Chrome OS Firmware Summit

Sundries

bhthompson & dparker

Feb 20th, 2014

The Beginning of the End

- Repo & Gerrit
- Portage
- Firmware Branches
- Keys & Signing
- Code Reviews
- CPFE
- Git
- Sausage?
- FAFT

Google

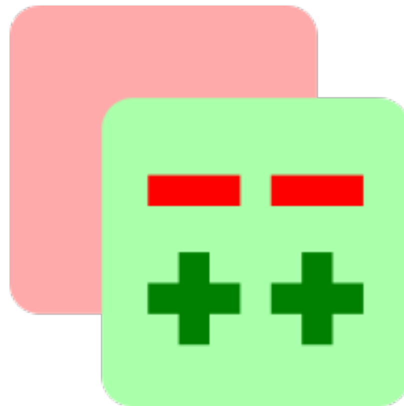
Disclaimer….

This presentation is not a quick-start guide.

See http://www.chromium.org/chromium-os/quick-start-guide for that.

Your Google contact will also help you get started with getting access to private code and sites.

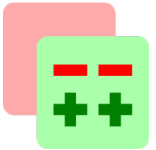
Repo & Gerrit



Repo

- Repo is a tool developed for Android for **repository management** that has also been adopted for Chromium OS.
- Repo lets you download and make changes to the ~156 GIT repositories that make up Chromium OS.
- Repo commands take the form of **repo COMMAND [ARGS].**
- The most common commands used with Chrome OS are:
  - **repo init** -- generate new repository files
  - **repo sync** -- bring your sources up to date
  - **repo start** -- start a local branch for development
  - **repo upload** -- upload changes for code review
  - **repo help** -- display help information
- Additional documentation on repo can be found at http://source.android.com/source/using-repo.html

Gerrit



- Gerrit is the **code review system** used by Chromium OS.
- Gerrit uses a web interface available at https://chromium-review.googlesource.com (crosreview.com)
- Submissions to Gerrit are made from within the Chromium OS development environment using the **repo upload** command.
- Anyone may create an account and upload changes but only chromium.org members may approve them.
- Changes and review comments are public. Please be careful about mentioning OEM names of unannounced devices.

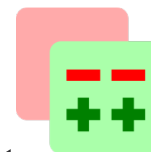
All | My | Admin | Documentation   Bernie Thompson <bhthompson@chromium.org> | Settings | Sign Out

Open Merged Abandoned   status:open Search

Search for status:open

ID	Subject	Owner	Project	Branch	Updated	V	R	CR
I5d8a7a5c	Fix incorrect TIME variables in two tests.	Scott Zawalski	chromiumos/third_party/autotest	master	1:08 PM	✓	✓	✓
Icc32a4b5	[cryptohome] [rfc] switch to Make	Elly Jones	chromiumos/platform/cryptohome	master	1:07 PM		-1	
I43988e7a	power: Rework the ambient light response logic.	Bryan Freed	chromiumos/platform/power_manager	master	1:06 PM			
I541c6a82	Re-enable realtimecomm_GTalk[Audio]Playground as a part of suite_HWQual.	Noah Richards	chromiumos/third_party/autotest	master	1:05 PM			
I5fc68c53	use --select for our core packages	Mike Frysinger	chromiumos/platform/crosutils	master	1:01 PM		✓	
I55c0ef58	Don't mark successful runs as failed in CLs.	David James	chromiumos/chromite	master	12:59 PM			

Gerrit-int



- An instance of Gerrit for making changes to board-specific private repositories. https://chrome-internal-review.googlesource.com (crosreview.com/i)
- Each board has a private repository (overlay) containing the files needed to build firmware. The system image.bin contains some private blobs (at least the x86 ones do).
- Anyone may register an account but you must be white-listed to see project-specific repositories. 'Read' access also controls which private repositories you can 'repo sync'.

(Screenshot redacted)

Portage

Portage

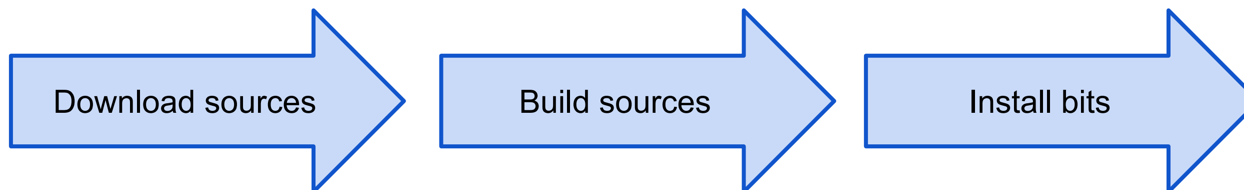
- Portage is a **package management system**, used in Gentoo Linux and Chromium OS

- Portage is designed to primarily work with **source** directly as opposed to a binary repository

- Portage is made up of:
    - A **portage tree** of **ebuilds**
    - Tools to use it, primarily **emerge**

Ebuilds

- Ebuilds are files that contain **Bash** script code to **acquire, build and install** a particular package.
- The **Portage Tree** consists of a set of ebuild files defining the various packages in the tree.
- The ebuild file names map directly to the package names you can pass into the emerge command.
- A **virtual** is essentially an ebuild pointer. It allows you to depend on something generic (like kernel) which maps to something specific (like kernel-vendor)



Eclasses

- An eclass is roughly the equivalent of a library or header file for an ebuild.
- Eclasses are written in Bash just like ebuilds and can be included in ebuilds to share common code.
- One example of an eclass specific to Chrome OS is cros_workon.

arbitrary.eclass

```
#arbitrary eclass comment
ECLASS_VALUE="1"
eclass_func1() {...}
eclass_func2() {...}
...
```

some.ebuild

```
#some ebuild comment
inherit arbitrary
eclass_func1
...
```

Emerge

- Emerge is the most common user interface to Portage.
- Similar to other package managers (for example, apt-get in Ubuntu) in the most general use it will bring in a package for you:
  - **emerge vim**
    - Bring vim into your filesystem
- In the Chromium OS development environment you can bring in packages for your target:
  - **emerge-arm-generic vim**
    - Bring vim into your target filesystem
- A common option for emerge is --unmerge which will remove a package from your filesystem.
- Emerge can be used to bring in specific packages or to work with a specific package as opposed to building all packages.

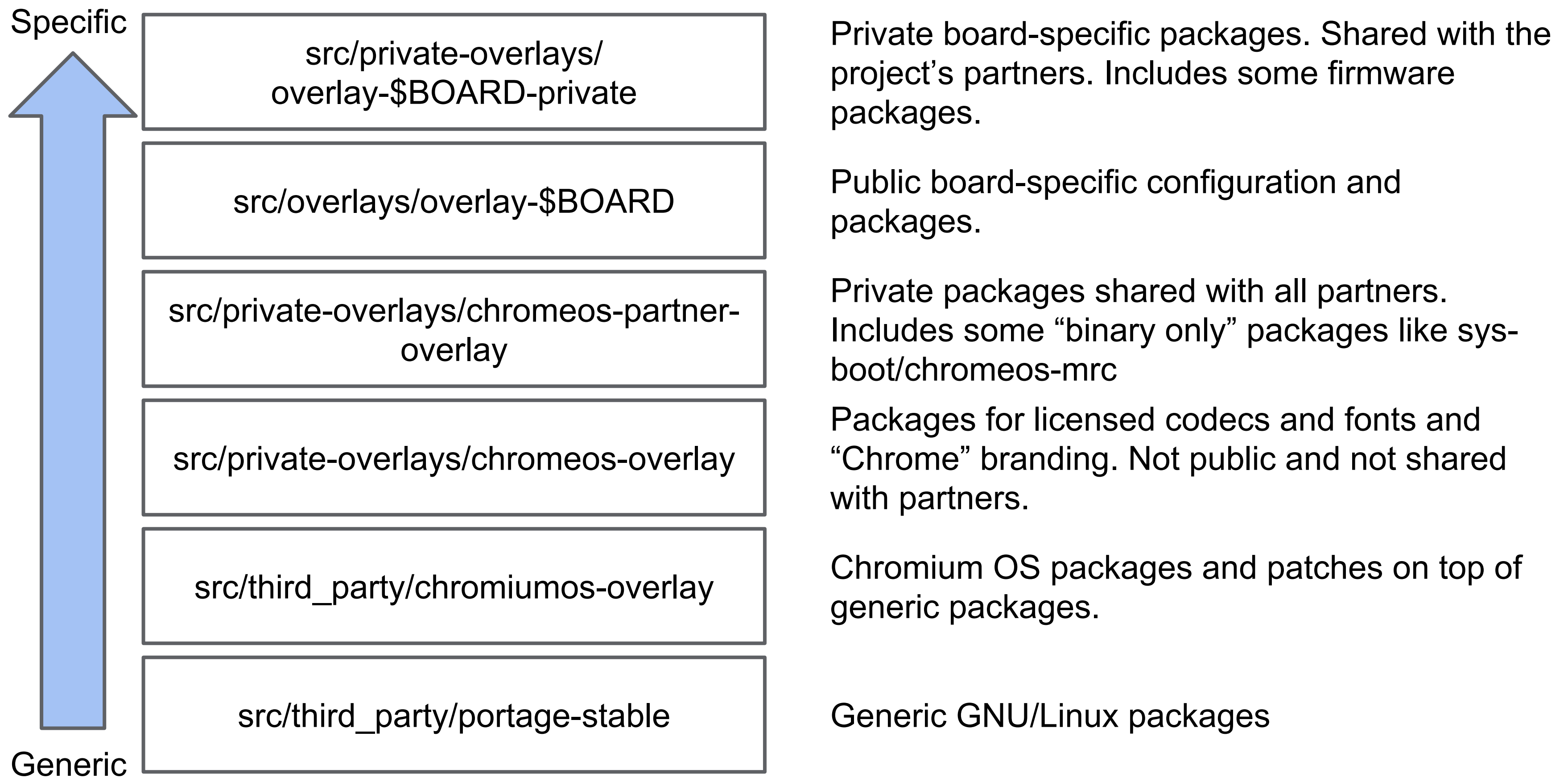
Equery

- Equery can be used to collect a variety of information from Portage. Some common commands are:
  - **equery belongs <file>** tells you what package a file comes from.
  - **equery which <package>** tells you which ebuild is used for a package.
  - **equery depends <package>** tells you what packages are dependent on a package.
  - **equery files <package>** tells you what files belong to a particular package.
  - **equery uses <package>** tells you what USE flags are enabled for a package.

Overlays

- An *overlay* is essentially a **partial Portage Tree** that goes over another Portage Tree.
- This allows a customized set of ebuilds to be used.
- With Chrome OS, a board will have an overlay for its own custom ebuilds, Portage files, and configurations.

Overlays



What is cros_workon?

- **cros_workon** is a bash script that is used to select which portage packages you intend to work on.

- The cros_workon **start** command will:
  - Use the -9999 ebuild for a package.
  - Build from local sources.

- The cros_workon **stop** command will:
  - Revert to the latest stable ebuild.
  - Use prebuilt binaries when possible.

- There are also cros_workon **info**, **list** and **list-all** commands which are useful.

- In summary, for cros_workon packages you should use the start command before trying to modify sources, otherwise your changes will not necessarily go into effect.

What is a cros_workon package?

- If an ebuild **inherits cros_workon** it is a cros_workon package.
- If you need to modify a cros_workon ebuild you must modify the **-9999** version of the ebuild.
    - When the change is submitted the builders will automatically up-rev the current stable ebuild and replace it with the contents of the -9999 ebuild.

What about non cros_workon packages?

- If an ebuild does not inherit cros_workon than it is not a cros_workon package.
- In this case you do not need to (and cannot) use the cros_workon command with the package.
- If you need to change an ebuild for a non cros_workon package than when submitting the change you will need to **manually up-rev** the ebuild.
    - For example: git mv arbitrary-0.0.1-r3 arbitrary-0.0.1-r4

Why must we up-rev?

Pre-builts

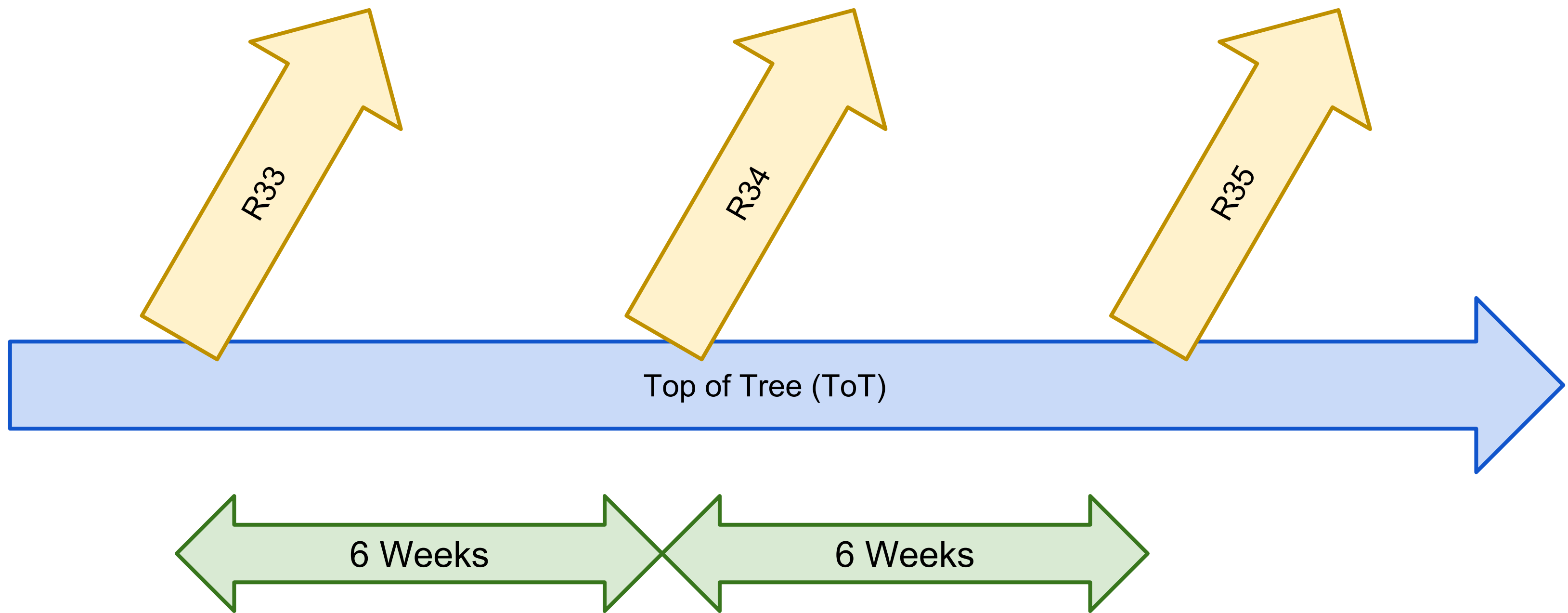
- Builders generated binary versions of packages.
- By default, if your development machine can download a pre-built for the correct architecture and USE flag (build options) combination it will do so.
- If a pre-built isn't available, it will fall back to build from source…. which might also involve downloading the source code. For some packages, like **chromeos-mrc** the source isn't avaialble.
- "cros worked on" (9999) packages are always built locally.
- Access tokens for private pre-builts are kept in the googlestorage_acl.boto file in the board's private overlay.
- The locations of the pre-builts is kept in chromeos-partner-overlay/chromeos/binhost/target

Firmware Branches

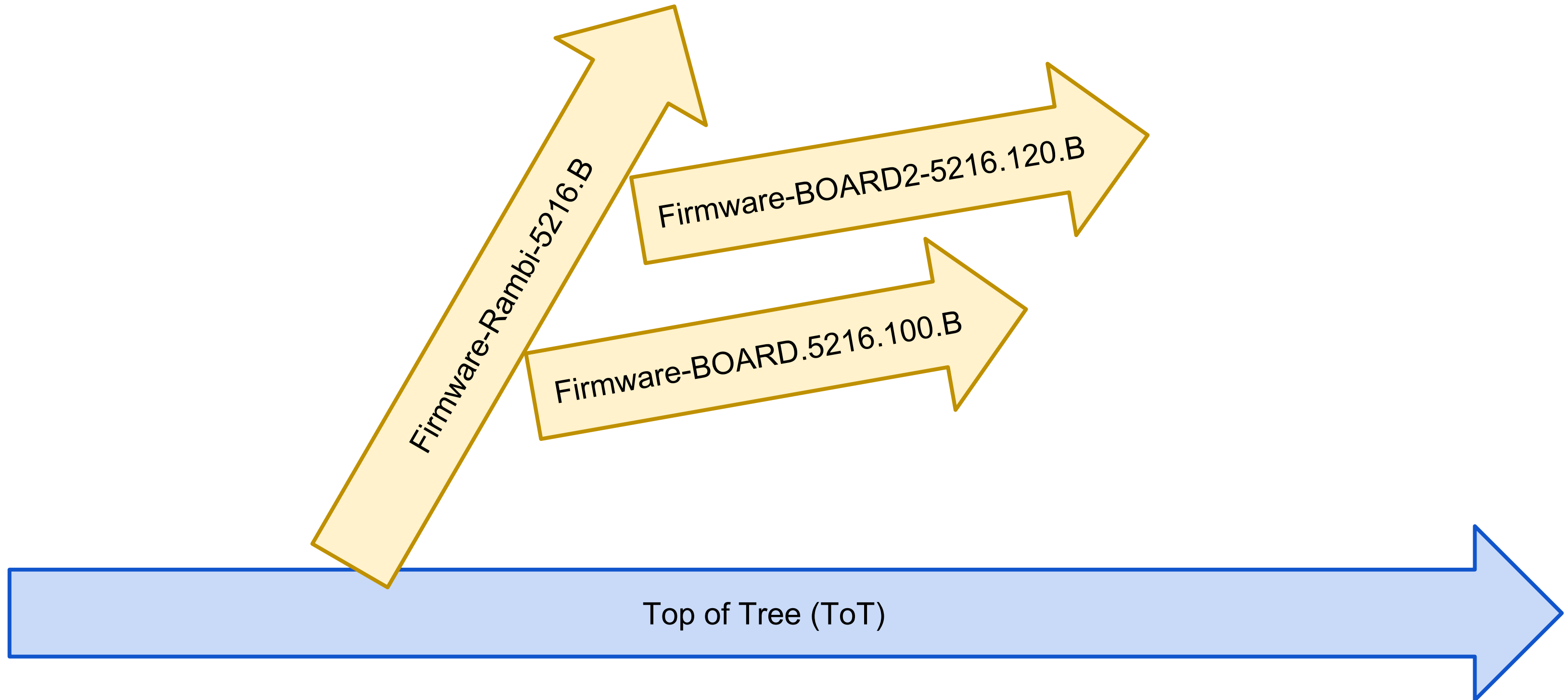
© Exo Terra - PT-3077

Google

Chrome OS Branching

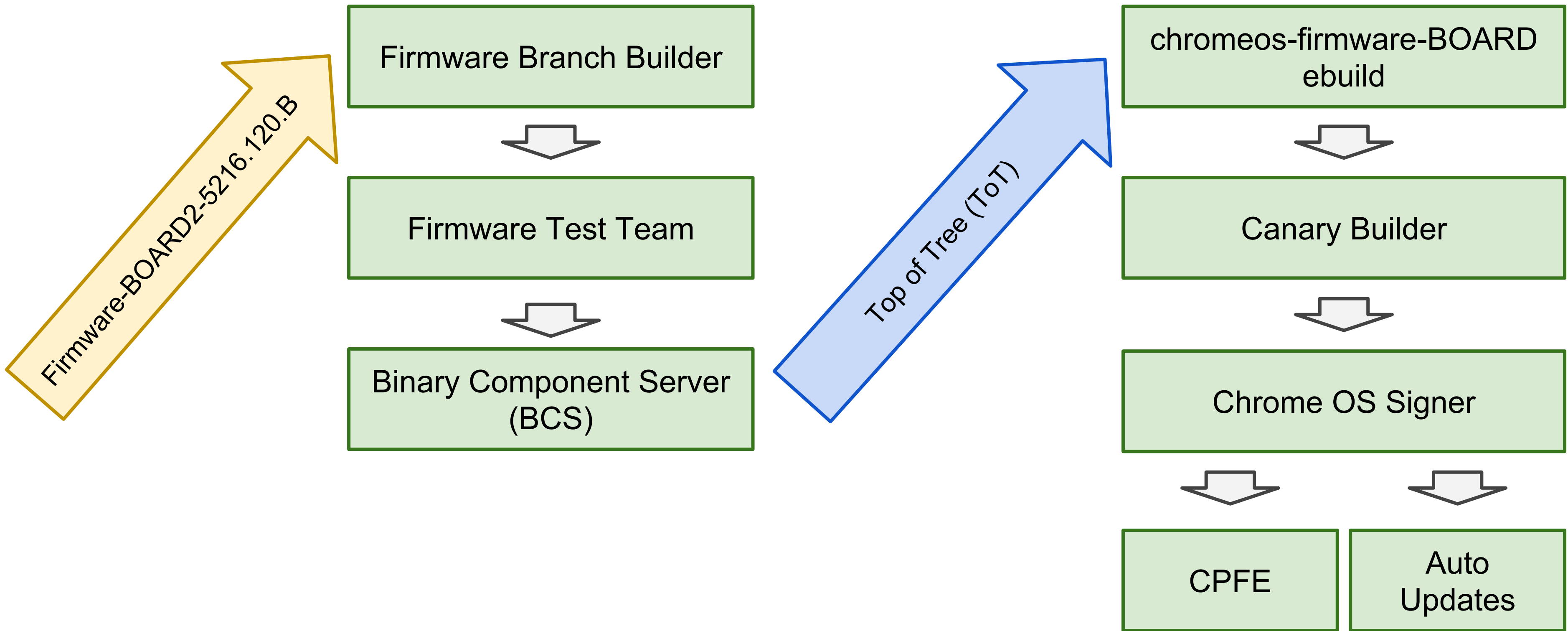


Chrome OS Firmware Branching

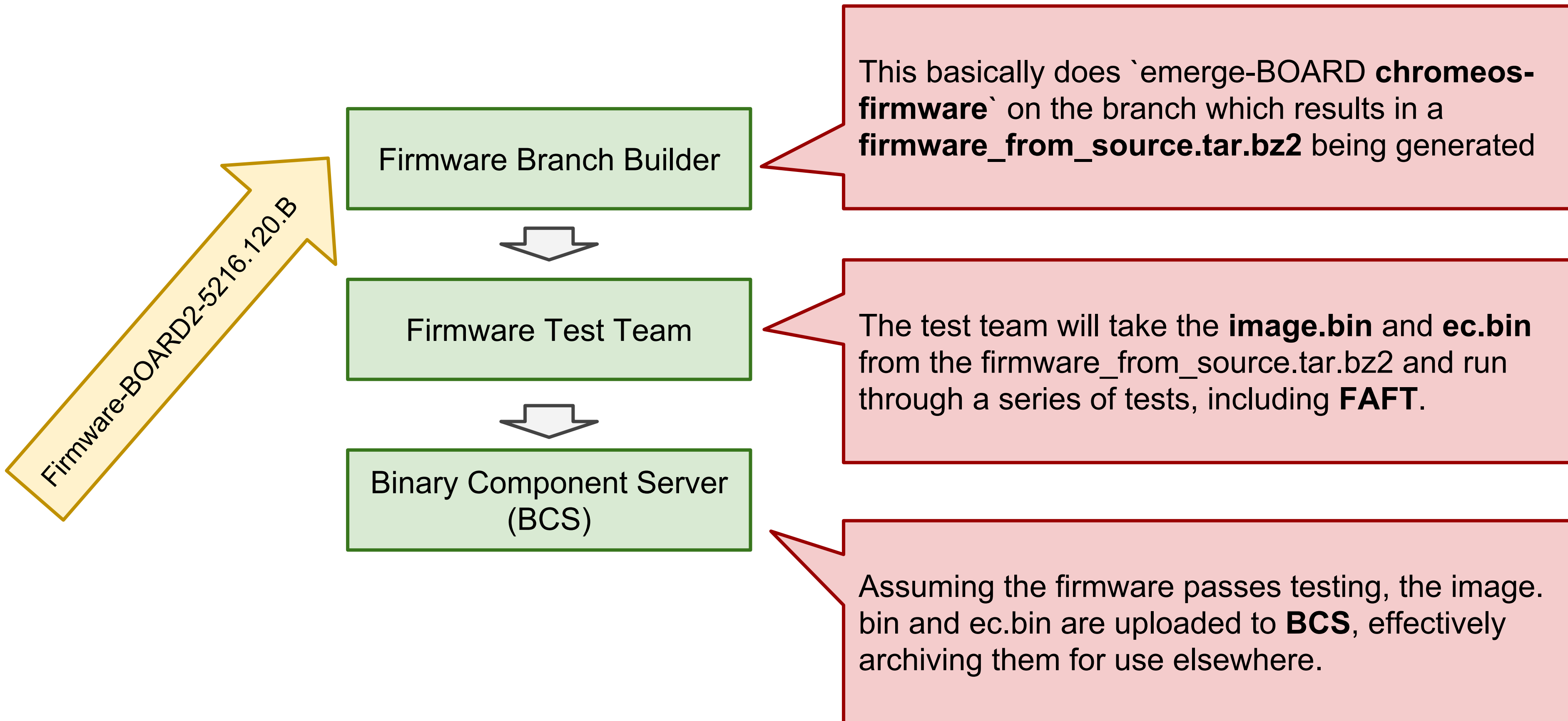


Google

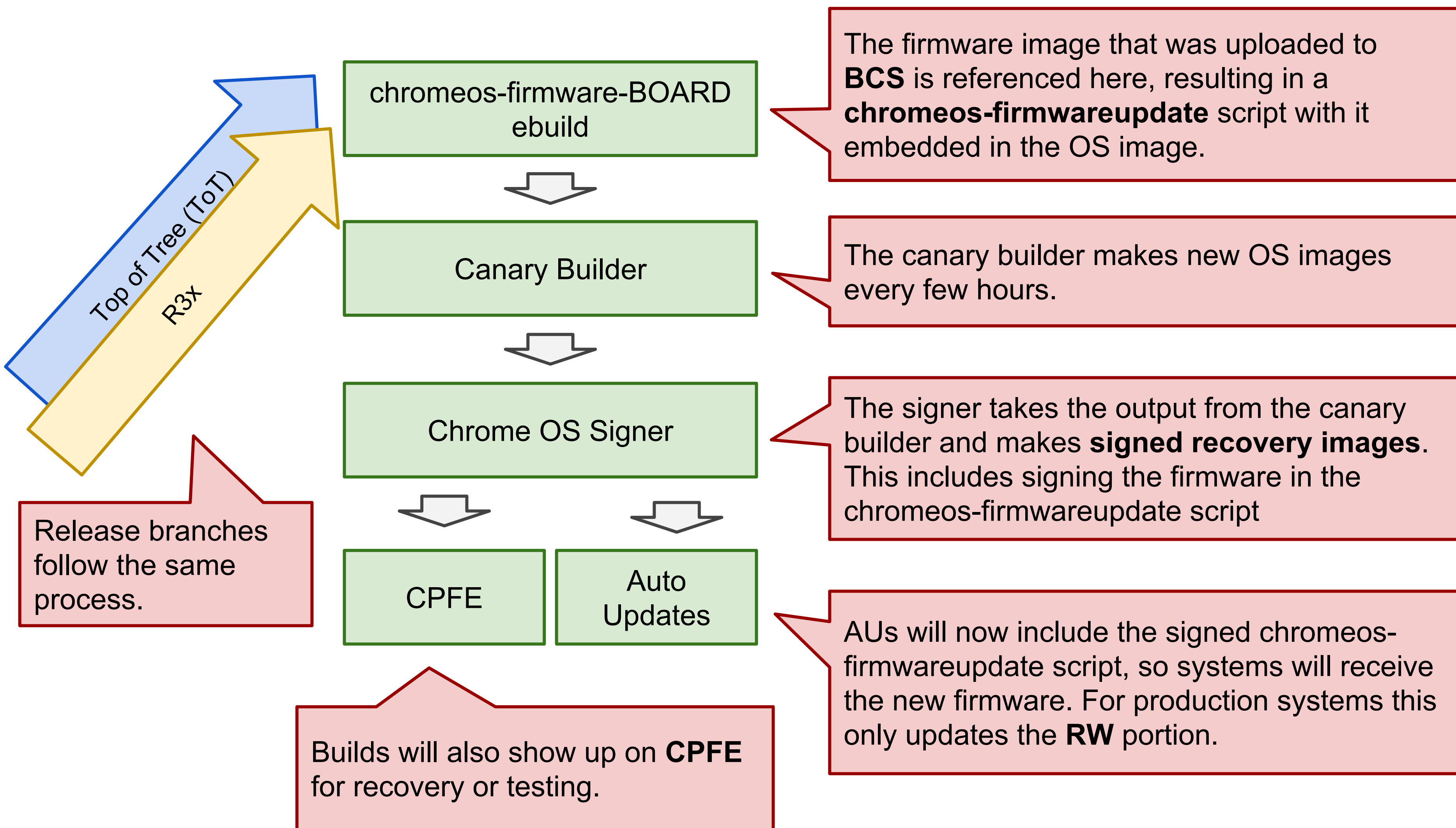
The Life of a Firmware Image



On the Firmware Branch Side...



On the ToT Side...



Keys & signing

What kind of signing keys are there?

- **Dev** - these are the developer keys checked into the public source code. They are used by any locally built image or very early units such as EVT. Boot a base image in recovery? No problem!
- **PreMP** - these are the pre mass production keys which are typically used in the earlier factory runs before mass production. These are required to enable automatic updates, and should be in place by DVT.
- **MP** - these are the mass production keys, these are expected to be the keys you would find on a device you actually buy as a end user. MP keys are generated in a secret underground bunker by security elves.

These keys are all of the same format, just different values.

Google Confidential and Proprietary

Google

How do I get a system to automatically update?

The main step is to recover the device with a Google signed recovery image which you can get from CPFE.

You may also need to set a valid HWID, you can set a HWID using some commands on the DUT like:

cd /tmp
flashrom -p host -r image.bin
gbb_utility --set --hwid="DEVICE ABC-123" image.bin
flashrom -p host -w image.bin
reboot

Note: This requires AUs to be enabled for the platform in question.

Code Reviews

“Cleanest code I’ve ever seen!” - Angela

“Super fast review, Would submit again! A++” - Shiva

Review Process

`repo upload `

Take feedback, Make fixes

`repo upload `

+1 Verified +1 Ready

Author

Provide Feedback

+2 Approved

Reviewer

CQ Time

CHROMIUM

Browse Repositories

olofj@google.com: Tree is open (cloud storage transient(?) error on amd64-generic, crbug.com/344914)

Change-Id:	I45d49123d369b7d075776215e709af5801ea696d
Owner	Bernie Thompson
Project	chromiumos/third_party/coreboot
Branch	firmware-rambi-5216.B
Topic	
Uploaded	Feb 12, 2014 1:20 PM
Updated	Feb 12, 2014 1:29 PM
Status	Merged

Commit Message — Permalink

```
CHERRY-PICK: baytrail: Fix hdmi audio choppy issue

Disable VC setting for HDA so hdmi audio choppy issue will be eliminated.
Change HDA initialize steps to sync with UEFI reference code.

BUG=chrome-os-partner:25651
BRANCH=Baytrail
TEST=Does not have choppy noise during video playing

Signed-off-by: Kein Yuan <kein.yuan@intel.com>

Change-Id: I45d49123d369b7d075776215e709af5801ea696d
Reviewed-on: https://chromium-review.googlesource.com/186146
Reviewed-by: Shawn Nematbakhsh <shawnn@chromium.org>
Commit-Queue: Bernie Thompson <bhthompson@chromium.org>
Tested-by: Bernie Thompson <bhthompson@chromium.org>
```

Reviewer	Code-Review	Commit-Queue	Verified
Bernie Thompson		+1	✓
Benson Leung			
Kein Yuan			
chrome-internal-fetch			
Aaron Durbin			
Shawn Nematbakhsh	✓		

- Need Commit-Queue

Name or Email or Group — Add Reviewer

▶ Included In

▶ Dependencies

Reference Version: Base

▶ Patch Set 1 — f71695556cbb1372b3298e35efb78f7e881988f2 (gitiles)

▶ Patch Set 2 — 177c70778f2a8b589aa60cba6a6219c3d4ae4765 (gitiles)

Home - Waterfall Grid T-Grid Console Builders Recent Builds Buildslaves Changesources - JSON API - About

Tree is open (cloud storage transient(?) error on amd64-generic, crbug.com/344914)

Builds:	status manage all
ChromiumOS:	sources | reviews | bugs | dev
Sheriffs:	pstew, olofj, None (channel is sheriff)
Build Deputy:	merlin (Chrome OS build deputy)
Gardeners:	tbarzic (Chrome)
Navigate:	about | customize | waterfall | failures | console |

closers · x86 · arm · commit queue · chromium.chromiumos

ChromiumOS (x86) · ChromiumOS (amd64) · ChromiumOS (daisy)

		x86-generic paladin	amd64-generic paladin	x86-generic incremental	amd64-generic incremental	daisy incremental	x86-generic full	amd64-generic full	daisy full	arm-generic full
Chromium last build		x86-generic paladin build successful	amd64-generic paladin build successful	build successful	build successful	build successful	build successful	build successful	build successful	build successful
current activity		building ETA in ~ 35 mins at 16:02	building ETA in ~ 37 mins at 16:04	idle	idle	idle	idle	idle	idle	idle
PST	changes	x86-generic paladin	amd64-generic paladin	x86-generic incremental	amd64-generic incremental	daisy incremental	x86-generic full	amd64-generic full	daisy full	arm-generic full
15:12:43		VMTest (attempt 1) stdio	VMTest (attempt 1) stdio							
15:11:26			BuildImage stdio							

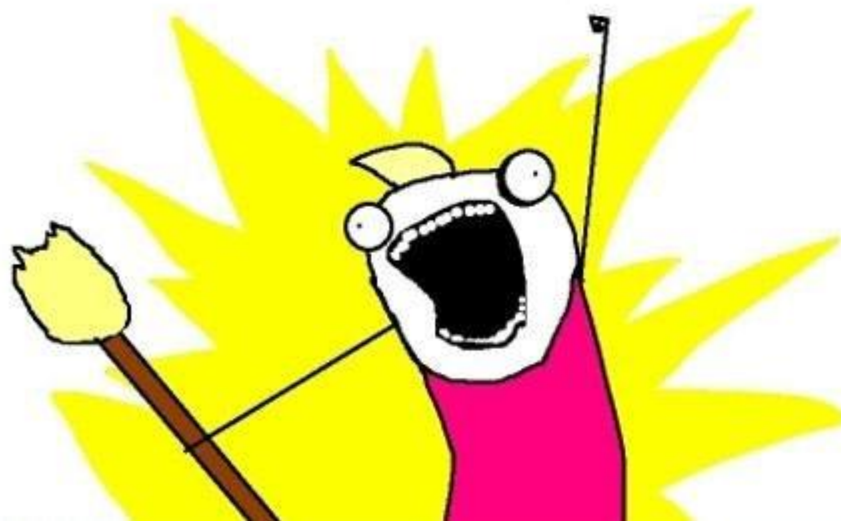
Guidelines for a Speedy Review

- Keep your patches small and on a single topic.
- Separate board-specific patches from global changes.
- Reference bugs so reviewers have more context.
- Add a reviewer! If you're unsure, use "git blame <filename>" and "git log --oneline -- <filename>" to see who has been modifying the same or similar files.

CPFE - Chrome Partner Front End

https://www.google.com/chromeos/partner/fe/

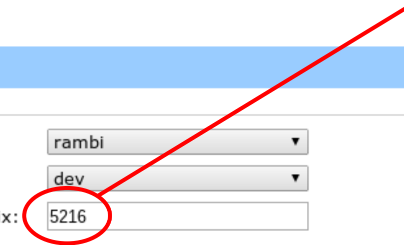
Download **ALL** the things!
(and upload some things)



CPFE - Download Builder Artifacts

Google Chrome OS Partners

Build number of the firmware branch for BayTrail



- **Home**
- **Releases**
  - **Image Files**
  - **Release Note Tool**
- **Binary Components**
  - **Uploads - Private**
- **Device Reports**
  - **Archive Upload**
  - **Report Search**
- **Device File Repository**
  - **Manage Files**
- **Components**
- **Admin**
  - **Configuration**
  - **Partners**
  - **Hwid Info**

Releases › Image Files

Partner: Select All
*Device: rambi
Board: Select All
Image Type: FIRMWARE_IMAGE_ARCHI
Channel: dev
Release/Milestone:
Version/prefix: 5216
Recommended Images Only
Search

Version	Release	Channel	Device	Board	Image Type	Filename	Size (MB)	Date
5216.53.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.53.0-rambi.tar.bz2	3	2014-02-14
5216.52.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.52.0-rambi.tar.bz2	3	2014-02-13
5216.51.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.51.0-rambi.tar.bz2	3	2014-02-13
5216.50.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.50.0-rambi.tar.bz2	3	2014-02-12
5216.46.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.46.0-rambi.tar.bz2	3	2014-02-11
5216.45.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.45.0-rambi.tar.bz2	3	2014-02-10
5216.44.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.44.0-rambi.tar.bz2	3	2014-02-05
5216.43.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.43.0-rambi.tar.bz2	3	2014-02-05
5216.42.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.42.0-rambi.tar.bz2	3	2014-02-05
5216.40.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.40.0-rambi.tar.bz2	3	2014-02-04
5216.39.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.39.0-rambi.tar.bz2	3	2014-02-04
5216.38.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.38.0-rambi.tar.bz2	3	2014-02-04
5216.36.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.36.0-rambi.tar.bz2	3	2014-02-04
5216.35.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.35.0-rambi.tar.bz2	3	2014-02-03
5216.34.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.34.0-rambi.tar.bz2	3	2014-02-03
5216.33.0	R34	dev	rambi	rambi	FIRMWARE_IMAGE_ARCHIVE	ChromeOS-firmware-R34-5216.33.0-rambi.tar.bz2	3	2014-02-03

Google

CPFE contains a variety of firmware...

- **Untested firmware**:
  - Set Device to the device in question.
  - Set Image Type to FIRMWARE_IMAGE_ARCHIVE.
  - Set Version/Prefix to your firmware branch number (5xxx), and search.
- **Tested and dev signed firmware**:
  - Set Device to device in question and search.
  - Download the latest TEST_IMAGE_ARCHIVE.
  - Image a USB stick with this, then boot it on a DUT and run `` `chromeos-firmwareupdate --mode=incompatible_update` ``.
- **Tested and Google signed firmware**:
  - Set Device to device in question and search.
  - Download the latest **recovery image**.
  - Image a USB stick with this, then boot it on a DUT.
    - If your firmware has matching keys, than recovery (esc+refresh+power) will work, if not than you will need to CTRL+U boot it in dev mode.
  - If you just let your system AU, you will get this automatically.

CPFE Uploads

New files for the Binary Component Server (BCS) can be uploaded through CPFE.

Home
Releases
Image Files
Release Note Tool
Binary Components
Uploads - Private
Device Reports
Archive Upload
Report Search
Device File Repository
Manage Files
Components
Admin
Configuration
Partners
Hwid Info

Binary Components › Uploads - Private

You may upload binary component packages to share with other Chrome OS and Chromium OS developers subject to the acceptance of a specified End-User License Agreement. Uploaded files are placed on the Binary Components server where Chromium OS developers may access them. You must submit an ebuild file in parallel that references the URI for your binary component on the server.

Uploaders Terms of Service

You accepted the below terms of service on Thursday, 2012 February 09 10:58:54 UTC-8
Chrome OS Binary Component Vendors Terms of Service

Upload New File

The maximum file upload size is 1GB. Once a file is uploaded it may take up to 5 minutes before it is available for download.

For files uploaded to a private overlay, all users with access to the source code in that private overlay will also have access to the file. For files uploaded to a public overlay, any user accepting the Developer Terms of Service on the Access tab will be able to access the file. Contact your Google technical account manager if you have any questions.

Select Component File: Choose File No file chosen
Select Partner: Select a Partner
Select Overlay: Select an Overlay
Relative path to file:

Affects full download path (/<overlay>/<relative path>/<filename>) from Google server.
Example relative path: chromeos-base/gobi-firmware.

Upload

Uploaded Files

Git

Git Basics

Git is an open source **version control system** created to wrangle the Linux Kernel

Create lots of local branches with “git checkout -b foo” or “repo start foo .” Don’t be afraid!

Learn git in your browser (which is Chrome, right?): http://try.github.io/

Pro Git book (Creative Commons): http://git-scm.com/book/

Google

An Example: Making a change to the EC code

1. cros_sdk
2. cd ~/trunk/src/platform/ec
3. repo sync .
4. repo start mychange .
5. vim board/rambi/board.c
6. make buildall -j
7. cros_workon-rambi start chromeos-ec
8. emerge-rambi chromeos-ec
9. util/flash_ec --board=rambi
10. git add board/rambi/board.c
11. git commit
12. repo upload .
13. ...time passes…got a -1 on the review. So sad :((Need to fix a type-o)
14. vim board/rambi/board.c
15. git add board/rambi/board.c
16. git commit --amend
17. repo upload .
18. repo abandon mychange .
19. cros_workon-rambi stop chromeos-ec

Google

Amending an earlier local commit

If you need to edit a change that has locally committed changes after it, you can go back and edit the change with interactive rebasing.

1. git log --oneline. Find the commit hash of the change you want to edit. Let's say it's a717a50.
2. git rebase -i a717a50^
3. In the text editor window, change "pick" to "edit" for the change you want to modify. Save the file.
4. Make changes.
5. Test changes.
6. "git add" the changed files.
7. git commit --amend
8. git rebase --continue
9. repo upload .

Diffing Branches & Cherry Picking

```
git --no-pager log --left-right --graph --cherry-pick --oneline \
cros/master...HEAD
```

```
> a717a50 Enable USB power in S3 if USB ports enabled at suspend
> 55c3e8e Added unit tests for lid angle calculation and acos
> 419e6d9 Squawks: Add smart battery temp sensor to temp sensors list
> 3ee48b7 Clapper: Update accelerometer translation matrices
...
< d778fab Fix some stupid.
< 33e967e Update util/lbplay.c to use the sysfs interface.
< e43074e cleanup: nyan: remove unnecessary dependence to pmu_tpschrome.h
< e7e0cf2 Optimize memmove
```

```
git cherry-pick d778fab
git commit --amend (change “Change-Id” to “Original-Change-Id”)
repo upload .
```

How the Sausage is Made

homefoodsafety.org

How the Sausage is Made

Firmware builders build **virtual/chromeos-firmware** whenever a changed is checked in to a firmware branch.

This virtual points to a board-specific package like **chromeos-base/chromeos-firmware-rambi**

This ebuild ultimately generates the **chromeos-firmwareupdate** “shell ball” containing the system (bios.bin) and EC (ec.bin) firmware along with supporting utilities (flashrom, mosys, vpd, dump_fmap). The firmware updater is assembled by **src/platform/firmware/pack_firmware.sh**

The firmware builders also gather some of the artifacts deposited under **/build/$BOARD/firmware/**, creates firmware_from_source.tar.bz2, and uploads it to the CPFE.

How the Sausage is Made

So how does the “bios.bin” get created so it can be assembled to go into the updater?

chromeos-firmware-rambi inherits from the **cros-firmware.eclass** in the chromiumos-overlay “eclass” directory.

USE flags steer the build process followed by the cros-firmware eclass to build the right blobs to package into image.bin (aka bios.bin) using **src/platform/dev/host/cros_bundle_firmware**

From: src/overlays/overlay-rambi/make.conf
USE="${USE} cros_ec”

From: src/private-overlays/overlay-rambi-private/make.conf
USE="${USE} bootimage coreboot depthcharge unified_depthcharge"

How the Sausage is Made

bootimage

Add a dependency on sys-boot/chromeos-bootimage which generates /firmware/image.bin from source.

coreboot

Depends on **virtual/chromeos-coreboot** which points to sys-boot/chromeos-coreboot-rambi which generates /firmware/coreboot.rom

depthcharge

Depend on **sys-boot/depthcharge**

cros_ec

Build **chromeos-base/chromeos-ec** resulting in /firmware/ec.bin

How the Sausage is Made

Looking more at **virtual/chromeos-coreboot** ...

This is satisfied by **sys-boot/chromeos-coreboot-rambi** which depends on chromeos-base/vboot_reference (verified boot).

chromeos-coreboot-rambi lives in the board's private overlay. It's "files" subdirectory contains:

descriptor.bin fitc.xml me.bin vgabios.bin vlv_1004.ssf

chromeos-coreboot-rambi inherits from the cros-coreboot eclass which pulls in **sys-boot/chromeos-mrc** and sys-apps/coreboot-utils

The chromeos-mrc ebuild is in the chromeos-partner-overlay. The source code isn't available to partners but the binary blob can be downloaded with: `emerge-$BOARD --getbinpkg chromeos-mrc`

In summary…

$ equery-rambi g --depth=4 chromeos-firmware

```
* dependency graph for virtual/chromeos-firmware-3
...
`--  virtual/chromeos-firmware-3
  `--  chromeos-base/chromeos-firmware-rambi-0.0.1-r1
   `--  chromeos-base/vboot_reference-1.0-r913
    `--  app-crypt/trousers-0.3.3-r35
    `--  dev-libs/libyaml-0.1.4
   `--  chromeos-base/vpd-0.0.1-r65
    `--  sys-apps/flashrom-0.9.4-r311
     `--  sys-apps/pciutils-3.1.10
     `--  sys-apps/dtc-1.4.0
     `--  dev-embedded/libftdi-1.0-r3
     `--  sys-apps/dmidecode-2.11-r1
    `--  dev-util/shflags-1.0.3-r1
    `--  virtual/chromeos-activate-date-1
     `--  chromeos-base/chromeos-activate-date-0.0.1-r1
   `--  sys-apps/mosys-1.2.03-r189
    `--  sys-apps/flashmap-0.3-r15
   `--  sys-boot/chromeos-bootimage-0.0.2-r84
    `--  virtual/chromeos-coreboot-3
     `--  sys-boot/chromeos-coreboot-rambi-0.0.1-r223
    `--  sys-apps/coreboot-utils-0.0.1-r988
    `--  sys-boot/chromeos-seabios-0.0.1-r51
    `--  virtual/u-boot-1
     `--  sys-boot/chromeos-u-boot-2013.04-r1644
    `--  chromeos-base/chromeos-ec-9999
    `--  sys-boot/chromeos-bmpblk-0.0.3-r2
    `--  sys-boot/chromeos-memtest-0.0.1-r5
    `--  sys-boot/depthcharge-0.0.1-r662
     `--  sys-boot/libpayload-0.0.1-r1000
```

It's complicated :-)

Start with a graph then use "equery w" and "equery f" to find ebuilds and see what files they install.

Increase the graph depth until you've had enough.

I'm building a local "cros_workon" EC firmware

So why do I care?

Because you'll forget to either "cros_workon" something or emerge something and you'll end up testing firmware you didn't intend to build.

Working on coreboot and want a new image.bin?

```
cros_workon-$BOARD start chromeos-coreboot-$BOARD
emerge-$BOARD chromeos-coreboot-$BOARD chromeos-bootimage
```

Changed something in src/platform/vboot_reference with headers that are linked from all over the place?

```
cros_workon-$BOARD start vboot_reference \
    chromeos-coreboot-$BOARD depthcharge
emerge-$BOARD vboot_reference chromeos-coreboot-$BOARD \
    depthcharge chromeos-bootimage
```

How the sausage is made on a release branch

Starts off the same way with **virtual/chromeos-firmware** which points to **chromeos-firmware-rambi**

However, chromeos-firmware-rambi on the release branch includes:

CROS_FIRMWARE_MAIN_IMAGE="bcs://Rambi.5216.50.0.tbz2"

CROS_FIRMWARE_EC_IMAGE="bcs://Rambi_EC.5216.50.0.tbz2"

Instead of building image.bin and ec.bin, the pre-built images are fetched from the Binary Component Server and are used to build the chromeos-firmwareupdate “shell ball.”

FAFT (Fully Automated Firmware Test)

FAFT is a framework to write autotests that test firmware. It consists of two components:

- A library of helper functions to perform firmware related tasks
- A defined structure/template for autotests

It is used in conjunction with the servo debug board connected to devices under test.

Running FAFT tests involve:

- Starting servod server, which provides control over the servo debug board
  - sudo servod --board=<BOARD>
- Running tests via the test_that script
  - test_that --board=<BOARD> <DUT IP> suite:faft_bios
  - test_that --board=<BOARD> <DUT IP> suite:faft_ec

All Done!

BACKUP

We are probably done...

Google Confidential and Proprietary

Useful portage equery commands

Show the ebuild for the package

equery-$BOARD w <package-name>

Show files installed by package

equery-$BOARD f <package-name>

Show the package to which a file belongs

equery-$BOARD b <filename without /build/$BOARD>

Show the dependency graph for a package

equery-rambi g <package-name>

Google

Moblab

- Moblab is an autotest server running on Chromebox.
-
- The moblab official image release will be managed and AUed by Google.
-
- The partners also can build their own moblab image from public overlay(overlay-variant-stumpy-moblab).
- The initial launch image will support Samsung Chromebox(Stumpy) only as a moblab host server, but we plan to add more Chromeboxes gradually.
-
- The initial launch will support only limited test set since it is more focusing on the moblab, but we plan to add FAFT for firmware testing in the later releases.

Depot Tools

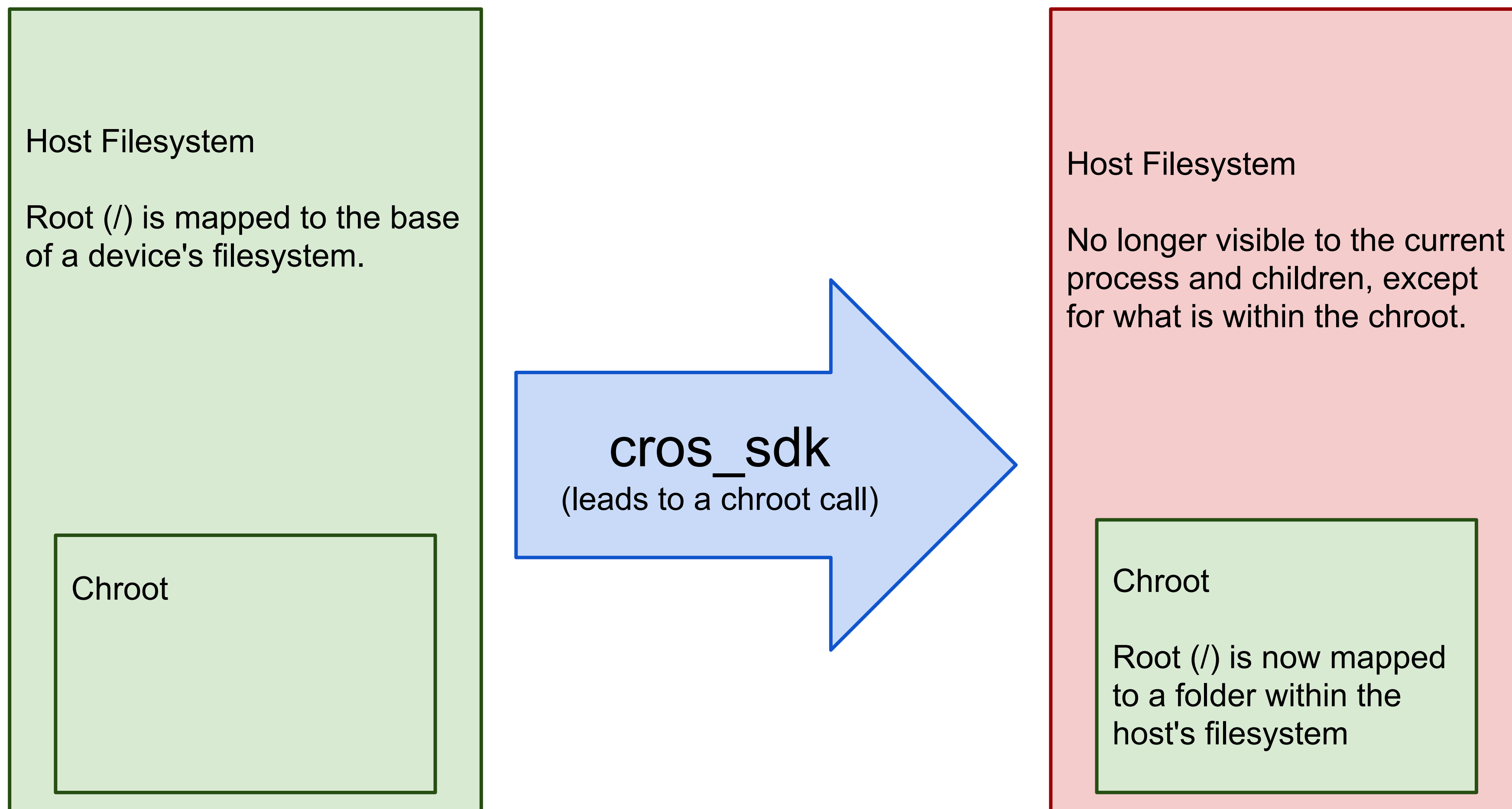
- The Depot Tools are a set of scripts used to manage interaction with the source repositories.
- Depot Tools can be installed using git via http://www.chromium.org/developers/how-tos/install-depot-tools
- The tool used most often with Chrome OS development is **repo**.
- Make sure you add it to your $PATH.

Chroot

- **chroot** is a Linux command which changes the apparent root directory (/) for the current process and children.
  - chroot is short for **change root**.

- In Chrome OS parlance 'the chroot' is the development environment one enters with the **cros_sdk** command which ultimately leads to a chroot call.
  - Effectively it is a folder which contains the **various tools and sources** needed to build Chrome OS.
  - The chroot is used to make the development environment somewhat **system agnostic** and to **protect the host**.

Chroot



Additional Resources

- Chromium OS Developer's Guide -- http://www.chromium.org/chromium-os/developer-guide
    - This is the quintessential guide to getting started with Chrome OS development.

- Gentoo Portage Documentation -- http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?part=2&chap=1
    - Much of what is described in the Gentoo documentation can be helpful in understanding the way Portage works even with Chrome OS.

Gerrit Searching

- **Clicking on a name or category** will run a search.
- Searching for a **user's email address** will find their CLs.
- Searching for **project:project/name** will find CLs for that particular project.
- By default you will see **status:open** which searches for open items (not yet merged or abandoned).
  - **status:merged** returns merged CLs.
  - **status:abandoned** returns abandoned CLs.
- Searching for a **commit ID** will return the CL it is from.
- Some additional search parameters can be found at https://review.typo3.org/Documentation/user-search.html

What is in a Chrome OS Overlay?

- A Chrome OS board has a primary overlay that at a minimum contains:
  - profiles/**repo_name** -- Unique name
  - **make.conf** -- Defines build info
  - **toolchain.conf** -- Defines the toolchain
- Typically there is also metadata**/layout.conf**
  - Defines the masters list for repositories which allows for disambiguation
- There can also be custom ebuilds.

```
bhthompson@ragnarok:
~/chromiumos/src/overlays/overlay-
arm-generic$ tree
.
|-- autotest-blacklist
|-- make.conf
|-- prebuilt.conf
|-- profiles
|   |-- kernel-next
|   |   |-- package.use
|   |   |-- parent
|   |   `-- virtuals
|   `-- repo_name
`-- toolchain.conf
```

$ willis

Willis tells you “what’s up” in your chroot.

Shows you all the local branches and untracked files across all the repositories.

Which overlay is used?

- There are multiple potential overlay locations, depending on if the overlay contents are public, private, and/or a variant of a base board.
- The locations in order of inclusion are:
  - src/overlays/overlay-<board>
  - src/overlays/overlay-variant-<board>-<variant>
  - src/private-overlays/overlay-<board>-private
  - src/private-overlays/overlay-variant-<board>-<variant>-private