Open Source Press

Peer Heinlein

LPIC-1

Vorbereitung auf die Prüfung des
Linux Professional Institute

Open Source Press

Alle in diesem Buch enthaltenen Programme, Darstellungen und Informationen wurden nach bestem Wissen erstellt. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grunde sind die in dem vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor(en), Herausgeber, Übersetzer und Verlag übernehmen infolgedessen keine Verantwortung und werden keine daraus folgende Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht, auch nicht für die Verletzung von Patentrechten, die daraus resultieren können. Ebenso wenig übernehmen Autor(en) und Verlag die Gewähr dafür, dass die beschriebenen Verfahren usw. frei von Schutzrechten Dritter sind.
Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. werden ohne Gewährleistung der freien Verwendbarkeit benutzt und können auch ohne besondere Kennzeichnung eingetragene Marken oder Warenzeichen sein und als solche den gesetzlichen Bestimmungen unterliegen.
Dieses Werk ist urheberrechtlich geschützt. Alle Rechte, auch die der Übersetzung, des Nachdrucks und der Vervielfältigung des Buches – oder Teilen daraus – vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlags in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

© 2004 Open Source Press GmbH, München
Gesamtlektorat: Dr. Markus Wirtz
Satz: Open Source Press GmbH (LaTeX)
Umschlaggestaltung: Fritz Design GmbH, Erlangen
Gesamtherstellung: Kösel, Krugzell

ISBN 3-937514-02-3

http://www.opensourcepress.de

Open Source Press

10 **Sie möchten Ihr System auf verschiedene Partitionen aufteilen. Was wäre eine sinnvolle Aufteilung?**

[ ] a) Drei Partitionen: /, /etc und /usr

[ ] b) Vier Partitionen: /, /tmp, /sbin und /usr

[ ] c) Vier Partitionen: /, /boot, /home und /var

[ ] d) Drei Partitionen: /, /bin, /home

Bei bestimmten Systemen kommen Sie um eine Partitionierung nicht herum, z. B. wenn Sie eine alte Version des Bootmanagers LILO einsetzen und den Kernel auf einer **/boot**-Partition platzieren müssen, so dass er unterhalb der 1024-Zylinder-Grenze liegt.

Allerdings können Sie nicht jedes Verzeichnis auf eine eigene Partition legen. Das sollte man wissen, schließlich ist das auch der Grund für die Verzeichnisse **/bin**, **/lib** und **/sbin**. Haben Sie sich schon einmal gefragt, warum sie existieren, obwohl es doch auch **/usr/bin**, **/usr/sbin** und **/var/lib** gibt?

Um eine Partition zu mounten, benötigen wir beispielsweise die Programme **mount** oder auch **fsck**. Wie sollten wir aber **/usr** (und damit auch **/usr/bin**) einbinden, wenn das Programm **mount** unter **/usr/bin/mount** liegt?

Daher liegen wichtige Programme, die schon beim Systemstart oder auch in einem Rescue-Modus verfügbar sein müssen, separat. Und selbstredend können wir **/bin**, **/sbin** oder **/lib** deshalb *nicht* auf eine eigene Partition legen. Antworten b) und d) scheiden damit schon einmal aus. Aus denselben Gründen muss auch **/etc** immer auf der Root-Partition liegen: Wie sollte **/etc/fstab** ausgelesen werden, um herauszufinden, wo **/etc** zu mounten ist? Auch a) kann nicht funktionieren.

Es bleibt c) als einzig mögliche Variante – und ist auch sinnvoll: **/boot** kommt auf die erste Partition, um jeglichen Problemen mit einer 1024-Zylinder-Grenze aus dem Weg zu gehen (auch wenn neuere LILO-Versionen und Grub damit keine Probleme mehr haben sollten). Auch die **/home**-Verzeichnisse liegen auf einer eigenen Partition, so können wir verhindern, dass Nutzer durch ihre privaten Dateien unsere Root-Partition des Servers volllaufen lassen. Zudem können wir dann von **mount**-Optionen wie **nodev**, **noexec** oder **nosuid** Gebrauch machen, um das System zu härten.

Auch **/var** kann man aus den genannten Sicherheitsüberlegungen auf eine separate Partition legen, muss man aber nicht. Im Einzelfall entscheidet das Aufwand-Nutzen-Verhältnis. Wer noch weiter gehen will, könnte übrigens auch **/usr** eine

Open Source Press

eigene Partition geben (zum Beispiel **readonly**), ebenso wie **/tmp**, damit wiederum normale Nutzer nicht die Root-Ebene des Servers fluten. Weitere Aufteilung ist denkbar, und es gibt sicherlich Gründe, die dafür sprechen, allerdings muss man sich natürlich stets die Frage nach dem Verhältnis zwischen Aufwand, Folgeproblemen und Nutzen stellen.

- **Sie sollten genau wissen, welche Verzeichnisse auf separate Partionen gelegt werden können und – wichtiger – welche Verzeichnisse auf der Root-Partition liegen müssen!**

Open Source Press

27 **Sie haben nachfolgendes Problem. Was wird wahrscheinlich die Ursache sein?**

```
linux:~ #  ls -l programm
-rwxr-x---    1 root     root        24680 Jul 15  2002 programm
linux:~ #  programm
bash: programm: command not found
linux:~ #
```

[ ] a) Der Nutzer hat keine Ausführungsrechte für diese Datei.

[ ] b) Diese Datei enthält keinen gültigen Programmcode.

[ ] c) Der Pfad „." ist nicht in der Umgebungsvariable **$PATH** enthalten. Mit dem Aufruf **./programm** würde es funktionieren.

[ ] d) Dieser Nutzer darf generell keine Programme starten.

Selbstverständlich hat der Nutzer Ausführungsrechte für diese Datei: An der Raute (**#**) ist zu erkennen, dass wir Superuser sind und als Dateibesitzer auch über Ausführungsrechte (x-Bit, Modus 750) verfügen. Auch Antwort d) ist Unsinn, denn es gibt keine Möglichkeit, einzelnen Nutzern das Ausführen von Programmen generell zu verbieten. Man kann allenfalls Partitionen so mounten, dass von diesen kein Programm gestartet werden kann, aber das ist ein anderes (LPIC-relevantes) Thema.

Würde die Datei keinen gültigen Programmcode enthalten, sähe die Fehlermeldung anders aus (**cannot execute binary file**).

Bleibt Antwort c) – und für jene, die damit nichts anfangen können, einige kurze Erläuterungen dazu; wer noch über frühe DOS-Kenntnisse verfügt, dem wird das Folgende vielleicht bekannt vorkommen.

Um Programme ohne Pfadangabe starten zu können, muss das System die Aufgabe übernehmen, Verzeichnisse nach dem gewünschten Programm zu durchsuchen. Um diesen Vorgang zu beschleunigen, wird die Suche auf die Verzeichnisse beschränkt, in denen Programme üblicherweise abgelegt sind – ein wichtiges Argument dafür, alle Programme in ein Verzeichnis **/usr/bin** zu packen, statt sie, wie bei Windows, über unzählige Ordner in **C:/Programme/xyz** zu verstreuen.

In welchen Verzeichnissen das System nach einem aufgerufenen Programm suchen soll, ist in der Umgebungsvariablen **PATH** gespeichert:

Open Source Press

```
user@linux:~> echo $PATH
/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:/usr/games:/opt/gnome2/bin:/
opt/gnome/bin:/opt/kde3/bin:/opt/kde2/bin:/usr/lib/java/jre/bin:/opt/gno
me/bin:.
```

Wir wir sehen, ist hier zuletzt noch der Punkt („.") als Pfad gespeichert – und repräsentiert bekanntlich das jeweils aktuelle Verzeichnis, in dem wir uns befinden. Der oben angezeigte Pfad gilt für einen normalen Benutzer; für ihn wird der Programmname auch im aktuellen Verzeichnis gesucht und die Anwendung gestartet.

Bei root sieht das anders aus:

```
linux:~ # echo $PATH
/usr/sbin:/bin:/usr/bin:/sbin:/usr/X11R6/bin
```

Hier fehlt der „." ebenso wie viele andere Verzeichnisse. Das heißt: Für root wird ausschließlich in den hier genannten Verzeichnissen gesucht. Liegt ein Programm (wie in der Frage angenommen) im aktuellen Verzeichnis, würde das System dieses einfach nicht finden.

Um das Programm aber dennoch starten zu können, müssen wir dem System mitteilen, wo genau es liegt, und zwar entweder über eine vollständige Pfadangabe oder durch ein simples vorangestelltes „./", das ja das aktuelle Verzeichnis bezeichnet. Dadurch weiß Linux nun genau, welches Programm wir meinen.

```
linux:~ # pwd
/root
linux:~ # ls -l programm
-rwxr-x---    1 root     root        24680 Jul 15  2002 programm
linux:~ # /root/programm

   Hallo Welt!

linux:~ # ./programm

   Hallo Welt!

linux:~ #
```

Der Grund für diesen Aufwand? Sicherheit! Es soll verhindert werden, dass gerade root fremde/manipulierte Programme untergeschoben werden, die die Sicherheit des Systems gefährden. Nehmen wir an, in /tmp (immerhin beschreibbar für alle!) legt ein Angreifer ein Programm namens jeo ab, vielleicht ein kleines Skript, das einen zweiten Superuser-Account mit einem bestimmten Passwort einrichtet.

Nun muss der Angreifer nur noch warten, bis sich root einmal im Verzeichnis /tmp befindet, sich vielleicht vertippt und statt des Texteditors joe versehentlich das Angreifer-Skript jeo startet. Wenn der Angreifer klug ist, lädt jeo anschließend

Open Source Press

/usr/bin/joe nach und löscht sich dann selbst, so dass root nicht einmal merkt, dass er sich vertippt hat.

Über die Frage, wie lange ein Angreifer denn hier auf einen solchen Vertipper warten müsste, kann man nun trefflich streiten, und sicher gibt es bessere „Tippfehlernamen", aber das Prinzip ist deutlich geworden: Man kann root hier ein „Kuckucksei" unterschieben, das er mit root-Rechten ausführt und über das man mit root-Rechten im System agieren kann.

Also limitiert man gerade bei root die zu durchsuchenden Programmpfade auf ein Minimum und schließt nur Pfade ein, die ausschließlich für root beschreibbar sind – zum Beispiel /usr/bin und Co. Aus Bequemlichkeit auch „." in die PATH-Umgebungsvariable aufzunehmen verbietet sich also kategorisch!

Für normale Benutzer hingegen ist der „." oft eingetragen. Das Risiko ist dort nicht sehr groß, da Kuckuckseier allenfalls die Benutzer-ID des jeweiligen Users erlangen könnten. Zudem würden wohl die wenigsten Benutzer verstehen, warum sie ein Programm mit **./programm** starten müssen, obwohl es doch „direkt vor ihnen liegt".

- **Wissen Sie, wo die** PATH**-Variable gesetzt wird? Schauen Sie in** /etc/profile **nach.**
- **Wie kann ein Benutzer seine** PATH**-Variable anpassen und/oder ändern? Schauen Sie in** ~/.profile **in den User-Homeverzeichnissen!**

Open Source Press

99

Ergänzen Sie die Angaben zu den üblichen Subnetzen.

255.255.255.0	**Kurzschreibweise:** /24	**Anzahl der Hosts:** 254
255.255.255. ______	**Kurzschreibweise:** ______	**Anzahl der Hosts:** ______
255.255.255. ______	**Kurzschreibweise:** ______	**Anzahl der Hosts:** ______
255.255.255. ______	**Kurzschreibweise:** ______	**Anzahl der Hosts:** ______
255.255.255. ______	**Kurzschreibweise:** ______	**Anzahl der Hosts:** ______
255.255.255. ______	**Kurzschreibweise:** ______	**Anzahl der Hosts:** ______
255.255.255. ______	**Kurzschreibweise:** ______	**Anzahl der Hosts:** ______
255.255.255. ______	**Kurzschreibweise:** ______	**Anzahl der Hosts:** ______

Subnetzmasken! – Für die meisten Einsteiger ein Angstthema. Glücklicherweise müssen Sie diese nicht binär berechnen (auch wenn es gar nicht so schwierig ist) oder gar detailliert erklären können. Sie sollten allerdings deren Bedeutung kennen und auch genau wissen, welche Subnetzmasken vorkommen können, wenn Sie ein Class-C-Netzwerk aufteilen. Insofern kommen wir an ein wenig Netzwerk-Theorie an dieser Stelle nicht vorbei.

Wie wohl leicht einzusehen ist, rechnen Computer mit IP-Nummern in Binärform und nicht im Dezimalsystem. Schauen wir uns also drei IP-Nummern in ihrer binären Schreibweise an:

192.168.0.80 entspricht 11000000.10101000.00000000.01010000
192.168.0.130 entspricht 11000000.10101000.00000000.10000010
192.168.5.130 entspricht 11000000.10101000.00000101.10000010

Die IP-Nummern bestehen also aus $4 \times 8 = 32$ Bits.

Wenn ein Rechner IP-Pakete verschickt, muss er zunächst feststellen, ob sich die Zieladresse im selben LAN befindet wie er selbst (dann muss er das Paket „einfach so" aussenden) oder ob sie sich außerhalb dieses Netzes befindet, er die Pakete also über ein Gateway weiterrouten muss (dann müssen die Pakete eben zum Gateway in diesem LAN geschickt werden).

Hier kommen nun die Subnetzmasken ins Spiel: Sie geben an, wie viele Bits der IP-Nummer zum so genannten „Netzwerkanteil" und wie viele zum so genannten „Hostanteil" gehören. Die Subnetzmaske **255.255.255.0** steht beispielsweise für binär **11111111.11111111.11111111.0000000**, wobei alle 1-Bits den Netzwerkanteil repräsentieren – hier also die ersten 24 Bits. Die hier frei bleibenden letzten 8 Bits geben den Hostanteil an.

Denn: Unterscheiden sich die Bits der Zieladresse gegenüber dem sendenden Host allein im Hostanteil, bzw. ist der Netzwerkanteil identisch, so liegt die Ziel-IP innerhalb *eines* LAN. Unterscheiden sich hingegen die Netzwerkanteile, muss das Paket über ein Gateway geroutet werden.

Open Source Press

Schauen Sie sich nochmal die drei oben angegebenen Binärmuster an: IP-Nummer eins und zwei unterscheiden sich allein in den letzten acht Bits. Bei der hier angenommenen Netzwerkmaske heißt das: Sie haben den gleichen Netzwerkanteil, sie liegen im selben Subnetz.

Bei IP-Nummer drei ist das anders: Sie unterscheidet sich bereits im 3. Block von den anderen beiden IP-Nummern, und damit bereits im Netzwerkanteil. Diese IP-Nummer liegt damit in einem anderen Subnetz. Wollen Host 1 und Host 2 IP-Pakete an Host 3 senden, benötigen Sie einen Host, der die Daten weiterleitet.

Nun wird es ein wenig komplizierter:

Wir beschließen, das Netz zu teilen, und wählen Subnetzmaske **255.255.255.128**, binär geschrieben **11111111.11111111.11111111.10000000**. Der Netzwerkanteil ist jetzt um ein Bit länger (die ersten 25 Bits) als im obigen Beispiel, der Hostanteil entsprechend ein Bit kürzer (die letzten 7 Bits). Für unseren Host 1 bedeutet das: Nur Hosts, die die ersten 25 Bits mit ihm gemein haben, liegen in seinem Netz.

Und wir sehen schon: Host 1 und Host 2 befinden sich plötzlich nicht mehr in einem Subnetz:

11111111.11111111.11111111.10000000 – Subnetzmaske **/25**
11000000.10101000.00000000.01010000 – Host 1
11000000.10101000.00000000.10000010 – Host 2

Noch in dem durch die 1-Bits der Subnetzmaske markierten Netzwerkanteil gibt es Unterschiede, so dass der Weg über das Gateway notwendig wird.

Nun können Sie fortfahren, indem Sie die Zahl der Bits des Netzwerkanteils weiter erhöhen; der Hostanteil wird entsprechend kleiner. Der Netzwerkanteil kann 27, 28, 29 oder mehr Bits lang sein, und wenn Sie die gesetzten Bits in Dezimalschreibweise umrechnen, kommen Sie zu folgender Tabelle. Die Kurzschreibweise der letzten Spalte gibt einfach nur an, wie viele Bits gesetzt sind:

11111111.11111111.11111111.00000000 – 255.255.255.000 – /24
11111111.11111111.11111111.10000000 – 255.255.255.128 – /25
11111111.11111111.11111111.11000000 – 255.255.255.192 – /26
11111111.11111111.11111111.11100000 – 255.255.255.224 – /27
11111111.11111111.11111111.11110000 – 255.255.255.240 – /28
11111111.11111111.11111111.11111000 – 255.255.255.248 – /29
11111111.11111111.11111111.11111100 – 255.255.255.252 – /30
11111111.11111111.11111111.11111110 – 255.255.255.254 – /31
11111111.11111111.11111111.11111111 – 255.255.255.255 – /32

Bis hierhin haben wir also schon einmal Dezimal- und Kurzschreibweise der Subnetzmasken zusammengetragen. Bleibt der zweite Teil der Frage, wie viele Hosts pro Subnetz vorhanden sein können.

Dazu muss man wissen: In jedem Subnetz gehen zwei IP-Nummern „verloren": Die eine wird als Netzwerk-Basisadresse benötigt (in der Regel die unterste IP-Nummer

Open Source Press

des jeweiligen Subnetzes), die andere für die Broadcast-Adresse, normalerweise die höchste IP-Nummer eines jeden Subnetzes.

In einem Class-C-Netz (alias **/24**) haben wir also 8 Bit Netzwerkanteil, alias 256 mögliche IP-Nummern, d. h. 254 nutzbare IPs für Hosts + 1 Netzwerkbasis-IP + 1 Broadcast-IP.

In einem halben Class-C-Netz sieht es etwas anders aus: Bei einer **/25**-Bit-Netzwerkmaske stehen 7 Bit Hostanteil zur Verfügung, also 128 IP-Nummern. Abzüglich Netzwerkbasis- und Broadcast-IP bleiben 126 nutzbare IPs für Hosts.

Und zu guter Letzt: Vierteln wir das Netz (Netzwerkanteil **/26**), haben wir 6 Bit Hostanteil, alias 64 mögliche IP-Nummern, also 62 nutzbare IP-Nummern für Hosts.

Allerdings lauert eine Falle: Nehmen wir Subnetzmaske **/31** (alias **255.255.255.254**), bleibt uns 1 Bit Hostanteil, was zwei (!) möglichen IP-Nummern entspricht – abzüglich Netzwerk-Basisadresse und Broadcast ergäbe das 0 nutzbare IP-Nummern. Ein Subnetz **/31** gibt es also nur theoretisch, nicht praktisch.

Etwas anders liegt der Fall bei **/32**, alias **255.255.255.255**. Hier haben wir eigentlich gar keinen Hostanteil und damit genau genommen auch kein Subnetz. Man gibt mit dieser Maske jedoch eine „single host route" an, also z. B. eine PPP-Verbindung zu einem einzelnen Host, der sich irgendwo über Modem oder ISDN eingewählt hat.

Tragen wir noch einmal die Musterantwort zusammen: Da die **/31** wegfällt, ergeben sich acht mögliche Subnetzmasken für ein Class-C-Netz:

255.255.255.0 – Kurz: **/24** – Hostanzahl: 254
255.255.255.128 – Kurz: **/25** – Hostanzahl: 126
255.255.255.192 – Kurz: **/26** – Hostanzahl: 62
255.255.255.224 – Kurz: **/27** – Hostanzahl: 30
255.255.255.240 – Kurz: **/28** – Hostanzahl: 14
255.255.255.248 – Kurz: **/29** – Hostanzahl: 6
255.255.255.252 – Kurz: **/30** – Hostanzahl: 2
255.255.255.255 – Kurz: **/32** – Hostanzahl: 1

Open Source Press