
Teil 1: Grundlagen und Konzepte von LFS

# Wunscheinrichtung

**Mithilfe von Linux from Scratch erstellen Sie eine für Ihre Zwecke maßgeschneiderte Distribution – etwa als Ablösung für ein bald nicht mehr unterstütztes CentOS.**

Markus Frei

**Nicht zuletzt** aufgrund des Rummels um die Abkündigung des Urgesteins CentOS 8 für Ende 2021 und daraus entstehenden Diskussionen um Alternativen wie Rocky Linux mag sich so mancher überlegt haben, ob es nicht möglich wäre, ein eigenes Linux zu bauen. Dieser Artikel erklärt auf der Basis von Linux from Scratch (LFS), wie das funktioniert.

Bei Linux from Scratch und Beyond Linux From Scratch (BLFS) handelt es sich um online verfügbare Anleitungen, die das Kompilieren und Zusammenstellen eines eigenen Linux-Systems direkt aus frei verfügbaren Quellen Schritt für Schritt beschreiben. LFS und BLFS werden meist zweimal im Jahr aktualisiert, im März und im September.

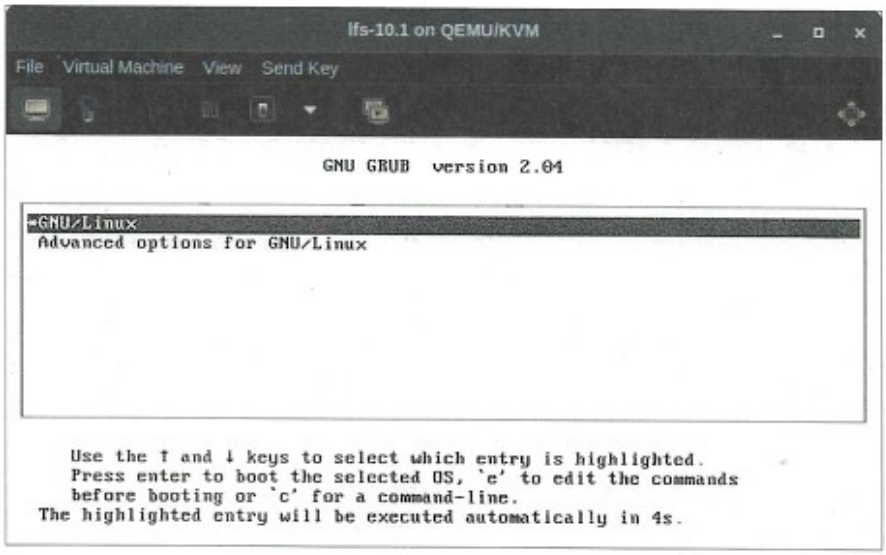
Gerard Beekmans, der Autor von LFS, begann 1998, sich mit Linux zu beschäftigen. Nach Einsatz diverser Distributionen reifte bei ihm die Erkenntnis, dass es für ihn kein „one size fits all" gab und er ein eigenes System benötigte. Open Source bot ihm per definitionem die Möglichkeit dazu. Er begann, zu kompilieren und sich mit Compile-Time-Errors sowie zirkulären Abhängigkeiten auseinanderzusetzen. Die gesammelten Erfahrungen, die er mit der Linux-Community teilte, stießen auf so breites Interesse, dass daraus das Projekt LFS entstand.

Neben LFS gibt es eine Reihe an Linux-Distributionsbaukästen, von denen sich die meisten auf den Embedded-Bereich spezialisieren und deren bekanntester Vertreter BitBake aus dem Yocto-Projekt ist. Daneben gibt es noch Linux Target Image Builder, OpenWrt und Scratchbox.

LFS versucht soweit wie möglich, gängigen Standards zu folgen. Dazu

gehören POSIX.1-2008, das eine Betriebssystemschnittstelle samt Environment zu Portabilitätszwecken definiert, der Filesystem Hierarchy Standard FHS in Version 3.0 sowie die Linux Standard Base LSB in Version 5.0 (2015). Letztere definiert vier teils kontrovers diskutierte Standards für Core, Desktop, Runtime Languages und Imaging, die LFS schon aufgrund seiner minimalen Softwareausstattung nur in Teilen umsetzen kann.

Folgen Sie der Anleitung, erhalten Sie ein minimales Linux-System 1 mit Ext4-Dateisystem, Grub 2.04, Kernel 5.10.17, Bash 5.1, GCC 10.2, Perl 5.32.1, Python 3.9.2 und Vim 8.2 (Stand LFS v10.1 von März 2021). Sie haben die Wahl zwischen LFS mit SysVinit und mit Systemd und D-Bus. Diese Artikelserie behandelt letztere Variante. Mit Systemd erhält Linux from Scratch nicht nur das modernere Init-System, sondern auch einen DHCP- und NTP-Client sowie andere Kleinigkeiten, die in der SysVinit-Variante erst in BLFS behandelt werden.




**1 LFS in der Grub-Auswahl. Der Bootloader gehört zur Minimalausstattung des selbst gebauten Linux.**

## Vorbemerkungen

Mit dem Herunterladen der Kernel-Quellen und dem Anwerfen des C-Compilers ist es allerdings nicht getan. LFS splittet das Erstellen eines Linux-Systems in drei relevante Teile 2: die Konfiguration eines Build-Hosts (Part II, Abschnitt 2 bis 4), das Bauen einer Cross-Toolchain (siehe Kasten Cross-Compilation erklärt) und nur temporär benötigter Tools (Part III, Abschnitt 5 bis 7) und erst daran anschließend das Erstellen des eigentlichen LFS-Systems (Part IV, Abschnitt 8 bis 11). Die Abschnitte 2 bis 11 beschreiben wir hier zusammengefasst und referenzieren sie so auch in den Shell-Kommentaren im nächsten Teil der Artikelserie.

Der Build-Prozess basiert auf dem Prinzip des Cross-Kompilierens: Ein Cross-Compiler produziert Code für andere Maschinentypen, dient also normalerweise dazu, Code für andere Systemarchitekturen zu übersetzen. Auf Ihrer Build-Maschine ist das theoretisch nicht nötig, da Sie mit hoher Wahrscheinlichkeit auf x86_64 für x86_64 bauten. LFS soll aber auf jeden Fall von hineingelinkter Software des Build-Hosts frei bleiben, was cross-kompilierte Software garantiert – daher der Aufwand.

Um die Vorgehensweise zu verdeutlichen, nimmt die LFS-Dokumentation drei Hosts mit jeweils unterschiedlichen Architekturen an. Host A besitzt einen Compiler für die eigene Architektur. Der Host ist langsam und seine Kapazitäten begrenzt. Host B besitzt keinen Compiler, ist jedoch schnell und gut ausgestattet. Er soll Software für Host C produzieren. Host C mit nochmals anderer Architektur besitzt ebenfalls keinen Compiler und ist klein und langsam.

Dieses Szenario erfordert zwei Cross-Compiler: Cross-Compiler A und Cross-Compiler B. Auf Host A baut der Compiler den Cross-Compiler A, der wiederum den Cross-Compiler B erstellt. Letzterer kommt auf Host B zum Einsatz, um den Compiler C sowie sämtliche Programme für Host C zu bauen. Auf Host B lässt sich diese Software aber nicht testen. Deswegen rebuildet und testet auf Host C der Compiler C sich anschließend selbst.

### Cross-Compilation erklärt

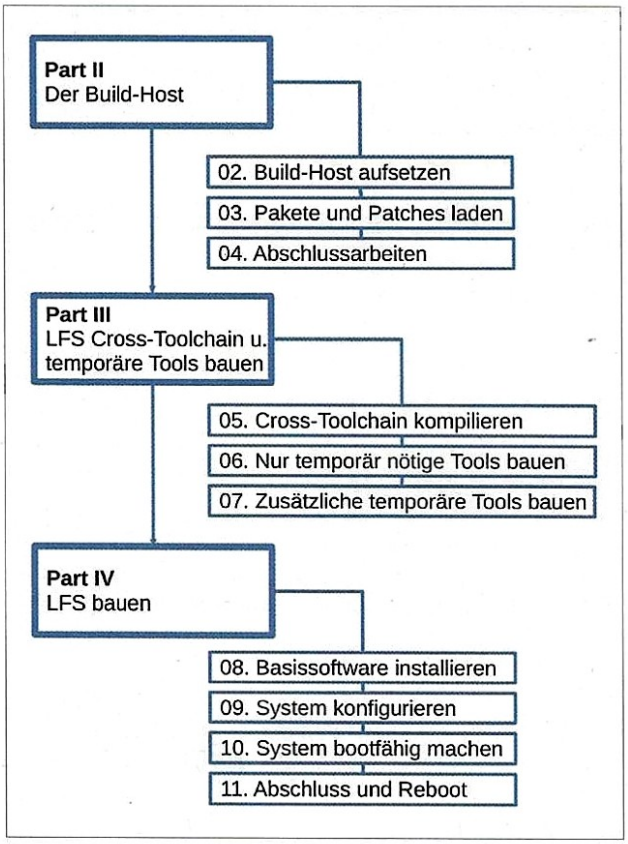
Um den Einfluss des Betriebssystems des Build-Hosts auf Linux from Scratch zu minimieren und davon unabhängig zu werden, erstellen Abschnitt 5 und 6 einen eigenen Cross-Compiler plus Tools, mit deren Hilfe Sie Linux from Scratch bauen. Sie setzen diese Werkzeuge nur für diesen Zweck in einer isolierten Chroot ein und entfernen sie anschließend.

## Auf einem Host

Damit für Linux from Scratch das Cross-Kompilieren auf demselben Host funktioniert, müssen Sie das bekannte Vendor-Triplet `x86_64-pc-linux-gnu` (das heutzutage immer vier Komponenten enthält) in `LFS_TGT` behutsam anpassen, um dem Compiler verschiedene Architekturen vorzugaukeln. Anschließend baut auf dem Build-Host der Compiler-Build-Host zunächst den Cross-Compiler-Build-Host, der wiederum den Compiler-LFS erstellt. In der LFS-Chroot auf dem Build-Host rebuildet und testet sich der Compiler-LFS dann selbst und steht danach zum Einsatz bereit.

Für LFS müssen Sie mithilfe des Cross-Compilers nicht nur den C-Compiler übersetzen, sondern auch die Glibc und die Libstdc++. Das Problem dabei: Der zu erstellende C-Compiler baut intern auf die Libgcc, die ihrerseits wiederum gegen die Glibc gelinkt sein muss – und die gibt es noch nicht.

Um diese zirkulären Abhängigkeiten zu umgehen, erstellen Sie zunächst eine Minimalversion des Cross-Compiler-Build-Hosts, die gerade dazu taugt, eine vollwertige Glibc sowie eine minimal funktionsfähige Libstdc++ zu übersetzen. Die Libstdc++ wird später in Chroot noch-




2 **Die drei Teile des Build-Prozesses von LFS.**

mals gebaut, diesmal komplett. Weitere Details nennt die LFS-Dokumentation.

## Beyond LFS

Anders als dem LFS-Buch folgt man der BLFS-Anleitung übrigens nicht linear, sondern sucht sich aus über 50 Abschnitten diejenigen aus, die das eigene Linux in die gewünschte Richtung bringen. Da BLFS von Security über Virtualisierung bis hin zu Desktop-Umgebungen so ziemlich alle Bereiche sowie die gängigsten Softwarepakete abhandelt, verwundert es nicht, dass es ungefähr drei Mal so umfangreich wie LFS ist.

## Nach Anleitung

Dieser Artikel fasst die LFS-Anleitung zusammen und erklärt zunächst die Prinzipien und die Vorgehensweise. In der nächsten Folge der Artikelserie finden sich dann die Kommandos, die Sie (soweit wie möglich) per Copy & Paste auf einem Build-Host unter CentOS 8 ausführen, um in einem Rutsch zu Ihrem ersten selbst gebauten Linux zu kommen. Die Anleitung verfrachtet `/home` darüber hinaus auf eine eigene Partition und bereitet die LFS-Disk auf ein autarkes Booten vor.

Das Bauen nimmt reichlich Zeit in Anspruch: Mit der in der nächsten Folge beschriebenen Test-VM unter KVM auf einem Rechner mit Core i7 10th Gen und NVMe-SSD dauert der Build auch mit abgeschalteten Tests mehrere Stunden.

Wenn Sie nach dem Build, dem Booten von LFS und den ersten Tests tiefer in die Materie eintauchen wollen, verweisen wir dazu auf die vorbildliche Originaldokumentation. Da linuxfromscratch.org so seine Mühen mit Geschwindigkeit und Erreichbarkeit hat, empfiehlt es sich, dafür einen der Mirror-Server zu nutzen.

## Vorzüge ...

Man kann sich freilich fragen, ob der Bau eines eigenen Linux-Systems lohnt, wo man doch aus einer breiten Masse an Distributionen auswählen kann. Das Team um den LFS-Projektleiter Gerard Beekmans hat darauf mehrere Antworten.

Zum einen möchte das Projekt zeigen, wie man eine Distribution baut, aus welchen Komponenten sie besteht, wie diese zusammenspielen und wie sie voneinander abhängen. Wer will, kann LFS auch als Ausgangspunkt für eigene Weiterentwicklungen nutzen.

Zum anderen erhält man mit LFS ein aktuelles und sehr kompaktes Linux-System, das nicht viel mehr umfasst als den Kernel und einige Werkzeuge. Die Doku berichtet von einem auf LFS laufenden Apache mit einer Gesamtgröße von 8 MByte und weniger, was besonders für Embedded Systeme von Interesse ist.

Wer selbst baut, bleibt zudem extrem flexibel. Mit LFS erhalten Sie quasi den Rohbau eines Hauses, das Sie ganz nach Geschmack individuell vom 1-Zimmer-Apartment bis zur Luxusvilla gestalten können. LFS lässt sich bei Bedarf komplett auditieren, und – vielleicht noch wichtiger: Sie haben das Management aller Security-Patches selbst in der Hand. Und zu guter Letzt: LFS ist einfach cool.

## ... und Nachteile

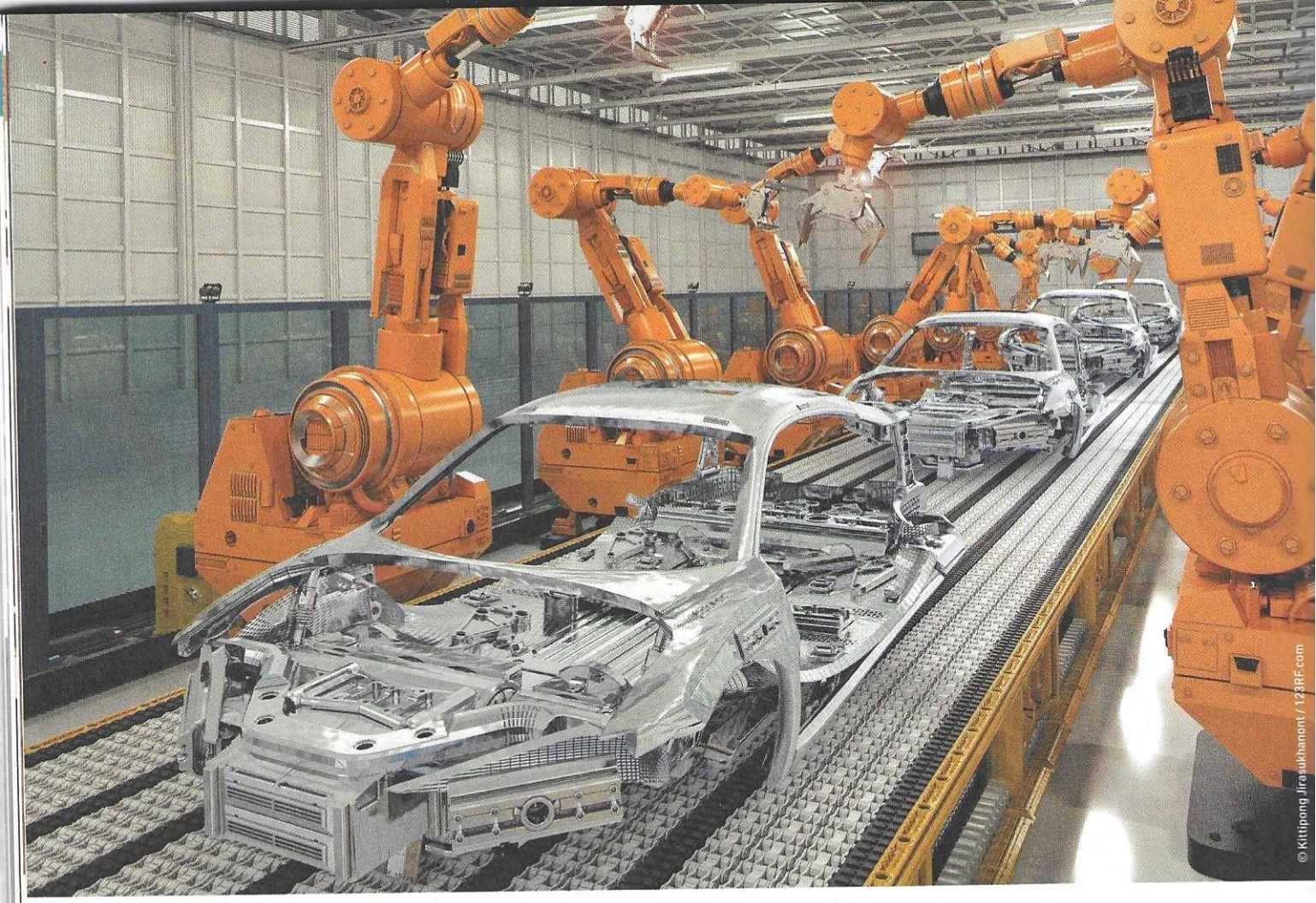
Um die Erwartungen etwas zu dämpfen: Im produktiven Betrieb lässt sich mit LFS nicht komfortabel arbeiten. Ein SSH-Daemon fehlt ebenso wie Sudo, Wget oder Parted. Kein LVM verwaltet das Dateisystem, der Netzwerk-Stack ist alles andere als komplett. BLFS liefert das zwar als Advanced Topic nach, einen Paketmanager gibt es aber im ganzen Projekt nicht.

Sie müssen Ihre eigene Distribution daher über Downloads und Kompilieren selbst aktuell halten und regelmäßig mit Updates und Sicherheitsaktualisierungen versorgen. Wer auf Slackware & Co. zu Hause ist, kann LFS jedoch auch in dieser Hinsicht erweitern, indem er einen Manager wie Pacman oder GNU Stow hinzufügt oder eine Strategie wie den Package-User-Ansatz verfolgt. Generell gilt bei LFS: Linux-Kenntnisse werden nicht vermittelt, sondern vorausgesetzt.

Von einer klassischen Linux-Distribution bleibt LFS ohnehin weit entfernt. Dazu fehlen ihm Merkmale wie eine aktive User Community, QA-geprüfte Pakete inklusive Errata und Patch-Management, ein umfangreiches Mirror-Netzwerk, zuständige Entwickler und Annehmlichkeiten wie ein Wiki, ein IRC-Chat, Mailing-Listen, Foren, Bugtracker oder eine FAQ.

Ob sich neben dem Lerneffekt auch ein praktischer Nutzen findet, hängt also von den eigenen Anforderungen ab. Mit LFS bekommen Sie jedenfalls ansatzweise ein Gefühl dafür, welche Investition an Zeit und Know-how im Linux-Kernel, in Open-Source-Software und in allen Linux-Distributionen steckt. (*jcb/jlu*)

**Weitere Infos und interessante Links**
www.lm-online.de/qr/46207




## Eine eigene Linux-Distribution zusammenbauen

# Endmontage

**Der erste Teil unserer kleinen Serie hat die Begriffe und Konzepte von LFS vorgestellt. In der zweiten Folge geht es nun darum, wie Sie in der Praxis ein Linux from Scratch aufsetzen.**

Markus Frei

**Ohne einen** perfekt funktionierenden Build-Host geht es nicht. Diese Anleitung setzt daher eine unbenutzte, virtuelle Maschine auf Basis eines voll gepatchten, aktuellen CentOS 8.3 Minimal Install mit BIOS-Boot voraus. Der C-Compiler GCC profitiert von einem schnellen Core.

Für den Build-Server genügen zwei Cores, 2 GByte RAM, ein SSD-basierter Hypervisor und eine Intel-e1000-NIC. Als Massenspeicher benötigen Sie eine Disk mit 10 GByte (VirtIO, `/dev/vda/`) für das System und eine mit 20 GByte (SATA oder SCSI, `/dev/sda/`) für LFS.

Vermeiden Sie VirtIO für die LFS-Disk und die Netzwerkkarte, da die Anleitung keinen VirtIO-Support in den Kernel einkompiliert. LFS freut sich über eine virtuelle SATA- oder SCSI-Disk, die es als `/dev/sda/` erkennt, sowie eine klassische e1000-Netzwerkkarte von Intel mit 1 Gbps Geschwindigkeit. Die Disk-Bezeichnung wird wichtig, wenn es darum geht, LFS bootfähig zu machen. Der Name der Netzwerkkarte (etwa `enp1s0`) spielt eine Rolle, falls Sie Predictable Network Interface Names verwenden wollen.

Nach der CentOS-Installation müssen Sie die für die ersten Build-Prozesse notwendigen Pakete installieren. Das benötigte `makeinfo` aus dem Paket *texinfo* erhalten Sie ab CentOS 8.3 aus dem PowerTools-Repository (Listing 1).

Die von CentOS installierten Versionen sollten Sie anschließend mithilfe eines kleinen Skripts prüfen: LFS v10.0 verlangt einen GCC ab Version 6.2 sowie GNU Make ab Version 4.0, die beispielsweise ein CentOS 7 nicht bereitstellt.

Passt hier alles, teilen Sie die zweite Disk des Build-Hosts in die Partitionen `/`, `swap` und `/home` auf, erstellen das Dateisystem mit Ext4 und definieren die für alle nachfolgenden Shell-Aufrufe essenzielle Umgebungsvariable $LFS,

**Der Autor**

Markus Frei beschäftigt sich seit über 25 Jahren mit Server-Anwendungen sowie dem Zusammenspiel von Infrastruktur und Softwareentwicklung. Er ist Mitinhaber der Linuxfabrik in Zürich, die Service und Support für Linux und Open-Source-Applikationen sowie Managed Hosting in einem Datacenter in der Schweiz anbietet.

die den Root-Mountpoint auflöst. Die Variable muss unter allen Umständen funktionieren. Weiter sollten Sie die Locale-Einstellungen in CentOS 8 noch erledigen und die zweite Disk einhängen.

Die Platte `/dev/sda/` machen Sie per BIOS/MBR bootfähig und statten sie mit diversen Partitionen aus. Um dem Grub-eigenen `core.img` genügend Platz einzuräumen, startet die erste Partition für / mit 10 GByte Größe erst ab 1 MiB. Danach legen Sie swap mit 2 GByte Größe an, der Rest der Disk bleibt für `/home`.

In diesem Setup benötigt `/boot` keine eigene Partition, sondern enthält als gewöhnliches Verzeichnis den Kernel sowie die Bootloader-Konfiguration. Eine eigene Partition brauchen Sie erst dann, wenn Sie ein Multiboot mit anderen Distributionen unterstützen wollen oder der Bootloader nicht mehr auf das Root-Dateisystem zugreifen kann. Letzteres kann an fehlenden Treibern, an Disk-Encryption, an einem Software-RAID oder LVM liegen. Alle Kommandos für die komplette Installation enthält das Open-Source-Admin-Handbuch der Linuxfabrik.

## Pakete und Patches laden

Das Kapitel 3 der LFS-Dokumentation definiert LFS in seiner späteren Ausstattung: Neben den Build-Tools laden Sie die gewünschten Quellen ausgewählter Softwarepakete herunter. Die Macher von LFS haben eine Vorauswahl getroffen, deren Versionsabhängigkeiten als Liste in einer Textdatei Sie herunterladen und ganz bequem Wget übergeben können (Listing 2).

Obendrein liefern die LFS-Entwickler eine Reihe von Patches, die Fehler beseitigen oder die Pakete an das zukünftige Zielsystem anpassen. Darüber hinaus erklären sie alle Optionen der jeweiligen `configure`-Aufrufe. Eine ebenfalls bereitgestellte Md5sum-Prüfsummen-Liste hilft dabei, die Checksummen der am Ende knapp 90 heruntergeladenen Pakete zu prüfen.

## Abschlussarbeiten

Im LFS-Kapitel 4 erstellt der Benutzer *root* die für den Build notwendigen Verzeichnisstrukturen `$LFS/{bin,etc,lib,lib64,sbin,tools,usr,var}` sowie den Benutzer *lfs* als Eigentümer der Build-Verzeichnisse. Anschließend gilt es, zum Benutzer *lfs* zu wechseln und dessen Shell-Environment zu optimieren.

Da es sich beim neuen Environment um eine Non-Login-Shell handelt, braucht es eine `.bashrc`-Datei, um die Shell-Umgebung zu erweitern und anzupassen, zum Beispiel mit Umgebungsvariablen wie $LFS, LC_ALL oder PATH. Das Verzeichnis `$LFS/tools/` enthält später den Cross-Compiler.

Haben Sie hier alles erledigt, dann steht ein Neustart des Build-Hosts an, um sicherzugehen, dass er ohne Altlasten daherkommt und funktioniert. Spätestens jetzt empfiehlt es sich, einen VM-Snapshot anzulegen.

## Cross-Toolchain kompilieren

In Kapitel 5 des LFS-Handbuchs bleibt man als Benutzer *lfs* angemeldet, für den die eigentliche Arbeit beginnt: das Bauen des Cross-Compilers und seiner Tools (Listing 3). Der Cross-Compiler samt Werkzeugen landet temporär in `$LFS/tools/`, die Bibliotheken installieren Sie bereits an ihren zukünftigen Orten.

Der Ablauf beim Bau der Softwarepakete bleibt bis zum Schluss gleich: Sie wechseln ins Verzeichnis `sources/`, wo Sie die Quelldateien mithilfe von `tar xf` auspacken. Dann wechseln Sie ins dabei erstellte neue Verzeichnis, wo Sie bei Bedarf ein `build/`-Verzeichnis erstellen.

Nun lässt sich das Paket kompilieren und installieren; anschließend löschen Sie das Quellcodeverzeichnis wieder. Das exerzieren Sie in einem ersten Durchlauf für Linker und Assembler, Cross-Compiler, die Linux-API-Headers, die Glibc und die Libstdc++ durch.

## Temporär benötigte Tools

Immer noch als Benutzer *lfs* kompilieren Sie mit der vorher erzeugten Cross-Toolchain im LFS-Kapitel 6 unter anderem Bash, Grep und den nativen LFS-GCC. Die Applikationen installieren Sie im bereits rudimentär erstellten LFS-Dateisystem, sie lassen sich aber noch nicht einsetzen; für bestimmte Aufgaben benötigen Sie also immer noch das Host-System. Immerhin werden die in LFS installierten Bibliotheken bereits zum Linken verwendet. Wirklich zum Einsatz kommen

**Listing 1: Benötigte Pakete installieren**

```
$ dnf -y install yum-utils
$ yum config-manager --set-enabled powertools
§ dnf -y install bison byacc bzip2 gcc-c++ patch perl python3 tar
texinfo wget
```

**Listing 2: Softwareauswahl herunterladen**

```
$ wget http://lfs.linux-sysadmin.com/lfs/downloads/stable-systemd/
wget-list
$ wget --input-file=wget-list --continue --directory-prefix=$LFS/
sources
```

**Listing 3: Test des cross-kompilierten GCC**

```
$ echo 'int main(){}' > dummy.c
$ $LFS_TGT-gcc dummy.c
### Zeigt die Information aus den Segment-Headern
### der Datei, falls vorhanden
$ readelf -l a.out | grep '/ld-linux'
### Sollte etwas zurückgeben wie:
### [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
```

die Applikationen aber erst im kommenden Abschnitt in der Chroot-Umgebung.

Temporär benötigen und kompilieren Sie im zweiten Durchlauf einen Makroprozessor, einen Linker und einen Assembler, den nativen GCC-Compiler, eine TUI-Bibliothek sowie die Bash. Daneben entstehen eine Reihe von Tools, darunter Awk, Chmod, Cp, Dd, Diff, File, Grep, Gzip, Make, Patch, Sed, Tar und Xz.

Bis zu dieser Stelle war der Benutzer *lfs* Eigentümer des LFS-Dateisystems; in Abschnitt 7 der LFS-Doku ändert sich das zu *root*. Anschließend erzeugen Sie auf der Platte die Devices `console` und `null`, die der Kernel beim Boot-Vorgang erwartet.

Durch die aufgelösten zirkulären Abhängigkeiten können Sie erstmals als Benutzer *root* in eine Chroot-Umgebung wechseln, die das Betriebssystem des Build-Hosts für weitere Aufgaben beinahe komplett aussperrt. Eine Ausnahme bildet der laufende Kernel: Damit die Chroot-Umgebung funktioniert, wird die Kommunikation mit ihm über das Virtual Kernel File System (VFS) konfiguriert.

Bei VFS handelt es sich um Dateisysteme wie etwa `tmpfs/`, die keinen Plattenplatz beanspruchen und komplett im Hauptspeicher liegen. Normalerweise mountet das System Geräte unterhalb von `/dev` als virtuelles Dateisystem und erstellt diese beim Erkennen oder beim ersten Zugriff während des Boot-Vorgangs via Udev. Im Augenblick fehlt Udev noch, also müssen Sie die benötigten Geräte manuell anlegen und per Bind-Mount einhängen. Stolperfalle: Falls Sie den LFS Build-Host später neu starten, müssen Sie auch die Mount-Befehle für die virtuellen Devices vor den weiteren Arbeiten erneut ausführen.

Beim ersten Wechsel in Chroot meldet sich ein ungewöhnlicher Bash-Prompt – es fehlt noch eine `/etc/passwd`. Nun kommt die endgültige Verzeichnisstruktur inklusive korrekt gesetzter Berechtigungen plus einiger Log-Dateien dran. Die Verzeichnisstruktur folgt dem Filesystem Hierarchy Standard (FHS) und legt unter anderem die Verzeichnisse `{boot,home,mnt,opt,srv}` an, lässt Unnötiges wie `/usr/local/games` aber weg.

Es fehlen noch Standardbenutzer und -gruppen. Da es hier keine einheitlichen Vorgaben gibt, orientiert sich LFS an Udev und anderen Distributionen. Sie kompilieren und installieren anschließend noch Software, um andere Programme zu testen. Hinzu kommen fehlende Bausteine der Toolchain: Zuerst erstellen Sie die Libstdc++, ein Paket für Internationalisierungs- und Lokalisierungssupport, einen Parser-Generator sowie die Programmiersprachen Perl und Python. Es folgen Tools für Info-Pages und kleinere Utilities.

Die derart komplettierte Toolchain läuft jetzt unabhängig vom Build-Host. Am Ende des Abschnitts kümmert sich die LFS-Doku noch um Stripping, Backups und Restore, worauf wir hier aber nicht genauer eingehen.

## Basissoftware installieren

Der nächste Abschnitt der LFS-Dokumentation gibt zunächst einen Ausblick auf das Thema Paketmanagement. Dabei kommt allerdings heraus, dass das Projekt keinen Paketmanager benutzt.

Anschließend ist Fleißarbeit angesagt. So kümmern Sie sich in Kapitel 8 per `configure`, `make` und `make install` darum, dass neben wichtigen Bibliotheken eine ganze Reihe an Applikationen auf der Platte landen. Dazu gehören neben der Bash auch Textprozessoren wie Awk, Grep und Sed sowie Packer und Archivprogramme wie Bzip, Gzip, Tar und Xz. Hinzu kommen der GCC, der Bootloader Grub, Man und Mandb, OpenSSL, Perl und Python, Vim sowie Udev.

In diesem Abschnitt ist das Übersetzen des GCC der mit Abstand der zeitintensivste Task im gesamten Projekt. Dieser Schritt beansprucht durch das Ausführen der Test-Suites fast die Hälfte der gesamten Kompilierungszeit. Sie erhalten hier jede Menge *FAILED*-Meldungen, um die Sie sich aber nicht zu kümmern brauchen,

**Listing 4: Klassische Interface-Namen nutzen**

```
# ln -s /dev/null /etc/systemd/network/99-default.link
```

**Tipps zum Troubleshooting**

| Symptom | Ursache | Lösung |
|---|---|---|
| Build-Vorgang meldet *libtool: warning: remember to run 'libtool --finish /usr/lib' (File-5.39)* | Durch Prefix-Parameter verursacht. | Nichts ausführen und Meldung ignorieren. |
| *The system has no more ptys. Ask your system administrator to create more.* | Möglicherweise haben Sie den Build-Host nach Schritt 7.3 neu gestartet und vergessen, die Device-Nodes vor dem Chroot erneut zu mounten. | Führen Sie Code-Abschnitt 7.3 und 7.4 nochmals aus. |
| LFS bleibt beim Boot einfach mit der Meldung *Grub* stehen. | Die LFS-Disk wurde mit BIOS/GPT eingerichtet. | Lagern Sie `/boot` auf eine eigene Partition aus. |
| In Grub erscheint beim Booten in LFS die Meldung *error: hd1 cannot get C/H/S values.* | Tritt oft bei manuell erstellter `/boot/grub/grub.cfg` auf. | Spezifizieren Sie die LFS-Disk als zweite Disk im System als erstes Boot-Medium, muss die Datei den Eintrag `set root=(hd0,1)` aufweisen, als zweites Boot-Medium `set root=(hd1,1)`. |
| *Kernel Panic – not syncing: VFS: Unable to mount root fs on unknown-block(0,0)* | Es fehlen höchstwahrscheinlich Treiber für das verwendete Medium. | Falls Sie VirtIO-Disks verwenden, fügen Sie die Treiber hinzu, oder steigen Sie auf SATA-Disks um. |

solange ein `grep` auf die von der Testsuite erzeugten Log-Dateien die erwarteten Ergebnisse liefert. Auch während der Compile-Vorgänge auftretende Warnungen in Bezug auf C-Syntax können Sie getrost ignorieren.

Die Unit-Tests erledigen verschiedene Variationen des Befehls `make check` oder `make test`, teils unter dem Benutzer *tester*. Überspringen Sie diese Unit-Tests, sparen Sie viel Zeit, riskieren aber auch das ein oder andere Problem.

Jetzt geht es ans Aufräumen: Nicht mehr benötigte Bibliotheken und Tools entfernen Sie ebenso wie den temporären Benutzer *tester*. Soll das erzeugte LFS nicht zum Programmieren dienen und benötigen Sie keine Debugging-Funktionen, können Sie durch Löschen der Debugging-Symbole 2 GByte Platz sparen (was der Code-Abschnitt berücksichtigt).

Zu guter Letzt verlassen Sie die Chroot-Umgebung, um sie gleich danach mit eingeschaltetem Bash-Path-Hashing (und von nun an immer mit dieser Konfiguration) erneut zu betreten.

## System konfigurieren

In Kapitel 9 konfigurieren Sie den Netzwerk-Stack: Hier lässt sich auf Wunsch die Verwendung der sogenannten Predictable Network Interface Device Names abschalten, was wir tun (Listing 4). Auf diese Weise erkennt LFS die erste Netzwerkkarte im System klassisch als `eth0`. Die IP-Adresse liefert DHCP, anschließend konfigurieren Sie die Namensauflösung und vergeben einen Host-Namen.

Kleinere Konfigurationsarbeiten an der Systemuhr, an der Linux-Konsole und an den Einstellungen der System-Locales runden das System ab (das entsprechende Linuxfabrik-Listing stellt ein schweizerdeutsches Tastaturlayout ein.) Bei Zweifeln hinsichtlich der passenden Locales suchen Sie sich in LFS aus `/usr/share/keymaps` und `/usr/share/consolefonts` das für Sie Passende heraus.

**Listing 5: Wichtige Kernel-Features**

```
General setup -->
  [ ] Auditing Support [CONFIG_AUDIT]
  [*] Control Group support [CONFIG_CGROUPS]
  [ ] Enable deprecated sysfs features to support old userspace tools
[CONFIG_SYSFS_DEPRECATED]
  [*] Configure standard kernel features (expert users) [CONFIG_EXPERT]
-->
    [*] open by fhandle syscalls [CONFIG_FHANDLE]
  Processor type and features  -->
    [*] Enable seccomp to safely compute untrusted bytecode [CONFIG_
SECCOMP]
  Firmware Drivers  -->
    [*] Export DMI identification via sysfs to userspace [CONFIG_DMIID]
  Networking support  -->
    Networking options  -->
      <*> The IPv6 protocol [CONFIG_IPV6]
  Device Drivers  -->
    Generic Driver Options  -->
      [ ] Support for uevent helper [CONFIG_UEVENT_HELPER]
      [*] Maintain a devtmpfs filesystem to mount at /dev [CONFIG_
DEVTMPFS]
  Firmware Loader -->
    [ ] Enable the firmware sysfs fallback mechanism [CONFIG_FW_LOADER_
USER_HELPER]
  File systems  -->
    [*] Inotify support for userspace [CONFIG_INOTIFY_USER]
    Pseudo filesystems  -->
      [*] Tmpfs POSIX Access Control Lists [CONFIG_TMPFS_POSIX_ACL]
```

## System bootfähig machen

Nun gilt es, LFS bootfähig einzurichten. Abschnitt 10 startet mit dem Anlegen einer `/etc/fstab`, die auf die LFS-Partitionen `sda1` (System) und `sda2` (Swap) verweist. Anschließend (und erst jetzt) bauen und installieren Sie den Linux-Kernel. Da spätere Erweiterungen wie in BLFS immer wieder Umkonfigurationen am Kernel erfordern, löschen Sie dessen Quellcode nach dem Build nicht. Mithilfe von Grub-install gelangt der Bootloader auf die Platte, den Sie anschließend noch per Grub-mkconfig konfigurieren.

LFS konfiguriert den Kernel im Code-Abschnitt per `make defconfig` ohne Interaktion mit sinnvollen Vorgabewerten. Bei Bedarf ändern Sie das und nutzen per `make menuconfig` die TUI-Variante, um die unzähligen Parameter des Kernels manuell zu setzen. Wichtig: Die Macher von Systemd empfehlen dringen die Verwendung von IPv6. Entsprechend sollten Sie die Kernel-Features aus Listing 4 ein- beziehungsweise abschalten. Auch zu Kapitel 10 finden sich passende Listings bei der Linuxfabrik.

## Abschluss und Reboot

Sie haben LFS nun vollständig installiert und dürfen sich stolz in den beiden Dateien `/etc/lsb-release` und `/etc/os-release` verewigen. Ein Logout aus der Chroot, das Setzen eines Root-Passworts, ein Aushängen von `$LFS` – und einem Reboot steht nichts mehr im Weg. Haben Sie die LFS-Disk als primäres Boot-Medium in der Build-VM ausgewählt, meldet sich kurz darauf ihr LFS. Listings gibt es bei der Linuxfabrik. *(jcb/jlu)*

**Weitere Infos und interessante Links**
www.lm-online.de/qr/46404