`git commit` 命令用于将更改记录(提交)到存储库。将索引的当前内容与描述更改的用户和日志消息一起存储在新的提交中。

简介

```
git commit [-a | --interactive | --patch] [-s] [-v] [-u<mode>] [--amend]
       [--dry-run] [(-c | -C | --fixup | --squash) <commit>]
       [-F <file> | -m <msg>] [--reset-author] [--allow-empty]
       [--allow-empty-message] [--no-verify] [-e] [--author=<author>]
       [--date=<date>] [--cleanup=<mode>] [--[no-]status]
       [-i | -o] [-S[<keyid>]] [--] [<file>…]
```

描述

`git commit` 命令将索引的当前内容与描述更改的用户和日志消息一起存储在新的提交中。

要添加的内容可以通过以下几种方式指定：

1. 在使用 `git commit` 命令之前，通过使用 `git add` 对索引进行递增的“添加”更改(注意：修改后的文件的状态必须为“`added`”);
2. 通过使用 `git rm` 从工作树和索引中删除文件，再次使用 `git commit` 命令;
3. 通过将文件作为参数列出到 `git commit` 命令(不使用`--interactive` 或`--patch` 选项)，在这种情况下，提交将忽略索引中分段的更改，而是记录列出的文件的当前内容(必须已知到 Git 的内容)；
4. 通过使用带有`-a` 选项的 `git commit` 命令来自动从所有已知文件(即所有已经在索引中列出的文件)中添加“更改”，并自动从已从工作树中删除索引中的“`rm`”文件，然后执行实际提交;
5. 通过使用`--interactive` 或`--patch` 选项与 `git commit` 命令一起确定除了索引中的内容之外哪些文件或 hunks 应该是提交的一部分，然后才能完成操作。

`--dry-run` 选项可用于通过提供相同的参数集(选项和路径)来获取上一个任何内容包含的下一个提交的摘要。

如果您提交，然后立即发现错误，可以使用 git reset 命令恢复。

示例

以下是一些示例 -

提交已经被 `git add` 进来的改动。

```
$ git add .
$ # 或者~
```

```
$ git add newfile.txt
$ git commit -m "the commit message" #
$ git commit -a # 会先把所有已经 track 的文件的改动`git add`进来，然后提交(有点像 svn 的一次提交,不用先暂存)。对于没有 track 的文件,还是需要执行`git add <file>` 命令。
$ git commit --amend # 增补提交，会使用与当前提交节点相同的父节点进行一次新的提交，旧的提交将会被取消。
```

录制自己的工作时，工作树中修改后的文件的内容将临时存储到使用 `git add` 命名为“索引”的暂存区域。 一个文件只能在索引中恢复，而不是在工作树中，使用 `git reset HEAD – <file>`进行上一次提交的文件，这有效地恢复了 git 的添加，并阻止了对该文件的更改，以参与下一个提交在使用这些命令构建状态之后，`git commit`(没有任何 `pathname` 参数)用于记录到目前为止已经进行了什么更改。 这是命令的最基本形式。一个例子：

```
$ vi hello.c
$ git rm goodbye.c
$ git add hello.c
$ git commit
```

可以在每次更改后暂存文件，而不是在 `git commit` 中关注工作树中跟踪内容的文件的更改，可使用相应的 `git add` 和 `git rm`。 也就是说，如果您的工作树中没有其他更改(*hello.c* 文件内容不变)，则该示例与前面的示例相同：

```
$ vi hello.c
$ rm goodbye.c
$ git commit -a
```

命令 `git commit –a` 首先查看您的工作树，注意您已修改 `hello.c` 并删除了 `goodbye.c`，并执行必要的 `git add` 和 `git rm`。

在更改许多文件之后，可以通过给出 `git commit` 的路径名来更改记录更改的顺序。当给定路径名时，该命令提交只记录对命名路径所做的更改：

```
$ edit hello.c hello.h # 修改了这两个文件的内容
$ git add hello.c hello.h
$ edit Makefile
$ git commit Makefile
```

这提供了一个记录 Makefile 修改的提交。 在 `hello.c` 和 `hello.h` 中升级的更改不会包含在生成的提交中。然而，它们的变化并没有消失 - 他们仍然有更改，只是被阻止。 按照上述顺序执行：

```
$ git commit
```

这个第二个提交将按照预期记录更改为 `hello.c` 和 `hello.h`。

合并后(由 `git merge` 或 `git pull` 发起)由于冲突而停止，干净合并的路径已经被暂存为提交，并且冲突的路径保持在未加载状态。 您必须首先检查哪些路径与 git 状态冲突，并在手工将其固定在工作树中之后，要像往常一样使用 `git add`：

```
$ git status | grep unmerged
unmerged: hello.c
$ edit hello.c
$ git add hello.c
```

解决冲突和暂存结果后，`git ls-files -u` 将停止提及冲突的路径。完成后，运行 `git commit` 最后记录合并：

```
$ git commit
```