Git 知识

git 基础操作

git diff

跳过使用暂存区域

尽管使用暂存区域的方式可以精心准备要提交的细节，但有时候这么做略显繁琐。Git 提供了一个跳过使用暂存区域的方式，只要在提交的时候，给 git commit 加上 -a 选项，Git 就会自动把所有已经跟踪过的文件暂存起来一并提交，从而跳过 git add 步骤：

git log

我们常用 -p 选项展开显示每次提交的内容差异，用 -2 则仅显示最近的两次更新： 该选项除了显示基本信息之外，还在附带了每次 commit 的变化。当进行代码审查，或者快速浏览某个搭档提交的 commit 的变化的时候，这个参数就非常有用了。

某些时候，单词层面的对比，比行层面的对比，更加容易观察。Git 提供了 --word-diff 选项。我们可以将其添加到 git log -p 命令的后面，从而获取单词层面上的对比。在程序代码中进行单词层面的对比常常是没什么用的。不过当你需要在书籍、论文这种很大的文本文件上进行对比的时候，这个功能就显出用武之地了。下面是一个简单的例子

该选项除了显示基本信息之外，还在附带了每次 commit 的变化。当进行代码审查，或者快速浏览某个搭档提交的 commit 的变化的时候，这个参数就非常有用了。

某些时候，单词层面的对比，比行层面的对比，更加容易观察。Git 提供了 --word-diff 选项。我们可以将其添加到 git log -p 命令的后面，从而获取单词层面上的对比。在程序代码中进行单词层面的对比常常是没什么用的。不过当你需要在书籍、论文这种很大的文本文件上进行对比的时候，这个功能就显出用武之地了。下面是一个简单的例子

--pretty 选项，可以指定使用完全不同于默认格式的方式展示提交历史。比如用 oneline 将每个提交放在一行显示，这在提交数很大时非常有用。另外还有 short，full 和 fuller 可以用，展示的信息或多或少有些不同，请自己动手实践一下看看效果如何。

但最有意思的是 format，可以定制要显示的记录格式，这样的输出便于后期编程提取分析，像这样：

$ git log --pretty=format:"%h - %an, %ar : %s" ca82a6d - Scott Chacon, 11 months ago : changed the version number 085bb3b - Scott Chacon, 11 months ago : removed unnecessary test code a11bef0 - Scott Chacon, 11 months ago : first commit

选项	说明
%H	提交对象（commit）的完整哈希字串

选项	说明
%h	提交对象的简短哈希字串
%T	树对象（tree）的完整哈希字串
%t	树对象的简短哈希字串
%P	父对象（parent）的完整哈希字串
%p	父对象的简短哈希字串
%an	作者（author）的名字
%ae	作者的电子邮件地址
%ad	作者修订日期（可以用 -date= 选项定制格式）
%ar	作者修订日期，按多久以前的方式显示
%cn	提交者(committer)的名字
%ce	提交者的电子邮件地址
%cd	提交日期
%cr	提交日期，按多久以前的方式显示
%s	提交说明

选项	说明
-p	按补丁格式显示每个更新之间的差异。
--word-diff	按 word diff 格式显示差异。
--stat	显示每次更新的文件修改统计信息。
--shortstat	只显示 --stat 中最后的行数修改添加移除统计。
--name-only	仅在提交信息后显示已修改的文件清单。
--name-status	显示新增、修改、删除的文件清单。
--abbrev-commit	仅显示 SHA-1 的前几个字符，而非所有的 40 个字符。
--relative-date	使用较短的相对时间显示（比如，“2 weeks ago”）。
--graph	显示 ASCII 图形表示的分支合并历史。
--pretty	使用其他格式显示历史提交信息。可用的选项包括 oneline，short，full，fuller 和 format（后跟指定格式）。
--oneline	--pretty=oneline --abbrev-commit 的简化用法。

限制输出长度 除了定制输出格式的选项之外，git log 还有许多非常实用的限制输出长度的选项，也就是只输出部分提交信息。之前我们已经看到过 -2 了，它只显示最近的两条提交，实际上，这是 - 选项的写法，其中的 n 可以是任何自然数，表示仅显示最近的若干条提交。不过实践中我们是不太用这个选项的，Git 在输出所有提交时会自动调用分页程序（less），要看更早的更新只需翻到下页即可。

另外还有按照时间作限制的选项，比如 --since 和 --until。下面的命令列出所有最近两周内的提交：

$ git log --since=2.weeks 你可以给出各种时间格式，比如说具体的某一天（“2008-01-15”），或者是多久以前（“2 years 1 day 3 minutes ago”）。

还可以给出若干搜索条件，列出符合的提交。用 --author 选项显示指定作者的提交，用 --grep 选项搜索提交说明中的关键字。（请注意，如果要得到同时满足这两个选项搜索条件的提交，就必须用 --all-match 选项。否则，满足任意一个条件的提交都会被匹配出来）

另一个真正实用的git log选项是路径(path)，如果只关心某些文件或者目录的历史提交，可以在 git log 选项的最后指定它们的路径。因为是放在最后位置上的选项，所以用两个短划线（--）隔开之前的选项和后面限定的路径名。

表 2-3 还列出了其他常用的类似选项。

选项	说明
`-(n)`	仅显示最近的 n 条提交
`--since, --after`	仅显示指定时间之后的提交。
`--until, --before`	仅显示指定时间之前的提交。
`--author`	仅显示指定作者相关的提交。
`--committer`	仅显示指定提交者相关的提交。

来看一个实际的例子，如果要查看 Git 仓库中，2008 年 10 月期间，Junio Hamano 提交的但未合并的测试脚本（位于项目的 t/ 目录下的文件），可以用下面的查询命令：

- $ git log --pretty="%h - %s" --author=gitster --since="2008-10-01" --before="2008-11-01" --no-merges -- t/
- 5610e3b - Fix testcase failure when extended attribute
- acd3b9e - Enhance hold_lock_file_for_{update,append}()
- f563754 - demonstrate breakage of detached checkout wi
- d1a43f2 - reset --hard/read-tree --reset -u: remove un
- 51a94af - Fix "checkout --track -b newbranch" on detac
- b0ad11e - pull: allow "git pull origin $something:$cur Git 项目有 20,000 多条提交，但我们给出搜索选项后，仅列出了其中满足条件的 6 条。

.4 Git 基础 - 撤消操作

撤消操作

任何时候，你都有可能需要撤消刚才所做的某些操作。接下来，我们会介绍一些基本的撤消操作相关的命令。请注意，有些撤销操作是不可逆的，所以请务必谨慎小心，一旦失误，就有可能丢失部分工作成果。

修改最后一次提交

有时候我们提交完了才发现漏掉了几个文件没有加，或者提交信息写错了。想要撤消刚才的提交操作，可以使用 `--amend` 选项重新提交：

```
$ git commit --amend
```

此命令将使用当前的暂存区域快照提交。如果刚才提交完没有作任何改动，直接运行此命令的话，相当于有机会重新编辑提交说明，但将要提交的文件快照和之前的一样。

启动文本编辑器后，会看到上次提交时的说明，编辑它确认没问题后保存退出，就会使用新的提交说明覆盖刚才失误的提交。

如果刚才提交时忘了暂存某些修改，可以先补上暂存操作，然后再运行 `--amend` 提交：

```
$ git commit -m 'initial commit'
$ git add forgotten_file
$ git commit --amend
```

上面的三条命令最终只是产生一个提交，第二个提交命令修正了第一个的提交内容。

取消已经暂存的文件

接下来的两个小节将演示如何取消暂存区域中的文件，以及如何取消工作目录中已修改的文件。不用担心，查看文件状态的时候就提示了该如何撤消，所以不需要死记硬背。来看下面的例子，有两个修改过的文件，我们想要分开提交，但不小心用 `git add .` 全加到了暂存区域。该如何撤消暂存其中的一个文件呢？其实，`git status` 的命令输出已经告诉了我们该怎么做：

```
$ git add .
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.txt
        modified:   benchmarks.rb
```

就在 “Changes to be committed” 下面，括号中有提示，可以使用 `git reset HEAD <file>...` 的方式取消暂存。好吧，我们来试试取消暂存 benchmarks.rb 文件：

```
$ git reset HEAD benchmarks.rb
Unstaged changes after reset:
M       benchmarks.rb
$ git status
```

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.txt

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:   benchmarks.rb
```

这条命令看起来有些古怪，先别管，能用就行。现在 benchmarks.rb 文件又回到了之前已修改未暂存的状态。

取消对文件的修改

如果觉得刚才对 benchmarks.rb 的修改完全没有必要，该如何取消修改，回到之前的状态（也就是修改之前的版本）呢？`git status` 同样提示了具体的撤消方法，接着上面的例子，现在未暂存区域看起来像这样：

```
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:   benchmarks.rb
```

在第二个括号中，我们看到了抛弃文件修改的命令（至少在 Git 1.6.1 以及更高版本中会这样提示，如果你还在用老版本，我们强烈建议你升级，以获取最佳的用户体验），让我们试试看：

```
$ git checkout -- benchmarks.rb
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.txt
```

可以看到，该文件已经恢复到修改前的版本。你可能已经意识到了，这条命令有些危险，所有对文件的修改都没有了，因为我们刚刚把之前版本的文件复制过来重写了此文件。所以在用这条命令前，请务必确定真的不再需要保留刚才的修改。如果只是想回退版本，同时保留刚才的修改以便将来继续工作，可以用下章介绍的 stashing 和分支来处理，应该会更好些。

记住，任何已经提交到 Git 的都可以被恢复。即便在已经删除的分支中的提交，或者用 `--amend` 重新改写的提交，都可以被恢复（关于数据恢复的内容见第九章）。所以，你可能失去的数据，仅限于没有提交过的，对 Git 来说它们就像从未存在过一样。

prev | next