Stanislav Komogorov, Fotolia

Configuring filesystems with mkfs, df, du, and fsck.

BUILDER

Although most Linux distributions today have simple-to-use graphical interfaces for setting up and managing filesystems, knowing how to perform those tasks from the command line is a valuable skill. We'll show you how to configure and manage filesystems with *mkfs*, *df*, *du*, and *fsck*. **BY NATHAN WILLIS**

Linux supports a wide array of filesystem types, including many that originated on other operating systems. The most common choices for hard disks, however, remain the native Linux ext2 and ext3 (the successor to ext2), followed by the high-performance XFS and ReiserFS filesystems. For compatibility, it is also important to know how to work with the VFAT filesystem, because it is the standard choice found pre-installed on many media, including USB thumb drives and flash disks.

Finally, several of the same utilities used to manage normal filesystems also apply to swap partitions, which the Linux kernel uses as virtual memory when RAM is scarce.

mkfs

The *mkfs* command (Figure 1) creates a new filesystem on a specified block device, such as a partition on a hard disk. The basic usage is

```
mkfs -t filesystem_type /the/device
```

where *filesystem_type* is a Linux-supported filesystem type (e.g., ext2 or xfs) and */the/device* is the location of the target disk partition (e.g., */dev/hda1* or */dev/sdc3*). Filesystem-specific options are added after *filesystem_type*.

mkfs hands off creation of the filesystem to one of several specialized utilities, depending on the filesystem type you specify: *mkfs.ext2*, *mkfs.xfs*, or *mkfs.vfat*, for example. Because filesystems differ so much from each other, having specialized tools maintained by experts in the individual filesystems results in more stable code.

Most of these utilities implement the same options, although they vary according to the features implemented in the different filesystems. Although *mkfs* calls these other utilities, note that you should generally run the standard *mkfs* command when creating a filesystem, instead of running any of the utilities directly.

Despite the differences, a few key options are common to all *mkfs.** utilities. Adding the *-c* flag will check the specified device for bad blocks, which will then be skipped over during the actual filesystem creation step. Adding the *-v* or *-V* flags produces verbose or extremely verbose output, respectively.

mkfs examples

To format the first partition of the first serial ATA drive on a system as ext3, you would run the command:

```
mkfs -t ext3 /dev/sda1
```

This command uses the default block size, inode parameters, and all other options, some of which are actually determined at run time when mkfs analyzes the geometry of the disk partition.

The following command

```
mkfs -t ext3 -b 4096 /dev/sda1
```

will also create an ext3 filesystem on */dev/sda1*, but it will force the use of 4,096-byte blocks. Running *mkfs -t ext3 -b 4096 -J device=/dev/sdb1 /dev/sda1* will create the same filesystem as the preceding command, but it will create the journal on a separate partition, */dev/sdb1*.

To create an XFS partition on */dev/sda1*, enter the following command:

```
mkfs -t xfs /dev/sda1
```

To specify the use of 4,096-byte blocks on this filesystem, use *mkfs -t xfs -b size=4096 /dev/sda1* – a different syntax than that used for ext3. Similarly, *mkfs -t reiserfs /dev/sda1* creates a ReiserFS filesystem with the default settings; to change the journal location to */dev/sdb1*, you would run:

```
mkfs -t reiserfs -j
  /dev/sdb1 /dev/sda1
```

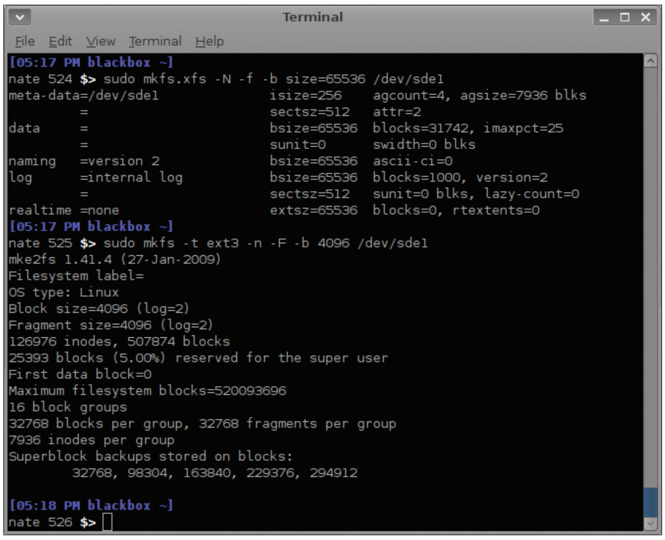

Figure 1: The simulated mkfs commands for XFS and ext3 differ. (The -N and -n flags specify a simulation, which does not actually create a filesystem.) The -f and -F flags tell mkfs to force filesystem creation, even if it detects a filesystem already in place.

The variations in syntax make it especially critical to refer to the man page for more on the use of *mkfs* with specific filesystem options.

Routine Maintenance

Although hard disk sizes increase every year, it still seems as though they fill up faster than they can be replaced. Running out of space on a filesystem is one of the most common problems you are likely to encounter on a Linux system, and it is not just an inconvenience for storage reasons – the system's use of temporary files means that a full or nearly full root filesystem could interfere with normal operations.

To check filesystem usage, use *df* (Figure 2). When given no arguments, *df* will return a table summarizing the usage of all of the mounted filesystems, in kilobytes and as a percentage of each filesystem's total size.

To get a report for a particular filesystem, specify it as an argument, such as *df /dev/sda1*. Also, you can pass a file name as an argument, and *df* will report on the filesystem that contains the specified file – which could be handy if you don't remember where a particular filesystem is mounted. Finally, a few options exist to make *df* more useful: *-i* reports inode usage instead of block usage of the filesystem(s); *-l* limits the report to local filesystems only; *--type=filesystem_type* and *--exclude-type=filesystem_type* allow you to limit or exclude output to a particular filesystem type, respectively.

On discovering a nearly full filesystem, you can further explore space usage with *du*. Executing *du /some/directory* will return a list of the disk space occupied by each subdirectory beneath */some/directory*, expressed in kilobytes. Adding the *-a* option tells *du* to report the space used by the files in addition to the directories.

Both commands are recursive. If you do not provide a directory as an argument to *du*, it will report on the current directory. The *-c* option produces a grand total in addition to individual usage statistics. Other option might help track down an errant large file, such as *-L*, which follows all symbolic links; *-x*, which limits the scope of the search to the current filesystem only; and *--max-depth=N*, which allows you to limit the number of recursive subdirectories into which you descend. This option is very helpful when dealing with a large file library.

Several utilities exist to help you get better performance out of your filesystems. The *tune2fs* program allows you to control many parameters of ext2 and ext3 filesystems; you can set the number of mounts between automatic filesystem integrity checks with *tune2fs -c N*, set the maximum time interval between checks with *tune2fs -i N[d|m|w]* (where

Filesystem Options

The ext2, ext3, XFS, and ReiserFS *mkfs.** utilities all support options that allow you to tweak filesystem settings, such as the size of the blocks used, the number and size of inodes, the fragment size, the amount of space reserved for use by root-privileged processes, the amount reserved to grow the group block descriptor if the filesystem ever needs to be resized, and settings for stripe, stride, and other details required for using the filesystem in a RAID array.

The good news is that all of these parameters have default settings, and unless you are sure you need to change them for a specific reason, it is safe to create a filesystem with the default settings. Nevertheless, it is a good idea to familiarize yourself with the basics of filesystem parameters in general, in case you ever run into problems.

The *block size* is the size of the chunks that the filesystem uses to store data – in a sense, it is the granularity of the pieces into which a file is split when stored on the disk. Ext2 can use 1,024, 2,048, or 4,096-byte blocks. Larger block sizes can improve disk throughput because the disk can read and write more data at a time before seeking to a new location, but a large block size can also waste space if there are a lot of small files, because a full block is consumed for each fragment of a file, even if only a small portion of it is actually needed. All four filesystems allow you to specify the block size by adding a *-b* flag, but the syntax that follows the flag varies, so you should consult the manual pages for each option.

Ext3, XFS, and ReiserFS all support filesystem journaling, which helps prevent filesystem corruption by maintaining a log of changes to files and directories. You can specify the size of the journal either in blocks or bytes, as well as whether the journal is kept on the same device as the filesystem or on a separate device. Here again, the exact syntax differs between filesystems, so check the manual page before proceeding.

VFAT filesystems differ quite a bit from the filesystems native to Linux and Unix-like operating systems. Because VFAT is simpler, you do not need to worry about inode numbers, sizes, fragment sizes, or RAID options. You can specify the number of reserved sectors, sectors per cluster, and sector size – options analogous to block settings in Unix-native filesystems – but in almost all cases, the defaults will suffice.

The *mkswap* command creates a swap area on a disk partition, just as *mkfs* creates a filesystem. The basic syntax is the same, *mkswap /the/swap/device*, with the optional *-c* flag again allowing you to check the partition for bad blocks before creating the swap area. Just as a new filesystem must be attached to Linux's root filesystem with *mount* before you can use it, a new swap partition must be attached with *swapon -L /the/swap/device*.

d, *m*, and *w* are days, months, and weeks, respectively), or add an ext3 journal to a filesystem that does not have one with *tune2fs -j*. Also, you can adjust RAID parameters, journal settings, and reserved block behavior, as well as change parameters manually, such as the time last checked and number of mounts, which are usually reported automatically.

ReiserFS has a *tunefs.reiserfs* utility in the same vein, although at present, its options are limited mostly to changing journaling options, including the journal size, location, and transaction size.

ReiserFS does have separate utilities for resizing filesystems (*resizefs.reiserfs*) and copying a filesystem from one device to another (*cpfs.reiserfs*). The *resize2fs* tool can resize both ext2 and ext3 filesystems. All of the resizing tools can both enlarge or shrink a filesystem, but enlarging a filesystem requires that the underlying disk partition have sufficient empty space – the filesystem cannot grow beyond the partition limit.

XFS has its own suite of utilities that cover many of the same options. The *xfs_growfs* command can resize an XFS filesystem, and it can make other changes as well. Using the *-m* option with *xfs_growfs*, you can change the amount of space reserved for inodes, and the *-l* and *-L* options allow you to make adjustments to the journal – both features found in other utilities in the other filesystems.

XFS also provides a defragmentation tool called *xfs_fsr* that can defragment a mounted XFS filesystem; no such utilities exist for ext2, ext3, or ReiserFS. Additionally, you can make backups and filesystem snapshots of XFS. The *xfs_freeze* tool freezes I/O on a filesystem. *xfsdump* writes a backup of a filesystem (in inode order, which allows it to be run on a mounted filesystem), and *xfsrestore* restores from a previous backup.

Troubleshooting

Linux checks each filesystem periodically and at boot time, looking for inconsistencies such as inodes that do not appear to be associated with a file, mismatches between the number of inode links and the inode link count, or mismatches between the number of free blocks in the filesystem and the expected number, as recorded in the superblock. All of these are indications that a crash or other problem might have occurred.

The utility that performs the check is *fsck*. If you suspect trouble on a filesystem, you can run

```
fsck /the/device
```

to perform a check manually and make any necessary repairs. If you run *fsck* with no target device specified, it will run checks sequentially on all of the filesystems in */etc/fstab*.

The filesystem-specific error-checking programs – *e2fsck* for ext2 and ext3, *reiserfsck* for ReiserFS, and *fsck.vfat* for VFAT (Figure 3) – support many of the same options, but again, the syntax may vary, so it is critical to read the man page for the filesystem checker before attempting any repairs.

When corrupted, VFAT filesystems suffer from a different set of problems: bad clusters, bad directory pointers, even bad file names. The *fsck.vfat* tool can find and correct many of these problems. Like the others, it can be called in non-interactive mode for use in scripts, and it can mark bad clusters automatically to prevent their reuse in the future. The *-V* flag tells *fsck.vfat* to run a second check after it has tried to correct any errors.

XFS has separate error-checking and repair utilities: *xfs_check* and *xfs_repair*. As with the other journaling filesystems, an option exists to point to the journal on an external device. Two helpful features unique to *xfs_check* are the *-f* flag, which runs the check on a filesystem image stored as a regular file (such as a filesystem backup created with *xfsdump*), and the *-s* flag, which specifies that only the most serious errors are reported. The *xfs_repair* tool can correct most of the same corruption problems tackled by *e2fsck*, *reiserfsck*, and the other *fsck.** utilities.

A more flexible examination of a problematic filesystem requires other programs. For ext2 and ext3 problems, the *debugfs* tool lets you examine a filesystem and correct errors interactively. *debugfs* can step through and work within a filesystem with commands similar to those of a typical Linux shell, such as *cd*, *open*, *close*, *pwd*, *mkdir*, and even *chroot*. Its real power comes from its ability to examine superblocks, blocks, and inodes directly, allocating and deallocating them individually, freeing blocks, and even creating links. ■

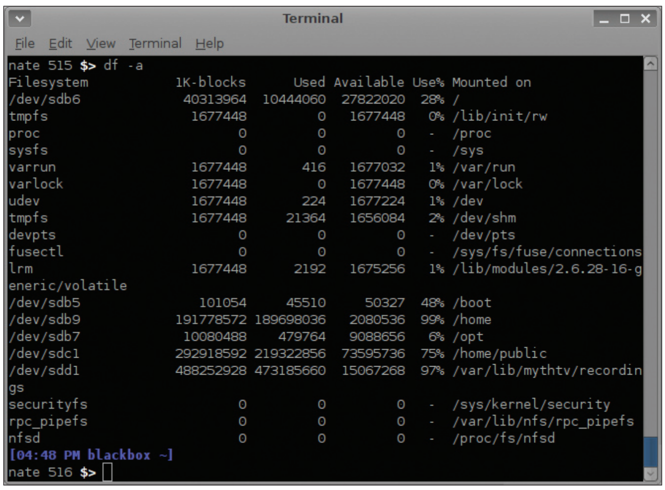

Figure 2: The results of a df command showing disk usage on a live system. The -a flag includes "dummy" filesystems like /proc in the output.

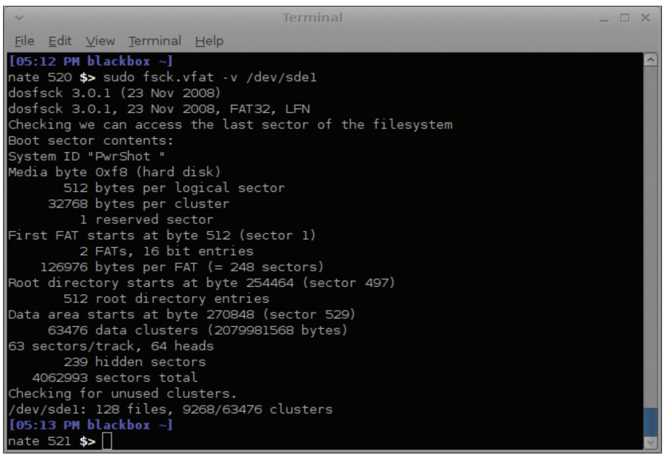

Figure 3: The results of an fsck with the default options on a VFAT filesystem. The -v flag gives verbose output. The journaling file systems (ext3 and ReiserFS) have options that allow you to specify the location of the journal if it is stored on a different device.