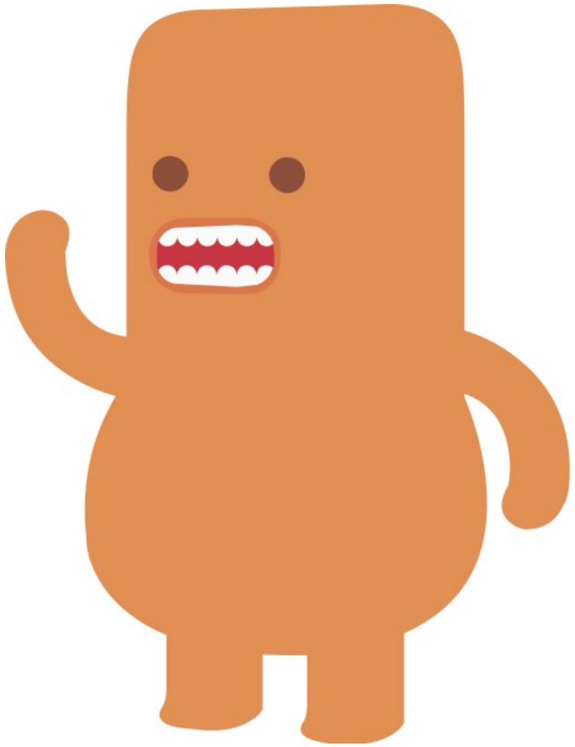
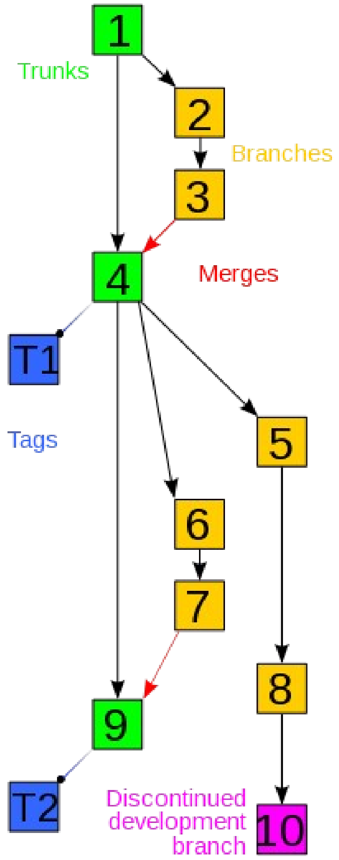
Git and Gerrit in Action

And lessons learned along the path to distributed version control

Chris Aniszczyk (Red Hat)
Principal Software Engineer
zx@redhat.com
http://aniszczyk.org





About Me

I've been using and hacking open source for ~12 years
- contribute{d} to Gentoo Linux, Fedora Linux, Eclipse

Hack on Eclipse, Git and other things at Red Hat

Member of the Eclipse Board of Directors

Member in the Eclipse Architecture Council

I like to run! (2 mins short of Boston qualifying ;/)

Co-author of RCP Book (www.eclipsercp.org)

Agenda

History of Version Control (VCS)

The Rise of Distributed Version Control (DVCS)

Code Review with Git and Gerrit

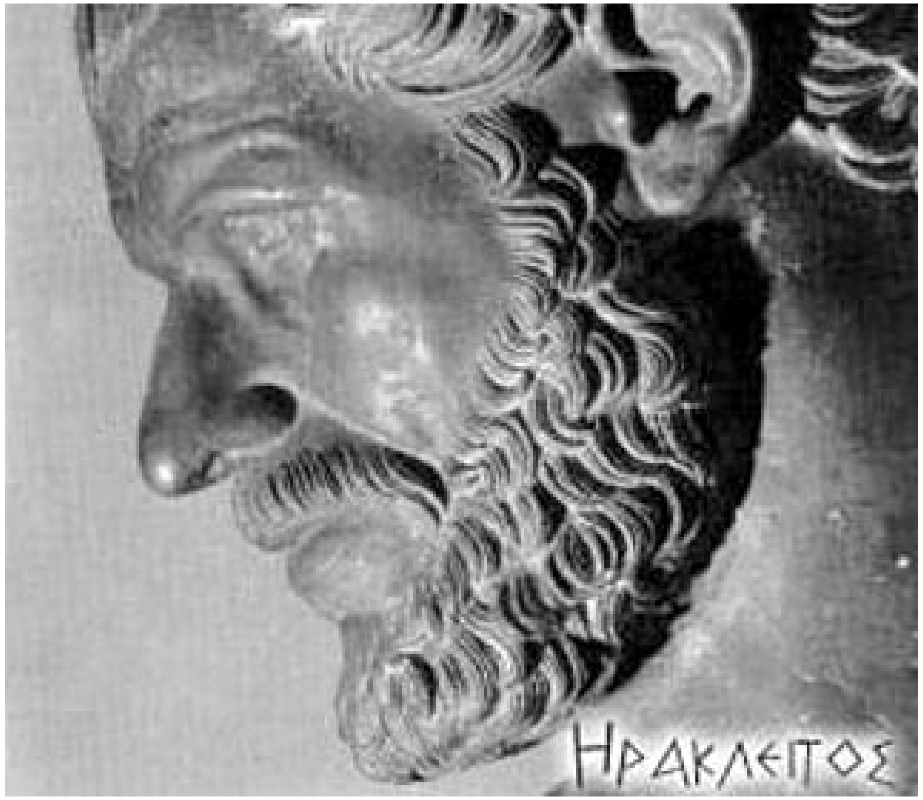
Lessons Learned at Eclipse moving to a DVCS

Conclusion

Q&A

Version Control

Version Control Systems manage **change**



“The only constant is change” (Heraclitus)

Why Version Control?

VCS became essential to software development because:

They allow teams to collaborate
They manage change and allow for inspection
They track ownership
They track evolution of changes
They allow for branching
They allow for continuous integration

Version Control: The Ancients

1972 – Source Code Control System (SCCS)

Born out of Bell Labs, based on interleaved deltas

No open source implementations as far as I know

1982 – Revision Control System (RCS)

Released as an alternative to SCCS

Operates on single files

Open source implementation hosted at GNU

Version Control: The Centralized

One centralized server with the revision information

Clients checkout a working copy locally

Most operations happen on the server

Linear revision history

Version Control: The Centralized

1990 – Concurrent Versions System (CVS)

Initially released as some scripts on top of RCS
Made branching possible for most people
Revisions by commits are per file :(
No atomic commit :(
Not really maintained anymore...

2000 – Subversion (SVN)

Released as an improvement to CVS
Atomic commits via transactions
Open source implementation hosted at Apache

“Hey, get back to work!”

… “My code's merging” - remember those days you spent merging in changes in CVS/SVN?

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S ~~COMPILING~~."
HEY! GET BACK TO WORK!
~~COMPILING!~~
OH. CARRY ON.

Version Control: The Distributed

Every client has a copy of the full repository locally

All repository operations are local (except sharing)

Intelligent network operations when sharing content

A very non linear revision history

Large online communities to share changes

Version Control: The Distributed

2001 – GNU arch

First open source DVCS

Deprecated; not maintained anymore

--- In 2005, Bitkeeper was no longer open source ---

2005 – Git

Created as the SCM for the Linux kernel by Linus

2005 – Mercurial (Hg)

Cross-platform DVCS

2007 – Bazaar (BZR)

Sponsored by Canonical

Agenda

History of Version Control (VCS)

The Rise of Distributed Version Control (DVCS)

- How does a DVCS like Git work?
- The benefits of a DVCS
- EGit Demo

Code Review with Git and Gerrit

Lessons Learned at Eclipse moving to a DVCS

Conclusion

Q&A

How does a DVCS like Git work?

A DVCS generally operates at the level of a *changeset*

Logically, a repository is made up from an initial empty state, followed by many changesets

Changesets are identified by a SHA-1 hash value

e.g., **0878a8189e6a3ae1ded86d9e9c7cbe3f**

It's all about the changesets

Changesets contain pointers to the previous changeset

```
previous: 48b2179994d494485b79504e8b5a6b23ce24a026
--- a/README.txt
+++ b/README.txt
@@ -1 +1 @@
-SVN is great
+Git is great

previous: 6ff60e964245816221414736d7e5fe6972246ead
--- a/README.txt
+++ b/README.txt
@@ -1 +1 @@
-Git is great
+SVN is great
```

Branches

The current version of your repository is simply a pointer to the end of the tree

The default "trunk" in Git is called "master"

The tip of the current branch is called "HEAD"

Any branch can be referred to by its SHA-1 hash id

Creating branches in a DVCS is fast, you simply point to a different element in the tree on disk already

Merging

DVCS are all about merging

Merges are just the weaving together of two (or more) local branches into one

However, unlike CVCS, you don't have to specify anything about where you're merging from and to; the trees automatically know what their split point was in the past, and can work it out from there.

Merging is much easier in a DVCS like Git

Pulling and Pushing

We've not talked about the distributed nature of DVCS

Changes flow between repositories by *push* and *pull*

Since a DVCS tree is merely a pointer to a branch...

There's three cases to consider for comparing two trees:

- Your tip is an ancestor of my tip
- My tip is an ancestor of your tip
- Neither of our tips are direct ancestors; however, we both share a common ancestor

Cloning and Remotes (git)

```
git clone git://egit.eclipse.org/egit.git
```

Where you can push or pull to is configured on a per (local) repository basis

```
git remote add github
http://github.com/caniszczyk/egit.git
```

origin is the default remote; you can have many remotes

Software Trends and Revolution

Most major open source projects use some form of DVCS

Git, Hg, Bazaar

Linux
MySQL
OpenJDK
Android
JQuery
Gnome
Fedora
Bugzilla and so on...

But why?

Benefits of Distributed Version Control

Can collaborate without a central authority

Disconnected operations

Easy branching and merging

Define your own workflow

Powerful community sharing tools

Easier path to contributor to committer

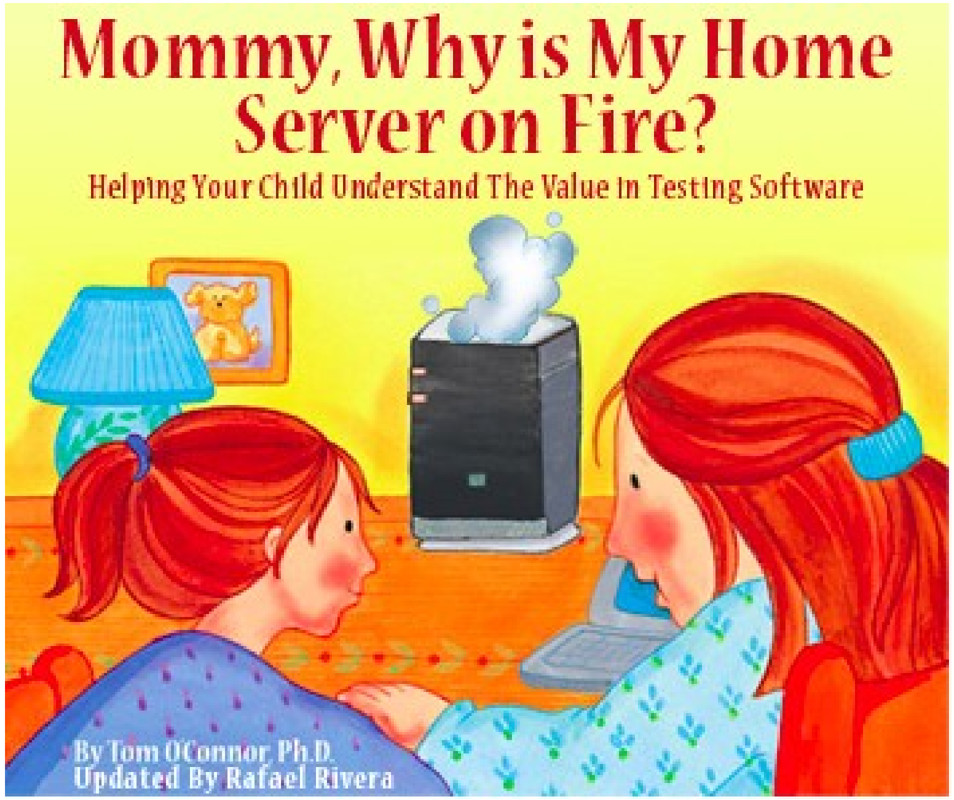
Collaboration

Developers can easily collaborate directly without needing a central authority or dealing with server administration costs

Disconnected operations rule!

Developers can still be productive and not worry about a central server going down... remember the days of complaining that CVS was down and you couldn't work?

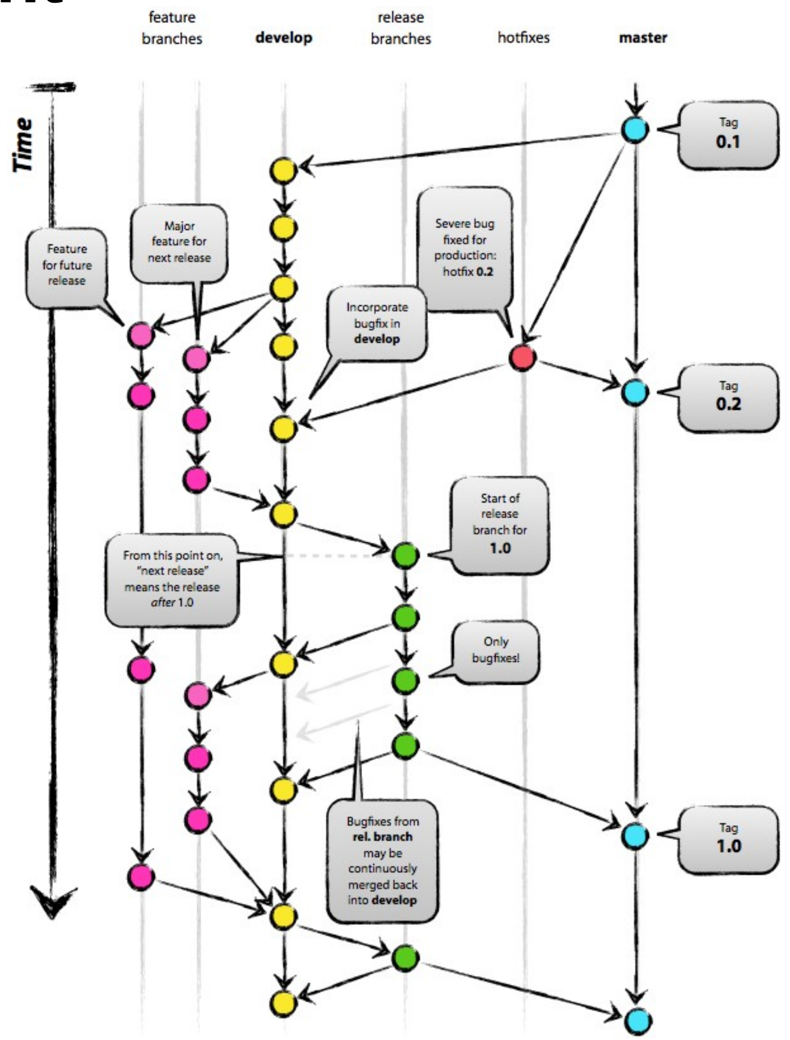
Also, there's a lighter server load for administrators!


Mommy, Why is My Home Server on Fire?
Helping Your Child Understand The Value in Testing Software
By Tom OConnor, Ph.D.
Updated By Rafael Rivera

Branches everywhere

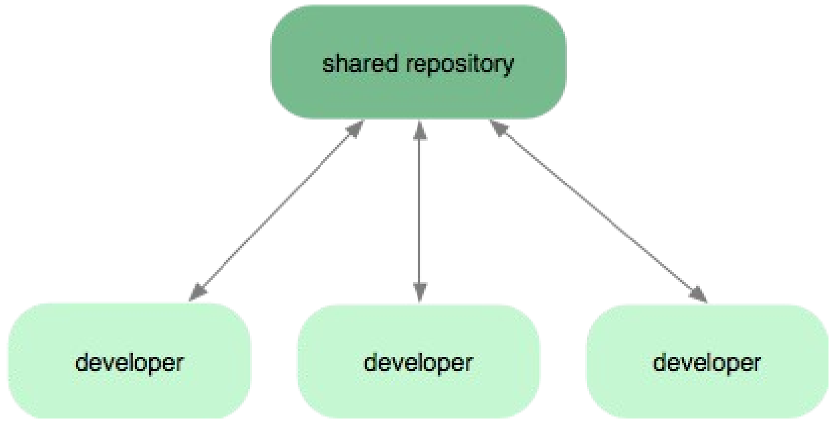
Creating and destroying branches are simple operations so it's easy to experiment with new ideas

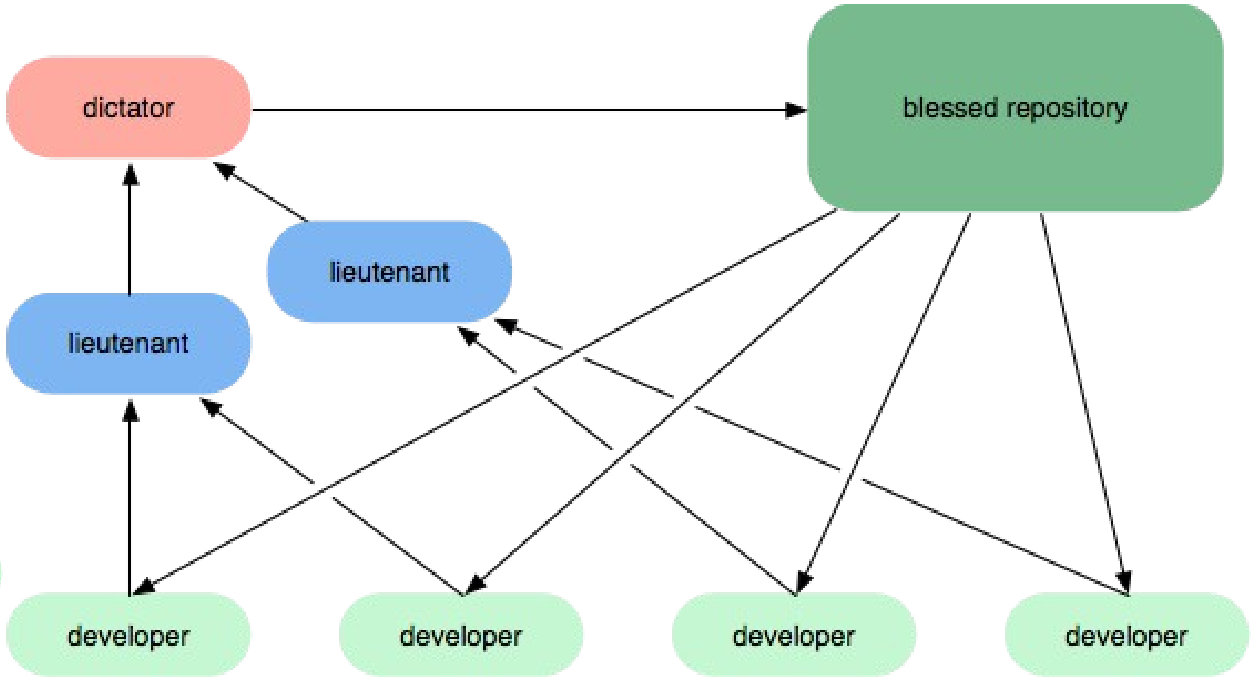
Very easy to isolate changes

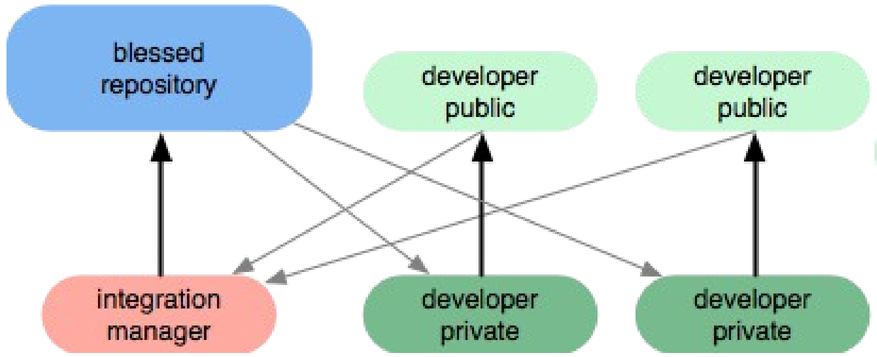

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Major feature for next release
Feature for future release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Define your own workflow

Define your own workflow to meet your team needs. Different workflows can be adopted as your team grows without changing VCS toolsets!





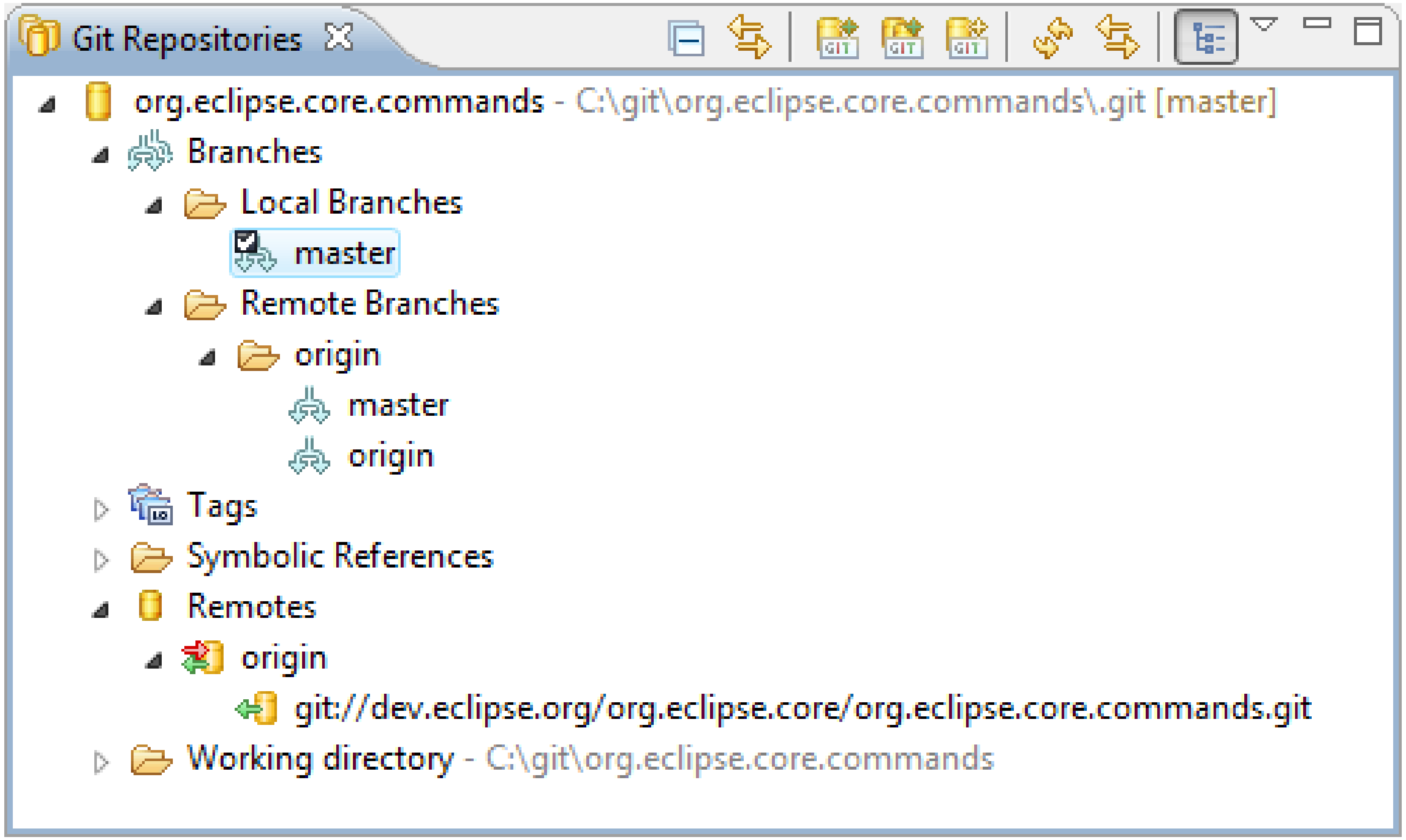


DVCS and Building Community

Developers can easily discover and fork projects. On top of that, it's simple for developers to share their changes

“Distributed version control is all about empowering your community, and the people who might join your community” - Mark Shuttleworth

EGit and GitHub Tooling Demo



Git Repositories

```
org.eclipse.core.commands - C:\git\org.eclipse.core.commands\.git [master]
  Branches
    Local Branches
      master
    Remote Branches
      origin
        master
        origin
  Tags
  Symbolic References
  Remotes
    origin
      git://dev.eclipse.org/org.eclipse.core/org.eclipse.core.commands.git
  Working directory - C:\git\org.eclipse.core.commands
```

Agenda

History of Version Control (VCS)

The Rise of Distributed Version Control (DVCS)

Code Review with Git and Gerrit

- **Why Code Review**
- **Introduction to Gerrit**
- **Gerrit and Mylyn Reviews Demo**

Lessons Learned at Eclipse moving to DVCS

Conclusion

Q&A

What is Code Review?

When one developer writes code, another developer is asked to review that code

A careful line-by-line critique

Happens in a non-threatening context

Goal is cooperation, not fault-finding

Often an integral part of coding process

Debugging someone else's broken code
– **Involuntary code review:** Not so good; emotions may flare

[1] http://code.google.com/p/rietveld/downloads/detail?name=Mondrian2006.pdf

Why Code Review?

Four eyes catch more bugs

- Catch bugs early to save hours of debugging

Enforce coding standards

- Keep overall readability & code quality high

Mentoring of new developers

- Learn from mistakes without breaking stuff

Establish trust relationships

- Prepare for more delegation

Good alternative to pair programming

- asynchronous and across locations

[1] http://code.google.com/p/rietveld/downloads/detail?name=Mondrian2006.pdf

Gerrit Code Review

Gerrit is a Code Review system based on JGit
http://code.google.com/p/gerrit/

Also serves as a git server
adding access control and workflow

Used by

• Android	https://review.source.android.com/
• JGit, EGit	http://egit.eclipse.org/r/
• Google, Red Hat, SAP, …	

History: Google and code review tools

Mondrian (Guido van Rossum)
- based on Perforce, Google infrastructure
- Google proprietary

Rietvield (Guido van Rossum)
- based on Subversion
- Open Source hosted on GoogleApp Engine

Gerrit (Shawn Pearce)
- started as a fork of Rietvield
- based on JGit and GWT
- Open Source (Android)
- Apache 2 license

One Branch One Feature

Master branch contains only reviewed and approved changes

- master moves from good to better state after each (approved) change

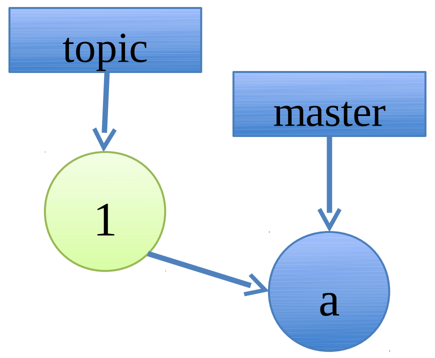
Each feature branch is based on master branch

- stable starting point

A change can really be abandoned because

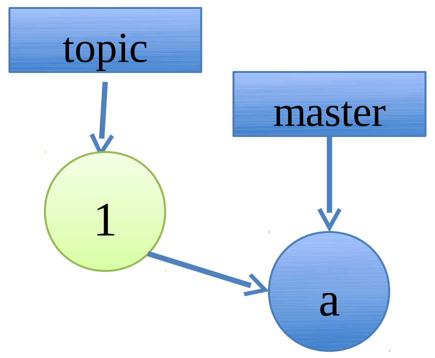
- no other approved change can depend on a not yet approved change
- Gerrit will automatically reject a successor change of an abandoned change

Gerrit – Lifecycle of a Change

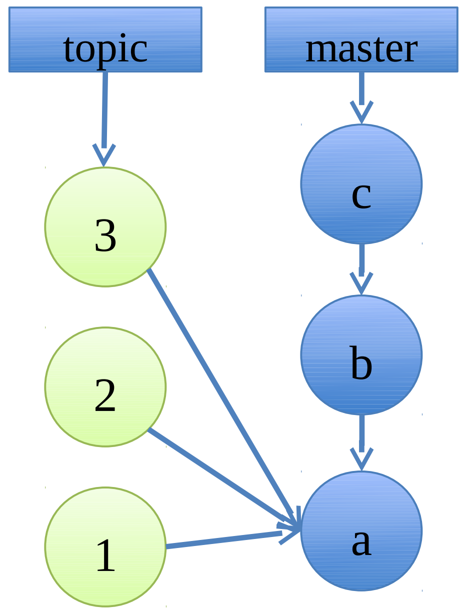


- create local topic branch
- commit change
- push it for review
- do review
- automated verification

Gerrit – Lifecycle of a Change

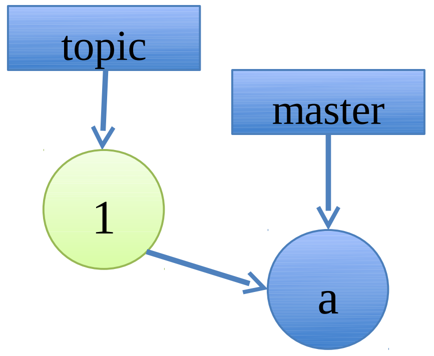


- create local topic branch
- commit change
- push it for review
- do review
- automated verification

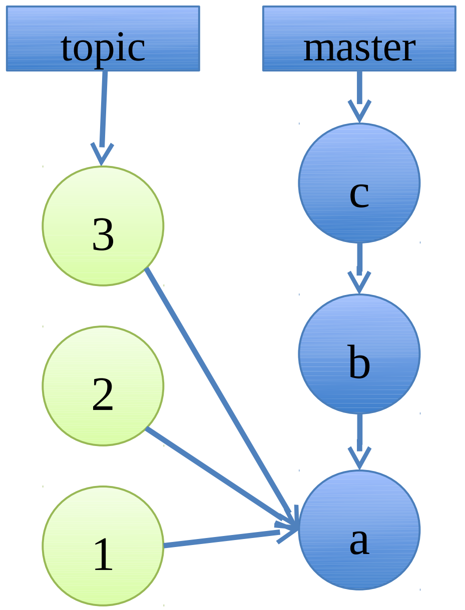


- refine based on review
- push new patchsets until review votes ok

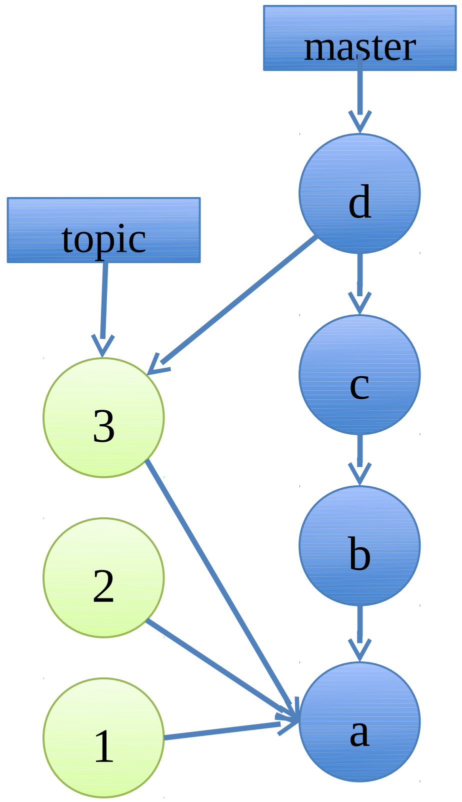
Gerrit – Lifecycle of a Change



- create local topic branch
- commit change
- push it for review
- do review
- automated verification

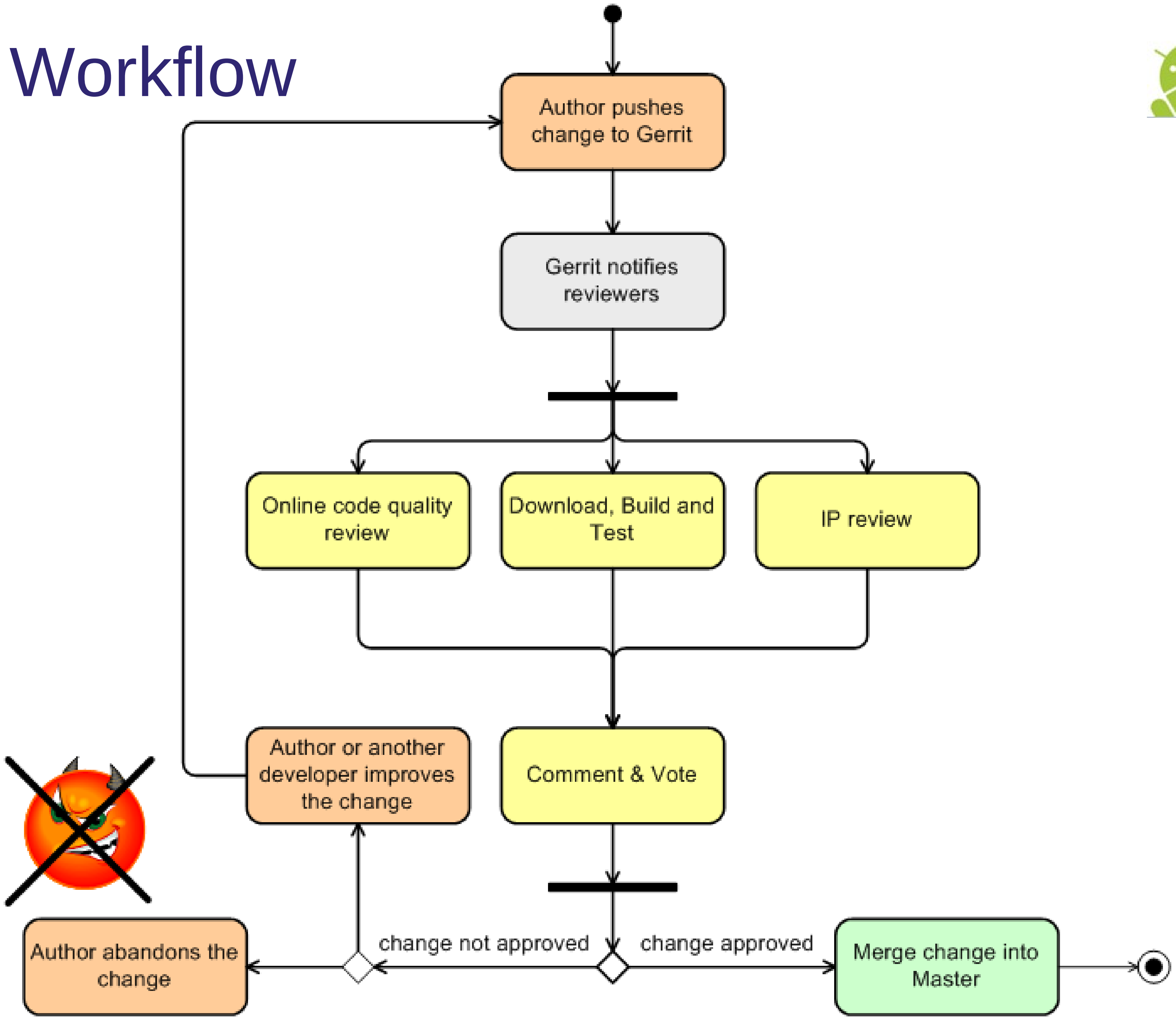


- refine based on review
- push new patchsets until review votes ok

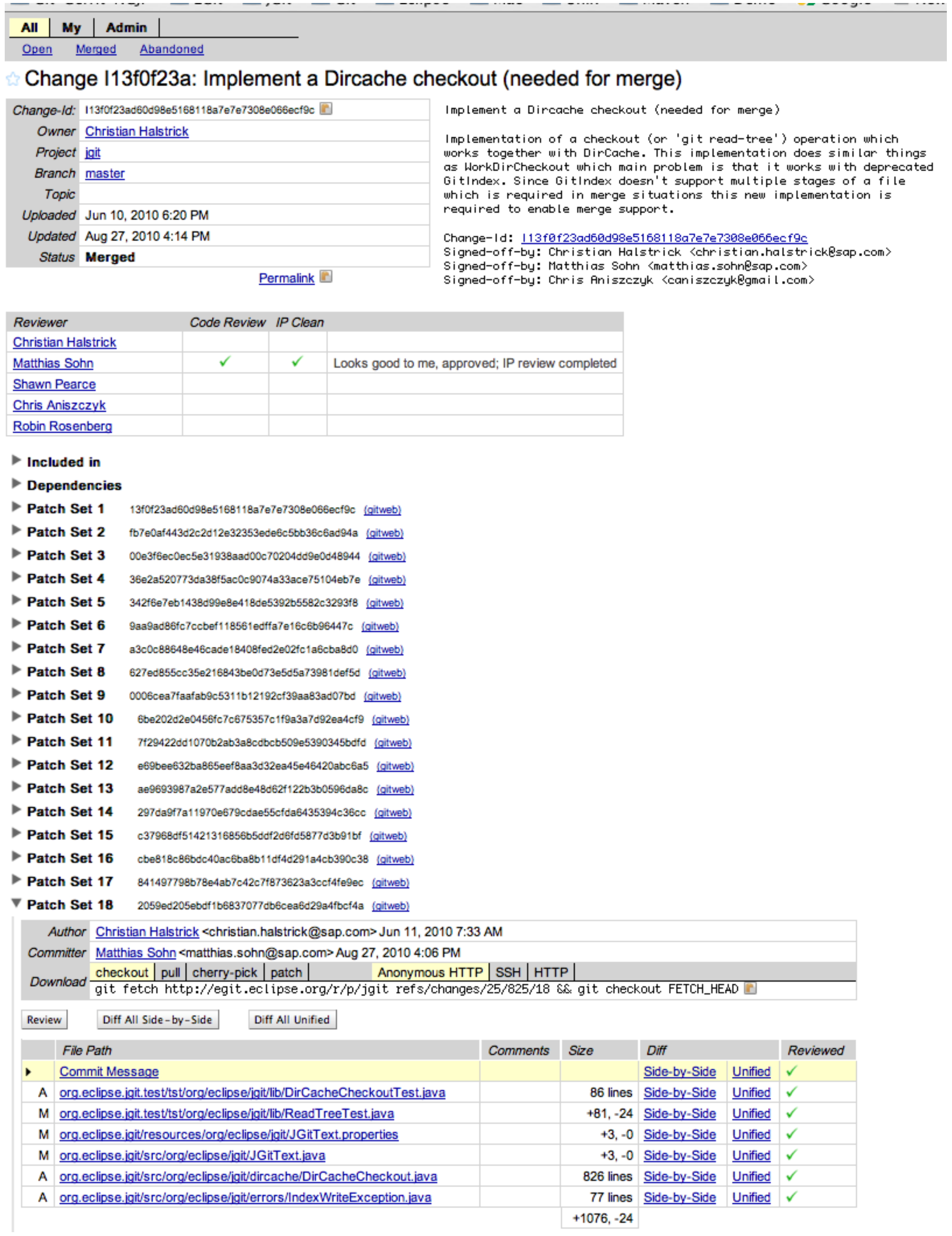


- Submit may lead to server-side merge
- or merge / rebase before push

Gerrit Workflow


Author pushes change to Gerrit
Gerrit notifies reviewers
Online code quality review
Download, Build and Test
IP review
Author or another developer improves the change
Comment & Vote
change not approved
change approved
Author abandons the change
Merge change into Master

Gerrit



All | **My** | **Admin**

Open Merged Abandoned

Change I13f0f23a: Implement a Dircache checkout (needed for merge)

Change-Id:	I13f0f23ad60d98e5168118a7e7e7308e066ecf9c
Owner	Christian Halstrick
Project	jgit
Branch	master
Topic	
Uploaded	Jun 10, 2010 6:20 PM
Updated	Aug 27, 2010 4:14 PM
Status	**Merged**

Permalink

```
Implement a Dircache checkout (needed for merge)

Implementation of a checkout (or 'git read-tree') operation which
works together with DirCache. This implementation does similar things
as WorkDirCheckout which main problem is that it works with deprecated
GitIndex. Since GitIndex doesn't support multiple stages of a file
which is required in merge situations this new implementation is
required to enable merge support.

Change-Id: I13f0f23ad60d98e5168118a7e7e7308e066ecf9c
Signed-off-by: Christian Halstrick <christian.halstrick@sap.com>
Signed-off-by: Matthias Sohn <matthias.sohn@sap.com>
Signed-off-by: Chris Aniszczyk <caniszczyk@gmail.com>
```

Reviewer	Code Review	IP Clean	
Christian Halstrick			
Matthias Sohn	✓	✓	Looks good to me, approved; IP review completed
Shawn Pearce			
Chris Aniszczyk			
Robin Rosenberg			

▶ **Included in**
▶ **Dependencies**

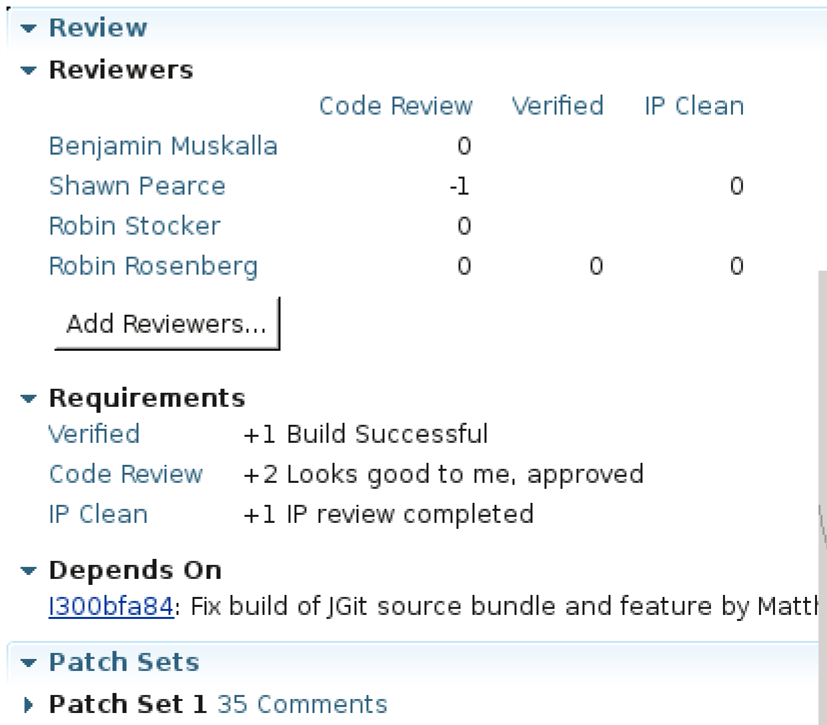
▶ **Patch Set 1**	13f0f23ad60d98e5168118a7e7e7308e066ecf9c	(gitweb)
▶ **Patch Set 2**	fb7e0af443d2c2d12e32353ede6c5bb36c6ad94a	(gitweb)
▶ **Patch Set 3**	00e3f6ec0ec5e31938aad00c70204dd9e0d48944	(gitweb)
▶ **Patch Set 4**	36e2a520773da38f5ac0c9074a33ace75104eb7e	(gitweb)
▶ **Patch Set 5**	342f6e7eb1438d99e8e418de5392b5582c3293f8	(gitweb)
▶ **Patch Set 6**	9aa9ad86fc7ccbef118561edffa7e16c6b96447c	(gitweb)
▶ **Patch Set 7**	a3c0c88648e46cade18408fed2e02fc1a6cba8d0	(gitweb)
▶ **Patch Set 8**	627ed855cc35e216843be0d73e5d5a73981def5d	(gitweb)
▶ **Patch Set 9**	0006cea7faafab9c5311b12192cf39aa83ad07bd	(gitweb)
▶ **Patch Set 10**	6be202d2e0456fc7c675357c1f9a3a7d92ea4cf9	(gitweb)
▶ **Patch Set 11**	7f29422dd1070b2ab3a8cdbcb509e5390345bdfd	(gitweb)
▶ **Patch Set 12**	e69bee632ba865eef8aa3d32ea45e46420abc6a5	(gitweb)
▶ **Patch Set 13**	ae9693987a2e577add8e48d62f122b3b0596da8c	(gitweb)
▶ **Patch Set 14**	297da9f7a11970e679cdae55cfda6435394c36cc	(gitweb)
▶ **Patch Set 15**	c37968df51421316856b5ddf2d6fd5877d3b91bf	(gitweb)
▶ **Patch Set 16**	cbe818c86bdc40ac6ba8b11df4d291a4cb390c38	(gitweb)
▶ **Patch Set 17**	841497798b78e4ab7c42c7f873623a3ccf4fe9ec	(gitweb)
▼ **Patch Set 18**	2059ed205ebdf1b6837077db6cea6d29a4fbcf4a	(gitweb)

Author	Christian Halstrick <christian.halstrick@sap.com> Jun 11, 2010 7:33 AM
Committer	Matthias Sohn <matthias.sohn@sap.com> Aug 27, 2010 4:06 PM
Download	checkout | pull | cherry-pick | patch Anonymous HTTP | SSH | HTTP

```
git fetch http://egit.eclipse.org/r/p/jgit refs/changes/25/825/18 && git checkout FETCH_HEAD
```

	File Path	Comments	Size	Diff		Reviewed
▶	Commit Message			Side-by-Side	Unified	✓
A	org.eclipse.jgit.test/tst/org/eclipse/jgit/lib/DirCacheCheckoutTest.java		86 lines	Side-by-Side	Unified	✓
M	org.eclipse.jgit.test/tst/org/eclipse/jgit/lib/ReadTreeTest.java		+81, -24	Side-by-Side	Unified	✓
M	org.eclipse.jgit/resources/org/eclipse/jgit/JGitText.properties		+3, -0	Side-by-Side	Unified	✓
M	org.eclipse.jgit/src/org/eclipse/jgit/JGitText.java		+3, -0	Side-by-Side	Unified	✓
A	org.eclipse.jgit/src/org/eclipse/jgit/dircache/DirCacheCheckout.java		826 lines	Side-by-Side	Unified	✓
A	org.eclipse.jgit/src/org/eclipse/jgit/errors/IndexWriteException.java		77 lines	Side-by-Side	Unified	✓
			+1076, -24			

Gerrit and Mylyn Reviews Demo



Review

Reviewers

	Code Review	Verified	IP Clean
Benjamin Muskalla	0		
Shawn Pearce	-1		0
Robin Stocker	0		
Robin Rosenberg	0	0	0

Requirements

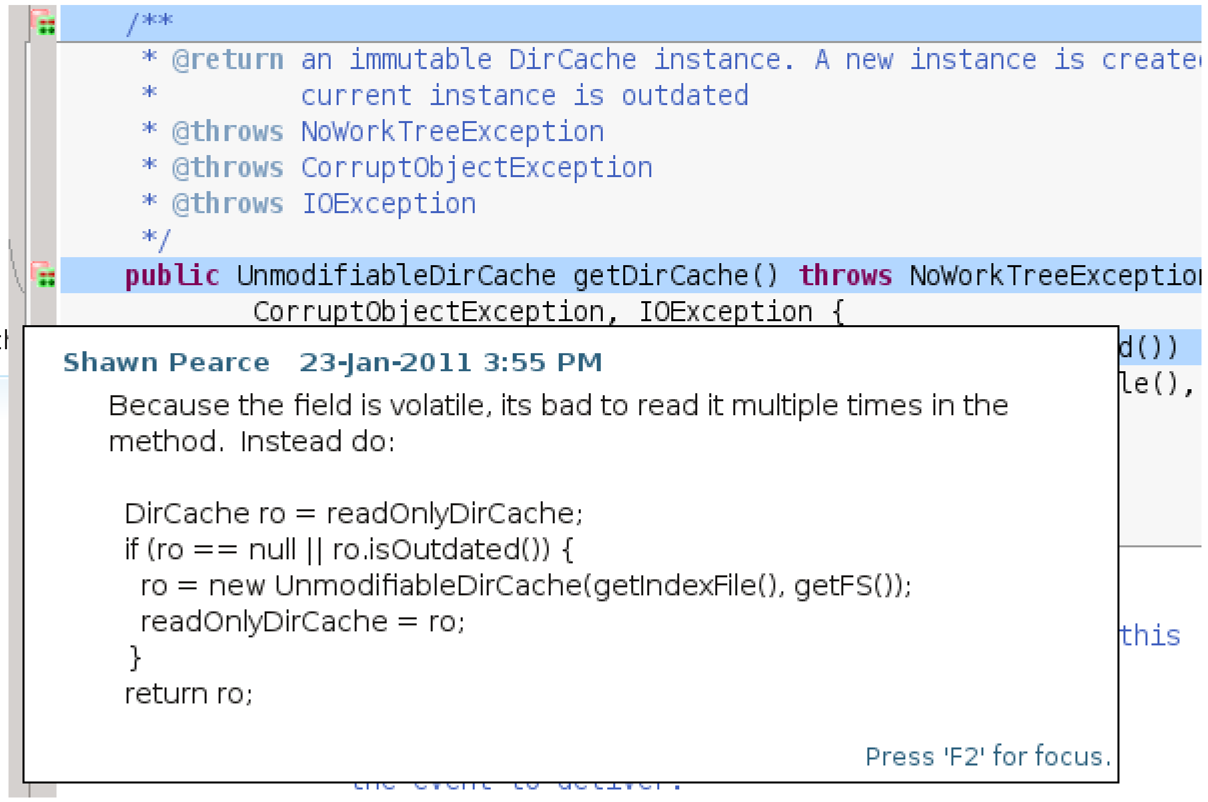
Verified	+1 Build Successful
Code Review	+2 Looks good to me, approved
IP Clean	+1 IP review completed

Depends On

I300bfa84: Fix build of JGit source bundle and feature by Matth

Patch Sets

Patch Set 1 35 Comments



```
/**
 * @return an immutable DirCache instance. A new instance is create
 *         current instance is outdated
 * @throws NoWorkTreeException
 * @throws CorruptObjectException
 * @throws IOException
 */
public UnmodifiableDirCache getDirCache() throws NoWorkTreeExceptio
        CorruptObjectException, IOException {
```

Shawn Pearce 23-Jan-2011 3:55 PM

Because the field is volatile, its bad to read it multiple times in the method. Instead do:

```
DirCache ro = readOnlyDirCache;
if (ro == null || ro.isOutdated()) {
  ro = new UnmodifiableDirCache(getIndexFile(), getFS());
  readOnlyDirCache = ro;
}
return ro;
```

Agenda

History of Version Control (VCS)

The Rise of Distributed Version Control (DVCS)

Code Review with Git and Gerrit

Lessons Learned at Eclipse moving to DVCS

- Version control at Eclipse

- Challenges in moving to a DVCS

Conclusion

Q&A

Version Control at Eclipse

Eclipse defined a roadmap to move to Git in 2009

CVS/SVN are deprecated now

Each project is moving to Git on its own timeline...

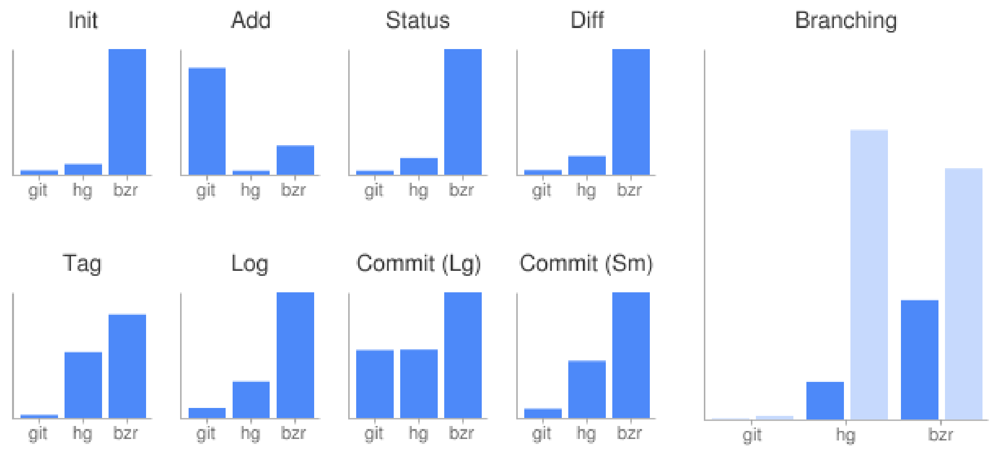
So why did Eclipse.org choose Git?

#1: Git-related projects at Eclipse.org

… both the core Git library (JGit) and tooling (EGit) are actively developed at Eclipse.org by a diverse set of committers and contributors with a common goal

#2: Git is fast

… Git is fast and scales well



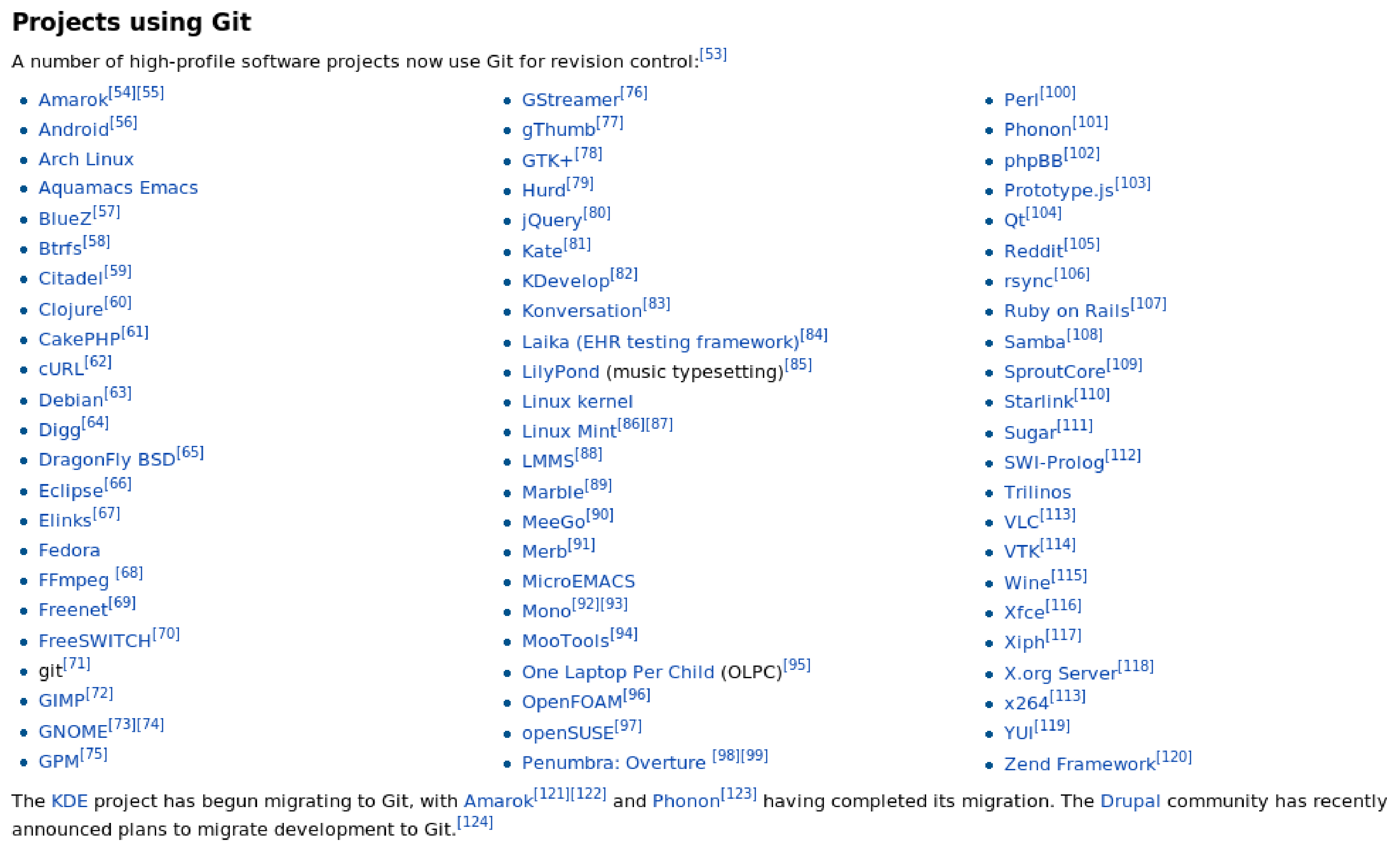
The end result was that for everything but adding new files, Git was fastest. (Also really large commits, which Hg was basically the same at, but the commit I tested was so large that you're unlikely to ever do anything like it—normal commits are much faster in Git.)

	Git	**Hg**	**Bzr**
Init	0.024s	0.059s	0.600s
Add	8.535s	0.368s	2.381s
Status	0.451s	1.946s	14.744s
Diff	0.543s	2.189s	14.248s
Tag	0.056s	1.201s	1.892s
Log	0.711s	2.650s	9.055s
Commit (Large)	12.480s	12.500s	23.002s
Commit (Small)	0.086s	0.517s	1.139s
Branch (Cold)	1.161s	94.681s	82.249s
Branch (Hot)	0.070s	12.300s	39.411s

*whyisgitbetterthanx.com

#3: Git is mature and popular

… Git is widely used and is the most popular distributed version control system



Projects using Git

A number of high-profile software projects now use Git for revision control:[53]

- Amarok[54][55]
- Android[56]
- Arch Linux
- Aquamacs Emacs
- BlueZ[57]
- Btrfs[58]
- Citadel[59]
- Clojure[60]
- CakePHP[61]
- cURL[62]
- Debian[63]
- Digg[64]
- DragonFly BSD[65]
- Eclipse[66]
- Elinks[67]
- Fedora
- FFmpeg [68]
- Freenet[69]
- FreeSWITCH[70]
- git[71]
- GIMP[72]
- GNOME[73][74]
- GPM[75]
- GStreamer[76]
- gThumb[77]
- GTK+[78]
- Hurd[79]
- jQuery[80]
- Kate[81]
- KDevelop[82]
- Konversation[83]
- Laika (EHR testing framework)[84]
- LilyPond (music typesetting)[85]
- Linux kernel
- Linux Mint[86][87]
- LMMS[88]
- Marble[89]
- MeeGo[90]
- Merb[91]
- MicroEMACS
- Mono[92][93]
- MooTools[94]
- One Laptop Per Child (OLPC)[95]
- OpenFOAM[96]
- openSUSE[97]
- Penumbra: Overture [98][99]
- Perl[100]
- Phonon[101]
- phpBB[102]
- Prototype.js[103]
- Qt[104]
- Reddit[105]
- rsync[106]
- Ruby on Rails[107]
- Samba[108]
- SproutCore[109]
- Starlink[110]
- Sugar[111]
- SWI-Prolog[112]
- Trilinos
- VLC[113]
- VTK[114]
- Wine[115]
- Xfce[116]
- Xiph[117]
- X.org Server[118]
- x264[113]
- YUI[119]
- Zend Framework[120]

The KDE project has begun migrating to Git, with Amarok[121][122] and Phonon[123] having completed its migration. The Drupal community has recently announced plans to migrate development to Git.[124]

#4: Git community tools

… interested in taking advantage of such Git tools like Gerrit Code Review (used by the Android community) and GitHub

Eclipse.org: Challenges moving to a DVCS

Convincing management and peers was tough

- At first, everyone is resistant to change

The learning curve of DVCS systems is high

- Initially, the Eclipse tooling was “alpha”
- People refuse to drop to the CLI

Legacy is a pain in the ass!

- 200+ projects at Eclipse used CVS/SVN
- The existing VCS tooling was of high quality

No free lunch!

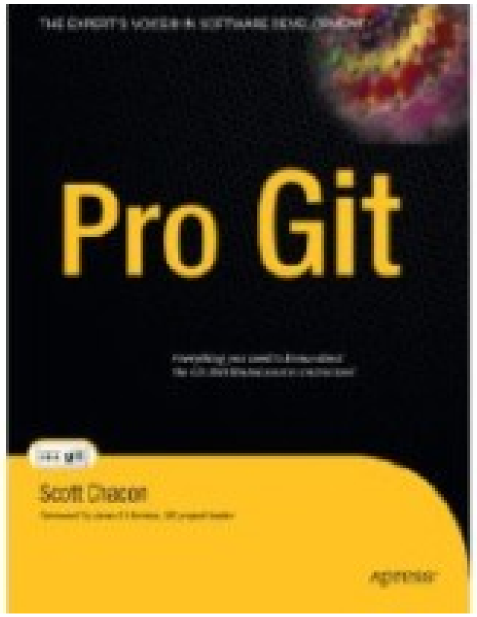
… trust me, the only way to learn DVCS is to start using it... there is a learning curve, you need to rewire your brain!

Git Resources

http://git-scm.com/documentation is your friend

Watch Linus' talk at Google
http://www.youtube.com/watch?v=4XpnKHJAok8

Read the Pro Git book - http://progit.org/book/



Agenda

History of Version Control (VCS)

The Rise of Distributed Version Control (DVCS)

Code Review with Git and Gerrit

Lessons Learned at Eclipse moving to a DVCS

Conclusion

Q&A

Conclusion

The future of version control is distributed!

Moving to a DVCS takes time, worth the effort

Gerrit enables a nice code review workflow for Git

Social coding systems like GitHub rock

Q&A

http://eclipse.org/egit
http://wiki.eclipse.org/EGit/GitHub
http://eclipse.org/reviews

Chris Aniszczyk (Red Hat)
zx@redhat.com
http://aniszczyk.org
twitter.com/caniszczyk