git的那些事儿

- xiaorui.cc

- github.com/rfyiamcool

agenda

- git的由来
- git实现原理
- git使用规范
- git日常技巧

git的诞生

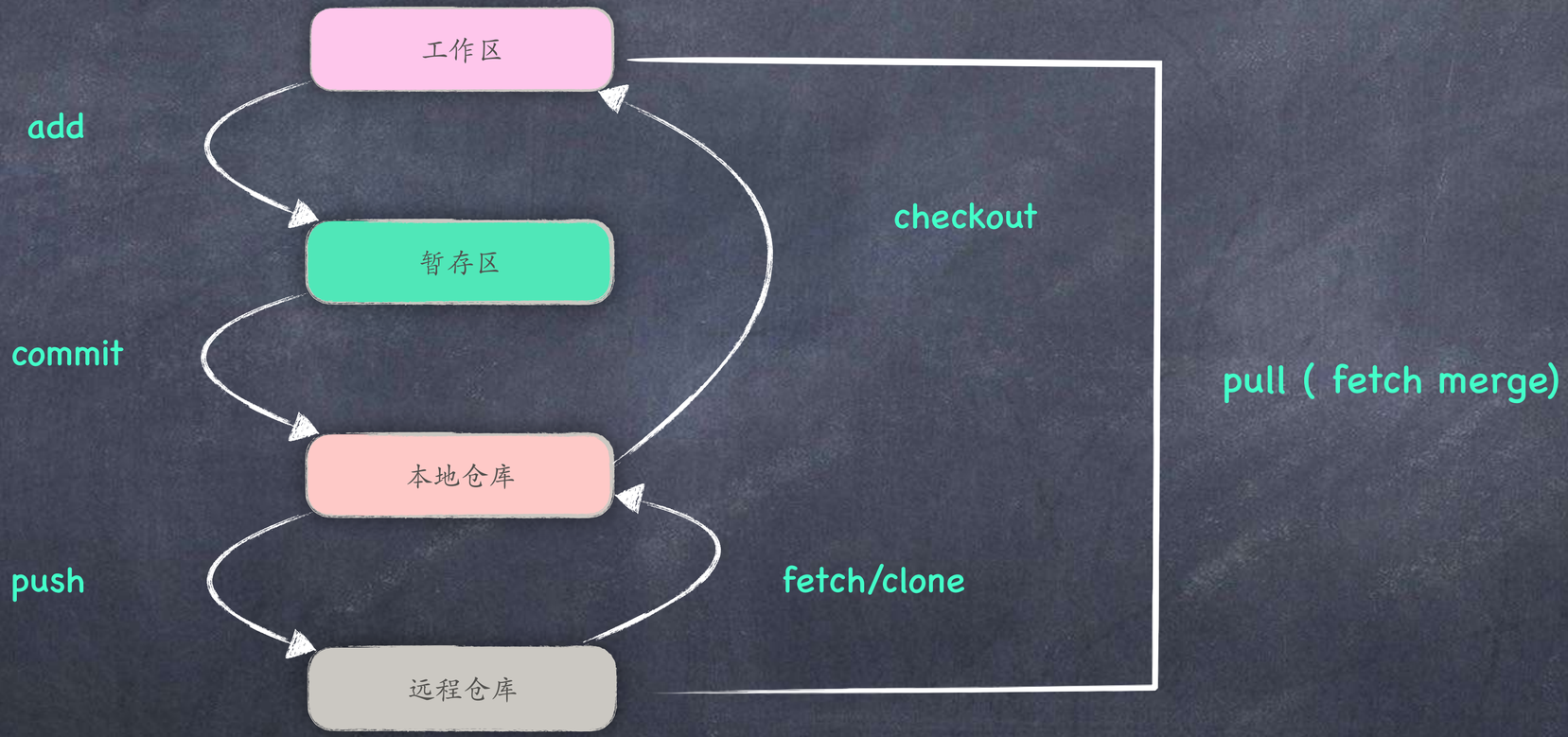
喷 nvidia 的视频地址

- 1991年
  - linux 本人手动合并 linux 代码
- 2002年
  - 商业公司 bitkeeper 免费赞助给linux开源社区
  - linux team里有个叫andrew哥们把bitkeeper给破解了，厂商不乐意.
- 2005年
  - linus 花了两周的时间构建了git的主体代码
  - 一个月后 linux 代码由于 git 托管
- 2008年
  - github上线开始接客

git实现原理

git

工作区


工作区
add
暂存区
commit
本地仓库
push
远程仓库
checkout
pull (fetch merge)
fetch/clone

```
.git
├── COMMIT_EDITMSG
├── HEAD
├── ORIG_HEAD
├── config
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── fsmonitor-watchman.sample
...
├── index
├── info
│   ├── exclude
│   └── refs
├── logs
│   ├── HEAD
│   └── refs
│       ├── heads
│       │   ├── develop
│       │   ├── feature
│       │   │   ├── issue-123
│       │   │   └── issue-781
│       │   └── master
│       └── stash
├── objects
│   ├── 06
│   │   └── e567b11dfdafeaf7d3edcc89864149383aeab6
│   ├── 44
│   │   └── 8230942a0d2f86e2d622f2c11af124deb33591
│   ├── 4f
│   │   └── 709134b40055d9f1ca199af3db7ca56e343f4e
...
│   ├── info
│   │   ├── commit-graph
│   │   └── packs
│   └── pack
│       ├── pack-88e4e89cadde5a638dda4edafd85b8e10bdc5328.idx
│       └── pack-88e4e89cadde5a638dda4edafd85b8e10bdc5328.pack
├── packed-refs
└── refs
    ├── heads
    │   ├── develop
    │   └── feature
    │       ├── issue-123
    │       └── issue-781
    ├── stash
    └── tags
```

目录结构

- commit_editmsg 最近提交描述
- HEAD 当前的分支指针
- index 暂存区的目录树，记录文件的相关信息
- FETCH_HEAD 最近的fetch信息
- objects 对象存储
  - object对应的SHA-1值的前 2 位为目录名，后 38 位为文件名
- object/pack 对象的压缩文件
- refs 分支、tag、stash对应的object信息
- logs 用于reflog做灾难恢复

object sha1

```
$ cat hello.txt
hello git


$ echo "hello git" | shasum
d6a96ae3b442218a91512b9e1c57b9578b487a0b  -

$ git add hello.txt


$ git commit -m "add hello.txt"
[master 2204d2e] add hello.txt
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt


$ echo "blob 10\0hello git"|sha1sum
8d0e41234f24b6da002d962a26c2495ea16a425f  -


$ git cat-file -t 8d0e41234f24b6da002d962a26c2495ea16a425f
blob

$ git cat-file -p 8d0e41234f24b6da002d962a26c2495ea16a425f
hello git


$ ll .git/objects/8d/0e41234f24b6da002d962a26c2495ea16a425f
-r--r--r--  1 ruifengyun  staff  26  4 21 08:27 .git/objects/8d/0e41234f24b6da002d962a26c2495ea16a425f
```

google sha1碰撞问题

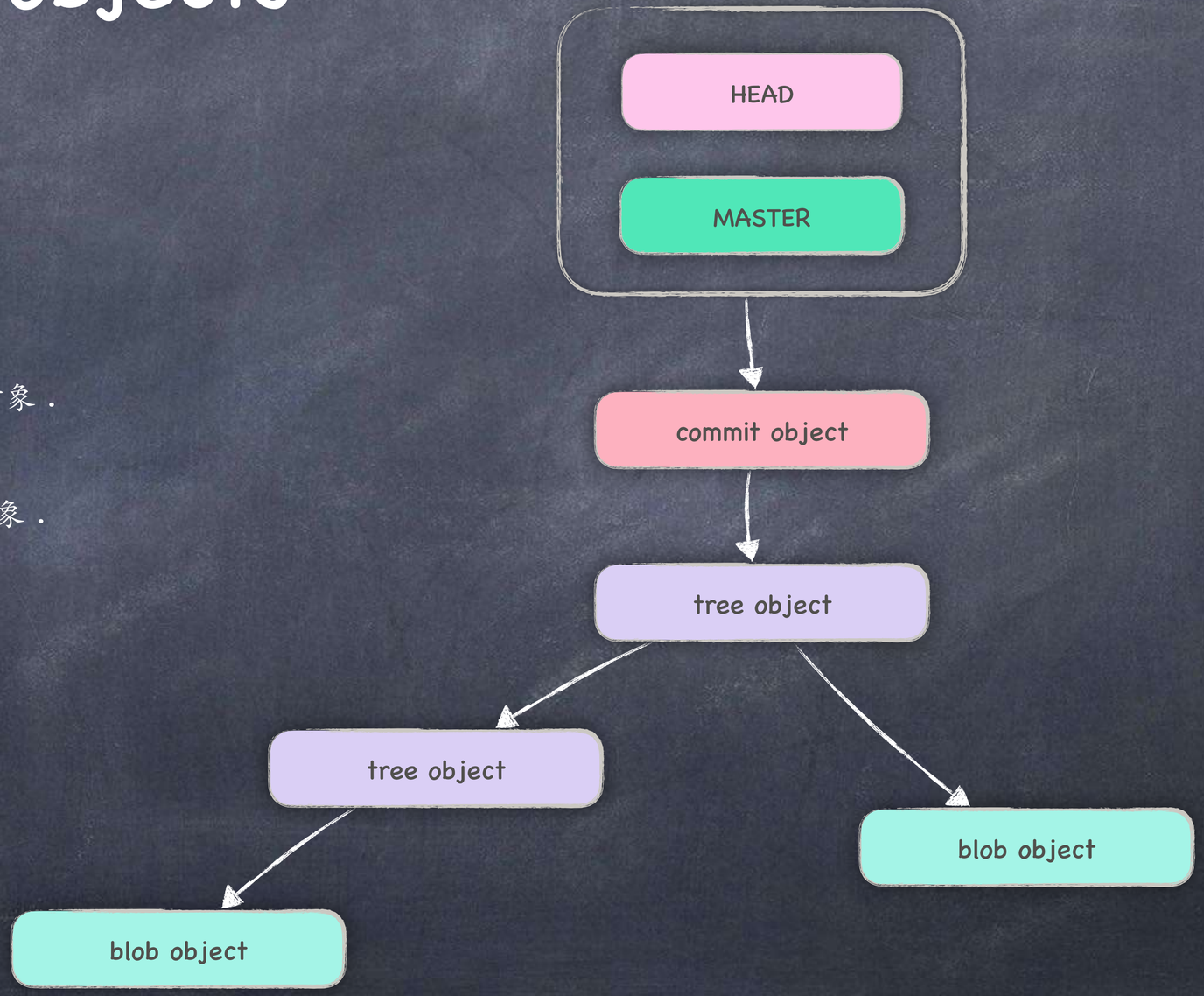
git同时对header和context进行hash

不同文件基本不会碰撞冲突

相同文件则忽略

objects

- object type
  - commit类型，显示提交信息及关联tree object对象 .
  - tree类型，关联其他子tree object对象及blob对象 .
  - blob类型，对应一个文件，也就是文件内容 .



一次提交过程

```
$ git init

$ echo xiaorui.cc > blog

$ git add .

$ git commit -m "add blog"

$ tree .git/objects
.git/objects
├── 6e
│   └── e4171157fb586fdacdddb7f3342cb65e9fd78d
├── a3
│   └── bd372af3ceb74dbbaa61cbbd9fe73569831489
├── cf
│   └── 5ff0c53144b0a5549ad523f369a99240050e12
├── info
└── ...

$ git cat-file -t 6ee417
commit

$ git cat-file -t cf5ff
tree

$ git cat-file -t a3bd372
blob
```

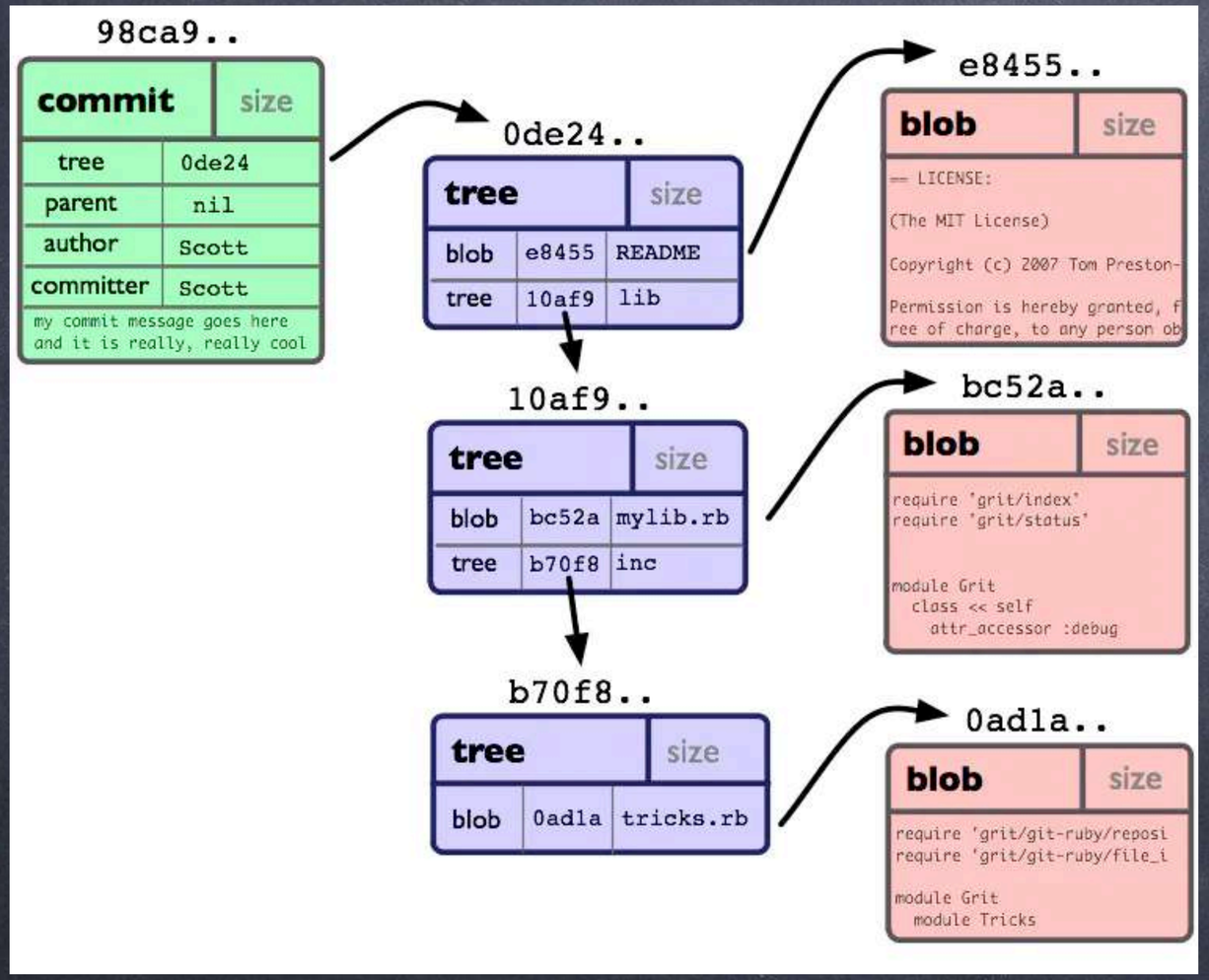
commit

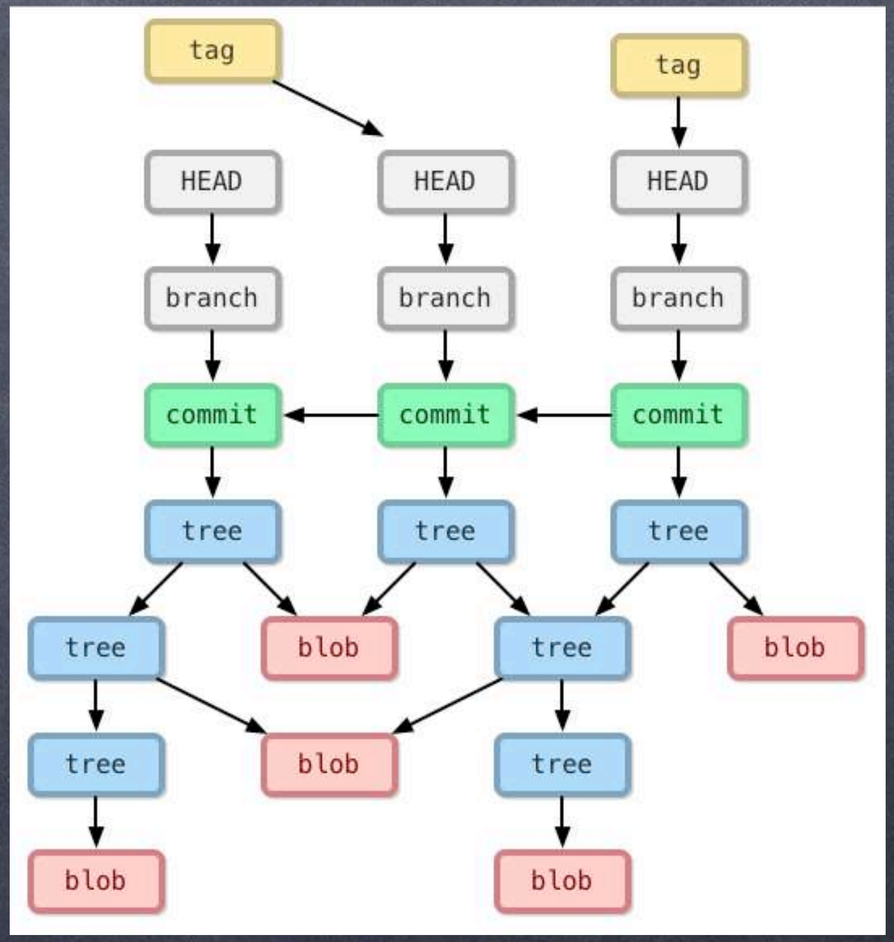
blob

tree

```
$ cat .git/refs/heads/master
6ee4171157fb586fdacdddb7f3342cb65e9fd78d

$ git cat-file -p 6ee417
tree cf5ff0c53144b0a5549ad523f369a99240050e12
author rfyiamcool <rfyiamcool@163.com> 1619225889 +0800
committer rfyiamcool <rfyiamcool@163.com> 1619225889 +0800

add blog

$ git cat-file -p cf5ff
100644 blob a3bd372af3ceb74dbbaa61cbbd9fe73569831489    blog

$ git cat-file -p a3bd372
xiaorui.c
```

git objects


98ca9..
commit
size
tree 0de24
parent nil
author Scott
committer Scott
my commit message goes here
and it is really, really cool
0de24..
tree
size
blob e8455 README
tree 10af9 lib
e8455..
blob
size
-- LICENSE:
(The MIT License)
Copyright (c) 2007 Tom Preston-
Permission is hereby granted, f
ree of charge, to any person ob
10af9..
tree
size
blob bc52a mylib.rb
tree b70f8 inc
bc52a..
blob
size
require 'grit/index'
require 'grit/status'
module Grit
class << self
attr_accessor :debug
b70f8..
tree
size
blob 0ad1a tricks.rb
0ad1a..
blob
size
require 'grit/git-ruby/reposi
require 'grit/git-ruby/file_i
module Grit
module Tricks


tag
tag
HEAD
HEAD
HEAD
branch
branch
branch
commit
commit
commit
tree
tree
tree
tree
blob
tree
blob
tree
blob
tree
blob
blob

gitlab repo

git server 是如何存储 git 项目的？

git server 是如何存储 git 项目的？

git server 是如何存储 git 项目的？

Gitlab服务端使用 git 目录存储 !!!

compress

```
$ ll -h

total 2488
drwxr-xr-x  12 ruifengyun  staff   384B  4 21 08:04 .git
-rw-r--r--   1 ruifengyun  staff   1.2M  4 21 08:04 big

$ du -sh .git/objects
0B  .git/objects

$ git add .

$ git commit -m "big"

$ du -sh .git/objects
204K    .git/objects
```

- 使用zlib进行数据压缩.
- 对commit、tree、blob都有压缩.
- 日常的代码文件压缩率在5倍.

文件的大小

```
$ git init
$ curl -s xiaorui.cc > big
$ du -sh big
 76K    big

# 第一次提交
$ git add .
$ git commit -m "add big"
$ du -sh .git/objects
 24K    .git/objects

# 第二次提交
$ echo "append xiaorui.cc" >> big
$ git add .
$ git commit -m "append big"
$ du -sh .git/objects
 48K    .git/objects

# size对比
$ git cat-file -s 57a146d2980841b750ec99703ee0711004d60313
76400

$ git cat-file -s 135454f7a8b19aabfa813b9271c4fd20dbed3609
76418
```

每次变更的提交都是一个全量blob对象文件，

而非文件的增量变更 !!!

git pack

- why
  - git的检索效率高
  - 利于网络传输，便于备份
  - 增量存储优化磁盘空间
  - 无需每个对象占用一个 inode
- git gc = git repack + git prune-packed

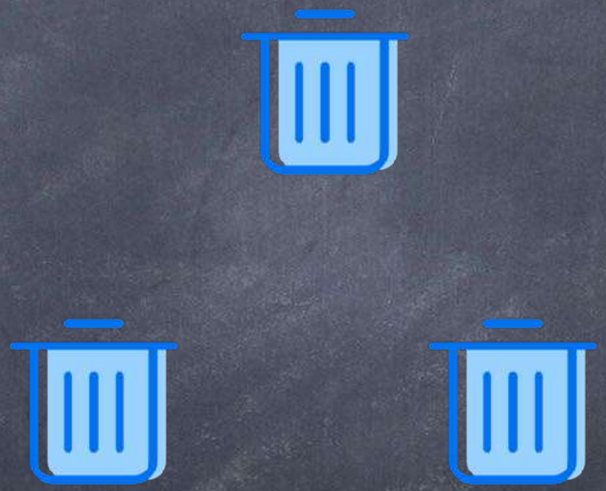
由于 blob 文件太多，导致 git 效率下降

```
$ tree .git/objects/ | wc -l
311

$ git gc
Enumerating objects: 288, done.
Counting objects: 100% (288/288), done.
Delta compression using up to 4 threads
Compressing objects: 100% (287/287), done.
Writing objects: 100% (288/288), done.
Total 288 (delta 131), reused 90 (delta 0)

$ tree .git/objects/ | wc -l
12
```

```
$ git verify-pack -v .git/objects/pack/pack-5963b552193021791c1a0ab9136c272f07124c98.pack

5978c2c79cd3e4711fb8edd3166c9f9f5c8c97f5 commit 245 153 12
2305588a632214f28646226042 8c4395f936b5b0 commit 252 156 165
1fa9735670eb952b5468d17b418525717c8e3527 commit 248 156 321
3ffb7fb9830e232689c95b3b65f0f8f3fc7a6027 commit 248 155 477
86a5912f97d7dd8f90a28cab6bffc8ee78997e2c commit 244 151 632
94e1a0d952f577fe1348d828d14550d3709e11e commit 249 156 783
86903f8f5024485afa8480020a04cc00f228d23c commit 243 150 939
6efdffad4fb725aa8d0f4d7d29feb5aee7ea5dff commit 242 151 1089
04c87c65f142f33945f2f5951cf7801a32dfa240 commit 73 85 1240 1 6efdffad4fb725aa8d0f4d7d29feb5aee7ea5dff
2a810017bfc85d7db2627f4aabdaa1583212bda3 blob   19 27 1325
e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 blob   0 9 1352
b5e810691433cf8a2960c27c1b33546fa96e2bef blob   16 26 1361
2f36e957afc2b3bcda988cb29a86e3a1490e8cc2 tree   153 106 1387
2ed6130bd33afa26817418308e29c4081ea056ec tree   5 15 1493 1 2f36e957afc2b3bcda988cb29a86e3a1490e8cc2
9df381ad27294a62ba1ae65abed489072d778c79 tree   123 103 1508
7d48a14b9ca1dca2f6a593eef19633ce45f81bee blob   12 21 1611
a448b4d6450de854dcc6fe658bdb72e22c726cbb tree   123 102 1632
9e56fd51f52d8b9d242c50c24a4cae586d76ec7e blob   7 16 1734
bde15b851f135327ada02c9deac0fb1ee01cf343 tree   123 102 1750
58c9bdf9d017fcd178dc8c073cbfcbb7ff240d6c blob   4 13 1852
3920a07c1d5694df5b8658592b0939241d70e9e5 tree   7 17 1865 1 bde15b851f135327ada02c9deac0fb1ee01cf343
16729e3b94f19bc95cb6f563f776bfb4694a6e5b tree   4 14 1882 2 3920a07c1d5694df6b8658592b0939241d70e9e5
b72c74792528892694c395b2c9a3d6af740f3fb2 tree   63 50 1896
098217953a6ca169bed33d2be8a07d584fcdaf30 tree   31 42 1946
non delta: 20 objects
chain length = 1: 3 objects
chain length = 2: 1 object
.git/objects/pack/pack-5963b552193021791c1a0ab9136c272f07124c98.pack: ok
```

git pack

- when
  - 对象太多时触发gc
    - 约超过约 7k 个松散对象时
    - 超过 50 个包文件时
  - 手动执行 git gc 回收
  - 推送到 git server 时



lock

```
open(".git/index.lock", O_RDWR|O_CREAT|O_EXCL, 0666) = 3
rename(".git/index.lock", ".git/index")    = 0

open(".git/HEAD.lock", O_WRONLY|O_CREAT|O_EXCL, 0666) = 3
rename(".git/HEAD.lock", ".git/HEAD")    = 0

open(".git/refs/heads/develop.lock", O_RDWR|O_CREAT|O_EXCL, 0666) = 3
rename(".git/refs/heads/develop.lock", ".git/refs/heads/develop") = 0
close(3)

open(".git/refs/heads/master.lock", O_RDWR|O_CREAT|O_EXCL, 0666) = 3
rename(".git/refs/heads/master.lock", ".git/refs/heads/master") = 0
close(3)
...
```

单单就本地操作 GIT 有毛的并发安全 ？

- 使用 O_CREAT | O_EXCL 实现 git 操作锁
- 该组合可实现 if exists else create 的原子性
- 解决可能并发操作引起的混乱问题

git checkout文件变更？

```
$ ls
aaa  bbb  ccc  ddd  eee  fff  log  uuu  www

$ cat www
this is www

$ strace -f git checkout feature

lstat("www", 0x7ffd14dbb100)            = -1 ENOENT (没有那个文件或目录)
open("www", O_WRONLY|O_CREAT|O_EXCL, 0666) = 4
open(".git/objects/cc/9ff6eb0107eae401ff27ed7abf9ecf9c5659a8", O_RDONLY|O_NOATIME) = 5
fstat(5, {st_mode=S_IFREG|0444, st_size=26, ...}) = 0
mmap(NULL, 26, PROT_READ, MAP_PRIVATE, 5, 0) = 0x7f173c079000
close(5)                                = 0
munmap(0x7f173c079000, 26)              = 0
open(".git/objects/cc/9ff6eb0107eae401ff27ed7abf9ecf9c5659a8", O_RDONLY|O_NOATIME) = 5
fstat(5, {st_mode=S_IFREG|0444, st_size=26, ...}) = 0
mmap(NULL, 26, PROT_READ, MAP_PRIVATE, 5, 0) = 0x7f173c079000
close(5)                                = 0
write(4, "this is www\n", 12)           = 12
munmap(0x7f173c079000, 26)              = 0
```

切换到其他分支，通过tree object检索

到本地无文件或者hash sha不匹配 !!!

则进行读取blob object并写入操作.

git checkout 文件变更 ?

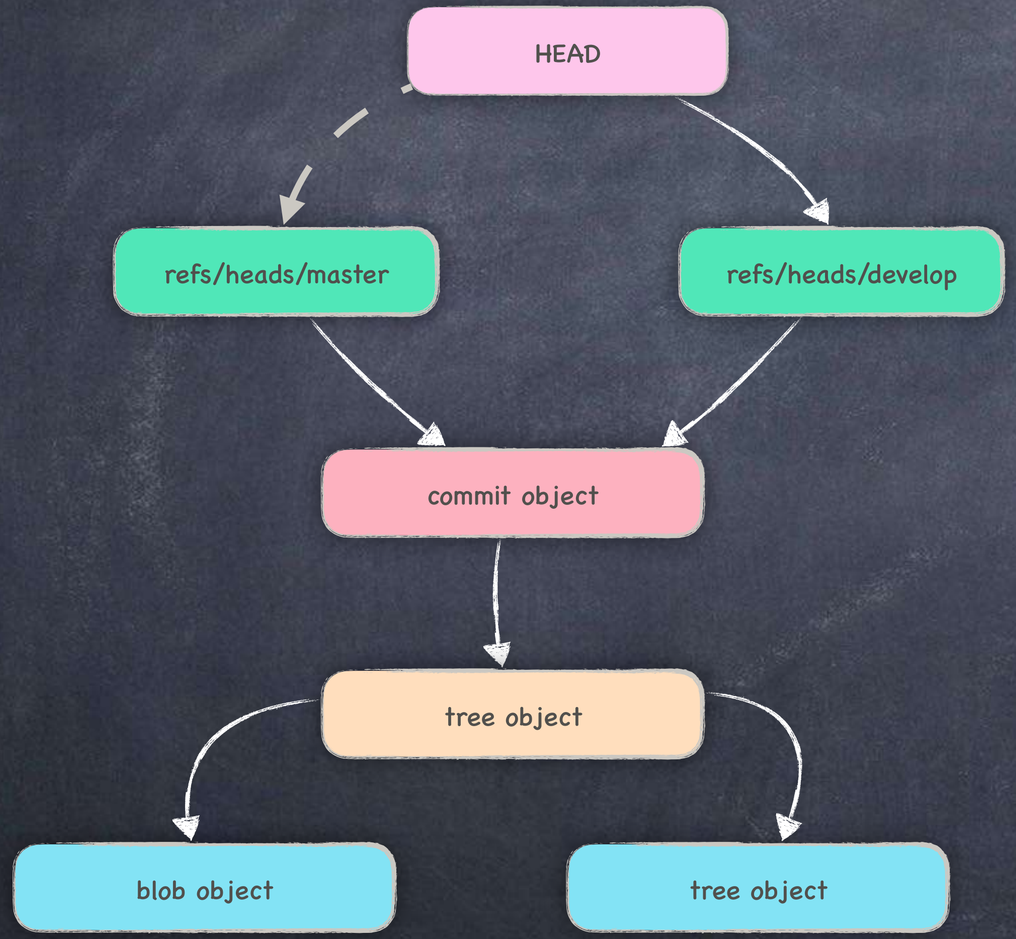
```
$ strace -f git checkout develop

lstat("aaa", {st_mode=S_IFREG|0644, st_size=4, ...}) = 0
lstat("bbb", {st_mode=S_IFREG|0644, st_size=4, ...}) = 0
lstat("ccc", {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
lstat("ddd", {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
lstat("eee", {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
lstat("fff", {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
lstat("log", {st_mode=S_IFREG|0644, st_size=6, ...}) = 0
lstat("uuu", {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
lstat("www", {st_mode=S_IFREG|0644, st_size=12, ...}) = 0
...
unlink("aaa")                           = 0
unlink("bbb")                           = 0
unlink("ccc")                           = 0
unlink("ddd")                           = 0
unlink("eee")                           = 0
unlink("fff")                           = 0
unlink("uuu")                           = 0
unlink("www")                           = 0
```

切换到其他分支, 如该分支不包含本地文件或者hash sha1不匹配 !!!

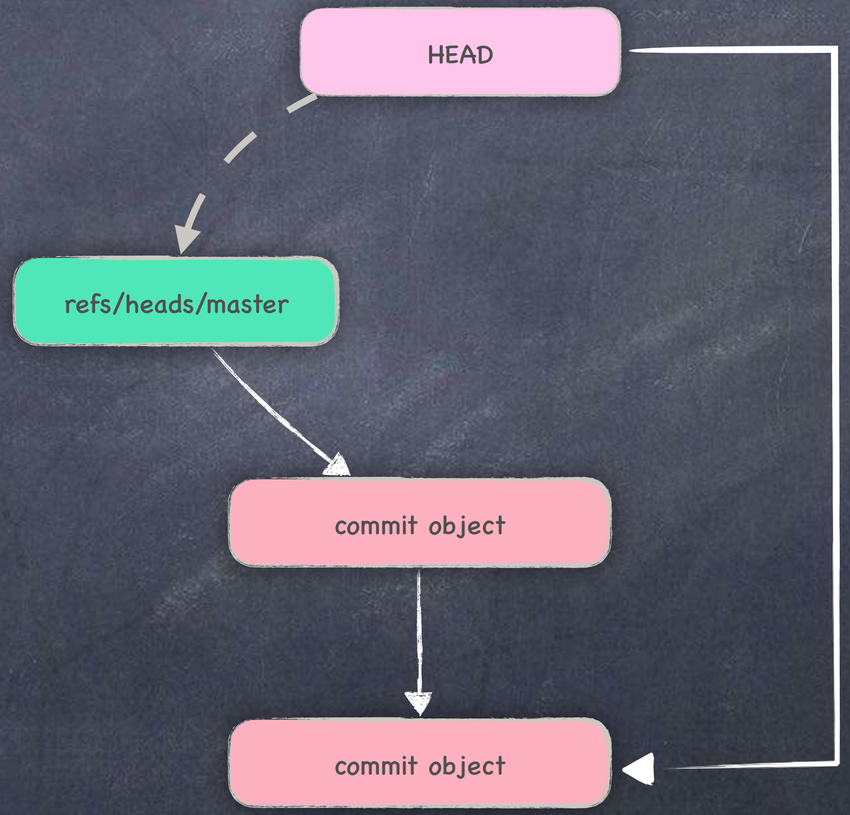
则 unlink 删除文件.

git checkout ?

git checkout <branch name>



git checkout <commit id>



git stash ?

```
[pid 3033564] lstat(".git/refs/stash", {st_mode=S_IFREG|0644, st_size=41, ...}) = 0
[pid 3033564] open(".git/refs/stash", O_RDONLY) = 3
[pid 3033564] read(3, "acc164560230a0b6a6fdaede3e397f53cdcd4e50\n", 255) = 41
[pid 3033564] read(3, "", 214)          = 0
[pid 3033564] close(3)
...
[pid 3033564] open(".git/logs/refs/stash", O_RDONLY) = 3
[pid 3033564] fstat(3, {st_mode=S_IFREG|0644, st_size=461, ...}) = 0
[pid 3033564] mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f75af62c000
[pid 3033564] read(3, "0000000000000000000000000000000000000000 fa382636ec18bc888e1c411105017db359d9bf2e Your Name <you@example.com> 1619059129 -0800\tWIP on master: a3f622b log\nfa382636ec18bc888e1c411105017db359d9bf2e 6c03e7957862a117f0ddc8020ea13ecb39ab3b4f Your Name <you@example.com> 1619059529 -0800\tWIP on master: a3f622b log\n6c03e7957862a117f0ddc8020ea13ecb39ab3b4f acc164560230a0b6a6fdaede3e397f53cdcd4e50 Your Name <you@example.com> 1619061580 +0800\tOn master: add stash3.txt\n", 4096) = 461
[pid 3033564] read(3, "", 4096)          = 0
[pid 3033564] close(3)                  = 0
[pid 3033564] munmap(0x7f75af62c000, 4096) = 0
[pid 3033564] ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
```

```
$ echo "xiaorui.cc" > stash1.txt

$ git add stash1.txt

$ git stash
Saved working directory and index state WIP on master: a3f622b log
HEAD 现在位于 a3f622b log

$ cat .git/refs/stash
fa382636ec18bc888e1c411105017db359d9bf2e

$ git cat-file -t fa382636ec18bc888e1c411105017db359d9bf2e
commit

$ git cat-file -p fa382636ec18bc888e1c411105017db359d9bf2e
tree 15f9d139f34d4ae0af745da3d3094d3aa9ec61e7
...

$ git cat-file -p 15f9d139f34d4ae0af745da3d3094d3aa9ec61e7
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    log
100644 blob a3bd372af3ceb74dbbaa61cbbd9fe73569831489    stash1.txt

$ git cat-file -p a3bd372af3ceb74dbbaa61cbbd9fe73569831489
xiaorui.cc
```

- .git/refs/stash
  - 最近的stash记录
  - commit -> tree -> blob
- .git/lgos/refs/stash
  - 所有的stash记录

git reset ?

```
$ strace -f git reset --hard a3f622ba80cc7cda2d1e2f59d09c5a7175d707dd

# 删除回滚提交记录不存在的文件
unlink("stash1.txt")                    = 0
unlink("stash2.txt")                    = 0
unlink("stash3.txt")
...

# 修改当前master headr指针
open(".git/refs/heads/master", O_RDONLY) = 4
write(3, "a3f622ba80cc7cda2d1e2f59d09c5a71"..., 40) = 40
write(3, "\n", 1)                       = 1
close(3)
...

# 在git reflog日志中追加操作记录
open(".git/logs/refs/heads/master", O_WRONLY|O_APPEND) = 3
write(3, "f7f900b2545b976d37c08616a6526507"..., 185) = 185
close(3)
...
```

回滚到一个不存在stash.txt文件的提交

```
$ strace -f git reset --hard f7f900b

# 当前目录不存在目标提交记录中的文件，需要从objet读取出来并复制一份.
open("stash1.txt", O_WRONLY|O_CREAT|O_EXCL, 0666) = 4
open(".git/objects/a3/bd372af3ceb74dbbaa61cbbd9fe73569831489", O_RDONLY|O_NOATIME) = 5
mmap(NULL, 27, PROT_READ, MAP_PRIVATE, 5, 0) = 0x7f0d9544e000
write(4, "xiaorui.cc\n", 11)            = 11

open("stash2.txt", O_WRONLY|O_CREAT|O_EXCL, 0666) = 4
open(".git/objects/5a/d28e22767f979da2c198dc6c1003b25964e3da", O_RDONLY|O_NOATIME) = 5
mmap(NULL, 20, PROT_READ, MAP_PRIVATE, 5, 0) = 0x7f0d9544e000
write(4, "haha\n", 5)                   = 5

lstat("stash3.txt", 0x7ffe2941c700)     = -1 ENOENT (没有那个文件或目录)
open("stash3.txt", O_WRONLY|O_CREAT|O_EXCL, 0666) = 4
open(".git/objects/f6/edd6e7a290f009aa685d3acd3153b495a69ea8", O_RDONLY|O_NOATIME) = 5
mmap(NULL, 27, PROT_READ, MAP_PRIVATE, 5, 0) = 0x7f0d9544e000
write(4, "git test\n", 9)               = 9

# 更改head master指针
open(".git/refs/heads/master", O_RDONLY) = 4
read(4, "a3f622ba80cc7cda2d1e2f59d09c5a71"..., 255) = 41
write(3, "f7f900b2545b976d37c08616a6526507"..., 40) = 40

# 把操作记录写到reflog里
open(".git/logs/refs/heads/master", O_WRONLY|O_APPEND) = 3
write(3, "a3f622ba80cc7cda2d1e2f59d09c5a71"..., 152) = 152
close(3)
```

回滚到一个存在stash.txt文件的提交

git filter-branch ?

```
# 重写提交记录，并删除匹配的文件
$ git filter-branch --force --index-filter \
  'git rm -rf -r --cached --ignore-unmatch {filename}' \
  --prune-empty --tag-name-filter cat -- --all


# 覆盖远端
$ git push origin master --force


# 本地刷新
$ rm -rf .git/refs/original/
$ git reflog expire --expire=now --all
$ git gc --prune=now
```

真正的删除文件 !!!

- 清理文件，回收空间
  - 为什么 git rm 不行 ?
  - 为什么 git reset —hard 无效 ?
  - 为什么还需要本地缓存删除 ?

git使用规范

git

git硬规范

- 分支管理
  - 代码提交在应该提交的分支上
  - 随时可以切换到线上稳定版本代码
  - 多个版本的开发工作同时进行
- 记录的可读性
  - commit内容按照格式执行.
  - 正确设置user.name和user.email信息.
- 版本号(tag)
  - 版本号(tag)命名规则
  - 主版本号.次版本号.修订号，如2.1.13
  - 对master标记tag意味着该tag能发布到生产环境
  - 版本号仅标记于master分支，用于标识某个可发布/回滚的版本代码
  - 仅项目管理员有权限对master进行合并和标记版本号

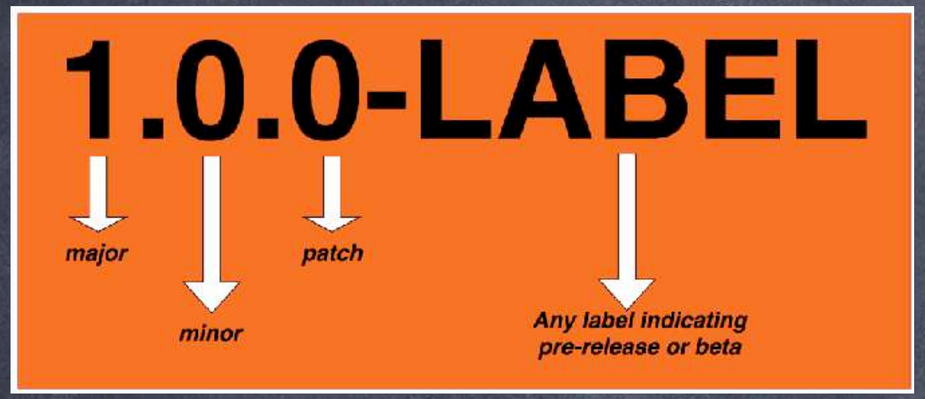
git commit规范

- type
  - feat: 新功能
  - fix: 修复 bug
  - docs: 文档变动
  - style: 单纯的格式调整
  - refactor: bug 修复和添加新功能之外的代码改动
  - perf: 提升性能的改动
  - test: 添加或修正测试代码
  - chore: 构建过程或辅助工具和库的更改

- <type>(<scope>): <subject>
  - feat(detail): 详情页修改样式
  - fix(login): 登录页面错误处理
  - test(list): 列表页添加测试代码

- scope 修改范围
  - 这次修改涉及到的部分简单概括
  - login、train-order
- subject 修改的描述
  - 具体的修改描述信息
  - provide issue id
- <type>(<scope>): <subject>
  - feat(detail): 详情页修改样式
  - fix(login): 登录页面错误处理
  - test(list): 列表页添加测试代码

语义化版本



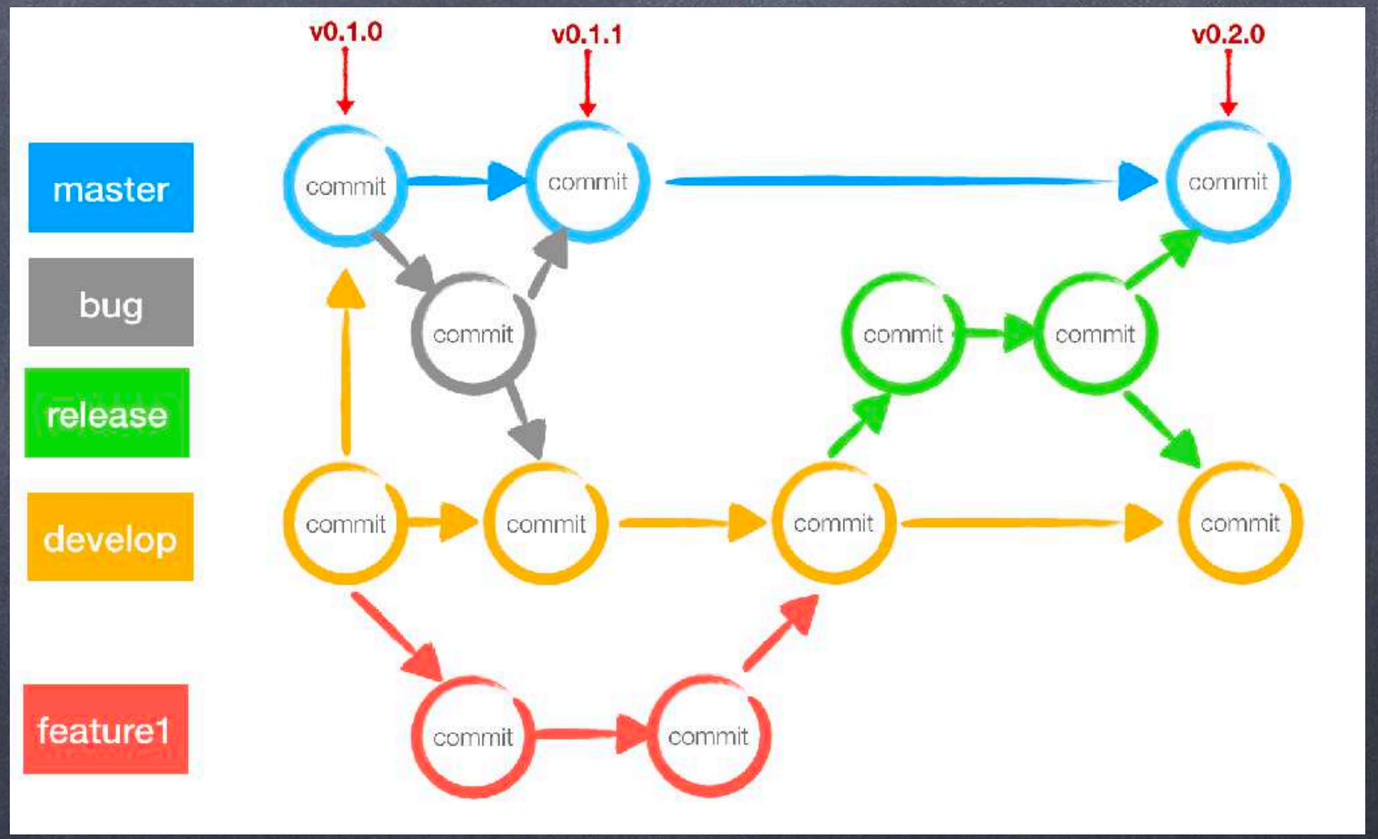
- 主版本号 (major)
  - 当有重大升级改动.
  - 当做了不兼容的 API 修改.
- 次版本号 (minor)
  - 当做了向下兼容的功能性新增.
- 修订号 (patch)
  - 当做了向下兼容的问题修正.

```
v1.0.0-alpha.0
v1.0.0-alpha.1
v1.0.0-beta.0
v1.0.0-rc.0
v1.0.0
```

- label
  - alpha：内部测试版
  - beta：公开测试版
  - rc: 候选版本，不再增加新功能.

git-flow

* master
  * git merge hotfix
  * git merge release/v1.0.0-rc.0
* hotfix
  * git checkout -b hotfix/fix-timeout master
* release
  * git checkout -b release/v1.0.0-rc.0 develop
* develop
  * git checkout -b develop master
* feature
  * git checkout -b feature/add-timeout develop
  * git checkout -b feature/issue-6379 develop



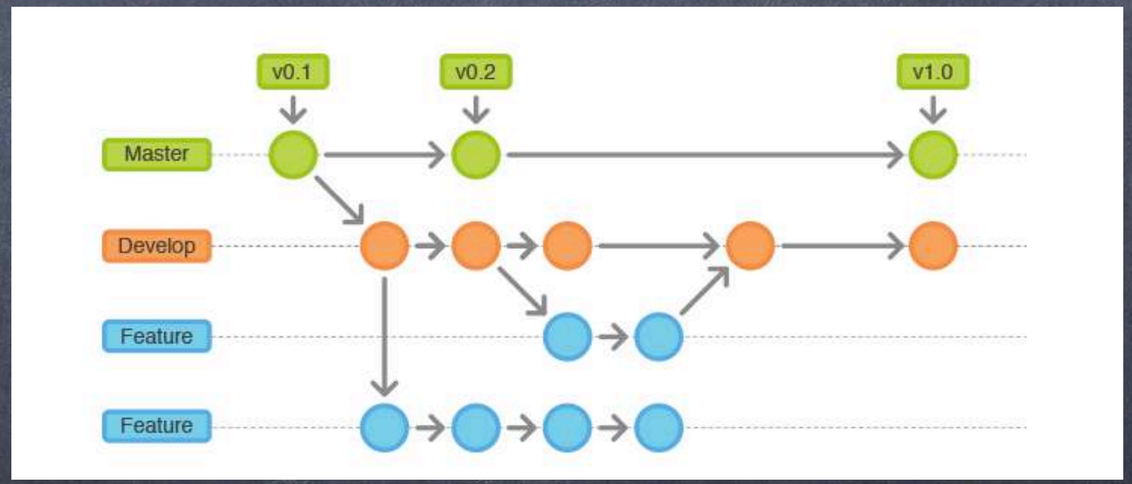
master

- 使用规范:
  - master分支存放的是随时可供在生产环境中部署的稳定版本代码
  - 使用 tag 标记一个版本用于发布或回滚
  - master分支是保护分支，不可直接push到远程仓master分支
  - 如当前版本出现异常，可先使用 tag 配合 cicd 进行回滚.

不能直接push，合并走 merge request 流程

develop 分支

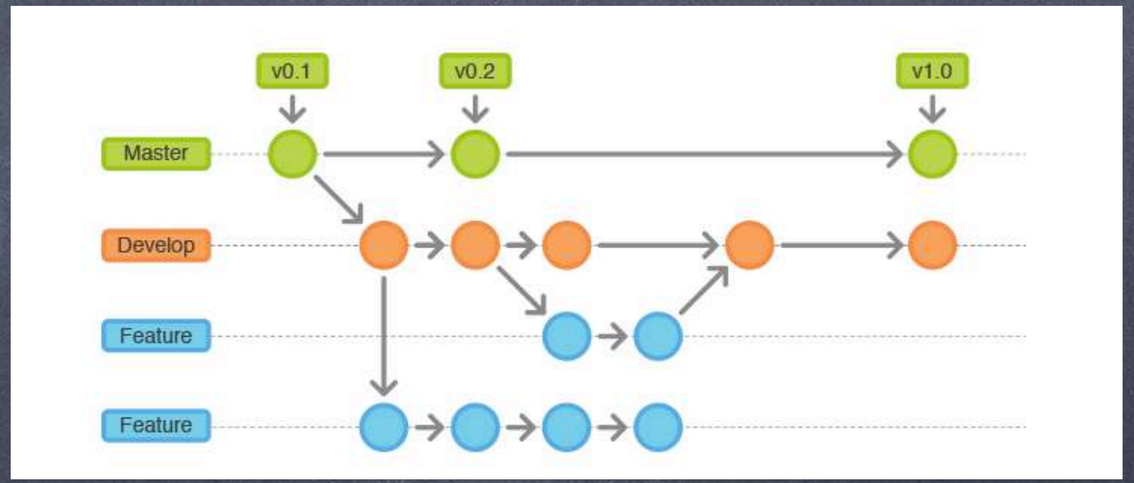
- 使用规范:
  - develop分支作为开发的主分支，保存当前最新开发成果的分支
  - 由 develop 分支衍生出各个 feature 分支
  - 建议新功能和修复开发都在 feature 分支进行



建议锁定 develop 分支，禁止push提交代码，代码合并走 merge request

feature 分支

- 使用规范:
  - 分支的命名格式建议 feature/login or feature/issue-3677
  - 以功能为单位从 develop 拉一个 feature 分支
  - 每个 feature 分支颗粒要尽量小，以利于快速迭代和避免冲突
  - 当其中一个feature分支完成后，需合并回 develop 分支，另需删除 feauter/xxx 分支
  - feature分支只与 develop 分支交互，不能与 master 分支直接交互



bugfix 分支

- 使用规范:
  - 分支的命名格式建议
    - bugfix/fix-login
    - bugfix/issue-1314
  - 由 develop 分支衍生用来修复问题用的分支
  - bugfix 分支跟 develop 分支合并，不能越界跟其他高分支交互

使用 feature 做解决问题的分支不是很合适？

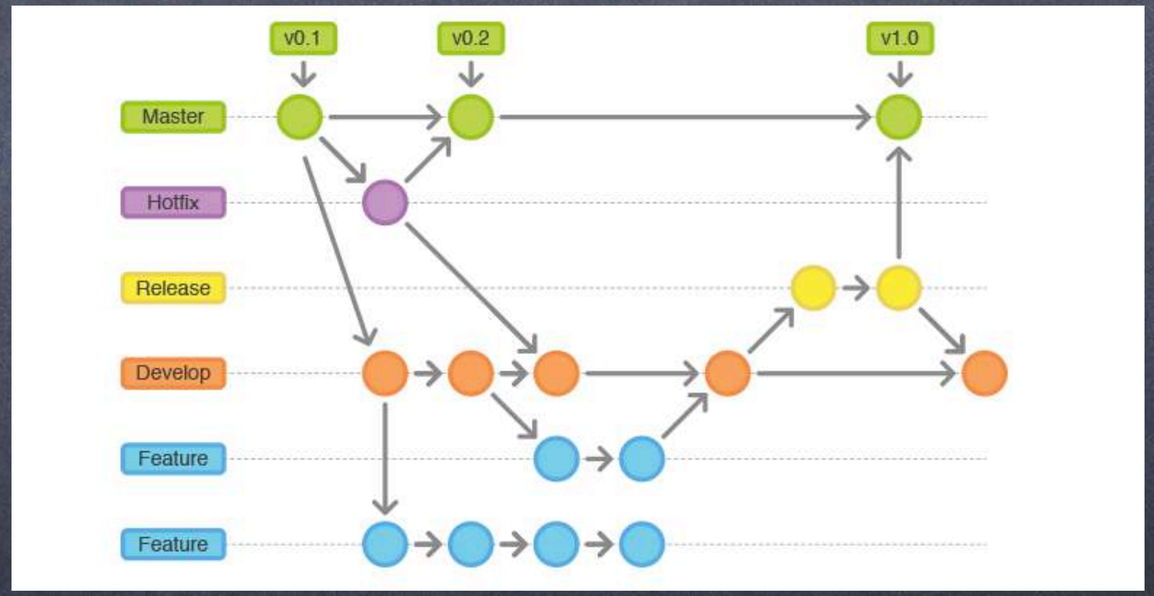
release 分支

- 使用规范:
  - 分支的命名格式建议 release/v1.0.0.rc.0 类似这种.
  - release主要用来为发布新版的测试、预发布、修复的分支.
  - release分支可以从develop分支上指定commit派生出.
  - release分支测试通过后，合并到master分支并且给master标记一个版本号.
  - 如在 release 分支发现 bug 可在 release 中修复，测试完成后合并到 develop 和 master 分支.
  - release分支一旦建立就将独立，不可再从其他分支pull代码.

如项目不大可省略该分支，避免繁琐的 gitflow 过程 !!!

hotfix 分支

- 使用规范:
  - 命名规则：hotfix/*，* 以本次发布的版本号为标识.
  - hotfix分支用来快速给已发布产品修复bug或微调功能.
  - 只能从master分支指定tag版本衍生出来.
  - 一旦完成修复bug，必须合并回master分支和develop分支.
  - master被合并后，应该被标记一个新的版本号.
  - hotfix分支一旦建立就将独立，不可再从其他分支pull代码.



git-flow

- 是否需要严格遵守 gitflow ?
- git-flow 真的就适合大家么 ?
- 大家经历的公司是否有严格要求 git-flow ?
- 为什么大厂和开源社区都有自己的 git-flow ?

建议用适合自己的 gitflow 来协助代码管理

git日常技巧

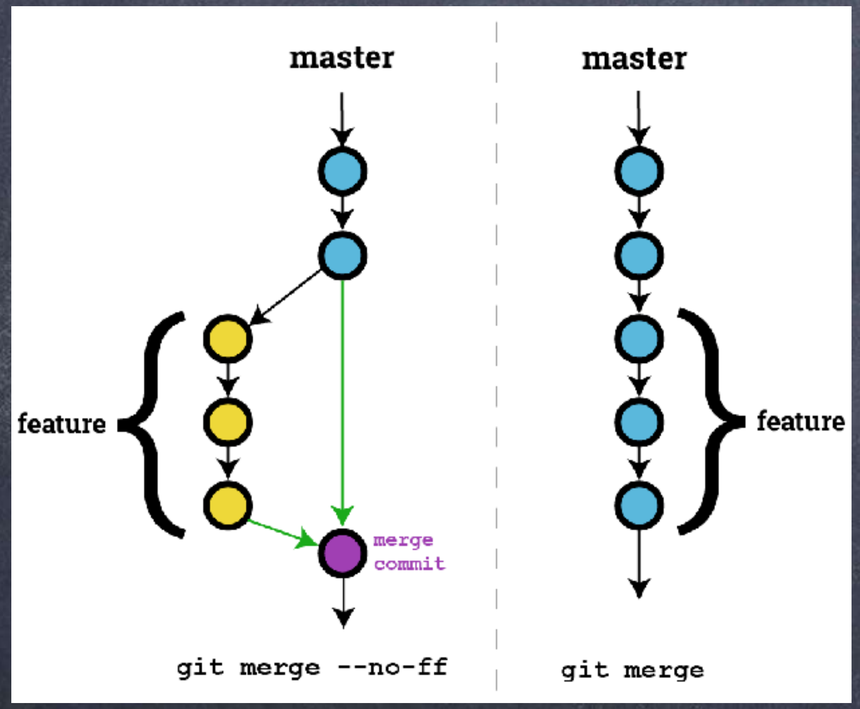
git

reset vs revert

- 强烈建议
  - 在公共分支做 revert .
    - revert 会删除某提交的变更，然后产生新的提交，并不会真正删除history.
  - 在自己分支做 reset .
    - soft: 回滚到某提交，但变更保留在工作区内，供开发者选择
    - hard: 彻彻底底的回滚到某提交，清空暂存区和工作区.

不要在公共分支做 reset 操作 !!!

fast-forward



从dev分支创建了feature分支，feature进行功能开发并提交后，在dev对feature分支进行合并，如feature提交期间dev无其他变更这时的合并就是 fast forward 模式.

```
git merge --no-ff -m "merge with no-ff" dev
```

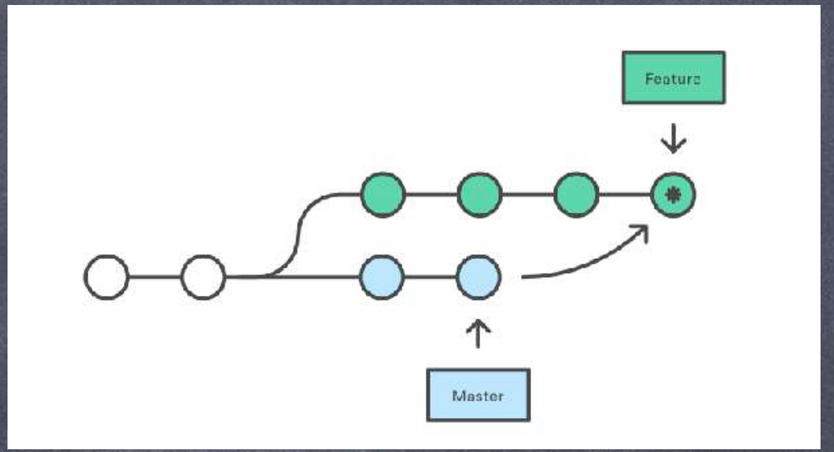
rebase vs merge

- 建议
  - 下游分支更新上游分支内容的时候使用 rebase
  - 上游分支合并下游分支内容的时候使用 merge
  - 更新当前分支的内容时一定要使用 --rebase 参数

不建议在公共分支上使用rebase

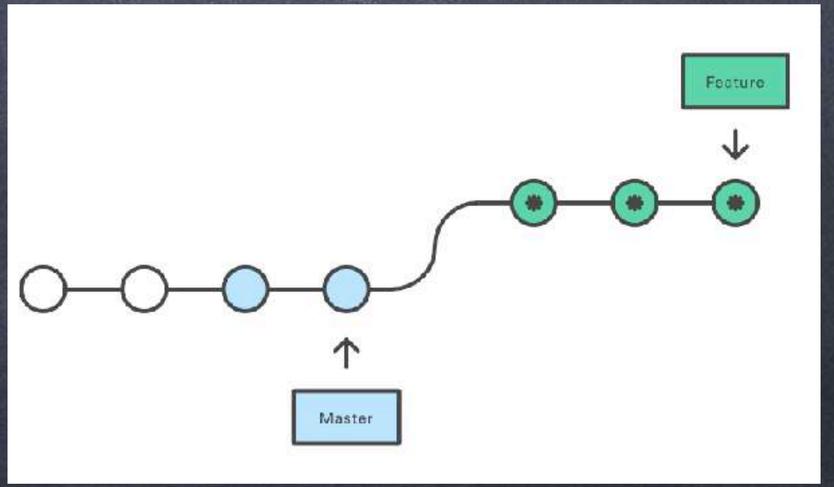
merge

注重历史记录



rebase

注重线性提交



合并多个commit记录

- merge squash
  - 在上游分支操作
  - git merg -squash <branch>
- rebase squash
  - 在下游分支操作
  - git rebase -i HEAD^4


```
s 9a54fd4 update1
s 0d4a808 下班提交
s 159db45 吃饭提交
pick 92f277a 改完了

# Rebase 326fc9f..0d4a808 onto d286baa
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

使代码的提交记录更加规整 !!!

git alias

三种添加方式，可根据自己情况添加

```
$ vim ~/.gitconfig
[alias]
  s = status
  d = diff
  co = checkout
  br = branch
  last = log -1 HEAD
  cane = commit --amend --no-edit
  lo = log --oneline -n 10
  pr = pull --rebase

$ vim ~/.zshrc
alias gst="git status"
alias gac="git add . && git commit -m" # + commit message
alias gbr="git branch" # + branch name
alias gco="git checkout" # + branch name

$ vim ~/.zhsrc
git config --global alias.s status
git config --global alias.d diff
git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.last "log -1 HEAD"
git config --global alias.cane "commit --amend --no-edit"
git config --global alias.pr "pull --rebase"
git config --global alias.lo "log --oneline -n 10"
```

skills

- 配置 git hook
  - 代码分析
  - 代码格式优化
  - 单元测试
  - ...
- 别放大文件，删除真的很麻烦！
- 使用 .gitignore 忽略文件

- 建议使用 git pull - -rebase 拉取代码
- 借用 stash 特性随意切换分支
- 使用 reflog 解决误操作的重定向 HEAD
- 通过git bisect的二分查找来快速揪出问题代码
- git clean -fd
- git cherry-pick
- git commit --amend
- git clone --depth 限制拉取深度，加快克隆代码

“ Q&A ”

- xiaorui.cc

- github.com/rfyiamcool