Ext2 to ext3 file conversation

[root@server1 ~]# **tune2fs -j /dev/sda5**

Ext3 to ext4 file system conversation

[root@server1 ~]# **tune2fs -O dir_index,uninit_bg,extents /dev/sda5**

Ext2 to ext4 file system conversation
[root@server1 ~]# tune2fs -O dir_index,uninit_bg,has_journal /dev/sda5

Ext3 to ext2 file system back conversation
# tune2fs -O ^has_journal /dev/sda5

Converting from Ext2 to Ext3

The conversion procedure is simple enough. Imagine /dev/hda10 mounted as /test – the procedure would be as follows:

- Log in as root
- Make sure `/etc/fstab` has `/dev/hda10` mounted to `/test` as ext2, read write
- `umount /dev/hda10`
    - `If you can't unmount it, then remount it read only (mount -o remount,ro /dev/hda10)`
- `tune2fs -j /dev/hda10`
- Edit `/etc/fstab`, and for `/dev/hda10`, change ext2 to ext3
- `mount /dev/hda10`
- `/sbin/shutdown -h now`
- `mount | grep /dev/hda10`
    - If it's not shown as ext3, reboot, if still not, troubleshoot
    - Otherwise, you're done.

A few explanations are in order. The tunefs command creates the journal file, which is kept in a special inode on the device (by default). You then must change the `/etc/fstab` entry to reflect it's a journalling filesystem, and then mount it.

Converting the / directory

First, think long and hard before deciding to convert the root directory. Ext3's primary purpose is shorter recovery from disaster rather than data loss prevention. Converting the root directory from

Ext2 to Ext3 isn't difficult, but converting it back from Ext3 to Ext2 is a treacherous process fraught with problems. But, if you really must perform the Ext2 to Ext3 conversion on the root directory, here's how, assuming `/dev/hda2` is mounted as the root directory and `/dev/hda1` is mounted as `/boot`:

- Log in as root
- Edit `/etc/fstab` and change `ext2` to `ext3` on the line referencing the root directory.
- `tune2fs -j /dev/hda2`
- `cd /boot`
- `mv initrd-2.4.18-26.8.0.img initrd-2.4.18-26.8.0.img.ext2`
- `mkinitrd initrd-2.4.18-26.8.0.img 2.4.18-26.8.0`
- `reboot`

In the preceding, you MUST perform all the steps, including the `mkinitrd`, before rebooting. Failing to perform all the steps before rebooting produces a "buried shovel" where if only you could boot the machine, you could run the `mkinitrd` command, and if only you could run the `mkinitrd` command, you could boot the machine.

Converting from Ext3 back to Ext2

There may come a time when you want to convert back to Ext2. For directories other than the root directory or `/usr`, it's pretty easy. The following once again uses the example of `/dev/hda10` mounted to directory `/test`:

- `umount /dev/hda10`
- `tune2fs -O ^has_journal /dev/hda10`
- `e2fsck /dev/hda10`
- Edit `/etc/fstab` to change `/dev/hda10` to mount type ext2
- `mount /dev/hda10`

The `tune2fs` command removes the journal inode, and the `e2fsck` command completes that removal.

Back-Converting the root directory

The root directory is a challenge for a number of reasons. First, it must be mounted for the system to run, but it must be unmounted to run the `e2fsck` command. Also, different distros behave different ways. The `mkinitrd` command varies widely between distros. The preceding works on a Red Hat 8.0 machine, but other machines might require other solutions. We'll assume that `/dev/hda1` is `/boot`, while `/dev/hda2` is the root directory (`/`).

- Log in as root
- Edit `/etc/fstab` to change the `/dev/hda2` line from `ext3` to `ext2`
- `reboot`

- Log in as root
- `mv initrd-2.4.18-26.8.0.img initrd-2.4.18-26.8.0.img.ext3`
- `mkinitrd initrd-2.4.18-26.8.0.img 2.4.18-26.8.0`
- Place your Knoppix CD in the CD drive
- `reboot`
- Notice you are now booted to Knoppix
- Ctrl+Alt+F2 to access a root prompt
- `umount /dev/hda1`
- `umount /dev/hda2`
- `tune2fs -O ^has_journal /dev/hda2`
- `e2fsck /dev/hda2`
- `mount -t ext3 /dev/hda2 /mnt/hda2`
  - This should fail. If it does, it proves that the partition no longer has a journaling inode.
- `reboot`
- Remove the Knoppix CD and press Enter as prompted
- Notice you are now booted to the original operating system (hopefully)
- Log in as root
- Execute a `mount` command to verify an Ext2 root partition.

Converting a non-root filesystem to ext4

As long as you are converting a filesystem that can be unmounted, it is fairly simple procedure. In this example we will be converting a */dev/sdc1* partition mounted as */home* directory.

First, unmount the partition.

```
umount /dev/sdc1
```

Next, run a filesystem check on it to make sure it is in sane condition. We are still on *ext3*.

```
fsck.ext3 -pf /dev/sdc1
```

Enable new features of *ext4* on the filesystem.

```
tune2fs -O extents,uninit_bg,dir_index /dev/sdc1
```

Option "extents" enables the filesystem to use extents instead of bitmap mapping for files, "uninit_bg" reduces file system check times by only checking used portions of the disk, and "dir_index" allows storing the contents of large directories in a htree for faster access. Option "dir_index" is also supported by *ext3*, so you may already be using it, but it makes no harm to specify it here.

Run a filesystem check. It will find errors. It is normal. Let it fix them. You may want to run the check twice to make sure that the filesystem is now clean.

```
fsck.ext4 -yfD /dev/sdc1
```

The "-D" parameter will actually enable the "dir_index" option by rebuilding directory index. It can be rebuilt (optimized) at any later time by running the check with the parameter.

Now edit your */etc/fstab* file to say "ext4" instead of "ext3" for */home*. Other options may differ for your system.

```
/dev/sdc1 /home ext4 defaults 0 2
```

Try to mount your new *ext4* filesystem.

```
mount /home
```

If it succeeds, congratulations. If not, do not panic. You have not lost your data. And you have a backup after all, right? Make sure you have all the latest tools listed in prerequisites. Get them form Debian unstable or experimental if needed. Upgrade and try again.

The /boot partition

If your */boot* is a separate partition, all is good and great. Just leave it as *ext3*. Latest development *grub* versions do have support for *ext4*, but it still may and will fail for some given snapshot of *grub*.

As *ext3* can be mounted *ext4* without conversion, you can just edit your */etc/fstab* to say "ext4" instead of "ext3" for boot partition.

```
/dev/sdb1 /boot ext4 defaults 0 1
```

Most features of *ext4* will not be used, but that makes little difference for */boot*, as it is only used early at boot time. And since this is essentially still an *ext3* partition, *grub* will have no problem booting it.

If, on the other hand, you do not have a separate */boot* partition, you should consider creating one. Otherwise you must be really careful, and not enable features not supported by grub, or make sure that you are using a version of grub that supports all of them.

Converting a root filesystem to ext4

Converting a root filesystem is a bit more tricky because you cannot unmount it, as your system is running on it. Nevertheless it is still possible to do it without using an external bootable media. You should do this in a single-user mode.

First step is to modify your */etc/fstab* file to say "ext4" instead of "ext3" for root partition. This is important because you will be operating on a read-only filesystem later, and will not be able to make the change, and this would result in your system unable to mount a root filesystem on next boot.

Let us assume that root partition is */dev/sda1*, so your */etc/fstab* should look something like this.

```
/dev/sda1 / ext4 defaults 0 1
```

Now remount the root filesystem read-only.

```
mount -o remount,ro /
```

Then run a filesystem check on the root filesystem.

```
fsck.ext3 -pf /dev/sda1
```

It will tell you to reboot the system. That may be a good idea, so simply boot into single-user mode and remount it read-only again. It is fine even though we have already modified */etc/fstab*, because *ext3* can be mounted as *ext4* without conversion.

Next, enable all the *ext4* features on the root filesystem.

```
tune2fs -O extents,uninit_bg,dir_index /dev/sda1
```

And run run a filesystem check on the root filesystem again. It will find and fix errors. This is normal.

```
fsck.ext4 -yfD /dev/sda1
```

You can now reboot to your new *ext4* system, and enjoy faster filesystem check times, better performance, and all the improvements of *ext4*. Well, almost; read the next section.

convert an ext4 partition to ext3 partition without formatting the HDD

not possible and even if it's somehow possible, not recommended. Unless you like to live extremely dangerous life.

All ext3 partitions can be mounted as ext4. But not all ext4 partitions can be mounted as ext3. Some can. It's backwards compatible in that you can mount ext2 and ext3 partitions as ext4 and they are perfectly valid ext4 partitions. It's forwards-compatible in certain circumstances in that you can mount some ext4 partitions as ext2 or ext3.

Convert the ext2 filesystem to ext4

```
[root@f11 ~]# umount /mnt2
[root@f11 ~]# tune2fs –O dir_index,uninit_bg,has_journal /dev/ram0
tune2fs 1.41.4 (27-Jan-2009)
Please run e2fsck on the filesystem.
Creating journal inode: done
This filesystem will be automatically checked every 39 mounts or
180 days, whichever comes first. Use tune2fs –c or –i to override.
[root@f11 ~]# e2fsck –pf /dev/ram0
/dev/ram0: Group descriptor 0 checksum is invalid. FIXED.
/dev/ram0: 140/4096 files (0.7% non-contiguous), 3291/16384 blocks
```

```
[root@f11 ~]# mount /dev/ram0 /mnt2
[root@f11 ~]# df –Thi /mnt2
Filesystem Type Inode  Iused Ifree IUse% Mounted on
/dev/ram0 ext4 4.0K 140 3.9K 4%  /mnt2
```