# Linux BootPrompt-Howto

Paul Gortmaker, Editor. v1.01, 18 August 1995

This is the BootPrompt-Howto, which is a compilation of all the possible boot time arguments that can be passed to the Linux kernel at boot time. This includes all kernel and device parameters. An overview of some of the popular software used to boot Linux kernels is included.

## Contents

**1 Introduction** **3**

1.1 Disclaimer and Copyright . . . . . 4

1.2 Related Documentation . . . . . 4

1.3 The Linux Newsgroups . . . . . 4

1.4 New Versions of this Document . . . . . 5

**2 Overview of Boot Prompt Arguments** **5**

2.1 LILO (LInux LOader) . . . . . 5

2.2 LoadLin . . . . . 5

2.3 The "rdev" utility . . . . . 6

2.4 How the Kernel Sorts the Arguments . . . . . 6

2.5 Setting Environment Variables. . . . . . 6

2.6 Passing Arguments to 'init' . . . . . 7

**3 General Non-Device Specific Boot Args** **7**

3.1 The 'no387' Argument . . . . . 7

3.2 The 'no-hlt' Argument . . . . . 7

3.3 The 'root=' Argument . . . . . 7

3.4 The 'ro' Argument . . . . . 8

3.5 The 'rw' Argument . . . . . 8

3.6 The 'debug' Argument . . . . . 8

3.7 The 'reserve=' Argument . . . . . 9

3.8 The 'ramdisk=' Argument . . . . . 9

3.9 The 'mem=' Argument . . . . . 9

**4 Boot Arguments for SCSI Peripherals. 10**

4.1 Maximum Probed LUNs (‘max_scsi_luns=’) . . . . . . . . . . 10
4.2 Parameters for SCSI Tape drives (‘st=’) . . . . . . . . . . 11
4.3 Adaptec aha151x, aha152x, aic6260, aic6360, SB16-SCSI (‘aha152x=’) . . . . . . . . . . 11
4.4 Adaptec aha154x (‘aha1542=’) . . . . . . . . . . 12
4.5 Adaptec aha274x, aha284x, aic7xxx (‘aic7xxx=’) . . . . . . . . . . 12
4.6 BusLogic SCSI Hosts (‘buslogic=’) . . . . . . . . . . 12
4.7 Future Domain TMC-8xx, TMC-950 (‘tmc8xx=’) . . . . . . . . . . 12
4.8 Pro Audio Spectrum (‘pas16=’) . . . . . . . . . . 13
4.9 Seagate ST-0x (‘st0x=’) . . . . . . . . . . 13
4.10 Trantor T128 (‘t128=’) . . . . . . . . . . 13
4.11 Cards that don’t Accept Boot Args . . . . . . . . . . 13

**5 Hard Disks 13**

5.1 IDE Disk/CD-ROM Driver Parameters . . . . . . . . . . 14
5.1.1 The ‘hd=cyls,heads,sects[,wpcom[,irq]]’ options . . . . . . . . . . 14
5.1.2 The ‘hd=serialize’ option . . . . . . . . . . 14
5.1.3 The ‘hd=dtc2278’ option . . . . . . . . . . 14
5.1.4 The ‘hd=noprobe’ option . . . . . . . . . . 14
5.1.5 The ‘hd=nowerr’ option . . . . . . . . . . 15
5.1.6 The ‘hd=cdrom’ option . . . . . . . . . . 15
5.2 Standard ST-506 Disk Driver Options (‘hd=’) . . . . . . . . . . 15
5.3 XT Disk Driver Options (‘xd=’) . . . . . . . . . . 15

**6 CD-ROMs (Non-SCSI/ATAPI/IDE) 15**

6.1 The Aztech Interface (‘aztcd=’) . . . . . . . . . . 16
6.2 The CDU-31A and CDU-33A Sony Interface (‘cdu31a=’) . . . . . . . . . . 16
6.3 The CDU-535 Sony Interface (‘sonycd535=’) . . . . . . . . . . 16
6.4 The GoldStar Interface (‘gscd=’) . . . . . . . . . . 16
6.5 The Mitsumi Standard Interface (‘mcd=’) . . . . . . . . . . 16
6.6 The Mitsumi XA/MultiSession Interface (‘mcdx=’) . . . . . . . . . . 17
6.7 The Optics Storage Interface (‘optcd=’) . . . . . . . . . . 17
6.8 The Phillips CM206 Interface (‘cm206=’) . . . . . . . . . . 17

6.9 The Sanyo Interface (‘sjcd=’) . . . . . 17
6.10 The SoundBlaster Pro Interface (‘sbpcd=’) . . . . . 17

**7 Other Hardware Devices 17**

7.1 Ethernet Devices (‘ether=’) . . . . . 18
7.2 The Floppy Disk Driver (‘floppy=’) . . . . . 18
7.2.1 floppy=mask,allowed_drive_mask . . . . . 18
7.2.2 floppy=all_drives . . . . . 18
7.2.3 floppy=asus_pci . . . . . 18
7.2.4 floppy=daring . . . . . 19
7.2.5 floppy=0,daring . . . . . 19
7.2.6 floppy=one_fdc . . . . . 19
7.2.7 floppy=two_fdc *or* floppy=address,two_fdc . . . . . 19
7.2.8 floppy=thinkpad . . . . . 19
7.2.9 floppy=0,thinkpad . . . . . 19
7.2.10 floppy=drive,type,cmos . . . . . 19
7.2.11 floppy=unexpected_interrupts . . . . . 19
7.2.12 floppy=no_unexpected_interrupts *or* floppy=L40SX . . . . . 19
7.3 The Sound Driver (‘sound=’) . . . . . 20
7.4 The Bus Mouse Driver (‘bmouse=’) . . . . . 20

**8 Closing 20**

# 1 Introduction

The kernel has a limited capability to accept information at boot in the form of a ‘command line’, similar to an argument list you would give to a program. In general this is used to supply the kernel with information about hardware parameters that the kernel would not be able to determine on its own, or to avoid/override the values that the kernel would otherwise detect.

However, if you just copy a kernel image directly to a floppy, (e.g. `cp zImage /dev/fd0`) then you are not given a chance to specify any arguments to that kernel. So most Linux users will use software like *LILO* or *loadlin* that takes care of handing these arguments to the kernel, and then booting it.

This present revision covers distribution kernels up to and including v1.2.13. Information pertaining to development kernels up to version 1.3.18 is also documented.

The BootPrompt-Howto is edited and maintained by:

Paul Gortmaker, `Paul.Gortmaker@anu.edu.au`

## 1.1 Disclaimer and Copyright

This document is *not* gospel. However, it is probably the most up to date info that you will be able to find. Nobody is responsible for what happens to your hardware but yourself. If your hardware goes up in smoke (...nearly impossible!) I take no responsibility. ie. THE AUTHOR IS NOT RESPONSIBLE FOR ANY DAMAGES INCURRED DUE TO ACTIONS TAKEN BASED ON THE INFORMATION INCLUDED IN THIS DOCUMENT.



If you are intending to incorporate this document into a published work, please contact me, and I will make an effort to ensure that you have the most up to date information available. In the past, out of date versions of the Linux howto documents have been published, which caused the developers undue grief from being plagued with questions that were already answered in the up to date versions.

## 1.2 Related Documentation

The most up-to-date documentation will always be the kernel source itself. Hold on! Don't get scared. You don't need to know any programming to read the comments in the source files. For example, if you were looking for what arguments could be passed to the AHA1542 SCSI driver, then you would go to the `linux/drivers/scsi` directory, and look at the file `aha1542.c` – and within the first 100 lines, you would find a plain english description of the boot time arguments that the 1542 driver accepts.

If you have figured out what boot-args you intend to use, and now want to know how to get that information to the kernel, then look at the documentation that comes with the software that you use to boot the kernel (e.g. LILO or loadlin).

## 1.3 The Linux Newsgroups

If you have questions about passing boot arguments to the kernel, please READ this document first. If this and the related documentation mentioned above does not answer your question(s) then you can try the Linux newsgroups.

General questions on how to configure your system should be directed to comp.os.linux.setup. We ask that you *please* respect this general guideline for content, and don't cross-post your request to other groups.

### 1.4 New Versions of this Document

New versions of this document can be retrieved via anonymous FTP from sunsite.unc.edu, in /pub/Linux/docs/HOWTO/* and various Linux ftp mirror sites. Updates will be made as new information / drivers becomes available. If this copy that you are reading is more than 3 months old, it is either out of date, or it means that I have been lazy and haven't updated it. This document was produced by using the SGML system that was specifically set up for the Linux Howto project, and there are various output formats available, including, postscript, dvi, ascii, html, and soon TeXinfo.

I would recommend viewing it in the html (via a WWW browser) or the Postscript/dvi format. Both of these contain cross-references that are lost in the ascii translation.

If you want to get the official copy off sunsite, here is URL.

*BootPrompt-HOWTO* `<http://sunsite.unc.edu/mdw/HOWTO/BootPrompt-HOWTO.html>`

If minor additions and changes have been made, you can view the latest working copy from this URL.

*Working Copy* `<http://rsphy1.anu.edu.au/~gpg109/BootPrompt-HOWTO.html>`

## 2 Overview of Boot Prompt Arguments

This section gives some examples of software that can be used to pass kernel boot-time arguments to the kernel itself. It also gives you an idea of how the arguments are processed, what limitations there are on the boot args, and how they filter down to each appropriate device that they are intended for.

### 2.1 LILO (LInux LOader)

The LILO program (LInux LOader) written by Werner Almesberger is the most commonly used. It has the ability to boot various kernels, and stores the configuration information in a plain text file. Most distributions ship with LILO as the default boot-loader. LILO can boot DOS, OS/2 Linux, FreeBSD, etc. without any difficulties, and is quite flexible.

LILO ships with excellent documentation, and for the purposes of boot args discussed here, the LILO `append=` command is of significant importance.

### 2.2 LoadLin

The other commonly used Linux loader is 'LoadLin' which is a DOS program that has the capability to launch a Linux kernel from the DOS prompt (with boot-args) assuming that certain resources are available. This is good for people that use DOS and want to launch into Linux from DOS.

It is also very useful if you have certain hardware which relies on the supplied DOS driver to put the hardware into a known state. A common example is 'SoundBlaster Compatible' sound cards that require the DOS driver to twiddle a few mystical registers to put the card into a SB compatible mode. Booting DOS with the supplied driver, and then loading Linux from the DOS prompt with loadlin avoids the reset of the card that happens if one rebooted instead. Thus the card is left in a Sb compatible mode and hence is useable under Linux.

There are also other programs that can be used to boot Linux. For a complete list, please look at the programs available on your local Linux ftp mirror, under `system/Linux-boot/`.

## 2.3 The "rdev" utility

There are a few of the kernel boot parameters that have their default values stored in various bytes in the kernel image itself. There is a utility called `rdev` that is installed on most systems that knows where these values are, and how to change them. It can also change things that have no kernel boot argument equivalent, such as the default video mode used.

The rdev utility is usually also aliased to swapdev, ramsize, vidmode and rootflags. These are the five things that rdev can change, those being the root device, the swap device, the RAM disk size, the default video mode, and the readonly/readwrite setting of root device.

More information on `rdev` can be found by typing `rdev -h` or by reading the supplied man page.

## 2.4 How the Kernel Sorts the Arguments

Most of the boot args take the form of:

```
        name[=value_1][,value_2]...[,value_11]
```

where 'name' is a unique keyword that is used to identify what part of the kernel the associated values (if any) are to be given to. Multiple boot args are just a space separated list of the above format. Note the limit of 11 is real, as the present code only handles 11 comma separated parameters per keyword. (However, you can re-use the same keyword with up to an additional 11 parameters in unusually complicated situations, assuming the setup function supports it.)

Most of the sorting goes on in `linux/init/main.c`. First, the kernel ckecks to see if the argument is any of the special arguments 'root=', 'ro', 'rw', or 'debug'. The meaning of these special arguments is described further on in the document.

Then it walks a list of setup functions (contained in the `bootsetups` array) to see if the specified argument string (such as 'foo') has been associated with a setup function ('foo_setup()') for a particular device or part of the kernel. If you passed the kernel the line `foo=3,4,5,6` then the kernel would search the `bootsetups` array to see if 'foo' was registered. If it was, then it would call the setup function associated with 'foo' (foo_setup()) and hand it the arguments 3, 4, 5 and 6 as given on the kernel command line.

## 2.5 Setting Environment Variables.

Anything of the form 'foo=bar' that is not accepted as a setup funtion as described above is then interpreted as an environment variable to be set. A (useless?) example would be to use 'TERM=vt100' as a boot argument.

### 2.6 Passing Arguments to 'init'

Any remaining arguments that were not picked up by the kernel and were not interpreted as environment variables are then passed onto process one, which is usually the `init` program. The most common argument that is passed to the `init` process is the word *single* which instructs `init` to boot the computer in single user mode, and not launch all the usual daemons. Check the manual page for the version of `init` installed on your system to see what arguments it accepts.

## 3 General Non-Device Specific Boot Args

These are the boot arguments that are not related to any specific device or peripheral. They are instead related to certain internal kernel parameters.

### 3.1 The 'no387' Argument

Some i387 coprocessor chips have bugs that show up when used in 32 bit protected mode. For example, some of the early ULSI-387 chips would cause solid lockups while performing floating point calculations. Using the 'no387' boot arg causes Linux to ignore the maths coprocessor even if you have one. Of course you must then have your kernel compiled with math emulation support!

### 3.2 The 'no-hlt' Argument

The i386 (and sucessors thereof) family of CPUs have a 'hlt' instruction which tells the CPU that nothing is going to happen until an external device (keyboard, modem, disk, etc.) calls upon the CPU to do a task. This allows the CPU to enter a 'low-power' mode where it sits like a zombie until an external device wakes it up (usually via an interrupt). Some of the early i486DX-100 chips had a problem with the 'hlt' instruction, in that they couldn't reliably return to operating mode after this instruction was used. Using the 'no-hlt' instruction tells Linux to just run an infinite loop when there is nothing else to do, and to *not* halt your CPU when there is no activity. This allows people with these broken chips to use Linux, although they would be well advised to seek a replacement through a warranty where possible.

### 3.3 The 'root=' Argument

This argument tells the kernel what device is to be used as the root filesystem while booting. The default of this setting is the value of the root device of the system that the kernel was built on. For example, if the kernel in question was built on a system that used '/dev/hda1' as the root partition, then the default root device would be '/dev/hda1'. To override this default value, and select the second floppy drive as the root device, one would use 'root=/dev/fd1'.

Valid root devices are partitions on any of the following disk devices:

(1) /dev/hdaN to /dev/hddN, which is partition N on ST-506 compatible disk 'a to d'.

(2) /dev/sdaN to /dev/sdeN, which is partition N on SCSI compatible disk 'a to e'.

(3) /dev/xdaN to /dev/xdbN, which is partition N on XT compatible disk 'a to b'.

(4) /dev/fdN, which is floppy disk drive number N. Having N=0 would be the DOS 'A:' drive, and N=1 would be 'B:'.

The more awkward and less portable numeric specification of the above possible root devices in major/minor format is also accepted. (e.g. /dev/sda3 is major 8, minor 3, so you could use `root=0x803` as an alternative.)

This is one of the few kernel boot arguments that has its default stored in the kernel image, and which can thus be altered with the `rdev` utility.

## 3.4 The 'ro' Argument

When the kernel boots, it needs a root filesystem to read basic things off of. This is the root filesystem that is mounted at boot. However, if the root filesystem is mounted with write access, you can not reliably check the filesystem integrity with half-written files in progress. The 'ro' option tells the kernel to mount the root filesystem as 'readonly' so that any filesystem consistency check programs (fsck) can safely assume that there are no half-written files in progress while performing the check. No programs or processes can write to files on the filesystem in question until it is 'remounted' as read/write capable.

This is one of the few kernel boot arguments that has its default stored in the kernel image, and which can thus be altered with the `rdev` utility.

## 3.5 The 'rw' Argument

This is the exact opposite of the above, in that it tells the kernel to mount the root filesytem as read/write. The default is to mount the root filesystem as read/write anyways. Do not run any 'fsck' type programs on a filesystem that is mounted read/write.

The same value stored in the image file mentioned above is also used for this parameter, accesible via `rdev`.

## 3.6 The 'debug' Argument

The kernel communicates important (and not-so important) messages to the operator via the `printk()` function. If the message is considered important, the `printk()` function will put a copy on the present console as well as handing it off to the `klogd()` facility so that it gets logged to disk. The reason for printing important messages to the console as well as logging them to disk is because under unfortunate circumstances (e.g. a disk failure) the message won't make it to disk and will be lost.

The threshold for what is and what isn't considered important is set by the `console_loglevel` variable. The default is to log anything more important than `DEBUG` (level 7) to the console. (These levels are defined in the include file `kernel.h`) Specifying `debug` as a boot argument will set the console loglevel to 10, so that *all* kernel messages appear on the console.

The console loglevel can usually also be set at run time via an option to the `klogd()` program. Check the man page for the version installed on your system to see how to do this.

## 3.7 The 'reserve=' Argument

This is used to *protect* I/O port regions from probes. The form of the command is:

```
reserve=iobase,extent[,iobase,extent]...
```

In some machines it may be necessary to prevent device drivers from checking for devices (auto-probing) in a specific region. This may be because of poorly designed hardware that causes the boot to *freeze* (such as some ethercards), hardware that is mistakenly identified, hardware whose state is changed by an earlier probe, or merely hardware you don't want the kernel to initialize.

The `reserve` boot-time argument addresses this problem by specifying an I/O port region that shouldn't be probed. That region is reserved in the kernel's port registration table as if a device has already been found in that region. Note that this mechanism shouldn't be necessary on most machines. Only when there is a problem or special case would it be necessary to use this.

The I/O ports in the specified region are protected against device probes. This was put in to be used when some driver was hanging on a NE2000, or misidentifying some other device as its own. A correct device driver shouldn't probe a reserved region, unless another boot argument explicitly specifies that it do so. This implies that `reserve` will most often be used with some other boot argument. Hence if you specify a `reserve` region to protect a specific device, you must generally specify an explicit probe for that device. Most drivers ignore the port registration table if they are given an explicit address.

For example, the boot line

```
reserve=0x300,32  blah=0x300
```

keeps all device drivers except the driver for 'blah' from probing 0x300-0x31f.

As usual with boot-time specifiers there is an 11 parameter limit, thus you can only specify 5 reserved regions per `reserve` keyword. Multiple `reserve` specifiers will work if you have an unusually complicated request.

## 3.8 The 'ramdisk=' Argument

This specifies the size in kB of the optional RAM disk device. For example, if one wished to have a root filesystem on a 1.44MB floppy loaded into the RAM disk device, they would use:

```
ramdisk=1440
```

This is one of the few kernel boot arguments that has its default stored in the kernel image, and which can thus be altered with the `rdev` utility.

## 3.9 The 'mem=' Argument

The original BIOS call defined in the PC specification that returns the amount of installed memory was only designed to be able to report up to 64MB. (Yes, another lack of foresight, just like the 1024 cylinder disks...

sigh.) Linux uses this BIOS call at boot to determine how much memory is installed. If you have more than 64MB of RAM installed, you can use this boot arg to tell Linux how much memory you have. Here is a quote from Linus on usage of the 'mem=' parameter.

"The kernel will accept any 'mem=xx' parameter you give it, and if it turns out that you lied to it, it will crash horribly sooner or later. The parameter indicates the highest addressable RAM address, so 'mem=0x1000000' means you have 16MB of memory, for example. For a 96MB machine this would be 'mem=0x6000000'.

NOTE NOTE NOTE: some machines might use the top of memory for BIOS cacheing or whatever, so you might not actually have up to the full 96MB addressable. The reverse is also true: some chipsets will map the physical memory that is covered by the BIOS area into the area just past the top of memory, so the top-of-mem might actually be 96MB + 384kB for example. If you tell linux that it has more memory than it actually does have, bad things will happen: maybe not at once, but surely eventually."

Note that the argument does not have to be in hex, and the suffixes 'k' and 'M' (case insensitive) can be used to specify kilobytes and Megabytes, respectively. (A 'k' will cause a 10 bit shift on your value, and a 'M' will cause a 20 bit shift.) The above warning still holds, in that a 96MB machine may work with `mem=97920k` but fail with either `mem=98304k` or `mem=96M`.

# 4 Boot Arguments for SCSI Peripherals.

This section contains the descriptions of the boot args that are used for passing information about the installed SCSI host adapters, and SCSI devices.

General notation for this section:

`iobase` – the first I/O port that the SCSI host occupies. These are specified in hexidecimal notation, and usually lie in the range from `0x200` to `0x3ff`.

`irq` – the hardware interrupt that the card is configured to use. Valid values will be dependant on the card in question, but will usually be 5, 7, 9, 10, 11, 12, and 15. The other values are usually used for common preipherals like IDE hard disks, floppies, serial ports, etc.

`scsi-id` – the ID that the host adapter uses to identify itself on the SCSI bus. Only some host adapters allow you to change this value, as most have it permanently specified internally. The usual default value is seven, but the Seagate and Future Domain TMC-950 boards use six.

`parity` – whether the SCSI host adapter expects the attached devices to supply a parity value with all information exchanges. Specifying a one indicates parity checking is enabled, and a zero disables parity checking. Again, not all adapters will support selection of parity behaviour as a boot argument.

## 4.1 Maximum Probed LUNs ('max_scsi_luns=')

Each SCSI device can have a number of 'sub-devices' contained within itself. The most common example is one of the new SCSI CD-ROMs that handle more than one disk at a time. Each CD is addressed as a 'Logical Unit Number' (LUN) of that particular device. But most devices, such as hard disks, tape drives and such are only one device, and will be assigned to LUN zero.

The problem arises with single LUN devices with bad firmware. Some poorly designed SCSI devices (old and unfortunately new) can not handle being probed for LUNs not equal to zero. They will respond by locking up, and possibly taking the whole SCSI bus down with them.

Newer kernels have the configuration option that allows you to set the maximum number of probed LUNs. The default is to only probe LUN zero, to avoid the problem described above.

To specify the number of probed LUNs at boot, one enters 'max_scsi_luns=n' as a boot arg, where n is a number between one and eight. To avoid problems as described above, one would use n=1 to avoid upsetting such broken devices

## 4.2 Parameters for SCSI Tape drives ('st=')

Some boot time configuration of the SCSI tape driver can be achieved by using the following:

```
st=buf_size[,write_threshold[,max_bufs]]
```

The fisrt two numbers are specified in units of kB. The default `buf_size` is 32kB, and the maximum size that can be specified is a ridiculous 16384kB. The `write_threshold` is the value at which the buffer is committed to tape, with a default value of 30kB. The maximum number of buffers varies with the number of drives detected, and has a default of two. An example usage would be:

```
st=32,30,2
```

Full details can be found in the `README.st` file that is in the `scsi` directory of the kernel source tree.

## 4.3 Adaptec aha151x, aha152x, aic6260, aic6360, SB16-SCSI ('aha152x=')

The aha numbers refer to cards and the aic numbers refer to the actual SCSI chip on these type of cards, including the Soundblaster-16 SCSI.

The probe code for these SCSI hosts looks for an installed BIOS, and if none is present, the probe will not find your card. Then you will have to use a boot arg of the form:

```
aha152x=iobase[,irq[,scsi-id[,reconnect[,parity]]]]
```

Note that if the driver was compiled with debugging enabled, a sixth value can be specified to set the debug level.

All the parameters are as described at the top of this section, and the `reconnect` value will allow device disconnect/reconnect if a non-zero value is used. An example usage is as follows:

```
aha152x=0x340,11,7,1
```

Note that the parameters must be specified in order, meaning that if you want to specify a parity setting, then you will have to specify an iobase, irq, scsi-id and reconnect value as well.

## 4.4 Adaptec aha154x ('aha1542=')

These are the aha154x series cards. The aha1542 series cards have an i82077 floppy controller onboard, while the aha1540 series cards do not. These are busmastering cards, and have parameters to set the "fairness" that is used to share the bus with other devices. The boot arg looks like the following.

```
aha1542=iobase[,buson,busoff[,dmaspeed]]
```

Valid `iobase` values are usually one of: `0x130, 0x134, 0x230, 0x234, 0x330, 0x334`. Clone cards may permit other values.

The `buson, busoff` values refer to the number of microseconds that the card dominates the ISA bus. The defaults are 11us on, and 4us off, so that other cards (such as an ISA LANCE Ethernet card) have a chance to get access to the ISA bus.

The `dmaspeed` value refers to the rate (in MB/s) at which the DMA (Direct Memory Access) transfers proceed at. The default is 5MB/s. Newer revision cards allow you to select this value as part of the soft-configuration, older cards use jumpers. You can use values up to 10MB/s assuming that your motherboard is capable of handling it. Experiment with caution if using values over 5MB/s.

## 4.5 Adaptec aha274x, aha284x, aic7xxx ('aic7xxx=')

These boards can accept an argument of the form:

```
aic7xxx=extended,no_reset
```

The `extended` value, if non-zero, indicates that extended translation for large disks is enabled. The `no_reset` value, if non-zero, tells the driver not to reset the SCSI bus when setting up the host adaptor at boot.

## 4.6 BusLogic SCSI Hosts ('buslogic=')

At present, the buslogic driver accepts only one parameter, that being the I/O base. It expects that to be one of the following valid values: `0x130, 0x134, 0x230, 0x234, 0x330, 0x334`.

## 4.7 Future Domain TMC-8xx, TMC-950 ('tmc8xx=')

The probe code for these SCSI hosts looks for an installed BIOS, and if none is present, the probe will not find your card. Or, if the signature string of your BIOS is not recognised then it will also not be found. In either case, you will then have to use a boot arg of the form:

```
tmc8xx=mem_base,irq
```

The `mem_base` value is the value of the memory mapped I/O region that the card uses. This will usually be one of the following values: `0xc8000, 0xca000, 0xcc000, 0xce000, 0xdc000, 0xde000`.

### 4.8 Pro Audio Spectrum ('pas16=')

The PAS16 uses a NC5380 SCSI chip, and newer models support jumper-less configuration. The boot arg is of the form:

```
pas16=iobase,irq
```

The only difference is that you can specify an IRQ value of 255, which will tell the driver to work without using interrupts, albeit at a performance loss. The `iobase` is usually `0x388`.

### 4.9 Seagate ST-0x ('st0x=')

The probe code for these SCSI hosts looks for an installed BIOS, and if none is present, the probe will not find your card. Or, if the signature string of your BIOS is not recognised then it will also not be found. In either case, you will then have to use a boot arg of the form:

```
st0x=mem_base,irq
```

The `mem_base` value is the value of the memory mapped I/O region that the card uses. This will usually be one of the following values: `0xc8000, 0xca000, 0xcc000, 0xce000, 0xdc000, 0xde000`.

### 4.10 Trantor T128 ('t128=')

These cards are also based on the NCR5380 chip, and accept the following options:

```
t128=mem_base,irq
```

The valid values for `mem_base` are as follows: `0xcc000, 0xc8000, 0xdc000, 0xd8000`.

### 4.11 Cards that don't Accept Boot Args

At present, the following SCSI cards do not make use of any boot-time parameters. In some cases, you can *hard-wire* values by directly editing the driver itself, if required.

Always IN2000, Adaptec aha1740, EATA-DMA, EATA-PIO, Future Domain 16xx, NCR5380 (generic), NCR53c7xx to NCR53c8xx, Qlogic, Ultrastor (incl. u?4f), Western Digital wd7000,

## 5 Hard Disks

This section lists all the boot args associated with standard MFM/RLL, ST-506, XT, and IDE disk drive devices. Note that both the IDE and the generic ST-506 HD driver both accept the 'hd=' option.

## 5.1 IDE Disk/CD-ROM Driver Parameters

The IDE driver accepts a number of parameters, which range from disk geometry specifications, to support for broken controller chips. Drive specific options are specified by using one of:

'hda=', 'hdb=', 'hdc=', or 'hdd='.

Non-drive specific options are specified with the prefix 'hd='. Note that using a drive specific prefix for a non-drive specific option will still work, and the option will just be applied as expected.

Also note that 'hd=' can be used to refer to the next unspecified drive in the (a, b, c, d) sequence. For the following discussions, the 'hd=' option will be cited for brevity. Please consult the file `README.ide` in the `linux/drivers/block` directory if more information is required.

### 5.1.1 The 'hd=cyls,heads,sects[,wpcom[,irq]]' options

These options are used to specify the physical geometry of the disk. Only the first three values are required. The cylinder/head/sectors values will be those used by `fdisk`. The write precompensation value is ignored for IDE disks. The IRQ value specified will be the IRQ used for the interface that the drive resides on, and is not really a drive specific parameter.

### 5.1.2 The 'hd=serialize' option

The dual IDE interface CMD-640 chip is broken as designed such that when drives on the secondary interface are used at the same time as drives on the primary interface, it will corrupt your data. Using this option tells the driver to make sure that both interfaces are never used at the same time. If you only have up to two drives, both on the primary interface, then you don't need to use this option.

### 5.1.3 The 'hd=dtc2278' option

This option tells the driver that you have a DTC-2278D IDE interface. The driver then tries to do DTC specific operations to enable the second interface and to enable faster transfer modes.

### 5.1.4 The 'hd=noprobe' option

If a particular drive (e.g. old IDE drive) has problems that are a result of being probed, this option can be used to disable the probe. An example usage could be:

```
hdb=noprobe hdb=1166,7,17
```

which would disable the probe, but still specify the drive geometry so that it would be registered as a valid block device, and hence useable.

#### 5.1.5 The ‘hd=nowerr’ option

Some drives apparently have the WRERR_STAT bit stuck on permanently. This enables a work-around for these broken devices.

#### 5.1.6 The ‘hd=cdrom’ option

This tells the IDE driver that there is an ATAPI compatible CD-ROM attached in place of a normal IDE hard disk. In most cases the CD-ROM is identified automatically, but if it isn’t then this may help.

### 5.2 Standard ST-506 Disk Driver Options (‘hd=’)

The standard disk driver can accept geometry arguments for the disks similar to the IDE driver. Note however that it only expects three values (C/H/S) – any more or any less and it will silently ignore you. Also, it only accepts ‘hd=’ as an argument, i.e. ‘hda=’, ‘hdb=’ and so on are not valid here. The format is as follows:

```
hd=cyls,heads,sects
```

If there are two disks installed, the above is repeated with the geometry parameters of the second disk.

### 5.3 XT Disk Driver Options (‘xd=’)

If you are unfortunate enough to be using one of these old 8 bit cards that move data at a whopping 125kB/s then here is the scoop. The probe code for these cards looks for an installed BIOS, and if none is present, the probe will not find your card. Or, if the signature string of your BIOS is not recognised then it will also not be found. In either case, you will then have to use a boot arg of the form:

```
xd=type,irq,iobase,dma_chan
```

The `type` value specifies the particular manufacturer of the card, and are as follows: 0=generic; 1=DTC; 2,3,4=Western Digital, 5,6,7=Seagate; 8=OMTI. The only difference between multiple types from the same manufacturer is the BIOS string used for detection, which is not used if the type is specified.

The `xd_setup()` function does no checking on the values, and assumes that you entered all four values. Don’t disappoint it. Here is an example usage for a WD1002 controller with the BIOS disabled/removed, using the ‘default’ XT controller parameters:

```
xd=2,5,0x320,3
```

## 6 CD-ROMs (Non-SCSI/ATAPI/IDE)

This section lists all the possible boot args pertaining to CD-ROM devices. Note that this does not include SCSI or IDE/ATAPI CD-ROMs. See the appropriate section(s) for those types of CD-ROMs.

## 6.1 The Aztech Interface ('aztcd=')

The syntax for this type of card is:

```
aztcd=iobase[,magic_number]
```

If you set the magic_number to 0x79 then the driver will try and run anyway in the event of an unknown firmware version. All other values are ignored.

## 6.2 The CDU-31A and CDU-33A Sony Interface ('cdu31a=')

This CD-ROM interface is found on some of the Pro Audio Spectrum sound cards, and other Sony supplied interface cards. The syntax is as follows:

```
cdu31a=iobase,[irq[,is_pas_card]]
```

Specifying an IRQ value of zero tells the driver that hardware interrupts aren't supported (as on some PAS cards). If your card supports interrupts, you should use them as it cuts down on the CPU usage of the driver.

The 'is_pas_card' should be entered as 'PAS' if using a Pro Audio Spectrum card, and otherwise it should not be specified at all.

## 6.3 The CDU-535 Sony Interface ('sonycd535=')

The syntax for this CD-ROM interface is:

```
sonycd535=iobase[,irq]
```

A zero can be used for the I/O base as a 'placeholder' if one wishes to specify an IRQ value.

## 6.4 The GoldStar Interface ('gscd=')

The syntax for this CD-ROM interface is:

```
gscd=iobase
```

## 6.5 The Mitsumi Standard Interface ('mcd=')

The syntax for this CD-ROM interface is:

```
mcd=iobase,[irq[,wait_value]]
```

The wait_value is used as an internal timeout value for people who are having problems with their drive, and may or may not be implemented depending on a compile time DEFINE.

### 6.6 The Mitsumi XA/MultiSession Interface ('mcdx=')

At present this 'experimental' driver has a setup function, but no parameters are implemented yet (as of 1.3.15). This is for the same hardware as above, but the driver has extended features.

### 6.7 The Optics Storage Interface ('optcd=')

The syntax for this type of card is:

```
optcd=iobase
```

### 6.8 The Phillips CM206 Interface ('cm206=')

The syntax for this type of card is:

```
cm206=[iobase][,irq]
```

The driver assumes numbers between 3 and 11 are IRQ values, and numbers between `0x300` and `0x370` are I/O ports, so you can specify one, or both numbers, in any order. It also accepts 'cm206=auto' to enable autoprobing.

### 6.9 The Sanyo Interface ('sjcd=')

The syntax for this type of card is:

```
sjcd=iobase[,irq[,dma_channel]]
```

### 6.10 The SoundBlaster Pro Interface ('sbpcd=')

The syntax for this type of card is:

```
sbpcd=iobase,type
```

where `type` is one of the following (case sensitive) strings: 'SoundBlaster', 'LaserMate', or 'SPEA'. The I/O base is that of the CD-ROM interface, and *not* that of the sound portion of the card.

## 7 Other Hardware Devices

Any other devices that didn't fit into any of the above categories got lumped together here.

## 7.1 Ethernet Devices ('ether=')

Different drivers make use of different parameters, but they all at least share having an IRQ, an I/O port base value, and a name. In its most generic form, it looks something like this:

```
ether=irq,iobase[,param_1[,param_2,...param_8]]],name
```

The first non-numeric argument is taken as the name. The `param_n` values (if applicable) usually have different meanings for each different card/driver. Typical `param_n` values are used to specify things like shared memory address, interface selection, DMA channel and the like.

The most common use of this parameter is to force probing for a second ethercard, as the default is to only probe for one. This can be accomplished with a simple:

```
ether=0,0,eth1
```

Note that the values of zero for the IRQ and I/O base in the above example tell the driver(s) to autoprobe.

Note that the Ethernet-HowTo has complete and extensive documentation on using multiple cards and on the card/driver specific implementation of the `param_n` values where used. Interested readers should refer to the section in that document on their particular card for more complete information.

## 7.2 The Floppy Disk Driver ('floppy=')

There are many floppy driver options, and they are all listed in `README.fd` in `linux/drivers/block`. This information is taken directly from that file.

### 7.2.1 floppy=mask,allowed_drive_mask

Sets the bitmask of allowed drives to `mask`. By default, only units 0 and 1 of each floppy controller are allowed. This is done because certain non-standard hardware (ASUS PCI motherboards) mess up the keyboard when accessing units 2 or 3. This option is somewhat obsoleted by the cmos option.

### 7.2.2 floppy=all_drives

Sets the bitmask of allowed drives to all drives. Use this if you have more than two drives connected to a floppy controller.

### 7.2.3 floppy=asus_pci

Sets the bitmask to allow only units 0 and 1. (The default)

### 7.2.4 floppy=daring

Tells the floppy driver that you have a well behaved floppy controller. This allows more efficient and smoother operation, but may fail on certain controllers. This may speed up certain operations.

### 7.2.5 floppy=0,daring

Tells the floppy driver that your floppy controller should be used with caution.

### 7.2.6 floppy=one_fdc

Tells the floppy driver that you have only floppy controller (default)

### 7.2.7 floppy=two_fdc *or* floppy=address,two_fdc

Tells the floppy driver that you have two floppy controllers. The second floppy controller is assumed to be at address. If address is not given, 0x370 is assumed.

### 7.2.8 floppy=thinkpad

Tells the floppy driver that you have a Thinkpad. Thinkpads use an inverted convention for the disk change line.

### 7.2.9 floppy=0,thinkpad

Tells the floppy driver that you don't have a Thinkpad.

### 7.2.10 floppy=drive,type,cmos

Sets the cmos type of `drive` to `type`. Additionally, this drive is allowed in the bitmask. This is useful if you have more than two floppy drives (only two can be described in the physical cmos), or if your BIOS uses non-standard CMOS types. Setting the CMOS to 0 for the first two drives (default) makes the floppy driver read the physical cmos for those drives.

### 7.2.11 floppy=unexpected_interrupts

Print a warning message when an unexpected interrupt is received (default behaviour)

### 7.2.12 floppy=no_unexpected_interrupts *or* floppy=L40SX

Don't print a message when an unexpected interrupt is received. This is needed on IBM L40SX laptops in certain video modes. (There seems to be an interaction between video and floppy. The unexpected interrupts only affect performance, and can safely be ignored.)

### 7.3 The Sound Driver ('sound=')

The sound driver can also accept boot args to override the compiled in values. This is not recommended, as it is rather complex. It is described in the `Readme.Linux` file, in `linux/drivers/sound`. It accepts a boot arg of the form:

```
        sound=device1[,device2[,device3...[,device11]]]
```

where each `deviceN` value is of the following format `0xTaaaId` and the bytes are used as follows:

T - device type: 1=FM, 2=SB, 3=PAS, 4=GUS, 5=MPU401, 6=SB16, 7=SB16-MPU401

aaa - I/O address in hex.

I - interrupt line in hex (i.e 10=a, 11=b, ...)

d - DMA channel.

As you can see it gets pretty messy, and you are better off to compile in your own personal values as recommended. Using a boot arg of 'sound=0' will disable the sound driver entirely.

### 7.4 The Bus Mouse Driver ('bmouse=')

The busmouse driver only accepts one parameter, that being the hardware IRQ value to be used.

## 8 Closing

If you have found any glaring typos, or outdated info in this document, please let me know. It is easy to overlook stuff.

Thanks,

Paul Gortmaker, `Paul.Gortmaker@anu.edu.au`