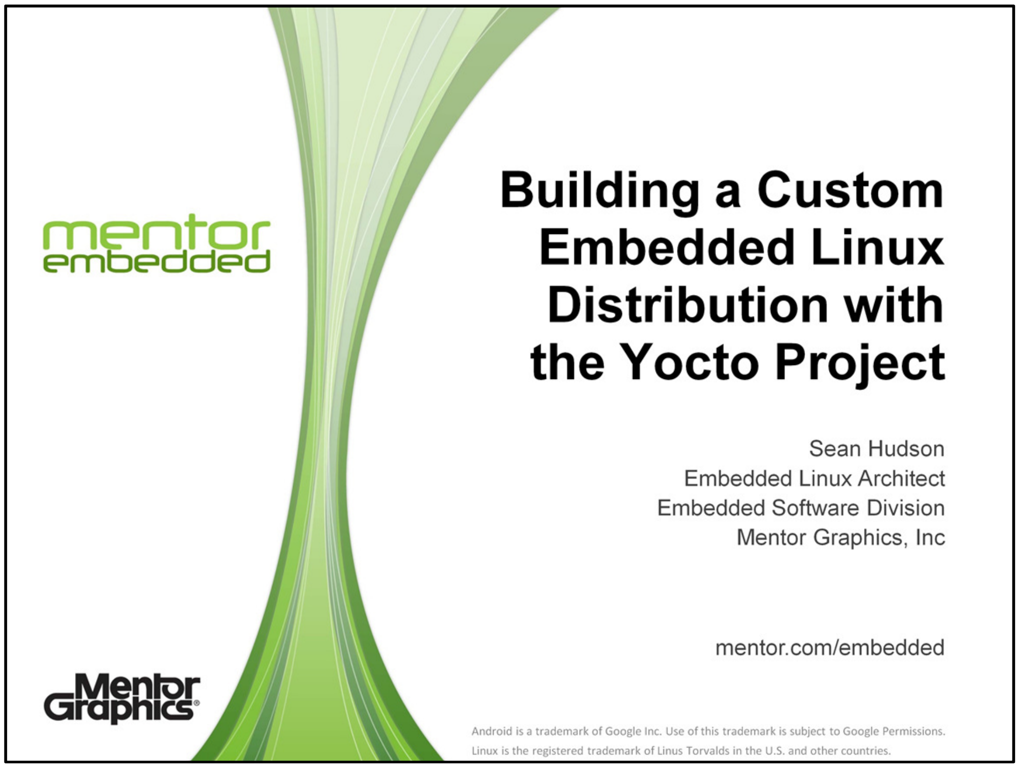
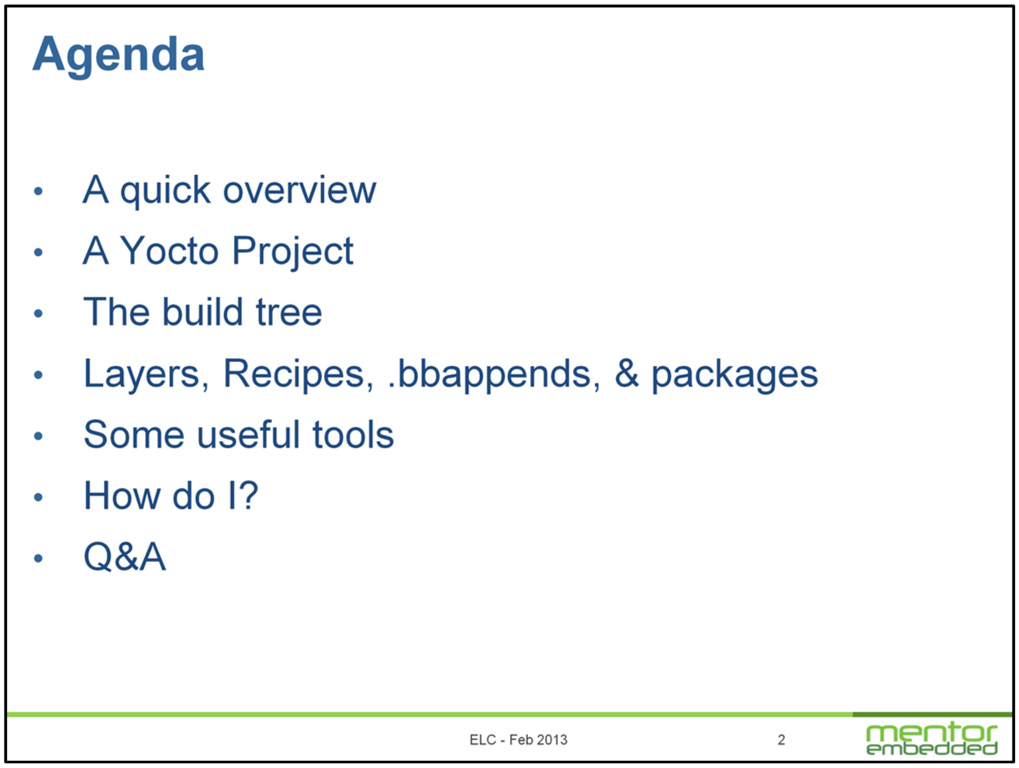


Building a Custom Embedded Linux Distribution with the Yocto Project

Sean Hudson
Embedded Linux Architect
Embedded Software Division
Mentor Graphics, Inc

mentor.com/embedded



Agenda

- A quick overview
- A Yocto Project
- The build tree
- Layers, Recipes, .bbappends, & packages
- Some useful tools
- How do I?
- Q&A

ELC - Feb 2013 2

The challenge for this presentation

1. Building an embedded Linux distribution from source is a complex task
2. The Yocto Project is a collection of powerful tools with lots of complexity of their own

A word of thanks

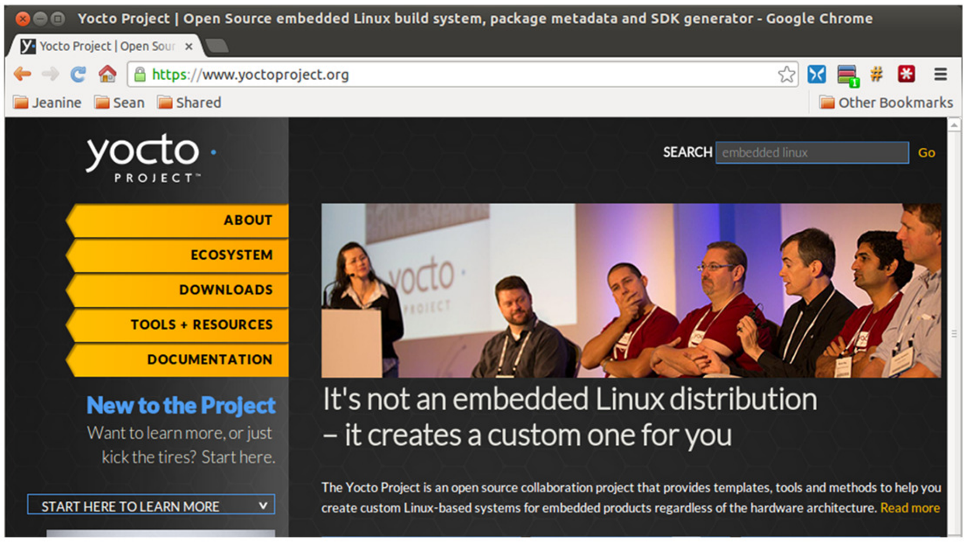
This presentation is built on the work of several others including: Chris Hallinan, Chris Larson, Robert P.J. Day, Khem Raj and everyone that's contributed to the documents in OE and YP.

Setting expectations

There is too much material to cover in a 50 minute presentation. So, in this presentation, I am going to make a quick survey of the basics and highlight the "big pieces". Next, I will highlight some useful tools and lastly, I will provide a set of quick "how to" steps for some common tasks.

Important to highlight the excellent documentation that can be found on the project site: https://www.yoctoproject.org/documentation/current

What is the Yocto Project?


Yocto Project | Open Source embedded Linux build system, package metadata and SDK generator - Google Chrome

https://www.yoctoproject.org

It's not an embedded Linux distribution – it creates a custom one for you

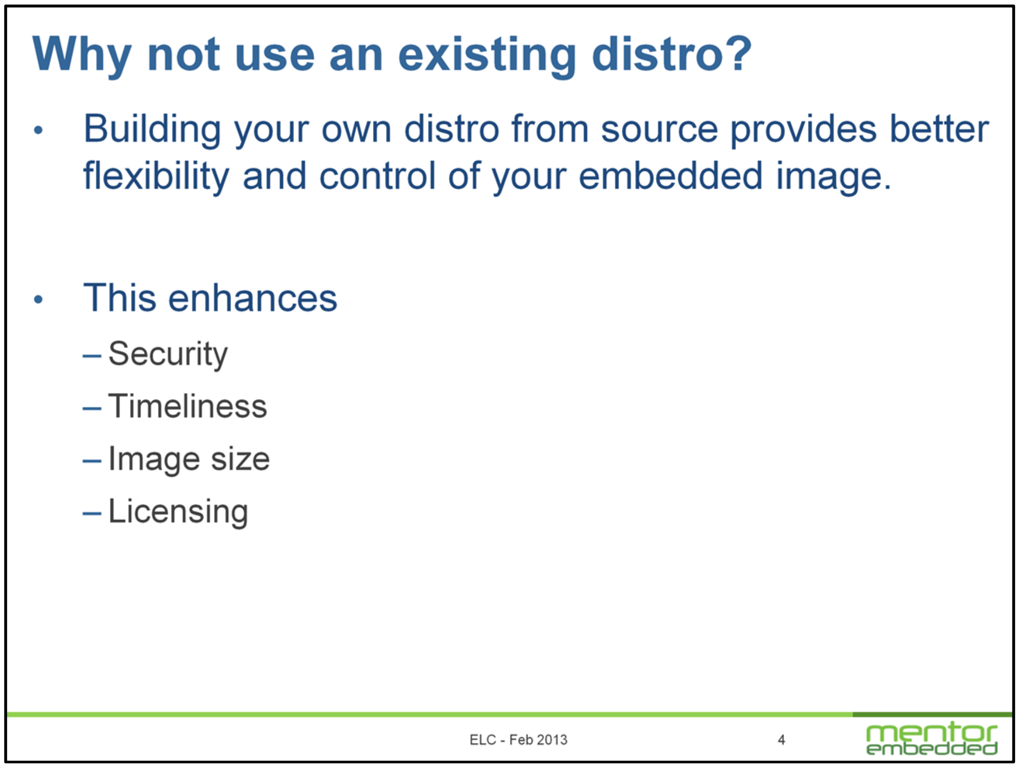
The Yocto Project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products regardless of the hardware architecture. Read more

It's not an embedded Linux distribution – it creates a custom one for you

The Yocto Project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products regardless of the hardware architecture.

Put another way:
The Yocto Project tries to provide the basic pieces for building an embedded Linux distribution. These pieces add little value for companies, but are necessary to build an end product. In short, the project should allow developers to focus on the features that matter to *their* customers.

Other points to make:
Customers can be internal and external.



Why not use an existing distro?

- Building your own distro from source provides better flexibility and control of your embedded image.
- This enhances
  - Security
  - Timeliness
  - Image size
  - Licensing

ELC - Feb 2013 4

Using an existing distro, e.g. Debian or Fedora is certainly a valid choice, but limits flexibility

There are reference distributions, e.g. Angstrom & Poky, however, Angstrom provides a binary feed and is primarily geared at enabling hobbyist boards.

Poky is mainly geared towards testing the YP build system and is therefore not as stable.

<Plug for Mentor Embedded Linux>

Regardless of the choice, building a distro from source enhances:

Security

– Since source code is visible, can tell what code is doing and prevent malicious code

– Also enables ability to patch source quickly when exploits are discovered (see timeliness)

Licensing

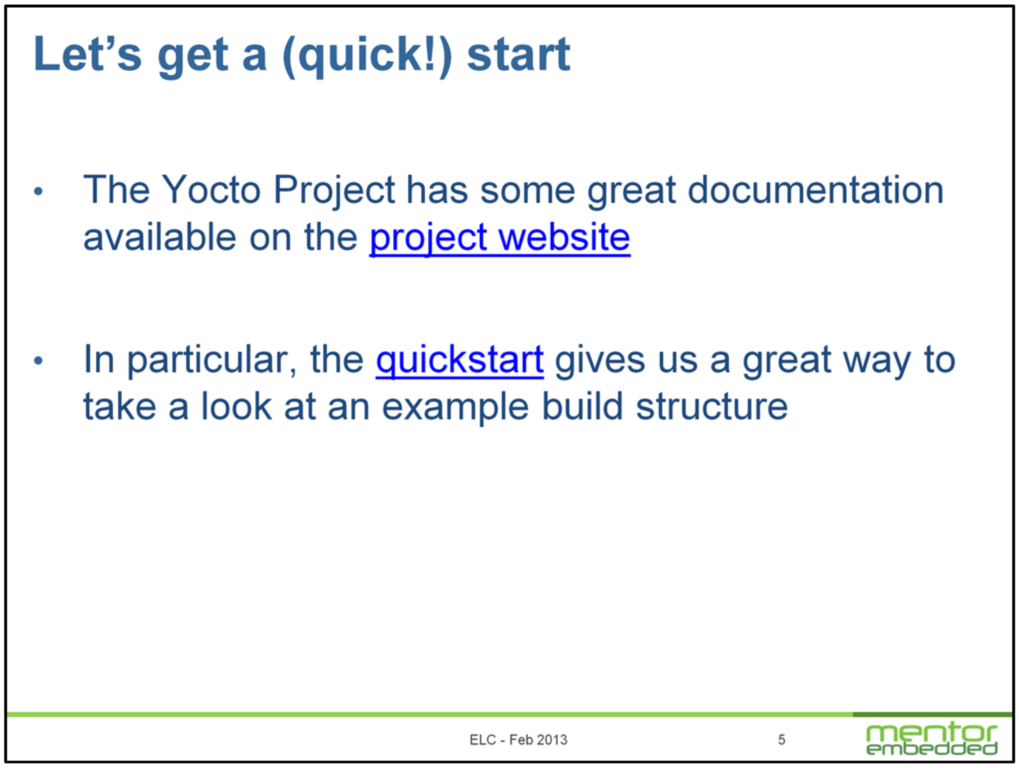
– Provides discreet control over the source going into the image, so licensing can be known/controlled

Tweaking/Image size

– Provides complete control over the image and therefore allows for optimizations and tweaking that may not otherwise be possible

Timeliness

– Reduced dependency on outside packagers, e.g. Ubuntu, to fix a problem. Instead, can incorporate new features and fixes on personal schedule



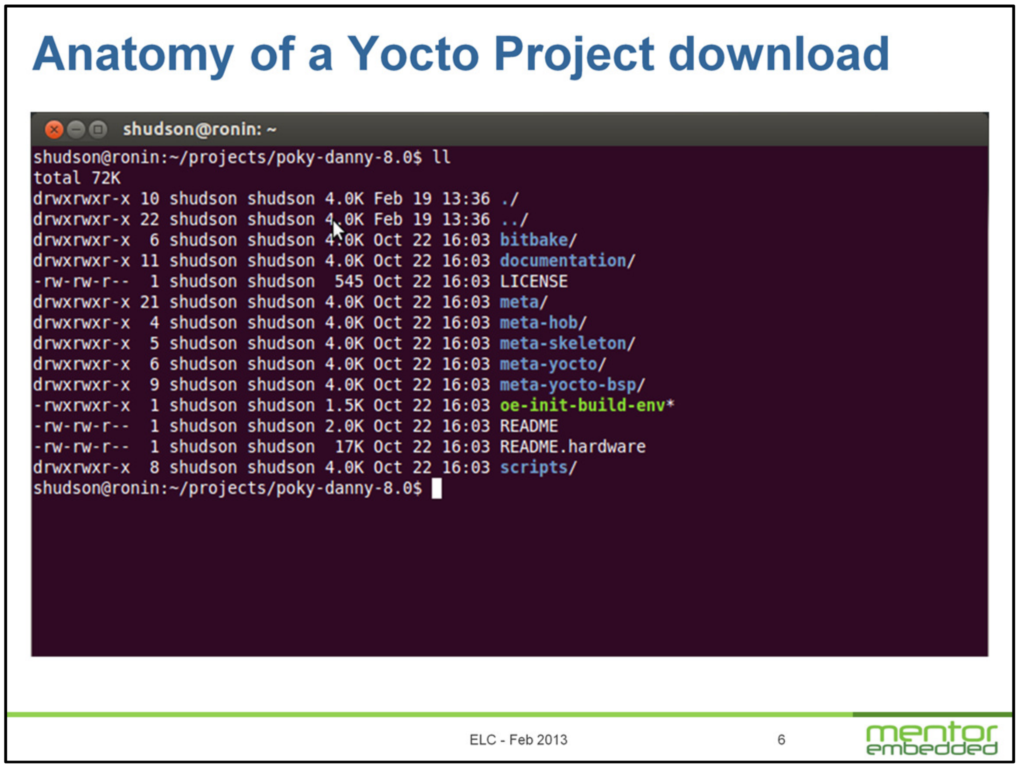
Let's get a (quick!) start

- The Yocto Project has some great documentation available on the project website
- In particular, the quickstart gives us a great way to take a look at an example build structure

ELC - Feb 2013 5

http://www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.html

Following the recipe of the quick start is a fun way to get started and will give us a good initial tree to break down further.



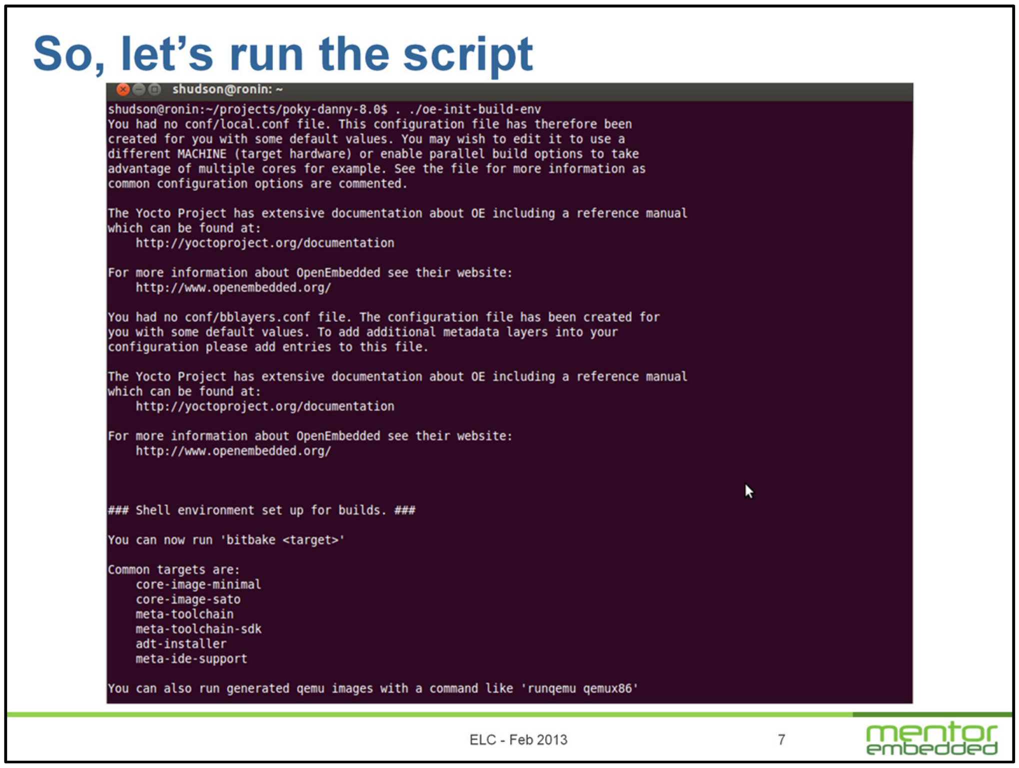
Anatomy of a Yocto Project download

```
shudson@ronin: ~
shudson@ronin:~/projects/poky-danny-8.0$ ll
total 72K
drwxrwxr-x 10 shudson shudson 4.0K Feb 19 13:36 ./
drwxrwxr-x 22 shudson shudson 4.0K Feb 19 13:36 ../
drwxrwxr-x  6 shudson shudson 4.0K Oct 22 16:03 bitbake/
drwxrwxr-x 11 shudson shudson 4.0K Oct 22 16:03 documentation/
-rw-rw-r--  1 shudson shudson  545 Oct 22 16:03 LICENSE
drwxrwxr-x 21 shudson shudson 4.0K Oct 22 16:03 meta/
drwxrwxr-x  4 shudson shudson 4.0K Oct 22 16:03 meta-hob/
drwxrwxr-x  5 shudson shudson 4.0K Oct 22 16:03 meta-skeleton/
drwxrwxr-x  6 shudson shudson 4.0K Oct 22 16:03 meta-yocto/
drwxrwxr-x  9 shudson shudson 4.0K Oct 22 16:03 meta-yocto-bsp/
-rwxrwxr-x  1 shudson shudson 1.5K Oct 22 16:03 oe-init-build-env*
-rw-rw-r--  1 shudson shudson 2.0K Oct 22 16:03 README
-rw-rw-r--  1 shudson shudson  17K Oct 22 16:03 README.hardware
drwxrwxr-x  8 shudson shudson 4.0K Oct 22 16:03 scripts/
shudson@ronin:~/projects/poky-danny-8.0$
```

ELC - Feb 2013 6

Highlight each of the directories and what they do.
Bitbake – is the build engine
Documentation – duh!
Meta-* - metadata – the meat! - critical information that drives the whole process
Scripts – “glue” pieces that makes things work, including wrappers, e.g. runqemu

Point out the layers, but save a deeper discussion for later.
Point out the script and explain what it does, leading to the next slide.

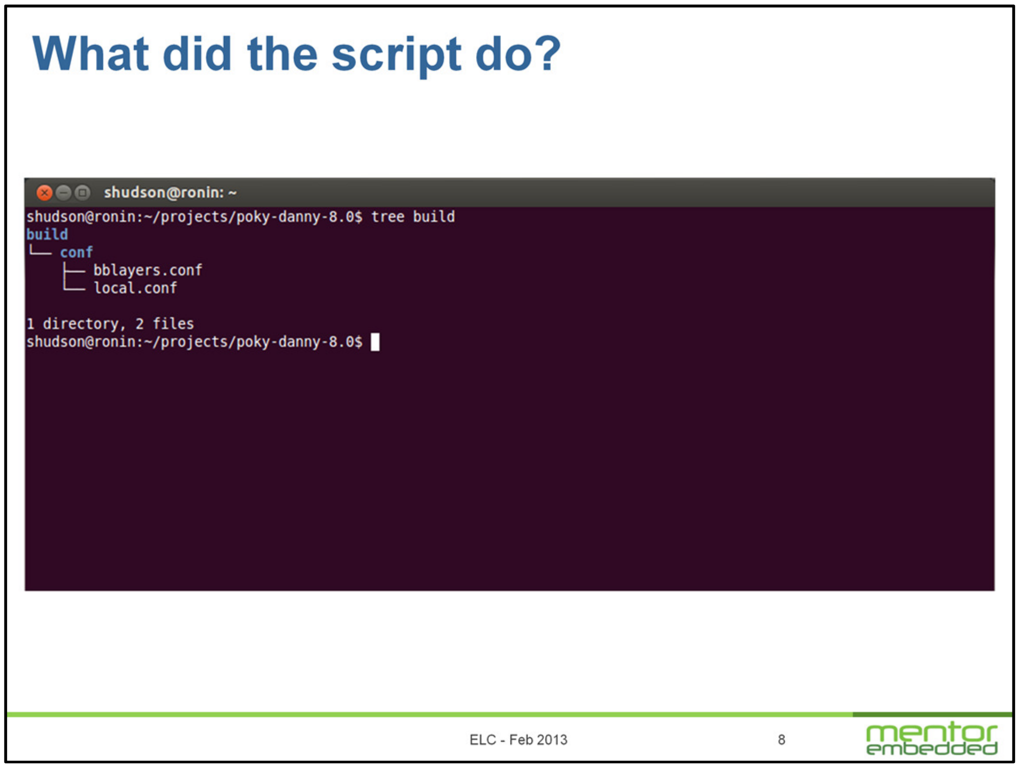


So, let's run the script

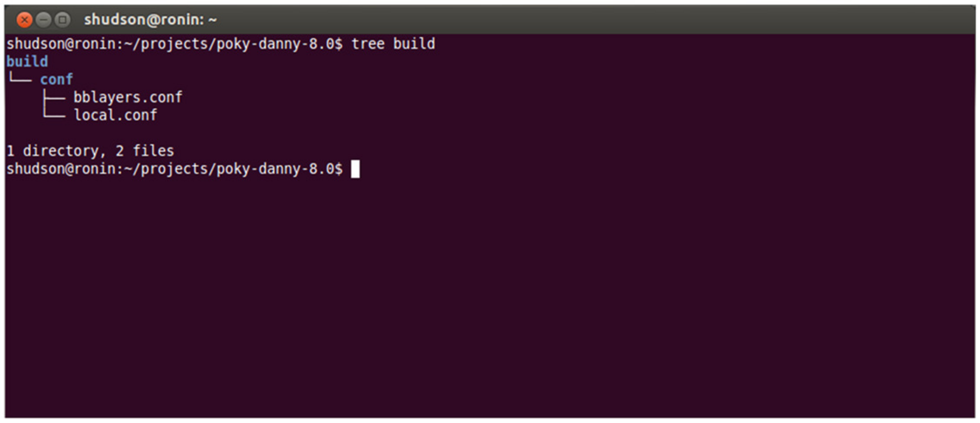
```
shudson@ronin:~/projects/poky-danny-8.0$ . ./oe-init-build-env
You had no conf/local.conf file. This configuration file has therefore been
created for you with some default values. You may wish to edit it to use a
different MACHINE (target hardware) or enable parallel build options to take
advantage of multiple cores for example. See the file for more information as
common configuration options are commented.

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

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

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

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

For more information about OpenEmbedded see their website:
    http://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-toolchain-sdk
    adt-installer
    meta-ide-support

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

ELC - Feb 2013 7

Initial script run:
Point out the bitbake targets, briefly describing each and point out the runqemu command



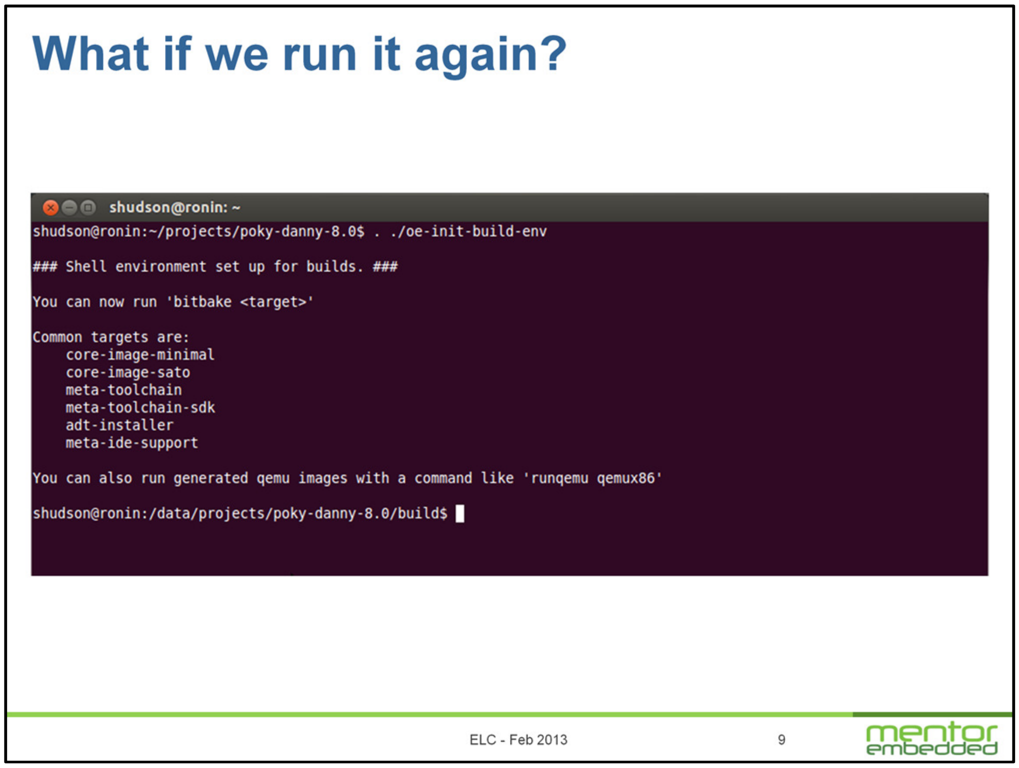
What did the script do?



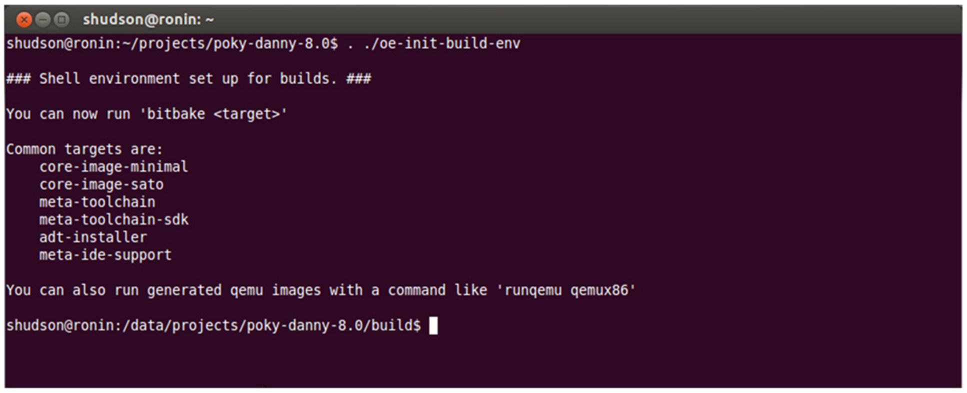
```
shudson@ronin:~/projects/poky-danny-8.0$ tree build
build
└── conf
    ├── bblayers.conf
    └── local.conf

1 directory, 2 files
shudson@ronin:~/projects/poky-danny-8.0$ 
```

ELC - Feb 2013 8

Created the initial build tree and seeded it with a starting local.conf
Show the initial build tree, including the local.conf.
Highlight the bblayers.conf and defer it again.
Pretty boring, as yet.



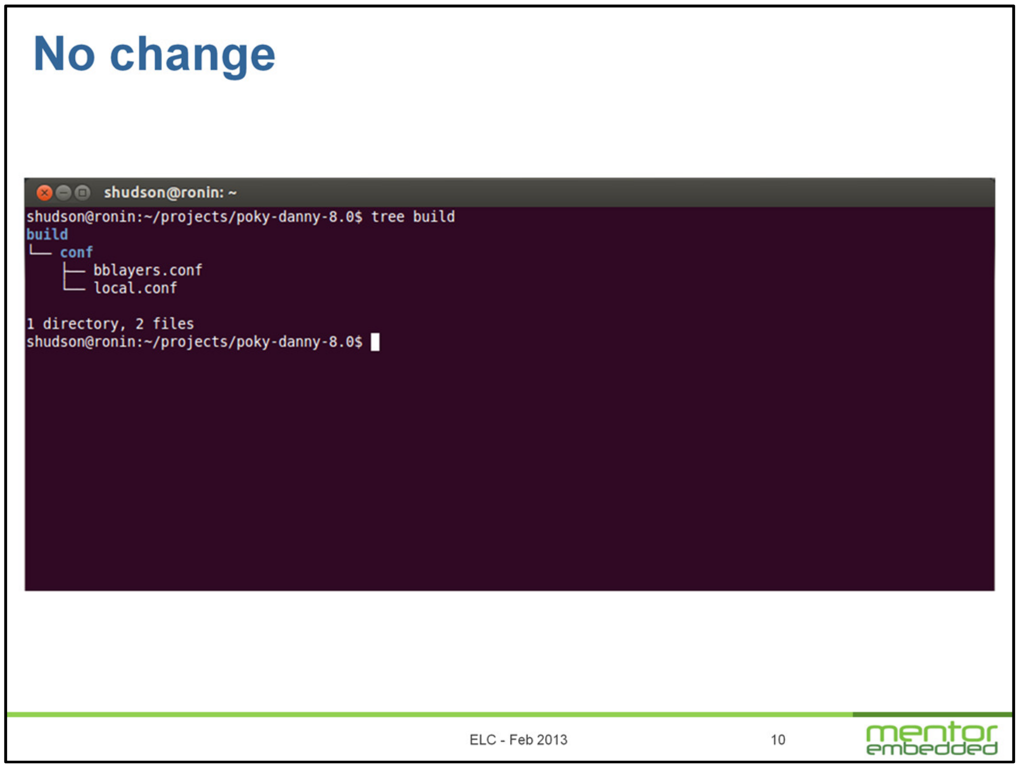
What if we run it again?



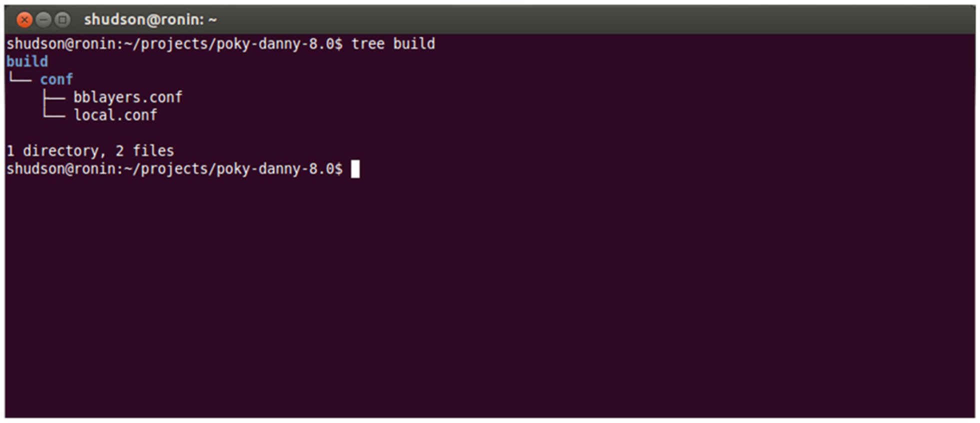
```
shudson@ronin: ~
shudson@ronin:~/projects/poky-danny-8.0$ . ./oe-init-build-env

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

You can now run 'bitbake <target>'

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

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

shudson@ronin:/data/projects/poky-danny-8.0/build$ █
```

ELC - Feb 2013 9

Sets the path and a few ENV variables.
Non-destructive to re-run, in fact, needed to set up the environment.
Observe that the last bit is the same and explain the env variables quickly, e.g. BB_EXTRA_WHITE



No change

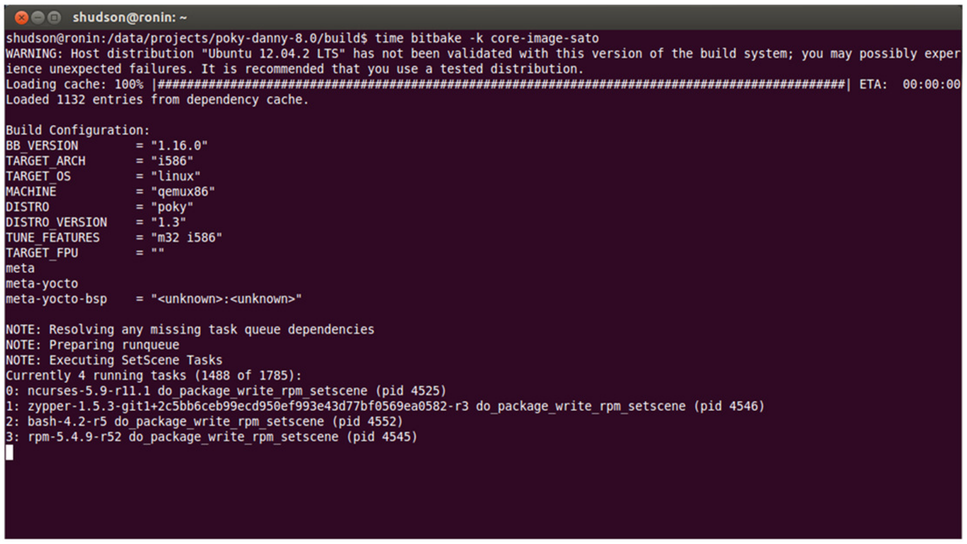


```
shudson@ronin:~/projects/poky-danny-8.0$ tree build
build
└── conf
    ├── bblayers.conf
    └── local.conf

1 directory, 2 files
shudson@ronin:~/projects/poky-danny-8.0$ 
```

ELC - Feb 2013 10

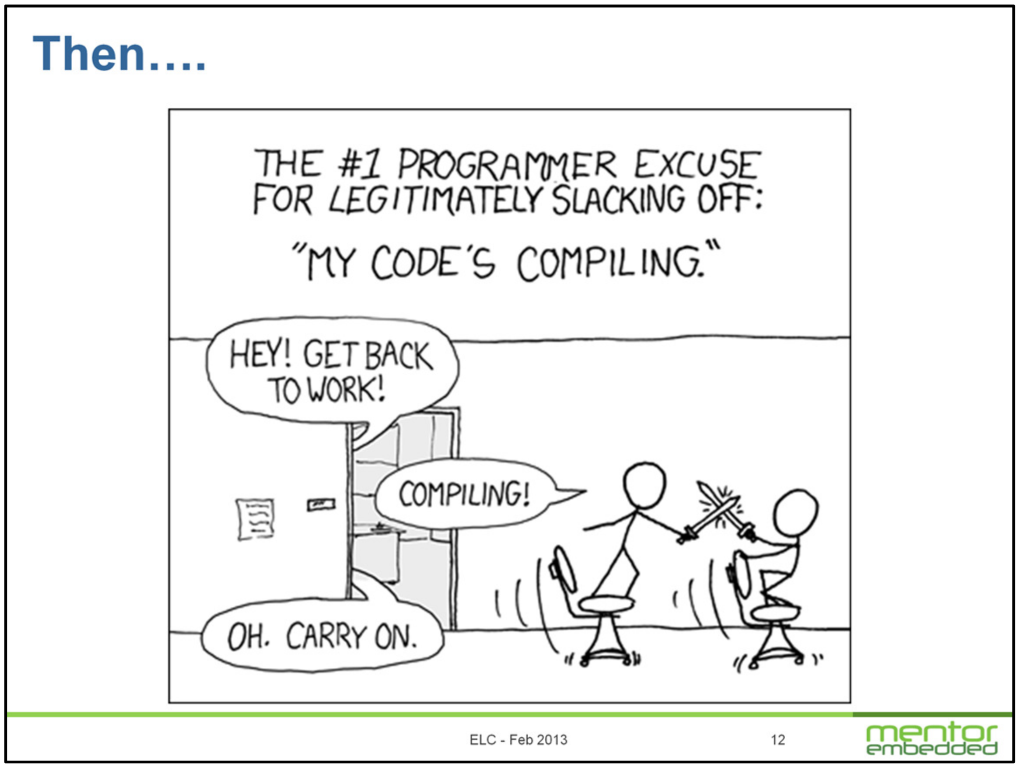
Show the initial build tree, including the local.conf.
Highlight the bblayers.conf and defer it again.
Pretty boring, as yet.

Let’s run a build

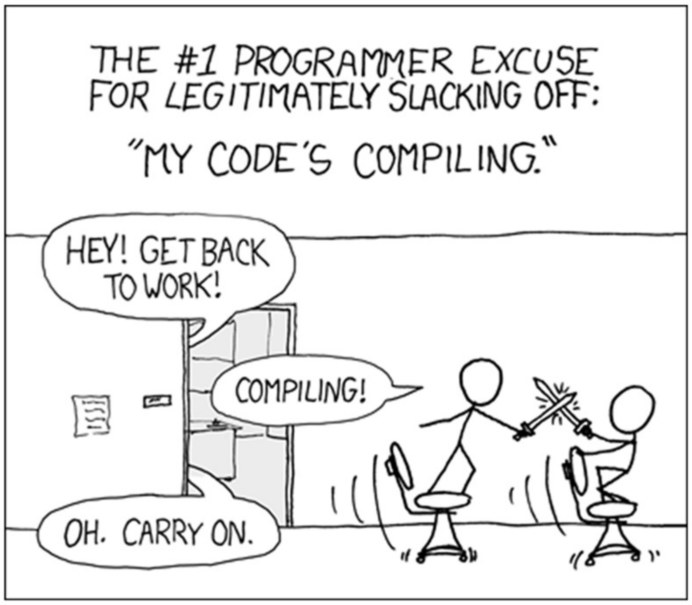


shudson@ronin: ~

```
shudson@ronin:/data/projects/poky-danny-8.0/build$ time bitbake -k core-image-sato
WARNING: Host distribution "Ubuntu 12.04.2 LTS" has not been validated with this version of the build system; you may possibly exper
ience unexpected failures. It is recommended that you use a tested distribution.
Loading cache: 100% |############################################################################################| ETA:  00:00:00
Loaded 1132 entries from dependency cache.

Build Configuration:
BB_VERSION        = "1.16.0"
TARGET_ARCH       = "i586"
TARGET_OS         = "linux"
MACHINE           = "qemux86"
DISTRO            = "poky"
DISTRO_VERSION    = "1.3"
TUNE_FEATURES     = "m32 i586"
TARGET_FPU        = ""
meta
meta-yocto
meta-yocto-bsp    = "<unknown>:<unknown>"

NOTE: Resolving any missing task queue dependencies
NOTE: Preparing runqueue
NOTE: Executing SetScene Tasks
Currently 4 running tasks (1488 of 1785):
0: ncurses-5.9-r11.1 do_package_write_rpm_setscene (pid 4525)
1: zypper-1.5.3-git1+2c5bb6ceb99ecd950ef993e43d77bf0569ea0582-r3 do_package_write_rpm_setscene (pid 4546)
2: bash-4.2-r5 do_package_write_rpm_setscene (pid 4552)
3: rpm-5.4.9-r52 do_package_write_rpm_setscene (pid 4545)
```

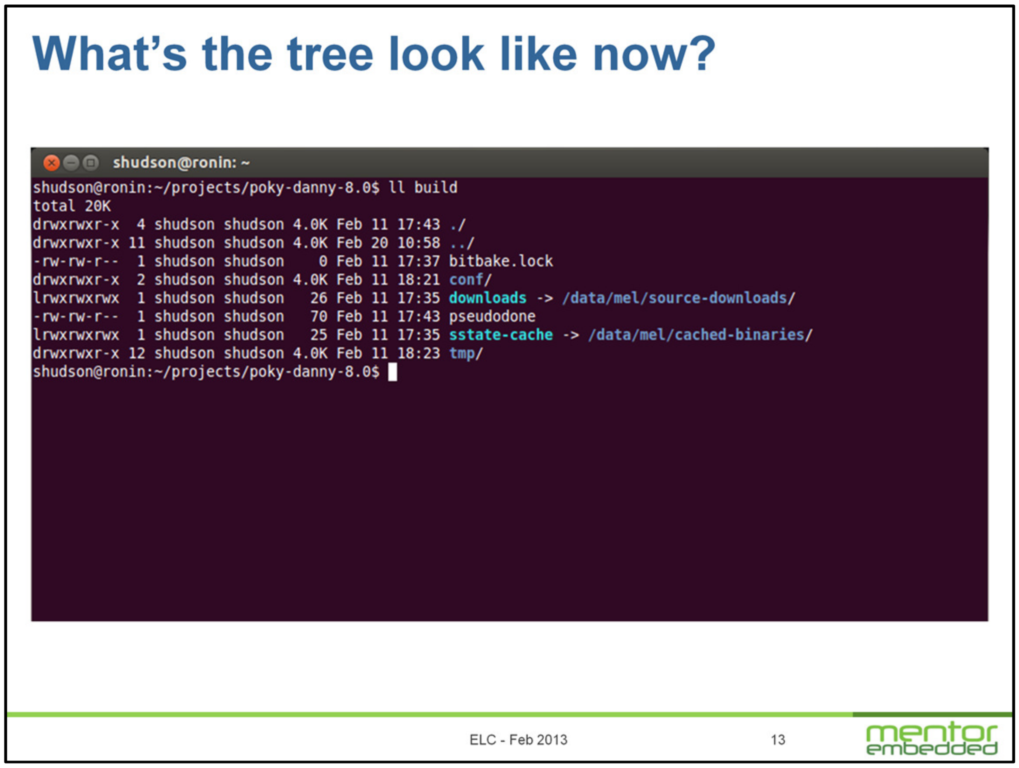


Then….



ELC - Feb 2013 12

Seriously, this may take a while depending on your machine and network speed. Typical times is 1-2 hours and don't forget to have enough disk space!



What's the tree look like now?

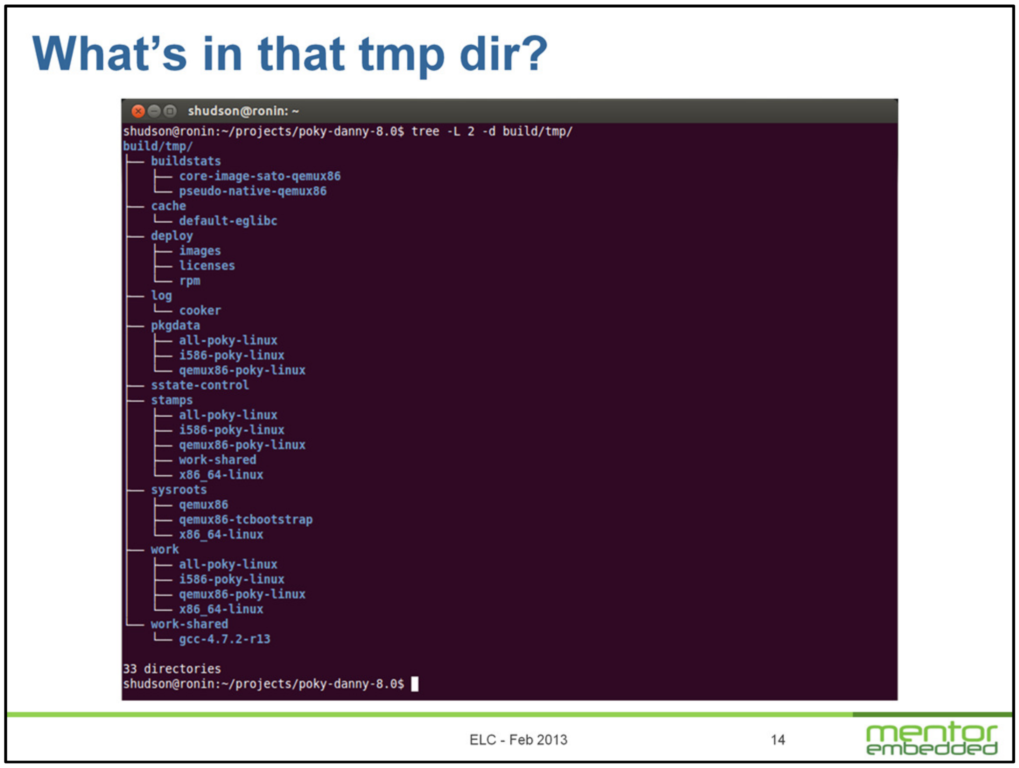
shudson@ronin: ~

```
shudson@ronin:~/projects/poky-danny-8.0$ ll build
total 20K
drwxrwxr-x  4 shudson shudson 4.0K Feb 11 17:43 ./
drwxrwxr-x 11 shudson shudson 4.0K Feb 20 10:58 ../
-rw-rw-r--  1 shudson shudson    0 Feb 11 17:37 bitbake.lock
drwxrwxr-x  2 shudson shudson 4.0K Feb 11 18:21 conf/
lrwxrwxrwx  1 shudson shudson   26 Feb 11 17:35 downloads -> /data/mel/source-downloads/
-rw-rw-r--  1 shudson shudson   70 Feb 11 17:43 pseudodone
lrwxrwxrwx  1 shudson shudson   25 Feb 11 17:35 sstate-cache -> /data/mel/cached-binaries/
drwxrwxr-x 12 shudson shudson 4.0K Feb 11 18:23 tmp/
shudson@ronin:~/projects/poky-danny-8.0$
```

ELC - Feb 2013 13 mentor embedded

So, what the heck is all this stuff?

1. downloads – source downloads (mention sharing the cache)
2. sstate-cache – package caching
3. tmp – the build output



What’s in that tmp dir?

```
shudson@ronin: ~
shudson@ronin:~/projects/poky-danny-8.0$ tree -L 2 -d build/tmp/
build/tmp/
├── buildstats
│   ├── core-image-sato-qemux86
│   └── pseudo-native-qemux86
├── cache
│   └── default-eglibc
├── deploy
│   ├── images
│   ├── licenses
│   └── rpm
├── log
│   └── cooker
├── pkgdata
│   ├── all-poky-linux
│   ├── i586-poky-linux
│   └── qemux86-poky-linux
├── sstate-control
├── stamps
│   ├── all-poky-linux
│   ├── i586-poky-linux
│   ├── qemux86-poky-linux
│   ├── work-shared
│   └── x86_64-linux
├── sysroots
│   ├── qemux86
│   ├── qemux86-tcbootstrap
│   └── x86_64-linux
├── work
│   ├── all-poky-linux
│   ├── i586-poky-linux
│   ├── qemux86-poky-linux
│   └── x86_64-linux
└── work-shared
    └── gcc-4.7.2-r13

33 directories
shudson@ronin:~/projects/poky-danny-8.0$ 
```

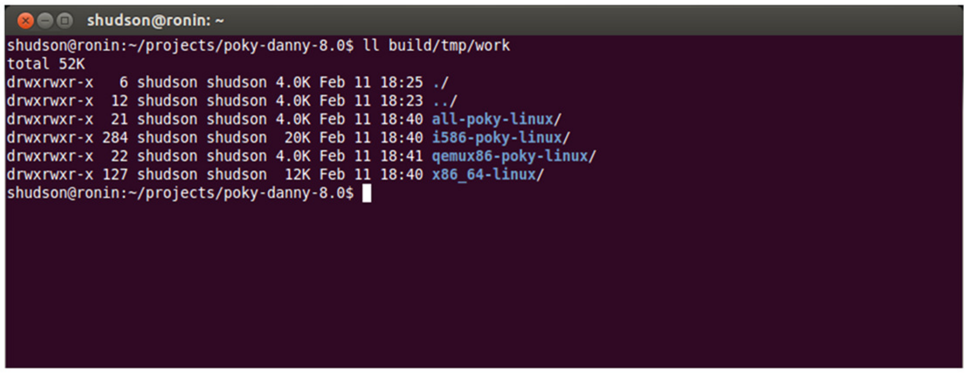
ELC - Feb 2013 14 mentor embedded

So, what the heck is all this stuff?

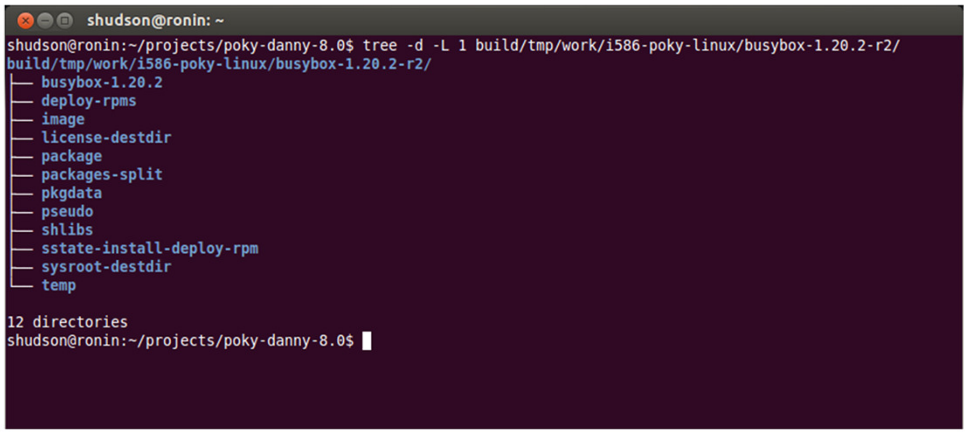
Important bits

1. Buildstats – useful build information
2. deploy – this is where final images go
3. pkgdata – information used to describe the packages
4. work – this is where the “action” is. Source archive, logs, and scripts for each build go here

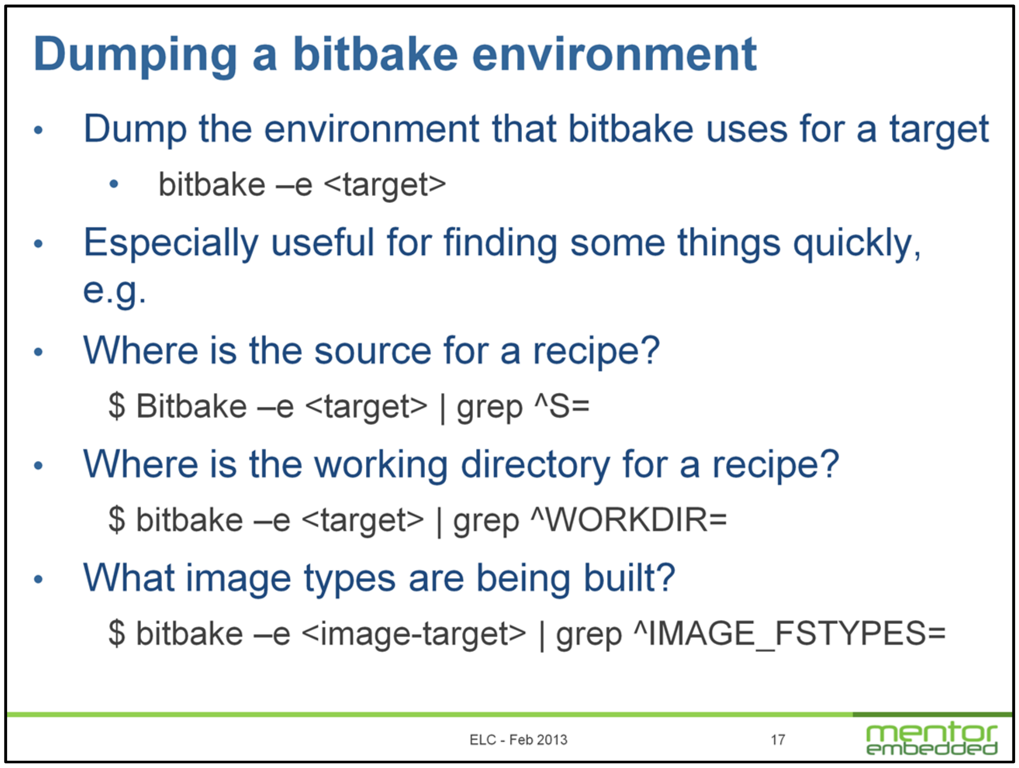
So, what's in the work dir?


```
shudson@ronin:~/projects/poky-danny-8.0$ ll build/tmp/work
total 52K
drwxrwxr-x   6 shudson shudson 4.0K Feb 11 18:25 ./
drwxrwxr-x  12 shudson shudson 4.0K Feb 11 18:23 ../
drwxrwxr-x  21 shudson shudson 4.0K Feb 11 18:40 all-poky-linux/
drwxrwxr-x 284 shudson shudson  20K Feb 11 18:40 i586-poky-linux/
drwxrwxr-x  22 shudson shudson 4.0K Feb 11 18:41 qemux86-poky-linux/
drwxrwxr-x 127 shudson shudson  12K Feb 11 18:40 x86_64-linux/
shudson@ronin:~/projects/poky-danny-8.0$
```

How far down do I need to go?!


shudson@ronin: ~

```
shudson@ronin:~/projects/poky-danny-8.0$ tree -d -L 1 build/tmp/work/i586-poky-linux/busybox-1.20.2-r2/
build/tmp/work/i586-poky-linux/busybox-1.20.2-r2/
├── busybox-1.20.2
├── deploy-rpms
├── image
├── license-destdir
├── package
├── packages-split
├── pkgdata
├── pseudo
├── shlibs
├── sstate-install-deploy-rpm
├── sysroot-destdir
└── temp

12 directories
shudson@ronin:~/projects/poky-danny-8.0$
```



Dumping a bitbake environment

- Dump the environment that bitbake uses for a target
  - bitbake –e <target>
- Especially useful for finding some things quickly, e.g.
- Where is the source for a recipe?

```
$ Bitbake –e <target> | grep ^S=
```

- Where is the working directory for a recipe?

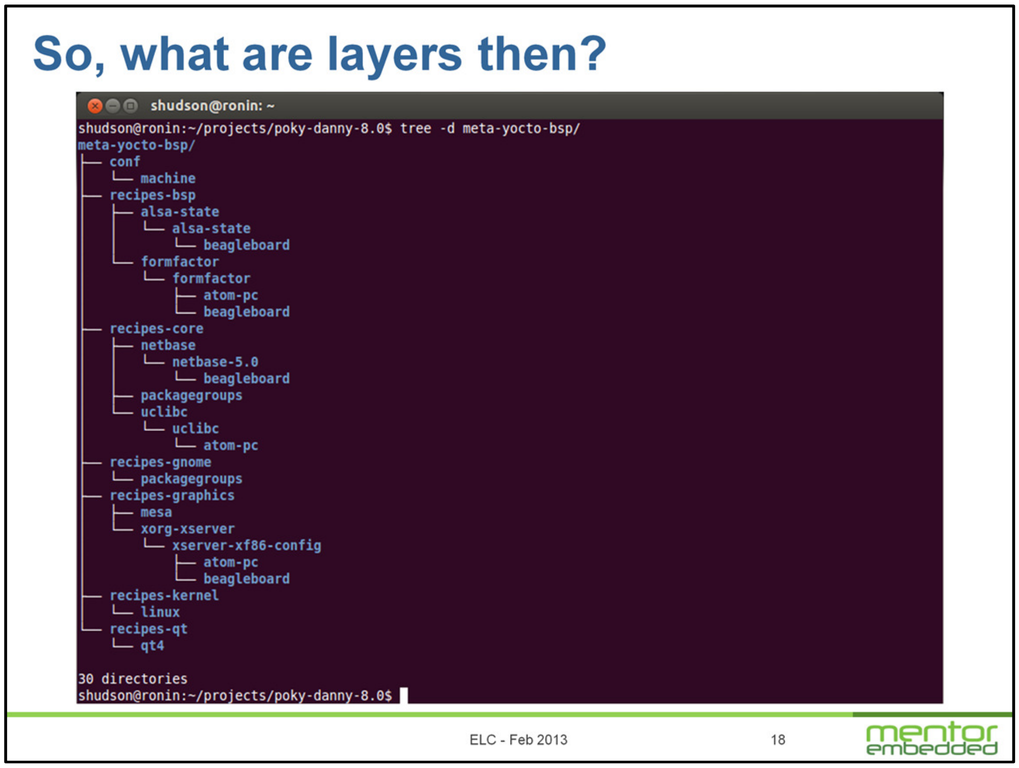
```
$ bitbake –e <target> | grep ^WORKDIR=
```

- What image types are being built?

```
$ bitbake –e <image-target> | grep ^IMAGE_FSTYPES=
```

ELC - Feb 2013 17

Note: bitbake –g also provides useful information by dumping the dependency graphs. Be careful, lots of information in this.



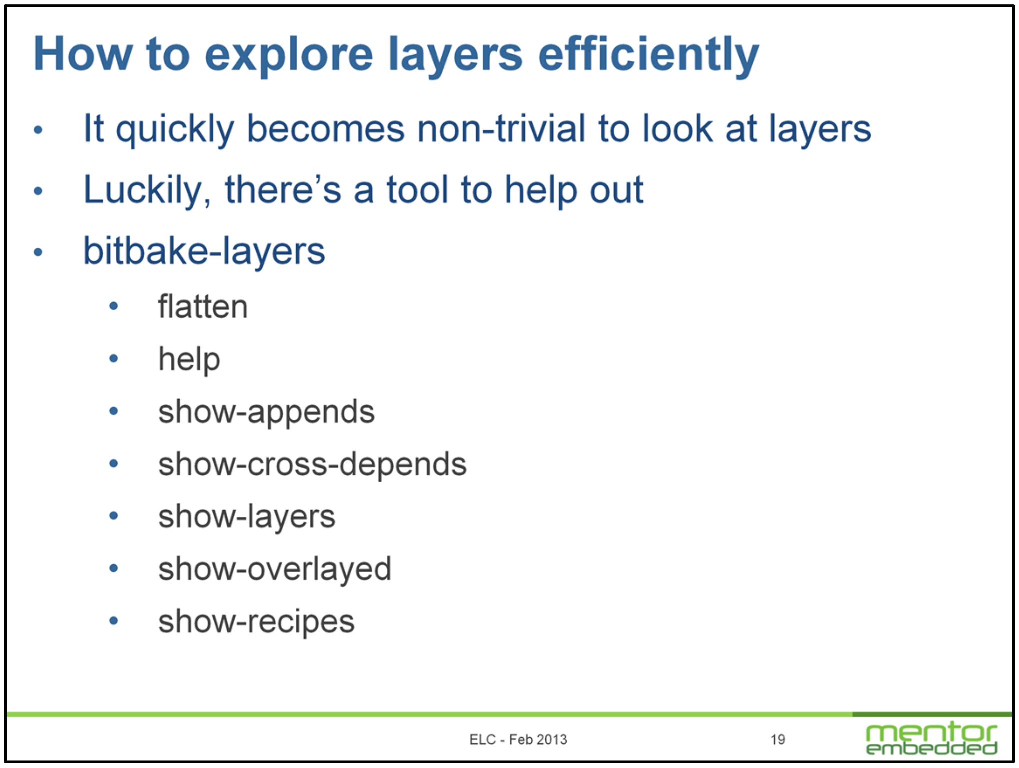
So, what are layers then?

```
shudson@ronin:~/projects/poky-danny-8.0$ tree -d meta-yocto-bsp/
meta-yocto-bsp/
├── conf
│   └── machine
├── recipes-bsp
│   ├── alsa-state
│   │   └── alsa-state
│   │       └── beagleboard
│   └── formfactor
│       └── formfactor
│           ├── atom-pc
│           └── beagleboard
├── recipes-core
│   ├── netbase
│   │   └── netbase-5.0
│   │       └── beagleboard
│   ├── packagegroups
│   └── uclibc
│       └── uclibc
│           └── atom-pc
├── recipes-gnome
│   └── packagegroups
├── recipes-graphics
│   ├── mesa
│   └── xorg-xserver
│       └── xserver-xf86-config
│           ├── atom-pc
│           └── beagleboard
├── recipes-kernel
│   └── linux
└── recipes-qt
    └── qt4

30 directories
shudson@ronin:~/projects/poky-danny-8.0$
```

ELC - Feb 2013 18 mentor embedded

Bitbake layers provide a way to collect related recipes together. This modularity, when used correctly, enables better reuse and easier maintenance.

Points to make:
1. Types of layers vary, call out bsp layers (refer to docs)
2. Other examples, networking, ivi, gui, etc

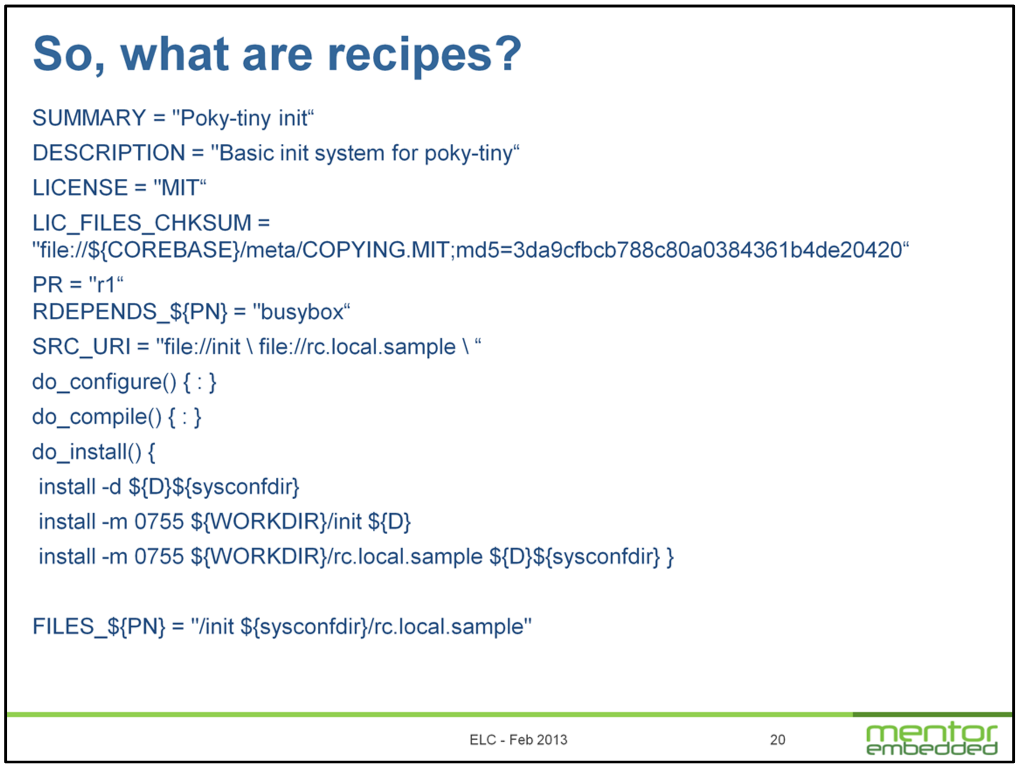


How to explore layers efficiently

- It quickly becomes non-trivial to look at layers
- Luckily, there's a tool to help out
- bitbake-layers
  - flatten
  - help
  - show-appends
  - show-cross-depends
  - show-layers
  - show-overlayed
  - show-recipes

ELC - Feb 2013 19

```
$ bitbake-layers
usage: bitbake-layers <command> [arguments]

Available commands:
  flatten
    flattens layer configuration into a separate output directory.
  help
    display general help or help on a specified command
  show-appends
    list bbappend files and recipe files they apply to
  show-cross-depends
    figure out the dependency between recipes that crosses a layer boundary.
  show-layers
    show current configured layers
  show-overlayed
    list overlayed recipes (where the same recipe exists in another layer)
  show-recipes
    list available recipes, showing the layer they are provided by
  help
    display general help or help on a specified command
```



So, what are recipes?

```
SUMMARY = "Poky-tiny init“
DESCRIPTION = "Basic init system for poky-tiny“
LICENSE = "MIT“
LIC_FILES_CHKSUM =
"file://${COREBASE}/meta/COPYING.MIT;md5=3da9cfbcb788c80a0384361b4de20420“
PR = "r1“
RDEPENDS_${PN} = "busybox“
SRC_URI = "file://init \ file://rc.local.sample \ “
do_configure() { : }
do_compile() { : }
do_install() {
 install -d ${D}${sysconfdir}
 install -m 0755 ${WORKDIR}/init ${D}
 install -m 0755 ${WORKDIR}/rc.local.sample ${D}${sysconfdir} }

FILES_${PN} = "/init ${sysconfdir}/rc.local.sample"
```

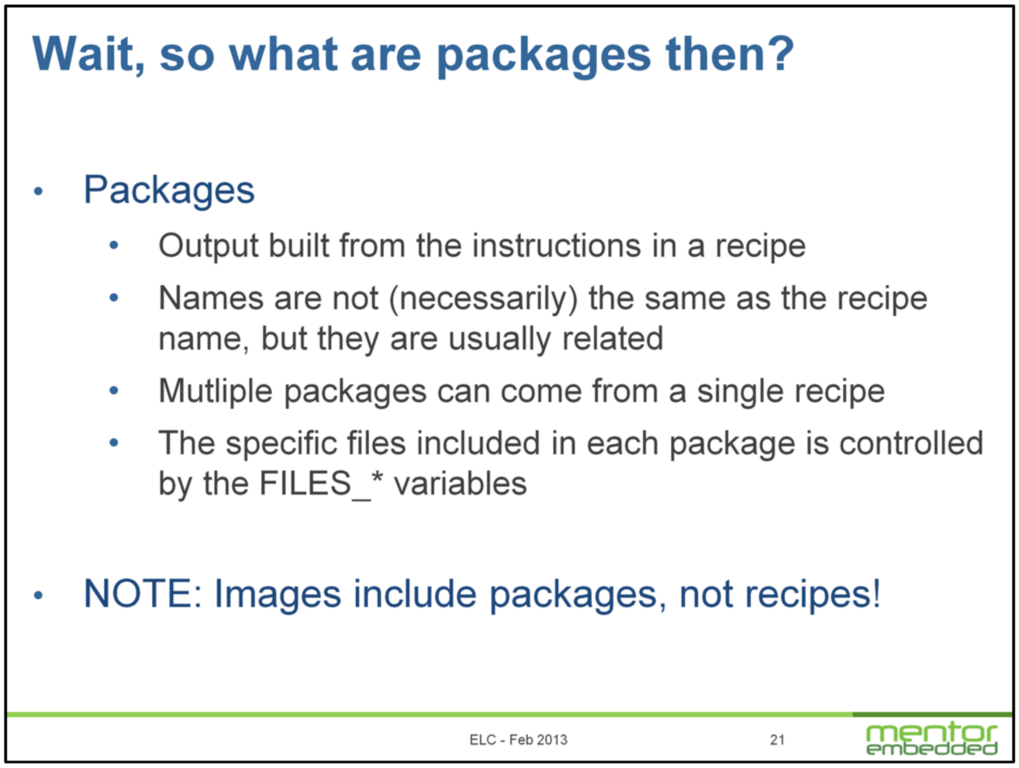
ELC - Feb 2013 20

Analyze an example recipe.

Recipes contain the instructions on how to take a set of source files and generate one or more output packages.
Name of the recipe is automatically inherited by many variable in total, or in part.

NOTE: Name-spacing is used a lot to separate items

Important pieces include:
SRC_URI
License fields
Checksum fields
…



Wait, so what are packages then?

- Packages
  - Output built from the instructions in a recipe
  - Names are not (necessarily) the same as the recipe name, but they are usually related
  - Mutliple packages can come from a single recipe
  - The specific files included in each package is controlled by the FILES_* variables
- NOTE: Images include packages, not recipes!

ELC - Feb 2013 21 mentor embedded

Refer to Chris Hallinan's blog post:
http://blogs.mentor.com/chrishallinan/blog/2012/04/27/more-on-yocto-terminology-recipes-and-packages/

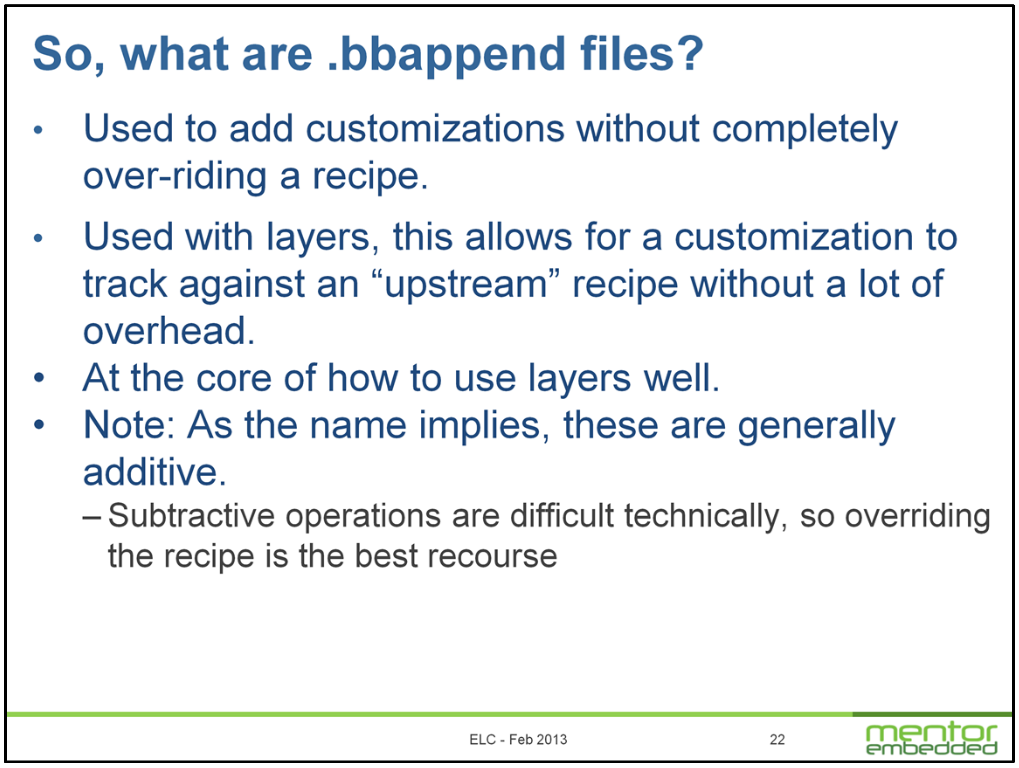
Default four packages, basic binary, -dev, -dbg, & -doc. Determined by the FILES_* variables

Example of multiple packages for a single recipe from Chris' post:

```
$ bitbake -e python | grep ^PACKAGES=
PACKAGES="libpython2 python-dbg python-2to3 python-audio python-bsddb
python-codecs python-compile python-compiler python-compression python-
core python-crypt python-ctypes python-curses python-datetime python-db
python-debugger python-dev python-difflib python-distutils-staticdev python-
distutils python-doctest python-elementtree python-email python-fcntl
python-gdbm python-hotshot python-html python-idle python-image python-
io python-json python-lang python-logging python-mailbox python-math
python-mime python-mmap python-multiprocessing python-netclient python-
netserver python-numbers python-pickle python-pkgutil python-pprint
python-profile python-pydoc python-re python-readline python-resource
python-robotparser python-shell python-smtpd python-sqlite3 python-sqlite3-
```

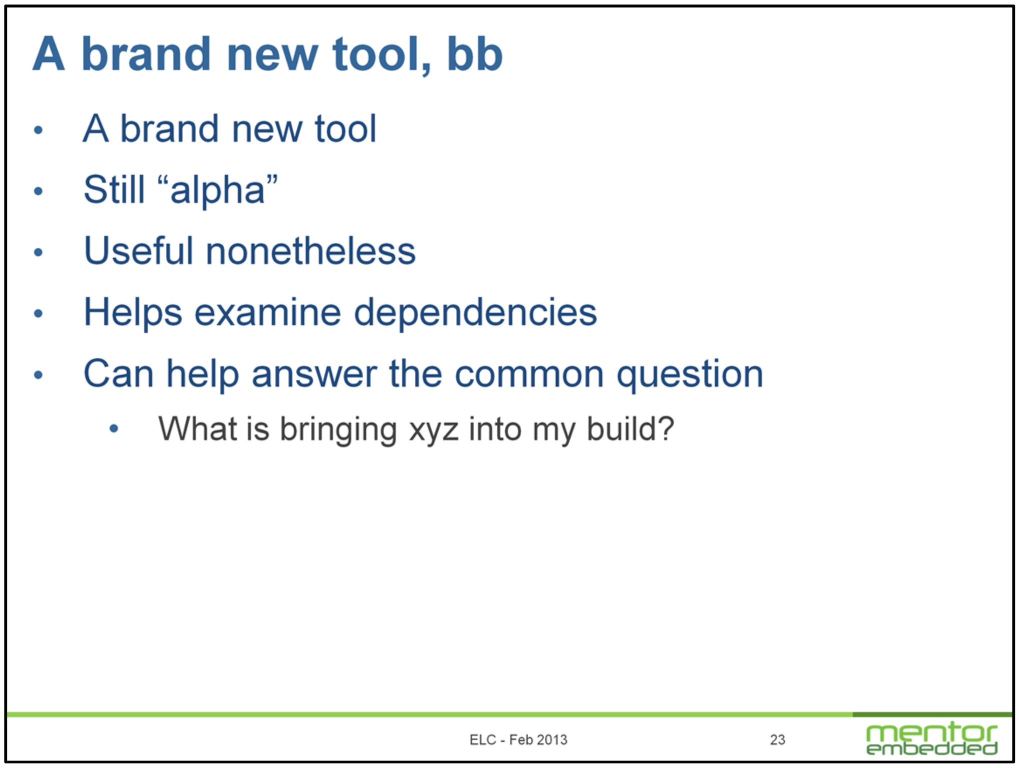
tests python-stringold python-subprocess python-syslog python-terminal python-tests python-textutils python-threading python-tkinter python-unittest python-unixadmin python-xml python-xmlrpc python-zlib python-modules python-misc python-man"

When BitBake completes "baking" the python recipe, a package is created for each of the named elements show above in the 'PACKAGES=' listing.



So, what are .bbappend files?

- Used to add customizations without completely over-riding a recipe.
- Used with layers, this allows for a customization to track against an “upstream” recipe without a lot of overhead.
- At the core of how to use layers well.
- Note: As the name implies, these are generally additive.
  - Subtractive operations are difficult technically, so overriding the recipe is the best recourse



A brand new tool, bb

- A brand new tool
- Still “alpha”
- Useful nonetheless
- Helps examine dependencies
- Can help answer the common question
  - What is bringing xyz into my build?

ELC - Feb 2013 23

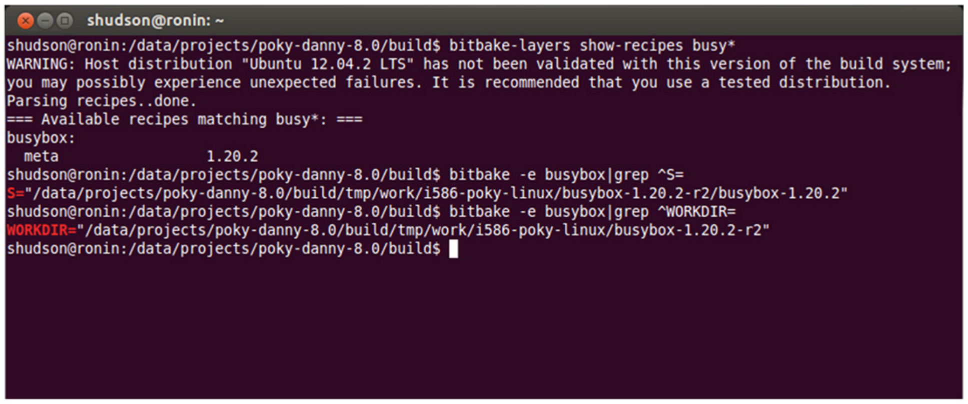
Chris Larson recently created a new tool, called ‘bb’.

Very useful for inspection of dependencies

```
$ bb
usage: bb [-h]


{whatdepends,help,showdepends,dependinfo,show,whatprovides,whatrprovides,showprov
ides}
          ...

Prototype subcommand-based bitbake UI

positional arguments:

{whatdepends,help,showdepends,dependinfo,show,whatprovides,whatrprovides,showprov
ides}
    dependinfo
    help             Show overall help, or help for a specific subcommand
    show             Show bitbake metadata (global or recipe)
    showdepends      Show what the specified target depends upon
    showprovides     Show what the specified target provides
    whatdepends         Show what depends on the specified target
    whatprovides        Show what recipes provide the specified target
    whatrprovides       Show what recipes provide the specified target

optional arguments:
  -h, --help            show this help message and exit
```

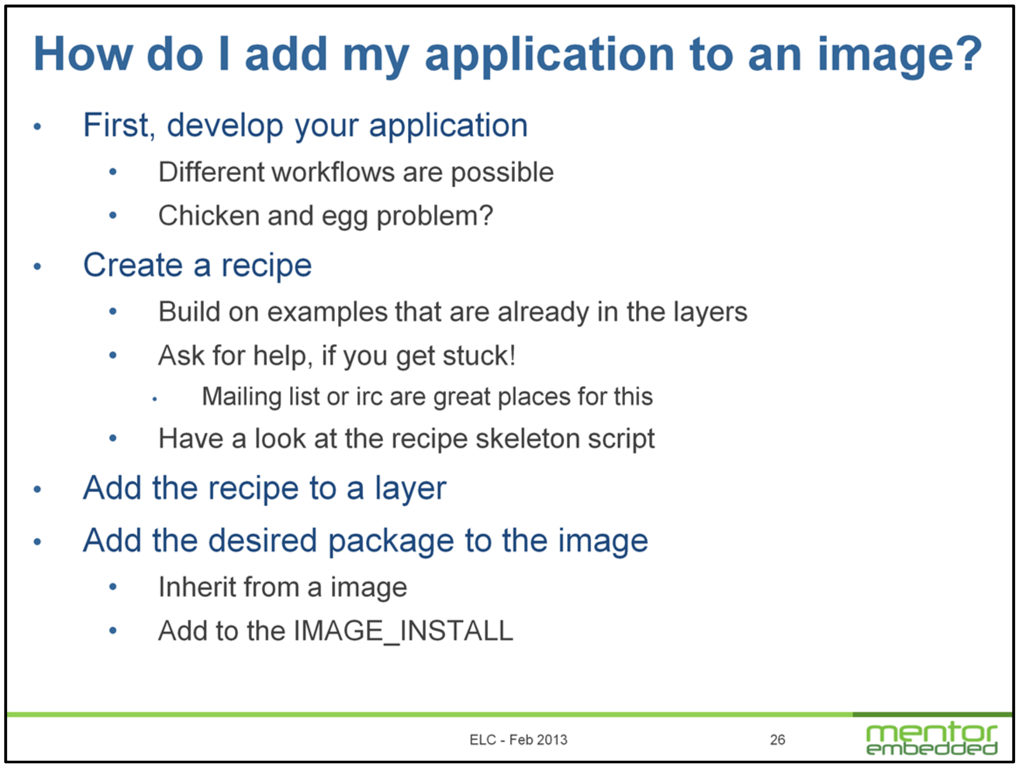
Putting it all together

Tracking down busybox



shudson@ronin: ~

```
shudson@ronin:/data/projects/poky-danny-8.0/build$ bitbake-layers show-recipes busy*
WARNING: Host distribution "Ubuntu 12.04.2 LTS" has not been validated with this version of the build system;
you may possibly experience unexpected failures. It is recommended that you use a tested distribution.
Parsing recipes..done.
=== Available recipes matching busy*: ===
busybox:
  meta                 1.20.2
shudson@ronin:/data/projects/poky-danny-8.0/build$ bitbake -e busybox|grep ^S=
S="/data/projects/poky-danny-8.0/build/tmp/work/i586-poky-linux/busybox-1.20.2-r2/busybox-1.20.2"
shudson@ronin:/data/projects/poky-danny-8.0/build$ bitbake -e busybox|grep ^WORKDIR=
WORKDIR="/data/projects/poky-danny-8.0/build/tmp/work/i586-poky-linux/busybox-1.20.2-r2"
shudson@ronin:/data/projects/poky-danny-8.0/build$
```

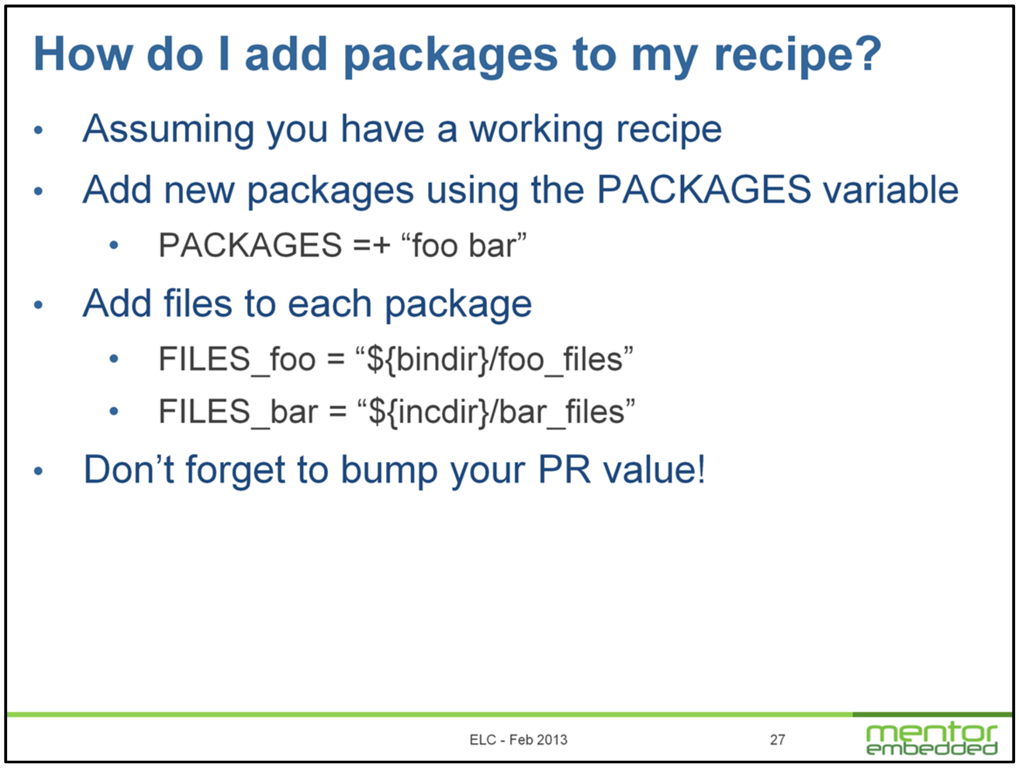


How do I add my application to an image?

- First, develop your application
  - Different workflows are possible
  - Chicken and egg problem?
- Create a recipe
  - Build on examples that are already in the layers
  - Ask for help, if you get stuck!
    - Mailing list or irc are great places for this
  - Have a look at the recipe skeleton script
- Add the recipe to a layer
- Add the desired package to the image
  - Inherit from a image
  - Add to the IMAGE_INSTALL

ELC - Feb 2013 26

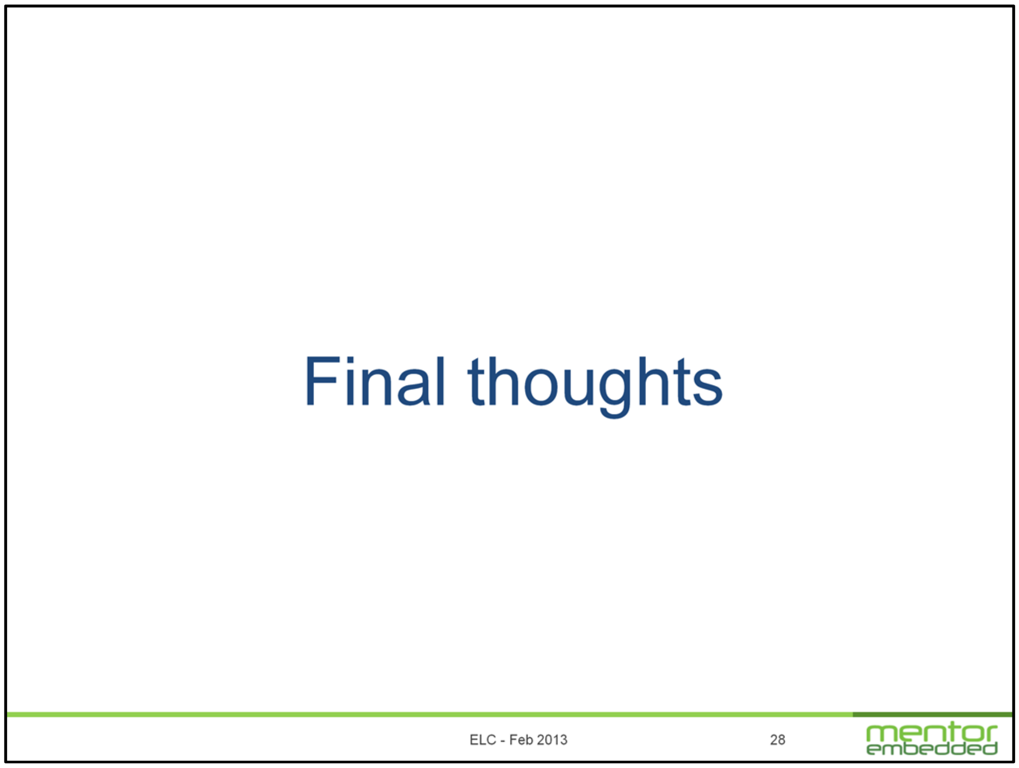
In almost all case, there are multiple ways to do things. Check out the docs for more alternatives.



How do I add packages to my recipe?

- Assuming you have a working recipe
- Add new packages using the PACKAGES variable
  - PACKAGES =+ “foo bar”
- Add files to each package
  - FILES_foo = “${bindir}/foo_files”
  - FILES_bar = “${incdir}/bar_files”
- Don’t forget to bump your PR value!

ELC - Feb 2013 27

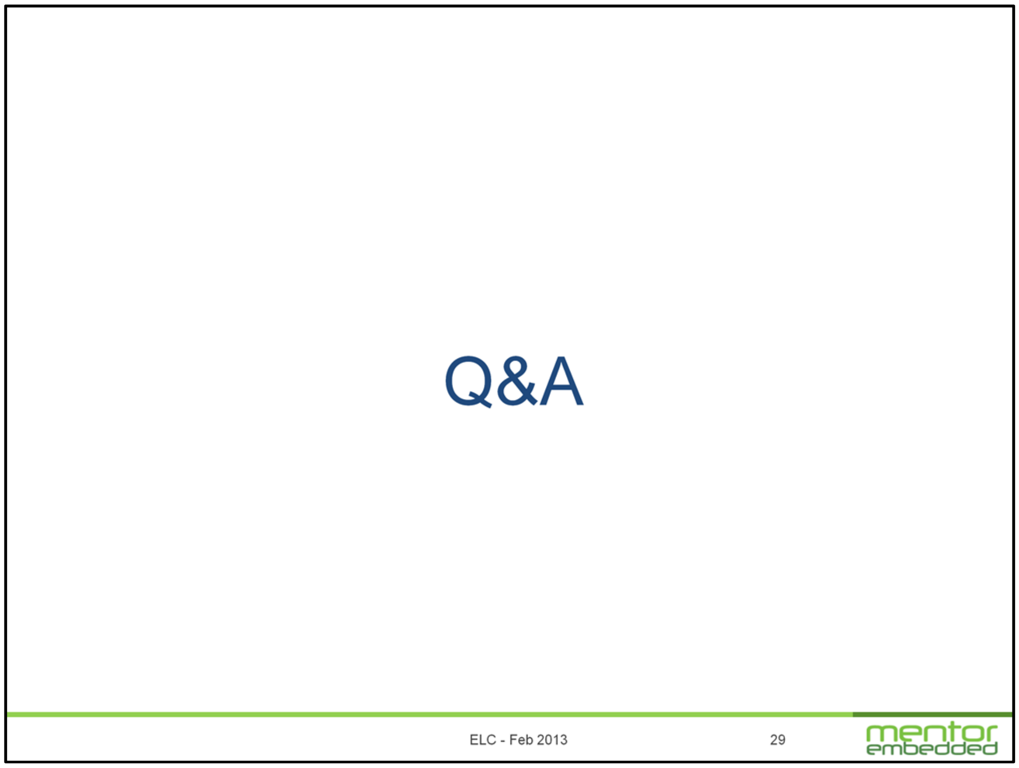


Final thoughts

ELC - Feb 2013 28

My final thoughts are mostly common sense.

1. Building a distibution from scratch is a daunting task, the YP gives you a great running start
2. Use an existing BSP as your starting point
3. Get comfortable with the process and understand the different roles/workflows that your organization will need to support
4. Play around with it and explore
5. The tools aren’t perfect and there are significant gaps, but that’s where we can use help!



Q&A