Hash Equivalence and Reproducible Builds

Embedded Linux Conference 2020
Joshua Watt
July 1st, 2020

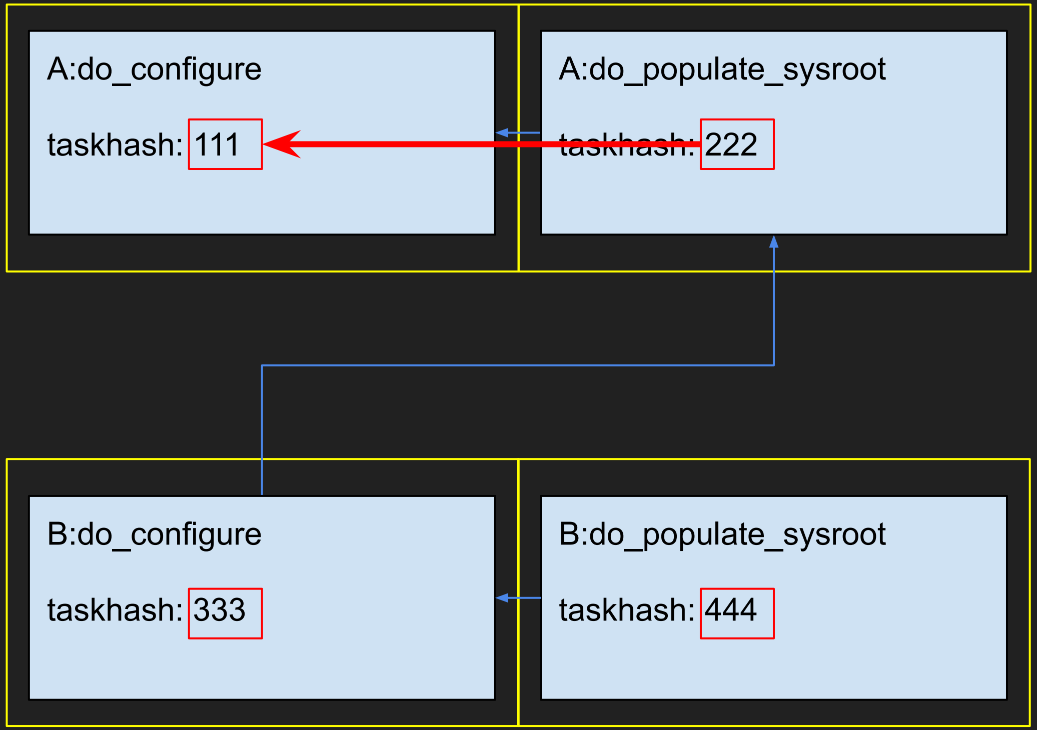
Hash Equivalence

What is Hash Equivalence?

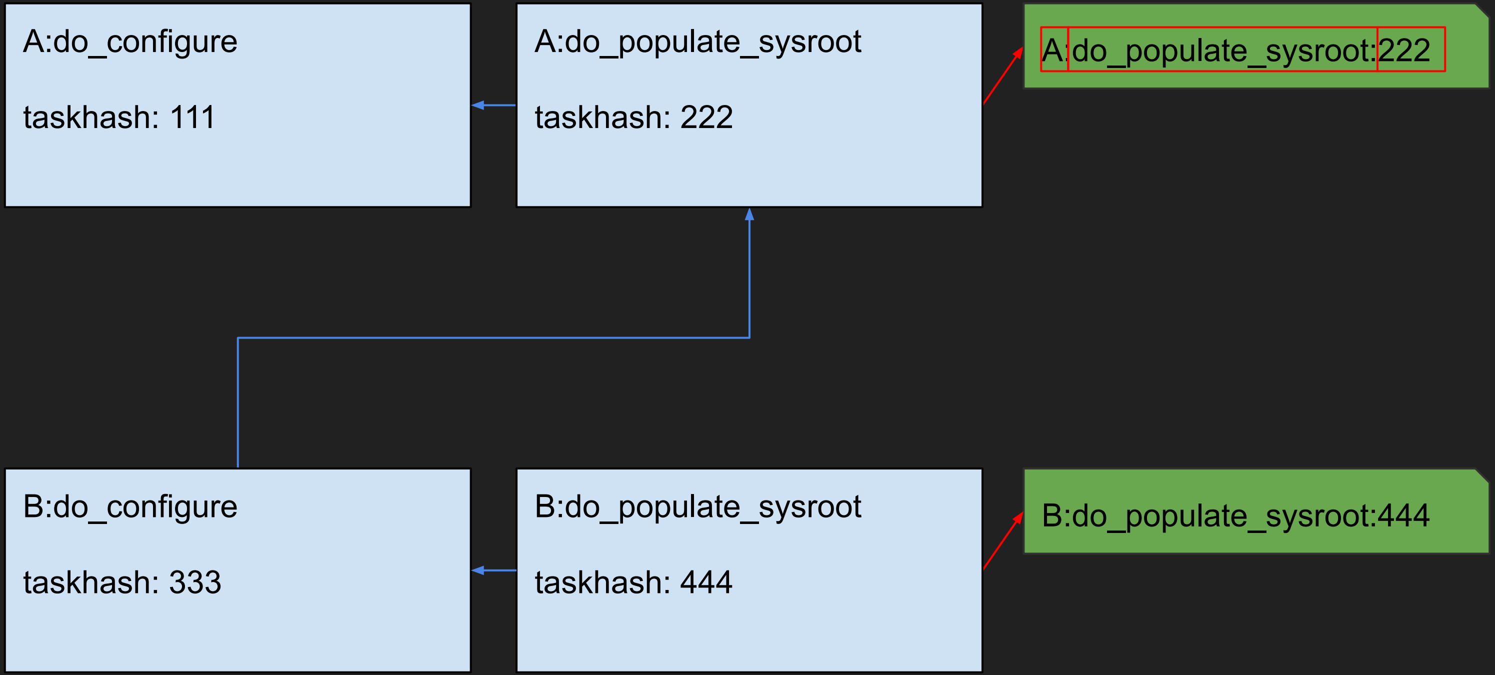
- Build accelerator
- Detects build tasks that have the same output even though the input signatures (hash) differ
- Enabled by default in poky starting with Yocto 3.0 (zeus)
- Enabled with:

```
BB_SIGNATURE_HANDLER = “OEEquivHash”
BB_HASHSERVER = “auto”
```

RunQueue example



RunQueue example



A:do_configure

taskhash: 111

A:do_populate_sysroot

taskhash: 222

A:do_populate_sysroot:222

B:do_configure

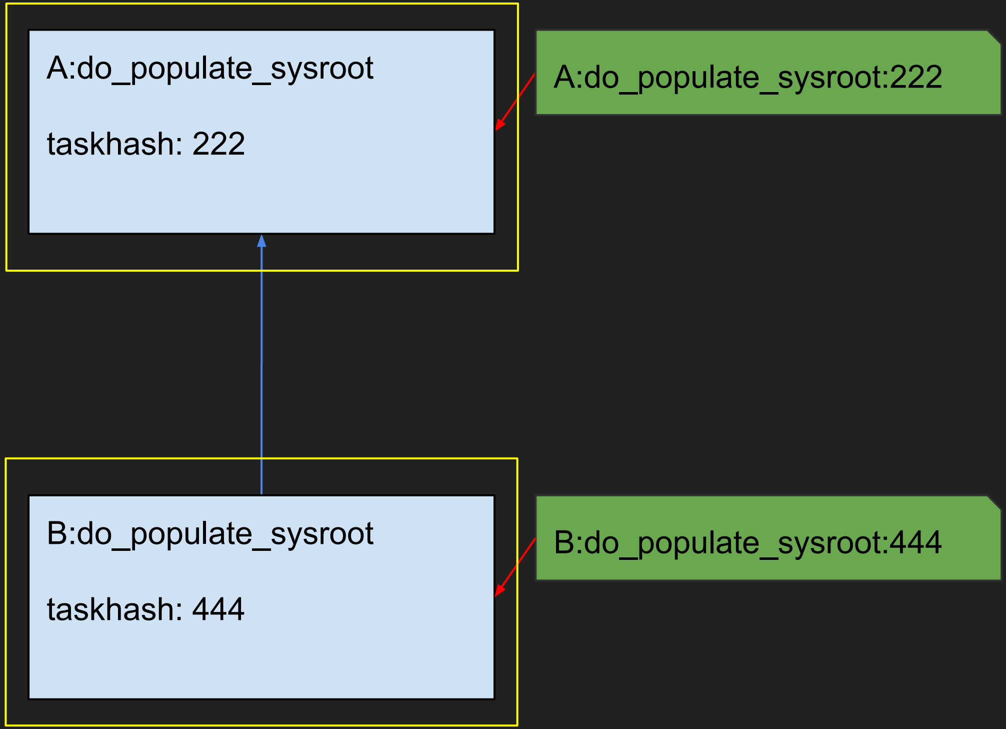
taskhash: 333

B:do_populate_sysroot

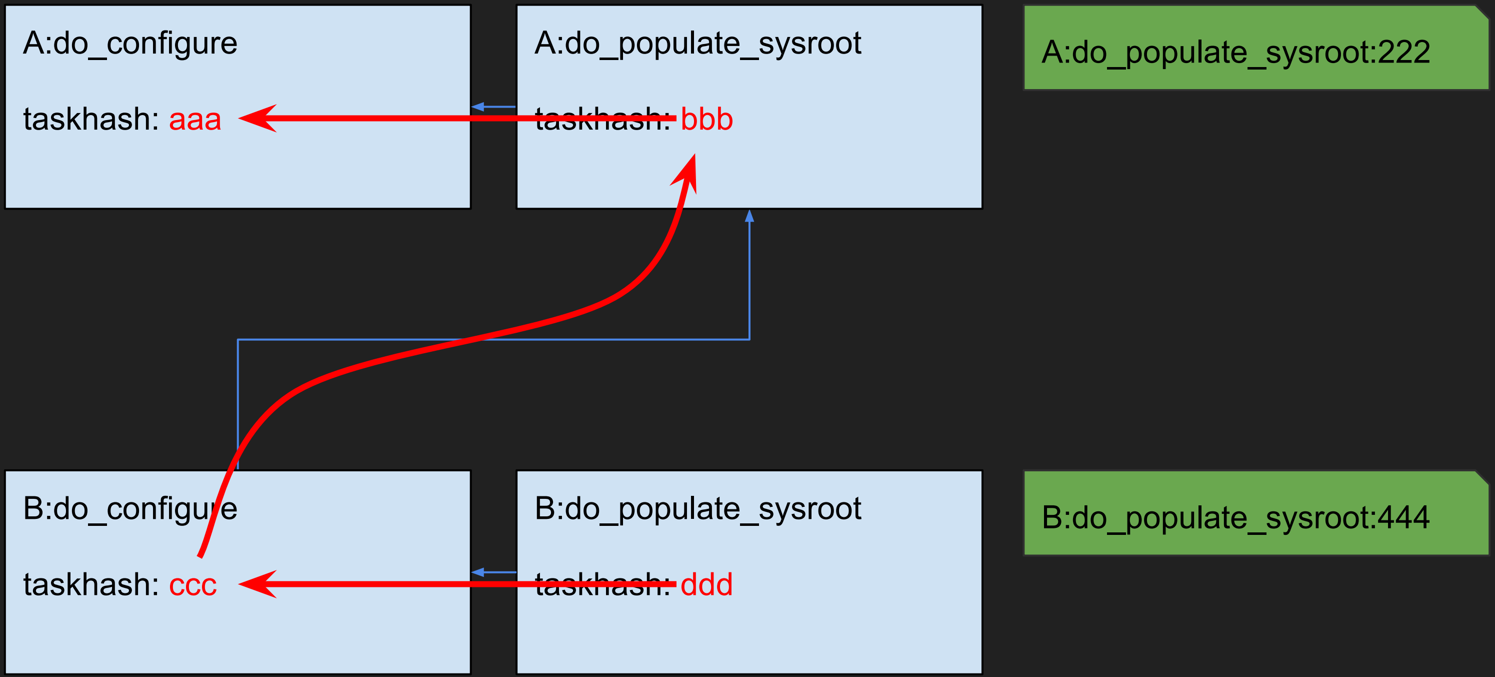
taskhash: 444

B:do_populate_sysroot:444

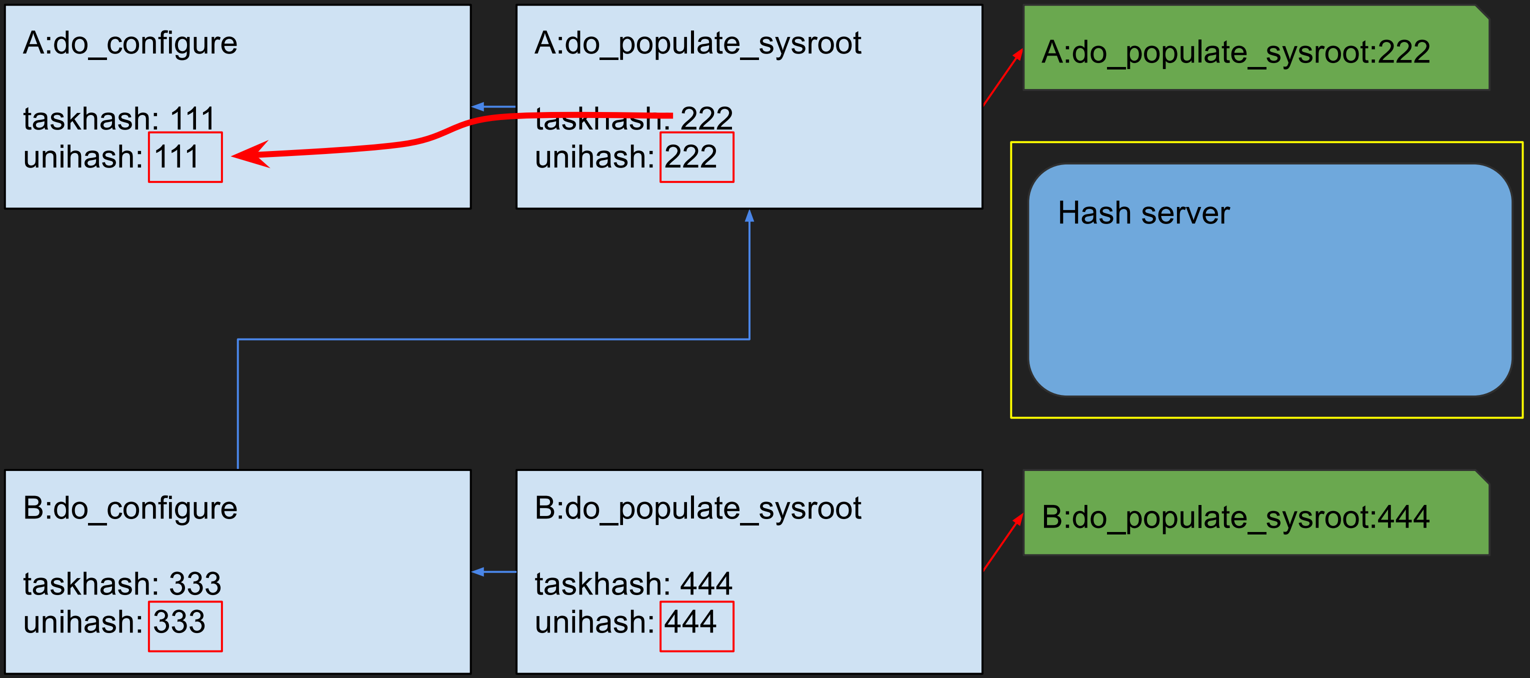
RunQueue example



RunQueue example



RunQueue with Hash Equivalence example



A:do_configure

taskhash: 111
unihash: 111

A:do_populate_sysroot

taskhash: 222
unihash: 222

A:do_populate_sysroot:222

Hash server

B:do_configure

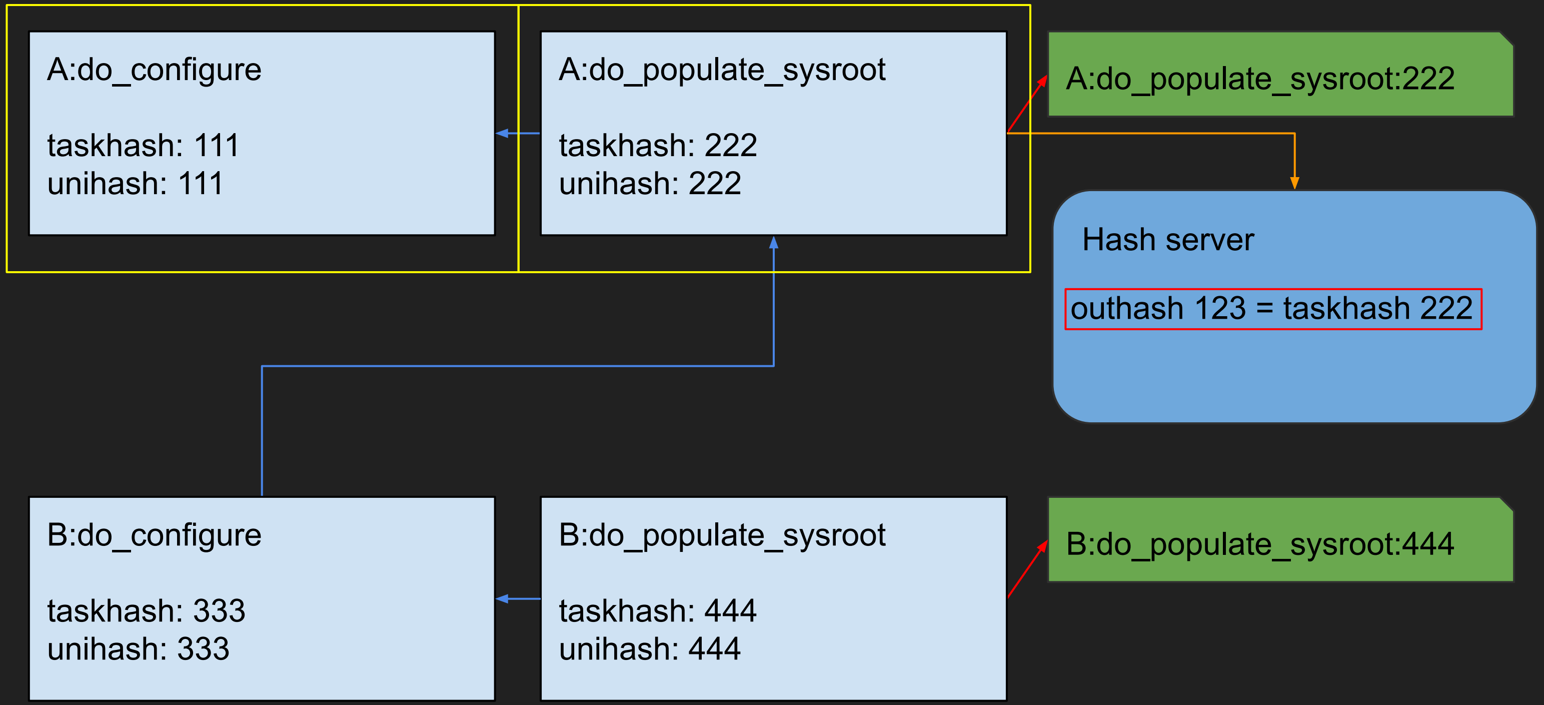
taskhash: 333
unihash: 333

B:do_populate_sysroot

taskhash: 444
unihash: 444

B:do_populate_sysroot:444

RunQueue with Hash Equivalence example



A:do_configure

taskhash: 111
unihash: 111

A:do_populate_sysroot

taskhash: 222
unihash: 222

A:do_populate_sysroot:222

Hash server

outhash 123 = taskhash 222

B:do_configure

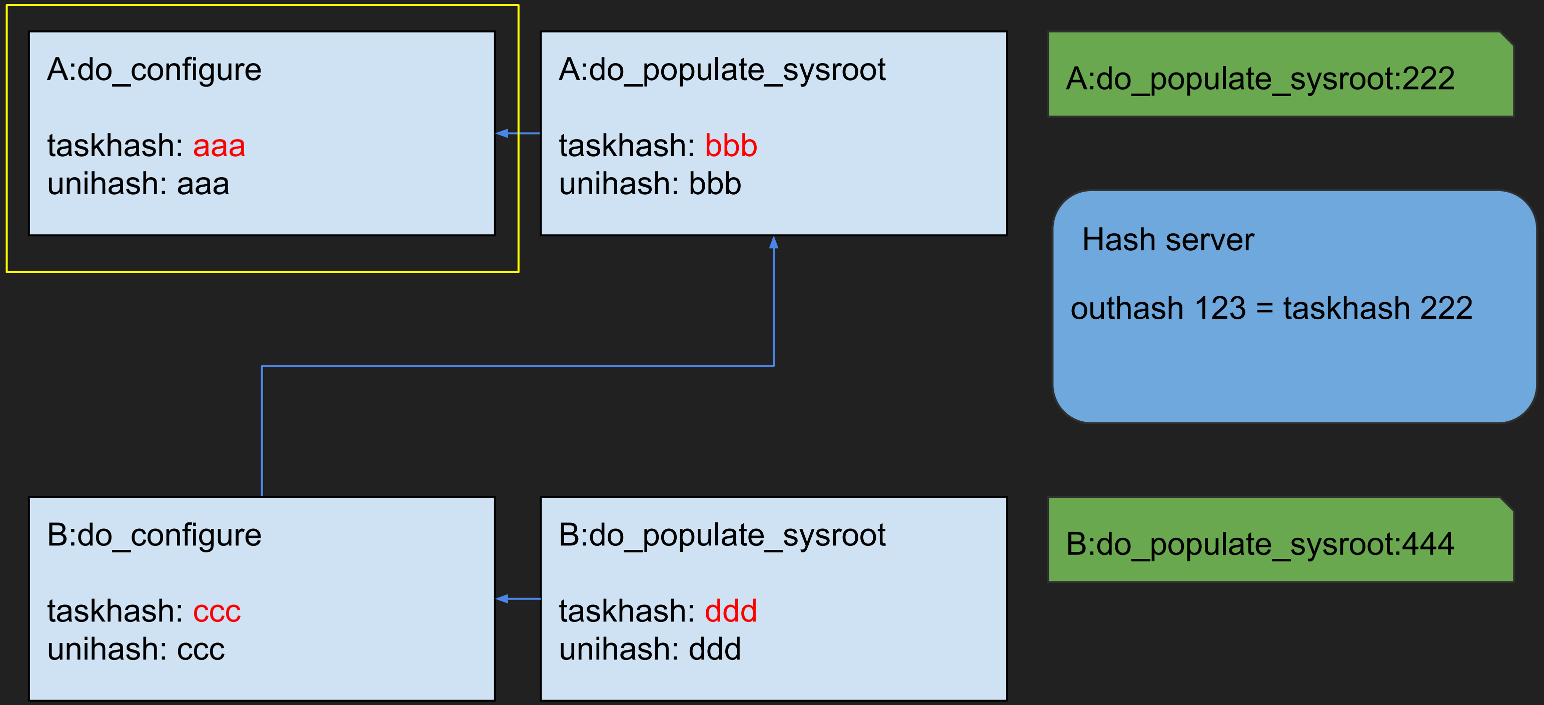
taskhash: 333
unihash: 333

B:do_populate_sysroot

taskhash: 444
unihash: 444

B:do_populate_sysroot:444

RunQueue with Hash Equivalence example



A:do_configure

taskhash: aaa
unihash: aaa

A:do_populate_sysroot

taskhash: bbb
unihash: bbb

A:do_populate_sysroot:222

Hash server

outhash 123 = taskhash 222

B:do_configure

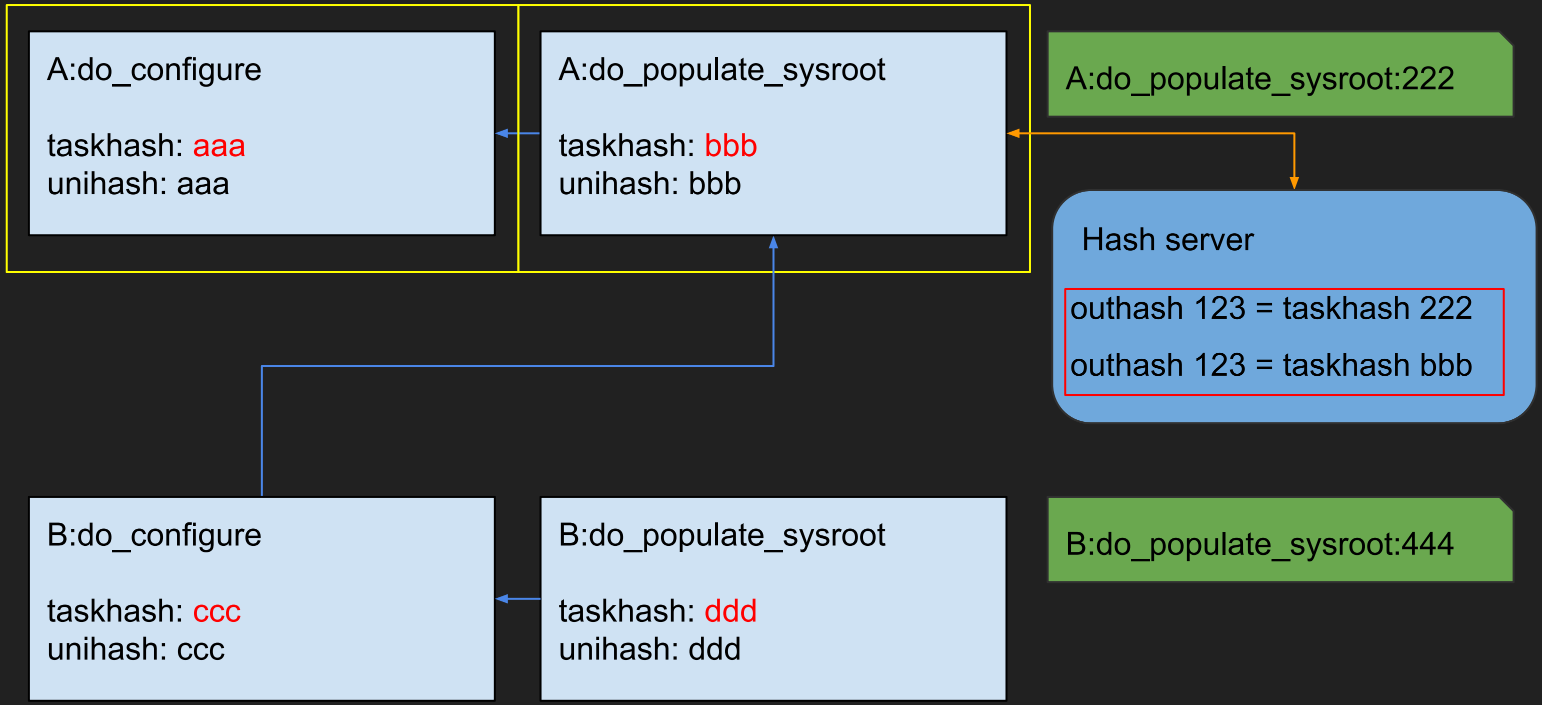
taskhash: ccc
unihash: ccc

B:do_populate_sysroot

taskhash: ddd
unihash: ddd

B:do_populate_sysroot:444

RunQueue with Hash Equivalence example



A:do_configure

taskhash: aaa
unihash: aaa

A:do_populate_sysroot

taskhash: bbb
unihash: bbb

A:do_populate_sysroot:222

Hash server

outhash 123 = taskhash 222

outhash 123 = taskhash bbb

B:do_configure

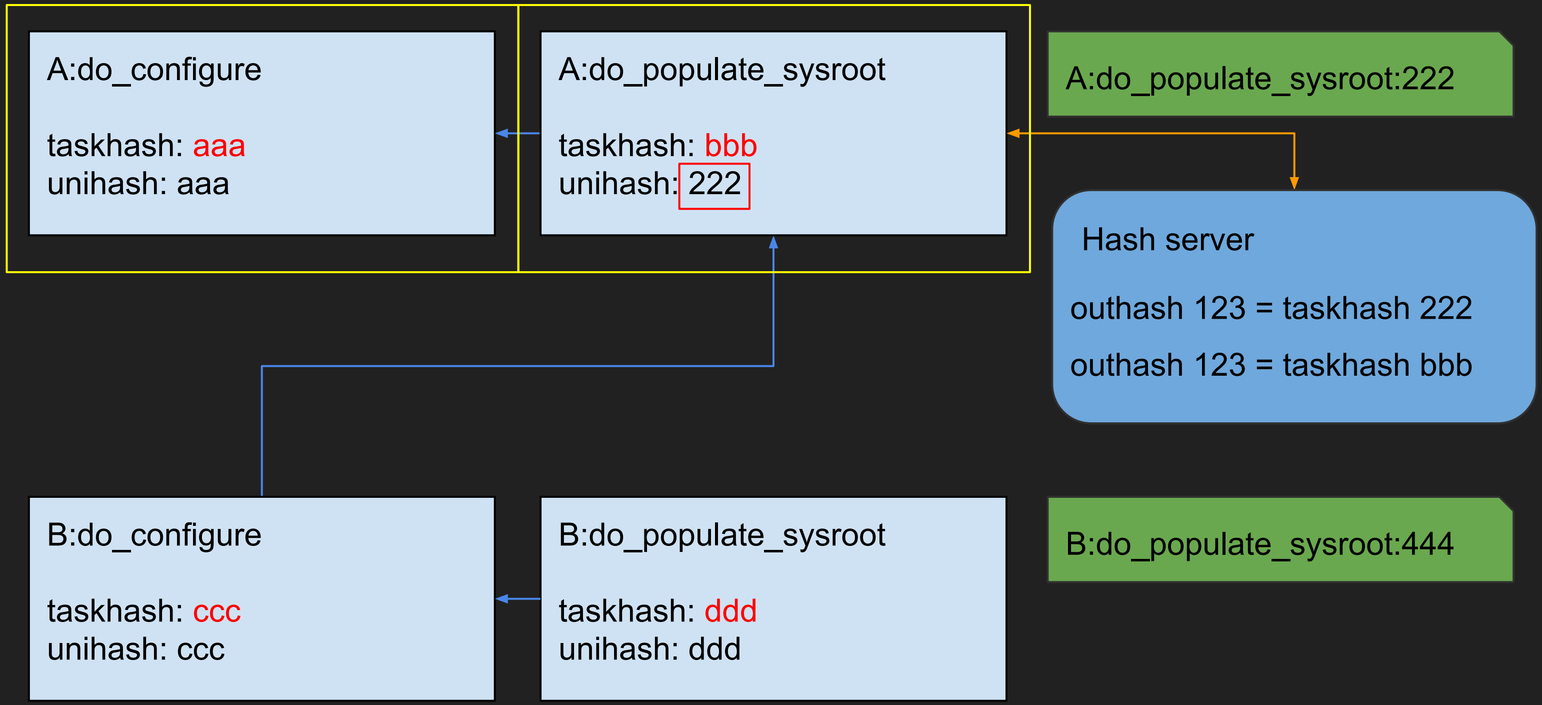
taskhash: ccc
unihash: ccc

B:do_populate_sysroot

taskhash: ddd
unihash: ddd

B:do_populate_sysroot:444

RunQueue with Hash Equivalence example



A:do_configure

taskhash: aaa
unihash: aaa

A:do_populate_sysroot

taskhash: bbb
unihash: 222

A:do_populate_sysroot:222

Hash server

outhash 123 = taskhash 222

outhash 123 = taskhash bbb

B:do_configure

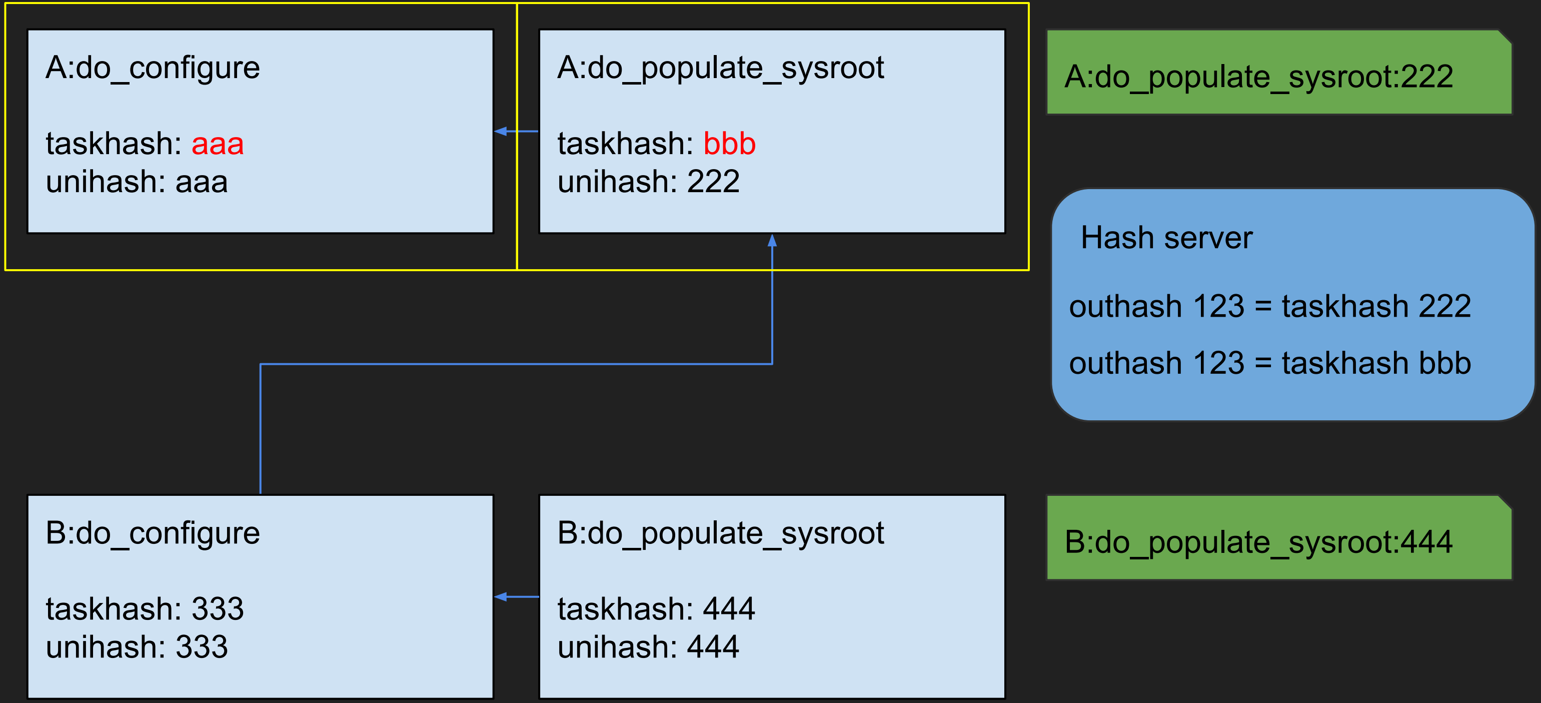
taskhash: ccc
unihash: ccc

B:do_populate_sysroot

taskhash: ddd
unihash: ddd

B:do_populate_sysroot:444

RunQueue with Hash Equivalence example



A:do_configure

taskhash: aaa
unihash: aaa

A:do_populate_sysroot

taskhash: bbb
unihash: 222

A:do_populate_sysroot:222

Hash server

outhash 123 = taskhash 222

outhash 123 = taskhash bbb

B:do_configure

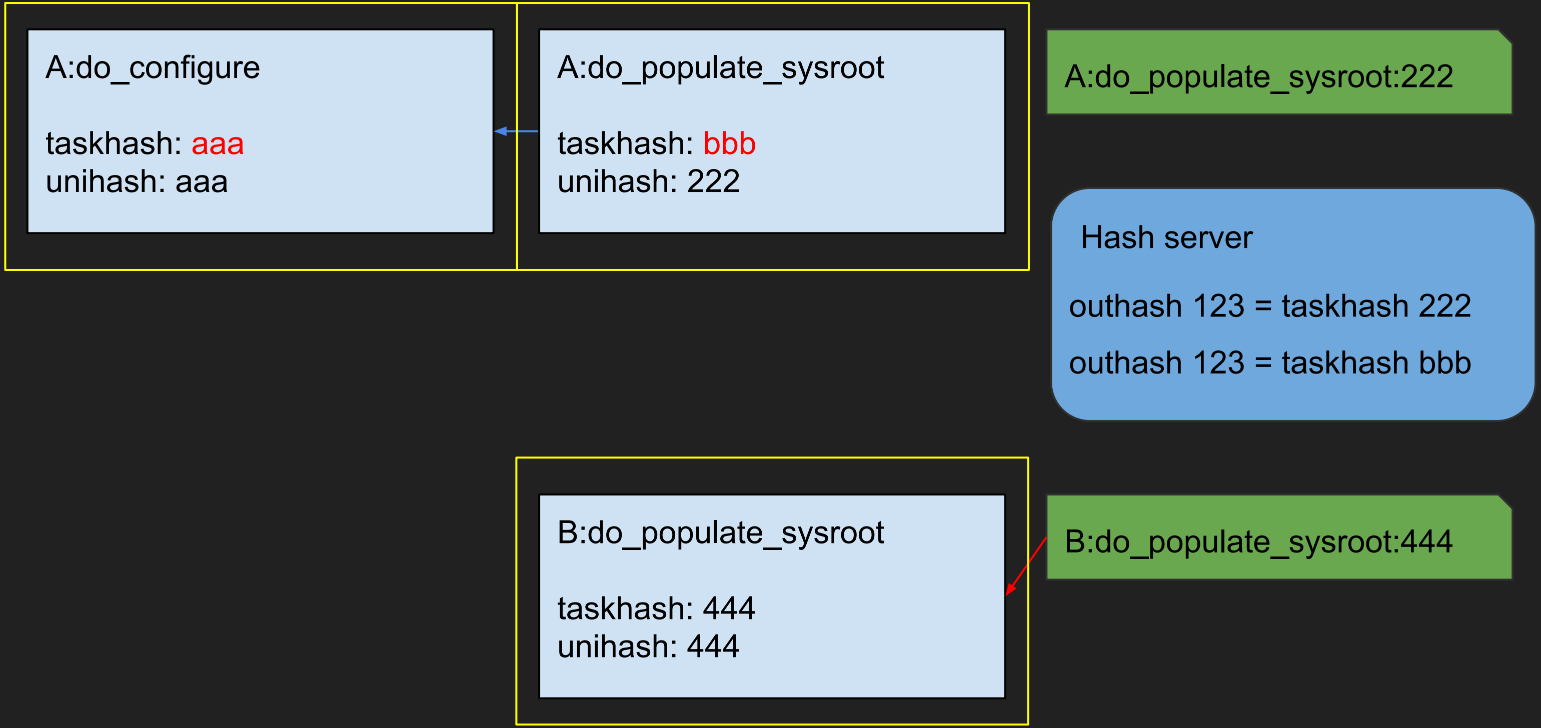
taskhash: 333
unihash: 333

B:do_populate_sysroot

taskhash: 444
unihash: 444

B:do_populate_sysroot:444

RunQueue with Hash Equivalence example



A:do_configure

taskhash: aaa
unihash: aaa

A:do_populate_sysroot

taskhash: bbb
unihash: 222

A:do_populate_sysroot:222

Hash server

outhash 123 = taskhash 222

outhash 123 = taskhash bbb

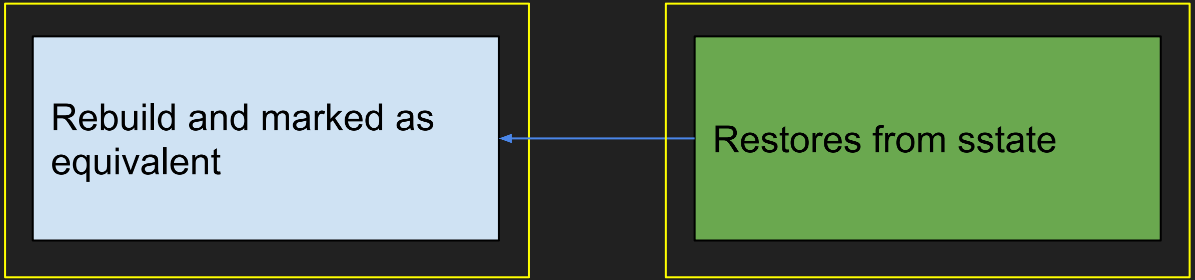
B:do_populate_sysroot

taskhash: 444
unihash: 444

B:do_populate_sysroot:444

Trivial Recipe Changes

- Whitespace changes
- Unused code paths
- Variable ordering



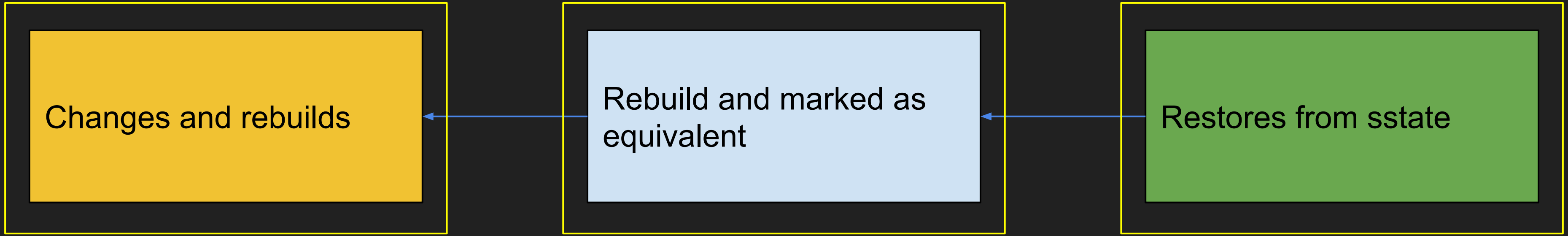
Rebuild and marked as equivalent

Restores from sstate

Recipe Updates

- Library updates & CVE Fixes
- Native tool updates

In many of these cases, downstream recipes will generate the same output when rebuilt and be marked as equivalent



Output Hash Calculation

- Checksum of all files and metadata that goes into an sstate object (archive)
- Signature files can be found in `${T}/depsig.`*`task`*

```
$ ls -1 temp/depsig.*
temp/depsig.do_deploy_source_date_epoch
temp/depsig.do_deploy_source_date_epoch.10403
temp/depsig.do_package
temp/depsig.do_package.35371
temp/depsig.do_packagedata
temp/depsig.do_packagedata.23347
temp/depsig.do_package_qa
temp/depsig.do_package_qa.7604
temp/depsig.do_package_write_ipk
temp/depsig.do_package_write_ipk.20118
temp/depsig.do_populate_lic
temp/depsig.do_populate_lic.16631
temp/depsig.do_populate_sysroot
temp/depsig.do_populate_sysroot.2568
```

Hash Equivalence Server

- Reference implementation included in bitbake
  - Started automatically over unix domain socket if `BB_HASHSERVE = “auto”`
- Hash Equivalence server can be shared between multiple clients
  - Specify a server with `BB_HASHSERVE = “host:port”`
  - Collective reporting of hash equivalence
- Server should be maintained with the sstate cache
  - Otherwise the server could report sstate hashes that don’t exist!

Debugging Hash Equivalence

Extra logging can be enabled using bitbake structured logging:

local.conf:

```
BB_LOGCONFIG += “log.json”
```

log.json:

```
{
  "version": 1,
  "loggers": {
    "BitBake.SigGen.HashEquiv": {
      "level": "VERBOSE",
      "handlers": ["BitBake.verbconsole"]
    },
    "BitBake.RunQueue.HashEquiv": {
      "level": "VERBOSE",
      "handlers": ["BitBake.verbconsole"]
    }
  }
}
```

Future Improvements

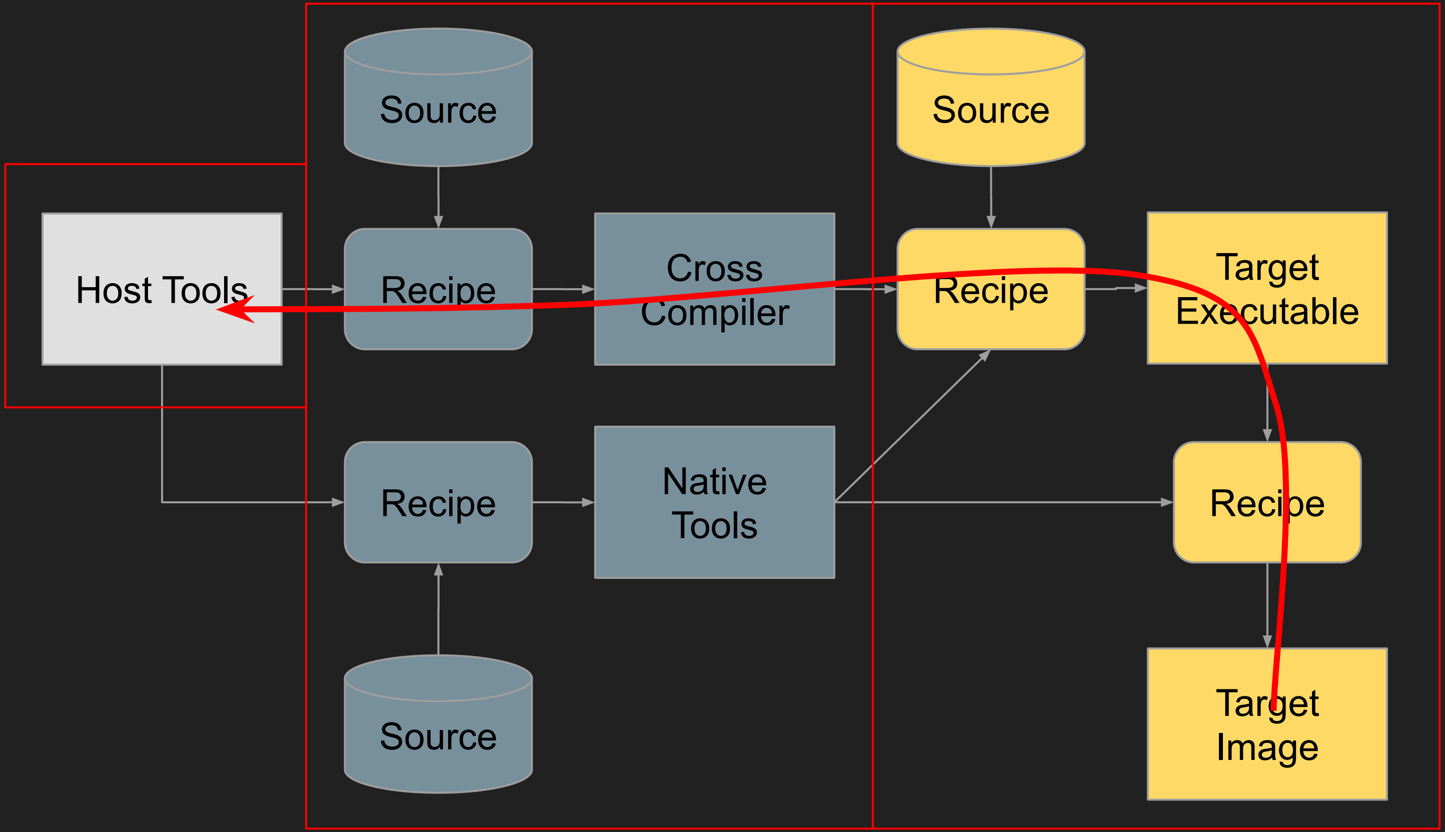
- Read-Only (or Read-mostly) hash server for CI centered workflows
- Better hash equivalence database introspection tools

Reproducible Builds

Why are reproducible builds important?

1. Improves Hash Equivalency effectiveness
2. Code archival
3. Software supply chain
   - Verify the toolchain is not compromising your target
   - https://reproducible-builds.org/docs/buy-in/

Build Flow


Source
Host Tools
Recipe
Cross Compiler
Recipe
Native Tools
Source
Source
Recipe
Target Executable
Recipe
Target Image

Quality Assurance Test

- Reproducible build tests are run regularly as part of the OE QA test suite since the 3.0 (zeus) release
- Ensure packages produced by recipes are binary reproducible
- Run regularly as part of patch verification

Features of Reproducibility QA Test

- Does an “A” and a “B” build. Build “A” can use sstate, build “B” is from scratch
- Tests core-image-minimal, core-image-full-cmdline, and core-image-sato
- Tests deb and ipk package formats
- Tests for build path differences
- Partial tests for timestamp differences
    - Test builds are done sequentially, can’t detect differences in day, month or year
- Partial tests for host differences
    - Yocto Project runs reproducible builds on a variety of hosts
- Uses diffoscope to report non-reproducible packages in browsable HTML format

Extending Quality Assurance Test

- The QA test for reproducibility is designed to be easy to extend and run for testing your own images:

```
$ cat lib/oeqa/selftest/cases/my-reproducible.py
from oeqa.selftest.cases.reproducible import ReproducibleTests

class MyReproTests(ReproducibleTests):
    images = [‘my-image’]
```

```
$ oe-selftest -r my-reproducible
```

Future improvements

- Improve the number of packages tested
  - SDK images (close…. just need perf to be reproducible!)
  - World images
- Test rpm packages
- Test final root filesystem images
- Test other deployed objects (e.g. Kernel, bootloader)
- Test native tools
- Improve the number of architectures the Yocto Project autobuilder tests
  - AArch64
- Use DisorderFS to test for file system ordering non-reproducibility
- Fake timestamps (e.g. libfaketime) to do proper timestamp testing
  - Needs work to interact properly with pseudo

Conclusion

- Hash Equivalence can help reduce build times
- Why we want reproducible builds
- There are lots of ways to get involved

Contacting the Community

- Freenode IRC
    - #yocto
    - #oe
- Monthly virtual planning meeting
    - First Tuesday of each month at 8:00 AM Pacific Time
- Weekly Bug Triage
    - Every Thursday at 7:30 AM Pacific Time

Contact Information

- Joshua.Watt@garmin.com
- JPEWhacker@gmail.com
- IRC: JPEW

References

- OpenEmbedded homepage
  - http://www.openembedded.org/
- OpenEmbedded Calendar
  - https://calendar.google.com/calendar/embed?src=gsu6m5g9utl4elkjlct144ihko%40group.calendar.google.com
- Yocto Project Calendar
  - https://calendar.google.com/calendar/embed?src=theyoctoproject%40gmail.com
- Yocto Project Public Virtual Meetings
  - https://www.yoctoproject.org/public-virtual-meetings/
- Reproducible Builds
  - https://reproducible-builds.org/
- DisorderFS
  - https://salsa.debian.org/reproducible-builds/disorderfs
- Libfaketime
  - https://github.com/wolfcw/libfaketime

Questions?