开源项目阅读与管理

五、开源项目阅读与管理-分支管理

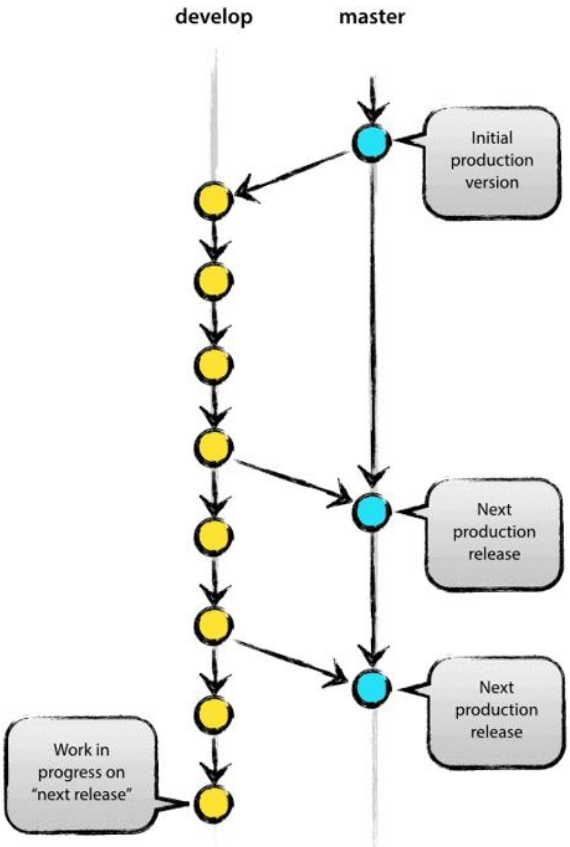
开源项目阅读与管理

步骤 1 : 创建与合并分支
步骤 2 : 解决冲突
步骤 3 : 分支管理策略

刚刚进入公司从事开发工作的时候，很多小伙伴很迷茫，一个项目组五六个人一起写一个项目，大家的写的代码是怎么拼到一起去的？拿 U 盘拷吗？等你正式进入项目以后，你就会知道有个东西叫 SVN，还有个东西叫 GIT。所以说刚毕业的同学一定要优先进入专业的大公司，就像年轻时候应该去大城市闯两年一样，眼界以及你遇到的牛人会大大加快你以后成功的进程。

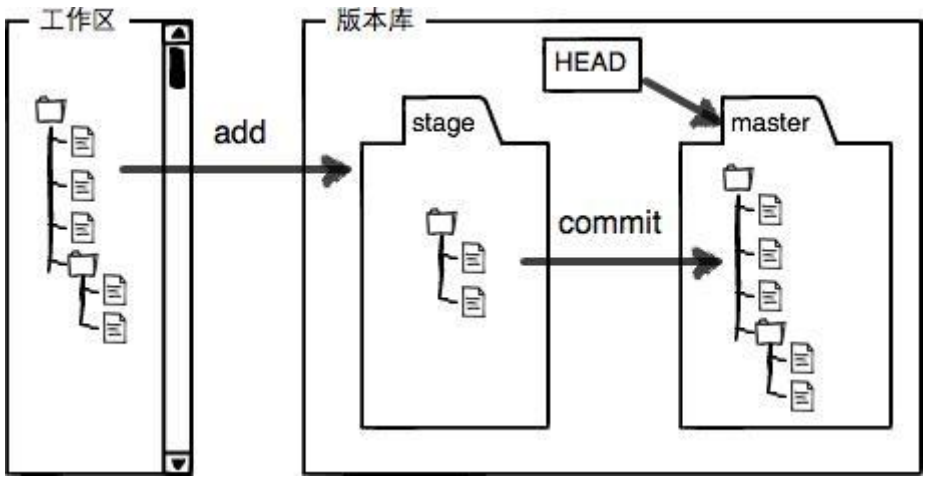
分支在实际中有什么用呢？假设你准备开发一个新功能，但是需要两周才能完成，第一周你写了 50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。

现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作。其他版本控制系统如 SVN 等都有分支管理，但是创建和切换分支慢而繁杂，但 Git 的分支管理无论创建、切换和删除分支，都能很快就能完成！

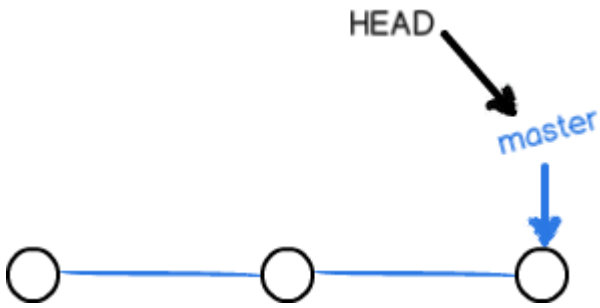


步骤 1：创建与合并分支

回顾一下我们的 Git 版本管理的结构，Git 为我们自动创建的第一个分支 master，以及指向 master 的一个指针叫 HEAD。



1、Git 单分支的结构如下图，master 是指向最新提交的指针，HEAD 是指向 master 的指针，每做一次提交，指针就向前移动一步：



运行查看分支的命令：$ git branch ，带星号的是当前所在分支 master

2、现在增加一个 dev 分支并切换到这个分支：

$ git checkout -b dev

也可以写成两步：（1）创建新分支 $ git branch dev （2）切换到目标分支 $ git checkout dev

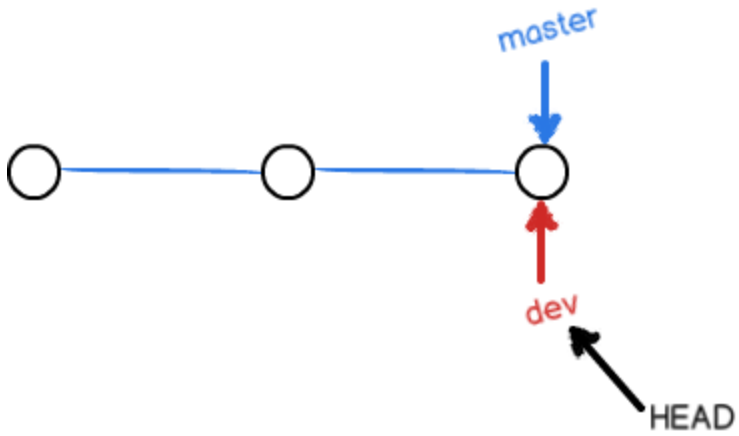
再次运行查看分支的命令：$ git branch ，带星号的变成了当前所在分支 dev

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git branch dev

李响1@LiXiang MINGW64 /d/myrepo (master)
$ git checkout dev
Switched to branch 'dev'

李响1@LiXiang MINGW64 /d/myrepo (dev)
$ git branch
* dev
  master
```

之后变成如下图这样：



从现在开始，对工作区的修改和提交就是针对 dev 分支了，我们把 readme.docx 做一些修改：

2022 年 6 月 1 日星期三
多云，今天是六一儿童节，又是开心的一天呢
2022 年 6 月 2 日星期四
中雨，今天是农历五月初四，明天就是端午节了。
2022 年 6 月 3 日星期五
中雨，今天是农历五月初五，是中国传统节日：端午节，这一天我们要吃粽子，赛龙舟。
2022 年 6 月 7 日星期二
晴，今天是高考第一天，上午考语文，下午考数学。今天天气不错，心情也很好。
2022 年 6 月 10 日星期五
多云转小于，今天学习了分支管理，创建了一个 dev 分支。

再执行命令 git add 然后 git commit ：

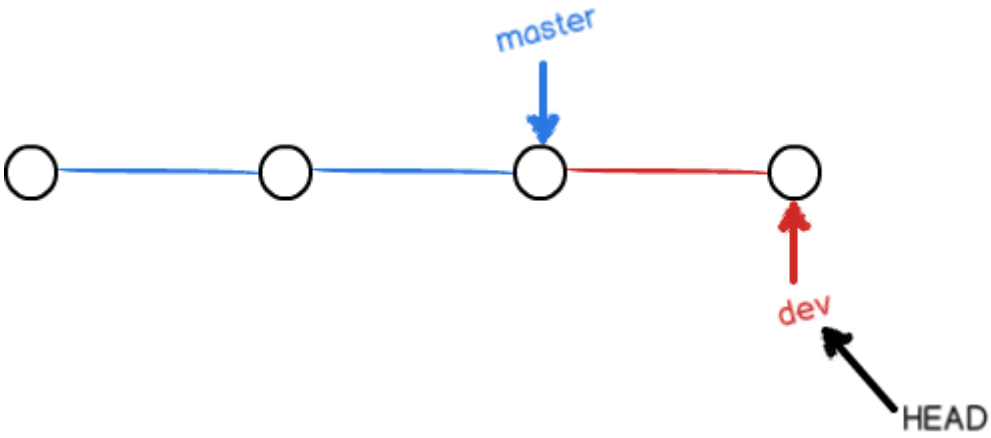
$ git add readme.docx

$ git commit -m "add dev branch"

```
李响1@LiXiang MINGW64 /d/myrepo (dev)
$ git add readme.docx
```

```
李响1@LiXiang MINGW64 /d/myrepo (dev)
$ git commit -m "add dev branch"
[dev 829caf2] add dev branch
 1 file changed, 0 insertions(+), 0 deletions(-)
```

这时，dev 指针往前移动一步，而 master 指针不变，这时结构变成了这样：

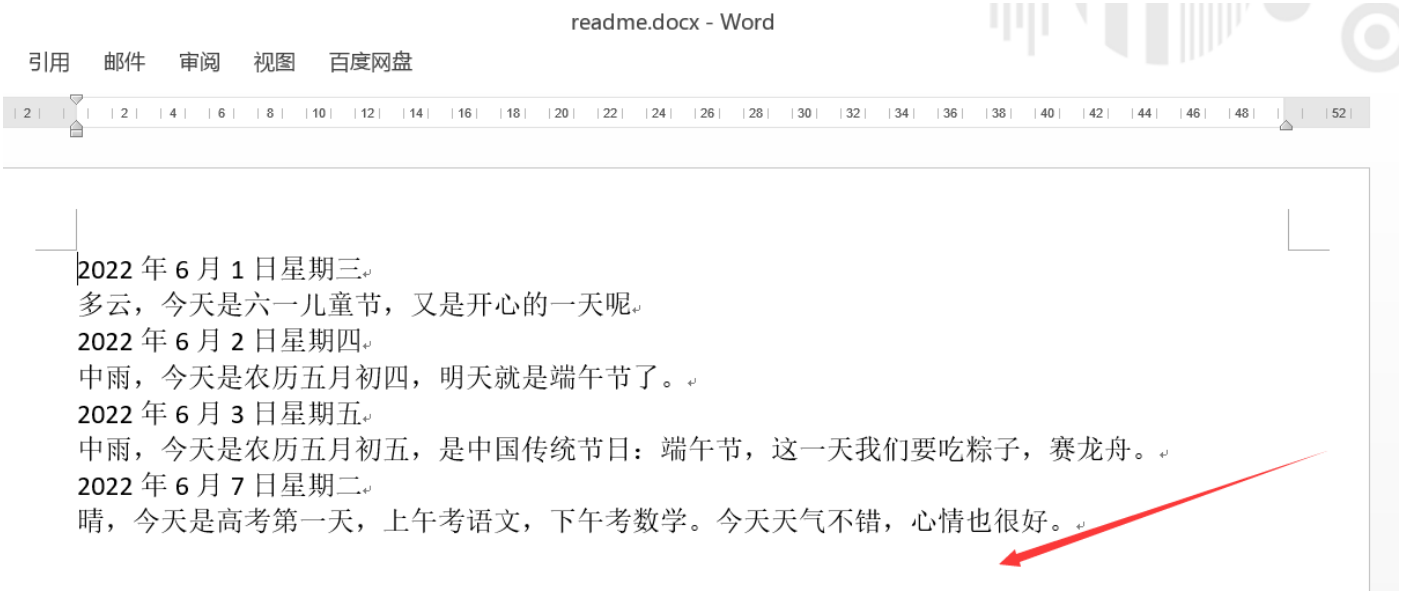


现在，dev 分支的工作完成，当我们执行命令：$ git checkout master 切换回 master 分支后

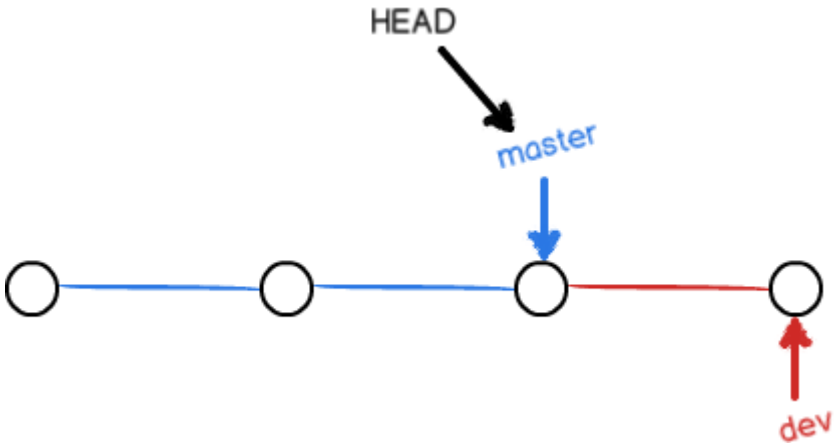
```
李响1@LiXiang MINGW64 /d/myrepo (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

李响1@LiXiang MINGW64 /d/myrepo (master)
$
```

查看 readme.docx 文件，发现改动不见了：



刚才所做的改动不见了，是因为改动在 dev 分支上。这时结构变成了下面这样：



现在，我们把 dev 分支的工作成果合并到 master 分支上：

$ git merge dev

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git merge dev
Updating c06dd0d..829caf2
Fast-forward
 readme.docx | Bin 13812 -> 13921 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)
```

git merge 命令用于合并指定分支到当前分支。合并后，再查看 readme.docx 的内容，就可以看到，和 dev 分支的最新提交是完全一样的。

注意到上面的 Fast-forward 信息，Git 告诉我们，这次合并是“快进模式”，也就是直接把 master 指向 dev 的当前提交，所以合并速度非常快。当然，也不是每次合并都能 Fast-forward，我们后面会讲其他方式的合并。

合并完成后，就可以放心地删除 dev 分支了：

$ git branch -d dev

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git branch -d dev
Deleted branch dev (was 829caf2).
```

删除后，查看 branch，就只剩下 master 分支了：

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git branch
* master
```

因为创建、合并和删除分支非常快，所以 Git 鼓励你使用分支完成某个任务，合并后再删掉分支，这和直接在 master 分支上工作效果是一样的，但过程更安全。

新版本的 Git 提供了新的 git switch 命令来切换分支：创建并切换到新的 dev 分支，可以使用：$ git switch -c dev ，直接切换到已有的 master 分支，可以使用：$ git switch master

总结：

- Git 鼓励大量使用分支：
- 查看分支：git branch
- 创建分支：git branch <name>
- 切换分支：git checkout <name>或者 git switch <name>
- 创建+切换分支：git checkout -b <name>或者 git switch -c <name>
- 合并某分支到当前分支：git merge <name>
- 删除分支：git branch -d <name>

步骤 2：解决冲突

当两个分支上对同一个文件有修改并分别有提交，最后 Git 无法自动合并，就会产生冲突。

1、准备新的 feature1 分支，继续我们的新分支开发：$ git switch -c feature1

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git switch -c feature1
Switched to a new branch 'feature1'
```

修改 readme.docx 最后一行，改为：

2022 年 6 月 10 日星期五
多云转小雨，今天学习了分支管理，创建了一个 dev 分支。使用 Git 创建分支简单又快速。

在 feature1 分支上提交：

$ git add readme.docx

$ git commit -m "add simple and quick"

```
李响1@LiXiang MINGW64 /d/myrepo (feature1)
$ git add readme.docx

李响1@LiXiang MINGW64 /d/myrepo (feature1)
$ git commit -m "add simple and quick"
[feature1 194e84d] add simple and quick
 1 file changed, 0 insertions(+), 0 deletions(-)
```

再切换到 master 分支：$ git switch master

```
李响1@LiXiang MINGW64 /d/myrepo (feature1)
$ git switch master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
```

Git 还会自动提示我们当前 master 分支比远程的 master 分支要超前 1 个提交。

在 master 分支上把 readme.txt 文件的最后一行改为：

2022 年 6 月 10 日星期五
多云转小于，今天学习了分支管理，创建了一个 dev 分支。使用 Git 创建分支简单又便捷。

添加并提交：

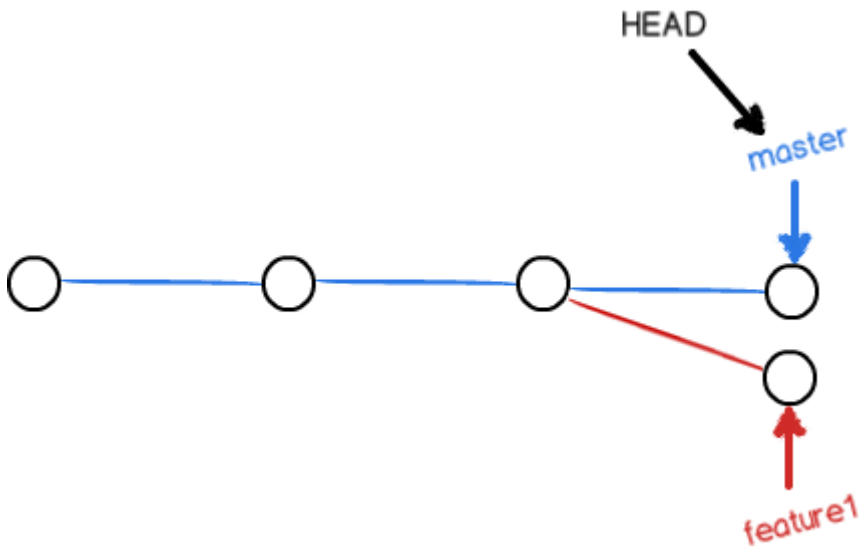
$ git add readme.docx

$ git commit -m " add simple and facile "

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git add readme.docx

李响1@LiXiang MINGW64 /d/myrepo (master)
$ git commit -m "add simple and facile"
[master 85d1948] add simple and facile
 1 file changed, 0 insertions(+), 0 deletions(-)
```

现在，master 分支和 feature1 分支各自都分别有新的提交，分支结构变成了这样：



这种情况下，Git 无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突，我们执行命令：$ git merge feature1

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git merge feature1
warning: Cannot merge binary files: readme.docx (HEAD vs. feature1)
Auto-merging readme.docx
CONFLICT (content): Merge conflict in readme.docx
Automatic merge failed; fix conflicts and then commit the result.
```

Automatic merge failed; fix conflicts and then commit the result，合并文件冲突了！Git 告诉我们，readme.txt 文件存在冲突，必须手动解决冲突后再提交。

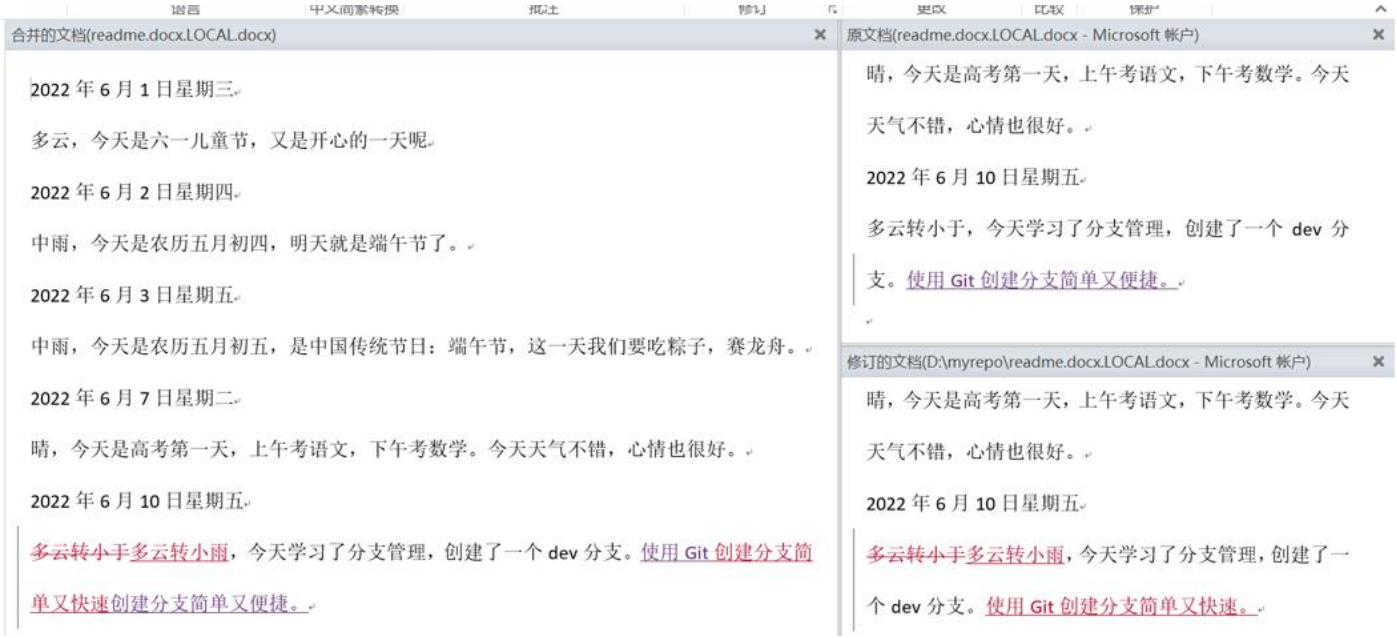
使用命令：$ git status

```
李响1@LiXiang MINGW64 /d/myrepo (master|MERGING)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   readme.docx
```

我们也可以打开 TortoiseGit 图形工具对比几个分支版本的区别。



我们把 readme.docx 最后一行修改为：

2022 年 6 月 10 日星期五。
多云转小雨，今天学习了分支管理，创建了一个 dev 分支。使用 Git 创建分支简单又快速。

修复冲突以后再次添加并提交：

$ git add readme.docx

$ git commit -m "conflict fixed"

```
李响1@LiXiang MINGW64 /d/myrepo (master|MERGING)
$ git add readme.docx

李响1@LiXiang MINGW64 /d/myrepo (master|MERGING)
$ git commit -m "conflict fixed"
[master 442ebd7] conflict fixed
```

再次使用命令：$ git status

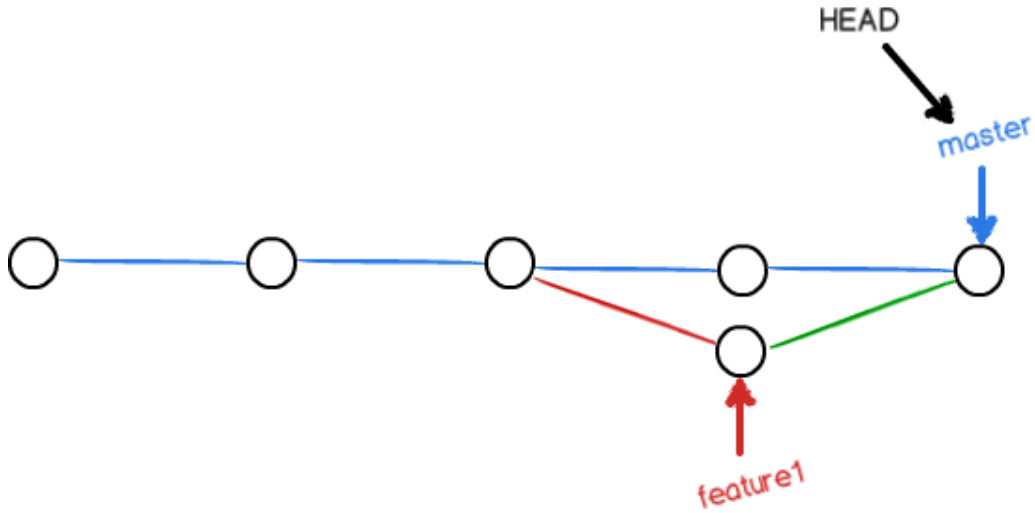
```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 4 commits.
  (use "git push" to publish your local commits)
```

我们执行命令合并分支：$ git merge feature1

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git merge feature1
Already up to date.
```

提示：Already up to date 已经更新

现在，master 分支和 feature1 分支结构如下图所示：



用带参数的 git log 也可以看到分支的合并情况：

$ git log --graph --pretty=oneline --abbrev-commit

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git log --graph --pretty=oneline --abbrev-commit
*   442ebd7 (HEAD -> master) conflict fixed
|\
| * 194e84d (feature1) add simple and quick
* | 85d1948 add simple and facile
|/
* 829caf2 add dev branch
* c06dd0d (origin/master) add test.txt
* dbf9ce6 remove test.txt
* e1cbd59 add test.txt
* fb31c6e add words in June 7th second
* dec163e add words in June 7th
* 4818645 understand how stage works
* 22a8b87 modify the word
* eb85296 The Dragon Boat Festival
* 4e29439 add something
* e2b2795 create a readme word file
```

最后，删除 feature1 分支：

$ git branch -d feature1

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git branch -d feature1
Deleted branch feature1 (was 194e84d).
```

■ 总结：

当 Git 无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

解决冲突就是把 Git 合并失败的文件手动编辑为我们希望的内容，再提交。

用 git log --graph 命令可以看到分支合并图。

步骤 3：分支管理策略

默认情况下，如果情况允许，Git 会自动用快进模式合并分支，但这样合并后不会留下分支存在过的痕迹。删除分支后就会丢失相应信息。

如果要强制禁用 Fast forward 模式，Git 就会在 merge 时生成一个新的 commit 提交，这样，从分支历史上就可以看出分支信息。

下面我们进行实验：--no-ff 方式的 git merge：

首先，仍然创建并切换 dev 分支：$ git switch -c dev

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git switch -c dev
Switched to a new branch 'dev'

李响1@LiXiang MINGW64 /d/myrepo (dev)
$
```

修改 readme.docx 文件：

2022 年 6 月 10 日星期五
多云转小雨，今天学习了分支管理，创建了一个 dev 分支，使用 Git 创建分支简单又快速。

添加 add 并提交一个新的 commit：

$ git add readme.docx

$ git commit -m "modify symbol"

```
李响1@LiXiang MINGW64 /d/myrepo (dev)
$ git add readme.docx
```

```
李响1@LiXiang MINGW64 /d/myrepo (dev)
$ git commit -m "modify symbol"
[dev 05ced57] modify symbol
 1 file changed, 0 insertions(+), 0 deletions(-)
```

现在，我们切换回 master：$ git switch master

```
李响1@LiXiang MINGW64 /d/myrepo (dev)
$ git switch master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 4 commits.
  (use "git push" to publish your local commits)

李响1@LiXiang MINGW64 /d/myrepo (master)
$
```

准备合并 dev 分支，请注意--no-ff 参数，表示禁用 Fast forward：

$ git merge --no-ff -m "merge with no-ff" dev

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git merge --no-ff -m "merge with no-ff" dev
Merge made by the 'ort' strategy.
 readme.docx | Bin 14026 -> 14056 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)
```

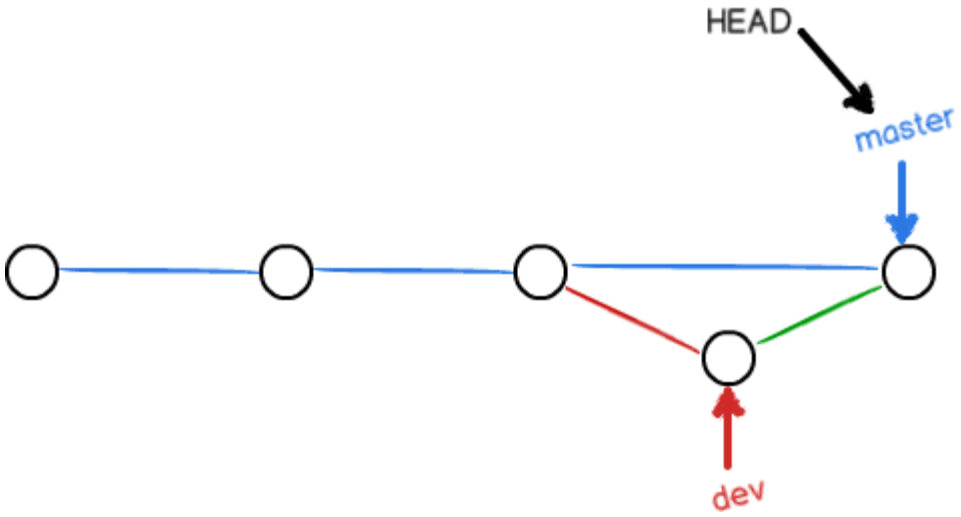
因为本次合并要创建一个新的 commit，所以加上-m 参数，把 commit 描述写进去。

合并后，我们用 git log 看看分支历史：

$ git log --graph --pretty=oneline --abbrev-commit

```
李响1@LiXiang MINGW64 /d/myrepo (master)
$ git log --graph --pretty=oneline --abbrev-commit
*   00ff21e (HEAD -> master) merge with no-ff
|\
| * 05ced57 (dev) modify symbol
|/
*   442ebd7 conflict fixed
|\
| * 194e84d add simple and quick
* | 85d1948 add simple and facile
|/
* 829caf2 add dev branch
* c06dd0d (origin/master) add test.txt
* dbf9ce6 remove test.txt
* e1cbd59 add test.txt
* fb31c6e add words in June 7th second
* dec163e add words in June 7th
* 4818645 understand how stage works
* 22a8b87 modify the word
* eb85296 The Dragon Boat Festival
* 4e29439 add something
* e2b2795 create a readme word file
```

可以看到，不使用 Fast forward 模式，merge 后分支结构图就像这样：



分支策略

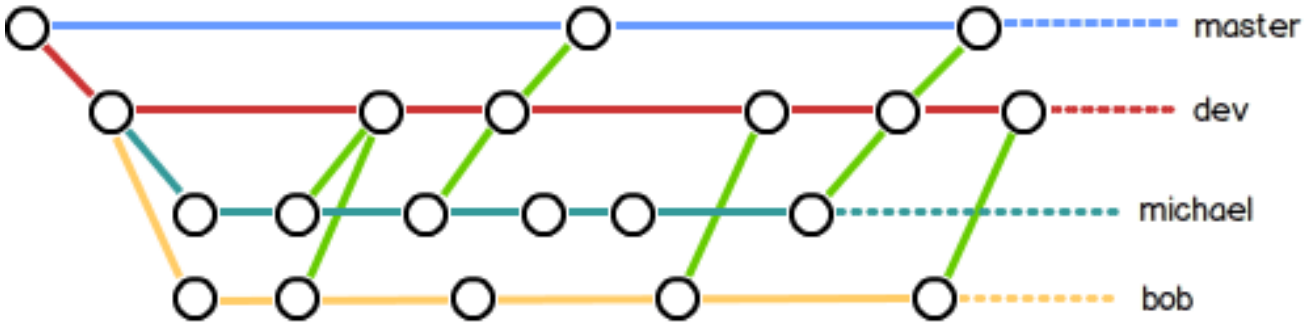
在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master 分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在 dev 分支上，也就是说，dev 分支是不稳定的，到某个时候，比如 1.0 版本发布时，再把 dev 分支合并到 master 上，在 master 分支发布 1.0 版本；

你和你的小伙伴们每个人都在 dev 分支上干活，每个人都有自己的分支，时不时地往 dev 分支上合并就可以了。

所以，团队合作的分支看起来就像这样：



总结：

Git 分支十分强大，在团队开发中应该充分应用。

合并分支时，加上--no-ff 参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而 fast forward 合并就看不出来曾经做过合并。