„May The Source Be With You“

Programme aus dem Quelltext übersetzen

Markus Dahms, Christoph Dahms

25. Februar 2008

Einführung – warum aus Quellen

- oft neuere Versionen verfügbar
- exotische Software manchmal gar nicht als Distributions-Paket zu bekommen
- man kann mal eben 'ne Kleinigkeit ändern:
  - Optimierung auf vorhandene Hardware
  - Konfiguration von optionalen Funktionen
  - Verbesserungen einbauen...

Umgang mit Quellpaketen – Beschaffen

- Quellpaket („Tarball“) herunterladen (***.tar.(gz|bz2)**, ***.zip**)
- Versionskontrollsystem anzapfen (CVS, Subversion, GIT, Mercurial, Darcs . . .)
- Quellen von der Distribution beziehen
  - Debian: **apt-get source <paketname>**

Umgang mit Quellpaketen – Entpacken

- ***.tar.gz**: **tar zxvf $PAKET**
- ***.tar.bz2**: **tar jxvf $PAKET**
- ***.zip**: **unzip $PAKET** (Verzeichnis im Zip?)

Überlegungen vor der Installation

- Lohnt sich die neue Version überhaupt?
- Sind alle Abhängigkeiten erfüllt?
- Wohin installieren? – Pfad-Präfix
- Ist genug Platz verfügbar?
- Root oder nicht Root?

Konfigurations- und Build-Systeme

- Konfiguration:
  - Prüfen der Abhängigkeiten
  - benutzte Verzeichnissen und Programmen
  - optionale Programmteile und Funktionen
- Build – das „Bauen“ eines Programms, also Kompilieren und Linken mit Bibliotheken

Konfigurations- und Build-Systeme – Vertreter

- **make** – der Klassiker
- **GNU Autotools** – sehr verbreitet
- **cmake** – Cross-platform
- **scons** – Python-basiert
- **ant** – alles in XML
- **imake**, **cons** . . . und viele andere mehr.

Die GNU Autotools

- **autoconf**:
  - erstellt das Konfigurationssystem
  - erweiterbar durch Skripte und M4-Makros
- **automake**:
  - erstellt Steuerdateien für **make**
  - anpassbar durch Vorlagendateien
- diverse kleine Zusatzprogramme...

Konfiguration

- *Bourne-Shell*-Skript
- **./configure**
  - **--help**
  - **--prefix** <*Pfad*>
  - **--enable-$X**, **--disable-$X**, **--with-$X**,
- Umgebungsvariablen werden beachtet:
  CC, **CPPFLAGS**, **LDFLAGS**, **LIBS** ...

„Bauen“ des Programms

- **make** tut es meistens
  - **-j<n>** – parallel bauen
  - **-k** – Fehler ignorieren

- `make check` – Testen, wenn möglich
- `make install` – Installation
  - üblicherweise als *Super-User*

Fehler aufspüren

- Compiler-Fehler

```
bla.c: In function 'main':
bla.c:6: error: syntax error before '}' token
```

  - fehlende Header-Dateien
  - falscher Suchpfad für Include-Dateien

- Linker-Fehler

```
bla.o: In function 'foo':bla.c:(.text+0x1d): undefined reference
to 'bar'
collect2: ld returned 1 exit status
```

  - fehlende Bibliotheken
  - falscher Suchpfad für Bibliotheken