

Enabling UBIFS

From PlugWiki

Contents

- 1 Quick summary
 - 1.1 Why UBIFS instead of JFFS2?
 - 1.2 Okay, how do I do it?
- 2 Making your UBIFS image
 - 2.1 Make the ubifs filesystem from the rootfs
 - 2.2 Circumnavigating "output file cannot be in the UBIFS root directory" errors
 - 2.3 Make ubi image out of the ubifs
 - 2.4 Format the NAND with your new image
 - 2.5 Attach the image to the UBI subsystem
 - 2.6 Mount the new device
 - 2.7 Reboot into the new system with ubifs root
- 3 Troubleshooting
 - 3.1 Kernel Resolutions
 - 3.1.1 UBIFS patch for Marvell LSP 1.7.3 Linux-2.6.22.18
 - 3.1.2 Marvell Ethernet Support, MV623XX, Issues in linux-2.6.30-rc7
 - 3.2 UBIFS Configuration Resolutions
 - 3.2.1 SELinux on UBIFS
- 4 Appendix
 - 4.1 JFFS2
 - 4.2 UBIFS (default lzo compression)
 - 4.3 UBIFS (zlib compression)
 - 4.4 UBIFS (no compression)
 - 4.5 Random UBI kernel spew

Quick summary

Why UBIFS instead of JFFS2?

Because it's fast! Also, in theory, the UBI layer should be even smarter than JFFS2 about wear-levelling. But mainly because it's really fast. For reference:

```
# Mounting a JFFS2 rootfs from /dev/mtdblock2
root# mount -t jffs2 /dev/mtdblock2 /mnt/nand
43.396s total
```

Compared to:

```
# Mounting an UBIFS rootfs from the same NAND partition
root# ubiattach /dev/ubi_ctrl -m 2 && mount -t ubifs ubi0:rootfs /mnt/nand
1.488s total
```

Okay, how do I do it?

Before you follow these instructions...

1. You need to build a custom kernel (I used orion.git 2.6.30-rc1) **Make sure you turn on UBI (under Device Drivers, Memory Technology Device (MTD) support) and UBIFS (under File systems)**
2. You can also use the Debian kernel if you follow the workarounds detailed in thread at <http://lists.debian.org/debian-arm/2009/07/threads.html#00039>
3. You should build an ARM version of mtd-tools from [git://git.infradead.org/mtd-utils.git](http://git.infradead.org/mtd-utils.git) (ubuntu and debian stable packages are missing ubiformat and mkfs.ubifs) (debian testing appears to have remedied this, as of at least august 22 2009 --Rektide 11:48, 24 August 2009 (UTC))
4. You might want to use an NFS root as the base system to start from (example: /mnt/nfs/rootfs-debian) but if you boot from NFS, you could easily transition the existing JFFS2 filesystem to UBIFS by mounting it (say on /mnt/nand) and then using /mnt/nand instead of rootfs-debian with `mkfs.ubifs`.

Making your UBIFS image

Make the ubifs filesystem from the rootfs

```
root# mkfs.ubifs -r /mnt/nfs/rootfs-debian -m 2048 -e 129024 -c 4096 -o ubifs.img -x zlib
```

The default compression option (if you don't specify `-x (zlib|none)`) is to compress with LZO - zlib tends to offer slightly better compression, and on the SheevaPlug CPU it appears to be faster. See performance benchmarks in the appendix for more details.

Circumnavigating "output file cannot be in the UBIFS root directory" errors

```
thuban:~# mkfs.ubifs -r / -o foo -m 2048 -e 129024 -c 4096
Error: output file cannot be in the UBIFS root directory
```

```
thuban:~# mkfs.ubifs -r / -o /tmp/foo -m 2048 -e 129024 -c 4096
Error: output file cannot be in the UBIFS root directory
```

Trying to create a ubifs image from a current image doesnt work; instead, you have to copy the disk (e.g.; `cp -aur /mnt/usb_stick`) and `mkfs.ubifs` that. The one particular irksome case of dealing with this is if you have a usb root drive, and presently you have no alternative location to copy the image to (eg you've been bitten by the [multiple usb peripheral uboot bug (<http://plugcomputer.org/plugforum/index.php?topic=653.0>)] and do not have a network drive accessible). In this case of last resort, repartition your usb drive into multiple partitions, such that you can mount an auxiliary non-root partition to hold the filesystem image `mkfs.ubifs` outputs. This will let you run `mkfs.ubifs` with only a single usb drive.

Alternatively, you can run `mkfs.ubifs` on a different machine, so long as that machine has access to the file system you wish to image.

Make ubi image out of the ubifs

```
root# ubinize -o ubi.img -m 2048 -p 128KiB -s 512 ubi.cfg
```

contents of ubi.cfg:

```
[ubifs]
mode=ubi
image=ubifs.img
vol_id=0
vol_size=256MiB
vol_type=dynamic
vol_name=rootfs
vol_flags=autoresize
```

Format the NAND with your new image

```
root# ubiformat /dev/mtd2 -s 512 -f ubi.img
```

Attach the image to the UBI subsystem

```
root# ubiattach /dev/ubi_ctrl -m 2
UBIFS: mounted UBI device 0, volume 0, name "rootfs"
UBIFS: file system size: 515837952 bytes (503748 KiB, 491 MiB, 3998 LEBs)
UBIFS: journal size: 9033728 bytes (8822 KiB, 8 MiB, 71 LEBs)
UBIFS: media format: w4/r0 (latest is w4/r0)
UBIFS: default compressor: lzo
UBIFS: reserved for root: 0 bytes (0 KiB)
UBI device number 0, total 4053 LEBs (522934272 bytes, 498.7 MiB),
available 0 LEBs (0 bytes), LEB size 129024 bytes (126.0 KiB)
```

Mount the new device

```
root# mount -t ubifs ubi0:rootfs /mnt/ubi
```

This should be much faster than mounting your old JFFS2 root. Hooray!

Reboot into the new system with ubifs root

```
Marvell>> setenv bootargs 'console=ttyS0,115200 ubi.mtd=2 root=ubi0:rootfs rootfstype=ubifs'
Marvell>> saveenv
Marvell>> nand read.e 0x20000000 0x100000 0x400000
Marvell>> bootm 0x2000000
```

Link title

Troubleshooting

There are some known incompatibilities and issues when setting up UBIFS because of the fairly recent introduction into the Linux kernel.

Kernel Resolutions

Most of the kernel-related issues for UBIFS are discussed on an OpenPlug Forum. You can find a discussion of released kernel and modules for the Sheeva Plug that have UBIFS support (<http://plugcomputer.org/plugforum/index.php?topic=337.0%7Cforum>) and the patches and information surrounding patches. Here is a brief list of some issues you might see.

UBIFS patch for Marvell LSP 1.7.3 Linux-2.6.22.18

The UBIFS patch for the Linux-2.6.22.18 kernel does not seem to allow recognition of the MTD devices with the newest mtd-utils. If you cannot get the 'mtdinfo -a' to work after compiling the mtd-utils as discussed above, you probably need to use the GIT Linux kernel tree (http://www.openplug.org/pluginwiki/index.php/Compiling_Linux_Kernel_for_the_Plug_Computer#Getting_the_kernel 7COrion) .

Marvell Ethernet Support, MV623XX, Issues in linux-2.6.30-rc7

If you happen to compile a new kernel and are using NFS to build the file system to then make a UBI file system to flash, there seems to be an issue with the Marvell ethernet support in linux-2.6.30-rc7 that keeps the NFS root file system from working. You probably need to use the GIT Linux kernel tree (http://www.openplug.org/pluginwiki/index.php/Compiling_Linux_Kernel_for_the_Plug_Computer#Getting_the_kernel 7COrion) . As of May 27, 2009, the Orion GIT tree was on 2.6.30-rc6, which is tested and worked.

UBIFS Configuration Resolutions

SELinux on UBIFS

The SELinux support on UBIFS is discussed elsewhere on the Raindrop SELinux pages.

Appendix

Some rough benchmarking information and a bunch of UBI-related kernel messages follow here.

JFFS2

```
# write the JFFS2 image to NAND
root# nandwrite -p /dev/mtd2 ubuntu-9.0.5.Release.jffs2
52.589s total
```

```
# mount the image to a local directory
root# mount -t jffs2 /dev/mtdblock2 /mnt/nand
43.396s total
```

```
# git clone the xserver
root# git clone /mnt/nfs/xserver
55.878s total
```

```
# remove xserver tree after cloning
root# rm -rf xserver && sync
12.550s total
```

```
# dd a blank 32MB image to the fs
root# dd if=/dev/zero of=test.img bs=1M count=32
32+0 records in
32+0 records out
33554432 bytes (34 MB) copied, 4.74679 s, 7.1 MB/s
4.747 s
```

UBIFS (default lzo compression)

```
# make lzo-compressed ubifs image (lzo compression = default)
root# mkfs.ubifs -r rootfs-debian -m 2048 -e 129024 -c 4096 -o ubifs.img
2m 59.21s total
```

```
# ubinize the image
root# ubinize -o ubi-lzo.img -m 2048 -p 128KiB -s 512 ubi-lzo.cfg
42.203s total
```

```
# flash the NAND with the lzo image
root# ubiformat /dev/mtd2 -s 512 -f ubi-lzo.img
48.718s total
```

```
# git clone the xserver tree
root# git clone /mnt/nfs/xserver
2.959s total
```

```
# remove xserver tree after cloning
root# rm -rf xserver && sync
0.669s total
```

```
# dd a blank 32MB image to the FS
root# dd if=/dev/zero of=test.img bs=1M count=32
32+0 records in
32+0 records out
33554432 bytes (34 MB) copied, 0.21514 s, 156 MB/s
0.227s total
```

UBIFS (zlib compression)

```
# make zlib-compressed ubifs image (zlib is usually more compact)
root# mkfs.ubifs -r rootfs-debian -m 2048 -e 129024 -c 4096 -o ubifs.img -x zlib
1m 58.09s total
```

```
# ubinize the image
root# ubinize -o ubi-zlib.img -m 2048 -p 128KiB -s 512 ubi-zlib.cfg
38.259s total
```

```
# flash the NAND with the zlib image
root# ubiformat /dev/mtd2 -s 512 -f ubi-zlib.img
45.580s total
```

```
# git clone the xserver tree
root# git clone /mnt/nfs/xserver
2.815s total
```

```
# remove xserver tree after cloning
root# rm -rf xserver && sync
0.959s total
```

```
# dd a blank 32MB image to the FS
root# dd if=/dev/zero of=test.img bs=1M count=32
32+0 records in
32+0 records out
33554432 bytes (34 MB) copied, 0.245113 s, 137 MB/s
0.256s total
```

UBIFS (no compression)

```
# make uncompressed ubifs image
root# mkfs.ubifs -r rootfs-debian -m 2048 -e 129024 -c 4096 -o ubifs.img -x none
26.386s total
```

```
# ubinize the image
root# ubinize -o ubi-none.img -m 2048 -p 128KiB -s 512 ubi-none.cfg
1m 13.73s total
```

```
# flash the NAND with the uncompressed image
root# ubiformat /dev/mtd2 -s 512 -f ubi-none.img
1m 16.92s total
```

```
# git clone the xserver tree
root# git clone /mnt/nfs/xserver
3.470s total
```

```
# remove xserver tree after cloning
root# rm -rf xserver && sync
0.922s total
```

```
# dd a blank 32MB image to the FS
root# dd if=/dev/zero of=test.img bs=1M count=32
32+0 records in
32+0 records out
33554432 bytes (34 MB) copied, 0.224067 s, 150 MB/s
0.237s total
```

Random UBI kernel spew

LZO-compressed UBI kernel logs

```
UBI: physical eraseblock size: 131072 bytes (128 KiB)
UBI: logical eraseblock size: 129024 bytes
UBI: smallest flash I/O unit: 2048
UBI: sub-page size: 512
UBI: VID header offset: 512 (aligned 512)
UBI: data offset: 2048
UBI: volume 0 ("rootfs") re-sized from 1626 to 4009 LEBs
UBI: attached mtd2 to ubi0
UBI: MTD device name: "root"
UBI: MTD device size: 507 MiB
UBI: number of good PEBs: 4053
UBI: number of bad PEBs: 3
UBI: max. allowed volumes: 128
UBI: wear-leveling threshold: 2048
UBI: number of internal volumes: 1
UBI: number of user volumes: 1
UBI: available PEBs: 0
UBI: total number of reserved PEBs: 4053
UBI: number of PEBs reserved for bad PEB handling: 40
UBI: max/mean erase counter: 1/1
UBI: background thread "ubi_bgt0d" started, PID 4760
UBIFS: mounted UBI device 0, volume 0, name "rootfs"
UBIFS: file system size: 515837952 bytes (503748 KiB, 491 MiB, 3998 LEBs)
UBIFS: journal size: 9033728 bytes (8822 KiB, 8 MiB, 71 LEBs)
UBIFS: media format: w4/r0 (latest is w4/r0)
UBIFS: default compressor: lzo
UBIFS: reserved for root: 0 bytes (0 KiB)
UBIFS: un-mount UBI device 0, volume 0
UBI: mtd2 is detached from ubi0
```

zlib-compressed UBI kernel logs

```
UBI: attaching mtd2 to ubi0
UBI: physical eraseblock size: 131072 bytes (128 KiB)
UBI: logical eraseblock size: 129024 bytes
UBI: smallest flash I/O unit: 2048
UBI: sub-page size: 512
UBI: VID header offset: 512 (aligned 512)
UBI: data offset: 2048
UBI: volume 0 ("rootfs") re-sized from 1626 to 4009 LEBs
UBI: attached mtd2 to ubi0
UBI: MTD device name: "root"
UBI: MTD device size: 507 MiB
UBI: number of good PEBs: 4053
UBI: number of bad PEBs: 3
```

Enabling UBIFS - PlugWiki

```

UBI: max. allowed volumes:      128
UBI: wear-leveling threshold:   2048
UBI: number of internal volumes: 1
UBI: number of user volumes:    1
UBI: available PEBs:            0
UBI: total number of reserved PEBs: 4053
UBI: number of PEBs reserved for bad PEB handling: 40
UBI: max/mean erase counter: 3/2
UBI: background thread "ubi_bgt0d" started, PID 4844
UBIFS: mounted UBI device 0, volume 0, name "rootfs"
UBIFS: file system size:      515837952 bytes (503748 KiB, 491 MiB, 3998 LEBs)
UBIFS: journal size:          9033728 bytes (8822 KiB, 8 MiB, 71 LEBs)
UBIFS: media format:          w4/r0 (latest is w4/r0)
UBIFS: default compressor:    zlib
UBIFS: reserved for root:     0 bytes (0 KiB)
UBIFS: un-mount UBI device 0, volume 0
UBI: mtd2 is detached from ubi0

```

Uncompressed UBI kernel logs

```

UBI: attaching mtd2 to ubi0
UBI: physical eraseblock size:  131072 bytes (128 KiB)
UBI: logical eraseblock size:   129024 bytes
UBI: smallest flash I/O unit:   2048
UBI: sub-page size:             512
UBI: VID header offset:         512 (aligned 512)
UBI: data offset:               2048
UBI: volume 0 ("rootfs") re-sized from 1626 to 4009 LEBs
UBI: attached mtd2 to ubi0
UBI: MTD device name:           "root"
UBI: MTD device size:           507 MiB
UBI: number of good PEBs:       4053
UBI: number of bad PEBs:        3
UBI: max. allowed volumes:      128
UBI: wear-leveling threshold:   2048
UBI: number of internal volumes: 1
UBI: number of user volumes:    1
UBI: available PEBs:            0
UBI: total number of reserved PEBs: 4053
UBI: number of PEBs reserved for bad PEB handling: 40
UBI: max/mean erase counter: 6/4
UBI: background thread "ubi_bgt0d" started, PID 4917
UBIFS: mounted UBI device 0, volume 0, name "rootfs"
UBIFS: file system size:      515837952 bytes (503748 KiB, 491 MiB, 3998 LEBs)
UBIFS: journal size:          9033728 bytes (8822 KiB, 8 MiB, 71 LEBs)
UBIFS: media format:          w4/r0 (latest is w4/r0)
UBIFS: default compressor:    none
UBIFS: reserved for root:     0 bytes (0 KiB)
UBIFS: un-mount UBI device 0, volume 0
UBI: mtd2 is detached from ubi0

```

Retrieved from "http://plugcomputer.org/plugwiki/index.php/Enabling_UBIFS"

Category: HowTo

■ This page was last modified on 6 January 2011, at 12:45.