommended), you can do so by replacing vmlinuz here.

One merely changes, adds, or removes files in rootfs/ to make changes to the firmware's file system. After you're done making changes, rebuild the firmware. Installing packages (.IPK) Packages are pre-built collections of files pertaining to a set of software. OpenWrt and DD-WRT both use packages, and most are cross-compatible. These packages are stored in a tar/gzip archive of a pre-defined structure that includes some control files. You can extract and copy the files manually, or use the ipkg_install.sh script included with this kit. For a list of some of the available packages (not all may work), see http://downloads.openwrt.org/backports/0.9/ . Example use:

```
$ ./ipkg_install.sh some_package-1.2.5.ipk working_directory/
```

some_package-1.2.5.ipkg
: would be the filename of the package.

working_directory
: is the same working directory you supplied to the extract_firmware.sh script.

Re-building the firmware (build_firmware.sh)

Rebuilding the firmware is as easy as extracting it. Use the build_firmware.sh script to automate the process. You must run this tool from inside the directory it exists in. Example use:

```
$ ./build_firmware.sh output_directory/ working_directory/
```

output_directory
: the path to which the created firmware images should be stored. If images already exist here, they will be over-written. Firmware images for various models will be emitted (these images are all the same but have different header patterns so they are accepted by the target models).

working_directory
: the working directory supplied to extract_firmware.sh.

A building log is created as build.log.

Building on Fedora 9 using DD-WRT V24

Source Retrieval

Customization

Build the source

Checking

Building on Debian Unstable using DD-WRT V24 - Work in progress [~matrix]

Initializing Workspace

Fill the Workspace with Sources

Customization

Building Sources

Testing

Building DD-WRT from Source

Building DD-WRT from source is difficult and according to the text here definitly not working on first try. You will see lots of strange errors and many confusing install-scripts. The forum is full of people who were not able to make this install-procedure running through. The infos in the forum is much newer than these here, but also very confusing and mixed up.

Brainslayer does not have the time to do everything. Until the day comes that DD-WRT will build without any extra steps, I've written some scripts that will set up a build environment for DD-WRT. Newer builds of DD-WRT may break compatibility with these scripts. If this happens and I don't update them, *please* take the time to update them **if** you are sure your changes are appropriate.

Requirements

To build DD-WRT, you need a Linux machine. It should work on any reasonably modern machine with basic development tools installed.

Pre-requisites (todo: someone please finish this list -- most/all of the openwrt pre-requisites are required):

- Automake v1.9.4
- GNU Make v?
- GCC/G++ v4
- ncurses
- binutils
- tar
- bzip2
- gzip
- g++
- patch
- flex
- bison
- make
- gettext
- unzip
- libz-dev
- libc headers
- build-essentials
- subversion

To install/update most of them at once:

```
   sudo apt-get install gcc g++ binutils patch bzip2 flex bison make gettext unzip  zlib1g-dev li
```

Organization

WiP DD-WRT's source is organized like so:

```
src/            source
  router/       packages
  linux/        kernel
opt/            make/control
```

Following is another listing of source tree checked out by svn.

```
[root@ibm ~]$ find  DD-WRT -maxdepth 1 -type d
DD-WRT
DD-WRT/redboot
DD-WRT/src
DD-WRT/opt
DD-WRT/tools
DD-WRT/image
DD-WRT/.svn
DD-WRT/.subversion
```

```
[root@ibm ~]$
```

Source Retrieval

The first step is to retrieve the desired DD-WRT version. You can do this by either downloading from the DD-WRT downloads section, or by checking out using Subversion.

Subversion

First you'll need to get subversion, available at https://subversion.apache.org/ For some Linux distributions, you should be able to get subversion using the package manager tool.

After installing, you can get the latest source from DD-WRT by running the following command:

```
svn co svn://svn.dd-wrt.com/DD-WRT
```

You can also visit the github mirror of dd-wrt for alternate download options (This is a just a mirror, pull requests on github will not be merged. Please report bugs to http://svn.dd-wrt.com/report instead of opening GitHub issues)

```
https://github.com/mirror/dd-wrt
```

Note: 1) src/linux/universal/linux-3.18/3.18-rc4.diff has been removed from GitHub /mirror/dd-wrt as this file (112.51 MB) exceeds github file size limit 100MB.

Important

Please be aware that the repository contains the sources for several linux kernel versions for each hardware platform, so be prepared for a very long download (18G).

To checkout everything except kernel's source code please have a look at ddwrt_selective_co.sh. As of May 2010, the selective checkout is 2.8G.

Do a simple check after svn checkout download.

```
[root@ibm DD-WRT]$ svn list
.cvsignore
.project
Makefile
image/
opt/
redboot/
src/
tools/
[root@ibm DD-WRT]$
```

Download Section

Another way to get the source is to download it from the Downloads Section. Afterwards you need to extract it which is done simply by using a command like:

```
tar -jxvf nameoffile.tar.bz
```

--jbrazio 13:07, 17 December 2008 (CET) The download section seems not to contain the source code for v24 and up.

Instructions

You'll need the 3.4.6 and 4.1.0 mipsl uclibc toolchains. They are available at [[1]] for x86 users. There is also a ppc version in the download section at [[2]] The mips-uclibc 3.4.6 toolchain is used for building the kernel and the 4.1.0 mipsel-uclibc toolchain is used for building the user-mode packages. Unpack the toolchains wherever you desire, the scripts below will set up symlinks appropriately.

To build DD-WRT you simply need to:

```
 0. Run ready_ddwrt.sh and ready_ddwrt_root.sh (the scripts are below).
 1. Add the 4.1.0 toolchains bin folder to your path environment variable. i.e.
   PATH=$PATH:/home/db90h/toolchains/4.1.0-uclibc/bin
 2. In DD-WRT/opt run ./install.sh. Running ./install_* to build a particular
    variant requires you rebuild shared code first.
```

These two scripts should be saved to ready_ddwrt.sh and ready_ddwrt_root.sh. You can download them [here] HOWEVER ready_ddwrt.sh is missing the last two sections to build write4 and webcomp, so you are probably best off just copying that script from this page.

NOTE: $DDROOT/src/linux/brcm may not exist on your file structure (it didn't on mine) so to fix the scripts below make a new sub-directory $DDROOT/src/linux/brcm and then move your $DDROOT/src/linux/linux.v2* folders to $DDROOT/src/linux/brcm or simply change the script below.

NOTE: If you are having issues with the first ready_ddwrt.sh script building mksquashfs-lzma and it's giving you some error about "ld: skipping incompatible ./lzma/C/7zip/Compress/LZMA_Lib/liblzma.a when searching for -llzma", then this is likely the result of the CPU tuning done in the CFLAGS in the makefiles. I had to edit src/squashfs-tools/Makefile and src/squashfs-tools/lzma/C/7zip/Compress/LZMA_Lib/makefile removing the following optimizations:

- -D_FILE_OFFSET_BITS=64
- -D_LARGEFILE_SOURCE
- -mtune=opteron
- -march=opteron
- -mfpmath=sse
- -m3dnow
- -msse2
- -m64
- -mmmx
- -msse3

I kept the "-funroll-loops" and "-O3" optimizations since those should be non-CPU-specific. By all mean leave any optimizations that apply to your CPU. For a 64-bit CPU, leave -D_FILE_OFFSET_BITS=64, -D_LARGEFILE_SOURCE, and -m64. If you're curious as to what your cpu has, look at the flags listed in /proc/cpuinfo.

```
#!/bin/sh
#
# title: ready_ddwrt.sh
# version: 1.14
# author: Jeremy Collake <jeremy@bitsum.com>
#
# This silly script will prepare a build environment
# for DD-WRT. You must also run ready_ddwrt_root.sh.
#
# RL 11/01/2006: Added line to compile tools/webcomp.c and write4.c
#
MINPARAMS=1
if [ $# -lt "$MINPARAMS" ]
        then
        echo usage:
        echo    ready_ddwrt.sh [ddwrt_base_path]
        echo
        echo i.e.:
        echo    ready_ddwrt.sh /home/db90h/DD-WRT
        echo
        exit 1
fi

ME=`whoami`
DDROOT=$1

echo I am $ME
echo DD-WRT is at $DDROOT

echo ...............................................................
echo creating some symlinks
echo ...............................................................
rm $DDROOT/src/linux/brcm/linux.v23/include/asm
ln -s $DDROOT/src/linux/brcm/linux.v23/include/asm-mips $DDROOT/src/linux/brcm/linux.v23/include/
# for CFE building
ln -s $DDROOT/src/linux/brcm/linux.v23 $DDROOT/src/linux/linux
echo done

echo ...............................................................
echo adjusting some attributes
echo ...............................................................
chmod +x $DDROOT/src/router/iptables/extensions/.dccp-test
chmod +x $DDROOT/src/router/iptables/extensions/.layer7-test
echo done

#echo ...............................................................
#echo fixing alconf's
#echo ...............................................................
#cd src/router/pptpd
#aclocal
#cd ../../..

echo ...............................................................
echo "re-building some tools"
echo ...............................................................
cd $DDROOT
```

```
# make bb_mkdep
cd src/router/busybox/scripts
rm bb_mkdep
make bb_mkdep
cd ../../../..

# make jsformat
cd src/router/tools
rm jsformat
make jsformat
cd ../../..

# make mksquashfs-lzma
cd src/squashfs-tools/
rm mksquashfs-lzma
make
cp mksquashfs-lzma ../linux/brcm/linux.v23/scripts/squashfs
cd ../..

# make strip
cd tools
rm ./strip
gcc strip.c -o ./strip
cd ..

# make write3
cd tools
rm ./write3
gcc write3.c -o ./write3
cd ..

# make write4
cd tools
rm ./write4
gcc write4.c -o ./write4
cd ..

# make webcomp
cd tools
rm ./webcomp
gcc -o webcomp -DUEMF -DWEBS -DLINUX webcomp.c
cd ..

echo done
```

This second script needs to be run as root ...

```
#!/bin/sh
#
# title: ready_ddwrt_root.sh
# version: 1.1
# author: Jeremy Collake <jeremy@bitsum.com> aka db90h
#
# This silly script will prepare a build environment
# for DD-WRT. You must also run ready_ddwrt.sh.
#
MINPARAMS=2
if [ $# -lt "$MINPARAMS" ]
        then
        echo
        echo This script needs root access.
        echo
        echo usage:
        echo    ready_ddwrt_root.sh [ddwrt_base_path] [3.4.6_toolchain_base_path]
        echo
        echo i.e.:
        echo    ready_ddwrt_root.sh /home/db90h/DD-WRT /home/db90h/3.4.6-ucliblc-0.9.28
        echo
        exit 1
fi

ME=`whoami`
DDROOT=$1
TCHAIN=$2

echo I am $ME
echo DD-WRT is at $DDROOT
echo mipsl-uclibc-x toolchain is at $TCHAIN

echo ............................................................
echo creating some symlinks
echo ............................................................
# duh, this will already be here
mkdir -p /opt
rm /opt/3.3.6
ln -s $TCHAIN /opt/3.3.6
rm /GruppenLW
ln -s $DDROOT/image /GruppenLW

echo All done!
```

Changing DD-WRT

Work in progress DD-WRT is easily reconfigured. In the /src/router folder you'll find .config* files for each type of distribution. Use 'make menuconfig' to edit the appropriate configuration files. The /src/router/busybox folder contains .config* files for its configuration. Just edit them in a similar way.

Change micro configuration: *WARNING: Before and after doing this you need to copy and restore the "Internal Options" section of the .config files. These options don't seem to be in #the Config template.*

```
cd DD-WRT/src/router
cp .config .config_micro
make menuconfig
cp .config_micro .config
```

Change micro Busybox configuration:

```
cd DD-WRT/src/router/busybox
cp .config .config_micro
make menuconfig
cp .config_micro .config
```

Change micro kernel configuration:

```
cd DD-WRT/src/router/busybox
cp .config .config_micro
make menuconfig
cp .config_micro .config
```

Compiling (custom) Kernel modules for DD-WRT

I have tested this on an Eko big build (2.6.23 or .24 kernel) for an Asus RT-N16. YMMV but I know this should work with minor modifications on other kernel versions.

If you want to build a custom kernel module (eg. to include drivers that are not in the suggested release for your router) you don't necessarily need to recompile all of DD-WRT. The kernel allows for modules to be loaded (manually) in runtime. You can find a list of pre-compiled modules on your router at /lib/modules/`uname -r`/drivers

If your kernel is 2.4 (check with uname -a), there are some kernel modules in ipkg and ipkg-opt.

Pre-requisites

See first these pre-requisites for more information on the tools you require before you begin. You'll need to execute the apt-get command to get your build tools as well as get your mipsel toolchain (4.1.0).

I recommend you have at least some knowledge or experience about the Linux kernel and/or how to build it from scratch before you start. You should also either have some jffs space or another form of (internal or external) media to save your modules and custom scripts once you built them. See also here

You should have a Linux or Unix-like environment (x86 or PPC). I recommend using a small Ubuntu install in VirtualBox or VMWare if you want the least problems. The disk where you store your workfiles needs to have at least a few 100MB and the filesystem needs to be case-sensitive (or you'll get errors during svn).

How to actually do it

Download the kernel sources for the kernel running. Some/most modules will run on one or more additional minor releases of the same kernel. Eg. some modules compiled for .23 will run on .24. Kernel modules compiled for major versions (2.4 vs 2.6) will not interchange, minor versions too far apart will likely fail also.

eg.:

```
svn co svn://svn.dd-wrt.com/DD-WRT/src/linux/brcm/linux-2.6.23
```

(you can probably use the original kernel sources from kernel.org too)

Understand, adapt and execute this (see prerequisites):

```
cd linux-2.6.23
PATH=$PATH:~/toolchains/4.1.0-uclibc-0.9.28/bin
cp .config_std .config
make config
```

Let me explain, .config_std comes from the dd-wrt svn and is a default .config for the MIPS BroadCom chipset and has a few defaults already set up to compile the kernel. I strongly suggest you use it and change that config for your needs.

If it complains about madwifi either get the madwifi drivers into place or take it out of the Kconfig script file on the line that is complaining (unless you really need the madwifi drivers in your modules).

Once that is done you can do:

```
make modules
```

Sit back and relax, get a cup of coffee or take a shower, all types of modules are being built for you. You have to keep track of where your modules are getting build though (this is where your prerequisite experience with the Linux kernel comes in). Eg, I built the Keyspan modules for a Keyspan USB-to-Serial module and those are in drivers/usb/serial

You do not need to do anything else. Definitely do not do make modules_install or make vmlinuz or use any type of make or script that will "install" the kernel as this will do something on your local computer (or vmware image) and possibly overwrite or corrupt your Linux boot (you can recover it but that's not in this scope).

Now go and find those kernel modules (.ko files). Mine was drivers/usb/serial/keyspan.ko, copy them (only the .ko files, we're not interested in the source) to your router using scp or so. Login on the router and do:

```
insmod /jffs/keyspan.ko
lsmod
ls /dev/usb
```

And if all went well you should see the modules in lsmod output and a USB device (or wherever it is for you) in /dev/usb