How to survive with large mono-repos in Gerrit

Luca Milanesio
Gerrit Code Review Maintainer
GerritForge
About GerritForge

Founded in the UK

HQ in London with presence in Europe (GerritForge Ltd) and the USA (GerritForge Inc.)

Committed to OpenSource and to Gerrit Code Review since 2009
What BIG means for you?
Large mono-repo false solutions: **git-lfs**

Git LFS often seen as “one-size-fits-all” solution
- Makes the repo smaller
- Reduces bandwidth utilization
- Less disk occupied locally

Dark-side of Git LFS
- Requires history rewrite
- Static definition of BLOB size to put onto LFS
- Does not solve large mono-repos issues in Gerrit
Git data model: recap

refs/heads/main

### first commit
- fdf4fc
- test.txt
- tree

### second commit
- cac8ca
- test.txt
- tree

### third commit
- 1a410e
- new.txt
- test.txt
- tree

bak

- 1f7a7a "version 2"
- fa49b0 "new file"
- 83baae "version 1"
Git data model: 

```
refs/heads/main
```

ref

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```

```
refs/heads/main
```
Git data model: recap

tree

objects

refs/heads/main
BIG repository: **dimensions**

![Graph showing a positive correlation between number of refs, number of objects, and disk space.]
BIG repository: **dimensions**

- Number of refs
- Number of objects

**Disk space**

**Git LFS**
Measuring repo metrics: *git-sizer*

Developed by GitHub and available on: https://github.com/github/git-sizer

Sample execution on the Chromium repository

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Level of concern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall repository size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* Commits</td>
<td>11.3 M</td>
<td>******************</td>
</tr>
<tr>
<td>* Count</td>
<td>7.01 G</td>
<td>!!!!!!!!!!!!!!!!!!!</td>
</tr>
<tr>
<td>* Trees</td>
<td>18.1 M</td>
<td>!!!!!!!!!!!!!</td>
</tr>
<tr>
<td>* Count</td>
<td>68.9 G</td>
<td>!!!!!!!!!!!!!!!!!!!!</td>
</tr>
<tr>
<td>* Total tree entries</td>
<td>1.42 G</td>
<td>!!!!!!!!!!!!!!!!!!!!</td>
</tr>
<tr>
<td>* Blobs</td>
<td>11.2 M</td>
<td>?????????????????</td>
</tr>
<tr>
<td>* Total size</td>
<td>1.06 T</td>
<td>!!!!!!!!!!!!!!!!!!!</td>
</tr>
<tr>
<td>* Annotated tags</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>* References</td>
<td>2.54 M</td>
<td>!!!!!!!!!!!!!!!!!!!!</td>
</tr>
<tr>
<td>Biggest objects</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* Commits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* Maximum size             [1]</td>
<td>6.63 M</td>
<td>!!!!!!!!!!!!!!!!!!!</td>
</tr>
<tr>
<td>* Maximum parents          [2]</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>* Trees</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* Blobs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>History structure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* Maximum history depth</td>
<td>879 k</td>
<td>*</td>
</tr>
<tr>
<td>* Maximum tag depth</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Biggest checkouts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* Number of directories      [5]</td>
<td>32.8 k</td>
<td>!!!!!!!!!!!!!!!</td>
</tr>
<tr>
<td>* Maximum path depth</td>
<td>20</td>
<td>**</td>
</tr>
<tr>
<td>* Maximum path length</td>
<td>284 B</td>
<td>**</td>
</tr>
<tr>
<td>* Number of files             [8]</td>
<td>385 k</td>
<td>!!!!!!!!!!!!</td>
</tr>
<tr>
<td>* Total size of files         [9]</td>
<td>5.25 G</td>
<td>****</td>
</tr>
<tr>
<td>* Number of symbolic links    [10]</td>
<td>363</td>
<td></td>
</tr>
<tr>
<td>* Number of submodules       [11]</td>
<td>114</td>
<td>*</td>
</tr>
</tbody>
</table>
Large mono-repo: Gerrit-specific issues

Very large number of refs

- Each change has:
  - refs/changes/NN/NNNN/\(<ps>\)
  - refs/changes/NN/NNNN/meta

- Example of medium-size Gerrit repo: 100k changes, 10 patch-sets per change
  - \(100k \times 11 = 1.1M\) refs
Large mono-repo: Gerrit-specific issues

Very old repositories = always increasing number of refs
- Merged changes stay in the history
- Abandoned changes stay in the history
Large mono-repo: Gerrit-specific issues

Mono-repo = lots of people working on the same repository
• 500 developers creating 1 change per day
• 1 year = 500 * 254 (working days) = 127k changes

What a project having 5 years of history looks like?
• 5 * 127k changes = 635k changes
• Assuming 10 patch-sets per change:
  635k * 11 ~ 7M refs !!!!
Large mono-repo: example = Chromium

- > 500k changes
- > 2.5M refs
- > 42M objects
- > 60GB of disk space
Large number of refs: problems

Storage:
- Loose refs => millions of files on the filesystem
- Packed refs => single file getting bigger over time

Access times:
- Loose refs: very slow on NFS: listing of directories is notoriously problematic
- Packed refs: very slow in any condition, but NFS is a killer
- Bitmaps are not helping, as they are not computed for changes
Large number of refs: benchmarks

JGit benchmark on refs lookup and filtering:
- Created by L.Milanesio, extended and measured by M.Sohn
- https://bugs.eclipse.org/bugs/show_bug.cgi?id=576165

### getExactRef benchmark:
- trustFolderStat=true
- trustFolderStat=false

<table>
<thead>
<tr>
<th>repo</th>
<th>refs</th>
<th>getExactRef [µs/op]</th>
<th>getExactRef [µs/op]</th>
</tr>
</thead>
<tbody>
<tr>
<td>jgit</td>
<td>34,396</td>
<td>71.549</td>
<td>17,027.238</td>
</tr>
<tr>
<td>All-Users-sap02</td>
<td>35,129</td>
<td>71.924</td>
<td>18,886.679</td>
</tr>
<tr>
<td>gerrit</td>
<td>42,216</td>
<td>71.902</td>
<td>20,700.065</td>
</tr>
<tr>
<td>All-Users-sap01</td>
<td>94,456</td>
<td>70.948</td>
<td>65,594.445</td>
</tr>
<tr>
<td>go</td>
<td>179,967</td>
<td>72.800</td>
<td>101,953.274</td>
</tr>
<tr>
<td>All-Users-Eclipse</td>
<td>304,258</td>
<td>64.669</td>
<td>156,373.621</td>
</tr>
<tr>
<td>large-repo-sap-02</td>
<td>1,526,738</td>
<td>73.575</td>
<td>873,704.665</td>
</tr>
<tr>
<td>large-repo-sap-01</td>
<td>1,904,551</td>
<td>73.744</td>
<td>1,246,369.098</td>
</tr>
<tr>
<td>chromium</td>
<td>2,540,957</td>
<td>101.902</td>
<td>2,155,881.338</td>
</tr>
</tbody>
</table>

### getRefsByPrefix benchmark:
- trustFolderStat=true
- trustFolderStat=false

<table>
<thead>
<tr>
<th>repo</th>
<th>getRefsByPrefix [µs/op]</th>
<th>getRefsByPrefix [µs/op]</th>
</tr>
</thead>
<tbody>
<tr>
<td>jgit</td>
<td>1,967.296</td>
<td>21,699.063</td>
</tr>
<tr>
<td>All-Users-sap02</td>
<td>451.381</td>
<td>19,661.338</td>
</tr>
<tr>
<td>gerrit</td>
<td>2,630.773</td>
<td>27,255.809</td>
</tr>
<tr>
<td>All-Users-sap01</td>
<td>2,396.374</td>
<td>64,784.646</td>
</tr>
<tr>
<td>go</td>
<td>14,213.907</td>
<td>125,918.218</td>
</tr>
<tr>
<td>All-Users-Eclipse</td>
<td>22,297.707</td>
<td>200,519.193</td>
</tr>
<tr>
<td>large-repo-sap-02</td>
<td>106,812.654</td>
<td>1,018,671.689</td>
</tr>
<tr>
<td>large-repo-sap-01</td>
<td>142,563.930</td>
<td>1,233,735.363</td>
</tr>
<tr>
<td>chromium</td>
<td>265,694.897</td>
<td>2,447,755.469</td>
</tr>
</tbody>
</table>
### Large number of refs: benchmarks

JGit benchmark on refs lookup and filtering:

- Created by L.Milanesio, extended and measured by M.Sohn
- [https://bugs.eclipse.org/bugs/show_bug.cgi?id=576165](https://bugs.eclipse.org/bugs/show_bug.cgi?id=576165)

<table>
<thead>
<tr>
<th>repo</th>
<th>refs</th>
<th>getExactRef trustFolderStat=true [µs/op]</th>
<th>getExactRef trustFolderStat=false [µs/op]</th>
</tr>
</thead>
<tbody>
<tr>
<td>jgit</td>
<td>34,396</td>
<td>71.549</td>
<td>17,027.238</td>
</tr>
<tr>
<td>All-Users-sap02</td>
<td>35,129</td>
<td>71.924</td>
<td>18,886.679</td>
</tr>
<tr>
<td>gerrit</td>
<td>42,216</td>
<td>71.902</td>
<td>20,700.065</td>
</tr>
<tr>
<td>All-Users-sap01</td>
<td>94,456</td>
<td>70.948</td>
<td>65,594.445</td>
</tr>
<tr>
<td>go</td>
<td>179,967</td>
<td>72.800</td>
<td>101,953.274</td>
</tr>
<tr>
<td>All-Users-Eclipse</td>
<td>304,258</td>
<td>64.669</td>
<td>156,373.621</td>
</tr>
<tr>
<td>large-repo-sap-02</td>
<td>1,526,738</td>
<td>73.575</td>
<td>873,704.665</td>
</tr>
<tr>
<td>large-repo-sap-01</td>
<td>1,904,551</td>
<td>73.744</td>
<td>1,246,360.098</td>
</tr>
<tr>
<td>chromium</td>
<td>2,540,957</td>
<td>101.902</td>
<td>2,155,881.338</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>repo</th>
<th>getRefsByPrefix trustFolderStat=true [µs/op]</th>
<th>getRefsByPrefix trustFolderStat=false [µs/op]</th>
</tr>
</thead>
<tbody>
<tr>
<td>jgit</td>
<td>1,967.296</td>
<td>21,699.063</td>
</tr>
<tr>
<td>All-Users-sap02</td>
<td>451.381</td>
<td>19,661.338</td>
</tr>
<tr>
<td>gerrit</td>
<td>2,630.773</td>
<td>27,255.809</td>
</tr>
<tr>
<td>All-Users-sap01</td>
<td>2,396.374</td>
<td>64,784.646</td>
</tr>
<tr>
<td>go</td>
<td>14,213.907</td>
<td>125,918.218</td>
</tr>
<tr>
<td>All-Users-Eclipse</td>
<td>22,297.707</td>
<td>200,519.193</td>
</tr>
<tr>
<td>large-repo-sap-02</td>
<td>106,812.654</td>
<td>1,018,671.689</td>
</tr>
<tr>
<td>large-repo-sap-01</td>
<td>142,563.930</td>
<td>1,233,735.363</td>
</tr>
<tr>
<td>chromium</td>
<td>265,694.897</td>
<td>2,447,755.469</td>
</tr>
</tbody>
</table>

Over 2s for a single ref lookup or filter!!!
Large number of objects: problems

Storage and bitmaps:
• Loose objects => tens millions of files on the filesystem
• Packfiles => bitmaps rely on a single packfile, which is getting bigger over time

Access times:
• Loose refs: very slow on NFS: listing of directories is notoriously problematic
• Packed refs: JGit is quite clever to be fast on NFS
Large number of objects: problems with packfiles

The struggle with packfiles:
• Objective: one single packfile with all objects
• Reality: hundreds of packfiles going up and down

Rationale:
• Single packfile obtained by a Git GC with pruning
• Pruning not practical for on-line Git GC (risk of removal of in-flight pushed objects)
• Every Git push creates a new packfile
Large number of objects: search-for-reuse/sizes

Digression on Git clone phases:
(https://gerrit-documentation.storage.googleapis.com/Documentation/3.4.1/logs.html#_sshd_log)

• Refs advertisement and negotiation
• Bitmaps lookup optimization
• Search-for-reuse / search-for-sizes
• Counting
• Compressing / Writing to the wire
Large number of objects: search-for-reuse/sizes

Digression on Git clone phases:
(https://gerrit-documentation.storage.googleapis.com/Documentation/3.4.1/logs.html#_sshd_log)

- Refs advertisement and negotiation
- Bitmaps lookup optimization
- **Search-for-reuse / search-for-sizes**
- Counting
- Compressing / Writing to the wire

This is a killer for large number of objects / packfiles
search-for-reuse/sizes: complexity

For each object, scan all packfiles looking for the best delta:
• 10M objects
• 100 packfiles
• 10M * 100 = 1BN of scans
What are YOUR problems with large mono-repos?
Large mono-repo: problems

- Disk space (server and client)
- Network bandwidth
- Long clone and fetch times
- Infinite refs negotiation
- High system load
- Timeouts
- Clucking git operations of cloned repo (logs, diffs)
- ...
Large number of refs: reftable

Available since Gerrit v3.1 (JGit 5.6)
Implemented by Shawn on Google’s JGit backend
Implemented for the OpenSource JGit by Han-Wen (thanks 😊)

Fully available on Gerrit v3.5 with the command to convert repos to ref-table storage

⚠️ Not yet available on Git (Han-Wen is working on it)

More details:
https://www.git-scm.com/docs/reftable
Large number of refs: refs caching (WIP)

M. Sohn working on an in-memory cache for refs: (https://git.eclipse.org/r/c/jgit/jgit/+/186205)

<table>
<thead>
<tr>
<th>repo</th>
<th>refs</th>
<th>getExactRef trustFolderStat=false [µs/op]</th>
<th>getExactRef cached [µs/op]</th>
</tr>
</thead>
<tbody>
<tr>
<td>jgit</td>
<td>34,396</td>
<td>17,027.238</td>
<td>0.646</td>
</tr>
<tr>
<td>All-Users-sap02</td>
<td>35,129</td>
<td>18,886.679</td>
<td>0.908</td>
</tr>
<tr>
<td>gerrit</td>
<td>42,216</td>
<td>20,700.065</td>
<td>0.744</td>
</tr>
<tr>
<td>All-Users-sap01</td>
<td>94,456</td>
<td>65,594.445</td>
<td>1.200</td>
</tr>
<tr>
<td>go</td>
<td>179,967</td>
<td>101,953.274</td>
<td>0.954</td>
</tr>
<tr>
<td>All-Users-Eclipse</td>
<td>304,258</td>
<td>156,373.621</td>
<td>1.067</td>
</tr>
<tr>
<td>large-repo-sap-02</td>
<td>1,526,738</td>
<td>873,704.665</td>
<td>1.800</td>
</tr>
<tr>
<td>large-repo-sap-01</td>
<td>1,904,551</td>
<td>1,246,369.098</td>
<td>1.912</td>
</tr>
<tr>
<td>chromium</td>
<td>2,540,957</td>
<td>2,155,881.338</td>
<td>1.636</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>repo</th>
<th>getRefsByPrefix trustFolderStat=false [µs/op]</th>
<th>getRefsByPrefix cached [µs/op]</th>
</tr>
</thead>
<tbody>
<tr>
<td>jgit</td>
<td>21,699.063</td>
<td>35.448</td>
</tr>
<tr>
<td>All-Users-sap02</td>
<td>19,661.338</td>
<td>6.381</td>
</tr>
<tr>
<td>gerrit</td>
<td>27,255.809</td>
<td>35.070</td>
</tr>
<tr>
<td>All-Users-sap01</td>
<td>64,784.646</td>
<td>35.594</td>
</tr>
<tr>
<td>go</td>
<td>125,918.218</td>
<td>365.114</td>
</tr>
<tr>
<td>All-Users-Eclipse</td>
<td>200,519.193</td>
<td>295.683</td>
</tr>
<tr>
<td>large-repo-sap-02</td>
<td>1,018,671.689</td>
<td>1,543.187</td>
</tr>
<tr>
<td>large-repo-sap-01</td>
<td>1,233,735.363</td>
<td>1,995.160</td>
</tr>
<tr>
<td>chromium</td>
<td>2,447,755.469</td>
<td>2,462.512</td>
</tr>
</tbody>
</table>
Large number of refs: refs filtering

Available as libModule from Gerrit v2.16 onwards
(https://gerrit.googlesource.com/modules/git-refs-filter/)

• Mimic the “hideRefs” config in Git
  (https://git-scm.com/docs/git-config/2.32.0#Documentation/git-config.txt-transferhideRefs)

• Define a new custom permission for refs-filtering of
  - refs/changes/NN/NNNN<meta
  - closed changes
  - other custom refs patterns
Large number of refs: refs filtering – case study

Real-life case-study of refs-filter:

- Repository with 1.5M refs
- Mirror clone without refs-filter: 45 mins
- Mirror clone with refs-filter: 3 mins (15x improvement)

Needs enabling of the persistent change cache, otherwise the change meta-data lookup eats up all the advantages.
Large number of objects: **search-for-reuse deadline**

Introduced by Ponch in JGit master

(https://git.eclipse.org/r/c/jgit/jgit/+/181355)

- Define a maximum deadline for search-for-reuse to complete pack.
  
  searchForReuseTimeout = 60s

- When deadline expires, a non-optimal object representation is taken (may transfer more data)

- Performance improvement observed:
  
  without deadline: 20 mins, with deadline: 5 mins (4x improvement)
Alternatives: repository pruning

Do you really need all the review history online?

• Two copies of the repository: full-history and pruned-history

• What to prune?
  • Auto-abandon changes (e.g. 1 month) and delete abandoned changes not resumed for a long time (e.g. 3 months)
  • Collapse merged patch-sets into latest
Q&A: excited about the future of Gerrit?
Wants to know more?

GerritForge.com/contact