Git Deep Dive

Éibhear Ó hAnluain

[2018-06-25 Mon]

Outline

Introduction

Assumes basic knowledge: `clone`, `add`, `commit`, `merge`, `pull`, `push`
Assumes comfort with the command line: `git` on Linux; `git bash`, `git cmd` on Windows

Paddling	config, fetch-and-merge, merge approaches
Snorkling and Scuba diving	refs, `HEAD`, annotated tags, submodules, `blame`, signing and verifying commits
Submarining	the structure of a commit, "Content Addressable Filesystem", git objects
Unmanned submersibles	the reflog, `fsck` and `gc`, finding and dealing with specific git objects

About Éibhear Ó hAnluain

- Solutions Architect
  - eibhear.ohanluain@version1.com
- Software engineer since 1994
- Using revision control since 1994: subversion, git, IBM/Rational/Atria ClearCase, RCS, VSS, CVS, Serena SCM, Serena PVCS, SCCS
- Introduced revision control in multiple environments
- Multiple revision-control migrations
  - RCS -> subversion
  - VSS -> subversion
  - CVS -> git

Paddling along the coast

By Frédéric de Goldschmidt www.frederic.net - Own work, CC BY 3.0,

`https://commons.wikimedia.org/w/index.php?curid=3972438`

Configuration

config

- Precedence: local (default), global, system, defaulted.

```
git config --local --list
git config --global --list
git config --system --list
git config [--system|--global|--local] <key> \
    <value>
git config -e
```

fetch and merge, not pull

fetch and merge, not pull

- "`git pull` is just `git fetch` and then `git merge`"
- When should you use `git pull`? When...
  1. ... there are no local changes to be committed or pushed; and
  2. you know the changes `pull` will make.
- Why?: `pull` doesn't stop to let you review the merged-in changes in case it didn't go well.
- Suggestions:
  - `git fetch --prune` followed by `git diff` followed by `git merge`
  - `git pull --prune --no-commit` followed by `git commit`
    - Allows review before commit, but doesn't apply if merge is `fast-forward`

fetch and merge, not pull

fetch and merge, not pull

- Other suggestions
  - `--prune`: Indicates the branches that have been removed from the remote, suggesing to you to remove the local branches
  - `git branch -va`: Allows you to get a full view of the local and remote branches

Merging approaches

Merging, merging strategies

fast-forward	where the tip commit of the target branch is a direct ancestor of the tip commit of the source branch. Merge is to re-locate the target branch label; no changes to files.
Merge	to apply the changes done on the source branch onto the target branch
rebase	to replay the changes made onto one commit onto another.

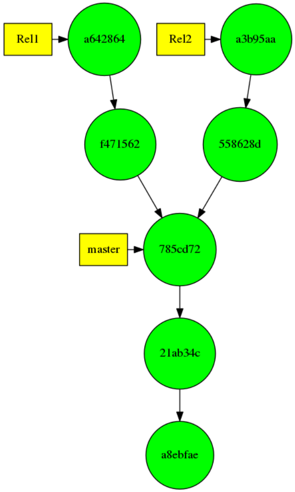
Merge strategies:

Strategies

- resolve
- recursive
- octopus
- ours
- subtree

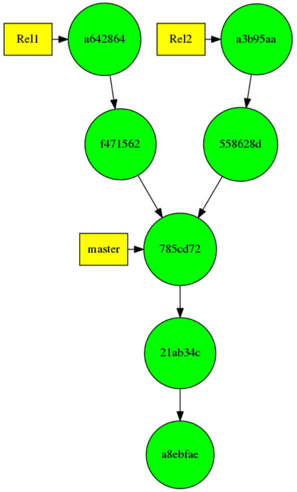
Merging approaches: fast-forward

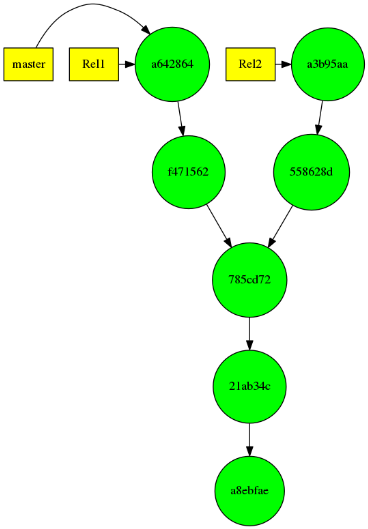
Fast forward



Merging approaches: fast-forward

Fast forward



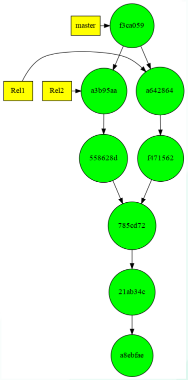


Merging approaches: merging strategies

Merging strategies

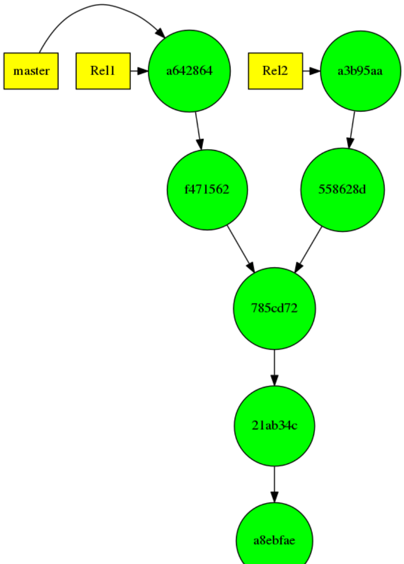
Git automatically decides which strategy to use, but the user can also specify:

- `recursive`: The default. For merging one branch into another, seeking to ensure previous merges are handled properly.
- `resolve`: For merging one branch into another. Safe and fast.
- `octopus`: For merging 3 or more branches.
- `ours`: For dealing with old branches; preserves the state of the target branch
- `subtree`: For merging trees that are at different levels.



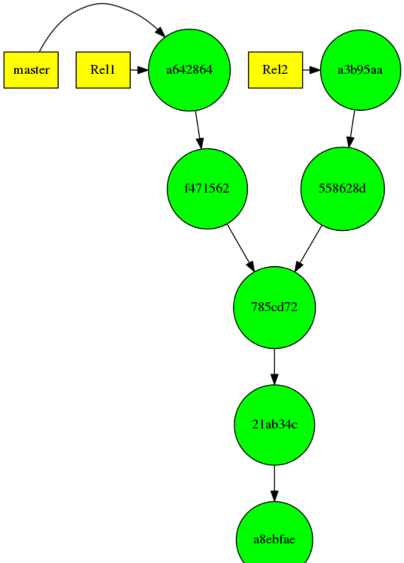
Merging approaches: Rebase

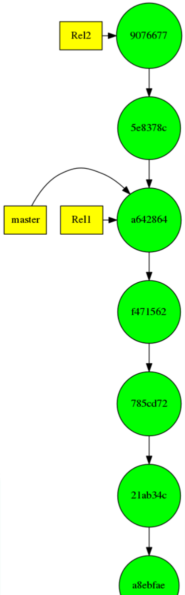
Rebase



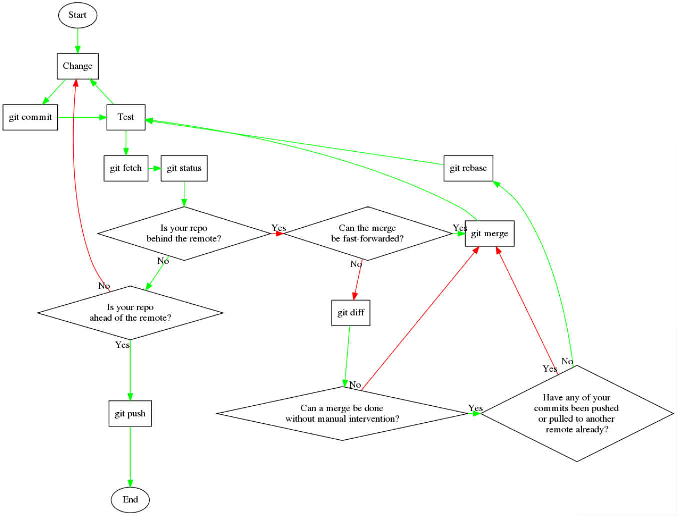
Merging approaches: Rebase

Rebase





Merging appraoches: Decision tree



Start
Change
git commit
Test
git fetch
git status
Is your repo behind the remote?
Yes
No
Can the merge be fast-forwarded?
Yes
No
git merge
git rebase
Is your repo ahead of the remote?
No
Yes
git diff
git push
Can a merge be done without manual intervention?
No
Yes
Have any of your commits been pushed or pulled to another remote already?
Yes
No
End

A little deeper

By Ggerdel - Foto de: Gustavo Gerdel (BAB Buceo) [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons

Refs

Refs

- Branches

```
$ ls -1 .git/refs/heads/*
.git/refs/heads/gitlab_10.x_upgrade
.git/refs/heads/gitlab-ce-8.17
.git/refs/heads/gitlab-ce-9.1.4-defectFix
.git/refs/heads/master

$ cat .git/refs/heads/gitlab-ce-8.17
c63767d0b9b520f36a533237624dfaa1256b463c
```

Refs

Refs

- tags

```
$ ls -1 .git/refs/tags/
gitlab-cookbook-for-10.7.3
gitlab-cookbook-initial
gitlab-cookbook-rebaseline

$ cat .git/refs/tags/gitlab-cookbook-for-10.7.3
f16b12027cef7052e09f3a483ace55999800ea5b
```

- HEAD

```
$ cat .git/HEAD
ref: refs/heads/gitlab_10.x_upgrade
```

Refs

Refs

- remotes

```
$ ls -1 .git/refs/remotes/origin/*
.git/refs/remotes/origin/ara_test
.git/refs/remotes/origin/brjones_aci_network
.git/refs/remotes/origin/gitlab_10.x_upgrade
.git/refs/remotes/origin/gitlab-ce-8.17
.git/refs/remotes/origin/gitlab-ce-9.1.4-defectFix
.git/refs/remotes/origin/HEAD
.git/refs/remotes/origin/master

$ cat .git/refs/remotes/origin/ara_test
7130b7ee374ad9f7ba784ec0b0d0b86dc99f41d4
```

HEAD

HEAD

- HEAD is a ref that points to where the commit currently checked out
- HEAD in a remote repository usually points to the default branch

```
$ cat .git/HEAD
ref: refs/heads/master
$ cat .git/refs/remotes/origin/HEAD
ref: refs/remotes/origin/REL2_0
```

Annotated tags

Annotated tags

- A normal tag is just a "ref", pointing to a commit.

  ```
  git tag Rel1.0 Rel1
  cat .git/refs/tags/Rel1.0
  git cat-file -t $(cat .git/refs/tags/Rel1.0)
  git cat-file -p $(cat .git/refs/tags/Rel1.0)
  ```
- Caveat: tags and branches use different namespaces, so temptation to tag the `Rel1` branch with `Rel1` would be high. However, this causes confusion for many git tools, so should be avoided.

Annotated tags

Annotated tags

- An annotated tag is a separate git object, that records information specfic to the tag:
  - Tag name; date; tagger; commit it's pointing to
- Advice is for annotated tags to be used for significant announcements/releases/snaphots.

```
git tag -a -m "Formal release of 1.0" \
    Rel1.0.prod Rel1
cat .git/refs/tags/Rel1.0.prod
git cat-file -t $(cat .git/refs/tags/Rel1.0.prod)
git cat-file -p $(cat .git/refs/tags/Rel1.0.prod)
```

blame

`blame`

- Identifies the commit, commit author and date for each line in a file.
- Doesn't give information for lines that have been removed or replaced.

Tag and commit signing

Tag and commit signing and verification

- PGP/GnuPG (gpg): expects familiarity with public/private key encryption.
- Sign tags and commits with private key to assure integrity

```
git commit -Seibhear.ohanluain@version1.com \
    -m "Update to information.md"
git cat-file -p $(cat .git/refs/heads/master)
git tag -s -u eibhear.ohanluain@version1.com \
    -m "Release 2." Rel2.0 Rel2
git cat-file -p $(cat .git/refs/tags/Rel2.0)
```

- Verify commits and tags with public key

```
git tag -v Rel2.0
git log --show-signature -1
```

Switch on the Sonar

By © Hans Hillewaert, CC BY-SA 4.0,

`https://commons.wikimedia.org/w/index.php?curid=385705`

Git Objects – blobs

Blob (file)

A file

```
$ cat ~/.profile | git hash-object --stdin
c9db4591825bd7a918df686ff04aeb3a87d3bda0
```

Git Objects – trees

Tree

A listing of files and other tree objects containing the following information on each: access mode (similar to UNIX permissions); type (e.g. `tree`, `blob`); SHA1; file/directory name

```
$ git cat-file -p \
      2c4e4782bcb6b13a0e11b6961004dec8745e9d35
040000 tree bcd2824258ddf007dae7f88da7d727fb3894691b
    Astro
100644 blob 94aea57573b92d9188a3df4cf748b60efd968803
    MyHelloWorldBean.java
100644 blob c55118d1afb2be6a6d0f728814c084d54e97db14
    MyHelloWorldServlet.java
```

Git Objects – commits

Commit

Information on a commit event:

- The SHA1 of the top-level tree of the project.
- The parent commit (or commits, if this commit is the result of a merge)
- The details of the author of the code that is being applied with the commit (including date and time)
- The details of the person who applied the commit (including date and time)
- The commit gpg signature, if present
- The commit comment

Git Objects – commits

Commit

```
$ git cat-file -p \
      37c5611a177c9eafbd17e4302b6d644434b1042b
tree 40b6262f3f3f5fc8cb8a0c78ca558a683dfc2323
parent 42608808a973b8e0c4a4b0105c2317d81b12851f
parent 36d56f097ca81f06d77f46cbde3fc10cbf6639f9
author Master O'Theuniverse \
    <master.otheuniverse@example.com> \
    1529840879 +0100
committer Master O'Theuniverse \
    <master.otheuniverse@example.com> \
    1529840879 +0100

REV18_5 now live
```

Git Objects – tags

Tag

A tag points to a commit object, and an annotated tag contains additional information

- What object it's pointing at and its type
- The name of the tag
- Who applied the tag (including date and time)
- A tag comment

Git Objects – tags

Tag

```
$ git cat-file -p \
      88a4c18867ccb1d7c398f285460d8abab3964e75
object 66d8336c2770d0d1cea3dcb0175611edb5e69f69
type commit
tag Rel1.0.a
tagger Éibhear Ó hAnluain \
    <eibhear.ohanluain@version1.com> \
    1529847749 +0100

An annotated tag for the Rel1 release

$ git cat-file -t \
      66d8336c2770d0d1cea3dcb0175611edb5e69f69
commit
```

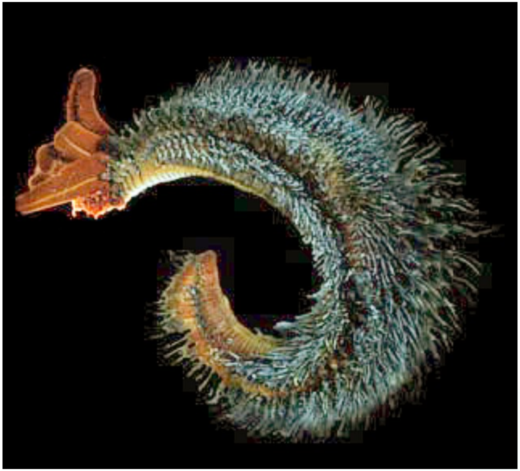
"Content Addressable Filesystem"

"Content Addressable Filesystem"

As the IDs of objects are based on their contents, they are located in the git database according to that name

```
git rev-list --all --objects
git cat-file -p \
    b5aea839bc89a0c7931af469ff9c145be18854d7
git cat-file -p \
    b5aea839bc89a0c7931af469ff9c145be18854d7 | \
    git hash-object --stdin
ls -l .git/objects/b5/aea839bc89a0c7931af469ff9c145be18
```

Exotic creatures



By National Science Foundation (University of Delaware College of Marine Studies) - http://www.nsf.gov/od/lpa/news/press/01/pr0190.htm, Public Domain,

The reflog

The reflog

- A log of all the changes to HEAD
- Local only: not involved in pulls, pushes or fetches.
- Useful for recovering accidentally removed commits
- Completely clear the reflog

```
git reflog expire --expire=now \
    --expire-unreachable=now --verbose --all
```

fsck and gc

fsck and gc

fsck	Analyse connectivity of the objects in the git database. Useful for finding objects that aren't of any use any more.
gc	"Garbage collector". Compress and pack objects, remove "danlging" and unreachable objects

Useful commands

Useful commands

- Find the largest object in your git database

```
git cat-file \
    --batch-check='%(objecttype) %(objectname) %(ob
    --batch-all-objects | sed -n 's/^blob //p' | \
    sort -n --key=2
```

- List all objects in your git database

```
git rev-list --all --objects
```

- Determine the filename related to a blob object

```
git rev-list --objects --all | grep <objectID>
```

- Determine commits that reference that file

```
git log --follow -- "<fileNameAndPath>"
```

Permanently removing a file from your git db

- e.g. contains password information, v. large file not required, etc.
- For each branch. . .
  - Check it out

    ```
    git checkout <branch>
    ```
  - Remove the file from all commits

    ```
    git filter-branch --tree-filter \
        'rm -f <filePathAndName>' \
        --prune-empty HEAD
    ```
  - Clean up the original refs

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

Permanently removing a file from your git db

Permanently removing a file from your git db

- . . . then . . .
  - Clear out the reflog

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

    The reflog maintains references to commits and objects you may want to remove.
  - Collect the garbage, regardless of age

    ```
    git gc --prune=now
    ```
  - Sanity-check all objects

    ```
    git fsck --unreachable --no-reflogs
    ```

Other interesting concepts

Other interesting concepts

- Porcelain vs Plumbing: The commands that we use vs the commands that they use.
  - Porcelain: `init`, `clone`, `fetch`, `push`, `add`, `commit`, etc.
  - Plumbing: `hash-object`, `cat-file`, `write-tree`, `count-objects`, etc.
- submodules: to link a separate git repository into yours
- fast-import: to import data into a git repository from other, non-git sources (e.g. cvs, subversion, etc.)

Resources

Resources

The git book `https://git-scm.com/book/en/v2/`

The git manual
```
git help --all
git help merge
git help cat-file
man git-hash-object
```

Stack overflow `https://stackoverflow.com/questions/tagged/git`

Emacs 24.5.1 (Org mode 9.1.12)