

A Guide to Git

Chase Zhang

July 7, 2017

Table of Contents

Basic Operations

- Warming Up

- History Management

- Other Toys

Inside Git

- Git's Object Model

- Git's Internal Files

- Optimizations

References

Basic Usage

Warming Up

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

Basic Usage

Warming Up

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

# Reset status of HEAD
git reset HEAD

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

Basic Usage

Warming Up

```
# Checkout a branch
git checkout branch_name

# Checkout a commit
git checkout 23abef

# Revert unstaged changes
git checkout .
```

Basic Usage

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
```

Basic Usage

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
```

Basic Usage

Warming Up

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

- ▶ Pro Git¹
- ▶ Git Documentation²
- ▶ Tutorials from Atlassian³

¹<https://git-scm.com/book>

²<https://git-scm.com/doc>

³<https://www.atlassian.com/git/tutorials/>

Basic Usage

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
```

Basic Usage

History Management

```
# Accidentally reverted a commit?  
# No panic, there is a way to restore it  
git reflow  
  
# Checkout the hash point you'd like be back to  
git checkout 42efba  
  
# Replace the origin branch  
git checkout -B master
```

Basic Usage

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
```

Basic Usage

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
```

Basic Usage

History Management

Use interactive rebase, we can change the history easily!

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

Basic Usage

History Management

What can interactive rebase do:

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

Basic Usage

History Management

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

Basic Usage

History Management

Question

What's the difference between `git merge` and `git rebase`?

Basic Usage

History Management

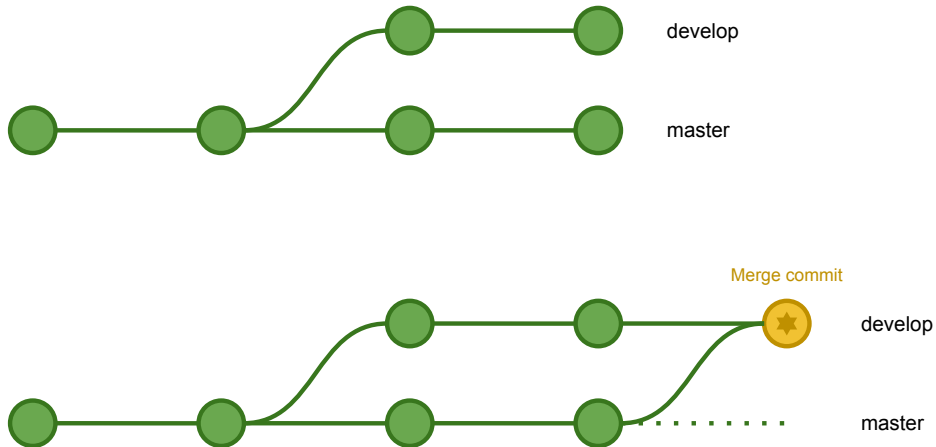


Figure: Branch Model for git merge

Basic Usage

History Management

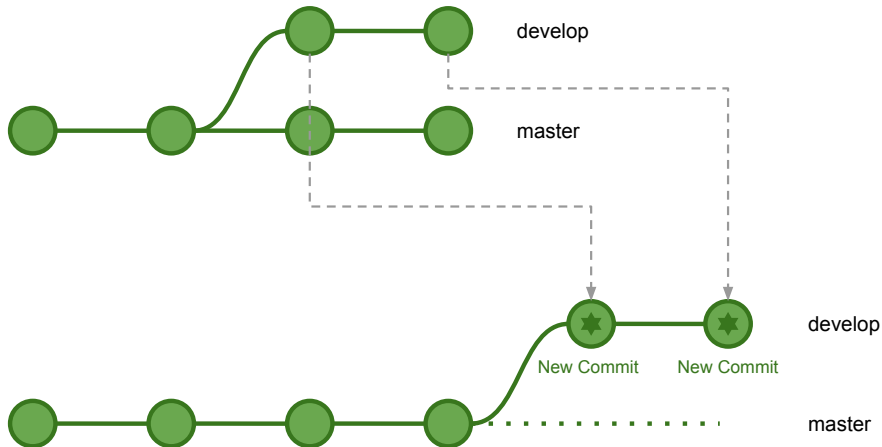


Figure: Branch Model for git **rebase**

Basic Usage

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

Basic Usage

History Management

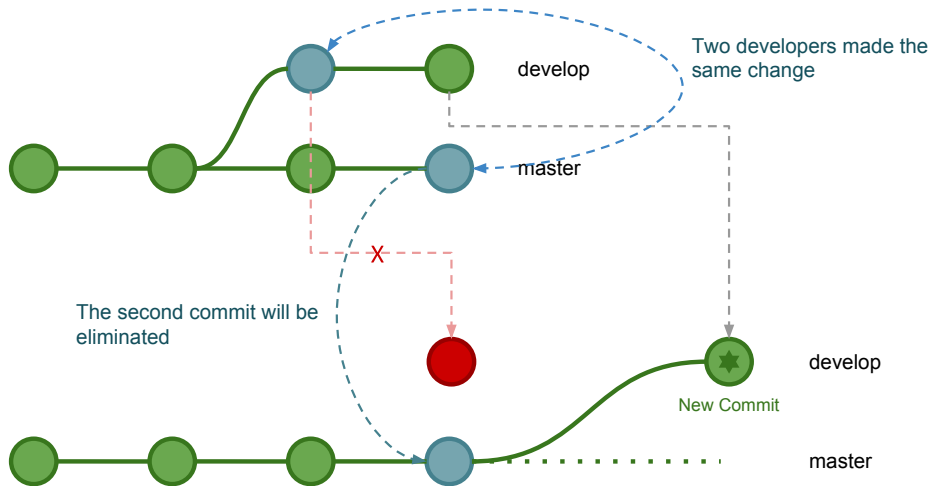


Figure: git **rebase** will eliminate empty commits

Basic Usage

History Management

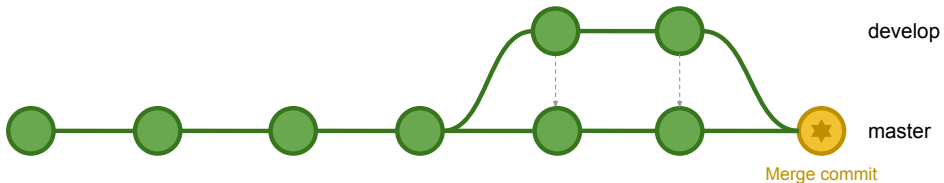
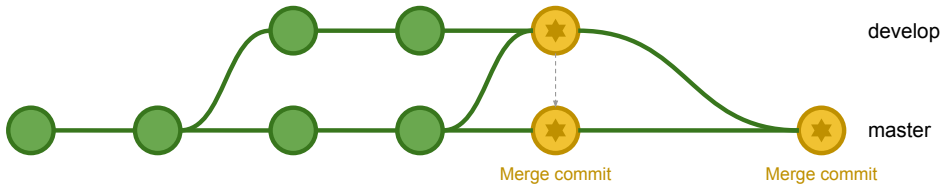


Figure: Branch flow: git merge vs. git rebase

Basic Usage

History Management

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

Basic Usage

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
```

Basic Usage

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

Basic Usage

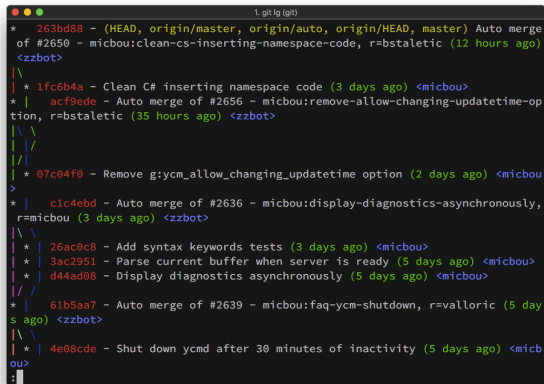
Other Toys

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

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

Basic Usage

Other Toys



```
1. git lg (git)
* 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:yem_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-yem-shutdown, r=valloric (5 day
s ago) <zzbot>
| \ \
| * 4e08cde - Shut down ycmd after 30 minutes of inactivity (5 days ago) <micbou>
ou>
:.
```

Figure: Show graph with git log

Basic Usage

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
```

Basic Usage

Other Toys

Toys and resources with **git hooks**:

- ▶ lolcommits⁴: take a photo of yourself every time you make a commit (see 1.6)
- ▶ Continuous Delivery
 - ▶ Heroku⁵
 - ▶ Dokku⁶
 - ▶ Deis⁷

⁴<https://lolcommits.github.io/>

⁵<https://www.heroku.com/>

⁶<https://github.com/dokku/dokku>

⁷<https://github.com/deis/workflow>

Basic Usage

Other Toys

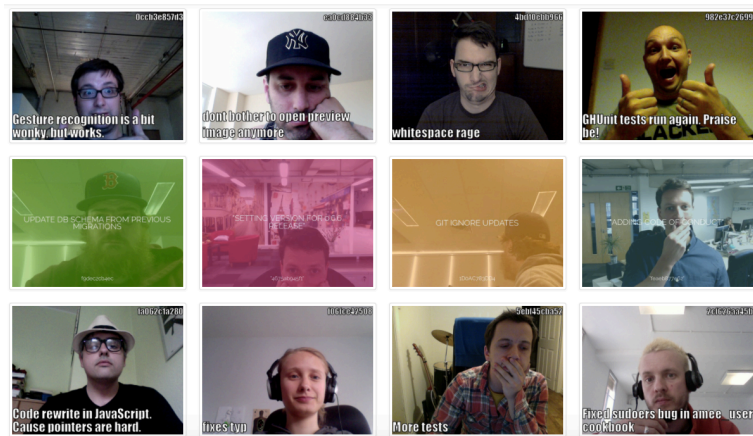


Figure: lolcommits

Basic Usage

Other Toys

What The Commit⁸!

⁸<http://whatthecommit.com/>

Inside Git

Git's Object Model

Question

How to implement **immutable stack** and **immutable tree**?

Inside Git

Git's Object Model

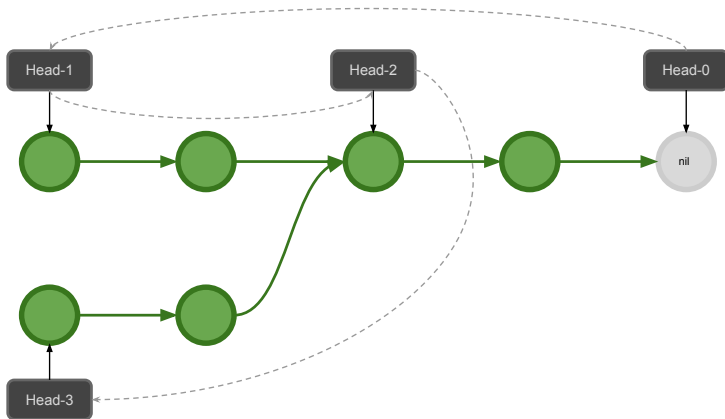


Figure: Immutable Stack

Inside Git

Git's Object Model

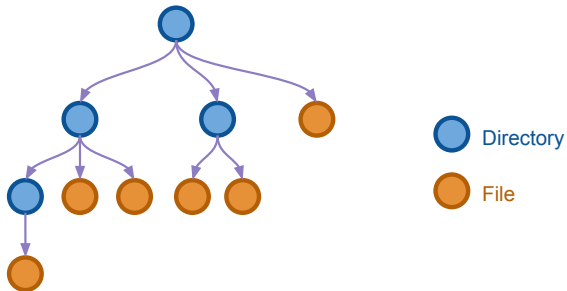


Figure: File Tree Model

Inside Git

Git's Object Model

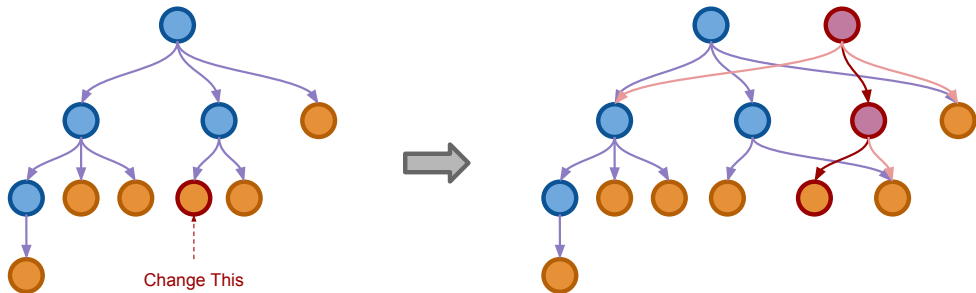


Figure: Immutable File Tree

Inside Git

Git's Object Model

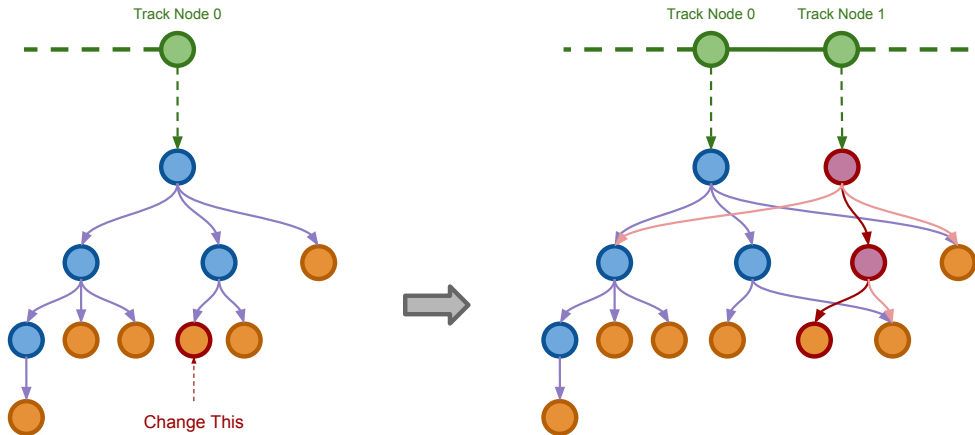


Figure: Git Object Tree with Commits

Inside Git

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

Inside Git

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
```

Inside Git

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

Inside Git

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:

```
#!/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])
```

Inside Git

Git's Internal Files

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

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


Inside Git

Git's Internal Files

Would like to try committing some files manually instead of using `git commit` directly?
Here are some commands might be useful⁹:

```
# 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
```

⁹Details at: <https://git-scm.com/book/en/v2/Git-Internals-Git-Objects>

Inside Git

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
```

Inside Git

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
```

Inside Git

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
```

Inside Git

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
```

Inside Git

Optimizations

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

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

Inside Git

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

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>.