【吐血整理】Git的各种撤销姿势

前提

如果要排一个程序员在git使用中遇到的最多问题的排名，我想**“怎么正确的撤销git操作？”**肯定是名列第一（第二应该是windows如何安装使用git）。就比如，“我提交了一个错误的文件咋办？”，“我push错了分支怎么办？”，“我分支合错了怎么办？”，这种问题在开发中总是能听到，不绝于耳。

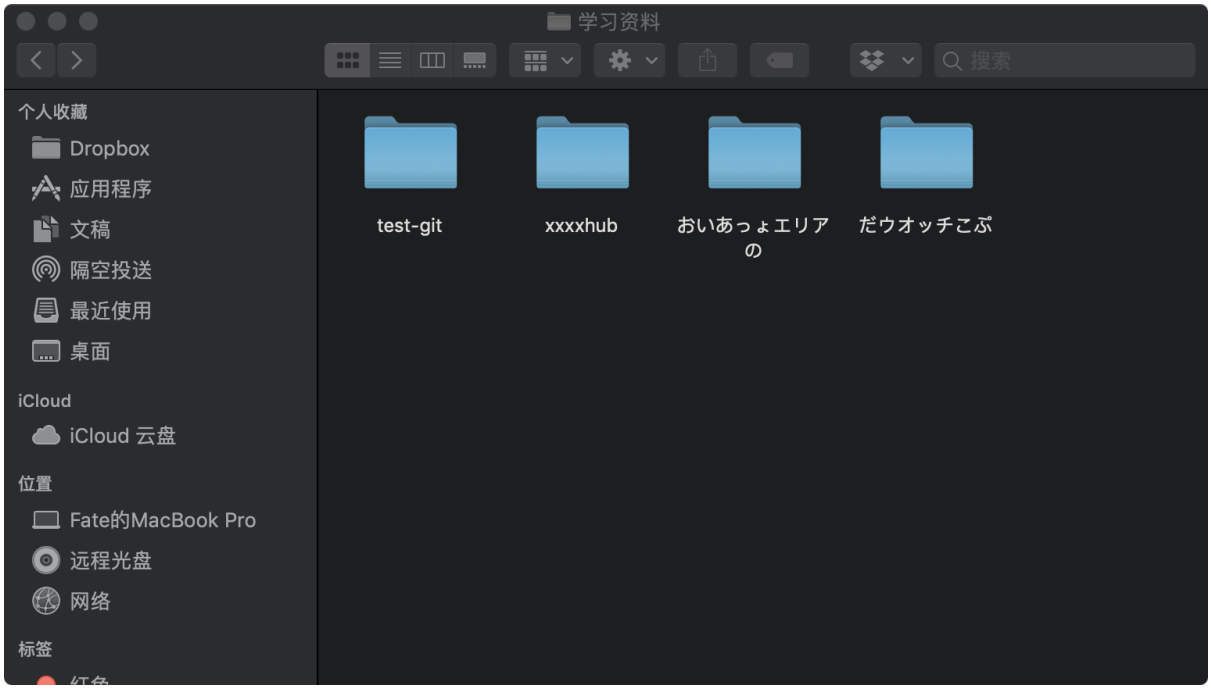
今天，这篇文章，就来好好说说git撤销的各种姿势，解决你的撤销恐惧症。

为了保证文章的生动形象，请允许我在图片中插播文章。

准备工作

我们为了方便演示，在本地的 **“学习资料目录”** 下新建一个git目录 **“test-git”**，并在目录下初始化git仓库。

```
⚡ mkdir test-git
⚡ cd test-git
⚡ git init
```



知识点整理：

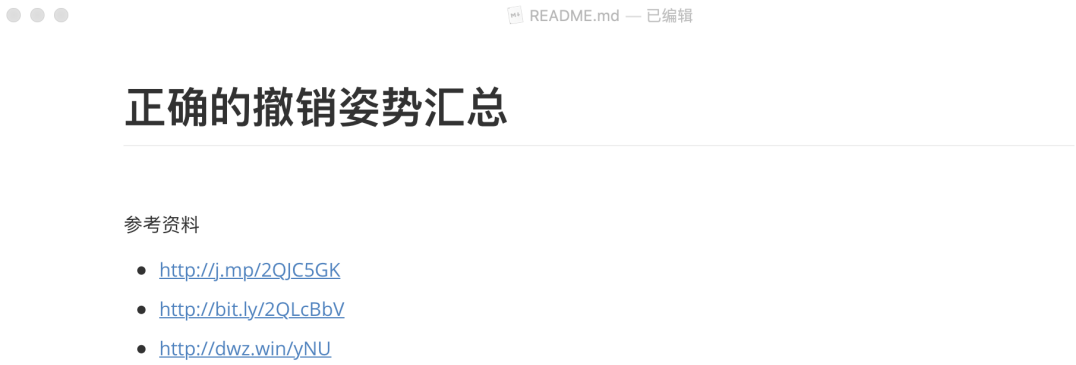
- 工作区

  工作区就是你本地仓库的所在目录，在我们这就是指的 **“test-git”** 这个目录。

- 暂存区

  暂存区就是暂时存放被修改文件的区域，比如 `git add` 就是把要提交的修改放进暂存区。

为了方面接下来的演示，我们先在master分支上创建并提交一个 `README.md` 文件，然后从 master上新建1个分支名为 test1。这里我就省略了具体操作步骤，纯新手请自行搜索下如何实现。



下面，我整理了多个高频率的疑难点，通过一个个场景来解答。

> PS：以下场景简单化了我们实际遇到的情况，请各位观众对号入座，举一反三。

场景1：我在暂存区添加了一个不想提交的文件，怎么办？

小A在test1分支修改了 `README.md` ，过了几天后新加了一个文件 `a.c` 。

```c
#include<stdio.h>
int fun(int n)
{
    if (n <= 1)
        return n;
    else
        return fun(n-1) + fun(n-2);
}

int main()
{
    int n = 10;
    int i;
    for (i = 0; i < n; i++) {
        printf("%d, ", fun(i));
    }
    return 0;
}
```

README.md — 已编辑

正确的撤销姿势汇总

参考资料

- http://j.mp/2QJC5GK
- http://bit.ly/2QLcBbV
- http://dwz.win/yNU

> 以上资料可以学习更多姿势

> 知道上图中算法名称的可以在评论下留言~

小a准备提交代码，只想提交 `a.c` 文件但是忘记了自己一共修改了2个文件，按以往习惯一并使用 `git add .` 添加修改到暂存区，然后一看 `git status` 人傻了发现不对劲，自己多添加了 `README.md` 的修改，这是这次提交不需要的，这可咋办？

别慌，下面命令可以帮到你：

```
⚡ git rm --cached [filename]
```

在我们这个场景中，filename 带入 `README.md` 就ok了，**它的作用就是撤销暂存区的文件修改**。

> PS：以上场景被我简化了可能不明显。实际情况是那些习惯在项目中用 `git add .` 统一添加修改到暂存区的童鞋，他们很容易遇到上面的问题。

场景二：我写错了提交信息，怎么办？

继续上面的场景，小a在完成撤销后打算提交这次修改，然而悲剧他又在填写提交信息的时候手快笔误：

```
⚡ git commit -m "add a.cpp"
```

虽然提交信息很多人不重视，但是小a同学还是很羞愧，把c文件“改成“了c++文件，这可咋办？

别慌，git可以让你反悔：

```
⚡ git commit --amend -m "add a.c"
```

这个命令的原理是产生一个新的提交对象，替换掉上一次提交产生的提交对象。

这时如果暂存区有发生变化的文件，会一起提交到仓库。所以，`--amend` 不仅可以修改提交信息，还可以整个把上一次提交替换掉。

场景三：我的文件改乱了，想要还原文件到上一次提交，怎么办？

成功提交后，不知道大家还记得不，此时我们的仓库中还有一个没处理的修改 `README.md` ：

```
⚡ git status
On branch test1
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:   README.md
```

因为已经过了一段时间，小a甚至都忘了自己当初为什么要去改 `README.md` ，觉得不说破更神秘，所以小a想要撤销这个文件的修改，这可咋办？

别慌，工作区的文件修改也能撤销：

```
$ git checkout -- README.md
```

它的原理是先找暂存区，如果该文件有暂存的版本，则恢复该版本，否则恢复上一次提交的版本。当然如果工作区的文件修改被撤销了，也就没法找回了。

> 我们可以看到checkout除了切换分支的作用还能撤销文件修改，同一个命令会让人困惑。针对切换分支，用 switch 命令更加符合语义。

场景四：我提交了代码，想要还原到以前的一个版本，怎么办？

让我们回顾下现在 git 仓库的提交历史：总共执行了2次提交。

```
⚡ git log --oneline
200e6d2 (HEAD -> test1) add a.c
9d7fb60 (master) Initial Commit
```

小a对最近的一次提交后悔了，原因是他做错了老师出的算法题，于是他决定先回退到最初的版本，重新写一个 `a.c` ，这可咋办？

别慌，我给出2种回退版本的方案，先讲第一种：

```
$ git revert HEAD
```

这个命令的原理是新增一次提交，抵消掉上一次提交的修改，所以不会丢失历史记录，只不过是新增一次历史。

如果想要撤销多次提交，需要依次写出：

```
$ git revert [上一次] [上上次] ......
```

再来讲下第二种：

```
$ git reset 9d7fb60
```

这个命令的原理是，让最新提交的指针重置到指定的提交上，所有指定提交之后的提交都不可见。

需要提到的是，reset 不会修改工作区的文件（会修改暂存区），如果想要同时修改工作区文件：

```
$ git reset --hard [banch]
```

小a成功这一下用了第二种方法回退了版本，重新写了 `a.c`，然后将 test1 分支合并到了 master，交给了老师：

```c
#include <stdio.h>
#include <string.h>
void Next(char*T,int *next){
    int i=1;
    next[1]=0;
    int j=0;
    while (i<strlen(T)) {
        if (j==0||T[i-1]==T[j-1]) {
            i++;
            j++;
            next[i]=j;
        }else{
            j=next[j];
        }
    }
}
int guessme(char * S,char * T){
    int next[10];
    Next(T,next);
    int i=1;
    int j=1;
    while (i<=strlen(S)&&j<=strlen(T)) {
        if (j==0 || S[i-1]==T[j-1]) {
            i++;
            j++;
        }
        else{
            j=next[j];
        }
    }
    if (j>strlen(T)) {
        return i-(int)strlen(T);
    }
    return -1;
}

int main() {
    int i=guessme("ababcabcacbab","abcac");
    printf("%d",i);
    return 0;
}
```

> 猜猜 guessme 是什么算法?

场景五：我合并错了分支，想要取消合并的操作，怎么办

由于老师布置的算法题太多了，小a邀请了室友小b帮他分担一些算法题，小b为了与小a区分，从master上拉了最新的内容，并自己新建了 test2 分支。聪明且“热心“的小b同学注意到小a写的 guessme 算法，觉得有更好的实现，于是他也修改了 a.c 文件。

```c
#include <stdio.h>
#include <string.h>
void Next(char*T,int *next){
    int i=1;
    next[1]=0;
    int j=0;
    while (i<strlen(T)) {
        if (j==0||T[i-1]==T[j-1]) {
            i++;
            j++;
            if (T[i-1]≠T[j-1]) {
               next[i]=j;
            }
            else{
                next[i]=next[j];
            }
        }else{
            j=next[j];
        }
    }
}
int guessme(char * S,char * T){
    int next[10];
    Next(T,next);
    int i=1;
    int j=1;
    while (i⩽strlen(S)&&j⩽strlen(T)) {
        if (j==0 || S[i-1]==T[j-1]) {
            i++;
            j++;
        }
        else{
            j=next[j];
        }
    }
    if (j>strlen(T)) {
        return i-(int)strlen(T);
    }
    return -1;
}

int main() {
    int i=guessme("ababcabcacbab","abcac");
    printf("%d",i);
    return 0;
}
```

修改完后没有告诉小a，自己准备合并到master的时候，正好被小a看到，小a感觉基友之情被伤害了。小b为了挽回，道歉并告诉小a：“我马上把合并的内容还原回去“，然后他快速的敲下了下面一行命令：

```
git merge --abort
```

大团圆结局，两人回归于好，并一起探讨了算法的实现，得到了老师的表扬。

> 该命令最多的场景在于当合并代码冲突后无从下手的时候可以先使用，避免代码混乱。

总结

1. 养成良好的git习惯，上面我也有提到，例如 `git add .` 谨慎使用，多使用 `git status` 查看当前工作区和暂存区的状态。
2. 使用idea内置的git工具，或者其他第三方的app，可以有效的防止命令行事故，也提高了开发效率。**但是，不能只当工具人，当你熟练了后更要看核心原理，会有一种豁然开朗的感觉。**
3. 根据大数据分析，评论+点赞+关注的童鞋买车票不排队，拿压岁钱拿的手软。

> 对本文有疑问或质疑，以及小彩蛋的看法可以在评论区留言~~