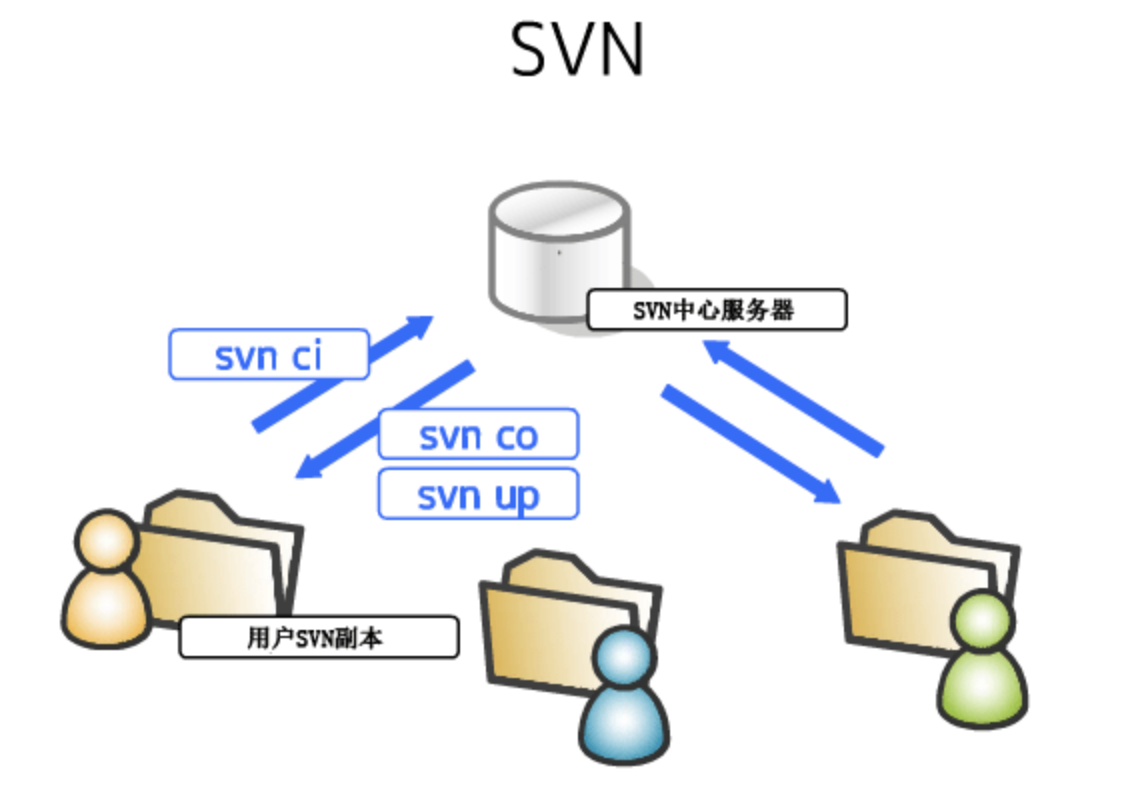
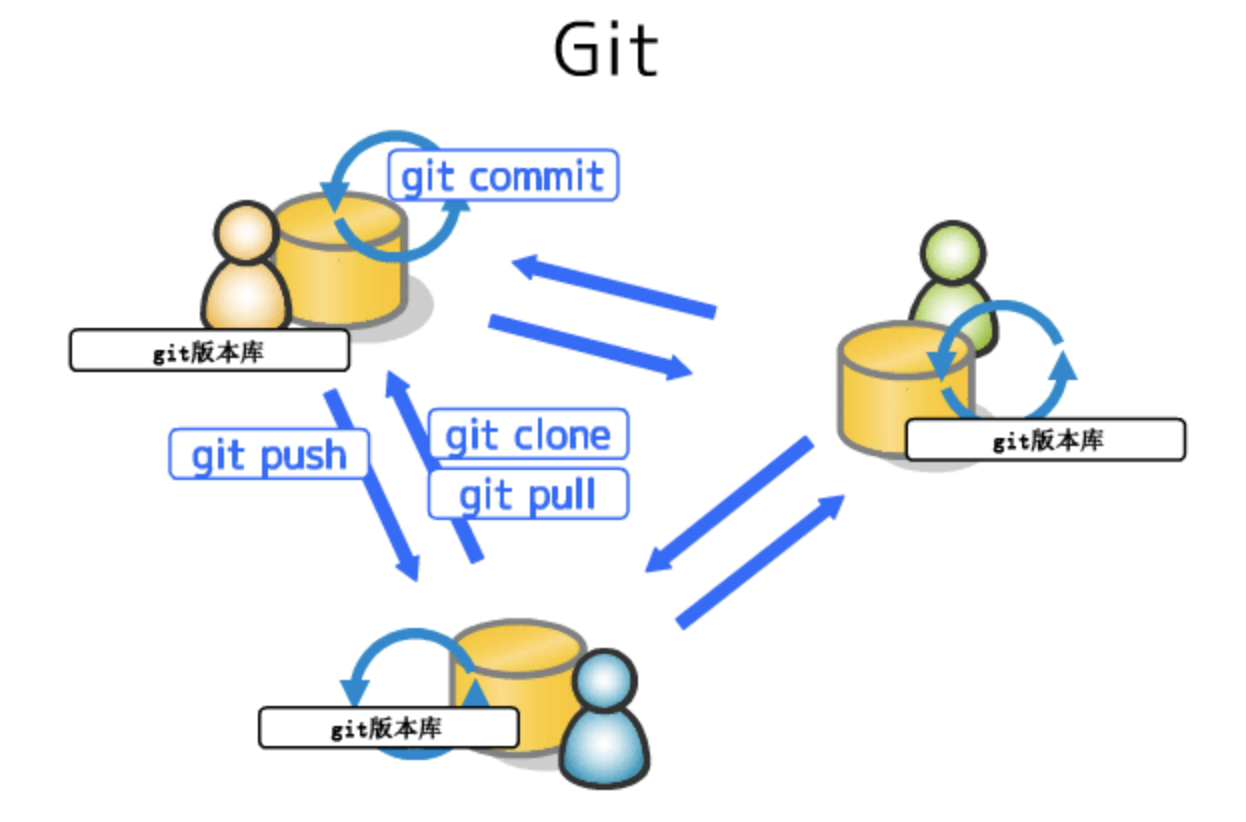
1. **Git是什么？**

Git是目前世界上最先进的分布式版本控制系统

集中式的版本控制系统与分布式版本控制系统的区别：







1. **安装git**

### 2.1在 Linux 上安装

基于Fedora 的发行版本，可以使用 yum：

$ sudo yum install git

基于 Debian 的发行版上，用 apt-get：

$ sudo apt-get install git

2.2 mac上安装git可参考<https://www.jianshu.com/p/7edb6b838a2e>

1. **本地Git仓库和远程Gitlab仓库**
   1. **git本地配置**

Git的设置文件为.gitconfig，它可以在用户主目录下（全局配置），也可以在项目目录下（项目配置）

# 显示当前的Git配置

$ git config --list

# 编辑Git配置文件

$ git config -e [--global]

# 设置提交代码时的用户信息

$ git config [--global] user.name "[name]"

$ git config [--global] user.email "[email address]"

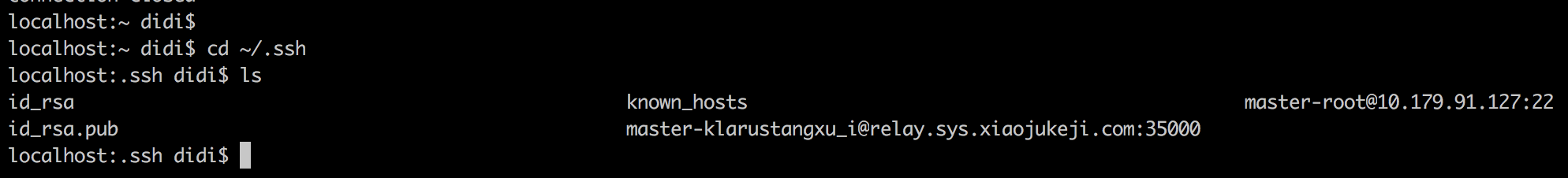
**3.2 本地git仓库和远程gitlab传输配置**

由于本地Git仓库和远程Gitlab仓库之间的传输是通过SSH加密的，所以，需要一点设置：

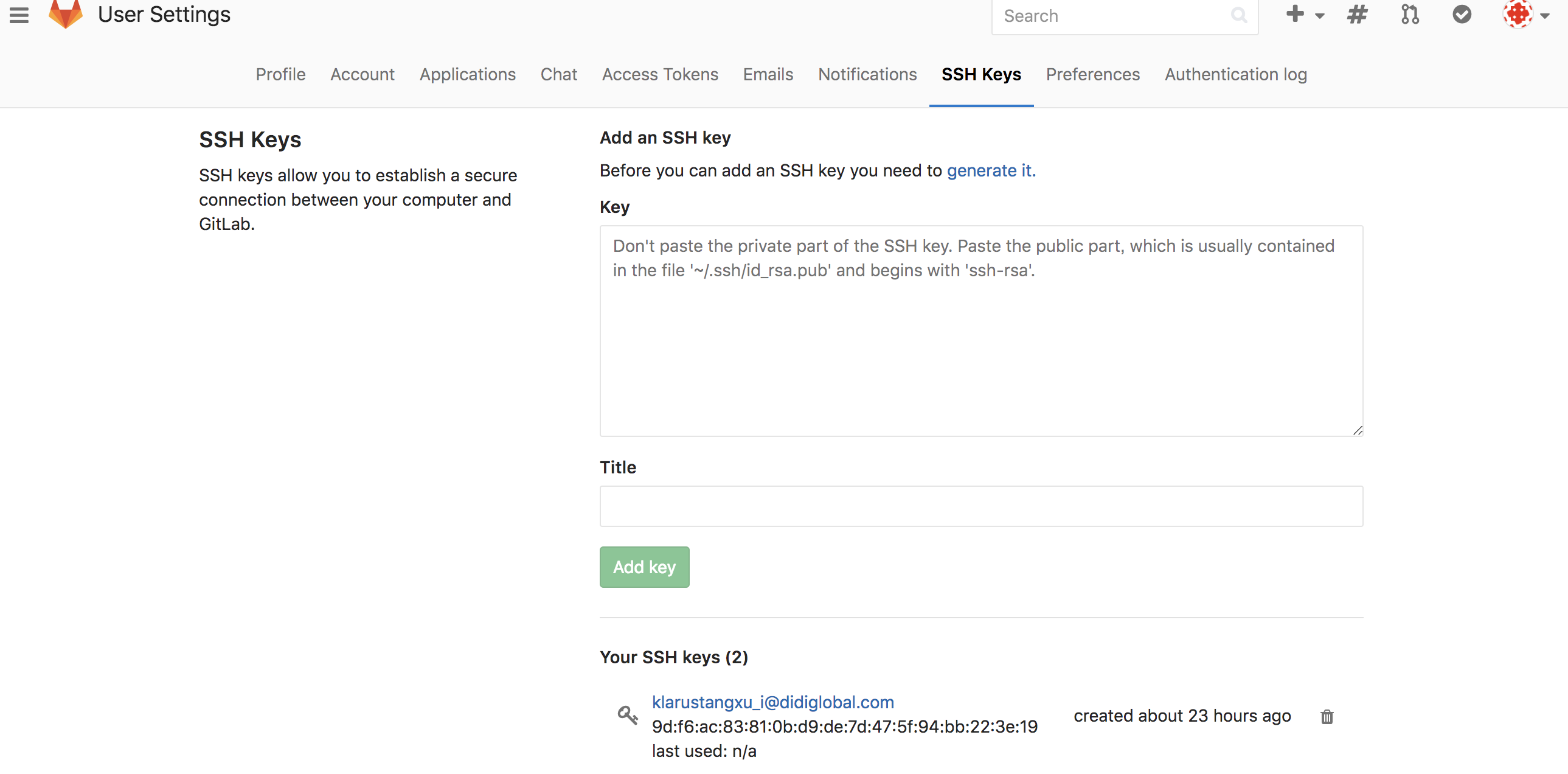
第1步：创建SSH Key

$ ssh-keygen -t rsa -C "youremail@example.com"

可以在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的秘钥对，id\_rsa是私钥，id\_rsa.pub是公钥



第2步：登陆Gitlab，打开“Account settings”，“SSH Keys”页面。然后，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容，保存即可



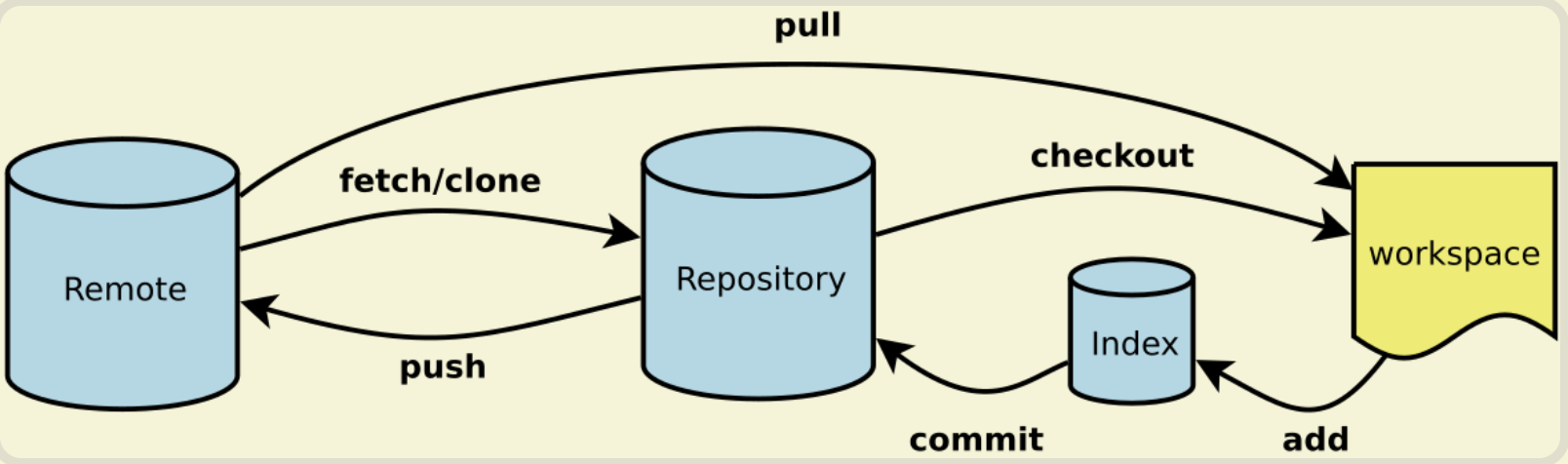
* 1. **git 中本地仓库与远程仓库关系**

Workspace：工作区

Index / Stage：暂存区

Repository：仓库区（或本地仓库）

Remote：远程仓库



1. **分支**
   1. **分支管理常用命令**

$git branch 列出所有本地分支

$git branch –r 列出所有远程分支

$git branch –a 列出所有本地分支和远程分支

$git branch [branch-name] 创建分支，但仍然停留在当前分支

$git checkout -b [branch-name] 创建一个分支，并切换到该分支

$git checkout [branch-name] 切换到指定分支，并更新工作区

$git checkout - 切换到上一个分支

$git branch --set-upstream [branch] [remote-branch]

$git merge dev 合并当前分支，一般先切换到需要被合并的分支，如切换到master（默认以Fast Forword合并）

$git merge --no-ff -m "merge with no-ff" dev --no-ff参数，表示禁用Fast forward，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。

$git branch -d [branch-name] 删除分支

$git push orign –delete [branch-name]

$git branch –dr [remote/branch] 删除远程分支

$git log --graph 看到分支合并图

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

 如果想把本地的test分支提交到远程仓库，并作为远程仓库的master分支，或者作为另外一个名叫test的分支,可用以下命令：

$ git push origin test:master         // 提交本地test分支作为远程的master分支

$ git push origin test:test              // 提交本地test分支作为远程的test分

查看远程分支commit信息用于回退到远程分支的某个版本

$ git fetch

$ git log

$ git reset --hard commit\_id

**4.2 分支策略**

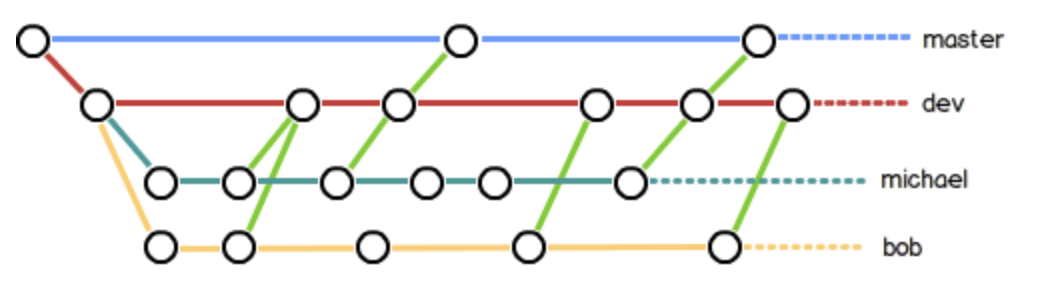
4.2.1 多人协作的基本原则

在实际开发中，都是遵循多人协作的模式，我们应该按照几个基本原则进行分支管理：

1、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；

2、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；

3、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



4.2.2 分支是否推送到远程

1．master分支是主分支，因此要时刻与远程同步；

2．dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；

3．bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；

4．feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

1. **增加/删除文件**

# 添加指定文件到暂存区

$ git add [file1] [file2] ...

# 添加指定目录到暂存区，包括子目录

$ git add [dir]

# 添加当前目录的所有文件到暂存区

$ git add .

# 添加每个变化前，都会要求确认

# 对于同一个文件的多处变化，可以实现分次提交

$ git add -p

# 删除工作区文件，并且将这次删除放入暂存区

$ git rm [file1] [file2] ...

# 停止追踪指定文件，但该文件会保留在工作区

$ git rm --cached [file]

# 改名文件，并且将这个改名放入暂存区

$ git mv [file-original] [file-renamed]

1. **代码提交**

# 提交暂存区到仓库区

$ git commit -m [message]

# 提交暂存区的指定文件到仓库区

$ git commit [file1] [file2] ... -m [message]

# 提交工作区自上次commit之后的变化，直接到仓库区

$ git commit -a

# 提交时显示所有diff信息

$ git commit -v

# 使用一次新的commit，替代上一次提交

# 如果代码没有任何新变化，则用来改写上一次commit的提交信息

$ git commit --amend -m [message]

# 重做上一次commit，并包括指定文件的新变化

$ git commit --amend [file1] [file2] ...

**7 远程同步**

# 下载远程仓库的所有变动

$ git fetch [remote]

# 显示所有远程仓库

$ git remote -v

# 显示某个远程仓库的信息

$ git remote show [remote]

# 增加一个新的远程仓库，并命名

$ git remote add [shortname] [url]

# 取回远程仓库的变化，并与本地分支合并

$ git pull [remote] [branch]

# 上传本地指定分支到远程仓库

$ git push [remote] [branch]

# 强行推送当前分支到远程仓库，即使有冲突

$ git push [remote] --force

# 推送所有分支到远程仓库

$ git push [remote] --all

**8 撤销**

# 恢复暂存区的指定文件到工作区

$ git checkout [file]

# 恢复某个commit的指定文件到暂存区和工作区

$ git checkout [commit] [file]

# 恢复暂存区的所有文件到工作区

$ git checkout .

# 重置暂存区的指定文件，与上一次commit保持一致，但工作区不变

$ git reset [file]

# 重置暂存区与工作区，与上一次commit保持一致

$ git reset --hard

# 重置当前分支的指针为指定commit，同时重置暂存区，但工作区不变

$ git reset [commit]

# 重置当前分支的HEAD为指定commit，同时重置暂存区和工作区，与指定commit一致

$ git reset --hard [commit]

# 重置当前HEAD为指定commit，但保持暂存区和工作区不变

$ git reset --keep [commit]

# 新建一个commit，用来撤销指定commit

# 后者的所有变化都将被前者抵消，并且应用到当前分支

$ git revert [commit]

# 暂时将未提交的变化移除，稍后再移入

$ git stash

$ git stash pop

1. **标签**

# 列出所有tag

$ git tag

# 新建一个tag在当前commit

$ git tag [tag]

# 新建一个tag在指定commit

$ git tag [tag] [commit]

# 删除本地tag

$ git tag -d [tag]

# 删除远程tag

$ git push origin :refs/tags/[tagName]

# 查看tag信息

$ git show [tag]

# 提交指定tag

$ git push [remote] [tag]

# 提交所有tag

$ git push [remote] --tags

# 新建一个分支，指向某个tag

$ git checkout -b [branch] [tag]

1. **查看信息**

# 显示有变更的文件

$ git status

# 显示当前分支的版本历史

$ git log

# 显示commit历史，以及每次commit发生变更的文件

$ git log --stat

# 搜索提交历史，根据关键词

$ git log -S [keyword]

# 显示某个commit之后的所有变动，每个commit占据一行

$ git log [tag] HEAD --pretty=format:%s

# 显示某个commit之后的所有变动，其"提交说明"必须符合搜索条件

$ git log [tag] HEAD --grep feature

# 显示某个文件的版本历史，包括文件改名

$ git log --follow [file]

$ git whatchanged [file]

# 显示指定文件相关的每一次diff

$ git log -p [file]

# 显示过去5次提交

$ git log -5 --pretty --oneline

# 显示所有提交过的用户，按提交次数排序

$ git shortlog -sn

# 显示指定文件是什么人在什么时间修改过

$ git blame [file]

# 显示暂存区和工作区的差异

$ git diff

# 显示暂存区和上一个commit的差异

$ git diff --cached [file]

# 显示工作区与当前分支最新commit之间的差异

$ git diff HEAD

# 显示两次提交之间的差异

$ git diff [first-branch]...[second-branch]

# 显示今天你写了多少行代码

$ git diff --shortstat "@{0 day ago}"

# 显示某次提交的元数据和内容变化

$ git show [commit]

# 显示某次提交发生变化的文件

$ git show --name-only [commit]

# 显示某次提交时，某个文件的内容

$ git show [commit]:[filename]

# 显示当前分支的最近几次提交

$ git reflog

1. **bug分支和暂存工作现场**

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又不想commit时，先把工作现场暂存，然后去修复bug，修复后，再git stash pop，回到工作现场。

假设暂存现场在dev分支，总体流程如下：

$git stash

去修bug

git checkout dev

git stash list 会出现stash@{0}: WIP on dev: 6224937 add mergegit

git stash pop 还可以通过 git stash apply stash@{0}进行恢复，再删除stask内容git stash drop

git stash list

**12 gitlab远程仓库**

第一种情况：远程只有仓库没有内容时，将本地已经写好的代码关联到远程：

$ git remote add origin git@\*\*\*\*/\*\*\*.git 关联远程库

$ git push -u origin master -u只在第一次推送分支时使用

$ git push origin master

$ git push origin dev

一般master分支和dev分支都会推到远程

第二种情况：远程仓库有内容，直接克隆到本地

$ git clone git@\*\*\*\*/\*\*\*.git

$ git branch

$ git checkout -b dev origin/dev

$ git add file\_name

$ git commit -m "记录提交信息"

$ git push origin dev

在GitHub上，可以任意Fork开源仓库；

自己拥有Fork后的仓库的读写权限；

可以推送pull request给官方仓库来贡献代码。

如果git checkout -b dev orgin/dev迁出分支出现问题

fatal: Cannot update paths and switch to branch 'dev' at the same time.

Did you intend to checkout 'orgin/dev' which can not be resolved as commit?

解决办法是先git fetch，再git checkout -b dev origin/dev

如果git push失败，先git pull

如果git pull失败，原因是没有指定本地dev分支与远程origin/dev分支的链接，根据提示，设置dev和origin/dev的链接：

git branch --set-upstream dev origin/dev

然后在git pull，git push origin master

有时候不想解决冲突时：

1，从远处拉取最新的东西到本地（不想解决冲突直接接受远程仓库内容）

$ git fetch --all //至少下载到本地不进行合并

$ git reset --hard origin/dev

$ git pull

2，将本地更新强制推到远程，不想解决和远程的冲突

$ git push --force origin dev 或者git push -f origin dev

git更多内容可参考如下链接：

<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000>

<http://www.ruanyifeng.com/blog/2018/10/git-internals.html>

<https://www.bookstack.cn/books/git-tutorial>