Solaris Package Management

Christoph Hartmann / Martin Sprengel

Gliederung

- Einleitung
- Aufbau eines Pakets
- Paketerstellung unter Solaris
- Paketadministration
- Zusammenfassung
- Literatur

- Einleitung
  - Übersicht Solaris
  - Was ist Condor?
- Aufbau eines Pakets
- Paketerstellung unter Solaris
- Paketadministration
- Zusammenfassung
- Literatur

Einleitung
Übersicht Solaris (1/2)

- Entwicklung von Sun Microsystems
- BSD-Unix-Variante
- 1982 erschien Version 1.0 (Motorola 68000)
- Version 5 (2.5) auf Basis von System V
  - Bündelung mit Java und GUI CDE
- Seit 5.7 Unterstützung für 64 Bit
- aktuelle Version 5.10

Einleitung
Übersicht Solaris (2/2)

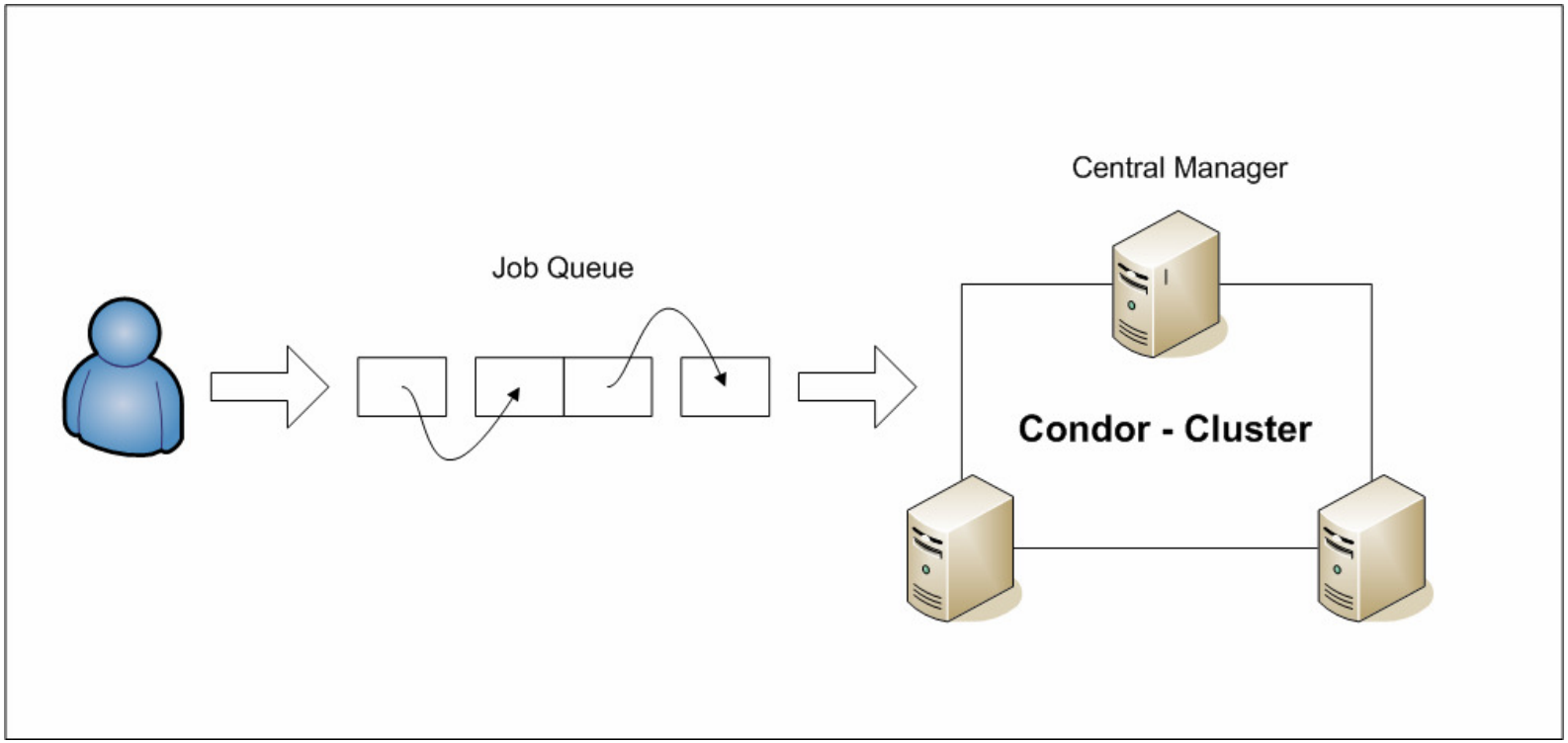
- Features
  - Sun Java System Directory Server
  - Sun Java Application Server
  - Sun Java System Message Quene
  - Sun Java Desktop System
  - Sun Management Center

Einleitung
Was ist Condor? (1/2)

- Condor-Projekt wurde 1988 an der University of Wisconsin-Madison, als Nachfolger des Remote-UNIX Projekts gestartet
- Condor ist ein Workload Management System das Mechanismen und Methoden für High Troughput Computing (HTC) bereitstellt
  - Job Queueing Mechanismus
  - Scheduling Verfahren
  - Prioritätsschemen
  - Ressourcen Monitoring und Ressourcen Management
- Aktuelle stabile Version 6.6.9

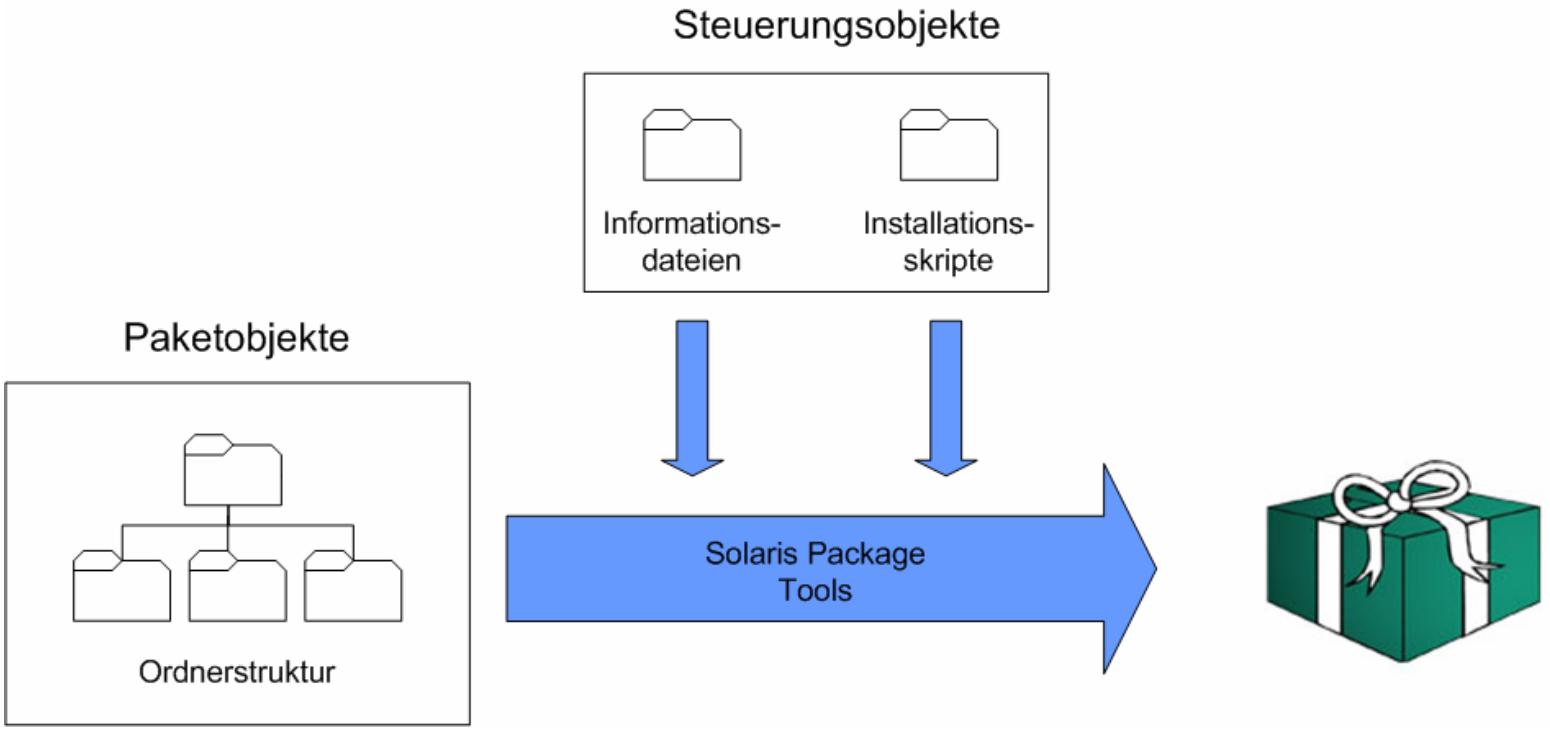
Einleitung - Was ist Condor? (2/2)

- Allgemeiner Ablauf



- Einleitung
- Aufbau eines Pakets
  - Allgemeines Vorgehen
  - Paketobjekte
  - Kontrollobjekte
    - Informationsdateien
    - Installationsskripte
- Paketerstellung unter Solaris
- Paketadministration
- Zusammenfassung
- Literatur

Aufbau eines Pakets
Begriffsklärung / Allgemeines Vorgehen



Aufbau eines Pakets
Paketobjekte

- Alle Objekte, welche die eigentliche Applikation ausmachen
- Mögliche Objekttypen
  - Dateien
  - Verzeichnisse
  - Named Pipes
  - Links
  - Devices
- Paketobjekte werden in einer Ordnerstruktur angeordnet
  - Struktur entspricht der auf dem Zielsystem

Aufbau eines Pakets
Kontrollobjekte - Informationsdateien

- Zwei benötigte Informationsdateien
  - `pkginfo` – Datei
  - `prototype` – Datei
- Optionale Informationsdateien
  - `compver` – Datei
  - `depend` – Datei
  - `space` – Datei
  - `copyright` – Datei

Aufbau eines Pakets
Variablen in Informationsdateien (1/4)

- Zwei Variablentypen in Informationsdateien möglich
  - Build-Variablen
    - Beginnen mit einem Kleinbuchstaben
    - Werden bei der Erzeugung durch `pkgmk` ausgewertet
  - Install-Variablen
    - Beginnen mit einem Großbuchstaben
    - Werden während der Installation durch `pkgadd` ausgewertet
- Variablendefinition
  - in der `pkginfo` – Datei: `VAR=value`
  - in der `prototype` – Datei : `!VAR=value`

Aufbau eines Pakets
pkginfo – Datei (1/3)

- ASCII – Datei, welche das Paket beschreibt und Informationen für die Installation bereitstellt
- Format
  - `Parameter="value"`
- Fünf benötigte Parameter
  - `PKG` – Name des Pakets
  - `NAME` – Beschreibung des Pakets
  - `ARCH` – Architektur, für die das Paket bestimmt ist
  - `VERSION` – Versionsstring des Pakets
  - `CATEGORY` – legt eine Kategorie fest

Aufbau eines Pakets
pkginfo – Datei (2/3)

- Optionale Parameter
  - `BASEDIR` – Standardmäßiges Installationsverzeichnis
  - `CLASSES` – Liste von Klassennamen
  - `DESC` – Beschreibung des Pakets
  - `EMAIL` – E-Mail-Adresse für Support
  - `HOTLINE` – Telefonnummer für Support
  - `INTONLY` – Paket darf nur interaktiv installiert werden, wenn der Parameter nicht Null ist
  - `ISTATES / RSTATES` – Paket darf nur in den aufgelisteten Runleveln installiert werden
  - `MAXINST` – zulässige Anzahl von Instanzen auf einem System
- Auch eigene Parameter möglich

Aufbau eines Pakets
pkginfo – Datei (3/3)

- Beispiel

```
# benoetigte Parameter
PKG="BSDAcondor"
NAME="Condor stellt Mechanismen und Verfahren für High
  Throughput Computing zur Verfügung. Die Software ist
  für Sparc Architektur verfügbar und in der /usr/
  Partition installiert"
ARCH=sparc
VERSION=1.0
CATEGORY=application

# optionale Parameter
BASEDIR=/usr/
```

Aufbau eines Pakets
prototype – Datei (1/5)

- In der `prototype` – Datei werden alle Paketobjekte aufgelistet
- Zwei Möglichkeiten der Erzeugung
  - manuell mit dem Texteditor
  - automatisch mit dem `pkgproto` – Tool
- Format
  - eine Zeile steht für einen Eintrag
  - besteht aus mehreren Feldern
    - [part] [ftype] [class] [path] [major] [minor] [mode] [owner] [group]

Aufbau eines Pakets
prototype – Datei (2/5)

- Felder
  - `part` – um ein Objekt einem bestimmten Teil des Pakets zuzuordnen
  - `ftype` – identifiziert den Typ des Objektes
    - d – Verzeichnis
    - f – Datei
    - i – Informationsdatei
  - `class` – Installationsklasse zu der das Objekt gehört
  - `path` – Pfad des Objektes auf dem Zielsystem

Aufbau eines Pakets
prototype – Datei (3/5)

- Weitere Felder
  - `mode` – Modus des Objektes auf dem Zielsystem als Oktalzahl angegeben
  - `owner` – Eigentümer des Objektes auf dem Zielsystem
  - `group` – Gruppe des Objektes auf dem Zielsystem
- Hinweis:
  - `mode`, `owner`, `group` des Objektes können beibehalten werden, indem `?` als Wert angegeben wird
  - auch Variablen möglich

Aufbau eines Pakets
prototype – Datei (4/5)

- Zusätzliche Funktionen für eine prototype – Datei
  - `!include`
    - zusätzliche `prototype` – Dateien einbinden
  - `!default`
    - legt Standardwerte für `mode`, `owner` und `group` fest
  - `!search`
    - gibt Pfad für Paketobjekte an, wenn dieser von der Position auf dem Zielverzeichnis abweicht

Aufbau eines Pakets prototype – Datei (5/5)

- Beispiel

```
# Variable setzen
!PKGINFOPATH=~/bsda/condor

# Datei
f none BSDAcondor/bin/condor 0777 root root

# Verzeichnis
d none BSDAcondor/etc/ 0777 root root

# Informationsdatei
i pkginfo=$PKGINFOPATH/pkginfo-file
```

Aufbau eines Pakets
depend – Datei

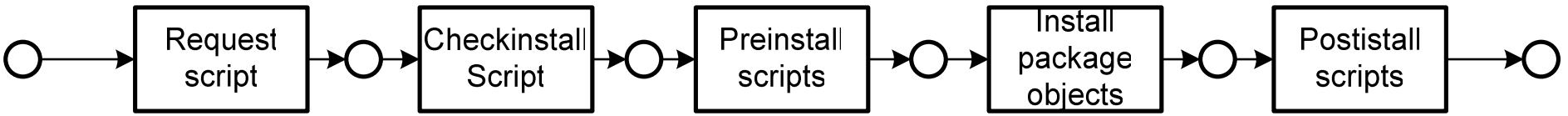
- Gibt Abhängigkeiten zu anderer Software an
- Format
  - `<type> <PKG> <NAME> [(<ARCH>) <VERSION>]`
  - `<type>` - Gibt die Art der Abhängigkeit an
    - `P` – Paket wird benötigt
    - `I` – Pakete die inkompatible sind
    - `R` – Umgekehrte Abhängigkeiten
- Beispiel
  - `P BSDAjoe joe (intel)`

Aufbau eines Pakets
Kontrollobjekte – Installationsskripte (1/3)

- Typ der Skripte
  - Bourne Shell Skripte (sh)
  - Dürfen nur Bourne Shell Commands und Text enthalten
  - Müssen nicht als ausführbar gekennzeichnet werden
  - Benötigen kein Identifierer !# /bin/sh

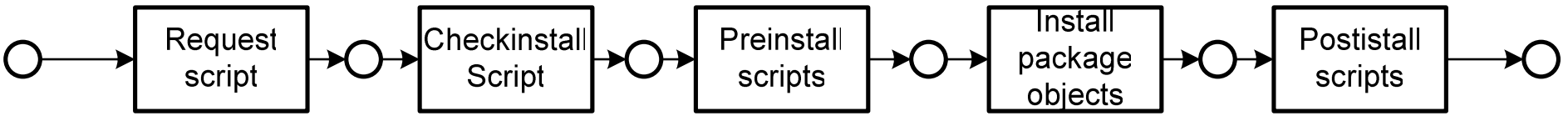
Aufbau eines Pakets
Kontrollobjekte – Installationsskripte (2/3)

- Installationsreihenfolge
  - 1. `request` Skript ausführen
  - 2. `checkinstall` Skript ausführen
  - 3. `preinstall` Skripts ausführen
  - 4. Erstellen von
    - a) named pipes, devices, symbolic links, required directories
    - b) normale Dateien anlegen
    - c) hard links
  - 5. `postinstall` Skripts ausführen



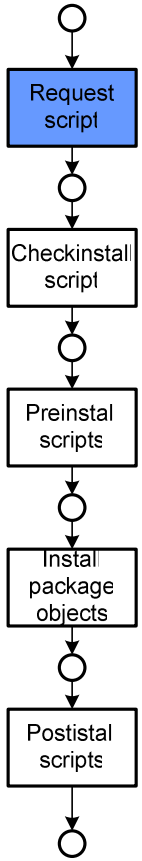
Aufbau eines Pakets
Kontrollobjekte – Installationsskripte (2/3)

- Installationsreihenfolge
  - 1. `request` Skript ausführen
  - 2. `checkinstall` Skript ausführen
  - 3. `preinstall` Skripts ausführen
  - 4. Erstellen von
    - a) named pipes, devices, symbolic links, required directories
    - b) normale Dateien anlegen
    - c) hard links
  - 5. `postinstall` Skripts ausführen



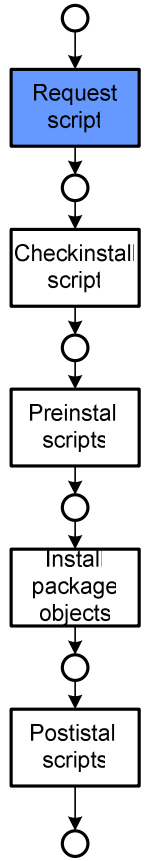
Aufbau eines Pakets
Request Skript (1/2)

- Einzigstes Skript um Interaktion mit dem installierenden Administrator zu bekommen
- Läuft nur wahrend der Installation, nicht beim Entfernen
- Ausgabe ist eine Liste von Umgebungsvariablen
- Kann keine Dateien modifizieren (Ausführung als user `install` / `nobody`)
- Ausführung durch `pkgadd`-Kommando



Aufbau eines Pakets
Request Skript (2/2)

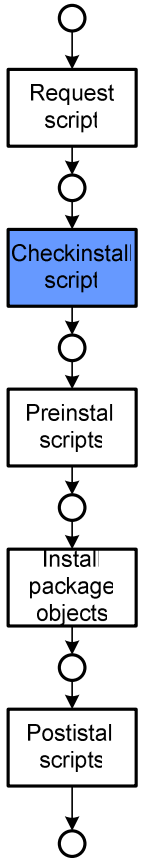
- Design-Regeln
  - Ein Skript pro Paket
  - Wird nicht unbedingt während der Installation ausgeführt
  - Administrator kann Antwortdatei erstellen, welches die Umgebungsvariablen setzt
  - In diesem Falle wird das Skript nicht mehr ausgeführt
  - Verwendete Variablen sollten in der `pkginfo` Datei mit Default-Werten belegt sein
  - Keine Umgebungsvariablen auf Basis des aktuellen Systems erstellen



Aufbau eines Pakets
Checkinstall Skript (1/2)

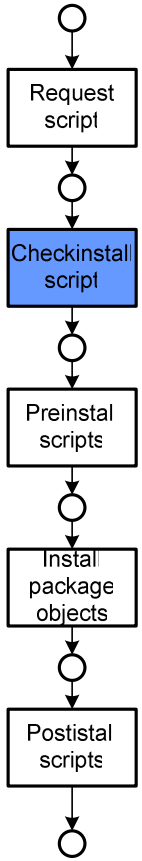
- Wird nach `request` Skript ausgeführt
- Läuft unter dem selben User wie das `request` Skript
- Kann keine Dateien verändern

- Geeignet für
  - Überprüfung, ob Dateien überschrieben werden
  - Abhängigkeiten überprüfen und behandeln
  - Hinweis: `depend` kann nur gesamte Pakete überprüfen



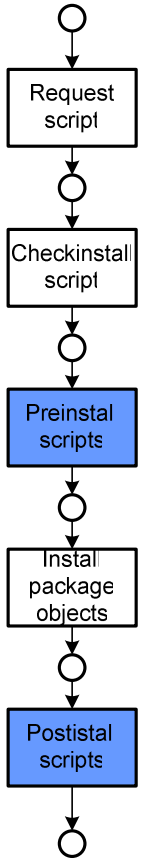
Aufbau eines Pakets
Checkinstall Skript (2/2)

- Design-Regeln
  - Ein Skript pro Paket
  - Verwendete Variablen sollten in der `pkginfo` Datei mit Default-Werten belegt sein
  - Keine Interaktion mit dem Administrator



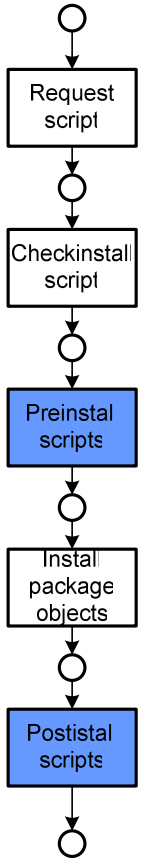
Aufbau eines Pakets
Procedure Skripte (1/2)

- 4 Procedure Skripte
  - `preinstall`
  - `postinstall`
  - `preremove`
  - `postremove`
- Werden als `uid=root` and `gid=other` ausgeführt



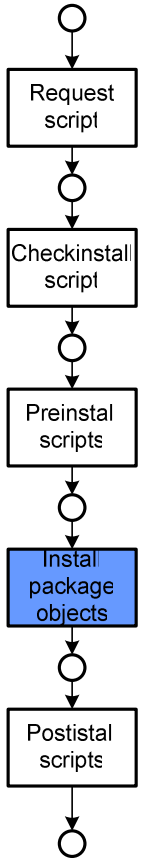
Aufbau eines Pakets
Procedure Skripte (2/2)

- Design-Regeln
  - Skript sollte mehrfach ausführbar sein
  - Keine Interaktion mit dem Administrator
  - Fremde Dateien mit `installf` installieren



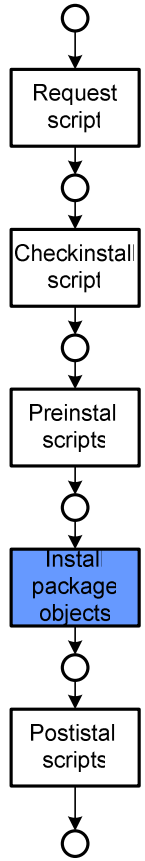
Aufbau eines Pakets
Class Action Skripte (1/3)

- Objekte werden in der `prototype` Datei einer `class` zugewiesen
  - `f none BSDAtest/src/testsrc 0777 root root`
- Liste der `CLASSES` in `pkginfo` bestimmen die Reihenfolge der Installation
- Aufbau der Namen für die Skripte
  - `i.class` (Paketinstallation)
  - `r.class` (Paketdeinstallation)
- Werden als `uid=root` and `gid=other` ausgeführt



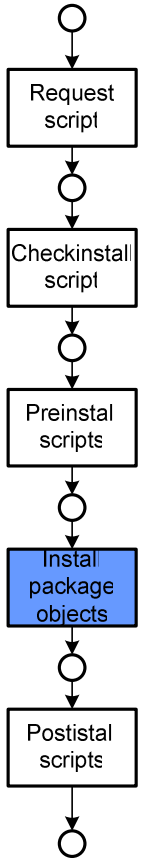
Aufbau eines Pakets
Class Action Scripte (2/3)

- Design-Regeln
  - Mehrfache Ausführung muss möglich sein (spanned Package)
  - Ist eine Datei einer CLASS zugeordnet, zu der es ein Action Script gibt, so muss das Action Script diese installieren (nicht pkgadd)
  - Sollte keine Veränderungen an Systemattributen machen
  - Keine Interaktion mit dem Administrator



Aufbau eines Pakets
Class Action Scripte (3/3)

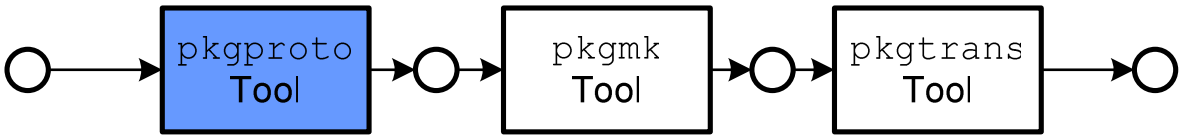
- Special System Classes
  - `sed` class
  - `awk` class
  - `build` class
  - `preserve` class



- Einleitung
- Aufbau eines Pakets
- Paketerstellung unter Solaris
  - Programme / Tools für die Paketerstellung
- Paketadministration
- Zusammenfassung
- Literatur

Tools zur Paketerstellung - pkgproto

- Erzeugt `prototype` – Dateieinträge als Eingabe für `pkgmk` – Tool
- Parameter:
  - `-i` – ignoriert symbolische Links und erzeugt Einträge mit ftype=f
  - `-c class` – setzt alle Einträge mit der angegebenen Klasse
- Operanden
  - `Pfad` – nutzt den angegebenen `Pfad` als Eingabe
  - `Pfad1=Pfad2` – ersetzt `Pfad1` im Output durch `Pfad2`



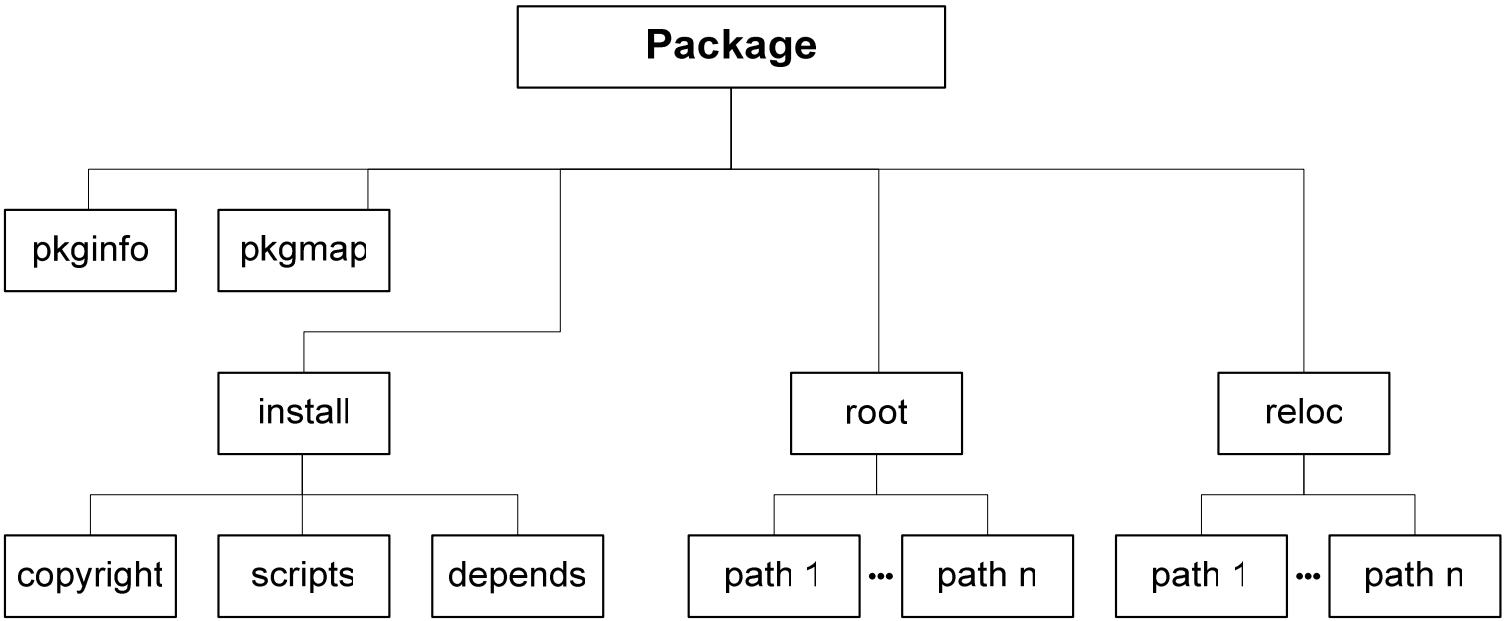
Tools zur Paketerstellung - pkgmk (1/2)

- `pkgmk` nimmt als Eingabe eine `prototype` Datei und erzeugt
  - ein installierbares Package in Ordnerstruktur
  - `pkgmap` Datei
- Jeder Eintrag in der `prototype` Datei wird an die entsprechende Stelle im angegebenen Verzeichnis kopiert
- Rückgabewert ist
  - = 0, wenn das Paket erfolgreich erstellt wurde
  - > 0, sonst

Tools zur Paketerstellung - pkgmk (2/2)

- Parameter
  - `-f prototype` – explizite Angabe einer prototype Datei
  - `-d device` – Angabe des Ausgabeverzeichnisses
  - `variable=value` – Belegung von Paketumgebungsvariablen

Tools zur Paketerstellung - Ordnerstruktur eines Pakets



Tools zur Paketerstellung - pkgmap - Datei

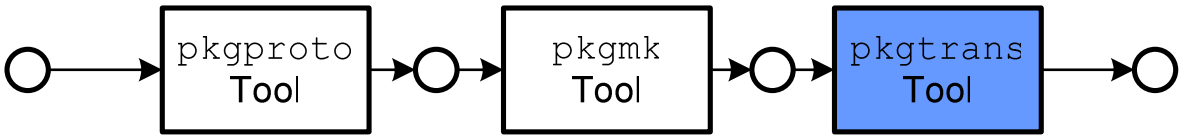
- `pkgmap` – Datei ersetzt die prototype Datei

- Beispiel

```
: 1 17658
1 f none /tmp/README 0644 root root 14430 63618 537826629
1 d none BSDAjoe 0755 root root
1 d none BSDAjoe/bin 0755 root root
1 f none BSDAjoe/bin/jmacs 0755 root root 1708164 61899 537826610
1 f none BSDAjoe/bin/joe 0755 root root 1708164 61899 537826612
1 f none BSDAjoe/bin/jpico 0755 root root 1708164 61899 537826615
```

Tools zur Paketerstellung - pkgtrans

- Übersetzt ein installierbares Paket von einem Format in ein anderes Format
  - Dateisystem-Format ↔ Datenstrom-Format
  - Dateisystem-Format ↔ Dateisystem-Format
- Parameter
  - `-s` – Erzeugt ein Datenstrom-Format
- Operanden
  - `device1 device2` – Quell- und Zielverzeichnis



- Einleitung
- Aufbau eines Pakets
- Paketerstellung unter Solaris
- Paketadministration
  - Programme / Tools zur Paketverwaltung
  - Paketsignierung
- Zusammenfassung
- Literatur

Paketadministration
Tools zur Paketverwaltung

- `pkgadd` – Fügt Paket zum System hinzu
  - `-d pkgname` (Angabe des Pfades zum Paket)
  - `-r resposefile` (Angabe des Pfades zum Antwortfile)
  - `-x proxy` (http-proxy zum downloaden von Paketen)

- `pkgrm` – Entfernt ein Paket vom System
  - `pkgname`

Paketadministration
Signierung von Paketen (1/4)

- Normales Paket mit einer digitalen Signatur
- Es lässt sich damit überprüfen
  - Wer das Paket signiert hat und wo es herkommt
  - Ob Paket geändert wurde
  - Das der signierten Person getraut werden kann

- Benötigt:
  - User Key (z.B. von Verisign)
  - Trusted Certificate

Paketadministration
Signierung von Paketen (2/4)

- Ein signiertes Paket ist binärkompatibel zu einem unsignierten Paket

- Es wird ein „package keystore“ benötigt
  - $HOME/.pkg/security

Paketadministration
Signierung von Paketen (3/4)

- Typischer Ablauf
  1. Erstellen eines normalen Pakets
  2. Importieren der digitalen Signaturen in des Keystore
  3. Paket signieren

Paketadministration
Signierung von Paketen (4/4)

- Kommandos:
  - Hinzufügen von “Trusted Certificates”

    ```
    pkgadm addcert -t mytrustedcert.pem
    ```
  - Hinzufügen eines User Certificate und Private Key

    ```
    pkgadm addcert -n myname -e myprivkey.pem
    mypubcert.pem
    ```
  - Anschauen der geladenen Key’s

    ```
    pkgadm listcert
    ```
  - Entfernen des geladenen Certificate

    ```
    pkgadm removecert -n myname
    ```
  - Signatur einem Paket zuordnen

    ```
    pkgtrans -g -k ~/mykeystore -n mycert .
    ./BSDpackage.signed BSDpackage
    ```

Paketadministration
Verifizierung von Paketen

- Überprüft, ob Paketintegrität stimmt
- Vergleicht
  - Dateigröße
  - Zugriffsrechte
- Format
  - `pkgchk`
- Fünf benötigte Parameter
  - `-v alle Infos ausgeben`
  - `-d paketstream kontrollieren`

Zusammenfassung

- Solaris bietet eine Vielzahl an Werkzeugen zur Erzeugung und Administration von Paketen
  - Gute Strukturierung der Programme
  - Programme sind einfach in der Bedienung
- Es gibt viele Möglichkeiten Einstellungen vorzunehmen
- Es war sehr interessant und lehrreich sich mit der Thematik auseinanderzusetzen

Literatur

- Homepage von SUN Microsystems
  http://docs.sun.com/app/docs/prod/solaris.10
- Condor-Projekt Hompage
  http://www.cs.wisc.edu/condor
- man-pages von Solaris 10
- Joe-Projekt Homepage
  http://sourceforge.net/projects/joe-editor