

Boot Linux from USB Hard Disk

Written 7. Oct. 2007 by Urs_Lindegger@Bluewin.Ch

About this Document

I have a laptop where I'm not allowed to install Linux. However on business trips I'd like to use Linux with all my files. May 2005 I managed to install Gentoo Linux on a USB Hard disk and like to share my share my hardly gained experience with others to make their life more easy.

Why to install Linux on a USB drive?

- You can turn your company laptop into your private one without touching the company hard disk. On business trips you will not carry two laptops.
- You can shrink your PC-environment to a small USB device, that you can carry around in your pocket.
- You need it to do some recovery work on computer.
- You can use the USB hard disk to install Linux on a computer not having a CD reader.
- You want to explore and learn how Linux boots

BIOS

First your BIOS has to make the job. It has to be able to boot from USB. Go into setup of your BIOS and check if your USB device is visible. Usually it appears as hard disk. Change the order of boot devices, so your USB device is on top of the list.

If your BIOS does not support booting from BIOS, than the following should work: Boot it from an other drive (maybe old fashion floppy) and then pass it to USB.

Grub the boot loader

BIOS starts Grub and Grub lets you select the kernel to boot. The kernel is copied into Ram and Grub starts the kernel and disappears.

Problem during boot

The kernel starts and looks for the root file system, but there is no chance so far with current kernels to find it when needed. The BIOS software was used to load the kernel in RAM. However when the kernel gets alive, it will start its own high sophisticated USB software. There is a gap where no connection to the USB hard disk exist. This causes a failure of the boot process (Kernel panics).

Solution with Kernel parameter

On 1 June 2007 I tested successfully a easy method to boot from the USB hard disk. There is the kernel parameter `rootdelay`, that gives the kernel time to setup the connection the USB hard disk and therefore the kernel panics do not appear. All necessary to be done are in `grub.config` to call the kernel with the `rootdelay` parameter. The example below shows what has to be put into `grub.config` to get a 10s delay:

```
title=Gentoo Linux 2.6.18-r6 rootdelay
root (hd0,0)
kernel /kernel-2.6.18-gentoo-r6 root=/dev/sda3 rootdelay=10
```

Solution with INITRD

Before I discovered the solution with the kernel parameter (it might not been available at the time) I used the following approach for over two years. Now it can be consider as a historic way to do the same. It might be helpful to solve other boot issues.

Quick install

If you don't care about how it works but you just would like that it works read here. Other wise jump to the [next chapter](#).

1. Install Gentoo as written in the handbook except that you install it on the USB device.
2. If you compile the kernel make sure you have set `CONFIG_BLK_DEV_RAM` and `CONFIG_BLK_DEV_INITRD`.
3. Download [initrd.gz](#) from the [internet](#) and copy it to `/boot`
4. Modify your [/boot/grub/grub.conf](#) to have something as:

```
title=Gentoo Linux 2.6.18-r6 linuxrc
root (hd0,0)
kernel /kernel-2.6.18-gentoo-r6 root=/dev/ram0 init=/linuxrc rw
```

`initrd /boot/initrd.gz`

5. Now boot your PC and make sure it boots from the USB device first

How it works in Details

I used Gentoo Linux and Grub, but it should also work with other distributions. However some distributions use also `linuxrc` and `initrd` to detect the hardware and modify the configuration files before they start the desired configuration (e.g. www.knoppix.org). This could cause conflicts with the method described here.

Maybe there are still missing things, problems, unnecessary things, or better ways. So let me know if you have any comment.

So far the kernel is optimized for just one computer type. The `linuxrc` script could be expanded to load device drivers according the hardware found.

Sample code is not included within this document to make updates easier. However you will find it on my [web page](#).

Way out

Kernel patches seem to be around. I did not find one for the 2.6 kernel. Additionally, each time a new kernel source comes out, the patch has to be applied and hopefully it is still compatible.

An other way is not using the USB at all for booting the kernel. Something more quick and more easy to access then the USB drive is the internal RAM. Grub supports to load a file into the RAM. The name of this file is usually `initrd` (initial RAM disk) or if it is compressed `initrd.gz`. This single compressed file is expanded to a root file system containing all necessary single files.

Additionally to grub, the kernel 2.6.xx allows dealing with such files. Add `root=/dev/ram0` in grub to indicate to the kernel to unzip the `initrd.gz` and mount it to the ram disk `/dev/ram0`. The file will be read from the kernel via the device `/dev/initrd` and unpacked and written to `/dev/ram0`.

By default the kernel starts `/sbin/init` after boot. `/sbin/init` accesses the `/etc/inittab` that manages the init V boot process with all its runlevels. To

have it simple, you can pass `init=/bin/sh` to the kernel via grub to let the kernel start the shell instead of `/sbin/init`, this might be helpful for debugging. With the same mechanism you can tell the kernel to start the `linuxrc` script that is found in `/dev/ram0`.

The `linuxrc` can be a script or an executable. However make sure that all files are in `initrd` that `linuxrc` requires. e.g.: If `linuxrc` is a script, it can not run without `bash`, so copy `bash` to the `initrd`. Note `bash` can not run, since its libraries are missing. Type `ldd /bin/bash` and you'll see what you have to copy to the `/lib` directory of the initial ram disk. How to create a `initrd.gz` file and `linuxrc` script will follow later in this document.

It is up to the `linuxrc` script what will happen after the kernel is alive. It can either start a shell for debugging or change root from the ram disk to the newly mounted USB hard disks. This is done via `pivot_root` command followed by a regularly start of `/sbin/init`.

Kernel

This document assumes that you are not use the Gentoo genkernel (since it already uses `initrd` and `linuxrc`, it would get complicated dealing with two different `initrd`).

Putting all necessary kernel modules for the boot process into the 2.6.xx kernel makes your live easier.

Make sure you have configured and compiled the kernel with `ramdisk` and `initrd` support.

Add also support for the `loop devices`, since the `mkinitrd` script will use them. The `/dev/loop` is also used when you want to customizing `initrd`.

How to install Gentoo on a USB Hard disk

First, a way how to work must be decided. One way is installing gentoo Linux on a USB hard disk using a regular PC, where gentoo already is installed on a hard disk (other methods are knoppix or gentoo live cd). The USB hard disk gets formatted and installed as written in the Gentoo installation handbook. Replaced `hda` with `sda`. Create a `/initrd` directory on your hard disk. The `pivot_root` command will move the initial ram disk this to this directory.

How to create an initrd and linuxrc

As already mentioned, `initrd` is a file containing the contents of the ram disk (=root file system) that will be used during the boot. To work with

it, create a directory e.g. `/root/USBHD` and put all necessary files in it, including the `linuxrc` script.

To convert the `/root/USBHD` files in `initrd.gz` first the directory `/mnt/initrd` has to be created. Then the script `mkinitrd` takes all files from `/root/USBHD`, creates the necessary `/dev` files and creates from that the `initrd.gz` file.

To do this work, an empty `initrd` file with a certain size is first created by `mkinitrd`. Then this file will be formatted as ext2 and mounted to `/mnt/initrd`. Now it can be accessed as it would be a regular filesystem on a memory stick. So all files can be copied there and `/dev` nodes can be created.

Finally the `/mnt/initrd` has to be unmounted and the now no more empty `initrd` has to be zipped to `initrd.gz`.

The script `mkinitrd` assumes that it has been put in `/root`. After running `/root/mkinitrd` move the `initrd.gz` to the USB hard disk e.g. `/mnt/gentoo/boot` and make sure the `/mnt/gentoo/boot/grub/grub.conf` points to it.

Customizing the sample `initrd.gz`

Instead of finding out yourself, what you have to copy to `/root/USBHD`, you can make it the other way round. Download my sample `initrd.gz` to `/root` and unpack it to have an `initrd` file. Mount this file: `mount -t ext2 -o loop /root/initrd /mnt/initrd`. Now you can do a `cp -R /mnt/initrd/* /root/USBHD` and customize the files in `/root/USBHD`. Then start `/root/mkinitrd` to create a customized `initrd.gz`.

An alternative to the above method is to first extract the `initrd.gz` to `initrd`, then `mount -t ext2 -o loop /root/initrd /mnt/initrd`. Edit the files directly in `/mnt/initrd`. Then `umount /mnt/initrd` and `gzip -9 initrd`. Now you find a customized `initrd.gz`.

The following samples files will give you a good start for your customization:

[initrd.gz](#)
[mkinitrd](#)

Sample code for `grub.conf`

```
default 0
timeout 5
splashimage=(hd0,0)/grub/splash.xpm.gz
```

```
title=Gentoo Linux 2.6.20-r7 linuxrc
root (hd0,0)
kernel /kernel-2.6.20-gentoo-r7 root=/dev/ram0 init=/linuxrc rw
initrd /boot/initrd.gz
```

```
title=Gentoo Linux 2.6.20-r7 /bin/sh
root (hd0,0)
kernel /kernel-2.6.20-gentoo-r7 root=/dev/ram0 init=/bin/sh rw
initrd /boot/initrd.gz
```

References

<usr/src/linux/Documentation/initrd.txt>

man initrd

man pivot_root

Hints

- **Kernel panic - not syncing: No init found.** Try passing `init=option to kernel`
It does not necessary mean that `init=option` did not happen, it can also be, that option or a depending file is missing.
- To check if the PC is alive plug in and out the USBHD and see if the system is alive and reports it.
- `Ldd` shows what libraries are required for your programs. e.g. type `ldd /bin/sh`
- `chroot initrd` to check if `linuxrc` would start running
- `linux-gate.so` seems to be a *.so ghost on my computer. I ignored it.
- Make two versions of Grub options, one that brings you into a shell where you can interactively work, the other version that starts `linuxrc`.
- Take care about synchronizing issues and add some sleep commands to the `linuxrc` script to give the kernel some time to find the USB hard disk and to mount it, before the `linuxrc` script goes too many some steps ahead (not very clean synchronizing, but simple). Does not delay boot up time too much since Linux is a real multitasking operating system. Other way around is looking into the `/proc` directory.
- Have every module in the kernel necessary to boot, so no worry and problems to put them into the `initrd`. `Initrd` and `linuxrc` are very specific to your application. It is some work to create them, but a lot to learn.
- To become familiar with the `initrd` approach, you can do a grub and `initrd` installation on a USB memory stick.