Git 学习笔记

Piex

UCAS

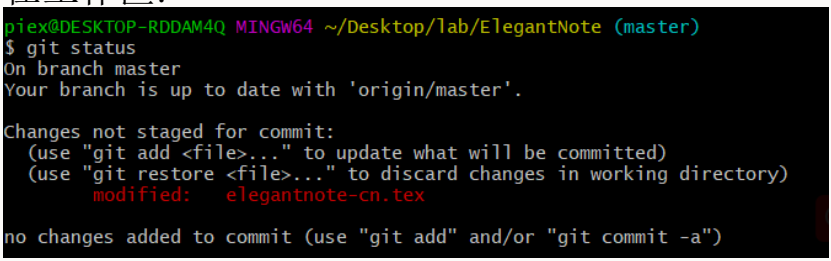
版本：1.0

更新：2021 年 1 月 23 日

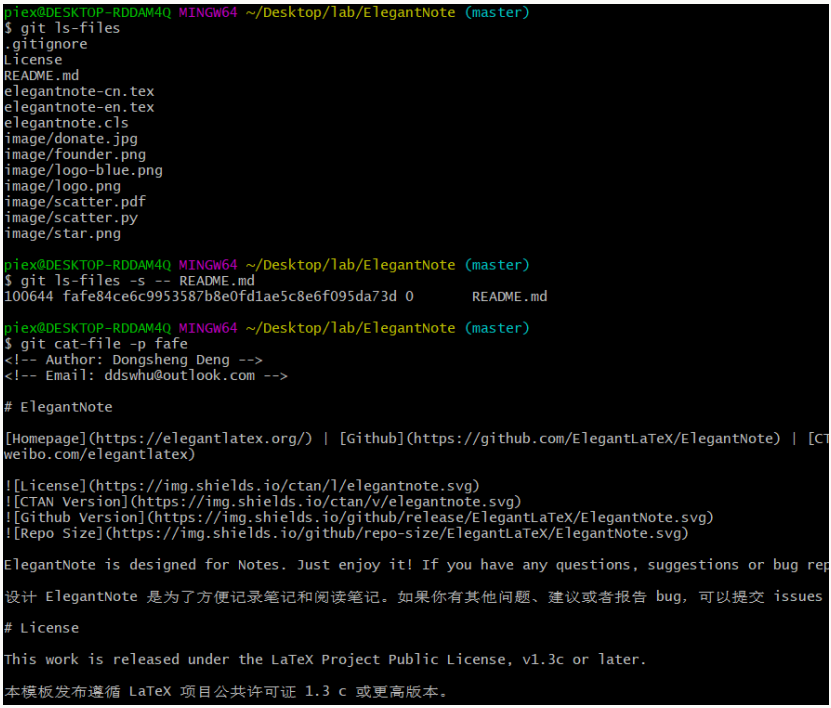
1 分区

1.1 三大分区

- 工作区：直接编辑的区域，对于新增的文件，如果没有 add 加入暂存区，就会以红色 (untracked/modified)的形式放置在工作区.



- 暂存区：数据暂时存放的区域（进入版本库之前），存放在.git/index 目录下，Untracked files 的文件，无法使用 git commit –am 命令将文件添加到本地仓库中。git ls-files 可以查看暂存区文件，要查看文件具体内容，需要使用 git ls-files -s – filename 获得文件对应的 blob 对象，然后使用 git cat-file -p [bolb 的 hash 值前 4 位即可] 查询内容。



- 版本库（本地仓库）：暂存区 commit 的代码会放入版本库中，存放在.git 目录下，push 的时候版本库的数据全部发送到远程仓库中。

1.2 涉及指令

1.2.1 分区转换指令

- git add: 工作区-> 暂存区
- git commit: 暂存区-> 版本库

- git push: 版本库-> 远程仓库

1.2.2 分区对比指令

- git diff: 工作区与暂存区对比

  git diff 输出结果分析：---代表源文件，+++ 代表目标文件（通常工作区的文件被当做目标文件），空格开头的行是源文件和目标文件中都出现的行，-开头的行是只在源文件中出现的，+ 开头的行是只在目标文件中出现的，差异按照差异小结进行总结，每个差异小结的第一行都是定位语句，以 @@ 开始，@@ 结束。如下图所示，@@ -1,7 +1,7 @@ 表示源文件第 1 行开始的 7 行和目标文件第 1 行开始的 7 行构成一个差异小结，差异内容是两行前面添加了% 和一个空格。

```
piex@DESKTOP-RDDAM4Q MINGW64 ~/Desktop/lab/ElegantNote (master)
$ git diff
diff --git a/elegantnote-cn.tex b/elegantnote-cn.tex
index 45e40fd..931060a 100644
--- a/elegantnote-cn.tex
+++ b/elegantnote-cn.tex
@@ -1,7 +1,7 @@
 %!TEX program = xelatex
 \documentclass[cn,hazy,blue,14pt,screen]{elegantnote}
-\definecolor{geyecolor}{RGB}{199,237,204}
-\pagecolor{geyecolor}
+% \definecolor{geyecolor}{RGB}{199,237,204}
+% \pagecolor{geyecolor}
 \title{ElegantNote：一个优美的 \LaTeX{} 笔记模板}

 \author{邓东升}
```

- git diff head: 工作区与版本库对比
- git diff - -cached: 暂存区与版本库对比

2 原理

2.1 git 如何存储文件/目录信息

git init：初始化一个新的 git 项目，会在项目的根目录下创建.git 的隐藏目录，其中.git 目录下的 objects 目录，是存储文件变化的核心。objects 下存放的文件名是哈希的“指纹”，文件内容就是 git 将信息压缩后形成的二进制文件。

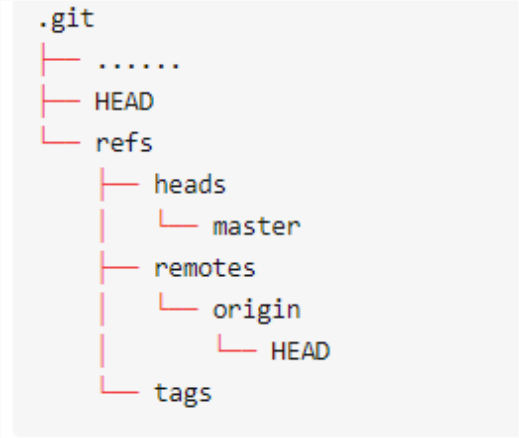
2.2 git object 的类型

三种：blob, tree, commit.

文件存储为 blob 类型，文件夹为 tree 类型，每次提交的节点被存储为 commit 类型。比较复杂，后面可以继续研究.尝试对同一个文件的 2 行增加/减少注释多次并加入暂存区，发现 objects 下只有 2 个 blob 组件，查询后得知内容相同时便不会增加 blob 组件，因此 2 个 blob 组件分别对应注释前和注释后的暂存区文件。

3 git 分支

3.1 git 的目录树



refs 目录就是用来记录当前对分支的引用信息，包括本地分支，远程分支，标签。

heads 记录的是本地所有分支，remotes 和\remotes\origin\HEAD一样，指向对应的某个远程分支。heads\master内容是“commit 节点的 hash 值”，HEAD 内容是“当前在哪个本地分支”，可以用 git branch 来创建其他分支，git chechkout 来切换到其他分支，git branch -vv 来查看分支信息。分支当前的指针指向最近一次 commit 的节点，通过谁创建的分支，就沿用谁的指针。（未被放入版本库的文件会在分支切换时被抛弃，造成严重后果）

3.2 分支的合并

merge 和 rebase 两种

- 相同点：都是从一个分支获取并合并到当前分支。
- 不同点：merge 自动创建一个新的 commit，如果遇到冲突，仅需要修改后重新 commit，每次都记录了详细的 commit，在 commit 频繁的时候会看到分支比较乱；rebase 是找公共的节点，直接合并之前 commit 历史，这样会是的分支发展历史比较简洁，去掉了 merge commit，但是如果合并时出现了问题，没有留下痕迹不好定位。

  git rebase - -abort: 遇到冲突时放弃合并，回到 rebase 之前的状态；

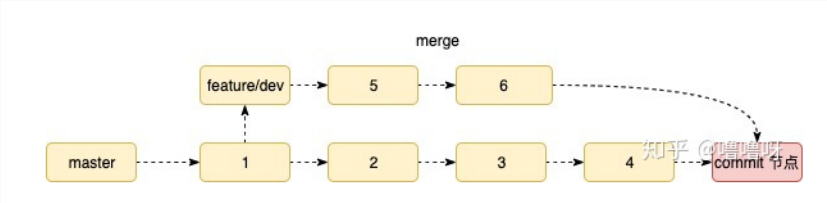
  git rebase - -continue: 合并冲突，结合"git add 文件" 命令一起，一步一步解决冲突；

  git rebase - -skip: 将引起冲突的 commits 丢弃掉.（最好不要在公共分支上使用 rebase）

3.3 分支的冲突

冲突的产生：冲突是从合并的时候产生的，git 分支的合并，其实就是 tree 的合并，在 feature/dev 上执行 git merge master 时，git 会先找到这两个分支（feature/1 & feature/2) 是从哪个指

针创建出来的，称之为"merge base"，然后检查这两次的 tree 是否一致，如果不一致说明一定有文件发生了修改。



对于一个文件来说，有三种可能的情况：

- 文件在节点 6，节点 3 以及 merge base 的 hash 值都相同，说明文件没有被修改过，不会有冲突；
- 文件在节点 6 和 merge base（在节点 3 和 merge base）的 hash 值相同，说明节点 3（节点 6）上的文件发生了修改，直接更新文件的变化即可；
- 文件在节点 6、节点 3 以及 merge bash 上的 hash 值均不相同，冲突就产生了，也就是说节点 6 和节点 3 的文件都发生了修改，不知道哪个是最新的修改。

4 版本的回滚

4.1 revert

执行 git revert 后，将回退到上一个 commit 的版本；

4.2 reset

git reset 分为三种模式：soft，mixed，hard。由于每一次的 commit 都会产生与之对应的 hash 值，所以借助这个进行重置。

- git reset –hard commit.hash: 会重置暂存区和工作区，完全重置为指定的 commit 节点，当前分支没有 commit 的代码会被清除；
- git reset –soft commit.hash: 会保留工作目录，并把指定的 commit 节点与当前分支的差异都存入暂存区，即没有被 commit 的代码也会被保留下来；
- git reset commit.hash: 不带参数即 mixed 模式，会保留工作目录，并把工作区，暂存区以及 reset 的差异都放到工作区，然后清除暂存区。因此执行后只要有所差异，文件都会变为红色，难以区分。

一般情况下，使用 soft 模式，既能保留暂存区，又能 reset 到某个分支。

5 代码暂存

在当前分支工作时不得已需要切换到其他分支处理事情而不想 commit 时（commit 多了会污染 log），可以使用 git stash

将那些数据都暂存到 git 提供的栈中。

- git stash：暂存修改过的代码，保存到 git 栈中，然后将工作区还原成上一次 commit 的内容；
- git stash list：显示之前压栈的所有记录；
- git stash clear：清空 git 栈；
- git stash apply：从 git 栈中读取上一次暂存的那些代码，恢复工作区。