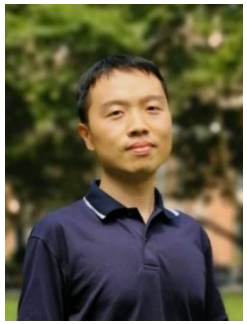
中山大學
SUN YAT-SEN UNIVERSITY

程序设计II



张书航 助理教授

电子邮件：zhangsh52@mail.sysu.ed
个人主页：shuhangz.github.io

李同文 助理教授

电子邮件：litw8@mail.sysu.edu.cn

2022/04/17

中山大學
SUN YAT-SEN UNIVERSITY

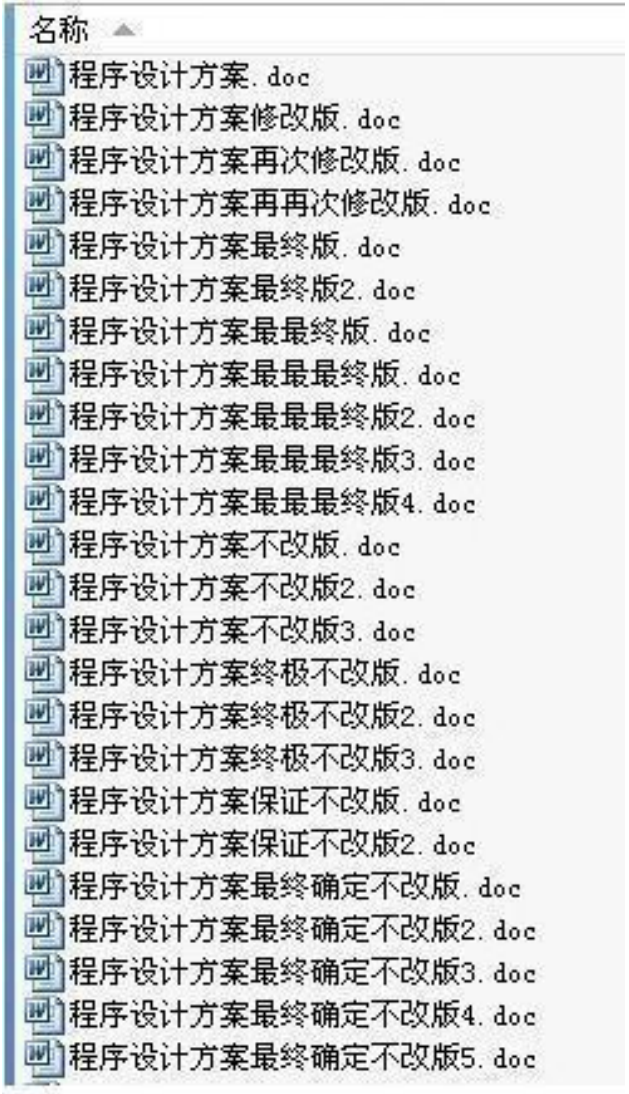
版本控制与团队协作编程

程序设计II

版本控制

版本控制系统

- 为什么要进行版本控制?
  - 改完之后发现要回退到某个版本?
  - 自己写的东西被别人覆盖了?
  - 每个版本到底改了什么?
  - 突然出现BUG，未来要发布的代码却不能动?



版本控制系统

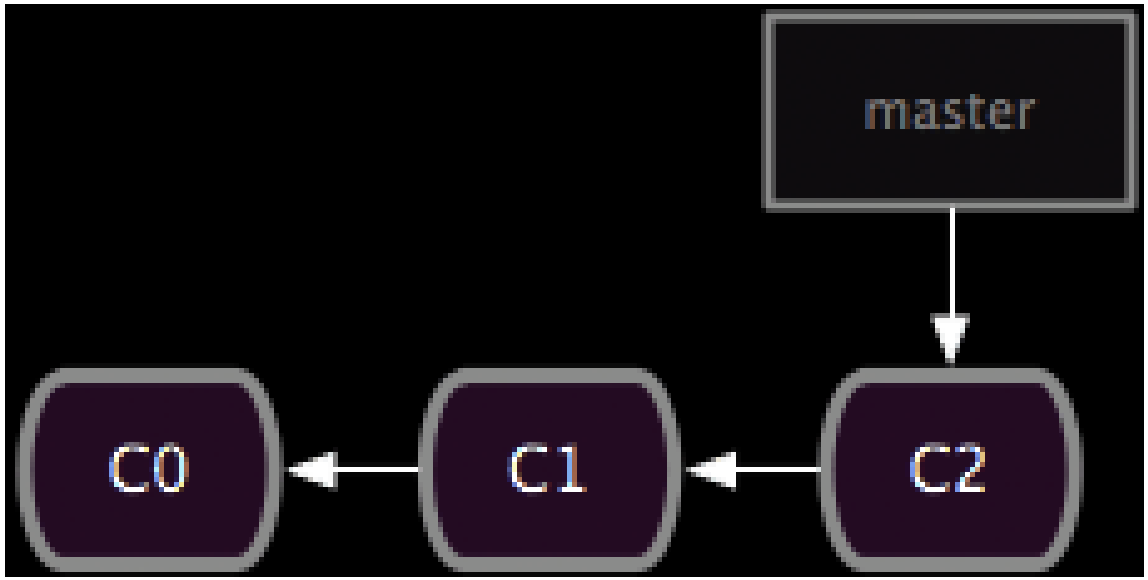
- 版本控制其实就是**控制版本**，帮助我们控制或管理某个事物的不同版本
  - 保留项目的详细历史记录
  - 迅速在各版本之间切换
  - 多人协作，内容不怕被覆盖
  - 谁修改都有记录
  - 回到项目的某个阶段，并恢复数据或文件

版本控制系统

- 一个版本控制系统 Version Control System (VCS)，通常有以下功能：
  - 建立 Repository（储存库），用来保存代码。
  - 有效率的协同开发，方便共享
  - 记录谁改变了什么，在什么时间，并可以写备注
  - Branch(分支)，可因不同情况分别开发
  - Tag(标签) 重要里程碑，以便参照

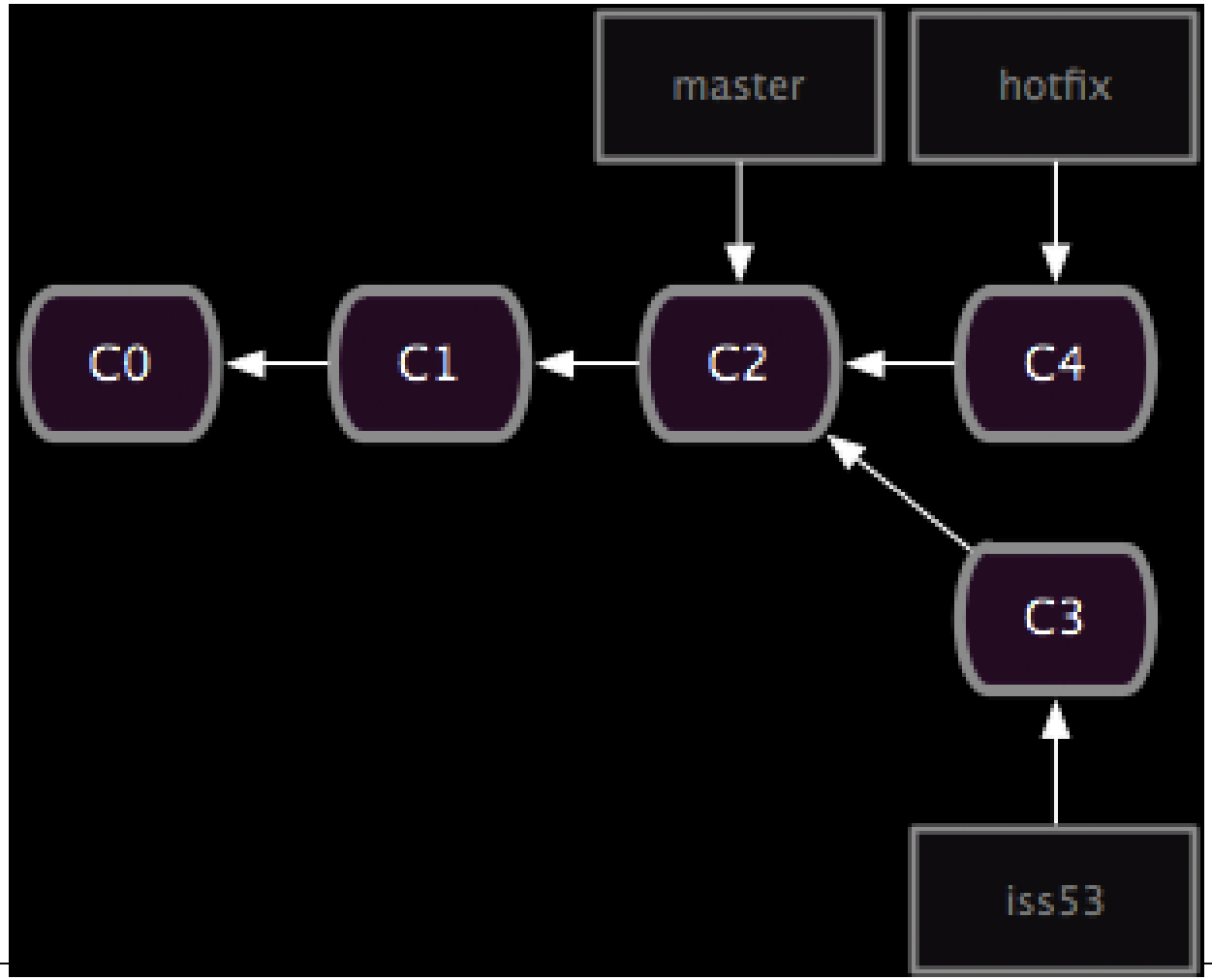
版本控制系统

回溯：可以进行备份，回溯之前的修改
C2回到C0



版本控制系统

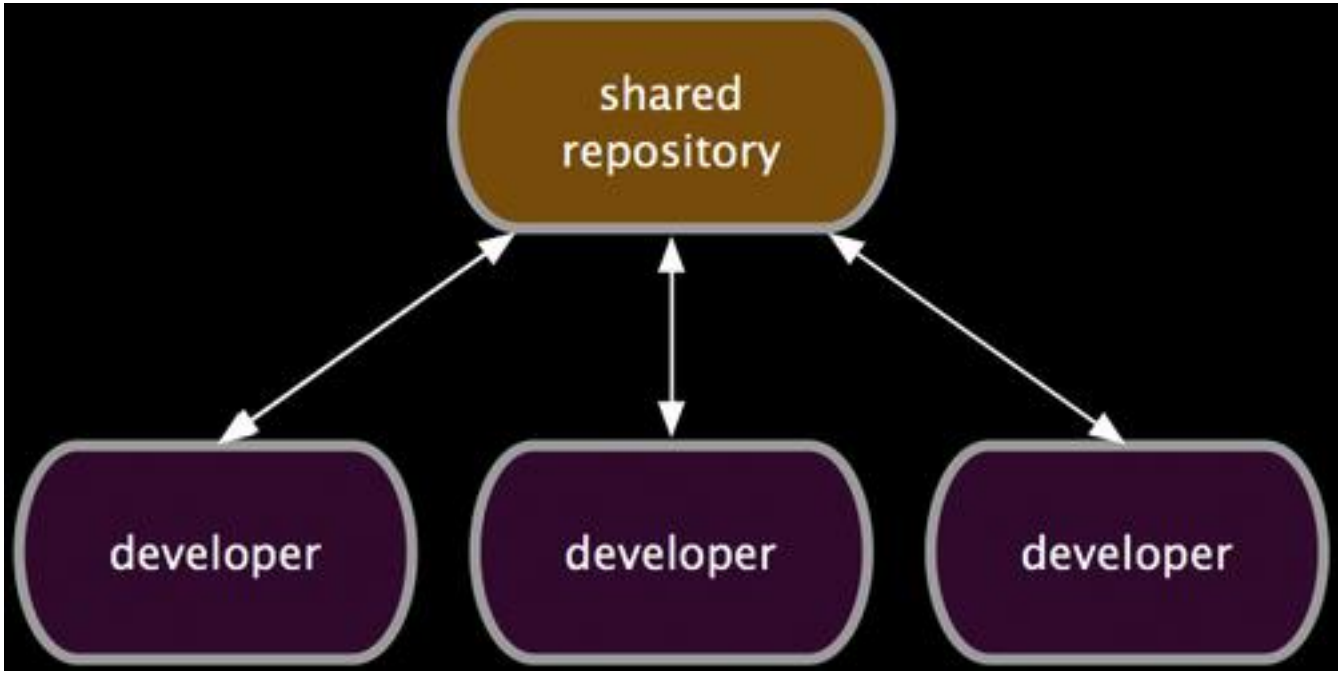
分支：在维护当前版本的同时，开发新版本



http://git-scm.com/book/en/Git-Branching-Basic-Branching-and-M...

版本控制系统

协作：多位成员共同开发



http://git-scm.com/book/en/Distributed-Git-Distributed-Wo

版本控制系统

- 可以对程序代码的修改进行追踪/管理与分发
- 在现代的软件开发中很普遍
- 例子:

older

- Revision Control System (RCS)
- Concurrent Versions System (CVS)
- Subversion (SVN)

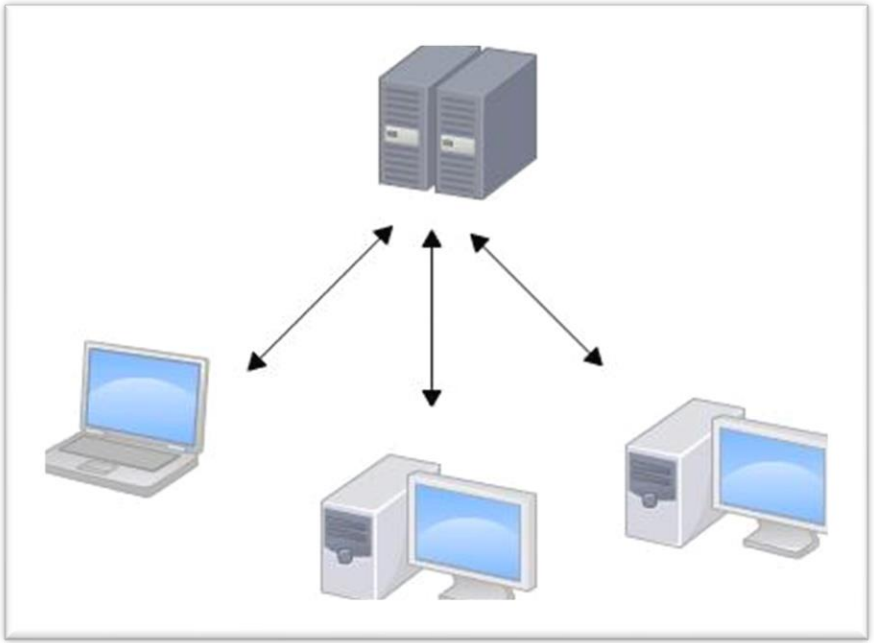
newer

- Git

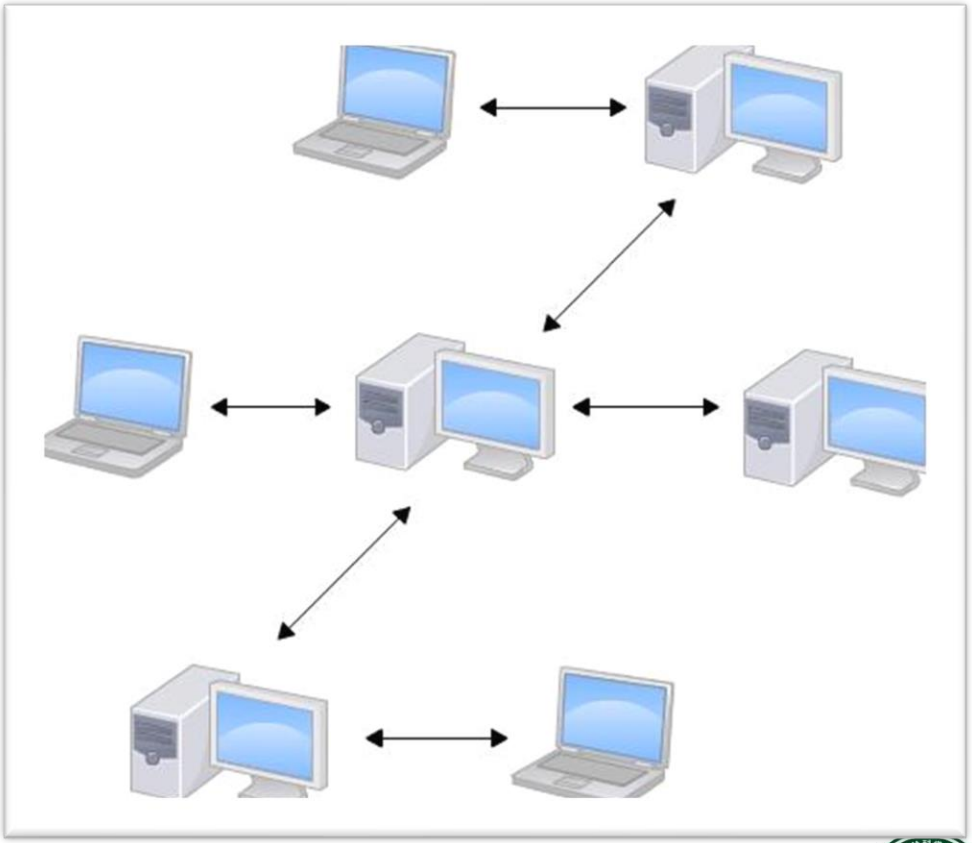
版本控制系统

- 根据原理的不同，又可以分为中心化版本控制系统和分布式版本控制系统

中心化版本控制系统 (eg: SVN)



分布式版本控制系统 (eg: Git)

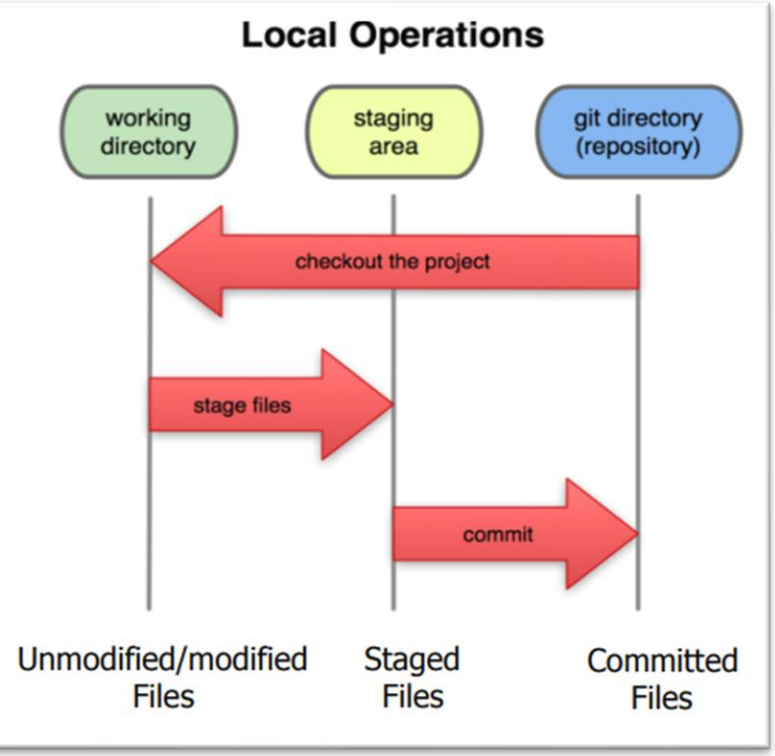


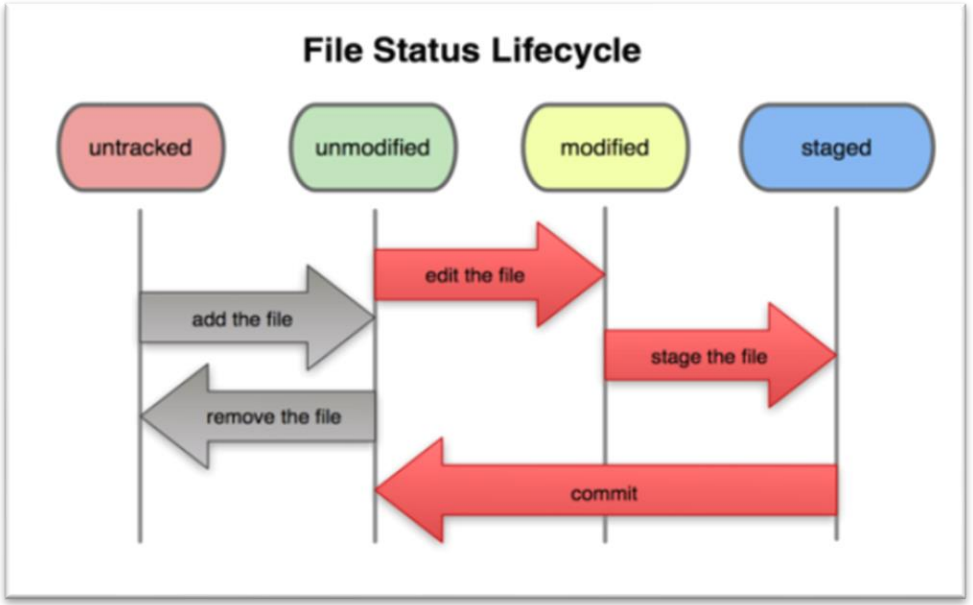
版本控制系统

- 有哪些软件使用了版本控制系统，并且是团队协作开发的?
  - 太多了，几乎所有软件都是
- 举个例子
  - Visual Studio Code: https://code.visualstudio.com/

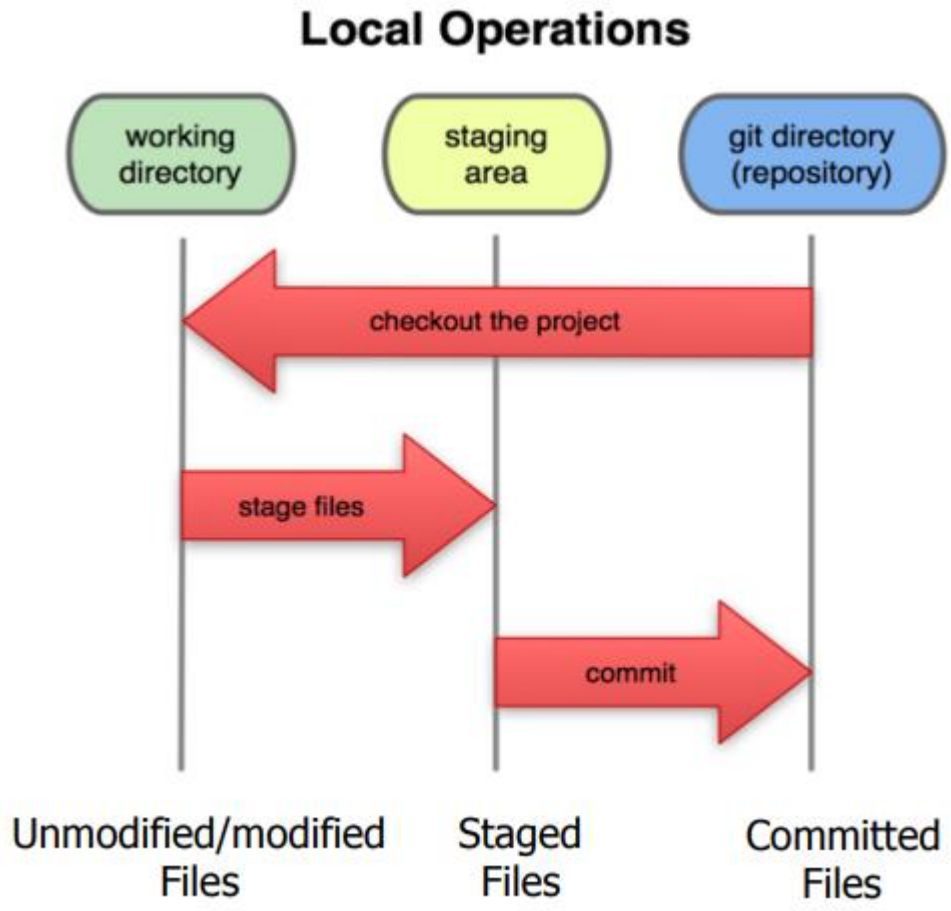
Git简介和团队协作

Git原理


Local Operations
working directory
staging area
git directory (repository)
checkout the project
stage files
commit
Unmodified/modified Files
Staged Files
Committed Files


File Status Lifecycle
untracked
unmodified
modified
staged
add the file
edit the file
stage the file
remove the file
commit

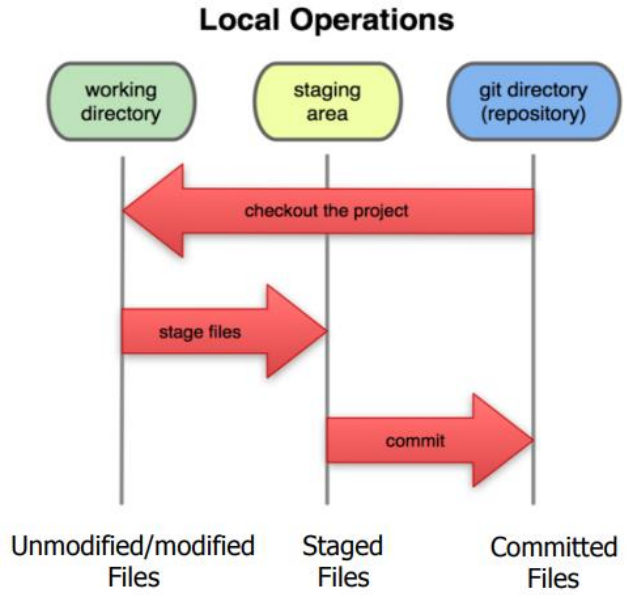
Git原理



名词解释

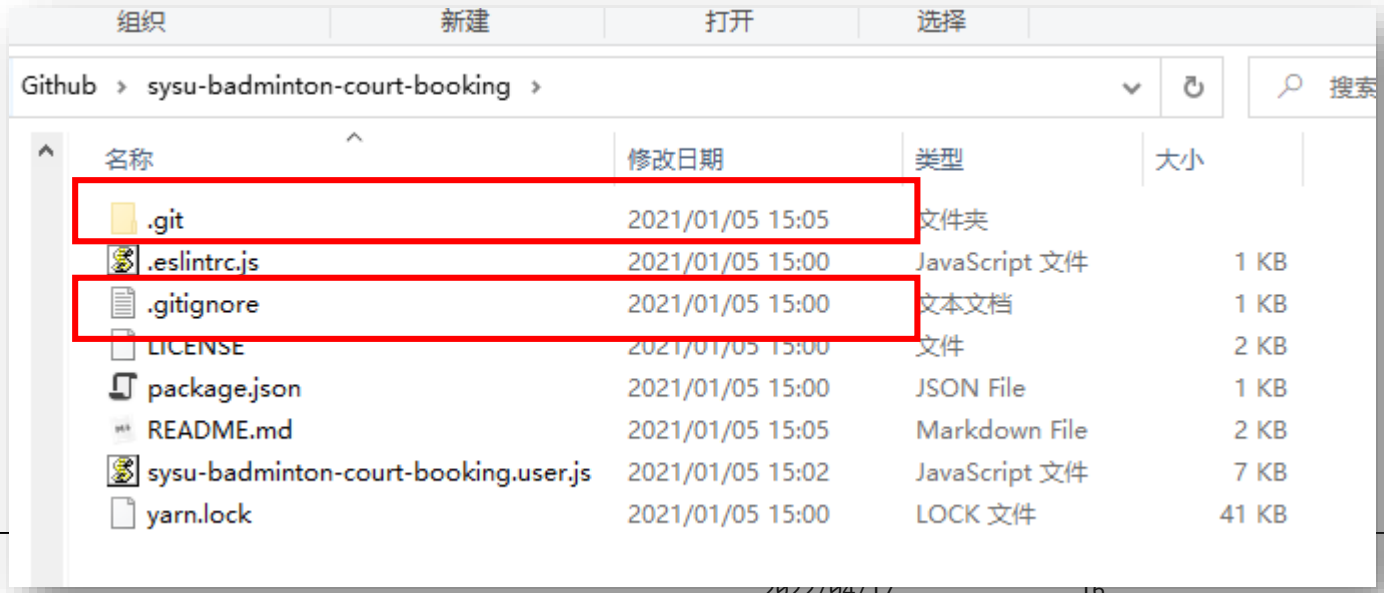
1. Working directory: 工作区
2. Staging area: 暂存区
3. Repository: 版本库
4. Commit: 提交
5. Checkout: 从某个地方复制东西到本地
6. Branch: 分支

Git原理



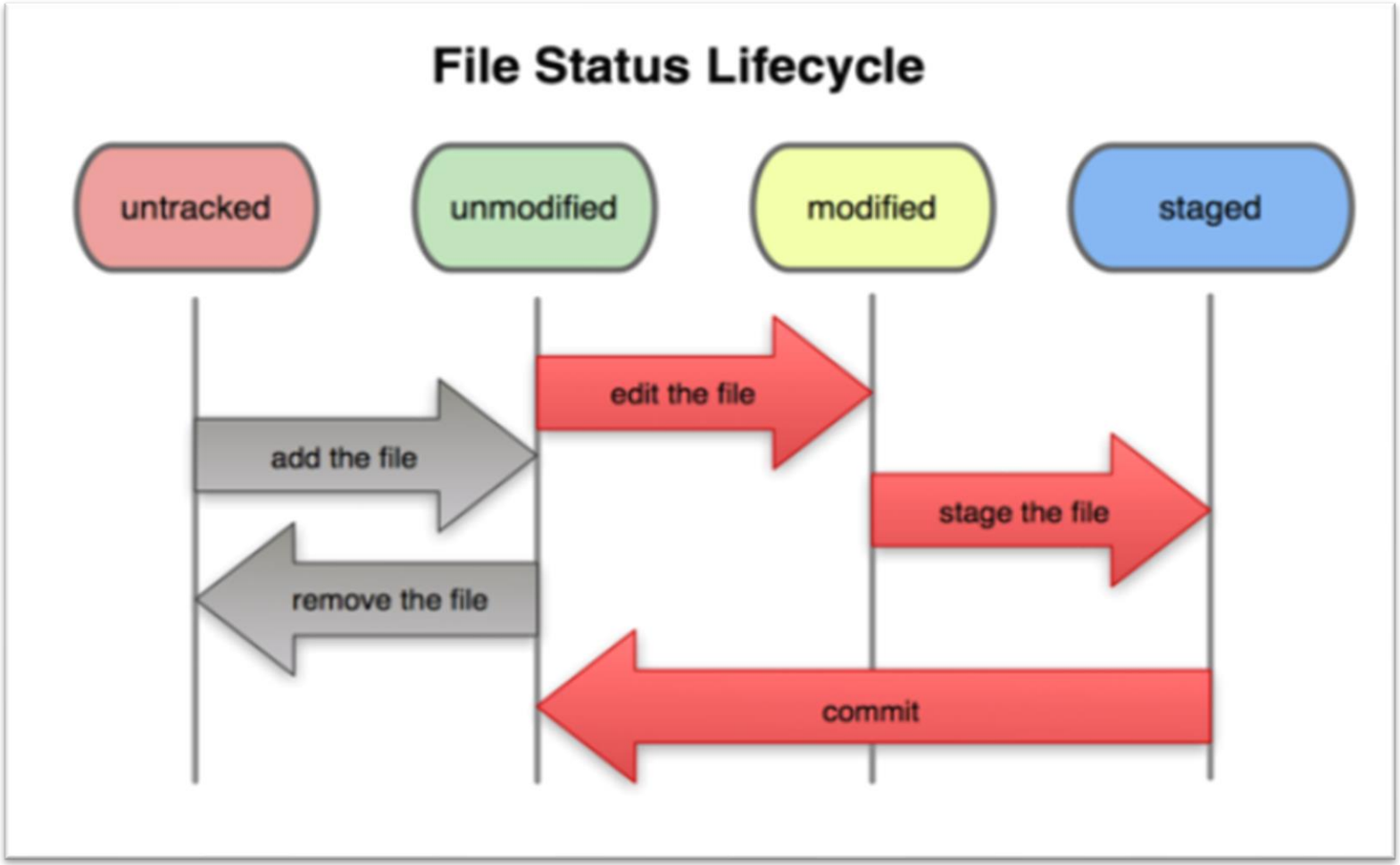
名词解释

1. Working directory: 工作区
2. Staging area: 暂存区
3. Repository: 版本库
4. Commit: 提交
5. Checkout: 从某个地方复制东西到本地
6. Branch: 分支



Git原理

- Git文件状态



Git安装

- 最早Git是在Linux上开发的，很长一段时间内，Git只能在Linux/Unix系统上运行。
- 随着Git的使用逐渐普及，一些开发者也慢慢将Git移植到了Windows平台上。
- 目前Git已经发展为可以在 Windows/ macOS/ Linux/ Unix 上运行的跨平台工具。
- **下载**
  - 可以从 https://git-scm.com/ 获得Git在Windows/macOS/Linux三个操作系统相关的安装包。也可以通过以下方式安装。
- **Windows下的安装**
  - 从 http://git-scm.com/download 上下载window版的客户端，以管理员身份运行后，一直选择下一步安装即可，请注意，如果你不熟悉每个选项的意思，请保持默认的选项

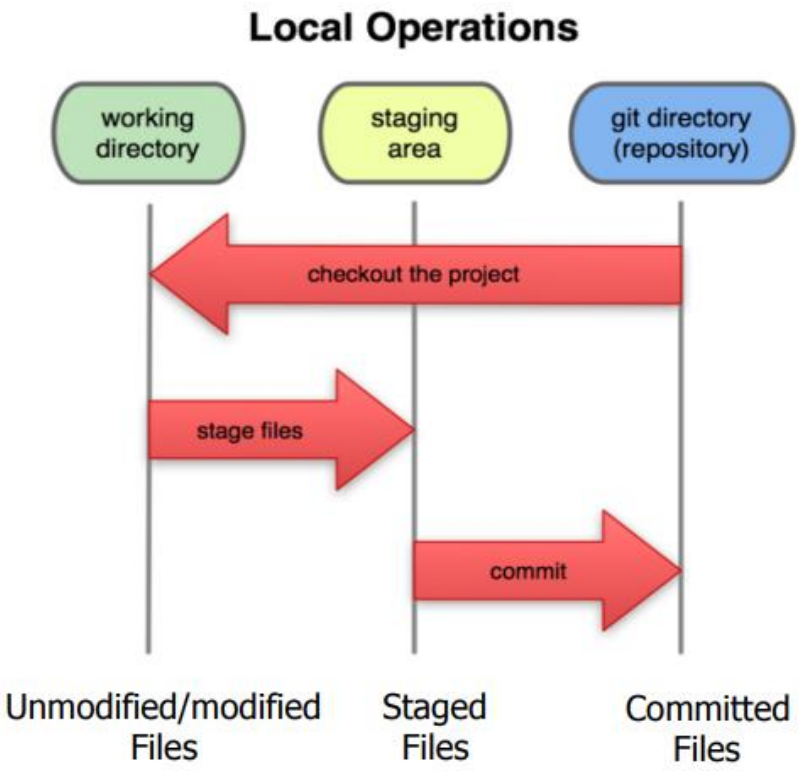
Git基本使用：初始化

: git init

如果之前没有安装过

```
: git config --global user.name "<your name>"
: git config --global user.email "<your email>"
```

Git基本使用：基本信息显示

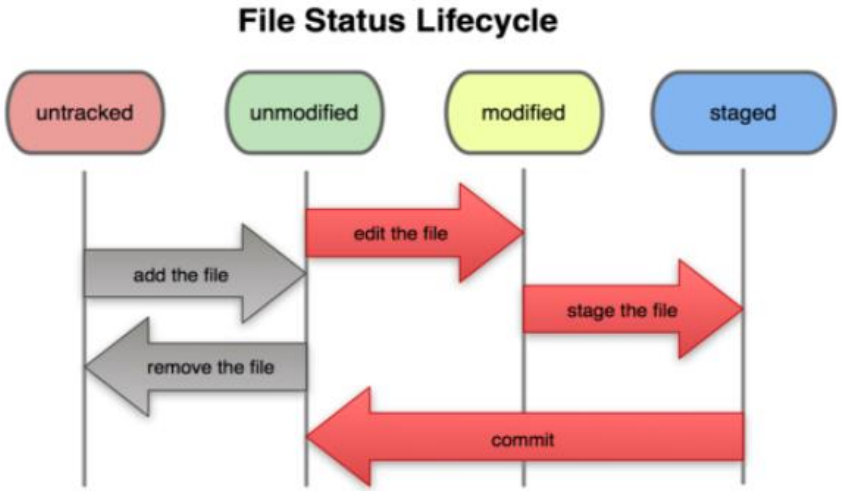


: **git log** 显示commit的历史信息
: **git reflog** 显示所有的提交信息
: **git log --graph** 用图显示commit历史

: **git status** (常用)
1. 显示当前分支名
2. 显示暂存区和HEAD指针指向的提交的差别（路径）
3. 显示工作区与暂存区的差别 （路径）

: **git diff** 显示暂存区和工作区的差别（文件内容）
: **git diff --staged** 显示暂存区和HEAD指针指向的提交的差别（文件内容）
: **git diff HEAD**显示工作区和HEAD指针指向的提交的差别（文件内容）

Git基本使用：文件状态变化



: **git add <file_name>** 暂存文件修改

: **git add .** 暂存所有文件修改

: **git commit –m “commit_msg”** 提交暂存区的修改

: **git rm <file_name>** 从工作区和暂存区删除文件

-r 递归删除（文件夹）

-cached 只从暂存区删除文件

: **git restore .** 将工作区未暂存的修改丢弃

: **git checkout <commit_id>/<branch>** 设置工作区（当branch时设置HEAD指针）

: **git reset –hard <commit_id>/<branch>** 设置HEAD指针和分支指针，清理工作区和暂存区

: **git reset HEAD <filename>** 丢弃暂存区的修改

远程代码仓库

GitHub 是一个面向开源及私有软件项目的托管平台。因为只支持 Git 作为唯一的版本库格式进行托管，故名 GitHub。

Gitee 是开源中国推出的基于 Git 的代码托管服务。目前已经有注册开发者 600 万，托管超过 1500 万代码仓库。

gitee

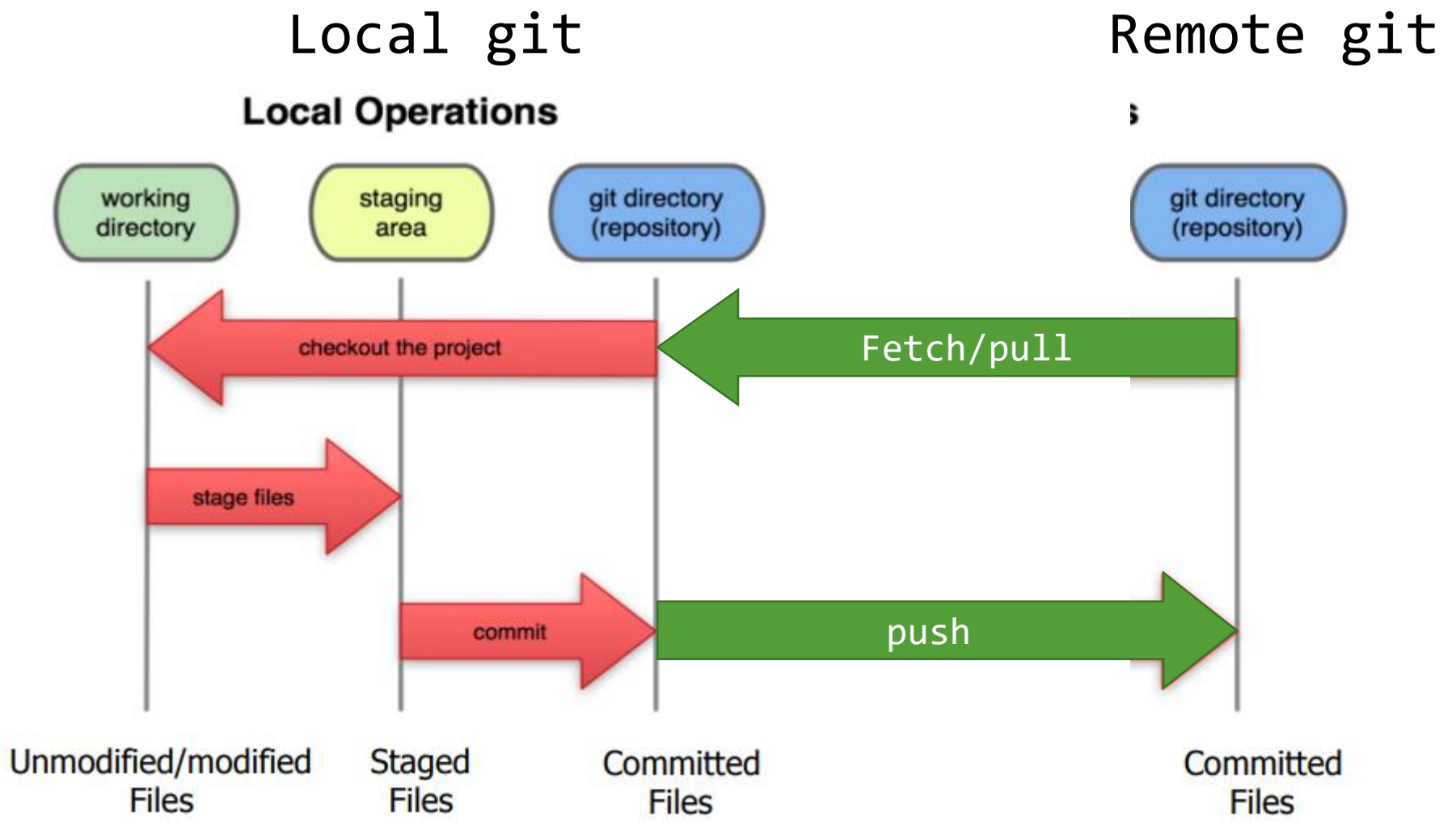
Coding.net 是一个面向开发者的云端开发平台,提供 Git/SVN 代码托管、任务管理、在线 WebIDE、Cloud Studio、开发协作、文件管理、Wiki 管理

CODING
CLOUD DEVELOPMENT

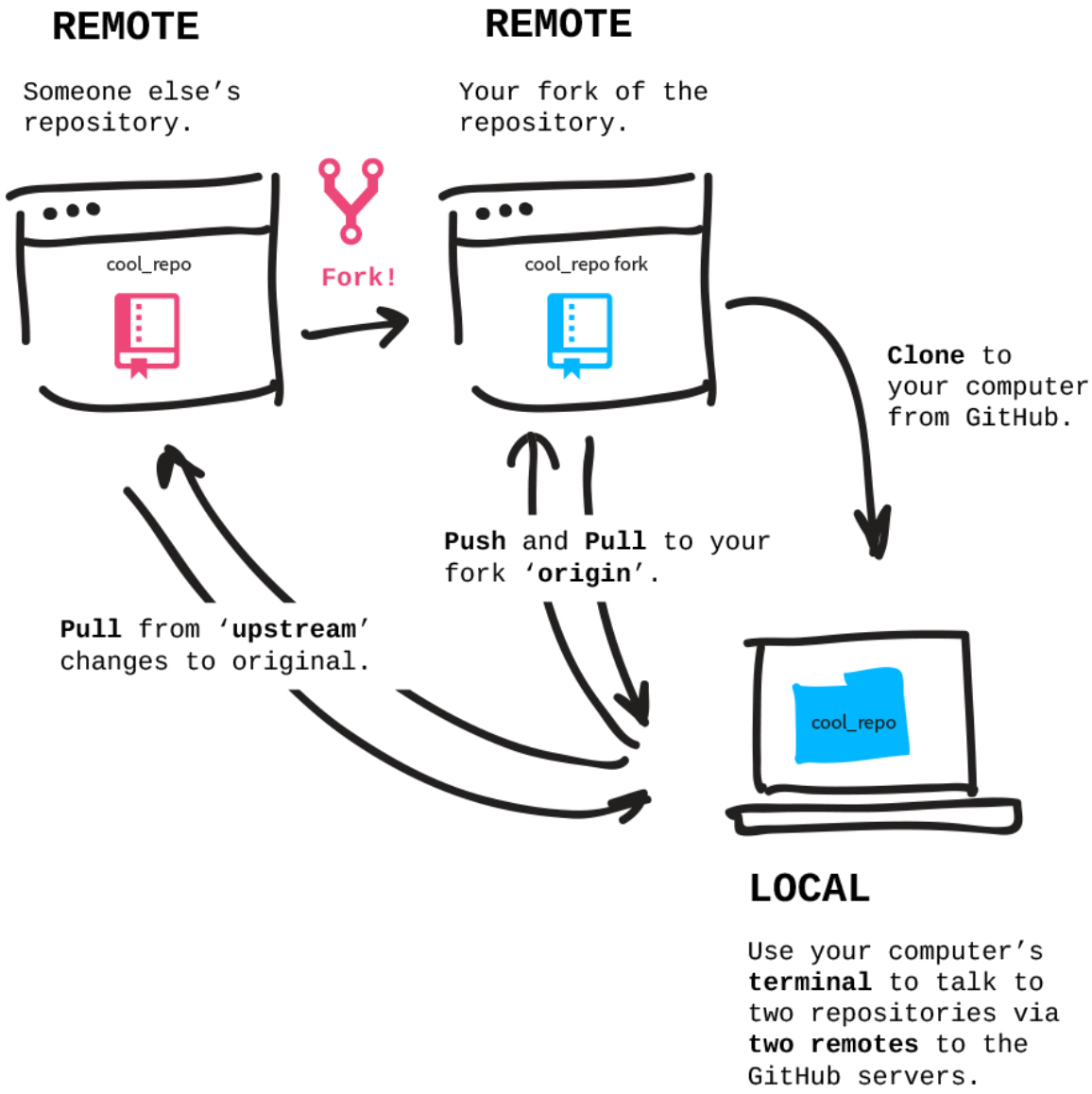

git


github
SOCIAL CODING

本地与远程代码仓库


Local git
Remote git
Local Operations
working directory
staging area
git directory (repository)
git directory (repository)
checkout the project
Fetch/pull
stage files
commit
push
Unmodified/modified Files
Staged Files
Committed Files
Committed Files

Fork（复刻）是什么


REMOTE
Someone else's repository.
REMOTE
Your fork of the repository.
cool_repo
Fork!
cool_repo fork
Clone to your computer from GitHub.
Push and Pull to your fork 'origin'.
Pull from 'upstream' changes to original.
cool_repo
LOCAL
Use your computer's terminal to talk to two repositories via two remotes to the GitHub servers.

1. 当你是项目的管理者时（有github仓库相应权限）

这就比较简单了

1. 如果github上建了项目，clone下来，就可以push origin或pull/fetch origin
2. 如果github上没有项目，建一个空项目，然后和本地的项目关联一下 git remote add origin xxx

2. 当你是项目的参与者时（没有github仓库相应的权限）

Origin: fork的github项目

Local git

Upstream: 真正的项目

总结：Git涉及的概念

- **仓库(repository)**
  - 在 Git 的概念中，仓库，就是**存在.git目录的那个文件夹内的所有文件**，包括隐藏的文件
  - 所以，我们如果想将某个文件夹当做一个Git仓库，你可以在那个文件夹下通过终端(Window为Cmd或者PoewrShell或者Bash)来执行：`git init`
- **版本(version)**
  - 在Git中，计数基础是提交，即我们常说的Commit，我们每做一点更改便可以产生一次提交
  - 当提交累计起来，可以作为产品定型时，就在当前的Commit上**打上一个标记**，将这个标记我们称之为版本多少多少

总结：Git涉及的概念

- **分支(branch)**
  - 分支功能解决了正在开发的版本与上线版本稳定性冲突的问题在Git使用过程中，我们的默认分支一般是Master，当然，这是可以修改的
  - 我们在Master完成一次开发，生成了一个稳定版本，那么当我们需要添加新功能或者做修改时，只需要新建一个分支，然后在该分支上开发，完成后合并到主分支即可
- **提交(commit)**
  - Git对于版本的管理其实是对于提交的管理，在整个Git仓库中，代码存在的形式并不是一份一份的代码，而是一个一个的提交
  - 关于Git的储存方式:Git是仅仅只储存有修改的部分，并不会储存整个文件

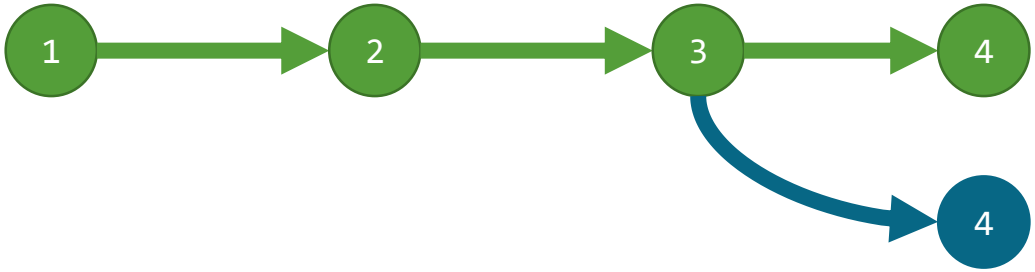
总结：Git涉及的概念

- **同步(sync)**
  - 同步，也可以称之为拉取pull，在Git中是非常频繁的操作
  - 所以，为了保证代码一致性，尽可能的在每次操作前进行一次同步操作，具体的为在工作目录下执行如下命令:`git pull origin master`
- **推送(push)**
  - 和拉取一样，也是一个非常频繁的操作，当你代码有更新时，你需要更新到远程仓库，这个动作被称之为推送
  - 如果远程仓库存在你本地仓库没有的更新，则在推送前你需要先进行一次同步，如果你确定你不需要远程的更新，则在推送时加上 `-f` 选项，则可以强制推送
  - 推送代码: `git push origin master`
  - 强制推送: `git push origin master -f`

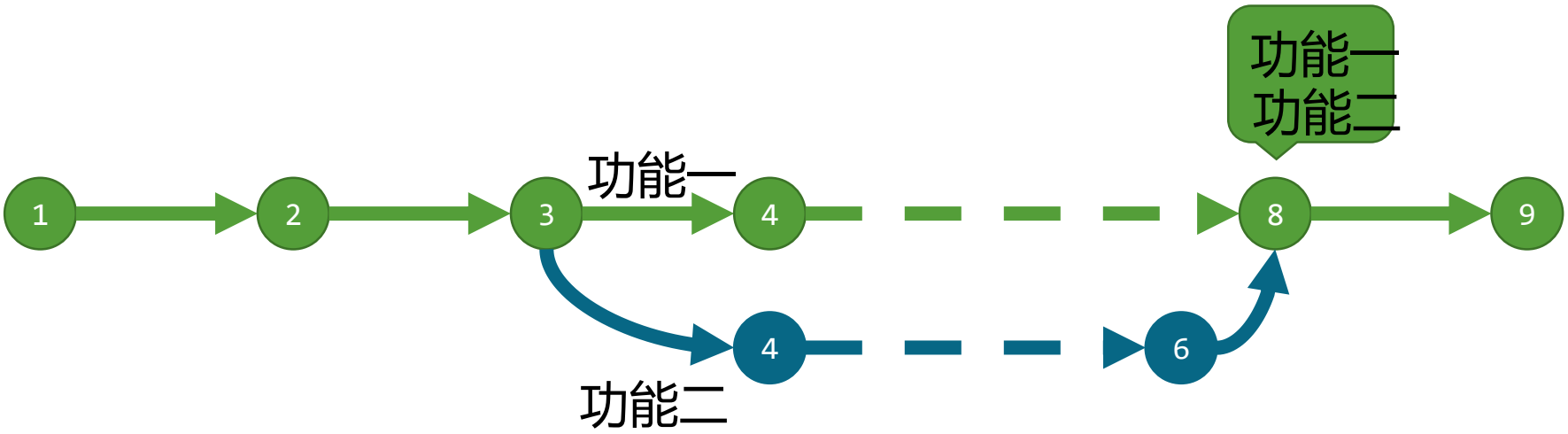
Git分支

Git分支

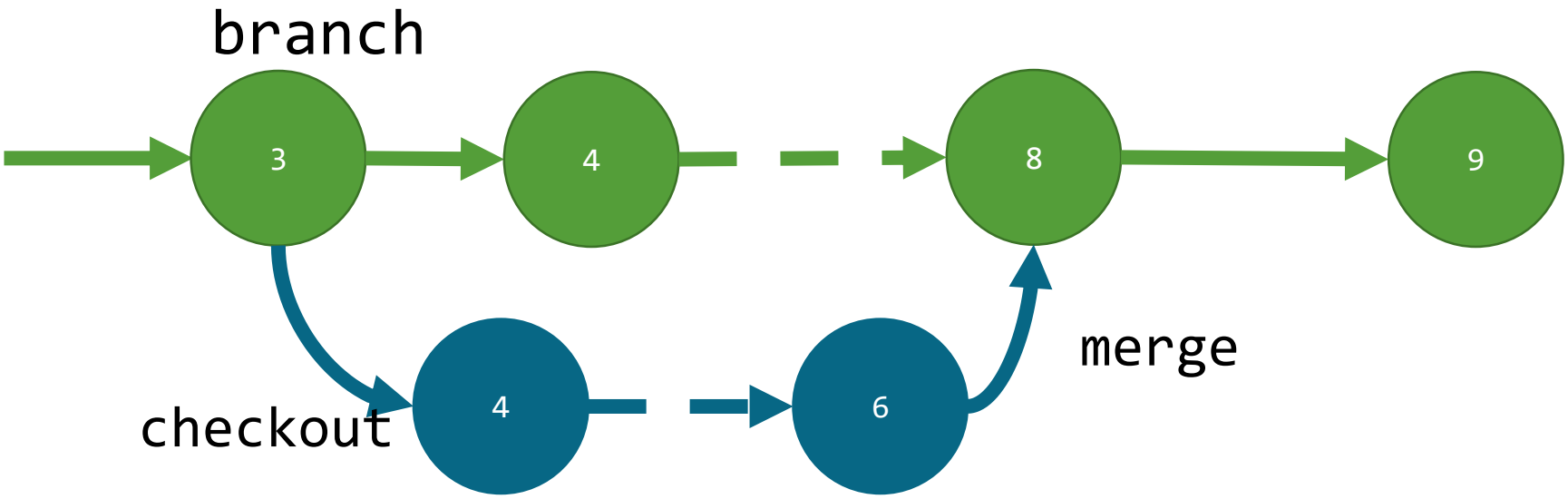
Git分支

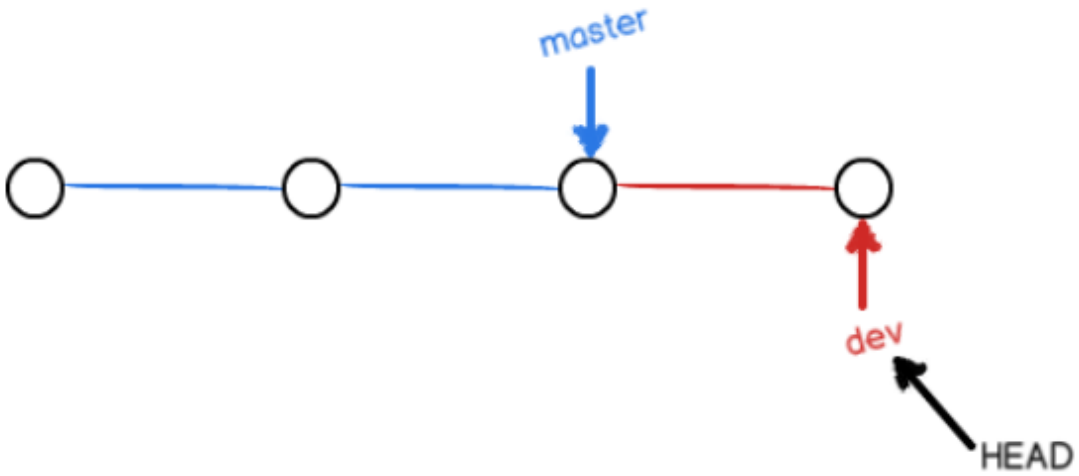


Git分支



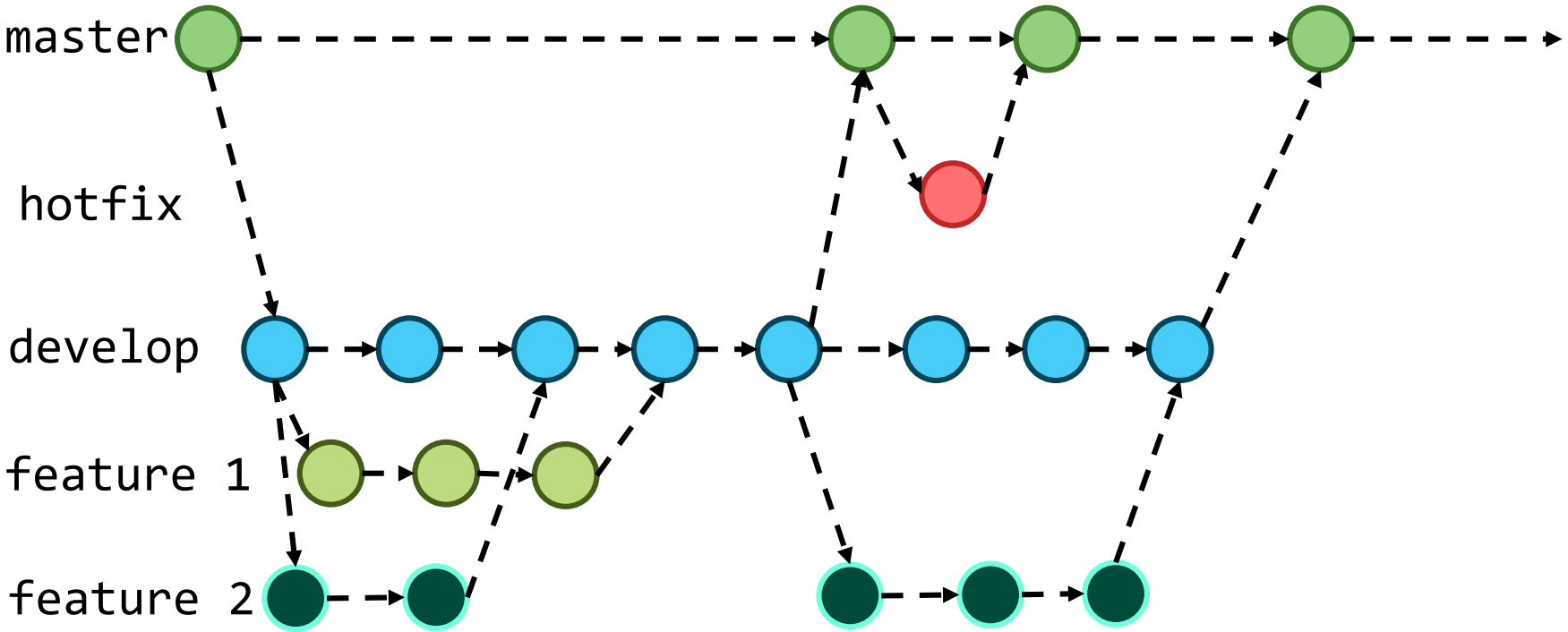
Git分支





: **git branch xxx** 新建分支
: **git checkout xxx** 切换分支
: **git merge xxx** 合并分支到当前分支
: **git stash** 储存工作区和暂存区的修改
: **git stash list**
: **git stash apply/pop** 使用/使用并删除

Git flow开发流程



Git分支命名规范

- 所有使用了本规范的项目，必须严格规范操作，否则不予以合并代码、提测、打包上线等后续操作。

名称	说明	命名规范	命名示例	合并目标	合并操作
master	线上稳定版本	master	master	--	--
release	待发布分支，下个版本需上线的版本	release/xxx	release/v1.0.0	master	merge request
develop	当前正在开发的分支	develop	develop	master	merge request
feature	功能分支，每个功能需分别建立自己的子分支	feature/版本号-功能名	feature/v1.0.0-Login	develop	merge request
hotfix	紧急修复分支	hotfix/xxx	hotfix/v1.0.1	master/develop	merge request

More information

- http://git-scm.com/about for introduction to **Git**
- http://git-scm.com/book for the nitty gritty
- http://gitref.org for a basic how-to guide
- https://github.com/ for overview of **Github**
- https://try.github.io for an interactive trial

开源软件与开源精神

开源软件与开源精神

- 截止2015年，世界上95%的主流IT公司都开始使用开源平台开发软件应用

http://www.neuni-lab.com/%E5%8D%81%E5%B9%85%E5%9B%BE%E5%91%8A%E8%AF%89%E4%BD%A0%E4%B8%BA%E4%BB%80%E4%B9%88%E5%BC%80%E6%BA%90%EF%BC%88open-source%EF%BC%89%E5%9C%A8%E6%88%91%E4%BB%AC%E7%9A%84%E7%94%9F%E6%B4%BB/

什么是开源软件

- 开源软件就是把**软件程序与源代码文件**一起打包提供给用户，用户既可以不受限制地使用该软件的全部功能，也可以根据自己的需求修改源代码，甚至编制成衍生产品再次发布出去。
- 用户具有使用自由、修改自由、重新发布自由和创建衍生品自由
- 这正好符合了黑客和极客对自由的追求，因此开源软件在国内外都有着很高的人气，大家聚集在开源社区，共同推动开源软件的进步。

开源软件的优点

优点	说明
低风险	使用闭源软件无疑是把命运交给他人，一旦封闭的源代码没有人来维护，你将进退维谷；而且相较于商业软件公司，开源社区很少存在倒闭的问题。
高品质	相较于闭源软件产品，开源项目通常是由开源社区来研发及维护的，参与编写、维护、测试的用户量众多，一般的 bug 还没有等爆发就已经被修补。
低成本	开源工作者都是在幕后默默且无偿地付出劳动成果，为美好的世界贡献一份力量，因此使用开源社区推动的软件项目可以节省大量的人力、物力和财力。
更透明	没有哪个笨蛋会把木马、后门等放到开放的源代码中，这样无疑是把自己的罪行暴露在阳光之下。

典型的开源软件

软件	说明
Linux	Linux 是一款开源的操作系统，它的内核由多名极客共同维护。Linux 是开源软件的经典之作、代表之作、巅峰之作。
Apache	世界使用排名第一的 Web 服务器软件。
MySQL	世界上最流行的关系型数据库，适合中小型网站。
Firefox	火狐浏览器。在 Chrome 推出之前，Firefox 几乎是最快速的浏览器，直到现在也是 Web 开发人员的调试利器。
OpenOffice	套跨平台的办公软件套件，类似 Microsoft Office。
GCC	C语言/C++编译器。
Java、PHP、Python	开源的编程语言。

- 什么是开源精神？
  - 开源精神是造福大众的开源精神，而不是"大众服务你，大众造福你"的开源精神。
  - 开源精神是知识共享，互惠互利的开源精神，而不是"知识垄断，弱者恒弱，强者恒强"的开源精神。
  - 开源精神是扶危济困，损有余而补不足，先进带动后进的开源精神，而不是"劫贫济富，以不足而奉有余，先进压制后进"的开源精神。
  - 开源精神是公正公开，维护开发者与使用者利益的开源精神，而不是"闭塞停滞，众乐乐不如独乐乐，不管他人死活"的开源精神。
  - 开源精神，应该是**自由，创新，团结，互助，友爱，积极，进取**的开源精神，而不是垄断，封闭，孤立，排异，冷漠，畏缩，退步的开源精神。

- 什么是开源精神？
  - 开源精神是造福大众的开源精神，而不是"大众服务你，大众造福你"的开源精神。
  - 开源精神是知识共享，互惠互利的开[illegible]而不是"知识垄断，弱者恒弱，强[illegible]源精神。
  - 开源精神是[illegible]不足，先进带动后进的[illegible]劫贫济富，以不足而奉[illegible]后进"的开源精神。
  - 开[illegible]公正公开，维护开发者与使用者利益的开源精神，而不是"闭塞停滞，众乐乐不如独乐乐，不管他人死活"的开源精神。
  - 开源精神，应该是**自由，创新，团结，互助，友爱，积极，进取**的开源精神，而不是垄断，封闭，孤立，排异，冷漠，畏缩，退步的开源精神。

人人为我，我为人人

什么是开源精神?

- 开源精神是 Free & Open Source，意思是自由及开放源代码，Free指的是自由不是免费。
- GNU Free Software是这么定义其所谓的“自由”的：
  1．出于任何目的使用该软件的自由；
  2．研究、修改该软件以使其实现你的目的的自由；源码的公开是实现此项的前提条件；
  3．传播该软件的拷贝以使其他人受益的自由；
  4．传播你所修改过的软件拷贝的自由。

https://www.zhihu.com/question/20559450

什么是开源精神?

- 开源和商业化是否矛盾？自由呢?
  - 从“精神”角度讲，**开源和商业化不矛盾**，和自由也不矛盾。不过自由和商业应该是矛盾的，核心问题在于：我如果根据自由许可拿到了一款商业软件，那么我可以无限的卖它的拷贝或衍生物。
- **开源≠自由≠非商业。**

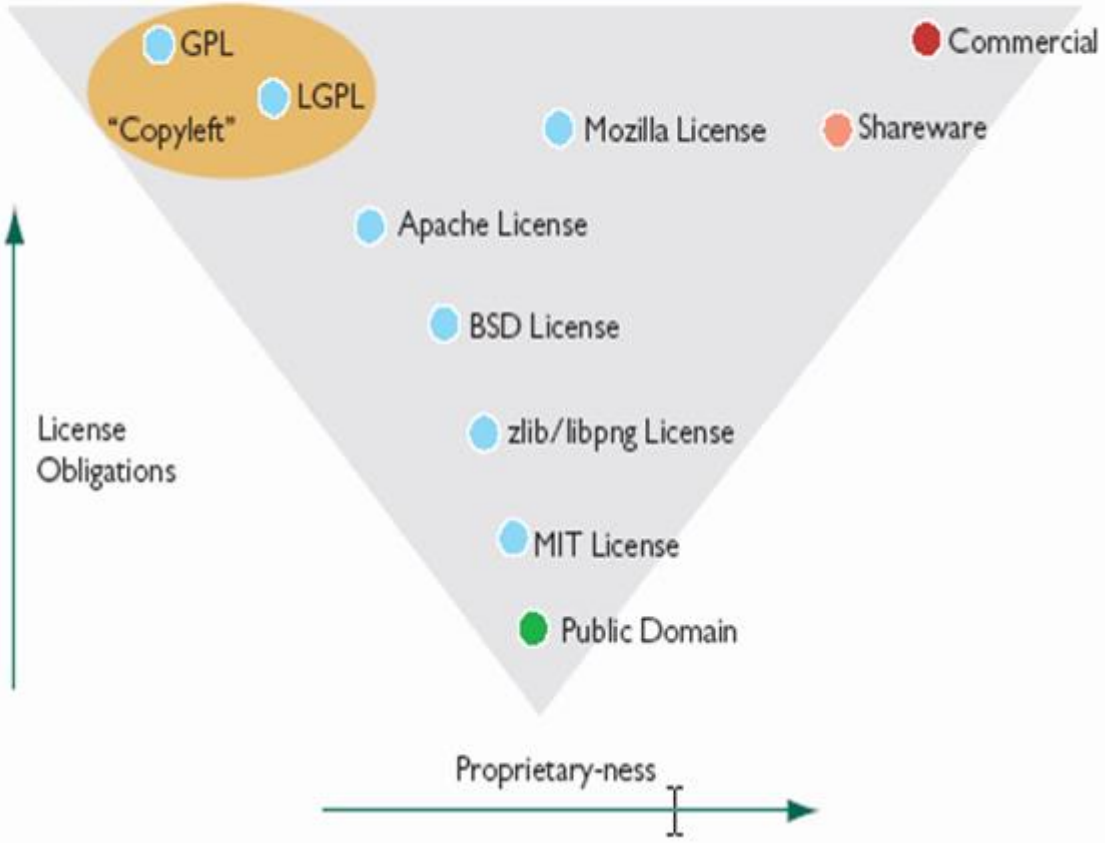
https://www.zhihu.com/question/20559450/answer/140647885

软件许可总览

SOFTWARE LICENSE OVERVIEW

In this overview of software licenses, the three corners of the triangle represent the primary license types: GPL, commercial and public domain. As you move away from the corners, you see license variations. GPL and commercial licenses carry the most license obligations but are at opposite ends of the proprietary scale regarding source code ownership and availability. Public domain software has no license at all.

Source: CSC



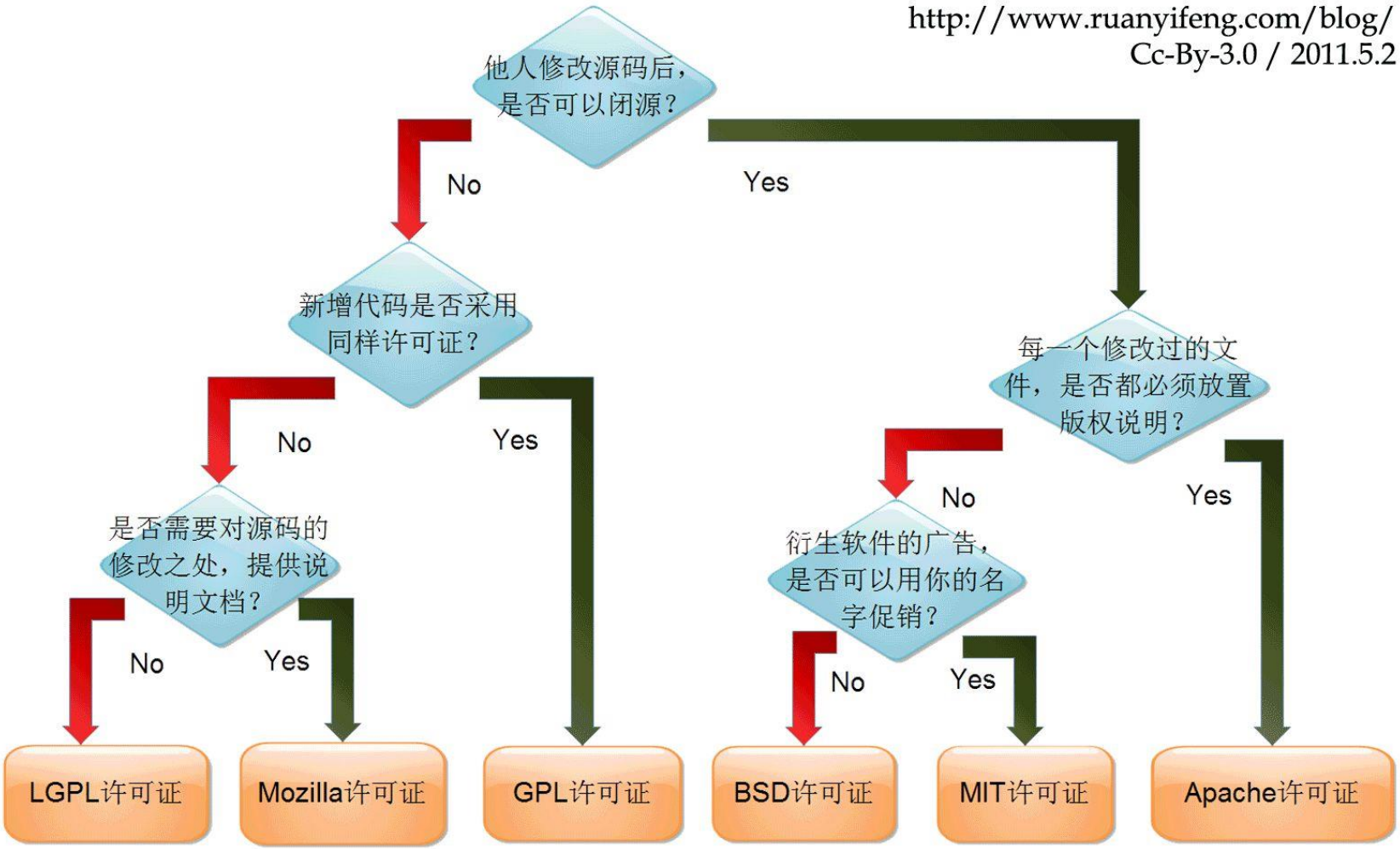
商业软件许可的一般性内容

- 1．协议许可的权利（安装的机器台数、并发用户数、可享有的支持和服务、可以修改的范围）
- 2．协议规定的约束和限制（安装和使用、转让和复制、出租或抵押、逆向工程）
- 3．有限担保和免责声明（免赔、连带责任、自担使用风险）
- 4、有关 件产品 授权包含的服务范围，付费方式等

BSD和GPL

- 最常见的开源软件和自由软件许可证是BSD和GPL
- 有评论说，“BSD保证了开发者的自由，GPL保证了使用者的自由”
  - 即GPL保证了使用者对程序及其今后的升级版的修改和免费使用的自由，BSD不能保证基于BSD开发的程序今后的所有改进版都能够允许修改和免费使用
  - BSD保证了开发者能够任意处置程序代码，包括封闭代码和商品化
  - GPL是基于用户的角度设计的授权，BSD是基于开发者的角度设计的授权

如何选择开源许可证?

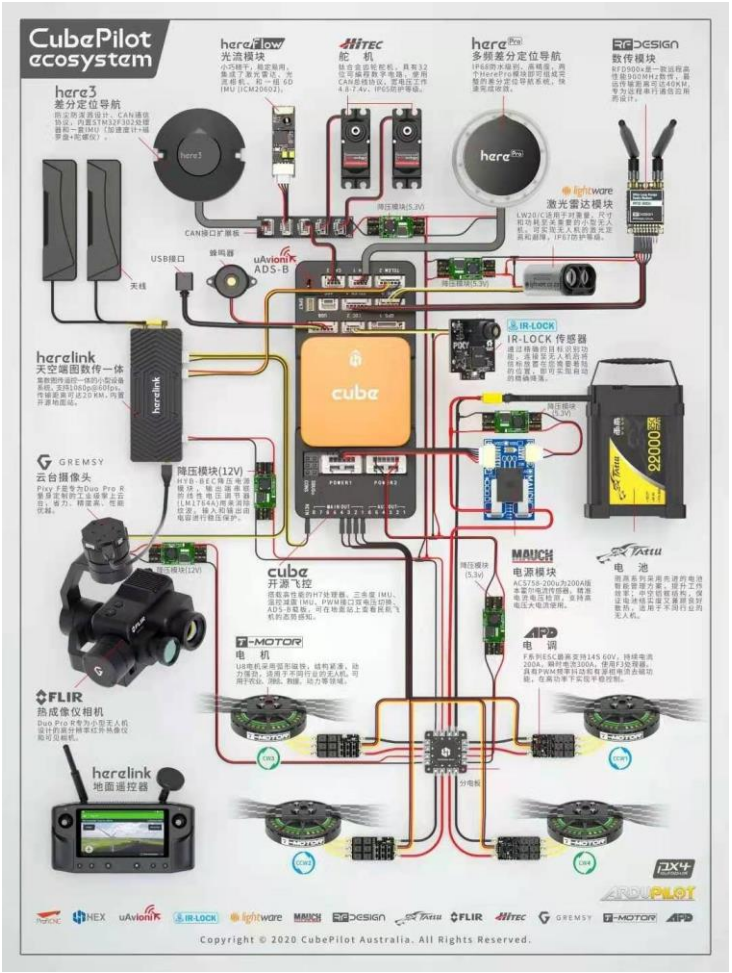

http://www.ruanyifeng.com/blog/
Cc-By-3.0 / 2011.5.2
他人修改源码后，是否可以闭源？
No
Yes
新增代码是否采用同样许可证？
No
Yes
是否需要对源码的修改之处，提供说明文档？
No
Yes
LGPL许可证
Mozilla许可证
GPL许可证
每一个修改过的文件，是否都必须放置版权说明？
No
Yes
衍生软件的广告，是否可以用你的名字促销？
No
Yes
BSD许可证
MIT许可证
Apache许可证

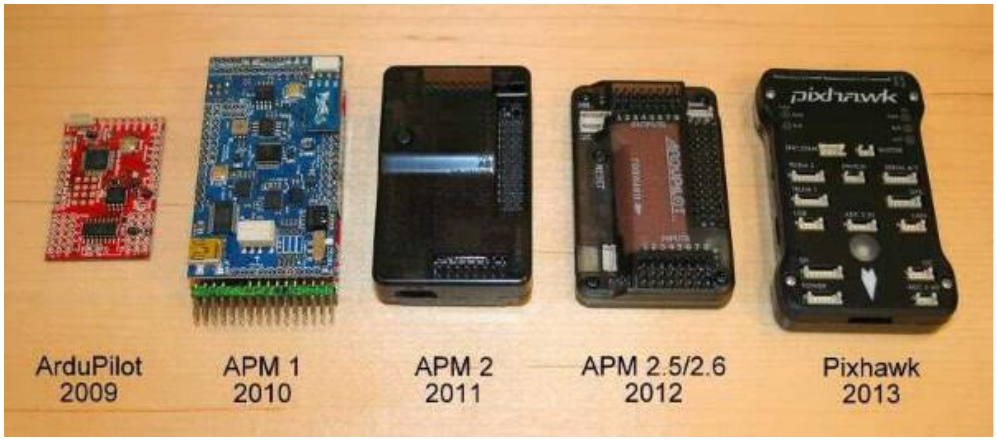
OFSS(Open /Free Source software)的启示

- 大力推广自由软件、开源软件的研究和应用，能帮用户节约开支，可以了解国际IT行业发展前沿和开发团队
- 参与先进计算机技术的开发、研究和应用，获得宝贵的工程和研究经验
- 著名的OFSS软件
  - Apache Web server系列
  - Linux操作系统
  - MySQL数据库
  - Moodle学习管理系统

开源项目介绍：Mission planner

- 开源无人机与地面站Mission planner



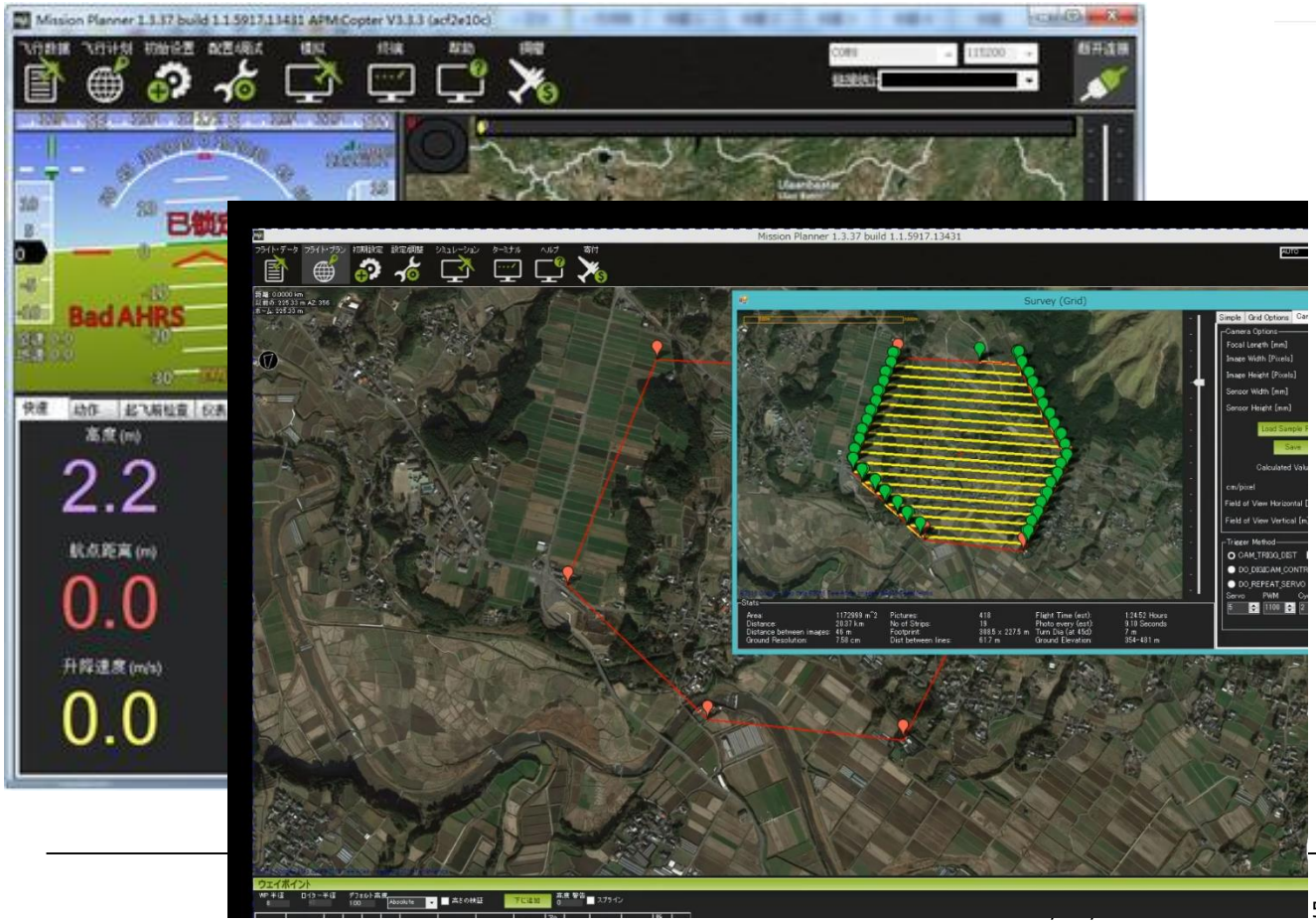


https://dronechina.net/t/topic/414?from=timeline

开源项目介绍：Mission planner

- The Mission Planner, created by Michael Oborne, does a lot more than its name. Here are some of the features:
  - Point-and-click waypoint/fence/rally point entry, using Google Maps/Bing/Open street maps/Custom WMS.
  - Select mission commands from drop-down menus
  - Download mission log files and analyze them
  - Configure autopilot settings for your vehicle
  - Interface with a PC flight simulator to create a full software-in-the-l
  - Run its own SITL simulation of many frames types for all the ArduPilot

无人机/无人车/无人船
规划/控制/调试/仿真……



Michael Oborne
meee1

Author of Mission Planner

Follow

142 followers · 0 following · 409

Albany, Australia

Highlights

Arctic Code Vault Contributor

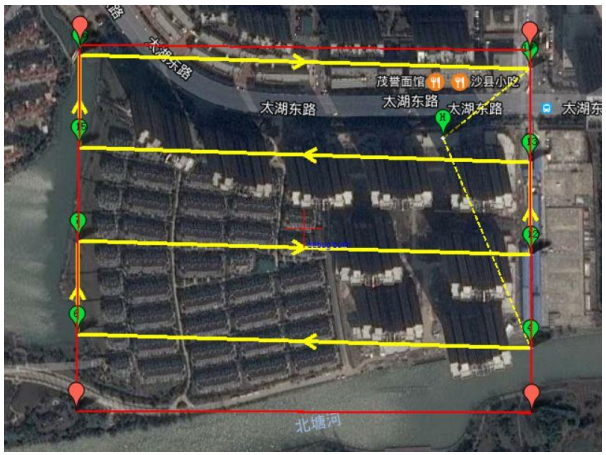
参与开源项目！

- 协助维护世界知名开源无人机地面站软件Mission Planner，将航线规划算法融入该软件中
- 发现了bug，并且自己解决了
- 给软件项目组发起pull request，主编同意Pull

Hi ,
I'm opening the pull request of fixing missing flight strip when doing segmentation in GridUI.
This is the one that we've talked about during the Ardupilot Unconference last week. **@meee1**
When planning a survey mission with 4 strips and 2 segments in 1.3.50, the latter segment missed a flight strip.

I've tested in some other conditions, and as far as I'm concerned, this change will not lead to other bugs.
Shuhang Zhang

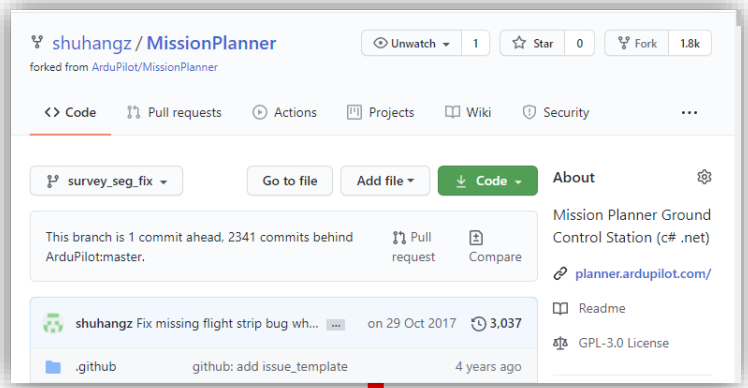
The bug

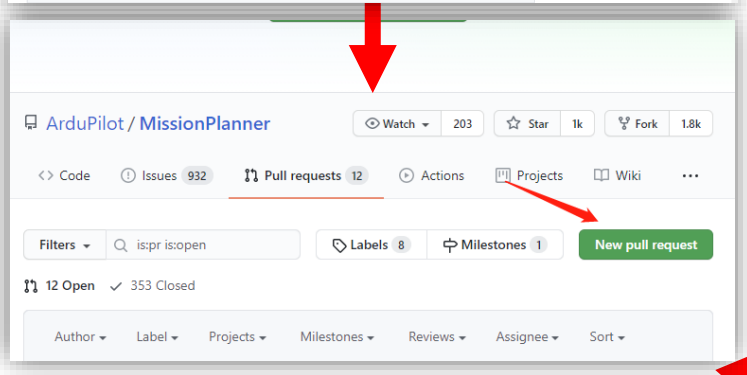


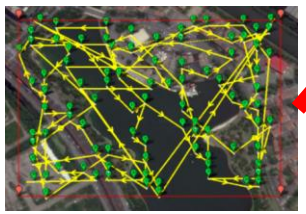
After the fixing.

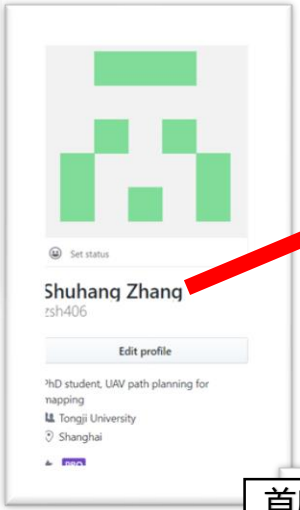
参与开源项目！

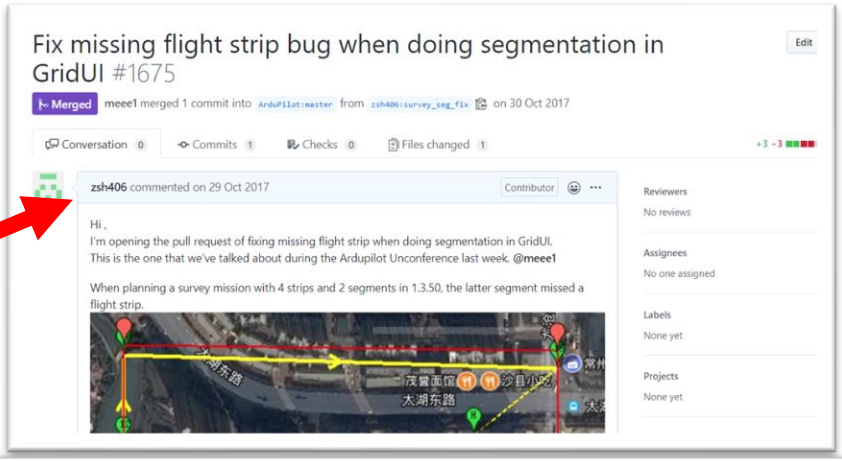
- 协助维护世界知名开源无人机地面站软件Mission Planner，将航线规划算法融入该软件中











https://github.com/ArduPilot/MissionPlanner/pull/1675

Git简介和团队协作

大纲

- Git简介
- Git基础
- Git操作
- Git版本管理
- GIT分支管理在项目中的实践

Git 简史

- 2002年，linux项目组开始启用分布式版本控制系统BitKeeper来管理和维护代码。
- 2005年的时候，开发BitKeeper的商业公司同 Linux 内核开源社区的合作关系结束，他们收回了免费使用BitKeeper的权力
- April 5, 2005 – Linus发布首个git版本
- June 15, 2005 - Git用作Linux源码版本控制

Git创建时的目标

1. 速度
2. 简单的设计
3. 对非线性开发模式的强力支持（允许上千个并行开发的分支）
4. 完全分布式
5. 有能力高效管理类似 Linux 内核一样的超大规模项目（速度和数据量）

Git简介

Git是一个快速，开源，分布式的版本控制系统，在开源和协作编程社区很快取代了svn。

- 可以利用它来追踪项目中的文件
- 可以和合作伙伴共享版本历史状态
- 可以将合作伙伴的工作和你的工作进行合并
- 可以对整个工程或某些文件跟历史版本进行比较或者恢复到早期的某个版本。

Git简介

- 它的特点在于：
  1. 开源
  2. 高速
  3. 节省大量空间
  4. 灵活、简洁、高效的分支管理
- 它能最大限度地发挥多人协同并发编程的效能，让分支管理更快速，版本管理更简单

GIT简介-开源

- GIT源码地址：git-scm.com/download
- GIT许可证：GNU通用公共许可证（GNU General Public License）

大纲

- Git简介
- Git基础
- Git操作
- Git版本管理
- GIT分支管理在项目中的实践

离线

- Git是完全的分布式处理，它可以离线工作。跟SVN完全不同，Git的所有操作几乎不需要网络连接，包括历史回顾，差异显示和提交

快速

- Git比其他的VCS工具要快很多，因为git绝大部分是离线操作，对网络依赖小

◆time git clone ssh://gufeiyong@scm.hz.netease.com:2222/backend/datastream.git >/dev/null

- real 0m26.559s
- user 0m2.568s
- sys 0m1.028s

◆time svn co https://svn.hz.netease.com/svn/datastream >/dev/null

- real 3m14.684s
- user 1m1.156s
- sys 0m20.897s

占用空间小

- git比较节省空间。Django项目为例。

  44M ./django-git

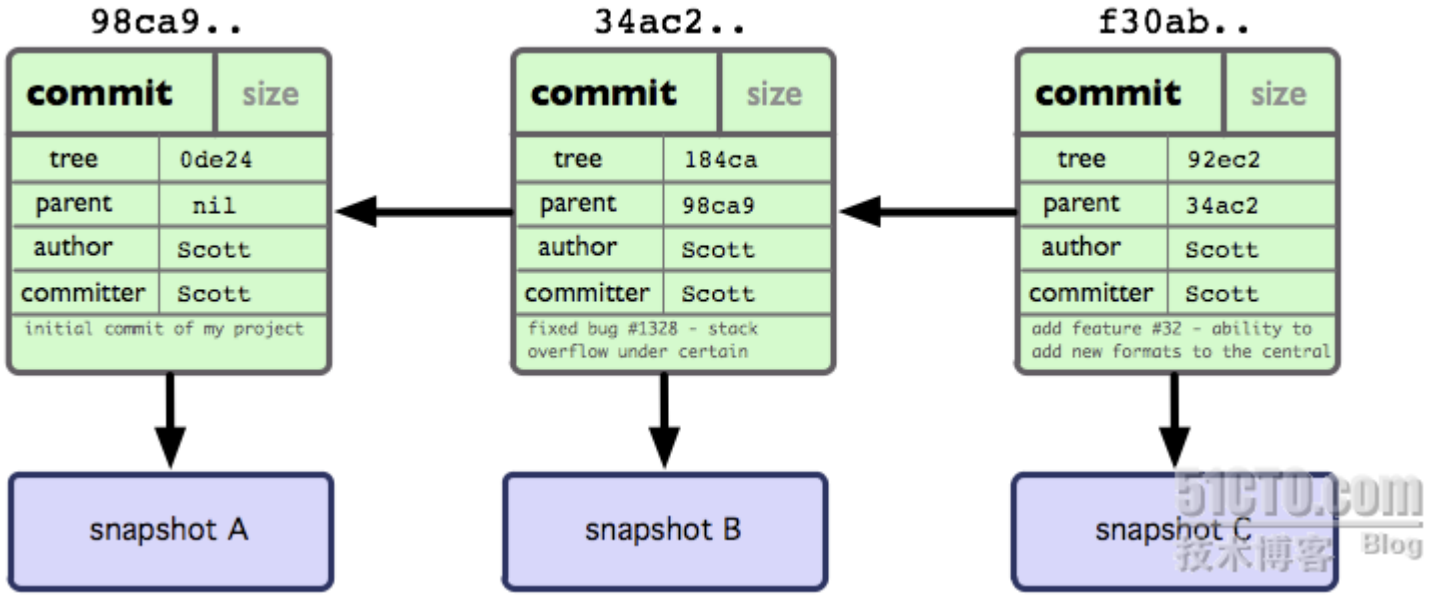
  53M ./django-svn

  git克隆比SVN要小很多，且git克隆包含整个项目的历史版本。

SVN只包含项目的最后一个版本。

快照

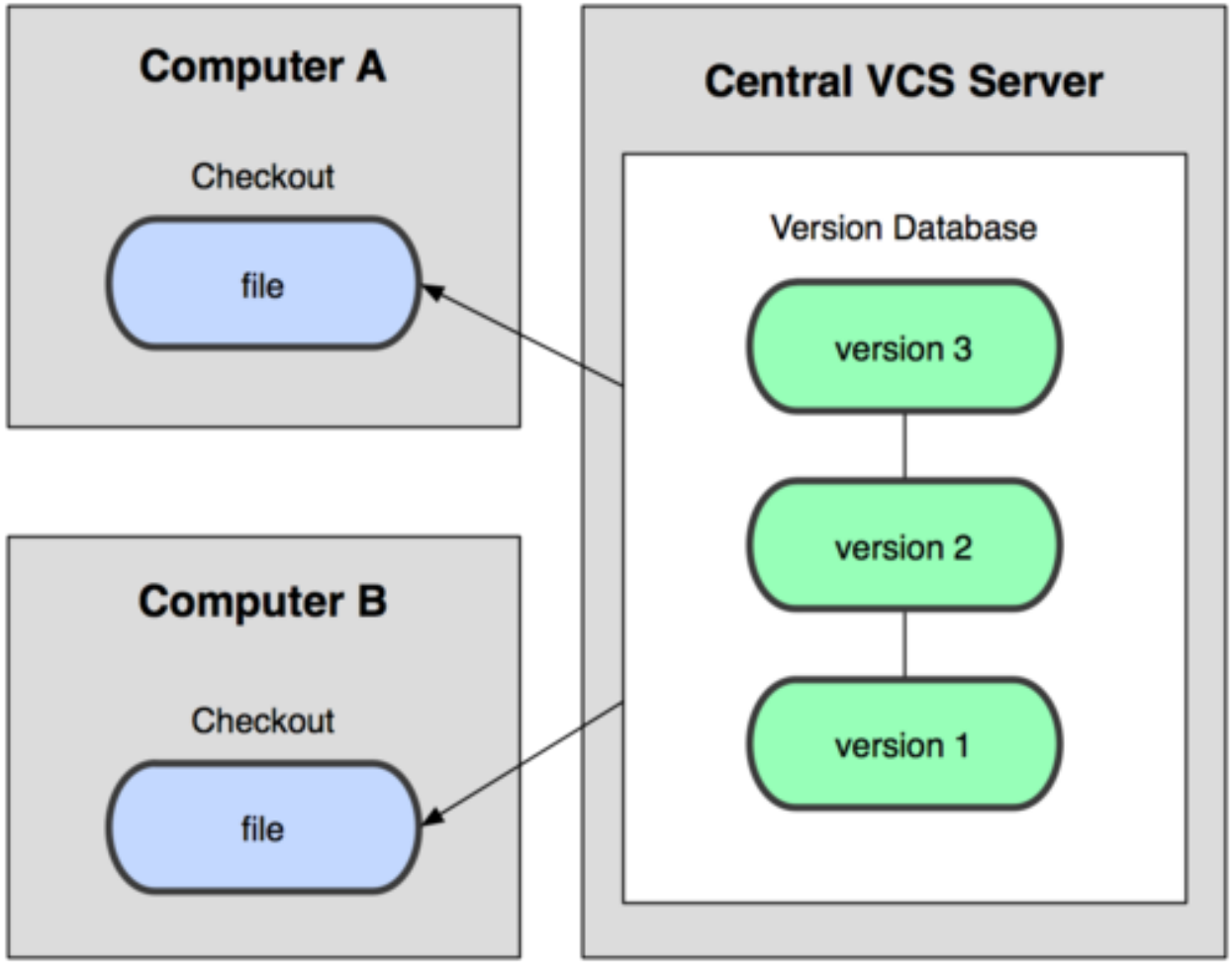
- Git是基于快照的，而不是补丁文件
- 包含一些元数据（提交信息（message），作者，日期等等），一个commit指向这次提交时项目的快照。



分支

- 以前的VCS工具分枝的方法是对每一个分枝都放到一个独立的目录中。而git可以让你在同一个工作目录中切换（switch）到不同的分枝。创建和切换分枝几乎是即时的（instant），并且存在本地分支
- git开发者可以随时创建，合并，删除多个分枝。它鼓励一种非线性的开发周期，它可以说是并行的多线程模式而不是多个步骤串行的模式。

集中化的版本控制系统


Computer A
Checkout
file
Computer B
Checkout
file
Central VCS Server
Version Database
version 3
version 2
version 1

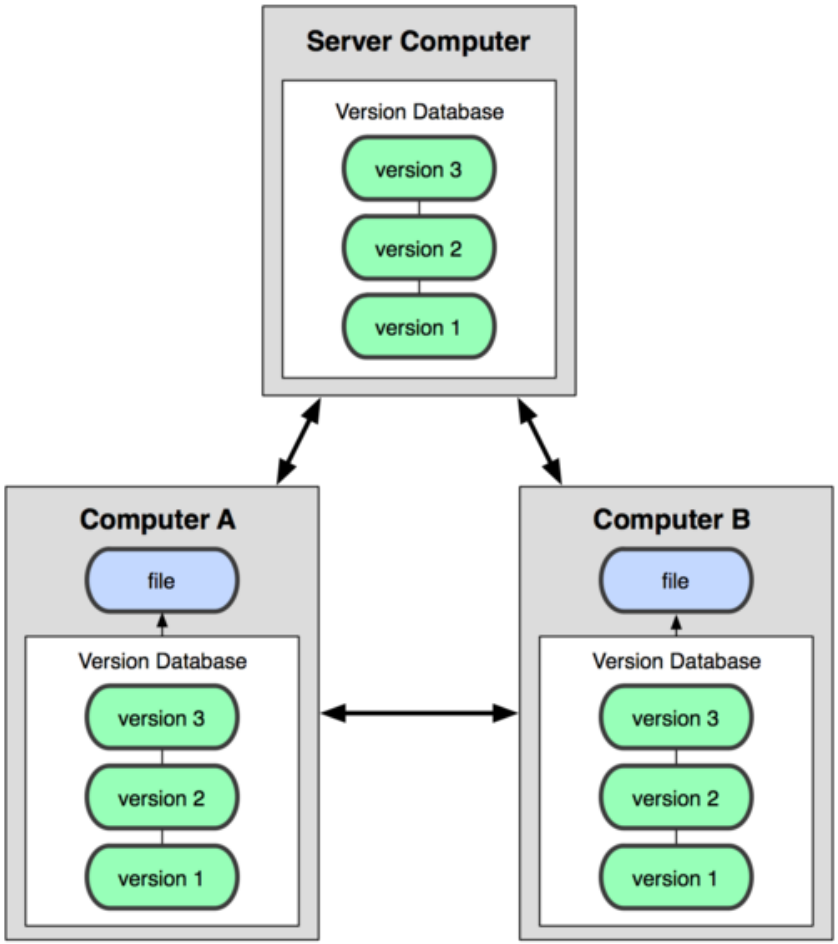
集中化的版本控制系统

- 坏处：好处：每个人都可以在一定程度上看到项目中的其他人正在做些什么。而管理员也可以轻松掌控每个开发者的权限。

  Ø（1）中央服务器的单点故障，无法提交更新，也就无法协同工作。

  Ø（2）要是中央服务器的磁盘发生故障，碰巧没做备份，会有丢失数据的风险。最坏的情况是彻底丢失整个项目的所有历史更改记录

分布式版本控制系统


Server Computer
Version Database
version 3
version 2
version 1
Computer A
file
Version Database
version 3
version 2
version 1
Computer B
file
Version Database
version 3
version 2
version 1

分布式版本控制系统

- 像GIT这种系统，客户端并不只提取最新版本的文件快照，而是把原始的代码仓库完整地镜像下来。

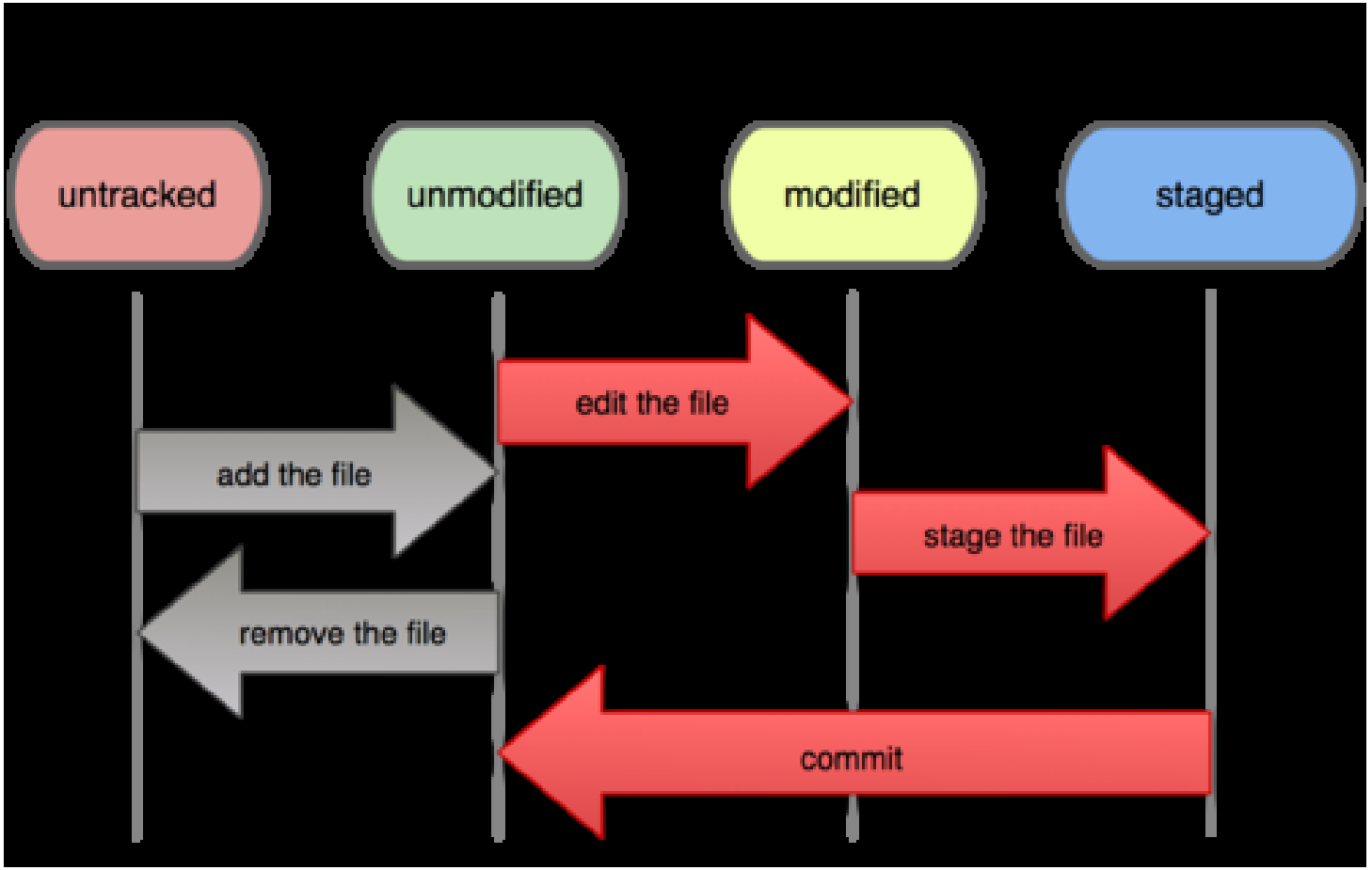
大纲

- Git简介
- Git基础
- Git操作
- Git版本管理
- GIT分支管理在项目中的实践

创建仓库(repository)

- Git init 创建一个空仓库
- git clone https://github.com/shuhangz/sysu-badminton-court-booking.git
- 复制一个远程仓库到本地

文件生命周期


untracked
unmodified
modified
staged
edit the file
add the file
stage the file
remove the file
commit

常用文件操作命令

- git status 查看文件状态
- git add <file> 跟踪新文件或暂存已修改文件
- git diff 查看文件变化
- git commit –m <msg> 提交更新
- git rm file 移除文件
- git log 查看提交日志
- git commit –ammend 修改最后一次提交
- git reset HEAD <file> 取消已暂存文件
- git checkout -- <file> 取消文件修改

远程仓库操作

- git clone <仓库地址>
- git remote –v 列出所有远程仓库
- git push <仓库名> <分支名> 推送本地分支更新到远程仓库
- git fetch 从远程仓库获取更新
- git pull 从远程仓库获取更新并merge本地分支

使用复刻（fork）

复刻通常在 GitHub 上的开源开发中使用。

大纲

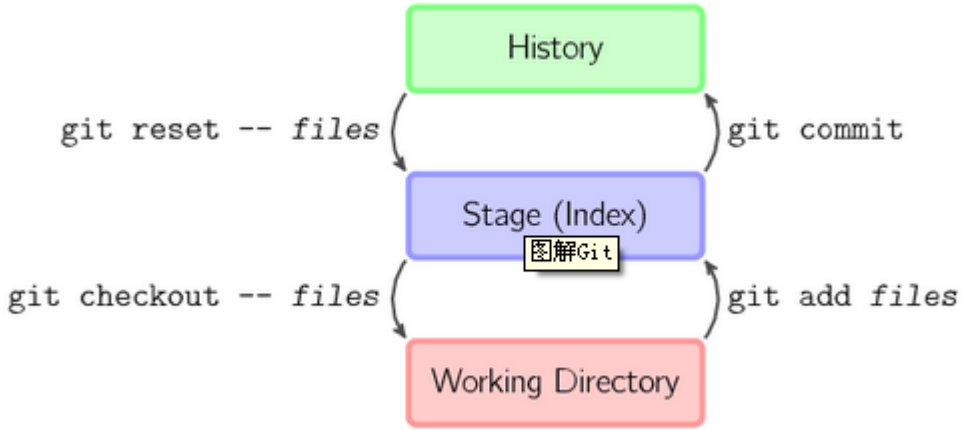
- Git简介
- Git基础
- Git操作
- GIT版本管理
- GIT分支管理在项目中的实践

Git目录

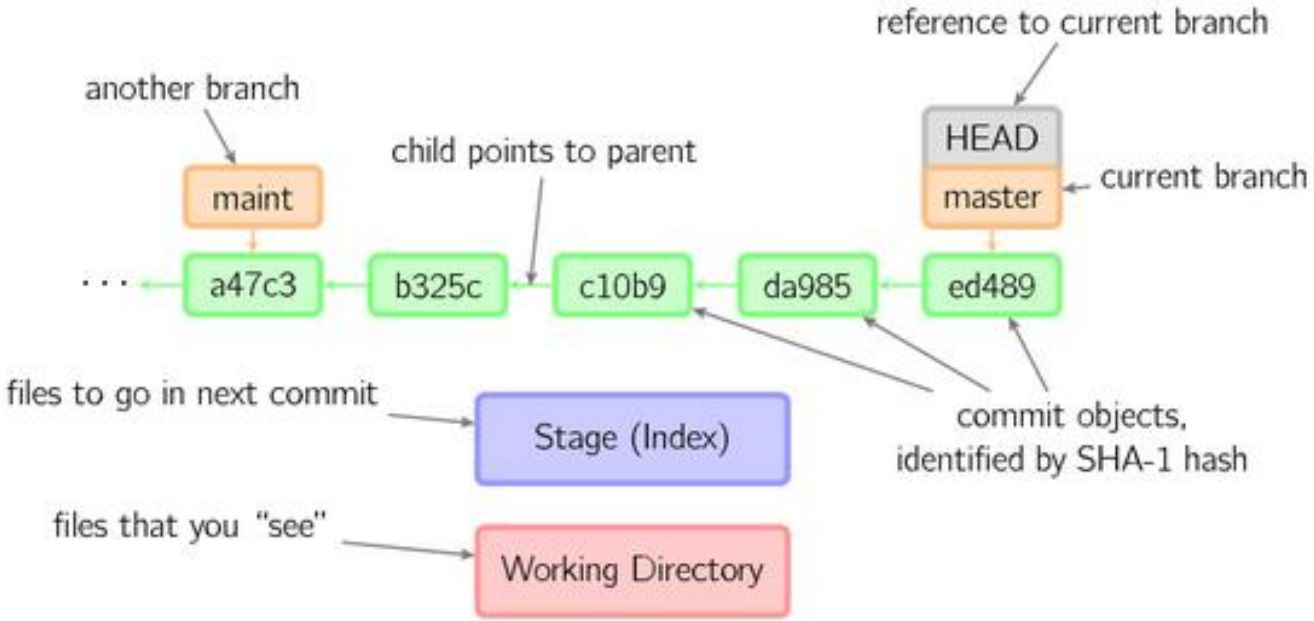
- Git一共有三个目录
  - Ø工作目录（Working Directory）
  - Ø暂存目录(index)
  - Ø仓库(History)

GIT文件状态切换

- git add *files* 把当前文件放入暂存区域。
- git commit 给暂存区域生成快照并提交。
- git reset -- *files* 用来撤销最后一次git add *files*，你也可以用git reset 撤销所有暂存区域文件。
- git checkout -- *files* 把文件从暂存区域复制到工作目录，用来丢弃本地修改。



GIT图示


reference to current branch
another branch
child points to parent
HEAD
maint
master
current branch
. . .
a47c3
b325c
c10b9
da985
ed489
files to go in next commit
Stage (Index)
commit objects,
identified by SHA-1 hash
files that you "see"
Working Directory

GIT版本

- Git版本号是一个40位的SHA-1编码的字符串
- 例如：4dd6bd612a121b24e1877dbc632e422e305dde6c
- 它不像svn那样版本号是连续的，很容易从版本号看出哪个是新版本，git的版本号是不连续的

查看历史版本

- 可以通过git log 命令来查看历史版本的提交
- git log的操作都是本地操作，基本都能瞬间完成，比SVN快很多，查看历史版本或进行diff比较都非常方便
- 也可以通过git revert操作来回退到历史版本

git log

- $ git log
- commit 734713bc047d87bf7eac9674765ae793478c50d3
- Author: Scott Chacon <schacon@gmail.com>
- Date: Fri Jan 2 18:32:33 2009 -0800
- fixed refs handling, added gc auto, updated tests
- commit d921970aadf03b3cf0e71becdaab3147ba71cdef
- Merge: 1c002dd... 35cfb2b...
- Author: Scott Chacon <schacon@gmail.com>

查看祖先引用

- git log --pretty=format:'%h %s' --graph

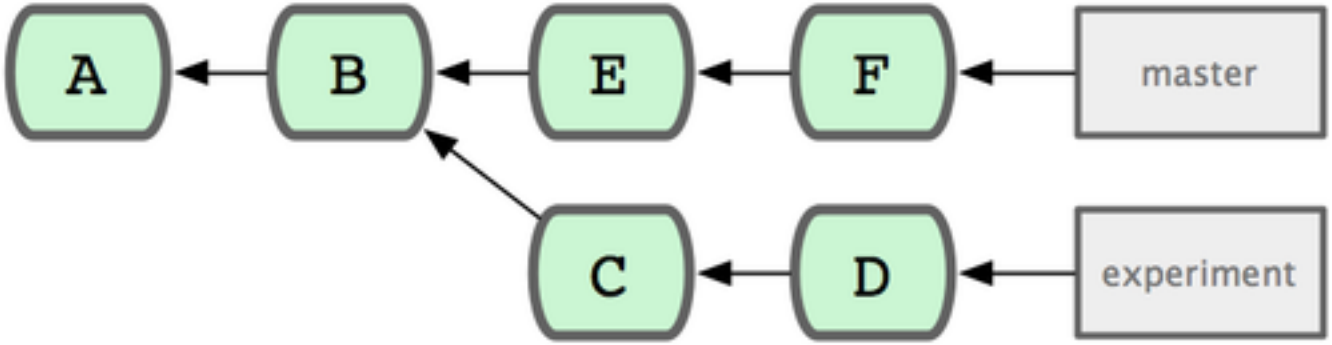
```
*   4da766c Merge branch 'dev' of ssh://scm.hz.netease.com:2222/backend/datastream into dev
|\
| * 43e61la 棋. . 结跨? 姥. ? ? 艹. ?
| *   eda6b1c Merge branch 'hotfix' into dev
| |\
| | * d368896 棋. ? dbcollector? 诲? ? . ?
| | * 6bb2f43 枵. . 覆辫触? . . 3结
| * |   dc3b9bc Merge branch 'dev' of ssh://gufeiyong@scm.hz.netease.com:2222/backend/datastream.git into dev
| |\ \
| | * \   87b0a2e Merge branch 'dev' of ssh://scm.hz.netease.com:2222/backend/datastream into dev
| | |\ \
| | * | | 8e5745c MongoHandler? 儿. ? ? vent? 版. ? 讹? 清? ull? . mpty? . . 娈佃? 嫁
| * | | |   a5db637 Merge branch 'hotfix' into dev
| |\ \ \ \
| | | | |_|/
```

SHA-1

- SHA-1摘要长度是20字节
- 如果地球上65亿的人类都在编程，每人每秒都在产生等价于整个Linux内核历史（一百万个Git 对象）的代码，并将之提交到一个巨大的Git仓库里面，那将花费5年的时间才会产生足够的对象，使其拥有50%的概率产生一次SHA-1对象冲突。

查看提交范围

```
git log master..experiemnt
D
C
```



查看提交范围

- `git log origin/master..HEAD`

- 这条命令显示任何在你当前分支上而不在远程 origin 上的提交。如果你运行`git push`
- 并且的你的当前分支正在跟踪`origin/master`, 被`git log origin/master..HEAD`

储藏stash

- 一个很实用的功能
- 在git切换分支的时候，他会提示你有未提交的更新，你需要commit才能切换，但可能当前代码很乱不能提交
- `git stash` - 将未提交代码储藏
- `git stash apply`- 取出储藏的代码

变更历史

- git允许你修改提交历史，这个很有用，可以减少那些乱七八糟的提交弄乱git仓库
- git commit -ammend
- 这会更改Sha-1值，不能再push之后再修改

git调试

- Git有个挺有用的功能，可以找出你觉得有问题的代码在哪个版本引入的
- `git blame -L 32,36  xxx.cpp`

子模块

- 第三方开发的库或者是你独立开发和并在多个父项目中使用的项目作为一个子模块放入git仓库
- `$ git submodule add git://github.com/chneukirchen/rack.git rack`

filter-branch

- 核弹级应用，尽可能小心使用
- 可以从所有历史提交中删除一个文件，悔棋用，可能你误提交了个私密文件，可以通过这个功能来从历史提交中全部删除

- `git filter-branch --tree-filter 'rm -f passwords.txt' HEAD`

大纲

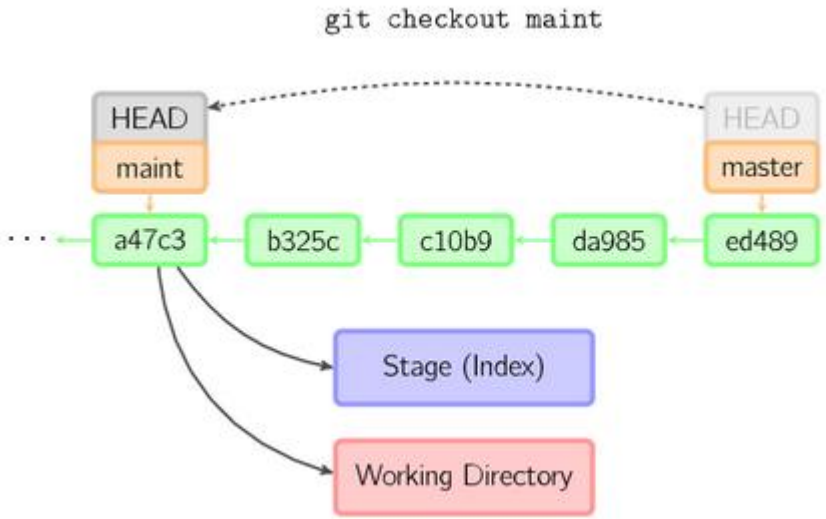
- Git简介
- Git基础
- Git操作
- GIT版本管理
- GIT分支管理在项目中的实践

GIT开分支

- Git开分支代价非常小
  - Ø仅增加几十字节的存储
  - Ø开分支不需要管理员来开
  - Ø开分支仅需要数秒钟

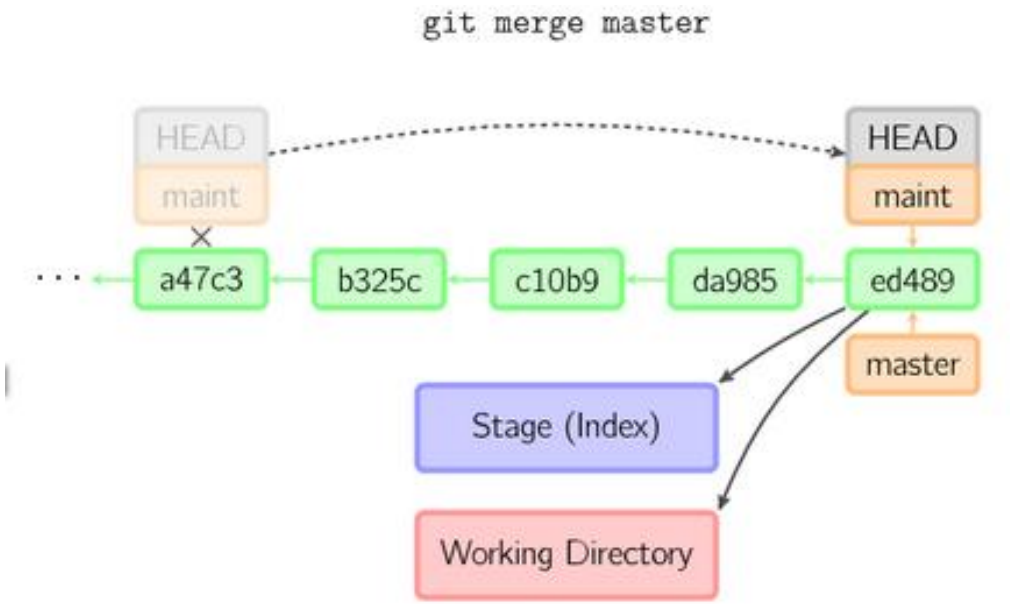
创建分支

- Git branch <branchName> 创建分支
- git checkout <branchName> 切换分支



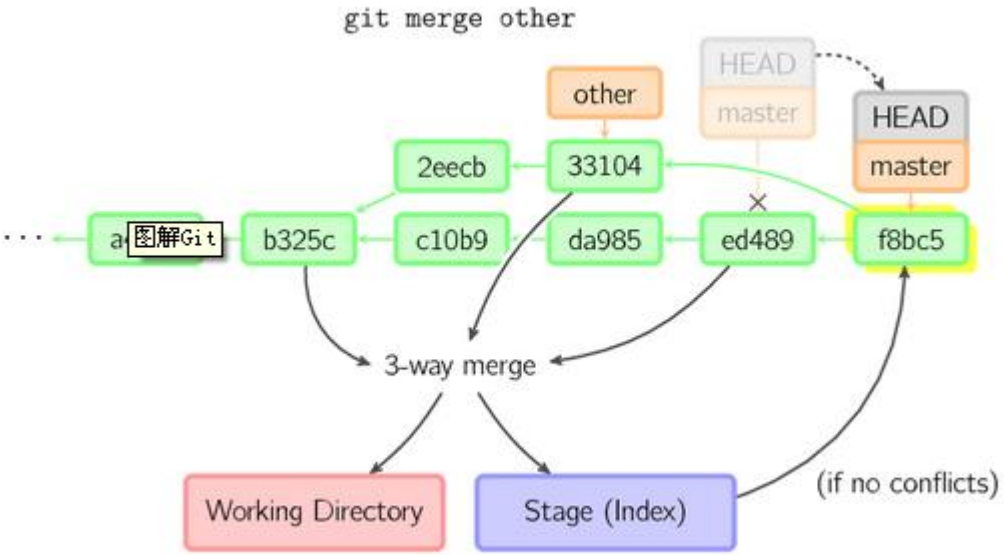
分支merge-ff

- 当一个分支是另一个分支的祖父节点，即从该分支分离后原来分支未做任何改变



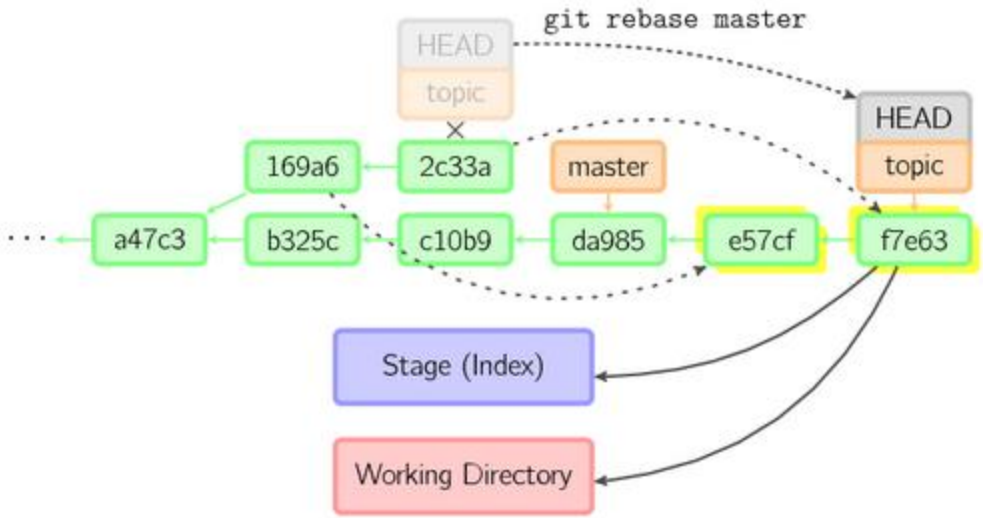
分支merge-non-ff

- 当不满足fast-forward条件时，就会发生三方合并



衍合分支-rebase

- 衍合是另一种分支合并策略，会在当前分支上重演被衍合分支的历史，他是一种线性的合并



分支管理遇到的问题

- 多人并行开发，开分支需求多，但分支开销太大，只能凑合着用
- 分支合并特别麻烦，项目后期经常为了合并分支要花费一下午的时间，特别是tree conflict问题
- 项目使用时间长后，svn服务器速度越来越慢，导致svn操作都很慢，喝杯咖啡回来继续

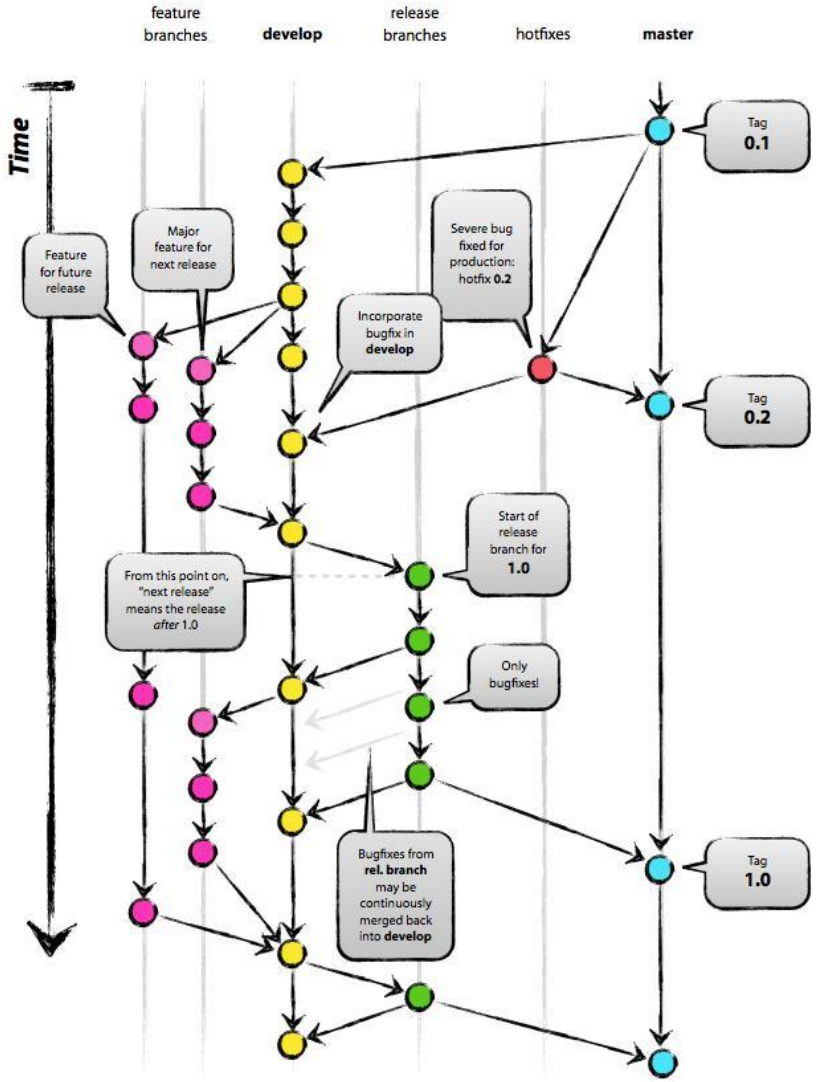
GIT的优势

- 使用GIT很大程度上解决了我们在项目中碰到的问题
  - 分支随意开，几乎0代价，鼓励大家并行开发
  - 本地分支，方便进行一些调研性的feature开发
  - 分支合并快，每天合并分支十来次，也没啥感觉
  - 图形化工具能方便管理分支，检查有无分支未被合并

Git在Datastream中的运用

- Datastream项目从今年下半年开始正式使用git作为代码管理的工具，摸索出了一些使用心得
  - GIT虽然是一个分布式版本管理框架，理论上没有中心库的概念。但在实际应用中还是要有一个中心库，方便多人开发的代码做同步
  - 尽量多使用本地分支进行开发、单元测试，测试通过再合并到协同开发的分支上

项目分支管理策略


feature branches
develop
release branches
hotfixes
master
Time
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.1
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

项目分支管理策略

- 一共有以下一些分支
  - Master:主干分支用于上线
  - Release branches：预发布分支
  - Develop：开发分支
  - Hotfix branches:线上bug修复分支
  - Feature branches:长期功能分支

分支-环境

- 如果仅仅有分支管理，还无法管理好一个大型的项目，需要针对不同的分支，有不同的环境对应
  - 开发环境->develop分支
  - QA环境->Release分支、hotfix分支
  - Per环境->性能测试相关的feature branch
  - 线上环境->Master分支

  只有分支和环境对应关系理清楚，我们才能规划好每一步，否则还是一团乱

开发

- Develop是一个长期分支，也是项目中最重要的一个分支，在项目里它主要承担协同开发、集成测试的作用。
- 开发人员在开发feature时，先本地开本地分支进行开发和单元测试（如果该模块开发需要其他模块协助，也可以把这个本地分支push到远端），单元测试通过后merge到develop分支进行集成测试。
- Develop分支不要求绝对稳定，但是也要保证一定的稳定度，至少新feature编译和单元测试要通过，否则会影响其他同事的开发

测试

- Release预发布分支一般会在项目有第一个feature提交时开，主要用于QA测试
- 只有develop分支集成测试通过的才可以提交到release分支供QA测试
- QA如果在Release分支测试出有bug，最好的方式是通过Release上开个分支，修复bug，并借用下develop环境调试bug（这步过程需要协商），修复完成后merge到Release分支以及Develop分支

上线

- QA在Release分支测试通过后，会通过配置管理员将Release分支通过fast-forward模式merge到master分支，这样不会有任何conflict。
- Master分支上打TAG，然后部署上线。这样一个项目流程就结束了。

线上bug

- 当发现线上bug后，会开启一个hotfix分支，修复bug，并在QA环境上测试
- 测试通过后merge到master，部署上线
- 并也要merge到develop分支

性能测试

- 项目过程中需要做性能测试的时候可以从develop上拉一个性能测试的feature branch，例如perf
- 当有性能相关的feature提交的时候merge develop到perf
- 如果有性能问题需要修复，可以在perf上进行修复，并merge到develop

git使用过程遇到的问题

- Git也存在一些问题
  - 没有目录的概念，因此不太容易对子目录控制权限
  - 可以使用submodule来控制权限，但是配置和管理代码的复杂度就会上升，分支管理会趋于复杂
  - Merge的时候整个branch的代码一起merge，在develop分支上有时会有部分人集成测试通过，部分人测试未通过的时候，这时候merge会出问题。目前通过沟通解决，merge之前通知所有人，如果有问题的人先把代码revert一下，大部分时候问题不大。

git使用惯例

- 要早提交，常提交，并且不要觉得麻烦

1. 每个提交的修订都会为你提供一个还原点。如果你完全把代码搞砸了（没骗你，我们都这么做过），你是希望恢复到一个小时前的工作还是一周前的工作?
2. 合并文件时会出现的危险会随着时间不断增加。合并文件一直很麻烦。如果你不是每天都保持提交代码，某一天你会突然发现你和其他人的更改内容会有50多个冲突。你不会为此感到高兴的。
3. 它促使你把任务分离成分散的单元。通常人们都是快完成的时候才提交的，因为他们想把代码做成一个完整的逻辑单元模块。不过庞大的任务不可避免地要分离出较小的分散功能，而频繁地提交它们会使你更了解它们，你可以一个个地构建并提交。

git使用惯例（续）

- 写提交信息时一定要认真

解释清楚为什么要提交新的代码，同一个程序员之后提交信息绝不能和前面的完全相同.

- 这里列出一些提交信息的**反面教材**：

1. 什么也没做
2. 能跑了
3. 解决了一些混帐问题
4. 解决了
5. 改进了一点bug
6. 上传了
7. 排字错误
8. 修订1024

git使用惯例（续）

- 常用git status查看工作目录状态
- 提交前要检查你更改了什么(使用git diff)
- 不要上传你自己的用户设置
- 附属文件也要集成在一起，比如更新了新的第三方库(无源码的.a .so文件)
- 临时文件，编译生成的文件不要放进源代码管理软件里(使用.gitignore来忽略)，也不要通过复制或者打包文件来进行备份

总结

- GIT从很大程度减轻了项目过程中代码管理、分支管理的代价，非常适合大型项目的多人并发开发
- 简单，可以让程序员不需要太多的代码分支管理技术也可以很好地管理代码，不容易出错
- 高速，大部分的本地操作，让程序员不会浪费时间在等待svn操作上

THANKS

感谢

声明

- 本课程PPT部分内容收集自网上公开资料，如有侵权，请邮件联系我们
- 如您是承担了类似课程的教师，需要原始pptx文件，请邮件联系张书航或李同文
- 联系方式在首页