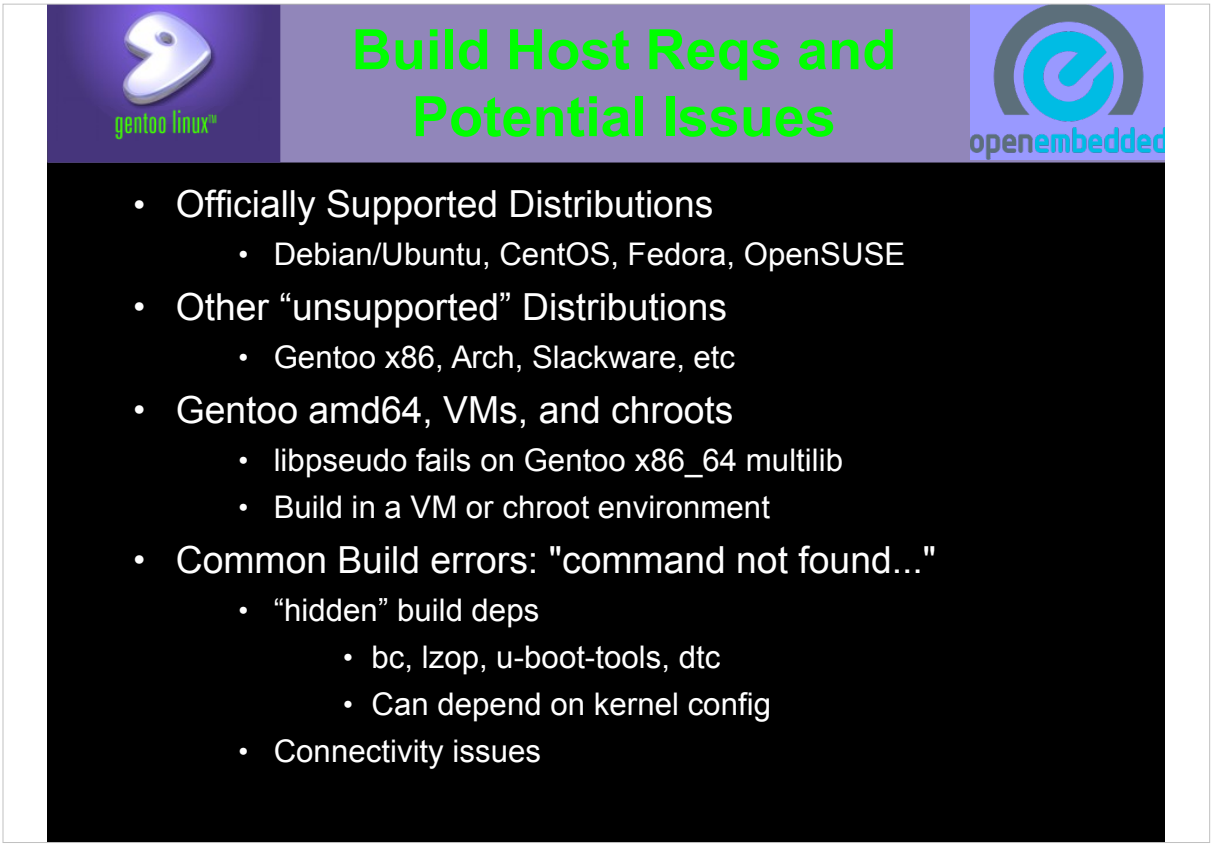
Upstream docs:

Quick Start:
http://tinyurl.com/yocto-1-7
Reference:
https://www.yoctoproject.org/documentation
Project wiki:
http://openembedded.org/wiki/Main_Page
Training:
https://www.yoctoproject.org/training/kernel-lab
Git repos:
https://www.yoctoproject.org/downloads
https://github.com/openembedded/meta-openembedded
Vendors:
http://beagleboard.org/project/yocto-project/
https://community.freescale.com/docs/DOC-1616
Other:
https://github.com/sarnold/meta-alt-desktop-extras
http://www.vctlabs.com/archives.html



See the getting-started guide and wiki for details; essentially you need python, git, tar, and the rest of the “normal” development tools and libraries, plus a few others. For example, a Gentoo x86 system with an ARM cross-compiler and U-boot tools should be almost ready to go:

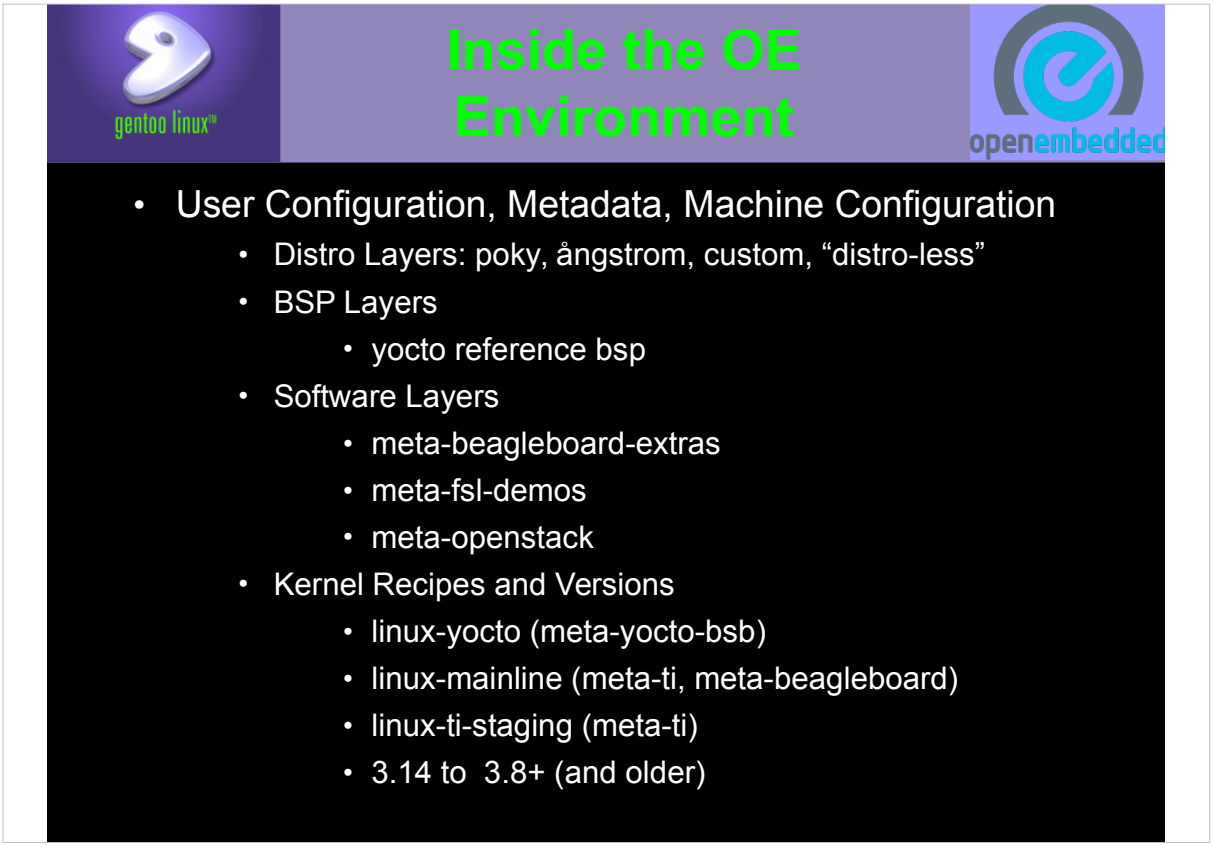
http://tinyurl.com/yocto-1-7

http://www.openembedded.org/wiki/OEandYourDistro

The wiki page above includes details for some of the “unsupported” distributions.

Other general considerations include disk space (you need plenty of it) and VM support (ie, KVM, qemu, libvirt, etc).

Network problems such as dropouts, bad name resolution, etc, can stop a build but you can pre-fetch required source packages and share downloads and cache data between builds.



Typical (manual) directory layout has poky as the top-level directory, with base BSP and additional layers inside.

The default environment script creates local build directories at the same level, however, user-configuration options are provided to specify paths for downloads, build output, and shared cache data (by default each build tree is self-contained).

Sharing downloads and cache data between builds is a good way to both speed up builds/rebuilds and save space.

You can add additional software layers as needed, however only one BSP layer should be enabled for a given build.

The two main local config files are conf/{bblayers.conf,local.conf} and local.conf is probably the easiest place to keep your custom build settings unless you're creating your own BSP or software layer.

Useful config options in local.conf include:

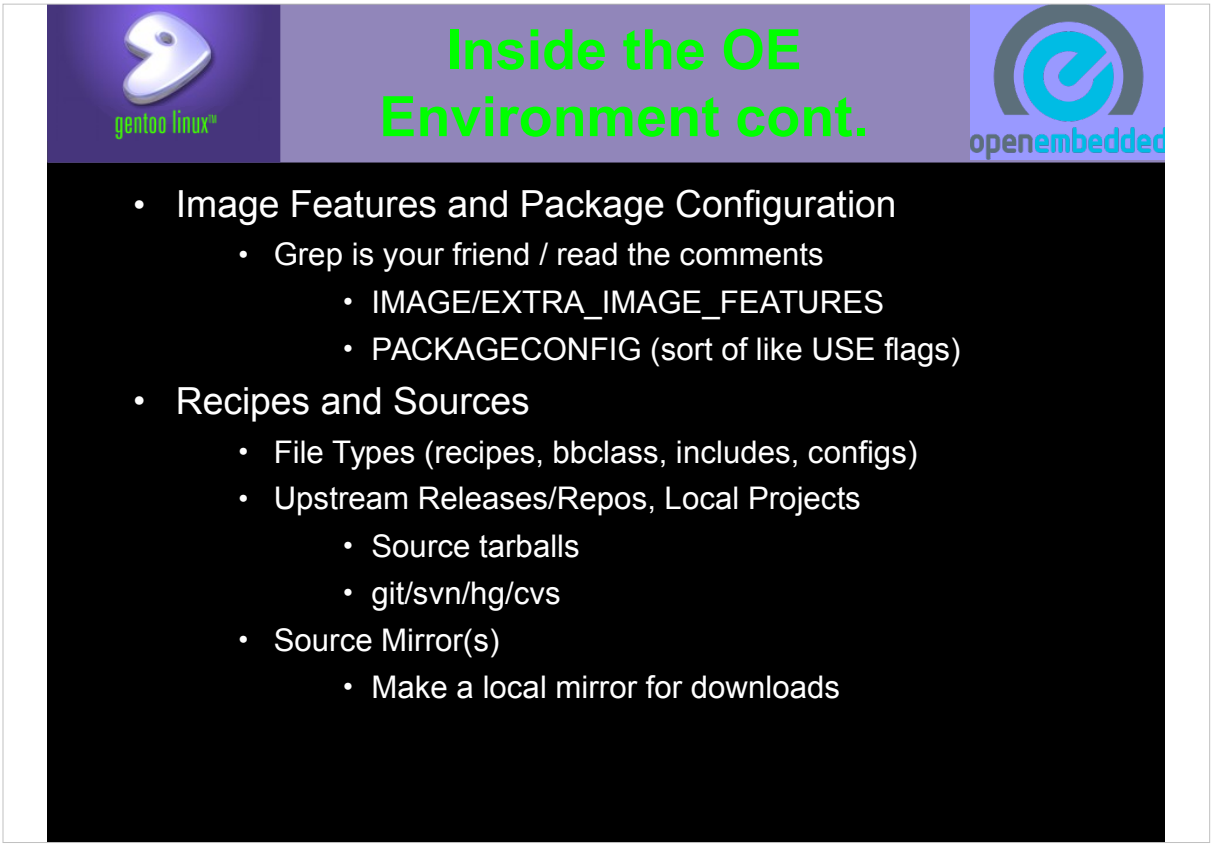
PREFERRED_VERSION

PREFERRED_PROVIDER

DISTRO_FEATURES

IMAGE_FEATURES / EXTRA_IMAGE_FEATURES

PACKAGECONFIG



build-foo/conf/bblayers.conf

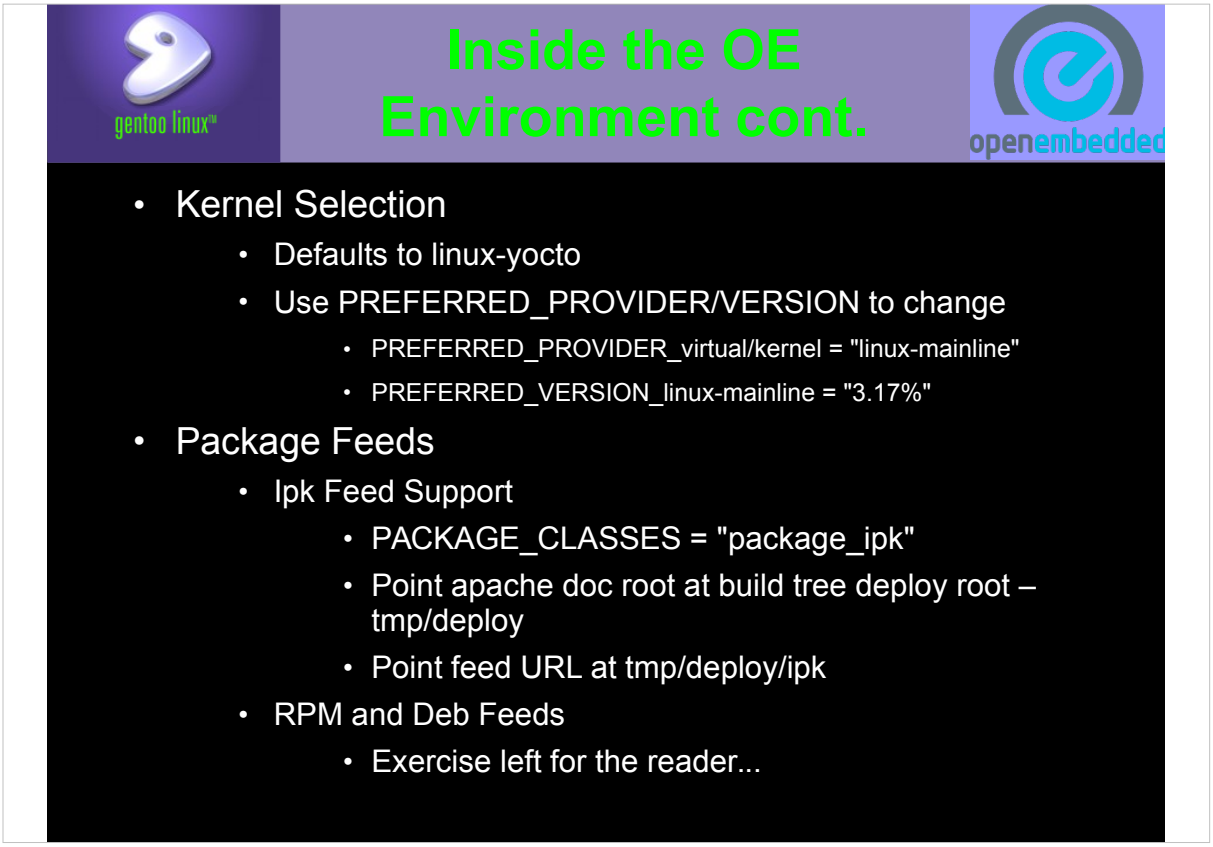
- Enable new metadata (software) layers
- Specify a BSP layer
- Set the full path to poky root

build-foo/conf/local.conf

- Set INHERIT options
- Set PKG_CLASS and PACKAGECONFIG options
- Set LICENSE options
- Set MACHINE and IMAGE options

Metadata file types include package and image recipes (.bb and .bbappend), include files for both recipes and configuration (.inc), configuration files (.conf), and class files (.bbclass). All of them are used to create and extend layers.

Package recipes can include everything from local files to remote git repos in their SRC_URIs (some control over remote fetching is provided via MIRROR settings).



References:

http://www.yoctoproject.org/docs/1.7.1/kernel-dev/kernel-dev.html

https://www.yoctoproject.org/training/kernel-lab

There are many ways to “skin” the kernel, depending on the specific BSP and kernel recipe:

1) KERNEL_FEATURES (poky-lsb distro config file)
2) Config parameters (linux-raspberrypi/linux.inc)
3) Config “fragment” (kernel recipe/.bbappend SRC_URI)
4) Custom defconfig (kernel recipe/.bbappend SRC_URI)
5) Kernel patches (kernel recipe/.bbappend SRC_URI)
6) Custom recipes or .bbappends (meta-mybsp)

When modifying kernel recipes, adding fragments/patches, etc, bitbake will normally detect the changes and rebuild the recipe. For example, the new kernel can be rebuilt, deployed, and then run with the following commands:

```
$ bitbake virtual/kernel -c deploy
$ runqemu tmp/deploy/images/bzImage-blah.bin \
  tmp/deploy/images/core-image-minimal-blah.ext3
```

References:

http://www.openembedded.org/wiki/Bitbake_cheat_sheet
https://community.freescale.com/docs/DOC-94953
http://tinyurl.com/bitbake-1-6

To keep an image build going after non-critical failure:

```
$ bitbake -k <recipe_name>
```

To list the contents of your build environment (can be large):

```
$ bitbake -e core-image-minimal
$ bitbake -e redis-ipc
```

To open a shell in the package source tree with the correct build environment:

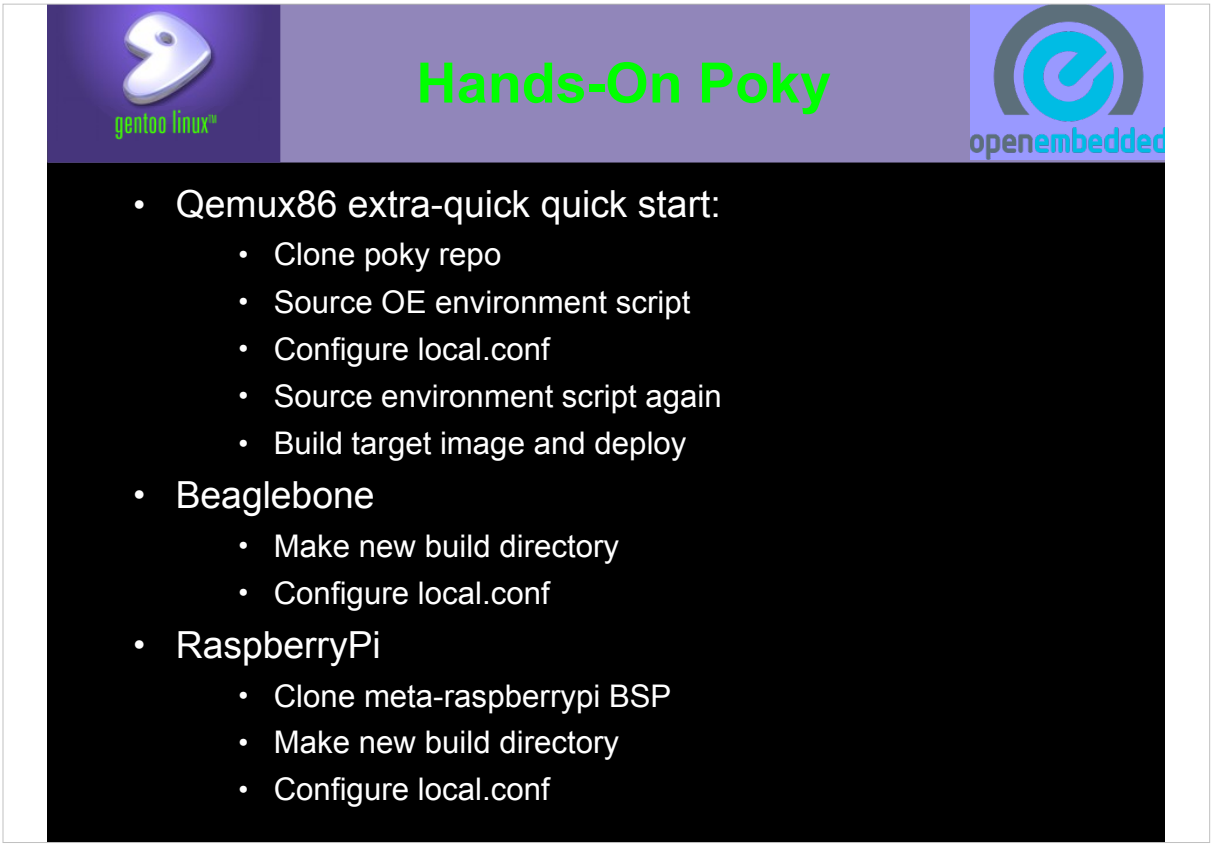
```
$ bitbake <recipe_name> -c devshell
```

To list the available tasks for a given build target:

```
$ bitbake <recipe_name> -c listtasks
```

To generate an SDK specific to a given image target:

```
$ bitbake <image_name> -c populate_sdk
```



Clone poky, check out release/master branch

```
$ git clone http://git.yoctoproject.org/git/poky
$ cd <poky-dir> && git checkout master
$ source oe-init-build-env build-x86
```

Changes to <poky-dir>/<build-dir>/conf/local.conf:

```
MACHINE = "qemux86"
DL_DIR ?= "/home/user/downloads"
SSTATE_DIR ?= "/home/user/shared-state/poky-std"
PACKAGE_CLASSES ?= "package_ipk"
INHERIT += "rm_work"
INHERIT += "buildhistory"
INHERIT += "toaster"
DISTRO_FEATURES_append = " pam"
```

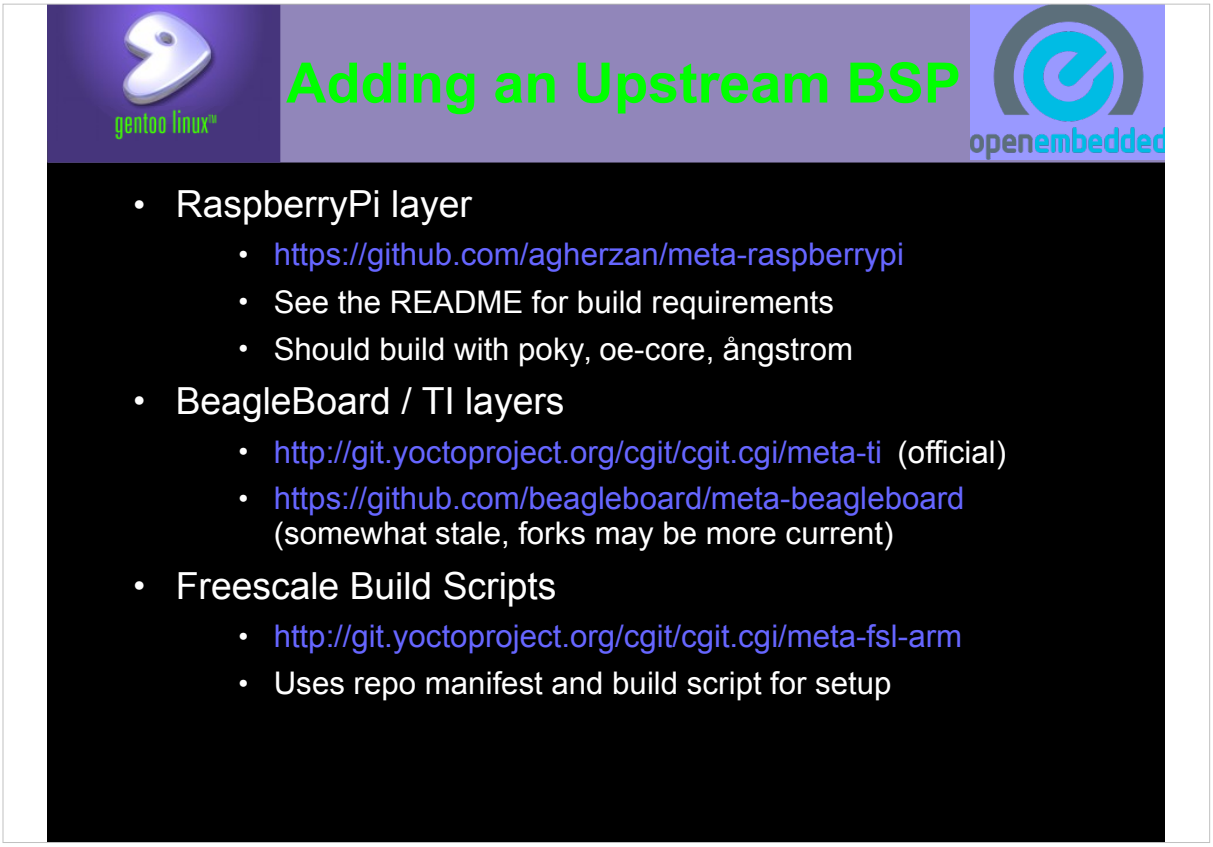
```
$ cd <poky-dir> && source oe-init-build-env build-x86
$ bitbake core-image-minimal
$ runqemu /path/to/kernel.bin /path/to/image.ext3
```

Official Yocto Project Quick Start Guide

http://tinyurl.com/yocto-1-7

OpenEmbedded OE-Core Quick Start

http://openembedded.org/wiki/OE-Core_Standalone_Setup



Since the meta-yocto-bsp layer supports the first two machines we built for this crash course, the defaults in bblayers.conf should work fine for the basic demo images and yocto BSP machines (eg, qemux86, beaglebone, etc). Other machines with Yocto support will have their own BSP and possibly application layers, eg, RaspberryPi.

The typical practice for Yocto-compliant layers is to document the build and layer requirements in the readme; notice it supports multiple OE build configurations, but only one is typically tested upstream (ie, poky + meta-raspberrypi).

The Yocto beaglebone support is both basic and somewhat less than current, so feel free to add the meta-ti layer and try their kernel recipes with support for the TI vendor blobs, etc.

The FreeScale Yocto support is somewhat different in that they do not document a “manual” layer setup as above, but do provide a repo manifest and set of build scripts that mostly automates the initial cloning and setup for building the fsl “community” layers for some of their iMX.6-based machines (eg, Wandboard).

References:

http://www.yoctoproject.org/docs/latest/kernel-dev/kernel-dev.html

http://www.yoctoproject.org/docs/latest/bsp-guide/bsp-guide.html

http://www.yoctoproject.org/docs/latest/adt-manual/adt-manual.html

Different kernel recipes from various BSPs can take somewhat different approaches to kernel builds and configuration (see the linux-raspberrypi vs. linux-yocto recipes). The following config fragment method is from the latest Yocto Kernel Dev Guide.

1) Complete a kernel build at least through the configuration task:

```
$ bitbake linux-yocto -c kernel_configme -f
```

2) Run the menuconfig command:

```
$ bitbake linux-yocto -c menuconfig
```

3) Run the diffconfig command to prepare a configuration fragment. fragment.cfg will be in the ${WORKDIR} directory:

```
$ bitbake linux-yocto -c diffconfig
```

The diffconfig command creates a file that is a list of kernel CONFIG_ assignments.

With recipes, less is more. See core-image-minimal.bb vs. core-image-sato.bb and <poky-dir>/meta-skeleton for examples. Don't copy a recipe - do make a .bbappend instead. Don't replicate an existing task - do use an append/prepend to add your changes instead. *Inherit, include,* or *require* as needed.

So what did we do to update the beaglebone kernel?

1) We made changes to local.conf

```
COMPATIBLE_MACHINE_beaglebone = "beaglebone"
PREFERRED_VERSION_linux-yocto = "3.17.%"
```

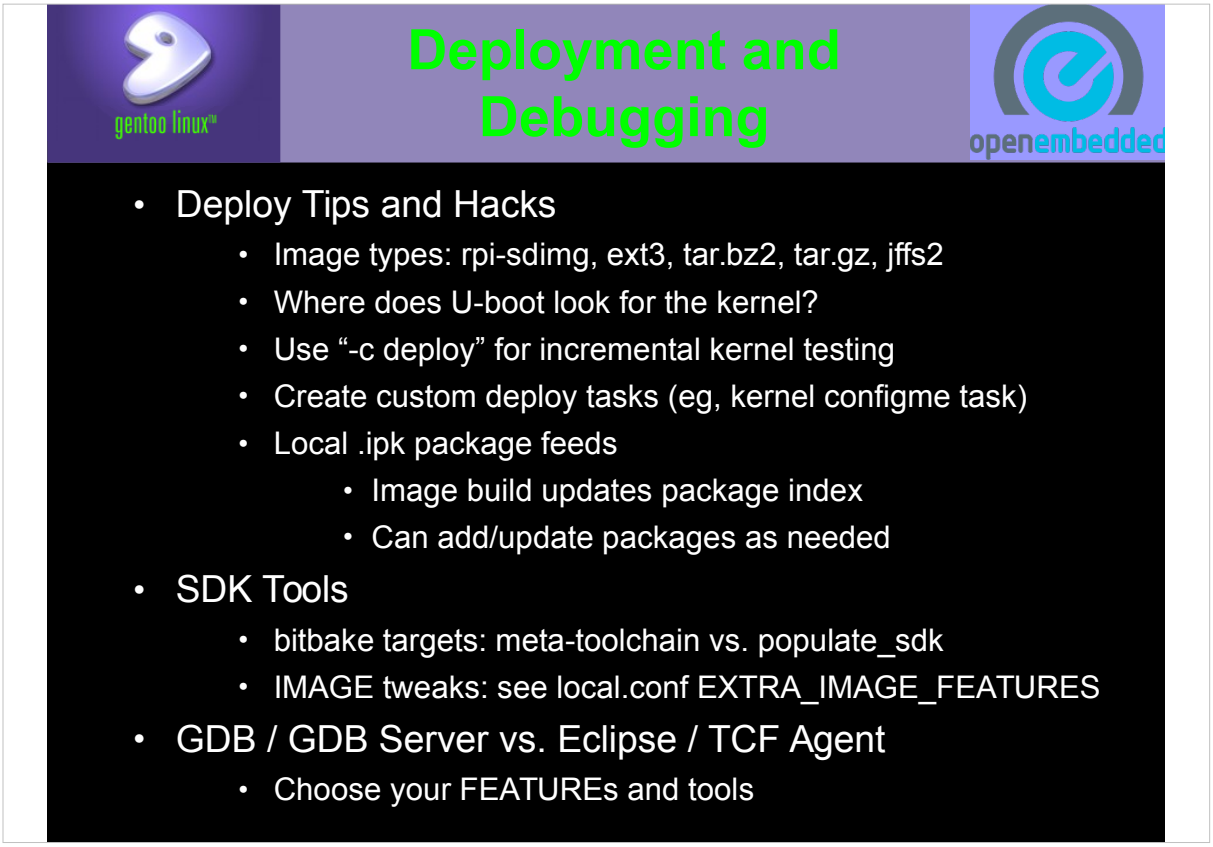
But, the linux-yocto recipe only sets qemu-compatible machines, and we also need to change the kernel configuration, so:

2) We created a new config fragment and .bbappend for linux-yocto_3.17.bb:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
SRC_URI_append_beaglebone = " file://ohci.cfg "
KBRANCH_beaglebone = "standard/beaglebone"
SRCREV_machine_beaglebone ?= "0409b1fbed221e61212e17b7637fa54f908d83f6"
COMPATIBLE_MACHINE_beaglebone = "beaglebone"
```

The config fragment in this case simply enables the OHCI OMAP support:

```
CONFIG_USB_OHCI_HCD=y
CONFIG_USB_OHCI_HCD_OMAP3=y
```



References:

http://tinyurl.com/local-pkg-feed

http://wiki.chumby.com/index.php?title=Advanced_OpenEmbedded

Different BSPs add/modify .bbclass files to provide additional image types (such as the RaspberryPi SDCard image type). As seen, the base beaglebone build produces both a jffs2 and tar.bz2 rootfs images, plus kernel, dtb, and u-boot files. In this case you must follow the TI version of the u-boot deploy dance:

1) Copy MLO first, then u-boot.img to boot **partition**

2) untar rootfs to root partition (use -p switch)

3) Copy zImage and am335x-boneblack.dtb to /boot **directory**

You can also create your own package feed by pointing your web server at:

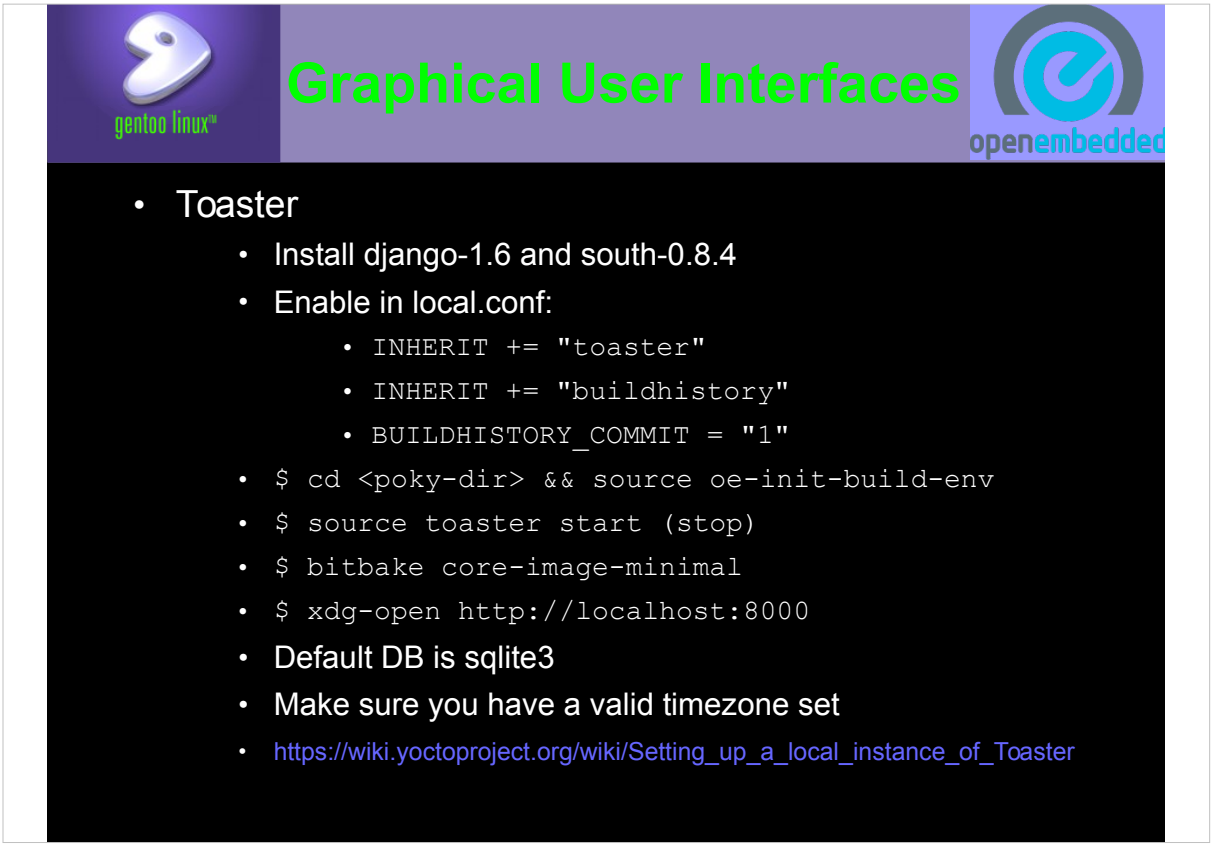
```
<poky-dir>/build/tmp/deploy/ipk
```

And adding this to local.conf:

```
FEED_DEPLOYDIR_BASE_URI = "http://ip-address/<machine>/ipk"
```

From your running device as root, try:

```
# opkg update && opkg list-installed
```



References:

https://www.yoctoproject.org/documentation/toaster-manual-17

https://wiki.yoctoproject.org/wiki/Toaster

How to use a GUI:

1) Click stuff

2) Scroll

3) Click more stuff