T-Head Yeying1520
Yocto User Guide

Revision	V1.0.0
Security	Secret
Date	Sept-16-2022

平头哥上海半导体技术有限公司 T-HEAD Semiconductor Co., LTD

地址: 中国(上海)自由贸易试验区上科路 366 号、川和路 55 弄 2 号 5 层

网址: www.t-head.cn

Revisions

Rev	Description	Author(s)	Date
V1.0.0	First draft	T-Head	Sept-16-2022

Contents

Figures & Tables

List of Abbreviations

Abbreviations	Full Spelling	Chinese Explanation

1 Overview

This document introduces the basic usage of Yocto and guides users to complete daily development based on Yocto.

2 Compile Environment

The Yocto compile environment uses Ubuntu 18.04. It is recommended to use Linux + docker for deployment. If docker cannot be used due to specific reasons, please refer to the Ubuntu environment chapter.

2.1 Docker Environment

Please install the basic Linux OS first. Users can choose Ubuntu, Centos, and other Linux releases according to their needs. The specific installation method will not be introduced in detail in this document. After the installation is complete, continue to install docker on the basis of this OS, then build a docker image, and build the relevant development environment in docker. The subsequent development is based on the environment in docker. The specific build method is as follows.

2.1.1 Install Docker

Use the official installation script to install automatically.

```
Bash | 复制代码
1  curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun
```

2.1.2 Download Dockerfile

Click to download linux-dev-master.7z, and enter the linux-dev-master directory after decompression. Open the dockerfile, and modify the user name and ID. Change “your the same user name asyour host” to the user name of your host OS.

```
Bash | 复制代码
1  ENV DOCKER_USER2 "your the same user name asyour host"
```

The value of “your user id” is recorded in /etc/passwd. After opening it, search for the line where your user name is located. The value of the anonuid field is the corresponding ID.

```
ENV USER2_ID "your user id"
```

2.1.3 Build Image

Run the following command to build your own environment:

docker build -t linux-dev-base:base

This docker image can compile Linux SDKs such as buildroot and yocto released by thead. The default password is 123.

To view the built docker image, you can see the following results:

```
@:~$ docker images
REPOSITORY       TAG       IMAGE ID       CREATED          SIZE
linux-dev-base   base      32207ad97b7a   12 minutes ago   2.13GB
ubuntu           18.04     886eca19e611   2 weeks ago      63.1MB
```

2.1.4 Start Docker

To start docker, use the docker run command.

```
docker run -u thead -dt --name linux-dev-{your_name}  -v
{your_lock_home}:{your_home} linux-dev-base:base /bin/bash
```

Note: You can mount one or more directories of the host through the -v option, which acts like a shared file, where: your_name: the name of the docker container, use your own name, and do not have the same name with others; your_lock_home: host local path; your_home: mount path of local path in docker。

Query the Started Docker Query:

```
docker ps |grep linux-dev-base
```

Normally you can see:

```
@~$ docker ps |grep linux-dev-base
CONTAINER ID   IMAGE                 COMMAND       CREATED
STATUS         PORTS     NAMES
017e0217cab0   linux-dev-base:base   "/bin/bash"   3 minutes ago   Up 3
minutes   22/tcp    linux-dev
```

2.1.5 Login to Docker

Run the following command to log in to docker.

```
docker exec -it linux-dev-{your_name} /bin/bash
```

At this point, you have logged in to Ubuntu 18.04 in docker via ssh, and run the following command to query the system version:

```
thead@017e0217cab0:/$ lsb_release -a No LSB modules are available.
Distributor ID: Ubuntu Description:    Ubuntu 18.04.6 LTS Release:
 18.04 Codename:       bionic
```

In the Start Docker section, the host local $HOME directory and the docker guest $HOME are mapped, so in the docker home/thead directory, you can see all the content in the host $HOME directory. All the files in that directory in docker are fully synchronized with host and are unaffected by docker shutdown or deletion.

2.1.6 Image Migration

In some environments, due to network restrictions and other reasons, it is impossible to build a complete docker image from scratch. At this time, you can use the docker save and docker load commands to complete the image

migration process. Build an image on a PC with an unrestricted network and then migrate it to the target machine. The specific steps are to save the image as a compressed package first, and then load the compressed package in another location.

Execute the following command on a host with an unrestricted network to generate a docker image.

```Bash
cd linux-dev-master docker build -t linux-dev-base:base .
```

View the generated docker image and confirm that the image is generated.

```Bash
docker images linux-dev-base:base
```

Start saving image.

```Bash
docker save  linux-dev-base:base| gzip >linux-dev-base:base.tar.gz
```

After this step, linux-dev-base:base.tar.gz can be generated in the current directory where the command is executed.

Copy the tar package linux-dev-base:base.tar.gz to the target machine, and then use the docker command to import image.

```Bash
docker load -i linux-dev-base\:base.tar.gz
```

At this point, you can see the image on the migration target machine.

docker images linux-dev-base:base

After the migration is complete, you can start the docker container, for example:

```
Bash | 复制代码
1  sudo docker run --network=host -u $user -dt --name $user_docker-ubuntu18
   -v /home/$user:/home/$user linux-dev-base:base /bin/bash
```

Then execute:

```
Bash | 复制代码
1  sudo docker exec -it $user_docker-ubuntu18  /bin/bash
```

2.2 Ubuntu Environment

This chapter introduces how to build a compile environment that is not based on the docker environment.

First, install Ubuntu 18.04. For the installation method, please refer to the Ubuntu official website.

Install the dependencies required to compile Yocto.

Bash 复制代码

```
sudo apt-get update
sudo apt-get install -y --assume-yes --no-install-recommends apt-utils
sudo apt-get install -y --assume-yes --no-install-recommends    automake
\
    autotools-dev \
    axel \
    bash \
    bash-completion \
    bc \
    bison \
    build-essential \
    busybox \
    ccache \
    chrpath \
    cpio \
    curl \
    debianutils \
    device-tree-compiler \
    diffstat \
    exuberant-ctags \
    fakeroot \
    file \
    flex \
    g++ \
    g++-multilib \
    gawk \
    gcc \
    gcc-multilib \
    git \
    gnupg \
    gperf \
    iputils-ping \
    less \
    libglib2.0-0 \
    libncurses-dev \
    libncurses5-dev \
    libssl-dev \
    locales \
    lsb-release \
    netcat-openbsd \
    nfs-common \
    openssh-server \
    pkg-config \
    procps \
    pv \
```

```
45        python-pip \
46        python3-pip \
47        rsync \
48        scons \
49        socat \
50        sshfs \
51        sudo \
52        texinfo \
53        tig \
54        tmux \
55        tree \
56        tzdata \
57        unzip \
58        vim \
59        wget \
60        x11-apps \
61        xz-utils \
62        zip \
63        zlib1g-dev \
64        zsh \
65        libevent-dev \
66        libgnutls-dane0 \
67        libgnutls-openssl27 \
68        libgnutls28-dev \
69        libgnutlsxx28 \
70        libidn2-dev \
71        libjsoncpp-dev \
72        liblog4cpp5-dev \
73        libopus-dev \
74        libunistring-dev \
75        libz-dev \
76        nettle-dev \
77        lib32ncurses5-dev \
78        lib32z-dev \
79        libaio-dev \
80        libattr1-dev \
81        libbluetooth-dev \
82        libbrlapi-dev \
83        libc6-dev-i386 \
84        libcap-dev \
85        libgl1-mesa-dev \
86        libglib2.0-dev \
87        liblzo2-dev \
88        libnuma-dev \
89        libpixman-1-dev \
90        libsnappy-dev \
91        libssl-dev \
92        libvdeplug-dev \
```

```
93        libx11-dev \
94        libxml2-utils \
95        openjdk-8-jdk \
96        x11proto-core-dev \
97        xsltproc \
98        libclang-dev \
99        cmake \
100       libvulkan-dev \
101       gtk-update-icon-cache \
102       module-init-tools \
```

3 Compile

This chapter introduces the common skills that will be used in the use of Yocto. For a systematic understanding of Yocto knowledge, you can refer to the following manuals:

- *Yocto Development Guide* (Chinese version) provided with SDK
- *BitBake User Guide* (English version)
- *Yocto Engineering Reference Guide* (English version)

3.1 Component

Yocto manages a large number of open source software components in package units. If you need to compile a package, the method is as follows:

```
Bash | 复制代码
1  bitbake “package-name”
```

Each package defines which tasks are supported in its own recipes file, and lists all tasks and help information supported by this package:

```
Bash | 复制代码
1  bitbake “package-name” -c listtasks
```

3.1.1 Source Directory

Yocto integrates with a large number of open source packages, which are usually compiled in the following directories:

- tmp-glibc/work/riscv64-oe-linux
- tmp-glibc/work/${MACHINE}

Take gnome-shell as an example, the directory result is usually as follows:

```
docker-ubuntu18:thead-build/light-fm/tmp-glibc/work/riscv64-oe-linux/gnome-shell$ tree -L 2
.
└── 3.34.5-r0
    ├── xxx.patch
    ├── build
    ├── debugsources.list
```

Where:

gnome-shell-3.34.5: Source directory

image: Compile output

build: Compile directory

How to query the package directory:

```
bitbake -e your_package _name  | grep ^S=
```

3.2 Linux Kernel

3.2.1 Source Path

Yocto downloads the source code to the following path when compiling:

```
tmp-glibc/work/d1-oe-linux/linux-thead
```

Take the 5.10 kernel as an example, after compiling the kernel once, you will see the following directory structure, where:

linux-5.10.y: Kernel source code, with git information

linux-light_fm_linux-standard-build: Compile intermediate files

temp: Log during compilation

image: The file mounted to the file system

```
└── 5.10.y-r0
    ├── defconfig
    ├── deploy-debs
    ├── deploy-linux-thead
    ├── image
    ├── license-destdir
    ├── linux-5.10.y
    ├── linux-light_fm_linux-standard-build
    ├── package
    ├── packages-split
    ├── pkgdata
    ├── pkgdata-pdata-input
    ├── pkgdata-sysroot
```

3.2.2 Build Linux Kernel

Generally, after modifying the Linux kernel source code, you only need to execute this command.

```
$ bitbake linux-thead -C compile
```

3.2.3 Clean Linux Kernel

This command will clear the build directory of the entire Linux kernel and is usually not required.

```
$ bitbake linux-thead -c clean
```

Note: clean will clear the kernel source code in temp/work.

3.2.4 menuconfig

```
$ bitbake linux-thead -c menuconfig
```

More commands can be viewed with bitbake linux-thead -c listtasks.

3.3 U-Boot

3.3.1 Source Path

When compiling, Yocto downloads the source code to the following path, where the penultimate directory name is the U-Boot version number.

```
tmp-glibc/work/d1-oe-linux/u-boot/1_2018.05-r0/git
```

3.3.2 Build U-Boot

Generally, after modifying the U-Boot kernel source code, you only need to execute this command.

```
$ bitbake u-boot -C compile
```

3.3.3 Clean U-Boot

This command will clear the build directory of the entire U-Boot kernel and is usually not required.

```
$ bitbake u-boot -c clean
```

More commands can be viewed with bitbake u-boot -c listtasks.

4 Add a Component

4.1 View a Component

For example, to see if a component with “perf” in its name exists:

```bash
$ bitbake-layers show-recipes *perf*
```

4.2 Add a Component

For example, to add the perf command to the image, the method is as follows:

Step 1: Add configuration to the bb file corresponding to the image. For example, adding the perf tool to d1-image-miniapp-dev image, then:

```bash
$ vi meta-d1/recipes-core/images/d1-image-miniapp-dev.bb

IMAGE_INSTALL += " perf "
```

Step 2: Execute the “build image” command, the method is shown in the previous chapter of this document.

Step 3: Execute the “package” command, the method is shown in the previous chapter of this document.

4.3 Enter the Component Directory

If you need to view the source code of a component after it has been added, you can enter the component build directory with the following command:

4.4 Build SYSROOTS SDK

The SDK built in this chapter contains a tool chain and is used to develop Linux applications. For example, to develop helloword running on Linux, this SDK is required. The SDK is already available in the SDK package provided and can be used directly by users without the need to follow the steps in this chapter.

```bash
$ bitbake perf -c devshell
```

Due to a bug in the Yocto version, you will need to delete coreutils before building the SDK, and then change it back after the build is complete.

In the openembedded-core directory, make the following changes (line 9):

```
diff --git a/meta/recipes-core/meta/target-sdk-provides-dummy.bb
b/meta/recipes-core/meta/target-sdk-provides-dummy.bb
index e3beeb796c..765611ffda 100644
--- a/meta/recipes-core/meta/target-sdk-provides-dummy.bb
+++ b/meta/recipes-core/meta/target-sdk-provides-dummy.bb
@@ -4,7 +4,6 @@ DUMMYPROVIDES_PACKAGES = "\
     busybox \
     busybox-dev \
     busybox-src \
-    coreutils \
     coreutils-dev \
     coreutils-src \
     bash \
```

For example, to build d1-image-miniapp-dev sysroots SDK:

```
$ bitbake d1-image-miniapp-dev -c populate_sdk
```

After the build is complete, the generated SDK is in the following directory, and the usage method can be found in *D1 Linux Quick Start Manual*.

```
thead-build/d1-miniapp/tmp-glibc/deploy/sdk/
```

4.5 Makefile Example

Using tree, you can see that there is a bb file, and then there is a directory where Makefile and source code are placed:

```
meta-thead/recipes-examples/hellomakefile$ tree
.
├── hello
│   ├── helloYocto.c
│   ├── makefile
│   └── zlibtest.c
├── hello.bb
└── README.md

1 directory, 5 files
```

The content of the bb file is as follows:

```
DESCRIPTION = "Hello World and Zlib test"
DEPENDS = "zlib"
SECTION = "libs"
LICENSE = "MIT"
PV = "3"
PR = "r0"

SRC_URI = " \
          file://helloYocto.c \
          file://zlibtest.c \
          file://makefile \
```

As you can see, the bb file specifies the values of the following variables:

- SRC_URI: Defines the source code
- LIC_FILES_CHKSUM: This is the checksum, if the source is based on the version management like git and svn, it does not need to be defined.
- FILES_$(PN): PN is the package number, which refers to the combination of PV and PR used by the software version, i.e.3-r0 as seen in the previous bitbake -s.

In addition to the variables, the bb file is reloaded in two ways:

- do_compile
- do_install

These two methods correspond to the compile and install tasks in the task list.

Before adding a package, confirm the following information:

- Where the original software, the address of the git repository, the branch, revision, the address of the tar package, and the sum value are.
- Whether there are additional patches and configuration files.
- Which method to use to compile, makefile, cmake, meson, ninja, script, to ensure that it can be compiled successfully.
- What the compiled product is and where it should be placed.

4.6 Cmake Example

The following is an example of how to integrate a cmake-based package in Yocto.

Bash 复制代码

```
meta-thead/recipes-examples/hellocmake$ tree
.
├── files
│   ├── A
│   │   ├── CMakeLists.txt
│   │   └── main.c
│   ├── B
│   │   ├── add.c
│   │   ├── add.h
│   │   └── CMakeLists.txt
│   ├── C
│   │   ├── CMakeLists.txt
│   │   ├── sub.c
│   │   └── sub.h
│   └── CMakeLists.txt
├── hellocmake.bb
└── README.md

4 directories, 11 files
```

The functions of the example are as follows:

B: The addition function written in C language is compiled into a dynamic library for use by C and A, that is, an example of compiling a dynamic library.

C: The subtraction function written in C language, and the example of calling the subtraction function in B, that is, the example of calling the dynamic library from the dynamic library.

A: An example of an executable program written in C language will call the dynamic library compiled by B and C, as well as an example of ffmpeg.

Compile method:

```
Bash | 复制代码
1  biatake hellocmake
```

5 Other

For other advanced usage of Yocto, please refer to *How to Use Yocto*.