

Hacking Git - a Blocktree

陈嘉杰

2018 年 10 月

Git 是个啥子

大家知道 Git 是什么吗?

NAME

git - the stupid content tracker

SYNOPSIS

```
git [--version] [--help] [-C <path>] [-c <name>=<value>]
      [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
      [-p|--paginate|-P|--no-pager] [--no-replace-objects] [--bare]
      [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
      [--super-prefix=<path>]
      <command> [<args>]
```

Git 是个啥子

大家知道 Git 是什么吗?

NAME

git - the stupid content tracker

SYNOPSIS

```
git [--version] [--help] [-C <path>] [-c <name>=<value>]
    [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
    [-p|--paginate|-P|--no-pager] [--no-replace-objects] [--bare]
    [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
    [--super-prefix=<path>]
    <command> [<args>]
```

the stupid content tracker

Git 是个啥子

大家知道 Git 是什么吗?

NAME

git - the stupid content tracker

SYNOPSIS

```
git [--version] [--help] [-C <path>] [-c <name>=<value>]
    [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
    [-p|--paginate|-P|--no-pager] [--no-replace-objects] [--bare]
    [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
    [--super-prefix=<path>]
    <command> [<args>]
```

the stupid content tracker

a block tree

先开始动手

咱们边做边讲，先请大家安装一下 Git：

```
$ apt install -y git
```

先开始动手

咱们边做边讲，先请大家安装一下 Git：

```
$ apt install -y git
```

然后设置一下自己的名字和邮箱：

```
$ git config --global user.name "Jiajie Chen"
```

```
$ git config --global user.email "jiegec@qq.com"
```

先开始动手

咱们边做边讲，先请大家安装一下 Git：

```
$ apt install -y git
```

然后设置一下自己的名字和邮箱：

```
$ git config --global user.name "Jiajie Chen"
```

```
$ git config --global user.email "jiegec@qq.com"
```

然后新建一个文件夹，在里面建立一个空的 Git 仓库：

```
$ mkdir learn_git
```

```
$ cd learn_git
```

```
$ git init
```

Git 能做什么

Git 是一个“版本控制系统”，这意味着：

- ▶ 保存你文件修改的历史记录，你可以看到你以前在什么时候都做了什么更改
- ▶ 可以多人同时工作在同一个项目里，合理地把大家写的代码合并
- ▶ 学好了 Git，有利于软工等课程的学习
- ▶ 学会了 Git，你就可以参与到 GitHub 的广阔开发者社区中
- ▶ 学会了 Git，你才能有朝一日成为 Linux Kernel 开发者

不要怕，让我们一点一点来。最后自然也有给大家的一个挑战环节。

保存文件历史记录

假如要记录和妹子的出游记录（基于 NanoApe 真实事件改编）

```
$ echo "2018-10-04 Shenyang" >> log.txt
```

```
$ git status
```

```
On branch master
```

```
No commits yet
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
    log.txt
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

git add

Git 默认情况下不会把你的文件考虑进来。你必须告诉 Git 让它替你管理这个文件的历史记录。我们采用的命令是：

```
$ git add log.txt
```

当然了，它还有别的写法，我们经常会用

```
$ git add .
```

来把当前目录和所有子目录里文件都加进来。小提示：你可以试试 `git add -i` 的效果。执行以后，是这个效果：

```
$ git status
```

```
On branch master
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file:   log.txt
```

年轻人的第一个提交！

Git 有暂存区 (stage) 的概念，允许你在最终决定一个版本前先做一些修改。git add 把当前文件系统的更改添加到暂存区，而 git commit 则把暂存区的更改保存在历史记录中。

首先我们可以看看暂存区有什么内容：

```
$ git diff --cached
```

```
diff --git a/log.txt b/log.txt
new file mode 100644
index 0000000..c52d43f
--- /dev/null
+++ b/log.txt
@@ -0,0 +1 @@
+2018-10-04 Shenyang
```

这个格式叫做 Unified Diff 。以后大家还会经常看到。

年轻人的第一个提交！（续）

然后 NanoApe 检查了一下，时间和地点都没错，保存！

```
$git commit -m "Add shenyang tour"
```

```
[master (root-commit) ffd4912] Add shenyang tour  
1 file changed, 1 insertion(+)  
create mode 100644 log.txt
```

我们再看一下当前的状态：

```
$ git status
```

```
On branch master  
nothing to commit, working tree clean
```

妙啊，我的文件都已经保存进去了。那我想看我之前都做了什么，怎么办？

git log

有一个命令可以查看最近的提交: git log

\$ git log

```
commit ffd4912fdd74c222787623d9263cb35771996611 (HEAD -> master)
Author: Jiajie Chen <jiegec@qq.com>
Date: Thu Oct 25 09:54:58 2018 +0800

    Add shenyang tour
```

可以看到我在这个时候添加了一个提交, 作者是我, 时间是这个时间, 然后这个提交的描述是这个。

git log (续)

怎么看这个提交具体做了什么呢？

```
$ git log -p
```

```
commit ffd4912fdd74c222787623d9263cb35771996611 (HEAD -> master)
Author: Jiajie Chen <jiegec@qq.com>
Date: Thu Oct 25 09:54:58 2018 +0800

    Add shenyang tour

diff --git a/log.txt b/log.txt
new file mode 100644
index 0000000..c52d43f
--- /dev/null
+++ b/log.txt
@@ -0,0 +1 @@
+2018-10-04 Shenyang
```

它还支持很多写法，比如 HEADn.HEAD 表示从显示当前提交往前走两个，从它开始到当前提交的历史记录。

一个提交太单调，我们加多一个

```
$ echo "2018-10-24 128 Days" >> log.txt
```

```
$ git commit -a -m "Add 1024 day"
```

```
$ git log -p
```

注：git commit -a -m "abc" 相当于 git add . && git commit -m "abc"

commit 7560697aa94262d1fec23a10300e77dd48a2c60e (HEAD -> master)

Author: Jiajie Chen <jiegec@qq.com>

Date: Thu Oct 25 10:07:03 2018 +0800

Add 1024 day

diff --git a/log.txt b/log.txt

index c52d43f..21b376d 100644

--- a/log.txt

+++ b/log.txt

@@ -1 +1,2 @@

2018-10-04 Shenyang

+2018-10-24 128 Days

commit ffd4912fdd74c222787623d9263cb35771996611

Author: Jiajie Chen <jiegec@qq.com>

Date: Thu Oct 25 09:54:58 2018 +0800

Add shenyang tour

diff --git a/log.txt b/log.txt

new file mode 100644

index 0000000..c52d43f

--- /dev/null

+++ b/log.txt

@@ -0,0 +1 @@

啊，忘了一件事情

NanoApe: 唔，忘记提妹子的 ID 了，这不行。我得改一下

```
$ cat log.txt
```

```
2018-10-04 Shenyang
```

```
2018-10-24 128 Days with muvseea
```

```
$ git status
```

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   log.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

啊，忘了一件事情（续）

我们可以看到刚刚的更改：

```
$ git diff
```

```
diff --git a/log.txt b/log.txt
index 21b376d..76656d2 100644
--- a/log.txt
+++ b/log.txt
@@ -1,2 +1,2 @@
 2018-10-04 Shenyang
-2018-10-24 128 Days
+2018-10-24 128 Days with muvseea
```

啊，忘了一件事情（续续）

我看刚才的提交不爽，可不可以更改我上一次的提交？可以：

```
$ git commit -a -amend -m "Add 128 days with muvseea"
```

```
$ git log
```

```
commit 182ffe291d1b3c041247b7ee1eb2a27ee61e9023 (HEAD -> master)
```

```
Author: Jiajie Chen <jiegec@qq.com>
```

```
Date: Thu Oct 25 10:07:03 2018 +0800
```

```
Add 128 days with muvseea
```

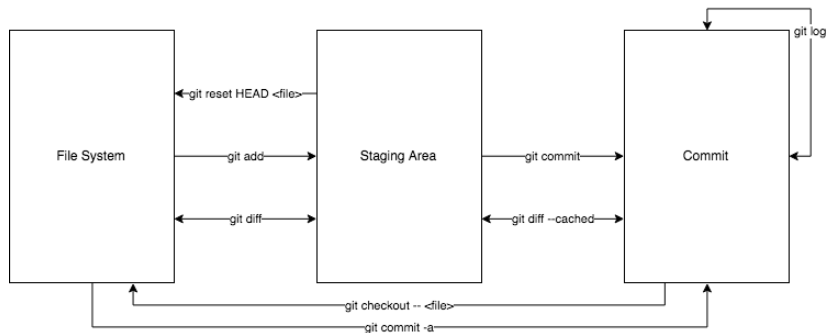
```
commit ffd4912fdd74c222787623d9263cb35771996611
```

```
Author: Jiajie Chen <jiegec@qq.com>
```

```
Date: Thu Oct 25 09:54:58 2018 +0800
```

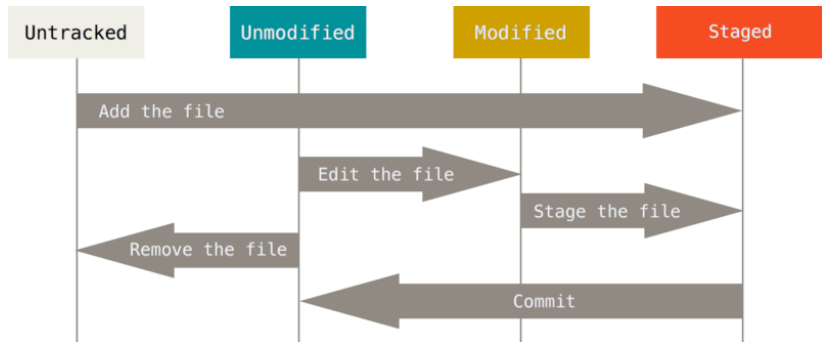
```
Add shenyang tour
```

谈谈暂存区



"git status" is your friend.

Pro Git 上的类似表述



发布虐狗版本!

```
$ git tag nue-gou-1.0
```

```
$ git log
```

```
commit 052a5268e8915a0c5156b22185b9e599a0fe101c (HEAD -> master, tag: nue-gou-1.0)
Author: Jiajie Chen <jiegec@qq.com>
Date: Thu Oct 25 10:07:03 2018 +0800

    Add 128 days with muvseea

commit ffd4912fdd74c222787623d9263cb35771996611
Author: Jiajie Chen <jiegec@qq.com>
Date: Thu Oct 25 09:54:58 2018 +0800

    Add shenyang tour
```

这样我们就创建了一个名为 `nue-gou-1.0` 的 tag。查看所有的 tag：

```
$ git tag
```

```
macbookair ➤ ... > Data > temp > learn_git ➤ git tag
nue-gou-1.0
```

虐狗回顾

假如 NanoApe 虐狗了 99 年的时候，想看一下当年的自己？（好浪漫啊）

```
$ git checkout nue-gou-1.0
```

```
Note: checking out 'nue-gou-1.0'.
```

```
You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.
```

```
If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example:
```

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 052a526 Add 128 days with muvseea
```

这样就可以回到当时的这个版本。实际上就是对某个 commit 进行了命名。

```
$ git checkout master
```

恢复到原来的地方

Git 协作

经过上面这部分，你应该已经学会如何用 Git 管理自己的代码了。但 Git 的强大很大程度在于，可以让很多人一起在一个项目上工作。

我们以清华 Git 为例。

`https://git.tsinghua.edu.cn`

欢迎使用清华大学代码托管服务

欢迎使用清华大学代码托管服务

Sign in

Sign in with

清华统一认证服务登录

☐ Remember me

使用清华 ID 登录即可。

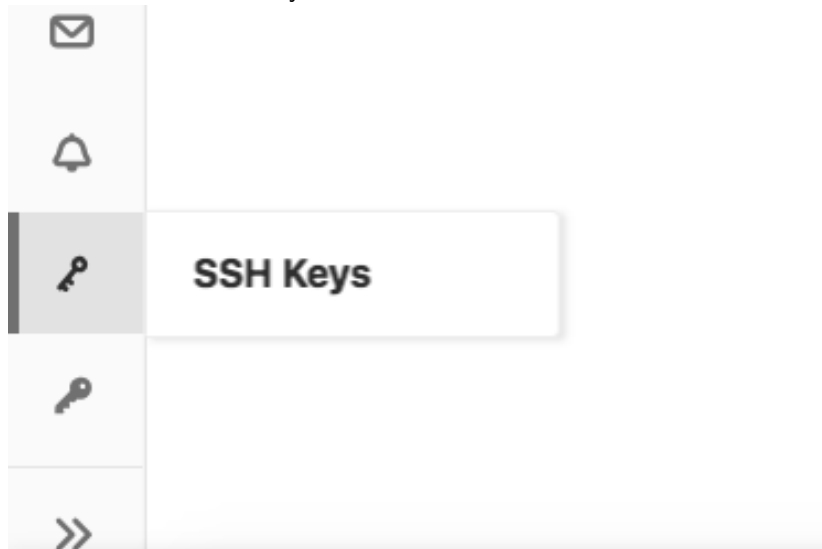
GitLab 配置

进去后，点击右上角的图标，点击 Settings



GitLab 配置 (续)

在左下角找到 SSH Keys



GitLab 配置 (续续)



如果大家还没有生成过，回到终端，生成一个自己的 SSH Key：

```
$ ssh-keygen
```

按照要求生成 SSH Key 。注意保存！拥有这个 Key 就可以对你拥有的 GitLab 仓库进行修改。

然后查看 `~/.ssh/id_rsa.pub` 文件的内容，复制到浏览器中，输入一个名称，最后添加到 GitLab 中。效果如图：

Your SSH keys (1)

 **jiegec@qq.com** 80:47:4d:fe:a8:d8:65:3f:fb:6d:5f:4a:50:cc:b0:8d created 6 months ago 
last used: 1 week ago

测试一下是否成功配置了：

```
$ ssh -T git@git.tsinghua.edu.cn
```

应该会输出 Welcome To GitLab 的信息。

创建仓库

找到 New Project , 进入创建仓库界面

`https://git.tsinghua.edu.cn/projects/new`

Blank project	Create from template	Import project
<div><div>Project path</div><div><div>https://git.tsinghua.edu.cn/</div><div>chenjj17</div><div>▼</div></div></div> <div><div>Project name</div><div>learn_git</div></div> <div>Want to house several dependent projects under the same namespace? Create a group</div> <div><div>Project description (optional)</div><div>Description format</div></div> <div><div>Visibility Level ⓘ</div><div><div><input type="radio"/> Private</div><div>Project access must be granted explicitly to each user.</div></div><div><input checked="" type="radio"/> Internal</div><div>The project can be accessed by any logged in user.</div></div> <div><div><input type="radio"/> Public</div><div>Public visibility has been restricted by the administrator.</div></div>		

Create project

Cancel


点击 Create Project 。


把刚才的仓库推到 GitLab 上


```
$ git remote add origin  
git@git.tsinghua.edu.cn:chenjj17/learn_git.git  
$ git push origin master
```

```
Enumerating objects: 6, done.  
Counting objects: 100% (6/6), done.  
Delta compression using up to 4 threads  
Compressing objects: 100% (3/3), done.  
Writing objects: 100% (6/6), 501 bytes | 250.00 KiB/s, done.  
Total 6 (delta 0), reused 0 (delta 0)  
To git.tsinghua.edu.cn:chenjj17/learn_git.git  
* [new branch]      master -> master
```

这时候就能在 GitLab 上看到我们刚才提交的更改

 Add 128 days with mvuseea
陈嘉杰 authored 1 hour ago

052a5268 

Name	Last commit	Last update
 log.txt	Add 128 days with mvuseea	1 hour ago

多人同时在不同分支协作

这次请出我们的两位主人公，muvseea 和 NanoApe 来给他们的虐狗日记添加一些内容。。。

NanoApe:

```
$ git checkout -b nanoape
```

```
$ ...
```

```
$ git commit -m "Add credit"
```

```
$ git push origin nanoape
```

muvseea:

```
$ git checkout -b muvseea
```

```
$ ...
```

```
$ git commit -m "Add new log and edit an old one"
```

```
$ git push origin muvseea
```

协作!

这时候可以看到，两人都对虐狗日记添加了自己的更改：

L learn_git

🏠 Project

📄 Repository

Files

Commits

Branches

Tags

Contributors

Graph

Compare

陈嘉杰 > learn_git > Graph

master

You can move around the graph by using the arrow keys.

Git revision

🔍

☐ Begin with the selected commit

Oct 25

muvsseea

nanoape

master

📄 Add 100days and edit shenyang

📄 Add credit

📄 Add 128 days with muvsseea

📄 Add shenyang tour

合并？

现在 NanoApe 和 muvseea 都对自己更新很满意了，想要把各自的更新合并回 master 分支里。那么，NanoApe 此时应该打开一个 Merge Request：

L learn_git

Project

Repository

Issues 0

Merge Requests 0

CI / CD


Operations

Wiki

Snippets

Settings

陈嘉杰 > learn_git > Merge Requests



Merge requests are a place to propose changes you've made to a project and discuss those changes with others

Interested parties can even contribute by pushing commits if they want to.

New merge request

合并？（续）

指定要从 nanoape 分支合并入 master 分支：

陈嘉杰 > learn_git > Merge Requests

New Merge Request

Source branch

chenji17/learn_git

nanoape



Add credit

NanoApe authored 4 hours ago

ed6e6937



Target branch

chenji17/learn_git

master



Add 128 days with muvseea

陈嘉杰 authored 6 hours ago

052a5268



Compare branches and continue

点击 Compare branches and continue ， 然后找到下面的 Submit merge request ， 这就创建了一个 Merge Request ：

陈嘉杰 > learn_git > Merge Requests > !1

Open

Opened 32 seconds ago by 陈嘉杰

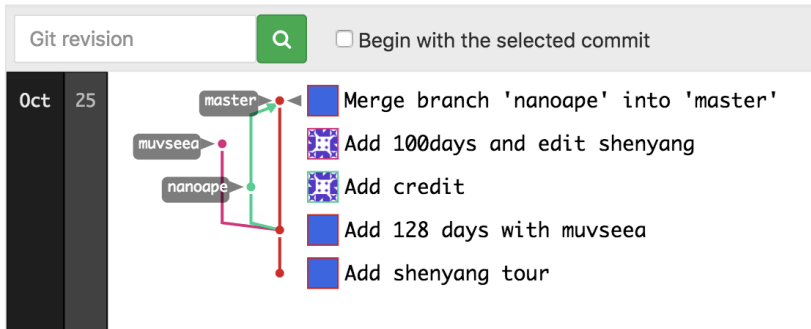
Add credit

Request to merge nanoape into master



合并!

点击 Merge ， 这时候看一下提交历史：



查看此时 master 分支上的 log.txt ， 发现已经有了 NanoApe 进行的更改。

接下来，我们让 muvseea 也来一次同样的操作。

冲突?

我刻意创造了一个情况，就是 NanoApe 和 movseea 对同一行内容进行了修改。此时在 movseea 创建 Merge Requests 的时候，就需要首先 Resolve Conflict：

Add 100days and edit shenyang

Request to merge **movseea**  into **master** (2 commits behind)



Merge

There are merge conflicts

Resolve conflicts

Merge locally

别吵架了！

点击 Resolve conflicts ，我们可以看到哪里冲突了
Add 100days and edit shenyang

Showing 1 conflict between mvuseea and master

Inline Side-by-side

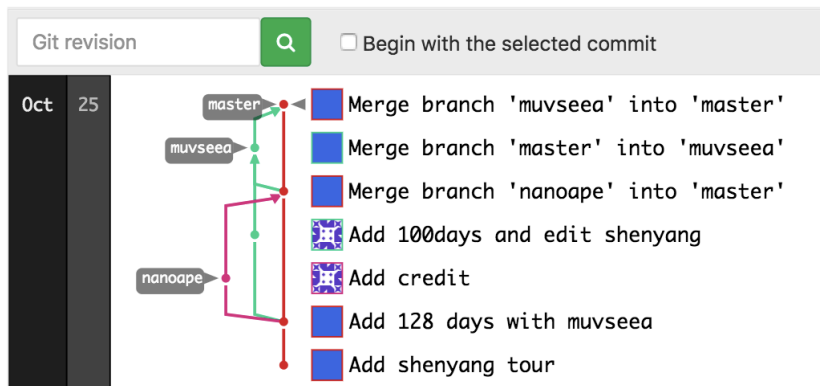
 **log.txt**

Interactive mode Edit inline View file @cf81848

HEAD//our changes		Use ours
1	2018-09-28 100 Days mvse&nano成品>3<	
2	2018-10-04 Shenyang 918	
3	2018-10-24 128 Days with mvuseea	
1	2018-10-04 Shenyang -- NanoApe	
2	2018-10-24 128 Days with mvuseea -- NanoApe	
origin//their changes		Use theirs

上面是 mvuseea 做的更改，下面是 NanoApe 做的更改。这里我们可以选择采用 mvuseea 修改的版本，也可以选择采用 NanoApe 修改的版本，也可以选择 Edit inline 自己进行冲突的合并。手动更改后，点击下面的 Commit 把冲突解决，然后点击 Merge 把修改合并到 master 分支。

等等...发生了什么?



现在, NanoApe 和 muvseea 做的更改都合并到了 master 分支中。
双方的更改都可以看到。

本地和 Gitlab 同步

刚才我们的操作都在 Gitlab 上进行。如何在本地获取到我们在网页上合并的结果呢？

```
$ git fetch
```

```
remote: Counting objects: 5, done.  
remote: Compressing objects: 100% (4/4), done.  
remote: Total 5 (delta 2), reused 0 (delta 0)  
Unpacking objects: 100% (5/5), done.  
From git.tsinghua.edu.cn:chenjj17/learn_git  
    052a526..e1a6a3b  master      -> origin/master  
    cf81848..fb172e2  mvvseea   -> origin/mvvseea
```

本地和 Gitlab 同步

此时的提交历史：

```
* e1a6a3b (26 hours ago) - (origin/master) Merge branch 'muvseea' into 'master' <陈嘉杰>
| \
| * fb172e2 (26 hours ago) - (origin/muvseea) Merge branch 'master' into 'muvseea' <陈嘉杰>
| | \
| | /
| /|
* | 4a7e9be (27 hours ago) - Merge branch 'nanoape' into 'master' <陈嘉杰>
| \ \
| * | ed6e693 (31 hours ago) - (origin/nanoape, nanoape) Add credit <NanoApe>
| / /
| * cf81848 (30 hours ago) - (HEAD -> muvseea) Add 100days and edit shenyang <muvseea>
| /
* 052a526 (33 hours ago) - (tag: nue-gou-1.0, master) Add 128 days with muvseea <Jiajie Chen>
* ffd4912 (34 hours ago) - Add shenyang tour <Jiajie Chen>
```

注意，这里的 origin 代表我们之前添加的 remote，即 Gitlab。这里的 origin/master 代表 Gitlab 上的 master 分支，它现在所在的位置和本地的 master 分支不同。

本地和 Gitlab 同步 (续)

当然了，我们更常用的命令是：

```
$ git pull origin master
```

它不仅会拉取当前上游的提交，而且会把本地的 master 分支更新到上游的版本。

不过，有时候，我们在本地更改了一些内容的时候，这个命令会报错：因为你进行了修改，不允许你直接更新到上游的版本。这时候我们可以这样：

```
$ git stash
```

它会把当前的更改保存为一个临时的 commit 并且回退到之前的状态。之后当你想回到这个修改后的状态的时候，你可以：

```
$ git stash apply
```

它会尝试把你的更改应用过来，不过并不会删掉这个临时的 commit 。你也可以选择

```
$ git stash pop
```

来应用并在成功的时候删除掉它。如果要删掉全部的临时 commit ，你也可以：

```
$ git stash clear
```


Recap

我们想想，如果我们要合并两个人做的事情，应该怎么做？
举个栗子：

1. 有个图书馆，一开始没有书，经过了若干天后，有 3 本书
2. 第一个人看中了一本书，想，今晚我要这本书。
3. 第二个人想还一本书，想，今晚我把这本书还到这里。
4. 到了晚上，图书管理员和这两个人见面了。他看了下第一个人要借书，嗯，拿去。第二个人要还书，嗯，放下。这时候是 $3 - 1 + 1$ 本书。
5. 他记下：书 [0] A 借了；书 [3] B 还了。

这实际上是三个状态的合并：原来的 3 本书；第一个人取走书后的 2 本书；第二个人还了书的 4 本书。

Conflict?

那在什么情况下可能出现冲突呢？
还是这个栗子：

1. 有个图书馆，一开始没有书，经过了若干天后，有 3 本书
2. 第一个人看中了一本书，想，今晚我要这本书。
3. 第二个人也看中了这本书，想，今晚我要这本书。
4. 到了晚上，图书管理员和这两个人见面了。他看了下第一个人和第二个人要借的书一样，唔，行，那我借出去了，但是给谁，你们自己打一架决定。

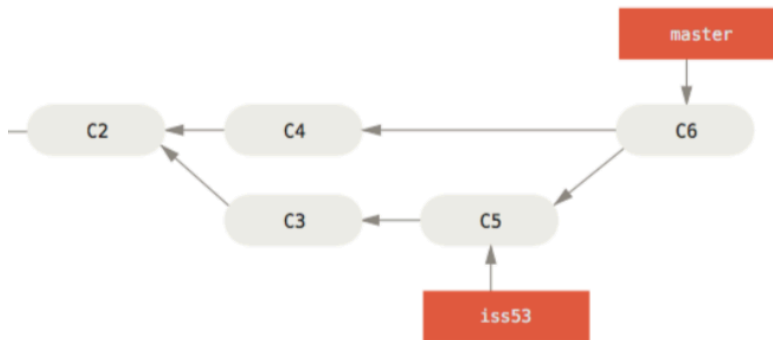
所以冲突的产生在于：不同人对同一个东西做了不同的更改。

Git 的智能合并策略

从上面这个栗子我们可以看到，一些时候我们是完全可以很好地合并两个人的更改的。

而这个更改，实际上就是，从两人的提交的最近共同祖先到这两个提交的改变。

通过把只有一方改变的地方合并起来，把双方都改变的地方要求用户去解决，这就是三路合并。



分支!?

刚才还提到了一个东西：分支。

分支实际上就是一个一直跟随 commit 在变的标记。

比如 master 分支，它代表着项目代码当前最新的一个提交。而 nanoape 分支，则代表 NanoApe 对代码更改的最新的提交。实际上在合并的时候可以直接指定和哪个提交进行合并。

在真实的项目中，我们往往会进一步细分：develop 表示正在活跃开发的版本，release 表示已经发布的最新版本，等等。

分支的常见操作

创建一个新分支，它指向的 commit 为当前 commit

```
$ git branch [name]
```

切换到指定的分支

```
$ git checkout [name]
```

创建并切换到分支

```
$ git checkout -b [name]
```

显示所有的分支

```
$ git branch
```

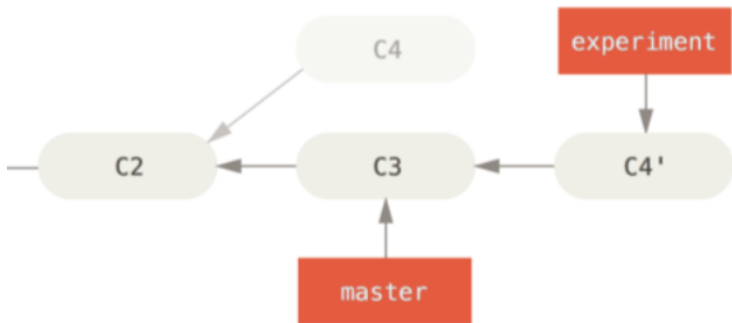
删除分支

```
$ git branch -d [name]
```

合并的骚操作??

不过，合并当然不仅有三路合并这一种方法。大家还有很多骚操作的，比如：

- ▶ 变基 (rebase)。意思就是在新的上游上重放你的更改。当然了，一样会有冲突的问题，但是最后的提交历史会是一条直线。变基可以用的很复杂，我就不深入讲了，太黑科技了。



- ▶ Squash，就是把你的更改全部合并成一个提交，再合并进去。

热热热

烫烫烫热热热 rerere

git 有个不为人知的功能，叫 Rerere (Reuse recorded resolution)。它会记录你解决冲突的历史记录，然后当你下次遇到同样的冲突的时候，它会自动帮你解决。

```
$ git config --global rerere.enabled true
```

即可开启。免费的人工智能！

讲一些奇怪的操作

如果你要魔改提交历史的话：

\$ git rebase [commit]

```
pick 052a526 Add 128 days with muvseea
pick cf81848 Add 100days and edit shenyang

# Rebase ffd4912..cf81848 onto ffd4912 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .      create a merge commit using the original merge commit's
# .      message (or the oneline, if no original merge commit was
# .      specified). Use -c <commit> to reword the commit message.
```

魔改完可以用 \$ git push --force 强制提交到上游

注意，Gitlab 默认禁用了 master 分支的强制 push 。如非必要，请勿打开这个开关。

如果你不小心把密码存在了仓库里并且传给了别人

可以一条命令把一段历史中某个文件都删掉：

```
$ git filter-branch --tree-filter 'rm -f password.txt' HEAD
```

魔改完可以用 `$ git push --force` 强制提交到上游

然后还是建议赶紧改密码。

还可以批量把提交者的名字改了：

```
$ git filter-branch --commit-filter 'GIT_AUTHOR_NAME="Linus Torvalds"
```

```
GIT_AUTHOR_EMAIL="torvalds@linux-foundation.org" git  
commit-tree "$@"
```

这样你就可以伪装成 Linus Torvalds 提交代码了。

当然了，如果不是密码这么敏感的信息，你可以用

```
$ git revert [commit]
```

生成一个效果和某个 commit 抵消的 commit。

如何保证以我的名字提交的就是我提交的？

需要大家配一下自己的 GPG Key ，过程比较复杂，这里就不详细讲了。

配好后，在提交的时候这么写：

```
$ git commit -S -m [message]
```

然后在网页上就会这么显示：



这个基于的是 GPG Key 的信任模型，大家有兴趣的话可以了解一下。

嘿，接锅嘞

突然有一天，代码出了 BUG，发现是一个很愚蠢的地方写错了。你想知道这行代码是谁改的，然后劈头盖脸骂 ta 一顿。怎么看？

\$ git blame [file]

```
cf81848d (muvseea 2018-10-25 13:07:09 +0800 1) 2018-09-28 100 Days mvs&nano成品>3<
fb172e25 (陈嘉杰 2018-10-25 17:00:24 +0800 2) 2018-10-04 Shenyang 918 -- NanoApe
fb172e25 (陈嘉杰 2018-10-25 17:00:24 +0800 3) 2018-10-24 128 Days with muvseea -- NanoApe
```

可以看到每一行最后都是谁改的，什么时候改的，这些内容。

嘿，接锅锅嘞

再设想一种情况：你看了看代码，不知道哪里有问题，但你知道啥时候肯定是好的。那我们自然会想，能不能二分呢？可以：

```
$ git bisect start
```

```
$ git bisect bad
```

这样，标记了当前的 commit 是坏的。然后，再标记一个没有问题的 commit：

```
$ git bisect good [commit]
```

之后，它会自动选择好和坏中间的 commit，你测试一下代码是否工作然后根据好坏继续执行：

```
$ git bisect good/bad
```

最后就可以定位到出错的地方了。

Git 里每个 commit 都有一个很长的十六进制值：它目前是这些内容的 SHA-1（以后会换哈希算法）

- ▶ 提交的信息
- ▶ 提交人
- ▶ 提交日期
- ▶ 作者
- ▶ 编辑日期
- ▶ 整个目录树结构的哈希值

举个栗子

```
$ git rev-parse HEAD | xargs git cat-file -p
```

```
tree df13ecd252f5c53fbfaefeeaf37dcabe639b09b0
parent 4a7e9be0373b80e514ac53664915bdfa7f9f2dab
parent fb172e25268b59acfa49f2ee6b65802dbfea06e0
author 陈嘉杰 <chenjj17@mails.tsinghua.edu.cn> 1540458044 +0800
committer 陈嘉杰 <chenjj17@mails.tsinghua.edu.cn> 1540458044 +0800
```

```
Merge branch 'muvseea' into 'master'
```

```
Add 100days and edit shenyang
```

```
See merge request chenjj17/learn_git!2
```

目录结构是个什么？

```
$ git cat-file -p df13ecd252f5c53fbfaefeeaf37dcabe639b09b0
```

```
100644 blob beea1aa47abadc02495171084f5afdc4f4eb4ade    log.txt
```

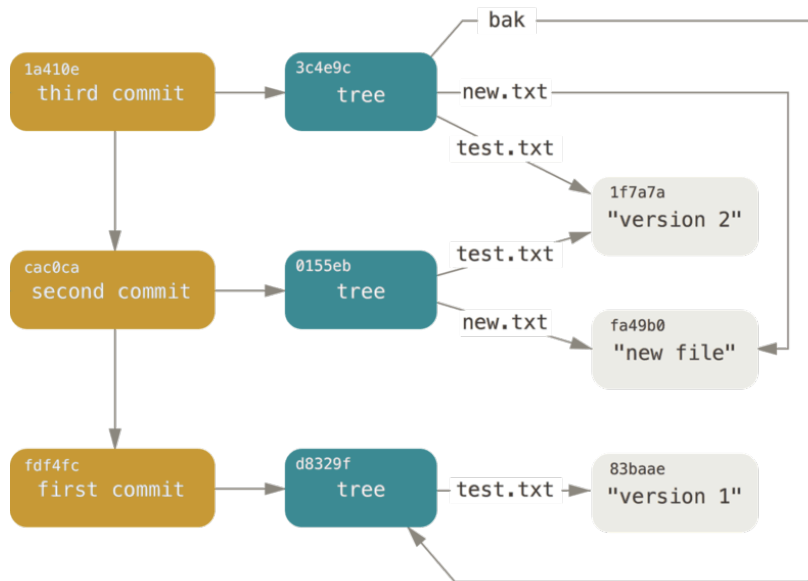
```
$ git cat-file -p beea1aa47abadc02495171084f5afdc4f4eb4ade
```

```
2018-09-28 100 Days mvs&nano成品>3<
```

```
2018-10-04 Shenyang 918 -- NanoApe
```

```
2018-10-24 128 Days with movseea -- NanoApe ↵
```

看个图



如何手动创建一个 commit ?

把文件保存到 Git 中，得到它的哈希值

```
$ git hash-object -w [file]
```

把该文件添加到暂存区

```
$ git update-index --add --cacheinfo 100644 [file_hash] [file]
```

生成当前目录树的哈希

```
$ git write-tree
```

生成提交的哈希

```
$ echo 'commit message' | git commit-tree [tree_hash] -p  
[parent_hash]
```

把当前的 master 分支指向刚刚创建的提交

```
$ git update-ref refs/heads/master [commit_hash]
```

查看 HEAD 的历史

我们知道 HEAD 是一个特殊的“指针”，一直指向目前的提交。而我们常常会不小心把某个 commit 弄丢。也就是说，它不在现在的任何一个分支上。这咋办呢？

```
$ git reflog
```

```
e1a6a3b (HEAD -> master, origin/master) HEAD@{0}: pull origin master: Fast-forward
052a526 (tag: nue-gou-1.0) HEAD@{1}: checkout: moving from mvuseea to master
cf81848 (mvuseea) HEAD@{2}: rebase -i (finish): returning to refs/heads/mvuseea
cf81848 (mvuseea) HEAD@{3}: rebase -i (start): checkout HEAD~
cf81848 (mvuseea) HEAD@{4}: rebase -i (finish): returning to refs/heads/mvuseea
cf81848 (mvuseea) HEAD@{5}: rebase -i (start): checkout HEAD~
cf81848 (mvuseea) HEAD@{6}: reset: moving to origin/mvuseea
ed6e693 (origin/nanoape, nanoape) HEAD@{7}: checkout: moving from nanoape to mvuseea
ed6e693 (origin/nanoape, nanoape) HEAD@{8}: checkout: moving from mvuseea to nanoape
cf81848 (mvuseea) HEAD@{9}: commit: Add 100days and edit shenyang
052a526 (tag: nue-gou-1.0) HEAD@{10}: reset: moving to master
```

它会记录你的 HEAD 变更历史。这样，即便你不小心弄丢了什么，还有很大机会可以找回来。

当然了，你也可以用

```
$ git fsck --full
```

它会找到所有的悬孤提交 (dangling commit)，也是一样的效果。

什么时候孤儿 commit 历史会丢呢？

有个命令，会在你的图上跑个搜索，标记上各个分支可以连接到的 commit，然后把没有标记到的文件删除。

```
$ git gc
```

也就是垃圾回收。同时，为了节省空间，它会把很多个小文件打包成一个大的压缩包，并把一些文件用 diff 的形式保存下来。这在很大的仓库里会非常有用。

Git 的本质是。。。?

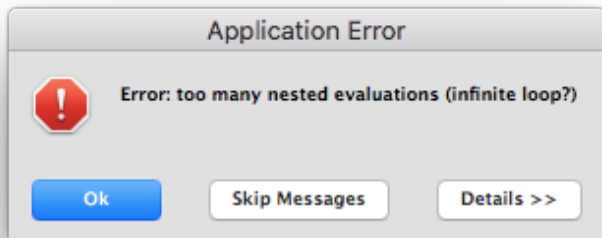
复读机（误

区块树 + Key-Value 存储

每个数据的 Key 是它本身的 SHA-1 哈希值，Value 就是文件本身的内容。

区块树：每个提交记录了它的父提交的 SHA-1 值。在 SHA-1 不能任意制造碰撞前，可以保证这是一棵树，并且确定了一个 commit 的 SHA-1 几乎无法伪造一个假的历史。

最新研究：Git 可以打环辣



dbq 假新闻

采用了一个命令：

```
$ git replace
```

它能做到，把一个 commit 替换成另一个 commit 。一个用途是可以拆分历史后再合并两个历史。

实际上并没有修改这颗区块树，只是人为假装修改了一些边而已。

实测，GitHub 和 GitLab 的绘制 Commit Graph 的工具会直接忽略这个。而 `git log --graph` 检测到打环后直接就停在中间不输出了。

不过，Gitk 就没有这么好运气了。它大概是没人维护了，于是就，无限循环了。

没有讲到的但可能会用到的

- ▶ diff 和 apply
- ▶ textconv 过滤器
- ▶ ident 和 filter
- ▶ hooks 和 CI
- ▶ svn 集成
- ▶ 如何用邮件和别人协作

扩展阅读：墙裂推荐《Pro Git 2》网上有开源版本可以自己构建、阅读。