CollabNet

Git/Gerrit with TeamForge

Secure, Scalable, Standards-Compliant for the Enterprise

Johannes Nicolai

Director of Engineering CollabNet

CollabNet

Agenda

- Introduction to Git, Gerrit and Jenkins
- Git/Gerrit's Place in the TeamForge Universe
  - Traceability
  - Searchability
  - Scalability
  - Security
- Gerrit Signature Features
  - History Protection
  - Code Quality Gates
- Q & A

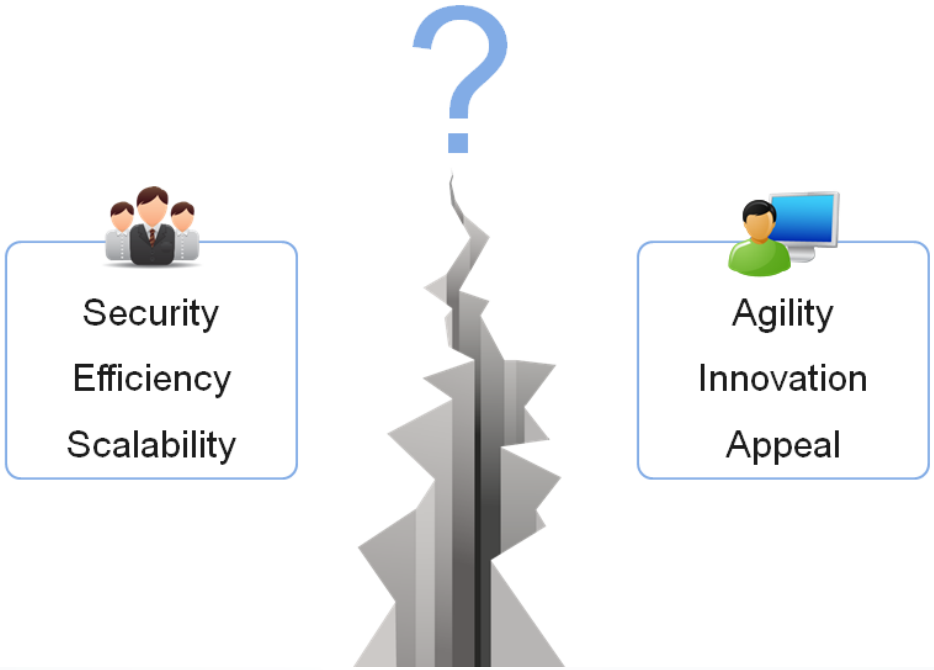
Introduction of Technologies

- Git
  - Most popular emerging distributed SCM
- Jenkins
  - Most popular open source Continuous Integration
- Gerrit Code Review
  - Widely used code-review tool and Git server backend
- CollabNet TeamForge®
  - Provides an unparalleled smooth integration of all these tools into your software development process

Git = Leading DVCS (Distributed Version Control System)

- Open source distributed SCM, used notably by:
  - Linux (it was originally developed for that very purpose)
  - Eclipse foundation
  - Android (based on Gerrit Code Review)
- Why Git?
  - All operations are local
    - Search, log, history, branching and merging
    - Offline development is possible
  - Powerful merging strategies (merge, rebase, cherry-pick)

Git Popularity Continues to Grow, But….



The adoption of DVCS has accelerated in small teams, but is moving more slowly in enterprise settings.

Source: Gartner, 27 July 2011 ID Number: G00214153; Hype Cycle for Application Development, 2011; Ian Finley, Mike Blechar

Gerrit = (Widely Used) Git Server + Code Review Tool

- 100% pure Java SSH and HTTP Git backend
  - Scalable and suitable for large Enterprises
- Web-based and Command Line administration
  - Users and Groups
  - Project and branch security (read and write)
- Collaboration and Code-review
  - B2B integration at code-level
  - Communicate and share code knowledge in the Team
  - Enforce workflow and code quality
- The most successful open source code review
  - Android OS
  - Eclipse, OpenStack, Qt, CouchDB and many others

CollabNet

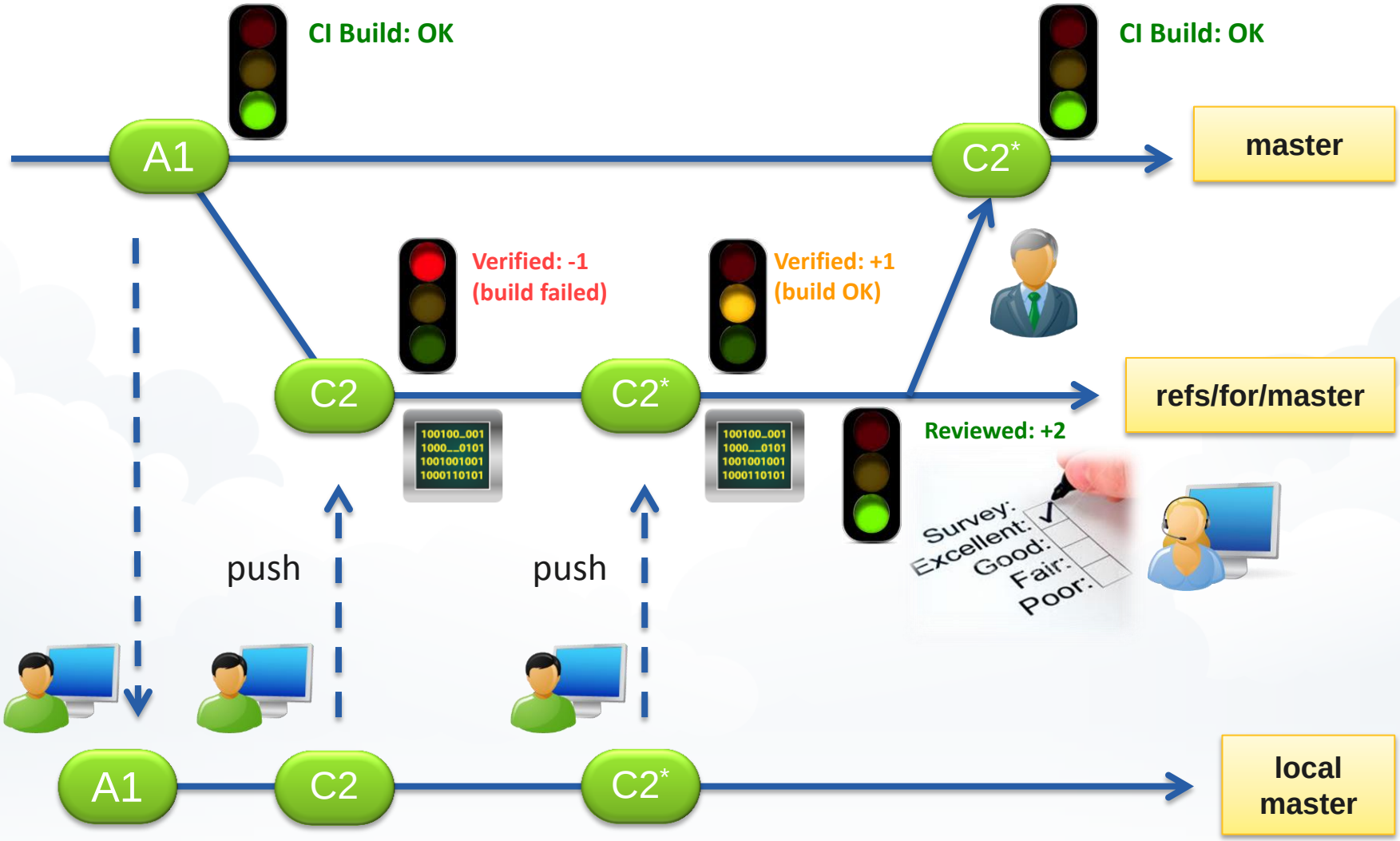
Who's Using Gerrit in Production Today?

source: Wikipedia

Jenkins = Most popular Open Source CI

- Has hundreds of plugins to
  - Interact with various SCMs (Git, SVN, CVS, …)
  - Build software (ant, maven, gradle, make)
  - Run unit/integration tests (JUnit, Selenium, …)
  - Perform static code analysis (findbugs, checkstyle, PMS, Sonar, …)
- Works brilliantly together with Gerrit over Gerrit Trigger Plugin
  - Keeps ssh connection open to learn about new review requests immediatly
  - Builds and verifies all configures quality gates (tests, coding conventions, code KPIs, you name it)
  - Sets ‘verified’ flag in review request according to result of verification build (either +1 or -1)
  - If Jenkins sets -1, then further code review isn’t possible, developer has to either re-work or abandon change

Tackling half baked review requests: Marrying Gerrit and Jenkins

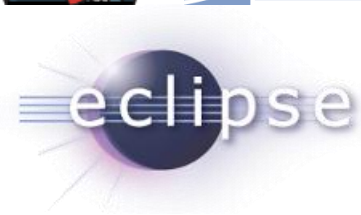


CollabNet

TeamForge = Enterprise-Grade Git Mgmt. + ALM

With TeamForge and Gerrit, Git is now ready for the enterprise. Realize all the benefits of Git, without compromising governance, security and compliance.

TeamForge = Enterprise-Grade Git Mgmt. + ALM



Integration of existing tools

Security, standards and audit compliance

Scalability (and manageability at

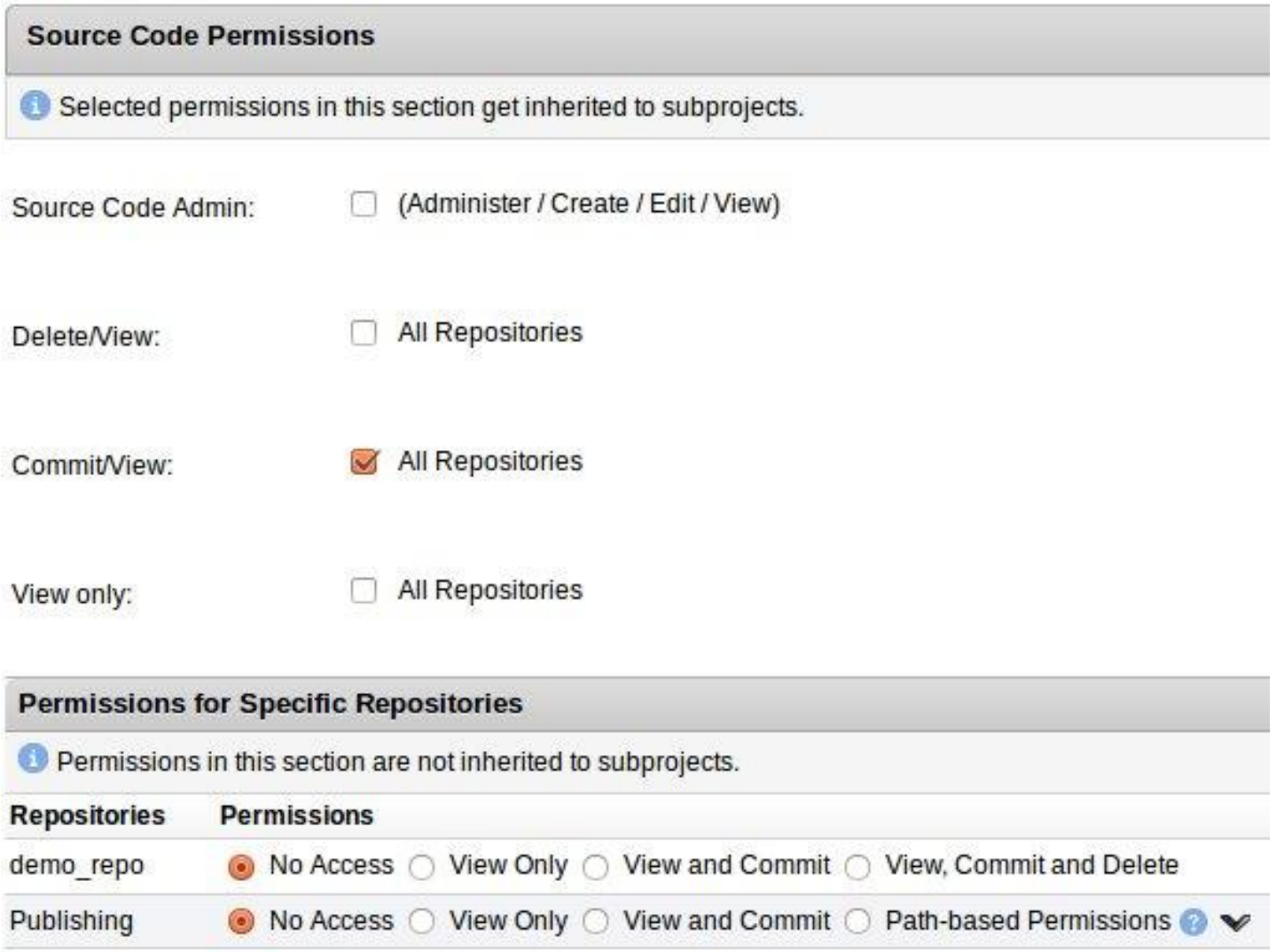
TeamForge = Enterprise-Grade Git Mgmt. + ALM

Searchability – across TF artifacts, including Code Search across multiple repositories, gerrit instances and other SCM tools (svn, cvs, perforce...)

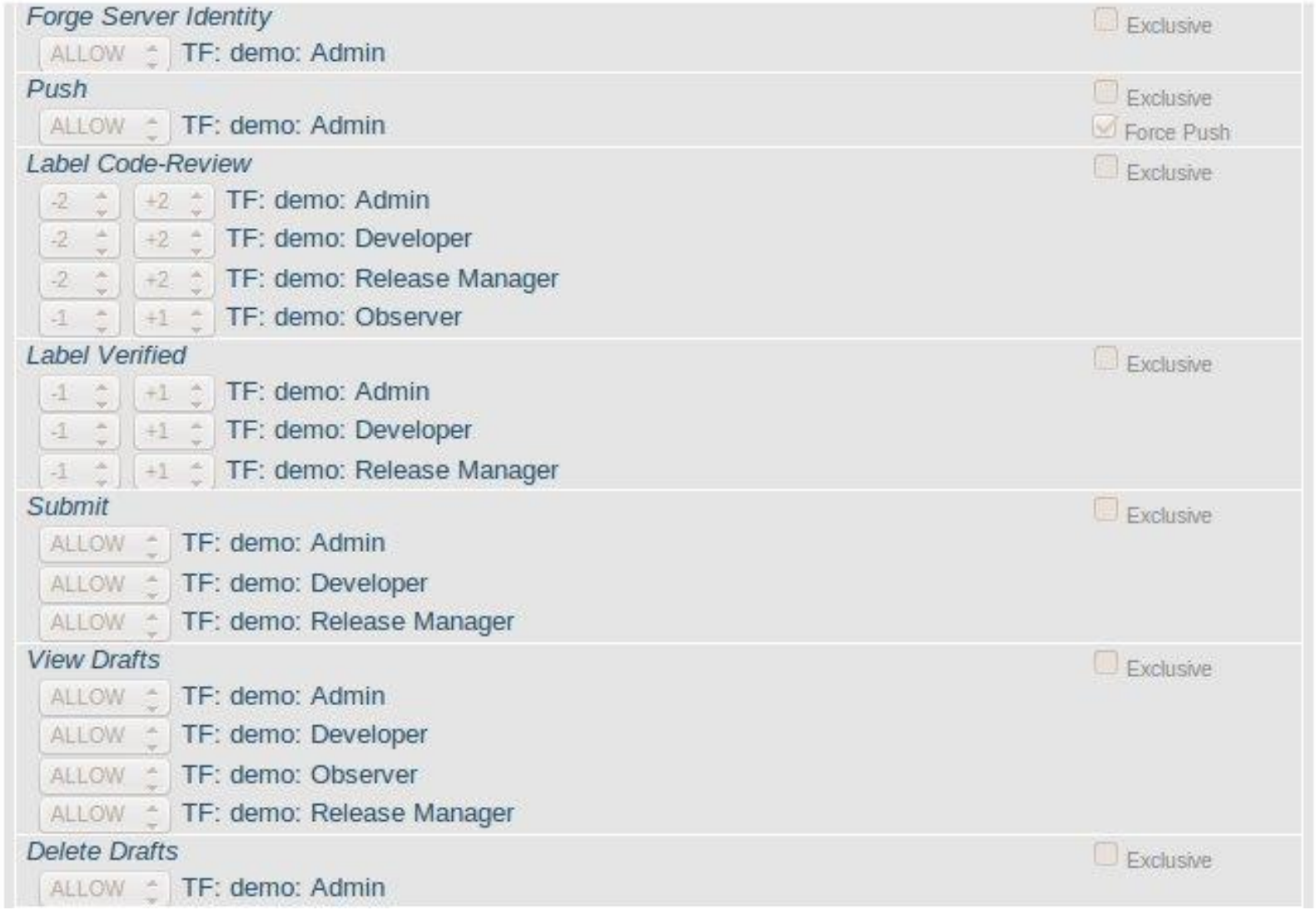
Traceability – associating the source code with requirements, issues, documents, tasks and other artifacts.

RBAC – Role Based Access Control - for each application's concepts, documents, file releases, trackers, and discussion forums, you can assign permissions globally based on user roles

Simple RBAC - TeamForge



Gerrit Acces Rights- Not simple at all



RepoCategories: Bridging the RBAC gap

Motivation:

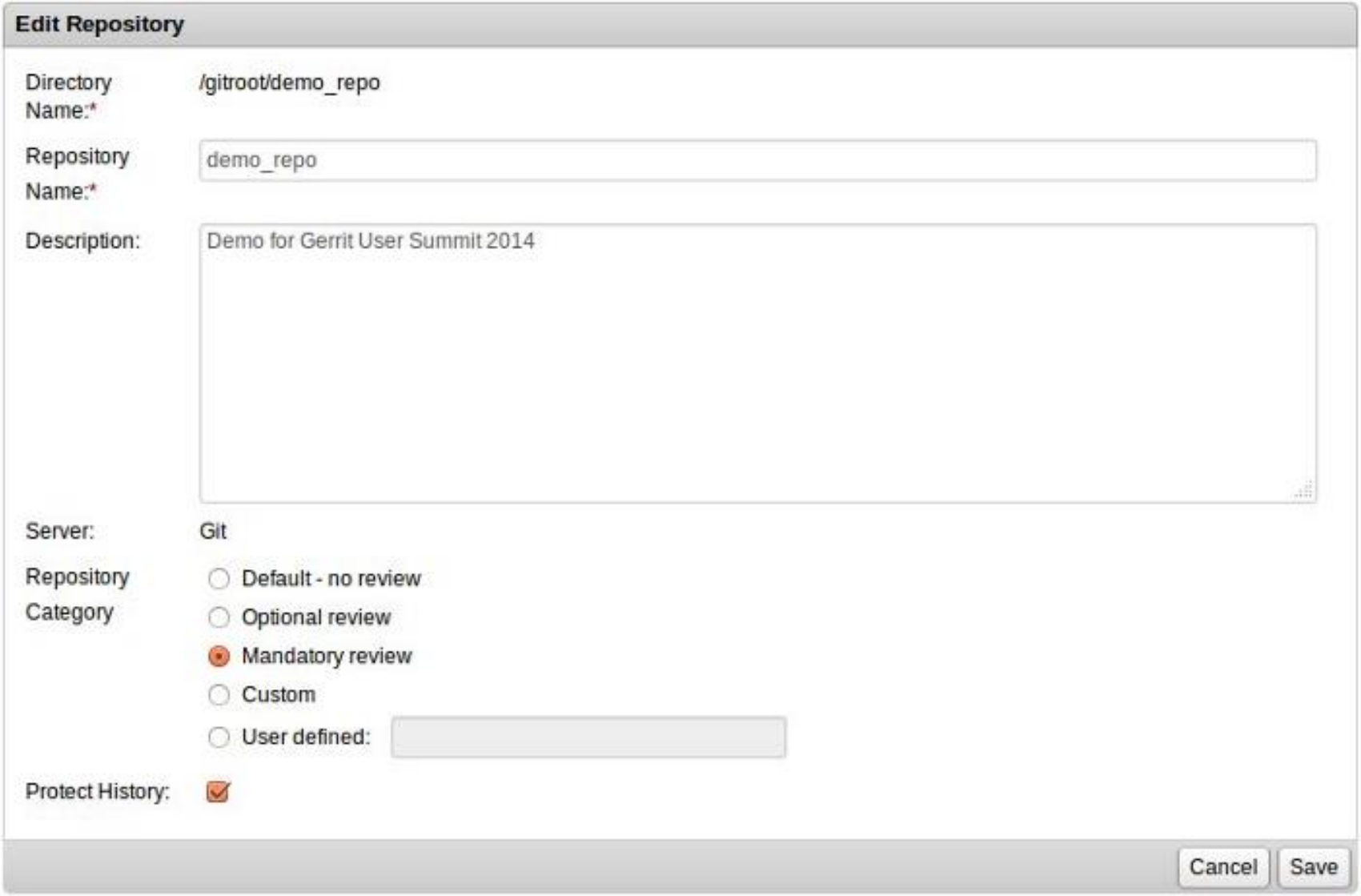
Shield most users from Gerrit's complexities by using pre-defined repo categories and generic TeamForge RBAC model with its SCM permissions.

without hiding advanced features for power users

Implementation:

- Map TeamForge SCM permissions to Gerrit Access Rights.
- Provide pre-defined **code review policies** so anyone can start to work with Gerrit within a few steps.
- **Code review policies** are applied per repository.

RepoCategories: Bridging the RBAC gap



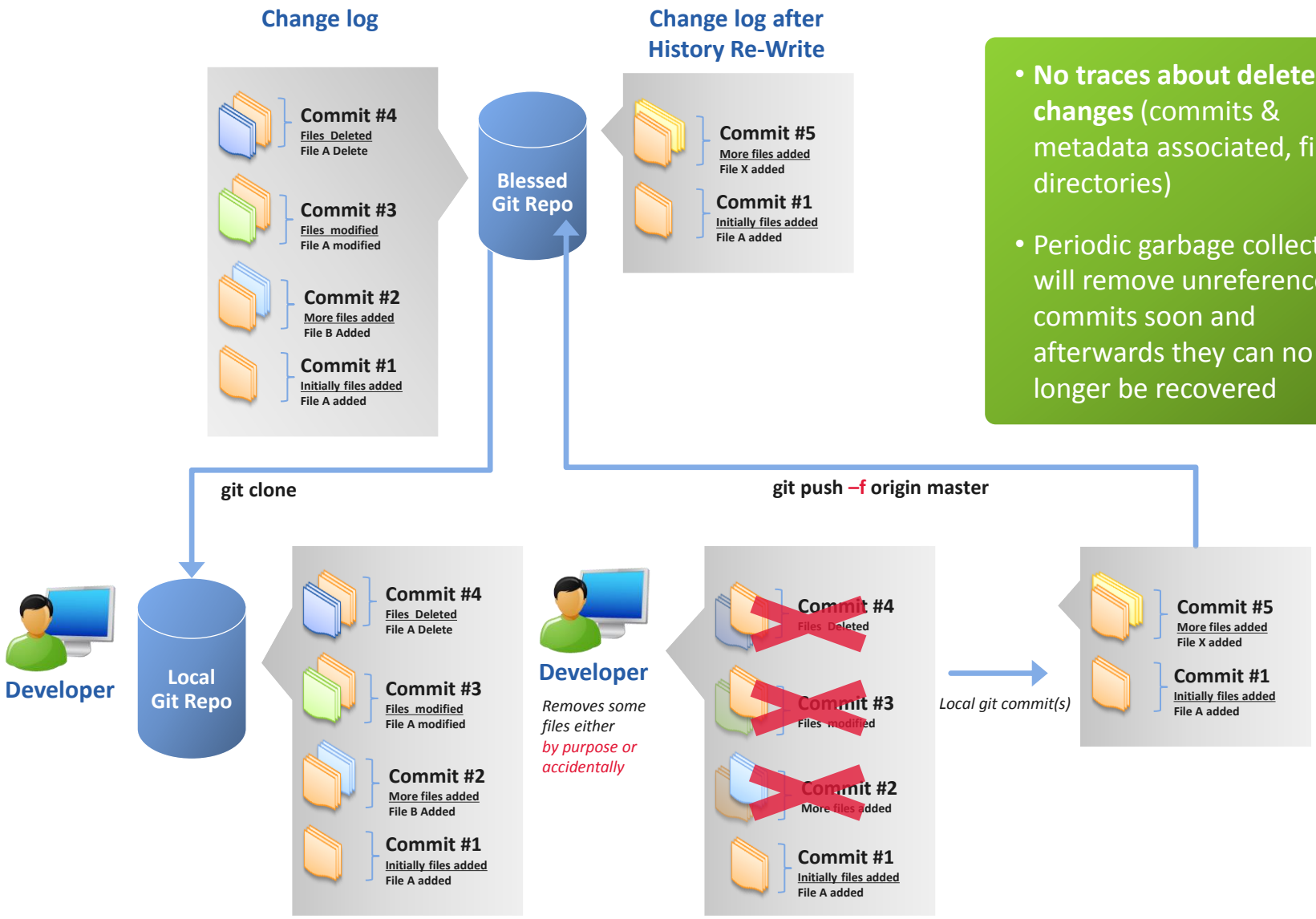
TeamForge = Enterprise-Grade Git Mgmt. + ALM

With TeamForge and Gerrit, Git is now ready for the enterprise. Realize all the benefits of Git, without compromising governance, security and compliance.

Git History Rewrite – Defined

"We define History Rewrite as non-fast forward updates of remote refs and its associated objects. This happens whenever a branch in the remote repository gets deleted, previously pushed commits get amended/tree filtered and forcefully re-pushed, or a remote branch/tag is pointed to an entire different commit history."

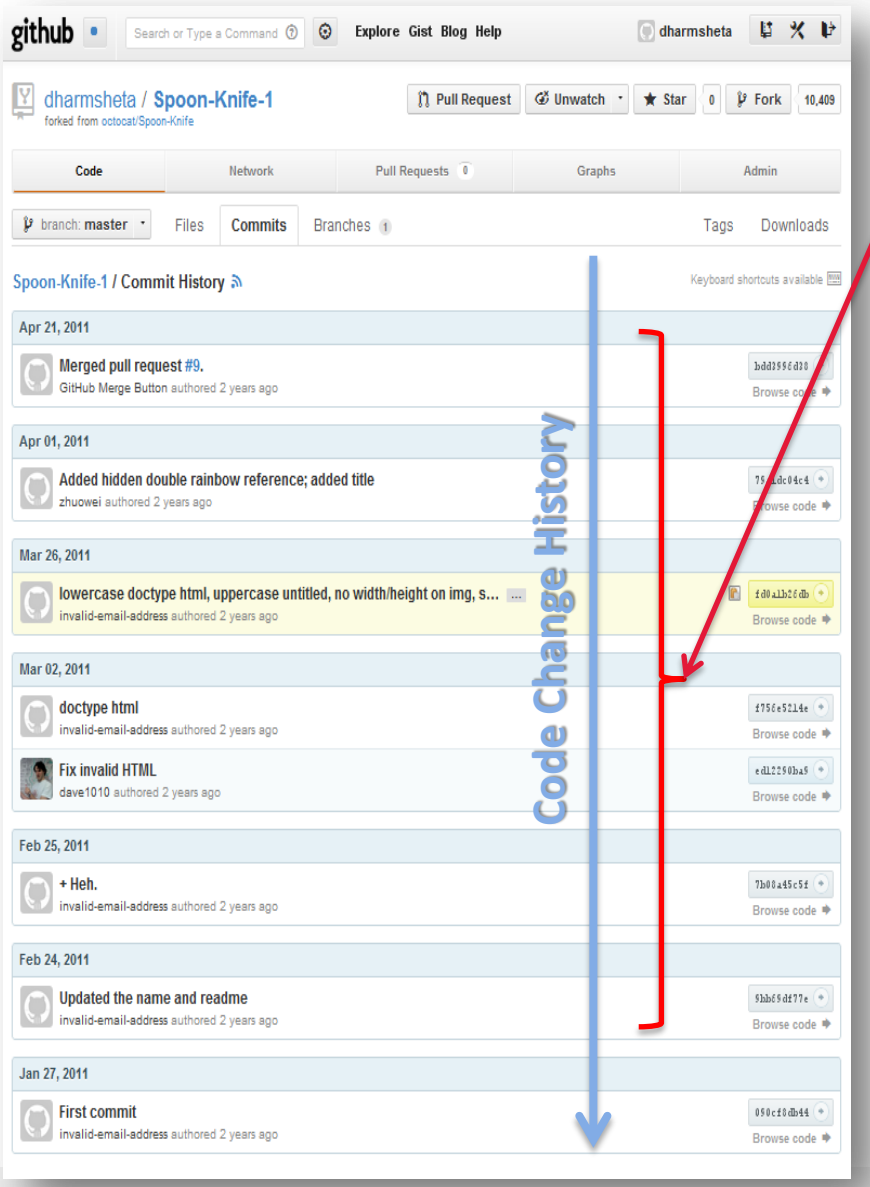
Git History Rewrite – Functionality in Detail



- **No traces about deleted changes** (commits & metadata associated, files, directories)
- Periodic garbage collection will remove unreferenced commits soon and afterwards they can no longer be recovered

CollabNet

Git History Rewrite – No Traces Left Behind!



Removing top 6 commits locally

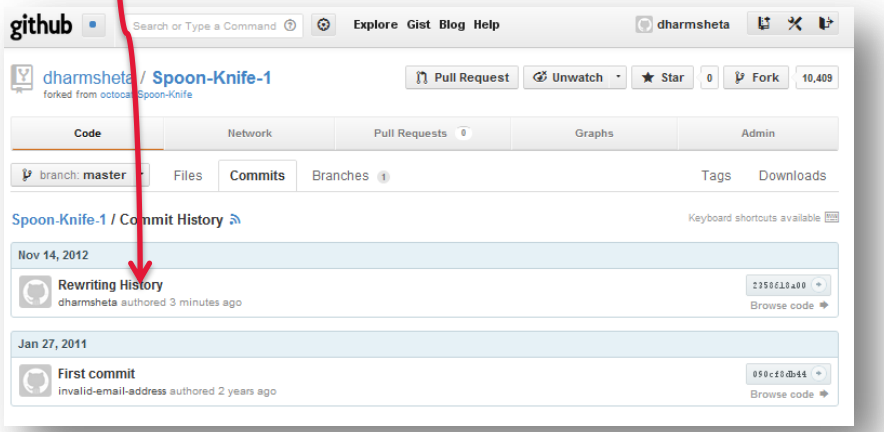
```
sheta@SHETA-THINK ~/Spoon-Knife-1 (master)
$ git reset --hard HEAD~6
HEAD is now at 090cf8d First commit
```

Creating new commit locally

Pushing all changes to remote

```
sheta@SHETA-THINK ~/Spoon-Knife-1 (master)
$ echo "deleted">>history.gone && git add history.gone

sheta@SHETA-THINK ~/Spoon-Knife-1 (master)
$ git commit -m "Rewriting History"
[master 2358618] Rewriting History
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 history.gone

sheta@SHETA-THINK ~/Spoon-Knife-1 (master)
$ git push -f origin master
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 352 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:dharmsheta/Spoon-Knife-1.git
 + bdd3996...2358618 master -> master (forced update)
```

No traces about deleted **6 commits in** Web UI



CollabNet

Git History Rewrite – This Can Happen to You, Too!

Single ‘space’ wipes entire history:

- git push origin production-quick-fix:stable

  instead of

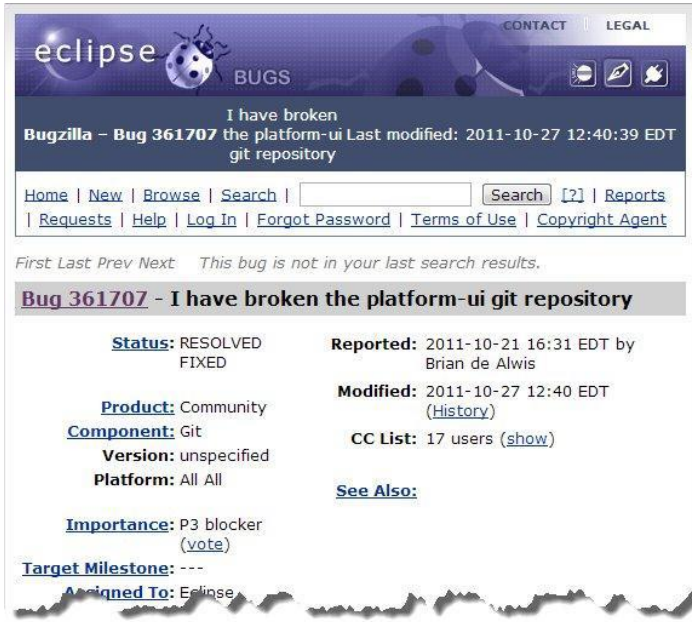
- git push origin production-quick-fix :stable

No traces left behind:

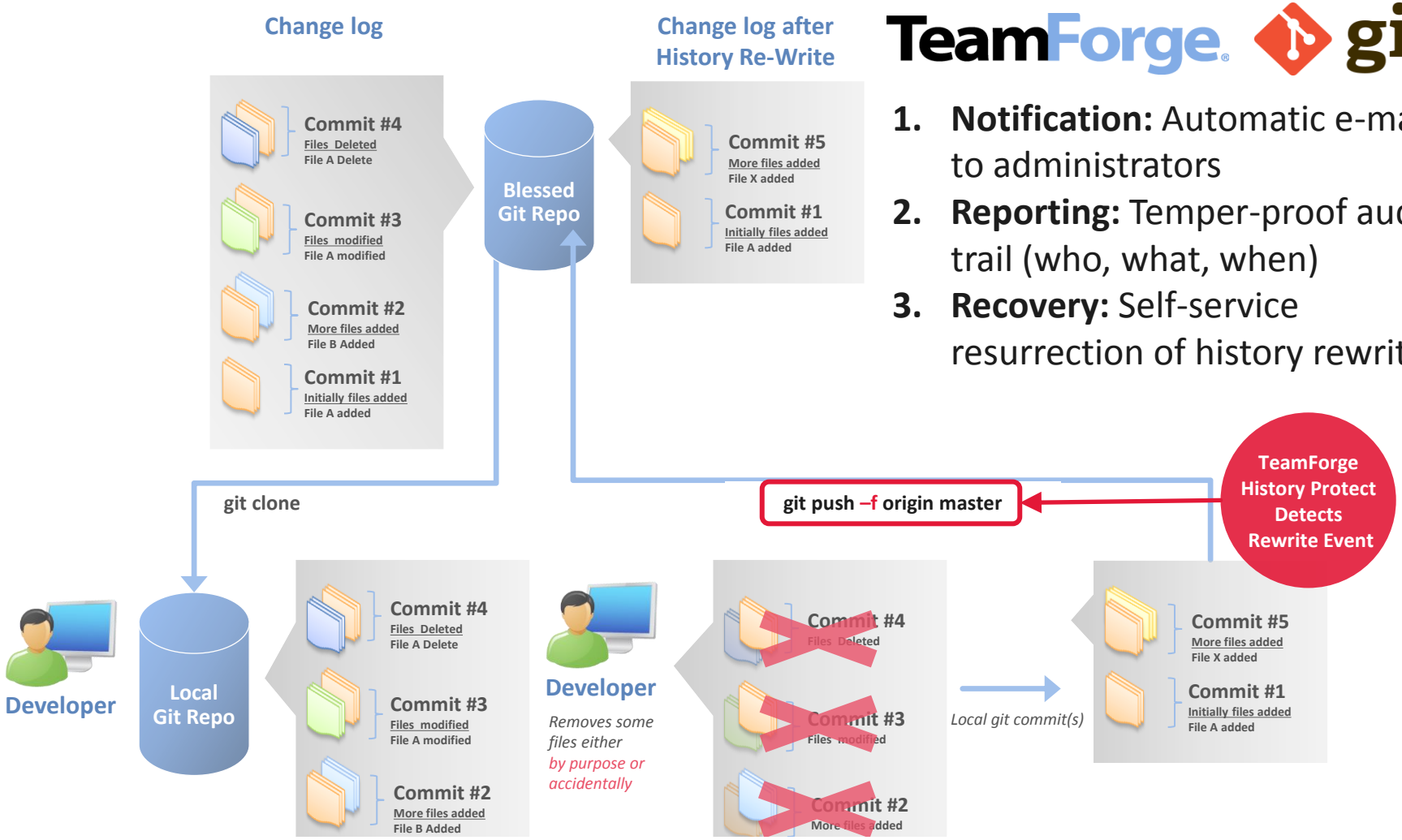
- No alerts
- No recovery (unless noticed quickly, via administrators)

Happens even to the experts:

- https://bugs.eclipse.org/bugs/show_bug.cgi?id=361707
- Eclipse Foundation lost almost all their branches and tags accidentally, garbage collection pruned unreferenced commits, administrators had to ask developers for a recent local backup
- http://jenkins-ci.org/content/summary-report-git-repository-disruption-incident-nov-10th
- 186 repositories from Jenkins Open Source Project got reset on GitHub



Introducing Git History Protection



TeamForge git

1. **Notification:** Automatic e-mail to administrators
2. **Reporting:** Temper-proof audit-trail (who, what, when)
3. **Recovery:** Self-service resurrection of history rewrite

TeamForge automatically secures history snapshots on 'blessed' master repository (under "`refs/rewrite`" or "`refs/deleted`"), for auditing & recovery.

CollabNet

TeamForge = Enterprise-Grade Git Mgmt. + ALM

- ✓ Basic SCM features
- ✓ jGit engine
- ✓ Native engine
- ✓ GitWeb

Gerrit

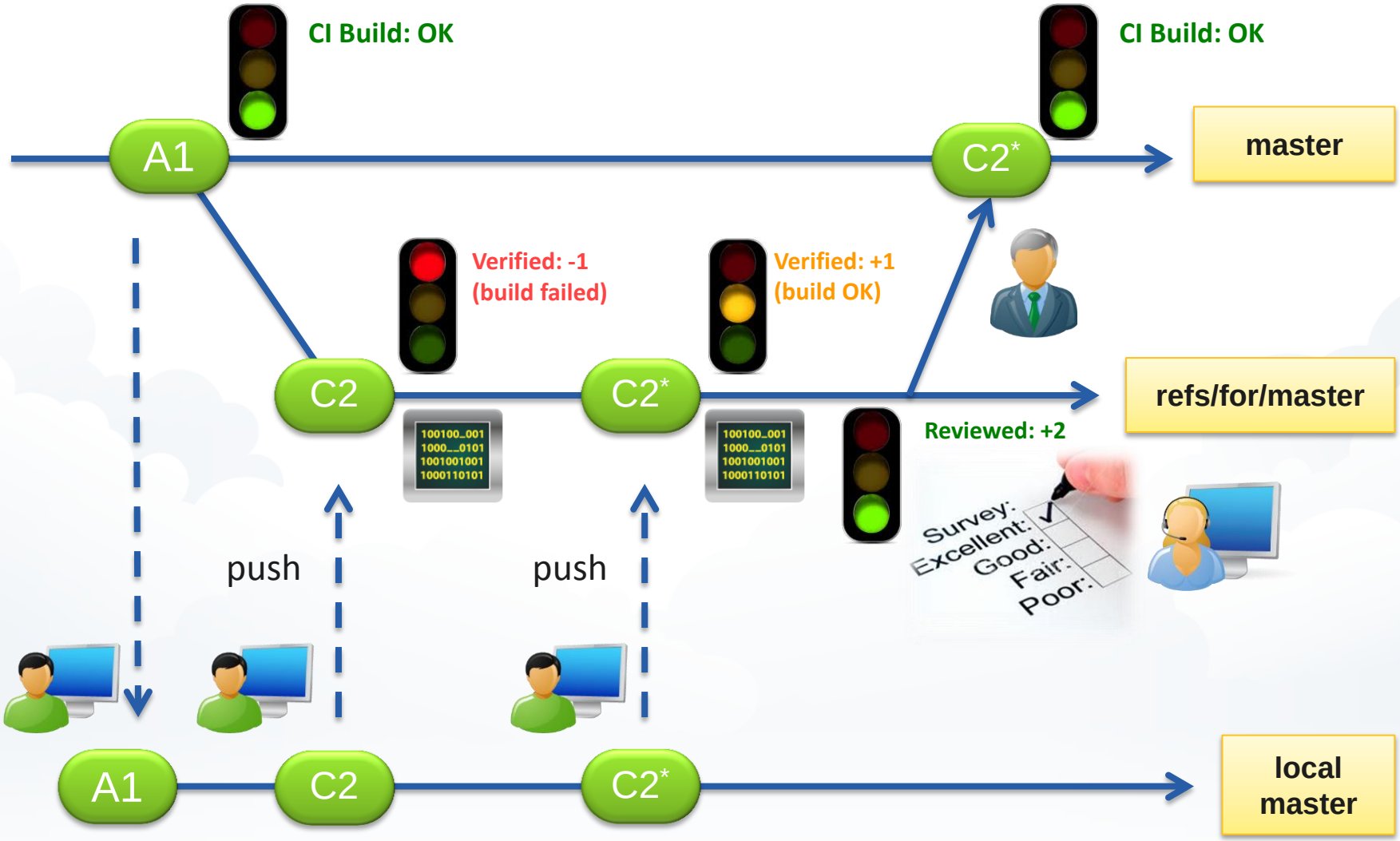
- ✓ Advanced Git security
- ✓ Git Projects organization
- ✓ Code-review
- ✓ Replication

TeamForge

- ✓ 24/7 Support
- ✓ Git/Gerrit Training
- ✓ Standards-Compliance
- ✓ Scalability
- ✓ ALM Integration
- ✓ History Protection
- ✓ Code Quality

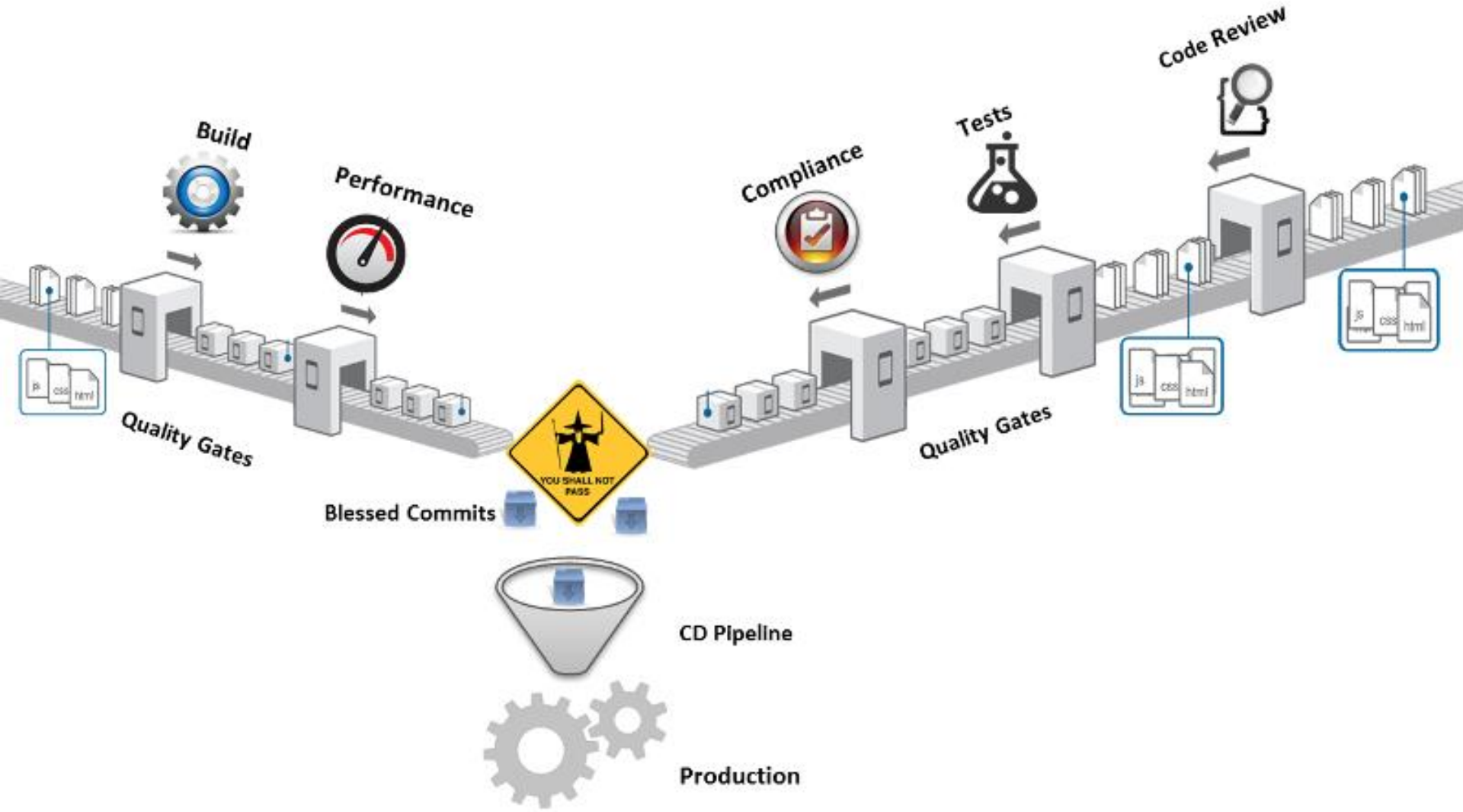
With TeamForge and Gerrit, Git is now ready for the enterprise. Realize all the benefits of Git, without compromising governance, security and compliance.

Tackling half baked review requests: Marrying Gerrit and Jenkins



CollabNet

Gerrit – Code Quality Gate Wizard – General Concept

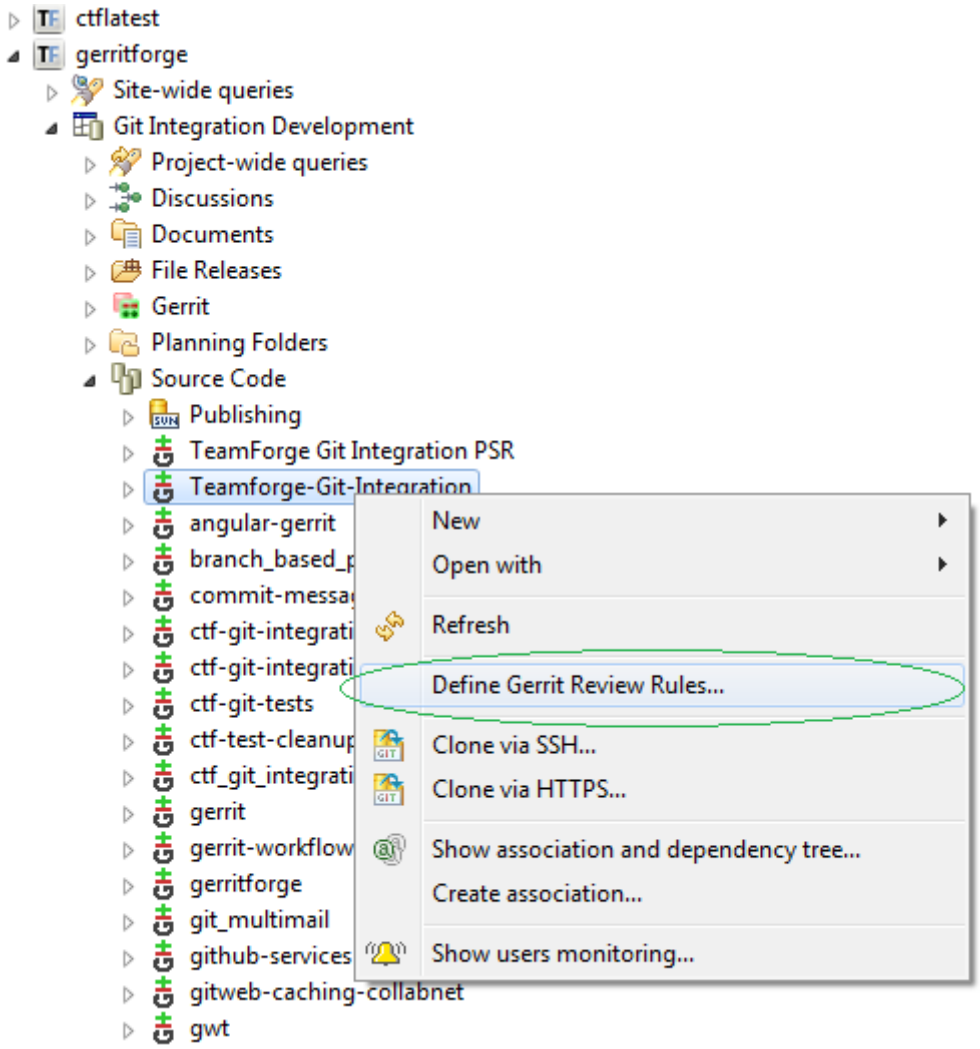


Code Quality Gate Wizard

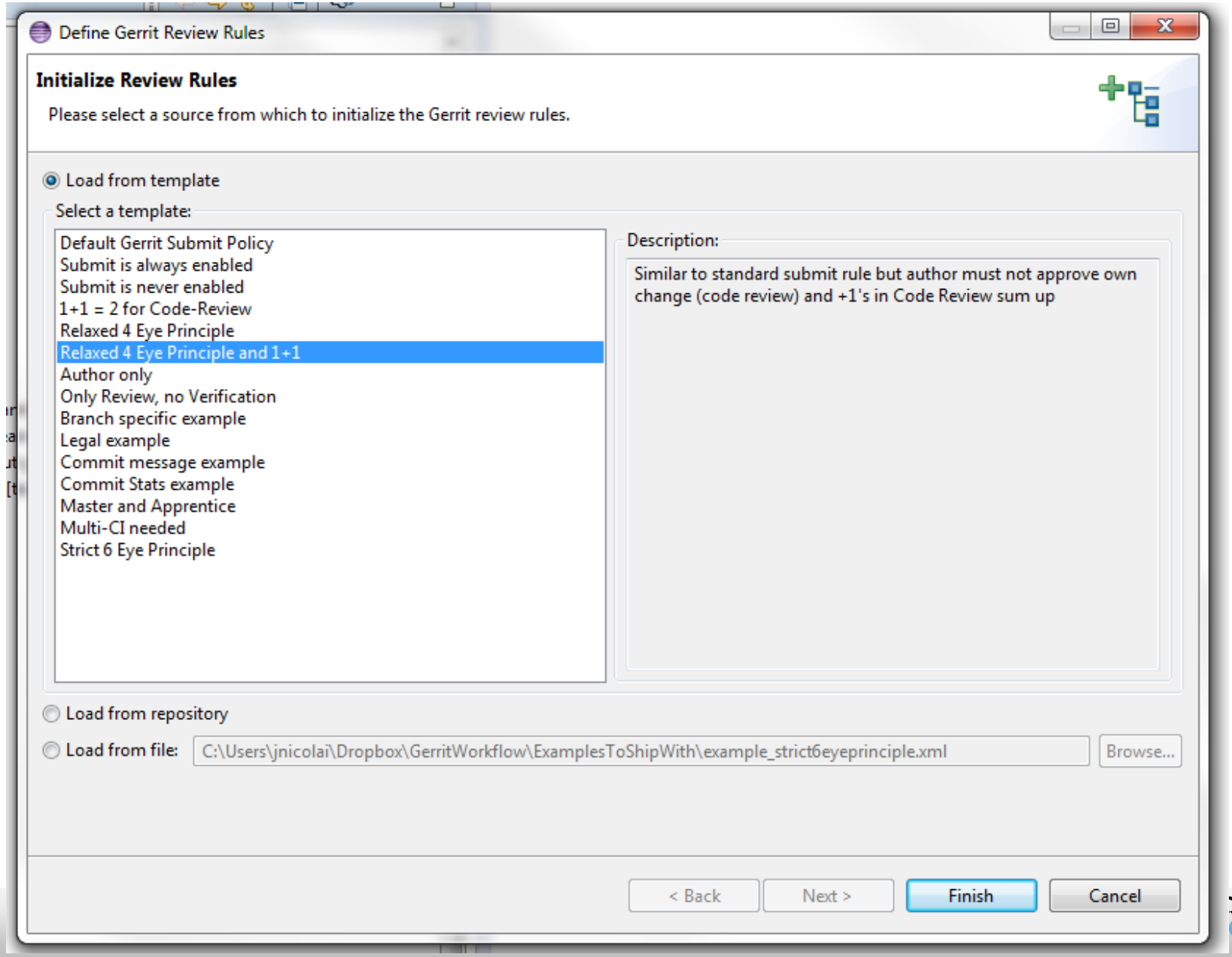
Code Quality Gate Wizard for Gerrit (Gerrit 2.8+)

- comes with a bunch of predefined policies and lets you graphically design your own quality gates as easy as defining email filter rules.
- Examples:
  - Four-eye peer review
  - Legal has to approve copyright file changes
  - Senior staff has to approve the work of juniors
  - Democratic feature voting
- More details under http://blogs.collab.net/git

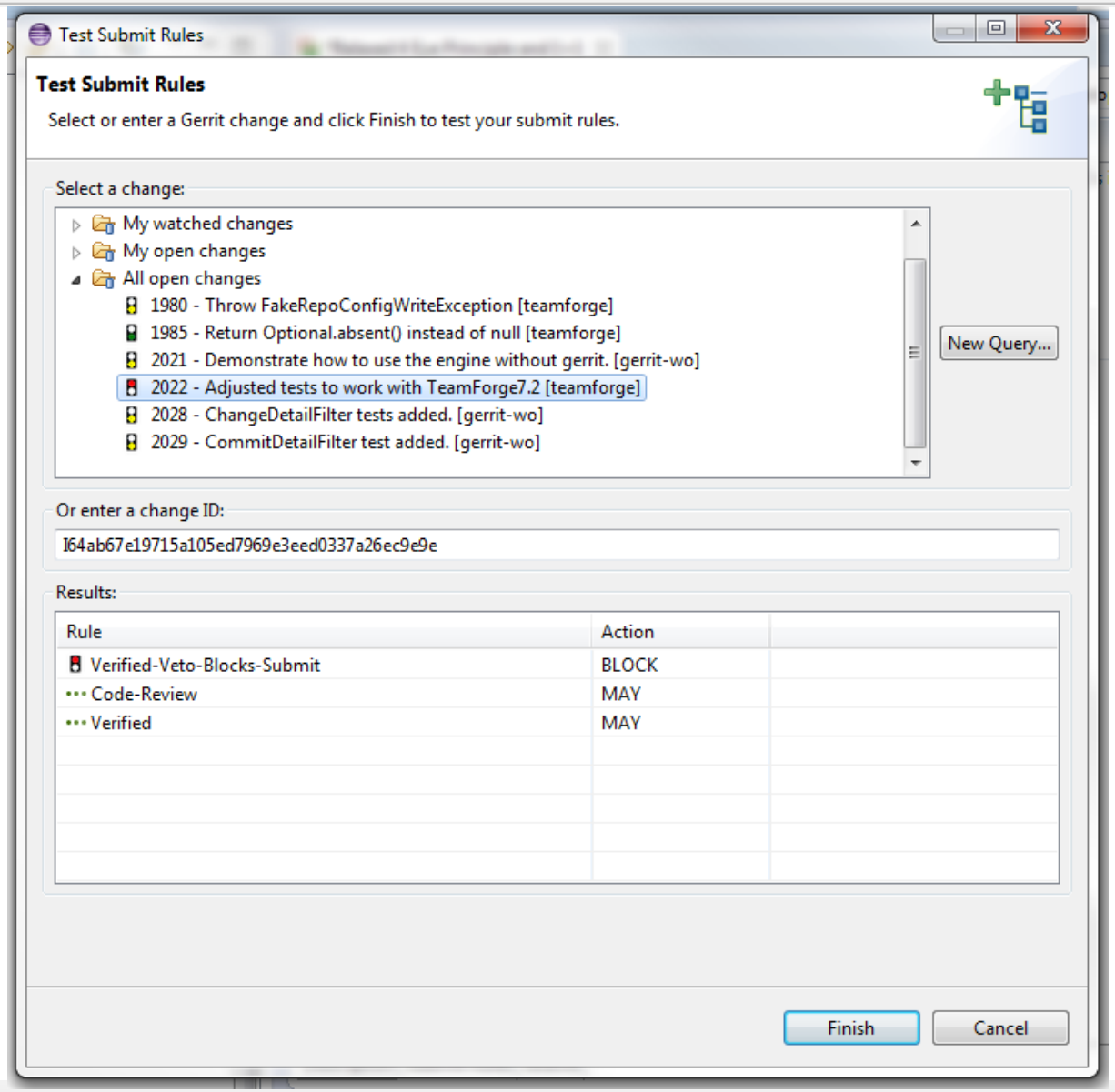
Gerrit – Code Quality Gate Wizard – How to launch



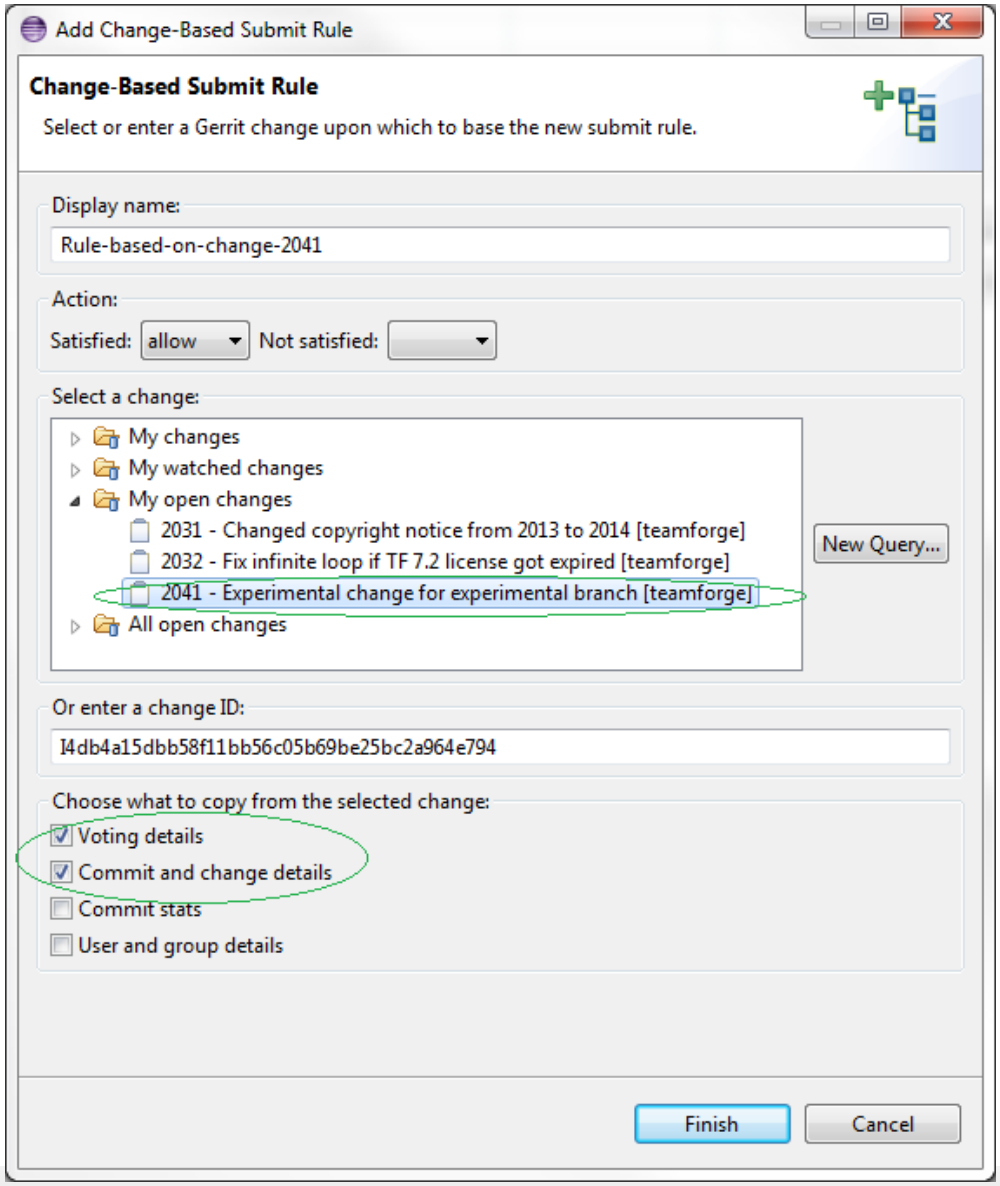
Gerrit – Code Quality Gate Wizard – Predefined templates



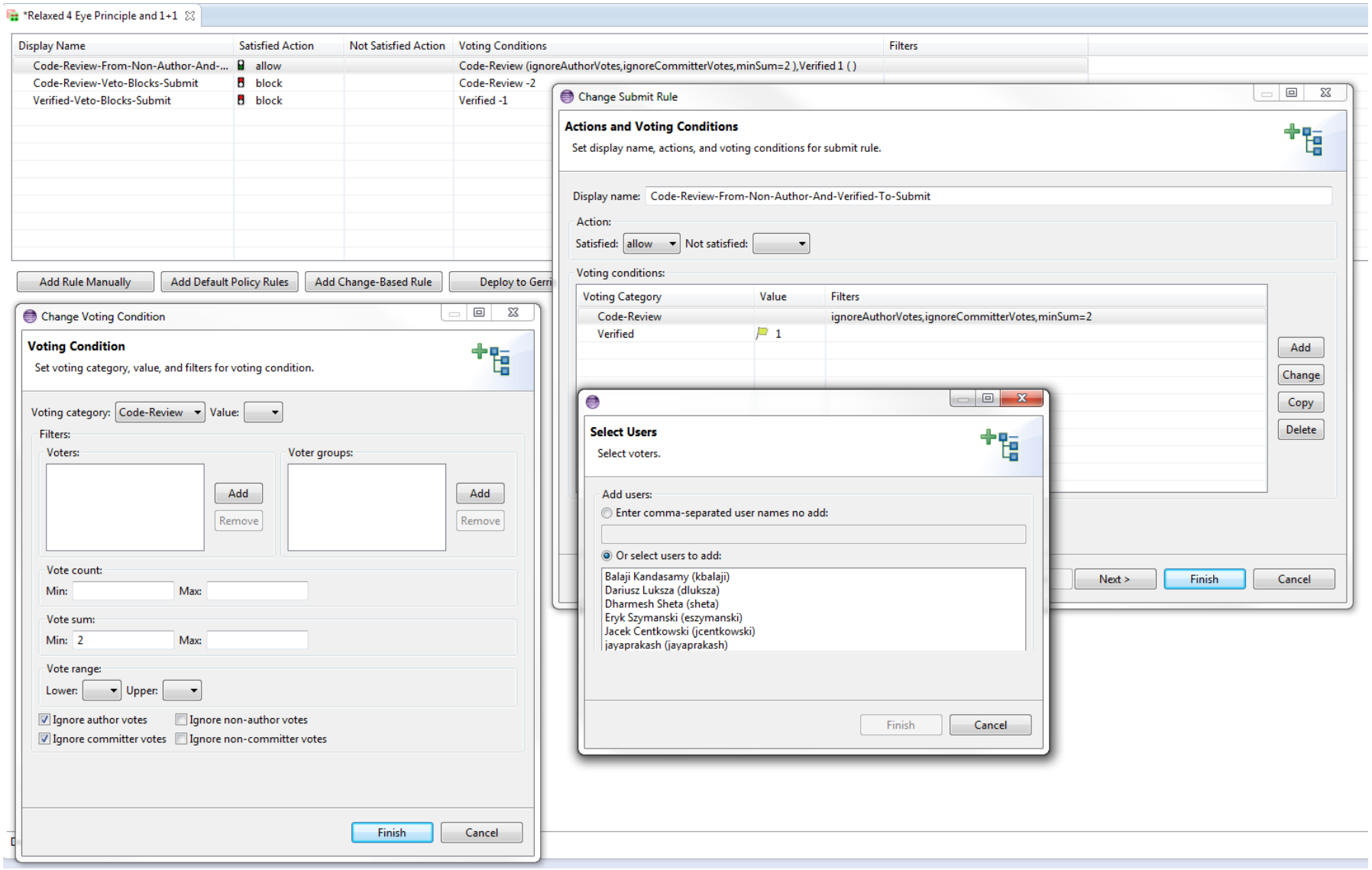
Gerrit – Code Quality Gate Wizard – Built In Test Mode



Gerrit – Code Quality Gate Wizard – Based on existing changes



Gerrit – Code Quality Gate Wizard – Great Flexibility



TeamForge = Enterprise-Grade Git Mgmt. + ALM

- ✓ Basic SCM features
- ✓ jGit engine
- ✓ Native engine
- ✓ GitWeb

Gerrit

- ✓ Advanced Git security
- ✓ Git Projects organization
- ✓ Code-review
- ✓ Replication

TeamForge

- ✓ 24/7 Support
- ✓ Security
- ✓ Standards-Compliance
- ✓ Scalability
- ✓ ALM Integration
- ✓ History Protection
- ✓ Code Quality

With TeamForge and Gerrit, Git is now ready for the enterprise. Realize all the benefits of Git, without compromising governance, security and compliance.

Learn More

3-Minute Video on History Protection
http://www.collab.net/products/teamforge/git-for-the-enterprise

Git Blogs
http://blogs.collab.net/git

Software Downloads (TeamForge, Git + clients)
http://www.collab.net/git

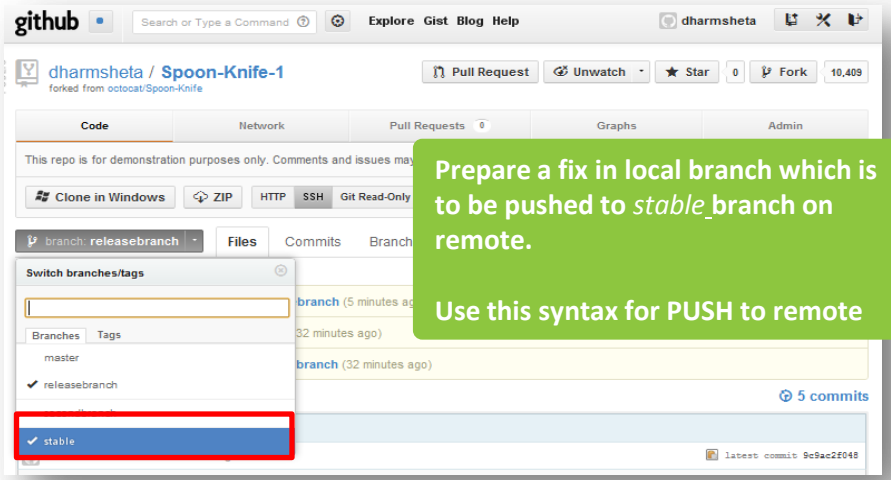
Q&A

Feature deep dive: Simple RBAC

Pre defined & user-defined categories

- **Default**: All Gerrit review features are turned off
- **Mandatory Review**: All code changes have to be reviewed
- **Optional Review**: Review feature is turned on but can be bypassed if necessary
- **Custom**: Access rights have to be set manually in Gerrit Web UI and will not be overridden by TeamForge
- **User-defined**: Possibility to define your own access rights controlled by TeamForge

Git History Rewrite – Example: Accidental Branch Deletion



Prepare a fix in local branch which is to be pushed to *stable* branch on remote.

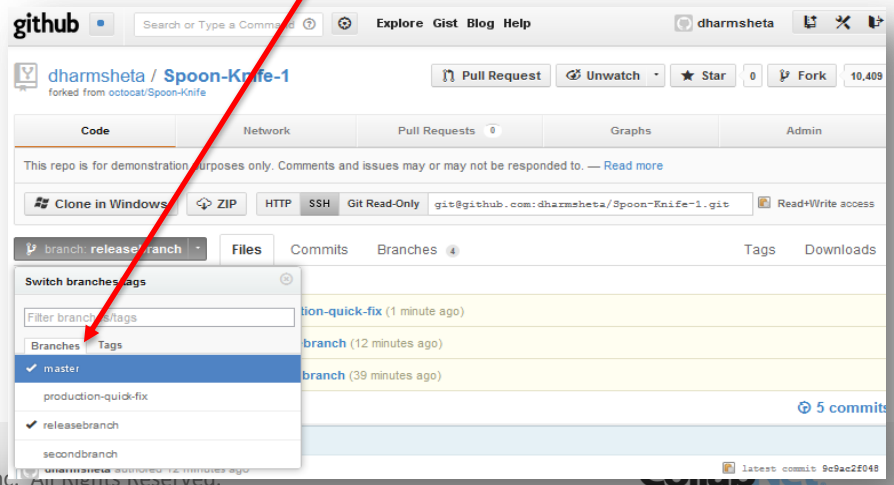
Use this syntax for PUSH to remote

```
sheta@SHETA-THINK ~/Spoon-Knife-1 (production-quick-fix)
$ echo "fix">>fix.patch && git add fix.patch

sheta@SHETA-THINK ~/Spoon-Knife-1 (production-quick-fix)
$ git commit -m "quick fix for production"
[production-quick-fix c951efd] quick fix for production
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 fix.patch

sheta@SHETA-THINK ~/Spoon-Knife-1 (production-quick-fix)
$ git push origin production-quick-fix:stable
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 270 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@github.com:dharmsheta/Spoon-Knife-1.git
   2358618..c951efd  production-quick-fix -> stable
```

Typo in push command...(accidentially added a space)

Results in branch deletion. No trace on Web UI on Git server side (*stable* branch got deleted)

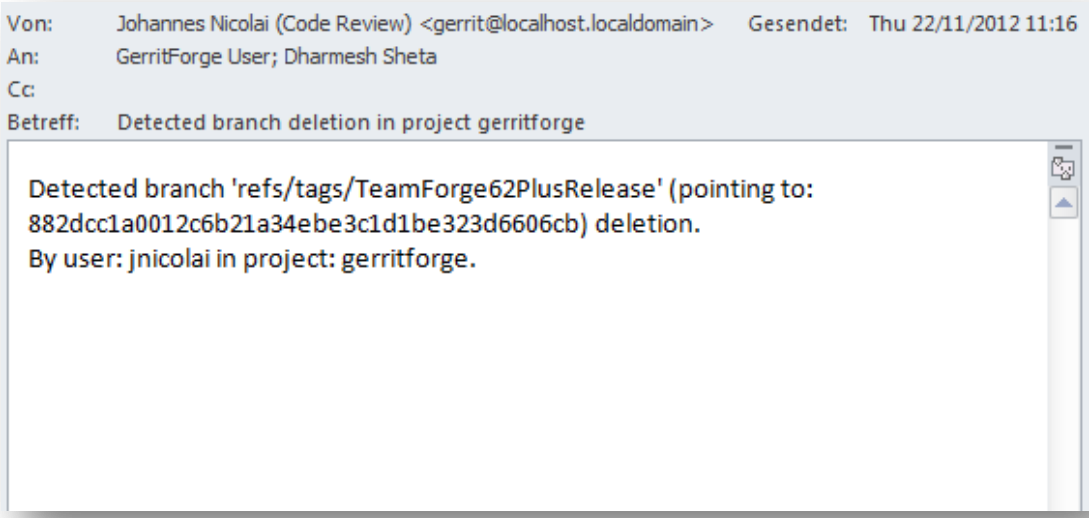
```
sheta@SHETA-THINK ~/Spoon-Knife-1 (production-quick-fix)
$ echo "fix REWORKED">>fix.patch && git add fix.patch

sheta@SHETA-THINK ~/Spoon-Knife-1 (production-quick-fix)
$ git commit -m "quick fix for production reworked"
[production-quick-fix a70264e] quick fix for production reworked
 1 files changed, 1 insertions(+), 0 deletions(-)

sheta@SHETA-THINK ~/Spoon-Knife-1 (production-quick-fix)
$ git push origin production-quick-fix :stable
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 282 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@github.com:dharmsheta/Spoon-Knife-1.git
 - [deleted]         stable
 * [new branch]      production-quick-fix -> production-quick-fix
```



Git History Protection – **Alerting**

TeamForge git

1. **Notification:** Automatic e-mail to administrators
2. **Reporting:** Temper-proof audit-trail (who, what, when)
3. **Recovery:** Self-service resurrection of history rewrite



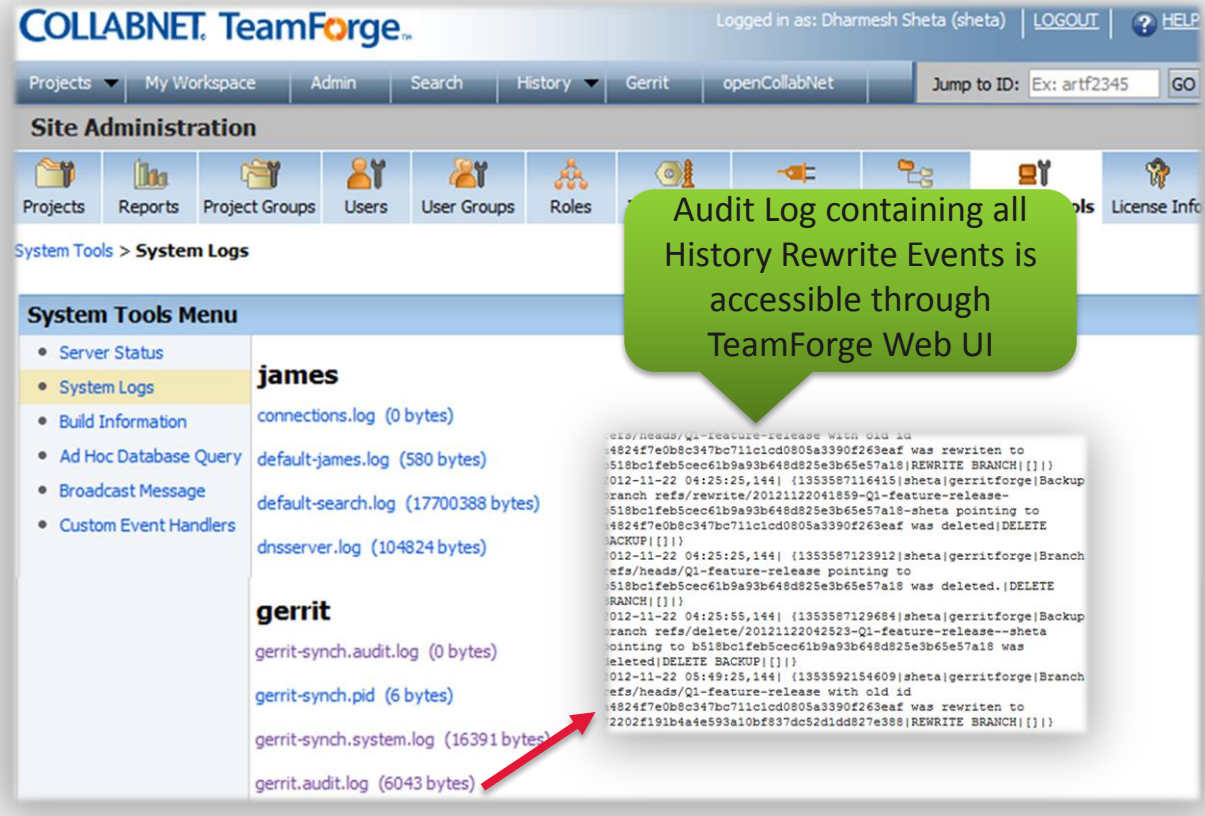
Whenever history gets ‘re-written’, an email gets sent out to ‘Gerrit Administrators’ containing details about old HEAD of branch and new HEAD after ‘re-write’.

CollabNet

Git History Protection – **Reporting** (Audit Log)

TeamForge git

1. **Notification:** Automatic e-mail to administrators
2. **Reporting:** Temper-proof audit-trail (who, what, when)
3. **Recovery:** Self-service resurrection of history rewrite

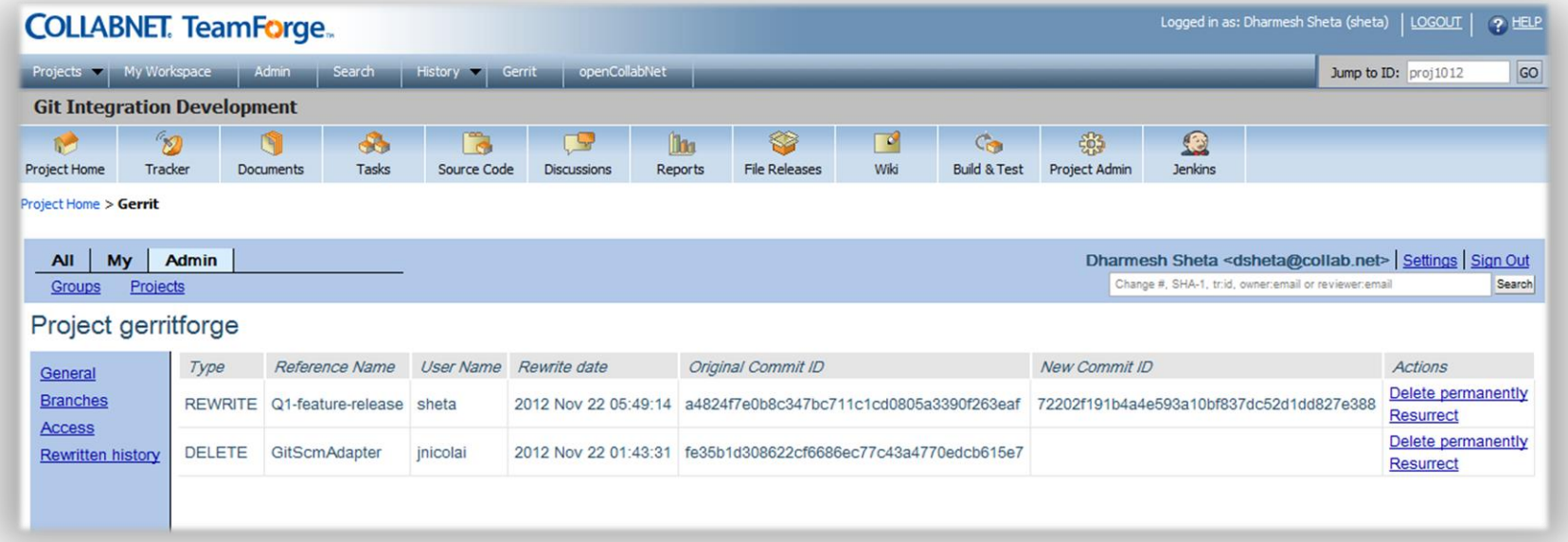


CollabNet

Git History Protection – **Reporting** (Web UI)

TeamForge git

1. **Notification:** Automatic e-mail to administrators
2. **Reporting:** Temper-proof audit-trail (who, what, when)
3. **Recovery:** Self-service resurrection of history rewrite



CollabNet

Git History Protection – **Reporting** (Command Line)

TeamForge git

1. **Notification:** Automatic e-mail to administrators
2. **Reporting:** Temper-proof audit-trail (who, what, when)
3. **Recovery:** Self-service resurrection of history rewrite

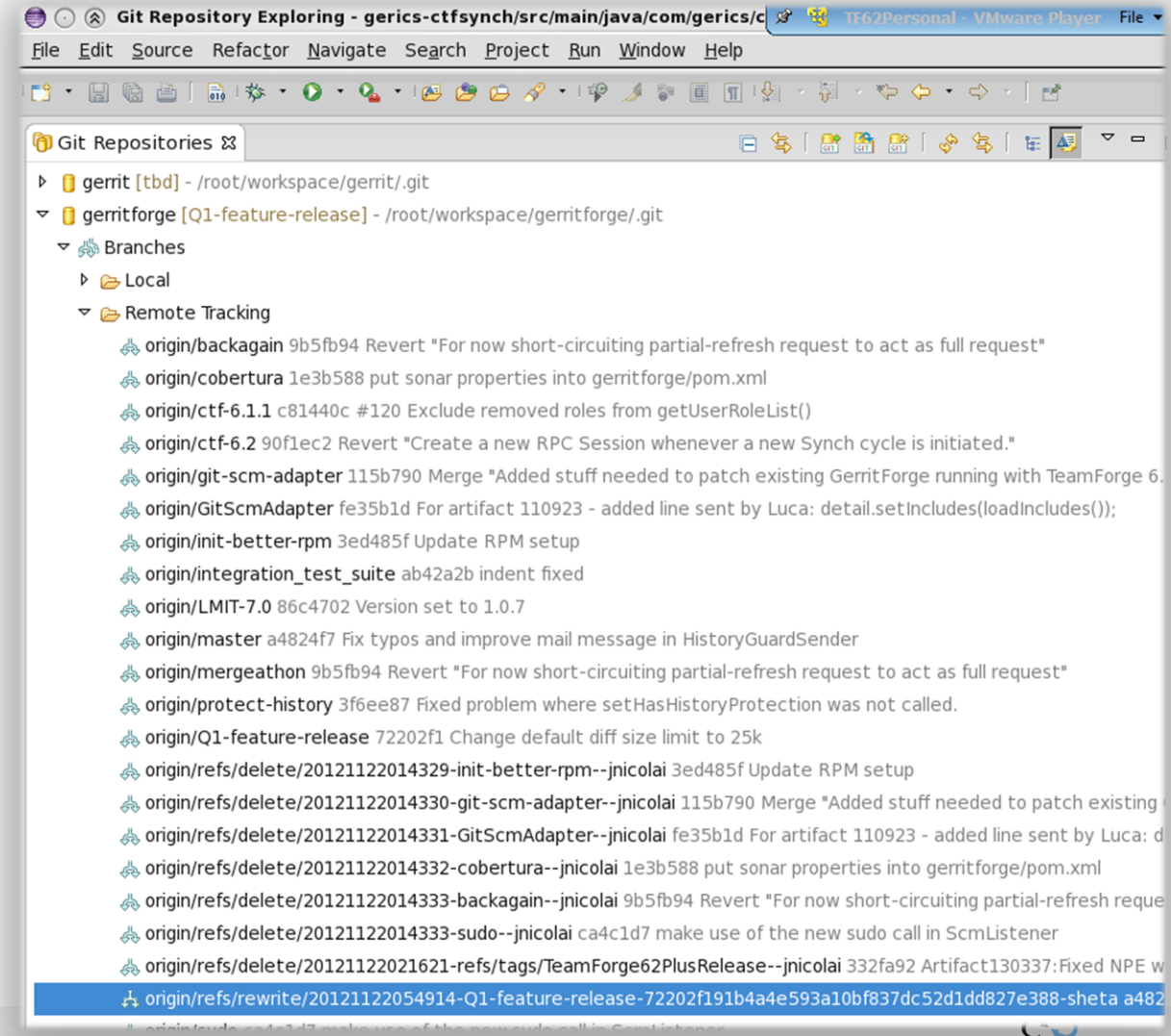
```
git ls-remote origin
fe35b1d308622cf6686ec77c43a4770edcb615e7        refs/delete/20121122014331-GitScmAdapter--jnicolai
a4824f7e0b8c347bc711c1cd0805a3390f263eaf        refs/rewrite/20121122054914-Q1-feature-release-72202f
837dc52d1dd827e388-sheta
```


CollabNet

Git History Protection – **Reporting** (Eclipse)

TeamForge git

1. **Notification:** Automatic e-mail to administrators
2. **Reporting:** Temper-proof audit-trail (who, what, when)
3. **Recovery:** Self-service resurrection of history rewrite

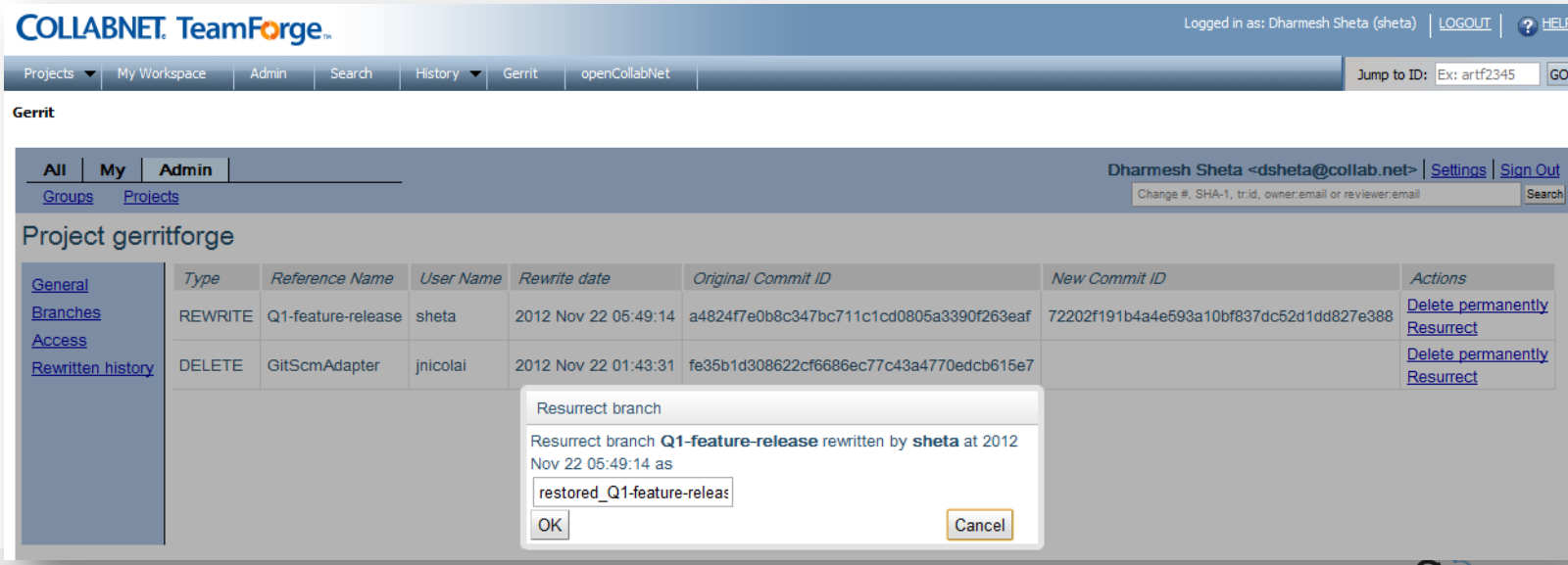


CollabNet

Git History Protection – **Recovery** (Web UI)

TeamForge git

1. **Notification:** Automatic e-mail to administrators
2. **Reporting:** Temper-proof audit-trail (who, what, when)
3. **Recovery:** Self-service resurrection of history rewrite



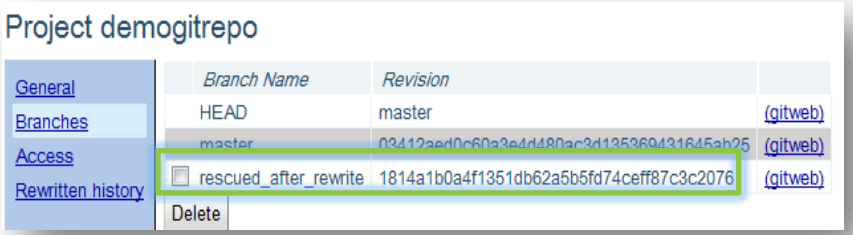
CollabNet

Git History Protection – **Recovery** (Command Line)

TeamForge git

1. **Notification:** Automatic e-mail to administrators
2. **Reporting:** Temper-proof audit-trail (who, what, when)
3. **Recovery:** Self-service resurrection of history rewrite

Alternatively users having permissions to create a new branch can restore history by using their git client



Resurrected branch via command line is now available for all user having atleast read access(view only) for this Git repository

Run this command on git command line

```
$ git ls-remote origin
03412aed0c60a3e4d480ac3d135369431645ab25        HEAD
03412aed0c60a3e4d480ac3d135369431645ab25        refs/heads/master
1814a1b0a4f1351db62a5b5fd74ceff87c3c2076        refs/rewrite/20121102215850-master-03412aed0c60a3e4d480ac3d135369431645ab25-david
```

Copy HEAD SHA1 of branch created after history 're-write'

Fetch SHA1 of rewritten history from server

```
$ git fetch origin refs/rewrite/20121102215850-master-03412aed0c60a3e4d480ac3d135369431645ab25-david
```

Use copied SHA1 to create new local branch in Git

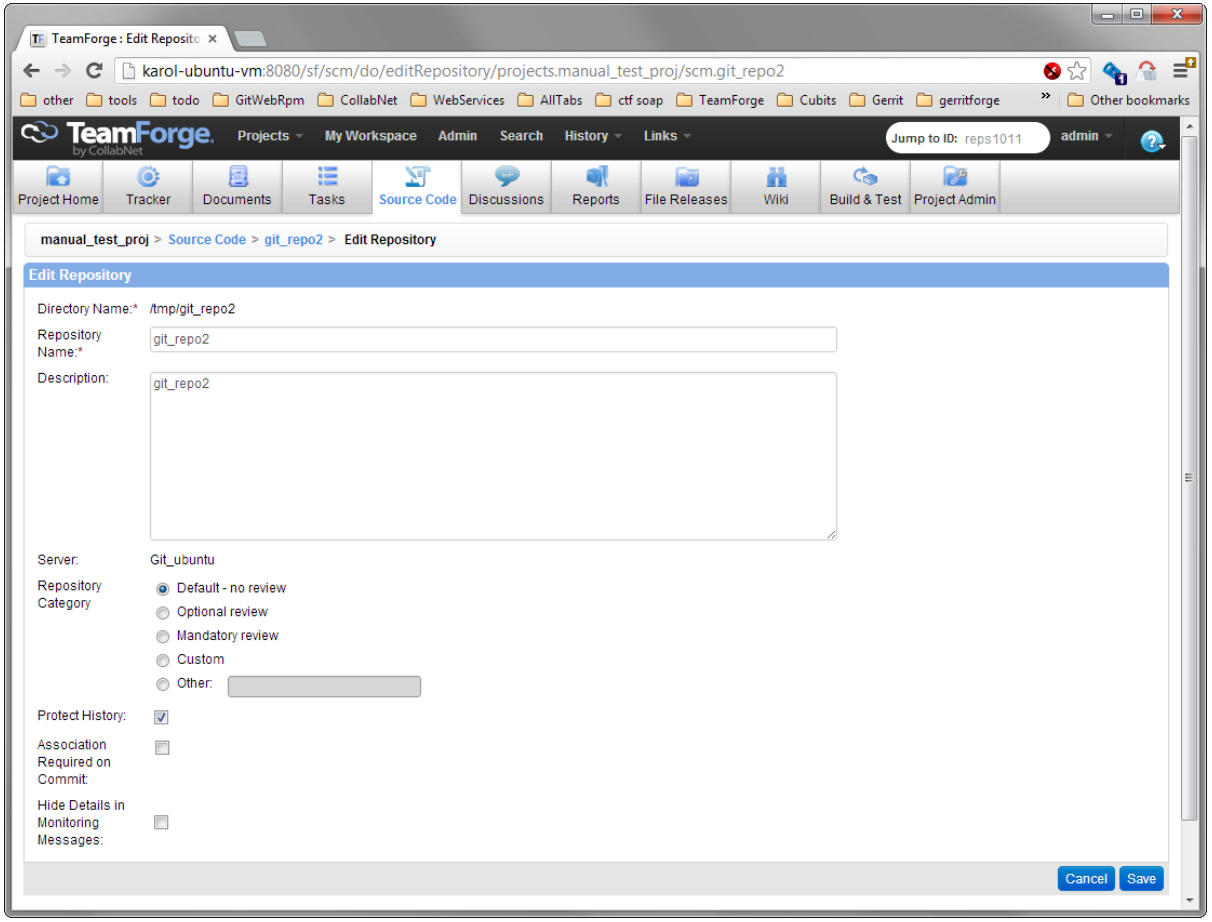
```
$ git checkout -b rescued_from_commandline 1814a1b0a4f1351db62a5b5fd74ceff87c3c2076
Switched to a new branch 'rescued_from_commandline'
```

Push this local branch to remote 'blessed reposiotry'

```
$ git push origin rescued_from_commandline:rescued_after_rewrite
Total 0 (delta 0), reused 0 (delta 0)
To ssh://david@tf62personal.potsdam.collab.net:29418/demogitrepo
 * [new branch]      rescued_from_commandline -> rescued_after_rewrite
```


CollabNet

Git History Protection – Administration



Gerrit Config-Option allows you to protect all Git repositories hosted by TeamForge: Not even TeamForge Site Admins can override

```
[gerrit]
        basePath = /gitroot
        forceHistoryProtection = true
[database]
```

Git History Rewrite – Why Blocking is NOT the Answer

Legitimate use cases

- Developers
  - Deleting accidently committed file(s)/change(s)
  - Change appearance of commits
    - squashing multiple commits into unified single commit
    - change order of commits
- Build / Release managers / Developers
  - Removing Copyrights/ Intellectual Property(IP) related resources from code base
  - Removing large file(s)
  - Removing feature branch(es) created temporarily and already merged

Not (so) Legitimate use cases

- Developers / Build / Release Managers
  - Remove somebody else's changes without leaving any trace
  - Pretending someone else's change as own (forgery)
  - Accidentally removing branches

Git History Rewrite – Why Blocking is NOT the Answer

Legitimate use cases

- Developers
  - Deleting accidently committed file(s)/change(s)
  - Change appearance of commits
    - squashing multiple commits into unified single commit
    - change order of commits
- Build / Release managers / Developers
  - Removing Copyrights/ Intellectual Property(IP) related resources from code base
  - Removing large file(s)
  - Removing feature branch(es) created temporarily and already merged

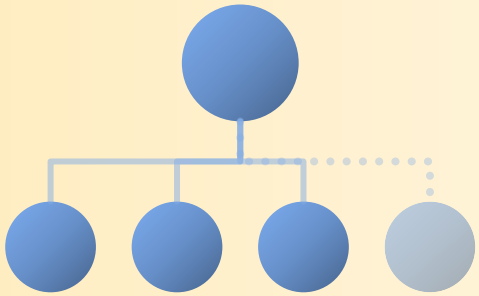
Not (so) Legitimate use cases

- Developers / Build / Release Managers
  - Remove somebody else's changes without leaving any trace
  - Pretending someone else's change as own (forgery)
  - Accidentally removing branches

Git History Rewrite – Why Blocking is NOT the Answer

Legitimate use cases

- Developers
  - Deleting accidently committed file(s)/change(s)
  - Change appearance of commits
    - squashing multiple commits into unified single commit
    - change order of commits
- Build / Release managers / Developers
  - Removing Copyrights/ Intellectual Property(IP) related resources from code base
  - Removing large file(s)
  - Removing feature branch(es) created temporarily and already merged



Not (so) Legitimate use cases

- Developers / Build / Release Managers
  - Remove somebody else's changes without leaving any trace
  - Pretending someone else's change as own (forgery)
  - Accidentally removing branches

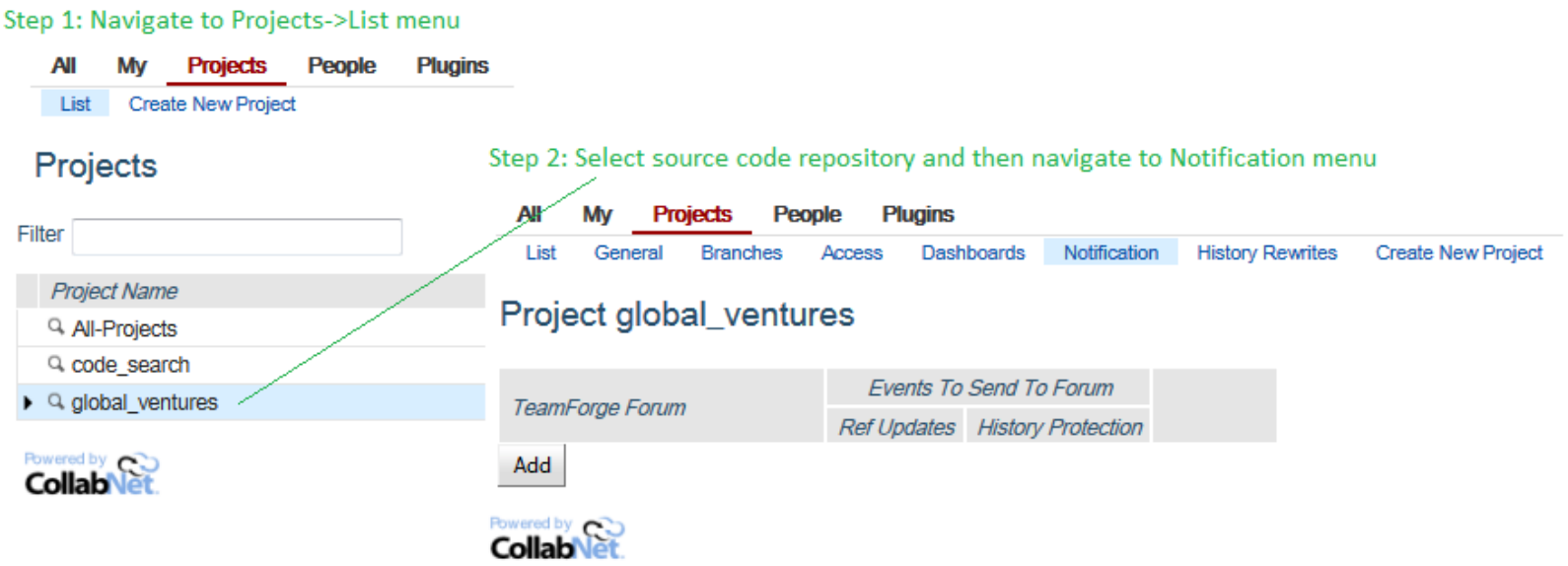
Git History Protection - Summary

- TeamForge + Gerrit is a powerful foundation for Enterprise Git
- Git History Rewrite covers branch deletions and forced pushes, two very powerful, often needed but quite dangerous operations which might lead to data loss and tampering
- TeamForge 6.2 with Gerrit introduced History Protection
- History protection ensures that potentially unnoticed events, such as remote branch deletions and forced pushes, are now detected and fully recoverable, any modifications to Git code and repositories are securely recorded with tamper-proof audit logs
- Recovery is possible from a Web interface at the push of a button, and using an ordinary Git client, IT operations teams don't have to be involved

Feature deep dive: Notifications

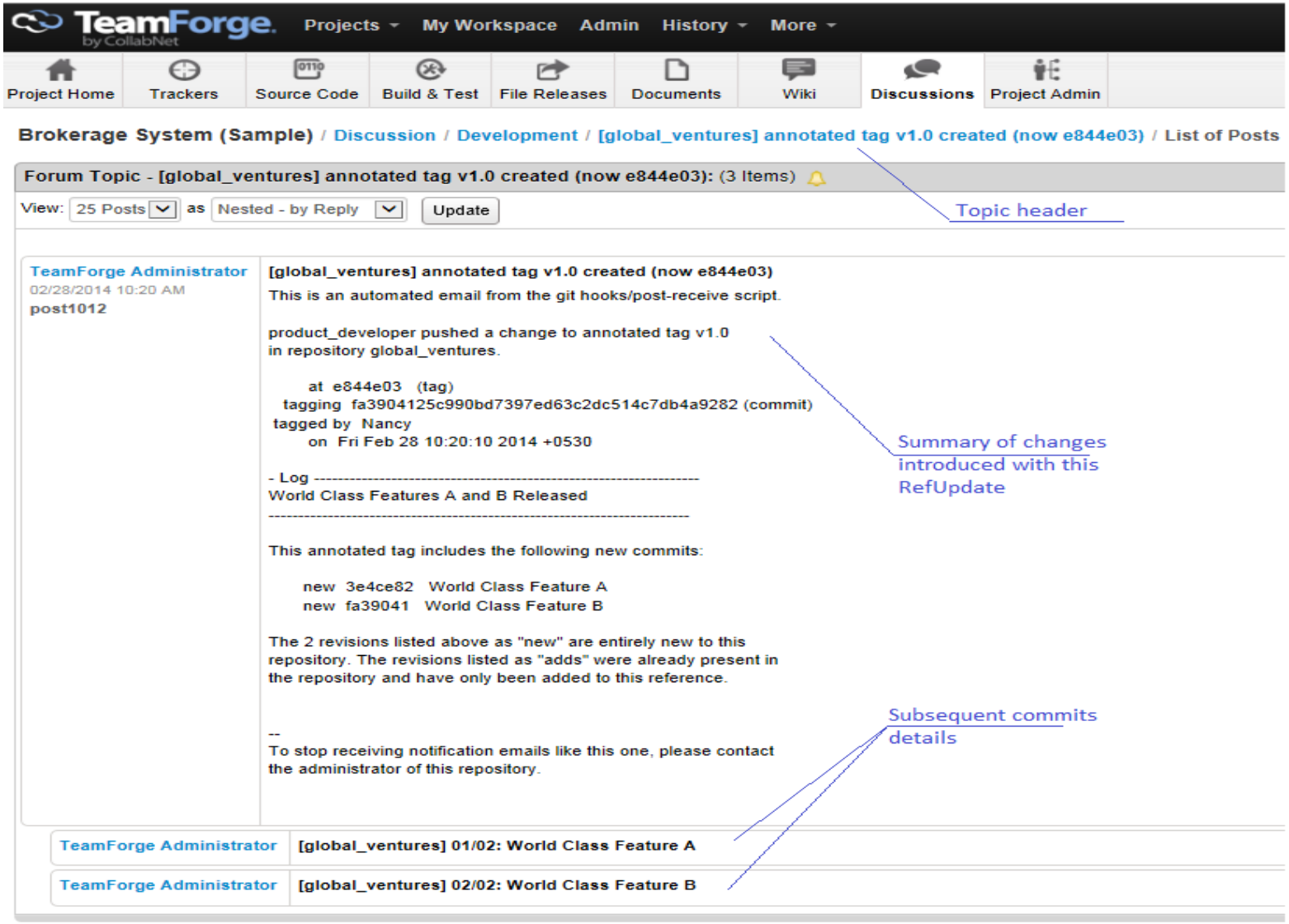
Motivation:

Bring the git push notifications mechanism from the gitmultimail script to TeamForge



CollabNet

Feature deep dive: Notifications



Appendix – History Rewrite, versus Git Reflog

	Git reflog	TeamForge Git Integration with 'History Protect'
Accessibility	Requires direct access to file system on server where 'blessed Git repository' is hosted which is very unlikely in huge organizations and will keep the server administrators busy	'Self –Service' approach. Users with appropriate permissions in TeamForge can find out/resurrect deleted/rewritten branches by themselves, decreasing work load of server administrators. Gerrit Administrators can also permanently delete selected branches/tags.
Signal-to- Noise Ratio	reflog records <all> changes in the repository • Any push (also ordinary fast forward) • Any merge • Any Branch creation/deletion • Any Tag creation / deletion Finding out about history rewrites/deleted branches is like searching for a needle in a haystack	History Protect only reports • Deleted branches/tags • History rewrites (non fast forward pushes)

Notification	No notification	• Email to Gerrit Administrators
		• Audit log entry whenever branch/tags gets
		  • Deleted
		  • Re-written (non fast forward)
		  • Resurrected
		  • Permanently deleted

Ease of use	• Only manually configurable by administrator having file system access • To be configured for each and every repository • Restoring requires running git commands on server	• Pre-configured in TeamForge/Gerrit • Configurable for all repositories by setting site-wide config option or on a per repository basis • User with appropriate permission can restore history using Gerrit WebUI / Git client
Protection against object pruning/reflog expiration	RefLog expiration and gc pruning settings have to be manually configured by server administrator. Only possibility to not lose commits no longer referenced in a branch is to set both values to <unlimited> which will consume huge amounts of disk space, slows down garbage collection and does not allow to permanently delete specific commits (all or nothing).	Preserved commits will never pruned by garbage collection unless permanently removed using Gerrit Web UI. No need to keep a large ref log. Garbage collection will run faster since all commits are still referenced in the repo.