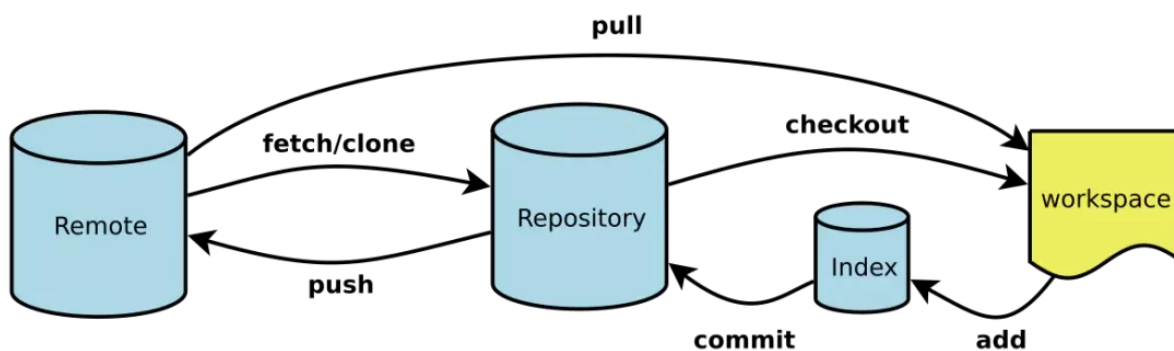


Git Notes

Git Notes

- 1 Git 简介
- 2 Git 基础操作
 - 2.1 Git 安装
 - 2.2 Git 配置
 - 2.3 Git 工作流程
 - 2.4 Git 工作区、暂存区和版本库
 - 2.3 新建仓库
 - 2.4 添加 / 删除文件
 - 2.5 代码提交
 - 2.6 代码版本 / 提交切换
- 3 Git 分支管理
- 4 Git 标签
- 5 远程仓库
- 6 Git 自定义
 - 6.1 忽略特殊文件
 - 6.2 配置别名
 - 6.3 Git 代理设置
 - 6.4 搭建 Git 服务器
 - 6.5 Pycharm 的 Terminal 使用 Git Bash

相关参考



Git 流程图

- Workspace: 工作区
- Index / Stage: 暂存区
- Repository: 仓库区 (或本地仓库)
- Remote: 远程仓库

1 Git 简介



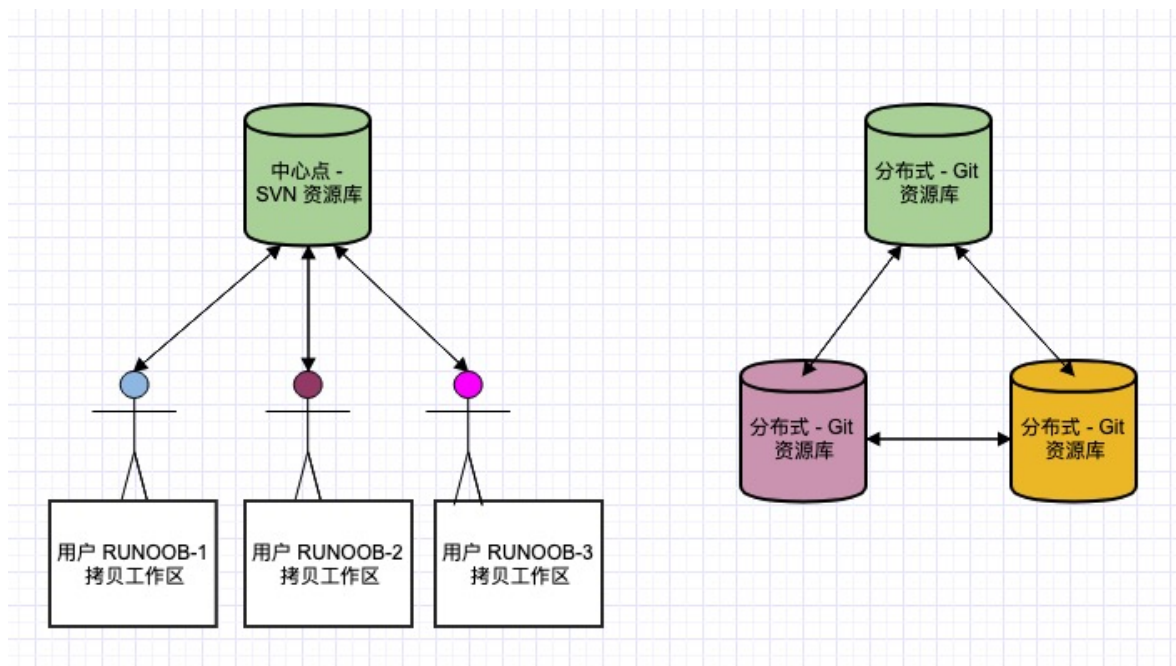
Git 是一个开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。

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。

Git 与常用的版本控制工具 CVS, Subversion 等不同，它采用了分布式版本库的方式，不必服务器端软件支持。

Git 与 SVN 区别点：

- 1、**Git 是分布式的，SVN 不是**：这是 Git 和其它非分布式的版本控制系统，例如 SVN，CVS 等，最核心的区别。
- 2、**Git 把内容按元数据方式存储，而 SVN 是按文件**：所有的资源控制系统都是把文件的元信息隐藏在一个类似 .svn、.cvs、.git 等的文件夹里。
- 3、**Git 分支和 SVN 的分支不同**：分支在 SVN 中一点都不特别，其实它就是版本库中的另外一个目录。
- 4、**Git 没有一个全局的版本号，而 SVN 有**：目前为止这是跟 SVN 相比 Git 缺少的最大的一个特征。
- 5、**Git 的内容完整性要优于 SVN**：Git 的内容存储使用的是 SHA-1 哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。



2 Git 基础操作

2.1 Git 安装

Git 目前支持 Linux/Unix、Solaris、Mac和 Windows 平台上运行。

Git 各平台安装包下载地址为：<https://git-scm.com/downloads>

2.2 Git 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）。

```
1  # 显示当前的 Git配置
2  $ git config --list
3
4  # 编辑 Git 配置文件
5  $ git config -e [--global]
6
7  # 设置提交代码时的用户信息
8  $ git config [--global] user.name "[name]"
9  $ git config [--global] user.email "[email address]"
```

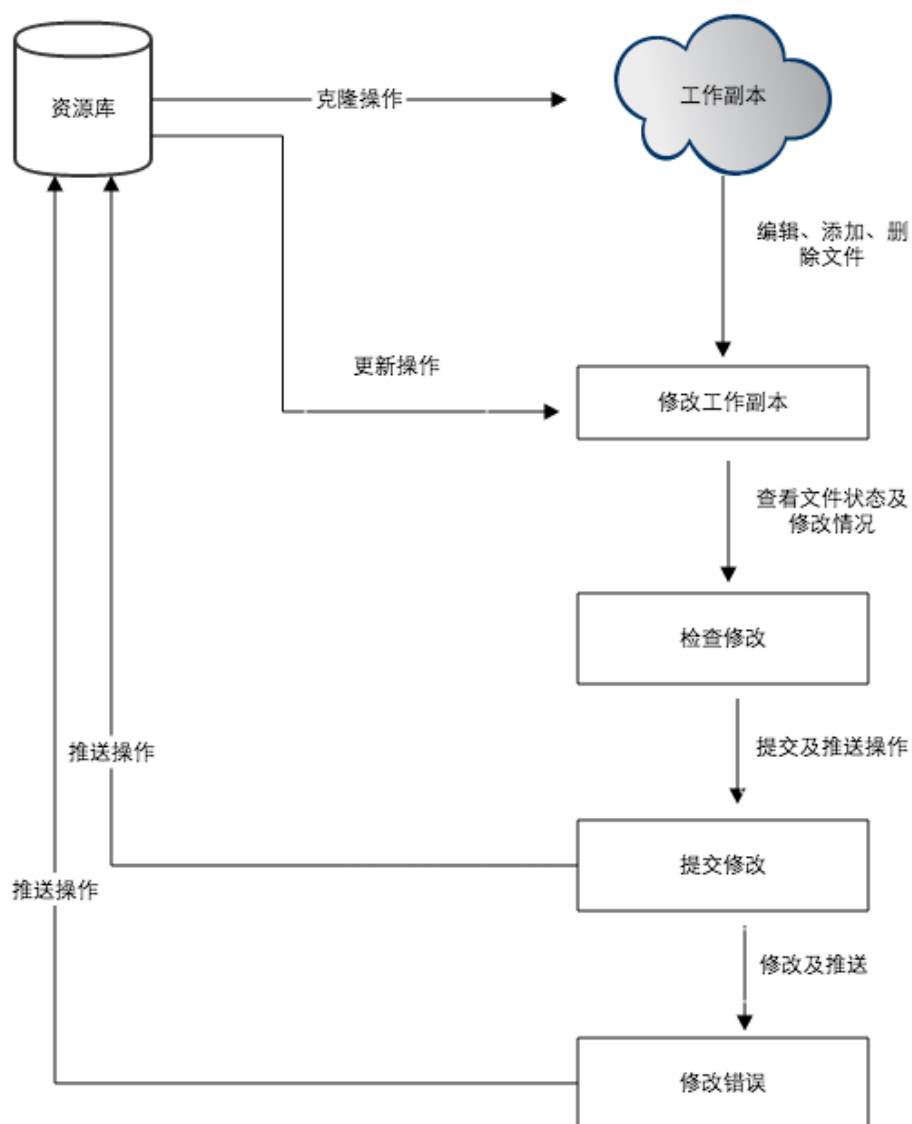
2.3 Git 工作流程

一般工作流程如下：

- 克隆 Git 资源作为工作目录。
- 在本地的资源上添加或修改文件。
- 如果其他人修改了，你可以更新资源。
- 在提交前查看修改。
- 提交修改。
- 在修改完成后，如果发现错误，可以撤回提交并再次修改并提交。

下图展示了 Git 的工作流程：

Git 工作流程



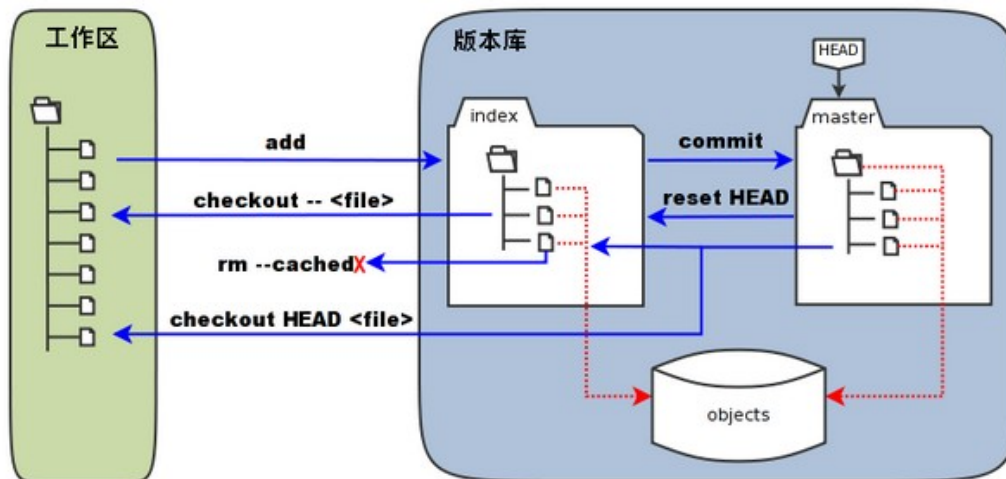
菜鸟教程: <http://www.runoob.com>

2.4 Git 工作区、暂存区和版本库

我们先来了解下 Git 工作区、暂存区和版本库概念

- **工作区**：就是你在电脑里能看到的目录。
- **暂存区**：英文叫stage, 或index。一般存放在 ".git目录下" 下的index文件 (.git/index) 中，所以我们把暂存区有时也叫作索引 (index) 。
- **版本库**：工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

下面这个图展示了工作区、版本库中的暂存区和版本库之间的关系：



图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区 (stage, index)，标记为 "master" 的是 master 分支所代表的目录树。

图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个"游标"。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。

图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。

当对工作区修改（或新增）的文件执行 "git add" 命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。

当执行提交操作 (git commit) 时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。

当执行 "git reset HEAD" 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。

当执行 "git rm --cached <file>" 命令时，会直接从暂存区删除文件，工作区则不做出改变。

当执行 "git checkout ." 或者 "git checkout -- <file>" 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。

当执行 "git checkout HEAD ." 或者 "git checkout HEAD <file>" 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

2.3 新建仓库

有两种新建 Git 项目仓库的方法。第一种是在现有项目或目录下导入所有文件到 Git 中；第二种是从一个服务器克隆一个现有的 Git 仓库。

1. 在现有目录中初始化仓库

```
1 $ git init
```

2. 克隆现有的仓库

```
1 # 使用 ssh 克隆
2 $ git clone git@gitee.com:Auto_SK/Git-Notes.git
3
4 # 使用 https 克隆
5 $ git clone https://gitee.com/Auto_SK/Git-Notes.git
```

2.4 添加 / 删除文件

```
1 # 添加指定文件到暂存区
2 $ git add file1 file2 ...
3
4 # 添加指定目录到暂存区, 包括子目录
5 $ git add dir
6
7 # 添加当前目录的所有文件到暂存区
8 $ git add .
9
10 # 添加每个变化前, 都会要求确认
11 # 对于同一个文件的多处变化, 可以实现分次提交
12 $ git add -p
13
14 # 删除工作区文件, 并且将这次删除放入暂存区
15 $ git rm file1 file2 ...
16
17 # 停止追踪指定文件, 但该文件会保留在工作区
18 $ git rm --cached file
19
20 # 改名文件, 并且将这个改名放入暂存区
21 $ git mv file-original file-renamed
```

2.5 代码提交

```
1 # 提交暂存区到仓库区
2 $ git commit -m "message"
3
4 # 提交暂存区的指定文件到仓库区
5 $ git commit file1 file2 ... -m "message"
6
7 # 提交工作区自上次commit之后的变化, 直接到仓库区
8 $ git commit -a
9
10 # add和commit的合并, 便捷写法 (未追踪的文件无法直接提交到暂存区/本地仓库)
11 $ git commit -am "message"
12
13 # 提交时显示所有diff信息
14 $ git commit -v
15
16 # 使用一次新的commit, 替代上一次提交
17 # 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
18 $ git commit --amend -m "message"
19
20 # 重做上一次commit, 并包括指定文件的新变化
21 $ git commit --amend file1 file2 ...
22
23 # 查看工作区和暂存区的状态
24 $ git status
```

2.6 代码版本 / 提交切换

1. 查看过去版本/提交

```
1 # 提交详情
2 $ git log
3
4 # 提交简介
5 $ git log --pretty=oneline
6
7 # 图形化查看提交
8 $ gitk
```

2. 回退版本/提交

```
1 # 回退到当前最新提交
2 $ git reset --hard HEAD
3
4 # 回退到上次提交
5 $ git reset --hard HEAD^
6
7 # 回退到上n次提交
8 $ git reset --hard HEAD~n
9
10 # 回退到某次提交
11 $ git reset --hard commit_id
```

3. 重返未来版本

```
1 # 查看历史提交以及被回退的提交，注意：该记录有时限，且只在本地
2 $ git reflog
3
4 # 回到未来版本
5 $ git reset --hard commit_id
```

4. 撤销修改

```
1 # 工作区文件撤销 没有提交到暂存区即没有 git add
2 # 撤销修改
3 $ git checkout file
4
5 # 暂存区文件撤销
6 # 将暂存区文件撤销到工作区
7 $ git reset HEAD file
8 # 撤销修改
9 $ git checkout file
```

3 Git 分支管理

1. 新建分支

```
1 # 创建分支
2 $ git branch branchname
3
4 # 创建分支并立即切换到该分支下
5 $ git checkout -b branchname
```

2. 切换分支

```
1 $ git checkout branchname
```

3. 删除分支

```
1 $ git branch -d branchname
```

4. 分支合并

```
1 $ git merge [branchname]
```

4 Git 标签

```
1 $ git tag -a version -m "version_message"
```

5 远程仓库

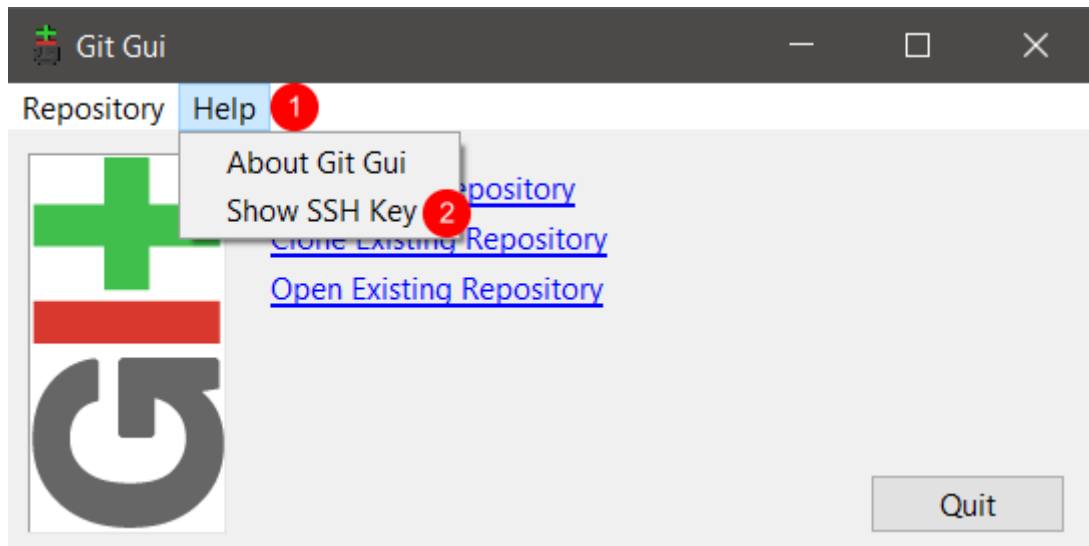
1. 生成 SSH Key

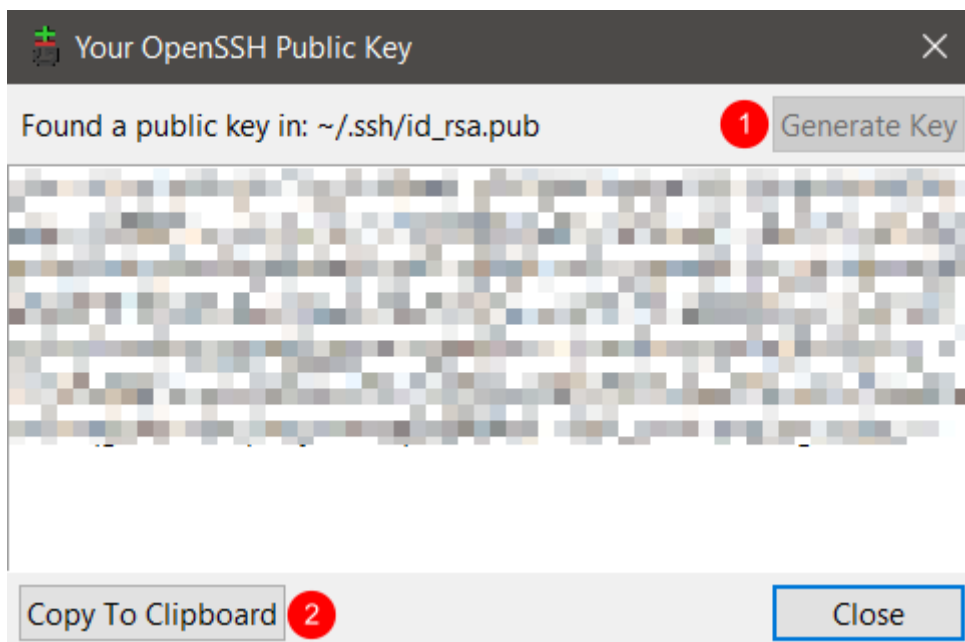
在 Terminal 中生成 SSH Key

```
1 $ ssh-keygen -t rsa -C "youremail@example.com"
```

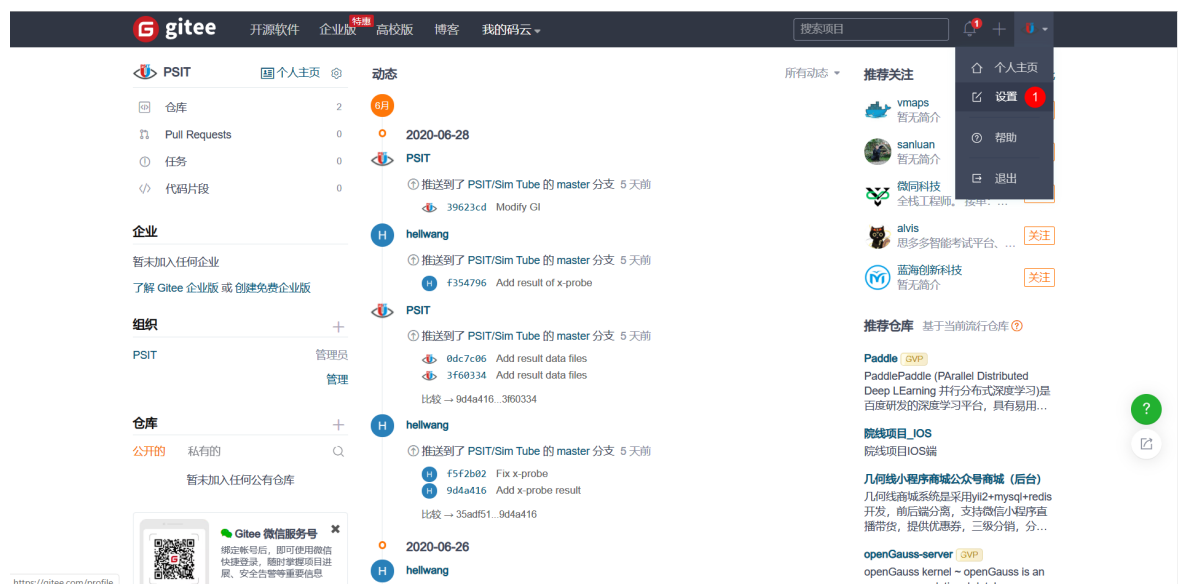
后面的 `your_email@youremail.com` 改为你 Gitee 账户的邮箱，之后会要求确认路径和输入密码，我们这使用默认的一路回车就行。打开 `~/.ssh` 下的 `id_rsa.pub`，复制里面的 key。

或使用 Git Gui 生成 SSH Key





进入 Gitee 的设置界面



左选 SSH 公钥选项，在公钥位置粘贴复制的 key，然后点击确定。



2. 添加远程仓库

```
1 $ git remote add origin git@gitee.com:Auto_SK/Git-Notes.git
```

3. 远程仓库管理

```
1 # 将本地分支和远程分支进行关联
2 $ git push -u origin branchName
3
4 # 将本地仓库的文件推送到远程分支
5 $ git push
6
7 # 将本地仓库的指定分支推送到远程仓库
8 $ git push origin branchName
9
10 # 拉取远程分支的代码
11 $ git pull origin branchName
```

6 Git 自定义

6.1 忽略特殊文件

在Git工作区的根目录下创建一个特殊的 .gitignore 文件，然后把要忽略的文件名填进去，Git就会自动忽略这些文件。Github 的 Git 忽略模板 <https://github.com/github/gitignore>

```
1 # 忽略指定文件
2 HelloWorld.py
3
4 # 忽略指定文件夹
5 bin/
6 bin/gen/
7
8 # 忽略所有的 .pyc 文件
9 *.pyc
10
11 # 忽略名称中末尾为 ignore 的文件夹
12 *ignore/
13
14 # 忽略名称中间包含 ignore 的文件夹
15 *ignore*/
16
17 # 在已经忽略的文件类型中不忽略指定文件
18 !bin/solver.exe
```

6.2 配置别名

```

1  # 配置别名
2  $ git config --global alias.co checkout
3  $ git config --global alias.ss status
4  $ git config --global alias.cm commit
5  $ git config --global alias.br branch
6  $ git config --global alias.rg reflog
7  # 这里只是美化 log 的输出, 实际使用时可以在 git lg 后面加命令参数, 如: git lg -10 显示最近10
   条提交
8  $ git config --global alias.lg "log --color --graph --
   pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)
   <%an>%Creset' --abbrev-commit"

```

6.3 Git 代理设置

1. http代理

```

1  # 走 http 代理
2  # Port 改为代理工具代理的端口
3  $ git config --global http.proxy "http://127.0.0.1:Port"
4  $ git config --global https.proxy "http://127.0.0.1:Port"
5
6  # 走 socks5 代理
7  # Port 改为代理工具代理的端口
8  $ git config --global http.proxy "socks5://127.0.0.1:Port"
9  $ git config --global https.proxy "socks5://127.0.0.1:Port"
10
11 # 取消设置
12 $ git config --global --unset http.proxy
13 $ git config --global --unset https.proxy

```

2. ssh代理

修改 ~/.ssh/config 文件（不存在则新建）：

```

1  # 这里的 -a none 是 NO-AUTH 模式, 参见 https://bitbucket.org/gotom/connect/wiki/Home
   中的 More detail 一节
2  # Port 改为代理工具代理的端口
3  ProxyCommand connect -S 127.0.0.1:Port -a none %h %p
4
5  Host github.com
6      # username 改为 GitHub 的邮箱地址
7      User username
8      Port 22
9      Hostname github.com
10     # 注意修改路径为你的路径
11     # username 改为你的用户名
12     IdentityFile "C:\Users\username\.ssh\id_rsa"
13     TCPKeepAlive yes
14
15  Host ssh.github.com
16     # username 改为 GitHub 的邮箱地址
17     User username
18     Port 443
19     Hostname ssh.github.com
20     # 注意修改路径为你的路径
21     # username 改为你的用户名
22     IdentityFile "C:\Users\username\.ssh\id_rsa"

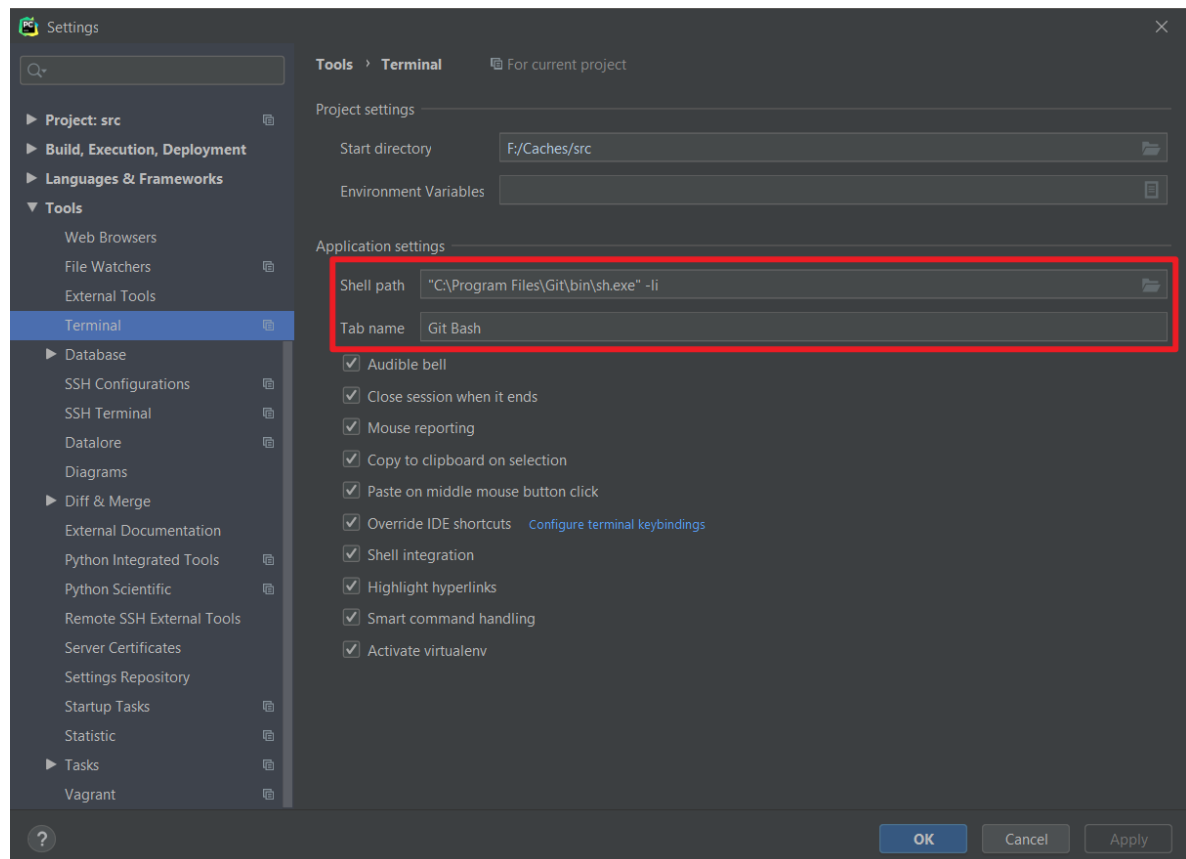
```

6.4 搭建 Git 服务器

使用 [Gitea](#) 搭建服务器，参考 [用 Gitea 搭建自己的 Git 服务器](#)

6.5 Pycharm 的 Terminal 使用 Git Bash

```
1 # 将 Shell Path 改为
2 "C:\Program Files\Git\bin\sh.exe" -li
3 # 将 Tab name 改为
4 Git Bash
```



相关参考

1. [Pro Git](#)
2. [Git 参考手册](#)
3. [Git 思维导图](#)
4. [廖雪峰的 Git 教程](#)
5. [菜鸟教程 | Git 教程](#)
6. [常用 Git 命令清单](#)
7. [A successful Git branching model](#)