创建版本库：

现在总结一下今天学的两点内容：

初始化一个Git仓库，使用`git init`命令。

添加文件到Git仓库，分两步：

- 第一步，使用命令`git add <file>`，注意，可反复多次使用，添加多个文件；
- 第二步，使用命令`git commit`，完成。

工作区和暂存区状态查看：

- 要随时掌握工作区的状态，使用`git status`命令。
- 如果`git status`告诉你有文件被修改过，用`git diff`可以查看修改内容。

版本回退：

- HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令`git reset --hard commit_id`。
- 穿梭前，用`git log`可以查看提交历史，以便确定要回退到哪个版本。
- 要重返未来，用`git reflog`查看命令历史，以便确定要回到未来的哪个版本。
- 查看head历史记录git log --oneline --stat

暂存区和工作区：

暂存区是介于工作区与分支之间的区域

git add <file>是提交到暂存区

git commit 是提交到分支

管理修改：

理解了Git是如何跟踪修改的，每次修改，如果不`add`到暂存区，那就不会加入到`commit`中。

撤销修改：

- 场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令`git checkout -- file`。

- 场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令`git reset HEAD file`，就回到了场景1，第二步按场景1操作。
- 场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考版本回退一节，不过前提是没有推送到远程库。

删除文件：

命令`git rm`用于删除一个文件。如果一个文件已经被提交到版本库，那么你永远不用担心误删，但是要小心，你只能恢复文件到最新版本，你会丢失**最近一次提交后你修改的内容**。

添加远程库：

要关联一个远程库，使用命令`git remote add origin git@server-name:path/repo-name.git`；

关联后，使用命令`git push -u origin master`第一次推送master分支的所有内容；

此后，每次本地提交后，只要有必要，就可以使用命令`git push origin master`推送最新修改；

分布式版本系统的最大好处之一是在本地工作完全不需要考虑远程库的存在，也就是有没有联网都可以正常工作，而SVN在没有联网的时候是拒绝干活的！当有网络的时候，再把本地提交推送一下就完成了同步，真是太方便了！

克隆远程库：

要克隆一个仓库，首先必须知道仓库的地址，然后使用`git clone`命令克隆。

Git支持多种协议，包括`https`，但通过`ssh`支持的原生`git`协议速度最快。

创建与合并分支：

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>`

创建+切换分支：`git checkout -b <name>`

合并某分支到当前分支：`git merge <name>`

删除分支：`git branch -d <name>`

查看远程分支：git branch -a

查看本地分支：git branch

创建分支：git branch test

把本地分支推送到远程主干：git push origin test

切换到test分支：git checkout test

删除本地test分支：git branch -d test

删除远程版本分支：git push origin :br-1.0.0

-

解决冲突：

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。

用`git log --graph`命令可以看到分支合并图。

分支管理策略：

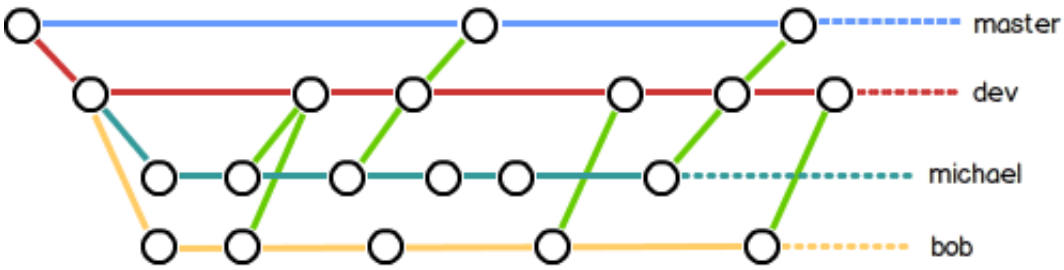
在实际开发中，我们应该按照几个基本原则进行分支管理：

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：



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。

Bug**分支管理**：

修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；

当手头工作没有完成时，先把工作现场`git stash`一下，然后去修复bug，修复后，再`git stash pop`，回到工作现场。

Feature**分支管理**：

开发一个新feature，最好新建一个分支；

如果要丢弃一个没有被合并过的分支，可以通过`git branch -D <name>`强行删除。

多人协作：

- 查看远程库信息，使用`git remote -v`；
- 本地新建的分支如果不推送到远程，对其他人就是不可见的；
- 从本地推送分支，使用`git push origin branch-name`，如果推送失败，先用`git pull`抓取远程的新提交；
- 在本地创建和远程分支对应的分支，使用`git checkout -b branch-name origin/branch-name`，本地和远程分支的名称最好一致；
- 建立本地分支和远程分支的关联，使用`git branch --set-upstream branch-name origin/branch-name`；
- 从远程抓取分支，使用`git pull`，如果有冲突，要先处理冲突。

创建标签：

- 命令`git tag <name>`用于新建一个标签，默认为HEAD，也可以指定一个commit id；
- `git tag -a <tagname> -m "blablabla..."`可以指定标签信息；
- `git tag -s <tagname> -m "blablabla..."`可以用PGP签名标签；
- 命令`git tag`可以查看所有标签。
- 命令git checkout <tag-id>可以切换到相应的版本

操作标签：

- 命令`git push origin <tagname>`可以推送一个本地标签；
- 命令`git push origin --tags`可以推送全部未推送过的本地标签；
- 命令`git tag -d <tagname>`可以删除一个本地标签；
- 命令`git push origin :refs/tags/<tagname>`可以删除一个远程标

签。

从远程获取更新
git -c diff.mnemonicprefix=false -c core.quotepath=false fetch origin

提交到远程分支
git push origin master

查看文件历史记录
git log --pretty=oneline
git log <filename>
git log -p <filename>查看版本间的diff
git show <commit-id> <filename> 查看某次提交中某个文件的变化

查看当前tag
git describe --tags

查看head历史记录
git log --oneline --stat

设置当前用户以及email
git config --global user.name <name>
git config --global user.email <emailaddress>
git config --list（列出所有已经设置的参数）