BFG Repo-Cleaner

Removes large or troublesome blobs like git-filter-branch does, but faster. And written in Scala

View project on GitHub

```
$ bfg --strip-blobs-bigger-than 100M --replace-text banned.txt repo.git
```

an alternative to git-filter-branch

The BFG is a simpler, faster alternative to `git-filter-branch` for cleansing bad data out of your Git repository history:

- Removing Crazy Big Files
- Removing Passwords, Credentials & other Private data

The `git-filter-branch` command is enormously powerful and can do things that the BFG can't - but the BFG is much better for the tasks above, because:

- **Faster** : 10 - 720x faster
- **Simpler** : The BFG isn't particularily clever, but is focused on making the above tasks easy
- Beautiful : If you need to, you can use the beautiful Scala language to customise the BFG. Which has got to be better than Bash scripting at least some of the time.

Usage

First clone a fresh copy of your repo, using the `--mirror` flag:

```
$ git clone --mirror git://example.com/some-big-repo.git
```

This is a bare repo, which means your normal files won't be visible, but it is a full copy of the Git database of your repository, and at this point you should make a backup of it to ensure you don't lose anything.

Now you can run the BFG to clean your repository up:

```
$ java -jar bfg.jar --strip-blobs-bigger-than 100M some-big-repo.git
```

The BFG will update your commits and all branches and tags so they are clean, but it doesn't physically delete the unwanted stuff. Examine the repo to make sure your history has been updated, and then use the standard `git gc` command to strip out the unwanted dirty data, which Git will now recognise as surplus to requirements:

```
$ cd some-big-repo.git
$ git reflog expire --expire=now --all && git gc --prune=now --aggressive
```

Finally, once you're happy with the updated state of your repo, push it back up (note that because your clone command used the `--mirror` flag, this push will update all refs on your remote server):

```
$ git push
```

At this point, you're ready for everyone to ditch their old copies of the repo and do fresh clones of the nice, new pristine data. It's best to delete all old clones, as they'll have dirty history that you don't want to risk pushing back into your newly cleaned repo.

Examples

In all these examples `bfg` is an alias for `java -jar bfg.jar`.

Delete all files named 'id_rsa' or 'id_dsa' :

```
$ bfg --delete-files id_{dsa,rsa}  my-repo.git
```

Remove all blobs bigger than 50 megabytes :

```
$ bfg --strip-blobs-bigger-than 50M  my-repo.git
```

Replace all passwords listed in a file (prefix lines 'regex:' or 'glob:' if required) with `***REMOVED***` wherever they occur in your repository :

```
$ bfg --replace-text passwords.txt  my-repo.git
```

Remove all folders or files named '.git' - a reserved filename in Git. These often become a problem when migrating to Git from other source-control systems like Mercurial :

```
$ bfg --delete-folders .git --delete-files .git  --no-blob-protection  my-repo.git
```

For further command-line options, you can run the BFG without any arguments, which will output text like this.

Your current files are sacred...

The BFG treats you like a reformed alcoholic: you've made some mistakes in the past, but now you've cleaned up your act. Thus the BFG assumes that your latest commit is a good one, with none of the dirty files you want removing from your history still in it. This assumption by the BFG protects your work, and gives you peace of mind knowing that the BFG is only changing your repo history, not meddling with the current files of your project.

By default the `HEAD` branch is protected, and while its history will be cleaned, the very latest commit (the 'tip') is a protected commit and its file-hierarchy won't be changed at all.

If you want to protect the tips of several branches or tags (not just HEAD), just name them for the BFG:

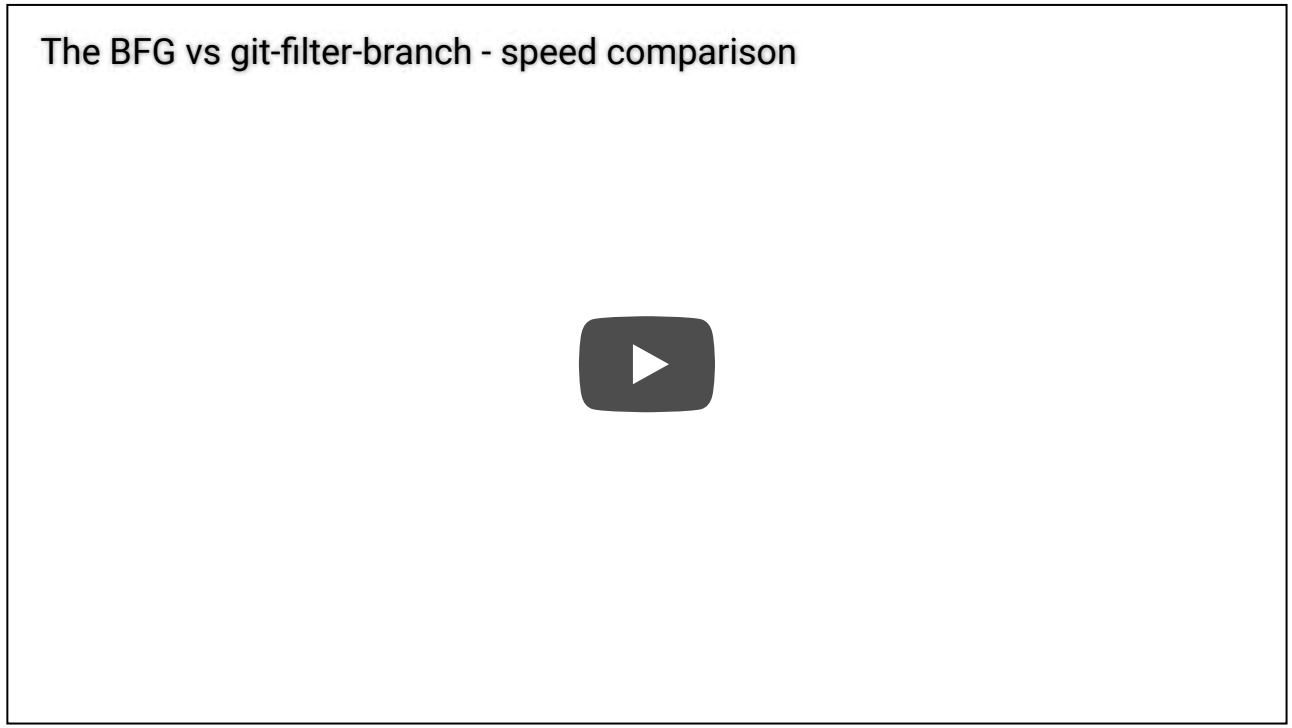
```
$ bfg --strip-biggest-blobs 100 --protect-blobs-from master,maint,next repo.git
```

Note:

- Cleaning Git repos is about completely eradicating bad stuff from history. If something 'bad' (like a 10MB file, when you're specifying `--strip-blobs-bigger-than 5M`) is in a protected commit, it won't be deleted - it'll persist in your repository, **even if the BFG deletes if from earlier commits**. If you want the BFG to delete something you need to make sure your current commits are clean.
- Note that although the files in those protected commits won't be changed, when those commits follow on from earlier dirty commits, their commit ids will change, to reflect the changed history - only the SHA-1 id of the filesystem-tree will remain the same.

Faster...

The BFG is 10 - 720x faster than `git-filter-branch`, turning an overnight job into one that takes less than ten minutes.



BFG's performance advantage is due to these factors:

- The approach of `git-filter-branch` is to step through every commit in your repository, examining the complete file-hierarchy of each one. For the intended use-cases of The BFG this is wasteful, as we don't care where in a file structure a 'bad' file exists - we just want it dealt with. Inherent in the nature of Git is that every file and folder is represented precisely once (and given a unique **SHA-1** hash-id). The BFG takes advantage of this to process each and every file & folder exactly once - no need for extra work.
- Taking advantage of the great support for parallelism in **Scala** and the JVM, the BFG does multi-core processing by default - the work of cleaning your Git repository is spread over every single core in your machine and typically consumes 100% of capacity for a substantial portion of the run.
- All action takes place in a single process (the process of the JVM), so doesn't require the frequent fork-and-exec-ing needed by `git-filter-branch` 's mix of Bash and C code.

Feedback

> I tried deleting using several "how to" blog entries for git filter-branch, but wasn't successful. Then tried The BFG; worked like a champ - very cool tool!
> — **Bill Hunt**, CTO at **OptTown**

> I found The BFG Repo-Cleaner and ran it to clean up some large files, and was amazed by the performance.
> — **Jason Frey**, Software Engineer at **Red Hat**

> I was able to shrink the current repository down to ~500 megabytes in about 10 minutes when using this tool. My hand crafted scripts clock in at 615 megabytes in 3 days time for comparison.
> — **Elliot Glaysher**, Google Software Engineer on **Google Chrome**

> The BFG was simple to set up and so fast that I had to ask Roberto, "Is that it?" and check for myself... it worked exactly as intended.
> — **Nicholas Tollervey**, Developer at **The Guardian**

> Roberto's creations (**Agit** and The BFG) are both very cool ;-)
> — **Junio C Hamano**, **Maintainer of Git**

Also see more feedback on Twitter...

Requirements

- The **Java Runtime Environment** (Java 7 or above - BFG **v1.12.3** was the last version to support Java 6)

That's it - the Scala library and all other dependencies are folded into the downloadable jar.

Links...

- **Rewriting Git project history with The BFG** - a blogpost for The Guardian
- **GitMinutes** podcast interview
- **Git Going Faster... with Scala** - talk for ScalaDays 2014, later Parleys **Presentation of the Day**
- **InfoQ interview**
- **Questions tagged** `git-rewrite-history` on Stack Overflow

License

Download v1.13.0
The BFG is by Roberto Tyley, the author of Prout, gu:who, Agit and the packager of Spongy Castle. Twitter Google+ PGP

This page was generated by GitHub Pages using the Architect theme by Jason Long.