

使用 Git 建立一个版本控制系统

侯智星 zxhou@njust.edu.cn 18362969947

摘要：软件开发中经常需要团队协作，代码也会不断地迭代更新，为了方便的管理团队代码，我们就非常需要一种版本控制系统。Git 是目前世界上最先进的分布式版本控制系统，而且其拥有免费的代码托管网站 Github，因此，本文简单介绍 Git 的基本概念、安装方法、版本库、远程仓库、分支管理以及 Github 网站的使用，并且借助一个小例子一步一步地构建了一个版本控制系统。

关键字：Git 创建 分布式版本控制系统

一、 引言

Git 是目前世界上最先进的分布式版本控制系统。它的开发者就是 Linux 操作系统的作者 Linus Torvalds。原本 Git 的开发只是为了更好的管理 linux 内核，而现在却被广泛应用到其他各种项目中。Git 诞生后，迅速成为了最流行的分布式版本控制系统，尤其是 2008 年，Github 网站正式上线，它为开源项目提供免费的 Git 存储，无数开源项目开始迁移至 Github，包括 jQuery，PHP，Ruby 等。

目前存在有许多中版本控制系统工具，如 CVS，SVN。CVS 和 SVN 都是集中式的版本控制系统，而 Git 是分布式版本控制系统。集中式版本控制系统的版本库都是集中存放在中央服务器，其缺点是对互联网有着强烈的依赖性。而分布式版本控制系统克服了这一缺点，其没有所谓的中央服务器，每个人的电脑上都是一个完整的版本库。和集中式版本控制系统相比，分布式版本控制系统的安全性要高很多，另外 Git 还用拥有强大的分支管理功能，使得其与其他版本控制系统相比有着非常的大的优越性。

本文接下来的部分将会从零开始，一步一步讲述如何建立 Git。

二、 建立过程

最早的 Git 是在 Linux 上开发的，因此本文的示例过程就建立在 Ubuntu14.04 的基础上，但是目前 Git 也可以在 Mac、Windows 及其他 Linux 平台上正常运行。

2.1 安装 Git

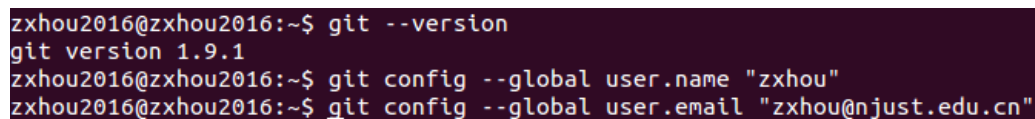
Linux 上安装 Git 非常简单，只需要一条 `sudo apt-get install git` 就成功了。

安装成功后，需要对 Git 进行简单的配置。因为 Git 是分布式版本控制系统，所以每台机器必须有自己的用户名和邮箱地址。这是需要用到命令：

```
git config --global user.name "Your name"
```

```
git config --global user.email email@example.com
```

如图 2.1



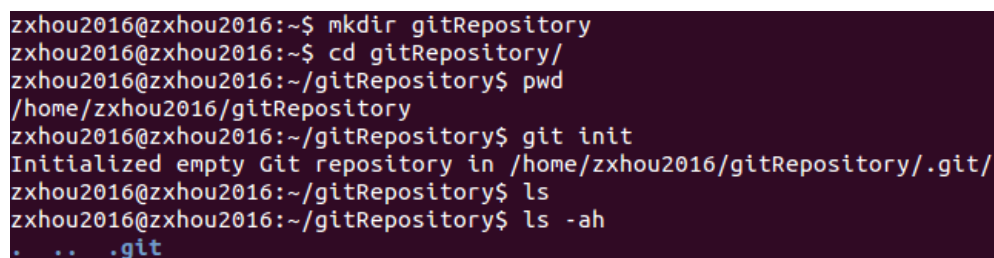
```
zxhou2016@zxhou2016:~$ git --version
git version 1.9.1
zxhou2016@zxhou2016:~$ git config --global user.name "zxhou"
zxhou2016@zxhou2016:~$ git config --global user.email "zxhou@njust.edu.cn"
```

图 2.1 配置 git 的用户名和邮箱

2.2 创建版本库

版本库又被称为仓库，可以简单地理解为一个目录这个目录里面所有的文件都可以被 Git 管理起来，每个文件的修改、删除，Git 都能进行跟踪，以便任何时候都可以追踪历史，或者在将来的某个时刻进行还原。

创建一个版本库非常简单，首先选择一个合适的地方，创建一个空目录。然后将该目录用 `git init` 命令变成 Git 可以管理的仓库。此时，Git 仓库就创建好了，此时的仓库是一个空的仓库，当前目录下多了一个 `.git` 目录，一般情况下这个目录是隐藏的，使用命令 `ls -ah` 可以看到。如图 2.2。



```
zxhou2016@zxhou2016:~$ mkdir gitRepository
zxhou2016@zxhou2016:~$ cd gitRepository/
zxhou2016@zxhou2016:~/gitRepository$ pwd
/home/zxhou2016/gitRepository
zxhou2016@zxhou2016:~/gitRepository$ git init
Initialized empty Git repository in /home/zxhou2016/gitRepository/.git/
zxhou2016@zxhou2016:~/gitRepository$ ls
zxhou2016@zxhou2016:~/gitRepository$ ls -ah
.  ..  .git
```

图 2.2 Git 仓库的建立

2.3 将文件添加进版本库

使用 `vim` 在 `/gitRepository` 目录或其子目录下创建 `readme.txt` 文件，并在其中输入：

Git is a version control system.

Git is a free software.

使用 `git add` 命令将文件添加到仓库中，然后使用 `git commit` 命令将文件提交到仓库，如图 2.3：

```
zxhou2016@zxhou2016:~/gitRepository$ vim readme.txt
zxhou2016@zxhou2016:~/gitRepository$ git add readme.txt
zxhou2016@zxhou2016:~/gitRepository$ git commit -m "creat a readme file."
[master (root-commit) da4bab3] creat a readme file.
1 file changed, 2 insertions(+)
create mode 100644 readme.txt
```

图 2.3 添加文件到仓库

图中的 `git commit` 命令有一个 `-m` 参数，后面输入的是本次提交的描述性说明。虽然这里可以输入任意的内容，但是对提交文件意义的描述对自己和别人阅读都非常的重要，方便自己和他人从历史记录中轻松地找到改动记录。如图中所示，`git commit` 命令执行成功之后会在窗口显示一个文件被改动，文件中插入了两行内容。

此外，`git commit` 可以一次性提交多个文件，所以可以多次 `git add`，最后一次性 `git commit` 提交。

2.4 版本管理

经过上述步骤，我们已经成功添加并提交了一个 `readme.txt` 文件，现在，我们尝试对该文件进行一定的修改，改成以下内容：

Git is a distributed version control system.

Git is a free software.

使用 `git status` 命令查看此时仓库的状态，状态显示 `readme.txt` 已经被修改过了，但是还没有准备提交的修改。此时，我们也可以使用 `git diff` 命令查看具体修改了什么内容，从窗口显示的命令来看，我们在第一行中添加了一个单词“distributed”。

了解了我们对文件进行了什么修改，我们在使用 `git add` 和 `git commit` 命令将其提交到仓库。在使用 `git commit` 命令的前后，我们可以用 `git status` 分别查看仓库的状态。如图 2.4 所示。

```

zxhou2016@zxhou2016:~/gitRepository$ vim readme.txt
zxhou2016@zxhou2016:~/gitRepository$ git status
On branch master
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.txt

no changes added to commit (use "git add" and/or "git commit -a")
zxhou2016@zxhou2016:~/gitRepository$ git diff readme.txt
diff --git a/readme.txt b/readme.txt
index b0354bd..5367626 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,2 +1,2 @@
-Git is a version control system.
+Git is a distributed version control system.
 Git is a free software.
zxhou2016@zxhou2016:~/gitRepository$ git add readme.txt
zxhou2016@zxhou2016:~/gitRepository$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   readme.txt

zxhou2016@zxhou2016:~/gitRepository$ git commit -m "add distributed"
[master 01b7ecd] add distributed
 1 file changed, 1 insertion(+), 1 deletion(-)
zxhou2016@zxhou2016:~/gitRepository$ git status
On branch master
nothing to commit, working directory clean

```

图 2.4 文件修改和状态

现在，我们再次对 readme.txt 文件进行修改：

Git is a distributed version control system.

Git is a free software distributed under the GPL.

在软件的开发过程中，我们像这样不断地提交修改到版本库里，每当觉得文件修改到一定程度的时候，就可以 commit 到 Git 中，一旦把文件修改乱了或者误删了文件，就可以从最近的一个 commit 中恢复文件，进而继续工作。然而，当我们多次修改文件中的内容后，我们自己是无法清楚地记住对每一次的修改，所以 Git 为我们提供了方便的解决方法，我们可以使用 git log 命令查看对文件的每一次修改。

git log 命令可以显示从近到远的提交日志，如图 2.5 可以看到我们一共进行了三次提交：

```

zxhou2016@zxhou2016:~/gitRepository$ git log
commit 1a498bb58caeb6565e7aa3a3a64afca39c374143
Author: zxhou <zxhou@njust.edu.cn>
Date:   Mon Jan 16 14:47:06 2017 +0800

    add GPL

commit 01b7ecd5421c7787017f6fe52408edd4b1c03ac3
Author: zxhou <zxhou@njust.edu.cn>
Date:   Mon Jan 16 14:31:38 2017 +0800

    add distributed

commit da4bab31d17406713028d798d02a55812af54bc2
Author: zxhou <zxhou@njust.edu.cn>
Date:   Mon Jan 16 13:52:03 2017 +0800

    creat a readme file.

```

图 2.5 Git 提交日志

现在，我们可以将 readme.txt 回退到前面任何一个版本，使用 `git reset` 命令可以完成这一操作，如图 2.6:

```

zxhou2016@zxhou2016:~/gitRepository$ git reset --hard HEAD^
HEAD is now at 01b7ecd add distributed
zxhou2016@zxhou2016:~/gitRepository$ git log
commit 01b7ecd5421c7787017f6fe52408edd4b1c03ac3
Author: zxhou <zxhou@njust.edu.cn>
Date:   Mon Jan 16 14:31:38 2017 +0800

    add distributed

commit da4bab31d17406713028d798d02a55812af54bc2
Author: zxhou <zxhou@njust.edu.cn>
Date:   Mon Jan 16 13:52:03 2017 +0800

    creat a readme file.

```

图 2.6 回退到前一个版本

2.5 添加远程仓库

Git 是分布式版本控制系统，同一个 Git 仓库，可以分布到不同的机器上。最开始，肯定只有一台机器上有原始的版本库，此后，别的机器就可以 clone 这个原始的版本库，并且每台机器上的版本库都是一样的，没有主次之分。

实际情况中，我们可以找一台电脑充当服务器的角色，每天 24 小时开机，其他每个人都从这个服务器仓库中克隆代码到自己的电脑上，并且把各自的代码提交到服务器仓库中。好在我们有一个 Github 网站，它是全球最大的代码托管网站，主要借助 Git 来进行版本控制。任何开源的代码都可以免费地将代码提交到 GitHub 上。

假设现在已经有了 Github 账号，首先要做的是创建 SSH Key，执行 `ssh-keygen -t rsa -C "zxhou@njust.edu.cn"`。后续的选项都可以采用默认值。操作结束后，可以在用户的主目录中找到 .ssh 目录，里面有 id_rsa 和 id_rsa.pub 两个文件，id_rsa 是私钥，id_rsa.pub 是公钥，

可以放心的告诉其他人。

然后登陆 Github，打开“Setting”，“SSH and GPG keys”页面，点击“New SSH key”，填写任意的 Title，在文本框里粘贴 id_rsa.pub 中的内容。单机“Add Key”，就可以看到添加的 key 了。如图 2.7 所示。

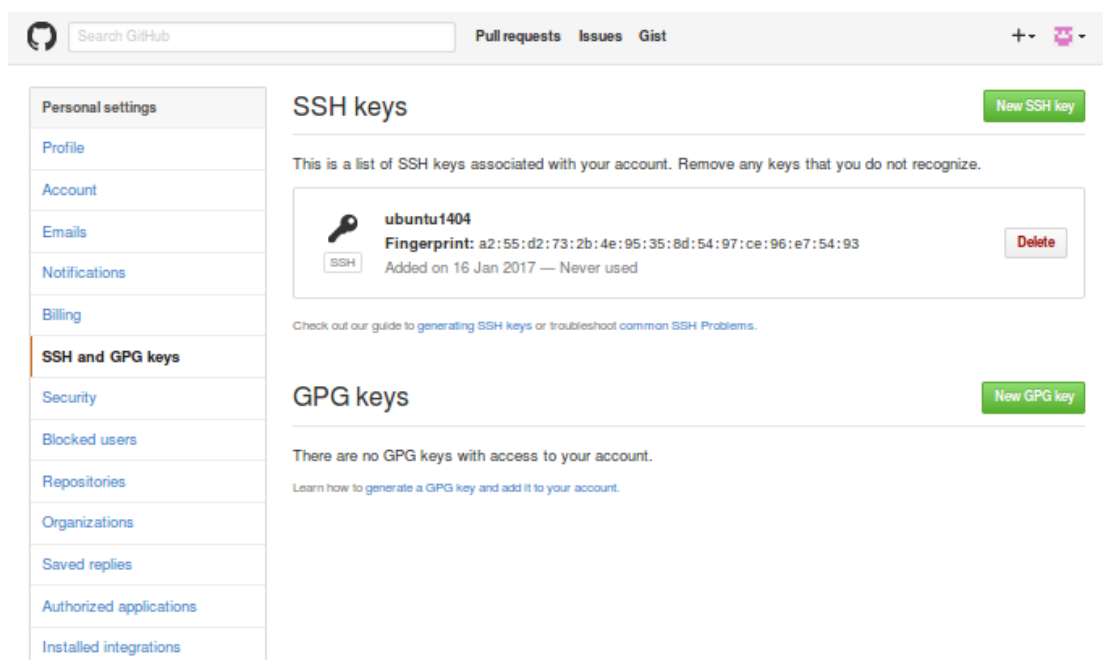


图 2.7 创建 SSH Key

Github 允许添加多个 Key，假如用户有多台电脑，就可以将每台电脑的 SSH Key 添加到 Github 账号中，这样就可以在任意一台电脑上向 Github 推送代码了。

我们已经有了一个本地的 Git 仓库，现在还要在 Github 上创建一个 Git 仓库，并且可以令本地仓库与 Github 仓库进行远程同步。

首先登陆 Github，在右上角找到一个十字符号，选择“New repository”，创建一个新的仓库，在 Repository name 里填入仓库名，保持其他选项为默认，点击“Create repository”，成功创建一个新的 Github 仓库。

接下来可以使用命令 `git remote add origin git@github.com:zxhou/testGit.git`，将本地仓库关联远程库。下一步，就可以把本地库的所有内容推送到远程库上：`git push origin master`。该命令实际上将当前分支 master 推送到远程仓库。如图 2.8 所示：

```
zxhou2016@zxhou2016:~/gitRepository$ git push origin master
Counting objects: 12, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (11/11), 1.03 KiB | 0 bytes/s, done.
Total 11 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To git@github.com:zxhou/testGit.git
ddcf828..23481f0 master -> master
```

图 2.8 将本地库推送到远程

推送成功后，刷新 Github 界面就可以看到远程库的内容与本地库一模一样，如图 2.9。

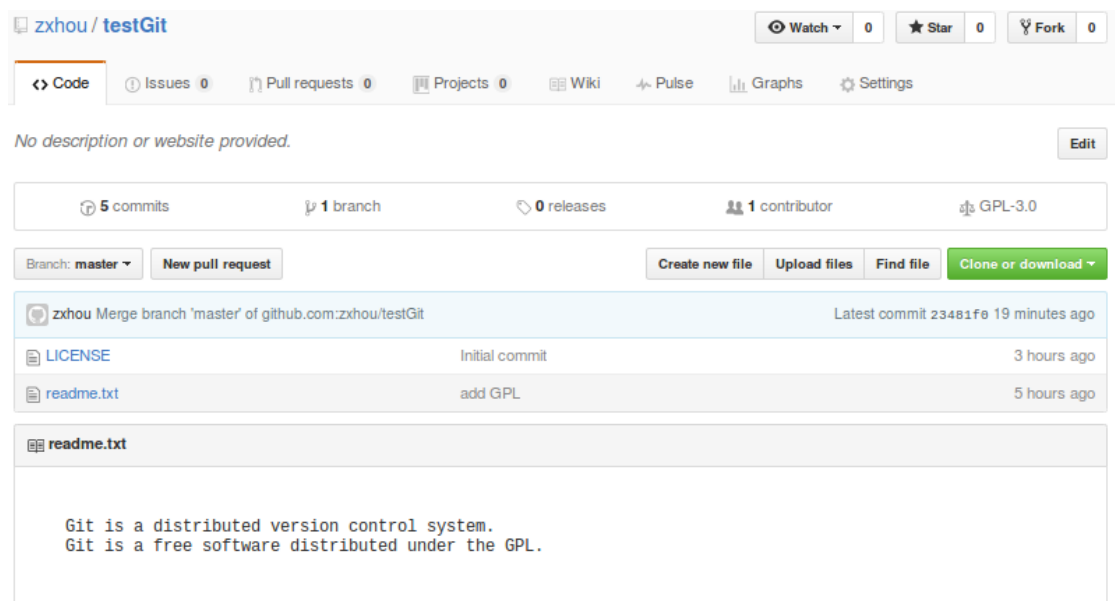


图 2.9 Github 远程仓库

2.6 从远程库克隆代码

从远程库克隆代码的前提是 Github 已经有了一个仓库，这里我建立了一个新的仓库，命名为 testGitClone，其中包含三个文件，分别为.gitignore，LICENSE，README.md。如图 2.10 所示。

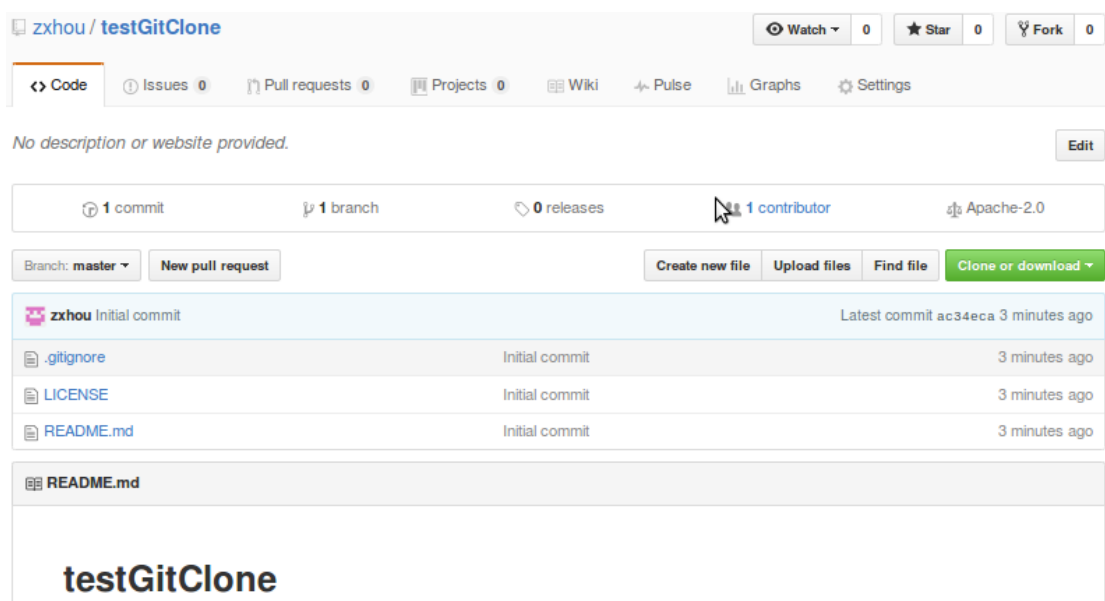


图 2.10 新建的 testGitClone 远程仓库

此时，我们使用命令

`git clone http://github.com/zxhou/testGitClone.git` 从远程库克隆一个本地库，如图 2.11 所示。

```
zxhou2016@zxhou2016:~/test/learnGit$ git clone http://github.com/zxhou/testGitClone.git
Cloning into 'testGitClone'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), done.
Checking connectivity... done.
zxhou2016@zxhou2016:~/test/learnGit$ ls
testGitClone
zxhou2016@zxhou2016:~/test/learnGit$ cd testGitClone/
zxhou2016@zxhou2016:~/test/learnGit/testGitClone$ ls
LICENSE  README.md
```

图 2.11 完成克隆的本地库

2.7 Git 的分支功能

分支功能对于多人协作的项目有着非常重大的作用。如果没有分支功能，假设要开发一个新功能，但是需要两个星期完成，可能第一周结束时只完成了 50%的代码，如果此时立刻提交，由于代码还没有写完，不完整的代码库会导致协作者无法在该工作的基础上继续工作。如果等到代码全部写完再一次性提交，就会使得本周没有工作进度。

但是有了分支功能，这个问题就得到了非常好的解决。首先创建一个只属于自己的分支，别人看不到，还继续在原来的分支上正常工作，而自己则在自己的分支上工作，可以随时提交自己的代码，直到全部开发完毕后，再一次性合并到原来的分支上，这样，安全又不会影响到其他人工作。

在 Git 中，master 分支即为主分支，一开始的时候，master 分支是一条线，Git 用 master 指向最新的提交，再用 HEAD 指向 master，就能确定当前分支以及当前分支的提交点。当新建分支后，HEAD 就指向了新建的分支，当出现新的提交时，HEAD 和新的分支在增加，而 master 不会发生任何变化，直到工作全部完成了，就可以把分支合并到 master 上了，之后删掉分支，就只剩下了一个 master 主分支了。

我们可是使用 `git checkout -b dev` 创建一个 dev 分支，并且切换到该分支上，-b 参数表示创建并切换，这条命令就相当于：

```
git branch dev
```

```
git checkout dev
```

两条命令。然后用 `git branch` 命令查看当前分支，它会列出所有分支，当前分支的前面会标有一个*号。之后就可以在 dev 分支上正常提交了。dev 分支的工作全部完成后，就可以切换回 master 主分支。如图 2.12 所示。



```
zxhou2016@zxhou2016:~/gitRepository$ git checkout -b dev
switched to a new branch 'dev'
zxhou2016@zxhou2016:~/gitRepository$ git branch
* dev
  master
zxhou2016@zxhou2016:~/gitRepository$ vim readme.txt
zxhou2016@zxhou2016:~/gitRepository$ git add readme.txt
zxhou2016@zxhou2016:~/gitRepository$ git commit -m "add branch"
[dev 4b6d9a9] add branch
1 file changed, 1 insertion(+)
zxhou2016@zxhou2016:~/gitRepository$ git checkout master
switched to branch 'master'
```

图 2.12 创建新的分支

然而当我们切换到 master 主分支上时，发现刚刚在 readme.txt 文件中添加的内容不见了。这是因为，刚刚的提交只发生在 dev 分支上，而 master 分支并没有发生变化。这个时候就需要将 dev 分支上的工作成果合并到 master 分支上。

我们可以执行 `git merge dev` 命令将制定分支合并到当前分支上，合并后再查看 readme.txt 的内容，可以看到刚刚添加的内容出现了。然后我们就可以放心的删除 dev 分支了 `git branch -d dev`。最后查看分支，发现只剩下了 master 分支，如图 2.13 所示。

```

zxhou2016@zxhou2016:~/gitRepository$ git merge dev
Updating 23481f0..4b6d9a9
Fast-forward
 readme.txt | 1 +
 1 file changed, 1 insertion(+)
zxhou2016@zxhou2016:~/gitRepository$ cat readme.txt
Git is a distributed version control system.
Git is a free software distributed under the GPL.
Create a new branch dev.
zxhou2016@zxhou2016:~/gitRepository$ git branch -d dev
Deleted branch dev (was 4b6d9a9).
zxhou2016@zxhou2016:~/gitRepository$ git branch
* master

```

图 2.13 合并分支到 master 主分支

2.8 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，平时不能在其上工作。

工作一般都会建立在 dev 分支之上，也就是说 dev 分支是不稳定的，直到开发到一定的阶段时才会把 dev 分支合并到 master 上，即在 master 分支上发布新版本。

所以团队中的每一个成员都在 dev 分支上工作，每个人都有自己的分支，并且时不时往 dev 分支上合并就可以了。如图 2.14 所示。

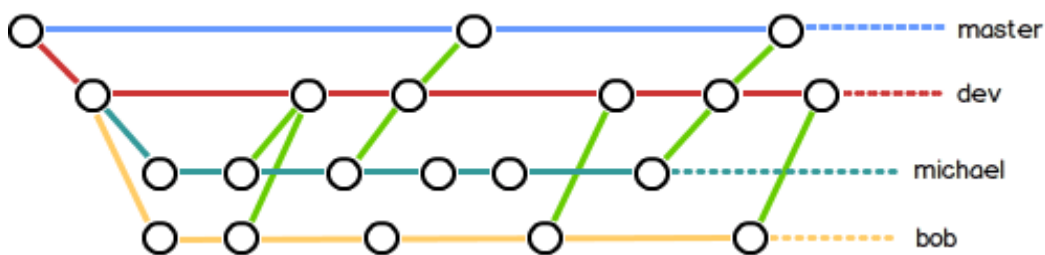


图 2.14 团队协作中的分支策略

三、 总结

Git 的功能非常强大，本文选取了其中一部分对其进行介绍，并且借助一个简单的例子来说明如何构建一个版本控制系统。更多更详细的说明在 Git 的官方网站[1]上都有详细的说明，本文在写作过程中参考了众多网站、博客[2][3]。因为 Git 的存在，将使得软件工程师的工作效率得到大幅提升，非常建议深入学习 Git 强大的功能，并且将 Git 真正运动到实际的开发过程中。

[1] <https://git-scm.com>

[2] <http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000>

[3] <http://www.cnblogs.com/smyhvae/p/4052539.html>