Yocto Workshop

vom 5.5.2017

Prof. Dipl.-Ing. Klaus Knopper
<knoppix@knopper.net>

Gliederung

1. Prerequisites:
    1. GNU/Linux-Entwicklungsumgebung (Host), Arbeiten mit der Shell
    2. GNU/Linux-Softwaremanagement
    3. Distributionen, Archive und Repositories

2. Yocto
    1. Sinn und Zweck
    2. Aufbau
    3. Installation + Dokumentation
    4. Layer-Prinzip
    5. Recipes (Rezepte), Includes und Klassen

3. Übungen zu yocto

Skills
required/recommended
for yocto

- Grundlagen Unix/Linux Kommandozeile (z.B. Bash)
- Dateisysteim-Navigation (s. Folie „Navigieren im Dateisystem mit der Shell“)
- Softwaremanagement auf dem Host-System (z.B. dpkg/apt für Debian/Ubuntu, rpm für Fedora/SuSE)
- Repositories für Upstream-Software: git/svn
- Beherrschen eines Text-Editors (vi, emacs, leafpad, ...)
- Hilfreich: Programmierkenntnisse in bourne-Shell (f. eigene Build-Prozeduren in yocto-Rezepten)

Host System

- Das yocto-Buildsystem setzt ein GNU/Linux-System voraus (besser: „empfiehlt“) mit mindestens den grundlegenden Entwicklungstools
  - Python
  - C-Compiler
  - Git (aktuelle Version)
- Weitere Entwicklungswerkzeuge werden heruntergeladen, compiliert und in einem host-nativen sysroot-Verzeichnis installiert. Hierzu gehört auch der passende Cross-Compiler für die Zielarchitektur.
- NFS-Server für diskless boot (nfsroot) und `minicom` (serielle Konsole) sind für Live-Tests sinnvoll.

Warum Shell?

- Direkte Mensch-Maschine-Kommunikation.
- Keine Missverständnisse durch falsch interpretierte Symbole (Icons).
- Direktausgabe von Fehlermeldungen ohne Interpretation durch GUI.
- Auch Systemfehler (Programm startet nicht wegen falscher Library) können erkannt werden.
- Nachteil: Kommandos müssen teilweise auswendig beherrscht werden (Online-Hilfe mit **`apropos Stichwort`**, **`man Kommando`**, Kommandoergänzung mit TAB-Taste)

Remote Kommandozeile
Remote Desktop

- **ssh benutzer@rechner** startet zunächst eine Shell für den Benutzer „benutzer“ auf dem Zielsystem.
- SSH unter Linux erlaubt auch den Direktstart graphischer Programme, die auf die eigene Desktop-Oberfläche „getunnelt“ werden. Voraussetzung ist ein laufender X-Server (Linux Desktopsystem).
- Mit entsprechenden public/private ssh keys (ssh-keygen) ist ein sicheres „passwortloses“ Login möglich.
- Desktop-Projektion bzw. Starten eines virtuellen Desktop per `tightvncserver` ist möglich (VNC). Für RDP-Clients (Windows) kann entsprechend ein RDP-Server installiert werden:
  `sudo apt-get install xrdp`

Navigieren im Dateisystem mit der Shell

pwd
Ausgabe aktuelles Arbeitsverzeichnis

cd Verzeichnisname
Wechsel des aktuellen Verzeichnisses

cat Datei...
Inhalt von Dateien lesen/ausgeben

ls -l [wildcards]
Ausführliches Auflisten von Dateien

mkdir [-p] Verz.
Lege [mit Unterverz.] Verzeichnis an.

cp [-a] Alt Neu
Kopiere [Klone Alles] von Alt nach Neu

mv Alt Neu
Benenne Alt nach Neu um

rm [-rf] wildcard
Lösche unwiderruflich [rekursiv forciert]

Speicherkapazität

df
Ausgabe der eingebundenen Dateisysteme mit "Füllstand"

mount
Ausgabe der eingebundenen Dateisysteme mit Optionen

du -s[km] [wildcards]
Aufsummieren der Verzeichnis-Inhalte [in Kilobytes/Megabytes]

free
Anzeige RAM- und Swap-Auslastung (zweite Zeile OHNE Cache ist relevant!)

cat /proc/meminfo
detaillierter als „free“

Übung zur Shell-Navigation

(1) Wechseln Sie in Ihr Heimverzeichnis. Wo befindet sich dies im Dateisystem?
(2) Erzeugen Sie einen Ordner **c**, der sich im Ordner **b** befindet, der sich im Ordner **a** befindet.
(3) Kopieren Sie die Datei **/etc/passwd** in den Ordner „**c**“ aus der vorigen Aufgabe.
(4) Editieren Sie die Datei **a/b/c/passwd** (z.B. mit **leafpad a/b/c/passwd**)
(5) Kopieren Sie die Datei **a/b/c/passwd** wieder nach **/etc/**
(Fehlermeldung ist korrekt!)
(6) Löschen Sie den Ordner **a** mit Inhalt.

Advanced: Dateien finden

```
find Verzeichnisse... [-iname 'Dateimuster']
                      [weitere Kriterien...]
```

z.B. alle Dateien finden, die nicht älter sind als 10 Tage:

```
find yocto/poky -type f -mtime -10
```

z.B. alle Dateien LÖSCHEN, die mit „**.tmp**“ enden:

```
find yocto/poky -type f -iname '*.tmp' \
                -exec rm -vf {} \;
```

Advanced: Ausgabe eines Kommandos filtern (grep)

- `dpkg -l | grep -i compiler`
- `ps auxwwww | grep bitbake`
- `grep -i 'error.*in.*file' make.log`

Permissions (in Kurzform)

- Rechte: Lesen (**r**), Schreiben (**w**), Ausführen/Betreten (**x**) ODER User-/Group-ID während Ausführen oder Betreten ändern (**s**)
- Rechte-Empfänger: Besitzer/User (**u**), Gruppe (**g**), Andere/Others (**o**)
- Ausgabe in Reihenfolge **rwx** (u) rwx (g) **rwx** (o) bei **ls -l**, fehlende Rechte werden mit **–** gekennzeichnet.
- Setzen z.B. mit **chmod u=rwx,g=rwx,o=rx datei**
- Besitzer / Gruppe einer Datei ändern nur mit entsprechenden Rechten: **chown username datei**, **chgrp username datei**
- Eigene Zugehörigkeit: **id**

Übung zu Permissions

Programme unter Unix-Systemen compilieren und installieren (1)

- Früher (klassisch):
  1. source.tar.gz downloaden,
  2. tar zxvf Source.tar.gz; cd Source
  3. make
  4. sudo make install
- Problem: Abhängigkeiten beim Compilieren und Laufzeitbibliotheken müssen zuvor untersucht und installiert werden.
- Problem: Vorhandene Libs und Programme können übschrieben werden.
- Problem: Update, Downgrade, Deinstallation und Anpassung von Konfigurationsdateien?

Programme unter Unix-Systemen compilieren und installieren (2)

- Heute: **Software-Paketmanagement**
- Datenbank für installierte Programme: Dateien, Abhängigkeiten, gemeinsam genutzte Daten
- Datenbank für verfügbare Programme: Dateien, Abhängigkeiten, Ersetzen oder Alternativauswahl
- Migration von Konfigurationsdaten bei Up- oder Downgrades
- Verhindern von Laufzeit-Problemen durch Markierung von Konflikten
- Teilweise Parallelinstallation verschiedener Versionen möglich (wichtig bei Bibliotheken)
- Problem: Zusätzlicher Speicherbedarf für Software-Datenbank oft zu groß für Target-System mit kleinem Flash-Speicher.

Programme unter Unix-Systemen compilieren und installieren (3)

- Debian, Ubuntu (deb), SuSE, Fedora (rpm), ...:
  Pakete sowohl für Binärinstallationen, als auch Source-Pakete zum Selbstcompilieren:

  `apt-get source paketname`
  `cd paketname-version`
  `dpkg-buildpackage` (erzeugt Binärpaket .deb und neues Source-Paket mit Patches)

- „Linux From Scratch“: Aus Sourcen wird ein optimiertes System ohne Redundante oder nicht gewünschte Komponenten erstellt. → Ziel und Vorgehensweise bei **ptxdist** und **yocto** ähnlich!

Software-Auswahl und Konfiguration anpassen (Debian/Ubuntu)

- Installations-/Konfigurations-Kommando als root ausführen:
  `sudo Kommando`
- Softwarepakete aus Debian/Ubuntu-Repository nachinstallieren (root-Kommandos schräggestellt):

Kommando	Wirkung
`apt-get update`	SW-Datenbank aktualisieren
`apt-get upgrade`	(VORSICHT!) Komplette System-Software aktualisieren
`apt-cache search Stichwort`	Software suchen
`apt-cache show paketname`	Details anzeigen
`apt-get install paketname`	Softwarepaket installieren oder aktualisieren

Apt Proxy

Durch Setzen einer Umgebungsvariablen

```
export http_proxy=http://referent-ip:9999
```

bzw.

```
export ftp_proxy=http://referent-ip:9999
```

können Programme wie **apt-get** in der aktuellen Shell angewiesen werden, einen Proxy/Cache zu verwenden. Dies spart Bandbreite, wenn ein solcher Proxy als Cache für größere Downloads bereitgestellt wird.

Lokal: dpkg (Debian/Ubuntu)

Kommando	**Wirkung**
`dpkg -l [paketname]`	SW-Datenbank [Paket] auflisten
`dpkg -i paketdatei.deb`	Software aus Datei installieren oder aktualisieren
`dpkg -s paketname`	Ausführliche Information über installiertes Paket ausgeben
`dpkg -c paketdatei.deb`	Inhalt der Paketdatei anzeigen
`dpkg --purge --force-all paketname`	Installiertes Paket und seine Konfigurationsdateien unter Missachtung der Abhängigkeiten deinstallieren (Vorsicht!!!)

Übung zu Software-Management

Distributionen, Archive, Repositories (1)

- Eine „Distribution“ ist eine Zusammenstellung (**Sammlung**) von System- und Anwendersoftware (Begriff meist für Linux-Systeme gebräuchlich, prinzipiell auch auf Mac OS / Windows anwendbar).
- Die meisten großen Distributionen haben einen kommerziellen Hersteller, der auch Support anbietet. Ausnahme: Debian (größtes GNU/Linux Derivat, ausschließlich Community-getragen, Basis für Ubuntu und Knoppix).
- Inhaltlich unterscheiden sich die Distributionen kaum, es wird aber unterschiedlicher Fokus auf Zielpublikumg und kommerzielle Anwendungen gelegt (Spezialisierung).

Distributionen, Archive, Repositories (2)

- Die meisten GNU/Linux-Distributionen bestehen aus einem Hauptteil, der Open Source Lizenzen benutzt und frei verteilbar ist, und einem proprietären Teil, der den „Mehrwert“ und die Abgrenzung gegenüber Wettbewerbern darstellen soll. GPL erlaubt die lose Kopplung (muss aber trennbar sein!) von Open Source und proprietärer Software.
- Nicht erlaubte Kombination: GPL-lizensierte Software als untrennbare Funktions-Basis einer proprietären Anwendung („Proprietarisierung“).
- Erlaubte Kombination: Proprietäre Anwendung läuft auf einem GPL-Lizensierten Betriebssystem.
- Erlaubte Kombination: BSD/MIT-lizensierte Komponente in einem proprietären Programm.

Distributionen, Archive, Repositories (3)

- Distributoren stellen aus eigenem Interesse kostenlose Updates und Erweiterungen auf öffentlich verfügbaren Servern („Repositories“) zur Verfügung.
- Es kommen unterschiedliche Container-Formate (Archive) zum Einsatz, z.B. .deb („ar“-Format) oder rpm („cpio“-Format), die auch Installationsskripte, Kompatibilitätshinweise und Abhängigkeitsinformationen enthalten.
- „From Scratch“-Distributionen oder „Meta-Distributionen“ beginnen oft mit einem Quelltext-Archiv mit Bauanleitung, das so wenige Abhängigkeiten wie möglich erfordert. → yocto „Meta-Distribution“ zum Erstellen von Cross-Plattform Distributionen.

yocto

https://www.yoctoproject.org/

About yocto - Summary

- Collaboration – Zusammenarbeits-Werkzeug
- yocto: 2010 als cross-plattform Projekt in Zusammenarbeit mit „den großen embedded Hardware-Herstellern“ von der Linux Foundation gegründet, Chief Architect: Richard Purdie.
- Verwendet Open Embedded („***bitbake***“) als Kern.
- „It's not an embedded Linux distribution – it creates a custom one for you“ → Stimmt, aber das Motto erzeugt eine hohe Erwartungshaltung (vor dem suggerierten „automatischen Erstellen“ ist natürlich einiges an Arbeit notwendig).

poky → *ist Referenzimplementation von* → Projekt **yocto** → *benutzt* → **OpenEmbedded** → *basiert auf* → **bitbake**

About yocto - Vererbung

- **Poky**

  → *ist Referenzimplementation von* →

- **Projekt yocto**

  → *benutzt* →

- **OpenEmbedded**

  → *basiert auf* →

- **bitbake**

yocto – Sinn und Zweck

- Bietet teilfertige **Vorlagen** für verschiedene Aufgaben in Form von Klassen und Rezepten
- Sichert **Reproduzierbarkeit** des Software-Builds, indem Cross-Compiler und andere Build-Tools in einer einheitlichen Versionierung zunächst heruntergeladen, compiliert, und in einem **nativen sysroot** Verzeichnis installiert werden.
- Rezepte **lösen Abhängigkeiten** und **korrekte Reihenfolge** von Build und Installation.
- Sehr **modular**, entkoppelt oder koppelt Builds von **Komponenten**.
- Stellt Bauvorlagen von **Images**/**Produkten** zur Verfügung.
- **Caching** von heruntergeladenen, jedoch unveränderten Sourcen.

→ Das Aktualisieren und „Durchbauen“ der Software und Erzeugen installationsfähiger Images soll **vereinfacht** werden und auf verschiedenen Host-Plattformen **einheitlich funktionieren** (Distributionsunabhängig).

Yocto - Trivia

Was bedeutet eigentlich „yocto“?

$$10^{-24}$$

Idee dahinter: „Elementarteilchen“ bzw. „Kleinster Gemeinsamer Nenner“? Interoperabilität?

→ https://www.yoctoproject.org/question/how-did-yocto-project-get-its-name

Übung: yocto installieren (.tar.xz)

- Hinweis: xz ist ein relativ neues Kompressionsverfahren, das v.a. bei Binaries effizienter und schneller ist als gzip oder bzip2. Linux-Kernels und Patches werden auf kernel.org im .xz-Format angeboten.

- Auspacken von tar.xz-Archiven:
  tar -Jxvf datei.tar.xz
  (d.h. tar-Option „J“ statt „z“)
  Inhalt des Archivs landet im aktuellen Verzeichnis.

(1) Entpacken Sie die Datei
`/mnt-system/poky-daisy-11.0.1.tar.xz`
in Ihr Heimverzeichnis.

(2) Benennen Sie den Ordner `poky-daisy-11.0.1` um in `poky`

yocto - Aufbau

Das yocto-Buildsystem ist nach Auschecken per GIT oder Download/Entpacken des tar-Archivs noch recht klein:

bitbake (Python):	10MB
build (fast leer):	1MB
documentation:	13MB
meta:	25MB
meta-selftest:	1MB
meta-skeleton:	1MB
meta-yocto:	1MB
meta-yocto-bsp:	1MB
scripts:	4MB
Makefile, oe-init-build-env,...	<1MB

Yocto Docs

yocto/poky/documentation

- adt-manual - The Yocto Project Application Developer's Guide.
- bsp-guide - The Yocto Project Board Support Package (BSP) Developer's Guide
- dev-manual - The Yocto Project Development Manual
- Kernel-dev - The Yocto Project Linux Kernel Development Manual
- ref-manual - The Yocto Project Reference Manual
- ***yocto-project-qs - The Yocto Project Quick Start***
- mega-manual - An aggregated manual comprised of all YP manuals and guides
- profile-manual - The Yocto Project Profile and Tracing Manual

Yocto Docs

yocto/poky/documentation

Übung: yocto-Dokumentation compilieren

(1) **cd** ins Dokumentations-Verzeichnis
(2) **make DOC=yocto-project-qs**
(3) **firefox yocto-project-qs/yocto-project-qs.html &**
(4) **make DOC=ref-manual pdf html**
(5) **okular ref-manual/ref-manual.pdf &**

yocto - Laufzeitgröße

- Nach einem Build von `core-image-sato` liegen im build-Verzeichnis >= 48GB Daten (v.a. ausgepackte Sourcen und compilierte Object Files).

- Tipp: Abhilfe gegen zu hohen temporären Platzverbrauch durch Angabe von

  `INHERIT += "rm_work"`

  in **`build/conf/local.conf`**. Hierdurch werden Sourcen und temporäre Build-Ordner nach erfolgreichem Ablauf eines Tasks sofort gelöscht. **Nachteil:** Fehlersuche in ausgepackten Sourcen ist aufwändiger, „Live-Patches“ nicht mehr möglich (z.B. bei hardwarespez. DTS-Patch-Generierung).

Übung: yocto-config anpassen

(1) Ausführen: **. oe-init-build-env**

(2)Editieren Sie im poky/build-Ordner die Datei
conf/local.conf
und fügen Sie am Ende die folgenden Zeilen hinzu:

```
INHERIT += "rm_work"

SOURCE_MIRROR_URL ?=
"file:///build/yocto_external_source/"

INHERIT += "own-mirrors"

BB_GENERATE_MIRROR_TARBALLS = "1"

BB_NO_NETWORK = "1"

ACCEPT_FSL_EULA = "1"
```

Übung: Externe Sourcen per NFS einbinden

(1) Erzeugen Sie den Ordner
/build/yocto_external_source
Hierzu müssen Sie ggf. das Kommando zum Erzeugen von Verzeichnissen als root aufrufen.

(2) Binden Sie den Sourcen-Ordner vom Referentenrechner als /build/yocto_external_sources in Ihr System ein (alles in einer Zeile!):

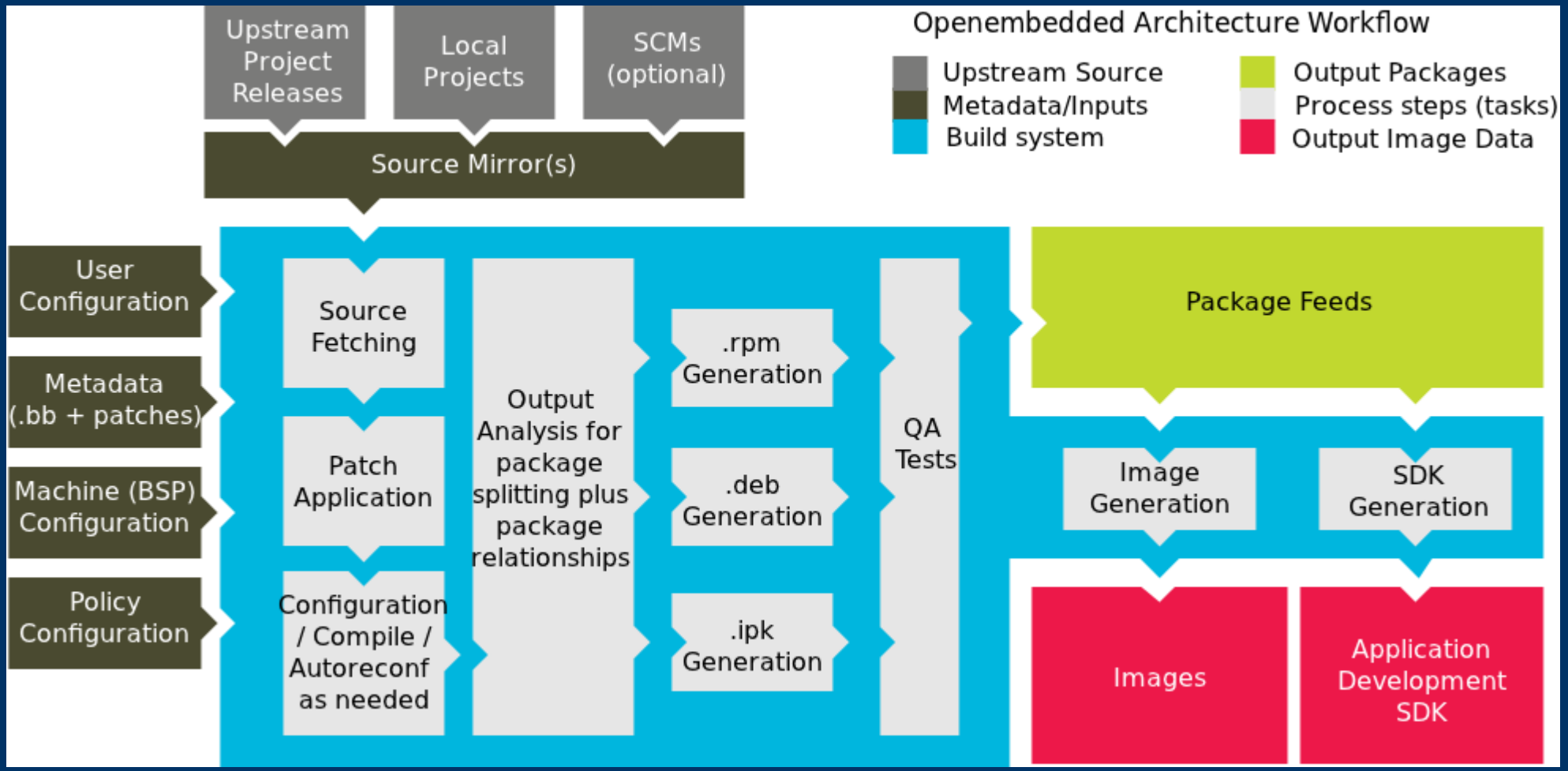
```
sudo mount -o nolock,tcp,ro \
     ip-referent:/build/yocto_external_source \
     /build/yocto_external_source
```

Tipps:

- „history“ zeigt die zuletzt eingegebenen Kommandos an (persistent) → „automatische Dokumentation“
- Was bedeutet
  `. ./oe-init-env` ?
  . = Führe den Inhalt des Skriptes so aus, als wären die Zeilen direkt hier eingegeben worden. Dabei werden auch Variablen gesetzt und Anweisungen in der AKTUELLEN Shell ausgeführt.
  Im Gegensatz dazu wird bei
  `./oe-init-env`
  eine NEUE Shell gestartet, bei der nach Beenden alle Variablen verschwinden.
  (source ./oe-init-env geht auch.)

yocto - Workflow

fetch → unpack → patch → configure → compile → install
→ target-sysroot+package → (qa tests) → image → ...



yocto – Weitere downloadbare Komponenten

- HOB - GUI für yocto
- Eclipse Plugin ADT – Application Development Toolkit
- EGLIBC (enthalten) – Embedded Version der GNU LIBC
- Matchbox - GUI für Tests auf Embedded Devices
- Build Appliance - Virtuelle Maschine mit yocto-Buildsystem geeignet zur Ausführung auch auf Nicht-Linux Plattformen (z.B. Windows)

Achtung:
Architekturproblem 32bit vs. 64bit

- Sehr speziell, aber kommt vor (z.B. bei Knoppix): Host-System benutzt 64bit Kernel, Userspace jedoch 32bit (Kompatibilitätsmodus).
- **`uname -m`** → „x86_64“
  aber
  `file -sk /bin/bash` → „ELF 32-bit LSB executable“
- Ergebnis: yocto erkennt „64bit host“ und baut native Binaries in 64bit, scheitert damit aber, weil Userspace und Libraries durchgehend 32bit.
- Abhilfe: Ausgabe des „**`uname`**“-Kommandos verändern:
  `setarch i386 /bin/bash`
  (Gilt NUR für die laufende Shell!)

Software mit apt-get installieren

- Ergänzung: Folgende Pakete müssen für yocto noch installiert werden:

```
export http_proxy=http://10.0.0.1:9999
sudo apt-get update
sudo apt-get install -t testing chrpath diffstat
libsdl1.2-dev
```

Alternativ:

```
sudo http_proxy=http://10.0.0.1:9999 apt-get update
sudo http_proxy=http://10.0.0.1:9999 apt-get install -t testing chrpath
diffstat libsdl1.2-dev
```

yocto Big Test

(1) Eigene Distribution als „tested“ deklarieren:
Datei **meta-yocto/conf/distro/poky.conf** editieren,
eigene Distro (hier: **Debian-7.6**) in **SANITY_TESTED_DISTROS** ergänzen.

(2)Im Verzeichnis yocto/poky:
. ./oe-init-build-env
(Wechselt automatisch ins „build“-Verzeichnis

(3) Anpassen conf/local.conf (Zielarchitektur z.B. qemux86)

(4) bitbake core-image-sato
oder
bitbake core-image-minimal

(5) Testlauf in virtueller Maschine:
```
for i in amd intel; do sudo modprobe kvm_$i; done
runqemu qemux86
```

Tipp: Systemlast während Build verringern

`nice -20 kommando`
(CPU-Priorität des Schedulers verringern)

`ionice -c 3 kommando`
(IO-Scheduler vergibt NUR „idle“-Resourcen an kommando)

Kombination der beiden, z.B. bei „**bitbake**“:

`nice -20 ionice -c 3 bitbake core-image-minimal`

Layer-Technik

- Yocto organisiert aufeinander aufbauende Sourcen und Rezepte in Layern (Ebenen). Später hinzugefügte Ebenen sollten in der Suchpriorität höher stehen.
- Angabe der Reihenfolge der Layer in **build/conf/bblayers.conf**
- Die Layer liegen als Verzeichnisse mit (Konvention) Präfix „**meta-**“ im poky-Verzeichnis. Sie enthalten die ggf. aufeinander aufbauenden Klassen und Rezepte, ggf. auch eigene Sourcen, Patches und Tools, wenn diese nicht per Anweisung aus externen Quellen (git, SVN, http-Download, ...) heruntergeladen werden sollen.

Klassen, Includes und Rezepte

- Innerhalb der meta-* Layer-Verzeichnisse werden „Rezepte“ zum Bauen von Softwarekomponenten, Installation und Integration ins Image in Unterverzeichnissen mit (Konvention) Namen „recipes-name“ abgelegt. Sie tragen die Endung .bb, und werden unter Berücksichtigung des Suchpfad-Variablen BBFILES aus meta-name/conf/layer.conf automatisch von bitbake gefunden (**.bb** = **b**it**b**ake Rezept)
- Klassen (`klasse.bbclass`) enthalten Definitionen, die an Rezepte mit **`inherit klasse`** „vererbt“ und erweitert werden können.
- Includes (`datei.inc`) werden wie Makros eingebunden („in-place“).

Variablen

- Informationen können zwischen Konfigurations-Dateien, Klassen, Includes und Rezepten mit Hilfe von Variablen übertragen werden. Hierbei ist zu beachten:
  - Nur die in Rezepten und Klassen mit „**export**“ deklarierten Variablen sind global und auch im Shell-Environment sichtbar.
  - Andere definierte Variablen sind in direkt erbenden Dateien sichtbar, werden aber nicht an Funktionen übermittelt.
  - Sehr wenige Variablen (z.B. **${WORKDIR}**) sind standardmäßig global. Z.B. der Source-Ordner wird nicht als Variable übermittelt.
    → Von Skripten benötigte Variablen am besten schon in den **.conf**-Dateien mit deklarieren! (z.B. Signatur-Keys)

To be continued...