配置引导加载程序

< Handbook:AMD64 | Installation

本页是页面Handbook:AMD64/Installation/Bootloader的翻译版本，翻译工作已完成100%。

其他语言： Deutsch • English • español • français • italiano • polski • português do Brasil • čeština • русский • தமிழ் • 中文（中国大陆） • 日本語 • 한국어

AMD64 手册
安装
关于安装
选择安装媒介
配置网络
准备磁盘
安装 stage3
安装基础系统
配置内核
配置系统
安装系统工具
配置引导程序
安装收尾
使用 Gentoo
Portage 介绍
USE 标记
Portage 功能特性
Initscript 系统
环境变量
使用 Portage
文件和目录
变量
混合使用不同的软件分支
额外的工具
自定义软件包仓库
高级特性
配置网络
开始
高级配置
模块化网络
无线网络
添加功能
动态管理

目录 [隐藏]

选择引导加载程序

完成配置Linux内核、安装系统工具和编辑配置文件之后，现在是时候去安装Linux安装的最后一个重要的部分：引导加载程序。

引导加载程序负责在引导过程中启动内核——若没有引导加载程序，系统将不知道按下电源键后将如何进行。

针对 **amd64**，我们将介绍如何在基于 DOS/传统 BIOS的系统上配置 GRUB 和 LILO，以及针对 UEFI 系统如何配置 GRUB 和 efibootmgr。

在本手册的这一部分中，描述了 "emerging" 引导加载程序包和 "installing" 引导加载程序到系统磁盘之间的区别。 这里，术语 "emerging" 将用于请求 Portage 使软件包安装于系统。 术语 "installing" 将表示引导加载程序复制文件或物理地修改系统的磁盘驱动器的适当部分，以便在下一次开机时使引导加载程序"激活并准备好操作"。

默认：GRUB

默认情况下，Gentoo 系统现在主要依赖于 GRUB（在`sys-boot/grub` 包中），它是 GRUB Legacy 的继任者。无需额外配置，GRUB 就能支持旧的BIOS("pc") 系统。 在安装之前加上少量的配置，GRUB 可以支持超过一半的平台。 有关详细信息，请参阅位于 GRUB 的准备章节。

Emerge

当使用只支持MBR分区表的旧版BIOS系统时，无需进行其他配置即可安装GRUB：

```
root # emerge --ask --verbose sys-boot/grub
```

UEFI用户注意：运行上述命令将在出现之前输出启用的*GRUB_PLATFORMS* 值。 当使用支持UEFI的系统时，用户需要确保启用 `GRUB_PLATFORMS="efi-64"` 参数（默认情况下是这样）。 如果设置不是这样，则需要在安装 GRUB 之前将 `GRUB_PLATFORMS="efi-64"` 添加到`/etc/portage/make.conf`:

```
root # echo 'GRUB_PLATFORMS="efi-64"' >> /etc/portage/make.conf
```

```
root # emerge --ask sys-boot/grub
```

如果 GRUB 在未先添加 `GRUB_PLATFORMS="efi-64"` 到 `make.conf` 时就已经 emerge 过了，可以添加这一行（像上面显示那样）然后可以通过 `--update --newuse` options to **emerge** 选项来重新计算 world package set：

```
root # emerge --ask --update --newuse --verbose sys-boot/grub
```

GRUB 现在已经安装到*系统*中了，但是他还没有成为辅助*引导加载程序*（SBL）。

安装

接下来，通过**grub-install**命令安装 GRUB 所需的文件到`/boot/grub/`目录。假设第一块磁盘（引导系统的那块）是`/dev/sda`，将使用下面的一条命令：

DOS/传统 BIOS 系统

针对 DOS/传统 BIOS 系统：

```
root # grub-install /dev/sda
```

EFI 系统

重要
确保 EFI 系统分区在运行 **grub-install** *之前*就已经挂载。否则它可能会把 **grub-install** 安装的 GRUB EFI 文件（`grubx64.efi`到错误的目录，并且**不会**提供*任何*辨识使用错误目录的信息。

针对 EFI 系统：

```
root # grub-install --target=x86_64-efi --efi-directory=/efi
Installing for x86_64-efi platform.
Installation finished. No error reported.
```

Upon successful installation, the output should match the output of the previous command. If the output does not match exactly, then proceed to Debugging GRUB, otherwise jump to the Configure step.

Optional: Secure Boot

The `sys-boot/grub` package does not recognize the `secureboot` USE flag, this is because the GRUB EFI executable is not installed by the package but is instead built and installed by the **grub-install** command. GRUB must therefore be *manually* signed after installation to the boot partition. Additionally, GRUB is a modular bootloader but loading modules is prohibited when Secure Boot is enabled. Therefore all necessary modules must be compiled into the GRUB EFI executable, below an example is shown including some basic modules, this may have to be adjusted for more advanced configurations:

```
root # emerge --noreplace sbsigntools
```

```
root # export GRUB_MODULES="all_video boot btrfs cat chain configfile echo efifwsetup efinet ext2 fat font gettext gfxmenu gfxterm
gfxterm_background gzio halt help hfsplus iso9660 jpeg keystatus loadenv loopback linux ls lsefi lsefimmap lsefisystab lssal memdisk minicmd normal
ntfs part_apple part_msdos part_gpt password_pbkdf2 png probe reboot regexp search search_fs_uuid search_fs_file search_label sleep smbios squash4
test true video xfs zfs zfscrypt zfsinfo"
```

```
root # grub-install --target=x86_64-efi --efi-directory=/efi --modules=${GRUB_MODULES} --sbat /usr/share/grub/sbat.csv
```

```
root # sbsign /efi/EFI/GRUB/grubx64.efi --key /path/to/kernel_key.pem --cert /path/to/kernel_key.pem --out /efi/EFI/GRUB/grubx64.efi
```

To successfully boot with secure boot enabled the used certificate must either be accepted by the UEFI firmware, or shim must be used as a pre-loader. Shim is pre-signed with the third-party Microsoft Certificate, accepted by default by most UEFI motherboards.

How to configure the UEFI firmware to accept custom keys depends on the firmware vendor, which is beyond the scope of the handbook. Below is shown how to setup shim instead:

```
root # emerge sys-boot/shim sys-boot/mokutil sys-boot/efibootmgr
```

```
root # cp /usr/share/shim/BOOTX64.EFI /efi/EFI/GRUB/shimx64.efi
```

```
root # cp /usr/share/shim/mmx64.efi /efi/EFI/GRUB/mmx64.efi
```

Shims MOKlist requires keys in the *DER* format, since the OpenSSL key generated in the example here is in the *PEM* format, the key must be converted first:

```
root # openssl x509 -in /path/to/kernel_key.pem -inform PEM -out /path/to/kernel_key.der -outform DER
```

附注
The path used here must be the path to the pem file containing the certificate belonging to the generated key. In this example both key and certificate are in the same pem file.

Then the converted certificate can be imported into Shims MOKlist:

```
root # mokutil --import /path/to/kernel_key.der
```

And finally we register Shim with the UEFI firmware. In the following command, `boot-disk` and `boot-partition-id` must be replaced with the disk and partition identifier of the EFI system partition:

```
root # efibootmgr --create --disk /dev/boot-disk --part boot-partition-id --loader '\EFI\GRUB\shimx64.efi' --label 'shim' --unicode
```

调试 GRUB

When debugging GRUB, there are a couple of quick fixes that may result in a bootable installation without having to reboot to a new live image environment.

In the event that "EFI variables are not supported on this system" is displayed somewhere in the output, it is likely the live image was not booted in EFI mode and is presently in Legacy BIOS boot mode. The solution is to try the removable GRUB step mentioned below. This will overwrite the executable EFI file located at `/EFI/BOOT/BOOTX64.EFI`. Upon rebooting in EFI mode, the motherboard firmware may execute this default boot entry and execute GRUB.

重要
如果 **grub-install** 返回了一个错误，类似 `Could not prepare Boot variable: Read-only file system`，那么为了成功安装，可能必须需要将 efivars 重新挂载为读写：

```
root # mount -o remount,rw,nosuid,nodev,noexec --types efivarfs efivarfs /sys/firmware/efi/efivars
```

```
root # mount -o remount,rw,nosuid,nodev,noexec --types efivarfs efivarfs /sys/firmware/efi/efivars
```

This is caused by certain non-official Gentoo environments not mounting the special EFI filesystem by default. If the previous command does not run, then reboot using an official Gentoo live image environment in EFI mode.

一些主板制造商似乎只支持EFI系统分区（ESP）中.EFI文件的 `/efi/boot/`目录。 GRUB安装程序可以使用 `--removable` 选项自动执行此操作。 在运行以下命令之前验证是否已安装ESP。 假设ESP安装在`/boot`（如前所述），执行：

```
root # grub-install --target=x86_64-efi --efi-directory=/efi --removable
```

这将创建UEFI规范定义的默认目录，然后将 `grubx64.efi` 文件复制到由同一规范定义的"默认"EFI文件位置。

配置

接下来，基于用户在`/etc/default/grub`文件和`/etc/grub.d`中特别配置的脚本文件来生成 GRUB。在大多数场景中，不需要由用户来配置，GRUB 就可以自动检测出哪个内核用于引导（位于`/boot/`中最高的那一个）以及根文件系统是什么。也可以使用*GRUB_CMDLINE_LINUX*>变量在`/etc/default/grub`中附加内核参数。

要生成最终的 GRUB 配置，运行**grub-mkconfig**命令：

```
root # grub-mkconfig -o /boot/grub/grub.cfg
Generating grub.cfg ...
Found linux image: /boot/vmlinuz-6.1.38-gentoo
Found initrd image: /boot/initramfs-genkernel-amd64-6.1.38-gentoo
done
```

需要注意至少找到一个Linux镜像在命令的输出中，它们是用来引导系统的。如果使用一个initramfs或用**genkernel**建立内核，同样会检测到正确的initrd 镜像。如果不是这样，进入到`/boot/`并使用**ls**命令检查内容。如果文件确实不存在，回到内核配置和安装的介绍。

提示
os-prober 工具可与 GRUB 配合使用，以检测所连接驱动器上的其他操作系统。可检测到 Windows 7, 8.1, 10 和其他 Linux 发行版。 那些希望双引导系统的应该出现 `sys-boot/os-prober` 包，然后重新运行 **grub-mkconfig** 命令（如上所示）。 如果遇到问题，请务必先阅读 GRUB 文章，然后再向Gentoo社区请求支持。

备选 1：LILO

Emerge

LILO (the **LI**nux**LO**ader,) 是Linux引导程序的久经考验的主力。但是它缺少GRUB所拥有的一些特性。LILO仍旧在一些系统上被使用的原因是GRUB无法使用但LILO却可以。当然还因为一些人是先认识了LILO而且对它忠心不二。不管怎样，Gentoo可以支持它们两个启动器。

安装LILO是一件轻而易举的事，使用emerge就可以了。

```
root # emerge --ask sys-boot/lilo
```

配置

要配置LILO，首先要创建 `/etc/lilo.conf`:

```
root # nano -w /etc/lilo.conf
```

在配置文件中，小节（sections）被用于指向可引导的内核。请确保内核文件（与内核版本号一起）和initramfs文件都可以被知晓，因为它们都需要被这个配置文件所引用。

附注
如果根文件系统是JFS，请在每一个引导条目之后增加 `append="ro"` 因为JFS在它被挂载为可读写之前需要重放它的日志。

文件 `/etc/lilo.conf` **LILO 配置样本**

```
boot=/dev/sda             # 在 MBR 安装 LILO
prompt                    # 让用户有机会选择其他项
timeout=50                # 启动默认项前等待 5（五）秒
default=gentoo            # 超时后，启动 "gentoo" 项
compact                   # 这极大地减少了加载时间，并使 map 文件更小。在有些系统上可能会失效

image=/boot/vmlinuz-6.1.38-gentoo
  label=gentoo            # 给这个项目起的名字
  read-only               # 从只读 root 启动。不要修改这个!
  root=/dev/sda3          # 根文件系统位置

image=/boot/vmlinuz-6.1.38-gentoo
  label=gentoo.rescue     # 给这个项目起的名字
  read-only               # 从只读 root 启动。不要修改这个!
  root=/dev/sda3         # 根文件系统位置
  append="init=/bin/bb"   # 启动 Gentoo 静态救援 shell

# 下面两行用于与 Windows 系统双启动。
# 这个实例中，Windows 在 /dev/sda6。
other=/dev/sda6
  label=windows
```

附注
如果您使用不同的分区方案或内核文件，请根据需要进行调整。

如果initramfs是必须的，那么就更改配置文件以便引用这个initramfs文件，并且告诉initramfs根设备的所在位置。

文件 `/etc/lilo.conf` **添加initramfs信息到引导条目**

```
image=/boot/vmlinuz-6.1.38-gentoo
  label=gentoo
  read-only
  append="root=/dev/sda3"
  initrd=/boot/initramfs-genkernel-amd64-6.1.38-gentoo
```

如果额外的选项需要被传递到内核，使用 `append` 语句。例如增加 `video` 语句来使能framebuffer：

文件 `/etc/lilo.conf` **添加视频参数到引导选项**

```
image=/boot/vmlinuz-6.1.38-gentoo
  label=gentoo
  read-only
  root=/dev/sda3
  append="video=uvesafb:mtrr,ywrap,1024x768-32@85"
```

使用 **genkernel**的用户应该了解他们的内核使用与安装CD相同的引导选项。例如，如果对SCSI设备的支持需要被使能，就增加 `doscsi` 到内核选项中。

现在保存这个文件并退出。

安装

为了彻底完成，运行 `/sbin/lilo`，这样 LILO 就会把 `/etc/lilo.conf` 中的设置应用到系统中（也就是说安装它自己到磁盘上）。要记住每一次一个新内核被安装或者 `lilo.conf` 文件被改变后，`/sbin/lilo` 都需要执行一次，以确保在内核文件名发生改变后系统仍然能够被引导起来。

```
root # /sbin/lilo
```

备选 2：efibootmgr

在基于UEFI的系统上，系统上的UEFI固件（换句话说，主引导加载程序）可以直接操作以查找UEFI引导条目。 这样的系统不需要具有额外的（也称为辅助）引导加载器，如GRUB，以帮助引导系统。 据说，基于EFI的引导加载程序（如GRUB）存在的原因是在引导过程中"扩展"UEFI系统的功能。 使用**efibootmgr**是真正的那些想要采取一个极简主义（虽然更僵硬的）方法来启动他们的系统; 使用GRUB（见上文）对于大多数用户更容易，因为它在引导UEFI系统时提供了灵活的方法。

System administrators who desire to take a minimalist, although more rigid, approach to booting the system can avoid secondary bootloaders and boot the Linux kernel as an EFI stub.

记住`sys-boot/efibootmgr`应用程序不是一个引导器，它是一个和UEFI固件相互作用并更新它的设置，因为之前安装的Linux内核可以通过额外的选项（如果需要）来引导，或允许多重引导条目。可以通过EFI变量（需要支持EFI变量的内核）来完成这个相互作用。

一定要阅读通过 EFI stub内核文章"'再*继续。 内核必须具有能够被系统的UEFI固件直接引导的特定选项。 可能需要重新编译内核。 看看**efibootmgr** 文章，这也是一个好主意。*

It is also a good idea to take a look at the **efibootmgr** article for additional information.

附注
要重申，**efibootmgr** *不是*引导UEFI系统的要求。Linux内核本身就可以启动即引导，其他内核命令行选项可以内置到Linux内核（有一个内核配置选项 *CONFIG_CMDLINE*）允许用户指定启动参数作为命令行选项，甚至initramfs 可以"内置"到内核。

安装 **efibootmgr** 软件：

```
root # emerge --ask sys-boot/efibootmgr
```

创建 `/efi/efi/gentoo` 目录，并复制内核文件到这个位置，并命名为 `bzImage.efi`:

```
root # mkdir -p /boot/efi/boot
root # cp /boot/vmlinuz-* /boot/efi/boot/bzImage.efi
```

附注
UEFI定义强制要求使用 `\` 作为目录分割符。

接下来，告诉UEFI固件创建一个叫作"Gentoo"的引导条目，它拥有全新编译的EFI stub内核：

```
root # efibootmgr --create --disk /dev/sda --part 2 --label "Gentoo" --loader "\efi\boot\bzImage.efi"
```

如果使用一个内存文件系统（initramfs），为它添加相应的引导选项：

```
root # efibootmgr -c -d /dev/sda -p 2 -L "Gentoo" -l "\efi\boot\bzImage.efi" initrd='\initramfs-genkernel-amd64-6.1.38-gentoo'
```

Note that the above command presumes an initramfs file was copied into the ESP inside the same directory as the `bzImage.efi` file.

完成这些变更后，当系统重新启动时，会有一个叫作"gentoo"的引导项。

Unified Kernel Image

If installkernel was configured to build and install unified kernel images. The unified kernel image should already be installed to the `EFI/Linux` directory on the EFI system partition, if this is not the case ensure the directory exists and then run the kernel installation again as described earlier in the handbook.

To add a direct boot entry for the installed unified kernel image:

```
root # efibootmgr --create --disk /dev/sda --part 1 --label "gentoo" --loader /efi/EFI/Linux/gentoo-x.y.z.efi
```

备选 3：Syslinux

Syslinux是 **amd64**架构的另一种引导加载程序替代方案。 它不仅支持MBR，从版本6.00开始，它开始支持EFI启动。 还支持PXE（网络）引导和鲜为人知的选项。 尽管Syslinux是许多流行的引导加载程序，但它并没有得到手册的支持。 读者可以在Syslinux文章中找到有关新兴然后安装此引导加载程序的信息。

备选 4: systemd-boot

Another option is systemd-boot, which works on both OpenRC and systemd machines. It is a thin chainloader and works well with secure boot.

To install systemd-boot:

```
root # bootctl install
```

重要
Make sure the EFI system partition has been mounted *before* running **bootctl install**.

When using this bootloader, before rebooting, verify that a new bootable entry exists using:

```
root # bootctl list
```

If no new entry exists, ensure the `sys-kernel/installkernel` package has been installed with the `systemd-boot` USE flag enabled, and re-run the kernel installation.

For the distribution kernels:

```
root # emerge --ask --config sys-kernel/gentoo-kernel
```

For a manually configured and compiled kernel:

```
root # make install
```

重要
When installing kernels for systemd-boot, no **root=** kernel command line argument is added by default. On systemd systems that are using an initramfs users may rely instead on systemd-gpt-auto-generator to automatically find the root partition at boot. Otherwise users should manually specify the location of the root partition by setting **root=** in `/etc/kernel/cmdline` as well as any other kernel command line arguments that should be used. And then reinstalling the kernel as described above.

Optional: Secure Boot

When the `secureboot` USE flag is enabled, the systemd-boot EFI executable will be signed automatically. **bootctl install** will automatically install the signed version.

To successfully boot with secure boot enabled the used certificate must either be accepted by the UEFI firmware, or shim must be used as a pre-loader. Shim is pre-signed with the third-party Microsoft Certificate, accepted by default by most UEFI motherboards.

How to configure the UEFI firmware to accept custom keys depends on the firmware vendor, which is beyond the scope of the handbook. A postinst hook to automatically update systemd-boot and set it up with shim instead is provided on the systemd-boot wiki page. However the first time this should be done manually by following the steps below:

```
root # emerge --ask sys-boot/shim sys-boot/mokutil sys-boot/efibootmgr
```

```
root # cp /usr/share/shim/BOOTX64.EFI /efi/EFI/BOOT/BOOTX64.EFI
```

```
root # cp /usr/share/shim/mmx64.efi /efi/EFI/BOOT/mmx64.efi
```

```
root # cp /efi/EFI/systemd/systemd-bootx64.efi /efi/EFI/BOOT/grubx64.efi
```

附注
Shim is hardcoded to load `grubx64.efi`. As such the systemd-boot bootloader must be named as if it were GRUB.

Shims MOKlist requires keys in the *DER* format, since the OpenSSL key generated in the example here is in the *PEM* format, the key must be converted first:

```
root # openssl x509 -in /path/to/kernel_key.pem -inform PEM -out /path/to/kernel_key.der -outform DER
```

附注
The path used here must be the path to the pem file containing the certificate belonging to the generated key. In this example both key and certificate are in the same pem file.

Then the converted certificate can be imported into Shims MOKlist:

```
root # mokutil --import /path/to/kernel_key.der
```

And finally we register Shim with the UEFI firmware. In the following command, `boot-disk` and `boot-partition-id` must be replaced with the disk and partition identifier of the EFI system partition:

```
root # efibootmgr --create --disk /dev/boot-disk --part boot-partition-id --loader '\EFI\BOOT\BOOTX64.EFI' --label 'shim' --unicode
```

重启系统

退出chroot环境并unmount全部已持载分区。然后敲入一条有魔力的命令来初始化最终的、真实的测试：**reboot**。

```
(chroot) livecd # exit
```

```
livecd~# cd
livecd~# umount -l /mnt/gentoo/dev{/shm,/pts,}
livecd~# umount -R /mnt/gentoo
livecd~# reboot
```

当别忘了移除 live 镜像，否则可能再次从 live 镜像启动，而不是新安装的 Gentoo 系统!

当重启进全新安装的 Gentoo 环境后，最好进行完成Gentoo安装。

← 安装工具 | Home | 收尾安装工作 →

分类： Handbook

此页面最后编辑于2015年10月11日 (星期日) 11:02。