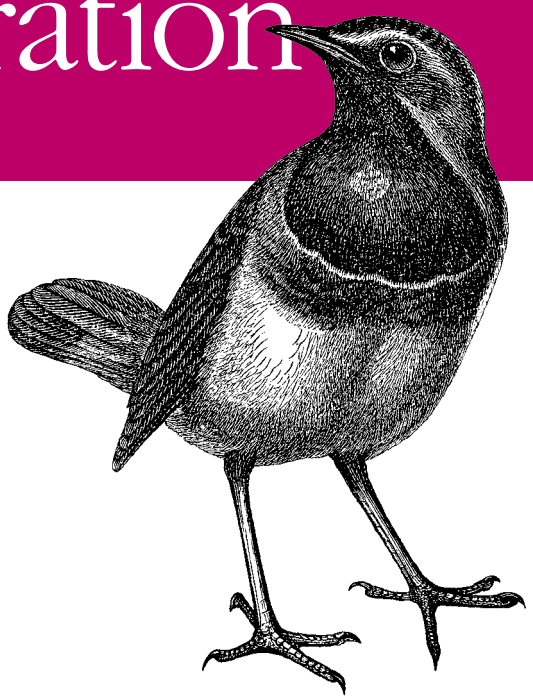
Die fortschrittlichste Open-Source-Datenbank

3. Auflage

Behandelt PostgreSQL 9.2

PostgreSQL Administration



Peter Eisentraut & Bernd Helmle

O'REILLY®

3. AUFLAGE

PostgreSQL-Administration

Peter Eisentraut & Bernd Helmle

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

Kommentare und Fragen können Sie gerne an uns richten:
O'Reilly Verlag GmbH & Co. KG
Balthasarstr. 81
50670 Köln
E-Mail: kommentar@oreilly.de

3. Auflage 2013

Bibliografische Information Der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.de* abrufbar.

Lektorat: Volker Bombien, Köln
Korrektorat: Tanja Feder, Bonn
Produktion: Karin Driesen, Köln
Umschlaggestaltung: Michael Oreal, Köln
Satz: Reemers Publishing Services GmbH, Krefeld, www.reemers.de
Belichtung, Druck und buchbinderische Verarbeitung:
Druckerei Kösel, Krugzell, www.koeselbuch.de

ISBN: 978-3-86899-361-5

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

Inhalt

Vorwort

Das fortschrittlichste Open-Source-Datenbankmanagementsystem der Welt, so lautet weithin unangefochten seit über einem Jahrzehnt der Untertitel zu PostgreSQL. Mittlerweile ist es millionenfach im Einsatz, als Teil der kritischen öffentlichen Infrastruktur des Internets und der Gesellschaft und als zentrales Element in der Zukunft der Datenbankwelt.

Doch jeder kann Teil dieser Erfolgsgeschichte sein. PostgreSQL ist Open Source, es ist kostenlos verfügbar und wird von einer großen, offenen Community von Anwendern und Entwicklern vorangetrieben. Dieses Buch möchte seinen Teil dazu beitragen, dieses Software-Produkt allen interessierten Anwendern zugänglich zu machen.

Zielgruppe

Dieses Buch richtet sich primär an Administratoren von PostgreSQL-Datenbanksystemen. Es soll dabei helfen, PostgreSQL-Datenbanksysteme erfolgreich, stabil und performant zu betreiben. Es wird davon ausgegangen, dass der Leser entweder schon Umgang mit PostgreSQL hatte oder über Erfahrungen mit der Administration von anderen Datenbanksystemen verfügt. Vertrautheit mit SQL und Unix-Shells wird von Vorteil sein.

Die Entwicklung von Datenbankanwendungen wird in diesem Buch nicht behandelt und fortgeschrittene Programmierkenntnisse sind auch nicht vonnöten. Allerdings wird im Zuge der Administration eines Datenbanksystems oft die Kommunikation zwischen Administration und Entwicklung notwendig sein. Daher sind Kenntnisse in Sachen Anwendungsentwicklung generell von Vorteil.

Dieses Buch soll die PostgreSQL-Dokumentation um praktische Erfahrungswerte ergänzen. Es kann aber dem PostgreSQL-Administrator im Alltag auch schon für sich genommen als eigenständige Referenz nützlich sein, wobei dieses Buch aber niemals den Anspruch haben kann, den gesamten Umfang des PostgreSQL-Systems abzudecken.

Struktur dieses Buchs

Dieses Buch besteht aus zehn Kapiteln. Die Kapitel sind so ausgelegt, dass sie der Reihenfolge entsprechen, in der man sich mit den entsprechenden Themen im Laufe des Lebens eines Datenbanksystems ungefähr befassen wird. Wer also schnell »von 0 auf 100« kommen möchte, kann dieses Buch von vorne bis hinten durchlesen. Jedes Kapitel soll aber auch für sich stehen und Anwendern, die schon einen gewissen Kenntnis- und Erfahrungsstand haben, die Möglichkeit geben, sich in bestimmten Themenbereichen weiterzubilden. Auf diese Weise kann das Buch außerdem als tägliche Referenz verwendet werden. Es ist also auch möglich – und in vielen Fällen wohl auch empfehlenswert –, das Buch in einer selbst gewählten Reihenfolge durchzuarbeiten.

Kapitel 1
: Das Leben jeder Software beginnt mit der Installation.

Kapitel 2
: Hier werden die Einstellungen der Konfigurationsparameter im PostgreSQL-Server erläutert.

Kapitel 3
: Hier werden wiederkehrende Aufgaben beschrieben, die zur Wartung eines PostgreSQL-Servers notwendig sind.

Kapitel 4
: Teil der Wartungsaufgaben ist die Datensicherung, der ein eigenes Kapitel gewidmet ist.

Kapitel 5
: Hier werden Verfahren vorgestellt, mit denen Zustand und Verhalten eines PostgreSQL-Servers überwacht und analysiert werden können.

Kapitel 6
: Hier wird beschrieben, was man tun kann, wenn irgendetwas beschädigt worden zu sein scheint.

Kapitel 7
: Die Absicherung der Daten vor unberechtigtem Zugriff ist Thema dieses Kapitels.

Kapitel 8
: Hier wird erläutert, wie man SQL-Befehle schneller machen kann.

Kapitel 9
: Hier werden verschiedene Lösungen vorgestellt, um PostgreSQL-Datenbanken zu replizieren und zu clustern, um bessere Verfügbarkeit oder bessere Leistung zu erzielen.

Kapitel 10
: Enthält Hinweise zu Auswahl und Einrichtung von Hardware für PostgreSQL-Systeme. Von der Logik her würde die Hardware-Auswahl wohl noch vor der Installation stattfinden, aber es ist auch sinnvoll, sich diesen Fragen erst dann zu widmen, wenn man die Interna eines PostgreSQL-Systems gut verstanden hat.

In diesem Buch behandelte Versionen

Dieses Buch behandelt hauptsächlich PostgreSQL 9.2 und 9.1. Die aktuellste PostgreSQL-Version zum Zeitpunkt der Drucklegung ist 9.2.2, aber alle Releases der Reihe 9.2 unterscheiden sich – wenn überhaupt – nur geringfügig bezüglich der Benutzerschnittstellen und der Verhaltensweise.

Wo es bedeutende Unterschiede gibt, wird auch kurz auf Version 9.0 und ältere Versionen eingegangen. Aber gerade bei der Datenbankadministration hat sich sowohl hinsichtlich der Möglichkeiten als auch bezüglich der Anforderungen über die letzten Jahre hinweg Hauptversionen sehr viel getan, weswegen ältere Versionen erstens aus Platzgründen nur kurz behandelt werden können und zweitens weniger zu empfehlen sind, wenn man die maximalen Möglichkeiten bei der Datenbankadministration ausnutzen möchte.

An den Stellen, an denen es um Betriebssystemeinstellungen und die Einbindung externer Programmpakete geht, haben wir natürlich eine Auswahl treffen müssen, die sich letztlich daran orientiert, womit wir selbst arbeiten und was wir weiterempfehlen wollen. Die allermeisten Teile dieses Buches gelten aber völlig unabhängig von der Wahl des Betriebssystems oder der Zusatzwerkzeuge.

Neues in der dritten Auflage

In der dritten Auflage wurden alle Kapitel dieses Buches überarbeitet und an neuere Software-Versionen und Hardware-Entwicklungen angepasst sowie um zusätzliche Erfahrungswerte erweitert. Wichtige Neuerungen gibt es insbesondere in den Bereichen Replikation, Überwachung sowie Performance-Optimierung.

Typografische Konventionen

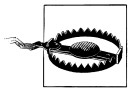
In diesem Buch werden die folgenden typografischen Konventionen verwendet:

Kursivschrift
: Wird für die Namen von Programmen, Befehlen, Dateien, Verzeichnissen sowie für URLs verwendet.

`Nichtproportionalschrift`
: Wird für SQL-Anweisungen sowie Codeteile, Codebeispiele und Systemausgaben verwendet.

`Nichtproportionalschrift kursiv`
: Wird in Codebeispielen für Platzhalter verwendet, für die eigene Werte eingesetzt werden müssen.

Dieses Symbol kennzeichnet einen Hinweis, der eine nützliche Anmerkung zum nebenstehenden Text enthält.



Dieses Symbol kennzeichnet eine Warnung, die sich auf den nebenstehenden Text bezieht.

Danksagungen

Treibende Kraft bei diesem Buch war wieder einmal unser Lektor Volker Bombien. Seine Geduld und Ausdauer waren unbezahlbar.

Wir danken dem Fachlektor Sven Riedel und allen Kollegen, Probelesern und Vorabkritikern für ihre Hinweise.

Die credativ GmbH hat es uns ermöglicht, unser Hobby zum Beruf zu machen. Unseren Erfahrungsschatz, den wir in diesem Buch teilen möchten, konnten wir nur so aufbauen.

Wir grüßen das Linuxhotel und alle Schulungsteilnehmer, die gewissermaßen unsere Versuchskaninchen und Betatester beim Aufbau dieses Materials waren.

KAPITEL 1

Installation

In diesem Kapitel wird die Installation der PostgreSQL-Software beschrieben. Geübten und erfahrenen Administratoren mit den geeigneten Werkzeugen geht die Installation eines PostgreSQL-Servers leicht von der Hand. Ein Großteil des Vorgangs ist automatisiert.

Softwareinstallation

Der erste Schritt, um ein PostgreSQL-System einzurichten, besteht in der Installation der Software. In den meisten Situationen bieten sich dem Administrator zwei Varianten, wie diese durchgeführt werden kann: Entweder man baut die Software aus dem Quellcode selbst, oder man installiert ein vorgefertigtes Paket.

Das PostgreSQL-Projekt (also die Entwickler der Software) veröffentlicht zunächst nur den Quellcode. Die Pakete werden daraufhin von oder in Zusammenarbeit mit den Anbietern der verschiedenen Betriebssysteme zusammengestellt, um die Installation der Software auf dem jeweiligen System zu vereinfachen und sicherzustellen, dass die Software mit dem System gut zusammenarbeitet.

Was wir hier verkürzt als »Paket« bezeichnen, heißt auf verschiedenen Systemen unterschiedlich: Auf vielen Linux-Systemen wird es als RPM oder Debian-Paket bezeichnet, auf BSD-Systemen entweder als Port oder als Package, auf Mac OS X finden sich MacPorts und Homebrew, und für Windows gibt es einen Installer.

Zu der Frage, welche Installationsart zu wählen ist, gibt es viele Meinungen. Ausschlaggebend sind dabei folgende Aspekte:

- Ist die gewünschte Version verfügbar?
- Ist absehbar, dass zukünftige Versionen rechtzeitig verfügbar sein werden?
- Wie ist die Qualität der Paketierung?
- Bestehen Sonderwünsche, die die Paketierung nicht erfüllt?
- Womit ist der Administrator am besten vertraut?

Wer hingegen den Quellcode selbst bauen und installieren will, sieht sich mit vielen zusätzlichen Aufgaben konfrontiert:

- Compiler pools müssen bereitgestellt werden.
- Abhängigkeiten müssen selbst verwaltet werden.
- Benutzer und Dateisystemrechte müssen eingestellt werden.
- Startskripten müssen geschrieben und konfiguriert werden.
- Logging, Wartung, Datensicherung und Ähnliches müssen erledigt werden.

Für diejenigen Betriebssysteme, die mit PostgreSQL am häufigsten eingesetzt werden, sieht die Situation so aus, dass sich der Einsatz einer paketierten Variante auf jeden Fall lohnt.

Versionierung

Bevor man die Einrichtung eines PostgreSQL-Systems angeht, sollte man sich klarmachen, welche Versionen es gibt und welche von ihnen man verwenden möchte.

PostgreSQL verwendet eine dreiteilige Versionsnummer, zum Beispiel 8.4.10 oder 9.2.0. Die erste Ziffer der Versionsnummer ändert sich nur sehr selten, nämlich dann, wenn eine neue »Ära« im Projekt anbricht. Die zweite Ziffer ändert sich, wenn ein neues großes Release mit neuen Features ausgeliefert wird. Die dritte Ziffer der Versionsnummer ändert sich mit jedem kleinen Release, das nur kritische Fehler berichtigt.

Dieses Versionsnummernschema ist aus technischer Sicht etwas unpassend, denn die erste und die zweite Zahl bilden im Prinzip eine Einheit. Aus Sicht der Entwickler gibt es einen großen Releasezweig »9.2« mit den Unterversionen 9.2.0, 9.2.1 und so weiter, je nachdem, wie oft Fehlerkorrekturen in dem Zweig notwendig werden. Insofern sind etwa 8.4 oder 9.2 als »Hauptversionsnummern« (*major version*) anzusehen, denn immer wenn sich die zweite Ziffer der Versionsnummer ändert (und eventuell auch die erste), enthält die neue Version neue Features. Wir verwenden den Begriff »Hauptversion« in diesem Buch in diesem Sinn. Die einzelne erste Ziffer (8 oder 9) hat eher repräsentativen Charakter, aber keine technischen Auswirkungen – die Unterschiede zwischen 8.4 und 9.0 sind vom Prinzip her nicht anders als die zwischen 8.3 und 8.4 oder die zwischen 9.0, 9.1 und 9.2. Es ist daher in jedem Fall falsch und sinnlos, etwa von einer Version »PostgreSQL 9« zu sprechen. (Das ist ähnlich wie beim Linux-Kernel. Auch da ist die Bezeichnung »Linux 2« oder »Linux 3« unsinnig.)

In der Praxis wird etwa einmal im Jahr eine neue Hauptversion herausgebracht. Es gibt dazu, wie bei Open Source-Projekten üblich, keinen lange voraus erdachten Releaseplan, aber dieser Zyklus hat sich über viele Jahre eingependelt. Man kann also, solange es in Zukunft keine gegenteiligen Bekanntmachungen gibt, davon ausgehen, dass es in etwa so weitergehen wird.

Eine Hauptversion bildet einen Zweig in der Entwicklung, der dann noch mehrere Jahre gewartet wird. Das heißt, es werden noch Fehlerkorrekturen eingespielt und etwa Über-

setzungen verbessert und aktualisierte Zeitzonenregeln eingebaut, aber keine neuen Features mehr entwickelt. Dies wird auch fortgesetzt, nachdem die nächste Hauptversion veröffentlicht wurde. Es gibt also immer mehrere gepflegte Hauptversionen. Der Wartungszeitraum einer Hauptversion beträgt aktuell fünf Jahre ab Veröffentlichung der Version x.y.0. Einzelheiten dazu sowie eventuelle Abweichungen werden unter anderem über die Website des PostgreSQL-Projekts veröffentlicht. Es wird davon ausgegangen, dass man in diesem Zeitraum die Möglichkeit findet, den Umstieg auf eine neuere Hauptversion anzugehen. Die Spanne von fünf Jahren deckt sich auch ungefähr mit den Supportzeiträumen von Linux-Distributionen der »Enterprise«- oder »Long-Term-Support«-Klasse.

Wenn Anwender oder Entwickler neue Projekte angehen, die auf PostgreSQL aufsetzen, ist ihnen prinzipiell zu empfehlen, die neueste Hauptversion einzusetzen. Das gilt auch, wenn diese zum Beispiel bei Projektstart erst kurz vor der Veröffentlichung stehen. Eine neue Version hat immer mehr Features und eine bessere Leistung, und mittelfristig wird man sowieso nicht um sie oder eine spätere Version herumkommen, wenn man PostgreSQL weiterhin einsetzen möchte.

Unterversionen (*minor releases*) werden etwa alle paar Monate herausgebracht, je nachdem, wie oft Korrekturen notwendig sind. Meist werden dabei Unterversionen von allen noch gewarteten Hauptversionen gleichzeitig herausgebracht, wenn die Fehlerkorrekturen auf alle zutreffen. Unterversionen sollten in der Regel umgehend von allen Anwendern eingespielt werden. Wer eine paketbasierte Installation hat, wird in der Regel entsprechende Updates automatisch erhalten.

Paketinstallation

Für einige populäre Linux-Betriebssysteme stellen wir hier kurz die Installation von PostgreSQL aus Binärpaketen vor.

Debian und Ubuntu

Das Debian-Paket für den PostgreSQL-Server heißt *postgresql*. Man installiert es also mit dem Befehl

```
apt-get install postgresql
```

Wenn man nur die Clientprogramme benötigt, installiert man das Paket *postgresql-client*. Das Serverpaket hängt vom Clientpaket ab, also ist es nicht notwendig, das Clientpaket zu installieren, wenn das Serverpaket schon installiert ist.

Debian und Ubuntu bieten die Möglichkeit, verschiedene Hauptversionen von PostgreSQL parallel zu installieren. Dazu besitzt jede Version eine eigene Paketgruppe. Die Pakete zu Version 9.1 heißen zum Beispiel *postgresql-9.1* und *postgresql-client-9.1*. Die unversionierten Pakete *postgresql* und *postgresql-client* sind eigentlich leere Pakete, die nur Abhängigkeiten in Bezug auf die Pakete der jeweils aktuellen Version aufweisen.

Ein stabiles Release von Debian oder Ubuntu enthält in der Regel nur eine oder maximal zwei Hauptversionen von PostgreSQL. Der Sinn dahinter ist der, dass neue Anwender die neue PostgreSQL-Version verwenden, existierende Anwendungen aber nach einem Upgrade des Betriebssystems die alte Version weiterverwenden können. Wer sich an die stabilen Releases von Debian oder Ubuntu halten möchte, dem sei empfohlen, die jeweils mitgelieferten Versionen zu verwenden. Darüber hinaus sind aber meist auch Backports von neueren Versionen in den entsprechenden Backport-Repositories verfügbar.

Red Hat

Unter Red Hat Linux (sowie Fedora, die Bezeichnung wird in diesem Buch synonym verwendet) heißt das Paket für den PostgreSQL-Server *postgresql-server*. Man beachte, dass das Paket namens *postgresql* lediglich einige clientseitige Programme enthält und nicht das darstellt, was man normalerweise unter einer vollständigen PostgreSQL-Installation versteht. (Gelegentlich ist es natürlich sinnvoll, ausschließlich den Client zu installieren.)

Installieren kann man die gewünschten Pakete zum Beispiel mit Yum

```
yum install postgresql-server
```

oder mit einem grafischen Paketverwaltungsprogramm.

SUSE

Auch auf SUSE Linux (sowie openSUSE) heißt das Serverpaket *postgresql-server* und das Clientpaket *postgresql*. Zur Installation verwendet man am besten YaST, oder *zypper* von der Konsole. Neben den in den Distributionen mitgelieferten PostgreSQL-Paketen werden auf dem openSUSE Build Service oft Backports von neueren PostgreSQL-Versionen angeboten.

Quellcode bauen

Wer trotz allem den Quellcode selber bauen oder verstehen möchte, wie die Pakete zustande kommen, um eventuell einige Änderungen vorzunehmen, der kann die Installation direkt aus dem Quellcode vornehmen.

Zuerst lädt man sich den Quellcode herunter. Dazu besucht man die Website *http://www.postgresql.org/*, wählt »Downloads« und »Source code« und hangelt sich dann durch die Verknüpfungen, bis man zu einer Verzeichnisübersicht gelangt. Der Quellcode befindet sich im Verzeichnis *source* im Unterverzeichnis mit der entsprechenden Versionsnummer. Als Beispiel verwenden wir hier Version *9.2.0*. Die Archivdatei mit dem Quellcode befindet sich dann im Unterverzeichnis *source/v9.2.0/* und heißt *postgresql-9.2.0.tar.bz2* oder *postgresql-9.2.0.tar.gz*. Wer `bunzip2` installiert hat, verwendet die erstere und spart damit Zeit beim Herunterladen; ansonsten nimmt man die zweite.

Ausgepackt wird das Quellcodearchiv mit folgenden Befehlen:

```
bunzip2 postgresql-9.2.0.tar.bz2
tar xf postgresql-9.2.0.tar
```

beziehungsweise

```
gunzip postgresql-9.2.0.tar.gz
tar xf postgresql-9.2.0.tar
```

Wenn GNU Tar installiert ist, was auf Linux- und BSD-Systemen normalerweise der Fall ist, dann kann man diese zwei Befehle auch zu jeweils einem zusammenfassen:

```
tar xjf postgresql-9.2.0.tar.bz2
```

beziehungsweise

```
tar xzf postgresql-9.2.0.tar.gz
```

Dadurch dann ein Verzeichnis namens *postgresql-9.2.0* im aktuellen Verzeichnis erzeugt. Für die Installation wechselt man in dieses Verzeichnis. In diesem Verzeichnis befindet sich eine Datei *INSTALL*, die die für die jeweilige Version aktuelle Installationsanleitung enthält.

Nun führt man das Programm *./configure* mit den für diese Installation gewünschten Optionen aus. Einzelheiten über die Optionen und die Features, die in der jeweiligen Version bereitgestellt werden, enthält die Datei *INSTALL*. Es ist sinnvoll, möglichst viele der Features anzuschalten, damit man später keine Probleme hat, wenn sich herausstellt, dass man doch eines mehr benötigt. Natürlich gibt es auch Argumente für möglichst schlanke Installationen, aber das hat sich bei PostgreSQL in der Praxis noch nie so richtig ausgewirkt. Man bedenke auch, dass ein Großteil der Anwender Pakete verwendet, die mit allen Optionen gebaut sind. Der Zweck der Optionen ist nicht etwa, experimentelle oder gefährliche Features auszublenden, sondern nur, Benutzern, die nicht alle Zusatzbibliotheken installiert haben, die Möglichkeit zu geben, nicht benötigte Features wegzulassen.

Wir empfehlen mindestens die folgenden Optionen, die auch für das übrige Buches ausreichen:

```
./configure --prefix=/usr/local/pgsql --enable-nls --with-tcl --
with-perl --with-python --with-gssapi --with-pam --with-ldap --with-openssl --with-libxml
--with-libxslt
```

Mit der Option `--prefix` wird das Installationsverzeichnis angegeben. Das hier verwendete Beispiel */usr/local/pgsql* ist auch die Voreinstellung. Man kann ein beliebiges Verzeichnis angeben. Beachten Sie, dass das Installationsverzeichnis der Programme mit der Voreinstellung dann */usr/local/pgsql/bin* lauten würde und sich damit nicht im normalen Pfad befände. Man müsste alle Programme mit vollem Verzeichnis aufrufen oder den Suchpfad in der Shell entsprechend anpassen. Mit Bash könnte man dazu Folgendes in die Datei *~/.bash_profile* schreiben:

```
PATH=$PATH:/usr/local/pgsql/bin
```

Zum Bauen von PostgreSQL benötigt man eine normale Entwicklungsumgebung mit C-Compiler, System-Headern und GNU Make. Um die erwähnten Zusatzoptionen verwenden zu können, müssen diverse zusätzliche Pakete installiert werden, nämlich Readline, Zlib, Perl, Python, Tcl, Kerberos, PAM, LDAP, OpenSSL, LibXML und LibXSLT, je nachdem, welche Optionen gewählt wurden. Diese Pakete sind auf allen üblichen Betriebssystemen erhältlich. Beachten Sie, dass Sie das jeweilige Entwicklungspaket installieren müssen, das meist einen Paketnamen aufweist, der auf *-dev* oder *-devel* endet, zum Beispiel *libreadline-dev* auf Debian und Ubuntu oder *readline-devel* auf Red Hat und SUSE.

Wenn `configure` abgeschlossen wurde, kann das eigentliche Kompilieren erfolgen. Dazu gibt man

```
make
```

ein.

Wenn GNU Make nicht das Standard-Make ist, zum Beispiel auf BSD-Systemen, gibt man stattdessen `gmake` ein.

Wenn das Kompilieren beendet ist, kann man die Installation starten, indem man

```
make install
```

(beziehungsweise `gmake install`) eingibt.

Bis zu diesem Zeitpunkt auftretende Fehler sind fast immer auf fehlende oder falsch installierte andere Software zurückzuführen, es sei denn, man verwendet ein nagelneues, außergewöhnliches System, das noch nicht ausreichend unterstützt wird.

Normalerweise führt man die einzelnen Schritte des Bauens als normaler Benutzer aus, also nicht als *root*. Lediglich im letzten Schritt `make install` werden je nach Konfiguration des Installationsverzeichnisses Dateien in Verzeichnisse kopiert, für die nur *root* Schreibrechte besitzt. Dann muss man für diesen Schritt zum Benutzer *root* wechseln:

```
$ su
# make install
```

Man beachte, dass man nicht `su -` eingibt, weil man sonst als *root* im falschen Verzeichnis landet.

Auf manchen Betriebssystemen (zum Beispiel Ubuntu und Mac OS X) geht es auch so:

```
$ sudo make install
```

Nun ist die PostgreSQL-Software installiert. Als Nächstes geht es daran, die Datenbank einzurichten. Darum geht es im Anschluss.

Contrib

In diesem Buch wird mehrmals davon die Rede sein, dass ein bestimmtes Zusatzmodul aus »contrib« nachinstalliert werden soll. Im Quellcode von PostgreSQL gibt es ein

Verzeichnis namens *contrib*, unter dem verschiedene Zusatzprogramme und -module zu finden sind, die aus verschiedenen Gründen nicht Teil des PostgreSQL-Kerns – also der Standardinstallation – sind, zum Beispiel weil sie zu neu sind oder eher wenig Anwender haben.

Wenn man ein solches Modul nachinstallieren möchte und aus dem Quellcode gebaut hat, wechselt man in das entsprechende Verzeichnis und gibt dort `make` und `make install` ein, zum Beispiel so:

```
$ cd contrib/auto_explain
$ make
$ sudo make install
```

Wenn man eine paketbasierte Installation nutzt, gibt es normalerweise ein Paket mit einem Namen wie *postgresql-contrib* (Debian- und RPM-basierte Systeme) oder eine andere Option bei der Installation.

Um die in den Modulen enthaltenen Funktionen oder Ähnliches zu installieren, muss eine »Erweiterung« (*extension*) in einer Datenbank installiert werden. Dazu führt man folgenden Befehl aus:

```
psql -d dbname -c 'CREATE EXTENSION name'
```

In Versionen vor PostgreSQL 9.1 wird meist eine SQL-Skriptdatei mitgeliefert, die in einer Datenbank ausgeführt werden muss. Das funktioniert in der Regel ungefähr so:

```
psql -d dbname -f pfad/datei.sql
```

Abweichendes wird in diesem Buch an den entsprechenden Stellen erwähnt werden.

PostgreSQL einrichten

Nach der Installation der Software besteht der nächste Schritt darin, die PostgreSQL-Instanz einzurichten und zu starten.

Datenverzeichnis initialisieren

Bei der einfachsten Konfiguration eines PostgreSQL-Systems werden alle Datenbankdaten und weitere Verwaltungsinformationen in einem einzigen Verzeichnis im Dateisystem abgelegt. Dieses Verzeichnis wird üblicherweise »Datenverzeichnis« genannt. Fortgeschrittene Konfigurationen verteilen die Daten auch über mehrere Verzeichnisse, aber diese einfache Konfiguration ist als Anfang ausreichend.

Datenverzeichnis bestimmen

Der Ort des Datenverzeichnisses kann vom Anwender frei gewählt werden, es gibt keine Voreinstellung. Für Quellcodeinstallationen, die den vorgegebenen Installationspfad */usr/local/pgsql/* verwenden, ist */usr/local/pgsql/data/* als Datenverzeichnis üblich. Für paketbasierte Installationen auf Linux-Systemen ergibt sich aus dem einschlägigen Filesystem

Hierarchy Standard (FHS) */var/lib/postgresql/* (oder */var/lib/pgsql/* oder so ähnlich) als passender Pfad. Die erste Möglichkeit wird zum Beispiel von Debian und Ubuntu verwendet, die zweite von Red Hat und SUSE. Ebenso passend wäre */srv/postgresql/*, was aber von Linux-Distributionen derzeit nicht verwendet wird.

Zum Ausprobieren, Testen oder Spielen ist es allerdings auch möglich, ein beliebiges Verzeichnis im eigenen Home-Verzeichnis als Datenverzeichnis zu verwenden.

Oft wird für ein Datenbanksystem ein besonderes Speichersystem – zum Beispiel NAS, SAN, RAID oder einfach eine zusätzliche Festplatte – angeschafft, das zusätzlich gemountet wird. Es gibt drei verschiedene Möglichkeiten, es einzubinden:

1. Man mountet das Speichersystem an eine beliebige Stelle, zum Beispiel */data1/*, */db/* oder */srv/postgresql/*, und konfiguriert die PostgreSQL-Software so um, dass diese Stelle als Datenverzeichnis verwendet wird.
2. Man mountet das Speichersystem an eine beliebige Stelle und setzt von der normalerweise verwendeten Stelle einen Symlink, also beispielsweise */var/lib/pgsql/data* → */data1*.
3. Man mountet das Speichersystem direkt an die normalerweise verwendete Stelle.

Der erste Ansatz funktioniert nur, wenn die verwendete Installationsmethode das Ändern des Datenverzeichnisses unterstützt. Die üblichen Pakete (RPM, Deb, Windows) tun das aber. Der zweite Ansatz sollte immer funktionieren (wenn ihm nicht irgendwelche Aversionen gegen Symlinks im Wege stehen). Die dritte Variante erzeugt möglicherweise ein Problem, wenn das Verzeichnis *lost+found* im Weg ist, da `initdb` (siehe später) ein leeres Verzeichnis erwartet. Man kann das Datenverzeichnis daher unter den Mountpunkt legen, aber nicht direkt in ihn. Weitere Informationen zur Einteilung und Verwaltung der Festplatte finden Sie in Kapitel 10.

Benutzerkonto einrichten

Für Serverdienste ist es üblich, dass sie unter einem separaten Benutzerkonto gestartet werden, damit bei einem Einbruch über das Netzwerk der Schaden eingedämmt werden kann und die dem Dienst zugeordneten Systemressourcen besser verwaltet werden können.

Für PostgreSQL ist es üblich, diesen Benutzer »postgres« zu nennen. Auf BSD-Systemen wird stattdessen teilweise »pgsql« verwendet; das ist aber ansonsten nicht zu empfehlen. In diesem Buch wird der Benutzername »postgres« repräsentativ für diesen Zweck verwendet.

Wichtig ist, dass dieser Benutzer keine Superuser- oder Administratorrechte besitzt, also nicht »root« auf Unix-Systemen oder »Administrator« auf Windows ist. PostgreSQL wird sich weigern, unter solchen Benutzerkonten zu laufen. Außerdem sollte man das Benutzerkonto für PostgreSQL nicht mit anderen Diensten (wie etwa dem Webserver) teilen. Natürlich gilt auch hier, dass man für Test- oder Spielinstallationen irgendeinen Benutzer (außer *root* beziehungsweise Administrator) verwenden kann.

Paketbasierte Installationen richten den Benutzer üblicherweise automatisch ein. Das gilt für RPMs, Debian-Pakete und Windows-Installationen. Wenn direkt aus dem Quellcode installiert wird, muss der Benutzer von Hand angelegt werden. Der Befehl dafür lautet auf Linux-Systemen wie folgt:

```
useradd -m postgres
```

Dieses Benutzerkonto hat zwei Funktionen:

- Alle Dateien im Datenverzeichnis müssen diesem Benutzer gehören.
- Der Serverprozess wird unter diesem Benutzer ausgeführt.

Achten Sie darauf, dass die Softwareinstallation (wie vorher schon erwähnt) nicht unter diesem Benutzer ausgeführt wird und die installierten Dateien möglichst nicht diesem Benutzer gehören – denn falls es wirklich zu einer Sicherheitsverletzung kommen sollte, möchte man dem PostgreSQL-Serverprozess ja nicht die Möglichkeit geben, seine eigenen Programmdateien zu verfälschen.

Datenverzeichnis initialisieren

Wenn man sich darüber im Klaren ist, welches Verzeichnis das Datenverzeichnis sein soll, und man ein Benutzerkonto angelegt hat, kann man das Datenverzeichnis initialisieren. Dazu dient der Befehl `initdb`, der mit PostgreSQL installiert wurde. Die Namensgebung ist etwas ungenau, da dieser Befehl nicht etwa eine Datenbank initialisiert, sondern ein Datenverzeichnis, aber egal ... Im einfachsten Fall wird dieser Befehl schlicht mit dem Datenverzeichnis als Argument aufgerufen, also zum Beispiel so:

```
initdb /usr/local/pgsql/data
```

Dieser Befehl muss unter dem für den PostgreSQL-Server vorgesehenen Benutzerkonto ausgeführt werden, also beispielsweise »postgres«.

Wenn das angegebene Verzeichnis noch nicht vorhanden ist, legt `initdb` es an. Normalerweise wird das aber nicht funktionieren, da nur *root* das Recht hat, beliebige Verzeichnisse anzulegen. Daher muss hier üblicherweise ein kleiner Umweg gegangen werden: Man legt zunächst das Verzeichnis als *root* an, ändert den Eigentümer auf *postgres*, loggt sich dann als *postgres* ein und führt `initdb` aus. Konkret könnte das also so aussehen:

```
root# mkdir /usr/local/pgsql/data
root# chown postgres /usr/local/pgsql/data
root# su - postgres
postgres$ initdb /usr/local/pgsql/data
```

Die Ausgabe von `initdb` sieht wie in Beispiel 1-1 aus. In den ersten beiden Zeilen wird darauf hingewiesen, dass der Benutzer, den man im Moment verwendet, für die Dateien im Datenverzeichnis genutzt wird und auch für den Serverprozess Verwendung finden muss. Wenn man versucht, `initdb` als *root* auszuführen, wird `initdb` an dieser Stelle mit einem Fehler abbrechen, da eine solche Verwendung als unsicher betrachtet wird. Ebenso wird `initdb` abbrechen, wenn der aktuelle Benutzer keine Schreibrechte für das angegebene Verzeichnis besitzt. Wenn man ansonsten feststellt, dass man sich beim Benutzer

oder sonstwie geirrt hat, kann man das Datenverzeichnis einfach leeren und wieder neu beginnen, zum Beispiel so:

```
rm -rf /usr/local/pgsql/data/*
```

(Man kann natürlich auch das Verzeichnis selbst löschen, aber dann muss man die Trickserei von oben wiederholen, um es neu anzulegen.)

Die dritte, vierte und fünfte Zeile geben an, mit welcher Locale, Kodierung und Textsuchkonfiguration das Datenbanksystem initialisiert wird. Weitere Informationen dazu folgen in Kapitel 2. Die Voreinstellung hier hängt von der Einstellung des Betriebssystems ab. Wenn also bei der Installation des Betriebssystems die Einstellung für »Deutsch« und »Deutschland« gewählt wurde, sollte die Ausgabe in etwa wie im nachfolgenden Beispiel aussehen.

Die folgenden Zeilen sind Fortschrittsanzeigen, die angeben, wie weit `initdb` schon durchlaufen wurde. Interessant sind eventuell die beiden Zeilen, in denen die Vorgabewerte für `max_connections` und `shared_buffers` gewählt werden. Hier versucht `initdb`, eine einigermaßen vernünftige Voreinstellung zu finden, die in die vom Betriebssystem vorgegebenen Ressourcengrenzen passt. Einige Betriebssysteme weisen an dieser Stelle extrem kleine Grenzen auf, aber auf Linux-Systemen sollte die Ausgabe immer in etwa in diesem Bereich landen. Trotzdem sind die hier gewählten Speichereinstellungen für ein vernünftiges Datenbanksystem auf aktueller Hardware immer zu klein. Bevor ein System produktiv geschaltet wird, sollte die Konfiguration also unbedingt überarbeitet werden. Informationen dazu finden Sie in Kapitel 2.

Nach den Fortschrittsanzeigen wird hier eine Warnung angezeigt, die darauf hinweist, dass die aktuelle Authentifizierungsmethode unsicher ist. Informationen dazu finden Sie in Kapitel 7. Auch mit diesem Thema sollten Sie sich befassen, bevor Sie das System produktiv schalten.

Zum Abschluss finden Sie Informationen darüber, wie Sie den Datenbankserver nun endlich starten können. Was es mit den vorgeschlagenen Befehlen auf sich hat, wird im folgenden Abschnitt erläutert.

Beispiel 1-1: Ausgabe von initdb

```
Die Dateien, die zu diesem Datenbanksystem gehören, werden dem Benutzer
»postgres« gehören. Diesem Benutzer muss auch der Serverprozess gehören.

Der Datenbankcluster wird mit der Locale »de_DE.utf8« initialisiert werden.
Die Standarddatenbankkodierung wurde entsprechend auf »UTF8« gesetzt.
Die Standardtextsuchekonfiguration wird auf »german« gesetzt.

berichtige Zugriffsrechte des bestehenden Verzeichnisses /usr/local/pgsql/data ... ok
erzeuge Unterverzeichnisse ... ok
wähle Vorgabewert für max_connections ... 100
wähle Vorgabewert für shared_buffers ... 32MB
erzeuge Konfigurationsdateien ... ok
erzeuge Datenbank template1 in /usr/local/pgsql/data/base/1 ... ok
initialisiere pg_authid ... ok
```

```
initialisiere Abhängigkeiten ... ok
erzeuge Systemsichten ... ok
lade Systemobjektbeschreibungen ... ok
erzeuge Sortierfolgen ... ok
erzeuge Konversionen ... ok
erzeuge Wörterbücher ... ok
setze Privilegien der eingebauten Objekte ... ok
erzeuge Informationsschema ... ok
lade Serversprache PL/pgSQL ... ok
führe Vacuum in Datenbank template1 durch ... ok
kopiere template1 nach template0 ... ok
kopiere template1 nach postgres ... ok

WARNUNG: Authentifizierung für lokale Verbindungen auf »trust« gesetzt
Sie können dies ändern, indem Sie pg_hba.conf bearbeiten oder beim
nächsten Aufruf von initdb die Option -A, oder --auth-local und
--auth-host, verwenden.

Erfolg. Sie können den Datenbankserver jetzt mit

    postgres -D /usr/local/pgsql/data
oder
    pg_ctl -D /usr/local/pgsql/data -l logdatei start

starten.
```

In paketbasierten Installationen wird `initdb` auf verschiedene Weise ausgeführt:

- Debian- und Ubuntu-Pakete führen den `initdb`-Schritt am Ende der Paketinstallation durch, in der sogenannten »postinst«-Phase. Wenn das Paket also vollständig installiert ist, wurde `initdb` bereits ausgeführt.
- Bei RPM-Paketen für neuere Red-Hat-Systeme, die Systemd verwenden, wird `initdb` über das Wrapper-Program `postgresql-setup` ausgeführt, was unter anderem auch den positiven Effekt hat, dass das passende Datenverzeichnis angelegt und verwendet wird. Dazu führt man als *root* `postgresql-setup initdb` aus.
- Mit RPM-Paketen für ältere Red-Hat-Systeme mit traditionellem System-V-Init-System führt man `initdb` manuell über das Init-Skript (siehe nächster Abschnitt) durch, indem man als *root* `/etc/init.d/postgresql initdb` ausführt. Auch hier wird auf diese Weise das passende Datenverzeichnis angelegt. Wenn man den Server über das Init-Skript zu starten versucht, bevor das Datenverzeichnis angelegt ist, erhält man eine Fehlermeldung.
- RPM-Pakete von SUSE führen `initdb` im Init-Skript (siehe nächster Abschnitt) automatisch aus. Wenn man den Server durch Ausführen von `/etc/init.d/postgresql start` startet und an der vorgesehenen Stelle (*/var/lib/pgsql/data*) kein Datenverzeichnis vorhanden zu sein scheint, wird *initdb* automatisch ausgeführt und danach der Server wie vorgesehen gestartet. Das funktioniert normalerweise genauso gut. Dieses Verfahren kann jedoch zu Problemen führen, wenn man das Datenverzeichnis auf einer separaten Speichereinheit ablegt, die nicht immer gemountet ist, etwa weil sie im Rahmen einer hochverfügbaren Cluster-Konstruktion von Zeit zu Zeit unterschiedlichen Rechnern zugeordnet ist. In dem Fall kann dieses Verfahren dazu

führen, dass `initdb` zu völlig unpassenden Zeiten ausgeführt wird. In solchen Umgebungen sollte man sich das Init-Skript selbst genau ansehen und eventuell anpassen. (Ältere RPMs von Red Hat wiesen denselben Fehler auf.)

Wegen der Vielzahl von Paketvarianten und der neuen Init-Systeme in einigen Linux-Distributionen empfielt es sich, die Dokumentation des jeweiligen Pakets einzusehen, um das empfohlene Verfahren zum Initialisieren und Starten des Datenbanksystems in Erfahrung zu bringen. Bei RPM-Paketen finden sich diese Informationen in der Regel in */usr/share/doc/postgresql.../README.rpm-dist*. Bei Debian-Paketen kann */usr/share/doc/postgresql.../README.Debian* hilfreich sein.

Server starten

Um das Datenbanksystem in Betrieb zu nehmen, wird der Serverdienst gestartet. Der Serverprozess läuft dann auf dem System im Hintergrund und wartet auf Verbindungen von Clientprogrammen. Wenn ein Clientprogramm eine Verbindung aufbaut, kann der Client durch den Server Datenbankabfragen und andere Operationen ausführen.

Server starten mit Programm postgres

Nachdem die PostgreSQL-Software installiert ist und auch ein Datenverzeichnis erzeugt wurde, kann der Datenbankserver gestartet werden. Das Datenbankserver-Programm heißt `postgres`. Im einfachsten Fall startet man den Datenbankserver, indem man den Befehl `postgres` unter Angabe eines Datenverzeichnisses, wie in der Ausgabe von `initdb` (siehe Beispiel 1-1) empfohlen, startet, also zum Beispiel so:

```
postgres -D /usr/local/pgsql/data
```

Die Option `-D` gibt hier das Datenverzeichnis an. Dieser Befehl muss ebenfalls unter dem für den PostgreSQL-Server vorgesehenen Benutzerkonto gestartet werden, und zwar unter demselben, unter dem `initdb` ausgeführt wurde. Anderenfalls wird sich das Serverprogramm beschweren und nicht starten. Ein Startversuch als root würde etwa diese Fehlermeldung ergeben:

```
Der PostgreSQL-Server darf nicht als »root« ausgeführt werden.  Der
Server muss unter einer unprivilegierten Benutzer-ID gestartet werden,
um mögliche Sicherheitskompromittierung zu verhindern.  In der
Dokumentation finden Sie weitere Informationen darüber, wie der
Server richtig gestartet wird.
```

Wenn man stattdessen einen anderen normalen Benutzer verwendet, wird man gar keine Zugriffsrechte auf das Datenverzeichnis besitzen und daher typischerweise eine Fehlermeldung dieser Art erhalten:

```
postgres kann nicht auf die Serverkonfigurationsdatei »/usr/local/pgsql/data/postgresql.
conf« zugreifen: Keine Berechtigung
```

Wenn der Server also erfolgreich gestartet ist, wird er im Vordergrund ausgeführt und gibt auf der Konsole Logmeldungen aus. Der Anfang sieht – je nach Version leicht unterschiedlich – in etwa so aus:

```
LOG:  Datenbanksystem wurde am 2012-09-06 22:41:50 CEST heruntergefahren
LOG:  Datenbanksystem ist bereit, um Verbindungen anzunehmen
LOG:  Autovacuum-Launcher startet
```

Der Server läuft so auf der Konsole weiter, bis man ihn etwa durch Drücken von *Sieg+C* beendet. Das ist für den Dauerbetrieb natürlich unpraktisch. Man möchte, dass der Server im Hintergrund unabhängig von der Konsole ausgeführt wird und er die Logmeldungen in einer Datei speichert. Das kann man mit etwas gespickter Shell-Syntax folgendermaßen erreichen:

```
nohup postgres -D /usr/local/pgsql/data >logdatei 2>&1 </dev/null &
```

Dieser Befehl schiebt `postgres` in den Hintergrund (`&`) und lenkt die Standardausgabe in eine Datei um (`>`), ebenso auch die Standardfehlerausgabe (`2>&1`). Außerdem wird die Standardeingabe stillgelegt (`<`), andernfalls würde sie weiterhin zum Terminal zeigen und der Prozess könnte sich nicht richtig von der Konsole lösen. Der Befehl `nohup` macht den Prozess schließlich immun gegen das HUP-Signal, das auftreten könnte, wenn man sich aus der Konsole ausloggt. Den Befehl kann man auf diese Weise verwenden, aber etwas kompliziert ist das schon. Alternativ gibt es in der PostgreSQL-Installation das Programm `pg_ctl`, das das Ganze für Sie erledigt, wie im nächsten Abschnitt beschrieben wird.

Server starten mit dem Programm pg_ctl

Bei *pg_ctl* handelt es sich um ein Programm, das das Starten des PostgreSQL-Servers vereinfacht. Der Aufruf erfolgt wie von `initdb` vorgeschlagen:

```
pg_ctl -D /usr/local/pgsql/data -l logdatei start
```

Intern macht `pg_ctl` letztlich genau das, was auch mit dem obigen Shell-Befehl bewirkt würde. (`pg_ctl` war früher sogar einmal ein Shell-Skript, ist es mittlerweile aber nicht mehr.)

Die entsprechende Ausgabe sieht im Erfolgsfall so aus:

```
Server startet
```

Das bedeutet, dass der Server jetzt mit den Starten beschäftigt ist, aber es bedeutet nicht, dass der Start auch erfolgreich durchgeführt wurde. Man kann ein paar Sekunden warten und dann versuchen, eine Verbindung zum Datenbankserver herzustellen. Wenn es Probleme gibt, sollte ein Blick in die Logdatei helfen. Das Warten lässt sich aber auch automatisieren, indem man die Option `-w` angibt:

```
pg_ctl -D /usr/local/pgsql/data -l logdatei -w start
```

Dann versucht *pg_ctl* intern, nach dem Starten wiederholt eine Verbindung mit dem Server herzustellen, und beendet sich erst, wenn die Testverbindung erfolgreich war. Die Ausgabe sieht dann so aus:

```
warte auf Start des Servers.... fertig
Server gestartet
```

In der Voreinstellung wird das Warten nach 60 Sekunden abgebrochen; das kann mit der Option `-t` geändert werden.

Das Warten beim Start ist oft sinnvoll, es ist aber manchmal nicht möglich, etwa wenn die Konfiguration der Zugangskontrolle keine Testverbindungen zulässt. Deswegen ist es nicht das Standardverhalten.

Server mit Init-Skript starten

Letztendlich möchte man es normalerweise so einrichten, dass PostgreSQL mit allen Systemdiensten beim Booten des Rechners gestartet wird. Wenn PostgreSQL als Paket installiert wurde, wurde das vom Paket auch so eingerichtet. Auf Debian und Ubuntu ist der PostgreSQL-Server automatisch zum Starten beim Booten konfiguriert. Auf (älteren) Red Hat- und SUSE-Systemen kann man mit folgendem Aufruf den PostgreSQL-Server zum Starten beim nächsten Booten konfigurieren:

```
chkconfig postgresql on
```

Auf Systemen mit Systemd ist der entsprechende Befehl dieser:

```
systemctl enable postgresql.service
```

Wenn ein Init-Skript eingerichtet ist, kann man es auch zum manuellen Starten verwenden. Der entsprechende Befehl auf Linux-Systemen ist folgender:

```
service postgresql start
```

(Auf älteren Systemen lautet er `/etc/init.d/postgresql start`; ältere Debian- und Ubuntu-Pakete verfügen stattdessen über ein versioniertes Startskript, wie beispielsweise `/etc/init.d/postgresql-9.0`.)

Unter Systemd startet man den Dienst folgendermaßen:

```
systemctl start postgresql.service
```

Auch hier empfiehlt sich im Zweifel ein Blick in die Dokumentation des jeweiligen Pakets.

All diese Befehle müssen als *root* ausgeführt werden; die Init-Skripten sorgen intern dafür, dass zum richtigen Benutzer umgeschaltet wird.

Wenn man PostgreSQL aus dem Quellcode installiert hat, ist Handarbeit vonnöten. Dazu wird der gewünschte Startbefehl in die vom Betriebssystem vorgesehene Konfigurationsdatei eingetragen. Auf Linux-Systemen ist das in der Regel eine Datei in */etc/init.d/*, also zum Beispiel */etc/init.d/postgresql*, die dann mit einem systemspezifischen Befehl in andere Verzeichnisse verlinkt wird, von wo aus sie dann beim Systemstart in der richtigen Reihenfolge aufgerufen wird. Einige Linux-Distributionen verwenden andere Systeme wie Systemd oder Upstart, die wieder andere Konfiguration benötigen. Das Schreiben der entsprechenden Skripten verlangt detailliertes Wissen um die Gegebenheiten des jeweiligen Betriebssystems; seine Beschreibung würde den Rahmen dieses Buches sprengen. Die Betriebssystem-Dokumentation sollte aber entsprechende Informationen liefern.

Um den PostgreSQL-Server zu starten, verwendet ein solches Skript letztlich auch nur einen Aufruf von `postgres` oder `pg_ctl` (wie oben beschrieben), jeweils mit den vom Betriebssystemhersteller oder Paketierer vorgesehenen Einstellungen bezüglich Datenverzeichnis und Logging.