6

2024년 10월 7일 월요일 오전 8:35

빌드 시스템 선택하기

<빌드 시스템 비교>

- 업스트림 소스 코드 빌드하는 것의 중요성

1. 외부 의존성이 없어도 언제든지 리빌드 가능
2. 소스 코드를 보유함으로써 디버깅이 가능
3. 필요한 사용자들에게 배포하기 위한 라이선스 요구 사항을 충족

- 빌드 시스템의 필수 기능

1. 업스트림 혹은 직접 소스 코드 제어 시스템이나 아카이브로부터 소스를 다운로드하고 로컬에 캐시한다.
2. 크로스 컴파일을 할 수 있도록 패치를 적용하고 아키텍처 의존 버그를 수정하고, 로컬 구성 정책을 적용하는 등의 작업을 수행한다.
3. 다양한 컴포넌트를 빌드한다.
4. 스테이징 영역을 생성하고 루트 파일시스템을 구성하며, 다양한 포맷의 이미지 파일을 생성한다.

빌드 시스템은 시스템 컴포넌트들을 패키지로 캡슐화 (호스트용 캡슐화, 타깃용 캡슐화)

패키지와 의존성을 해결하고 필요한 패키지 세트를 빌드하기 위한 빌드 매커니즘 사이에 의존성이 있음

- 빌드 시스템

Buildroot	GNU make와 Kconfig를 이용하는 시스템으로 사용이 편리
EmboToolKit	루트 파일시스템과 툴체인을 생성하기 위한 간단한 시스템이며 지금까지 기본적으로 유일하게 LLVM/Clang을 지원하는 시스템
OpenEmbedded	Yocto 프로젝트와 다른 시스템들의 코어 컴포넌트인 파워풀한 시스템
OpenWrt	무선 라우터용 펌웨어 빌드를 지향하고 있는 빌드 툴이며 기본적으로 런타임 패키지 관리를 지원
PTXdist	Pengutronix에서 후원하고 있는 오픈소스 빌드 시스템
The Yocto Project:	메타데이터, 툴, 문서와 함께 OpenEmbedded 코어를 확장한 것으로 아마도 가장 인기 있는 시스템

- Buildroot

  부트로더, 커널 이미지, 툴체인 등을 빌드할 수 있지만 루트 파일시스템 이미지 빌드를 주요 목적으로 갖고 있어서, 빠른 시간 안에 타깃 이미지를 생성해 설정하고 설치하는 것이 쉬움

- Yocto 프로젝트

  타깃 시스템을 정의하는 방법이 좀 더 일반적이어서 복잡한 임베디드 장치들을 적절히 빌드할 수 있음
  모든 컴포넌트는 기본적으로 RPM 포맷을 사용해 바이너리 패키지로 생성
  패키지들은 파일시스템 이미지를 만들기 위해 한꺼번에 묶음
  파일시스템 이미지 안에 런타임에 패키지들을 업데이트할 수 있게 해주는 패키지 관리자를 설치할 수 있음

  -> Yocto 프로젝트로 빌드하면 효과적으로 자신의 커스텀 리눅스 배포판을 만들 수 있음

바이너리 배포

- 메인 스트림 리눅스 배포판의 패키지 포맷

1. RPM (Red Hat Package Manager)

- 사용 배포판: 레드햇, 수세, 페도라 등
- 설명: RPM 포맷의 바이너리 패키지를 관리하는 시스템

2. DEB

- 사용 배포판: 우분투, 민트, 데비안
- 설명: 데비안 패키지 관리자를 위한 DEB 포맷 사용

3. IPK

- 사용 배포판: DEB 기반
- 설명: Itsy 패키지 포맷으로, 임베디드 장치에 적합한 경량 패키지 포맷

Buildroot 소개

Buildroot 버전들은 툴체인, 부트로더, 커널, 루트 파일시스템의 빌드가 가능
기본 빌드 툴로는 GNU make를 사용

<배경>
테스트용으로 작은 루트 파일시스템을 만들어 내는 방법으로서 uClinux와 uClibc 프로젝트들의 일부로 시작
Openwrt 의 시초

<안정 버전 릴리스 및 장기간 지원 버전>

1년에 4번 안정버전 릴리스
<년도>.02, <년도>.05 <년도>.08, <년도>.11 와 같은 git 태그로 표시

<설치>

```
user@raspberrypi:~ $ git clone git://git.buildroot.net/buildroot -b 2024.02.6
```

<구성>

Buildroot는 Kconfig/Kbuild 매커니즘 사용

<실행>

샘플 구성들 중 일부는 board/폴터에 해당 항목이 있는데, 커스텀 구성 요소 파일들과 타깃에 결과물을 설치하기 위한 정보를 포함

<실제 하드웨어에 타기팅>

```
master@build-master:/home/ysj/buildroot$ sudo make clean
rm -rf /home/ysj/buildroot/output/target /home/ysj/buildroot/output/images /home/ysj/buildroot/output/host  \
       /home/ysj/buildroot/output/build /home/ysj/buildroot/output/staging \
       /home/ysj/buildroot/output/legal-info /home/ysj/buildroot/output/graphs /home/ysj/buildroot/output/per-package
master@build-master:/home/ysj/buildroot$ sudo make raspberrypi4_64_defconfig
```

```
master@build-master:/home/ysj/buildroot$ sudo make
```

빌드가 완료되면 output/images/sdcard.img의 이름으로 이미지 생성
Post-image.sh 스크립트와 genimage-raspberrypi4-64.cfg 환경 설정 파일은 board/와 raspberrypi/ 두 폴더에 이미지 파일을 생성하는데 사용

- 라즈베리 파이 4에 운영체제 이미지 설치하는 방법

1. 리눅스 호스트 컴퓨터에 마이크로SD 카드를 연결
2. Etcher를 시작
3. Etcher에서 Flash from file 을 클릭
4. 라즈베리 파이 4를 위해 빌드한 sdcard.img를 선택
5. Etcher에서 Select target을 클릭
6. 첫 번째 단계에서 연결한 마이크로SD 카드를 선택
7. Etcher에서 Flash를 클릭하고 이미지를 생성
8. Etcher에서 플래싱이 종료되면 마이크로SD 카드를 꺼냄
9. 마이크로SD 카드를 라즈베리 파이 4에 넣음
10. 라즈베리 파이 4 의 USB-C 포트를 통해 전원을 공급

SSH를 통해 라즈베리 파이 4에 로그인 하려면 Buildroot 이미지 환경 설정에 dropbear나 openssh 같은 SSH 서버를 추가해야 함

<오버레이>

마지막 빌드 프로세스 단계에서 루트 파일시스템의 가장 상위에 복사되는 간단한 디렉터리 구조
실행 파일, 라이브러리와 그 외 포함하고 싶은 다른 것을 추가할 수 있음
Buildroot가 사용한 같은 툴체인으로 컴파일 돼야 함(Buildroot 툴체인을 사용하는 방법은 PATH에 추가하면 됨)

툴체인 접두어 -> <ARCH>-linux-

<패키지 추가>

Buildroot 패키지들은 각각의 서브디렉터리를 갖고 있으면서 package 디렉터리에 저장돼 있음
패키지는 최소한 구성 메뉴에서 패키지를 볼 수 있도록 해주는 Kconfig코드 조각을 포함하는 Config.in과 <패키지이름>.mk 형태인 Makefile로 구성
Makefile은 buildroot용 포맷으로 만들어지며 다운로드, 구성, 컴파일, 프로그램 설치를 할 수 있도록 지시 사항을 포함

<라이선스 준수>

라이선스 확인
```
$ make legal-info
```
Output/legal-info/ 폴더에 정보 저장

Yocto 프로젝트 소개

툴체인, 부트로더, 커널과 루트 파일시스템을 빌드할 수 있을 뿐 아니라 런타임에 설치될 수 있는 바이너리 패키지와 함께 전체 리눅스 배포판을 생성 가능
파이썬과 셸 스크립트의 조합으로 작성되는 레시피 그룹을 중심으로 구성
레시피에서 설정한 모든 것을 생성하는 BitBake라는 태스크 스케줄러가 포함돼 있음

<배경>
OpenEmbedded에 있으며 샤프 자우러스와 컴팩 ipaq을 포함한 다양한 핸드헬드 컴퓨터들에 리눅스를 포팅하기 위한 수많은 프로젝트 중에서 성장해옴

Yocto 프로젝트는 여러 컴포넌트로 구성

OE-Core	OpenEmbedded와 공유하는 코어 메타데이터
BitBake	OpenEmbedded 및 다른 프로젝트들과 공유하는 태스크 스케줄러
Poky	레퍼런스 배포판
Documentation	사용자 매뉴얼들과 각 컴포넌트에 대한 개발자 지침들
Toaster	BitBake와 그 메타데이터를 위한 웹 기반 인터페이스

Openembedded
2003년 핸드헬드 컴퓨터들을 위한 빌드 시스템으로 시작
컴팩트 IPK 포맷을 사용해 바이너리 패키지를 만들고, 이를 다양한 방법으로 타깃 시스템에 통합하거나 런타임에 설치할 수 있음
각 패키지를 위한 레시피들을 만들고 태스크 스케줄러로서 BitBake를 사용해 바이너리 패키지를 만듦
자신의 스펙에 맞는 전체 리눅스 배포판을 만들 수 있음

Yocto 프로젝트는 안정적인 기반의 버전을 제공하고 현재 상태에서 그대로 사용할 수 있거나 메타 레이어들을 사용해 확장될 수 있음

많은 SoC업체는 이런 방법으로 디바이스에 대해 BSP (Board Support Package)를 제공

<안정적인 릴리스와 지원>

프로젝트의 릴리스는 6개월마다 있음

코드 네임	릴리스 날짜	Yocto 버전	Poky 버전
Gatesgarth	2020년 10월	3.2	24
Dunfell	2020년 4월	3.1	23
Zeus	2019년 10월	3.0	22
Warrior	2019년 4월	2.7	21
Thud	2018년 11월	2.6	20
Sumo	2018년 4월	2.5	19

안정 버전은 현재 릴리스 사이클과 다음 사이클 동안 보안과 중대한 버그 수정을 지원
각 버전은 릴리스 후 약 12개월동안 지원
Dunfell은 Yocto의 첫번째 LTS 릴리스 (2년동안 결함 수정과 업데이트를 받을 수 있다는 뜻)

<Yocto 프로젝트 설치>

```
master@build-master:/home/ysj$ sudo git clone -b dunfell git://git.yoctoproject.org/poky.git
Cloning into 'poky'...
remote: Enumerating objects: 659585, done.
remote: Counting objects: 100% (1296/1296), done.
remote: Compressing objects: 100% (238/238), done.
remote: Total 659585 (delta 1145), reused 1134 (delta 1058), pack-reused 658289
Receiving objects: 100% (659585/659585), 208.49 MiB | 3.01 MiB/s, done.
Resolving deltas: 100% (480222/480222), done.
```

<설정>

```
master@build-master:~/ysj/test_poky$ source oe-init-build-env

### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    core-image-sato
    meta-toolchain
    meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'

Other commonly useful commands are:
 - 'devtool' and 'recipetool' handle common recipe tasks
 - 'bitbake-layers' handles common layer tasks
 - 'oe-pkgdata-util' handles common target package tasks
```

위의 코드는 Build/라는 이름의 작업 디렉터리가 생성돼 현재 디렉터리가 됨
모든 설정 파일, 중간에 생성되는 파일과 타깃 이미지 파일들은 이 build/ 디렉터리에 저장

처음 build/ 디렉터리에는 conf/ 이름의 서브디렉터리 하나만 가지고 있음

```
master@build-master:~/junhee/poky/book_poky/build/conf$ ls
bblayers.conf  local.conf  templateconf.cfg
```

local.conf :빌드하려고 하는 장치의 스펙과 빌드 환경을 포함
bblayers.conf: 사용하려고 하는 메타 레이어들의 패스를 포함

<빌드>

실제로 빌드를 진행하기 위해 생성하고자 하는 루트 파일시스템 이미지를 알려주면서 BitBake를 실행하는 것이 필요

- Yocto Project에서 사용하는 기본 이미지 유형

Core-image-minimal	테스트에 유용하고 커스텀 이미지의 기초가 되는 작은 콘솔 기반의 시스템
Core-image-minimal-nitramfs	core-image-minimal 과 비슷하나 램 디스크로 빌드
Core-image-x11	x11 서버와 xterminal 터미널 앱을 통한 그래픽 지원용 기본 이미지
Core-image-full-cmdline	콘솔 기반 시스템으로 표준 CLI 환경과 타깃 하드웨어에 대한 완전한 지원

BitBake에 최종 타깃을 지정함으로써 툴체인을 시작으로 역방향으로 동작하고 모든 의존성을 빌드

```
master@build-master:~/ysj/test_poky/build$ bitbake core-image-minimal
Loading cache: 100% |################################################################################
Loaded 1335 entries from dependency cache.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION           = "1.46.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "universal"
TARGET_SYS           = "arm-poky-linux-gnueabi"
MACHINE              = "qemuarm"
DISTRO               = "poky"
DISTRO_VERSION       = "3.1.33"
TUNE_FEATURES        = "arm armv7ve vfp thumb neon callconvention-hard"
TARGET_FPU           = "hard"
meta-nova
meta
meta-poky
meta-yocto-bsp       = "dunfell:63d05fc061006bf1a88630d6d91cdc76ea33fbf2"

Initialising tasks: 100% |###############################################################################
Sstate summary: Wanted 349 Found 347 Missed 2 Current 797 (99% match, 99% complete)
NOTE: Executing Tasks
NOTE: Tasks Summary: Attempted 3137 tasks of which 3137 didn't need to be rerun and all succeeded.
```

---

tmp/ 내 디렉터리

```
master@build-master:~/junhee/poky/book_poky/build/tmp$ ls
abi_version  buildstats  cache  deploy  hosttools  log  pkgdata  saved_tmpdir  sstate-control  stamps  sysroots-components  sysroots-uninative  work  work-shared
```

work/	빌드 디렉터리와 루트 파일시스템을위한 스테이징 영역을 포함
deploy/	타깃에배포를 위한최종 바리너리들을 포함
deploy/images/[machine name]/	타깃에 실행되기 위해 준비된 부트로더, 커널과 루트 파일시스템 이미지를 포함
deploy/rpm/	이미지를 구성했던 RPM 패키지들을 포함
deploy/lincenses/	각 패키지에서 추출된 라이선스 파일들을 포함

<레이어>

Yocto 프로젝트의 메타데이터는 레이어로 구성

- 코어 레이어

meta	OpenEmbedded 의 코어이며 poky에 대한 몇가지 변경사항을 가지고 있음
meta-poky	poky 배포판에 특정한 메타데이터
meta-yocto-bsp	yocto 프로젝트에서 지원하는 시스템의 BSP를 포함

BitBake 가 레시피를 검색하는 레이어의 목록은 <빌드 디렉터리>/conf/bblayers.conf에 저장 (위의 3개는 모두 포함)

이런 방식으로 레시피와 다른 설정 데이터를 구성함으로서 새로운 레이어들을 추가해 Yocto 프로젝트를 확장하기가 매우 용이

- 유용한 레이어 리스트

Meta-qt5	Qt5 라이브러리와 유틸리티
Meta-intel	Intel CPU와 SoC용 BSP
Meta-raspberrypi	라즈베리 파이 보드용 BSP
Meta-ti	T1 ARM 기반 SoC용 BSP

레이어 동작 방법 확인을 위한 실습 (Nova board 용 레이어)

meta-nova 레이어 생성

```
ysj@build-master:~/poky$ source oe-init-build-env build-nova
You had no conf/local.conf file. This configuration file has therefore been
created for you with some default values. You may wish to edit it to, for
example, select a different MACHINE (target hardware). See conf/local.conf
for more information as common configuration options are commented.

You had no conf/bblayers.conf file. This configuration file has therefore been
created for you with some default values. To add additional metadata layers
into your configuration please add entries to conf/bblayers.conf.

The Yocto Project has extensive documentation about OE including a reference
manual which can be found at:
    https://docs.yoctoproject.org

For more information about OpenEmbedded see their website:
    https://www.openembedded.org/


### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    core-image-sato
    meta-toolchain
    meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'

Other commonly useful commands are:
 - 'devtool' and 'recipetool' handle common recipe tasks
 - 'bitbake-layers' handles common layer tasks
 - 'oe-pkgdata-util' handles common target package tasks
ysj@build-master:~/poky/build-nova$ bitbake-layers create-layer nova
NOTE: Starting bitbake server...
Add your new layer with 'bitbake-layers add-layer nova'
ysj@build-master:~/poky/build-nova$ mv nova ../meta-nova
```

위 명령어는 build-nova라는 이름의 작업 디렉터리로 이동

```
ysj@build-master:~/poky$ cd meta-nova
ysj@build-master:~/poky/meta-nova$ ls
COPYING.MIT  README  conf  recipes-example
ysj@build-master:~/poky/meta-nova$ cd conf
ysj@build-master:~/poky/meta-nova/conf$ ls
layer.conf
```

Conf/layer.conf, README 파일과 MIT 라이선스의 COPYING.MIT를 포함하는 meta-nova레이어를 생성

<layer.conf 파일>

```
# We have a conf and classes directory, add to BBPATH
BBPATH .= ":${LAYERDIR}"

# We have recipes-* directories, add to BBFILES
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \
            ${LAYERDIR}/recipes-*/*/*.bbappend"

BBFILE_COLLECTIONS += "nova"
BBFILE_PATTERN_nova = "^${LAYERDIR}/"
BBFILE_PRIORITY_nova = "6"

LAYERDEPENDS_nova = "core"
LAYERSERIES_COMPAT_nova = "dunfell"
```

레이어를 빌드 설정에 추가 필요

```
ysj@build-master:~/poky/meta-nova/conf$ bitbake-layers add-layer ../meta-nova
NOTE: Starting bitbake server...
Specified layer directory /home/ysj/poky/meta-nova/meta-nova doesn't exist
```

환경 설정을 적용한 후 build-nova 작업 디렉터리에서 위 명령어 실행

구조 세팅 확인

```
ysj@build-master:~/poky/meta-nova/conf$ bitbake-layers show-layers
NOTE: Starting bitbake server...
layer                 path                                      priority
==========================================================================
meta                  /home/ysj/poky/meta                       5
meta-poky             /home/ysj/poky/meta-poky                  5
meta-yocto-bsp        /home/ysj/poky/meta-yocto-bsp             5
meta-nova             /home/ysj/poky/meta-nova                  6
ysj@build-master:~/poky/meta-nova/conf$
```

새로운 레이어 확인 (우선순위 5), 낮은 우선 순위를 갖고 있는 다른 레이어의 레시피를 우선할 수 있다는 뜻)

< BitBake와 레시피 >

Recipes (.bb)	소스 코드 사본과 다른 컴포넌트들의 의존성 파일을 얻는 방법, 빌드와 설치법을 포함한 소프트웨어 유닛의 빌드에 대한 정보를 포함
Append (.bbappend)	여러 레시피를 위한 공통의 정보를 갖고 있으며 정보가 레시피들 사이에서 공유될 수 있게 해줌 파일들은 include 나 require 키워드들을 사용해 포함될 수 있음, 차이점은 파일이 존재하지 않을 시, require는 에러를 발생, include는 그렇지 않음
Classes (.bbclass)	공통 빌드 정보를 포함하는데 , 예를 들어 커널을 빌드하는 방법 혹은 autotools 프로젝트를 빌드하는 방법 클래스들은 inherit 키워드를 이용해 레시피와 다른 클래스들로 상속되며 확장, classes/base.bbclass 클래스는 묵시적으로 모든 레시피에게 상속
Include (.inc)	여러 레시피를 위한 공통의 정보를 갖고 있으며, 정보가 레시피들 사이에서 공유될 수 이게 해줌 파일들은 include나 require 키워드들을 사용해 포함할 수 있음
Confguration (.conf)	프로젝트의 빌드 프로세스를 통제하는 다양한 구성 변수를 정의

레시피는 파이썬과 셸 코드의 조합으로 만들어진 태스크들의 모음 (태스크는 do_fetch, do_patch, do_unpack등의 이름을 가짐)

이런 태스크들을 실행하기 위해 **BitBake**를 이용
기본 태스크는 do_build, 이 태스크는 레시피를 빌드하기 위해 필요한 모든 서브태스크를 실행

레시피 안에서 사용 가능한 태스크 리스트 얻기

BitBake -c listtasks [recipe]

Ex) core-image-minial안의 레시피 리스트 얻기

```
ysj@build-master:~/poky$ bitbake -c listtasks core-image-minimal
```

```
Build Configuration:
BB_VERSION           = "1.46.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "universal"
```

```
TARGET_SYS           = "x86_64-poky-linux"
MACHINE              = "qemux86-64"
DISTRO               = "poky"
DISTRO_VERSION       = "3.1.33"
TUNE_FEATURES        = "m64 core2"
TARGET_FPU           = ""
meta
meta-poky
meta-yocto-bsp
meta-nova            = "dunfell:63d05fc061006bf1a88630d6d91cdc76ea33fbf2"

Initialising tasks: 100% |#################################################################
Sstate summary: Wanted 0 Found 0 Missed 0 Current 0 (0% match, 0% complete)
NOTE: No setscene tasks
NOTE: Executing Tasks
do_build                             Default task for a recipe - depends on all other norma
do_checkuri                          Validates the SRC_URI value
do_clean                             Removes all output files for a target
do_cleanall                          Removes all output files, shared state cache, and down
do_cleansstate                       Removes all output files and shared state cache for a
do_compile                           Compiles the source in the compilation directory
do_configure                         Configures the source by enabling and disabling any bu
do_deploy_source_date_epoch
do_deploy_source_date_epoch_setscene  (setscene version)
do_devpyshell                        Starts an interactive Python shell for development/deb
do_devshell                          Starts a shell with the environment set up for develop
do_fetch                             Fetches the source code
do_flush_pseudodb
do_image
do_image_complete
do_image_complete_setscene            (setscene version)
do_image_ext4
do_image_qa
do_image_qa_setscene                  (setscene version)
do_image_tar
do_install                           Copies files from the compilation directory to a holdi
do_listtasks                         Lists all defined tasks for a target
do_package                           Analyzes the content of the holding area and splits it
do_package_qa_setscene               Runs QA checks on packaged files (setscene version)
do_package_setscene                  Analyzes the content of the holding area and splits it
do_package_write_rpm_setscene        Creates the actual RPM packages and places them in the
do_packagedata                       Creates package metadata used by the build system to g
do_packagedata_setscene              Creates package metadata used by the build system to g
do_patch                             Locates patch files and applies them to the source coo
do_populate_lic_deploy
do_populate_lic_setscene             Writes license information for the recipe that is coll
do_populate_sdk                      Creates the file and directory structure for an instal
do_populate_sdk_ext
do_populate_sysroot_setscene         Copies a subset of files installed by do_install into
do_prepare_recipe_sysroot
do_rootfs                            Creates the root filesystem (file and directory struct
do_rootfs_wicenv
do_sdk_depends
do_unpack                            Unpacks the source code into a working directory
do_write_qemuboot_conf
NOTE: Tasks Summary: Attempted 1 tasks of which 0 didn't need to be rerun and all succeeded.
```

-> -c 옵션은 BitBake에게 do_ 부분을 뺀 이름의 태스크로 레시피에서 특정 태스크를 실행하는 것을 알려주는 옵션

do_listtasks는 간단하게 레시피 안에 정의된 모든 태스크의 리스트를 보여주는 특별한 태스크의 리스트를 보여주는 태스크

레시피의 소스코드를 다운로드하는 fetch 태스크

$bitbake -c fetch busybox

Ex)

```
ysj@build-master:~/poky/meta-nova/conf$ bitbake -c fetch busybox
Loading cache: 100% |######################################################################
Loaded 1334 entries from dependency cache.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION           = "1.46.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "universal"
TARGET_SYS           = "x86_64-poky-linux"
MACHINE              = "qemux86-64"
DISTRO               = "poky"
DISTRO_VERSION       = "3.1.33"
TUNE_FEATURES        = "m64 core2"
TARGET_FPU           = ""
meta
meta-poky
meta-yocto-bsp
meta-nova            = "dunfell:63d05fc061006bf1a88630d6d91cdc76ea33fbf2"

Initialising tasks: 100% |#################################################################
Sstate summary: Wanted 0 Found 0 Missed 0 Current 0 (0% match, 0% complete)
NOTE: No setscene tasks
NOTE: Executing Tasks
NOTE: Tasks Summary: Attempted 1 tasks of which 1 didn't need to be rerun and all succeeded.
```

이미지를 구성하는 모든 패키지의 소스 코드를 다운로드

$bitbake core-image-minimal --runall=fetch

Ex)

```
ysj@build-master:~/poky/meta-nova$ bitbake core-image-minimal --runall=fetch
Loading cache: 100% |################################################################################
Loaded 1334 entries from dependency cache.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION           = "1.46.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "universal"
TARGET_SYS           = "x86_64-poky-linux"
MACHINE              = "qemux86-64"
DISTRO               = "poky"
DISTRO_VERSION       = "3.1.33"
TUNE_FEATURES        = "m64 core2"
TARGET_FPU           = ""
meta
meta-poky
meta-yocto-bsp
meta-nova            = "dunfell:63d05fc061006bf1a88630d6d91cdc76ea33fbf2"

Initialising tasks: 100% |###############################################################################
Sstate summary: Wanted 0 Found 0 Missed 0 Current 0 (0% match, 0% complete)
NOTE: No setscene tasks
NOTE: Executing Tasks
NOTE: Tasks Summary: Attempted 235 tasks of which 235 didn't need to be rerun and all succeeded.
```

레시피 파일들은 **<패키지_이름>_<version>.bb** 로 명명

(예제) meta-nova 안에 helloworld 프로그램

다음과 같은 디렉터리 구조 생성

```
ysj@build-master:~/poky/meta-nova/recipes-local/helloworld$ tree
.
├── files
│   └── helloworld.c
└── helloworld_1.0.bb
```

레시피 명령들

```
DESCRIPTION = "A friendly program that prints Hello World!"
PRIORITY = "optional"
SECTION = "examples"

LICENSE = "GPLv2"
LICENSE_FLAGS = "commercial"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/GPL2.0;md5=801f80980d171dd6425610833a22dbe6"

SRC_URI = "file://helloworld.c"

S = "${WORKDIR}"

do_compile() {
${CC} ${CFLAGS} ${LDFLAGS} helloworld.c -o helloworld

}

do_install() {
install -d ${D}${bindir}
install -m 0755 helloworld ${D}${bindir}

}
```

SRC_URI = "file://helloworld.c" = file://URC라는 코드가 recipe 디렉터리에 로컬로 있다는 뜻
${D} = 레시피의 스테이징 영역 확장
${bindir} = 기본 바이너리 디렉터리

모든 레시피는 LICENSE에 의해 정의된 라이선스를 갖고 있으며 GPLv2로 설정

Helloworld 레시피가 컴파일하는 것을 확인하기 위해 Bitbake로 빌드

```
ysj@build-master:~/poky/meta-nova$ bitbake helloworld
Loading cache: 100% |################################################################################
Loaded 1334 entries from dependency cache.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION           = "1.46.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "universal"
TARGET_SYS           = "x86_64-poky-linux"
MACHINE              = "qemux86-64"
DISTRO               = "poky"
DISTRO_VERSION       = "3.1.33"
TUNE_FEATURES        = "m64 core2"
TARGET_FPU           = ""
meta
meta-poky
meta-yocto-bsp
meta-nova            = "dunfell:63d05fc061006bf1a88630d6d91cdc76ea33fbf2"

Initialising tasks: 100% |###############################################################################
Sstate summary: Wanted 9 Found 0 Missed 9 Current 126 (0% match, 93% complete)
NOTE: Executing Tasks
NOTE: Tasks Summary: Attempted 542 tasks of which 525 didn't need to be rerun and all succeeded.
```

<local.conf 를 통한 이미지 커스터마이징>

기본 이미지에 추가하고 싶은 기능을 정의할 수 있음
EXTRA_IMAGE_FEATURES를 통해 좀 더 포괄적으로 변경할 수 있음

dbg-pkgs	이미지 안에 설치돼 있는 모든 패키지를 위한 디버그 심벌 패키지를 설치
Debug-tweaks	쉽게 개발하기 위해 패스워드나 다른 수정 내역 없이 root 로그인을 가능하게 해줌
Package	패키지 관리 툴을 설치하고 패키지 관리 데이터베이스를 유지보수
Read-only-rootfs	루트 파일시스템을 읽기 전용으로 만들음
X11	X 서버를 설치
X11-base	최소 환경 구성으로 X 서버를 설치
X11-sato	OpenEmbedded Sato 환경을 설치

<이미지 레시피 쓰기>

사용가능한 이미지 리스트 얻기

$ meta*/recipes*/images/*.bb

```
master@build-master:~/ysj/test_poky$ ls meta*/recipes*/images/*.bb
meta/recipes-core/images/build-appliance-image_15.0.0.bb  meta/recipes-extended/images/core-image-testmaster.bb           meta/recipes-sato/images/core-image-sato-ptest-fast.bb
meta/recipes-core/images/core-image-base.bb               meta/recipes-extended/images/core-image-testmaster-initramfs.bb  meta/recipes-sato/images/core-image-sato-sdk.bb
meta/recipes-core/images/core-image-minimal.bb            meta/recipes-graphics/images/core-image-clutter.bb               meta/recipes-sato/images/core-image-sato-sdk-ptest.bb
meta/recipes-core/images/core-image-minimal-dev.bb        meta/recipes-graphics/images/core-image-weston.bb                meta-selftest/recipes-test/images/error-image.bb
meta/recipes-core/images/core-image-minimal-initramfs.bb  meta/recipes-graphics/images/core-image-x11.bb                  meta-selftest/recipes-test/images/oe-selftest-image.bb
meta/recipes-core/images/core-image-minimal-mtdutils.bb   meta/recipes-rt/images/core-image-rt.bb                         meta-selftest/recipes-test/images/test-empty-image.bb
meta/recipes-core/images/core-image-tiny-initramfs.bb     meta/recipes-rt/images/core-image-rt-sdk.bb                      meta-selftest/recipes-test/images/wic-image-minimal.bb
meta/recipes-extended/images/core-image-full-cmdline.bb   meta/recipes-sato/images/core-image-sato.bb                      meta-skeleton/recipes-multilib/images/core-image-multilib-example.bb
meta/recipes-extended/images/core-image-kernel-dev.bb     meta/recipes-sato/images/core-image-sato-dev.bb
```

손쉬운 접근 방법은 기존의 이미지 레시피를 가져오거나 local.conf 에서 사용했던 것들과 비슷한 명령을 이용해 수정하는 것

BitBake는 타깃 장치에서 실행하기 위한 이미지 빌드할 뿐만 아니라 호스트 시스템에서 개발하기 위한 SDK도 빌드할 수 있음

<SDK 생성>

개발 라이브러리들과 타깃에 설치되는 모든 라이브러리에 대한 헤더파일들이 포함된 툴체인을 원함
Populate_sdk 태스크를 사용해 모든 이미지에 대해 이 작업을 수행

```
$ bitbake -c populate_sdk nova-image
```

tmp/deploy/sdk안의 자체 설치 셸 스크립트가 만들어짐

```
ysj@build-master:~/poky/build-nova$ bitbake meta-toolchain
Loading cache: 100% |################################################################################
Loaded 1334 entries from dependency cache.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION           = "1.46.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "universal"
TARGET_SYS           = "x86_64-poky-linux"
MACHINE              = "qemux86-64"
DISTRO               = "poky"
DISTRO_VERSION       = "3.1.33"
TUNE_FEATURES        = "m64 core2"
TARGET_FPU           = ""
meta
meta-poky
meta-yocto-bsp
meta-nova            = "dunfell:63d05fc061006bf1a88630d6d91cdc76ea33fbf2"

Initialising tasks: 100% |###########################################################################
Sstate summary: Wanted 660 Found 0 Missed 660 Current 1017 (0% match, 60% complete)
NOTE: Executing Tasks
NOTE: Tasks Summary: Attempted 4159 tasks of which 2753 didn't need to be rerun and all succeeded.
```

C나 C++ 크로스 컴파일러들, C 라이브러리, 헤더 파일을 가진 기본 툴체인을 원한다면 다음과 같이 실행

SDK를 설치하기 위해서 셸 스크립트만 실행하면 됨
기본 설치 디렉터리: /opt/poky/이지만, 설치 스크립트는 설치 디렉터리를 수정할 수 있게 해줌

```
$ tmp/deploy/sdk/poky-glibc-x86_64-nova-image-cortexa8hf-neonbeaglebone-
yocto-toolchain-3.1.5.sh
Poky (Yocto Project Reference Distro) SDK installer version
3.1.5
Enter target directory for SDK (default: /opt/poky/3.1.5):
You are about to install the SDK to "/opt/poky/3.1.5". Proceed [Y/n]? Y
[sudo] password for frank:
Extracting SDK.....................................done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source
the environment setup script e.g.
$ . /opt/poky/3.1.5/environment-setup-cortexa8hf-neon-pokylinux-gnueabi
```

Arm 프로세서용 동작하도록 구성돼 있기 때문에 올바른 플래그 세트를 사용해 실행하면 세부적인 조정이 완료되기 때문임

크로스 컴파일링을 위해 environment-setup 스크립트를 적용할 때 생성되는 셸 변수를 사용 해야하며, 다음 것들을 포함

- CC: C 컴파일러
- CXX: C++ 컴파일러
- CPP: C전처리기
- AS: 어셈블러
- LD: 링커

<라이선스 검사>

Yocto 프로젝트는 각 패키지가 라이선스를 갖도록 돼 있음
빌드되면 각 패키지용으로 tmp/deploy/licenses/[package name] 안에 라이선스의 복사본이 위치
이미지에 사용된 패키지와 라이선스의 요약본은 <image name>-<machine name>-<datestamp>/ 디렉터리에 들어가 있음