

# OpenEmbedded-devtool

## 내용

- 1 개요
- 2 예
  - 2.1 새로운 어플리케이션 또는 새로운 라이브러리 추가
    - 2.1.1 목표
    - 2.1.2 작업 방법 (단계별)
    - 2.1.3 재료
  - 2.2 소유권이있는 레시피로 관리되는 기존 어플리케이션 또는 라이브러리 수정
    - 2.2.1 목표
    - 2.2.2 작업 방식 (단계별)
    - 2.2.3 재료
  - 2.3 응용 프로그램 또는 라이브러리에서 추가 만들기 / 업데이트
    - 2.3.1 목표
    - 2.3.2 작업 방식 (단계별)
    - 2.3.3 재료
- 3 devtool 명령 목록
  - 3.1 레시피 작업 시작
    - 3.1.1 devtool 추가
    - 3.1.2 devtool 수정
    - 3.1.3 devtool 업그레이드
  - 3.2 정보 얻기
    - 3.2.1 devtool 상태
    - 3.2.2 devtool 검색
  - 3.3 작업 공간에서 레시피 작업
    - 3.3.1 devtool 빌드
    - 3.3.2 devtool 이름 바꾸기
    - 3.3.3 devtool 편집 레시피
    - 3.3.4 devtool 찾기 레시피
    - 3.3.5 devtool 구성 도움말
    - 3.3.6 devtool 업데이트 레시피
    - 3.3.7 devtool 재설정
    - 3.3.8 devtool 마무리
  - 3.4 목표 변경 테스트
    - 3.4.1 devtool 배포 대상
    - 3.4.2 devtool 배포 취소 대상
    - 3.4.3 devtool 빌드 이미지
  - 3.5 고급
    - 3.5.1 devtool 작성 작업 공간

- [3.5.2 devtool 가져 오기](#)
- [3.5.3 devtool 추출](#)
- [3.5.4 devtool 동기화](#)
- [3.5.5 devtool 내보내기](#)

## 1 개요

OpenEmbedded는 이미지를 통해 **배포** 를 생성하거나 SDK 를 생성하는 빌드 시스템 입니다.

OpenEmbedded에서 도구를 사용하여 OpenEmbedded 배포와 함께 개발을 **시작할 수 있으며** 도구의 이름은 **devtool** 입니다.  
이 도구를 사용하면 다음을 수행 할 수 있습니다.

- 새로운 레시피 추가
- bbappend 파일을 생성하여 실제 레시피에 추가 생성
- 기존 bbappend 파일 업데이트



devtool 공식 문서 : [SDK 워크 플로우에서 devtool 사용 \(www.yoctoproject.org\)](http://www.yoctoproject.org).

**devtool** 도움말 표시 :

```
$ devtool --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool [--basepath BASEPATH] [--bbpath BBPATH] [-d] [-q]
        [-<컬러 컬러>] [-h]
        <하위 명령> ...
```

OpenEmbedded 개발 도구

옵션 :

```
--basepath BASEPATH SDK의 기본 디렉토리 / 빌드 디렉토리
--bbpath BBPATH BBPATH를 가져 오지 않고 명시 적으로 지정하십시오.
                        메타 데이터에서
-d, --debug 디버그 출력 활성화
-q, --quiet 오류 만 인쇄
--color COLOR Colorize 출력 (COLOR이 자동, 항상, 항상)
-h, --help이 도움말 메시지를 표시하고 종료
```

부속 명령 :

레시피 작업 시작 :

## 2 예

### 2.1 새 응용 프로그램 또는 새 라이브러리 추가

#### 2.1.1 목표

목표는 OpenEmbedded 빌드 설정에서 새 애플리케이션 또는 새 라이브러리를 통합하는 것입니다.

응용 프로그램 또는 라이브러리에는 OpenEmbedded 빌드 설정에서 참조하는 레이어에서 사용할 수 있는 레시피가 없습니다.

Devtool을 사용하면 OpenEmbedded 빌드 설정에서, 특히 관리하는 레이어에서 레시피를 생성 할 수 있습니다.

Devtool은 구성 유형 (cmake, autotools)과 같은 응용 프로그램이나 라이브러리를 통합하는 데 도움이되는 몇 가지 필수 사항을 감지 할 수 있지만 구성을 위해 전달할 매개 변수는 정의하지 않습니다. 적어도 수동으로 레시피를 필요에 따라 수동으로 조정해야 합니다. -올바르게 SRC\_URI를 채우고-configure, compile, install 명령을 추가 하여

① - 대상에 설치할 파일 목록을 지정하여 -...

### 2.1.2 작업 방식 (단계별)

1. 존재하지 않는 경우 소스 디렉토리를 작성하십시오 (빌드 디렉토리의 'mysources'예).

```
$> mkdir mysources / myapp
```

2. 응용 프로그램 소스 (git clone, extract tarball 등)를 가져 와서 소스 디렉토리에 넣습니다.

3. 작업 공간에 응용 프로그램을 추가합니다

```
$> devtool add myapp mysources / myapp
```

(devtool은 작업 공간에서 myapp에 대한 레시피를 작성 합니다 . devtool add 명령에 대한 설명 참조 )

4. 빌드 :

```
$> devtool 빌드 myapp
```

5. 네트워크 액세스를 통해 대상 장치 빌드 출력에 배포

```
$> devtool deploy-target myapp root @ <ip of board>
```

6. 소스 코드를 편집하고 필요에 따라 4-5를 반복합니다

. 7. 새로운 레시피로 레이어를 채웁니다.

먼저 사용자 정의 레이어 **meta-my-custo-layer** 를 생성 / 활성화해야 합니다 . 이를 위해 새로운 오픈 임베디드 레이어 아티클 을 생성하는 방법을 참조 할 수 있습니다 .

```
$> mkdir ../meta-st/meta-my-custo-layer/recipes-custom/myapp
$> cp workspace / recipes / myapp / myapp.bb ../meta-st/meta-my-custo-layer/recipes-custom/myapp
$> mkdir ../meta-st/meta-my-custo-layer/recipes-custom/myapp/myapp
$> cp mysources / myapp / * ../meta-st/meta-my-custo-layer/recipes-custom/myapp/myapp
$> cp <적절한 라이선스 파일> ../meta-st/meta-my-custo-layer/recipes-custom/myapp/myapp
```

../meta-st/meta-my-custo-layer/recipes-custom/myapp/myapp 폴더에 복사 된 모든 파일 은 **myapp.bb**의 SRC\_URI 필드에 추가되어야 합니다.

그러면 새 레시피 (myapp.bb)가 있어야합니다. 컴파일 한 커스텀 이미지 안에 추가

```
$> cd meta-st / meta-my-custo-layer / 레시피 샘플 / 이미지 /
```

my-custom-image.bb를 열고 다음 행을 추가하십시오. IMAGE\_INSTALL += "myapp"

### 2.1.3 재료

- 레시피 파일 내용

```
$> 고양이 작업 공간 / 레시피 /myapp/myapp.bb
# recipetool로 만든 레시피
# 이것은 레시피의 기초이며 완전히 기능하기 위해서는 추가 편집이 필요할 수 있습니다.
# (편집 할 때 이러한 주석을 제거하십시오.)
#
# 경고 : 다음의 LICENSE 및 LIC_FILES_CHKSUM 값이 가장 추측하기 좋습니다.
# 값이 완전하고 정확한지 확인하는 책임.
LICENSE = " 알 수없는 "
LIC_FILES_CHKSUM = " file://LICENSE;md5 = 6dc31330b6fcb6a82dea131bf3d33d33 "
# SRC_URI에 대한 정보가 아직 없습니다 (외부 소스 트리 만 지정됨)
SRC_URI = ""
DEPENDS += "wayland"는
cmake를 상속받습니다
# EXTRA_OECMAKE를 사용하여 cmake에 전달할 옵션을 지정하십시오.
EXTRA_OECMAKE = ""
```

**빨간색 텍스트** : devtool이 자동으로 감지 한 정보.

- 작업 공간 트리

```
$> 트리 작업 공간 / mysources /
작업 공간 /
|-추가
| ` -myapp.bbappend
|-conf
| ` -layer.conf
|-읽어보기
`-레시피
`-myapp
`-myapp.bb
mysources /
`-myapp
```

## 2.2 소유권이있는 레시피로 관리되는 기존 애플리케이션 또는 라이브러리 수정



devtool은 외부 소스 (externalsrc)를 기반으로하며 이미 외부 소스를 사용하는 레시피에서 작업 할 수 없습니다

### 2.2.1 목표

목표는 귀하가 소유 한 레시피를 업데이트하는 것입니다.

다음 예제에서는 수정을 적용하려는 라이브러리 **libsmat** 를 사용합니다.

### 2.2.2 작업 방식 (단계별)

1. 소스 디렉토리를 만듭니다 (예 : 빌드 디렉토리의 'mysources')

```
$> mkdir mysources
```

## 2. 추출 원

```
$> devtool modify -x libsmaf mysources / libsmaf  
(옵션 -x : 소스 코드를 추출하고  
libsmaf 레시피에서 사용할 수 있는 규칙에 따라 패치하도록 devtool에 요청 )
```

3. 로컬 git 저장소에서 코드를 편집하거나 변경 사항을 커밋합니다

. 4. 빌드합니다.

```
$> devtool 빌드 libsmaf
```

5. 네트워크 액세스를 통해 대상 장치 빌드 출력에 배포

```
$> devtool deploy-target libsmaf root @ <ip of board>
```

6. 소스 코드 편집 및 필요에 따라 3-5 회 반복

7. 레시피 위에 패치로 작성

```
$> devtool update-recipe -a ../meta-st/meta-my-custo-layer libsmaf
```

먼저 사용자 정의 레이어 **meta-my-custo-layer** 를 생성 / 활성화해야 합니다 . 이를 위해 새로운 오픈 임베디드 레이어 아티클 을 생성하는 방법을 참조 할 수 있습니다 .

8. 정상으로 돌아옵니다 (작업 영역에서 구성 요소 제거).

```
$> devtool 재설정 libsmaf
```

## 2.2.3 재료

- libsmaf.bb의 devtool로 수정

```
$> git diff recipessecurity / smaf / libsmaf.bb
diff git a / recipessecurity / smaf / libsmaf.bb b / recipessecurity / smaf / libsmaf.bb
색인 78d5882..6c04a2f 100644
a / recipessecurity / smaf / libsmaf.bb
+++ b / recipessecurity / smaf / libsmaf.bb
@@ 8,7 +8,9 @@ PV = "1.0 + git"

autotools gettext pkgconfig 상속

-SRC_URI = "git : //git.linaro.org/people/benjamin.gaignard/libsmaf.git; protocol = http"
+ SRC_URI = "git : //git.linaro.org/people/benjamin.gaignard/libsmaf.git; protocol = http \
+           파일 : //0001Someschanges.patch \
+ "
S = "$ {WORKDIR} / git"
```

빨간색 텍스트 : devtool이 자동으로 수정 한 정보

```
$> ls recipessecurity / smaf / libsmaf
0001Someschanges.patch
```

빨간색 텍스트 : devtool이 추가 한 패치 파일

## 2.3 응용 프로그램 또는 라이브러리에서 추가 만들기 / 업데이트



devtool은 외부 소스 (externalsrc)를 기반으로하며 이미 외부 소스를 사용하는 레시피에서 작업 할 수 없습니다

### 2.3.1 목표

목표는 특정 레시피에 대한 추가를 업데이트 (또는 작성)하는 것입니다.

이 예에서는 수정을 적용하려는 **pixman** 라이브러리 를 사용합니다.

### 2.3.2 작업 방식 (단계별)

1. 존재하지 않는 경우 소스 디렉토리를 작성하십시오 (여기서 mydirs는 빌드 디렉토리에 있습니다).

```
$> mkdir mysources
```

2. 추출 원

```
$> devtool 수정 -x pixman mysources / pixman
```

3. 코드를 편집하거나 로컬 git 저장소에서 변경 사항을 커밋하십시오
4. 빌드하십시오.

```
$> devtool 빌드 pixman
```

## 5. 네트워크 액세스를 통해 대상 장치 빌드 출력에 배포

```
$> devtool deploy-target pixman root @ <ip of board>
```

## 6. 소스 코드 편집 및 필요에 따라 3-5 회 반복

## 7. 레시피 추가 파일 위에 패치로 작성

```
$> devtool update-recipe -a ../meta-st/meta-my-custo-layer pixman
```

먼저 사용자 정의 레이어 **meta-my-custo-layer** 를 생성 / 활성화해야 합니다 . 이를 위해 새로운 오픈 임베디드 레이어 아티클 을 생성하는 방법을 참조 할 수 있습니다 . 그런 다음 :

## 8. 정상으로 돌아옵니다 (작업 공간의 구성 요소 제거).

```
$> devtool 재설정 pixman
```

## 2.3.3 재질

- 생성 된 추가 파일의 내용 : pixman.bbappend

```
$> cat meta-st / meta-st-framework / recipes-graphics / xorg-lib / pixman_0.32.6.bbappend
# 레이어에서 먼저 파일을 찾습니다
FILESEXTRAPATHS_prepend := "$ {THISDIR} / $ {PN} :"
SRC_URI += " file://0001-NV12-format-support.patch \
            file://0001-Somes-Changes.patch \
            "
```

- 레이어에 추가 된 트리

```
$> tree meta-st / meta-st-framework / recipes-graphics / xorg-lib
|-pixman
|  |-0001- NV12 -format-support.patch
|  `--0001-Somes-Changes.patch
`--pixman_0.32.6.bbappend
```

## 3 devtool 명령 목록

## 3.1 레시피 작업 시작

## 3.1.1 devtool 추가

작업 공간에 새로운 레시피 추가

- 도움말 명령 :

```
$ devtool add --help
```

참고 : 비트 베이크 서버 시작 중 ...

```
사용법 : devtool add [-h] [--same-dir | --no-same-dir] [-패치 URI]
          [--fetch-dev] [-버전 버전] [--no-git]
          [--srcrev SRCREV | --autorev] [--srcbranch SRCBRANCH]
          [-이진] [-또한 네이티브] [--src-subdir SUBDIR]
          [-미러] [-제공 제공]
          [레시피 이름] [srctree] [fetchuri]
```

지정된 소스 트리를 빌드하기 위해 새 레시피를 작업 공간에 추가합니다. 할 수있다  
선택적으로 원격 URI를 가져 와서 압축을 풀어 소스 트리를 만듭니다.

인수 :

recipeName 추가 할 새 레시피의 이름 (이름 만-버전 없음,  
경로 또는 확장자). 지정하지 않으면 시도합니다  
자동 감지합니다.

srctree 외부 소스 트리의 경로. 지정하지 않으면  
하위 디렉토리

<WORKSPACE\_LAYER\_PATH> / workspace / sources가 사용됩니다.

- 예

```
$ devtool mylibrary <소스 경로 / mylibrary> 추가
```

- 작업 공간 : add 명령의 결과

```
$ 트리 작업 공간
|-추가
|  `--mylibrary.bbappend
|-conf
|  `--layer.conf
|-읽어보기
`--레시피
   `--mylibrary
      `--mylibrary.bb
```

.bb : mylibrary의 레시피 파일이 mysource / mylibrary 디렉토리에있는 정보로 자동으로 채워  
집니다. bbbend : externalsrc 클래스를 통해 mylibrary의 소스 코드를 관리하기위한 레시피 파일.

### 3.1.2 devtool 수정

기존 레시피의 소스를 수정하고 작업 공간에 추가하십시오.

- 도움말 명령 :



```
$ devtool 수정 --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool 수정 [-h] [-와일드 카드] [-추출 | -추출 불가]
        [-같은 디렉토리 | --no-same-dir] [-분기 브랜치]
        [-보유]
        레시피 이름 [srctree]
```

기존 레시피의 소스를 수정하도록 빌드 환경을 설정합니다. 그만큼 기본 동작은 레시피에서 가져온 소스를 추출하여 자식 트리를 사용하면 작업 할 수 있습니다. 또는 이미 자신의 것이 있다면 미리 준비된 소스 트리 -n /-no-extract를 지정할 수 있습니다.

인수 :

- recipeName 편집 할 기존 레시피 이름 (이름 만-아니오 버전, 경로 또는 확장자)
- srctree 외부 소스 트리의 경로. 지정하지 않으면 하위 디렉토리 <WORKSPACE\_LAYER\_PATH> // workspace / sources가 사용됩니다.

- 예

```
$ devtool modify -x mylibrary <소스 경로 / mylibrary>
(옵션 -x : 소스 라이브러리를 추출하여
mylibrary 레시피에 사용 가능한 규칙 에 따라 <source path / mylibrary>에 패치하도록 devtool에 요청 하십시오)
```

- 작업 공간 :

```
$ 트리 작업 공간
작업 공간 /
|-추가
| ` -mylibrary_ <version> .bbappend
|-conf
| ` -layer.conf
|-읽어보기
`-레시피
```

### 3.1.3 devtool 업그레이드

기존 레시피 소스를 새 버전으로 업그레이드

- 도움말 명령 :

```
$ devtool 업그레이드 --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool 업그레이드 [-h] [-버전 VERSION] [--srcrev SRCREV]
          [--srcbranch SRCBRANCH] [--branch BRANCH] [-패치 없음]
          [-같은 디렉토리 | --no-same-dir] [--keep-temp]
          레시피 이름 [srctree]
```

기존 레시피를 새로운 업스트림 버전으로 업그레이드합니다. 업그레이드를 넣습니다.  
 관련 파일과 함께 레시피 파일을 작업 공간으로 가져와 추출  
 패치를 재배치해야 할 경우를 대비하여 소스 트리를 지정된 위치로  
 업그레이드의 결과로 추가).

인수 :

- recipeName 업그레이드 할 레시피 이름 (이름 만-버전 없음,  
경로 또는 확장자)
- srctree 소스 트리를 추출 할 경로입니다. 그렇지 않다면  
지정된 하위 디렉토리  
<WORKSPACE\_LAYER\_PATH> / workspace / sources가 사용됩니다.

- 예

```
$ devtool 업그레이드 mylib --srcrev <신규 SHA1>
```

## 3.2 정보 얻기

### 3.2.1 devtool 상태

작업 공간 상태를 표시합니다 (작업 공간의 레시피 목록).

- 도움말 명령 :

```
$ devtool 상태 --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool 상태 [-h]

현재 작업 공간에있는 레시피와 각각에 대한 경로를 나열합니다
외부 소스 트리

옵션 :
-h, --help이 도움말 메시지를 표시하고 종료
```

- 예

```
$ devtool 상태
```

### 3.2.2 devtool 검색

사용 가능한 레시피를 검색하십시오 ( **bitbake -s** 와 동일 하지만 패턴이 일치 함).

- 도움말 명령 :

```
$ devtool 검색 --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool 검색 [-h] 키워드

사용 가능한 대상 레시피를 검색합니다. 레시피 이름, 패키지 이름,
설명 및 설치된 파일을 검색하고 레시피 이름을 일치하게 인쇄합니다.

인수 :
    키워드 검색 할 키워드 (정규 표현식 구문 허용)

옵션 :
    -h, --help이 도움말 메시지를 표시하고 종료
```

- 예

```
$ devtool 검색 웨스턴
glibc GLIBC (GNU C 라이브러리)
웨이 랜드 웨스턴 컴퍼니
libgcc GNU cc 및 gcc C 컴파일러
GCC의 gcc- 런타임 런타임 라이브러리
Weston Wayland 컴포 지터에 대한 weston-init 시작 스크립트 및 시스템 단위 파일
```

### 3.3 작업 공간에서 레시피 작업

#### 3.3.1 devtool 빌드

작업 공간에서 관리되는 특정 패키지에 대해 do\_configure, do\_compile 및 do\_install을 수행하도록 레시피를 작성하고 devtool에 요청하십시오.

- 도움말 명령 :

```
$ devtool 빌드 --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool 빌드 [-h] [-s] recipename

비트 베이크를 사용하여 지정된 레시피를 작성합니다 (최대 포함
do_populate_sysroot, do_packagedata)

인수 :
    recipename 제조법

옵션 :
    -h, --help이 도움말 메시지를 표시하고 종료
    -s, --disable-parallel-make
        병렬 처리 비활성화
```

- 예

```
$ devtool 빌드 myapp
```

#### 3.3.2 devtool 이름 변경

레시피 파일 이름을 새 이름으로 바꾸십시오.

- 도움말 명령 :

```
$ devtool 이름 바꾸기 --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool 이름 바꾸기 [-h] [-버전 버전] [--no-srctree]
        레시피 이름 [newname]

작업 공간에서 레시피의 레시피 파일 이름을 바꾸거나 이름을 변경하거나
작업 영역 내의 모든 참조가
동시에 업데이트되었습니다. 레시피 파일 자체가
작업 영역 (예 : devtool 추가 후) devtool이 추가했을 때 특히 유용합니다.
올바른 이름을 자동으로 결정하지 않습니다.

인수 :
  recipename 이름을 바꿀 레시피의 현재 이름
  newname 레시피의 새 이름 (선택 사항, 필요한 경우에만 필요하지 않음)
        버전을 변경하고 싶습니다)

옵션 :
  -h, --help이 도움말 메시지를 표시하고 종료
  --version 버전, -V 버전
```

- 예

```
$ devtool 이름을 mylib mylib_newname으로 바꿉니다.
```



이 명령은 레시피 파일 자체가 작업 공간에 있어야하기 때문에 'devtool add'또는 'devtool upgrade'명령을 사용한 후에 만 작동합니다.

### 3.3.3 devtool 편집 레시피

구성된 편집기에서 레시피 파일 편집

- 도움말 명령 :

```
$ devtool edit-recipe --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool edit-recipe [-h] [--any-recipe] recipeName

지정된 편집기에서 EDITOR 변수로 지정된 기본 편집기를 실행합니다.
레시피. 레시피 파일 자체는 작업 공간에 있어야합니다 (예 :
"devtool add"또는 "devtool upgrade"의 결과); 당신은 이것을 재정의 할 수 있습니다
-a /-모든 레시피 옵션.

인수 :
  recipeName 편집 할 레시피

옵션 :
  -h, --help이 도움말 메시지를 표시하고 종료
  --any-recipe, -a 레시피 파일 자체가있는 위치뿐만 아니라 모든 레시피를 편집하십시오.
        작업 공간에서
```

- 예

```
$ devtool 편집 레시피 mylib
```

### 3.3.4 devtool 발견 레시피

레시피 파일 찾기

- 도움말 명령 :

```
$ devtool find-recipe --help
```

참고 : 비트 베이크 서버 시작 중 ...

사용법 : devtool find-recipe [-h] [--any-recipe] recipename

기본적으로, 이것은 작업 공간에서 레시피 파일을 찾습니다. 당신은 무시할 수 있습니다 이것은 -a /-any-recipe 옵션과 함께 사용됩니다.

인수 :

recipename 레시피 찾기

옵션 :

-h, --help이 도움말 메시지를 표시하고 종료

--any-recipe, -a 레시피 파일 자체의 위치뿐만 아니라 모든 레시피를 찾습니다.  
작업 공간에서

- 예

```
$ devtool find-recipe mylib
```

### 3.3.5 devtool 구성 도움말

이러한 스크립트를 사용하여 레시피에 대한 구성 도움말을 표시하십시오.

- 도움말 명령 :

```
$ devtool configure-help --help
```

참고 : 비트 베이크 서버 시작 중 ...

사용법 : devtool configure-help [옵션] recipename [--arg ...]

지정된 레시피에 대한 구성 스크립트에 대한 도움말을 표시합니다 (예 : 실행).

./configure --help) 앞에 현재 옵션을 설명하는 헤더가 붙습니다.

지정되었습니다. 출력이 적게 (또는 PAGER가 설정되어있는 경우)

쉽게 탐색 할 수 있습니다.

인수 :

recipename 구성 도움말을 표시하는 레시피

옵션 :

-h, --help이 도움말 메시지를 표시하고 종료

-p, --no-pager 페이지징 된 출력 비활성화

-n, --no-header 설명 헤더 텍스트 비활성화

--arg ... 나머지 인수를 구성 스크립트 대신 전달

--help (스크립트에 추가 도움말 옵션이있는 경우 유용)

- 예

```
$ devtool configure-help mylib
```

### 3.3.6 devtool 업데이트 레시피

조리법에 외부 소스 트리에서 변경 사항을 적용 :

- 업데이트 BB 파일에 의해
- 업데이트 또는 bbappend 파일을 만들어

- 도움말 명령 :

```
$ devtool update-recipe --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool update-recipe [-h] [--mode MODE] [--initial-rev INITIAL_REV]
          [--LAYERDIR 추가] [-와일드 카드 버전]
          [-제거하지 않음]
          레시피 이름
```

외부 소스 트리에서 레시피로 변경 사항을 적용합니다  
(필요에 따라 패치 업데이트 / 추가 / 제거 또는 SRCREV 업데이트). 노트  
이러한 변경 사항을 git 저장소에 순서대로 커밋해야 합니다.  
인정 될 것이다.

인수 :  
    recipeName 업데이트 할 레시피 이름

옵션 :  
    -h, --help이 도움말 메시지를 표시하고 종료  
    --mode MODE, -m MODE 업데이트 모드 (여기서 MODE는 패치, srcrev, auto입니다.  
        기본값은 자동입니다)

- 예

특정 레시피에 대한 bbappend를 만드십시오. 여기 pixman

```
$ devtool update-recipe -a ../meta-st/meta-st-framework pixman
```

레시피 libsmf 업데이트

```
$ devtool 업데이트 레시피 libsmf
```

### 3.3.7 devtool 재설정

작업 공간에서 레시피를 제거하십시오 (작업 공간에서만).

- 도움말 명령 :

```
$ devtool reset --help
```

참고 : 비트 베이크 서버 시작 중 ...

사용법 : devtool 재설정 [-h] [--all] [--no-clean] [recipeName [recipeName ...]]

작업 공간에서 지정된 레시피를 제거합니다 (상태 재설정).  
메타 데이터에 의해 정의 된 것).

인수 :

recipeName 레시피 재설정

옵션 :

-h, --help이 도움말 메시지를 표시하고 종료

--all, -a 모든 레시피 재설정 (작업 공간 지우기)

--no-clean, -n 레시피 출력을 제거하기 위해 sysroot를 정리하지 마십시오

- 예

```
$ devtool myapp 재설정
```

### 3.3.8 devtool 마감

수행 한 작업으로 레이어를 업데이트하여 devtool을 통해 개발을 완료 할 수 있습니다.

- 도움말 명령 :

```
$ devtool finish-도움말
```

참고 : 비트 베이크 서버 시작 중 ...

사용법 : devtool finish [-h] [--mode MODE] [--initial-rev INITIAL\_REV]  
레시피 이름 대상

지정된 레시피의 커밋 된 변경 사항을 지정된 레이어로 푸시합니다.

작업 공간에서 제거합니다. 업데이트 레시피와 거의 동일

업데이트-레시피 단계가 "올바른 일"을 수행한다는 점을 제외하고 재설정

레시피와 지정된 대상 레이어에 따라

인수 :

recipeName 레시피 완료

레시피를 넣을 레이어 / 경로입니다. 의 이름이 될 수 있습니다

bbayers.conf에 구성된 레이어

레이어의 밑면 또는 레이어 내부의 부분 경로.

devtool finish는 경로 기반을 완성하려고 시도합니다.

층의 구조에.

옵션 :

- 예

```
$ devtool finish mylib 레이어 / 경로
```

## 3.4 대상 변경 테스트

### 3.4.1 devtool 배포 대상

레시피 출력 파일을 라이브 대상 머신에 배치하십시오.

- 도움말 명령 :

```
$ devtool deploy-target --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool deploy-target [-h] [-c] [-s] [-n] [-p] [--no-check-space]
        [-P 포트] [-S | -스트립 없음]
        레시피 이름 대상


레시피의 빌드 출력 (예 : do_install 태스크 출력)을
ssh를 통한 라이브 대상 머신. 기본적으로 기존 파일은 유지됩니다
devtool undeploy-를 실행하면 덮어 쓰지 않고 복원됩니다.
표적. 참고 : 이것은 런타임이 아닌 레시피 자체 만 배포합니다.
의존성이므로 대상에 설치된 것으로 가정합니다.
미리.


인수 :
  recipename 배치 할 레시피
  ssh 서버를 실행하는 대상 라이브 대상 시스템 :
      user @ hostname [: destdir]

옵션 :
```

- 예

```
$ devtool deploy-target myapp root @ <ip of board>
```

 패키지가 런타임에 다른 패키지 (RDEPENDS)에 종속 된 경우 rootfs에 종속 패키지를 강제로 설치하지 않습니다. 따라서 rootfs가 업데이트되도록하려면 "devtool build-image"를 사용해야 합니다.

 '배포 대상'은 'D' 디렉토리를 복사 합니다. 레시피에 의해 생성 된 모든 파일을 포함합니다. 따라서 내용은 대상에 설치된 레시피 패키지에 따라 다릅니다. 그러나 'D' 에는 대상에 설치할 필요가 없는 파일이 포함될 수 있습니다. 특히 대상에 설치되지 않은 'vmlinux'파일을 생성하는 가상 / 커널 레시피의 경우입니다. 따라서 커널 레시피를 사용하여 '배포 대상'을 수행하면 '여유 공간 없음'오류가 발생할 수 있습니다 (배포 명령을 실행하기 전에 'vmlinux'를 수동으로 제거하여 해결할 수 있음)

### 3.4.2 devtool 배포 취소 대상

라이브 대상 기계에서 레시피 출력 파일을 제거하십시오.

- 도움말 명령 :



```
$ devtool undeploy-target --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool undeploy-target [-h] [-c] [-s] [-a] [-n] [-P PORT]
        [레시피 이름] 대상
```

이전에 라이브 대상 머신에 배치 한 레시피 출력 파일을 배치 취소합니다.  
devtool deploy-target.

인수 :

- recipeName 배치 취소 할 레시피 (-a /-all을 사용하지 않는 경우)
- ssh 서버를 실행하는 대상 라이브 대상 시스템 :  
user @ hostname

옵션 :

- h, --help이 도움말 메시지를 표시하고 종료
- c, --no-host-check ssh 호스트 키 검사 비활성화
- s, --show-status 진행률 / 상태 출력 표시
- a, --all 대상에 배치 된 모든 레시피를 배포 해제합니다
- n, --dry-run 배포 취소 할 파일 만 나열합니다.

- 예

```
$ devtool 배포 취소 대상 myapp root @ <ip of board>
```



이 명령은 실제 대상 시스템에서 파일을 영구적으로 제거 할 수 있습니다.

### 3.4.3 devtool 빌드 이미지

작업 공간 레시피 패키지를 포함하여 이미지를 빌드하십시오.

- 도움말 명령 :

```
$ devtool 빌드 이미지 --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool 빌드 이미지 [-h] [-p 패키지] [이미지 이름]
```

레시피의 패키지를 포함하도록 이미지를 확장하고  
작업 공간

인수 :

- imagename 빌드 할 이미지 레시피

옵션 :

- h, --help이 도움말 메시지를 표시하고 종료
- p 패키지, --add- 패키지 패키지  
전체 작업 공간에 패키지를 추가하는 대신  
이미지에 추가 할 패키지를 지정합니다 (별도)  
선택으로 여러 패키지)

- 예

```
$ devtool 빌드 이미지 st-image-weston
```

## 3.5 고급

### 3.5.1 devtool 작성 작업 공간

기본 폴더 대신 특정 작업 영역 폴더를 만듭니다.

- 도움말 명령 :

```
$ devtool create-workspace --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool create-workspace [-h] [--create-only] [layerpath]
```

새로운 작업 공간을 설정합니다. 참고 : 다른 devtool 하위 명령은 작업 영역은 필요에 따라 자동으로 수행되므로 devtool create-작업 공간이 위치해야하는 위치를 지정하려는 경우 작업 공간.

인수 :  
layerpath 작업 공간 레이어를 생성해야하는 경로

옵션 :  
-h, --help이 도움말 메시지를 표시하고 종료  
--create-only 작업 공간 계층 만 작성하고 구성을 변경하지 마십시오

- 예

```
$ devtool create-workspace NewWorkspace / 경로
```

### 3.5.2 devtool 가져 오기

내 보낸 작업 영역을 작업 영역으로 가져옵니다.

- 도움말 명령 :

```
$ devtool 가져 오기 --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool 가져 오기 [-h] [-덮어 쓰기] 파일
```

"devtool export"로 이전에 작성된 tar 아카이브를 작업 공간으로 가져 오기

인수 :  
FILE 가져올 tar 아카이브의 이름

옵션 :  
-h, --help이 도움말 메시지를 표시하고 종료  
--overwrite, -o 추출시 파일 덮어 쓰기

- 예

```
$ devtool import ExportedWorkspaceFileName
```

### 3.5.3 devtool 추출

-패치 (do\_patch).

- ```
$ devtool extract-도움말
```
- 참고 : 비트 베이크 서버 시작 중 ...
- 사용법 : devtool extract [-h] [--branch BRANCH] [--keep-temp] recipeName srctree
- 기존 레시피의 소스를 추출합니다.
- 인수 :
- recipeName 소스를 추출 할 레시피 이름
  - srctree 소스 트리를 추출 할 경로
- 옵션 :
- h, --help이 도움말 메시지를 표시하고 종료
  - 브랜치 지점, -b 지점
    - 체크 아웃 할 개발 브랜치 이름 (기본값)  
"devtool")
  - keep-temp 임시 디렉토리 유지 (디버깅 용)

- ```
$ devtool extract pixman <소스 경로 / pixman>
```

## 추출 된 소스 동기화

- ```
$ devtool 동기화 --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool 동기화 [-h] [-분기 브랜치] [--keep-temp] recipename srctree

기존 레시피에 대해 이전에 추출한 소스 트리 동기화

인수 :
    recipename 소스를 동기화 할 레시피 이름
    srctree 소스 트리의 경로

옵션 :
    -h, --help이 도움말 메시지를 표시하고 종료
    -브랜치 지점, -b 지점
                        체크 아웃 할 개발 브랜치 이름 (기본값 :
                        devtool)
    --keep-temp 임시 디렉토리 (디버그 용)를 유지합니다 (기본값 :
                        그릇된)
```

- 19/20

```
$ devtool sync libsmf mysources / libsmf
```

### 3.5.5 devtool 내보내기

작업 공간을 tarball로 내보내십시오.

- 도움말 명령 :

```
$ devtool 내보내기 --help
참고 : 비트 베이크 서버 시작 중 ...
사용법 : devtool 내보내기 [-h] [-파일 파일] [-덮어 쓰기]
          [-포함 포함 [INCLUDE ...] | -배제 제외
          [배제 ...]]
```

하나 이상의 레시피를 현재 작업 공간에서 tar 아카이브로 내보내기

옵션 :

```
-h, --help이 도움말 메시지를 표시하고 종료
--file FILE, -f FILE 출력 아카이브 파일 이름
--overwrite, -o 이전 내보내기 tar 아카이브를 덮어 씁니다.
-INCLUDE [INCLUDE ...] 포함, -i INCLUDE [INCLUDE ...] 포함
                    tar 아카이브에 레시피 포함
--EXCLUDE [EXCLUDE ...] 제외, -e EXCLUDE [EXCLUDE ...] 제외
                    tar 아카이브로 레시피 제외
```

- 예

```
$ devtool 내보내기 ExportWorkspaceFileName
```