An Intermediate Guide to Git

Chase Zhang

Splunk Inc.

May 21, 2017

Table of Contents

Advanced Techniques

Warming Up

```
# Show current status
git status

# Show commit logs on current branch
git log
```

Advanced Techniques

Warming Up

```
# Add changes under current directory
git add .

# Reset status of HEAD
git reset HEAD

# Commit
git commit -m "commit message"
```

Advanced Techniques

Warming Up

```
# Checkout a branch
git checkout branch_name

# Checkout a commit
git checkout 23abef

# Revert unstaged changes
git checkout .
```

Advanced Techniques

Warming Up

```
# Create a new branch
git checkout -b branch_name

# Rename a branch
git branch -m new_branch_name

# Delete a branch
git branch -d branch_name

# Merge from another branch
git merge another_branch_name
```

Advanced Techniques

Warming Up

```
# Add a remote upstream
git remote add origin git@remote.repo.url

# Fetch from upstream repo
git fetch

# Pull from upstream repo
git pull

# Push onto upstream repo
git push
```

Advanced Techniques

Warming Up

If you are not familiar with previous commands, please refer to some tutorials on the Internet:

- Pro Git[1]
- Git Documentation[2]
- Tutorials from Atlassian[3]

[1] https://git-scm.com/book
[2] https://git-scm.com/doc
[3] https://www.atlassian.com/git/tutorials/

Advanced Techniques

History Management

```
# Revert commit softly. Changes will be kept
git reset --soft HEAD~3

# Revert commit hardly. Changes will lose
git reset --hard HEAD~3

# Revert commit mixed. Stage will be clear,
# changes will be kept (default)
git reset --mixed HEAD~3
```

Advanced Techniques

History Management

```
# Accidentally reverted a commit?
# No panic, there is a way to restore it
git reflog

# Checkout the hash point you'd like be back to
git checkout 42efba

# Replace the origin branch
git checkout -B master
```

Advanced Techniques

History Management

```
# Pick a commit from other branch, without merging it
git cherry-pick 892bfe

# Pick several commits at once
git cherry-pick 892bfe..42fdab

# If there is a conflict, you have to resolve it
# And use this to continue
git cherry-pick --continue
# Or use this to abort
git cherry-pick --abort
```

Advanced Techniques

History Management

A more convenient way is to use rebase:

```
# Rebase HEAD with a previous commit in the same branch
git rebase HEAD^10

# Rebase current branch to master
git rebase master
```

Advanced Techniques

History Management

Use interactive rebase, we can change the history easily!

```
git rebase -i HEAD~3
```

Advanced Techniques

History Management

What can interactive rebase do:

- Pick or drop a commit
- Change commit orders
- Change commit message
- Modify edit contents
- Squash two commints into one

Advanced Techniques

History Management

It is highly recommended that we always use rebase instead of merge when syncing local developing branches with upstream.

Advanced Techniques

History Management

Question

What’s the difference between git merge and git rebase?

Advanced Techniques

History Management

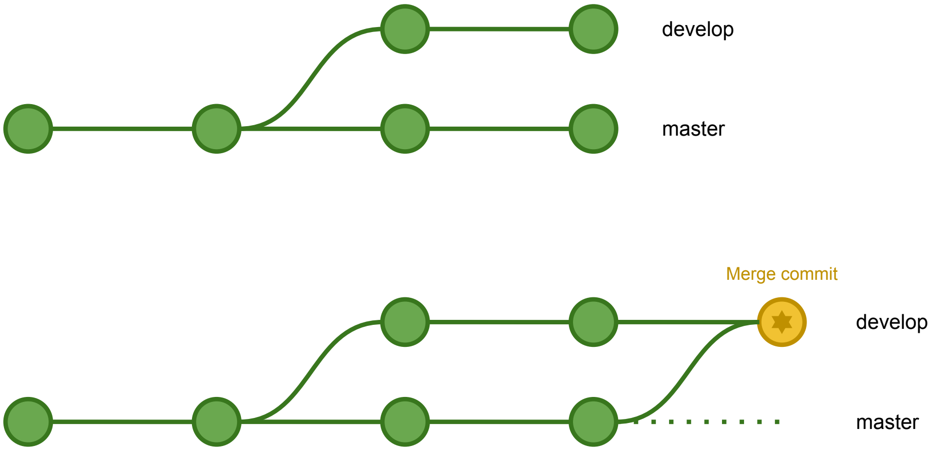


Figure: Branch Model for git merge

Advanced Techniques

History Management

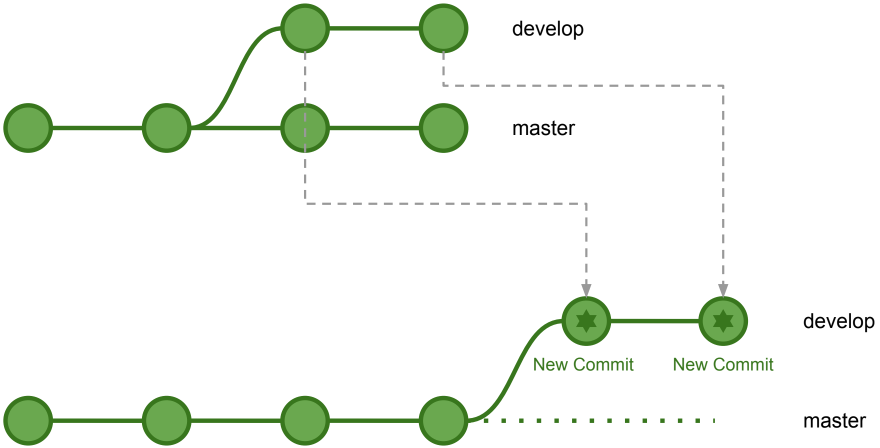


Figure: Branch Model for git rebase

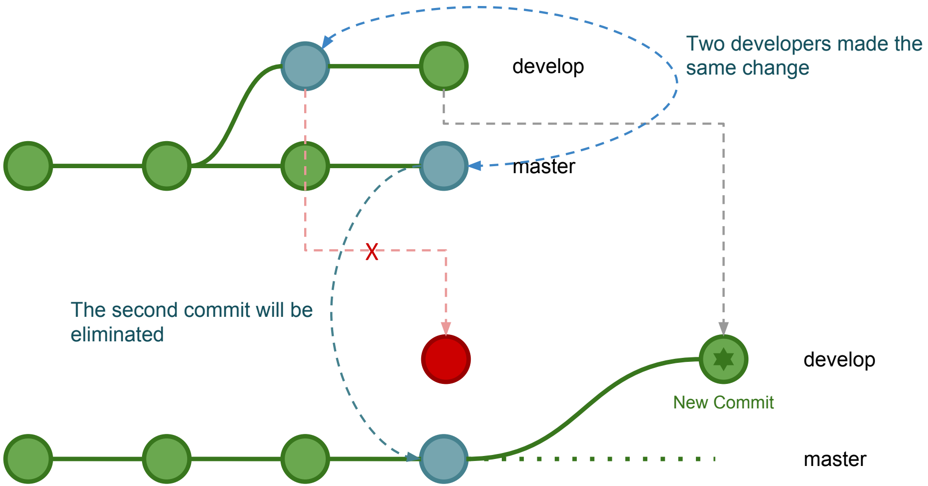
Advanced Techniques

History Management

- git merge
  - Will not change any commits in both master and develop branches
  - Will generate a new merge commit at develop branch
  - Once merged back, merge commits in develop will appears in master
- git rebase
  - Will rewrite commits from develop branch
  - Will eliminate empty commits and keep logs clean

Advanced Techniques

History Management



develop
master
Two developers made the same change
X
The second commit will be eliminated
develop
New Commit
master

Figure: git rebase will eliminate empty commits

Advanced Techniques

History Management

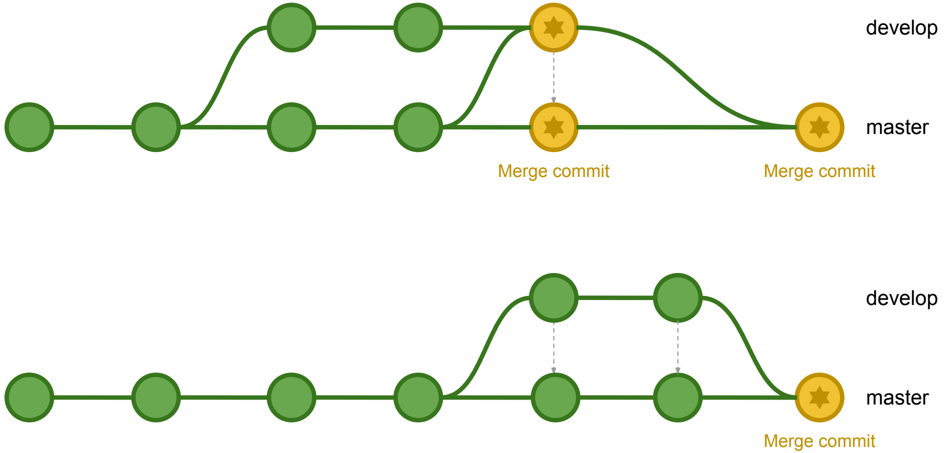


Figure: Branch flow: git merge vs. git rebase

Advanced Techniques

History Management

The golden rule of git rebase is to never use it on public branches[1].

Advanced Techniques

History Management

Additional notices:

1. Once rebase is applied, you may have to use git push -f to push local changes to remote. You should never do this on master, but it's ok for your own branches
2. git pull is actually equivalent to git fetch + git merge by default. You can override this by

```
git pull --rebase
```

Advanced Techniques

Other Toys

Some git commands you may not know:

- git grep
  - Multi threads, will be much faster than bare grep command
  - Output to less by default
- git clean
  - Clean uncommitted files
  - Can’t be reverted!
- gitk, git gui : build-in GUI client of Git

Advanced Techniques

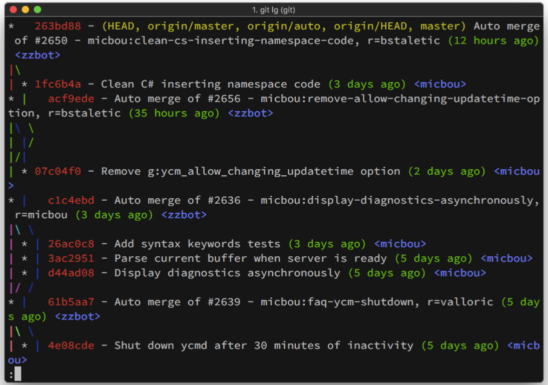
Other Toys

Let git log show tree graph other than just a commit list (see 1.5)

```
git log --graph --oneline --decorate
```

Advanced Techniques

Other Toys



```
*   263bd88 - (HEAD, origin/master, origin/auto, origin/HEAD, master) Auto merge
 of #2650 - micbou:clean-cs-inserting-namespace-code, r=bstaletic (12 hours ago)
 <zzbot>
|\
| * 1fc6b4a - Clean C# inserting namespace code (3 days ago) <micbou>
* |   acf9ede - Auto merge of #2656 - micbou:remove-allow-changing-updatetime-op
tion, r=bstaletic (35 hours ago) <zzbot>
|\ \
| |/
|/|
| * 07c04f0 - Remove g:ycm_allow_changing_updatetime option (2 days ago) <micbou
>
* |   c1c4ebd - Auto merge of #2636 - micbou:display-diagnostics-asynchronously,
 r=micbou (3 days ago) <zzbot>
|\ \
| * | 26ac0c8 - Add syntax keywords tests (3 days ago) <micbou>
| * | 3ac2951 - Parse current buffer when server is ready (5 days ago) <micbou>
| * | d44ad08 - Display diagnostics asynchronously (5 days ago) <micbou>
|/ /
* |   61b5aa7 - Auto merge of #2639 - micbou:faq-ycm-shutdown, r=valloric (5 day
s ago) <zzbot>
|\ \
| * | 4e08cde - Shut down ycmd after 30 minutes of inactivity (5 days ago) <micb
ou>
:
```

Figure: Show graph with git log

Advanced Techniques

Other Toys

It is convenient to make an alias command

```
git config alias.lg "log --color --graph --pretty=format:'%Cred%h%
    Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%
    Creset' --abbrev-commit"

git config alias.ci commit
git config alias.st status
```

Advanced Techniques

Other Toys

Toys and resources with git hooks:

- lolcommits[4]: take a photo of yourself every time you make a commit (see 1.6)
- Continous Delivery: Heroku[5], Dokku[6], Deis[7]
- git-jira-hook: Automatically retrieve JIRA number from branch name and prepend to each commit[8]

[4] https://lolcommits.github.io/
[5] https://www.heroku.com/
[6] https://github.com/dokku/dokku
[7] https://github.com/deis/workflow
[8] https://github.com/joyjit/git-jira-hook

Advanced Techniques

Other Toys

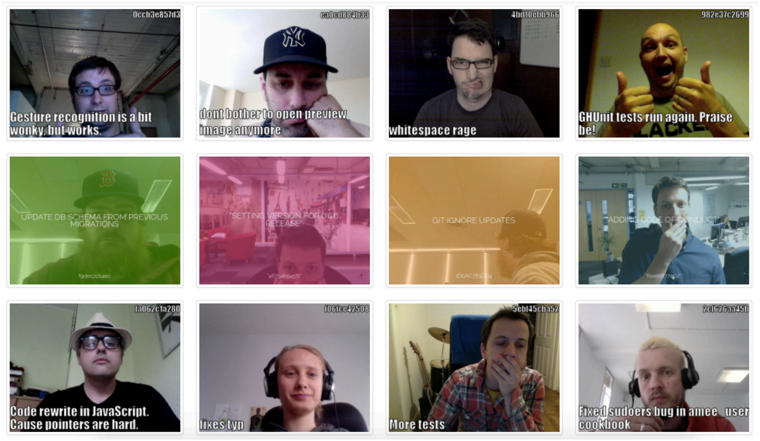

0ccb3e857d3
Gesture recognition is a bit wonky, but works.
ca0cd884b33
dont bother to open preview image anymore
4bd10ebb966
whitespace rage
982e37c2699
GHUnit tests run again. Praise be!
UPDATE DB SCHEMA FROM PREVIOUS MIGRATIONS
"SETTING VERSION FOR 0.0.6 RELEASE"
GIT IGNORE UPDATES
"ADDING CODE OF CONDUCT"
fa062c1a280
Code rewrite in JavaScript. Cause pointers are hard.
f061ce42508
fixes typ
5ebf45cba52
More tests
2cf626aa45b
Fixed sudoers bug in amee_user cookbook

Figure: lolcommits

Advanced Techniques

Other Toys

What The Commit[9]!

[9] http://whatthecommit.com/

Underneath Git Command

Git's Object Model

To describe Git's Object Model is actually the same as answering the question below because our file system is just a tree like data structure

Question

How can we make an immutable in memory tree, which we can look up its modification history and revert to previous state at will?

Underneath Git Command

Git's Object Model

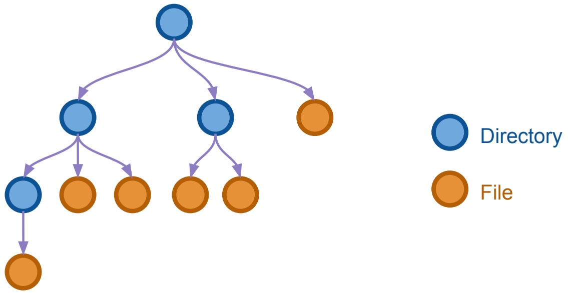


Figure: Git Tree Model

Underneath Git Command

Git's Object Model

The simplest way might be:

- For every changes we copy everything and save them somewhere
- We use an array of pointers pointing to each copies according to their order
- Once we'd like to find how the tree is like in an old time point, we look up it in the array and retrieve the copy of trees

But, it will cost too much memory and is very slow.

Underneath Git Command

Git's Object Model

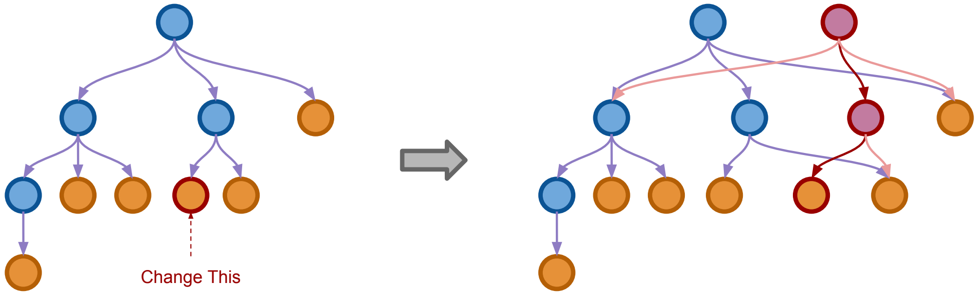


Figure: Git Object Tree

Underneath Git Command

Git's Object Model

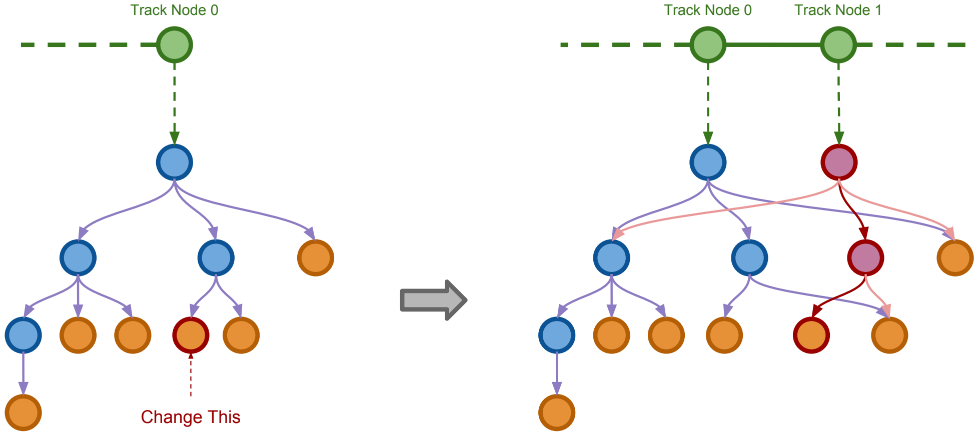


Figure: Git Object Tree with Commits

Underneath Git Command

Git's Object Model

Conclusion

We can improve our previous solution by doing the following optimizations:

1. Once there is a change, we don't copy everything, but only the path from root to the node that has been modified. This will save a lot of memory and time
2. We use a series of track nodes to preserve the order of each time point in history. Each node has a pointer to the corresponding copy of root. A track node also has a pointer to previous node
3. When we're seeking for an history state, we just find the corresponding track node, get the copy of root. With that root as tip, to traverse the tree, we'll get the snapshot of that time point

Underneath Git Command

Git's Object Model

Summary

You've already learnt Git's object model!

- Git makes an object for every directories and files and saved them into `.git/objects`
- Each TrackNode corresponding to a commit in Git's object model, commits are also saved in `.git/objects`
- Every operation you performed with Git will carefully maintain the object files and their relationships. It will make sure the objects will keep consistent with your sources directories

Underneath Git Command

Git's Internal Files

Let's have an exploration of Git's internal files (some files are omitted):

```
.git
├── HEAD
├── objects
│   ├── 0c
│   ├── 0d
│   ├── info
│   └── pack
├── refs
│   ├── heads
│  └── tags
│
```

Underneath Git Command

Git's Internal Files

- objects sub directory contains all the objects
  - Each objects are indexed by their SHA1 hash value
  - The first two characters are picked out to make a level of category directories
  - info and pack's function will be explain later
- refs sub directory contains all the pointers, like branches, tags and so on
- HEAD points to commit hash of current working dir

Underneath Git Command

Git's Internal Files

Git's object are compressed with zlib by default, we can uncompress and view its content by the following python program:

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import zlib


def main(fname):
    with open(fname) as f:
        print zlib.decompress(f.read())

if __name__ == "__main__":
    main(sys.argv[1])
```

Underneath Git Command

Git's Internal Files

A better way is just to use the method Git itself provides

```
git cat-file -p 5b67fd90081
```

Underneath Git Command

Git's Internal Files

Would like to try commiting some files manually instead of using git commit directly? Here are some commands might be useful[10]:

```
# Write a new file object
git hash-object -w test.txt

# Create a new tree object and write a tree
git update-index --add --cacheinfo 100644 \
  83baae61804e65cc73a7201a7252750c76066a30 test.txt
git write-tree

# Commit a tree
git commit-tree d8329f
```

[10] Details at: https://git-scm.com/book/en/v2/Git-Internals-Git-Objects

Underneath Git Command

Git's Internal Files

Git will keep content of working dir consistent to what HEAD ref is pointing to. Once you checkout a branch, commit or files, Git retrieve objects from its store and apply the tree content to root dir and update the HEAD

```
# Read a tree from storage and apply
# to current working directory
git read-tree 5fd87
```

Underneath Git Command

Git's Internal Files

Update and maintain working directory is costly and unsafe, as for remote repo, we usually init them with the following command to get a repo without working directory

```
git init --bare
```

Underneath Git Command

Optimizations

Once we made a change to a file, the whole new file content will be saved as a new object. Ideally we should only save the diff if a huge file has changed just a little. Git can pack your files into a format that will save more space. The pack files also will speed up object looking up and reading.
This command will also eliminate objects that is not referenced any more.

```
# Let git perform GC and pack files
git gc

# git gc will run every time you execute
git push
```

Underneath Git Command

Optimizations

Packfiles are saved under `.git/objects/packs` and metadata will be saved to `.git/objects/info`

```
find .git/objects -type f
.git/objects/bd/9dbf5aae1a3862dd1526723246b20206e5fc37
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
.git/objects/info/packs
.git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.
    idx
.git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.
    pack
```

Underneath Git Command

Optimizations

Obviously, Git's object model have two major shortcomings:

- If we're commiting a huge file, it will be very slow
- If we have too many files, it will works very slow

Underneath Git Command

Optimizations

Currently, we have some solutions to the two problems:

- GLFS (Git Large File System)[2] is an open source tool that will save huge files to external cloud storage to speed up reading and writing
- GVFS (Git Virtual File System)[3, 4] is a project from Microsoft which provide a git file system which won't download an object until it is read for the first time

Git HTTP Server

An implementation of a Git HTTP server might be simpler than your wildest imagination.

Git HTTP Server

Provide info/refs

- Before both fetch and push action, client git will send requests to info/refs first
- The client finds the minimum mount of objects to transfer through this endpoint.
- This is an RPC invoke and we need a Git client at the server side to handle it

Git HTTP Server

Provide info/refs

Requests to info/refs will have an URL parameter named service, we make use of this parameter to call a local git command and let it do the rest for us.

```
git [service] --stateless-rpc --advertise-refs
```

Git HTTP Server

Provide info/refs

We'll just stream STDOUT of previous git command calls back to the client through HTTP protocol. But firstly we have to prepend a head block

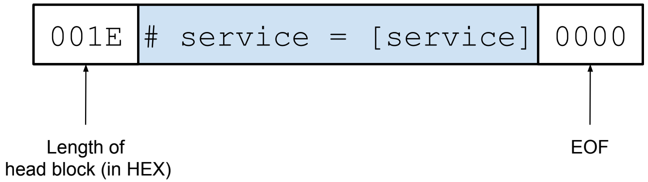


Figure: Head Block Structure

Git HTTP Server

Serving Fetch Request

Support fetching request is very simple. As the client has already known what to fetch from the info/refs call. It will fetch all files by itself and we'll just serve objects directory as static files.

Git HTTP Server

Accept Push Request

Push request is a little more complex than fetch. Although client git has already known what to push to the server, for the sake of security and concurrency. Writing action must be performed in a safer and faster way. It is an RPC invoke too.

Git HTTP Server

Accept Push Request

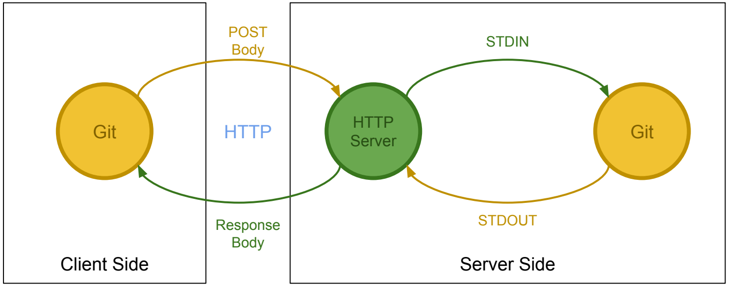


Figure: Handle Push Requests

Git HTTP Server

Accept Push Request

Push requests will have an URL parameter named command, as before, we pass this command to git directly:

```
git [command] --stateless-rpc
```

If command name is receive-pack, we have to do one step more:

```
git update-server-info
```

Git HTTP Server

That's it! We now have a workable Git HTTP Server[11][12].

[11] More info about the protocol: https://git-scm.com/book/en/v2/Git-Internals-Transfer-Protocols

[12] My implematation in Golang: https://io-meter.com/2014/07/09/simple-git-http-server/

Thank you!

References I

Atlassian.
Merging vs. rebasing.
https://www.atlassian.com/git/tutorials/merging-vs-rebasing.

Git lfs.
https://git-lfs.github.com/.

Gvfs.
https://github.com/Microsoft/GVFS.

Microsoft.
Announcing gvfs (git virtual file system).
https://blogs.msdn.microsoft.com/visualstudioalm/2017/02/03/announcing-gvfs-git-virtual-file-system/.

References II

- Mary Rose Cook.
  Git from the inside out.
  https://codewords.recurse.com/issues/two/git-from-the-inside-out.

- Git internals.
  https://git-scm.com/book/en/v2/Git-Internals-Plumbing-and-Porcelain.

- Atlassian.
  Reset, checkout, and revert.
  https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting.