

Git:分支

约瑟夫·哈利特

2023 年 1 月 11 日



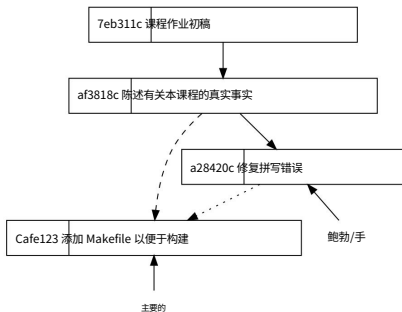
这是关于什么的？

上次我们讨论了如何使用Git从远程获取更改并将其与我们自己的代码合并。

这次我们来谈谈

分支并展示一些使用它们的技巧！

上次…



我们遇到过这样的情况,爱丽丝和鲍勃的树分开了…… ▶

……但他们有共同的历史 ▶

……我们可以将它们重新组合在一起

但为什么我们需要将其限制在其他人的树上呢?

分枝

计划

让我们的目标是保持主分支清洁 ► 主分支始终工作

当我们做一些工作时,我们从 main 中取出一个分支

► (或者可能是其他一些合理的地方) ► 做工作……

► 完成后合并回来。

让我们尝试一下吧!

\$ git 分支 新功能 主要

\$ git 状态
在分支主干上没有
什么可提交的,工作树干净

\$ git checkout 新功能
切换到“新功能”分支

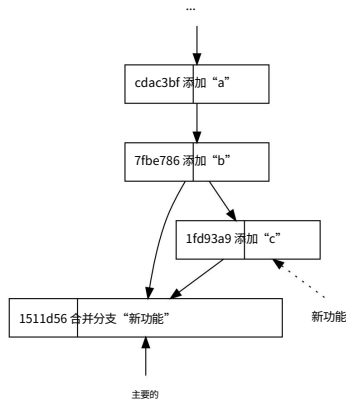
\$ 触摸c; git 添加 c; git commit -m 添加 c [新功能 1fd93a9] 添加
c 1 个文件已更改,0 次插入(+),0 次删除(-) 创建
模式 100644 c

\$ git checkout 主要
切换到分支“主”

\$ git merge --no-ff 新功能提示:正在等待
编辑器关闭文件...

通过“ort”策略进行合并。c | 0 1 个文件已更改,0 个插
入(+),0
个删除(-) 创建模式 100644 c

\$ gitbranch -d new-feature 删除了
分支 new-feature (原为 1fd93a9)。



你到底为什么要这么做？

当您同时处理多个功能时,它开始派上用场。

假设您正在开发新功能时,您的朋友需要您紧急开发另一个功能 ,

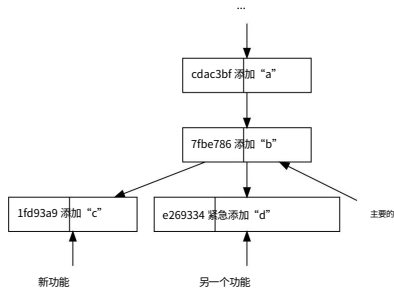
.....

► 不过,您还不想合并新功能,因为您仍在研究它。 ► 您不想在另一个功能的工作中为新功能添加不相关的代码

\$ git 分支 another-feature 7f867be

\$ git checkout 另一个功能
切换到分支 “另一个功能”

\$ 触摸d; git 添加 d; git commit -m 紧急添加 d [another-feature e269334] 紧急添加 d 1 个文件已更改,0 次插入(+),0 次删除(-) 创建模式 100644
d



将它们全部合并!

\$ git checkout 主要
切换到分支 “主”

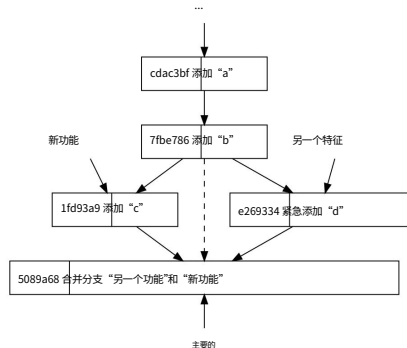
\$ git merge --no-ff another-feature new-feature main 快进到:
another-feature 尝试与 new-feature 提示进行
简单合并:正在等待编辑器关闭文件...

通过“章鱼”策略进行合并。c | 0 天 | 0 2 个文件已更改,0 次插

入(+),0

次删除

(-) 创建模式 100644 c 创建模式 100644 d



通常您不会打扰 no-ff,虚线就会消失! ►有时您喜欢额外的信息 (尤其是

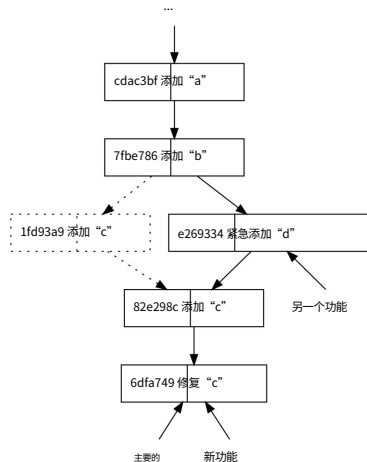
好吧,除了.....

通常你不会这样做►如果合并另一个功

能破坏了新功能中的某些内容怎么办? ►在合并之前先测

试一下就好了! ►但是new-feature

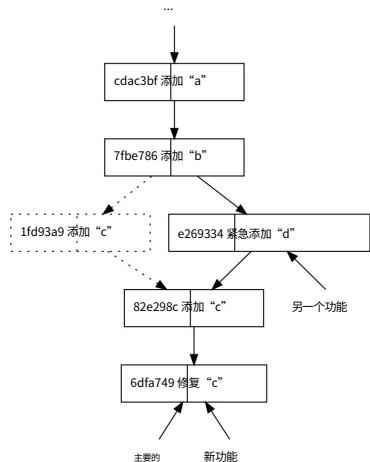
现在还没有将作品合并到main中!



我们可以将 main 合并到 new-feature 我们可以尝试将 main 合并到 new-feature... ► 测试并看看是否需要任何额外的更改...

► 根据需要添加额外的提交 ► 然后将新功能合并回主功能

还是有点混乱,因为我们将至少进行一次合并 ► (假设我们不禁用快进)



相反,我们可以重新设置基准

我通常会做的是将新功能重新设置为主要功能

- 本质上重写历史记录,因此看起来新功能是在合并另一个功能之后完成的
- 作为原始添加“C”提交的一部分修复所有冲突
- (可能更改其

ID) ► 然后重新测试并修复问题并提交

► 然后合并回main。

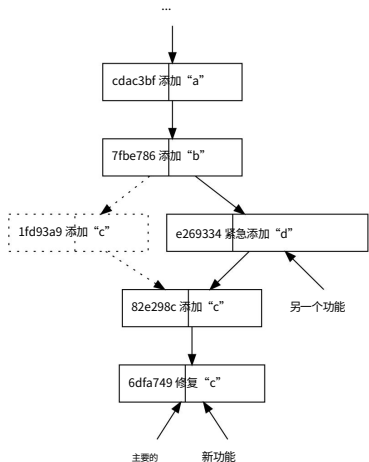
\$ git rebase main new-feature 成功地

重新建立基础并更新了 refs/heads/new-feature。

\$ git checkout 新功能

已经开启“新功能”

(我总是以错误的方式得到变基命令) ► (说真的,我花了3次尝试.....) ► (变基之前一定要做好备份)



我们可以通过 rebase 做更多事情！

假设我们要将新功能作为一系列补丁发送以由项目维护者合并

► `git format-patch` 将为每次提交生成一个补丁 ► 这对于维护者来说应用起来很麻烦 ►

这意味着我们在修复 C 之前提交了整个事情都被破坏的情况。

Rebase 让我们可以编辑存储库历史记录!

因此,一旦我们完成修复,我们就可以交互式地重新设置分支并决定要做什么

```
$ git rebase -i main new-feature [detached HEAD
```

7d2180a] 添加 c,并将其修复为与 d 一起使用 日期:2022 年 11 月 25 日星期五 14:31:08 +0000 1 个文件已更改,0 个插入(+),0 个删除(-) 创建模式 100644 c [分

离 HEAD 5af45b8] 添加 c,并将其修复为与 d 一起使用 日期:2022 年 11 月 25 日星期五 14:31:08 +0000 1 个文

件已更改,1 个插入(+) 创建模式 100644 c 成功重新设置基础并更新了 refs/heads/new-feature。

这在软件工程师中被认为是一种专业礼貌►也有利于隐藏所有那些我弄坏了的东西

提交

►在发送给客户之前删除脏话

```
1 reword 82e298c Adds c
2 squash 6dfa749 Fixes 'c'
3
4 # Rebase e269334..6dfa749 onto e269334 (2 commands)
5 #
6 Commands:
7 # p, pick <commit> = use commit
8 # r, reword <commit> = use commit, but edit the commit message
9 # e, edit <commit> = use commit, but stop for amending
10 # s, squash <commit> = use commit, but meld into previous commit
11 # f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
12 #      commit's log message, unless -C is used, in which case
13 #      keep only this commit's message; -c is same as -C but
14 #      opens the editor
15 # x, exec <command> = run command (the rest of the line) using shell
16 # b, break = stop here (continue rebase later with 'git rebase --continue')
17 # d, drop <commit> = remove commit
18 # l, label <label> = label current HEAD with a name
19 # t, reset <label> = reset HEAD to a label
20 # m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
21 #      create a merge commit using the original merge commit's
22 #      message (or the oneline, if no original merge commit was
23 #      specified); use -c <commit> to reword the commit message
24 # u, update-ref <ref> = track a placeholder for the <ref> to be updated
25 #      to this position in the new commits. The <ref> is
26 #      updated at the end of the rebase
27 #
28 # These lines can be re-ordered; they are executed from top to bottom.
29 #
30 # If you remove a line here THAT COMMIT WILL BE LOST.
31 #
32 # However, if you remove everything, the rebase will be aborted.
33 #
```

```
.git/rebase-merge/git-rebase-todo [+ 1,7 ALL
-- INSERT --
```

警告!

过于聪明地使用 rebase 会破坏你的仓库

有时以无法修复的方式 ► 在聪明之前一

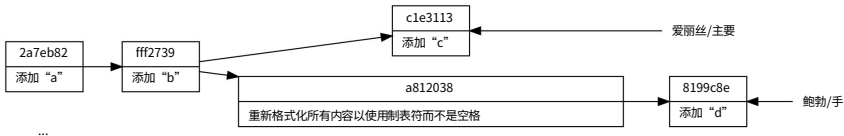
定要备份 ► rebase 被认为是高级Git

最后一招……

假设 Bob 在他们的主分支上做了一些有趣的工作,也做了一些不太有趣的工作。▶他们修复了一些错误,

▶但他们也将您的所有文件

从使用空格切换为制表符 您如何挑选您想要的内容并忽略您不需要的内容?



git 樱桃挑选

```
$ git cherry-pick 8199c8e [main
```

```
031d8d6] 添加 "d"
```

日期:2022 年 11 月 28 日星期一 09:10:43 +0000 1 个文件

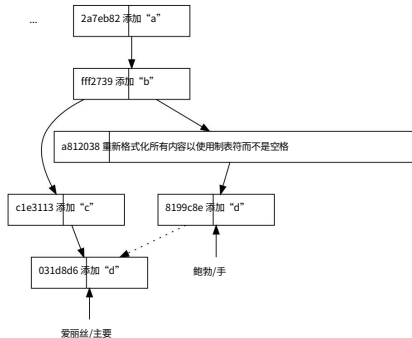
已更改,0 个插入(+),0 个删除(-) 创建模式 100644 d

你为什么要这样做? ► Git 通常会

智能地了解如何复制工作►如果需要更多提交,它也会拉取它们►如果

稍后合并,Git 会智能地了解事物的来源,并

且不会导致冲突



在内部,这只是一个 rebase...►但是 Git 隐藏

了很多复杂性►...实际上 Git 中的所有内容实际上都只是一

个rebase;-)

包起来

没错,那就是 Git!

▶ 您可以用它做无限多的事情 ▶ ...但希望这是您通常会做的 90%

黄金法则

▶ 不要破坏构建

▶ 带着恐惧重新设定基准 (但有时您确实必须这样做) ▶

编写有用的日志消息

(我添加了一些我在幻灯片中没有提到的内容.....阅读手册页很有趣！)

