What is Baserock?

A way of making our lives easier as embedded systems developers and consultants

?
Dreamed up on a late night working on a complex Meego project with
a super long develop-deploy-test-run-review-merge cycle, 800MB images
which take 4 hours to build, random RPM's floating around in different
directions, various systems of version numbering which may or may not
be adhered to, no consistency between git repo / packaging / executable
names, tarballs checked into git, super stale versions of lots of
components etc., etc.
Worst of all: no concept among developers on how to contribute upstream,
and no easy way to show them.
This is a desktop conference, it's not about embedded, but we can take
the same principles and apply them - should my desktop OS really be 12GB;
> 2000 packages? And, it's nice being paid to hack on stuff, which happens
a lot more often in embedded than the desktop space.

What is Baserock?

Not a distribution

A set of tools for managing and building systems for Linux-based devices
Maybe you saw Lars' talks, either in Montreal or at FOSDEM: sketching ideas that have turned into Baserock
No packages. Packaging goes out of date so fast. Meego: it's scraped from all over; often Fedora from 4 years ago. And it's largely duplicated effort!

? How often do you build something manually / from jhbuild because packaged version is too old?
? How often do you opt to update / create packaging for your distro vs. being lazy and just using your built version (and there are probably a lot of distro people at Guadec!)
Choice between stable (but out of date) / medium (every 6 months - OK) / unstable (randomly broken day-to-day but at least you get shiny features) - we can do better.
Closest is Yocto - but without the complexity or legacy

What is Baserock?

The cool stuff we've been wanting for years

* Atomic OS update / rollback

##* Branch whole system
##* Check out and hack on individual components and test in isolation
##http://codethink.branchable.com/baserock/developer-story/

What is Baserock?

A good way to learn about geology

* Strata
* Chunks
* Erratics
* Morphologies
* Lorry
* Roadtrain
* Montoya
* Systems (not all terms are from geology)

100m overview: strata, chunks, morph, morpologies

Key points

* Starting from scratch, from the ground up

* Minimalist

* Easy to get started

Hence "Baserock" - it's a foundation for building real-world systems
on top of. 'foundation' aims to be little more than linux + systemd +
busybox (really? + dbus, glib, libstdc++, ...) but then, you can always
strip more stuff out if you want ...
If anyone was going to ask "Why not use Yocto", there's part of the answer
We care about disk space, boot speed etc. and being minimal from the get-go
helps a lot! Also, no packages!
Lars built an 11MB x86 image .. but this contradicts with the fact
that baserock systems must be able to build themselves, right?
I've been working on the project 2 weeks, personally,

Key points

* Everything is in git

* Reproducible builds

* Continuous integration testing

* No cross-compiling

* btrfs

it's cool these days. Also, we've seen a lot of people who don't know how to use git at all, and don't learn. Object files checked in! Having a fixed git workflow for everything means we all have to learn to use it and have one place to look to make sure we aren't getting commit logs like this:

 "fix"
 "fix"
 "fix #1"

Sane upstreams can be built directly, others we mirror ('lorry') and add a chunk morphology, describing how to build

If you use OBS - just imagine that you don't have a separate VCS for packaging now.

cf. Debian - here you can store your packaging in whatever VCS you want, storing just the packaging, or the whole upstream, plus patches, or not the patches, they could go in Quilt instead ... doing work on a large number of patches is a *nightmare*

Inputs are hashed - like NixOS / (like OSTree???) and compilation takes place in an isolated chroot. Should be FAST (it isn't yet, but should be) because we're not installing 600 packages, we're installing like 3 strata, and there's work on the way to make faster still with hardlinks inside the chroot etc. etc. ? ??

Is this FREE? should I mention it?

Strata, chunks and erratics

Erratics:

http://codethink.branchable.com/baserock/ideas/erratics/

Verified because we always build in a staging chroot.

Baserock by numbers

Gnome on Baserock

DEMO!!!!

Basically, this is an entire operating system built from git!
Suprisingly stable ;)

What can it do?

* Rebuild itself from source

* Branch and hack on it

* Caveats

include the cached git repos
caveats

libtasn1 - files deleted from source tree as part of configure

No cross-compiling

* It's an alternative code path, and often not widely used

* Some things don't work at all
  * gobject-introspection being one of them:
    https://bugzilla.gnome.org/show_bug.cgi?id=592311
  * Also, binary tools that need to run as part of the build process
  * Automated testing

* Cross-compiling isn't going to disappear overnight, but we are saving ourselves a lot of pain

* ARM devices are faster than they used to be

* One of our goals is to have systems which are easy to hack: if the system simply can't compile itself then we may as well give up.

* Distributed work, distcc, caching of chunk compile results, ccache.

Especially when building everything from git is a goal, lots of
upstreams have to be patched to fix cross-compiling for certain
targets, Yocto / OpenEmbedded carry patches that upstreams won't
take to fix crosscompile in various ways ...
autoconf does wildly different stuff when cross-compiling, to the
point where you often want to configure on target instead of host,
taking extra effort ...
It's the free software dream!
Distributed work is super cool but nonfree!!! No problem for x86 though.
We farm out stuff to machines with distcc -- easier than cross-compiling
because no configure hassle, nothing more than a crosscompiler required
really. Less fragile than Scratchbox - ?

http://codethink.branchable.com/baserock/ideas/distcc-cross-compiling/

None of this is theoretical - baserock *is* bootstrapped on ARM.
Javier can say more about this!
VM's - why didn't we use Scratchbox ????? you may not have a VM. and
the great thing is you can just go with Baserock and fix speed problems
later, whereas if we took the scratchbox approach maybe you first have
to implement your device in qemu or do a bunch of setup on some target HW
...
Main argument against of course is speed ---- what do you say here ???
http://www.scratchbox.org/documentation/user/scratchbox-1.0/html/tutorial.html

If you've been comparing us to OSTree in yr head, here's difference #1: instead of reimplementing git, we use btrfs subvolumes and snapshots. Only time will tell which is the better approach I guess.
eg. When I was looking through Morph logs for the first time, I noticed we copied the whole .git directory to the build root every time. WHAT! But of course we have copy-on-write, so it's OK!
For example, the kernel on my laptop crashed while I was debugging morph's test suite -- and btrfs driver crashed the kernel while unmounting a loopback mount.
"mixed block groups" since 2.6.37 - now forced for partitions < 1GiB. Makes it work OK apparently.

Development cycle

Not yet

* x86_32

* Lots of things need root

* Plenty of stuff unfinished

just for simplicity; but you need 64bit to test right now
linux-user-chroot will help in future

Contribute!

http://www.baserock.org/

Source: http://roadtrain.codethink.co.uk/gitweb/

Prepared answers

Have you considered parallel / distributed building?

Free Morph supports make -j$N and even distcc

For a fee we will provide code to fully distribute work: running
multiple instances of Morph on multiple machines so systems get built
really fast. Intended for use in companies building big systems.

Branching - can I branch a whole system at the latest stable version of each
chunk? What about per-strata (eg. Gnome 3.6)? Mix and match?

Codethink will offer paid 'curated upstream' service. *** Need info on this ***

(How) do you plan to monetize Baserock?

Paid support features will include:
 - parallel distributed build: separate worker instances of Morph and a
   master instance, on different machines. (distcc is free, but you only
   have one chunk at a time).
 - curated upstream: we'll handle branching etc. whereas our public mirrors
   will have just master of everything. Of course, you could handle this
   yourself (quite trivially for certain components, eg. all of Gnome has
   a 'gnome-3-6' branch) ...
 *** Need more info on that! ***