

一、Git

1. 软件配置管理

软件配置管理(SCM)-Software configuration management 是指通过执行版本控制、变更控制的规程，以及使用合适的配置管理软件，来保证所有配置项的完整性和可跟踪性。配置管理是对工作成果的一种有效保护。

目前国内常用的配置管理工具大概有SourceSafe、CVS、subversion、Git和ClearCase,其中CVS,subversion和Git属于开源版本控制系统。

2. 为什么需要配置管理

如果没有软件配置管理，最大的麻烦是工作成果无法回溯。

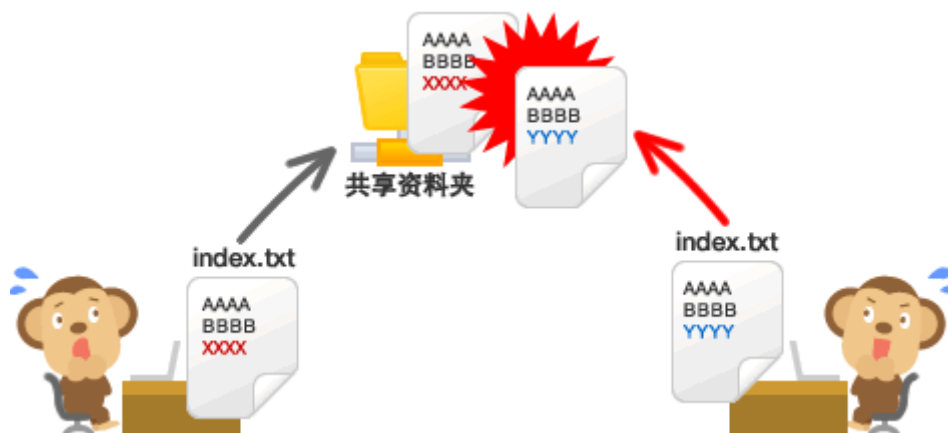
最简单的方法就是先备份编辑前的文档。使用这个方法时，我们通常都会在备份的文档名或目录名上添加编辑的日期。但是，每次编辑文档都要事先复制，这样非常麻烦，也很容易出错。

| Name |
|----------------------|
| 120525_文档_更新.txt |
| 120604_文档.txt |
| 120605_文档_monkey.txt |
| 120605_文档_最新.txt |
| 120605_文档_最新复制.txt |
| 120605_文档_修改版.txt |
| 120605_文档.txt |
| 1200602_文档.txt |
| 文档_会议用.txt |



再加上，如果像上图那样毫无命名规则的话，就无法区分哪一个文档是最新的了。而且，如果是共享文件的话，应该加上编辑者的名字。还有，那些文档名字没有体现修改内容。

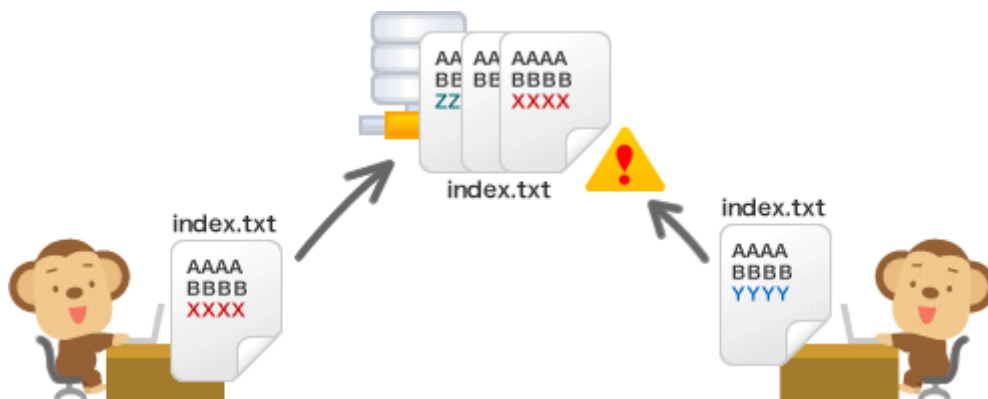
另外，如果两个人同时编辑某个共享文件，先进行编辑的人所做的修改内容会被覆盖，相信大家都有这样的经历。



以Git为代表的软件配置（版本）管理系统就是为了解决这些问题应运而生的。

版本管理工具可以在任何时间点，把文档的状态作为更新记录保存起来。因此可以把编辑过的文档复原到以前的状态，也可以显示编辑前后的内容差异。

而且，编辑旧文件后，试图覆盖较新的文件的时候（即上传文件到服务器时），系统会发出警告，因此可以避免在无意中覆盖了他人的编辑内容。

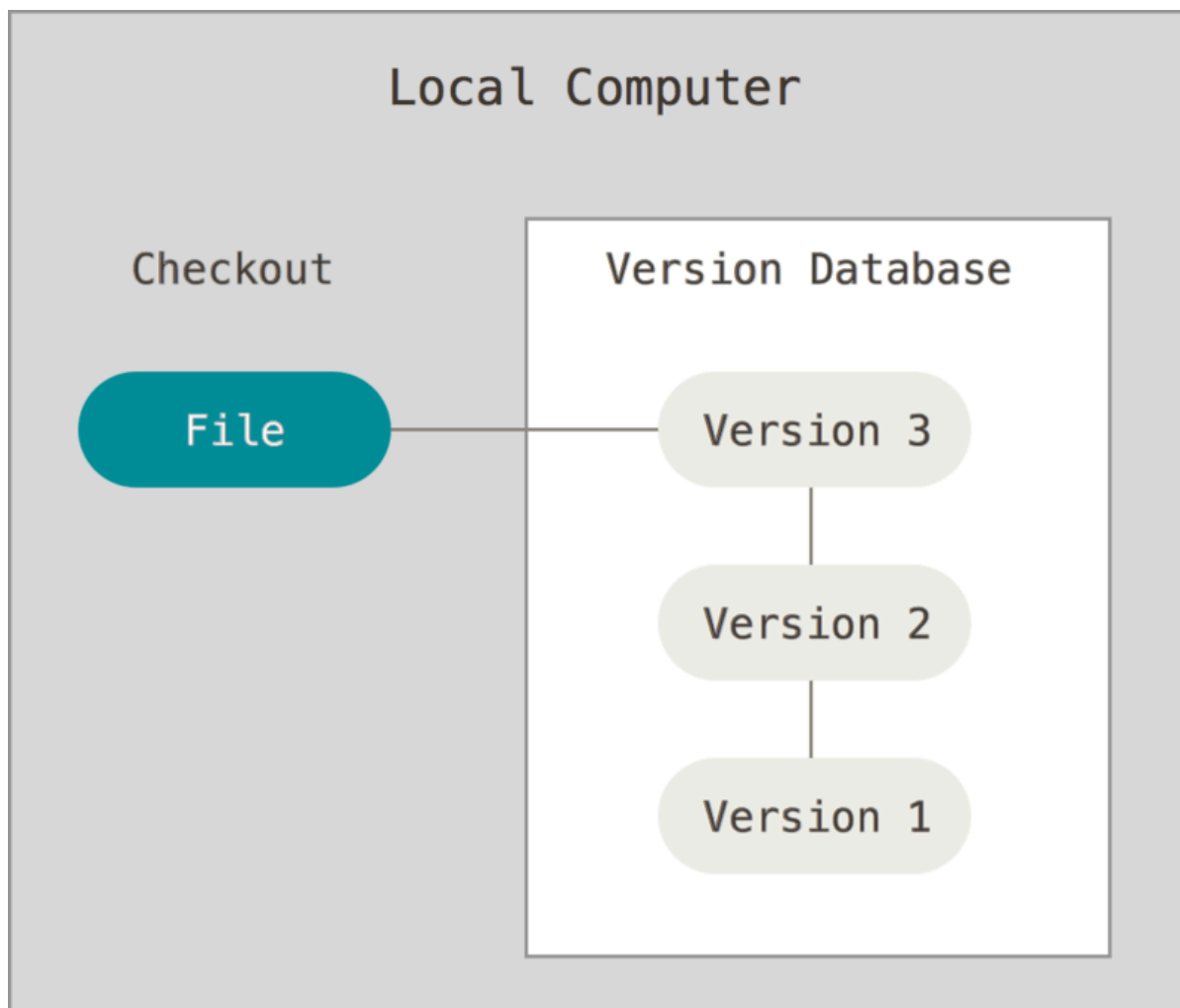


3. 配置管理工具分类

3.1. 本地版本控制系统

许多人习惯用复制整个项目目录的方式来保存不同的版本，或许还会改名加上备份时间以示区别。这么做唯一的好处就是简单，但是特别容易犯错。有时候会混淆所在的工作目录，一不小心会写错文件或者覆盖意想不到的文件。

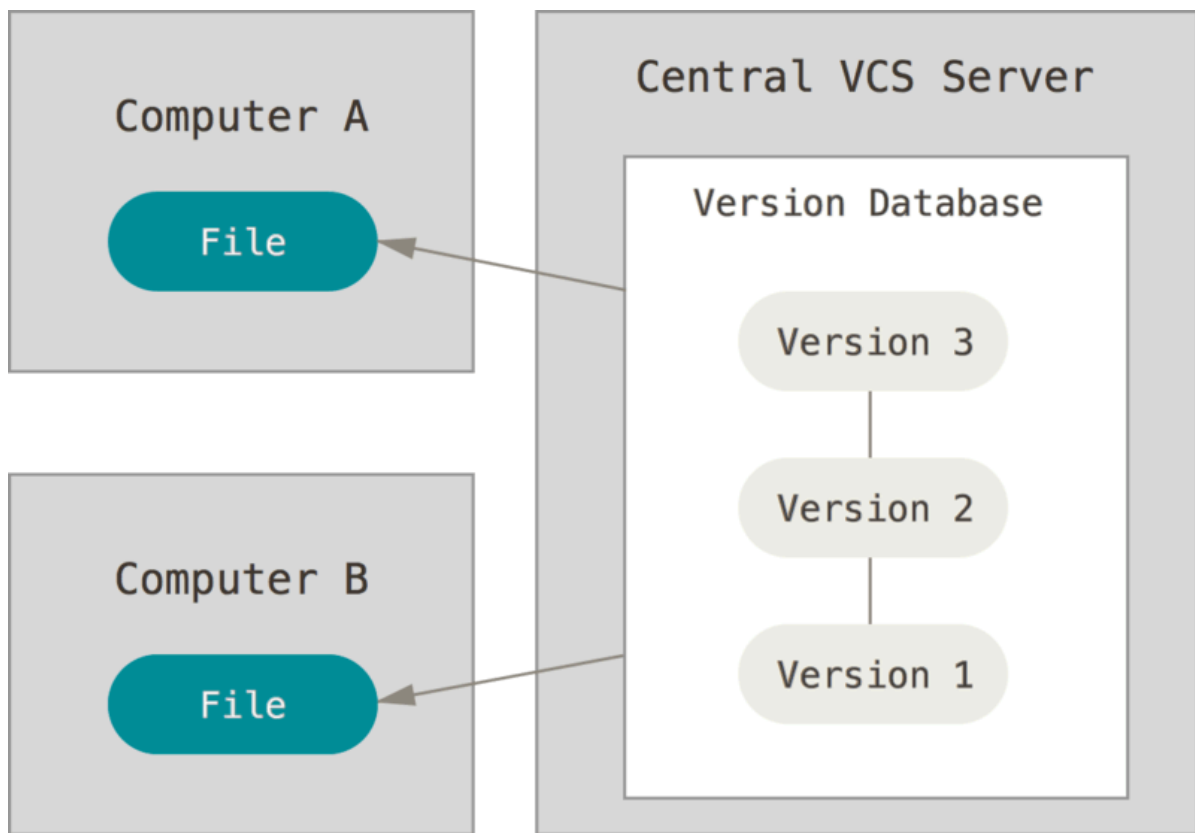
为了解决这个问题，人们很久以前就开发了许多种本地版本控制系统，大多都是采用某种简单的数据库来记录文件的历次更新差异。



其中最流行的一种叫做 RCS，现今许多计算机系统上都还看得到它的踪影。甚至在流行的 Mac OS X 系统上安装了开发者工具包之后，也可以使用 `rcs` 命令。它的工作原理是在硬盘上保存补丁集（补丁是指文件修订前后的变化）；通过应用所有的补丁，可以重新计算出各个版本的文件内容。

3.2. 集中化的版本控制系统

接下来人们又遇到一个问题，如何让在不同系统上的开发者协同工作？于是，集中化的版本控制系统（Centralized Version Control Systems，简称 CVCS）应运而生。这类系统，诸如 CVS、Subversion 以及 Perforce 等，都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。多年以来，这已成为版本控制系统的标准做法。

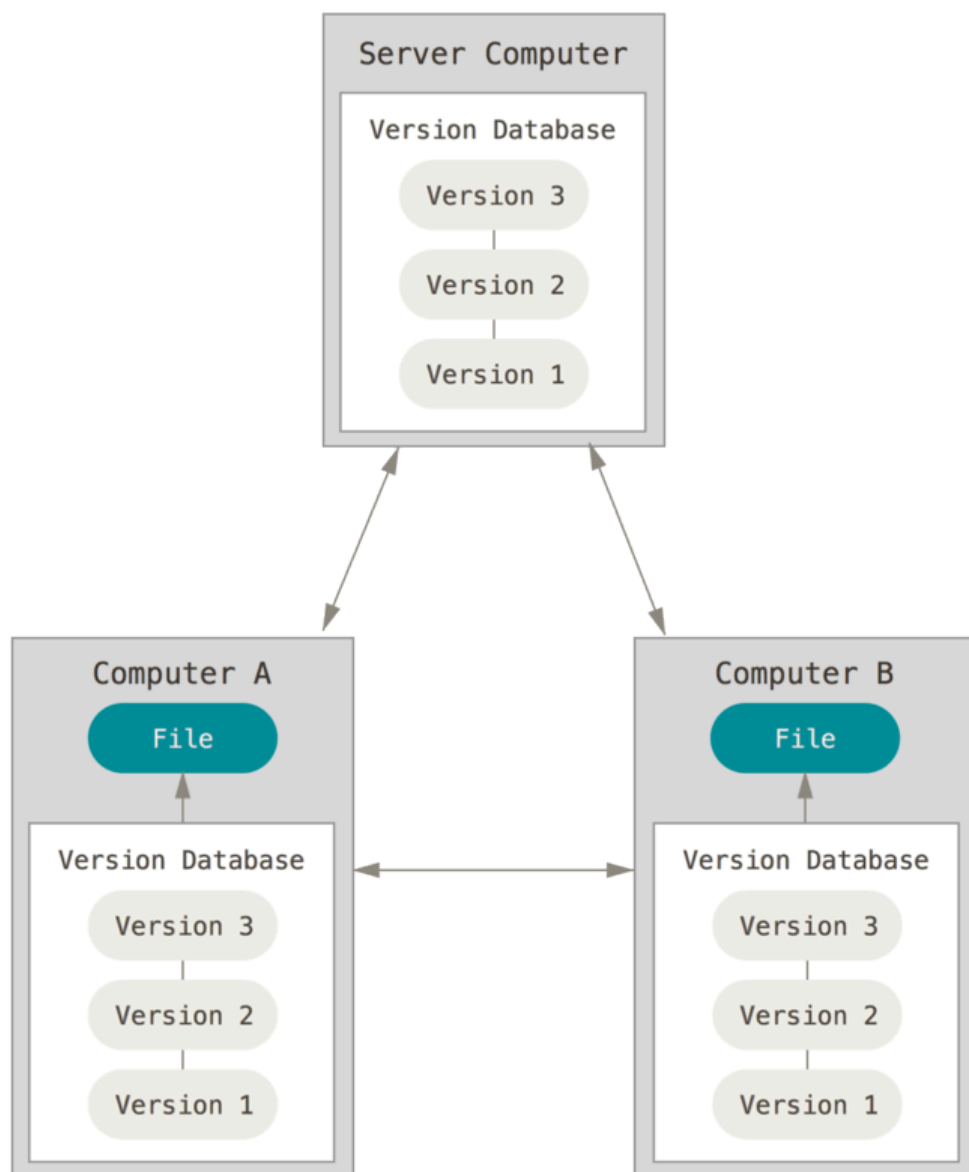


这种做法带来了许多好处，特别是相较于老式的本地 VCS 来说。现在，每个人都可以在一定程度上看到项目中的其他人正在做些什么。而管理员也可以轻松掌控每个开发者的权限，并且管理一个 CVCS 要远比在各个客户端上维护本地数据库来得轻松容易。

事分两面，有好有坏。这么做最显而易见的缺点是中央服务器的单点故障。如果宕机一小时，那么在这一小时内，谁都无法提交更新，也就无法协同工作。如果中心数据库所在的磁盘发生损坏，又没有做恰当备份，毫无疑问你将丢失所有数据——包括项目的整个变更历史，只剩下人们在各自机器上保留的单独快照。本地版本控制系统也存在类似问题，只要整个项目的历史记录被保存在单一位置，就有丢失所有历史更新记录的风险。

3.3. 分布式版本控制系统

于是分布式版本控制系统（Distributed Version Control System，简称 DVCS）面世了。在这类系统中，像 Git、Mercurial、Bazaar 以及 Darcs 等，客户端并不只提取最新版本的文件快照，而是把代码仓库完整地镜像下来。这么一来，任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复。因为每一次的克隆操作，实际上都是一次对代码仓库的完整备份。



更进一步，许多这类系统都可以指定和若干不同的远端代码仓库进行交互。籍此，你就可以在同一个项目中，分别和不同工作小组的人相互协作。你可以根据需要设定不同的协作流程，比如层次模型式的工作流，而这在以前的集中式系统中是无法实现的。

4. Git（分布式版本控制系统）

Git(读音为/git/)是一个开源的分布式版本控制系统，可以有效、高速地处理从很小到非常大的项目版本管理。[1] Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

分布式相比于集中式的最大区别在于开发者可以提交到本地，每个开发者通过克隆（Git clone），在本地机器上拷贝一个完整的Git仓库。

详细参考书：

- <https://Git-scm.com/book/zh/v2>
- <https://backlog.com/Git-tutorial/cn/>

5. Git 简史

同生活中的许多伟大事物一样，Git 诞生于一个极富纷争大举创新的年代。

Linux 内核开源项目有着为数众多的参与者。绝大多数的 Linux 内核维护工作都花在了提交补丁和保存归档的繁琐事务上（1991 - 2002年间）。到 2002 年，整个项目组开始启用一个专有的分布式版本控制系统 BitKeeper 来管理和维护代码。

到了 2005 年，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 Linux 内核社区免费使用 BitKeeper 的权力。这就迫使 Linux 开源社区（特别是 Linux 的缔造者 Linus Torvalds）基于使用 BitKeeper 时的经验教训，开发出自己的版本系统。他们对新的系统制订了若干目标：

- 速度
- 简单的设计
- 对非线性开发模式的强力支持（允许成千上万个并行开发的分支）
- 完全分布式
- 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）

自诞生于 2005 年以来，Git 日臻成熟完善，在高度易用的同时，仍然保留着初期设定的目标。它的速度飞快，极其适合管理大项目，有着令人难以置信的非线性分支管理系统。

6. Git的基础

6.1. 管理历史记录的数据库

数据库 (Repository) 是记录文件或目录状态的地方，存储着内容修改的历史记录。在数据库的管理下，把文件和目录修改的历史记录放在对应的目录下。

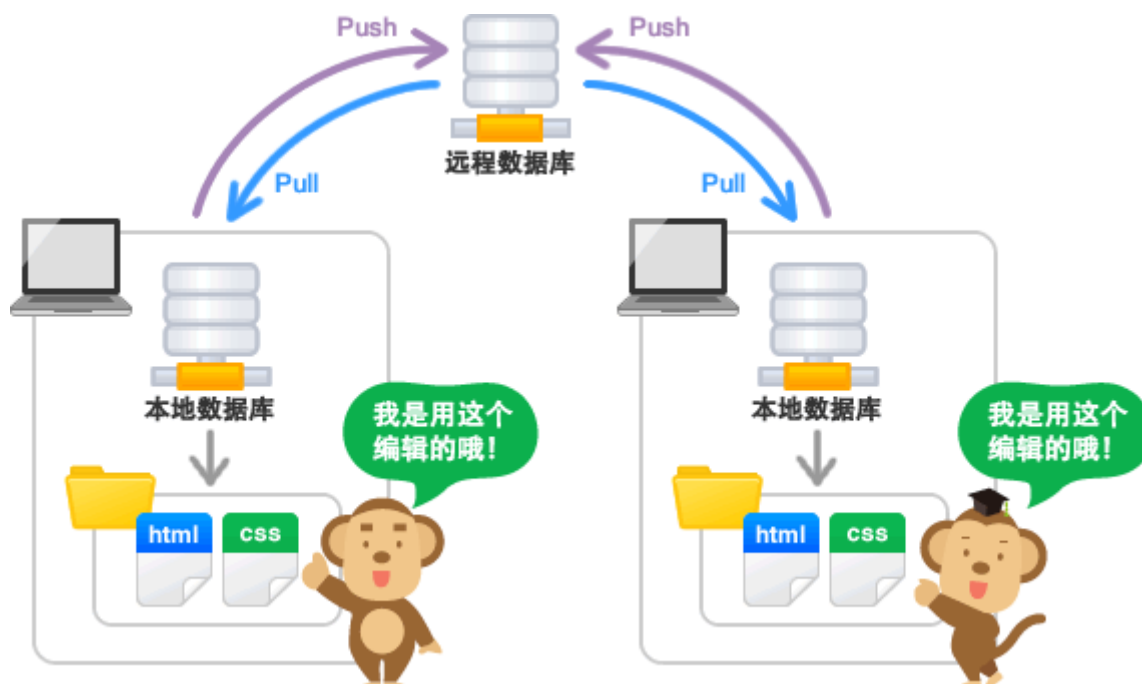


6.2. 远程数据库和本地数据库

首先，Git的数据库分为远程数据库和本地数据库的两种。

- 远程数据库: 配有专用的服务器，为了多人共享而建立的数据库。
- 本地数据库: 为了方便用户个人使用，在自己的机器上配置的数据库。

数据库分为远程和本地两种。平时用手头上的机器在本地数据库上操作就可以了。如果想要公开在本地数据库中修改的内容，把内容上传到远程数据库就可以了。另外，通过远程数据库还可以取得其他人修改的内容。



6.3. 创建数据库

创建本地数据库的方法有两种：一种是创建全新的数据库，另一种是复制远程数据库。



6.4. 修改记录的提交

若要把文件或目录的添加和变更保存到数据库，就需要进行提交。

执行提交后，数据库中会生成上次提交的状态与当前状态的差异记录（也被称为revision）。

如下图，提交是以时间顺序排列状态被保存到数据库中的。凭借该提交和最新的文件状态，就可以知道过去的修改记录以及内容。



系统会根据修改的内容计算出没有重复的40位英文及数字来给提交命名。指定这个命名，就可以在数据库中找到对应的提交。

不同类别的修改 (如: Bug修复和功能添加) 要尽量分开提交, 以方便以后从历史记录里查找特定的修改内容。

执行提交时, 系统会要求输入提交信息。请务必输入提交信息, 因为在空白的状态下执行提交会失败的。

⚡ Tips (小贴士)

查看其他人提交的修改内容或自己的历史记录的时候, 提交信息是需要用到的重要资料。所以请用心填写修改内容的提交信息, 以方便别人理解。

以下是Git的标准注解:

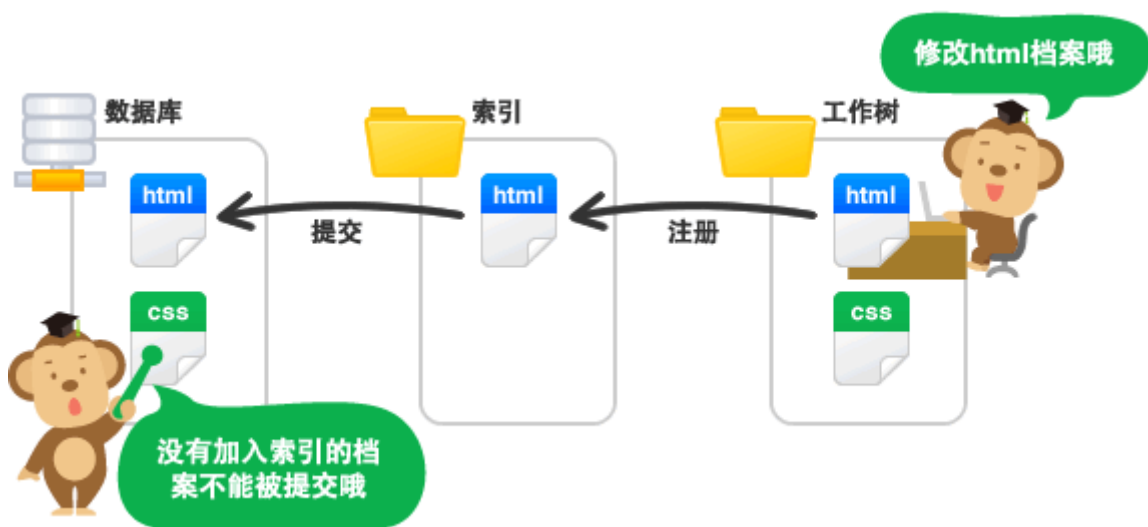
- 1 第1行: 提交修改内容的摘要
- 2 第2行: 空行
- 3 第3行以后: 修改的理由

请以这种格式填写提交信息。

6.5. 工作树和索引

在Git管理下, 大家实际操作的目录被称为工作树。

在数据库和工作树之间有索引, 索引是为了向数据库提交作准备的区域。



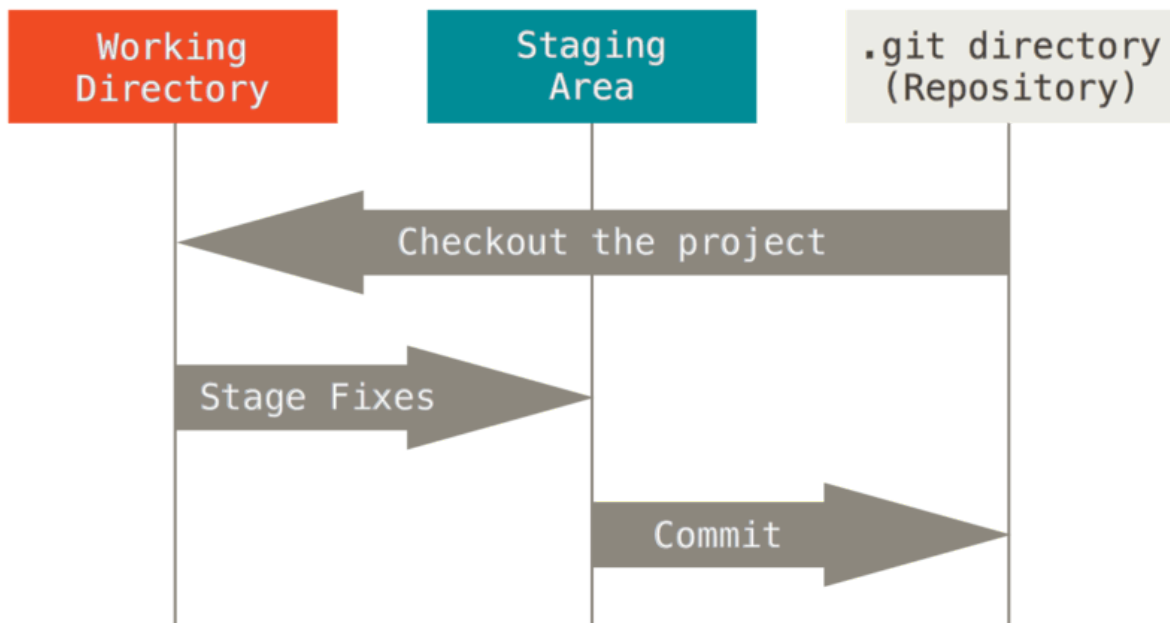
Git在执行提交的时候, 不是直接将工作树的状态保存到数据库, 而是将设置在中间索引区域的状态保存到数据库。因此, 要提交文件, 首先需要把文件加入到索引区域中。

所以, 凭借中间的索引, 可以避免工作树中不必要的文件提交, 还可以将文件修改内容的一部分加入索引区域并提交。

6.6. Git的三种状态

Git 有三种状态, 你的文件可能处于其中之一: 已提交 (committed)、已修改 (modified) 和已暂存 (staged)。已提交表示数据已经安全的保存在本地数据库中。已修改表示修改了文件, 但还没保存到数据库中。已暂存表示对一个已修改文件的当前版本做了标记, 使之包含在下次提交的快照中。

由此引入 Git 项目的三个工作区域的概念: Git 仓库、工作目录以及暂存区域。



- Git 仓库目录是 Git 用来保存项目的元数据和对象数据库的地方。这是 Git 中最重要的部分，从其它计算机克隆仓库时，拷贝的就是这里的数据。
- 工作目录是对项目的某个版本独立提取出来的内容。这些从 Git 仓库的压缩数据库中提取出来的文件，放在磁盘上供你使用或修改。
- 暂存区域是一个文件，保存了下次将提交的文件列表信息，一般在 Git 仓库目录中。有时候也被称作“索引”，不过一般说法还是叫暂存区域。

基本的 Git 工作流程如下：

1. 在工作目录中修改文件。
2. 暂存文件，将文件的快照放入暂存区域。
3. 提交更新，找到暂存区域的文件，将快照永久性存储到 Git 仓库目录。

如果 Git 目录中保存着特定版本的文件，就属于已提交状态。如果作了修改并已放入暂存区域，就属于已暂存状态。如果自上次取出后，作了修改但还没有放到暂存区域，就是已修改状态。

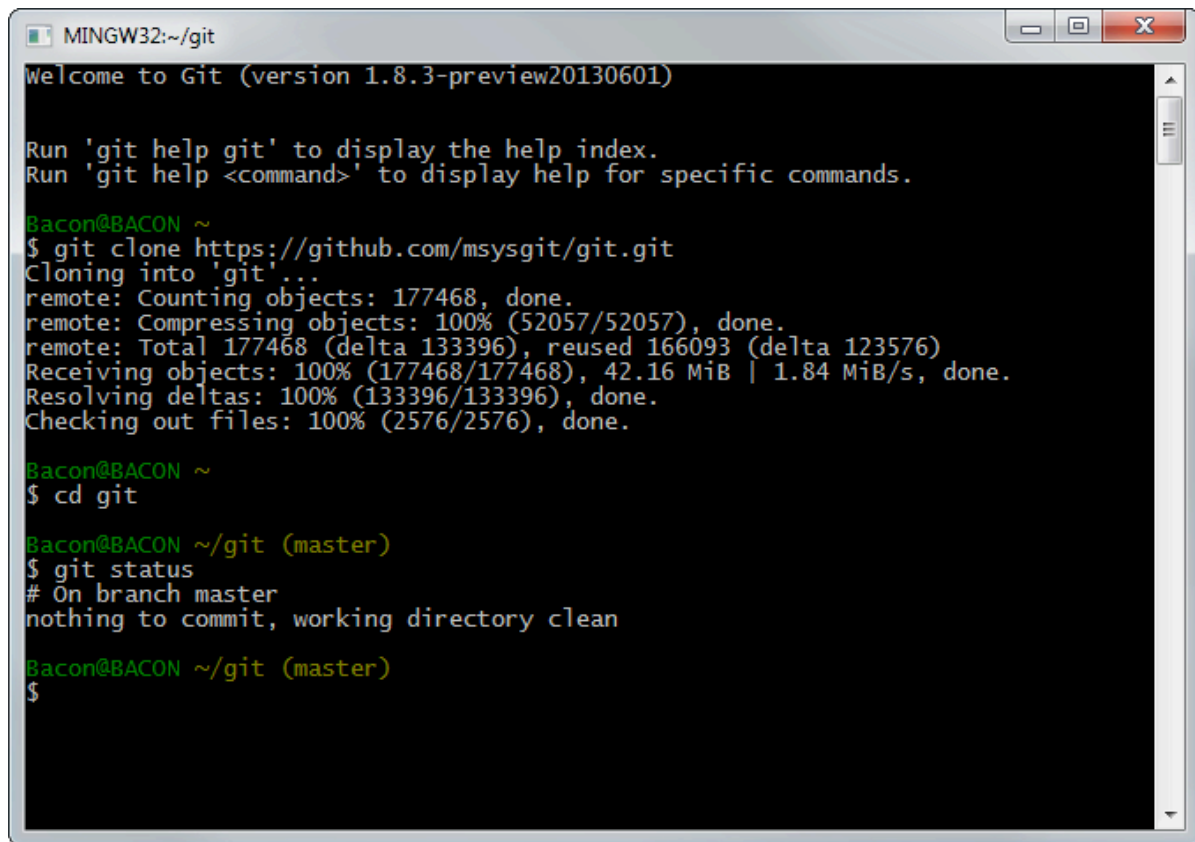
7. Git的实践

7.1. 在 Windows 上安装

在 Windows 上安装 Git 也有几种安装方法。官方版本可以在 Git 官方网站下载。打开 <http://Git-scm.com/download/win>，下载会自动开始。要注意这是一个名为 Git for Windows 的项目（也叫做 msysGit），和 Git 是分别独立的项目；更多信息请访问 <http://msysGit.Github.io/>。

另一个简单的方法是安装 GitHub for Windows。该安装程序包含图形化和命令行版本的 Git。它也能支持 Powershell，提供了稳定的凭证缓存和健全的 CRLF 设置。稍后我们会对这方面有更多了解，现在只要一句话就够了，这些都是你所需要的。你可以在 GitHub for Windows 网站下载，网址为 <http://windows.Github.com>。

7.1.1. Git BASH



```
MINGW32:~/git
Welcome to Git (version 1.8.3-preview20130601)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

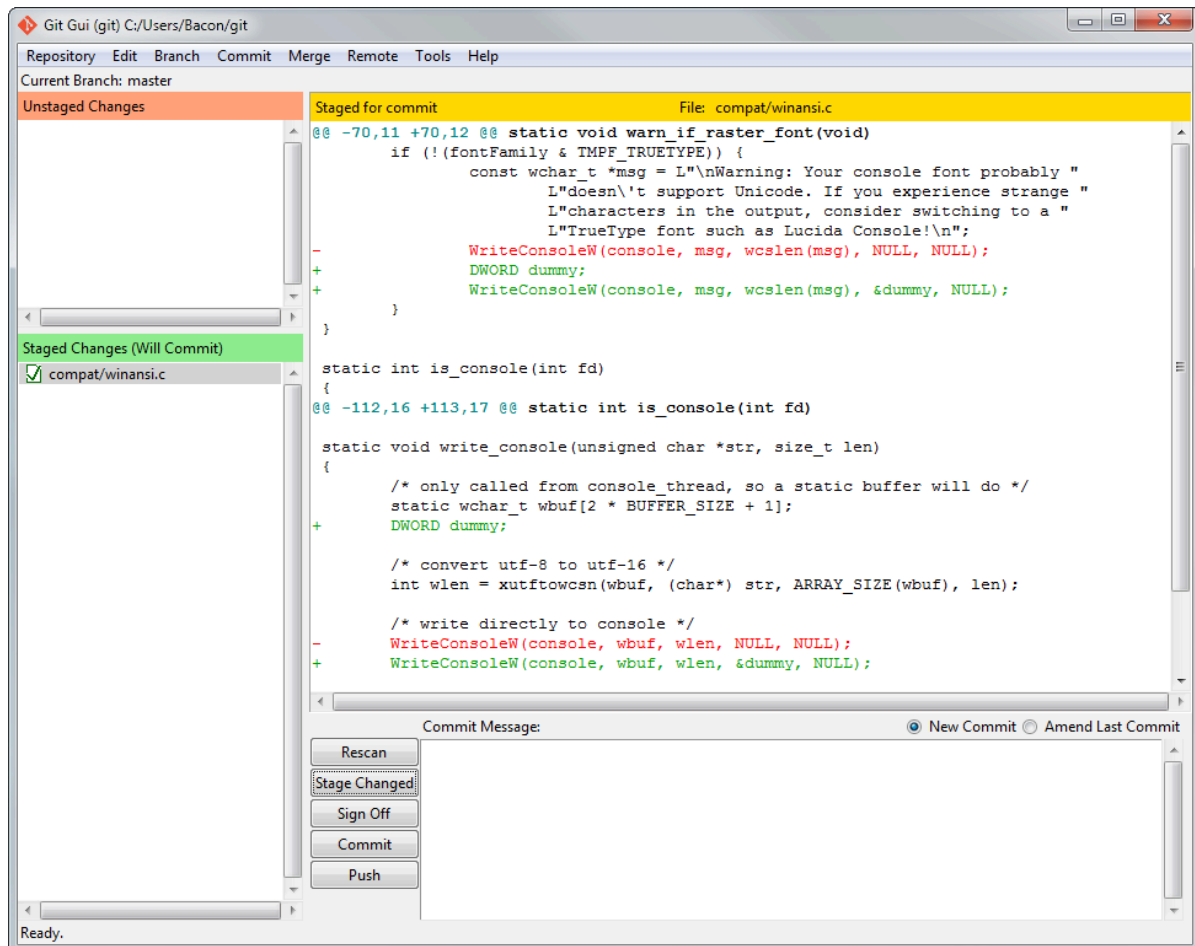
Bacon@BACON ~
$ git clone https://github.com/msysgit/git.git
Cloning into 'git'...
remote: Counting objects: 177468, done.
remote: Compressing objects: 100% (52057/52057), done.
remote: Total 177468 (delta 133396), reused 166093 (delta 123576)
Receiving objects: 100% (177468/177468), 42.16 MiB | 1.84 MiB/s, done.
Resolving deltas: 100% (133396/133396), done.
Checking out files: 100% (2576/2576), done.

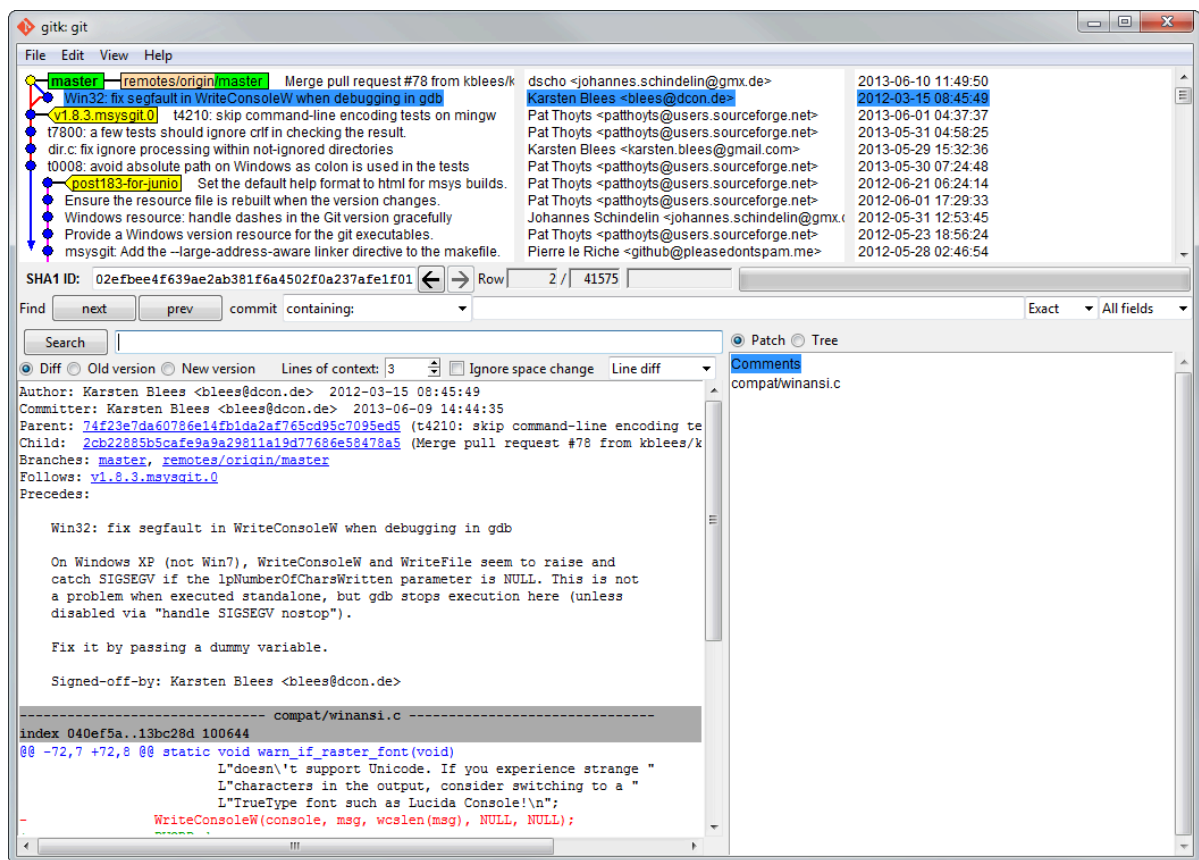
Bacon@BACON ~
$ cd git

Bacon@BACON ~/git (master)
$ git status
# On branch master
nothing to commit, working directory clean

Bacon@BACON ~/git (master)
$
```

7.1.2. Git GUI





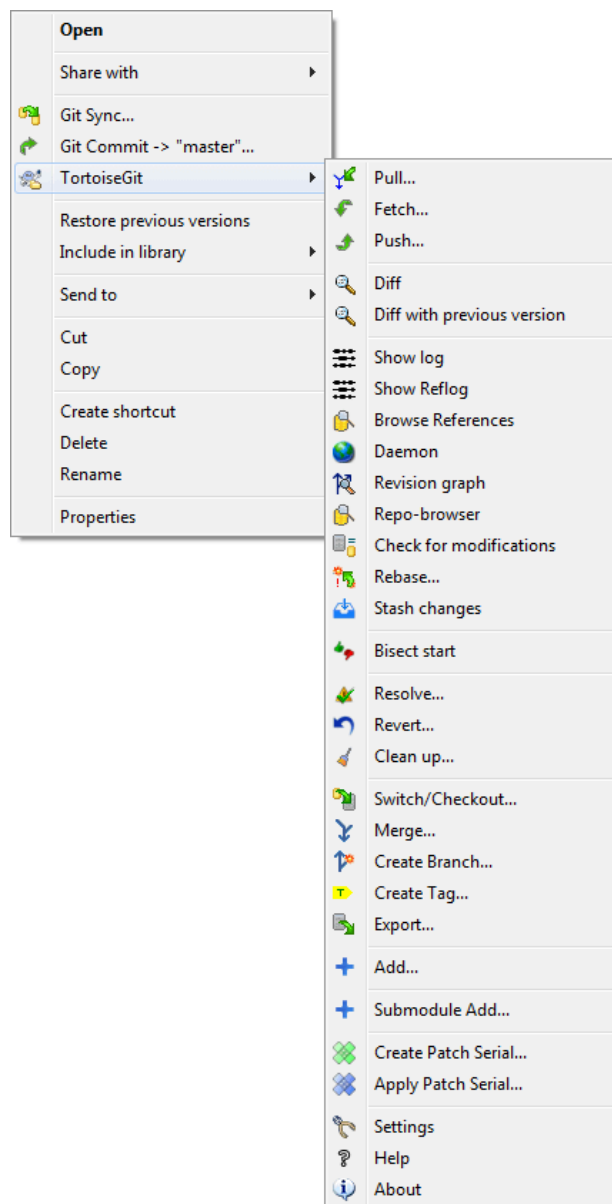
7.2. TortoiseGit

TortoiseGit, Git客户端, 32/64位最新版及对应的语言包下载地址: <https://tortoiseGit.org/download/>

资源管理器中的图标集成



资源管理器的右键菜单集成



7.3. Github

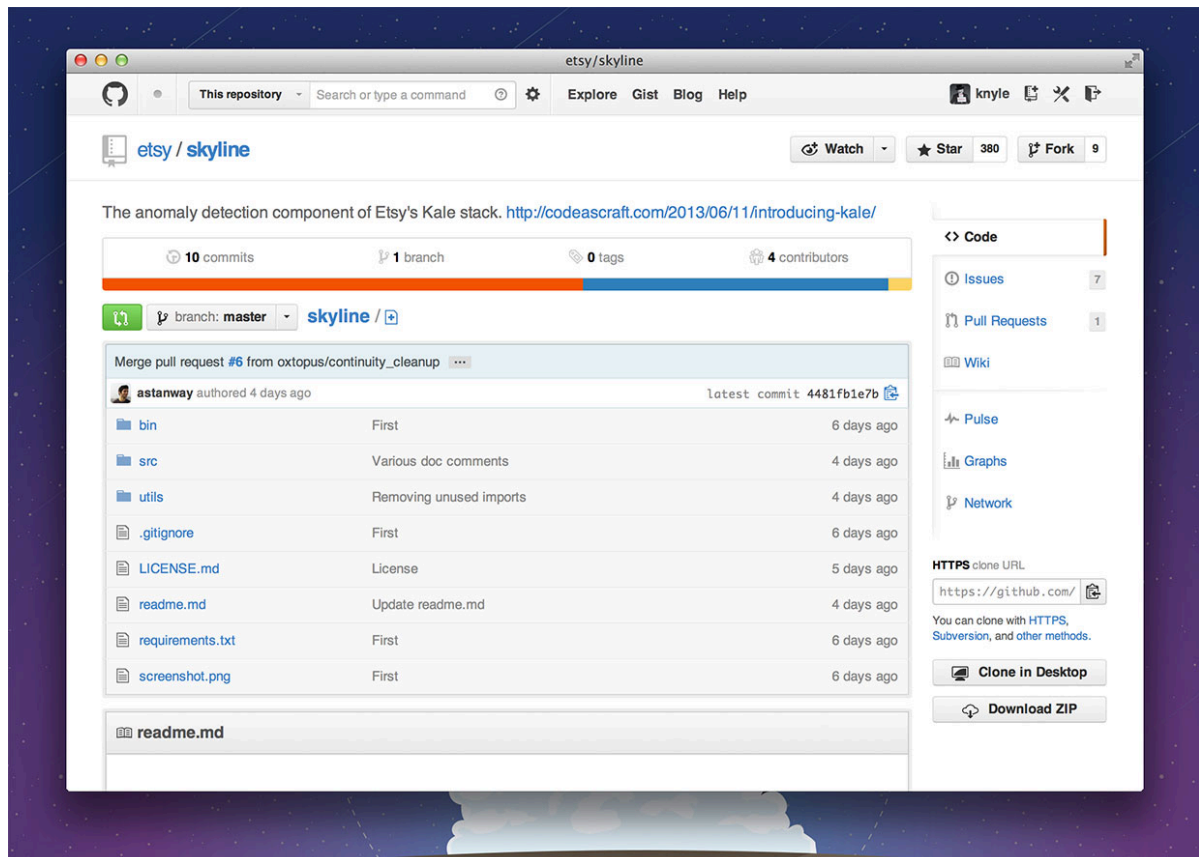
Github是一个面向开源及私有软件项目的托管平台，因为只支持Git 作为唯一的版本库格式进行托管，故名Github。

Github于2008年4月10日正式上线，除了Git代码仓库托管及基本的 Web管理界面以外，还提供了订阅、讨论组、文本渲染、在线文件编辑器、协作图谱（报表）、代码片段分享（Gist）等功能。目前，其注册用户已经超过350万，托管版本数量也是非常之多，其中不乏知名开源项目 Ruby on Rails、jQuery、python 等。

2018年6月4日，微软宣布，通过75亿美元的股票交易收购代码托管平台Github。

<https://Github.com/>

GitHub

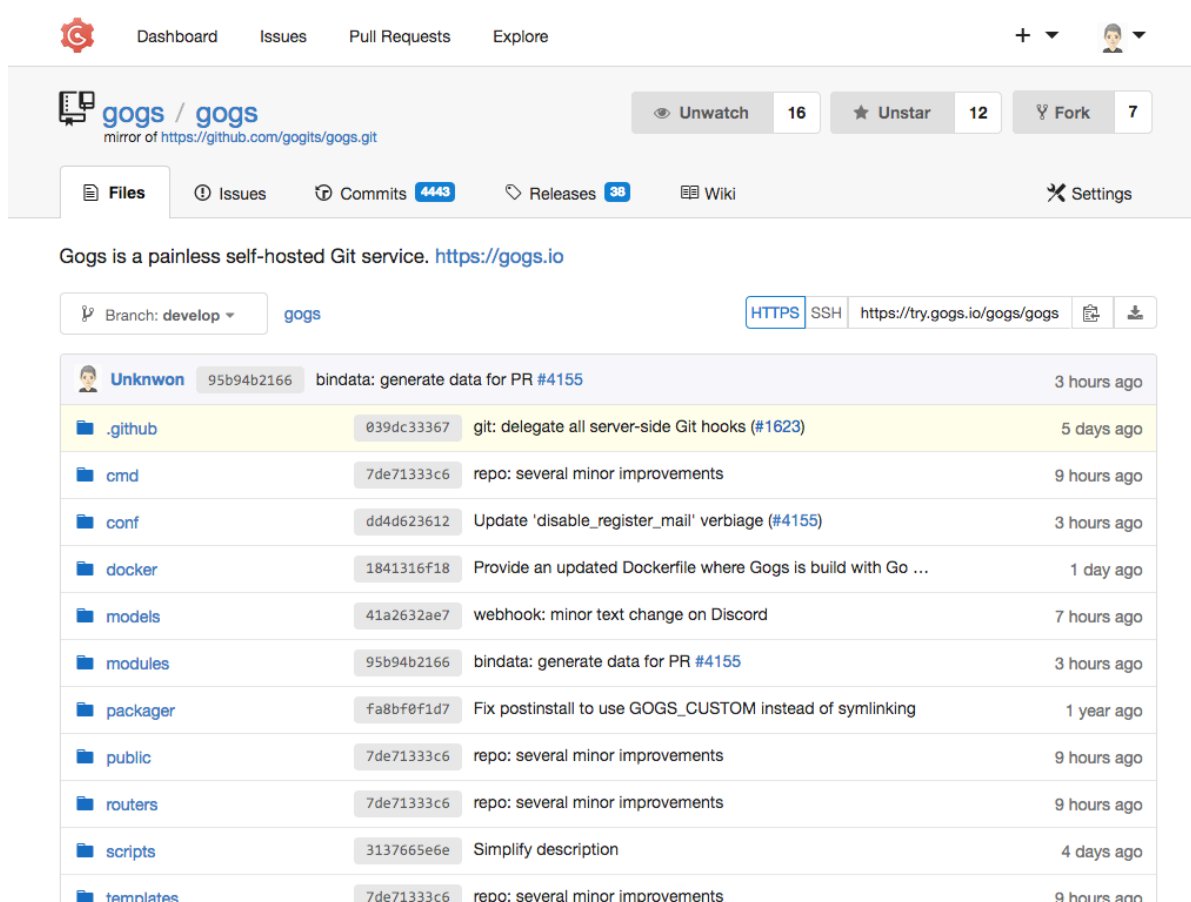


8. 使用Gogs搭建自助 Git 服务

8.1. 什么是 Gogs?

Gogs 是一款极易搭建的自助 Git 服务。

Gogs 的目标是打造一个最简单、最快速和最轻松的方式搭建自助 Git 服务。使用 Go 语言开发使得 Gogs 能够通过独立的二进制分发，并且支持 Go 语言支持的 **所有平台**，包括 Linux、Mac OS X、Windows 以及 ARM 平台。



8.2. 环境要求

- 数据库（选择以下一项）：
 - **MySQL**：版本 ≥ 5.7
 - **PostgreSQL**
 - **MSSQL**
 - **TiDB**（实验性支持，使用 MySQL 协议连接）
 - 或者 **什么都不安装** 直接使用 SQLite3
- Git
 - (bash)：
 - 服务端和客户端均需版本 $\geq 1.7.1$
 - Windows 系统建议使用最新版
- SSH 服务器：
 - **如果您只使用 HTTP/HTTPS 的话请忽略此项**

- 如果您选择在 Windows 系统使用内置 SSH 服务器，请确保添加 `ssh-keygen` 到您的 `%PATH%` 环境变量中
- 推荐 Windows 系统使用 `Cygwin OpenSSH` 或 `Copssh`

8.3. 配置与运行

8.3.1. 安装git

```
1 curl https://setup.ius.io | sudo sh
2 sudo yum remove -y git | sudo yum -y install git2u
```

8.3.2. 默认配置文件

默认配置都保存在 `conf/app.ini`，您 **永远不需要** 编辑它。该文件从 `v0.6.0` 版本开始被嵌入到二进制中。

8.3.3. 自定义配置文件

那么，在不允许修改默认配置文件 `conf/app.ini` 的情况下，怎么才能自定义配置呢？很简单，只要创建 `custom/conf/app.ini` 就可以！在 `custom/conf/app.ini` 文件中修改相应选项的值即可。

例如，需要改变仓库根目录的路径：

```
1 [repository]
2 ROOT = /home/jiahuachen/gogs-repositories
```

当然，您也可以修改数据库配置：

```
1 [database]
2 PASSWD = root
```

8.4. 运行 Gogs 服务

8.4.1. 开发者模式

- 您需要在 `custom/conf/app.ini` 文件中将选项 `security -> INSTALL_LOCK` 的值设置为 `true`。
- 您可以使用超能的 `make` 命令：

```
1 $ make
2 $ ./gogs web
```

8.4.2. 部署模式

脚本均放置在 `scripts` 目录，但请在仓库根目录执行它们

- Gogs 支持多种方式的启动：
 - 普通：只需执行 `./gogs web`
 - 守护进程：详见 `scripts` 文件夹
- 然后访问 `/install` 来完成首次运行的配置工作

8.5. 其它要点

```
1 # 后台启动gogs
2 nohup /work/gogs/gogs web > /work/gogs/log/gogs_web.log 2>&1 &
3 # 访问: http://192.168.99.102:3000/
```

8.6. 类似解决方案

- <https://Gitea.io/>
- <https://Gitlab.com/>

9. Git目录级别的管理

9.1. 思路

Git无法做到目录级别的权限控制。只能使用 **submodule** 或者 **subtree** 实现模块化的git管理。

使用submodule将项目模块化，通过第三方托管平台(**Gogs**)给不同的子模块赋予不同的权限。

9.2. 实现

现有项目Project1，分为3大模块， module1, module2, module3。

在Gogs上分别创建module1, module2, module3三个Git仓库 ,同时创建Project1仓库。将Project1克隆到本地工作目录后进行如下操作：

```
1 git submodule add [url:module1.git] module1 # 这时会在Project1目录下生成module1文件夹，
   里面存放的就是module1.git的所有内容
2
3 git submodule add [url:module2.git] module2 # 这时会在Project1目录下生成module2文件夹，
   里面存放的就是module2.git的所有内容
4
5 git submodule add [url:module3.git] module3 #这时会在Project1目录下生成module3文件夹，
   里面存放的就是module3.git的所有内容
```

最后通过Gogs的权限分配，不同角色分配不同Git库的权限，达到代码目录隔离的效果。

最终编译的时候使用如下命令获取三个Git库的更新：

```
1 cd Project1
2 git submodule foreach git pull
```

10. GitLab

很多程序员在内网搭建 gitlab 都搭建的坑坑洼洼，不支持 https，或者装个 gitlab 就把服务器弄得乱七八糟的，根本不知道该怎么维护和迁移。

然后其他人按照 ssh 的协议来克隆的刀耕火种的方法，还有项目用 php 写 fastcgi 来提供 git 服务。真的有那么麻烦么？正确使用 Docker 搭建 Gitlab 明明就是半分钟的事情。


```
1  docker run \  
2  -itd \  
3  -p 9980:80 \  
4  -p 9922:22 \  
5  -v /usr/local/gitlab-test/etc:/etc/gitlab \  
6  -v /usr/local/gitlab-test/log:/var/log/gitlab \  
7  -v /usr/local/gitlab-test/opt:/var/opt/gitlab \  
8  --restart always \  
9  --privileged=true \  
10 --name gitlab-test \  
11 gitlab/gitlab-ce
```

命令解释:

- -i 以交互模式运行容器，通常与 -t 同时使用命令解释:
- -t 为容器重新分配一个伪输入终端，通常与 -i 同时使用
- -d 后台运行容器，并返回容器ID -p 9980:80 将容器内80端口映射至宿主机9980端口，这是访问gitlab的端口
- -p 9922:22 将容器内22端口映射至宿主机9922端口，这是访问ssh的端口
- -v /usr/local/gitlab-test/etc:/etc/gitlab 将容器/etc/gitlab目录挂载到宿主机/usr/local/gitlab-test/etc目录下，若宿主机内此目录不存在将会自动创建，其他两个挂载同这个一样
- --restart always 容器自启动
- --privileged=true 让容器获取宿主机root权限
- --name gitlab-test 设置容器名称为gitlab-test
- gitlab/gitlab-ce 镜像的名称，这里也可以写镜像ID

11. Gitea

Gitea 是一个轻量级的 DevOps 平台软件。从开发计划到产品成型的整个软件生命周期，他都能够高效而轻松的帮助团队和开发者。包括 Git 托管、代码审查、团队协作、软件包注册和 CI/CD。它与 GitHub、Bitbucket 和 GitLab 等比较类似。Gitea 最初是从 Gogs 分支而来，几乎所有代码都已更改。

11.1. 安装

Gitea使用go开发，全平台兼容，安装非常简单

在Docker中安装Gitea

```
1  docker run --name gitea -d -p 3000:3000 gitea/gitea
```

配置Gitea，如果就是普通使用一下，啥都不用改

初始配置

如果您正在使用 Docker 容器运行 Gitea，请务必先仔细阅读 [官方文档](#) 后再对本页面进行填写。

数据库设置

Gitea 需要使用 MySQL、PostgreSQL、MSSQL、SQLite3 或 TiDB (MySQL协议) 等数据库

数据库类型 *

SQLite3

数据库文件路径 *

/data/gitea/gitea.db

SQLite3 数据库的文件路径。
如果以服务的方式运行 Gitea，请输入绝对路径。

一般设置

站点名称 *

Gitea: Git with a cup of tea

您可以在此输入您公司的名称。

仓库根目录 *

/data/git/repositories

所有远程 Git 仓库将保存到此目录。

LFS根目录

/data/git/lfs

存储为Git LFS的文件将被存储在此目录。留空禁用LFS

以用户名运行 *

git

输入 Gitea 运行的操作系统用户名。请注意，此用户必须具有对仓库根路径的访问权限。

服务器域名 *

192.168.56.104

服务器的域名或主机地址。

SSH 服务端口

22

SSH 服务器的端口号，为空则禁用它。

HTTP 服务端口 *

3000

Gitea web 服务器将侦听的端口号。

基础URL *

http://192.168.56.104:3000/

用于 HTTP (S) 克隆和电子邮件通知的基本地址。

日志路径 *

/data/gitea/log

日志文件将写入此目录。

☐ 启用更新检查

通过连接到 gitea.io 定期检查新版本发布。

可选设置

▶ 电子邮箱设置

▶ 服务器和第三方服务设置

▶ 管理员帐号设置

These configuration options will be written into: /data/gitea/conf/app.ini

立即安装

然后注册用户，第一个注册的用户即超管，然后就可以正常使用了。

11.2. Actions

Gitea支持actions，可以参考：[Gitea Actions Quick Start](#)




注意token在这里 </settings/actions/runners>








```
1 ./act_runner register --no-interactive --instance http://192.168.56.104:3000/ --  
  token P2U1U0oB4XaRCi8azcngmPCLbRpUGapalhmdh23  
2 ./act_runner daemon
```



color_tasks.gitea\workflows\demo.yaml


```
1  name: Gitea Actions Demo  
2  run-name: ${ gitea.actor }} is testing out Gitea Actions 🚀  
3  on: [push]  
4  
5  jobs:  
6    Explore-Gitea-Actions:  
7      runs-on: ubuntu-latest  
8      steps:  
9        - run: echo "🎉 The job was automatically triggered by a ${ gitea.event_name }} event."  
10       - run: echo "🐙 This job is now running on a ${ runner.os }} server hosted by Gitea!"  
11       - run: echo "🔗 The name of your branch is ${ gitea.ref }} and your repository is ${ gitea.repository }}."  
12       - name: Check out repository code  
13         uses: actions/checkout@v4  
14       - run: echo "💡 The ${ gitea.repository }} repository has been cloned to the runner."  
15       - run: echo "💻 The workflow is now ready to test your code on the runner."  
16       - name: List files in the repository  
17         run: |  
18           ls ${ gitea.workspace }}  
19       - run: echo "🍏 This job's status is ${ job.status }}."
```




lgc653 / color_tasks 


 取消关注 1  点赞 0  0


 代码  工单  合并请求 ...


 **Gitea Actions** 重新运行所有任务


demo.yaml: 提交 71e3329873 推送者 lgc653 master


 **Explore-Gitea-Actions** 32s


Explore-Gitea-Actions
成功 


>  Set up job 23s


>  echo " 🇬🇧 The job was automatically t... 0s

>  echo " 🐧 This job is now running on ... 0s

>  echo " 🌐 The name of your branch is... 0s

>  Check out repository code 1s

>  echo " 💡 The \${{ gitea.repository }} r... 0s

>  echo " 🖥 The workflow is now ready ... 0s

11.3. 总结

- 比Gogs功能全面且持续更新
- 比GitLab轻量级，系统要求极低
- 小团队自建DevOps平台的最佳选择