NAME

installboot − make a device bootable

SYNOPSIS

installboot −i(mage) *image* [*label*:]*kernel mm fs ... init*
installboot −(e)x(tract) *image*
installboot −d(evice) *device bootblock boot* [[*label*:]*image* ...]
installboot −b(oot) *device bootblock boot* [*label*:]*image* ...
installboot −m(aster) *device masterboot* [*keys* [*logical*]]

DESCRIPTION

Installboot may be used to make a device bootable by constructing a kernel image and installing bootstrap code into the boot block of a Minix file system. To understand how this can be done one first has to know what happens when a PC is booted.

When the power is turned on the typical PC will try to read the first sector from the first floppy disk or from the first hard disk into memory and execute it. The code obtained from the hard disk (from the so-called master boot sector) will immediately replace itself by the code found in the first sector of the active partition. Thus the PC is now executing the bootstrap code found in the first sector of /dev/fd0, /dev/c0d0p0, /dev/c0d0p1, /dev/c0d0p2, or /dev/c0d0p3 (assuming the boot disk is attached to controller 0.) The bootstrap will locate the operating system on the device it itself was loaded from, load it, and execute it.

To make a Minix file system **/dev/fd0** mounted on **/mnt** bootable, enter the following:

cp /usr/mdec/boot /mnt/boot

installboot −i /mnt/minix kernel mm fs init

installboot −d /dev/fd0 /usr/mdec/bootblock boot

The "boot" program in the example is named the "Boot Monitor". It is loaded by the bootblock code placed in the boot sector of /dev/fd0 and it will take care of loading the kernel image "minix" from the root directory of the file system. See **monitor**(8) for a description of the Boot Monitor. Note that **boot** is a name in the file system on **/dev/fd0** in this example, the same file as **/mnt/boot**. Making **/mnt/minix** is normally not necessary, there is usually a kernel image in the **tools** directory.

OPTIONS

−i(mage) *image* [*label*:]*kernel mm fs ... init*

The **−image** option (or the **−i** shorthand) combines the executable files needed to run Minix in one file. Only the names and a few zero bytes are inserted into the image. The name is for identification and the zeros are used to pad separate pieces to sector boundaries for fast loading.

An executable may be prefixed by a label. The Monitor may be instructed to load processes by label. So more than one kernel process may be included in the image, each with a different winchester driver for instance. So if you have compiled two different kernels with an AT or XT driver then

installboot −i *image AT:at_kernel XT:xt_kernel mm fs init*

will make an image with two different labeled kernels and one unlabeled set of the other binaries.

−(e)x(tract) *image*

Extract the binaries from *image* under the names stored in the image. (The name includes the optional label.)

−d(evice) *device bootblock boot* [[*label*:]*image* ...]

Installs *bootblock* in the boot sector of *device* together with the disk addresses to *boot*. These disk addresses are needed to load *boot* from the file system at boot time. The argument *boot* is first searched in the file system on *device*. If it is not found then it is read as a normal file and added at the end of the file system. The file system should be smaller than the device it is on to allow this. Any extra images are also added to the end as described under **−boot**. (Make sure you understand all this.)

The device need not be mounted when **installboot** is run, nor does it matter if it is.

Installboot needs to be run again if *boot* is rewritten, because it will then occupy a new place on the disk.

Old boot parameters are kept if there are no images added.

−b(oot) *device bootblock boot* [*label*:]*image ...*

This option fills a blank floppy in *device* with boot code and kernel images. This "boot disk" does not have a root file system, only the Boot Monitor and Minix kernels. The boot parameters sector is filled with code that enables menu options for selecting an image. After loading an image, the Monitor will ask you to insert a root file system diskette before starting Minix.

The labels used on the images should match those on the executables used inside the image. You can put a comma separated list of labels on an image for each label used within the image. For the image created earlier one would create a boot floppy like this:

> **installboot −b /dev/fd0 bootblock boot** *AT,XT:image*

If a label-list is omitted on an image, then that image will be selected by default. (Like in the normal one image, no labels case.)

Note that **−device** and **−boot** together allow you to make a boot floppy with or without a root file system. With the boot code in the file system, attached to the end of it, or after the boot block. And with one or more kernel images in the file system or at the end of the device. Somewhat confusing.

−m(aster) *device masterboot* [*keys* [*logical*]]

This option installs the *masterboot* program into the boot sector of the given device. If another device is given instead of *masterboot* then its bootstrap code is copied to *device*. The master bootstrap on a hard disk boots the active partition on that disk at boot time. The MS-DOS fdisk command normally puts a master bootstrap on the hard disk. Minix has two bootstraps that can be used as a master bootstrap, **masterboot** and **jumpboot.**

Masterboot is a fairly normal master bootstrap that works as follows:

> If installed on a hard disk then it will load the bootstrap of the active partition and run it. **Masterboot** can be put in the first sector of a hard disk to boot the active partition, or in the first sector of a Minix partition to boot the active subpartition.
>
> If installed on a Minix floppy then it will try to boot the next floppy or the first hard disk. Ideal for floppies with just data on it, they will no longer obstruct the boot process if left in the drive. Also a very useful trick to boot from floppy drive 1.

The other bootstrap named **jumpboot** is used for the weird cases:

> If your default operating system is installed on another disk then **jumpboot** can be installed on the first disk and instructed to boot the disk, partition or subpartition that must be booted by default.
>
> If one of your operating systems insists on being active when booted then use **jumpboot** to ignore the active flag and boot your preferred O.S. instead. The Boot Monitor's "**boot** ∗" trick to activate the partition to boot is useful here.
>
> To boot a logical partition within an extended partition. Note that you can put **jumpboot** in the first sector of the extended partition in this case, with the extended partition marked active.
>
> If you hold down the ALT key while **jumpboot** is being executed, then you can type the disk, partition or subpartition you want to boot as one to three digits followed by typing ENTER.

Jumpboot can be programmed to boot a certain partition with the *keys* argument and optionally also the *logical* argument. *Keys* are one to three digits naming the disk, partition or subpartition. If the device to boot is **/dev/c0d1p3s0**, then *keys* is **130**. These are the same three digits you can type manually if you hold down ALT at boot. To program **jumpboot** to boot a logical partition within an extended partition, let *keys* be just a disk number, and specify *logical* as the name of the logical partition on that disk that is to be booted. (Actually *logical* can be any device name, but this form should be avoided because it offers less checking to see if the device is still there after a disk rearrangement.)

A backup copy of the current master bootstrap (including the partition table) can be made with:

dd if=*device* of=*backup-file* count=1

A simple 'cp *backup-file device*' will put it back. You can also use **fdisk /mbr** under MS-DOS 5.0 (or newer) to restore the master bootstrap.

FILES

/usr/mdec/bootblock Minix bootstrap for the Minix root device. To be placed in the boot sector.

/usr/mdec/boot Minix Boot Monitor. Can usually be found in the root directory of a bootable device.

/usr/mdec/masterboot Master bootstrap. Can be placed in the first sector of a disk to select the active partition. In a Minix primary partition it selects the active subpartition.

/usr/mdec/jumpboot Special "boot this" bootstrap.

SEE ALSO

part(8), **monitor**(8).

DIAGNOSTICS

File is not an executable
What you think is boot code or part of the kernel isn't.

Program will crash, text/data segment larger then 64K
One of the 16-bit programs added to an image has a text or data segment that is larger than 64K. You probably enabled too many drivers, or configured too many buffers.

File can't be attached to *device*
You are trying to put the boot monitor or an image after a file system, but there is no or not enough space. Did you specify the full path of the monitor instead of just "boot"?

Device is not a Minix file system
You are using **−device** on a device that doesn't contain a file system. Maybe you specified the wrong device, maybe you should make a file system, or maybe you should use **−boot**.

Device contains a file system
You are about to destroy a file system with **−boot**. Maybe you meant to use **−device**? You have 10 seconds to make up your mind...

File is too big
Several types of messages like these will tell you that *file* can't be installed in a boot sector, or that there is no room to add some parameters, etc. Is *file* really a bootstrap?

Bootstrap errors

Read error
A read error trying to get the next bit of boot code. You may even get the BIOS error code in hex. Either the device has a bad block, or jumpboot is told to read a nonexistent disk.

No active partition
None of the partitions in a partition table is marked active.

Not bootable
Partition does not exist (jumpboot), or it's bootstrap isn't executable.

NOTES

The Minix bootstraps can boot beyond the 8G disk size limit if the BIOS supports the IBM/MS INT 13 Extensions. Alas only Minix-vmd can make use of this, standard Minix has a 4G disk size limit.

BUGS

It has four more options than the SunOS installboot program it is modeled after.

The bootblock code has been crunched to such ugliness that you can use it to scare little kids out of your garden.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)