git快速入门

光玉

想象一下你正在玩Flappy Bird, 你今晚的目标是拿到100分, 不然就不睡觉. 经过千辛万苦, 你拿到了99分, 就要看到成功的曙光的时候, 你竟然失手了! 你悲痛欲绝, 滴血的心在呼喊着, "为什么上天要这样折磨我? 为什么不让我存档?"

想象一下你正在写代码, 你今晚的目标是实现某一个新功能, 不然就不睡觉. 经过千辛万苦, 你终于把代码写好了, 保存并编译运行, 你看到调试信息一行一行地在终端上输出. 就要看到成功的曙光的时候, 竟然发生了段错误! 你仔细思考, 发现你之前的构思有着致命的错误, 但之前正确运行的代码已经永远离你而去了. 你悲痛欲绝, 滴血的心在呼喊着, "为什么上天要这样折磨我?" 你绝望地倒在屏幕前... 这时, 你发现身边渐渐出现无数的光玉, 把你包围起来, 耀眼的光芒令你无法睁开眼睛... 等到你回过神来, 你发现屏幕上正是那份之前正确运行的代码! 但在你的记忆中, 你确实经历过那悲痛欲绝的时刻... 这一切真是不可思议啊...

人生如戏, 戏如人生

人生就像不能重玩的Flappy Bird, 但软件工程领域却并非如此, 而那不可思议的光玉就是"版本控制系统". 版本控制系统给你的开发流程提供了比朋也收集的更强大的光玉, 能够让你在过去和未来中随意穿梭, 避免上文中的悲剧降临你的身上.

没听说过版本控制系统就完成实验, 艰辛地排除万难, 就像游戏通关之后才知道原来游戏可以存档一样, 其实玩游戏的时候进行存档并不是什么丢人的事情.

在实验中, 我们使用 `git` 进行版本控制. 下面简单介绍如何使用 `git` .

游戏设置

首先你得安装 `git` :

```
apt-get install git
```

安装好之后, 你需要先进行一些配置工作. 在终端里输入以下命令

```
git config --global user.name "Zhang San"        # your name
git config --global user.email "zhangsan@foo.com"    # your email
git config --global core.editor vim            # your favourite editor
git config --global color.ui true
```

经过这些配置, 你就可以开始使用 `git` 了.

在实验中, 你会通过 `git clone` 命令下载我们提供的框架代码, 里面已经包含一些 `git` 记录, 因此不需要额外进行初始化. 如果你想在别的实验/项目中使用 `git` , 你首先需要切换到实验/项目的目录中, 然后输入

```
git init
```

进行初始化.

查看存档信息

使用

```
git log
```

查看目前为止所有的存档.

使用

```
git status
```

可以得知, 与当前存档相比, 哪些文件发生了变化.

存档

你可以像以前一样编写代码. 等到你的开发取得了一些阶段性成果, 你应该马上进行"存档".

首先你需要使用 `git status` 查看是否有新的文件或已修改的文件未被跟踪, 若有, 则使用 `git add` 将文件加入跟踪列表, 例如

```
git add file.c
```

会将 `file.c` 加入跟踪列表. 如果需要一次添加所有未被跟踪的文件, 你可以使用

```
git add -A
```

但这样可能会跟踪了一些不必要的文件, 例如编译产生的 `.o` 文件, 和最后产生的可执行文件. 事实上, 我们只需要跟踪代码源文件即可. 为了让 `git` 在添加跟踪文件之前作筛选, 你可以编辑 `.gitignore` 文件(你可以使用 `ls -a` 命令看到它), 在里面给出需要被 `git` 忽略的文件和文件类型.

把新文件加入跟踪列表后, 使用 `git status` 再次确认. 确认无误后就可以存档了, 使用

```
git commit
```

提交工程当前的状态. 执行这条命令后, 将会弹出文本编辑器, 你需要在第一行中添加本次存档的注释, 例如"fix bug for xxx". 你应该尽可能添加详细的注释, 将来你需要根据这些注释来区别不同的存档. 编写好注释之后, 保存并退出文本编辑器, 存档成功. 你可以使用 `git log` 查看存档记录, 你应该能看到刚才编辑的注释.

读档

如果你遇到了上文提到的让你悲痛欲绝的情况, 现在你可以使用光玉来救你一命了. 首先使用 `git log` 来查看已有的存档, 并决定你需要回到哪个过去. 每一份存档都有一个hash code, 例如 `b87c512d10348fd8f1e32ddea8ec95f87215aaa5` , 你需要通过hash code来告诉 `git` 你希望读哪一个档. 使用以下命令进行读档:

```
git reset --hard b87c
```

其中 `b87c` 是上文hash code的前缀: 你不需要输入整个hash code. 这时你再看看你的代码, 你已经成功地回到了过去!

但事实上, 在使用 `git reset` 的hard模式之前, 你需要再三确认选择的存档是不是你的真正目标. 如果你读入了一个较早的存档, 那么比这个存档新的所有记录都将被删除! 这意为着你不能随便回到"将来"了.

第三视点

当然还是有办法来避免上文提到的副作用的, 这就是 `git` 的分支功能. 使用命令

```
git branch
```

查看所有分支. 其中 `master` 是主分支, 使用 `git init` 初始化之后会自动建立主分支.

读档的时候使用以下命令

```
git checkout b87c
```

而不是 `git reset` . 这时你将处于一个虚构的分支中, 你可以

- 查看 `b87c` 存档的内容
- 使用以下命令切换到其它分支

```
git checkout 分支名
```

- 对代码的内容进行修改, 但你不能使用 `git commit` 进行存档, 你需要使用

```
git checkout -B 分支名
```

把修改结果保存到一个新的分支中, 如果分支已存在, 其内容将会被覆盖

不同的分支之间不会相互干扰, 这也给项目的分布式开发带来了便利. 有了分支功能, 你就可以像第三视点那样在一个世界的不同时间(一个分支的多个存档), 或者是多个平行世界(多个分支)之间来回穿梭了.

更多功能

以上介绍的是 `git` 的一些基本功能, `git` 还提供很多强大的功能, 例如使用 `git diff` 比较同一个文件在不同版本中的区别, 使用 `git bisect` 进行二分搜索来寻找一个bug在哪次提交中被引入...

其它功能的使用请参考 `git help` , `man git` , 或者在网上搜索相关资料.