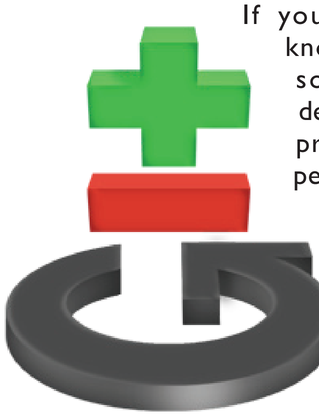
Git Going with Distributed Version Control

by Matthew J. McCullough

"Centralized" is the Cobol of Version Control Systems.

The Word is Getting Around



If you circulate in a group of knowledge-hungry developers, socialize at local software development user groups, attend programming conferences, or peruse just one leading coding-news web site, it would be a safe bet to say that the topic of distributed version control systems (DVCS) has been brought to your attention. But as with any water-cooler-worthy topic, there is a vast chasm between "cool technology" and something that can be leveraged for productivity gains in your everyday work environment. I will introduce you to and support the argument that DVCS-es, and Git in particular, have successfully reached a level of maturity and acceptance that demand your review of their capabilities for use in your next project. The feature set of Git is so rich, the productivity gains so significant, and alignment with agile team goals so congruent that you will wonder how you ever survived using a centralized version control system.

Born of Anger

Many great pieces of software are written in reaction to a dramatic event. Angst provides an unmatched level of adrenaline and motivation to outperform the catalytic offender. In the case of Git, it was birthed from just days of Linus Torvald's sleep-deprived hacking after the well-publicized BitKeeper debacle. There was a vitriolic disagreement about the licensing of BitKeeper, the VCS used for the Linux kernel, and on April 3rd, 2005, Linus' work on Git began. Four days later it was self-hosted. Eleven days after that, it was capable of merging multiple branches. And after just 59 more days, the Linux kernel issued its first release from the Git platform. Linus, referring to this timeframe during a Google campus presentation, less than modestly said, "I decided I can write something better than anything out there in two weeks, and I was right."

Approachable, Yet Deep

Now that we've had a quick history lesson on Git's origins, I'll win your attention to read the remainder of this article by a quick demonstration of using this radical VCS tool. "But wait, I'm not on Linux!" you exclaim.

Platforms

In the early days, a competitor in the DVCS space named Mercurial, authored in Python, kidnapped new DVCS users into their camp claiming a better cross-platform experience. This was true for a while, but in 2008 that premise no longer rang true. Significant efforts to bring Git to the other widely used development platforms yielded a bounty of actively maintained ports. Today, there are solid and up to date binary distributions of Git on Linux, Mac OSX, Solaris and Windows, just to name the principle OSes. In a move that you are welcome to replicate, I bolstered my geek standing with the technical crowd by wrangling Git to run natively on my iPhone. That is evidence of Git's expansive platform support, visible in the very palm of your hand.

Simple Setup

Returning to the mechanics of getting Git up and running on your development machine, you'll need only three minutes to spare:

```
# Step 1
#Download & install the binaries (for some
# distros, just copy onto your path):
http://git-scm.com/download


# Step 2
# Setup your system-wide identity
# for checkins
git config --global user.name 'Matthew
McCullough'
git config --global user.email
matthewm@ambientideas.com

# Step 3 (optional)
# Enable syntax highlighting in Git
# console output
git config --global color.ui true
```

We're done. Really. There's no background daemon, no Apache server, no mod_dav_svn, and no execution of the svnadmin command. You are now ready to create your first Git repository.

```
# Step A1
#Create a folder to contain an
# empty repository
mkdir myproject

# or Step A2
#CD to the new directory or CD to an
# existing folder you want to put under
# version control
cd myproject

# Step B
#Initialize the Git repository
#(a simple ".git" folder + files)
git init
# Step C
#Edit some source files
echo //Code, code, and more
code >> HelloWorld.java

# Step D
#Recursively add all source files to the
# staging area and check them in.
git add .
git commit -m'The initial
checkin of project files'
```

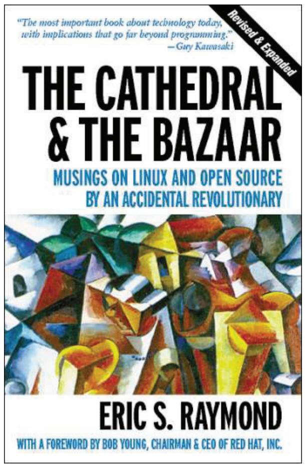
In just several more keystrokes, we've built a git repository and put all its contents under version control. I challenge you to find an equally rich SCM tool that has fewer setup steps.

Layered Plumbing and Porcelain

The getting-started steps allowed you to dip your toe into the water of DVCS with just three commands, but there is an ocean of capabilities and commands awaiting your beck and call. Git, in the purest form of a properly abstracted system, has a core of 150 binaries that comprise all the intricate operations that can be performed on a repository. Java programmers can think of these as protected base-class methods; Git-heads call this the "plumbing." The aggregation of that fine-grained power is termed the "porcelain" and takes the form of around 20 frequently used commands. This modular design can be leveraged to literally build your own versioned file system inside an application. Such an innovative use is already taking place by an automatic document versioning tool for writers called FlashBake; For developers, Gerrit, an interactive code-review tool that leverages Git plumbing is gaining traction even beyond the borders of its high-profile sponsor, Google.

The Untethered Bazaar

I am often telling colleagues that Git fosters the "crowd-source" approaches of the famous Eric S. Raymond essay, "The Cathedral and the Bazaar." If you haven't read it, don't worry, I'll wait while you give it a review; I've linked to it in the references section of this article. In sum, Raymond's dissertation says that having many contributors on a project and low barriers to participation increases its overall health from the aspect of bug reduction and feature contributions. Though Raymond's case study was focused on open source projects, these same benefits can be found in applying a very transparent and empowering source code control tool like Git to a commercial closed-source project. Teams find that Git, like a good employee, is a source control tool that silently works for them. This is a refreshing contrast to developers extolling a feeling of subservience to the intricate checkin rules and rituals of many traditional version control systems;This is the essence of an agile process.


"The most important book about technology today, with implications that go far beyond programming." —Guy Kawasaki
Revised & Expanded
THE CATHEDRAL & THE BAZAAR
MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY
ERIC S. RAYMOND
WITH A FOREWORD BY BOB YOUNG, CHAIRMAN & CEO OF RED HAT, INC.

Desirable Disconnectedness

Git fulfills the "bazaar" promise by offering options to fit the developer at almost every turn. The most important of those options, and the one that we will touch on in this article, is in the area of connectivity. Nearly every Git command operates on a repository on disk, forsaking the requirement of connectivity for the range of commands from "log" through "diff" to "commit" and "branch". You might ask, "Why in this time of ever-more-prevalent connections, is operating in a disconnected fashion a positive trait?" The answer is three-fold. First, the performance gains of running intense operations of viewing repository history and diff-ing arbitrary versions locally is undeniable. The results are on screen as fast as your disk can serve them to the CPU. Second, many developers are using time-boxing techniques such as Pomodoro and Getting Things Done (GTD) to self-impose Internet radio-silence time periods. Third, many developers are finding themselves working in airplanes, cars, and locations that are out of reach of reliable connectivity.

The Sneakernet Is Back

A feature unique to Git is the ability to capture binary-compressed segments of the repository between two points in time to a single portable file. These deltas can then be transported by any means available -- email, FTP or memory stick -- and used to "catch up" repository clones that live inside firewalls or on machines that literally never have network connectivity. These are not your grandfather's patch-files; These are binary segments of a repository that contain branch, revision, merge, move and deletion information.

Models of Use

Progressive Adoption

An insightful observation by one developer was that "Git doesn't force you into a new way of working, it just progressively rewards you for leveraging its new approach." Another acquaintance had a similar observation: "The strength of Git is found in the duality of its support for the 'the old way' of working while its tutorials sing a siren-like song to lure you to its radical new features." I'm irrevocably jaded. I nondexpecf all my developer tools to encourage me to migrate to using their new featureh by means of productivind incentives; In short, a carrot instead of a stick.

Subversion Silently Subverted

Git offers a rich round trip experience to Subversion repositories. Git knows the language of, and thus can "clone" a Subversion repository, which is a thorough one-time troll through every historical revision of a repository, building up a local Git equivalent. Once "git svn clone"-ed, your newfound Git capabilities such as network-disconnected history review, arbitrary version diffs, merging and branching can all be performed on your favorite Subversion project via Git. And then, in what might seem a pure wave of a magic wand, Git can transactionally "replay" mutable offline Git operations back to the Subversion repository, bringing your offline operations back into the Subversion repository for consumption by your colleagues of the non-Git persuasion. Depending on your level of deviousness, you can silently use Git in this mode for months against a Subversion repository and yield about 70% of its benefits.* Not that I'm admitting to having ever done this...but the experience is simply splendid.

Centralized, But Less Strongly

When first moving to Git, you might find yourself using it in a Subversion-like manner. That's perfectly ok. After a few days or weeks though, depending on your nature, you'll grow dissatisfied with the remnants of the concept of a central repository that all changes must be 'git push'-ed to. You'll begin to yearn to transmit changes on-the-sly to a colleague for confidential review and improvement. You'll start itching to branch "just to try an idea out." And I promise you'll begin relying on having the entire history of the repository local and at your fingertips to provide mental context of "why something is the way it is." These Git features will become springboards for your next coding effort; Your Git-ification has officially begun.

Bargain Branching

One of the features that makes Git a perfect fit for agile teams or solo developers wanting to leverage agile techniques is instantaneous branching that doesn't pollute the global namespace. You can create a branch in just milliseconds with no network interaction.

```
#Create the branch and check it out (start
#working on it)
git checkout -b storycardbranch015
```

* Git also offers a great compatibility experience for CVS users. While round trip features are offered for Git and CVS, it is commonly recommended to do a one-way conversion with CVS due to some field-reports of Git to CVS issues.

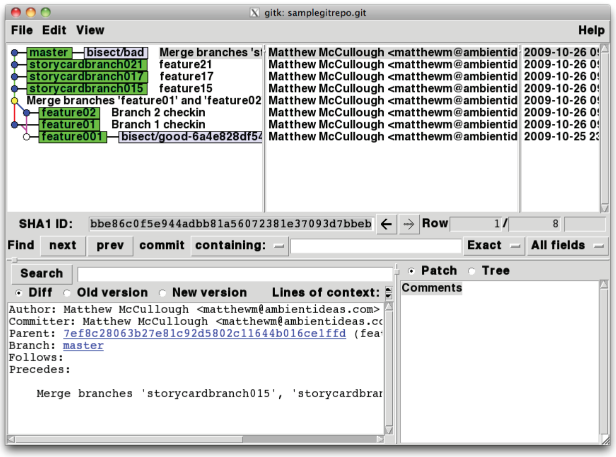
If the branch gains some useful code via your coding session and git commits, share it with colleagues:

```
#Share the branch with an associate
git push colleaguerepo storycardbranch015
```

But if the effort turns out to be a dead end, no one needs to know.

```
#Delete the less-than-useful branch
git branch -D storycardbranch015
```

With branching being such a quick and easy operation to reach for, you'll find, as many Git users do, creating a branch for each "concept" or story card becomes a natural approach to development. Git's powerful support for merging makes pulling an individual feature into a given branch an option at any point in time. Recurring releases, in the true spirit of agility, can easily call-in or scope-out a feature based on its ready-ness at almost any point in the sprint



```
master  bisect/bad  Merge branches 'st  Matthew McCullough <matthewm@ambientid  2009-10-26 0
storycardbranch021  feature21           Matthew McCullough <matthewm@ambientid  2009-10-26 0
storycardbranch017  feature17           Matthew McCullough <matthewm@ambientid  2009-10-26 0
storycardbranch015  feature15           Matthew McCullough <matthewm@ambientid  2009-10-26 0
Merge branches 'feature01' and 'feature02  Matthew McCullough <matthewm@ambientid  2009-10-26 0
feature02  Branch 2 checkin             Matthew McCullough <matthewm@ambientid  2009-10-26 0
feature01  Branch 1 checkin             Matthew McCullough <matthewm@ambientid  2009-10-26 0
feature001  bisect/good-6a4e828df54     Matthew McCullough <matthewm@ambientid  2009-10-25 23

SHA1 ID: bbe86c0f5e944adbb81a56072381e37093d7bbeb   Row 1 / 8

Author: Matthew McCullough <matthewm@ambientideas.com>
Committer: Matthew McCullough <matthewm@ambientideas.cc
Parent: 7ef8c28063b27e81c92d5802c11644b016ce1ffd (fea
Branch: master
Follows:
Precedes:

    Merge branches 'storycardbranch015', 'storycardbrar
```

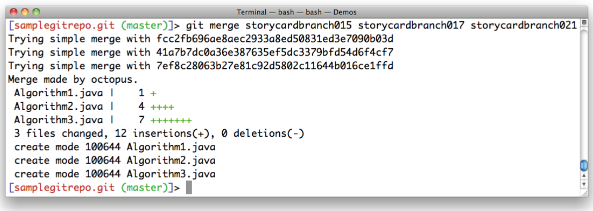
Mild Mannered Merging

Many SCMs elevate merging of branches to a special ceremony, as equally involved as the Norse rituals of spring. Git brushes aside this accidental complexity with the position that if branching is easy, then merging has to be quick to balance the picture.

```
#Merge a feature branch into the mainline
git checkout master
git merge storycardbranch015
```

But what if we have multiple features? It's just as easy

```
#Merge three branches into the current
#branch
git merge storycardbranch015
storycardbranch017 storycardbranch021
```

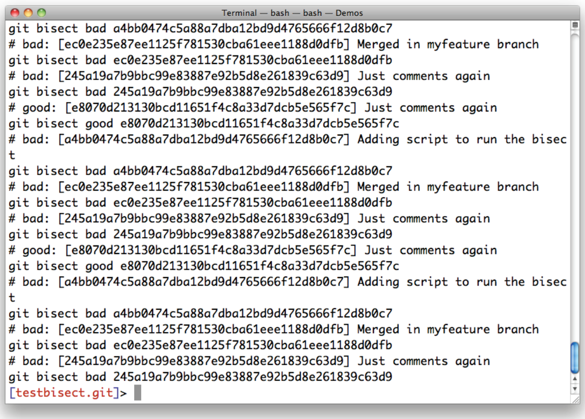


```
[samplegitrepo.git (master)]> git merge storycardbranch015 storycardbranch017 storycardbranch021
Trying simple merge with fcc2fb696ae8aec2933a8ed50831ed3e7090b03d
Trying simple merge with 41a7b7dc0a36e387635ef5dc3379bfd54d6f4cf7
Trying simple merge with 7ef8c28063b27e81c92d5802c11644b016ce1ffd
Merge made by octopus.
 Algorithm1.java |    1 +
 Algorithm2.java |    4 ++++
 Algorithm3.java |    7 +++++++
 3 files changed, 12 insertions(+), 0 deletions(-)
 create mode 100644 Algorithm1.java
 create mode 100644 Algorithm2.java
 create mode 100644 Algorithm3.java
[samplegitrepo.git (master)]> 
```

Unit Test Support Leaked Into In My SCM

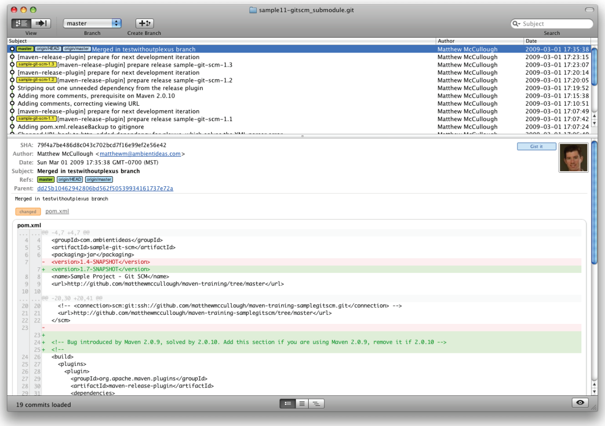
To wrap up some ways in which Git changes your approach to development, 'git bisect' deserves a very honorable mention. Whether you strive for significant test coverage or go all the way with Test Driven Design (TDD), this feature of Git will multiply the value of your unit tests. Bisect is a binary search that progressively checks out each revision of code in the specified range, runs your battery of unit tests, records the success or failure of the tests against each tested "commit" transaction, and presents it in a log-result style format. Voila, the checkin that caused the test(s) to start failing has been specifically identified.

```
#Start the binary search (test executions)
# between a failing (current) and
# successful (six checkins ago)
# range of revisions
git bisect start HEAD HEAD^^^^^

#Tell Git what command (Ant, Maven, Rake,
# make, etc.)
# executes your tests and provides
# a success/fail return code.
git run mvn tet

#Visualize the result as a log
git bisect lo
```



```
git bisect bad a4bb0474c5a88a7dba12bd9d4765666f12d8b0c7
# bad: [ec0e235e87ee1125f781530cba61eee1188d0dfb] Merged in myfeature branch
git bisect bad ec0e235e87ee1125f781530cba61eee1188d0dfb
# bad: [245a19a7b9bbc99e83887e92b5d8e261839c63d9] Just comments again
git bisect bad 245a19a7b9bbc99e83887e92b5d8e261839c63d9
# good: [e8070d213130bcd11651f4c8a33d7dcb5e565f7c] Just comments again
git bisect good e8070d213130bcd11651f4c8a33d7dcb5e565f7c
# bad: [a4bb0474c5a88a7dba12bd9d4765666f12d8b0c7] Adding script to run the bisec
t
git bisect bad a4bb0474c5a88a7dba12bd9d4765666f12d8b0c7
# bad: [ec0e235e87ee1125f781530cba61eee1188d0dfb] Merged in myfeature branch
git bisect bad ec0e235e87ee1125f781530cba61eee1188d0dfb
# bad: [245a19a7b9bbc99e83887e92b5d8e261839c63d9] Just comments again
git bisect bad 245a19a7b9bbc99e83887e92b5d8e261839c63d9
# good: [e8070d213130bcd11651f4c8a33d7dcb5e565f7c] Just comments again
git bisect good e8070d213130bcd11651f4c8a33d7dcb5e565f7c
# bad: [a4bb0474c5a88a7dba12bd9d4765666f12d8b0c7] Adding script to run the bisec
t
git bisect bad a4bb0474c5a88a7dba12bd9d4765666f12d8b0c7
# bad: [ec0e235e87ee1125f781530cba61eee1188d0dfb] Merged in myfeature branch
git bisect bad ec0e235e87ee1125f781530cba61eee1188d0dfb
# bad: [245a19a7b9bbc99e83887e92b5d8e261839c63d9] Just comments again
git bisect bad 245a19a7b9bbc99e83887e92b5d8e261839c63d9
[testbisect.git]> 
```

What was a tedious manual task is now fully automated thanks to Git. Identifying exactly which checkin broke the unit test suite across any small or vast expanse of time just became trivial. I've found that merely demonstrating this surgically precise identification capability in a lunch-and-learn session bolsters a desire to never break tests across a development team, thus relegating the need for its use to rare occasions.



```
Subject                                                                   Author              Date
master origin/HEAD origin/master Merged in testwithoutplexus branch        Matthew McCullough  2009-03-01 17:35:38
[maven-release-plugin] prepare for next development iteration             Matthew McCullough  2009-03-01 17:23:15
sample-git-scm-1.3 [maven-release-plugin] prepare release sample-git-scm-1.3  Matthew McCullough  2009-03-01 17:23:07
[maven-release-plugin] prepare for next development iteration             Matthew McCullough  2009-03-01 17:20:14
sample-git-scm-1.2 [maven-release-plugin] prepare release sample-git-scm-1.2  Matthew McCullough  2009-03-01 17:20:05
Stripping out one unneeded dependency from the release plugin             Matthew McCullough  2009-03-01 17:19:52
Adding more comments, prerequisite on Maven 2.0.10                        Matthew McCullough  2009-03-01 17:15:38
Adding comments, correcting viewing URL                                   Matthew McCullough  2009-03-01 17:10:51
[maven-release-plugin] prepare for next development iteration             Matthew McCullough  2009-03-01 17:07:49
sample-git-scm-1.1 [maven-release-plugin] prepare release sample-git-scm-1.1  Matthew McCullough  2009-03-01 17:07:42
Adding pom.xml.releaseBackup to gitignore                                 Matthew McCullough  2009-03-01 17:07:24

    SHA: 79f4a7be486d8c043c702bcd7f16e99ef2e56e42
 Author: Matthew McCullough <matthewm@ambientideas.com>
   Date: Sun Mar 01 2009 17:35:38 GMT-0700 (MST)
Subject: Merged in testwithoutplexus branch
   Refs: master origin/HEAD origin/master
 Parent: dd25b10462942806bd562f50539934161737e72a

Merged in testwithoutplexus branch

changed pom.xml
```

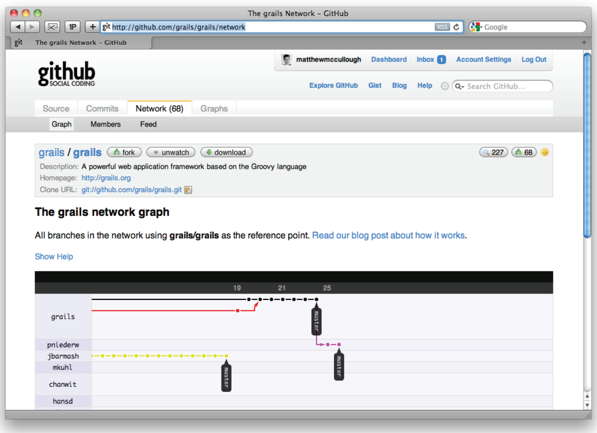
```
pom.xml
@@ -4,7 +4,7 @@
   <groupId>com.ambientideas</groupId>
   <artifactId>sample-git-scm</artifactId>
   <packaging>jar</packaging>
-  <version>1.4-SNAPSHOT</version>
+  <version>1.7-SNAPSHOT</version>
   <name>Sample Project - Git SCM</name>
   <url>http://github.com/matthewmccullough/maven-training/tree/master</url>

@@ -20,30 +20,41 @@
     <!-- <connection>scm:git:ssh://github.com/matthewmccullough/maven-training-samplegitscm.git</connection> -->
     <url>http://github.com/matthewmccullough/maven-training-samplegitscm/tree/master</url>
   </scm>
-
+
+  <!-- Bug introduced by Maven 2.0.9, solved by 2.0.10. Add this section if you are using Maven 2.0.9, remove it if 2.0.10 -->
+  <!--
   <build>
     <plugins>
       <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-release-plugin</artifactId>
         <dependencies>
```

A Full Plate of Supporting Tools

The maturity of the tools supporting Git has expanded faster than a universe without a Higgs boson particle. Seemingly overnight, Eclipse gained ownership of and added sanctioned support of the eGit plugin, IntelliJ IDEA added official Git integration and UI symbolics, NetBeans gained the NBGit module, TextMate has a keyboard-driven bundle, and OS-specific UIs are popping onto the scene via Slashdot more frequently than I can track. Getting onto Git is now a near seamless transition for the most hardened mouse or keyboard-loving developer.

Who's On Board With Git?

While you are likely convinced to leverage Git on your next project, looking at the trending behavior of the community is a good validation of your decision. You'll find your conclusion to use Git fully substantiated. Many open source hosting firms are quickly bringing DVCS to their menu of services based on public demand, and a majority of those are making Git their DVCS of choice based on the rich feature set and significant volume of public adoption. Examples of Git hosting include Sun's OSS community, Kenai.com, the venerable Sourceforge.com, Unfuddle.com, and Gitorious.org just to name a few



grails / grails

Description: A powerful web application framework based on the Groovy language

Homepage: http://grails.org

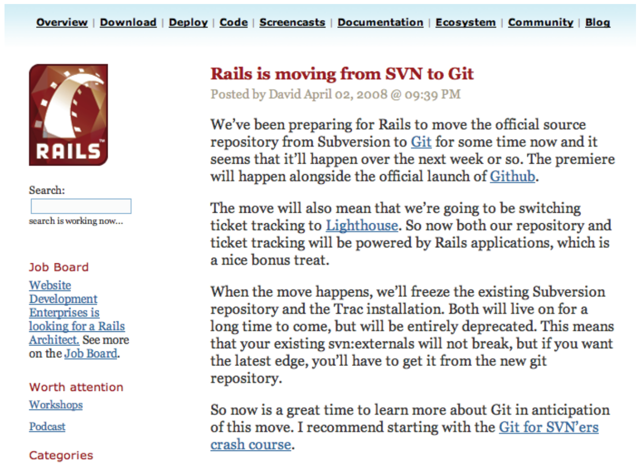
Clone URL: git://github.com/grails/grails.git

The grails network graph

All branches in the network using **grails/grails** as the reference point. Read our blog post about how it works.

Show Help

It is equally emboldening to prattle off the list of projects that are now based on the Git DVCS for their source code control: Grails, Perl, Android, Linux, MooTools, YUI, and Rails just to enumerate the tip of the iceberg of émigrés. When you switch to using Git, you will be in the very best of compay


Overview | Download | Deploy | Code | Screencasts | Documentation | Ecosystem | Community | Blog
RAILS
Search:
search is working now...
Job Board
Worth attention
Workshops
Podcast
Categories

Website Development Enterprises is looking for a Rails Architect. See more on the Job Board.

Rails is moving from SVN to Git

Posted by David April 02, 2008 @ 09:39 PM

We've been preparing for Rails to move the official source repository from Subversion to Git for some time now and it seems that it'll happen over the next week or so. The premiere will happen alongside the official launch of Github.

The move will also mean that we're going to be switching ticket tracking to Lighthouse. So now both our repository and ticket tracking will be powered by Rails applications, which is a nice bonus treat.

When the move happens, we'll freeze the existing Subversion repository and the Trac installation. Both will live on for a long time to come, but will be entirely deprecated. This means that your existing svn:externals will not break, but if you want the latest edge, you'll have to get it from the new git repository.

So now is a great time to learn more about Git in anticipation of this move. I recommend starting with the Git for SVN'ers crash course.

Closing with a vitriolic thought from the gentleman who started it all:

"[In version control systems,] if you aren't distributed, you aren't worth using."

-Linus Torvalds