Deploying Gerrit Code Review

Shawn Pearce, Magnus Bäck,
Stefan Lay, Matthias Sohn

eclipse CON
BOSTON 2013

Today's Agenda

Introduction to Gerrit
Access Controls
Scaling Gerrit
Advanced Workflows

Gerrit's History

Android Open Source Project

- Peer code review is central to development at Google
- Android needed a tool to support open source on Git

Project Lineage

- Google Mondrian ⇉ Rietveld ⇉ Gerrit 1.x ⇉ Gerrit 2.x

Contributors

- Google, Qualcomm, SAP, Sony Mobile, Wikimedia, GerritForge, CollabNet, Spotify, Garmin, Kitware, ...

Deployment Options

Database

H2 *(built-in)*, PostgreSQL, MySQL

Servlet Container

Jetty *(built-in)*, Tomcat *(war deployment)*

Authentication

LDAP, OpenID, Container

For this tutorial we will use H2 and Jetty

Exercise 1 – Install Gerrit

(optional) Create a non-privileged user to run Gerrit

```
$ sudo adduser gerrit2
$ sudo su gerrit2
```

Initialize review site (batch mode) and start Gerrit

```
$ java -jar gerrit.war init --batch -d ~/gerrit_testsite
```

Server starts automatically on Linux and Mac OS X

On Windows, start by hand in 2 more slides

http://code.google.com/p/gerrit/downloads/detail?name=gerrit-2.6-rc0.war

Exercise 1 – Configure Gerrit

Edit settings in '~/gerrit_testsite/etc/gerrit.config'

Change

```
[auth]
    type = OPENID
```

into

```
[auth]
    type = DEVELOPMENT_BECOME_ANY_ACCOUNT
```

OPENID: requires network access

DEVELOPMENT_BECOME_ANY_ACCOUNT: good for experimentation

Exercise 1 – Restart Gerrit

Restart Gerrit server to pickup config changes:

```
$ ~/gerrit_testsite/bin/gerrit.sh restart
```

or if you like to type, restart in two steps:

```
$ ~/gerrit_testsite/bin/gerrit.sh stop
$ ~/gerrit_testsite/bin/gerrit.sh start
```

On Windows, start the daemon:

```
$ java -jar gerrit.war daemon -d ~/gerrit_testsite
```

Use Ctrl + C to stop

Exercise 2 – Admin User

Check if you have a ssh key pair

```
$ ls ~/.ssh
authorized_keys  config  id_rsa  id_rsa.pub  known_hosts
```

If necessary generate a key pair with a passphrase

```
$ ssh-keygen -t rsa
```

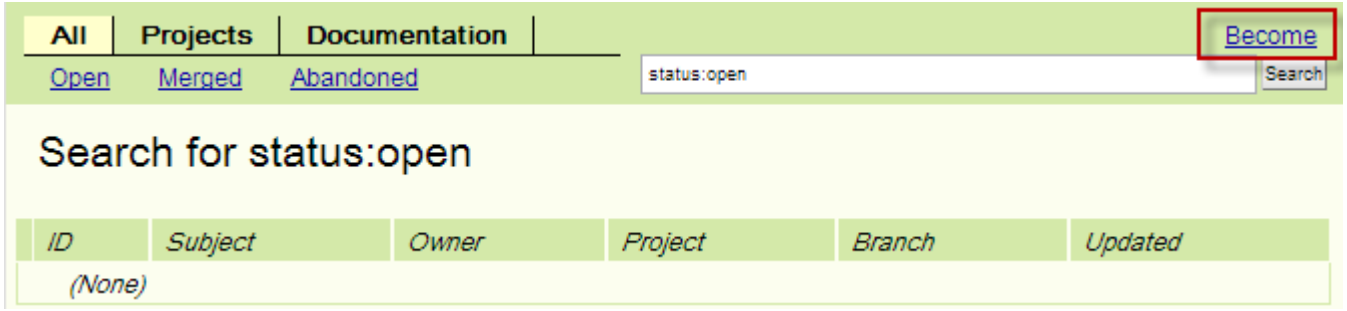
Open Gerrit UI (URL and listen port defined in gerrit.config)

http://localhost:8080/

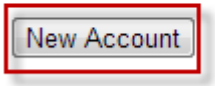
Exercise 2 – Admin User

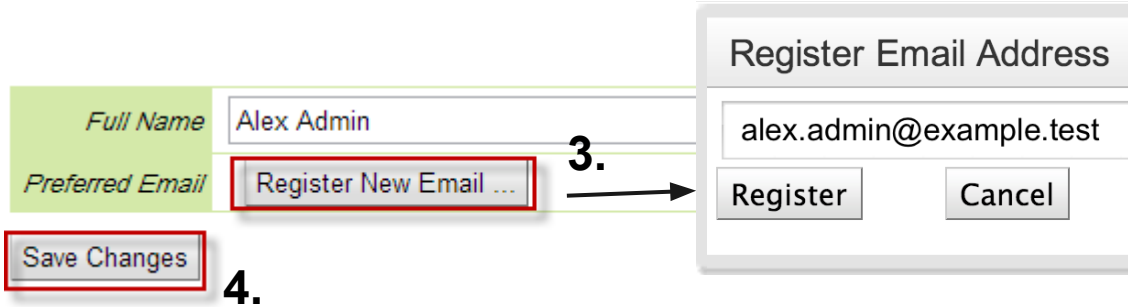
Register a new user (The first user has admin rights)

1.



Register 2.





Exercise 2 – Admin User

Add public ssh key (content of ~/.ssh/id_rsa.pub)

Exercise 3 – Developer User

The first user created is granted admin rights automatically.
Additional users are ordinary users with no special powers.

Create an ordinary user
- Sign out your admin user
- Click on "Become"
- Click on "New Account"
- Enter name and email

    The email address should be the one you use for git

- Enter a different username, "`developer`" *(used later in slides)*
- Upload your same public ssh key

Create a Project

A Project corresponds to a git repo located in `gerrit.basePath`

Use ssh command

```
$ ssh -p 29418 alexadmin@localhost gerrit create-project --name myproject
```

Web alternative

http://localhost:8080/ Projects > Create New Project

Command line alternative

Copy bare Git repository into ~/gerrit_testsite/git/myproject.git

Make Gerrit look for new repositories:

```
$ ssh -p 29418 alexadmin@localhost gerrit flush-caches
  --cache project_list
```

Exercise 4 – Create a Project

Use ssh command

```
$ ssh -p 29418 alexadmin@localhost gerrit create-project
--name myproject
```

Inspect the newly created project

- Go to the WebUI, click Projects > List
- Click on the new project and inspect its properties

- type ls ~/gerrit_testsite/git

Reviewing Code

Push to the "magic" `refs/for/<branch_name>`

- Gerrit creates a special ref for the commit
- Gerrit creates a Change with a Patch-Set

A Change-Id line is needed to match further versions of the Change

- Install the Change-id hook before pushing

Every logged-in user can see and review the change

Exercise 5 – Reviewing Code

```
$ git clone ssh://developer@localhost:29418/myproject
$ cd myproject
```

Install the Change-Id hook

```
$ scp -p -P 29418 developer@localhost:hooks/commit-msg .
git/hooks/
```

Create a change

```
$ echo "Hello Gerrit" > hello.txt
$ git add hello.txt
$ git commit -m "My first change"
```

Push it for review

```
$ git push origin HEAD:refs/for/master
```

Exercise 5 – Reviewing Code

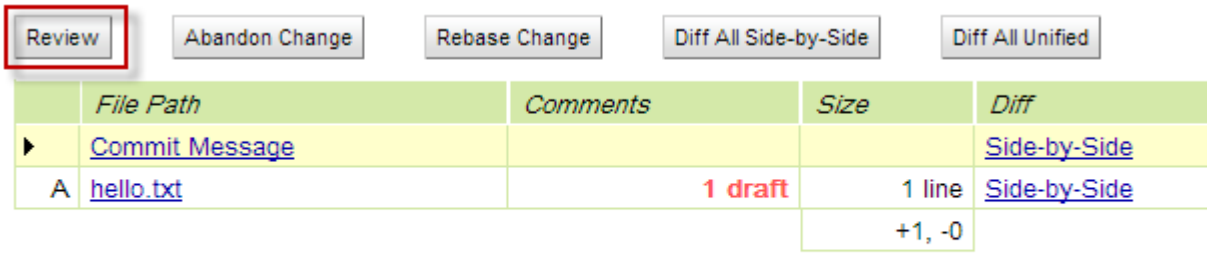
Login as the developer user.

Access the Change in the browser:

Direct link: http://localhost:8080/1

Click on hello.txt, double click on the content and enter some text

Click on "Review"



Exercise 5 – Reviewing Code

By default every logged-in user can review

But: +1 is not sufficient to submit (merge) the change.

Submitting a Change

Submitting: merging into the target branch

Default workflow:

A change can be submitted if it:

- has highest vote in every label category
- has no lowest vote in any label category

Normally project committers are allowed to submit.

Exercise – Create Group

The developer user needs permissions to submit
Permissions can be modified by any admin *(more details later)*

- Log in as admin
- People > Create New Group
- Name "myproject-committers"
- Add developer user

Exercise – Grant Access

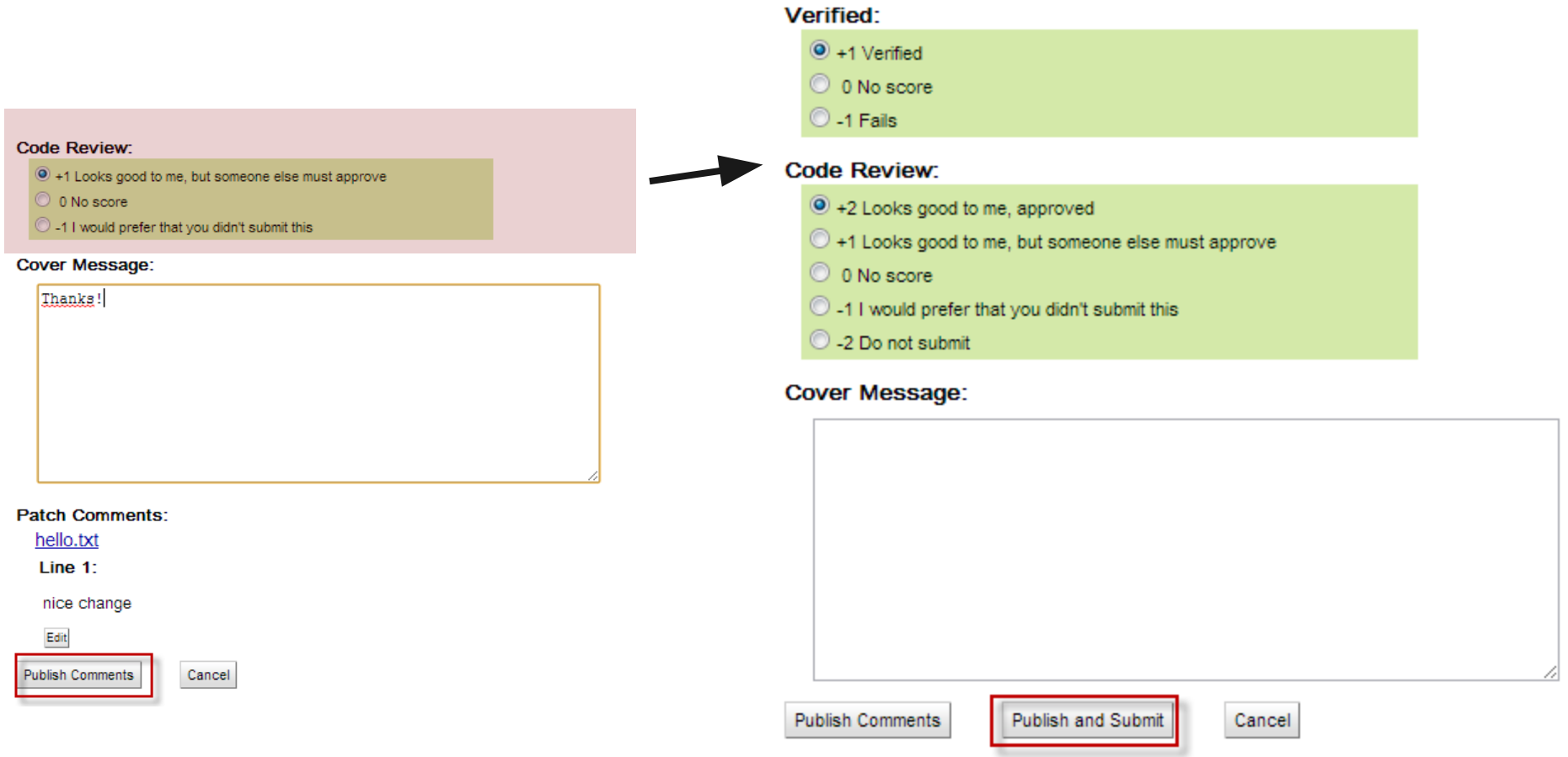
Projects > List > myproject

Projects > Access

Add permissions:

Exercise – Submit Change

Log in as developer again, click Review button



Addressing Review Issues

- A version of a Change is called a Patch Set
- A Patch Set is represented by a Git commit
- A new Patch Set "replaces" an older one

- A Patch Set must not use another Patch Set as parent

- A new Patch Set is created by amend:

```
git commit --amend
```

Exercise – Be Human

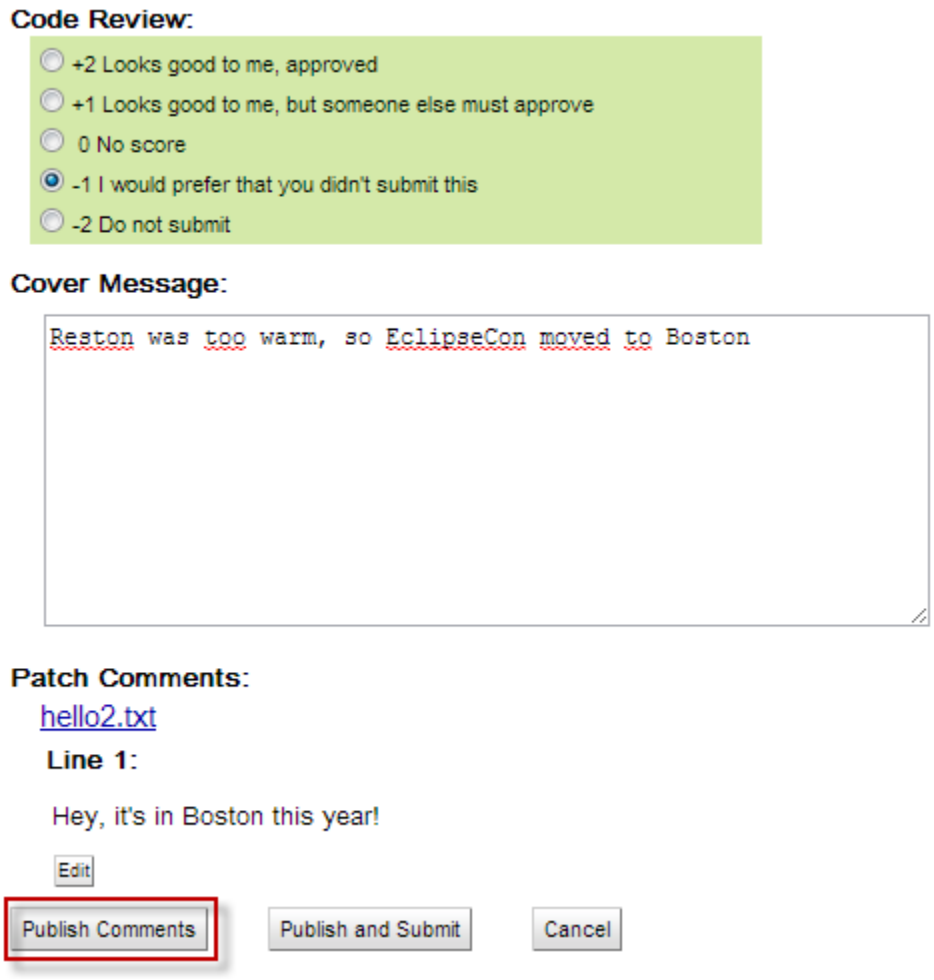
Create another change

```
$ echo "Hello EclipseCon in Reston" > conference.txt
$ git add conference.txt
$ git commit -m "Greet EclipseCon"
```

Push this for review

```
$ git push origin HEAD:refs/for/master
```

Exercise – Being Human



Exercise – Address Issues

Create another version of the change

```
$ echo "Hello EclipseCon in Boston" > conference.txt
$ git add conference.txt
$ git commit --amend
```

Note: Do NOT use -m here, the Change-Id would be replaced

Push it for review

```
$ git push origin HEAD:refs/for/master
```

Submit Behavior

Submit Action: set on project level

- **Fast Forward Only:** no merges, linear history
- **Merge If Necessary:** default, like git merge*
- **Always Merge:** like git merge --no-ff*
- **Rebase If Necessary:** rebase*
- **Cherry Pick:** cherry pick the patch set, ignoring lineage

* content merge only if `Automatically resolve conflicts` is set

Scaling Gerrit

The Boondocks

Scaling Gerrit

Memory

- Size of Git on disk, plus 2G (or more)
- ~8G Java heap, 24G heap not uncommon

Replication and Slaves

- Multiple servers with local Git repositories
- Geographical distribution
- Shared ACLs, users, groups

Use Optimized git clone

"Import" and register two new Git repositories:

```
$ mv linux_*.git ~/gerrit_testsite/git
$ ssh -p 29418 alexadmin@localhost gerrit flush-caches
```

Clone Linux kernel repository:

```
$ git clone ssh://alexadmin@localhost:29418/linux_orig t1
Cloning into 't1'...
remote: Counting objects: ...
```

Clone with optimized support:

```
$ git clone ssh://alexadmin@localhost:29418/linux_opt t2
Cloning into 't2'...
remote: ...
```

JGit and Gerrit magic.
Unlike anything else.

Scaling Up JGit
Wed. 11:15 - 11:50
Harborview Ballroom 2

Access Controls

Access Control Overview

- Fine-grained, per-reference controls of creations, deletions, reviews, reads, and writes, ...

- Server-wide capabilities allow administrators to delegate some administrative duties.

Permission Assignment

Permissions can only be granted to groups.
Users *(or groups)* may be added to groups.

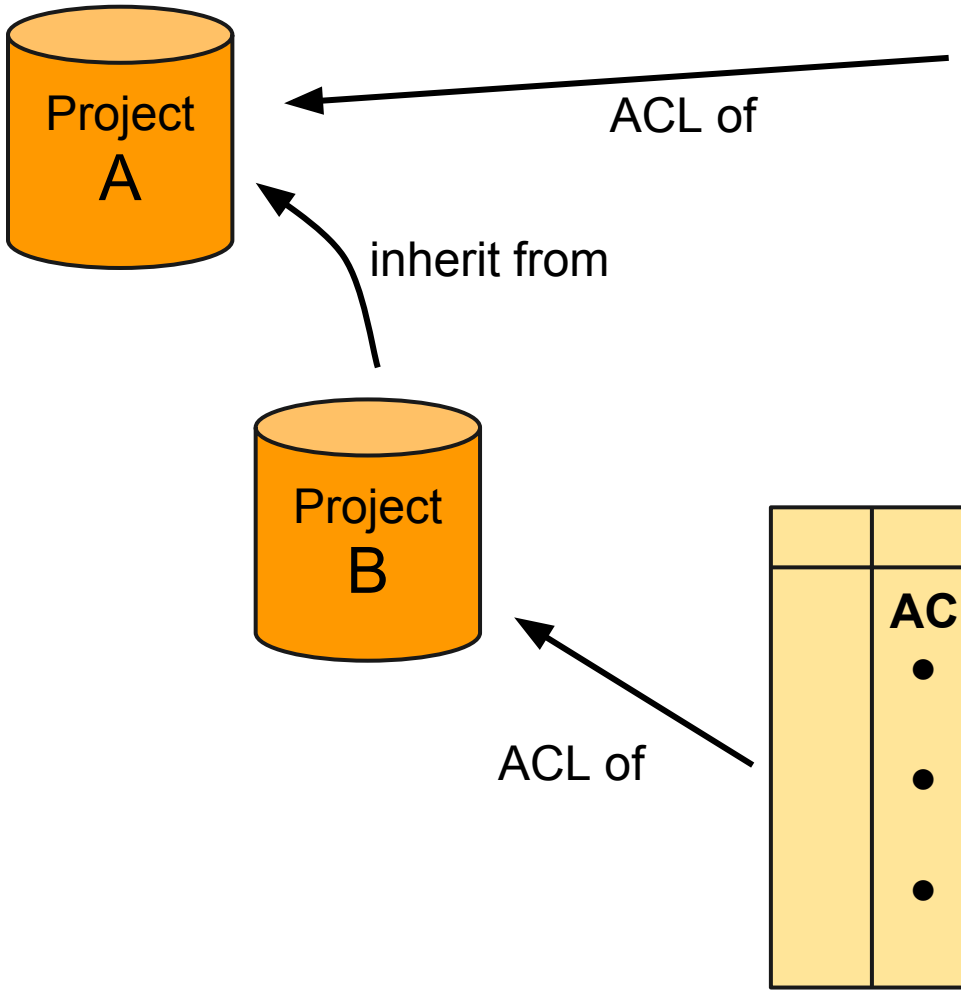
Special built-in groups:

Anonymous Users	-- users not signed in *(anyone! everyone!)*
Registered Users	-- any signed in user *(everyone!)*
Project Owners	-- substitution placeholder *(more later)*

Default groups:

Administrators	-- granted all capabilities
Non-Interactive Users	-- assigned to

Inheritance



ACL

- "Registered users" can read all branches.
- "Super users" can approve changes.

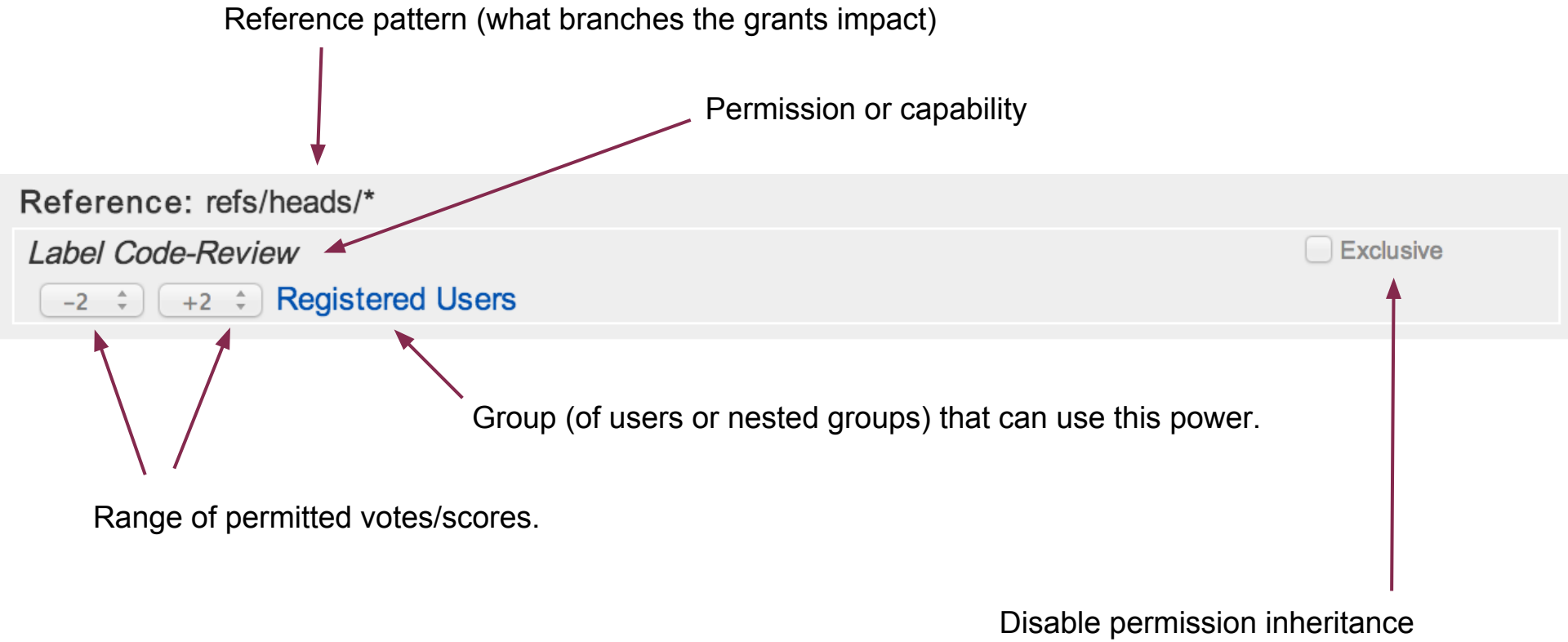
ACL

- "Registered users" can read all branches.
- "Super users" can approve changes.
- "CI system user" can mark changes as verified.

An ACL Entry


Reference: refs/heads/*
Label Code-Review
Exclusive
-2
+2
Registered Users

An ACL Entry


Reference pattern (what branches the grants impact)
Permission or capability
Reference: refs/heads/*
Label Code-Review
Exclusive
-2
+2
Registered Users
Group (of users or nested groups) that can use this power.
Range of permitted votes/scores.
Disable permission inheritance

References in ACLs

Read	refs/heads/*
Push	refs/for/refs/heads/*
Label Code-Review	refs/heads/master
Label Verify	^refs/heads/release-[0-9]\.[0-9]$

Exclusive Flag


```
git ls-remote --heads ssh://alexadmin@localhost:
29418/myproject
.... refs/heads/master
.... refs/heads/sekret


git ls-remote --heads ssh://developer@localhost:
29418/myproject
.... refs/heads/master
```

Example: Contributor

refs/heads/*
ALLOW Read
Label Code-Review −1..+1

Permits accessing branches
Can see all changes

Enable users to vote thumbs up or thumbs down on changes.
No real impact, just warm-fuzzy.

refs/for/refs/heads/*
ALLOW Push

Enable contributions

Example: Maintainer

refs/heads/*

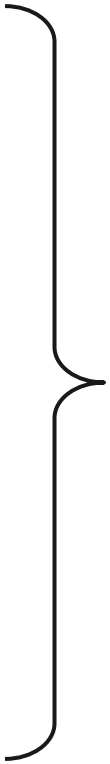
Label Code-Review −2..+2

Label Verify −1..+1

ALLOW Submit

refs/tags/*

ALLOW Push Annotated Tag



− Can reject a change
+ Can accept a change

Submit to project branch, adding change to next version, and to the project's permanent history.

Create new release tags

Example: CI system

refs/heads/*

Label Verified −1..+1

Continuous integration can vote:
−1 = change did not compile
+1 = compiles, passed tests

refs/tags/nightly/*

ALLOW Push Annotated Tag

Create only nightly tags

Exercise – Typical ACL

- All registered users can upload and submit changes and have the full range of the Verified and Code-Review labels.

- The “Wizards” group can also push annotated tags and are the only ones who may approve (i.e. Code Review +2) and submit changes for branches whose names end with “-release”.

Read Permission

Controls who can:

- see a branch in the web UI
- access the branch via Git
  - fetch
  - clone
  - push
- see changes uploaded for the branch
- see changes merged into the branch

Push Permission

Controls who can:

- upload changes for review
  - if reference is refs/for/refs/heads/...
- push commits directly into a branch
  - if reference is refs/heads/...
- delete a branch or non-fast forward update
  - if “Force” flag is set

Submit Permission

Controls who can:

Submit a change that has required approvals

Unlike Push refs/heads/... enforces workflow

Forge {Author,Committer}

Controls who can:

Forge Author	upload commits by another
Forge Committer	upload commits or tags by another

Compares email address stored by Git...
... with email addresses in Gerrit.

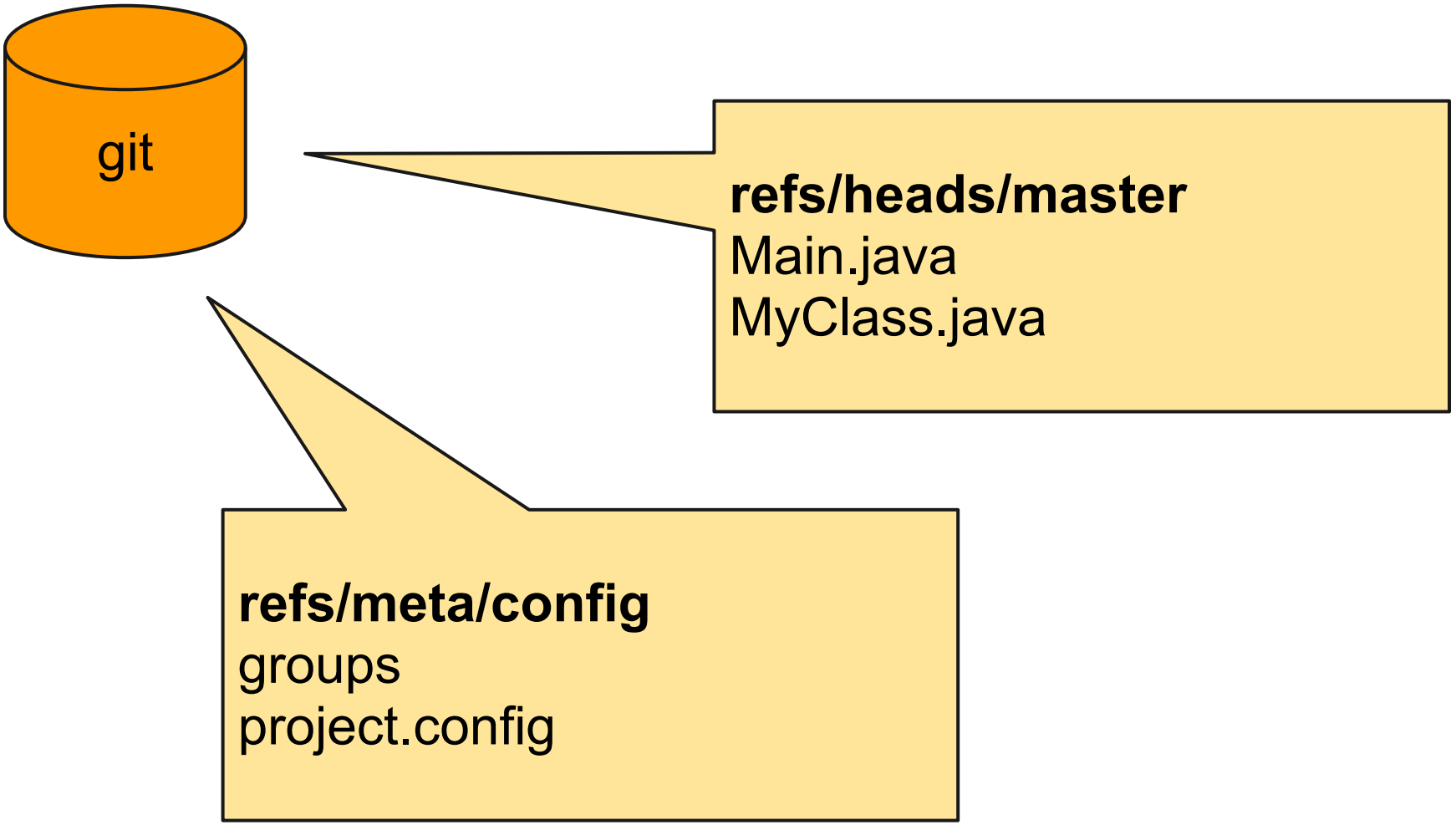
Push Annotated Tag

Controls who can

- push annotated tags (“git tag -a”).

Tags live in the refs/tags namespace, so assigning this to refs/tags/* makes sense. refs/heads/* does not.

ACL Data Store



project.config

```
[project]
   description = Rights inherited by all other projects
   state = active

[access "refs/*"]
   read = group Administrators
   read = group Anonymous Users
   forgeAuthor = group Registered Users

[access "refs/for/refs/*"]
   push = group Registered Users

[access "refs/heads/*"]
   label-Code-Review = -1..+1 group Registered Users
```

Exercise – Propose ACL

Mortal users that can't change the permissions directly can propose changes in the web UI and have them reviewed like any other code change.

Let's try it out! Update some permissions but instead of clicking *Save Changes*, click *Save for Review*.

You'll get an error message. Why?

Exercise – Explore refs/meta/config

Update project configuration through refs/meta/config:

```
$ git init cfg ; cd cfg
$ git remote add origin ssh://developer@localhost:29418/All-Projects
$ git pull origin refs/meta/config
$ vi project.config
$ git commit -a -m 'Updated permissions'
$ git push origin HEAD:refs/meta/config
```

Try doing this for a “real” project (instead of a permission-only project like All-Projects).

What existing tool can you use to batch edit these files?

Advanced Workflows

Default Rules

Define default rules in Prolog:

```
$ git init rules ; cd rules
$ git remote add origin ssh://adminalex@localhost:29418/myproject
$ git pull origin refs/meta/config
$ vi rules.pl
```

```
submit_rule(submit(CR, V)) :-
  gerrit:max_with_block(-1, 1, 'Verified', V),
  gerrit:max_with_block(-2, 2, 'Code-Review', CR).
```

```
$ git add rules.pl
$ git commit -a -m 'Default Prolog rules'
$ git push origin HEAD:refs/meta/config
```

Verified-by Skips Verified

Update rules.pl to make Verified label optional:

```
submit_rule(submit(CR)) :-
  gerrit:commit_message_matches('^Verified-by: '),
  !,
  gerrit:max_with_block(-2, 2, 'Code-Review', CR).

submit_rule(submit(CR, V)) :-
  gerrit:max_with_block(-1, 1, 'Verified', V),
  gerrit:max_with_block(-2, 2, 'Code-Review', CR).
```

Create a change with commit message:

```
Greetings EclipseCon, Boston

Verified-by: Person on my left
```

Notice Verified label is not used in web interface.

Need CQ >= 2 Lines

Append to project.config to declare CQ label in web UI

```
[label "CQ"]
    value = 0 CQ Pending
    value = +1 CQ Approved
```

Replace rules.pl to check for 2 new lines and need CQ

```
submit_rule(submit(CR, V, CQ)) :-
  gerrit:commit_stats(_, Inserted, _), Inserted >= 2, !,
  base(CR, V),
  gerrit:max_with_block(0, 1, 'CQ', CQ).

submit_rule(submit(CR, V)) :-
  base(CR, V).

base(CR, V) :-
  gerrit:max_with_block(-1, 1, 'Verified', V),
  gerrit:max_with_block(-2, 2, 'Code-Review', CR).
```

No Self Approvals

Replace rules.pl to ignore self-approved Code-Review +2

```
submit_rule(submit(CR, V, N)) :-
  gerrit:commit_author(A),
  gerrit:max_with_block(-2, 2, 'Code-Review', label(_, ok(A))),
  N = label('Non-Author Code-Review', need(_)),
  base(CR, V),
  !.

submit_rule(submit(CR, V)) :-
  base(CR, V).

base(CR, V) :-
  gerrit:max_with_block(-1, 1, 'Verified', V),
  gerrit:max_with_block(-2, 2, 'Code-Review', CR).
```

Thanks! Other Cool Stuff

Plugins/Extensions	External group systems; User avatars; External accounts *(future)*
Integrations	Jenkins/Hudson CI; JIRA/Bugzilla; index by bug numbers Mylyn Reviews Gerrit Connector; Skalli Project Portal
stream-events	Monitor server activity in real time, react to events (e.g. CI)
Draft changes	Private reviews, double check diff before review
Workflow by file	Add (or simplify) review flow for Documentation, etc.
REST API	Stable JSON based REST API
	CI systems can insert comments directly on lines of code.

code.google.com/p/gerrit

Scaling Gerrit

(continued)

Memory Usage

Gerrit Loves Memory

Git data

- Paged into Java heap on demand
- Custom block cache implementation
- Works around Java mmap() limitations

ACLs, Accounts, Groups, Diffs

- Cached to speed up authorization, display

Java Container

container.heapLimit

Bytes of memory JVM can use for Gerrit (Java -Xmx flag), e.g. `24g`

container.javaOptions

Additional flags to pass to JVM, e.g. `-d64 -server`

core.packedGitOpenFiles

Defines open files ulimit. Gerrit sets process ulimit to `MAX(1024, packedGitOpenFiles * 2)`
Minimum ulimit selected by gerrit.sh is 1024.

JGit Cache Settings

core.packedGitLimit (10 MiB)

Max. bytes to load and cache in memory from pack files.

core.packedGitOpenFiles (128)

Max. number of pack files to have open at once.

core.packedGitWindowSize (8 kiB)

Bytes of a pack file to load into memory in a single read operation.

core.deltaBaseCacheLimit (10 MiB)

Max. bytes for caching base objects that multiple deltafied objects reference.

core.streamFileThreshold (25% of heap)

Largest object size, in bytes, allocated as a contiguous byte array.

Servers should set this to be larger than the size of their common big files.

Gerrit Cache Settings

cache.directory	Local disk cache, holds data across restarts
cache.<name>.maxAge	Maximum age to keep an entry in the cache
cache.<name>.memoryLimit	Total cost (size) of entries to retain in memory
cache.<name>.diskLimit	Total size in bytes of cache entries on disk

http://localhost:8080/Documentation/config-gerrit.html#_a_id_cache_a_section_cache

```
$ ssh -p 29418 alexadmin@localhost gerrit show-caches
Gerrit Code Review        2.6-rc0                  now    00:09:18   PDT
                                                uptime    44 min 17 sec

  Name                          |Entries              |  AvgGet |Hit Ratio|
                                |   Mem   Disk   Space|         |Mem  Disk|
--------------------------------+---------------------+---------+---------+
  accounts                      |     2               |   2.0ms | 99%     |
  accounts_byemail              |     3               | 700.0us | 62%     |
...
```

Scaling Gerrit

Concurrent Requests

Database Connections

database.poolLimit (8)

Limit on open database connections; resources of DB must be considered.

Need at least sshd.threads + 4 database connections to avoid deadlocks.

database.poolMinIdle (4)

Minimum number of connections to keep idle in the pool.

database.poolMaxIdle (4)

Maximum number of connections to keep idle in the pool.

database.poolMaxWait (30sec)

Max. time request processing thread will wait to acquire a database connection.

Jetty HTTP Daemon

httpd.acceptorThreads (2)

Worker threads dedicated to accepting new incoming TCP connections.

httpd.minThreads / httpd.maxThreads (25)

Minimum/Maximum number of spare threads to keep.

httpd.maxQueued (50)

Maximum number of connections which can enter the queue waiting for a worker thread.

httpd.maxWait (5min)

Maximum time for a project clone, fetch or push request over smart HTTP.

SSH Daemon

sshd.threads (1.5xCPUs)

Number of threads to execute SSH command requests.

sshd.batchThreads (0)

Number of threads for SSH command requests from Non-Interactive Users.

sshd.streamThreads (1+CPUs)

Number of threads for formatting events to asynchronous streaming clients.

sshd.commandStartThreads (2)

Number of threads used to start new SSH commands.

sshd.maxConnectionsPerUser (64)

Maximum number of concurrent SSH sessions for a user account.

receive-pack (git push)

receive.maxObjectSizeLimit (0)

Maximum Git object size that receive-pack will accept.

Use this to prevent pushing objects which are too large to Gerrit.

receive.threadPoolSize (number of CPUs)

Number of threads to process received Git data.

Database updates use changeUpdateThreads.

receive.changeUpdateThreads (*disabled*)

Number of threads to perform database metadata updates.

Slower databases can benefit from parallel updates if users frequently push multiple changes.

receive.timeout (2min)

Upper bound on time taken to process change data received from client.

Exercise – Inspect State

Display active client SSH connections

```
$ ssh -p 29418 localhost gerrit show-connections
```

Display the background work queues, including replication

```
$ ssh -p 29418 localhost gerrit show-queue
```

Display current cache statistics

```
$ ssh -p 29418 localhost gerrit show-caches
```

Flush some/all server caches from memory

```
$ ssh -p 29418 localhost gerrit flush-caches --all | --list | --cache
<NAME>
```

Find command details here

http://localhost:8080/Documentation/cmd-index.html

Exercise – Monitor Activity

Monitor events occurring in real time

```
$ ssh -p 29418 localhost gerrit stream-events
```

keep this open and try this from a second shell:
- upload a new patchset
- submit a change

Explore the server logs at '~/gerrit_testsite/logs'

If a command hangs `jstack` and `jconsole` are your friend

Scaling Gerrit

Going Faster

Exercise – Manage Git GC

$ `ssh -p 29418 localhost gerrit gc [--all] <NAME> ...`

--all run gc for all projects sequentially

GC configuration:

defined in ~/.gitconfig of the system user that runs the Gerrit server
or in specific <project>.git/config

Exercise:

- type `$ git help config` and explore the gc configuration options
- run gc on a project (if it's new you may need to tweak the gc configuration)
- find the gc log and check gc statistics

Speed Up git clone

Make our project history larger:

```
$ for a in {1..99};do echo $a >a;git add a;git commit -m $a;done
$ git push ~/gerrit_testsite/git/myproject.git master
```

Clone to a new client:

```
$ git clone ssh://alexadmin@localhost:29418/myproject t2
Cloning into 't2'...
remote: Counting objects: 300, done
```

Make it faster:

```
$ ssh -p 29418 alexadmin@localhost gerrit gc myproject
$ git clone ssh://alexadmin@localhost:29418/myproject t3
Cloning into 't3'...
remote: Total 300 (delta 0), reused 300 (delta 0)
```

Distribute Load

Replicate to other servers

- Enable replication plugin
- Configure remotes in site_path/etc/replication.config

Gerrit Slaves

- Only serve Git over SSH
- Enforce same Read access permissions, using same user and groups.

```
container.slave = true
database.database = ... same address as master ...
cache.<name>.maxAge = 15min    # or some other low value
```