The Linux BootPrompt−HowTo

Table of Contents

Table of Contents

Table of Contents

The Linux BootPrompt−HowTo

by Paul Gortmaker.

v1.2, May 1999

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. A discussion of how the kernel sorts boot time arguments, along with an overview of some of the popular software used to boot Linux kernels is also included.

1.Introduction

2.Overview of Boot Prompt Arguments

3.General Non−Device Specific Boot Args

4.Boot Arguments to Control PCI Bus Behaviour (`pci=')

5.Boot Arguments for Video Frame Buffer Drivers

6.Boot Arguments for SCSI Peripherals.

7.Hard Disks

8.CD−ROMs (Non−SCSI/ATAPI/IDE)

9.Serial and ISDN Drivers

10.Other Hardware Devices

11.Copying, Translations, Closing, etc.

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 kernels up to and including v2.2.9. Some features that are unique to development/testing kernels up to v2.3.2 are also documented.

The BootPrompt−Howto is by:

Paul Gortmaker, `p_gortmaker@yahoo.com`

1.1 Disclaimer and Copyright

1.2 Intended Audience and Applicability

Most Linux users should never have to even look at this document. Linux does an exceptionally good job at detecting most hardware and picking reasonable default settings for most parameters. The information in this document is aimed at users who might want to change some of the default settings to optimize the kernel to their particular machine, or to a user who has `rolled their own' kernel to support a not so common piece of hardware for which automatic detection is currently not available.

IMPORTANT NOTE: Driver related boot prompt arguments only apply to hardware drivers that are compiled directly into the kernel. They have *no effect* on drivers that are loaded as modules. Most Linux distributions come with a basic `bare−bones' kernel, and the drivers are small modules that are loaded after the kernel has initialized. If you are unsure if you are using modules then look at `man depmod` and `man modprobe` along with the contents of your `/etc/conf.modules`.

1.3 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.

The `linux` directory is usually found in `/usr/src/` for most distributions. All references in this document to files that come with the kernel will have their pathname abbreviated to start with `linux` − you will have to append the `/usr/src/` or whatever is appropriate for your system. (If you can't find the file in question, then make use of the `find` and `locate` commands.)

The next best thing will be any documentation files that are distributed with the kernel itself. There are now quite a few of these, and most of them can be found in the directory `linux/Documentation` and subdirectories from there. Sometimes there will be `README.foo` files that can be found in the related driver directory (e.g. `linux/drivers/???/`, where examples of `???` could be `scsi`, `char`, or `net`).

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). A brief overview is given below, but it is no substitute for the documentation that comes with the booting software.

1.4 The Linux Newsgroups

If you have questions about passing boot arguments to the kernel, please check 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 the comp.os.linux.setup newsgroup. We ask that you *please* respect this general guideline for content, and don't cross−post your request to other groups.

Of course you should try checking the group before blindly posting your question, as it may even be a Frequently Asked Question (a FAQ). A quick browse of the Linux FAQ before posting is a *good* idea. You should be able to find the FAQ somewhere close to where you found this document. If it is not a FAQ then use newsgroup archives, such as those at `http://www.dejanews.com` to quickly search years worth of postings for your topic. Chances are someone else has already asked (and another person answered!) the question that you now have.

1.5 New Versions of this Document

New versions of this document can be retrieved via anonymous FTP from most Linux FTP sites in the directory `/pub/Linux/docs/HOWTO/`. Updates will be made as new information and/or drivers becomes available. If this copy that you are presently reading is more than six months old, then you should probably check to see if a newer copy exists. I would recommend viewing this via a WWW browser or in the Postscript/dvi format. Both of these contain cross−references that are lost in a simple plain text version.

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

BootPrompt−HOWTO

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.

It is *important* to note that spaces should *not* be used in a boot argument, but only between separate arguments. A list of values that are for a single argument are to be separated with a comma between the values, and again without any spaces. See the following examples below.

```
        ether=9,0x300,0xd0000,0xd4000,eth0  root=/dev/hda1            *RIGHT*
        ether = 9, 0x300, 0xd0000, 0xd4000, eth0  root = /dev/hda1    *WRONG*
```

Once the Linux kernel is up and running, one can view the command line arguments that were in place at boot by simply typing `cat /proc/cmdline` at a shell prompt.

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.

A typical configuration will have LILO stop and print `LILO:` shortly after you turn on your computer. It will then wait for a few seconds for any optional input from the user, and failing that it will then boot the default system. Typical system labels that people use in the LILO configuration files are `linux` and `backup` and `msdos`. If you want to type in a boot argument, you type it in here, after typing in the system label that you want LILO to boot from, as shown in the example below.

```
        LILO: linux root=/dev/hda1
```

LILO comes with excellent documentation, and for the purposes of boot args discussed here, the LILO `append=` command is of significant importance when one wants to add a boot time argument as a permanent addition to the LILO config file. You simply add something like `append = "foo=bar"` to the `/etc/lilo.conf` file. It can either be added at the top of the config file, making it apply to all sections, or to a single system section by adding it inside an `image=` section. Please see the LILO documentation for a more complete description.

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 set a few proprietary 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.EXE` 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 parameters, 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 (`man rdev`).

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.) Also note that the kernel splits the list into a maximum of ten integer arguments, and a following string, so you can't really supply 11 integers unless you convert the 11th arg from a string to an int in the driver itself.

Most of the sorting goes on in `linux/init/main.c`. First, the kernel checks 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,bar` 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 integer arguments 3, 4, 5 and 6 as given on the kernel command line, and also hand it the string argument `bar`.

2.5 Setting Environment Variables.

Anything of the form `foo=bar' that is not accepted as a setup function as described above is then interpreted as an environment variable to be set. An example would be to use `TERM=vt100` or `BOOT_IMAGE=vmlinuz.bak` as a boot argument. These environment variables are typically tested for in the initialization scripts to enable or disable a wide range of things.

2.6 Passing Arguments to the `init' program

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, such as memory handling, ramdisk handling, root file system handling and others.

3.1 Root Filesystem options

The following options all pertain to how the kernel selects and handles the root filesystem.

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 any of the following 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:'.

(5) /dev/nfs, which is not really a device, but rather a flag to tell the kernel to get the root fs via the network.

The more awkward and less portable numeric specification of the above possible disk 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.

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.

The `rw' Argument

This is the exact opposite of the above, in that it tells the kernel to mount the root filesystem as read/write. The default is to mount the root filesystem as read/write anyway. 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, accessible via `rdev`.

3.2 Options Relating to RAM Disk Management

The following options all relate to how the kernel handles the RAM disk device, which is usually used for bootstrapping machines during the install phase, or for machines with modular drivers that need to be installed to access the root filesystem.

The `ramdisk_start=' Argument

To allow a kernel image to reside on a floppy disk along with a compressed ramdisk image, the `ramdisk_start=<offset>' command was added. The kernel can't be included into the compressed ramdisk filesystem image, because it needs to be stored starting at block zero so that the BIOS can load the bootsector and then the kernel can bootstrap itself to get going.

Note: If you are using an uncompressed ramdisk image, then the kernel can be a part of the filesystem image that is being loaded into the ramdisk, and the floppy can be booted with LILO, or the two can be separate as is done for the compressed images.

If you are using a two−disk boot/root setup (kernel on disk 1, ramdisk image on disk 2) then the ramdisk would start at block zero, and an offset of zero would be used. Since this is the default value, you would not need to actually use the command at all.

The `load_ramdisk=' Argument

This parameter tells the kernel whether it is to try to load a ramdisk image or not. Specifying `load_ramdisk=1' will tell the kernel to load a floppy into the ramdisk. The default value is zero, meaning that the kernel should not try to load a ramdisk.

Please see the file `linux/Documentation/ramdisk.txt` for a complete description of the new boot time arguments, and how to use them. A description of how this parameter can be set and stored in the kernel image via `rdev' is also described.

The `prompt_ramdisk=' Argument

This parameter tells the kernel whether or not to give you a prompt asking you to insert the floppy containing the ramdisk image. In a single floppy configuration the ramdisk image is on the same floppy as the kernel that just finished loading/booting and so a prompt is not needed. In this case one can use `prompt_ramdisk=0'. In a two floppy configuration, you will need the chance to switch disks, and thus `prompt_ramdisk=1' can be used. Since this is the default value, it doesn't really need to be specified. ((Historical note: Sneaky people used to use the `vga=ask' LILO option to temporarily pause the boot process and allow a chance to switch from boot to root floppy.)

Please see the file `linux/Documentation/ramdisk.txt` for a complete description of the new boot time arguments, and how to use them. A description of how this parameter can be set and stored in the kernel image via `rdev' is also described.

The `ramdisk_size=' Argument

While it is true that the ramdisk grows dynamically as required, there is an upper bound on its size so that it doesn't consume all available RAM and leave you in a mess. The default is 4096 (i.e. 4MB) which should be large enough for most needs. You can override the default to a bigger or smaller size with this boot argument.

Please see the file `linux/Documentation/ramdisk.txt` for a complete description of the new boot time arguments, and how to use them. A description of how this parameter can be set and stored in the kernel image via `rdev' is also described.

The `ramdisk=' Argument (obsolete)

(NOTE: This argument is obsolete, and should not be used except on kernels v1.3.47 and older. The commands that should be used for the ramdisk device are documented above.)

This specifies the size in kB of the 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.

The `noinitrd' (initial RAM disk) Argument

The v2.x and newer kernels have a feature where the root filesystem can be initially a RAM disk, and the kernel executes `/linuxrc` on that RAM image. This feature is typically used to allow loading of modules needed to mount the real root filesystem (e.g. load the SCSI driver modules stored in the RAM disk image, and then mount the real root filesystem on a SCSI disk.)

The actual `noinitrd' argument determines what happens to the initrd data after the kernel has booted. When specified, instead of converting it to a RAM disk, it is accessible via `/dev/initrd`, which can be read once before the RAM is released back to the system. For full details on using the initial RAM disk, please consult `linux/Documentation/initrd.txt`. In addition, the most recent versions of `LILO` and `LOADLIN` should have additional useful information.

3.3 Boot Arguments Related to Memory Handling

The following arguments alter how Linux detects or handles the physical and virtual memory of your system.

The `mem=' Argument

This argument has two purposes: The original purpose was to specify the amount of installed memory (or a value less than that if you wanted to limit the amount of memory available to linux). The second (and hardly used) purpose is to specify `mem=nopentium` which tells the Linux kernel to not use the 4MB page table performance feature.

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 argument to tell Linux how much memory you have. Here is a quote from Linus on the 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'. 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.) A typical example for a 128MB machine would be "`mem=128m`".

The `swap=' Argument

This allows the user to tune some of the virtual memory (VM) parameters that are related to swapping to disk. It accepts the following eight parameters:

```
        MAX_PAGE_AGE
        PAGE_ADVANCE
        PAGE_DECLINE
        PAGE_INITIAL_AGE
        AGE_CLUSTER_FRACT
        AGE_CLUSTER_MIN
        PAGEOUT_WEIGHT
        BUFFEROUT_WEIGHT
```

Interested hackers are advised to have a read of `linux/mm/swap.c` and also make note of the goodies in `/proc/sys/vm`. Kernels come with some useful documentation on this in the `linux/Documentation/vm/` directory.

The `buff=' Argument

Similar to the `swap=' argument, this allows the user to tune some of the parameters related to buffer memory management. It accepts the following six parameters:

```
        MAX_BUFF_AGE
        BUFF_ADVANCE
        BUFF_DECLINE
        BUFF_INITIAL_AGE
        BUFFEROUT_WEIGHT
        BUFFERMEM_GRACE
```

Interested hackers are advised to have a read of `linux/mm/swap.c` and also make note of the goodies in `/proc/sys/vm`. Kernels come with some useful documentation on this in the `linux/Documentation/vm/` directory.

3.4 Boot Arguments for NFS Root Filesystem

Linux supports systems such as diskless workstations via having their root filesystem as NFS (Network FileSystem). These arguments are used to tell the diskless workstation which machine it is to get its system from. Also note that the argument `root=/dev/nfs` is required. Detailed information on using an NFS root fs is in the file `linux/Documentation/nfsroot.txt`. You should read that file, as the following is only a quick summary taken directly from that file.

The `nfsroot=' Argument

This argument tells the kernel which machine, what directory and what NFS options to use for the root filesystem. The form of the argument is as follows:

```
        nfsroot=[<server-ip>:]<root-dir>[,<nfs-options>]
```

If the nfsroot parameter is not given on the command line, the default `/tftpboot/%s' will be used. The other options are as follows:

<server−ip> −− Specifies the IP address of the NFS server. If this field is not given, the default address as determined by the nfsaddrs variable (see below) is used. One use of this parameter is for example to allow using different servers for RARP and NFS. Usually you can leave this blank.

<root−dir> −− Name of the directory on the server to mount as root. If there is a `%s' token in the string, the token will be replaced by the ASCII−representation of the client's IP address.

<nfs−options> −− Standard NFS options. All options are separated by commas. If the options field is not given, the following defaults will be used:

```
        port            = as given by server portmap daemon
        rsize           = 1024
        wsize           = 1024
        timeo           = 7
        retrans         = 3
        acregmin        = 3
        acregmax        = 60
        acdirmin        = 30
        acdirmax        = 60
        flags           = hard, nointr, noposix, cto, ac
```

The `nfsaddrs=' Argument

This boot argument sets up the various network interface addresses that are required to communicate over the network. If this argument is not given, then the kernel tries to use RARP and/or BOOTP to figure out these parameters. The form is as follows:

```
        nfsaddrs=<my-ip>:<serv-ip>:<gw-ip>:<netmask>:<name>:<dev>:<auto>
```

<my−ip> −− IP address of the client. If empty, the address will either be determined by RARP or BOOTP. What protocol is used de− pends on what has been enabled during kernel configuration and on the <auto> parameter. If this parameter is not empty, neither RARP nor BOOTP will be used.

<serv−ip> −− IP address of the NFS server. If RARP is used to determine the client address and this parameter is NOT empty only replies from the specified server are accepted. To use different RARP and NFS server, specify your RARP server here (or leave it blank), and specify your NFS server in the nfsroot parameter (see above). If this entry is blank the address of the server is used which answered the RARP or BOOTP request.

<gw−ip> −− IP address of a gateway if the server in on a different subnet. If this entry is empty no gateway is used and the server is assumed to be on the local network, unless a value has been received by BOOTP.

<netmask> −− Netmask for local network interface. If this is empty, the netmask is derived from the client IP address, unless a value has been received by BOOTP.

<name> −− Name of the client. If empty, the client IP address is used in ASCII−notation, or the value received by BOOTP.

<dev> −− Name of network device to use. If this is empty, all devices are used for RARP requests, and the first one found for BOOTP. For NFS the device is used on which either RARP or BOOTP replies have been received. If you only have one device you can safely leave this blank.

<auto> −− Method to use for autoconfiguration. If this is either `rarp' or `bootp' the specified protocol is being used. If the value is `both' or empty, both protocols are used so far as they have been enabled during kernel configuration Using 'none' means no autoconfiguration. In this case you have to specify all necessary values in the fields before.

The <auto> parameter can appear alone as the value to the nfsaddrs parameter (without all the `:' characters before) in which case autoconfiguration is used. However, the `none' value is not available in that case.

3.5 Other Misc. Kernel Boot Arguments

These various boot arguments let the user tune certain internal kernel parameters.

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.

The `init=' Argument

The kernel defaults to starting the `init' program at boot, which then takes care of setting up the computer for users via launching getty programs, running `rc' scripts and the like. The kernel first looks for `/sbin/init`, then `/etc/init` (depreciated), and as a last resort, it will try to use `/bin/sh` (possibly on `/etc/rc`). If for example, your init program got corrupted and thus stopped you from being able to boot, you could simply use the boot prompt `init=/bin/sh` which would drop you directly into a shell at boot, allowing you to replace the corrupted program.

The `kbd−reset' Argument

Normally on i386 based machines, the Linux kernel does not reset the keyboard controller at boot, since the BIOS is supposed to do this. But as usual, not all machines do what they should. Supplying this option may help if you are having problems with your keyboard behaviour. It simply forces a reset at initialization time. (Some have argued that this should be the default behaviour anyways).

The `maxcpus=' Argument

The number given with this argument limits the maximum number of CPUs activated in SMP mode. Using a value of 0 is equivalent to the `nosmp` option.

The `mca−pentium' Argument

The IBM model 95 Microchannel machines seem to lock up on the test that Linux usually does to detect the type of math chip coupling. Since all Pentium chips have a built in math processor, this test (and the lock up problem) can be avoided by using this boot option.

The `md=' Argument

If your root filesystem is on a Multiple Device then you can use this (assuming you compiled in boot support) to tell the kernel the multiple device layout. The format (from the file `linux/Documentation/md.txt`) is:

```
md=md_device_num,raid_level,chunk_size_factor,fault_level,dev0,dev1,...,devN
```

Where `md_device_num` is the number of the md device, i.e. 0 means md0, 1 means md1, etc. For `raid_level`, use −1 for linear mode and 0 for striped mode. Other modes are currently unsupported. The `chunk_size_factor` is for raid−0 and raid−1 only and sets the chunk size as PAGE_SIZE shifted left the specified amount. The `fault_level` is only for raid−1 and sets the maximum fault number to the specified number. (Currently unsupported due to lack of boot support for raid1.) The `dev0−devN` are a commaseparated list of the devices that make up the individual md device: e.g. `/dev/hda1,/dev/hdc1,/dev/sda1`

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, apparently due to a bug in the FRSAV/FRRESTOR instructions. Using the `no387' boot argument causes

Linux to ignore the math coprocessor even if you have one. Of course you must then have your kernel compiled with math emulation support! This may also be useful if you have one of those *really* old 386 machines that could use an 80287 FPU, as Linux can't use an 80287.

The `no−hlt' Argument

The i386 (and successors 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.

The `no−scroll' Argument

Using this argument at boot disables scrolling features that make it difficult to use Braille terminals.

The `noapic' Argument

Using this option tells a SMP kernel to not use some of the advanced features of the interrupt controller on multi processor machines. See `linux/Documentation/IO−APIC.txt` for more information.

The `nosmp' Argument

Use of this option will tell a SMP kernel on a SMP machine to operate single processor. Typically only used for debugging and determining if a particular problem is SMP related.

The `panic=' Argument

In the unlikely event of a kernel panic (i.e. an internal error that has been detected by the kernel, and which the kernel decides is serious enough to moan loudly and then halt everything), the default behaviour is to just sit there until someone comes along and notices the panic message on the screen and reboots the machine. However if a machine is running unattended in an isolated location it may be desirable for it to automatically reset itself so that the machine comes back on line. For example, using `panic=30` at boot would cause the kernel to try and reboot itself 30 seconds after the kernel panic happened. A value of zero gives the default behaviour, which is to wait forever.

Note that this timeout value can also be read and set via the `/proc/sys/kernel/panic` sysctl interface.

The `pci=' Argument

The `pirq=' Argument

Using this option tells a SMP kernel information on the PCI slot versus IRQ settings for SMP motherboards which are unknown (or known to be blacklisted). See `linux/Documentation/IO−APIC.txt` for more information.

The `profile=' Argument

Kernel developers can enable an option that allows them to profile how and where the kernel is spending its CPU cycles in an effort to maximize efficiency and performance. This option lets you set the profile shift count at boot. Typically it is set to two. You can also compile your kernel with profiling enabled by default. In either case, you need a tool such as `readprofile.c` that can make use of the `/proc/profile` output.

The `reboot=' Argument

This option controls the type of reboot that Linux will do when it resets the computer (typically via `/sbin/init` handling a Control−Alt−Delete). The default as of v2.0 kernels is to do a `cold' reboot (i.e. full reset, BIOS does memory check, etc.) instead of a `warm' reboot (i.e. no full reset, no memory check). It was changed to be cold by default since that tends to work on cheap/broken hardware that fails to reboot when a warm reboot is requested. To get the old behaviour (i.e. warm reboots) use `reboot=w` or in fact any word that starts with `w` will work.

Why would you bother? Some disk controllers with cache memory on board can sense a warm reboot, and flush any cached data to disk. Upon a cold boot, the card may be reset and the write−back data in your cache card's memory is lost. Others have reported systems that take a long time to go through the memory check, and/or SCSI BIOSes that take longer to initialize on a cold boot as a good reason to use warm reboots.

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 (with the name `reserved`). 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 that do a `check_region()` prior to probing blindly into a region of I/O space. 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.

The `vga=' Argument

Note that this is not really a boot argument. It is an option that is interpreted by LILO and not by the kernel like all the other boot arguments are. However its use has become so common that it deserves a mention here. It can also be set via using `rdev −v` or equivalently `vidmode` on the vmlinuz file. This allows the setup code to use the video BIOS to change the default display mode before actually booting the Linux kernel. Typical modes are 80x50, 132x44 and so on. The best way to use this option is to start with `vga=ask` which will prompt you with a list of various modes that you can use with your video adapter before booting the kernel. Once you have the number from the above list that you want to use, you can later put it in place of the `ask'. For more information, please see the file `linux/Documentation/svga.txt` that comes with all recent kernel versions.

Note that newer kernels (v2.1 and up) have the setup code that changes the video mode as an option, listed as `Video mode selection support` so you need to enable this option if you want to use this feature.

4.Boot Arguments to Control PCI Bus Behaviour (`pci=')

The `pci=' argument (not avail. in v2.0 kernels) can be used to change the behaviour of PCI bus device probing and device behaviour. Firstly the file `linux/drivers/pci/pci.c` checks for architecture independent `pci=` options. The remaining allowed arguments are handled in `linux/arch/???/kernel/bios32.c` and are listed below for ???=i386.

4.1 The `pci=bios' and `pci=nobios' Arguments

These are used to set/clear the flag indicating that the PCI probing is to take place via the PCI BIOS. The default is to use the BIOS.

4.2 The `pci=conf1' and `pci=conf2' Arguments

If PCI direct mode is enabled, the use of these enables either configuration Type 1 or Type 2. These implicitly clear the PCI BIOS probe flag (i.e. `pci=nobios') too.

4.3 The `pci=io=' Argument

If you get a message like `PCI: Unassigned IO` space for.../ then you may need to supply an I/O value with this option. From the source:

``Several BIOS'es forget to assign addresses to I/O ranges. We try to fix it here, expecting there are free addresses starting with `0x5800`. Ugly, but until we come with better resource management, it's the only simple solution.''

4.4 The `pci=nopeer' Argument

This disables the default peer bridge fixup, which according to the source does the following:

``In case there are peer host bridges, scan bus behind each of them. Although several sources claim that the host bridges should have header type 1 and be assigned a bus number as for PCI2PCI bridges, the reality doesn't pass this test and the bus number is usually set by BIOS to the first free value.''

4.5 The `pci=nosort' Argument

Using this argument instructs the kernel to not sort the PCI devices during the probing phase.

4.6 The `pci=off' Argument

Using this option disables all PCI bus probing. Any device drivers that make use of PCI functions to find and initialize hardware will most likely fail to work.

4.7 The `pci=reverse' Argument

This option will reverse the ordering of the PCI devices on that PCI bus.

5.Boot Arguments for Video Frame Buffer Drivers

The `video=' argument (not avail. in v2.0 kernels) is used when the frame buffer device abstraction layer is built into the kernel. If that sounds complicated, well it isn't really too bad. It basically means that instead of having a different video program (the X11R6 server) for each brand of video card (e.g. XF86_S3, XF86_SVGA, ...), the kernel would have a built in driver available for each video card and export a single interface for the video program so that only one X11R6 server (XF86_FBDev) would be required. This is similar to how networking is now − the kernel has drivers available for each brand of network card and exports a single network interface so that just one version of a network program (like Netscape) will work for all systems, regardless of the underlying brand of network card.

The typical format of this argument is `video=name:option1,option2,...` where `name` is the name of a generic option or of a frame buffer driver. The `video=` option is passed from `linux/init/main.c` into `linux/drivers/video/fbmem.c` for further processing. Here it is checked for some generic options before trying to match to a known driver name. Once a driver name match is made, the comma separated option list is then passed into that particular driver for final processing. The list of valid driver names can be found by reading down the `fb_drivers` array in the file `fbmem.c` mentioned above.

Information on the options that each driver supports will eventually be found in `linux/Documentation/fb/` but currently (v2.2) only a few are described there. Unfortunately the number of video drivers and the number of options for each one is content for another document itself and hence too much to list here.

If there is no Documentation file for your card, you will have to get the option information directly from the driver. Go to `linux/drivers/video/` and look in the appropriate `???fb.c` file (the ??? will be based on the card name). In there, search for a function with `_setup` in its name and you should see what options the driver tries to match, such as `font` or `mode` or...

5.1 The `video=map:...' Argument

This option is used to set/override the console to frame buffer device mapping. A comma separated list of numbers sets the mapping, with the value of option N taken to be the frame buffer device number for console N.

5.2 The `video=scrollback:...' Argument

A number after the colon will set the size of memory allocated for the scrollback buffer. (Use Shift and Page Up or Page Down keys to scroll.) A suffix of `k' or `K' after the number will indicate that the number is to be interpreted as kilobytes instead of bytes.

5.3 The `video=vc:...' Argument

A number, or a range of numbers (e.g. `video=vc:2−5`) will specify the first, or the first and last frame buffer virtual console(s). The use of this option also has the effect of setting the frame buffer console to *not* be the default console.

6.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.

6.1 Arguments for Mid−level Drivers

The mid level drivers handle things like disks, CD−ROMs and tapes without getting into host adapter specifics.

Maximum Probed LUNs (`max_scsi_luns=')

Each SCSI device can have a number of `sub−devices' contained within itself. The most common example is any of the 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.

The kernel has a 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

SCSI Logging (`scsi_logging=')

Supplying a non−zero value to this boot argument turns on logging of all SCSI events (error, scan, mlqueue, mlcomplete, llqueue, llcomplete, hlqueue, hlcomplete). Note that better control of which events are logged can be obtained via the `/proc/scsi/scsi` interface if you aren't interested in the events that take place at boot before the `/proc/` filesystem is accessible.

Parameters for the SCSI Tape Driver (`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 first 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.

6.2 Arguments for SCSI Host Adapters

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 dependent 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 peripherals like IDE hard disks, floppies, serial ports, etc.

`dma` −− the DMA (Direct Memory Access) channel that the card uses. Typically only applies to bus−mastering cards. PCI and VLB cards are native bus−masters, and do not require and ISA DMA channel.

`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.

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 argument 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.

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 argument 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.

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.

AdvanSys SCSI Host Adaptors (`advansys=')

The AdvanSys driver can accept up to four i/o addresses that will be probed for an AdvanSys SCSI card. Note that these values (if used) do not effect EISA or PCI probing in any way. They are only used for probing ISA and VLB cards. In addition, if the driver has been compiled with debugging enabled, the level of debugging output can be set by adding an `0xdeb[0−f]` parameter. The `0−f` allows setting the level of the debugging messages to any of 16 levels of verbosity.

Always IN2000 Host Adaptor (`in2000=')

Unlike other SCSI host boot arguments, the IN2000 driver uses ASCII string prefixes for most of its integer arguments. Here is a list of the supported arguments:

ioport:addr −− Where addr is IO address of a (usually ROM−less) card.

noreset −− No optional args. Prevents SCSI bus reset at boot time.

nosync:x −− x is a bitmask where the 1st 7 bits correspond with the 7 possible SCSI devices (bit 0 for device #0, etc). Set a bit to PREVENT sync negotiation on that device. The driver default is sync DISABLED on all devices.

period:ns −− ns is the minimum # of nanoseconds in a SCSI data transfer period. Default is 500; acceptable values are 250 to 1000.

disconnect:x −− x = 0 to never allow disconnects, 2 to always allow them. x = 1 does 'adaptive' disconnects, which is the default and generally the best choice.

debug:x If `DEBUGGING_ON' is defined, x is a bitmask that causes various types of debug output to printed − see the DB_xxx defines in in2000.h

proc:x −− If `PROC_INTERFACE' is defined, x is a bitmask that determines how the /proc interface works and what it does − see the PR_xxx defines in in2000.h

Some example usages are listed below:

```
in2000=ioport:0x220,noreset
in2000=period:250,disconnect:2,nosync:0x03
in2000=debug:0x1e
in2000=proc:3
```

AMD AM53C974 based hardware (`AM53C974=')

Unlike other drivers, this one does not use boot parameters to communicate i/o, IRQ or DMA channels. (Since the AM53C974 is a PCI device, there shouldn't be a need to do so.) Instead, the parameters are used to communicate the transfer modes and rates that are to be used between the host and the target device. This is best described with an example:

```
AM53C974=7,2,8,15
```

This would be interpreted as follows: `For communication between the controller with SCSI−ID 7 and the device with SCSI−ID 2, a transfer rate of 8MHz in synchronous mode with max. 15 bytes offset should be negotiated.' More details can be found in the file `linux/drivers/scsi/README.AM53C974`

BusLogic SCSI Hosts with v1.2 kernels (`buslogic=')

In older kernels, 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`.

BusLogic SCSI Hosts with v2.x kernels (`BusLogic=')

With v2.x kernels, the BusLogic driver accepts many parameters. (Note the case in the above; upper case B and L!!!). There are simply too many to list here. A complete description is tucked away in the middle of the driver `linux/drivers/scsi/BusLogic.c` and searching for the string `BusLogic=` will put you right on it.

EATA SCSI Cards (`eata=')

As of late v2.0 kernels, the EATA drivers will accept a boot argument to specify the i/o base(s) to be probed. It is of the form:

```
eata=iobase1[,iobase2][,iobase3]...[,iobaseN]
```

The driver will probe the addresses in the order that they are listed.

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 recognized then it will also not be found. In either case, you will then have to use a boot argument 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`.

Future Domain TMC−16xx, TMC−3260, AHA−2920 (`fdomain=')

The driver detects these cards according to a list of known BIOS ROM signatures. For a full list of known BIOS revisions, please see `linux/drivers/scsi/fdomain.c` as it has a lot of information at the top of that file. If your BIOS is not known to the driver, you can use an override of the form:

```
        fdomain=iobase,irq[,scsi_id]
```

IOMEGA Parallel Port / ZIP drive (`ppa=')

This driver is for the IOMEGA Parallel Port SCSI adapter which is embedded into the IOMEGA ZIP drives. It may also work with the original IOMEGA PPA3 device. The boot argument for this driver is of the form:

```
        ppa=iobase,speed_high,speed_low,nybble
```

with all but iobase being optionally specified values. If you wish to alter any of the three optional parameters, you are advised to read `linux/drivers/scsi/README.ppa` for details of what they control.

NCR5380 based controllers (`ncr5380=')

Depending on your board, the 5380 can be either i/o mapped or memory mapped. (An address below 0x400 usually implies i/o mapping, but PCI and EISA hardware use i/o addresses above 0x3ff.) In either case, you specify the address, the IRQ value and the DMA channel value. An example for an i/o mapped card would be: `ncr5380=0x350,5,3`. If the card doesn't use interrupts, then an IRQ value of 255 (`0xff`) will disable interrupts. An IRQ value of 254 means to autoprobe. More details can be found in the file `linux/drivers/scsi/README.g_NCR5380`

NCR53c400 based controllers (`ncr53c400=')

The generic 53c400 support is done with the same driver as the generic 5380 support mentioned above. The boot argument is identical to the above with the exception that no DMA channel is used by the 53c400.

NCR53c406a based controllers (`ncr53c406a=')

This driver uses a boot argument of the form:

```
        ncr53c406a=PORTBASE,IRQ,FASTPIO
```

where the IRQ and FASTPIO parameters are optional. An interrupt value of zero disables the use of interrupts. Using a value of one for the FASTPIO parameter enables the use of `insl` and `outsl` instructions instead of the single−byte `inb` and `outb` instructions. The driver can also use DMA as a compile−time option.

Pro Audio Spectrum (`pas16=')

The PAS16 uses a NCR5380 SCSI chip, and newer models support jumper−less configuration. The boot argument 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`.

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 recognized then it will also not be found. In either case, you will then have to use a boot argument 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`.

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`.

Ultrastor SCSI cards (`u14−34f=')

Note that there appears to be two independent drivers for this card, namely `CONFIG_SCSI_U14_34F` that uses `u14−34f.c` and `CONFIG_SCSI_ULTRASTOR` that uses `ultrastor.c`. It is the u14−34f one that (as of late v2.0 kernels) accepts a boot argument of the form:

```
u14-34f=iobase1[,iobase2][,iobase3]...[,iobaseN]
```

The driver will probe the addresses in the order that they are listed.

Western Digital WD7000 cards (`wd7000=')

The driver probe for the wd7000 looks for a known BIOS ROM string and knows about a few standard configuration settings. If it doesn't come up with the correct values for your card, or you have an unrecognized BIOS version, you can use a boot argument of the form:

```
wd7000=irq,dma,iobase
```

6.3 SCSI Host Adapters 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.

```
Adaptec aha1740 (EISA probing),
NCR53c7xx,8xx (PCI, both drivers)
Qlogic Fast (0x230, 0x330)
Qlogic ISP (PCI)
```

7.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.

7.1 IDE Disk/CD−ROM Driver Parameters

The IDE driver accepts a number of parameters, which range from disk geometry specifications, to support for advanced or broken controller chips. The following is a summary of all the possible boot arguments. For full details, you *really* should consult the file `ide.txt` in the `linux/Documentation` directory, from which this summary was extracted.

```
"hdx="  is recognized for all "x" from "a" to "h", such as "hdc".
"idex=" is recognized for all "x" from "0" to "3", such as "ide1".

"hdx=noprobe"          : drive may be present, but do not probe for it
"hdx=none"             : drive is NOT present, ignore cmos and do not probe
"hdx=nowerr"           : ignore the WRERR_STAT bit on this drive
"hdx=cdrom"            : drive is present, and is a cdrom drive
"hdx=cyl,head,sect"    : disk drive is present, with specified geometry
"hdx=autotune"         : driver will attempt to tune interface speed
                               to the fastest PIO mode supported,
                               if possible for this drive only.
                               Not fully supported by all chipset types,
                               and quite likely to cause trouble with
                               older/odd IDE drives.

"idex=noprobe"         : do not attempt to access/use this interface
"idex=base"            : probe for an interface at the addr specified,
                               where "base" is usually 0x1f0 or 0x170
                               and "ctl" is assumed to be "base"+0x206
"idex=base,ctl"        : specify both base and ctl
"idex=base,ctl,irq"    : specify base, ctl, and irq number
"idex=autotune"        : driver will attempt to tune interface speed
                               to the fastest PIO mode supported,
                               for all drives on this interface.
                               Not fully supported by all chipset types,
                               and quite likely to cause trouble with
                               older/odd IDE drives.
"idex=noautotune"      : driver will NOT attempt to tune interface speed
                               This is the default for most chipsets,
                               except the cmd640.
"idex=serialize"       : do not overlap operations on idex and ide(x^1)
```

The following are valid ONLY on ide0, and the defaults for the base,ctl ports must not be altered.

```
"ide0=dtc2278"         : probe/support DTC2278 interface
"ide0=ht6560b"         : probe/support HT6560B interface
"ide0=cmd640_vlb"      : *REQUIRED* for VLB cards with the CMD640 chip
                         (not for PCI -- automatically detected)
"ide0=qd6580"          : probe/support qd6580 interface
```

```
"ide0=ali14xx"         : probe/support ali14xx chipsets (ALI M1439/M1445)
"ide0=umc8672"         : probe/support umc8672 chipsets
```

Everything else is rejected with a "BAD OPTION" message.

7.2 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.

7.3 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 recognized then it will also not be found. In either case, you will then have to use a boot argument 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
```

8.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.

Note that most of these CD−ROMs have documentation files that you *should* read, and they are all in one handy place: `linux/Documentation/cdrom`.

8.1 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.

8.2 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.

8.3 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.

8.4 The GoldStar Interface (`gscd=')

The syntax for this CD−ROM interface is:

```
        gscd=iobase
```

8.5 The ISP16 Interface (`isp16=')

The syntax for this CD−ROM interface is:

```
        isp16=[port[,irq[,dma]]][[,]drive_type]
```

Using a zero for `irq` or `dma` means that they are not used. The allowable values for `drive_type` are `noisp16, Sanyo, Panasonic, Sony,` and `Mitsumi`. Using `noisp16` disables the driver altogether.

8.6 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`.

8.7 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.

8.8 The Optics Storage Interface (`optcd=')

The syntax for this type of card is:

```
        optcd=iobase
```

8.9 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.

8.10 The Sanyo Interface (`sjcd=')

The syntax for this type of card is:

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

8.11 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.

9.Serial and ISDN Drivers

9.1 The ICN ISDN driver (`icn=')

This ISDN driver expects a boot argument of the form:

```
icn=iobase,membase,icn_id1,icn_id2
```

where `iobase` is the i/o port address of the card, `membase` is the shared memory base address of the card, and the two `icn_id` are unique ASCII string identifiers.

9.2 The PCBIT ISDN driver (`pcbit=')

This boot argument takes integer pair arguments of the form:

```
        pcbit=membase1,irq1[,membase2,irq2]
```

where `membaseN` is the shared memory base of the N'th card, and `irqN` is the interrupt setting of the N'th card. The default is IRQ 5 and membase `0xD0000`.

9.3 The Teles ISDN driver (`teles=')

This ISDN driver expects a boot argument of the form:

```
        teles=iobase,irq,membase,protocol,teles_id
```

where `iobase` is the i/o port address of the card, `membase` is the shared memory base address of the card, `irq` is the interrupt channel the card uses, and `teles_id` is the unique ASCII string identifier.

9.4 The DigiBoard Driver (`digi=')

The DigiBoard driver accepts a string of six comma separated identifiers or integers. The 6 values in order are:

```
        Enable/Disable this card
        Type of card: PC/Xi(0), PC/Xe(1), PC/Xeve(2), PC/Xem(3)
        Enable/Disable alternate pin arrangement
        Number of ports on this card
        I/O Port where card is configured (in HEX if using string identifiers)
        Base of memory window (in HEX if using string identifiers)
```

An example of a correct boot prompt argument (in both identifier and integer form) is:

```
        digi=E,PC/Xi,D,16,200,D0000
        digi=1,0,0,16,512,851968
```

Note that the driver defaults to an i/o of `0x200` and a shared memory base of `0xD0000` in the absence of a `digi=` boot argument. There is no autoprobing performed. More details can be found in the file `linux/Documentation/digiboard.txt`.

9.5 The RISCom/8 Multiport Serial Driver (`riscom8=')

Up to four boards can be supported by supplying four unique i/o port values for each individual board installed. Other details can be found in the file `linux/Documentation/riscom8.txt`.

9.6 The Baycom Serial/Parallel Radio Modem (`baycom=')

The format of the boot argument for these devices is:

```
        baycom=modem,io,irq,options[,modem,io,irq,options]
```

Using modem=1 means you have the ser12 device, modem=2 means you have the par96 device. Using options=0 means use hardware DCD, and options=1 means use software DCD. The `io` and `irq` are the i/o port base and interrupt settings as usual. There is more details in the file `README.baycom` which is currently in the `/linux/drivers/char/` directory.

10.Other Hardware Devices

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

10.1 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.

IMPORTANT NOTE TO MODULE USERS: The above will *not* force a probe for a second card if you are using the driver(s) as run time loadable modules (instead of having them complied into the kernel). Most Linux distributions use a bare bones kernel combined with a large selection of modular drivers. The `ether=` only applies to drivers compiled directly into the kernel.

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. Ethernet−HowTo

10.2 The Floppy Disk Driver (`floppy=')

There are many floppy driver options, and they are all listed in `README.fd` in `linux/drivers/block`. There are too many options in that file to list here. Instead, only those options that may be required to get a Linux install to proceed on less than normal hardware are reprinted here.

`floppy=0,daring` Tells the floppy driver that your floppy controller should be used with caution (disables all daring operations).

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

`floppy=nodma` Tells the floppy driver not to use DMA for data transfers. This is needed on HP Omnibooks, which don't have a workable DMA channel for the floppy driver. This option is also useful if you frequently get "Unable to allocate DMA memory" messages. Use of `nodma' is not recommended if you have a FDC without a FIFO (8272A or 82072). 82072A and later are OK). The FDC model is reported at boot. You also need at least a 486 to use nodma.

`floppy=nofifo` Disables the FIFO entirely. This is needed if you get `Bus master arbitration error' messages from your Ethernet card (or from other devices) while accessing the floppy.

`floppy=broken_dcl` Don't use the disk change line, but assume that the disk was changed whenever the device node is reopened. Needed on some boxes where the disk change line is broken or unsupported. This should be regarded as a stopgap measure, indeed it makes floppy operation less efficient due to unneeded cache flushings, and slightly more unreliable. Please verify your cable connection and jumper settings if you have any DCL problems. However, some older drives, and also some Laptops are known not to have a DCL.

`floppy=debug` Print (additional) debugging messages.

`floppy=messages` Print informational messages for some operations (disk change notifications, warnings about over and underruns, and about autodetection).

10.3 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 and the documentation for it in the kernel mysteriously vanished (a hint). You are better off to use sound as a module, or compile in your own values.

If you choose to use it regardless, then processing of the argument takes place in the file `dev_table.c` 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 `0xDTaaaId` and the bytes are used as follows:

D − second DMA channel (zero if not applicable)

T − device type: 1=FM, 2=SB, 3=PAS, 4=GUS, 5=MPU401, 6=SB16, 7=SB16−MIDI,... The listing of soundcard types up to 26 (don't forget to convert back to hex for command line use) are listed in the file `linux/include/linux/soundcard.h` and 27 to 999 (newer models) can be found in the file `linux/drivers/sound/dev_table.h`.

aaa − I/O address in hex.

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

d − First DMA channel.

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

10.4 The Bus Mouse Driver (`bmouse=')

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

10.5 The MS Bus Mouse Driver (`msmouse=')

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

10.6 The Printer Driver (`lp=')

With this boot argument you can tell the printer driver what ports to use and what ports *not* to use. The latter comes in handy if you don't want the printer driver to claim all available parallel ports, so that other drivers (e.g. PLIP, PPA) can use them instead.

The format of the argument is multiple i/o, IRQ pairs. For example, `lp=0x3bc,0,0x378,7` would use the port at `0x3bc` in IRQ−less (polling) mode, and use IRQ 7 for the port at `0x378`. The port at `0x278` (if any) would not be probed, since autoprobing only takes place in the absence of a `lp=` argument. To disable the printer driver entirely, one can use `lp=0`.

11.Copying, Translations, Closing, etc.

Hey, you made it to the end! (Phew...) Now just the legal stuff.

11.1 Copyright and Disclaimer

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 stuff goes up in smoke, or anything else bad happens, we 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.

A hint to people considering doing a translation. First, translate the SGML source (available via FTP from the HowTo main site) so that you can then generate other output formats. Be sure to keep a copy of the original English SGML source that you translated from! When an updated HowTo is released, get the new SGML source for that version, and then a simple `diff −u old.sgml new.sgml` will show you exactly what has changed so that you can easily incorporate those changes into your translated SMGL source without having to re−read or re−translate everything.

If you are intending to incorporate this document into a published work, please make contact (via e−mail) so that you can be supplied with 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.

11.2 Closing

If you have found any glaring typos, or outdated info in this document, please let me know. It is easy to overlook stuff, as the kernel (and the number of drivers) is huge compared to what it was when I started this.

Thanks,

Paul Gortmaker, `p_gortmaker@yahoo.com`
