YP Dependencies Issues: Tools and Techniques

David Reyna, Wind River Systems
david.reyna@windriver.com
YP Conference Team, Advocacy Group, Working Group, Toaster Maintainer

Yocto Project Summit, 2022.05

Goals of this Presentation

- Capture the current state of dependency execution and analysis
- Describe dependencies and the problem space
- Present the many tools and techniques available today to analyze dependency issues
- Share new tools available or in progress to help resolve these issues going forward
- Dependency analysis is a “break glass in case of emergency” situation, so it needs to be easy to pick up and use when needed

Dependencies: An Introduction

- The dependency mechanism is how bitbake can resolve the correct order to build recipes
- Bitbake enforces dependencies at the task level in order to maximize parallelization
- This system rarely fails, but when it does it can be very hard to resolve

Dependencies: An Introduction

- There are several tools and techniques available for analyzing the dependency trees
- There are occasional gaps between Build dependencies and Run dependencies
- There are several new tool features being released to enhance this process

From the Bitbake Manual

- Each target BitBake builds consists of multiple tasks such as fetch, unpack, patch, configure, and compile. For best performance on multi-core systems, BitBake considers each task as an independent entity with its own set of dependencies.

- Dependencies are defined through several variables. You can find information about variables BitBake uses in the Variables Glossary near the end of this manual. At a basic level, it is sufficient to know that BitBake uses the DEPENDS and RDEPENDS variables when calculating dependencies.

Some Definitions

- (Forward) dependency: what packages the given package requires to be pre-built
- Backwards dependency: what packages require the given package to be pre-built
- Cached: the results came from the cache and were not explicitly built
- Not executed: the task noexec flag set to 1

The Build Dependency

- There is one "Build" dependency variable called "DEPENDS".
- In principle, a recipe waits for all of its "DEPENDS" recipes to complete
- In practice, it is the recipe's "do_prepare_recipe_sysroot" that waits until all its "DEPENDS" recipe's task "do_populate_sysroot" are finished, which in turn allows everything up to "do_fetch" task to run in parallel.
- Some task types require that a respective host tool is already built. For example, "do_patch" in general always waits for "quilt-native"

The Four Runtime Dependencies

- **RDEPENDS: Lists runtime dependencies of a package**
  - The practical effect of the RDEPENDS assignment is that those recipes will be declared as dependencies inside the package when it is written out by one of the do_package_write_* tasks.
- **RRECOMMENDS: Extends the usability of a package**
  - The package manager will automatically install the RRECOMMENDS list of packages when installing the built package.
  - Packages specified in RRECOMMENDS need not actually be produced, and the build will continue without error
- **RCONFLICTS, RREPLACES : observed but not used by bitbake**

Runtime Dependencies 2

- Bitbake's automatic runtime dependency calculations:

  "There are three automatic mechanisms (`shlibdeps`, `pcdeps`, and `depchains`) that handle shared libraries, package configuration (pkg-config) modules, and `-dev` and `-dbg` packages, respectively. For other types of runtime dependencies, you must manually declare the dependencies."

- This may result in un-anticipated packages in the build

General Techniques for Analyzing dependency issues

- DOT files
- Taskexp.py
- Toaster
- RPM/opkg/ipkg

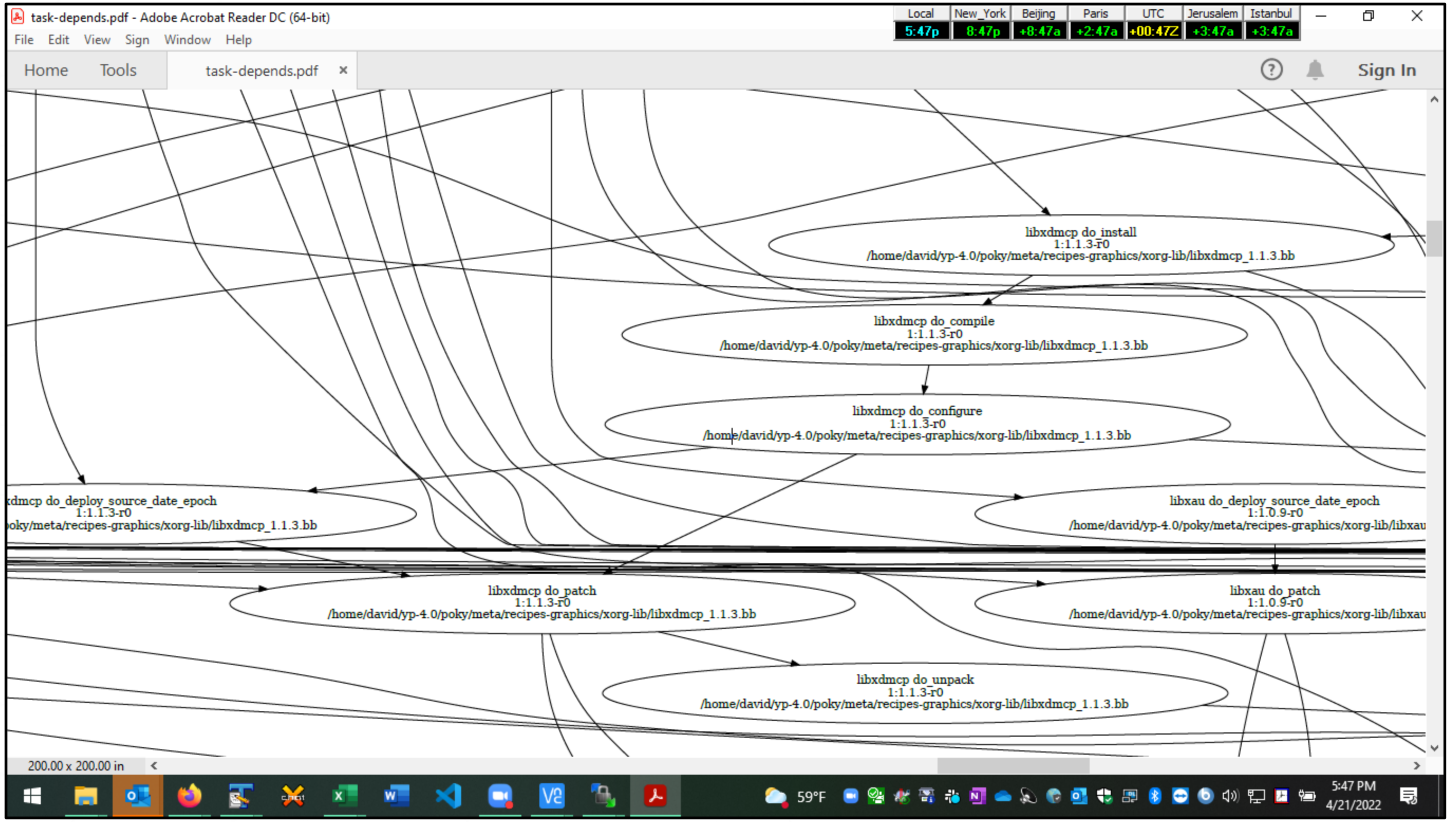
Tools: Dependency .dot files

- **Built-in dependency output from bitbake**
- **Advantages:**
  - Industry standard format for capturing relationships
  - Many tools available for viewing .dot files
- **Limitations:**
  - The count of tasks is in the hundreds, so hard to manage
  - Does not list the implicit reverse dependencies, so one has to search – *experts: “regex’ing dot files is no way to live your life*”
  - Graphic Tools (like graphviz) can take hours to render
  - The huge dependency tree can be very hard to read in a visualization
  - You can use filters to help reduce the noise with the “-I” option
    - bitbake -g -I virtual/kernel -I eglibc foo

$ bitbake -g zlib (-g, --graphviz: Save dependency tree information)

```
[build]$ bitbake -g zlib
Loading cache: 100% |############################################| Time: 0:00:00
Loaded 1642 entries from dependency cache.
NOTE: Resolving any missing task queue dependencies
NOTE: PN build list saved to 'pn-buildlist'
NOTE: Task dependencies saved to 'task-depends.dot'
[build]$ cat task-depends.dot | wc
   2114    6340  216350
[build]$ more task-depends.dot
digraph depends {
"acl-native.do_compile" [label="acl-native do_compile\n:2.3.1-r0\nvirtual:native:/home/david/yp
-master/poky/meta/recipes-support/attr/acl_2.3.1.bb"]
"acl-native.do_compile" -> "acl-native.do_configure"
"acl-native.do_configure" [label="acl-native do_configure\n:2.3.1-r0\nvirtual:native:/home/davi
d/yp-master/poky/meta/recipes-support/attr/acl_2.3.1.bb"]
"acl-native.do_configure" -> "acl-native.do_deploy_source_date_epoch"
…
```

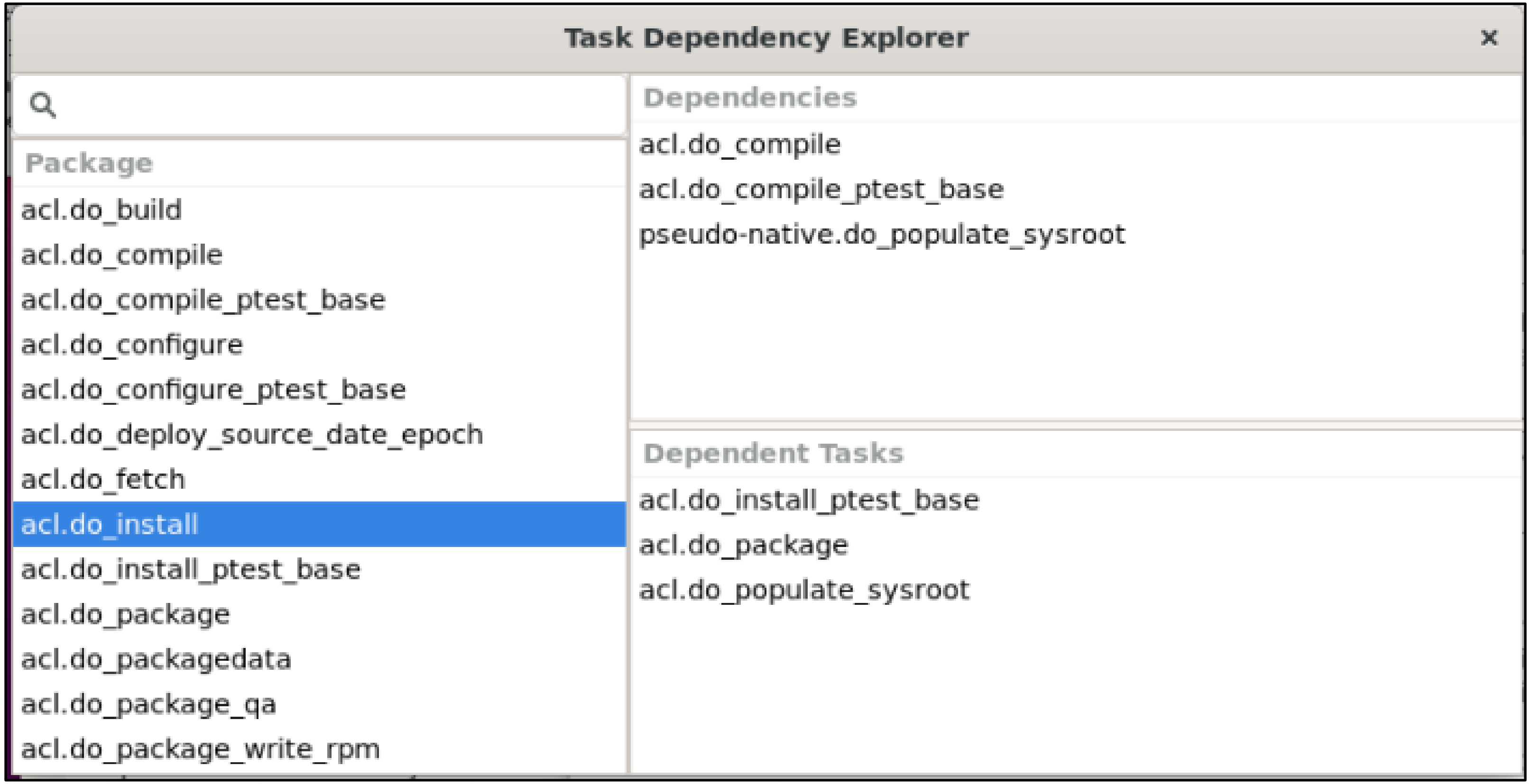
$ dot -Tpdf task-depends.dot -o task-depends.pdf


task-depends.pdf - Adobe Acrobat Reader DC (64-bit)
File Edit View Sign Window Help
Home Tools task-depends.pdf
Sign In
libxdmcp do_install
1:1.1.3-r0
/home/david/yp-4.0/poky/meta/recipes-graphics/xorg-lib/libxdmcp_1.1.3.bb
libxdmcp do_compile
1:1.1.3-r0
/home/david/yp-4.0/poky/meta/recipes-graphics/xorg-lib/libxdmcp_1.1.3.bb
libxdmcp do_configure
1:1.1.3-r0
/home/david/yp-4.0/poky/meta/recipes-graphics/xorg-lib/libxdmcp_1.1.3.bb
xdmcp do_deploy_source_date_epoch
1:1.1.3-r0
oky/meta/recipes-graphics/xorg-lib/libxdmcp_1.1.3.bb
libxau do_deploy_source_date_epoch
1:1.0.9-r0
/home/david/yp-4.0/poky/meta/recipes-graphics/xorg-lib/libxau
libxdmcp do_patch
1:1.1.3-r0
/home/david/yp-4.0/poky/meta/recipes-graphics/xorg-lib/libxdmcp_1.1.3.bb
libxau do_patch
1:1.0.9-r0
/home/david/yp-4.0/poky/meta/recipes-graphics/xorg-lib/libxau
libxdmcp do_unpack
1:1.1.3-r0
/home/david/yp-4.0/poky/meta/recipes-graphics/xorg-lib/libxdmcp_1.1.3.bb
200.00 x 200.00 in
59°F
5:47 PM
4/21/2022

Taskexp.py: A GTK+ application included with bitbake

- **Advantages:**
  - In Bitbake since 2007 (Ross Burton)
  - Quick to start
  - Shows forward and backward dependencies for recipes
  - Listens to the "generateDepTreeEvent“ event
- **Limitations:**
  - The tasks are shown in dictionary order
  - Requires GTK+ on host
  - No way to export the results for review

$ bitbake -g -u taskexp acl (-u UI: The user interface to use)


Task Dependency Explorer

Package

acl.do_build
acl.do_compile
acl.do_compile_ptest_base
acl.do_configure
acl.do_configure_ptest_base
acl.do_deploy_source_date_epoch
acl.do_fetch
acl.do_install
acl.do_install_ptest_base
acl.do_package
acl.do_packagedata
acl.do_package_qa
acl.do_package_write_rpm

Dependencies

acl.do_compile
acl.do_compile_ptest_base
pseudo-native.do_populate_sysroot

Dependent Tasks

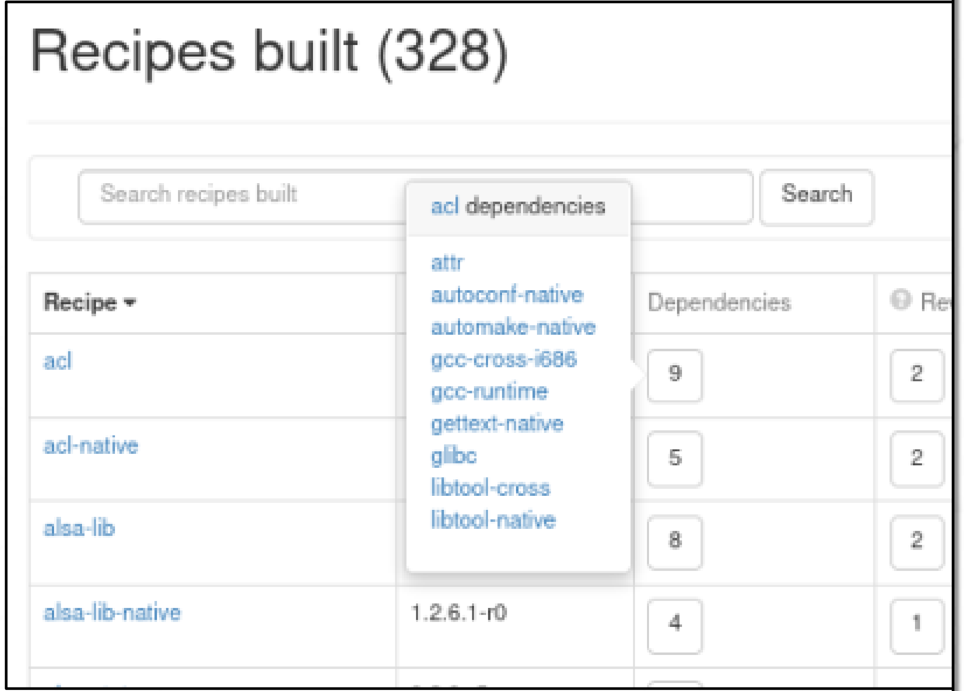
acl.do_install_ptest_base
acl.do_package
acl.do_populate_sysroot

Toaster: A web-based application included with bitbake

- **Advantages:**
  - Has deep additional information about tasks from the bitbake event stream
    - Built or loaded from cached, which task may have "covered" it
    - Disk I/O and Build time statistics
  - Has deep information about recipes from the layer Index
  - Has information about when tasks were started and stopped

- **Limitations:**
  - The dependencies at the recipe level are hard to corelate with others
  - Requires a build in the Toaster context to capture the additional task data

Toaster: Usage Example



Recipes built (328)

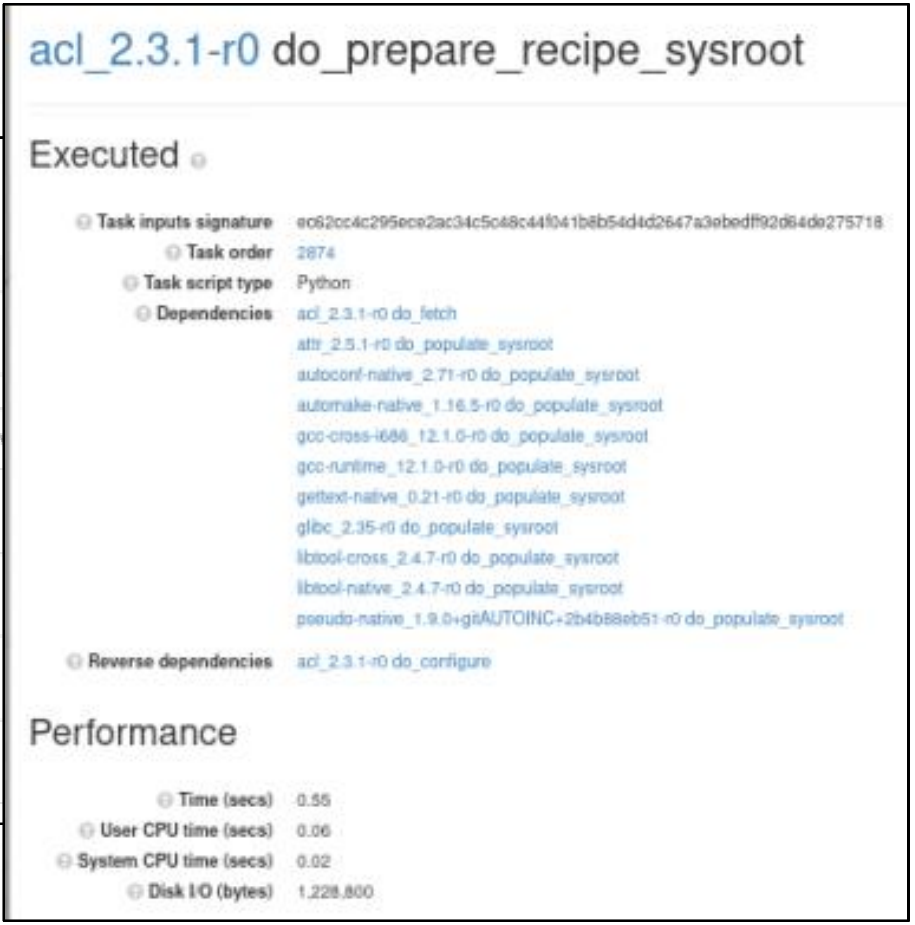
Search recipes built

Search

acl dependencies

attr
autoconf-native
automake-native
gcc-cross-i686
gcc-runtime
gettext-native
glibc
libtool-cross
libtool-native

Recipe ▾		Dependencies	Re
acl		9	2
acl-native		5	2
alsa-lib		8	2
alsa-lib-native	1.2.6.1-r0	4	1



acl_2.3.1-r0 do_prepare_recipe_sysroot

Executed

Task inputs signature	ec62cc4c295ece2ac34c5c48c44f041b8b54d4d2647a3ebedff92d64de275718
Task order	2874
Task script type	Python
Dependencies	acl_2.3.1-r0 do_fetch attr_2.5.1-r0 do_populate_sysroot autoconf-native_2.71-r0 do_populate_sysroot automake-native_1.16.5-r0 do_populate_sysroot gcc-cross-i686_12.1.0-r0 do_populate_sysroot gcc-runtime_12.1.0-r0 do_populate_sysroot gettext-native_0.21-r0 do_populate_sysroot glibc_2.35-r0 do_populate_sysroot libtool-cross_2.4.7-r0 do_populate_sysroot libtool-native_2.4.7-r0 do_populate_sysroot pseudo-native_1.9.0+gitAUTOINC+2b4b88eb51-r0 do_populate_sysroot
Reverse dependencies	acl_2.3.1-r0 do_configure

Performance

Time (secs)	0.55
User CPU time (secs)	0.06
System CPU time (secs)	0.02
Disk I/O (bytes)	1,228,800

RPM/opkg/ipkg: application included with Linux

- **Advantages:**
  - The packages for the generated file system image are rendered in one of these formats, so it makes sense to use their package manager to directly examine their raw declared dependencies and other meta data
  - This information reveals exactly what bitbake encoded as dependencies, and viewing this data can reveal unexpected dependencies or mistaken/mistranslated dependencies
  - Old school, tried and true

- **Limitations:**
  - These tools are complex and can be hard to work with, especially when trying to corelate between packages

Issues with Runtime Dependencies

- There can be times when a package will appear in a build with no evidence why it was included, not in the build dependencies and not apparently in the runtime dependencies.

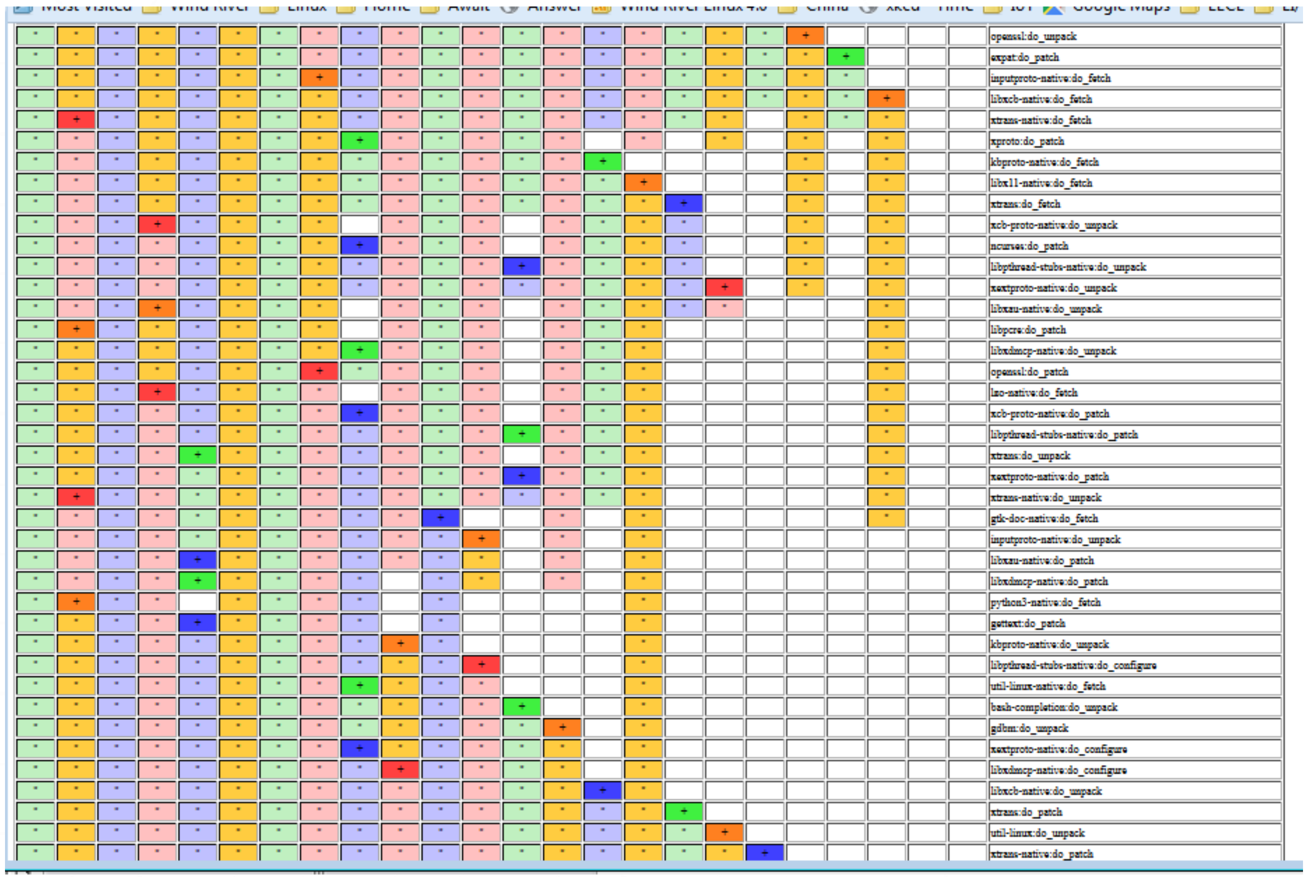
- This may be a case where runtime dependencies have an unexpected need, and where the build dependencies diverge.

- In this situation, you can simply block the package in question from the build, re-run the build, and see exactly what dependency was broken.

Task Execution Overlap

- Another potential hidden source of dependency issues can come from the packages that are being built in parallel due to unexpected dependent resource issue
- These issues could be persistent and/or can be transient UFOs if there is a race condition on how those tasks are scheduled and how fast they run
- I gave a paper titled “Extracting analytics from complex OpenEmbedded builds” at ELC in Portland 2017, which describes how to use the bitbake event stream to extract the exact task start and stop times to determine these build overlaps, specifically using the Toaster database and simple command line scripts
- Including this data, together with the dynamic runtime dependencies, can give a deeper picture into dependency issues

Sample HTML Output of Task Overlap



openssl:do_unpack
expat:do_patch
inputproto-native:do_fetch
libxcb-native:do_fetch
xtrans-native:do_fetch
xproto:do_patch
kbproto-native:do_fetch
libx11-native:do_fetch
xtrans:do_fetch
xcb-proto-native:do_unpack
ncurses:do_patch
libpthread-stubs-native:do_unpack
xextproto-native:do_unpack
libxau-native:do_unpack
libpcre:do_patch
libxdmcp-native:do_unpack
openssl:do_patch
lzo-native:do_fetch
xcb-proto-native:do_patch
libpthread-stubs-native:do_patch
xtrans:do_unpack
xextproto-native:do_patch
xtrans-native:do_unpack
gtk-doc-native:do_fetch
inputproto-native:do_unpack
libxau-native:do_patch
libxdmcp-native:do_patch
python3-native:do_fetch
gettext:do_patch
kbproto-native:do_unpack
libpthread-stubs-native:do_configure
util-linux-native:do_fetch
bash-completion:do_unpack
gdbm:do_unpack
xextproto-native:do_configure
libxdmcp-native:do_configure
libxcb-native:do_unpack
xtrans:do_patch
util-linux:do_unpack
xtrans-native:do_patch

Leveraging the powerful bitbake event system, which is used by Knotty, Toaster, IncrediBuild, and other tools (and you can too!)

This example is focused on task execution overlap

Bitbake UIs: Three Quick Caveats

1. Hidden errors on first use

```
[build]$ bitbake -g -u taskexp acl
Exception in startup:
 Traceback (most recent call last):
  File "/home/david/yp_master/poky/…
    params.updateToServer(server, …
  File "/home/david/yp_master/poky/bitbake…
    ret, error = server.runCommand(["update…
 File "/home/david/yp_master/poky/bitbake/l…
    raise bb.BBHandledException()
bb.BBHandledException

[build]$ bitbake quilt-native
ERROR: The following required tools (as specified
  by HOSTTOOLS) appear to be unavailable in
  PATH, please install them in order to proceed:
  pzstd zstd
[build]$
```

2. UIs lock bitbake

```
[build]$ # Run taskexp in the background …
[build] bitbake -g -u taskexp zlib

^Z
[1]+  Stopped                 bitbake -g -u taskexp zlib
[build]$
[build]$ bg
[1]+ bitbake -g -u taskexp zlib &
[build]$ bitbake quilt-native      (or “. toaster start …”)
```

[bitbake hang, or bitbake fails with a bitbake lock]

Obvious for Knotty, but not as obvious for GUIs like taskex and Toaster that float in separate windows

Bitbake UIs: Three Quick Caveats

3. Misspelling the UI name leads bitbake to somehow try to open Toaster from the side, resulting in massive errors

```
[build]$ bitbake -g -u taskexp_foobar sometask
Unable to init server: Could not connect: Connection refused
Unable to init server: Could not connect: Connection refused
FATAL: Unable to import extension module "taskexp_foobar" from bb.ui. Valid extension modules:
knotty, ncurses, taskexp, taskexp_cli, teamcity or toasterui
…
WARNING: /host/yp-2022.05/poky/bitbake/lib/toaster/toastermain/settings.py:97:
ResourceWarning: unclosed file <_io.BufferedReader name='/etc/localtime'>
  TIME_ZONE = zonefilelist[hashlib.md5(open('/etc/localtime', 'rb').read()).hexdigest()]
[build]$
```

Quick Questions and Answers

- **Question**: Tool commands like “taskexp recipename” seem very narrowly specific. Is it the case that a developer generally explores one recipe at a time?

- **Answer**: That is usually the use case. They're building say an image and they want to explore the dependencies in that image. Other dependencies would just confuse their search.
- To be clear:
  - recipename may be an image or packagegroup or any bitbake target
  - You can also do "bitbake imageA imageB imageC -u taskexp -g“
- Bitbake only computes the taskgraph for the targets you ask it to process so you will have to limit the scope to some target(s). One could do it for a world build but not sure the results would be as useful as you'd think.

Quick Questions and Answers

- **Question**: Can we assume that recipe dependencies change depending on how a build is configured, and that the “bitbake -g -u taskexp recipename” command is indeed context sensitive?

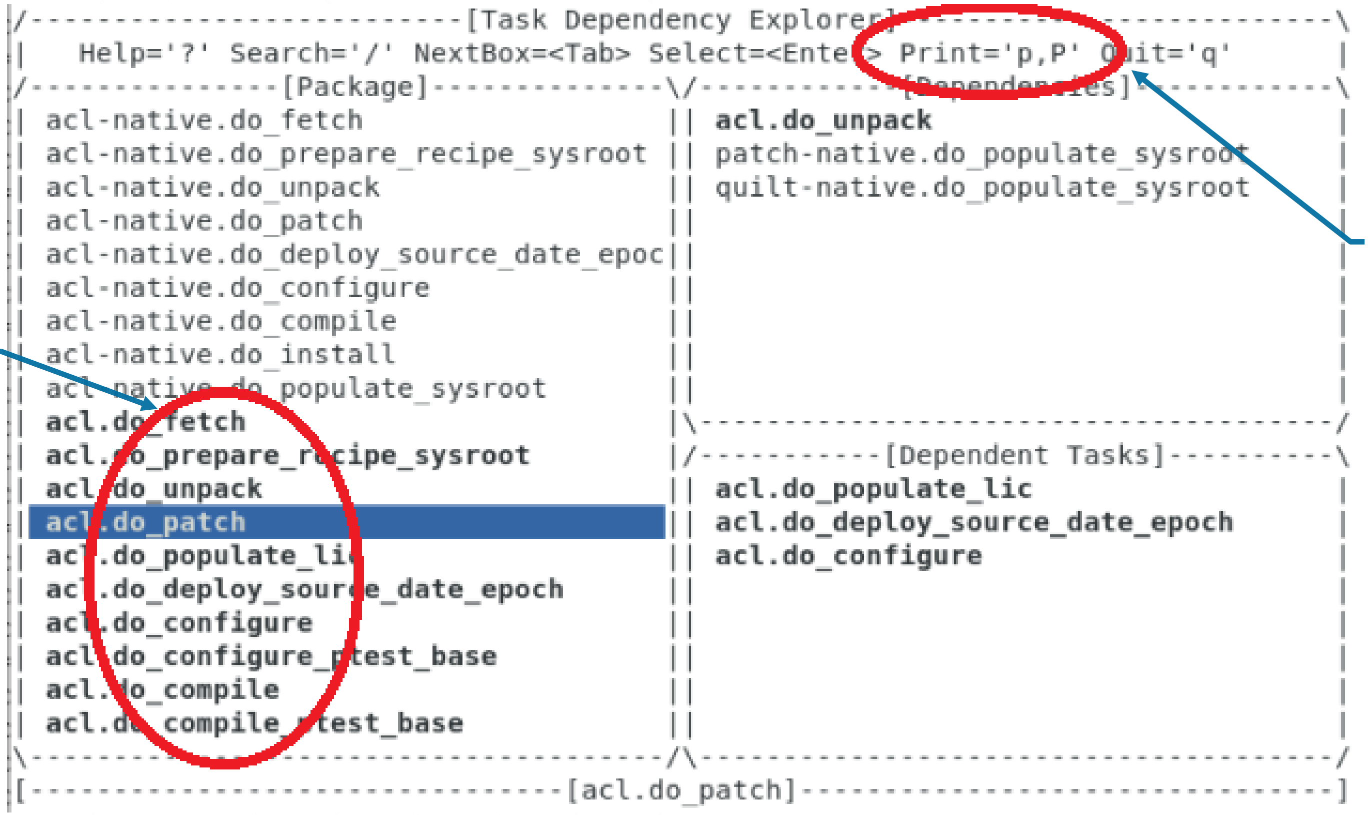
- **Answer**: Yes, very much so (and this is what we want)

New Work for Dependency Analysis

New: TaskExp_cli.py – Based on Ncurses

- **Presentationally equivalent to the existing “taskexp.py” GTK app**
- **Many new features:**
  - Can run as-is on any Linux host (that also supports bitbake’s knotty)
  - No reliance on GTK or web service: can run on minimal hosts
  - Shows in bold all tasks of the primary recipe name, for easy context
  - Shows the tasks of each recipe in the internal dependency order
    - Shows the detailed bitbake task order
    - Supports tasks that are recipe-specific, not just regular bitbake tasks
    - You can toggle the sort option back to the regular alpha-numeric
  - Has a print-to-text-file feature:
    - Print and append the dependencies of specific tasks
    - Print the dependencies of all tasks of parent recipe, in sort order
    - Hopefully, this feature kills the need to regex the DOT files

$ bitbake -g -u taskexp_cli acl (-u UI The user interface to use)



```
/---------------------------[Task Dependency Explorer]---------------------------\
|   Help='?' Search='/' NextBox=<Tab> Select=<Enter> Print='p,P' Quit='q'         |
/--------------[Package]--------------\/-----------[Dependencies]-----------\
| acl-native.do_fetch                  || acl.do_unpack                      |
| acl-native.do_prepare_recipe_sysroot || patch-native.do_populate_sysroot   |
| acl-native.do_unpack                 || quilt-native.do_populate_sysroot   |
| acl-native.do_patch                  ||                                    |
| acl-native.do_deploy_source_date_epoc||                                    |
| acl-native.do_configure              ||                                    |
| acl-native.do_compile                ||                                    |
| acl-native.do_install                ||                                    |
| acl-native.do_populate_sysroot       ||                                    |
| acl.do_fetch                         |\------------------------------------/
| acl.do_prepare_recipe_sysroot        |/----------[Dependent Tasks]---------\
| acl.do_unpack                        || acl.do_populate_lic                |
| acl.do_patch                         || acl.do_deploy_source_date_epoch    |
| acl.do_populate_lic                  || acl.do_configure                   |
| acl.do_deploy_source_date_epoch      ||                                    |
| acl.do_configure                     ||                                    |
| acl.do_configure_ptest_base          ||                                    |
| acl.do_compile                       ||                                    |
| acl.do_compile_ptest_base            ||                                    |
\--------------------------------------/\------------------------------------/
[-----------------------------[acl.do_patch]-----------------------------------]
```

Inner Dependency Order

Export Data

Taskexp_cli.py "Print" File Output

```
[build]$ more taskdep.txt
=== Dependendency Snapshot ===
Dep     =
Package=acl.do_fetch
RDep    =acl.do_prepare_recipe_sysroot
         acl.do_unpack
=== Dependendency Snapshot ===
         acl.do_fetch
         attr.do_populate_sysroot
         autoconf-native.do_populate_sysroot
         automake-native.do_populate_sysroot
         gcc-cross-x86_64.do_populate_sysroot
         gcc-runtime.do_populate_sysroot
         gettext-native.do_populate_sysroot
         glibc.do_populate_sysroot
         libtool-cross.do_populate_sysroot
         libtool-native.do_populate_sysroot
Dep     =pseudo-native.do_populate_sysroot
Package=acl.do_prepare_recipe_sysroot
RDep    =acl.do_configure
=== Dependendency Snapshot ===
Dep     =acl.do_fetch
Package=acl.do_unpack
RDep    =acl.do_patch
...
```

Coming: Toaster Enhancements

- **Add the task-level dependency view, beyond just recipe-level dependencies**
- **Include in the new view the extra task data that dot files and taskexp do not have, specifically:**
  - If the task was actually built
  - If it was instead loaded from cached (perhaps mistakenly)
  - Which task may have "covered" it (perhaps mistakenly)
- **Leverage the bitbake event stream capture feature, so that Toaster can import that existing output from an existing command line build and fully visualize it**
- **Leverage the task timing data so that the user can see what tasks were being built simultaneous to the task in question, in case there is a race condition**
- **Capture the automatic runtime dependency events, so that all runtime dependencies can be explicitly seen?**

Resources:

- **The YP Technical FAQs ("Dependencies")**
  - https://wiki.yoctoproject.org/wiki/Technical_FAQ
- **The YP Documentation ("Generating Dependency Graphs")**
  - https://www.yoctoproject.org/docs/latest/bitbake-user-manual/bitbake-user-manual.html
- **The YP Documentation ("Automatic Runtime Dependencies")**
  - https://docs.yoctoproject.org/current/overview-manual/concepts.html#automatically-added-runtime-dependencies
- **"Extracting analytics from complex OpenEmbedded builds", ELC 2017**
  - http://events17.linuxfoundation.org/sites/events/files/slides/BitbakeAnalytics_ELC_Portland.pdf
  - https://wiki.yoctoproject.org/wiki/DevDay_US_2017 (the example task overlap script)

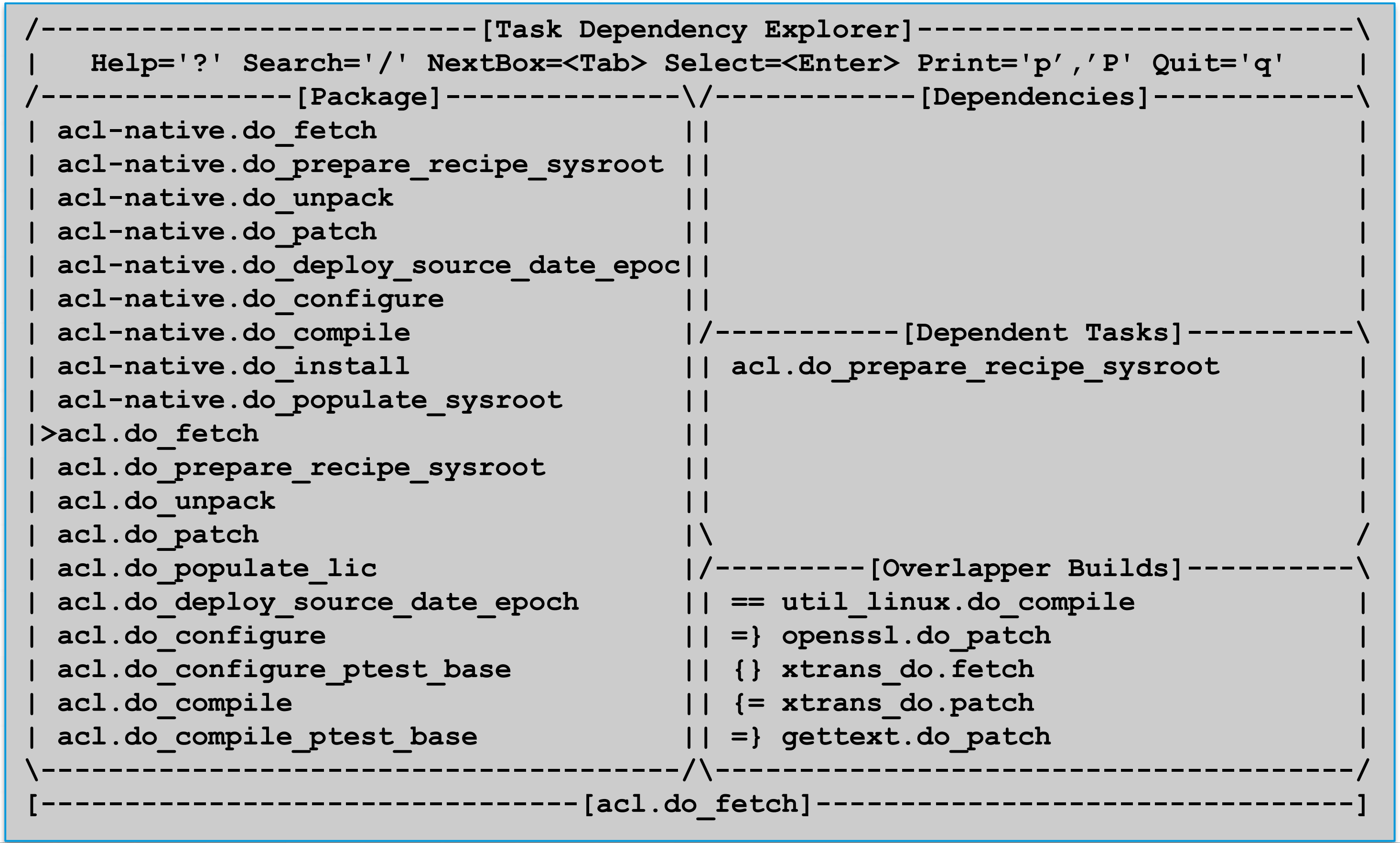
Thanks for your time

Example Task and Recipe Build Analysis Script, from the ELC Presentation

```
$ ./event_overlap.py --help
Commands: ?
 ?                               : show help
 b,build    [build_id]           : show or select builds
 d,data                          : show histogram data
 t,task     [task]               : show task database
 r,recipe   [recipe]             : show recipes database
 e,events   [task]               : show task time events
 E,Events   [recipe]             : show recipe time events
 o,overlap  [task|0|n]           : show task|zero|n_max execution overlaps
 O,Overlap  [recipe|0|n]         : show recipe|zero|n_max execution overlaps
 g,graph    [task]    [> file]   : graph task execution overlap
 G,Graph    [recipe]  [> file]   : graph recipe execution overlap
 h,html     [task]    [> file]   : HTML graph task execution overlap [to file]
 H,Html     [recipe]  [> file]   : HTML graph recipe execution overlap [to file]
 q,quit                          : quit

Examples:
  * Recipe/task filters accept wild cards, like 'native-*', '*-lib*'
  * Recipe/task filters get an automatic wild card at the end
  * Task names are in the form 'recipe:task', so 'acl*patch'
    will specifically match the 'acl*:do_patch' task
  * Use 'o 2' for the tasks in the two highest overlap count sets
  * Use 'O 0' for the recipes with zero overlaps
```

Example taskexp_cli showing overlapping task execution



```
/--------------------------[Task Dependency Explorer]--------------------------\
|   Help='?' Search='/' NextBox=<Tab> Select=<Enter> Print='p’,’P' Quit='q'    |
/---------------[Package]--------------\/------------[Dependencies]------------\
| acl-native.do_fetch                  ||                                      |
| acl-native.do_prepare_recipe_sysroot ||                                      |
| acl-native.do_unpack                 ||                                      |
| acl-native.do_patch                  ||                                      |
| acl-native.do_deploy_source_date_epoc||                                      |
| acl-native.do_configure              ||                                      |
| acl-native.do_compile                |/-----------[Dependent Tasks]----------\
| acl-native.do_install                || acl.do_prepare_recipe_sysroot        |
| acl-native.do_populate_sysroot       ||                                      |
|>acl.do_fetch                         ||                                      |
| acl.do_prepare_recipe_sysroot        ||                                      |
| acl.do_unpack                        ||                                      |
| acl.do_patch                         |\                                      /
| acl.do_populate_lic                  |/---------[Overlapper Builds]----------\
| acl.do_deploy_source_date_epoch      || == util_linux.do_compile             |
| acl.do_configure                     || =} openssl.do_patch                  |
| acl.do_configure_ptest_base          || {} xtrans_do.fetch                   |
| acl.do_compile                       || {= xtrans_do.patch                   |
| acl.do_compile_ptest_base            || =} gettext.do_patch                  |
\--------------------------------------/\--------------------------------------/
[--------------------------------[acl.do_fetch]--------------------------------]
```

“{“ : task started
“}” : task stopped
“=“ : task in progress