

# Udacity: Git 版本控制笔记

By Allen Huang

## 1. 集中式模型和分布式模型

集中式模型：所有用户都连接到一个中央的主仓库（master repository）

分布式模型 - 每个用户都在自己的计算机上拥有完整的仓库

Git 属于分布式模型

## 2. Git & GitHub



## 3. 术语

**版本控制系统 (VCS) 或源代码管理器 (SCM)**: VCS 的作用包括：将文件或整个项目还原到之前的状态；查看一段时间内所做的更改；查看是谁做了最后修改而引发问题；引发问题的具体内容以及时间等。

**提交 (snapshot)** : Git 的数据就像一组迷你文件系统的快照。每次在 Git 中提交或保存项目状态时，Git 会立刻对所有文件进行拍照，并存储这组快照的索引。

**仓库 (repo)** : 一个包含项目工作以及用于与 Git 联络的文件（在 Mac OS X 中默认隐藏）的目录，既可存在于本地计算机上，也可作为另一台计算机的远程副本。

**工作目录** : 即计算机文件系统中的文件。当你在代码编辑器中打开项目文件时，就是在处理工作目录中的文件。

工作目录中的文件与仓库中保存（提交中）的文件不同。

使用 Git 时，工作目录也不同于当前工作目录命令行的概念，当前工作目录是你的 shell 正在“查看”的目录。

**检出 / 新建**: 仓库中的内容被复制到工作目录。**文件、提交、分支**等都可从仓库检出。

**暂存区、暂存索引或索引**: Git 目录中的一个文件，用于存储下一次提交的相关信息，相当于 Git 处理下一次提交的准备区域。暂存区中的文件将被添加进仓库。

**SHA:** SHA 基本上就是每次提交的 ID 号码，是由 40 个字符（0-9 和 a-f）组成的字符串，根据 Git 中的文件内容或目录结构计算而成。“SHA”是“SHA 散列”的简写。以下就是一个 SHA 的例子：

e2adf8ae3e2e4ed40add75cc44cf9d0a869afeb6

**分支:** 当创造出一个新的开发线，并从开发主线分离后，就产生了一个分支。这条开发线可在不影响主线的同时继续工作。就像游戏中的进度保存点，你可以把分支想象成游戏中决定尝试冒险之前点击保存点的位置，若冒险失败，就可以回到保存点。分支功能强大的关键在于，你可以在一个分支上创建保存点，然后切换到不同的分支再次创建保存点。

#### 4.3 Areas

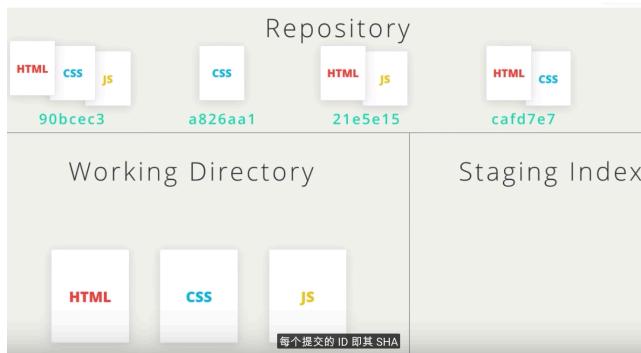


**新的文件:** 首先，现在有一个空的仓库，工作目录下的文件需要先到暂存区（储存所以即将提交的文件）。这时候 commit 就可以将文件传递到仓库中。

**文件修改情况 1:** 这时，我们对 CSS 文件做了一些修改，同理，先到达暂存区，再 commit 到仓库。

**文件修改情况 2:** 暂存区具有 HTML 和 CSS 更改，并在工作目录中具有 HTML 更改。这时候，当我们 commit 只会提交暂存区的内容。除非我们先暂存 HTML 更改，之后关于 HTML 的更改在暂存区会合并。通过 commit 一同到仓库中。

每一次 commit，都会有一个 ID，也就是 SHA。图中显示的是前七位。



#### 5. 安装和终端配置

安装：

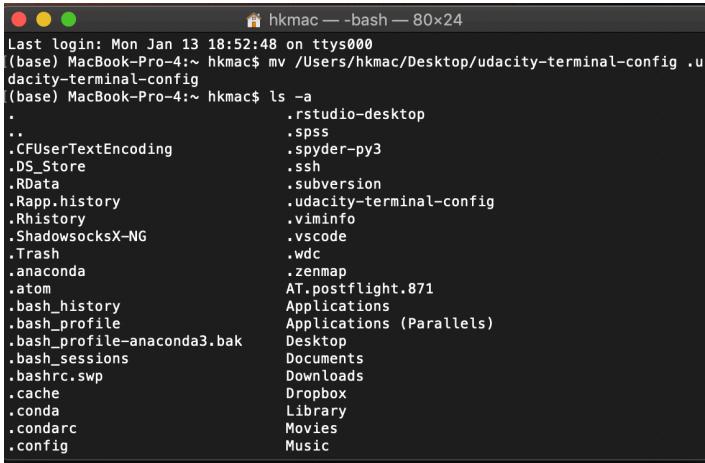
转到 <https://git-scm.com/downloads>  
下载 Mac 版软件  
安装 Git 并选择所有默认选项

## 配置步骤

要配置终端，我们将执行以下步骤：

1. 下载 zip 文件
2. 将目录 `udacity-terminal-config` 移到你的主目录下，并命名为 `.udacity-terminal-config` (注意前面有个点)
3. 将 `bash_profile` 文件移到你的主目录下，并命名为 `.bash_profile` (注意前面有个点)
4. 如果主目录下已经有 `.bash_profile` 文件，则将下载的 `bash_profile` 文件中的内容复制到现有的 `.bash_profile` 文件中

具体终端步骤：



```
hkmac — bash — 80x24
Last login: Mon Jan 13 18:52:48 on ttys000
(base) MacBook-Pro-4:~ hkmac$ mv /Users/hkmac/Desktop/udacity-terminal-config .udacity-terminal-config
(base) MacBook-Pro-4:~ hkmac$ ls -a
.
..
.CFUserTextEncoding
.DS_Store
.RData
.Rapp.history
.Rhistory
.ShadowsocksX-NG
.Trash
.anaconda
.atom
.bash_history
.bash_profile
.bash_profile-anaconda3.bak
.bash_sessions
.bashrc.swp
.cache
.condarc
.config
.rstudio-desktop
.spss
.spyder-py3
.ssh
.subversion
.udacity-terminal-config
.viminfo
.vscode
.wdc
.zenmap
AT.postflight.871
Applications
Applications (Parallels)
Desktop
Documents
Downloads
Dropbox
Library
Movies
Music
```

通过 `ls -a` 发现已经有 `bash_profile` 这个文件。终端输入 `open -e .bash_profile`，进行编辑。粘贴 zip 文件中 `bash_profile` 里面的内容。重启终端。

MAC 访问主目录的常用组合键：(<https://blog.csdn.net/fanshouyizhi/article/details/70141952>)

## 6. 配置 Git

```
# 设置你的 Git 用户名
git config --global user.name "<Your-Full-Name>"
```

```
# 设置你的 Git 邮箱
git config --global user.email "<your-email-address>"
```

```
# 确保 Git 输出内容带有颜色标记
```

```
git config --global color.ui auto  
  
# 对比显示原始状态  
git config --global merge.conflictstyle diff3  
  
git config -list
```

## 7. 创建 Git 仓库

### git init 创建一个仓库

涉及终端命令：

ls - 用来列出文件和目录  
mkdir - 用来新建目录  
cd - 用来更改目录  
rm - 用来删除文件和目录

首先，为了将所有相关文件储存在一个特定目录下，需要以下步骤：

1. 创建一个目录，叫做 udacity-git-course
2. 在该目录中，创建另一个目录，叫做 new-git-project
3. 使用 cd 命令移到 new-git-project 目录下

```
mkdir -p udacity-git-course/new-git-project && cd $_
```

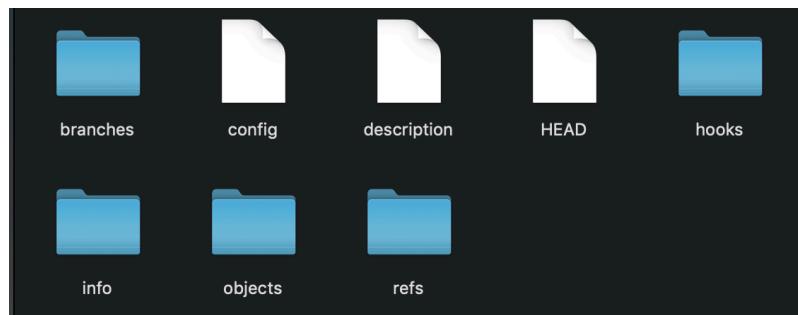
```
Last login: Tue Jan 14 18:36:12 on ttys000  
hkmac (master #) ~  
$ pwd  
/Users/hkmac  
hkmac (master #) ~  
$ mkdir -p udacity-git-course/new-git-project && cd $_  
hkmac (master #) new-git-project  
$ █
```

在当前目录下初始化生成一个空的 git 仓库

```
$ git init  
Initialized empty Git repository in /Users/hkmac/udacity-git-course/new-git-project/.git/
```

运行 git init 命令会初始化 Git 跟踪所有内容会用到的所有必要文件和目录。所有这些文件都存储在叫做 .git (注意开头有个 ., 表示在 Mac/Linux 上，它将是一个隐藏目录) 的目录下。这个 .git 目录是一个库！Git 会将所有 commit 记录在这里，并跟踪所有内容！

我们来大致了解下 .git 目录下的内容。注意显示 MAC 上隐藏文件用 Command+shift+.



**config** 文件 - 存储了所有与项目有关的配置设置。Git 会查看 Git 目录下你当前所使用仓库对应的配置文件 (.git/config) 中的配置值。这些值仅适用于当前仓库。例如，假设你将 Git 全局配置为使用你的个人电子邮箱。如果你想针对某个项目使用你的工作邮箱，则此项更改会被添加到该文件中。

**description** 文件 - 此文件仅用于 GitWeb 程序，因此可以忽略。

**hooks** 目录 - 我们会在此处放置客户端或服务器端脚本，以便用来连接到 Git 的不同生命周期事件。

**info** 目录 - 包含全局排除文件。

**objects** 目录 - 此目录将存储我们提交的所有 commit。

**refs** 目录 - 此目录存储了指向 commit 的指针（通常是“分支”和“标签”）

## 8. 克隆现有仓库

在克隆任何内容之前，确保命令行工具已定位于正确的目录下。克隆项目会新建一个目录，并将克隆的 Git 仓库放在其中。问题是无法创建嵌套的 Git 仓库。因此，确保终端的当前工作目录没有位于 Git 仓库中。如果当前工作目录没有在 shell 的提示符中显示，输入 pwd 输出工作目录。

克隆 blog 仓库：\$ git clone <https://github.com/udacity/course-git-blog-project> (要克隆的仓库的路径，这里是一个 URL)

```
hkmac (master #) ~  
$ git clone https://github.com/udacity/course-git-blog-project  
Cloning into 'course-git-blog-project'...  
remote: Enumerating objects: 131, done.  
remote: Total 131 (delta 0), reused 0 (delta 0), pack-reused 131  
Receiving objects: 100% (131/131), 2.04 MiB | 407.00 KiB/s, done.  
Resolving deltas: 100% (57/57), done.
```

第一行是“Cloning into 'course-git-blog-project'...”。Git 正在创建一个目录（名称与我们要克隆的项目一样）

其余输出结果基本都是验证信息——也就是统计远程仓库的项目数，然后压缩并接收这些项目，并解压。

如果想要克隆 blog 项目仓库并将其存储在叫做 blog-project 的目录下：在后面空格 blog-project。

将目录转移到这个项目中，可以用 open index.html 打开查看。

```
$ cd course-git-blog-project  
hkmac (master) course-git-blog-project  
$
```

### 总结：

该命令会获取现有仓库的路径

默认地将创建一个与被克隆的仓库名称相同的目录

可以提供第二个参数，作为该目录的名称

将在现有工作目录下创建一个新的仓库

## 9. Git status 判断仓库的状态

git status 是了解 Git 的核心所在。它将告诉我们 Git 正在考虑什么，以及 Git 所看到的我们仓库的状态。当你第一次使用 Git 时，你应该一直都要使用 git status 命令！说真的，你应该习惯于运行任何其他命令之后，都运行下该命令。这样可以帮助你了解 Git 的工作原理，并避免你对文件 / 仓库状态做出不正确的推论。

当下：

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
hkmac (master) course-git-blog-project
```

On branch master – 这部分告诉我们 Git 位于 master 分支上，那么这是"master"分支（也就是默认分支）。

Your branch is up-to-date with 'origin/master'. – 因为我们使用 git clone 从另一台计算机上复制了此仓库，因此这部分告诉我们项目是否与所复制的仓库保持同步状态。我们不会在其他计算机上处理该项目，因此这一行可以忽略。

nothing to commit, working directory clean – 表示没有任何待定的更改。

没有新的文件、没有对文件作出更改、暂存区没有任何需要 commit 的内容。

对于之前新建的那个 project：

```
$ cd udacity-git-course/new-git-project
hkmac (master #) new-git-project
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

如果你将此结果与 course-git-blog-project 项目的 git status 输出结果进行对比，你会发现它们很相似。不同之处在于这个输出结果包含 No commits yet。

### 总结：

告诉我们已在工作目录中被创建但 Git 尚未开始跟踪的新文件

Git 正在跟踪的已修改文件

以及我们将在这门课程的后续阶段学习的很多其他信息 ;-)。

## 10. git log

可以看到 log 日志，包括：SHA，提交者，以及提交时间，还有简单描述

```
hkmac (master) course-git-blog-project
$ code index.html
hkmac (master) course-git-blog-project
$ git log
commit a3dc99a197c66ccb87e3f4905502a6c6eddd15b1 (HEAD -> master, origin/master, origin/HEAD)
Author: Richard Kalehoff <richardkalehoff@gmail.com>
Date:   Mon Dec 5 16:34:15 2016 -0500

    Center content on page

commit 6f04ddd1fb41934c52e290bc937e45f9cd5949aa
Author: Richard Kalehoff <richardkalehoff@gmail.com>
Date:   Mon Dec 5 16:30:40 2016 -0500

    Add breakpoint for large-sized screens

commit 50d835d7b53f46deb1365fe7598e0ea7011dbc3e
Author: Richard Kalehoff <richardkalehoff@gmail.com>
Date:   Mon Dec 5 10:39:19 2016 -0500
```

对于 command line 的 less 分页器：

- 要向下滚动，按下 j 或 ↓ 一次向下移动一行
- d 按照一半的屏幕幅面移动
- f 按照整个屏幕幅面移动
- 要向上滚动，按上 k 或 ↑ 一次向上移动一行
- u 按照一半的屏幕幅面移动
- b 按照整个屏幕幅面移动
- 按下 q 可以退出日志（返回普通的命令提示符）

Git log 具有一个选项，`--oneline`，包含简略版 SHA（前七位），和 commit message.

```
$ git log --oneline
a3dc99a (HEAD -> master, origin/master, origin/HEAD) Center content on page
6f04ddd Add breakpoint for large-sized screens
50d835d Add breakpoint for medium-sized screens
0768f3d Add space around page edge
f9f20a9 Style page header
8aa6668 Convert social links from text to images
c165069 Add divider between content/footer
39ff53c Add divider between main/side content
20a3470 Add missing profile picture
8d3ea36 Style 'read more' links
e6a8e07 Set paragraph line-height
4a60beb Set default text color
5de135a Set article timestamp color
921c387 Align article header content
7471693 Make article images responsive
8dec321 Add 'visuallyhidden' helper class
d638b69 Add article images
70f6352 Set default fonts
9a21730 Give body a default color
cdce4fa Add Normalize.css CSS reset to project
10ea3fe Add colors & set better box-sizing
fdf5493 Add sidebar content
8a11b3f Update article dates
4e16c7e Add article content
b4521d7 Add starting HTML structure
40e2199 Add starter files
9362088 Initial commit
```

The git log 命令有一个选项可以用来显示 commit 中更改的文件以及添加或删除的行数。该选项为 **--stat** (stat 是“统计信息 statistics”的简称) :

```
commit 6f04ddd1fb41934c52e290bc937e45f9cd5949aa
Author: Richard Kalehoff <richardkalehoff@gmail.com>
Date:   Mon Dec 5 16:30:40 2016 -0500

    Add breakpoint for large-sized screens

css/app.css | 31 ++++++-----+
index.html  | 118 ++++++-----+
2 files changed, 91 insertions(+), 58 deletions(-)

commit 50d835d7b53f46deb1365fe7598e0ea7011dbc3e
Author: Richard Kalehoff <richardkalehoff@gmail.com>
Date:   Mon Dec 5 10:39:19 2016 -0500

    Add breakpoint for medium-sized screens

css/app.css | 16 ++++++-----+
index.html  | 33 ++++++-----+
2 files changed, 34 insertions(+), 15 deletions(-)
```

比如，css/app.css 改变了，增加了 91 行，删除了 58 行。

git log 命令具有一个可用来显示对文件作出实际更改的选项。该选项是 **--patch**, 可以简写为 **-p**.

patch 显示原始版本和更改版本之间的区别，也就是第一行的 diff, a) 表示第一个版本，b) 表示第二版本，如果 rename 就会改遍这里的名字。

Index 表示原始的 hash 和更改后的 hash。

--- 是 old version, +++是新的 version

```
hkmac (master) course-git-blog-project
[$ git log -p
commit a3dc99a197c66ccb87e3f4905502a6c6eddd15b1 (HEAD --> master, origin/master, origin/HEAD)
Author: Richard Kalehoff <richardkalehoff@gmail.com>
Date:   Mon Dec 5 16:34:15 2016 -0500

    Center content on page

diff --git a/css/app.css b/css/app.css
index 07c36fa..3cbd0b8 100644
--- a/css/app.css
+++ b/css/app.css
@@ -38,6 +38,11 @@ p {
    line-height: 1.5;
}

+.container {
+  margin: auto;
+  max-width: 1300px;
+}
+
/** Header Styling ***/
.page-header {
```

绿色加号后面的是这次添加的，青色表示是在什么位置添加的。旧版本中，这个 code 从第 38 行开始，并显示了 6 行代码，也就是空格行和白色行，加上新的 5 行，变成了 11 行。

用红色标示并以减号 (-) 开头的行是位于文件原始版本中，但是被 commit 删除的行；用绿色标示并以加号 (+) 开头的行是 commit 新加的行。

另外，`git log -p -stat` 同时显示两种信息，并且统计信息显示在补丁信息上方。`Git log -p -m` 会忽略空格的更改。

`git log -p fdf5493` 还可以提供 SHA 作为最后一个参数，显示所提供 SHA 之前提交的所有 commit 信息

## 11. git show

直接运行，`显示最近的 commit`

`git show fdf5493` 添加 SHA 作为参数，`git show` 命令将仅显示一个 commit。

并且 `git show` 可以与我们了解过的大部分其他选项一起使用：

- `--stat` - 显示更改了多少文件，以及添加/删除的行数
- `-p` 或 `--patch` - 显示默认补丁信息，但是如果使用了 `--stat`，将不显示补丁信息，因此传入 `-p` 以再次添加该信息
- `-w` 忽略空格变化

## 12. git add

```
hkmac (master #) ~
$ cd udacity-git-course/new-git-project
hkmac (master #) new-git-project
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
hkmac (master #) new-git-project
$
```

添加了文件之后，注意 git 没有跟踪这个文件，但是它已经在观察这个目录。

```
hkmac (master #) new-git-project
[$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    css
    index.html

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

- 我们新建了几个希望 git 开始跟踪的文件

- 为了让 git 能跟踪文件，需要将该文件提交到仓库中
- 要提交文件，需要将该文件放入暂存区
- 可以使用 git add 命令将文件从工作目录移到暂存区
- 工作目录中目前有三个未跟踪文件
  - index.html
  - css 目录下的 app.css
  - js 目录下的 app.js

首先，使用 git add，将 index.html 添加到暂存区，可接受多个文件名（用空格分隔）

```
hkmac (master #) new-git-project
[$ git add index.html
hkmac (master +) new-git-project
[$ git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file:   index.html

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

输出结果中现在出现了全新的区域："Changes to be committed" 区域！这一新的"Changes to be committed" 区域显示了位于暂存区的文件！目前只显示了 index.html 文件，因此暂存区只有这个文件。继续这一思路，如果我们现在提交 commit，则只有 index.html 文件会被提交。

`git rm --cached`，可以从暂存区删掉文件，但是不会删除原始文件。

`git add .` 句点表示当前所有文件和目录

## 13. git commit

## 14. gitignore

如果你想将某个文件保留在项目的目录结构中，但是确保它不会意外地提交到项目中，可以使用名称特殊的文件 `.gitignore`

将此文件添加到 new-git-project 项目根目录。你只需列出希望 git ignore（忽略，不跟踪）的文件名，git 将忽略这些文件。

通配符：

- 空白行作为空格
- `#` - 将行标记为注释
- `*` - 与 0 个或多个字符匹配
- `?` - 与 1 个字符匹配
- `[abc]` - 与 a、b 或 c 匹配
- `**` - 与嵌套目录匹配 - `a/**/z` 与以下项匹配
  - `a/z`
  - `a/b/z`
  - `a/b/c/z`

比如：在 samples 文件夹中有一些 jpg 未见，可以在.gitignore 中添加：

Samples/\*.jpg

## 15. git tag

目前为止项目的 git log 输出结果如下：

```
$ git log --oneline
638a287d (HEAD -> master, origin/master, origin/HEAD) Time Series Analysis
64de65b2 delete Practice
9ba8fce0 Merge branch 'master' of https://github.com/allen1881996/CrazyAllenData
Science NLP:wq NLP# the commit.
01fa00ff NLP
fd7305bf Update README.md
092065ee Update README.md
4fb0542b Pandas & Matplotlib
839d9ab4 Update README.md
65377428 Update README.md
2b035d0c Create README.md
f9c82726 2019-11-14
faa02574 Pytorch
45d0cac9 2019-11-9
fbe8dbc7 new commit
94a30b75 Third_commit
b4479a22 Delete Numpy.ipynb
2946d7ef second-commit
0ec5ecaa first_commit
```

git tag -a v1.0 中，使用了 -a 选项。该选项告诉 git 创建一个带注释的标签。如果你没有提供该选项（即 git tag v1.0），那么它将创建一个轻量级标签。带注释的标签包含标签创建者，创建日期，标签消息等。标签与 commit 相绑定。因此，该标签与 commit 的 SHA 位于同一行。

```
hkmac (master *)$ CrazyAllenDataScience
$ git tag -a v1.0
hkmac (master *)$ CrazyAllenDataScience
$ git tag
v1.0
hkmac (master *)$ CrazyAllenDataScience
```

在弹出的编辑器界面中输入完毕后，再次输入 git tag 会显示仓库中的所有标签。

使用 git log --decorate (实际上现在已经默认自动包含了这个选项)

```
commit 638a287de22a6c5b79f91db0da2e3a4703d4a32a (HEAD -> master, tag: v1.0, origin/master, origin/HEAD)
Author: Allen <913124925@qq.com>
Date:   Mon Dec 2 16:41:10 2019 -0500
```

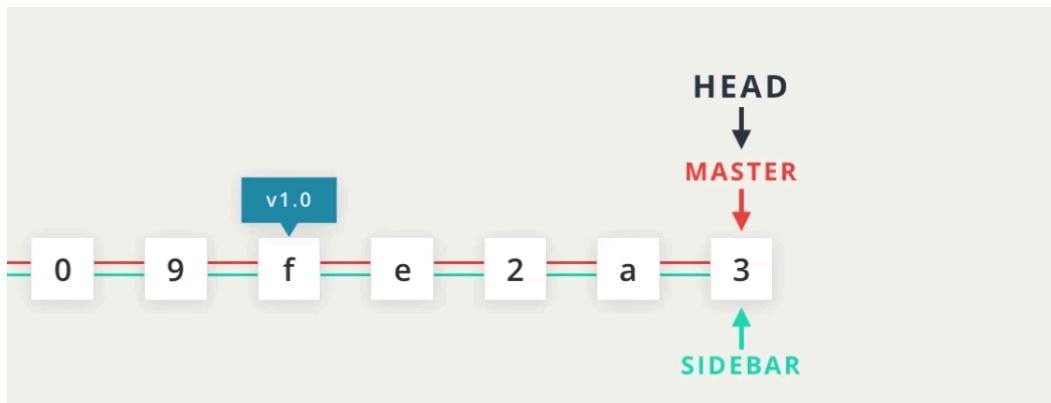
### Time Series Analysis

删除标签 git tag -d v1.0

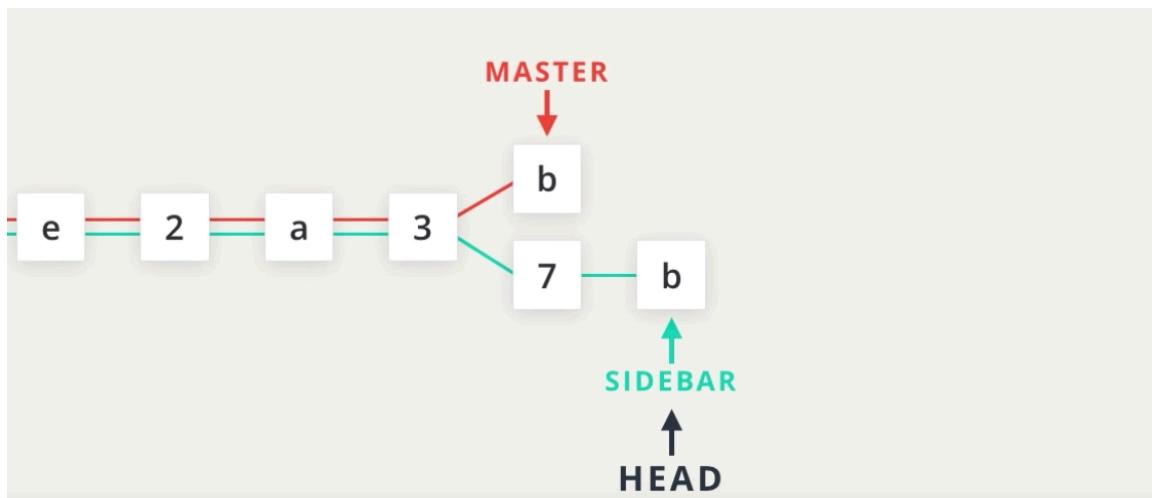
指定某个 commit 添加标签 git tag -a v1.0 a87984

## 16. 分支

第一个分支默认为 master。当我们进行一次提交，这个提交就会被添加到分支上，分支的指针也会移动指向它，tag 是针对某个 commit 的永久指针，不会移动。Sidebar 是我们添加的另外一个分支。



HEAD 指针指向活跃的分支



Git check out sidebar

Git check out master

在不同的分支之间切换。但是注意，git 只会显示 master 分支中的更改。但其他的也在仓库中安全的保存了，只需要将 HEAD 指针切换过去即可。

Git branch: git 的分支进行交互

- 列出仓库中的所有分支名称
- 创建新的分支
- 删除分支

Git branch 显示现在所有的分支

git branch sidebar 创建分支 sidebar

git branch alt-sidebar-loc 42a69f 创建分支，指向特定 commit

git branch -d sidebar 删除分支(如果有独有的 commit, 无法删除)

git branch -D sidebar 强制删除分支

git checkout sidebar 切换到 sidebar

工作方式:

Git 会从工作目录中删除 git 目前跟踪的所有文件和目录，但是这些文件储存在仓库中，因此是不会丢失的。

转到仓库，提取分支指向的 commit 所对应的所有文件和目录。

git checkout -b richards-branch-for-awesome-changes -b 能够用一个命令创建分支并切换到该分支  
git log --oneline --decorate --graph --all 同时查看所有分支

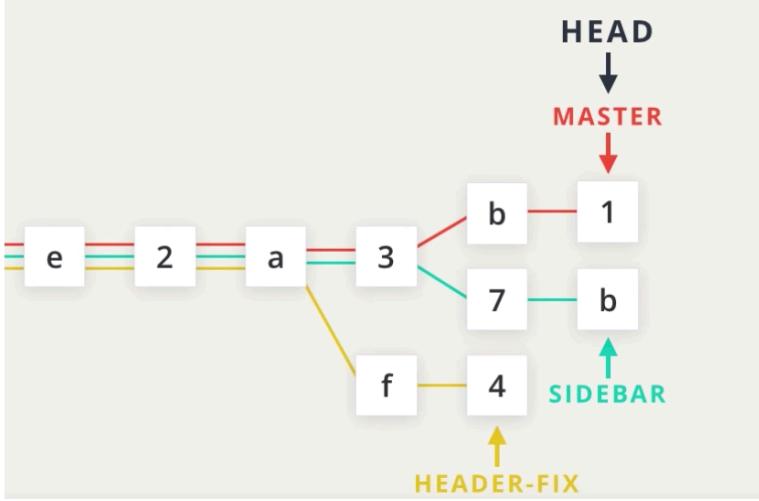
--graph 选项将条目和行添加到输出的最左侧。显示了实际的分支。--all 选项会显示仓库中的所有分支。

运行此命令将显示仓库中的所有分支和 commit:

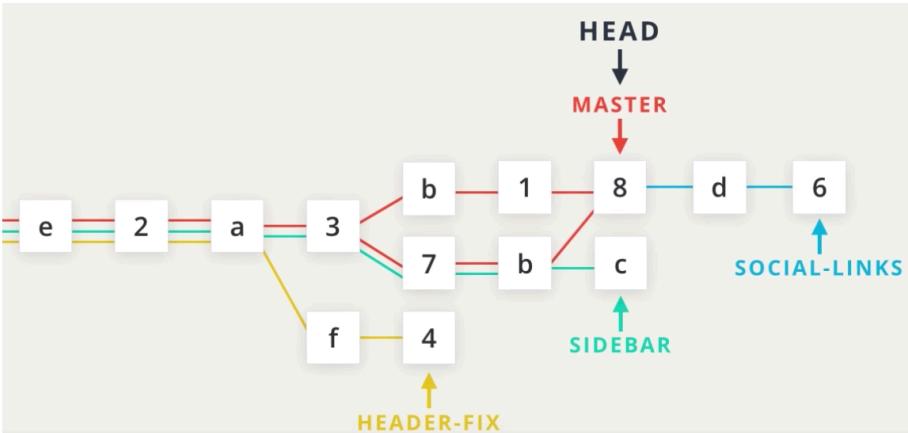
```
* e014d91 (HEAD -> footer) Add links to social media
* 209752a (master) Improve site heading for SEO
* 3772ab1 Set background color for page
| * f69811c (sidebar) Update sidebar with favorite movie
| * e6c65a6 Add new sidebar content
|/
* 5bfe5e7 Add starting HTML structure
* 6fa5f34 Add .gitignore file
* a879849 Add header to blog
* 94de470 Initial commit
(END)
```

## 17. 合并

可以将该分支上的更改与其他分支上的更改合并。



比如，现在想要合并 master 和 sidebar，因为现在指针在 master，合并提交将被放置在 master 分支，它会向前移动。使用 `git merge sidebar` 就可以将 sidebar 的内容合并到 master 上，当然，sidebar 并不会受影响，你依然可以在这个分支上继续向前（C）。

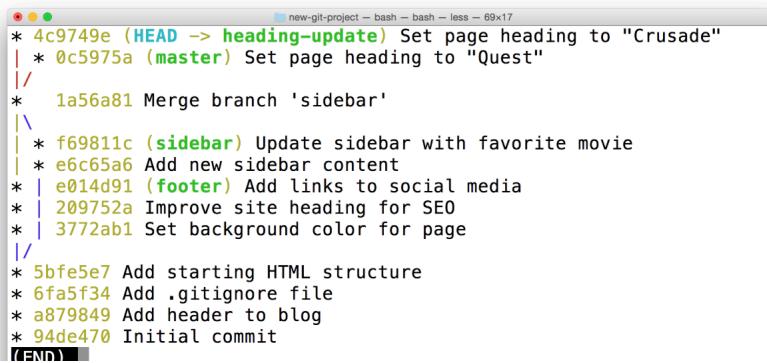


- 查看将合并的分支
- 查看分支的历史记录并寻找两个分支的 commit 历史记录中都有的单个 commit
- 将单个分支上更改的代码行合并到一起
- 提交一个 commit 来记录合并操作

因为 Social-links 直接在 master 前面，因此这种合并最简单。将 Social-links 合并到 master 中将导致快进合并（Fast-forward merge）。快进合并将使当前检出的分支向前移动，直到它指向与另一个分支（这里是 Social-links）指向的 commit 一样为止。

普通类型的合并就是两个完全不同的分支被合并，会创建一个合并 commit，比如 master 和 sidebar 的合并。

合并冲突：git 会跟踪文件中的代码行。如果完全相同的行在不同的文件中更改了，将产生合并冲突。例如，如果你在 alternate-sidebar-style 分支上并将侧栏的标题改为"Information About Me"，git 应该选择哪个标题？你在两个分支上都更改了标题，因此 git 根本不知道你要保留哪个标题。它肯定不会随机选择一个标题！



```
* 4c9749e (HEAD -> heading-update) Set page heading to "Crusade"
| * 0c5975a (master) Set page heading to "Quest"
|/
* 1a56a81 Merge branch 'sidebar'
|\
| * f69811c (sidebar) Update sidebar with favorite movie
| * e6c65a6 Add new sidebar content
* | e014d91 (footer) Add links to social media
* | 209752a Improve site heading for SEO
* | 3772ab1 Set background color for page
|/
* 5bfe5e7 Add starting HTML structure
* 6fa5f34 Add .gitignore file
* a879849 Add header to blog
* 94de470 Initial commit
(END)
```

### 合并冲突指示符解释

编辑器具有以下合并冲突指示符：

<<<<< HEAD 此行下方的所有内容（直到下个指示符）显示了当前分支上的行

||||| merged common ancestors 此行下方的所有内容（直到下个指示符）显示了原始行的内容

===== 表示原始行内容的结束位置，之后的所有行（直到下个指示符）是被合并的当前分支上的行的内容

>>>>> heading-update 是要被合并的分支（此例中是 heading-update 分支）上的行结束指示符

## 18. 删除和撤销

git commit –amend：更改最近的 commit

如果仓库中没有任何未 commit 的更改，就会重启编辑器，显示原始 commit 消息，你只需要修改，关闭编辑器即可。

- 编辑文件
- 保存文件
- 暂存文件
- 运行 git commit –amend

当你告诉 git 还原 (revert) 具体的 commit 时，git 会执行和 commit 中的更改完全相反的更改。我们详细讲解下。假设 commit A 添加了一个字符，如果 git 还原 commit A，那么 git 将创建一个新的 commit，并删掉该字符。如果删掉了一个字符，那么还原该 commit 将把该内容添加回来！

git revert <SHA-of-commit-to-revert>：还原之前创建的 commit

重置：还原会创建一个新的 commit，并还原或撤消之前的 commit。但是重置会清除 commit！

## 相关 commit 引用

你已经知道可以使用 SHA、标签、分支和特殊的 `HEAD` 指针引用 commit。有时候这些并不足够，你可能需要引用相对于另一个 commit 的 commit。例如，有时候你需要告诉 git 调用当前 commit 的前一个 commit，或者是前两个 commit。我们可以使用特殊的“祖先引用”字符来告诉 git 这些相对引用。这些字符为：

- `^` - 表示父 commit
- `~` - 表示第一个父 commit

我们可以通过以下方式引用之前的 commit：

- 父 commit - 以下内容表示当前 commit 的父 commit
- `HEAD^`
- `HEAD~`
- `HEAD~1`
- 祖父 commit - 以下内容表示当前 commit 的祖父 commit
- `HEAD^^`
- `HEAD~2`
- 曾祖父 commit - 以下内容表示当前 commit 的曾祖父 commit
- `HEAD^^^`
- `HEAD~3`

`^` 和 `~` 的区别主要体现在通过合并而创建的 commit 中。合并 commit 具有两个父级。对于合并 commit，`^` 引用来表示第一个父 commit，而 `^2` 表示第二个父 commit。第一个父 commit 是当你运行 `git merge` 时所处的分支，而第二个父 commit 是被合并的分支。

`git reset`

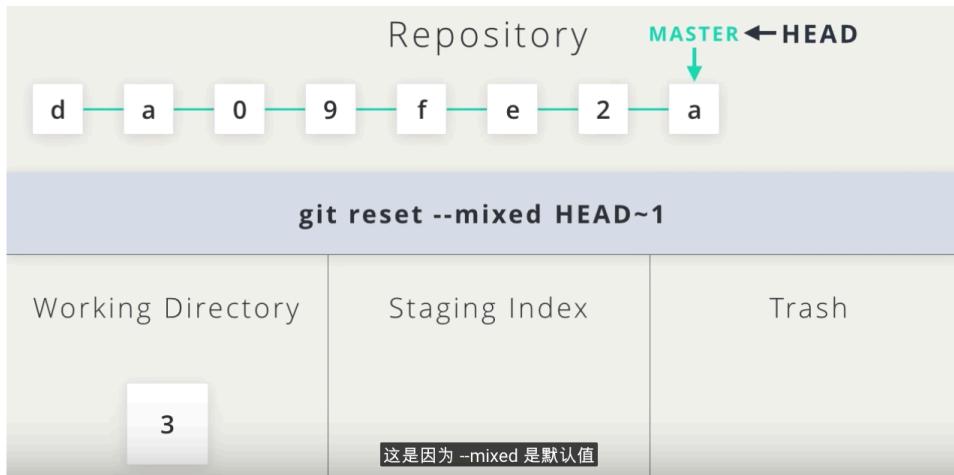
`git reset <reference-to-commit>`

可以用来：

- 将 `HEAD` 和当前分支指针移到目标 commit
- 清除 commit
- 将 commit 的更改移到暂存区
- 取消暂存 commit 的更改

首先，运行 `git reset` 会让指针向后移动，从 3 这个 commit 退回到 a。默认情况下，`--mixed` 是默认的，3 被放在工作目录下。这时候我们如果重新 commit，会 cimmit 一样的内容，只是 SHA 变了。

如果使用 `-soft`, 会放到暂存区, 也就是你只需要 commit  
`--hard` 则会丢进垃圾站



一般在运行 git reset 之前, 先创建一个 backup 分支。如果出现错误可以返回这些 commit.