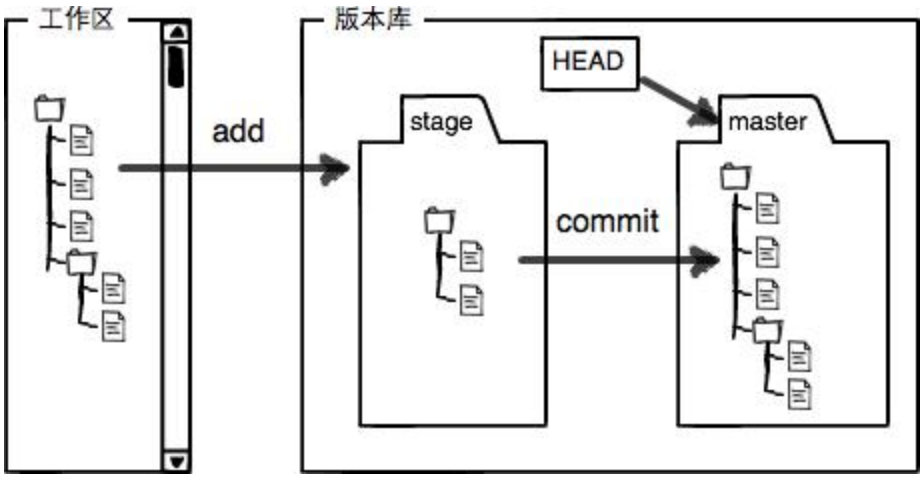
Git



Git 相当于版本控制工具

git init

git add xxx

将当前目录下的所有文件加入

git add .

git commit -m xxxx

git status

git diff

git log

输入字母q退出

git rest —hard 回退或者前进一个版本

```
git reset --hard 1094a
```

git reflog 记录每一次命令

暂存区 概念: 暂存区在版本库里

master 默认分配的第一个分支

> 为什么Git比其他版本控制系统设计得优秀，因为Git跟踪并管理的是修改，而非文件。

> 两次修改，只add一次，会丢失修改，可见文件更改但是不提交不会被git更改

git checkout -- readme.txt

回到最近一次add或者commit状态

git reset HEAD <file> 可以将文件从暂存区移动到工作区 HEAD的意思是永远指向最新版本

git restore <文件>... 丢弃工作区的改动

`git reset --hard HEAD^` 对于已经提交到库的文件，用这个进行版本回退

`rm test.txt` 删除文件，删除的肯定是工作区的，Linux指令

`git checkout` 其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。git checkout 是对版本库进行操作的，如果文件的相关修改都没提交到版本库，就无法恢复了

对于文件删除操作使用 `git restore --staged <deleted file>`

来丢弃暂存区该次删除操作, 在 `git checkout -- <delete file>，文件就还原了`

checkout相当于对HEAD指针进行操作

远程库

关联远程库

git remote add origin git@server-name:path/repo-name.git

将本地master分支上传到git远程库

git push -u origin master

正常的顺序应该是先建立远程库，然后从远程库克隆自己的代码下来

git clone xxx

分支

分支创建

```
git switch -c dev
```

创建dev分支，并将head指向dev

只创建，不改HEAD

git branch dev

改head 切换分支

git switch <name>

合并分支

```
git merge dev
```

这种方式是快速合并，快速合并会删除分支信息

删除分支

```
git branch -d dev
```

查看当前的分支

git branch

无论是什么分支，在暂存区是公共的

分支策略

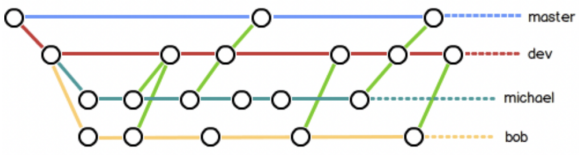
在实际开发中，我们应该按照几个基本原则进行分支管理：

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：




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

不进行快速合并

stash功能

比如要中途出去改某个bug，但是当前的工作区不想上传，就用stash暂时保存

`git stash`

查看保存过的现场

git stash list

恢复现场 + 删除保存的现场

git stash apply + git stash drop

恢复+删除

git stash pop

指定恢复到哪个

`git stash apply stash@{0}`

在一个分支当中修改，修改之后复制到另一个分支，避免再修改一遍

`git cherry-pick xxx`

分支没有合并之前不能直接-d删除，需要

`git branch -D feature-vulcan`

强行删除

查看远程库

`git remote`

多人协作方式

> 因此，多人协作的工作模式通常是这样：
>
> 1. 首先，可以试图用 `git push origin <branch-name>` 推送自己的修改；
> 2. 如果推送失败，则因为远程分支比你的本地更新，需要先用 `git pull` 试图合并；
> 3. 如果合并有冲突，则解决冲突，并在本地提交；
> 4. 没有冲突或者解决掉冲突后，再用 `git push origin <branch-name>` 推送就能成功！
>
> 如果 `git pull` 提示 `no tracking information` ，则说明本地分支和远程分支的链接关系没有创建，用命令 `git branch --set-upstream-to <branch-name> origin/<branch-name>` 。

• 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；

都需要先关联分支再pull

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。

为了减少冲突，会规定使用统一的ide进行程序编写，这样在下载之后再更改 能够避免因版本和格式造成的无法pul

标签

对commit标签可以不用再输commit号了

- 命令`git tag <tagname>`用于新建一个标签，默认为`HEAD`，也可以指定一个commit id；
- 命令`git tag -a <tagname> -m "blablabla..."`可以指定标签信息；
- 命令`git tag`可以查看所有标签。
- 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。

配置别名

`git config --global alias.st status`