Learn Linux, 101: Maintain the integrity of filesystems

Keeping track of your disk space

Ian Shields
Senior Programmer
IBM

24 August 2010

Learn how to check the integrity of your Linux® filesystems, monitor free space, and fix simple problems. Use the material in this article to study for the Linux Professional Institute (LPI) 101 exam for Linux system administrator certification—or just to check your filesystems and keep them in good working order, especially after a system crash or power loss.

View more content in this series

Overview

In this article, learn to:

- Verify the integrity of filesystems
- Monitor free space and inodes
- Repair simple filesystem problems

This article covers standard and journaling (also called journaling) filesystems with an emphasis is on ext2 (standard filesystem) and ext3 (journaling filesystem), but tools for other filesystems are mentioned too. Most of this material applies to both 2.4 and 2.6 kernels. Most examples in this article use Fedora 12, with a 2.6.32 kernel. Your results on other systems may differ.

This article helps you prepare for Objective 104.2 in Topic 104 of the Linux Professional Institute Certification level 1 (LPIC-1) exams. The objective has a weight of 2.

Prerequisites

To get the most from the articles in this series, you should have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this article. Sometimes different versions of a program will format output differently, so your results may not always look exactly like the listings and figures shown here.

About this series

This series of articles helps you learn Linux system administration tasks. You can also use the material in these articles to prepare for Linux Professional Institute Certification level 1 (LPIC-1) exams.

See our developerWorks roadmap for LPIC-1 for a description of and link to each article in this series. The roadmap is in progress and reflects the *current objectives (April 2009)* for the LPIC-1 exams. As we complete articles, we add them to the roadmap. In the meantime, in our previous LPI certification exam prep tutorials, find earlier versions of similar material that supports LPIC-1 *objectives prior to April 2009*.

You should also be familiar with the material in our article "Learn Linux 101: Create partitions and filesystems."

Checking filesystems

Connect with Ian

Ian is one of our most popular and prolific authors. Browse all of Ian's articles on developerWorks. Check out Ian's profile and connect with him, other authors, and fellow readers in My developerWorks.

In cases when your system crashes or loses power, Linux may not be able to cleanly unmount your filesystems. Thus, your filesystems may be left in an inconsistent state, with some changes completed and some not. Operating with a damaged filesystem is not a good idea as you are likely to further compound any existing errors.

The main tool for checking filesystems is `fsck`, which, like `mkfs`, is really a front end to filesystem-checking routines for the various filesystem types. Some of the underlying check routines are shown in Listing 1.

Listing 1. Some of the fsck programs

```
[ian@echidna ~]$ ls /sbin/*fsck*
/sbin/btrfsck  /sbin/fsck         /sbin/fsck.ext3     /sbin/fsck.msdos
/sbin/dosfsck  /sbin/fsck.cramfs  /sbin/fsck.ext4     /sbin/fsck.vfat
/sbin/e2fsck   /sbin/fsck.ext2    /sbin/fsck.ext4dev  /sbin/fsck.xfs
```

You may be surprised to learn that several of these files are hard links to just one file as shown in Listing 2. Remember that these programs may be used so early in the boot process that the filesystem may not be mounted and symbolic link support may not yet be available. See our article Learn Linux, 101: Create and change hard and symbolic links for more information about hard and symbolic links.

Listing 2. One fsck program with many faces

```
[ian@echidna ~]$ find /sbin -samefile /sbin/e2fsck
/sbin/fsck.ext4dev
/sbin/e2fsck
/sbin/fsck.ext3
/sbin/fsck.ext4
/sbin/fsck.ext2
```

The system boot process use `fsck` with the `-A` option to check the root filesystem and any other filesystems that are specified for checking in the /etc/fstab control file. If the filesystem was not cleanly unmounted, a consistency check is performed and repairs are made, if they can be done safely. This is controlled by the *pass* (or *passno*) field (the sixth field) of the /etc/fstab entry. Filesystems with pass set to zero are not checked at boot time. The root filesystem has a pass value of 1 and is checked first. Other filesystems will usually have a pass value of 2 (or higher), indicating the order in which they should be checked.

Multiple `fsck` operations can run in parallel if the system determines it is advantageous, so different filesystems are allowed to have the same pass value, as is the case for the /grubfile and // mnt/ext3test filesystems shown in Listing 3. Note that `fsck` will avoid running multiple filesystem checks on the same physical disk. To learn more about the layout of /etc/fstab, check the man pages for `fstab`.

Listing 3. Boot checking of filesystems with /etc/fstab entries

```
filesystem                              mount point  type   options    dump pass
UUID=a18492c0-7ee2-4339-9010-3a15ec0079bb /              ext3    defaults        1   1
UUID=488edd62-6614-4127-812d-cbf58eca85e9 /grubfile      ext3    defaults        1   2
UUID=2d4f10a6-be57-4e1d-92ef-424355bd4b39 swap           swap    defaults        0   0
UUID=ba38c08d-a9e7-46b2-8890-0acda004c510 swap           swap    defaults        0   0
LABEL=EXT3TEST                            /mnt/ext3test  ext3    defaults        0   2
/dev/sda8                                 /mnt/xfstest   xfs     defaults        0   0
LABEL=DOS                                 /dos           vfat    defaults        0   0
tmpfs                   /dev/shm                       tmpfs   defaults        0   0
devpts                  /dev/pts                       devpts  gid=5,mode=620  0   0
sysfs                   /sys                           sysfs   defaults        0   0
proc                    /proc                          proc    defaults        0   0
```

Some journaling filesystems, such as ReiserFS and XFS, might have a pass value of 0 because the journaling code, rather than `fsck`, does the filesystem consistency check and repair. On the other hand, some filesystems, such as /proc, are built at initialization time and therefore do need to be checked.

You can check filesystems after the system is booted. You will need root authority, and the filesystem you want to check should be unmounted first. Listing 4 shows how to check two of our filesystems, using the device name, label, or UUID. You can use the `blkid` command to find the device given a label or UUID, and the label and UUID, given the device.

Listing 4. Using fsck to check filesystems

```
[root@echidna ~]# # find the device for LABEL=EXT3TEST
[root@echidna ~]# blkid -L EXT3TEST
/dev/sda7
[root@echidna ~]# # Find label and UUID for /dev/sda7
[root@echidna ~]# blkid /dev/sda7
/dev/sda7: LABEL="EXT3TEST" UUID="7803f979-ffde-4e7f-891c-b633eff981f0" SEC_TYPE="ext2"
 TYPE="ext3"
[root@echidna ~]# # Check /dev/sda7
[root@echidna ~]# fsck /dev/sda7
fsck from util-linux-ng 2.16.2
e2fsck 1.41.9 (22-Aug-2009)
EXT3TEST: clean, 11/7159808 files, 497418/28637862 blocks
[root@echidna ~]# # Check it by label using fsck.ext3
[root@echidna ~]# fsck.ext3 LABEL=EXT3TEST
e2fsck 1.41.9 (22-Aug-2009)
```

```
EXT3TEST: clean, 11/7159808 files, 497418/28637862 blocks
[root@echidna ~]# # Check it by UUID using e2fsck
[root@echidna ~]# e2fsck UUID=7803f979-ffde-4e7f-891c-b633eff981f0
e2fsck 1.41.9 (22-Aug-2009)
EXT3TEST: clean, 11/7159808 files, 497418/28637862 blocks
[root@echidna ~]# # Finally check the vfat partition
[root@echidna ~]# fsck LABEL=DOS
fsck from util-linux-ng 2.16.2
dosfsck 3.0.9, 31 Jan 2010, FAT32, LFN
/dev/sda9: 1 files, 1/513064 clusters
```

If you attempt to check a mounted filesystem, you will usually see a warning similar to the one in Listing 5 where we try to check our root filesystem. Heed the warning and do not do it!

Listing 5. Do not attempt to check a mounted filesystem

```
[root@echidna ~]# fsck UUID=a18492c0-7ee2-4339-9010-3a15ec0079bb
fsck from util-linux-ng 2.16.2
e2fsck 1.41.9 (22-Aug-2009)
/dev/sdb9 is mounted.

WARNING!!!  Running e2fsck on a mounted filesystem may cause
SEVERE filesystem damage.

Do you really want to continue (y/n)? no

check aborted.
```

It is also a good idea to let `fsck` figure out which check to run on a filesystem; running the wrong check can corrupt the filesystem. If you want to see what `fsck` would do for a given filesystem or set of filesystems, use the `-N` option as shown in Listing 6.

Listing 6. Finding what fsck would do to check /dev/sda7, /dev/sda8, and /dev/sda9

```
[root@echidna ~]# fsck -N /dev/sda7 /dev/sda[89]
fsck from util-linux-ng 2.16.2
[/sbin/fsck.ext3 (1) -- /mnt/ext3test] fsck.ext3 /dev/sda7
[/sbin/fsck.xfs (2) -- /mnt/xfstest] fsck.xfs /dev/sda8
[/sbin/fsck.vfat (3) -- /dos] fsck.vfat /dev/sda9
```

So far, we have checked ext and vfat filesystems. Let's now check the XFS filesystem on /dev/sda8. As you can see in Listing 7, the `fsck` command simply tells us that we should use the `xfs_check` command. If there are no errors, then `xfs_check` does not display any output. There is a `-v` option for verbose output, but it is much too verbose for a simple check.

Listing 7. Using fsck with XFS

```
[root@echidna ~]# fsck /dev/sda8
fsck from util-linux-ng 2.16.2
If you wish to check the consistency of an XFS filesystem or
repair a damaged filesystem, see xfs_check(8) and xfs_repair(8).
[root@echidna ~]# xfs_check /dev/sda8
```

Monitoring free space

On a storage device, a file or directory is contained in a collection of *blocks*. Information about a file is contained in an *inode*, which records information such who the owner is, when the file was

last accessed, how large it is, whether it is a directory, and who can read from or write to it. The inode number is also known as the file serial number and is unique within a particular filesystem. See our article Learn Linux, 101: File and directory management for more information on files and directories.

Data blocks and inodes each take space on a filesystem, so you need to monitor the space usage to ensure that your filesystems have space for growth.

The df command

The `df` command displays information about mounted filesystems. If you add the `-T` option, the filesystem type is included in the display; otherwise, it is not. The output from `df` for the Fedora 12 system that we used above is shown in Listing 8.

Listing 8. Displaying filesystem usage

```
[ian@echidna ~]$ df -T
Filesystem    Type   1K-blocks      Used Available Use% Mounted on
/dev/sdb9     ext3    45358500  24670140  18384240  58% /
tmpfs        tmpfs     1927044       808   1926236   1% /dev/shm
/dev/sda2     ext3      772976     17760    716260   3% /grubfile
/dev/sda8      xfs    41933232      4272  41928960   1% /mnt/xfstest
/dev/sda7     ext3   112754024    192248 106834204   1% /mnt/ext3test
/dev/sda9     vfat     2052256         4   2052252   1% /dos
```

Notice that the output includes the total number of blocks as well as the number used and free. Also notice the filesystem, such as /dev/sbd9, and its mount point: / /dev/sdb9. The `tmpfs` entry is for a virtual memory filesystem. These exist only in RAM or swap space and are created when mounted without need for a `mkfs` command. You can read more about tmpfs in "Common threads: Advanced filesystem implementor's guide, Part 3".

For specific information on inode usage, use the `-i` option on the `df` command. You can exclude certain filesystem types using the `-x` option, or restrict information to just certain filesystem types using the `-t` option. Use these multiple times if necessary. See the examples in Listing 9.

Listing 9. Displaying inode usage

```
[ian@echidna ~]$ df -i -x tmpfs
Filesystem            Inodes   IUsed   IFree IUse% Mounted on
/dev/sdb9            2883584  308920 2574664   11% /
/dev/sda2              48768      41   48727    1% /grubfile
/dev/sda8            20976832       3 20976829    1% /mnt/xfstest
/dev/sda7            7159808      11 7159797    1% /mnt/ext3test
/dev/sda9                  0       0       0    -  /dos
[ian@echidna ~]$ df -iT -t vfat -t ext3
Filesystem    Type    Inodes   IUsed   IFree IUse% Mounted on
/dev/sdb9     ext3   2883584  308920 2574664   11% /
/dev/sda2     ext3     48768      41   48727    1% /grubfile
/dev/sda7     ext3   7159808      11 7159797    1% /mnt/ext3test
/dev/sda9     vfat         0       0       0    -  /dos
```

You may not be surprised to see that the FAT32 filesystem does not have inodes. If you had a ReiserFS filesystem, its information would also show no inodes. ReiserFS keeps metadata for

files and directories in *stat items*. And since ReiserFS uses a balanced tree structure, there is no predetermined number of inodes as there are, for example, in ext2, ext3, or xfs filesystems.

There are several other options you may use with `df` to limit the display to local filesystems or control the format of output. For example, use the `-H` option to display human readable sizes, such as 1K for 1024, or use the `-h` (or `--si`) option to get sizes in powers of 10 (1K=1000).

If you aren't sure which filesystem a particular part of your directory tree lives on, you can give the `df` command a parameter of a directory name or even a filename as shown in Listing 10.

Listing 10. Human readable output for df

```
[ian@echidna ~]$ df --si ~ian/index.html
Filesystem             Size   Used  Avail Use% Mounted on
/dev/sdb9               47G    26G    19G  58% /
```

The tune2fs command

The ext family of filesystems also has a utility called `tune2fs`, which can be used to inspect information about the block count as well as information about whether the filesystem is journaled (ext3 or ext4) or not (ext2). The command can also be used to set many parameters or convert an ext2 filesystem to ext3 by adding a journal. Listing 11 shows the output for a near-empty ext3 filesystem using the `-l` option to simply display the existing information.

Listing 11. Using tune2fs to display ext4 filesystem information

```
[root@echidna ~]# tune2fs -l /dev/sda7
tune2fs 1.41.9 (22-Aug-2009)
Filesystem volume name:   EXT3TEST
Last mounted on:          <not available>
Filesystem UUID:          7803f979-ffde-4e7f-891c-b633eff981f0
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode dir_index filetype
 needs_recovery sparse_super large_file
Filesystem flags:         signed_directory_hash
Default mount options:    (none)
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              7159808
Block count:              28637862
Reserved block count:     1431893
Free blocks:              28140444
Free inodes:              7159797
First block:              0
Block size:               4096
Fragment size:            4096
Reserved GDT blocks:      1017
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         8192
Inode blocks per group:   512
Filesystem created:       Mon Aug  2 15:23:34 2010
Last mount time:          Tue Aug 10 14:17:53 2010
Last write time:          Tue Aug 10 14:17:53 2010
Mount count:              3
Maximum mount count:      30
Last checked:             Mon Aug  2 15:23:34 2010
Check interval:           15552000 (6 months)
```

```
Next check after:         Sat Jan 29 14:23:34 2011
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:           256
Required extra isize:     28
Desired extra isize:      28
Journal inode:            8
Default directory hash:   half_md4
Directory Hash Seed:      2438df0d-fa91-4a3a-ba88-c07b2012f86a
Journal backup:           inode blocks
```

xfs_info

For XFS filesystems you can display the same information that `mkfs.xfs` displayed when the filesystem was created using the `xfs_info` as shown in Listing 12. You need to use `xfs_info` on a mounted filesystem.

Listing 12. Using xfs_info to display XFS filesystem information

```
[root@echidna ~]# xfs_info /dev/sda8
meta-data=/dev/sda8              isize=256    agcount=4, agsize=2622108 blks
         =                       sectsz=512   attr=2
data     =                       bsize=4096   blocks=10488429, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =internal               bsize=4096   blocks=5121, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

The du command

The `df` command gives information about a whole filesystem. Sometimes you might want to know how much space is used by your home directory, or how big a partition to use if you wanted to move /usr to its own filesystem. To answer this kind of question, use the `du` command.

The `du` command displays information about the filename (or filenames) given as parameters. If a directory name is given, then `du` recurses and calculates sizes for every file and subdirectory of the given directory. The result can be a lot of output. Fortunately, you can use the `-s` option to request just a summary for a directory. If you use `du` to get information for multiple directories, then add the `-c` option to get a grand total. You can also control output format with the same set of size options (`-h`, `-H`, `--si`, and so on) that are used for `df`. Listing 13 shows two views of the home directory of a newly created user who has logged in once and created an index.html file.

Listing 13. Using du

```
[testuser1@echidna ~]$ du -hc *
4.0K Desktop
4.0K Documents
4.0K Downloads
16K index.html
4.0K Music
4.0K Pictures
4.0K Public
4.0K Templates
4.0K Videos
48K total
[testuser1@echidna ~]$ du -hs .
1.1M .
```

The reason for the difference between the 48K total from `du -c *` and the 1.1M summary from `du -s` is that the latter includes the entries starting with a dot, such as .bashrc, while the former does not.

One other thing to note about `du` is that you must be able to read the directories that you are running it against.

So now, let's use `du` to display the total space used by the /usr tree and each of its first-level subdirectories. The result is shown in Listing 14. Use root authority to make sure you have appropriate access permissions.

Listing 14. Using du on /usr

```
[root@echidna ~]# du -shc /usr/*
394M /usr/bin
4.0K /usr/etc
4.0K /usr/games
156M /usr/include
628K /usr/kerberos
310M /usr/lib
1.7G /usr/lib64
110M /usr/libexec
136K /usr/local
30M /usr/sbin
2.9G /usr/share
135M /usr/src
0 /usr/tmp
5.7G total
```

Repairing filesystems

Occasionally, very occasionally we hope, the worst will happen and you will need to repair a filesystem because of a crash or other failure to unmount cleanly. The `fsck` command that you saw above can repair filesystems as well as check them. Usually the automatic boot-time check will fix the problems and you can proceed.

If the automatic boot-time check of filesystems is unable to restore consistency, you are usually dumped into a single user shell with some instructions to run `fsck` manually. For an ext2 filesystem, which is not journaled, you may be presented with a series of requests asking you to confirm proposed actions to fix particular blocks on the filesystem. You should generally allow `fsck` to attempt to fix problems, by responding `y` (for yes). When the system reboots, check for any missing data or files.

If you suspect corruption, or want to run a check manually, most of the checking programs require the filesystem to be unmounted, or at least mounted read-only. Because you can't unmount the root filesystem on a running system, the best you can do is drop to single user mode (using `telinit 1`) and then remount the root filesystem read-only, at which time you should be able to perform a consistency check. A better way to check a filesystem is to boot a recovery system, such as a live CD or a USB memory key, and perform the check of your unmounted filesystems from that.

If `fsck` cannot fix the problem, you do have some other tools available, although you will generally need advanced knowledge of the filesystem layout to successfully fix it.

Why journal?

An `fsck` scan of an ext2 disk can take quite a while to complete, because the internal data structure (or *metadata*) of the filesystem must be scanned completely. As filesystems get larger and larger, this takes longer and longer, even though disks also keep getting faster, so a full check may take one or more hours.

This problem was the impetus for *journaled*, or *journaling*, filesystems. Journaled filesystems keep a log of recent changes to the filesystem metadata. After a crash, the filesystem driver inspects the log in order to determine which recently changed parts of the filesystem may possibly have errors. With this design change, checking a journaled filesystem for consistency typically takes just a matter of seconds, regardless of filesystem size. Furthermore, the filesystem driver will usually check the filesystem on mounting, so an external `fsck` check is generally not required. In fact, for the xfs filesystem, `fsck` does nothing!

If you do run a manual check of a filesystem, check the man pages for the appropriate `fsck` command (`fsck.ext3`, `e2fsck` , `reiserfsck`, and so on) to determine the appropriate parameters. The `-p` option, when used with ext2, ext3, or ext4 filesystems will cause `fsck` to automatically fix all problems that can be safely fixed. This is, in fact, what happens at boot time.

We'll illustrate the use of `e2fsck` and `xfs_check` by first running `e2fsck` on an empty XFS filesystem and then using `xfs_check` to fix it. Remember we suggested that you use the `fsck` front end to be sure you are using the right checker, and we warned you that failure to do so may result in filesystem corruption.

In Listing 15, we start running `e2fsck` against /dev/sda8, which contains an XFS filesystem. After a few interactions we use ctrl-Break to break out, but it is too late. **Warning**: Do **NOT** do this unless you are willing to destroy your filesystem.

Listing 15. Deliberately running e2fsck manually on an XFS filesystem

```
[root@echidna ~]# e2fsck /dev/sda8
e2fsck 1.41.9 (22-Aug-2009)
/dev/sda8 was not cleanly unmounted, check forced.
Resize inode not valid.  Recreate<y>? yes

Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
Free blocks count wrong for group #0 (31223, counted=31224).
Fix<y>? ctrl-Break

/dev/sda8: e2fsck canceled.

/dev/sda8: ***** FILE SYSTEM WAS MODIFIED *****
```

Even if you broke out at the first prompt, your XFS filesystem would still have been corrupted. Repeat after me. Do **NOT** do this unless you are willing to destroy your filesystem.

Now let's use `xfs_check` to repair the XFS filesystem. The `xfs_check` command is quite verbose, but it has a `-s` option which reports only serious errors. The output is shown in Listing 16.

Listing 16. Repairing the XFS filesystem using xfs_check

```
[root@echidna ~]# xfs_check -s /dev/sda8
cache_node_purge: refcount was 1, not zero (node=0x1cf3ee0)
xfs_check: cannot read root inode (117)
cache_node_purge: refcount was 1, not zero (node=0x1cf7400)
xfs_check: cannot read realtime bitmap inode (117)
bad magic # 0x1040000 in btbno block 0/1
bad magic # 0x4000 in btcnt block 0/2
bad magic # 0x58465342 in inobt block 0/0
```

You can also use `xfs_repair` to repair an XFS filesystem. Like `xfs_check`, it is quite verbose, and it does not have an `-s` option. If you'd like just to see what needs repair without actually repairing it, use `xfs_repair -n`.

Superblocks

You may be wondering how all these checking and repairing tools know where to start. Linux and UNIX filesystems usually have a *superblock*, which describes the filesystem *metadata*, or data describing the filesystem itself. This is usually stored at a known location, frequently at or near the beginning of the filesystem, and replicated at other well-known locations. You can use the `-n` option of `mke2fs` to display the superblock locations for an existing filesystem. If you specified parameters such as the bytes per inode ratio, you should invoke `mke2fs` with the same parameters when you use the `-n` option. Listing 17 shows the location of the superblocks on /dev/sda7.

Listing 17. Finding superblock locations

```
[root@echidna ~]# mke2fs -n /dev/sda7
mke2fs 1.41.9 (22-Aug-2009)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
7159808 inodes, 28637862 blocks
1431893 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
874 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
 32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
 4096000, 7962624, 11239424, 20480000, 23887872
```

Advanced tools

There are several more advanced tools that you can use to examine or repair a filesystem. Check the man pages for the correct usage and the Linux Documentation Project (see Resources) for

how-to information. Almost all of these commands require a filesystem to be unmounted, although some functions can be used on filesystems that are mounted read-only. A few of the commands are described below.

You should always back up your filesystem before attempting any repairs.

Tools for ext2 and ext3 filesystems

tune2fs
: Adjusts parameters on ext2 and ext3 filesystems. Use this to add a journal to an ext2 system, making it an ext3 system, as well as display or set the maximum number of mounts before a check is forced. You can also assign a label and set or disable various optional features.

dumpe2fs
: Prints the super block and block group descriptor information for an ext2 or ext3 filesystem.

debugfs
: Is an interactive file system debugger. Use it to examine or change the state of an ext2 or ext3file system.

Tools for Reiserfs filesystems

reiserfstune
: Displays and adjusts parameters on ReiserFS filesystems.

debugreiserfs
: Performs similar functions to dumpe2fs and debugfs for ReiserFS filesystems.

Tools for XFS filesystems

xfs_info
: Displays XFS filesystem information.

xfs_growfs
: Expands an XFS filesystem (assuming another partition is available).

xfs_admin
: Changes the parameters of an XFS filesystem.

xfs_repair
: Repairs an XFS filesystem when the mount checks are not sufficient to repair the system.

xfs_db
: Examines or debugs an XFS filesystem.

We will wrap up our tools review with an illustration of the `debugfs` command, which allows you to explore the inner workings of an ext family filesystem. By default, it opens the filesystem in read-only mode. It does have many commands that allow you to attempt undeletion of files or directories, as well as other operations that require write access, so you will specifically have to enable write access with the `-w` option. Use it with extreme care. Listing 18 shows how to open the root filesystem on my system; navigate to my home directory; display information, including the inode number, about a file called index.html; and finally, map that inode number back to the pathname of the file.

Listing 18. Using debugfs

```
[root@echidna ~]# debugfs /dev/sdb9
```

```
debugfs 1.41.9 (22-Aug-2009)
debugfs:  cd home/ian
debugfs:  pwd
[pwd]   INODE: 165127  PATH: /home/ian
[root]  INODE:      2  PATH: /
debugfs:  stat index.html
Inode: 164815   Type: regular    Mode:  0644   Flags: 0x0
Generation: 2621469650    Version: 0x00000000
User:  1000   Group:  1000   Size: 14713
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 32
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x4bf1a3e9 -- Mon May 17 16:15:37 2010
atime: 0x4c619cf0 -- Tue Aug 10 14:39:44 2010
mtime: 0x4bf1a3e9 -- Mon May 17 16:15:37 2010
Size of extra inode fields: 4
Extended attributes stored in inode body:
  selinux = "unconfined_u:object_r:user_home_t:s0\000" (37)
BLOCKS:
(0-2):675945-675947, (3):1314836
TOTAL: 4

debugfs:  ncheck 164815
Inode Pathname
164815 /home/ian/index.html
debugfs:  q
```

Conclusion

We've covered many tools you can use for checking, modifying, and repairing your filesystems. Remember to always use extreme care when using the tools discussed in this article or any other tools. Data loss may be only a keystroke away.

Resources

Learn

- Develop and deploy your next app on the IBM Bluemix cloud platform.
- Use the developerWorks roadmap for LPIC-1 to find the developerWorks articles to help you study for LPIC-1 certification based on the April 2009 objectives.
- At the LPIC Program site, find detailed objectives, task lists, and sample questions for the three levels of the Linux Professional Institute's Linux system administration certification. In particular, see their April 2009 objectives for LPI exam 101 and LPI exam 102. Always refer to the LPIC Program site for the latest objectives.
- Review the entire LPI exam prep series on developerWorks to learn Linux fundamentals and prepare for system administrator certification based on earlier LPI exam objectives prior to April 2009.
- Learn more about tmpfs in "Common threads: Advanced filesystem implementor's guide, Part 3" (developerWorks, September 2001).
- The Linux Documentation Project has a variety of useful documents, especially its HOWTOs.
- In the developerWorks Linux zone, find hundreds of how-to articles and tutorials, as well as downloads, discussion forums, and a wealth of other resources for Linux developers and administrators.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools, as well as IT industry trends.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- Follow developerWorks on Twitter, or subscribe to a feed of Linux tweets on developerWorks.

Get products and technologies

- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the SOA Sandbox learning how to implement Service Oriented Architecture efficiently.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the My developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

Ian Shields

Ian Shields works on a multitude of Linux projects for the developerWorks Linux zone. He is a Senior Programmer at IBM at the Research Triangle Park, NC. He joined IBM in Canberra, Australia, as a Systems Engineer in 1973, and has since worked on communications systems and pervasive computing in Montreal, Canada, and RTP, NC. He has several patents and has published several papers. His undergraduate degree is in pure mathematics and philosophy from the Australian National University. He has an M.S. and Ph.D. in computer science from North Carolina State University. Learn more about Ian in Ian's profile on developerWorks Community.