Linux Partition HOWTO

Table of Contents

Linux Partition HOWTO

Tony Harris
version 3.0, 1 May 2000
(added Partitioning with fdisk). All other sections written by
Kristan Koehntopp
Linux Partition HOWTO (version 2.4, 3 November 1997)

This Linux Mini−HOWTO teaches you how to layout disk space for your Linux system. It talks about disk hardware, partitions, swap space sizing and positioning considerations. file systems, file system types and related topics.

How to rebuild a deleted partition table

Appendix

What is a partition?

When PC hard disks were invented people soon wanted to install multiple operating systems, even if their system had only one disk. So a mechanism was needed to divide a single physical disk into multiple logical disks. So that's what a partition is: A contiguous section of blocks on your hard disk that is treated like a completely seperate disk by most operating systems.

It is fairly clear that partitions must not overlap: An operating system will certainly not be pleased, if another operating system installed on the same machine were overwriting important information because of overlapping partitions. There should be no gap between adjacent partitions, too. While this constellation is not harmful, you are wasting precious disk space by leaving space between partitions.

A disk need not be partitioned completely. You may decide to leave some space at the end of your disk that is not assigned to any of your installed operating systems, yet. Later, when it is clear which installation is used by you most of the time, you can partition this left over space and put a file system on it.

Partitions can not be moved nor can they be resized without destroying the file system contained in it. So repartitioning usually involves backup and restore of all file systems touched during the repartitioning. In fact it is fairly common to mess up things completely during repartitioning, so you should back up anything on any disk on that particular machine before even touching things like `fdisk`.

Well, some partitions with certain file system types on them actually *can* be split into two without losing any data (if you are lucky). For example there is a program called "fips" for splitting MS−DOS partitions into two to make room for a Linux installation without having to reinstall MS−DOS. You are still not going to touch these things without carefully backing up everything on that machine, aren't you?

Related HOWTOs

To learn how to estimate the various size and speed requirements for different parts of the filesystem, see "Linux Multiple Disks Layout mini−HOWTO", by Gjoen Stein.

For instructions and considerations regarding disks with more than 1024 cylinders, see "Linux Large Disk mini−HOWTO", AndriesBrouwer.

For instructions on limiting disk space usage per user (quotas), see "Linux Quota mini−HOWTO", by Albert M.C. Tam <bertie@scn.org>

Currently, there is no general document on disk backup, but there are several documents with pointers to specific backup solutions. See "Linux ADSM Backup mini−HOWTO", by Thomas Koenig <Thomas.Koenig@ciw.uni−karlsruhe.de> for instructions on integrating Linux into an IBM ADSM backup environment. See "Linux Backup with MSDOS mini−HOWTO", by Christopher Neufeld <neufeld@physics.utoronto.ca> for information about MS−DOS driven Linux backups.

For instructions on writing and submitting a HOWTO document, see the Linux HOWTO Index, by Tim Bynum <linux−howto@sunsite.unc.edu>.

Browsing through /usr/src/linux/Documentation can be very instructive, too. See ide.txt and scsi.txt for some background information on the properties of your disk drivers and have a look at the filesystems/ subdirectory.

Device numbers and device names

In Linux, partitions are represented by device files. A device file is a file with type c (for "character" devices, devices that do not use the buffer cache) or b (for "block" devices, which go through the buffer cache). In Linux, all disks are represented as block devices only. Unlike other Unices, Linux does not offer "raw" character versions of disks and their partitions.

The only important thing with a device file are its major and minor device number, shown instead of the files size:

```
$ ls −l /dev/hda
brw−rw−−−−   1 root     disk       3,   0 Jul 18  1994 /dev/hda
                                    ^    ^
                                    |    minor device number
                                   major device number
```

When accessing a device file, the major number selects which device driver is being called to perform the input/output operation. This call is being done with the minor number as a parameter and it is entirely up to the driver how the minor number is being interpreted. The driver documentation usually describes how the driver uses minor numbers. For IDE disks, this documentation is in `/usr/src/linux/Documentation/ide.txt`. For SCSI disks, one would expect such documentation in `/usr/src/linux/Documentation/scsi.txt`, but it isn't there. One has to look at the driver source to be sure (`/usr/src/linux/driver/scsi/sd.c:184−196`). Fortunately, there is Peter Anvin's list of device numbers and names in

`/usr/src/linux/Documentation/devices.txt`; see the entries for block devices, major 3, 22, 33, 34 for IDE and major 8 for SCSI disks. The major and minor numbers are a byte each and that is why the number of partitions per disk is limited.
device files have certain names and many system programs have knowledge about these names compiled in. They expect your IDE disks to be named `/dev/hd*` and your SCSI disks to be named `/dev/sd*`. Disks are numbered a, b, c and so on, so `/dev/hda` is your first IDE disk and `/dev/sda` is your first SCSI disk. Both devices represent entire disks, starting at block one. Writing to these devices with the wrong tools will destroy the master boot loader and partition table on these disks, rendering all data on this disk unusable or making your system unbootable. Know what you are doing and, again, back up before you do it.

Primary partitions on a disk are 1, 2, 3 and 4. So `/dev/hda1` is the first primary partition on the first IDE disk and so on. Logical partitions have numbers 5 and up, so `/dev/sdb5` is the first logical partition on the second SCSI disk.

Each partition entry has a starting and an ending block address assigned to it and a type. The type is a numerical code (a byte) which designates a particular partition to a certain type of operating system. For the benefit of computing consultants partition type codes are not really unique, so there is always the probability of two operating systems using the same type code.

Linux reserves the type code 0x82 for swap partitions and 0x83 for "native" file systems (that's ext2 for almost all of you). The once popular, now outdated Linux/Minix file system used the type code 0x81 for partitions. OS/2 marks it's partitions with a 0x07 type and so does Windows NT's NTFS. MS−DOS allocates several type codes for its various flavors of FAT file systems: 0x01, 0x04 and 0x06 are known. DR−DOS used 0x81 to indicate protected FAT partitions, creating a type clash with Linux/Minix at that time, but neither Linux/Minix nor DR−DOS are widely used any more. The extended partition which is used as a container for logical partitions has a type of 0x05, by the way.

Partitions are created and deleted with the `fdisk` program. Every self respecting operating system program comes with an `fdisk` and traditionally it is even called `fdisk` (or `FDISK.EXE`) in almost all OSes. Some `fdisk`s, noteable the DOS one, are somehow limited when they have to deal with other operating systems partitions. Such limitations include the complete inability to deal with anything with a foreign type code, the inability to deal with cylinder numbers above 1024 and the inability to create or even understand partitions that do not end on a cylinder boundary. For example, the MS−DOS fdisk can't delete NTFS partitions, the OS/2 fdisk has been known to silently "correct" partitions created by the Linux fdisk that do not end on a cylinder boundary and both, the DOS and the OS/2 fdisk, have had problems with disks with more than 1024 cylinders (see the "large−disk" Mini−Howto for details on such disks).

Partition Types

Primary partitions on a disk are 1, 2, 3 and 4. So `/dev/hda1` is the first primary partition on the first IDE disk and so on. Logical partitions have numbers 5 and up, so `/dev/sdb5` is the first logical partition on the second SCSI disk.

Each partition entry has a starting and an ending block address assigned to it and a type. The type is a numerical code (a byte) which designates a particular partition to a certain type of operating system. For the benefit of computing consultants partition type codes are not really unique, so there is always the probability of two operating systems using the same type code.

Linux reserves the type code 0x82 for swap partitions and 0x83 for "native" file systems (that's ext2 for almost all of you). The once popular, now outdated Linux/Minix file system used the type code 0x81 for partitions. OS/2 marks it's partitions with a 0x07 type and so does Windows NT's NTFS. MS−DOS allocates several type codes for its various flavors of FAT file systems: 0x01, 0x04 and 0x06 are known. DR−DOS used 0x81 to indicate protected FAT partitions, creating a type clash with Linux/Minix at that time, but neither Linux/Minix nor DR−DOS are widely used any more. The extended partition which is used as a container for logical partitions has a type of 0x05, by the way.

Partitions are created and deleted with the `fdisk` program. Every self respecting operating system program comes with an `fdisk` and traditionally it is even called `fdisk` (or `FDISK.EXE`) in almost all OSes. Some `fdisk`s, noteable the DOS one, are somehow limited when they have to deal with other operating systems partitions. Such limitations include the complete inability to deal with anything with a foreign type code, the inability to deal with cylinder numbers above 1024 and the inability to create or even understand partitions that do not end on a cylinder boundary. For example, the MS−DOS fdisk can't delete NTFS partitions, the OS/2 fdisk has been known to silently "correct" partitions created by the Linux fdisk that do not end on a cylinder boundary and both, the DOS and the OS/2 fdisk, have had problems with disks with more than 1024 cylinders (see the "large−disk" Mini−Howto for details on such disks).

The number of partitions on an Intel based system was limited from the very beginning: The original partition table was installed as part of the boot sector and held space for only four partition entries. These partitions are now called primary partitions. When it became clear that people needed more partitions on their systems, logical partitions were invented. The number of logical partitions is not limited: Each logical partition contains a pointer to the next logical partition, so you can have a potentially unlimited chain of partition entries.

For compatibility reasons, the space occupied by all logical partitions had to be accounted for. If you are using logical partitions, one primary partition entry is marked as "extended partition" and its starting and ending block mark the area occupied by your logical partitions. This implies that the space assigned to all logical partitions has to be contiguous. There can be only one extended partition: no `fdisk` program will create more than one extended partition.

Linux cannot handle more than a limited number of partitions per drive. So in Linux you have 4 primary partitions (3 of them useable, if you are using logical partitions) and at most 15 partitions altogether on an SCSI disk (63 altogether on an IDE disk).

What Partitions do I need?

Okay, so what partitions do you need? Well, some operating systems do not believe in booting from logical partitions for reasons that are beyond the scope of any sane mind. So you probably want to reserve your primary partitions as boot partitions for your MS−DOS, OS/2 and Linux or whatever you are using. Remember that one primary partition is needed as an extended partition, which acts as a container for the rest of your disk with logical partitions.

Booting operating systems is a real−mode thing involving BIOSes and 1024 cylinder limitations. So you probably want to put all your boot partitions into the first 1024 cylinders of your hard disk, just to avoid problems. Again, read the "large−disk" Mini−Howto for the gory details.

To install Linux, you will need at least one partition. If the kernel is loaded from this partition (for example by LILO), this partition must be readable by your BIOS. If you are using other means to load your kernel (for example a boot disk or the LOADLIN.EXE MS−DOS based Linux loader) the partition can be anywhere. In any case this partition will be of type 0x83 "Linux native".

Your system will need some swap space. Unless you swap to files you will need a dedicated swap partition. Since this partition is only accessed by the Linux kernel and the Linux kernel does not suffer from PC BIOS deficiencies, the swap partition may be positioned anywhere. I recommed using a logical partition for it (/dev/?d?5 and higher). Dedicated Linux swap partitions are of type 0x82 "Linux swap".

These are minimal partition requirements. It may be useful to create more partitions for Linux. Read on.

How large should my swap space be?

If you have decided to use a dedicated swap partition, which is generally a Good Idea [tm], follow these guidelines for estimating its size:

- In Linux RAM and swap space add up (This is not true for all Unices). For example, if you have 8 MB of RAM and 12 MB swap space, you have a total of about 20 MB virtual memory.
- When sizing your swap space, you should have at least 16 MB of total virtual memory. So for 4 MB of RAM consider at least 12 MB of swap, for 8 MB of RAM consider at least 8 MB of swap.
- In Linux, a single swap partition can not be larger than 128 MB. That is, the partition may be larger than 128 MB, but excess space is never used. If you want more than 128 MB of swap, you have to create multiple swap partitions.
- When sizing swap space, keep in mind that too much swap space may not be useful at

all.

Every process has a "working set". This is a set of in−memory pages which will be referenced by the processor in the very near future. Linux tries to predict these memory accesses (assuming that recently used pages will be used again in the near future) and keeps these pages in RAM if possible. If the program has a good "locality of reference" this assumption will be true and prediction algorithm will work.

Holding a working set in main memory does only work if there is enough main memory. If you have too many processes running on a machine, the kernel is forced to put pages on disk that it will reference again in the very near future (forcing a page−out of a page from another working set and then a page−in of the page referenced). Usually this results in a very heavy increase in paging activity and in a sustantial drop of performance. A machine in this state is said to be "thrashing" (For you German readers: That's "thrashing" ("dreschen", "schlagen", "haemmern") and not trashing ("muellen")).

On a thrashing machine the processes are essentially running from disk and not from RAM. Expect performance to drop by approximately the ratio between memory access speed and disk access speed.

A very old rule of thumb in the days of the PDP and the Vax was that the size of the working set of a program is about 25% of its virtual size. Thus it is probably useless to provide more swap than three times your RAM.

But keep in mind that this is just a rule of thumb. It is easily possible to create scenarios where programs have extremely large or extremely small working sets. For example, a simulation program with a large data set that is accessed in a very random fashion would have almost no noticeable locality of reference in its data segment, so its working set would be quite large.

On the other hand, an xv with many simultaneously opened JPEGs, all but one iconified, would have a very large data segment. But image transformations are all done on one single image, most of the memory occupied by xv is never touched. The same is true for an editor with many editor windows where only one window is being modified at a time. These programs have − if they are designed properly − a very high locality of reference and large parts of them can be kept swapped out without too severe performance impact.

One could suspect that the 25% number from the age of the command line is no longer true for modern GUI programs editing multiple documents, but I know of no newer papers that try to verify these numbers.

So for a configuration with 16 MB RAM, no swap is needed for a minimal configuration and more than 48 MB of swap are probably useless. The exact amount of memory needed depends on the application mix on the machine (what did you expect?).

Where should I put my swap space?

- Mechanics are slow, electronics are fast.
  Modern hard disks have many heads. Switching between heads of the same track is fast, since it is purely electronic. Switching between tracks is slow, since it involves moving real world matter.

  So if you have a disk with many heads and one with less heads and both are identical in other parameters, the disk with many heads will be faster.

Splitting swap and putting it on both disks will be even faster, though.

- Older disks have the same number of sectors on all tracks. With this disks it will be fastest to put your swap in the middle of the disks, assuming that your disk head will move from a random track towards the swap area.
- Newer disks use ZBR (zone bit recording). They have more sectors on the outer tracks. With a constant number of rpms, this yields a far greater performance on the outer tracks than on the inner ones. Put your swap on the fast tracks.
- Of course your disk head will not move randomly. If you have swap space in the middle of a disk between a constantly busy home partition and an almost unused archive partition, you would be better of if your swap were in the middle of the home partition for even shorter head movements. You would be even better off, if you had your swap on another otherwise unused disk, though.

Summary: Put your swap on a fast disk with many heads that is not busy doing other things. If you have multiple disks: Split swap and scatter it over all your disks or even different controllers.

Even better: Buy more RAM.

Partitioning with fdisk

This section shows you how to actually partition your hard drive with the fdisk utility. Linux allows only 4 primary partitions. You can have a much larger number of logical partitions by sub−dividing one of the primary partitions. Only one of the primary partitions can be sub−divided.
Examples:

1. Four primary partitions
2. Mixed primary and logical partitions

Notes about fdisk: fdisk is started by typing (as root) "fdisk *device*" at the command prompt. "*device*" might be something like /dev/hda or /dev/sda. The basic fdisk commands you need are:

p print the partition table
n create a new partition
d delete a parition
q quit without saving changes
w write the new partition table
and exit

Changes you make to the partition table do not take effect until you issue the write (w) command. Here is a sample partition table:

```
Disk /dev/hdb: 64 heads, 63 sectors, 621 cylinders
```

```
Units = cylinders of 4032 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hdb1   *         1       184    370912+  83  Linux
/dev/hdb2           185       368    370944   83  Linux
/dev/hdb3           369       552    370944   83  Linux
/dev/hdb4           553       621    139104   82  Linux swap
```

The first line shows the geometry of your hard drive. It may not be physically accurate, but you can accept it as though it were. The hard drive in this example is made of 32 double−sided platters with one head on each side (probably not true). Each platter has 621 concentric tracks. A 3−dimensional track (the same track on all disks) is called a cylinder. Each track is divided into 63 sectors. Each sector contains 512 bytes of data. Therefore the block size in the partion table is 64 heads * 63 sectors * 512 bytes.
The start and end values are cylinders. The first cylinder (0) is reserved for information about the drive layout.

Four primary partitions

*The overview:*Decide on the size of your swap space and where it ought to go. Divide up the remaining space for the three other partitions.
Example:
I start fdisk from the shell prompt:
`fdisk /dev/hdb`
which indicates that I am using the second drive on my IDE controller. (See device numbers.) Now I need to plan my layout. In this example, I want to use only primary partitions for my linux partitions and my swap space. When I print the (empty) partition table, I just get configuration information.

```
Command (m for help): p

Disk /dev/hdb: 64 heads, 63 sectors, 621 cylinders
Units = cylinders of 4032 * 512 bytes
```

I knew that I had a 1.2Gb drive, but now I really know: 64 * 63 * 512 * 621 = 1281982464 bytes. I decide to reserve 128Mb of that space for swap, leaving 1153982464. If I use one of my primary partitions for swap, that means I have three left for ext2 partitions. Divided equally, that makes for 384Mb per partition. Now I get to work.

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1−4)
p
Partition number (1−4): 1
First cylinder (1−621, default 1):RETURN>
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1−621, default 621): +384M
```

Next, I set up the partition I want to use for swap:

```
Command (m for help): n
Command action
   e   extended
```

```
   p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (197-621, default 197):
Using default value 197
Last cylinder or +size or +sizeM or +sizeK (197-621, default 621): +128M
```

Now the partition table looks like this:

```
   Device Boot    Start       End    Blocks   Id  System
/dev/hdb1             1       196    395104   83  Linux
/dev/hdb2           197       262    133056   83  Linux
```

I set up the remaining two partitions the same way I did the first. Finally, I make the first partition bootable:

```
Command (m for help): a
Partition number (1-4): 1
```

And I make the second partition of type swap:

```
Command (m for help): t
Partition number (1-4): 2
Hex code (type L to list codes): 82
Changed system type of partition 2 to 82 (Linux swap)
Command (m for help): p
```

The end result:

```
Disk /dev/hdb: 64 heads, 63 sectors, 621 cylinders
Units = cylinders of 4032 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hdb1   *         1       196    395104+  83  Linux
/dev/hdb2           197       262    133056   82  Linux swap
/dev/hdb3           263       458    395136   83  Linux
/dev/hdb4           459       621    328608   83  Linux
```

Finally, I issue the write command (w) to write the table on the disk.

[How to format partitions]
[How to mount partitions]
[How to activate swap space]

Mixed primary and logical partitions

The overview: create one use one of the primary partitions to house all the extra partitions. Then create logical partitions within it. Create the other primary partitions before or after creating the logical partitions.

Example:

I start fdisk from the shell prompt:

`fdisk /dev/sda` which indicates that I am using the first drive on my SCSI chain. (See [device numbers].

First I figure out how many partitions I want. I know my drive has a 183Gb capacity and I want 26Gb partitions (because I happen to have back-up tapes that are about that size).

183Gb / 26Gb = ~7

so I will need 7 partitions. Even though fdisk accepts partition sizes expressed in Mb

and Kb, I decide to calculate the number of cylinders that will end up in each partition because fdisk reports start and stop points in cylinders. I see when I enter fdisk that I have 22800 cylinders.

```
> The number of cylinders for this disk is set to 22800.  There is
> nothing wrong with that, but this is larger than 1024, and could in
> certain setups cause problems with: 1) software that runs at boot
> time (e.g., LILO) 2) booting and partitioning software from other
> OSs  (e.g., DOS FDISK, OS/2 FDISK)
```

So, 22800 total cylinders divided by seven partitions is 3258 cylinders. Each partition will be about 3258 cylinders long. I ignore the warning msg because this is not my boot drive.

Since I have 4 primary partitions, 3 of them can be 3258 long. The extended partition will have to be (4 * 3258), or 13032, blocks long in order to contain the 4 logical partitions.

I enter the following commands to set up the first of the 3 primary partitions (stuff I type is bold):

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4):
First cylinder (1-22800, default 1): RETURN>
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-22800, default 22800):
```

The last partition is the extended partition:

```
Partition number (1-4): 4
First cylinder (9775-22800, default 9775): RETURN>
Using default value 9775
Last cylinder or +size or +sizeM or +sizeK (9775-22800, default 22800): RETURN>
Using default value 22800
```

The result, when I issue the print table command is:

```
/dev/sda1             1      3258  26169853+  83  Linux
/dev/sda2          3259      6516  26169885   83  Linux
/dev/sda3          6517      9774  26169885   83  Linux
/dev/sda4          9775     22800 104631345    5  Extended
```

Next I segment the extended partition into 4 logical partitions, starting with the first logical partition, into 3258-cylinder segments. The logical partitions automatically start from /dev/sda5.

```
Command (m for help):  n
First cylinder (9775-22800, default 9775): RETURN>
Using default value 9775
Last cylinder or +size or +sizeM or +sizeK (9775-22800, default 22800): 13032
```

The end result is:

```
   Device Boot    Start       End    Blocks   Id  System
/dev/sda1             1      3258  26169853+  83  Linux
/dev/sda2          3259      6516  26169885   83  Linux
/dev/sda3          6517      9774  26169885   83  Linux
/dev/sda4          9775     22800 104631345    5  Extended
/dev/sda5          9775     13032  26169853+  83  Linux
/dev/sda6         13033     16290  26169853+  83  Linux
```

```
/dev/sda7         16291     19584  26459023+  83  Linux
/dev/sda8         19585     22800  25832488+  83  Linux
```

Finally, I issue the write command (w) to write the table on the disk. The next thing I need to do is format each partition.

Formating Partitions

At the shell prompt, I begin making the file systems on my partitions. Continuing with the previous example, this is:

mke2fs /dev/sda1

I need to do this for each of my partitions, but not for /dev/sda4 (my extended partition). If I want non−ext2 partitions, this is the time to make that decision. The kinds of file systems your kernel supports is in:

/usr/src/linux/include/linux/fs.h

The most common file systems can be made with programs in /sbin that start with "mk" like mkfs.msdos and mke2fs.

To set up a swap partition:

```
# mkswap −f /dev/hda5
```

To activate the swap area:

```
# swapon /dev/hda5
```

Normally, the swap area is activated by the initialization scripts at boot time.

Mounting Partitions

Mounting a partition means attaching it to the linux file system. To mount a linux partition:

```
# mount −t ext2 /dev/sda1 /opt
```

−t ext2	File system type. Other types you are likely to use are: ♦ msdos (DOS) ♦ hfs (mac) ♦ iso9660 (CDROM) ♦ nfs (network file system)
/dev/sda1	Device name. Other device names you are likely to use: ♦ /dev/hdb2 (second partition in second IDE drive) ♦ /dev/fd0 (floppy drive A) ♦ /dev/cdrom (CDROM)
/opt	mount point. This is where you want to "see" your partition. When you type `ls /opt`, you can see what is in /dev/sda1. If there are already some directories and/or files under /opt, they will be invisible after this mount command.