Filter-Branch

This chapter was inspired by the Grails repo-splitting script from Jeff Scott Brown of SpringSource.

Pruning out folders from a repo

`filter-branch` is commonly used on a clone of the repo to split a too-large repo into smaller ones.

`filter-branch` Reference Page

Filter Branch Command Documentation

Process

We'll start with `bigrepo` and create a new repo that contains only `c`. If we wanted to split into multiple repos, we would simply clone for each desired final repo and run a `filter-branch` on each.

Preparation

This is a destructive command and thus we need to clone the repository before we start operating on it. We will effectively be creating a new repository and leaving the old repository behind.

Since git can optimize local-disk clones of repositories with hardlinks, we want to create a clone that is entirely separate from the original one:

```
git clone --no-hardlinks /path/to/originalrepo newrepo
```

Pruning

With a Subdirectory

If we wish to only save the files in the `c` directory while purging all branches, we can run the following command. Oddly lightweight tags are kept here. This *relocates* all files in the `c` subfolder to the *root* of the new repository. This is usually not what we want. Users typically want to specify what to prune away, leaving all other folder structures intact.

```
git filter-branch --subdirectory-filter c HEAD
```

The same command can have an additional option to keep all the branches.

- the `--` separates filter-branch options from revision options
- the `--all` rewrites all branches and tags
- the `--prune-empty` removes commits that would no longer have any content

  ```
  git filter-branch —prune-empty —subdirectory-filter c HEAD — —all
  ```

With the Tree and Checkouts

Alternatively, we can use a tree filter which *prunes away* the selected folder or filename pattern using shell commands. It checks out each commit and runs the command against it. This allows for the full power of any shell command to be leveraged, including greps.

- `-f` force the `rm` or else commits where that file didn't exist would fail on the shell command.
- `--prune-empty` removes any commits that have no files (blank, empty) after the shell command performs its surgery.

  ```
  git filter-branch —tree-filter “rm -rf c” —prune-empty HEAD
  ```

With the Index

A variation on this is the `--index-filter` which is much faster. It only operates on the DAG, not on checkouts and the staging area. It only uses git commands, not full shell commands.

- `--cached` is supplied to leave untracked files alone. Only operate on tracked files.
- `--ignore-unmatch` is supplied to allow the command to always succeed for every commit, even if the file didn't exist.

  ```
  git filter-branch —index-filter “git rm -r -f —cached —ignore-unmatch c”
  —prune-empty HEAD
  ```

And this variation that adds the `--tag-name-filter` and `-- --all` which keeps the `.git/refs/heads/original/refs/tags` folder, keeps all references to the original tags in the `/git/info/refs` file, and re-writes the tag to `.git/refs/tags/AGOODPOINT` and `.git/refs/heads/addingonefile` branch.

- `--tag-name-filter cat` re-writes all tags

  git filter-branch —index-filter “git rm -r -f —cached —ignore-unmatch c” —prune-empty —tag-name-filter cat — —all

Cleanup

Remove any `original` refs

Many of the `filter-branch` invocations will create a `.git/refs/original` folder to allow for a restore after a `filter-branch` execution. These are still first class references and will cause the objects to be retained. If you have reviewed the results of the filter and are satisfied with the result, remove these refs so that the objects can be cleaned up in the next steps.

```
git for-each-ref --format="%(refname)" refs/original/ | xargs -n 1 git update-ref -d
```

Expire all entries from the reflog

Keep in mind that filter-branch only removes references from the DAG (history), but doesn’t purge the `.git/objects` directory. Git always partitions repository cleanup into a separate step that is usually run on a scheduled basis.

Expire all the old `reflog` entries now instead of at the scheduled time:

```
git reflog expire --expire=now --all
```

Reset Working Directory

Reset to the “new” (possibly different) `HEAD` state now that entries have been removed with `filter-branch`.

```
git reset --hard
```

Garbage Collection

Garbage collect any orphaned entries. From the git-gc man page, please note that:

> `git gc` tries very hard to be safe about the garbage it collects. In particular, it will keep not only objects referenced by your current set of branches and tags, but also objects referenced by the index, remote-tracking branches, refs saved by git `filter-branch` in `refs/original/`, or reflogs (which may reference commits in branches that were later amended or rewound).

If you are expecting some objects to be collected and they aren't, check all of those locations and decide whether it makes sense in your case to remove those references.

- `--prune=now` Prune all unreachable (orphaned) objects from the DAG without a separate invocation of prune

  git gc —aggressive —prune=now