

# 日常使用 Git 的 19 个建议

2021-08-04

如果你对git一无所知，那么我建议先去读一下Git 常用命令速查。本篇文章主要适合有一定 git 使用基础的人群。

目录：

1. 日志输出参数
  2. 查看文件的详细变更
  3. 查看文件中指定位置的变更
  4. 查看尚未合并（merge）的变更
  5. 查看其他分支中的文件
  6. 关于变更基线(rebase)的几点说明
  7. 本地合并之后保留分支结构
  8. 修复而非新建提交
  9. 的三种状态以及它们的相互转换
  10. 优雅地回退
  11. 使用第三方工具查看整个项目（而非单独文件）的变更
  12. 忽略空格变更
  13. 追加文件中的部分变更
  14. 发现并清理无用分支
  15. 暂存部分文件
  16. 如何写好提交信息
  17. 自动补全
  18. 创建常用命令的别名
  19. 快速定位问题版本
- 1.日志输出参数

命令示例：

*git log --oneline --graph*

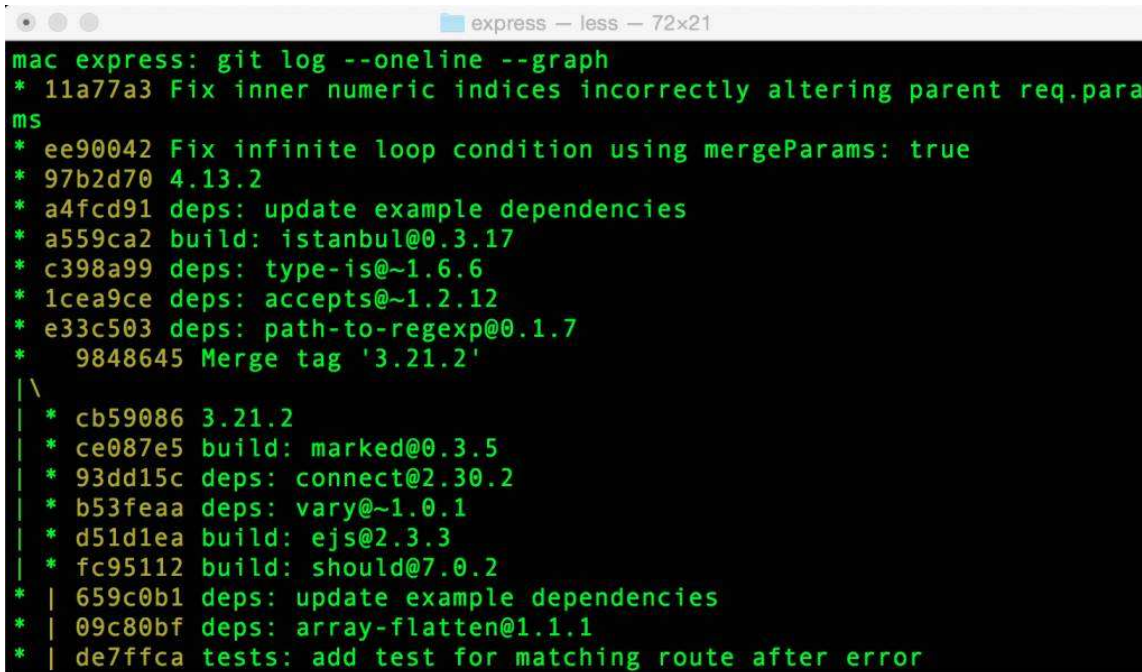
也许你用过git log。它支持很多命令行参数，将这些参数结合起来使用，功能尤为强大。下面是我经常使用的一些参数：

- `-author="Alex Kras"` ——只显示某个用户的提交任务
- `-name-only` ——只显示变更文件的名称
- `-oneline` ——将提交信息压缩到一行显示
- `-graph` ——显示所有提交的依赖树
- `-reverse` ——按照逆序显示提交记录（最先提交的在最前面）
- `-after` ——显示某个日期之后发生的提交
- `-before` ——显示发生某个日期之前的提交

例如，曾经有位主管要求在每周五提交周报。所以我每周五都运行一下这个指令：`git log --author="Alex Kras" --after="1 week ago" --oneline`，然后将输出结果编辑一下，发送给主管审核。

Git有很多命令行参数，使用起来非常方便。运行 `man git log`，来看一下这些参数的作用。

如果这些都不好用，git还有一个 `--pretty` 参数，可以用来创建高级自定义输出。

A terminal window titled 'express - less - 72x21' showing the output of the command 'git log --oneline --graph'. The output is a list of commit hashes followed by their messages, with a graph structure indicated by vertical and diagonal lines. The commits include updates to dependencies, builds, and a merge tag.

```
mac express: git log --oneline --graph
* 11a77a3 Fix inner numeric indices incorrectly altering parent req.params
* ee90042 Fix infinite loop condition using mergeParams: true
* 97b2d70 4.13.2
* a4fcd91 deps: update example dependencies
* a559ca2 build: istanbul@0.3.17
* c398a99 deps: type-is@~1.6.6
* 1cea9ce deps: accepts@~1.2.12
* e33c503 deps: path-to-regexp@0.1.7
* 9848645 Merge tag '3.21.2'
| \
| * cb59086 3.21.2
| * ce087e5 build: marked@0.3.5
| * 93dd15c deps: connect@2.30.2
| * b53feaa deps: vary@~1.0.1
| * d51d1ea build: ejs@2.3.3
| * fc95112 build: should@7.0.2
* | 659c0b1 deps: update example dependencies
* | 09c80bf deps: array-flatten@1.1.1
* | de7ffca tests: add test for matching route after error
```

## 2. 查看文件的详细变更

命令示例：

`git -log -p filename`

`git log -p` 或者 `git log -p filename` 不仅显示提交说明、提交者以及提交日期，还会显示这每次提交实际修改的内容。

然后你就可以使用 `less` 中常用的检索命令即“斜杠”后面加检索词/{{在此处添加你的检索词}})，在变更内容日志中检索指定的关键词（使用小写的`n`跳到下一条检索结果，大写的`N`跳到上一条检索结果）。

A terminal window titled 'express - less - 72x21' showing the output of the command 'git log -p'. The output displays the commit message, author, date, and the diff of the commit. The diff shows changes to a file named 'docs' where 'visionmedia' is replaced with 'tj'.

```
Author: Douglas Christopher Wilson <doug@somethingdoug.com>
Date: Thu Oct 23 02:20:51 2014 -0400

docs: visionmedia is now tj on Github

commit 5f7a37ee517e172c0762fc3debaf94c066e531f9
Author: Fishrock123 <fishrock123@rocketmail.com>
Date: Sat Oct 11 14:12:03 2014 -0400
```

```
docs: misc. tweaks

closes #2394

commit ff3a368b2f9a7e640fb3fd18f116de5352e73d58
Author: Douglas Christopher Wilson <doug@somethingdoug.com>
Date: Thu Oct 23 02:08:34 2014 -0400

    deps: update example dependencies

commit ccc45a74f89b45a6551bc8ff305581dbc8175bca
/visionmedia
```

### 3. 查看文件中指定位置的变更

命令示例：

```
git log -L 1,1:some-file.txt
```

你可以使用 `git blame filename` 追查文件中每一行是由谁变更的。

```
mac express: git blame package.json
903c2aa6 (visionmedia) 2010-03-16 08:31:33 -0700 1) {
ea82ee9 (TJ Holowaychuk) 2010-06-15 13:50:17 -0700 2)   "name": "express",
00000000 (Not Committed Yet) 2015-09-07 14:42:56 -0700 3)   "description": "Sinatra inspired web development framework",
00000000 (Not Committed Yet) 2015-09-07 14:42:56 -0700 4)   "version": "3.20.3",
ccea1dddf (visionmedia) 2010-06-03 16:31:08 -0700 5)   "author": "TJ Holowaychuk <tj@vision-media.ca>",
e2ad0d3d (TJ Holowaychuk) 2012-11-21 08:46:36 -0800 6)   "contributors": [
2e257d1c (Douglas Christopher Wilson) 2014-06-05 19:45:00 -0400 7)     "Aaron Heckmann <aaron.heckmann+github@gmail.com>",
2e257d1c (Douglas Christopher Wilson) 2014-06-05 19:45:00 -0400 8)     "Ciaran Jessup <ciaranj@gmail.com>",
2e257d1c (Douglas Christopher Wilson) 2014-06-05 19:45:00 -0400 9)     "Douglas Christopher Wilson <doug@somethingdoug.com>",
2e257d1c (Douglas Christopher Wilson) 2014-06-05 19:45:00 -0400 10)     "Guillermo Rauch <rauchg@gmail.com>",
2e257d1c (Douglas Christopher Wilson) 2014-06-05 19:45:00 -0400 11)     "Jonathan Ong <me@jongleberry.com>",
00000000 (Not Committed Yet) 2015-09-07 14:42:56 -0700 12)     "Roman Shtylman <shtylman+expressjs@gmail.com>"
faf8095 (Aaron Heckmann) 2010-06-10 21:21:32 -0400 13)   ],
7a7f18c2 (Fishrock123) 2014-10-13 13:41:45 -0400 14)   "license": "MIT",
7a7f18c2 (Fishrock123) 2014-10-13 13:41:45 -0400 15)   "repository": "strongloop/express",
7a7f18c2 (Fishrock123) 2014-10-13 13:41:45 -0400 16)   "homepage": "http://expressjs.com/",
0f49d896 (Douglas Christopher Wilson) 2014-05-18 01:16:38 -0400 17)   "keywords": [
0f49d896 (Douglas Christopher Wilson) 2014-05-18 01:16:38 -0400 18)     "express",
0f49d896 (Douglas Christopher Wilson) 2014-05-18 01:16:38 -0400 19)     "framework",
0f49d896 (Douglas Christopher Wilson) 2014-05-18 01:16:38 -0400 20)     "sinatra",
```

`git blame` 是一个非常强大的工具，但是又是无法提供足够的信息。

`git log` 提供了一个 `-L` 的选项。这个选项允许指定文件中的某些行。Git 只会输出与这些行的变更日志。这有点像带焦点的 `git log -p`。

```
git log -L 1,1:some-file.txt
```

```
mac express: git log -L 6,13:package.json
commit d046208ca2400cfd83e474bf2d4340c60fffb520
Author: Douglas Christopher Wilson <doug@somethingdoug.com>
Date: Wed Jul 16 13:10:19 2014 -0400

    build: add Young Jae Sim as contributor

diff --git a/package.json b/package.json
--- a/package.json
+++ b/package.json
@@ -6,7 +6,8 @@
   "contributors": [
     "Aaron Heckmann <aaron.heckmann+github@gmail.com>",
     "Ciaran Jessup <ciaranj@gmail.com>",
     "Douglas Christopher Wilson <doug@somethingdoug.com>",
     "Guillermo Rauch <rauchg@gmail.com>",
     "Jonathan Ong <me@jongleberry.com>",
     "Roman Shtylman <shtylman+expressjs@gmail.com>",
     "Roman Shtylman <shtylman+expressjs@gmail.com>",
     "Young Jae Sim <hanul@hanul.me>"
   ]

commit 2e257d1cf74440d0ed4c80cd0e2fb7b01ec3896
```

#### 4. 查看尚未合并的变更

命令示例：

```
git log --no-merges master..
```

如果你曾经与很多小伙伴工作在同一个持久分支上，也许会有这样的经历，父分支（例如：master）上的大量合并同步到你当前的分支。这使得我们很难分辨哪些变更时发生主分支，哪些变更发生在当前分支，尚未合并到master分支。

`git log --no-merges master..`可以解决这个问题。注意`--no-merges` 标志意味着只显示没有合并到任何分支的变更，`master..`选项，意思是指显示没有合并到master分支的变更（在master后面必须有`..`）。

你也可以运行 `git show --no-merges master..` 或者 `git log -p --no-merges master..` 命令（输出结果相同）来查看一下尚未合并的文件变更。

#### 5. 查看其他分支中的文件

示例：

```
git show some-branch:some-file.js
```

用这个命令可以很方便地查看其他分支上的文件而无需切换到那个分支。

当然你也可以通过 `git show some-branch-name:some-file-name.js` 命令在终端中显示指定的文件。

你还可以将输出重定向到一个临时文件，这样你就可以再指定的编辑器中，以并排视图来查看它了。

```
git show some-branch-name:some-file-name.js > deleteme.js
```

如果你想查看另一个分支上文件与当前分支上文件的差异，只要运行下面的命令就可以了：

```
git diff some-branch some-filename.js
```

#### 6.关于变更基线的几点说明

示例·

*git pull --rebase*

之前我们说过在远程分支上工作会有大量的合并提交。使用 git rebase 可以避免这些提交。

总的来说我认为变更基线是高级特征，最好是留到另一篇文章中详细介绍。

甚至在git book中也有这样的论述：

*但是，令人狂喜的变更基线并不是任何情况下都适用，一言以蔽之：*

*若是工作区中存在尚未提交到仓库的变更，请不要使用变更基线。*

*如果遵照这条指南，不会有什么问题。不然，你可能会招致厌恶与谩骂。*

<https://git-scm.com/book/en/v2/Git-Branching-Rebasing#The-Perils-of-Rebasing>

也就是说，变更基线本身并不可怕，关键在于使用方式。

或许，最好的方法是使用交互式变更基线，调用命令为 `git rebase -i {{某个提交序列号}}`。运行这条命令，会打开一个带有自解释指令的编辑器。由于变更基线不在本文的叙述范围之内，我们就此而止，不再深究。

```

express — vim — 76x30
.g/r/git-rebase-todo
pick afece96 Commit 1
pick a6671aa Commit 2
pick 2261d76 Commit 3
pick 8f462e7 Commit 4
pick 7c9d609 Fist bad commit
pick e549232 Commit 5
pick e5efc0e Commit 6

# Rebase 11a77a3..e5efc0e onto 11a77a3
#
# 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
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out

```



```
<rebase-merge/git-rebase-todo CWD: /Users/alexk/Desktop/express Line: 1
```

**变更基线一个非常有用的特殊用法 `git pull --rebase`。**

举个例子，假设你正在master分支的一个本地版本上工作，你已经向仓库提交了一小部分变更。与此同时，也有人向master分支提交了他一周的工作成果。当你尝试推送本地变更时，git提示你需要先运行一下 `git pull`，来解决冲突。作为一个称职的工程师，你运行了一下 `git pull`，并且git自动生成了如下的提交信息。

*Merge remote-tracking branch 'origin/master'*

尽管这不是什么大问题，也完全安全，但是不太有利于历史记录聚合。

这种情况下，`git pull --rebase` 是一个不错的选择。

这个命令会迫使git将远程分支上的变更同步到本地，然后将尚未推送的提交重新应用到这个最新版本，就好象它们刚刚发生一样。这样就可以避免合并以及随之而来的丑陋的合并信息了。

## 7. 本地合并之后保留分支结构

示例：

*git merge --no-ff*

我喜欢为每一个bug或者特征创建一个新的分支。最大的好处就是，可以清楚地知道一系列的提交与某个任务的关系。如果你曾经合并过github 或者类似工具上的同步请求，那么可以通过运行 `git log --oneline --graph` 显示的视图，清楚地看到合并分支的历史。

如果你试图合并一个本地分支到另一个本地分支，也许会注意到git将两个分支平滑地衔接在一起，在git历史中看到的是一条直线。。

如果你想强迫git保存分支的历史，与合并同步请求的状况类似，你可以加一个 `--no-ff` 标志，最后可以看到非常清楚的提交历史树。

`git merge --no-ff some-branch-name`

```
mac express: git merge --no-ff no-ff-demo
Merge made by the 'recursive' strategy.
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
mac express:
mac express: git log --oneline --graph
```

```

* 397215d Merge branch 'no-ff-demo'
| \
| * b4ae5f8 Another commit
| * 8abe7f9 Some commit
| /
* 11a77a3 Fix inner numeric indices incorrectly altering parent req.params
* ee90042 Fix infinite loop condition using mergeParams: true
* 97b2d70 4.13.2
* a4fcd91 deps: update example dependencies
* a559ca2 build: istanbul@0.3.17
* c398a99 deps: type-is@~1.6.6
* 1cea9ce deps: accepts@~1.2.12
* e33c503 deps: path-to-regexp@0.1.7
* 9848645 Merge tag '3.21.2'

```

## 8. 修复而非新建提交

示例：

*git commit --amend*

这个指令顾名思义。

假设提交之后，你意识到自己犯了一个拼写错误。你可以重新提交一次，并附上描述你的错误的提交信息。但是，还有一个更好的方法：

如果提交尚未推送到远程分支，那么按照下面步骤简单操作一下就可以了：

1. 修复你的拼写错误
2. 将修正过的文件暂存，通过 `git add some-fixed-file.js`
3. 运行 `git commit --amend` 命令，将会把最近一次的变更追加到你最新的提交。同时也会给你一个编辑提交信息的机会。
4. 准备好之后，将干净的分支推送到远程分支。

如果你工作在自己的分支，甚至可以在已经推送之后修正提交，你需要使用 `git push -f`（-f 代表强制执行），这条指令可以重写历史。但是，不要试图在一个很多人共同工作的分支（正如我们在变更基线那一部分讨论的分支）上这样做。此时，新建一次提交，在提交信息中描述错误，应该是最好的补救措施。

## 9. Git 中的三种状态以及它们之间的转换

示例：

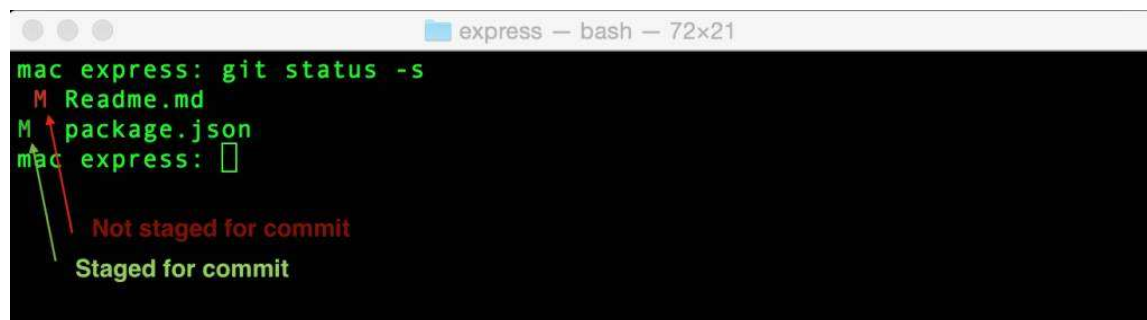
*git reset --hard HEAD* 与 *git status -s*

目前你或许已经了解，git 中的文件可以有三种不同的状态：

1. 没有暂存
2. 暂存并准备提交
3. 已经提交

通过运行 `git status` 可以看到关于文件的描述以及文件的状态。运行 `git add filename.js` 命令可以将文件从未暂存状态移动到暂存并准备提交的状态，或者使用 `git add .` 命令一次性暂存所有的文件。

通过运行 `git status -s` 命令可以看到状态图，其中 `-s` 是简短 (short) 的意思 (个人认为)，最终输出结果如图所示：



```
mac express: git status -s
M README.md
M package.json
mac express: 
Not staged for commit
Staged for commit
```

显然，`git status` 不显示已经提交了的文件，你可以使用 `git log` 命令来查看。

若要将文件在不同阶段之间转换，有很多可以用的命令供你选择。

## 重置文件

在git中，有3种类型的重置。重置是让文件回到git历史中的一个特定版本。

1. `git reset --hard {{some-commit-hash}}` —— 回退到一个特定的历史版本。丢弃这次提交之后的所有变更。
2. `git reset {{some-commit-hash}}` —— 回滚到一个特定的历史版本。将这个版本之后的所有变更移动到“未暂存”的阶段。这也就意味着你需要运行 `git add .` 和 `git commit` 才能把这些变更提交到仓库。
3. `git reset --soft {{some-commit-hash}}` —— 回滚到一个特定的历史版本。将这次提交之后所有的变更移动到暂存并准备提交阶段。意味着你只需要运行 `git commit` 就可以把这些变更提交到仓库。

这些命令似乎并没有什么用处，但当你尝试着将文件在不同版本间移动时，使用它们会非常方便。

我平时使用重置的一些用例如下：

1. 如果想清除变更记录，可以使用清理命令——`git reset --hard HEAD`（最常用）
2. 如果想编辑，以不同的顺序，重新暂存，重新提交文件——`git reset {{some-start-point-hash}}`
3. `git reset --soft {{some-start-point-hash}}` 如果想把之前3次的提交，作为一次提交 `git reset --soft {{some-start-point-hash}}`

## 签出部分文件



如果你想取消某些文件在本地的变更，而同时保留另外一些文件在本地的变更，一个比较简单的方法是通过 `git checkout forget-my-changes.js` 签出那些你想取消本地的变更的文件。

正如前面提到的那样，你也可以从其他分支或者之前的提交中签出文件的不同版本。

`git checkout some-branch-name file-name.js` 和 `git checkout {{some-commit-hash}} file-name.js`

你应该注意到了签出的文件处于“暂存并准备提交”的状态。如果想回到未暂存的状态，需要执行一下 `git reset HEAD file-name.js`。然后再次执行 `git checkout file-name.js`，文件回到了初始状态。

注意，运行 `git reset --hard HEAD file-name.js` 不起作用。总而言之，在git的不同阶段之间移动有点复杂，没有一个清晰的模式，我希望能通过这一部分有所改观。

## 10. 撤销而不产生提交信息

示例：

```
git revert -n
```

如果打算撤销之前一次或者两次的提交，查看这些提交都做了哪些变更，哪些变更又有可能引发问题，这个命令非常方便。

通常，`git revert` 会自动将回退的文件提交到仓库，需要你写一个新的提交信息。`-n` 标志告诉git先别急着提交，因为我只是想看一眼罢了。

## 11. 用第三方差异工具查看整个工程而非单个目录的差异

示例：

```
git difftool -d
```

我最喜欢的差异工具是Meld。我在使用Linux的时候就开始使用它，并且一直持续到现在。

我并不是要推销Meld。假设你已经选好了比较工具，并且git能够将它作为一个合并和差异工具使用。接下来需要运行一下下面的命令，注意用你选择的差异工具的名字代替Meld：

```
git config --global diff.tool.meld
```

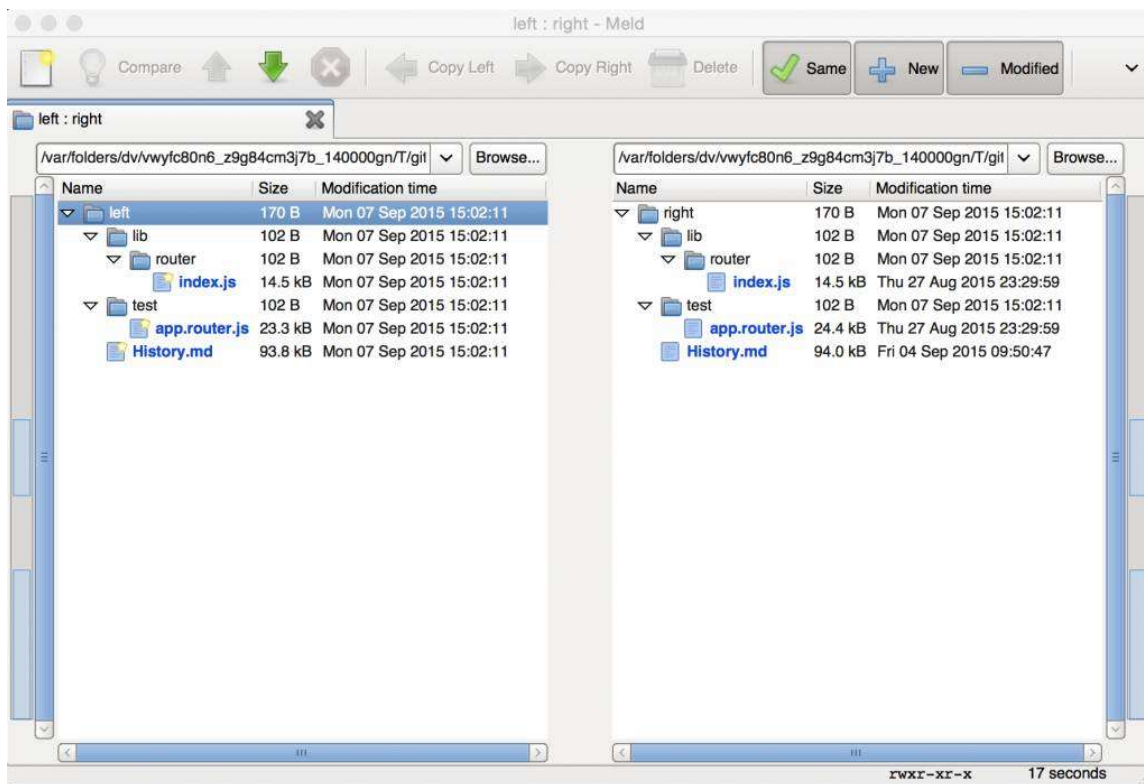
```
git config --global merge.tool meld
```

之后你就可以运行 `run git difftool some-file.js` 来查看文件的差异了。

但是，有些比较工具（例如 `meld`）支持全路径比较。

如果你调用 `git difftool` 时加 `-d` 标志，将会对整个文件夹进行比较。有时会非常有用。

`git difftool -d`



## 12. 忽略空格变更

示例：

`git diff -w` 或者 `git blame -w`

你是否遇到这样的情况：直到 `git blame` 显示你应该为文件中的一切负责时才意识到自己重新调整了文件的缩进或者格式？

结果证明，`git` 足够聪明来分辨文件中不同类型的变更。你可以调用许多命令（例如：`git diff`, `git blame`），加一个 `-w` 标志，`git` 将会忽略空白的变更。

## 13. 追加文件中的部分变更

示例：

*git add -p*

```
mac express: git add -p
diff --git a/package.json b/package.json
index 3b4389e..5112b4d 100644
--- a/package.json
+++ b/package.json
@@ -1,5 +1,5 @@
 {
-  "name": "express",
+  "name": "express-custom",
   "description": "Fast, unopinionated, minimalist web framework",
   "version": "4.13.2",
   "author": "TJ Holowaychuk <tj@vision-media.ca>",
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? ?
y - stage this hunk
n - do not stage this hunk
q - quit; do not stage this hunk or any of the remaining ones
a - stage this hunk and all later hunks in the file
d - do not stage this hunk or any of the later hunks in the file
g - select a hunk to go to
/ - search for a hunk matching the given regex
j - leave this hunk undecided, see next undecided hunk
J - leave this hunk undecided, see next hunk
k - leave this hunk undecided, see previous undecided hunk
K - leave this hunk undecided, see previous hunk
s - split the current hunk into smaller hunks
e - manually edit the current hunk
? - print help
@@ -1,5 +1,5 @@
 {
-  "name": "express",
+  "name": "express-custom",
   "description": "Fast, unopinionated, minimalist web framework",
   "version": "4.13.2",
   "author": "TJ Holowaychuk <tj@vision-media.ca>",
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? 
```

## 14. 发现并清理无用分支

示例:

*git branch -a*

仓库中存在大量远程分支的现象非常常见，甚至其中某些分支已经被合并到了master分支。如果你跟我一样有洁癖（至少有代码洁癖），这些分支可能会令你难以忍受。

你可以通过运行git branch查看所有的远程分支，还可以带有 -a 标志（显示所有的分支），或者带上 -merged标志 只显示那些完全合并到master分支的分支。

你或许首先想到的是运行git fetch -p（获取和清除旧数据），来确保你的数据是最新的。

```

mac express: git branch -a
* master
  remotes/origin/1.x
  remotes/origin/2.x
  remotes/origin/3.x
  remotes/origin/4.x
  remotes/origin/5.0
  remotes/origin/5.x
  remotes/origin/HEAD -> origin/master
  remotes/origin/benchmark
  remotes/origin/external-isAbsolute
  remotes/origin/master
  remotes/origin/mscdex-router-optimize
  remotes/origin/slim-benchmark
  remotes/origin/streaming-render
mac express: git branch -a --merged
* master
  remotes/origin/3.x
  remotes/origin/4.x
mac express: 

```

如果你要获取真正的fancy，你可以得到一个所有远程分支的列表，以及这些分支最后一次提交的列表，通过运行：

`git for-each-ref --sort=committerdate --format='%(refname:short) * %(authorname) * %(committerdate:relative)' refs/remotes/ | column -t -s '*'`.

```

mac express: git for-each-ref --sort=committerdate --format='%(refname:short) * %(authorname) * %(committerdate:relative)' refs/remotes/ | column -t -s '*'
origin/1.x                Tj Holowaychuk                4 years, 6 months ago
origin/2.x                TJ Holowaychuk                3 years ago
origin/streaming-render   Roman Shtylman                1 year, 10 months ago
origin/benchmark          Douglas Christopher Wilson     1 year, 2 months ago
origin/mscdex-router-optimize Douglas Christopher Wilson     1 year, 1 month ago
origin/slim-benchmark     Douglas Christopher Wilson     1 year, 1 month ago
origin/5.x                Douglas Christopher Wilson     10 months ago
origin/external-isAbsolute Jeremiah Senkpiel              5 months ago
origin/4.x                Douglas Christopher Wilson     9 weeks ago
origin/5.0                Douglas Christopher Wilson     9 weeks ago
origin/3.x                Douglas Christopher Wilson     5 weeks ago
origin/master             Brendan Ashworth              5 weeks ago
origin/HEAD               Brendan Ashworth              5 weeks ago
mac express: 

```

不幸地是，没有简单的方法（至少我不知道）可能只显示合并过的分支。所以你可能不得不比较两个输出或者写一个脚本来做这些事情。

## 15. 暂存部分文件

示例：

`git stash --keep-index` 或者 `git stash -p`

如果你还不了解 `git stash` 的功能，只是把当前工作区中的变更保存到一个有序的“git stack”。之后你可以用 `git stash pop`，恢复你的变更。你也可以使用 `git stash list` 查看git栈里面你做的所有备份。通过 `man git stash` 查看更多可以用的选项。

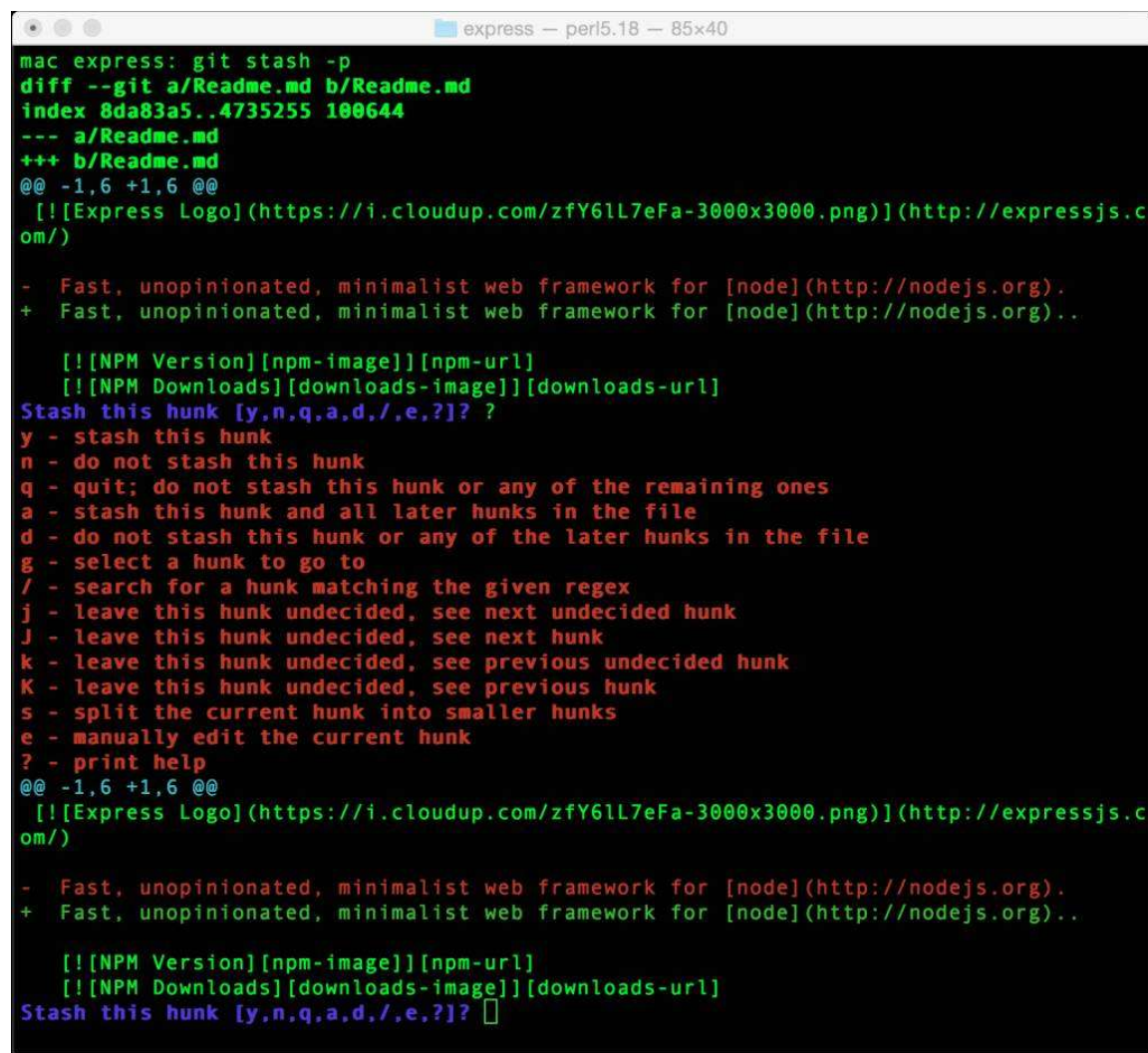


历。通过 `man git stash` 且有更多可以用的选项。

常规 `git stash` 的一个限制是它会一下暂存所有的文件。有时，只备份某些文件更为方便，让另外一些与代码库保持一致。

还记得神奇的 `-p` 命令吗？是的，它与 `git stash` 一起用会非常方便。你现在或许已经猜到了，它会询问你想备份哪些文件的变更。

为了确认一下，点击？你可以看到所有可用的选项。

A terminal window titled 'express - perl5.18 - 85x40' showing the execution of 'git stash -p'. It displays a diff of 'a/Readme.md' vs 'b/Readme.md' with a hunk of code for the Express.js logo and a list of options for stashing the hunk. The options include 'y' (stash this hunk), 'n' (do not stash this hunk), 'q' (quit), 'a' (stash this hunk and all later hunks), 'd' (do not stash this hunk), 'g' (select a hunk), '/' (search for a hunk), 'j' and 'J' (leave hunk undecided), 'k' and 'K' (leave hunk undecided), 's' (split hunk), 'e' (manually edit hunk), and '?' (print help). The prompt 'Stash this hunk [y,n,q,a,d,/,e,?]? ' is shown at the end of the diff output.

```
mac express: git stash -p
diff --git a/Readme.md b/Readme.md
index 8da83a5..4735255 100644
--- a/Readme.md
+++ b/Readme.md
@@ -1,6 +1,6 @@
 [[Express Logo](https://i.cloudup.com/zfY6lL7eFa-3000x3000.png)](http://expressjs.com/)

- Fast, unopinionated, minimalist web framework for [node](http://nodejs.org).
+ Fast, unopinionated, minimalist web framework for [node](http://nodejs.org)..

  [[NPM Version]][npm-image]][npm-url]
  [[NPM Downloads]][downloads-image]][downloads-url]
Stash this hunk [y,n,q,a,d,/,e,?]? ?
y - stash this hunk
n - do not stash this hunk
q - quit: do not stash this hunk or any of the remaining ones
a - stash this hunk and all later hunks in the file
d - do not stash this hunk or any of the later hunks in the file
g - select a hunk to go to
/ - search for a hunk matching the given regex
j - leave this hunk undecided, see next undecided hunk
J - leave this hunk undecided, see next hunk
k - leave this hunk undecided, see previous undecided hunk
K - leave this hunk undecided, see previous hunk
s - split the current hunk into smaller hunks
e - manually edit the current hunk
? - print help
@@ -1,6 +1,6 @@
 [[Express Logo](https://i.cloudup.com/zfY6lL7eFa-3000x3000.png)](http://expressjs.com/)

- Fast, unopinionated, minimalist web framework for [node](http://nodejs.org).
+ Fast, unopinionated, minimalist web framework for [node](http://nodejs.org)..

  [[NPM Version]][npm-image]][npm-url]
  [[NPM Downloads]][downloads-image]][downloads-url]
Stash this hunk [y,n,q,a,d,/,e,?]? 
```

另一个非常实用的技巧，用来备份部分文件：

1. add 那些你不想备份的文件（例如： `git add file1.js, file2.js`）
2. 调用 `git stash -keep-index`。只会备份那些没有被add的文件。
3. 调用 `git reset` 取消已经add的文件的备份，继续自己的工作。

## 16. 写好提交信息

刚刚读过一篇很好的文章，关于如何写好提交信息，点击这个链接阅读：How to Write a Git Commit Message

有一个规则真的是为我量身订做的：“每一个好的提交应该能完善下面的这个句子”



应用到实际中，提交信息应该是这样的：{{你的提交信息}}

例如：

—应用这次提交，可以更新\*\*README文件\*\*

—应用这次提交:为调用GET/user/:id API追加确认

—应用这次提交：会回退到\*\*12345版本\*\*

## 17. Git 自动补全

某些操作系统（例如：Ubuntu）的git包自带并且默认开启自动补全。如果你的操作系统没有这个功能（Mac就没有），你可以按照下面的指南为自己添加。

<https://git-scm.com/book/en/v1/Git-Basics-Tips-and-Tricks#Auto-Completion>

## 18. 创建常用命令的别名

常用的较长的git命令应该使用git或者bash别名

使用Git最好的方式是通过命令行，学习命令行的最好方式就是先用最困难的方法做每一件事。（把一切都打印出来）。

然而，一段时间之后，最好将你常用的命令总结出来，为它们创建一个简单的别名。

Git 支持别名，例如，你可以运行一下下面的命令：

```
git config --global alias.l "log --oneline --graph"
```

这个命令行会创建一个新的git别名l，你可以使用：

git l 代替 git log --oneline --graph。

注意你可以在alias后面附加其他的参数（例如：git l --author = "Alex"）

其他的选项，是好的就得Bash别名

例如，在我的.bashrc文件中有下面的词条：

Alias gil="git log --oneline --graph"，允许我使用gil代替长命令，甚至比git l还要少两个字母

## 19. 快速定位故障版本

示例：

*git bisect*

git bisect 使用分治算法查找出错版本号。

假设休假一周回来，你看了一下最新代码，发现走之前完全正常的代码现在出问题了。

你查看了一下休假之前最后一次提交的代码，功能尚且正常。不幸的是，你离开的这段时间，已经