



# Rebuilding the airplane at 10 000m

Continuous Deployment with Jenkins and Gerrit

R. Tyler Croy  
[tyler@linux.com](mailto:tyler@linux.com)

Hello and thanks for coming. I'm R. Tyler Croy, and today I'm going to talk to you about two things at the same time!

I'm going to tell you how we rebuilt our engineering organization "mid-flight" for Continuous Deployment and at the same time, I'm going to tell you how you can too with Jenkins and Gerrit



I work for Lookout



First of all, I work at Lookout Mobile Security, if you're an Android user you might already be familiar with some of our products.

If you're not familiar with us, we are primarily known for our security app on Android.

My job at Lookout primarily involves ->



```
ast.rb ~/.viarc
self.fail(Puppet::ParseError, message)
end
```

Wrap a state in a reusable way so we always throw a parse error.

```
def parsewrap
  exceptwrap :type => Puppet::ParseError do
    yield
  end
end
```

The version of the evaluate method that should be called, because it correctly handles errors. It is critical to use this method because it can enable you to catch the error where it happens, rather than much higher up the stack.


```
def safeevaluate(*options)
  # We duplicate code here because the evaluate method is called so many times
  begin
    return self.evaluate(*options)
  rescue Puppet::Error => detail
    raise adderrorcontext(detail)
  rescue => detail
    error = Puppet::ParseError.new(detail.to_s, nil, nil, detail)
    # We can't use self.fail here because it always expects strings,
    # not exceptions.
    raise adderrorcontext(error, detail)
  end
end
```

Initialize the object. Requires a hash as the argument, and takes each of the parameters of the hash and calls the setter method for them. This is probably pretty inefficient and should

# Hacking with Ruby

Hacking with Ruby, you see while we have a nice fancy Java-based Android application, we also have a *\*large\** server-backend which handles device notifications, backups, analysis and much more.

That entire backend is written in Ruby, and can benefit from ->

The background of the slide is a composite image. The upper portion shows the tail of a white airplane with a red stripe and a 'TA' logo on the vertical stabilizer, set against a clear blue sky. The lower portion shows a close-up of a large, white rocket engine with red and blue decorative stripes around its base, also against a blue sky. A dark, semi-transparent rectangular box is centered over the image, containing the text.

Let's talk about:  
**continuous deployment**

Continuous Deployment.

Before I talk too much about what it \*is\*, I'd like to talk about ->




Project \*  ServerIssue Type \*  Story ?Summary \* We should install Continuous Deployment  
Use "should" or "shouldn't". Example: "Logo on website homepage should be green, not blue"

Priority P1 ?

Urgency from Engineering's perspective:

P1 = Needed ASAP; won't release again without this change. P2 = Very important, but may not delay a release for it; P3 = Like to have, but won't delay a release for it. Do only if have the time and doesn't risk anything.

Component/s  x

Start typing to get a list of possible matches. Press down to select.

Fix Version/s  xStart typing to get a list of possible matches. Press down to select.  
Identifies the first Release this was fixed/resolved in.Due Date 29/Jul/12 

Optional customer-committed date (if no Fix Version specified).


Assignee  R. Tyler Croy [Assign To Me](#)

Description All the other people are using it and they seem to like it!

What it is:  
NOT

What Continuous Deployment is NOT.

Above all else, it is not something you do \*once\* and then you're finished and you can move on. Continuous Deployment is a process and mind-set you and your team stick with

A photograph of a messy room. In the foreground, a wooden table is cluttered with various items: a yellow container, a blue and white striped bag, a glass jar, and some papers. In the background, a potted plant sits on a dark surface, and a wooden chair is visible. A large, semi-transparent grey box with white text is overlaid on the center of the image.

"Release *everything*  
as soon as possible!"

Continuous Deployment doesn't mean you release EVERYTHING as soon as it's committed. Nor does it mean you must deploy every single commit.

(photo by thomen: <<http://www.flickr.com/photos/thomen/364890522/>>)

A photograph of a messy room. A light-colored wooden table is tipped over on its side. On the floor next to it is a potted plant with green leaves. Various items are scattered on the floor, including a yellow container, a blue and white striped bag, a glass jar, and some papers. A dark-colored sofa is visible in the background.

"Great! No need for a QA team"

One of the interesting things I've discovered at Lookout and at other organizations, is that Continuous Deployment, and some of the practices involved in it is that it will free up the QA team to \*do their jobs\*.

Good QA engineers are most useful when they're exploring, hunting for bugs. Having QA engineers running through test plans every day of the week is boring, slow and is a good candidate for replacement with automated testing tools



A photograph of a messy room. A light-colored wooden table is tipped over on its side. On the floor next to it is a potted plant with green leaves. Various items are scattered on the floor, including a yellow container, a blue and white striped bag, a glass jar, and some papers. A dark-colored sofa is visible in the background.

"Our users will be our testers!"

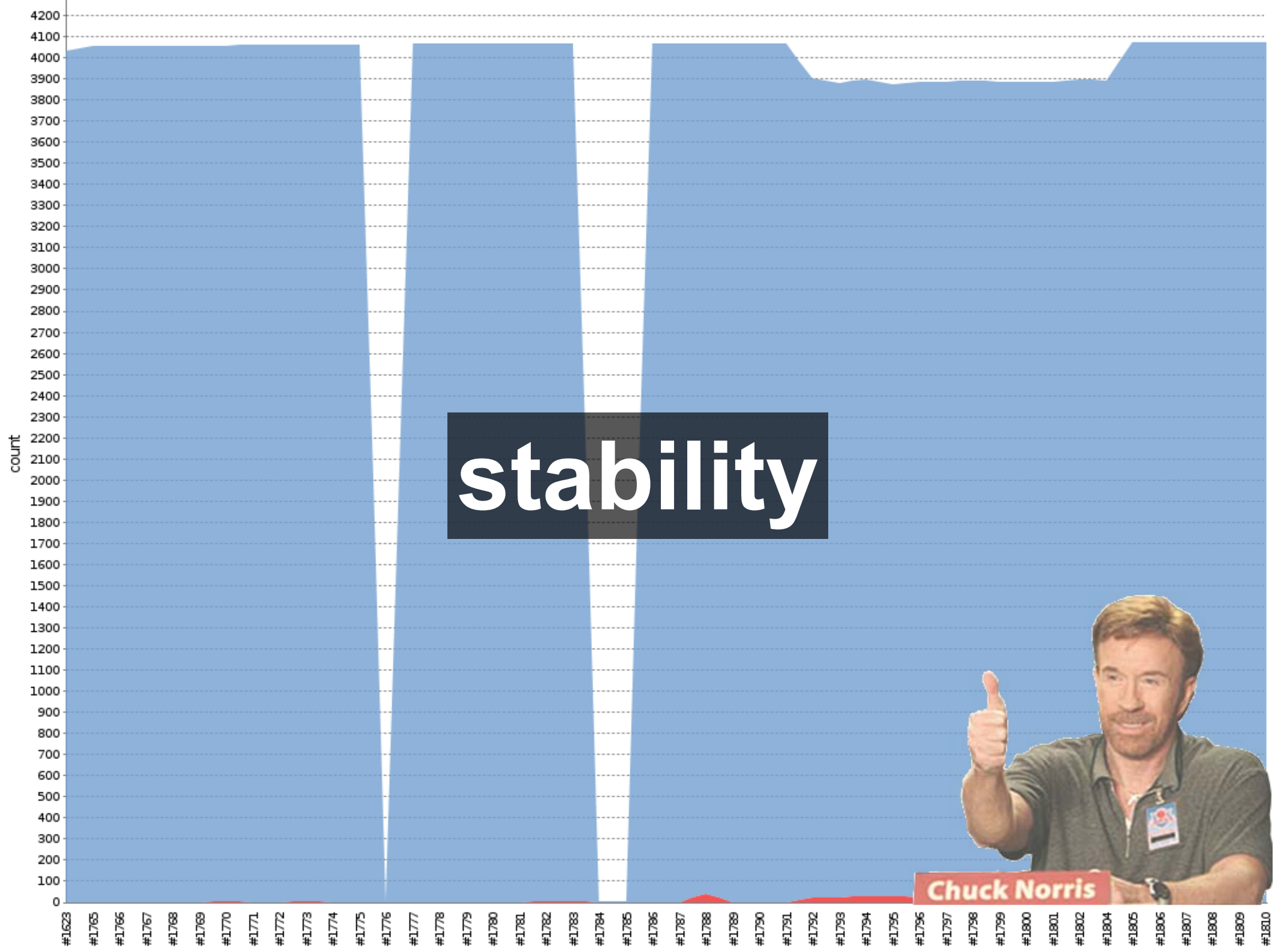
Lastly, I don't think Continuous Deployment means you can offload testing to your users.

To some extent this will be inevitable, as changes are more rapidly deployed, but I believe you should try everything you can to avoid your users experiencing issues because of bugs you've introduced

A large commercial airplane is shown from a low-angle perspective, flying towards the viewer. The aircraft is white with red and blue stripes along the fuselage and tail. The tail features a logo with the letters 'TA' inside a red square. The plane is set against a clear blue sky. A dark, semi-transparent banner is overlaid across the middle of the image, containing the text 'Continuous Deployment is about' in white, sans-serif font.

Continuous Deployment  
is about

Continuous Deployment, in my opinion, is all about ->



stability. It is about being able to deploy changes ->





**Faster  
with  
More Confidence**

Continuous Deployment is about releasing changes faster, and with more confidence in the changes you're preparing the release.

In order to make that happen, it is important to have \*good\* procedures for rapid deployment and an excellent feedback loop from production. These two factors, above anything else will enable you to ship code rapidly and maintain quality.

A large commercial airplane is shown from a low-angle perspective, flying towards the viewer. The aircraft is white with red and blue stripes along the fuselage and tail. The tail features a logo with the letters 'TA' inside a red circle. The plane is set against a clear blue sky. A dark, semi-transparent rectangular box is centered over the middle of the image, containing the text 'Continuous Deployment is GOOD'.

Continuous Deployment is  
**GOOD**

Continuous Deployment is GOOD for your organization. Even if you don't end up rapidly deploying your software, the practice of \*striving\* for continuous deployment will help improve so many other parts of your development process

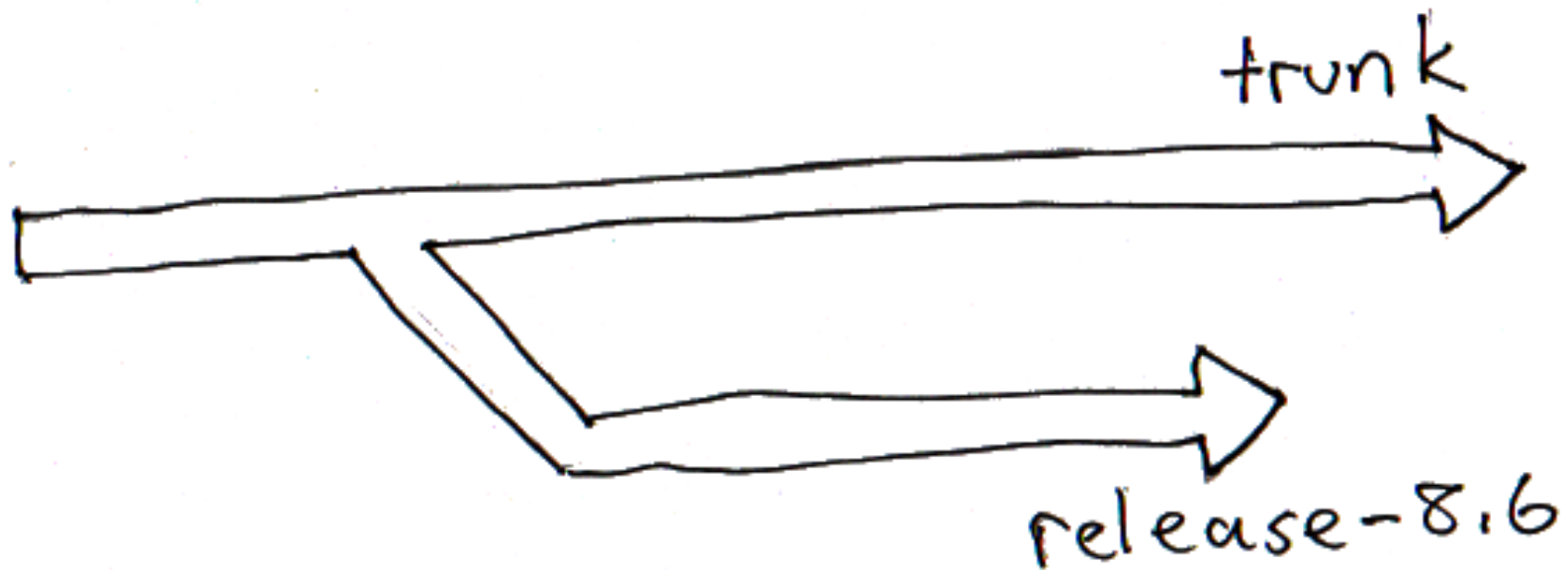


# Meanwhile at Lookout

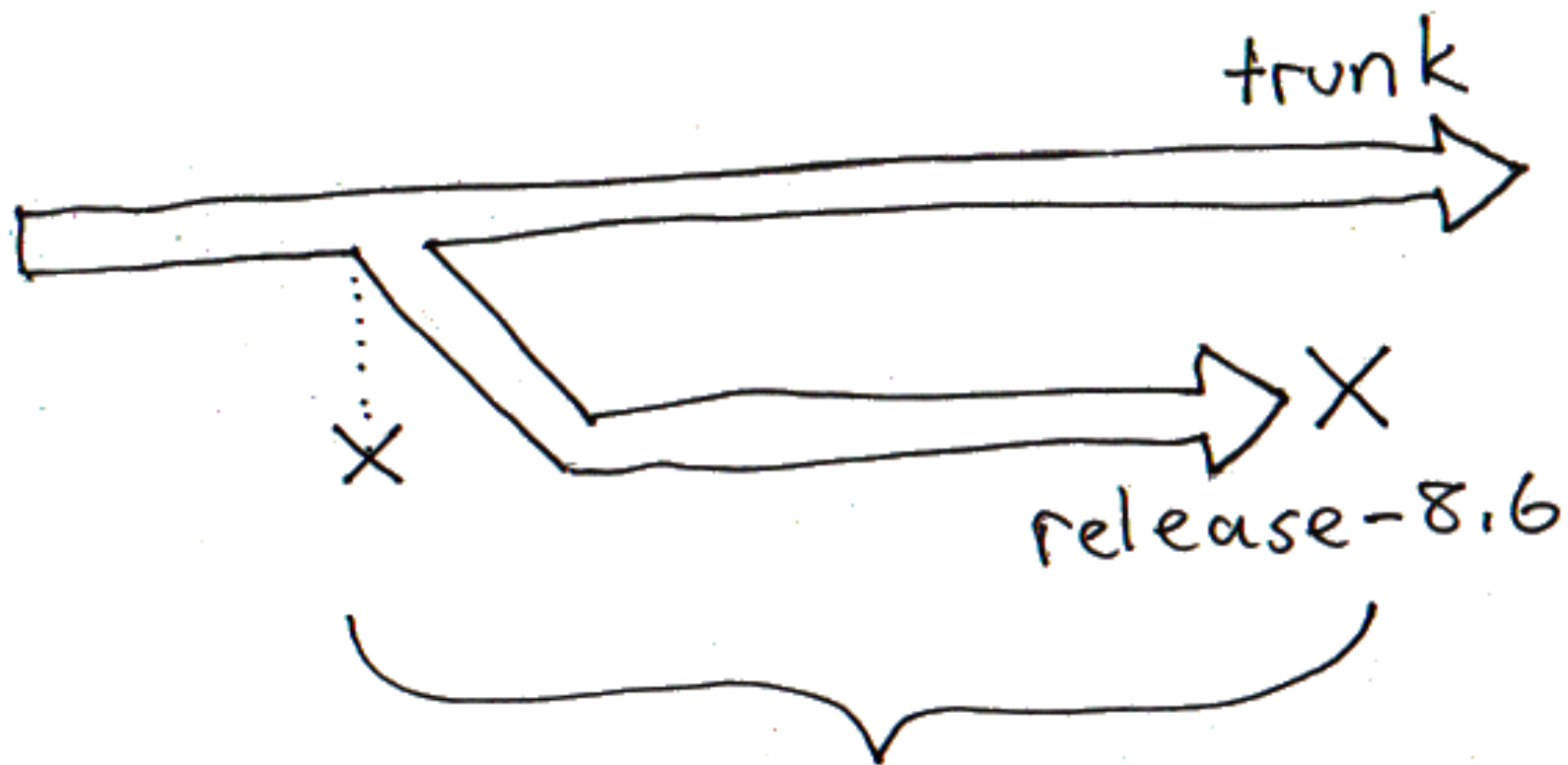




# Subversion branches for releases







10-18 days per release branch



manual code review

TODO: Fill out a good description of code review pre-git+gerrit



very little automation



# Sadness with Numbers



**36%**  
of deployments failed





# 68

commits per deployment



**62%**  
of deployments slipped

A large white passenger jet is shown flying upside down against a light blue sky. The aircraft has a red and white striped tail fin with a 'TA' logo. The fuselage is white with red and blue stripes along the windows. The wings are spread, and the engines are visible. The text 'Let's fix this.' is overlaid in a black box with white text.

**Let's fix this.**



A photograph of an industrial manufacturing environment, likely a car factory. Several yellow robotic arms are visible, working on car chassis. The scene is filled with metal structures, pipes, and safety railings. The lighting is bright, typical of a large industrial hall. The text "Step One: Automate" is overlaid in the center in a large, white, sans-serif font.

# Step One: Automate



A large industrial factory setting with multiple yellow robotic arms (likely KUKA) working on assembling car chassis. The robots are positioned around a car body, which is visible in the foreground. The background shows a complex network of pipes, structural beams, and other industrial equipment. The word "Jenkins" is overlaid in the center of the image.

# Jenkins



Before we used a tool called Bitten, I won't tell you too much about Bitten, but it's not a great tool and we had a number of issues with it:

- \* Practically zero developer insight into the test/build process
- \* All the tests ran on \*one\* build machine which was hand-crafted by the Operations team for the task

We installed Jenkins and started to work on migration "jobs" over to Jenkins.





"Why don't our tests pass?"



The first major issue we had was that we noticed that we had tests that didn't actually \*pass\* reliably. Previously this was hidden from us, but after running the tests after every commit with Jenkins, we noticed that we had some technical debt in the test suite



A large yellow industrial robotic arm is the central focus, positioned over a car chassis. The arm has a red emergency stop button and various cables. In the background, another similar robotic arm is visible, working on another car. The factory floor is filled with industrial equipment, pipes, and structural beams. The text "Never stop automating." is overlaid in white on a dark horizontal band across the middle of the image.

Never stop automating.

TODO: Need to dive into how automation is an on-going commitment





Step Two  
Better tools,  
processes

The second important step we took towards continuous deployment was to use  
\*better\* tools and processes than we were using





*(I don't like SVN)*

I'm not going to rant against Subversion here, if you like it, that's fine.  
There are ways to accomplish continuous deployment with Subversion. At  
Lookout however, I viewed it as one of the things standing in our way.

So we got rid of Subversion and instead opted for ->

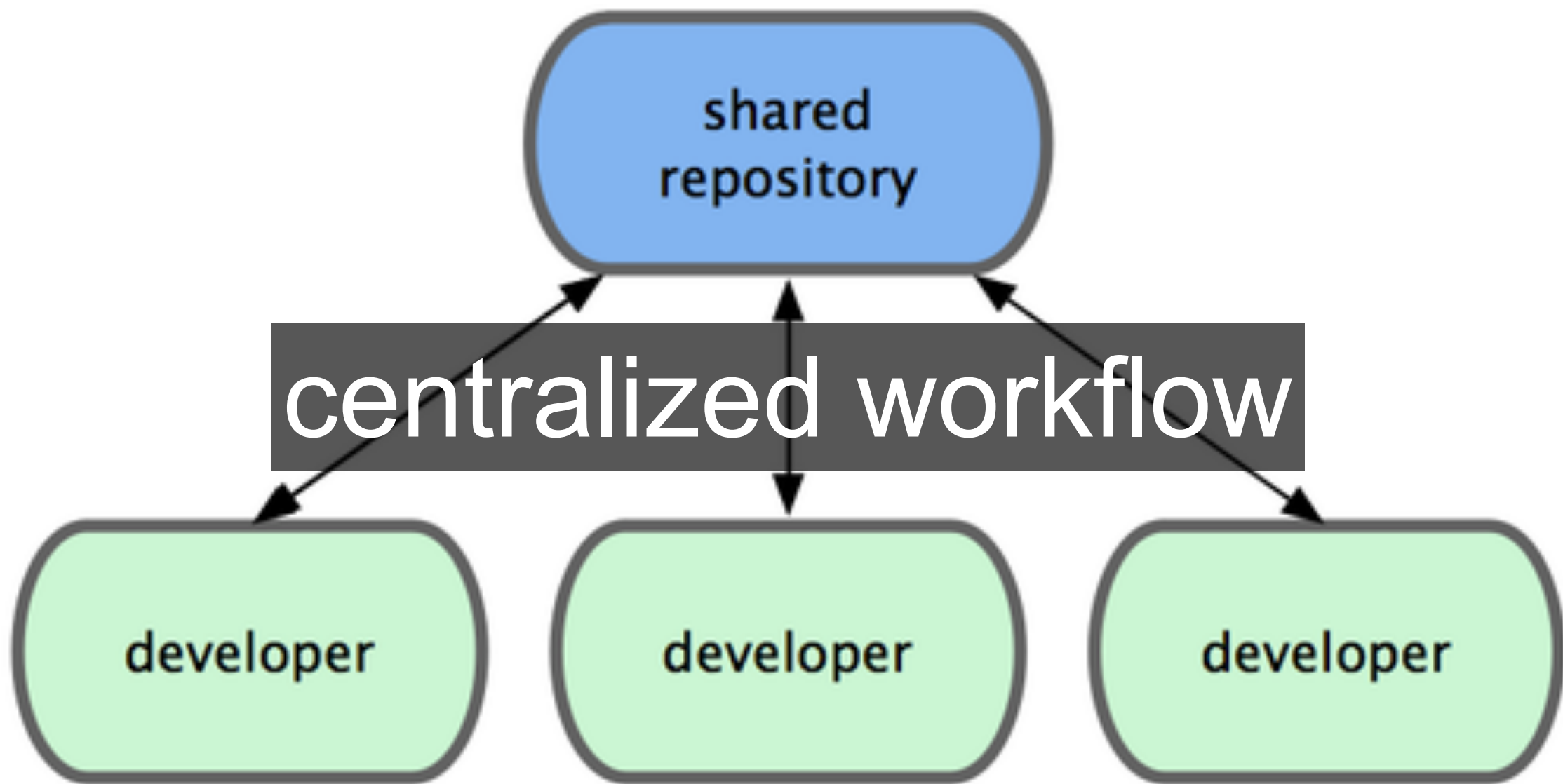




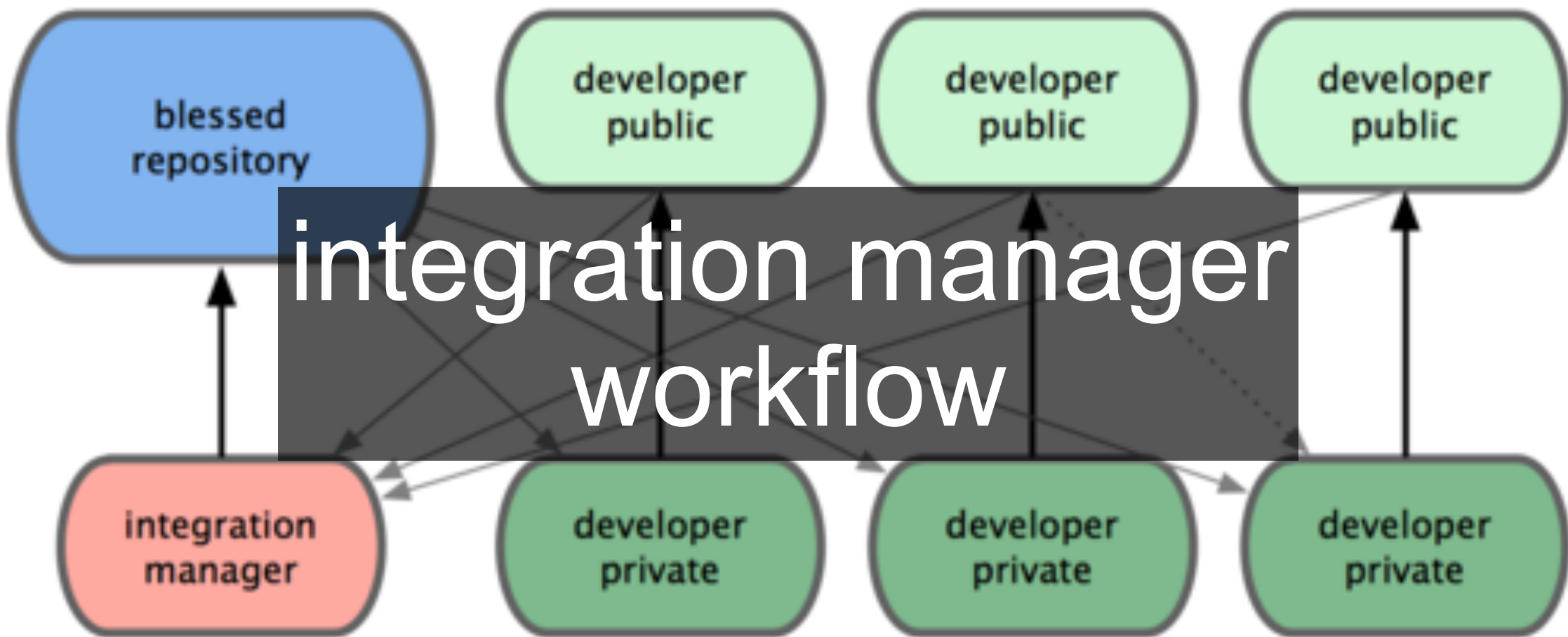
Git

Git can be used a number of different ways, there's the familiar ->

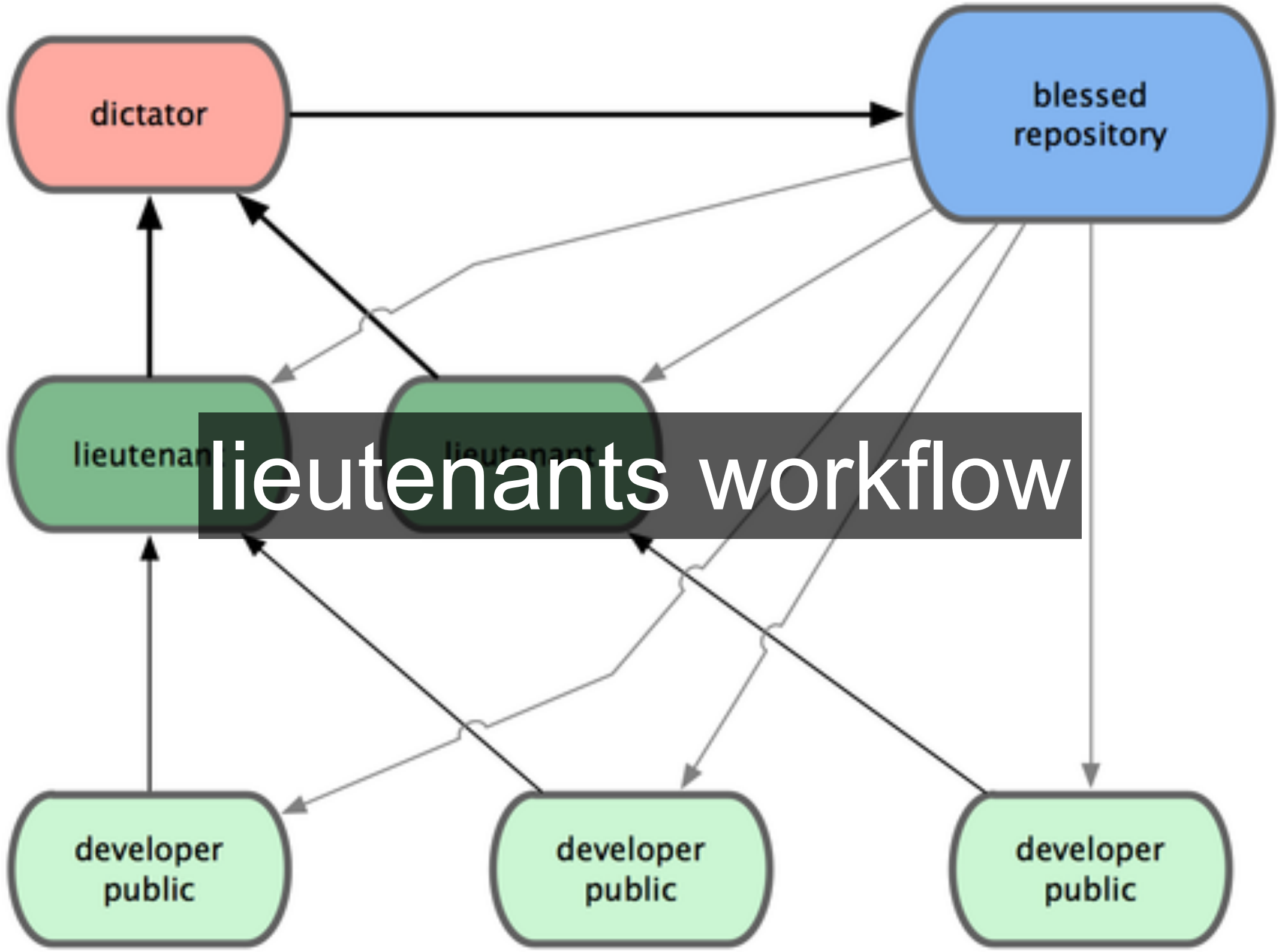




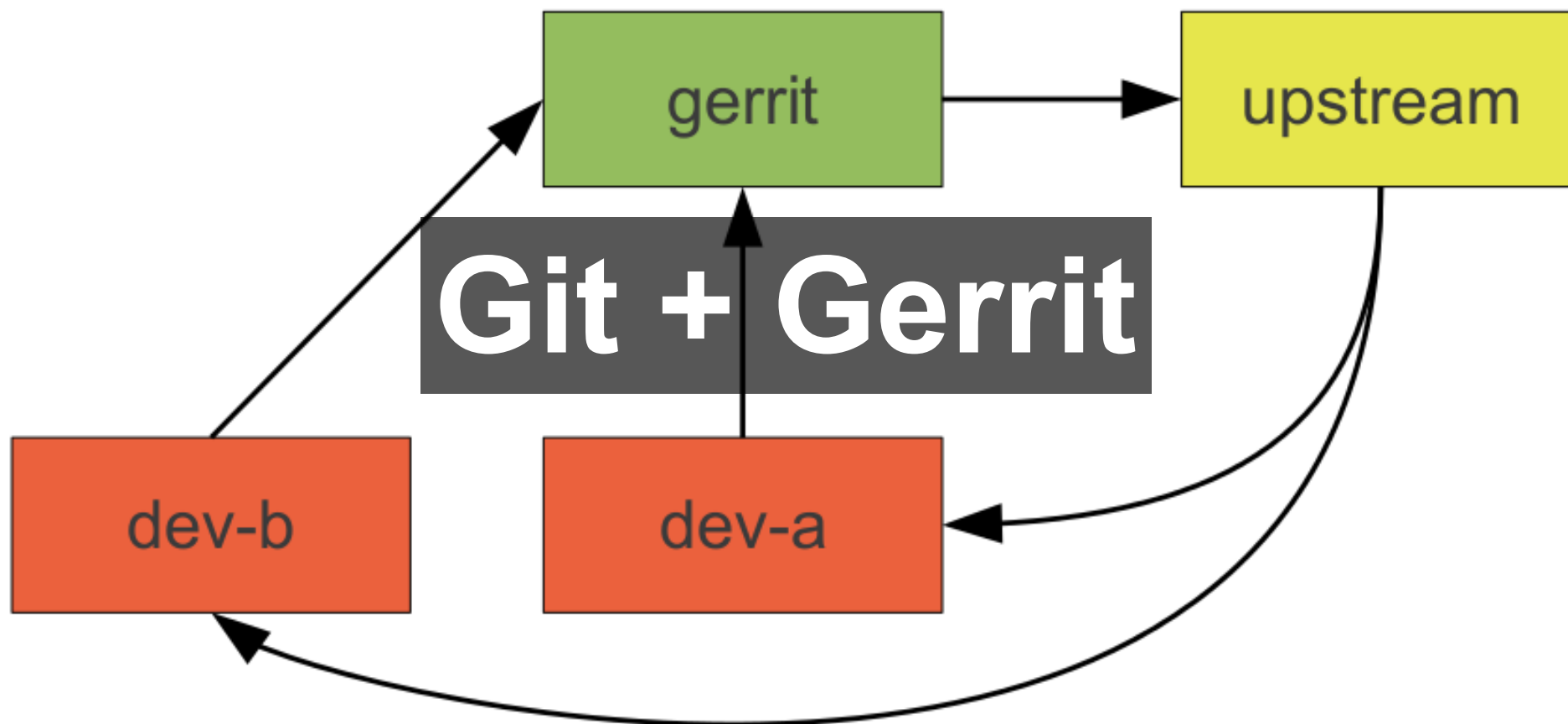
This is more common for smaller companies



This is common for GitHub-based projects



This is how the Linux kernel is developed



## Git and Gerrit





## Dashboard for Tyler Croy

Commit	Owner	Project	Branch
	Tyler Gray	feed	master (frontend-emulator)
	Tyler Gray	feed	master (frontend-emulator)
Tyler Gray		feed	master
		feed	master
		feed	master
		feed	master
(MERGED)		feed	master (in-app-recurring)
(MERGED)		feed	master (in-app-recurring)
(MERGED)		gemshwecippings	master
(MERGED)		gemshwecippings	master
(MERGED)		gemshwecippings	master
(MERGED)		gemshwecippings	master
(MERGED)		gemshwecippings	master
(MERGED)	Tyler Gray	feed	deploy-2012-07-19
(MERGED)	Tyler Gray	feed	master
(MERGED)	Tyler Gray	feed	deploy-2012-07-19
(MERGED)	Tyler Gray	feed	deploy-2012-07-17
(MERGED)	Tyler Gray	feed	master (5v4b-7576)

Gerrit is a Git-based code review tool

# code review

```
#include "ttclock.h"
#include <stdlib.h>
```

```
void)
```

```
struct sigaction sig;
ttclock->bg = COLOR_BLACK;
```

```
/* Init ncurses */
initscr();
cbreak();
noecho();
keypad(stdscr, True);
start_color();
```

```
#include "ttclock.h"
#include <stdlib.h>
```

```
void
init(void)
{
    struct sigaction sig;
    ttclock->bg = COLOR_BLACK;
    int *derp = NULL;
```

(Draft)

Draft saved

Poor variable naming

Edit

```
int foo = 1 + *derp;
```

(Draft)

Draft saved

This is a seriously terrible idea.

Edit

```
/* Init ncurses */
initscr();
cbreak();
noecho();
keypad(stdscr, True);
start_color();
```

# collaboration

```
struct sigaction sig;
```

```
ttclock->bc = 0; /* 1000 Hz */
```

```
int *derp = NULL;
```

**rieger** Poor variable naming

```
int foo = 1 + *derp;
```

**rieger** This is a seriously terrible idea.

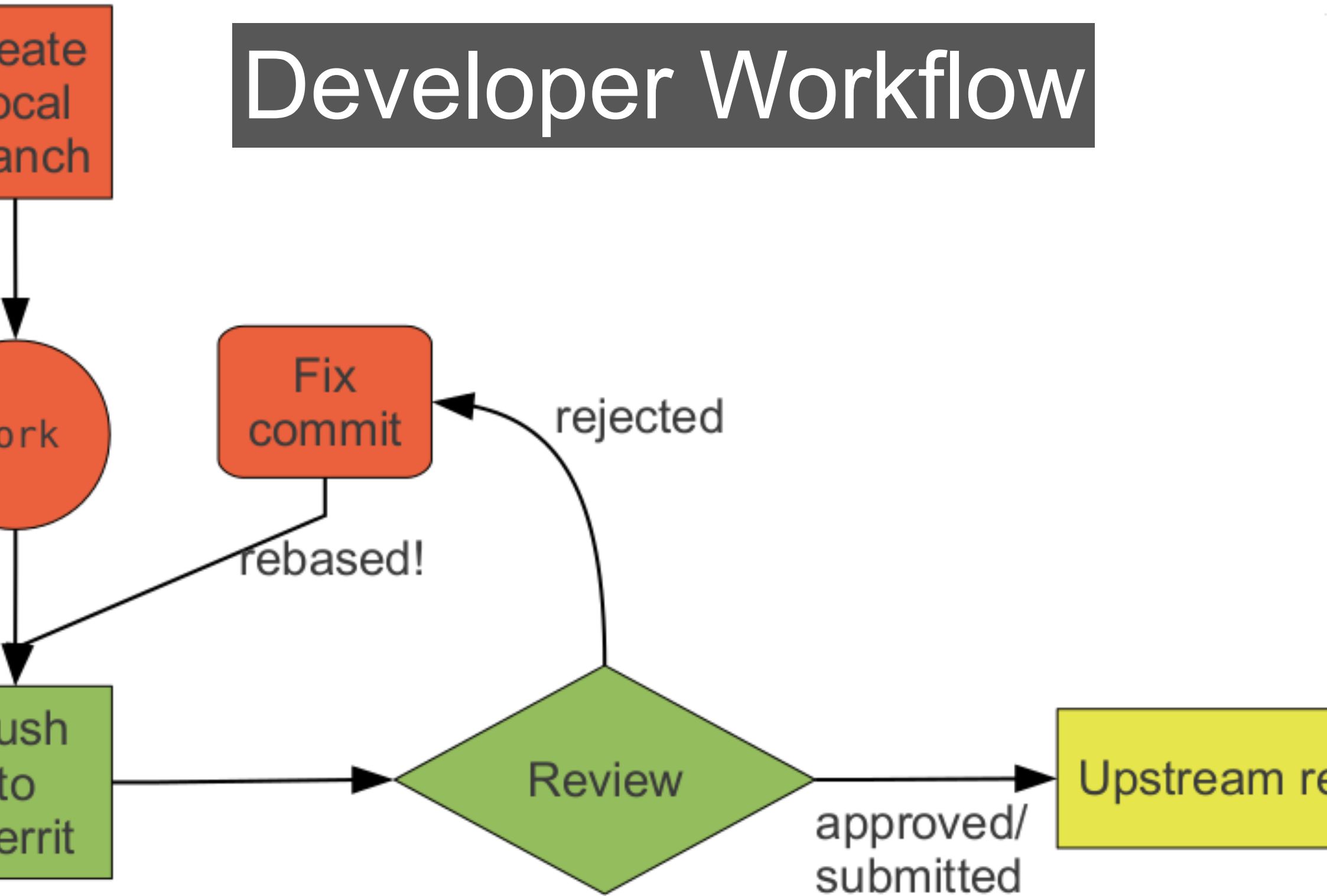
**tte** Girl please

**n Kane** Jesus christ Archer! Are you an idiot?

**ling Archer** Watch it Lana, you're in the \*danger zone\*

```
/* Init ncurses */
```

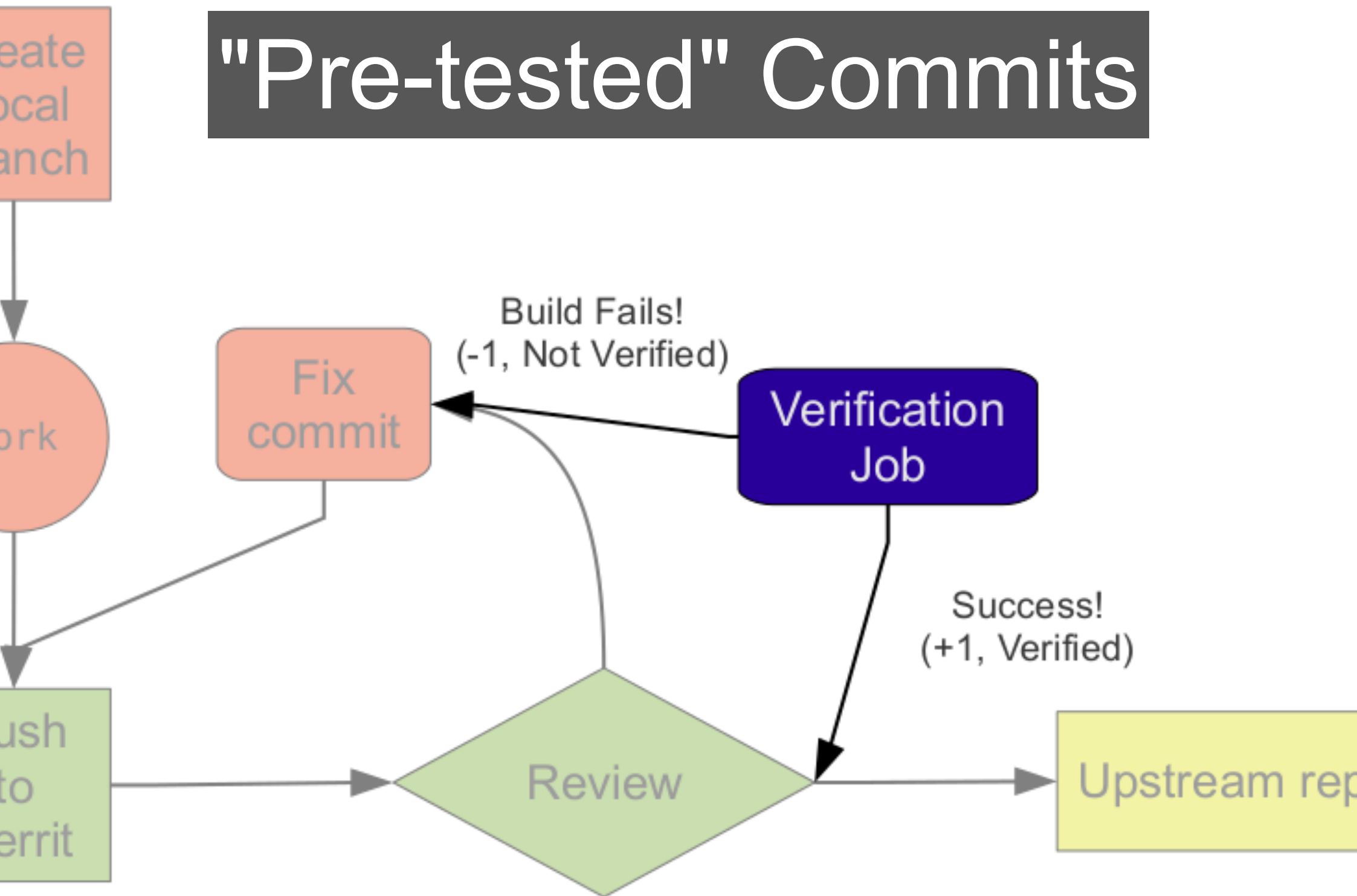
# Developer Workflow



What this means for an individual developer is that they can iterate on their code in Gerrit, based on feedback from their colleagues.

Once the code is all polished up, it can then be integrated into the "main" repository

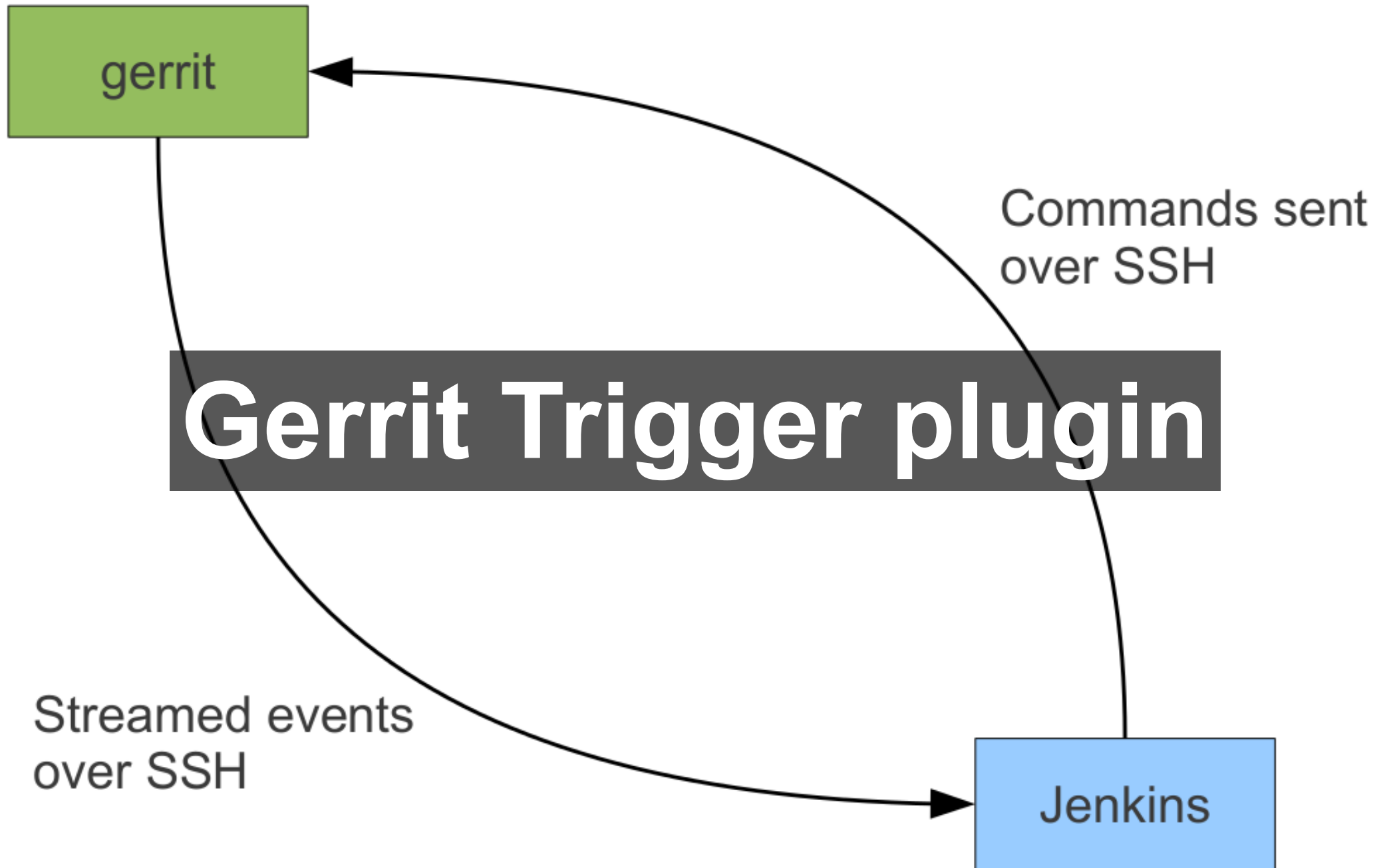
# "Pre-tested" Commits



An integral part of our Git + Gerrit workflow involved pre-testing commits.

The whole concept behind "pre-testing" a commit is that only changes which have passed the "tests" will be allowed to be integrated or merged.





Gerrit Trigger plugin

Uploaded Oct 1, 2011 12:42 PM

Updated Oct 1, 2011 12:44 PM

Status Review in Progress

# Feedback in Gerrit

[Permalink](#)

Reviewer	Verified	Code Review
<a href="#">Leeroy</a>	✗	Fails

- Need Verified +1 (Verified)
- Need Code Review +2 (Looks good to me, approved)

## ► Dependencies

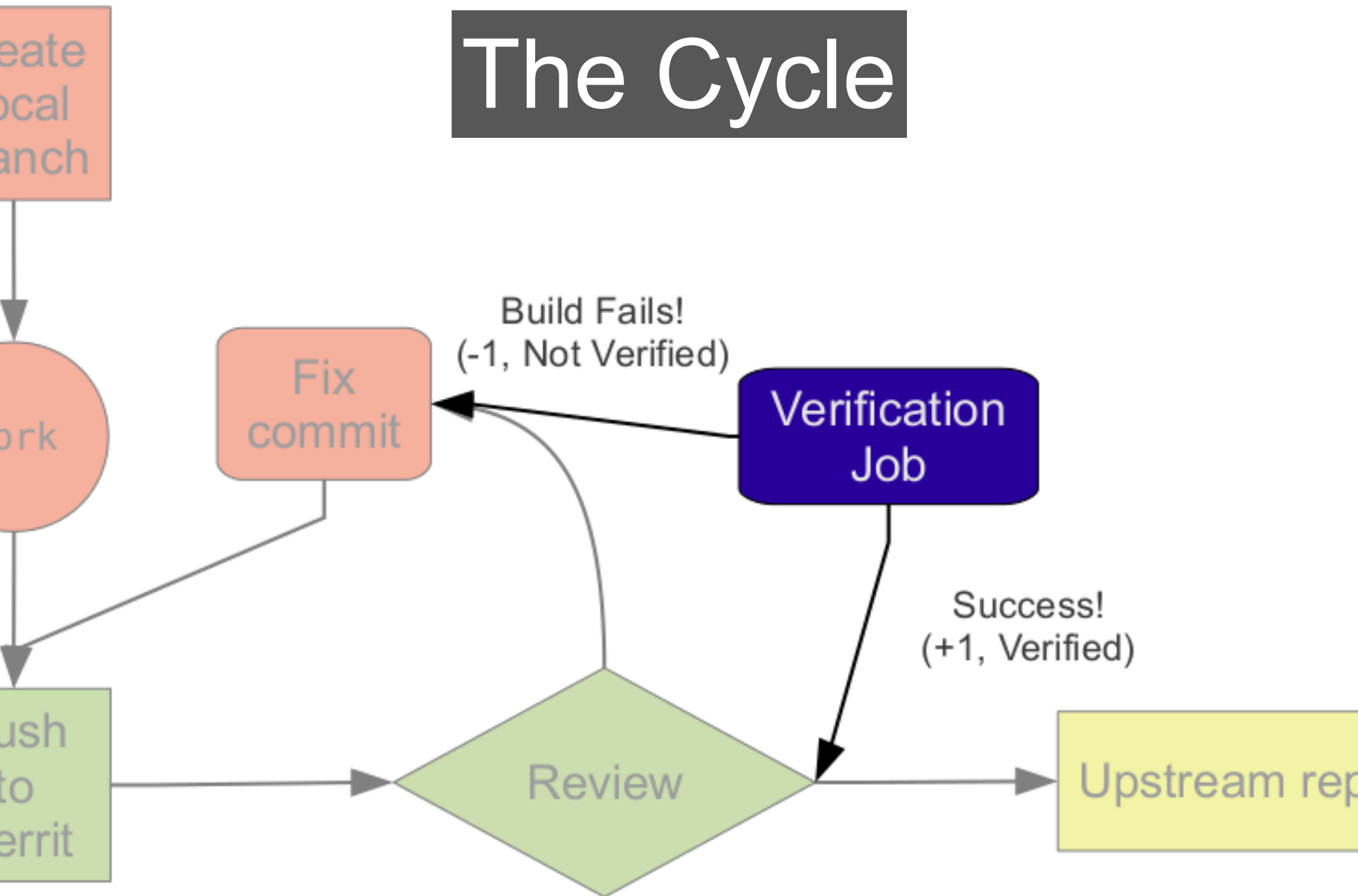
Old Version History:

► Patch Set 1 2b46eeeb675841f5e796e2e2910d2eee743c234e

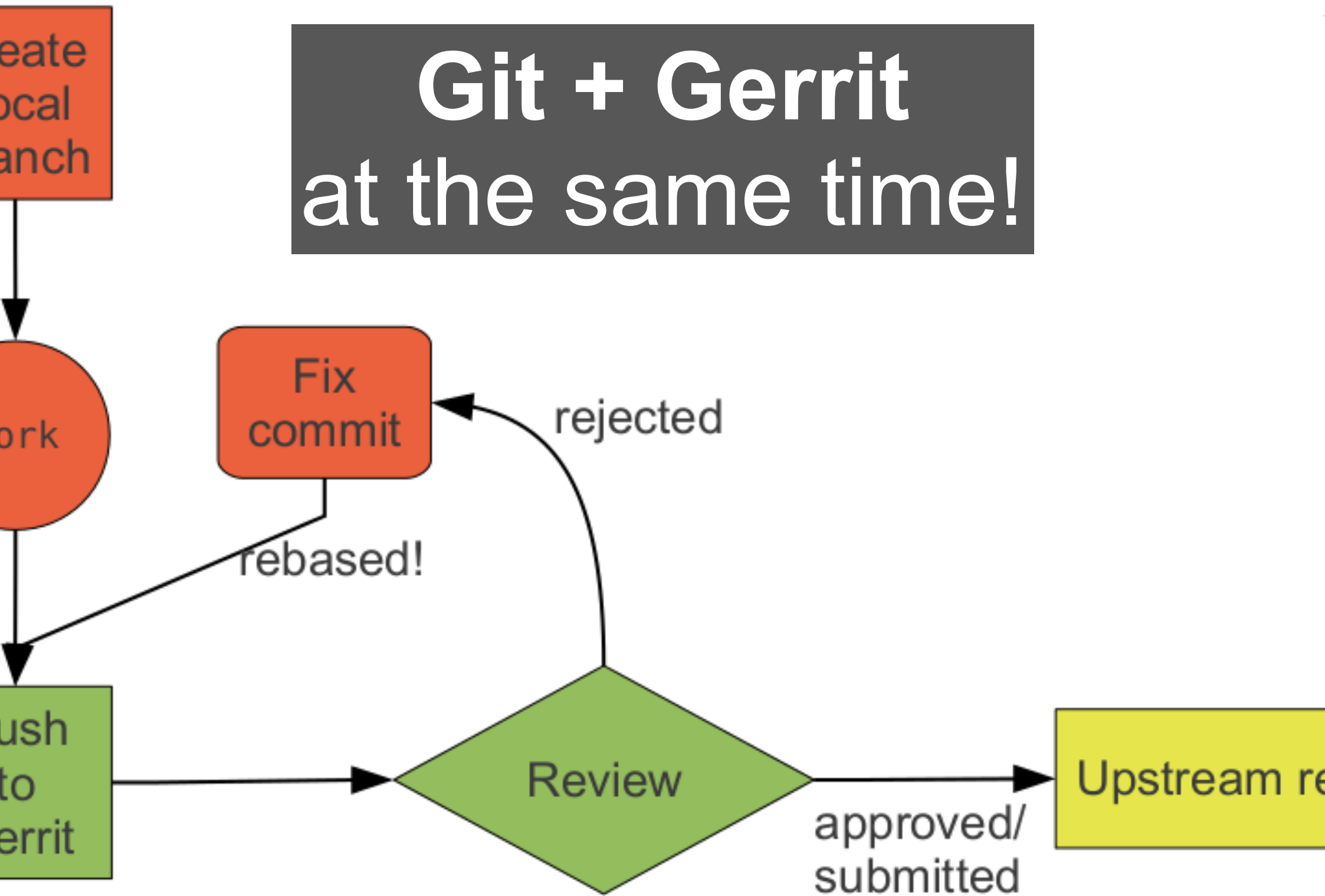
► Patch Set 2 a607d3649f17c69070d06163bcc91442ec1eae0f

► Patch Set 3 2a32f079abcb0ad09a4c59a2a2737a76b11591e9

# The Cycle



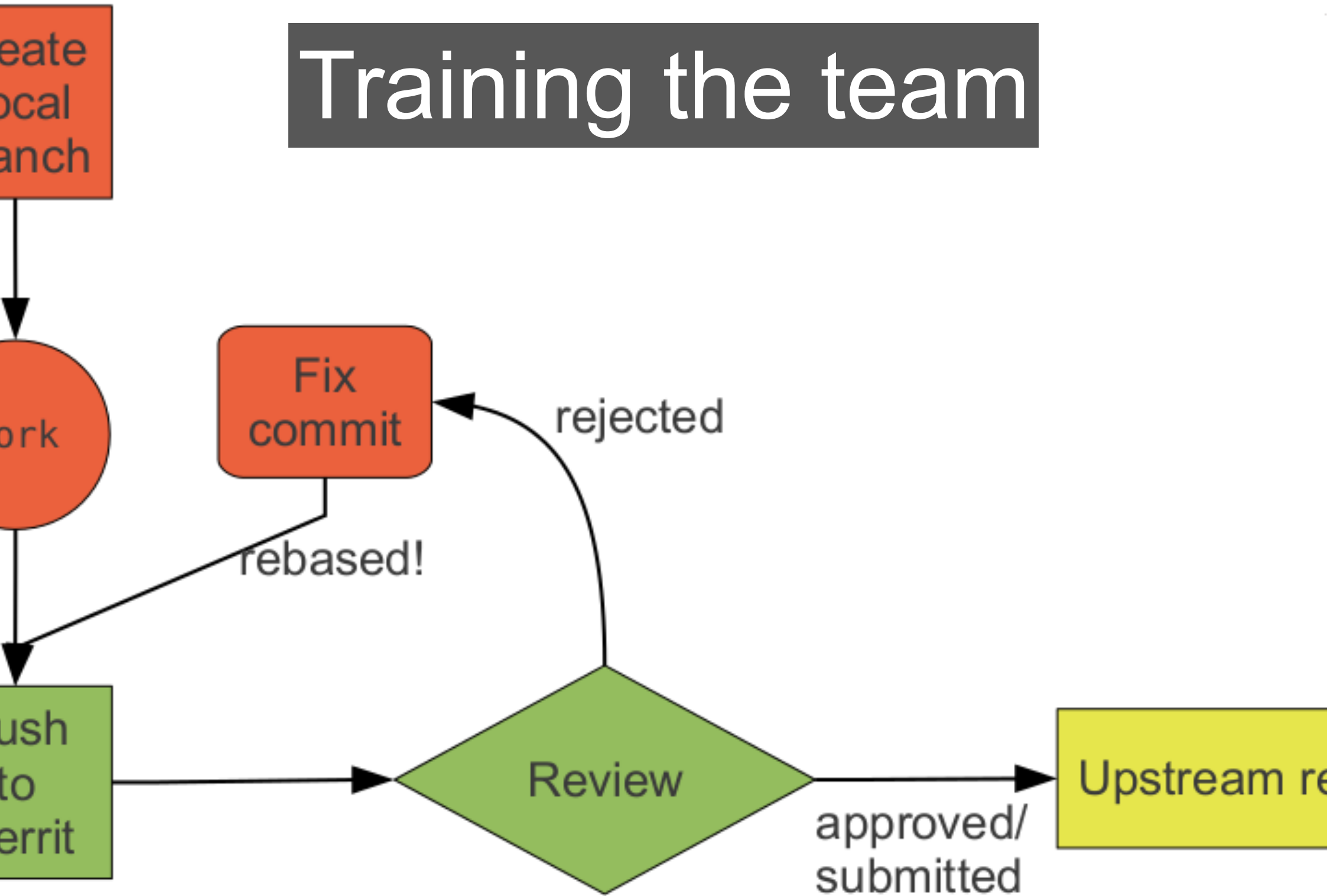
# Git + Gerrit at the same time!



TODO: Perhaps this should go after the pre-tested commits bit  
We switched from Subversion directly to Git and Gerrit, all at once.

Instead of introducing Git as a separate tool to developers, we introduced at the same time so developers never learned a Git-based workflow that *\*didn't\** involve Gerrit at its core.

# Training the team



We scheduled 3 different 1 hour training sessions with various groups of engineers in order to provide a hands-on walk-through of the Git + Gerrit workflow

This included a fully set-up "demo" project to use for experimentation of creating commits, code reviewing them, verifying them with Jenkins and finally merging them into the "master" branch

During the course of these training sessions, we used the feedback and common problems encountered by engineers to fill out a "getting started" wiki page which new hires now use to come up to speed with Git + Gerrit.



A large white passenger jet with red and blue stripes is flying over a smaller white propeller plane. The background is a light blue sky with white clouds. The text "We need more builders!" is overlaid on the image in a black box with white text.

"We need more builders!"

TODO: Make this less out of place

When we first started moving things into Jenkins we had 3 slaves that were properly configured for running our tests. As we started using pre-tested commits with Gerrit and Jenkins, we *\*very\** quickly realized that we needed

In the very early stages of this, we used the same hand-crafted VM base image with VMWare ESX, and then spun up multiple new machines.

This process has changed a bit since then, which I'll dig into in a moment



**Recap**

TODO: Diagram and cover "the world thus far"

A large commercial airplane, likely a Boeing 747, is shown in flight against a light blue sky. The plane is white with red and blue stripes along the fuselage and tail. The tail features a logo with the letters 'TA' inside a red square. The plane is flying over a large, stylized globe that is partially visible. A dark, semi-transparent banner is overlaid across the center of the image, containing the text 'Step Three Automate Everything' in white. The text is in a sans-serif font, with 'Automate' in a bold, italicized style.

# Step Three **Automate *Everything***



Deploying the test environment

Once the deployment of our test environment was managed through Jenkins, we created pipelines with Jenkins, chaining off of a successful deployment to the test environment.

TODO: Discuss selenium testing/SI testing after QA deploy

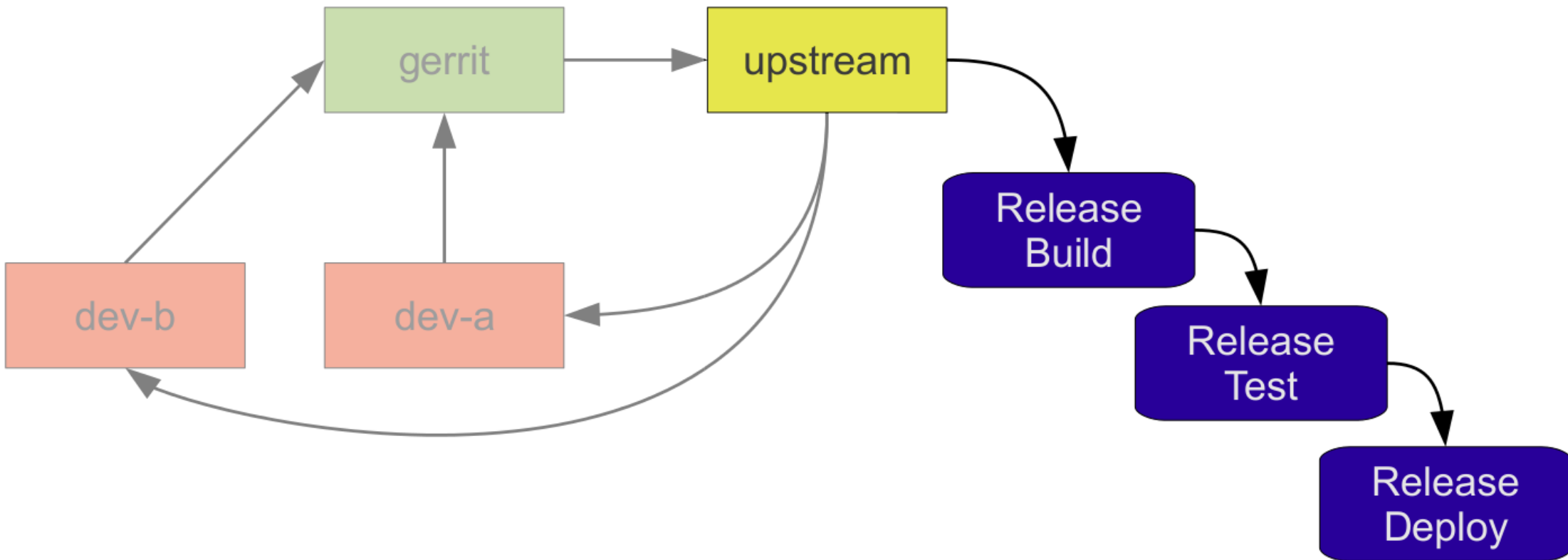


New kinds of tests!



TODO: Discuss Jasmine/Selenium tests with Sauce Labs

# Automating deployment

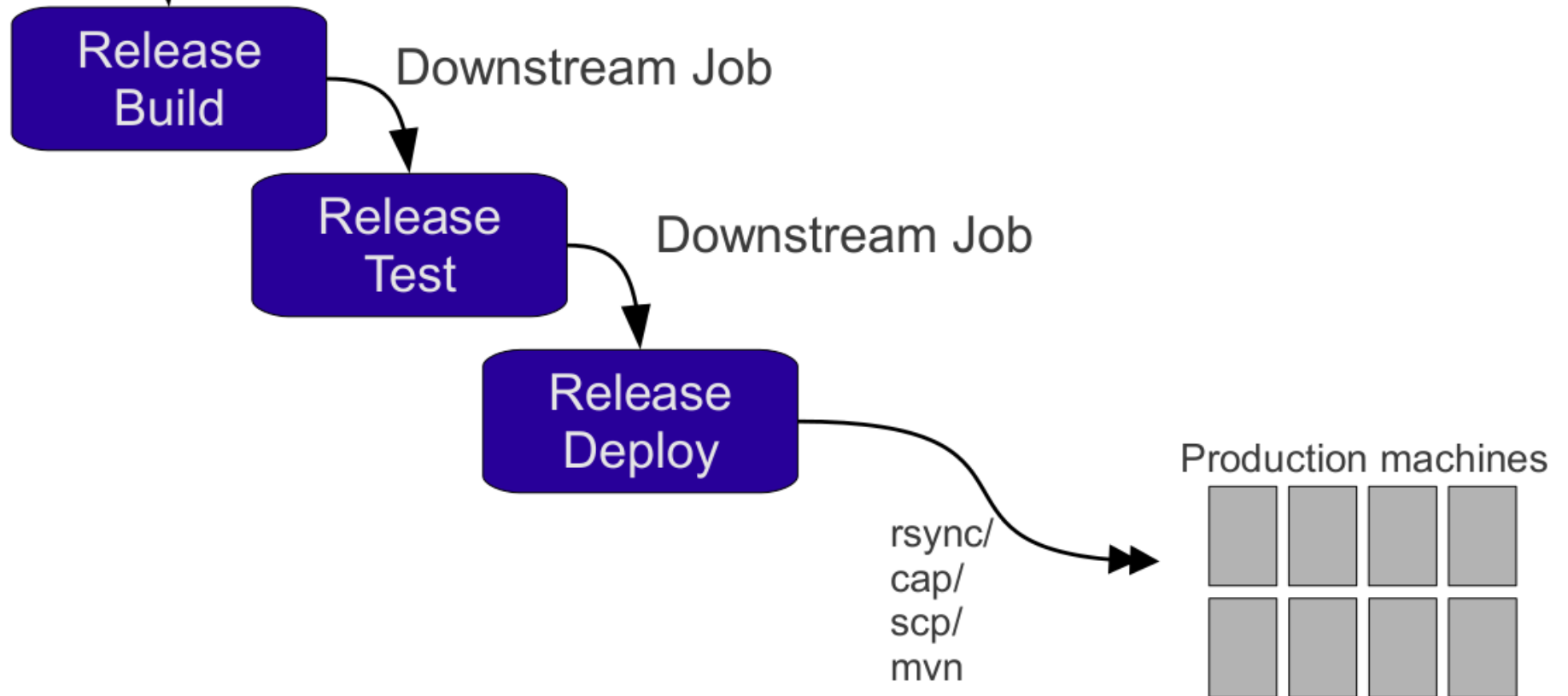


First automating with `infra\_update\_faithful2` then moving into the  
`infra\_deploy\_qa` territory. Finally automating the actual deployment of  
production

# Deploying to the

upstream

# *production* environment



TODO: Discuss the use of build promotions to stage, deploy and finally mark the deployment as successful

A large crowd of green Android robots is gathered on a cobblestone street. The robots are of various sizes and are all looking towards the camera. They have a simple, friendly design with large heads, small bodies, and two antennae on each head. The background is filled with more robots, creating a sense of a large assembly.

# Step Four More Robots



A large crowd of green Android robots, resembling the character Wall-E, is gathered on a cobblestone street. The robots are of various sizes and are looking in different directions. A semi-transparent black rectangular box is centered over the image, containing white text.

# OpenStack *and the* jclouds plugin

TODO: Discuss the investment in OpenStack for adding more build capacity.  
Looking forward to the work done with the jclouds plugin

A large number of green Android robots are standing in rows on a dark, textured surface. The robots are arranged in a grid-like pattern, filling the background. They have a classic Android design with a round head, two antennae, and a cylindrical body. The lighting is dramatic, with strong highlights and shadows, giving the scene a somewhat ominous or industrial feel.

# Slave Management *with* Puppet

TODO: discuss Puppetizing the slave machines

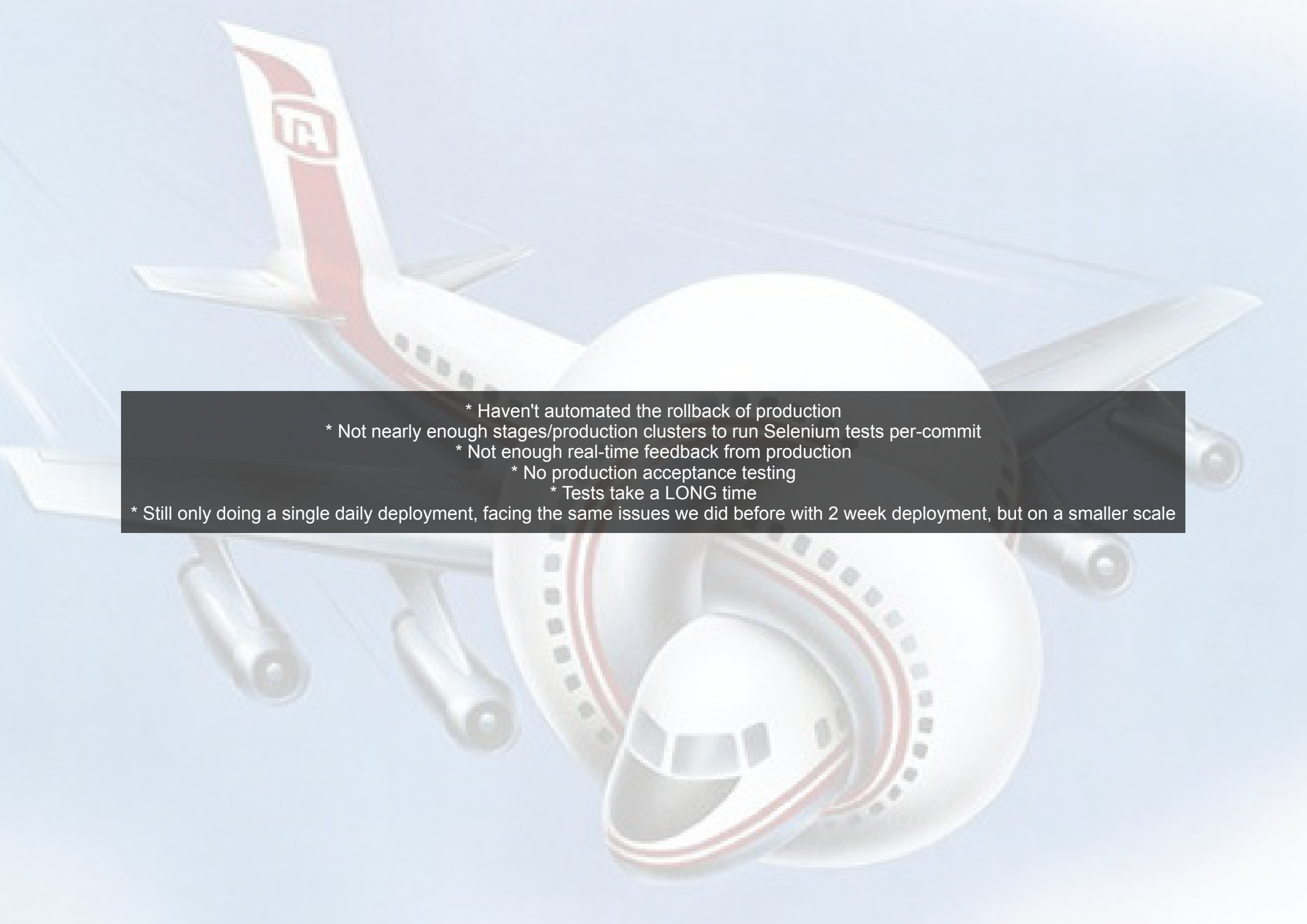


TODO



*(nobody's perfect)*



- 
- \* Haven't automated the rollback of production
  - \* Not nearly enough stages/production clusters to run Selenium tests per-commit
  - \* Not enough real-time feedback from production
    - \* No production acceptance testing
    - \* Tests take a LONG time
  - \* Still only doing a single daily deployment, facing the same issues we did before with 2 week deployment, but on a smaller scale



# Happiness with Numbers



**2%**  
of deployments failed



14

commits per deployment

A large commercial airplane is shown from a low-angle perspective, flying against a clear blue sky. The aircraft's white fuselage, wings, and tail are visible, with red and blue decorative stripes. A dark rectangular box is superimposed over the center of the image, containing the text "3% of deployments slipped" in white. The tail fin features a logo with the letters "TA" inside a red-bordered square.

**3%**  
of deployments slipped

***(that rocks!)***







Questions?