8

2024년 10월 23일 수요일 오후 3:34

Yocto 의 내부를 살펴보자

Yocto의 아키텍처와 워크플로 분석

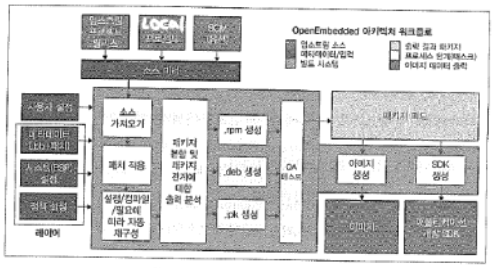


그림 8.1 OpenEmbedded 아키텍처와 워크플로

Yocto는 openembedded 프로젝트에서 워크플로를 가져옴
소스 재료는 bitbake 레시피 형태로 메타데이터를 통해 시스템에 입력으로 공급
빌드 시스템은 이 메타데이터를 사용해 소스 코드를 바이너리 패키지 피드로 가져오고, 구성하고, 컴파일함
개별 패키지는 스테이징 영역에서 생성되며, 라이선스가 포함된 매니페스트와 함께 완성
(매니페스트 : 소프트웨어 패키지나 시스템의 구성 요소에 대한 메타 정보를 담고 있는 파일)

YOCTO 빌드 시스템 워크플로의 7단계

1. 정책, 시스템, 소프트웨어 메타데이터에 대한 레이어를 정의
2. 소프트웨어 프로젝트의 소스 URI에서 소스를 가져옴
3. 소스 코드를 추출하고, 패치를 적용한 다음, 소프트웨어를 컴파일
4. 패키징을 위해 스테이징 영역에 빌드 아티팩트를 설치
5. 설치된 빌드 아티팩트를 루트 파일시스템을 위해 패키지 피드에 번들링
6. 제출하기 전에 바이너리 패키지 피드에 대해 QA 검사를 실행
7. 완성된 리눅스 이미지와 SDK를 병렬로 생성

첫단계와 마지막 단계 제외하고 워크플로의 모든 단계는 레시피별로 수행됨
코드 린팅, 안정성 검사, 다른 형태의 정적 분석은 컴파일 전후에 발생할 수 있음

Yocto가 생성하는 패키지는 rpm이나 deb, ipk 포맷일 수 있음
기본 바이너리 외에도 레시피에 대해 다음과 같은 패키지를 생성하려고 시도

dbg	디버그 심볼을 포함한 바이너리 파일
static-dev	헤더 파일과 정적 라이브러리
dev	헤더 파일과 공유 라이브러리 심볼릭 링크
doc	매뉴얼 페이지를 포함한 문서
locale	언어 번역 정보

ALLOW_EMPTY 변수가 활성화돼 있지 않으면 파일을 포함하지 않은 패키지가 생성되지 않음
기본적으로 생성되는 패키지 세트는 PACKAGES 변수에 의해 결정
=> 두 변수 모두 meta/classes/packagegroup.bbclass에 정의 (해당 bitbake 클래스에서 상속되는 패키지 그룹 레시피에 의해 재정의 될 수 있음)

<메타데이터>

메타데이터는 빌드 시스템으로 들어가는 입력이며, 빌드 대상과 방법을 제어
Yocto는 이러한 모든 입력을 구문 분석하고 완전한 리눅스 이미지로 변환

아키텍처별 설정
=>Yocto의 meta/conf/machine/include 디렉터리에 있는 tunes라는 파일과 개별 BSP 레이어 자체에 정의

```
ysj@build-master:~/poky/meta/conf/machine$ cd include/
ysj@build-master:~/poky/meta/conf/machine/include$ ls
README             qemuboot-x86.inc      tune-arm9tdmi.inc    tune-cortexa35.inc           tune-cortexa9.inc     tune-mips32.inc     tune-power6.inc     tune-ppce500.inc        tune-supersparc.inc
arm                riscv                 tune-atom.inc        tune-cortexa5.inc            tune-ep9312.inc       tune-mips32r2.inc   tune-power7.inc     tune-ppce500mc.inc      tune-thunderx.inc
m68k               sh                    tune-c3.inc          tune-cortexa53.inc           tune-i586-nlp.inc     tune-mips32r6.inc   tune-power9.inc     tune-ppce500v2.inc      tune-xscale.inc
microblaze         soc-family.inc        tune-core2.inc       tune-cortexa57-cortexa53.inc tune-i586.inc         tune-mips64.inc     tune-ppc476.inc     tune-ppce5500.inc       x86
mips               tune-arm1136jf-s.inc  tune-corei7.inc      tune-cortexa7.inc            tune-i686.inc         tune-mips64r2.inc   tune-ppc603e.inc    tune-ppce6500.inc       x86-base.inc
powerpc            tune-arm1176jz-s.inc  tune-cortexa15.inc   tune-cortexa72-cortexa53.inc tune-iwmmxt.inc       tune-mips64r6.inc   tune-ppc7400.inc    tune-sh3.inc
qemu.inc           tune-arm920t.inc      tune-cortexa17.inc   tune-cortexa72.inc           tune-mcf5441x.inc     tune-octeon.inc     tune-ppce300c2.inc  tune-sh4.inc
qemuboot-mips.inc  tune-arm926ejs.inc    tune-cortexa32.inc   tune-cortexa8.inc            tune-microblaze.inc   tune-power5.inc     tune-ppce300c3.inc  tune-strongarm1100.inc
ysj@build-master:~/poky/meta/conf/machine/include$
```

메타데이터를 사용하려면 소스 코드가 필요

- **do_fetch 태스크:**
  소스 코드를 가져오는 역할

1. **압축 파일 다운로드**:

필요한 소프트웨어를 다른 누군가가 개발하고 있다면, bitbake에게 프로젝트의 tar 압축 파일 릴리스를 다운로드 할 수 있도록 알려주기

2. **깃허브에서 소스 복제**:
   다른 사람의 오픈소스 소프트웨어를 확장하려면 깃허브에서 저장소를 포크하기. 그 다음 bitbake의 do_fetch 작업은 깃을 사용해 주어진 SRC_URI에서 소스 파일을 복제할 수 있음

소프트웨어 작업 환경:
소프트웨어를 팀의 로컬 프로젝트로 포함할 때는 서브디렉터리로 중첩하거나 externalsrc 클래스를 사용할 수 있음
externalsrc로 외부 소스를 사용하는 경우, 모든 빌드 인스턴스에서 동일한 경로가 필요하여 재사용에 어려움이 있음
Embedding: 소스가 레이어 저장소에 묶여 있어 다른 곳에서 사용하기 어려움

=> 두 기술 모두 개발 촉진 도구일 뿐, 실제 제품 단계에서 사용해서는 안 됨

배포 레이어:
배포 레이어는 리눅스 배포판에 필요한 기능과 속성을 포함
각 배포 레이어는 conf/distro 서브디렉터리를 가지며, 그 안의 .conf 파일이 배포판 또는 이미지에 대한 최상위 정책을 정의함

<빌드 태스크>

소스 코드 패치 적용(do_patch), 설정(do_configure), 컴파일(do_compile)

- **do_patch 태스크:**
  FILESPATH 변수와 레시피 SRC_URI 변수를 사용해 패치 파일을 찾고 원하는 소스 코드에 적용
  FILESPATH 변수 (Meta/classes/base.bbclass) : 빌드 시스템이 패치 파일을 검색하는데 사용하는 기본 디렉터리 세트를 정의
  패치 파일(.diff, .patch): 해당 레시피 파일이 있는 서브 디렉터리에 위치

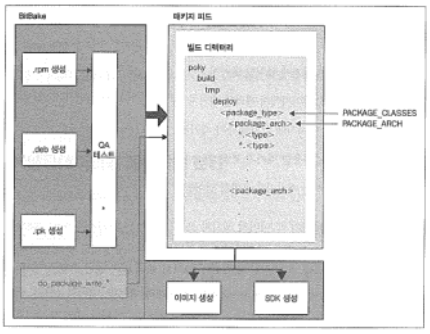


그림 8.2 패키지 피드

- **do_configure 태스크**:
  소스 코드 설정

- **do_compile 태스크:**
  소스코드를 컴파일하고 링크

- **do_install 태스크:**
  컴파일된 결과 파일을 패키징을 위해 준비된 스테이징 영역에 복사

- **do_package & do_package_data 태스크:**
  스테이징 영역에서 빌드 아티팩트를 처리하고 패키지로 나눔

- **do_package_qa 태스크:**
  패키지 피드 영역으로 전달되기 전에 패키지 아티팩트를 대상으로 일련의 QA 검사 수행
  => _자동으로 생성된 QA 검사는 meta/classes/insane.bbclass에 정의)

- **do_package_write_* 태스크:**
  개별 패키지를 만들어 패키지 피드 영역으로 보냄

=> 패키지 영역이 채워지면 bitbake는 이미지와 SDK를 생성 가능

<이미지 생성>

이미지 생성은 여러 변수에 의존해 일련의 태스크를 수행하는 다단계 프로세스

- **do_rootfs 태스크:**
  이미지에 대한 루트 파일시스템을 생성

IMAGE_INSTALL	이미지에 설치할 패키지
PACKAGE_EXCLUDE	이미지에서 제외할 패키지
IMAGE_FEATURES	이미지에 설치할 추가 패키지
PACKAGE_CLASSES	사용할 패키지 포맷
IMAGE_LINGUAS	지원 패키지를 포함할 언어

- **IMAGE_INSTALL 변수**:
  설치할 패키지 목록을 패키지 관리자에 전달

- **패키지 관리자**:
  호출되는 패키지 관리자는 패키지 피드 포맷에 따라 다르며, 런타임 패키지 관리자가 타깃에 포함돼 있는지와 관계없이 패키지 설치를 수행
  패키지 관리자가 없는 경우, 설치 후 불필요한 파일이 이미지에서 삭제되어 안정성과 공간을 절약

- **설치 후 스크립트 실행**:
  패키지 설치가 완료되면 설치 후 스크립트가 실행
  스크립트가 성공적으로 실행되면, 매니페스트가 작성되고 루트 파일시스템 이미지 최적화가 수행
  최상위 .manifest 파일에는 이미지에 설치된 모든 패키지가 나열
  기본 라이브러리 크기와 실행 시작 시간 최적화는
  ROOTFS_POSTPROCESS_COMMAND 변수에 의해 정의

- **do_image 태스크 시작**:
  루트 파일시스템이 완성되면, do_image 태스크가 이미지 처리를 시작
  IMAGE_PREPROCESS_COMMAND 변수에 정의된 모든 전처리 명령이 실행

- **최종 이미지 출력 파일 생성**:
  IMAGE_FSTYPES 변수에 지정된 모든 이미지 유형에 대해 do_image_* 태스크가 실행
  빌드 시스템은 IMAGE_ROOTFS 디렉터리의 내용을 가져와 하나 이상의 이미지 파일로 변환하며, 이 파일은 지정된 파일시스템 포맷에 따라 압축

- **do_image_complete 태스크**:
  마지막으로, do_image_complete 태스크가 IMAGE_POSTPROCESS_COMMAND 변수에 의해 정의된 모든 후처리 명령을 실행하여 이미지를 완성

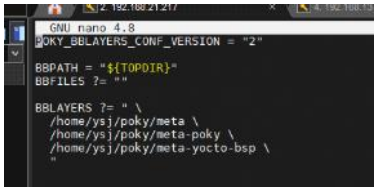
메타데이터를 레이어로 분리

메타데이터는 다음과 같은 개념을 중심으로 구성

distro	C라이브러리 선택, init 시스템, 윈도우 관리자를 포함한 OS 기능들
machine	CPU 아키텍처, 커널, 드라이버, 부트로더
recipe	애플리케이션 바이너리/스크립트
image	개발, 제조 또는 생산

Yocto로 시작하는 모든 주요 프로젝트에 대해 최소한 개별 배포 레이어, BSP 레이어, 애플리케이션 레이어를 생성
배포 레이어-> 애플리케이션이 실행될 타깃 OS를 빌드
(프레임 버퍼와 윈도우 관리자 설정 파일은 배포 레이어에 속함)
BSP 레이어-> 하드웨어가 동작하는 데 필요한 부트로더, 커널, 장치 트리를 지정함
애플리케이션 레이어-> 커스텀 애플리케이션을 구성하는 모든 패키지를 빌드하는데 필요한 레시피가 포함돼 있음

Poky의 기본 BBLAYERS 정의의 예



bitbake가 메타데이터 파일에 대한 경로와 검색 패턴을 설정할 수 있도록 모든 레이어에는 layer.conf 파일을 포함하는 conf 디렉터리가 있어야 함
BSP 레이어와 베포 레이어는 machine 이나 distro 서브 디렉터리가 있을수도 있음
Classes 서브 디렉터리는 자체 bitbake 클래스를 정의하는 레이어만 필요
Package-a와 package-z 는 실제 패키지의 자리 표시자 일 뿐

빌드 실패에 대한 문제 해결

Yocto 빌드 실패에 대응하기 위한 유용한 디버깅 기술을 다룸

1. Yocto를 복제한 디렉터리의 한 단계 위로 이동하여 bitbake 작업 환경 설정

```
ysj@build-master:~/poky/build-mine$ cd ..
ysj@build-master:~/poky$ source oe-init-build-env build-rpi/

### 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-rpi$
```

```
xZJbmwAiJJ+BnjmpV/n6o1kbj/oVYrUYvp/qQGhdrdmDFDdv3eR9AptrRrkp2zevPiKD5yNx57Xi+A3TfAwRM1BrIsq4n/aY8+
ysj@build-master:~/s32g-bsp-300-master/build_s32g274ardb2ubuntu$ bitbake fsl-image-ubuntu-base
Loading cache: 100% |#########################################################################
Loaded 2067 entries from dependency cache.
WARNING: /home/ysj/s32g-bsp-300-master/sources/meta-openembedded/meta-networking/recipes-daemons/p
be found
Parsing recipes: 100% |#########################################################################
Parsing of 3295 .bb files complete (1385 cached, 1910 parsed). 4892 targets, 266 skipped, 6 masked,
NOTE: Resolving any missing task queue dependencies
NOTE: Multiple providers are available for runtime ncurses-terminfo-base (ubuntu-base, ncurses)
Consider defining a PREFERRED_RPROVIDER entry to match ncurses-terminfo-base

Build Configuration:
BB_VERSION           = "1.48.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "ubuntu-20.04"
TARGET_SYS           = "aarch64-fsl-linux"
MACHINE              = "s32g274ardb2"
DISTRO               = "fsl-auto"
DISTRO_VERSION       = "30.0"
TUNE_FEATURES        = "aarch64"
TARGET_FPU           = ""
meta
meta-poky
meta-yocto-bsp
meta-oe
meta-multimedia
meta-python
meta-python2
meta-networking
meta-gnome
meta-filesystems
meta-webserver
meta-perl
meta-xfce
meta-virtualization
meta-optee
meta-security
meta-freescale
meta-alb             = "<unknown>:<unknown>"

NOTE: Fetching uninative binary shim http://downloads.yoctoproject.org/releases/uninative/2.9/x86_
eck PREMIRRORS first)
Initialising tasks: 100% |#######################################################################
Checking sstate mirror object availability: 100% |###############################################
Sstate summary: Wanted 414 Found 0 Missed 414 Current 0 (0% match, 0% complete)
NOTE: Executing Tasks
```

<오류 격리하기>

1. 먼저 bitbake 빌드 오류 메시지를 보고 패키지나 태스크 이름을 알 수 있는지 확인
   작업 공간에 어떤 패키지가 있는지 확실하지 않은 경우, 다음 명령을 사용해 패키지 목록을 가져올 수 있음

```
Summary: There were 14 WARNING messages shown.
ysj@build-master:~/s32g-bsp-300-master/build_s32g274ardb2ubuntu$ bitbake-layers show-layers
NOTE: Starting bitbake server...
layer                 path                                      priority
==========================================================================
meta                  /home/ysj/s32g-bsp-300-master/sources/poky/meta  5
meta-poky             /home/ysj/s32g-bsp-300-master/sources/poky/meta-poky  5
meta-yocto-bsp        /home/ysj/s32g-bsp-300-master/sources/poky/meta-yocto-bsp  5
meta-oe               /home/ysj/s32g-bsp-300-master/sources/meta-openembedded/meta-oe  6
meta-multimedia       /home/ysj/s32g-bsp-300-master/sources/meta-openembedded/meta-multimedia  6
meta-python           /home/ysj/s32g-bsp-300-master/sources/meta-openembedded/meta-python  7
meta-python2          /home/ysj/s32g-bsp-300-master/sources/meta-openembedded/meta-python2  7
meta-networking       /home/ysj/s32g-bsp-300-master/sources/meta-openembedded/meta-networking  5
meta-gnome            /home/ysj/s32g-bsp-300-master/sources/meta-openembedded/meta-gnome  7
meta-filesystems      /home/ysj/s32g-bsp-300-master/sources/meta-openembedded/meta-filesystems  6
meta-webserver        /home/ysj/s32g-bsp-300-master/sources/meta-openembedded/meta-webserver  6
meta-perl             /home/ysj/s32g-bsp-300-master/sources/meta-openembedded/meta-perl  6
meta-xfce             /home/ysj/s32g-bsp-300-master/sources/meta-openembedded/meta-xfce  7
meta-virtualization   /home/ysj/s32g-bsp-300-master/sources/meta-virtualization  8
meta-optee            /home/ysj/s32g-bsp-300-master/sources/meta-linaro/meta-optee  8
meta-security         /home/ysj/s32g-bsp-300-master/sources/meta-security  8
meta-freescale        /home/ysj/s32g-bsp-300-master/sources/meta-freescale  5
meta-alb              /home/ysj/s32g-bsp-300-master/sources/meta-alb  8
ysj@build-master:~/s32g-bsp-300-master/build_s32g274ardb2ubuntu$
```

2. 빌드에 실패한 패키지를 확인 했으면, 다음과 같이 현재 레이어에서 레시피를 검색하거나 해당 패키지와 관련된 파일을 인수로 추가 (다음 예시는 connman 패키지)

```
$ find ../poky -name "*connman*.bb*"
```

3. Connman 레시피에 사용 가능한 모든 태스크 나열

```
find: 's32g-bsp-300-master': No such file or directory
ysj@build-master:~/s32g-bsp-300-master/build_s32g274ardb2ubuntu$ bitbake -c listtasks connman
Loading cache: 100% |#########################################################################################
Loaded 4886 entries from dependency cache.
Parsing recipes: 100% |#######################################################################################
Parsing of 3295 .bb files complete (3291 cached, 4 parsed). 4892 targets, 266 skipped, 6 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
NOTE: Multiple providers are available for runtime python3-dbus (python3-dbus, ubuntu-base)
Consider defining a PREFERRED_RPROVIDER entry to match python3-dbus
NOTE: Multiple providers are available for runtime ncurses-terminfo-base (ubuntu-base, ncurses)
Consider defining a PREFERRED_RPROVIDER entry to match ncurses-terminfo-base

Build Configuration:
BB_VERSION           = "1.48.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "universal"
TARGET_SYS           = "aarch64-fsl-linux"
MACHINE              = "s32g274ardb2"
DISTRO               = "fsl-auto"
DISTRO_VERSION       = "30.0"
TUNE_FEATURES        = "aarch64"
TARGET_FPU           = ""
meta
meta-poky
meta-yocto-bsp
meta-oe
meta-multimedia
meta-python
meta-python2
meta-networking
meta-gnome
meta-filesystems
meta-webserver
meta-perl
meta-xfce
meta-virtualization
meta-optee
meta-security
meta-freescale
meta-alb             = "<unknown>:<unknown>"
```

```
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 normal tasks required to 'build' a recipe
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 downloaded source files for a target
do_cleansstate                      Removes all output files and shared state cache for a target
do_compile                          Compiles the source in the compilation directory
do_configure                        Configures the source by enabling and disabling any build-time and configuration options for the software being built
do_deploy_source_date_epoch
do_deploy_source_date_epoch_setscene  (setscene version)
do_devpyshell                       Starts an interactive Python shell for development/debugging
do_devshell                         Starts a shell with the environment set up for development/debugging
do_fetch                            Fetches the source code
do_install                          Copies files from the compilation directory to a holding area
do_listtasks                        Lists all defined tasks for a target
do_package                          Analyzes the content of the holding area and splits it into subsets based on available packages and files
do_package_qa                       Runs QA checks on packaged files
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 into subsets based on available packages and files (setscene version)
do_package_write_rpm                Creates the actual RPM packages and places them in the Package Feed area
do_package_write_rpm_setscene       Creates the actual RPM packages and places them in the Package Feed area (setscene version)
do_packagedata                      Creates package metadata used by the build system to generate the final packages
do_packagedata_setscene             Creates package metadata used by the build system to generate the final packages (setscene version)
do_patch                            Locates patch files and applies them to the source code
do_populate_lic                     Writes license information for the recipe that is collected later when the image is constructed
do_populate_lic_setscene            Writes license information for the recipe that is collected later when the image is constructed (setscene version)
do_populate_sysroot                 Copies a subset of files installed by do_install into the sysroot in order to make them available to other recipes
do_populate_sysroot_setscene        Copies a subset of files installed by do_install into the sysroot in order to make them available to other recipes (setscene version)
do_prepare_recipe_sysroot
do_unpack                           Unpacks the source code into a working directory
NOTE: Tasks Summary: Attempted 1 tasks of which 0 didn't need to be rerun and all succeeded.
ysj@build-master:~/s32g-bsp-300-master/build_s32g274ardb2ubuntu$
```

4. 오류를 재현하기 위해 connman 을 다시 빌드 할 수 있음

```
$ bitbake -c clean connman && bitbake connman
```

<환경 설정 덤프>

빌드 실패시 디버깅하면서 bitbake 환경 내 변수의 현재 값을 확인할 필요가 있음

1. 전역 환경을 덤프하고 DISTRO_FEATURES 값을 검색

```
ysj@build-master:~/s32g-bsp-300-master/sources/poky/build-rpi$ bitbake -e |less
```

```
#   set /home/ysj/s32
#     [_defaultval] "
/DISTRO_FEATURES=
```

```
DISTRO_FEATURES="acl alsa argp bluetooth ext2 ipv4 ipv6 largefile pcmcia usbgadget usbhost wifi xattr nfs zeroconf pci 3g nfc x11 vfat benchmark vfat ext2 mtd     perl python usbhost virtualization pam polk
it largefile opengl  multiarch wayland vulkan   pfe usrmerge docker systemd pulseaudio gobject-introspection-data ldconfig"
#
# $DISTRO_FEATURES_BACKFILL [2 operations]
#   set /home/ysj/s32g-bsp-300-master/sources/poky/meta/conf/documentation.conf:144
#     [doc] "Features to be added to DISTRO_FEATURES if not also present in DISTRO_FEATURES_BACKFILL_CONSIDERED. This variable is set in the meta/conf/bitbake.conf file and it is not intended to be user-con
figurable."
#   set /home/ysj/s32g-bsp-300-master/sources/poky/meta/conf/bitbake.conf:840
#     "pulseaudio sysvinit gobject-introspection-data ldconfig"
# pre-expansion value:
#   "pulseaudio sysvinit gobject-introspection-data ldconfig"
DISTRO_FEATURES_BACKFILL="pulseaudio sysvinit gobject-introspection-data ldconfig"
#
# $DISTRO_FEATURES_BACKFILL_CONSIDERED [2 operations]
#   _append /home/ysj/s32g-bsp-300-master/sources/poky/meta/conf/distro/include/init-manager-systemd.inc:3
#     " sysvinit"
#   set /home/ysj/s32g-bsp-300-master/sources/poky/meta/conf/documentation.conf:145
#     [doc] "Features from DISTRO_FEATURES_BACKFILL that should not be backfilled (i.e. added to DISTRO_FEATURES) during the build."
# pre-expansion value:
#   " sysvinit"
DISTRO_FEATURES_BACKFILL_CONSIDERED=" sysvinit"
#
# $DISTRO_FEATURES_DEFAULT [3 operations]
#   _append /home/ysj/s32g-bsp-300-master/sources/meta-alb/conf/distro/fsl-auto.conf:15
#     " benchmark vfat ext2 mtd     perl python usbhost virtualization"
#   _append /home/ysj/s32g-bsp-300-master/sources/meta-alb/conf/distro/fsl-auto.conf:23
#     " pam polkit"
#   set? /home/ysj/s32g-bsp-300-master/sources/poky/meta/conf/distro/include/default-distrovars.inc:13
#     "acl alsa argp bluetooth ext2 ipv4 ipv6 largefile pcmcia usbgadget usbhost wifi xattr nfs zeroconf pci 3g nfc x11 vfat"
# pre-expansion value:
#   "acl alsa argp bluetooth ext2 ipv4 ipv6 largefile pcmcia usbgadget usbhost wifi xattr nfs zeroconf pci 3g nfc x11 vfat benchmark vfat ext2 mtd     perl python usbhost virtualization pam polkit"
DISTRO_FEATURES_DEFAULT="acl alsa argp bluetooth ext2 ipv4 ipv6 largefile pcmcia usbgadget usbhost wifi xattr nfs zeroconf pci 3g nfc x11 vfat benchmark vfat ext2 mtd     perl python usbhost virtualization
pam polkit"
#
# $DISTRO_FEATURES_FILTER_NATIVE
#   set? /home/ysj/s32g-bsp-300-master/sources/poky/meta/conf/bitbake.conf:837
#     "api-documentation opengl"
DISTRO_FEATURES_FILTER_NATIVE="api-documentation opengl"
#
# $DISTRO_FEATURES_FILTER_NATIVESDK
#   set? /home/ysj/s32g-bsp-300-master/sources/poky/meta/conf/bitbake.conf:838
#     "api-documentation opengl"
DISTRO_FEATURES_FILTER_NATIVESDK="api-documentation opengl"
#
# $DISTRO_FEATURES_NATIVE
#   set? /home/ysj/s32g-bsp-300-master/sources/poky/meta/conf/bitbake.conf:832
#     "x11 ipv6 xattr"
DISTRO_FEATURES_NATIVE="x11 ipv6 xattr"
#
# $DISTRO_FEATURES_NATIVESDK
#   set? /home/ysj/s32g-bsp-300-master/sources/poky/meta/conf/bitbake.conf:833
#     "x11"
DISTRO_FEATURES_NATIVESDK="x11"
#
# $DISTRO_NAME [4 operations]
#   set /home/ysj/s32g-bsp-300-master/sources/poky/meta/conf/bitbake.conf:733
#     [_defaultval] "OpenEmbedded"
:
```

2. Busybox의 패키지 환경을 덤프하고 소스 디렉터리를 확인
3. Connman의 패키지 환경을 덤프하고 작업 디렉터리를 확인
   (이미지에 대해서도 동일한 명령 적용)

```
ysj@build-master:~/s32g-bsp-300-master/build_s32g274ardb2ubuntu$ bitbake -e busybox | grep ^S=
S="/home/ysj/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/work/aarch64-fsl-linux/busybox/1.32.0-r0/busybox-1.32.0"
ysj@build-master:~/s32g-bsp-300-master/build_s32g274ardb2ubuntu$ bitbake -e connman | grep ^WORKDIR=
WORKDIR="/home/ysj/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/work/aarch64-fsl-linux/connman/1.38-r0"
ysj@build-master:~/s32g-bsp-300-master/build_s32g274ardb2ubuntu$ bitbake -e core-image-minimal | grep ^S=
S="/home/ysj/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/work/s32g274ardb2-fsl-linux/core-image-minimal/1.0-r0/core-image-minimal-1.0"
ysj@build-master:~/s32g-bsp-300-master/build_s32g274ardb2ubuntu$
```

<태스크 로그 읽기>

Bitbake는 모든 셸 태스크에 대한 로그 파일을 생성하고 패키지의 작업 디렉터리에 있는 임시 폴더에 저장
(connman의 경우 다음과 같은 경로)

```
ysj@build-master:~/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/work/aarch64-fsl-linux/connman/1.38-r0/temp$ ls
depsig.do_deploy_source_date_epoch         log.do_populate_lic                    run.do_fetch                    run.extend_recipe_sysroot.638264        run.sstate_hardcode_path_unpack.644343
depsig.do_deploy_source_date_epoch.641703  log.do_populate_lic.644343             run.do_fetch.638264             run.extend_recipe_sysroot.639001        run.sstate_report_unihash.641703
depsig.do_populate_lic                     log.do_unpack                          run.do_patch                    run.extend_recipe_sysroot.639733        run.sstate_report_unihash.644343
depsig.do_populate_lic.644343              log.do_unpack.639001                   run.do_patch.639733             run.patch_do_patch.639733               run.sstate_task_postfunc.641703
log.do_deploy_source_date_epoch            log.task_order                         run.do_populate_lic             run.populate_lic_qa_checksum.644343     run.sstate_task_postfunc.644343
log.do_deploy_source_date_epoch.641703     run.base_do_fetch.638264               run.do_populate_lic.644343      run.sstate_create_package.641703        run.sstate_task_prefunc.641703
log.do_fetch                               run.base_do_unpack.639001              run.do_qa_patch.639733          run.sstate_create_package.644343        run.sstate_task_prefunc.644343
log.do_fetch.638264                        run.create_source_date_epoch_stamp.639001  run.do_qa_unpack.639001      run.sstate_hardcode_path.641703         run.uninative_changeinterp.641703
log.do_patch                               run.do_deploy_source_date_epoch        run.do_unpack                   run.sstate_hardcode_path.644343         run.uninative_changeinterp.644343
log.do_patch.639733                        run.do_deploy_source_date_epoch.641703 run.do_unpack.639001            run.sstate_hardcode_path_unpack.641703
ysj@build-master:~/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/work/aarch64-fsl-linux/connman/1.38-r0/temp$
```

<더 많은 로깅 추가>

파이썬에서 로깅하는 법
Bitbake 모듈을 사용해 파이썬의 표준 로거 모듈을 호출
셸에서 로깅하기 위해 bitbake의 로깅 클래스를 사용 (meta/classes/logging.bbclass에서 확인)

```
# The following logging mechanisms are to be used in bash functions of recipes.
# They are intended to map one to one in intention and output format with the
# python recipe logging functions of a similar naming convention: bb.plain(),
# bb.note(), etc.

LOGFIFO = "${T}/fifo.${@os.getpid()}"

# Print the output exactly as it is passed in. Typically used for output of
# tasks that should be seen on the console. Use sparingly.
# Output: logs console
bbplain() {
        if [ -p ${LOGFIFO} ] ; then
                printf "%b\0" "bbplain $*" > ${LOGFIFO}
        else
                echo "$*"
        fi
}

# Notify the user of a noteworthy condition.
# Output: logs
bbnote() {
        if [ -p ${LOGFIFO} ] ; then
                printf "%b\0" "bbnote $*" > ${LOGFIFO}
        else
                echo "NOTE: $*"
        fi
}

# Print a warning to the log. Warnings are non-fatal, and do not
# indicate a build failure.
# Output: logs console
bbwarn() {
        if [ -p ${LOGFIFO} ] ; then
                printf "%b\0" "bbwarn $*" > ${LOGFIFO}
        else
                echo "WARNING: $*"
        fi
}

# Print an error to the log. Errors are non-fatal in that the build can
# continue, but they do indicate a build failure.
# Output: logs console
bberror() {
        if [ -p ${LOGFIFO} ] ; then
                printf "%b\0" "bberror $*" > ${LOGFIFO}
        else
                echo "ERROR: $*"
        fi
}

# Print a fatal error to the log. Fatal errors indicate build failure
# and halt the build, exiting with an error code.
# Output: logs console
bbfatal() {
```

base.bbclass 를 상속하는 모든 레시피는 자동으로 logging.bbclass 상속
다음의 모든 로깅 기능은 대부분의 셸 레시피 파일에서 이미 사용할 수 있음을 의미

Bbplain	전달된 내용을 그대로 정확히 출력
Bbnote	NOTE 접두어가 있는 주목할 만한 조건을 출력
Bbwarn	WARNING 접두어가 있는 치명적이지 않은 경고를 출력
bbfatal	치명적인 오류를 출력하고 빌드를 중지
bbdebug	로그 레벨에 따라 디버그 메시지를 출력
bberror	ERROR 접두어가 있는 치명적인 오류를 출력

<devshell에서 명령 실행>

Devshell은 명령줄에서 개별 항목을 구성하고 컴파일 하는데 사용
Connman을 빌드하기 위해 devshell에 들어가는 명령어 (실패)

```
ysj@build-master:~/s32g-bsp-300-master/build_s32g274ardb2ubuntu$ bitbake -c devshell connman
Loading cache: 100% |#########################################################################################################################################################| Time: 0
Loaded 4886 entries from dependency cache.
Parsing recipes: 100% |#######################################################################################################################################################| Time: 0
Parsing of 3295 .bb files complete (3291 cached, 4 parsed). 4892 targets, 266 skipped, 6 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
NOTE: Multiple providers are available for runtime python3-dbus (python3-dbus, ubuntu-base)
Consider defining a PREFERRED_RPROVIDER entry to match python3-dbus
NOTE: Multiple providers are available for runtime ncurses-terminfo-base (ubuntu-base, ncurses)
Consider defining a PREFERRED_RPROVIDER entry to match ncurses-terminfo-base

Build Configuration:
BB_VERSION           = "1.48.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "universal"
TARGET_SYS           = "aarch64-fsl-linux"
MACHINE              = "s32g274ardb2"
DISTRO               = "fsl-auto"
DISTRO_VERSION       = "30.0"
TUNE_FEATURES        = "aarch64"
TARGET_FPU           = ""
meta
meta-poky
meta-yocto-bsp
meta-oe
meta-multimedia
meta-python
meta-python2
meta-networking
meta-gnome
meta-filesystems
meta-webserver
meta-perl
meta-xfce
meta-virtualization
meta-optee
meta-security
meta-freescale
meta-alb             = "<unknown>:<unknown>"

Initialising tasks: 100% |####################################################################################################################################################| Time: 0
Sstate summary: Wanted 43 Found 0 Missed 43 Current 237 (0% match, 84% complete)
NOTE: Executing Tasks
WARNING: mobile-broadband-provider-info-1_20201225-r0 do_fetch: Failed to fetch URL git://gitlab.gnome.org/GNOME/mobile-broadband-provider-info.git;protocol=https, attempting MIRRORS if available
ERROR: mobile-broadband-provider-info-1_20201225-r0 do_fetch: Fetcher failure: Fetch command export PSEUDO_DISABLED=1; unset _PYTHON_SYSCONFIGDATA_NAME; export DBUS_SESSION_BUS_ADDRESS="unix:path=/ru
1001/bus"; export PATH="/home/ysj/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/sysroots-uninative/x86_64-linux/usr/bin:/home/ysj/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/work/aarch64-fsl-l
obile-broadband-provider-info/1_20201225-r0/recipe-sysroot-native/usr/bin/tar-native:/home/ysj/s32g-bsp-300-master/sources/poky/scripts:/home/ysj/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/work/
4-fsl-linux/mobile-broadband-provider-info/1_20201225-r0/recipe-sysroot-native/usr/bin/aarch64-fsl-linux:/home/ysj/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/work/aarch64-fsl-linux/mobile-broadb
ovider-info/1_20201225-r0/recipe-sysroot/usr/bin/crossscripts:/home/ysj/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/work/aarch64-fsl-linux/mobile-broadband-provider-info/1_20201225-r0/recipe-sysr
tive/usr/sbin:/home/ysj/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/work/aarch64-fsl-linux/mobile-broadband-provider-info/1_20201225-r0/recipe-sysroot-native/usr/bin:/home/ysj/s32g-bsp-300-master
_s32g274ardb2ubuntu/tmp/work/aarch64-fsl-linux/mobile-broadband-provider-info/1_20201225-r0/recipe-sysroot-native/sbin:/home/ysj/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/work/aarch64-fsl-linux
e-broadband-provider-info/1_20201225-r0/recipe-sysroot-native/bin:/home/ysj/s32g-bsp-300-master/sources/poky/bitbake/bin:/home/ysj/s32g-bsp-300-master/build_s32g274ardb2ubuntu/tmp/hosttools"; export H
home/ysj"; LANG=C git -c core.fsyncobjectfiles=0 fetch -f --progress https://gitlab.gnome.org/GNOME/mobile-broadband-provider-info.git refs/*:refs/* failed with exit code 128, output:
fatal: unable to access 'https://gitlab.gnome.org/GNOME/mobile-broadband-provider-info.git/': server certificate verification failed. CAfile: none CRLfile: none
```

<의존성 그래프 작성>

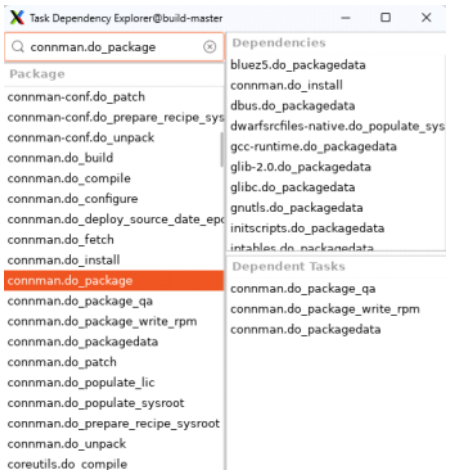
패키지의 의존성 중 하나에서 빌드할 때 실제로 오류가 발생하면 빌드 오류의 원인을 패키지 파일 내에서 찾을 수 없는 경우가 있음

Connman 패키지에 대한 의존성 목록을 가지고 오는 명령어 (실패)

```
- 'oe-pkgdata-util' handles common target package tasks
ysj@build-master:~/s32g-bsp-300-master/sources/poky/build-rpi$ bitbake -v connman
Loading cache: 100% |#################################################################################
Loaded 1354 entries from dependency cache.
```

Connman을 분석한 후 태스크 탐색기의 그래픽 UI 를 실행

```
Summary: There were 2 ERROR messages shown, returning a non-zero exit code.
ysj@build-master:~/s32g-bsp-300-master/sources/poky/build-rpi$ bitbake -g connman -u taskexp
ysj@build-master:~/s32g-bsp-300-master/sources/poky/build-rpi$
```



Bitbake 구문과 의미의 이해

bitbake는 태스크 실행자
bitbake의 강점은 작업 간 의존성을 충족하면서 동시에 태스크를 실행하는 기능이 있음

<태스크>

태스크는 bitbake가 레시피를 실행하기 위해 순서대로 실행 해야하는 함수
태스크 이름 do_접두사로 시작 (recipe-core/systemd)

```
do_deploy () {
        install ${B}/src/boot/efi/systemd-boot*.efi ${DEPLOYDIR}
}
addtask deploy before do_build after do_compile
```

addtask 명령어를 사용해 즉시 태스크로 승격 (단일 레시피 내 태스크 간의 의존성 표현)
addtask 명령은 태스크 간 의존성도 지정
deltask 명령을 사용해 태스크 삭제도 가능

```
deltask do_deploy
```

레시피에서 태스크가 삭제 되지만, 기존 do_deploy 함수 정의는 그대로 남아있어 호출은 가능

<의존성>

효율적인 병렬 처리를 보장하기 위해 bitbake는 태스크 레벨에서 의존성을 처리

서로 다른 레시피의 태스크 간 의존성 존재

- **태스크 간 의존성**

  변수 플래그(varflags)는 특성이나 속성을 변수에 연결하는 수단
  키를 값으로 설정하고 해당 키로 값을 검색할 수 있다는 점에서 해시 맵의 키처럼 동작
  bitbake 변수 플래그는 태스크 간 의존성을 표현하는 또 다른 방법 제공

- **빌드 타임 의존성**

  Bitbake는 빌드 타임 의존성을 관리하기 위해 DEPENDS 변수 사용
  태스크에 대한 deptask 변수 플래그는 해당 태스크가 실행되기 전에 DEPENDS의 각 항목에 대해 완료해야 하는 태스크

  ```
  do_package[deptask] += "do_packagedata"
  ```

  do_packagedata 태스크는 do_package를 실행하기 전에 완료 되어야 함
  또는 DEPENDS 변수를 무시하고 depends 플래그를 사용해 명시적으로 빌드 타임 의존성 정의

- **런타임 의존성**

  Bitbake는 런타임 의존성을 관리하기 위해 PACKAGES 와 RDEPENDS 변수 사용
  PACKAGES 변수: 레시피에서 생성하는 모든 런타임 패키지 나열
  각 패키지에는 RDEPENDS 런타임 의존성이 있을 수 있음
  태스크에 대한 rdeptask 변수 플래그는 해당 태스크를 실행하기 전에 모든 런타임 의존성에 대해 완료해야 하는 태스크 지정

  ```
  do_package_qa[rdeptask] = "do_packagedata"
  ```

  RDEPENDS의 각 항목에 대한 do_package_data 태스크는 do_package_qa가 실행하기 전에 완료되어야 함
  Rdepends 플래그는 RDEPENDS 변수를 우회할 수 있도록 함으로써 depends 플래그와 유사동작
  차이점은 rdepends 가 빌드 타임 대신 런타임에 적용됨

<변수>

Bitbake 에서 변수의 범위는 변수가 정의된 메타데이터 파일의 유형에 따라 달라짐
레시피 파일(.bb)에 선언된 모든 변수는 로컬 변수
설정 파일(conf) 에 선언된 모든 변수는 전역 변수

- **할당과 확장**

  변수 할당과 확장은 셸에서와 같이 동작
  할당은 명령문이 구문 분석되는 즉시 발생하며 무조건적임
  $ 문자는 변수 확장을 트리거
  둘러싸는 중괄호는 선택 사항이며 바로 뒤에 오는 문자로부터 확장할 변수를 보호하는 역할
  확장 변수는 실수로 단어를 분할하거나 묶는 것을 방지하기 위해 일반적으로 큰 따옴표로 묶음

  ```
  OLDPKGNAME = "dbus-x11"
  PROVIDES_${PN} = "${OLDPKGNAME}"
  ```

  변수 할당이 오른쪽에서 참조되는 경우, 왼쪽의 변수가 확장될 때까지 참조된 변수가 평가되지 않음을 의미
  오른쪽의 값이 시간이 지남에 따라 변경되면 왼쪽의 변수 값도 변경

  조건부 할당은 변수가 구문 분석 시 정의되지 않은 경우에만 변수를 정의

  ```
  PREFERRED_PROVIDER_virtual/kernel ?= "linux-yocto"
  ```

  해당 동작을 원하지 않을 때 재할당이 방지됨

  조건부 할당은 빌드 시스템에 의해 이미 설정됐을 수 있는 변수를 덮어쓰지 않도록 하기 위해 makefile의 맨 위에 사용
  조건부 할당은 나중에 레시피에서 정의되지 않은 변수를 추가하거나 앞쪽에 추가하지 않도록 해줌
  ?=를 사용한 지연 할당은 즉시 할당되는 것이 아니라 구문 분석 프로세스의 마지막에 이뤄진다는 점을 제외하고는 ?=와 동일하게 동작함

  ```
  TOOLCHAIN_TEST_HOST ??= "localhost"
  ```

  이는 변수 이름이 여러 개의 지연 할당의 왼쪽 편에 있으면 마지막 지연 할당문으로 변수가 할당된다는 것을 의미
  다른 형태의 변수 할당은 구문 분석 시 할당의 오른쪽 편을 즉시 강제로 평가

  ```
  target_datadir := "${datadir}"
  ```

  즉시 할당을 위한 :=연산자는 셸이 아니라 make에서 제공된다는 점을 유의

- **앞과 뒤에 추가**

  ```
  CXXFLAGS += "-std=c++11"
  PACKAGES =+ "gdbserver"
  ```

  왼쪽의 값과 오른쪽에서 앞이나 뒤에 추가되는 값 사이에 단일 공백을 삽입
  += 연산자는 문자열 값이 아닌 정수에 적용되는 경우 추가가 아니라 증가를 의미

  ```
  BBPATH .= ":${LAYERDIR}"
  FILESEXTRAPATHS =. "${FILE_DIRNAME}/systemd:"
  ```

  앞과 뒤에 추가 시 단일 공백을 생략하려는 경우 다음 할당 연산자를 사용하면 됨
  앞과 뒤에 추가하는 할당 연산자의 단일 공백 버전은 bitbake 메타데이터 파일 전체에서 사용

- **재정의**

  bitbake는 변수에 앞과 뒤 추가를 위한 대체 구문을 제공 (재정의 구문)

```
CFLAGS_append = " -DSQLITE_ENABLE_COLUMN_METADATA"
PROVIDES_prepend = "${PN} "
```

_append와 _prepend 접미사는 기존 변수의 값을 수정하거나 재정의
단일 공백을 생략하기에 bitbake의 .= 및 =.와 유사하게 동작

OVERRIDES 변수 (meta/conf/bitbake.conf 에 정의): 사용자가 원하는 조건 목록으로 콜론으로 구분돼 있음
이 목록은 동일한 변수의 여러 버전 중에서 선택하는 데 사용되며 각 버전은 서로 다른 접미사로 구별 (다양한 접미사가 조건의 이름과 일치)

bitbake는 또한 특정 항목이 OVERRIDE 목록에 포함돼 있는지 여부에 따라 변수 값 앞과 뒤에 추가하는 작업을 지원

bitbake의 변수 추가는 조건 없이 재정의할 수 있으며, 구문 분석 시점에 할당

- **인라인 파이썬**

bitbake의 @ 기호를 사용하면 변수 내부에 파이썬 코드를 삽입하고 실행 가능
라인 파이썬 표현식은 = 연산자의 왼쪽에 있는 변수가 확장될 때마다 평가
:= 연산 자의 오른쪽에 있는 인라인 파이썬 표현식은 구문 분석 시 한 번만 평가

```
PV = "${@bb.parse.vars_from_file(d.getVar('FILE', False),d)[1] or '1.0'}"
BOOST_MAJ = "${@"_".join(d.getVar("PV").split(".")[0:2])}"
GO_PARALLEL_BUILD ?= "${@oe.utils.parallel_make_argument(d, '-p %d')}"
```

bb와 oe는 BitBake와 OpenEmbedded의 파이썬 모듈에 대한 별칭
d.getVar("PV")는 태스크의 런타임 환경에서 PV 변수 값을 검색하는 데 사용
d 변수는 BitBake가 원래 실행 환경의 복사본을 저장하는 데이터 저장소 개체를 나타냄

<함수>

함수는 bitbake 태스크를 구성하는 요소
셸 또는 파이썬으로 작성되고, .bbclass, .bb, .int 파일 내에 정의

- **셸**

셸에서 작성된 함수는 함수나 태스크로 실행
태스크로 실행되는 함수는 일반적으로 do_ 접두사로 시작하는 이름을 가짐
호스트 배포판에 따라 Bash가 아닐 수 있으므로, Bash 전용 기능을 피해야 함
(확인하기 위해 scripts/verify-bashisms linter를 실행하여 체크)

- **파이썬**

Bitbake는 순수 함수, bitbake 스타일 함수, 익명 함수 세가지 유형의 파이썬 함수 사용 가능

- **순수 파이썬 함수**
  일반 파이썬으로 작성되고 다른 파이썬 코드에 의해 호출

(meta/recipes-connetivity/bluez5/bluez5.inc 예)

```
def get_noinst_tools_paths (d, bb, tools):
    s = list()
    bindir = d.getVar("bindir")
    for bdp in tools.split():
        f = os.path.basename(bdp)
        s.append("%s/%s" % (bindir, f))
    return "\n".join(s)
```

위 함수는 실제 파이썬 함수처럼 매개변수를 사용
a. 데이터 저장소 개체를 사용할 수 없으므로 이 개체를 함수 매개변수로 전달 해야 함
b. Os 모듈은 자동으로 사용 가능하므로 가져오거나 전달할 필요가 없음

순수 파이썬 함수는 @ 기호를 사용한 셸 변수에 할당된 인라인 파이썬에서 호출할 수 있음

- **Bitbake 스타일 파이썬 함수**

Bitbake 스타일 파이썬 함수 정의는 파이썬의 네이티브 def 키워드 대신 파이썬 키워드로 표시
Bitbake 자체의 내부 함수들을 포함해 다른 파이썬 함수에서 bb.build.exec_func() 를 호출하면서 실행
Bitbake 스타일 함수는 매개변수를 사용하지 않음

(meta/classes/sign_rpm.bbclass)의 bitbake 스타일 파이썬 함수 정의

```
python sign_rpm () {
    import glob
    from oe.gpg_sign import get_signer

    signer = get_signer(d, d.getVar('RPM_GPG_BACKEND'))
    rpms = glob.glob(d.getVar('RPM_PKGWRITEDIR') + '/*')

    signer.sign_rpms(rpms,
                     d.getVar('RPM_GPG_NAME'),
                     d.getVar('RPM_GPG_PASSPHRASE'),
                     d.getVar('RPM_FILE_CHECKSUM_DIGEST'),
                     int(d.getVar('RPM_GPG_SIGN_CHUNK')),
                     d.getVar('RPM_FSK_PATH'),
                     d.getVar('RPM_FSK_PASSWORD'))
}
sign_rpm[vardepsexclude] += "RPM_GPG_SIGN_CHUNK"

do_package_index[depends] += "signing-keys:do_deploy"
do_rootfs[depends] += "signing-keys:do_populate_sysroot"

PACKAGE_WRITE_DEPS += "gnupg-native"
```

```
python sign_rpm () {
    import glob
    from oe.gpg_sign import get_signer

    signer = get_signer(d, d.getVar('RPM_GPG_BACKEND'))
    rpms = glob.glob(d.getVar('RPM_PKGWRITEDIR') + '/*')

    signer.sign_rpms(rpms,
                     d.getVar('RPM_GPG_NAME'),
                     d.getVar('RPM_GPG_PASSPHRASE'),
                     d.getVar('RPM_FILE_CHECKSUM_DIGEST'),
                     int(d.getVar('RPM_GPG_SIGN_CHUNK')),
                     d.getVar('RPM_FSK_PATH'),
                     d.getVar('RPM_FSK_PASSWORD'))
}
sign_rpm[vardepsexclude] += "RPM_GPG_SIGN_CHUNK"

do_package_index[depends] += "signing-keys:do_deploy"
do_rootfs[depends] += "signing-keys:do_populate_sysroot"

PACKAGE_WRITE_DEPS += "gnupg-native"
ysj@build-master:~/s32g-bsp-300-master/sources/poky/meta/classes$
```

- **익명 파이썬 함수**

  Bitbake 스타일 함수와 유사하지만 구문 분석 중에 실행
  익명 함수는 먼저 실행되기 때문에 변수를 초기화 하거나 기타 설정을 초기화하는 것과 같이 rrnans 분석 시 수행할 수 있는 작업에 적합
  익명 함수 정의는 _anonymous 함수 이름을 사용하거나 혹은 사용하지 않고 작성

  익명 함수 내 변수를 설정하면 해당 함수가 실행될 때 전역 데이터 저장소 개체를 통해 다른 함수 사용 가능