

Git学习

姓名：吴文芳

学号：2101210428

Git学习

1. 快速入门

- 1.1 什么是版本控制
- 1.2 什么是Git
- 1.3 Git与SVN
- 1.4 Git与Github

2. Git使用

- 2.1 Git安装与配置
 - 2.1.1 Git安装
 - 2.1.2 Git配置
 - 2.1.3 创建账户
 - 2.1.4 配置SSH Key, 添加SSH Key到GitHub

2.2 概念

- 2.2.1 四个工作区域
- 2.2.2 工作流程
- 2.2.3 文件的四种状态

2.3 常用命令

- 2.3.1 新建代码库
- 2.3.2 查看文件状态
- 2.3.3 工作区<-->暂存区
- 2.3.4 工作区<-->资源库（版本库）
- 2.3.5 远程操作
- 2.3.6 其他常用命令

2.4 本地仓库

- 2.4.1 创建本地仓库
- 2.4.2 基本操作
 - 跟踪文件
 - 添加文件
 - 删除文件
 - 查看状态
 - 查看修改
 - 版本回退

2.5 远程仓库

- 2.5.1 创建远程仓库
- 2.5.2 克隆远程仓库
- 2.5.3 上传本地项目

2. Git协议

- 2.1 本地协议
- 2.2 HTTP协议
 - 2.2.1 智能HTTP 协议
 - 2.2.2 哑HTTP 协议
- 2.3 SSH 协议
- 2.4 Git 协议

3. Git使用小神器

- 3.1 搜索小妙招
- 3.2 GitHub插件

1. 快速入门

1.1 什么是版本控制

版本控制（Revision control）是一种在开发的过程中用于管理我们对文件、目录或工程等内容的修改历史，方便查看更改历史记录，备份以便恢复以前的版本的软件工程技术。

- 实现跨区域多人协同开发
- 追踪和记载一个或者多个文件的历史记录
- 组织和保护你的源代码和文档
- 统计工作量
- 并行开发、提高开发效率
- 跟踪记录整个软件的开发过程
- 减轻开发人员的负担，节省时间，同时降低人为错误

没有进行版本控制或者版本控制本身缺乏正确的流程管理，在软件开发过程中将会引入很多问题，如软件代码的一致性、软件内容的冗余、软件过程的事物性、软件开发过程中的并发性、软件源代码的安全性，以及软件的整合等问题。

1.2 什么是Git

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。

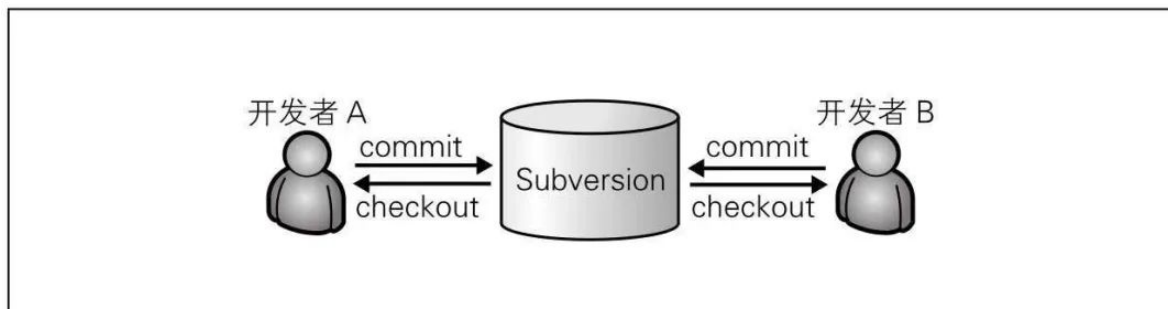
Git 与常用的版本控制工具 CVS, Subversion 等不同，它采用了分布式版本库的方式，不必服务器端软件支持。

Git 不仅仅是个版本控制系统，它也是个内容管理系统(CMS)，工作管理系统等。

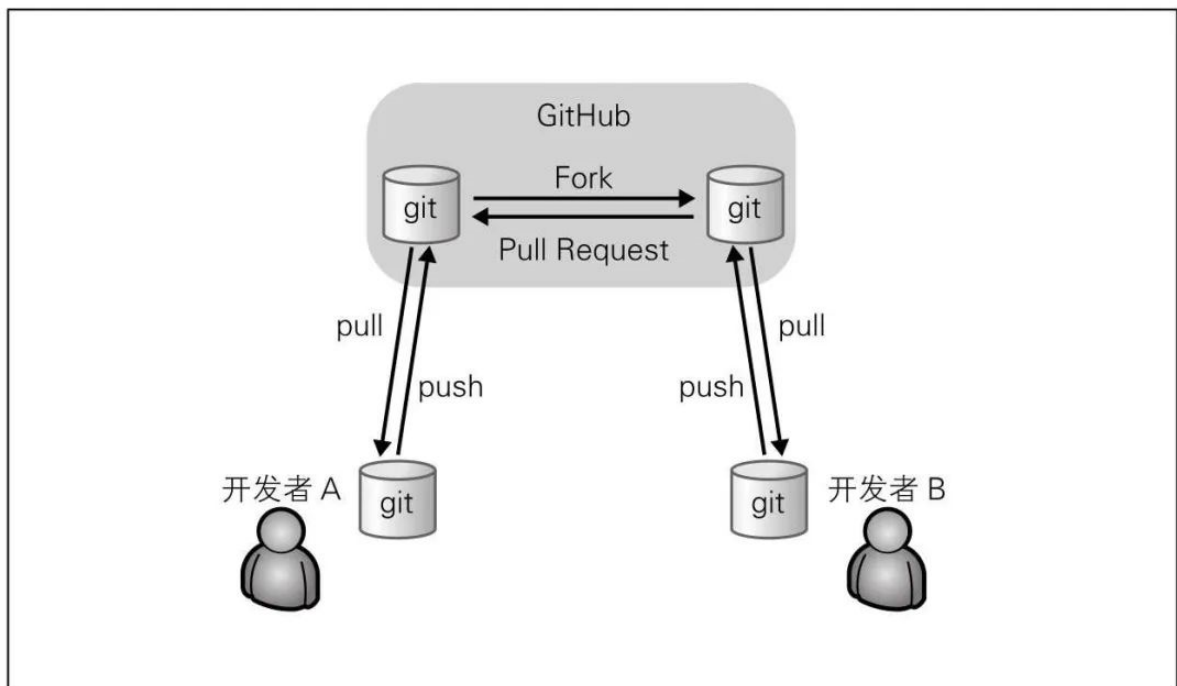
1.3 Git与SVN

Git出现之前，大家主要是使用Subversion（简称为SVN）作为版本控制的工具。

SVN是属于集中型的版本管理系统，其特点是将仓库集中存放在服务器中，所以只存在一个仓库。集中型将所有特点是方便管理，但是如果开发者所处的环境无法联网，则无法获取到最新的源码，进而无法进行开发工作。



Git是**分散型**的版本管理系统。从下图中我们可以观察出来，GitHub将仓库fork给每个用户。fork的仓库和原始的仓库是两个不同的仓库，开发者是可以随意编辑的。



1.4 Git与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。

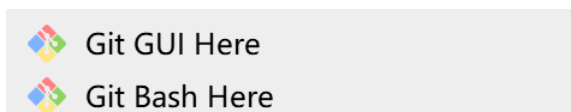
如果是付费用户可以建立自己的私人仓库，一般用户的话只能建立公用仓库，也就是说仓库的代码必须是公开的。

2. Git使用

2.1 Git安装与配置

2.1.1 Git安装

从 Git 官网下载安装程序，根据指引完成安装。安装完成后，在开始菜单中会有Git项，菜单下有2个程序：



选择Git Bash Here，输入git --version，如果出现如下图的情况，即安装成功。

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/桌面
$ git --version
git version 2.34.0.windows.1
```

2.1.2 Git配置

配置个人的用户名称和电子邮件地址。

```
// 设置用户名、邮箱
$ git config --global user.name "dfdg777"
$ git config --global user.email "2849805393@qq.com"
```

配置成功后，查看配置：

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/桌面
$ git config --global -l
core.editor="D:\下载软件\VS code\Microsoft VS Code\bin\code" --wait
user.name=dfdg777
user.email=2849805393@qq.com
http.sslverify=
https.proxy=https://
```

2.1.3 创建账户

进入创建用户的页面: <http://github.com/join>, 按界面提示填写相关信息再点击Create an account即可。

2.1.4 配置SSH Key, 添加SSH Key到GitHub

GitHub上连接现有仓库的认证, 是通过使用了SSH的公开密钥认证方式进行的。需要创建公开密钥所需的SSH Key, 并将其添加到GitHub。

```
ssh-keygen -t rsa -C # 创建SSH Key
```

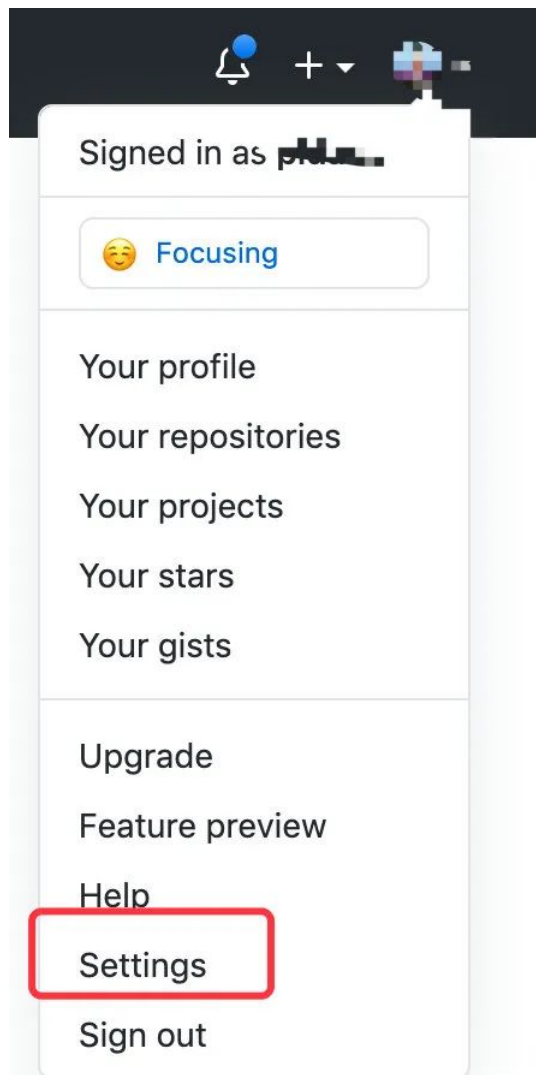
接下来需要输入两次密码, 最终会生成两个文件:

- id_rsa: 私有密钥
- id_rsa.pub: 公开密钥

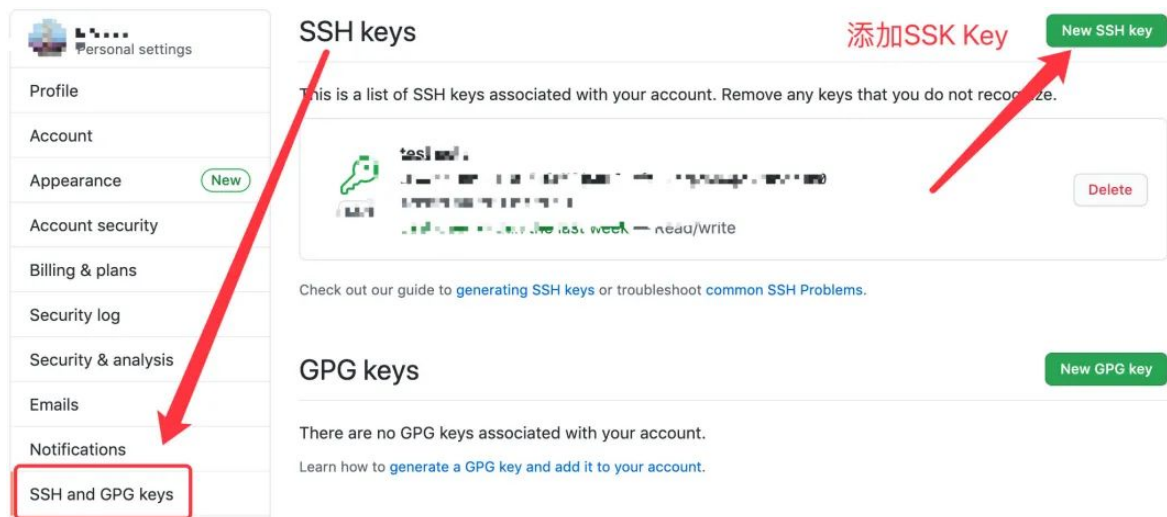
检查SSH Key是否创建成功。若显示如下, 则说明创建成功。

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/桌面
$ ~/.ssh
bash: /c/Users/86186/.ssh: Is a directory
```

然后在GitHub中添加公开密钥, 今后就可以使用私有密钥进行认证。点击右上角的账户设定按钮:



进入settings之后，添加新的SSH Key：



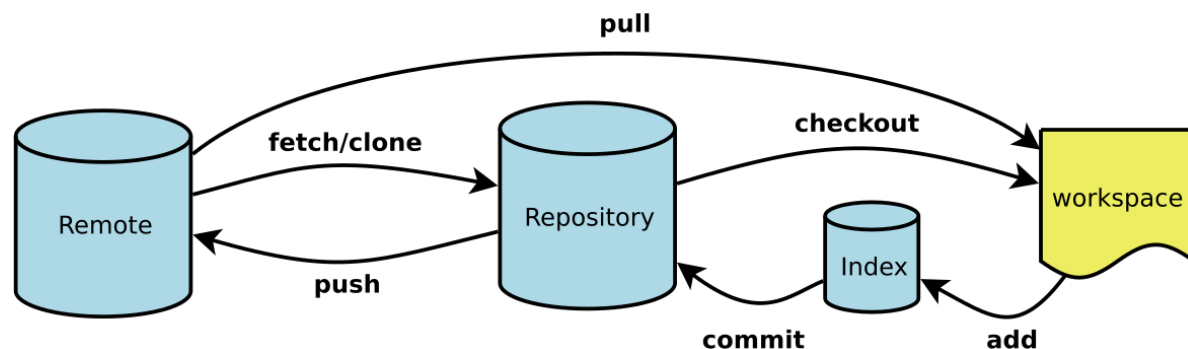
接下来会出现Title和Key两个输入框，在Title中输入适当的密钥名称，Key部分复制上面id_rsa.pub文件中的内容。

添加完成之后，会在我们的邮箱中收到一封提示“公开密钥添加完成”的邮件，确认即可。这样便完成了整个手中的私人密钥和GitHub的认证和通信问题。

2.2 概念

2.2.1 四个工作区域

Git本地有四个工作区域：工作目录（Working Directory）、暂存区(Stage/Index)、资源库(Repository或Git Directory)、git仓库(Remote Directory)。文件在这四个区域之间的转换关系如下：



Workspace: 工作区，平时存放项目代码的地方。

Index/Stage: 暂存区，用于临时存放改动，事实上它只是一个文件，保存即将提交到文件列表信息。

Repository: 仓库区（或版本库），就是安全存放数据的位置，这里面有提交的所有版本的数据。其中HEAD指向最新放入仓库的版本。

Remote: 远程仓库，托管代码的服务器，可以简单的认为是项目组中的一台电脑，用于远程数据交换。

2.2.2 工作流程

Git的工作流程一般为：

- 1、在工作目录中添加、修改文件；
- 2、将需要进行版本管理的文件放入暂存区域；
- 3、将暂存区域的文件提交到git仓库。

因此，Git管理的文件有三种状态：已修改（modified）,已暂存（staged）,已提交(committed)。

2.2.3 文件的四种状态

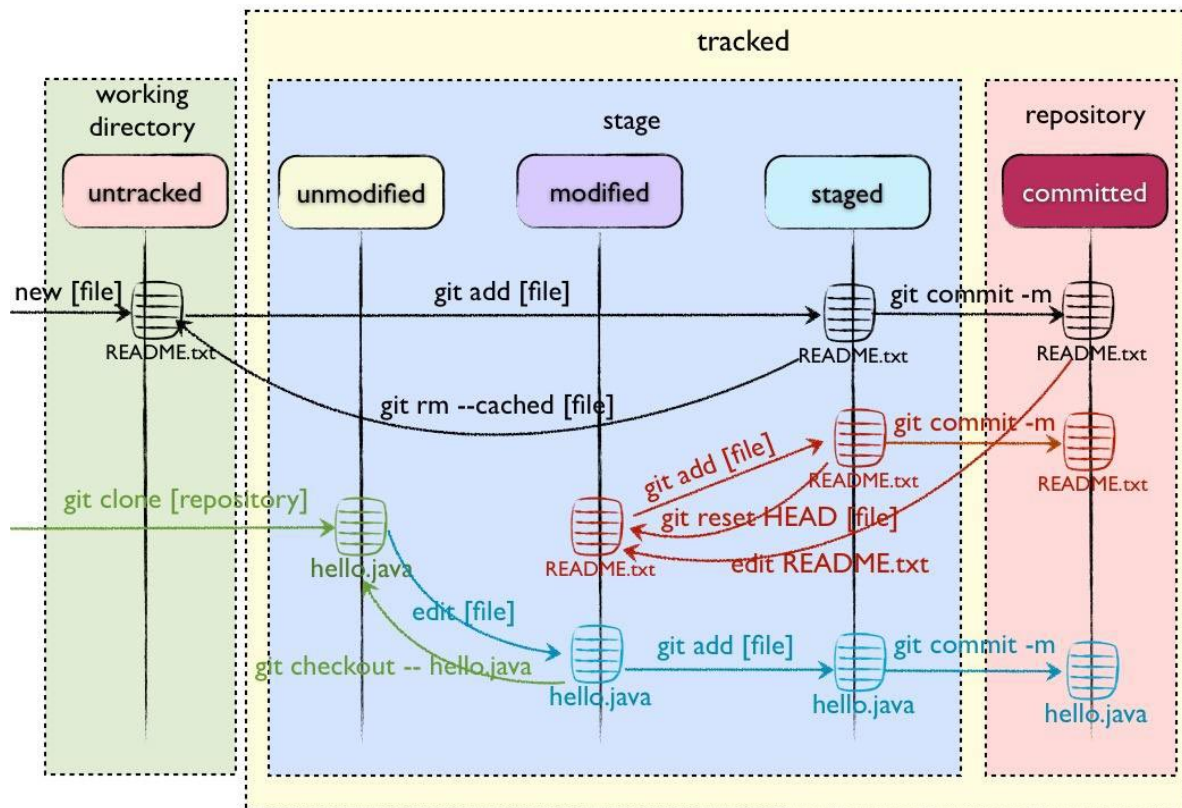
版本控制就是对文件的版本控制，要对文件进行修改、提交等操作，首先要知道文件当前在什么状态，不然可能会提交了现在还不想提交的文件，或者要提交的文件没提交上。

Untracked: 未跟踪, 此文件在文件夹中, 但并没有加入到git库, 不参与版本控制。通过git add 状态变为 Staged.

Unmodify: 文件已经入库，未修改，即版本库中的文件快照内容与文件夹中完全一致。这种类型的文件有两种去处：如果它被修改, 而变为Modified；如果使用git rm移出版本库, 则成为Untracked文件。

Modified: 文件已修改, 仅仅是修改，并没有进行其他的操作。这个文件也有两个去处, 通过git add可进入暂存staged状态；使用git checkout 则丢弃修改，返回到unmodify状态。

Staged: 暂存状态. 执行git commit则将修改同步到库中, 这时库中的文件和本地文件又变为一致, 文件为Unmodify状态。执行git reset HEAD filename取消暂存，文件状态为Modified。



新建文件--->Untracked

使用add命令将新建的文件加入到暂存区--->Staged

使用commit命令将暂存区的文件提交到本地仓库--->Unmodified

如果对Unmodified状态的文件进行修改---> modified

如果对Unmodified状态的文件进行remove操作--->Untracked

2.3 常用命令

2.3.1 新建代码库

```
# 在当前目录新建一个Git代码库
git init
# 新建一个目录，将其初始化为Git代码库
git init [project-name]
# 下载一个项目和它的整个代码历史
git clone [url]
```

2.3.2 查看文件状态

```
#查看指定文件状态
git status [filename]
#查看所有文件状态
git status
```

2.3.3 工作区<-->暂存区

```
# 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 .
#当我们需要删除暂存区或分支上的文件，同时工作区也不需要这个文件了，可以使用。
git rm file_path
#当我们需要删除暂存区或分支上的文件，但本地又需要使用，这个时候直接push那边这个文件就没有，如果push之前重新add那么还是会有。
git rm --cached file_path
#直接加文件名 从暂存区将文件恢复到工作区，如果工作区已经有该文件，则会选择覆盖
#加了【分支名】 +文件名 则表示从分支名为所写的分支名中拉取文件 并覆盖工作区里的文件
git checkout
```

2.3.4 工作区<-->资源库（版本库）

```
#将暂存区-->资源库（版本库）
git commit -m '该次提交说明'
#如果出现:将不必要的文件commit 或者 上次提交觉得是错的 或者 不想改变暂存区内容，只是想调整提交的信息
#移除不必要的添加到暂存区的文件
git reset HEAD 文件名
#去掉上一次的提交（会直接变成add之前状态）
git reset HEAD^
#去掉上一次的提交（变成add之后，commit之前状态）
git reset --soft HEAD^
```

2.3.5 远程操作

```
# 取回远程仓库的变化，并与本地分支合并
git pull
# 上传本地指定分支到远程仓库
git push
```

2.3.6 其他常用命令

```
# 显示当前的Git配置
git config --list
# 编辑Git配置文件
git config -e [--global]
#初次commit之前，需要配置用户邮箱及用户名，使用以下命令：
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
#调出Git的帮助文档
git --help
#查看某个具体命令的帮助文档
git +命令 --help
#查看git的版本
git --version
```


2.4 本地仓库

2.4.1 创建本地仓库

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/桌面
$ cd D:

86186@LAPTOP-HT9G16B0 MINGW64 /d
$ git init git_local_test
Initialized empty Git repository in D:/git_local_test/.git/
```

此电脑 > Data (D:)

| 名称 | 修改日期 | 类型 |
|----------------|------------------|-----|
| | 2021/12/25 22:58 | 文件夹 |
| | 2021/11/23 13:40 | 文件夹 |
| | 2021/12/14 9:46 | 文件夹 |
| | 2021/9/7 16:31 | 文件夹 |
| | 2021/9/28 11:15 | 文件夹 |
| | 2021/11/10 22:59 | 文件夹 |
| | 2021/12/17 17:20 | 文件夹 |
| | 2021/10/5 23:03 | 文件夹 |
| | 2021/9/6 21:20 | 文件夹 |
| git_local_test | 2021/12/25 23:17 | 文件夹 |

此电脑 > Data (D:) > git_local_test

| 名称 | 修改日期 | 类型 |
|------|------------------|-----|
| .git | 2021/12/25 23:17 | 文件夹 |

2.4.2 基本操作

跟踪文件

将需要Git管理的文件放入刚初始化的目录下（以readme.md文件为例）， 需要先用 git add 命令告诉Git 开始对这些文件进行跟踪，然后提交。

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git add readme.md

86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git commit -m "add file description"
[master (root-commit) 3cf42cc] add file description
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 readme.md
```

添加文件

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ touch test.txt
```

此电脑 > Data (D:) > git_local_test

| 名称 | 修改日期 | 类型 | 大小 |
|-----------|-----------------|---------------|------|
| .git | 2021/12/26 0:03 | 文件夹 | |
| readme.md | 2021/12/26 0:03 | Markdown File | 1 KB |
| test.txt | 2021/12/26 0:04 | 文本文档 | 0 KB |

删除文件

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git add test.txt

86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git commit -m "first commit"
[master 256ccdc] first commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 test.txt

86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git rm test.txt
rm 'test.txt'
```

查看状态

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git status
On branch master
changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    test.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.md
```

查看修改

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git diff
diff --git a/readme.md b/readme.md
index e69de29..af27ff4 100644
--- a/readme.md
+++ b/readme.md
@@ -0,0 +1 @@
+This is a test file.
\ No newline at end of file
```

版本回退

使用git log查看历史记录。

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git log
commit 256ccdca84044b8bedeea2f13dbd30fc7b6aa759 (HEAD -> master)
Author: dfdg777 <2849805393@qq.com>
Date: Sun Dec 26 00:19:50 2021 +0800

    first commit

commit 3cf42cc7b1d1350bf7c99100a6562c2f5b9cc8b2
Author: dfdg777 <2849805393@qq.com>
Date: Sun Dec 26 00:03:58 2021 +0800

    add file description
```

使用 git reset 命令回退到上一个版本。

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git reset
Unstaged changes after reset:
M   readme.md
D   test.txt
```

再使用git reset 命令，通过指定commit id 回到未来最新版本。

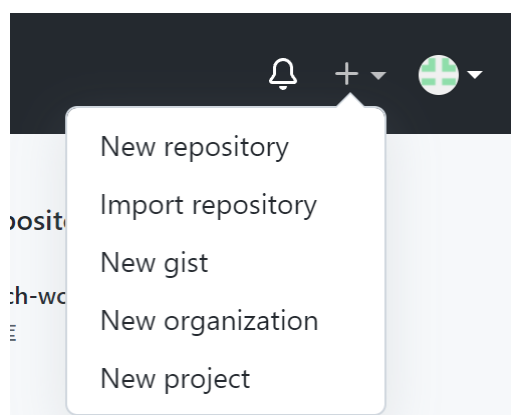
```
86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git reset --hard HEAD^
HEAD is now at 3cf42cc add file description


86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git reset --hard 256ccdca84044b8bedeea2f13dbd30fc7b6aa759 |
HEAD is now at 256ccdc first commit
```

2.5 远程仓库




2.5.1 创建远程仓库

打开GitHub账号主页，在账号首页右上角找到一个加号，点击New repository选项，填写页面相关信息即创建新的仓库。





[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)


  

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *


Repository name *




dfdg777 /

Great repository names are short and memorable. Need inspiration? How about [bookish-carnival?](#)

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)




[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)

 [dfdg777 / Yu-Thursday-hw](#) Public




2.5.2 克隆远程仓库


```
# 克隆一个项目和它的整个代码历史(版本信息)
$ git clone [url]
```

比如我们要从克隆的远程仓库托管在github上，地址为：<https://github.com/oxygenxml/com.oxygenxml.pdf2.ug>，这是一个公开的项目。



[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)

 [oxygenxml / com.oxygenxml.pdf2.ug](#) Public

[Watch](#) 26 [Fork](#) 2 [Starred](#) 8

[Code](#) [Issues](#) 1 [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#)

master 4 branches 0 tags

Go to file


Add file

Code

Clone

HTTPS SSH GitHub CLI

https://github.com/oxygenxml/com.oxygenxml



Use Git or checkout with SVN using the web URL.

Open with GitHub Desktop

Download ZIP

raducoravu

Wrap everything inside an foiblock to avoid generating fo:inline

Update to oxygenxml

.travis

Customization

fonts

gh-pages

lib

pdf2-customization-wiki-topics/imag...

.project

.travis.yml

CODE_OF_CONDUCT.md

CONTRIBUTING.md

LICENSE

Modularizing File and...

5 years ago

5 years ago

5 years ago

3 years ago

5 years ago

3 years ago

3 years ago

5 years ago

3 years ago

About

DITA-OT PDF Customization Plugin for oXygen User Manual

[dita-ot-plugin](#) [dita-ot-pdf-plugin](#)

Readme

Apache-2.0 License

Code of conduct

8 stars

26 watching

2 forks

Releases

No releases published

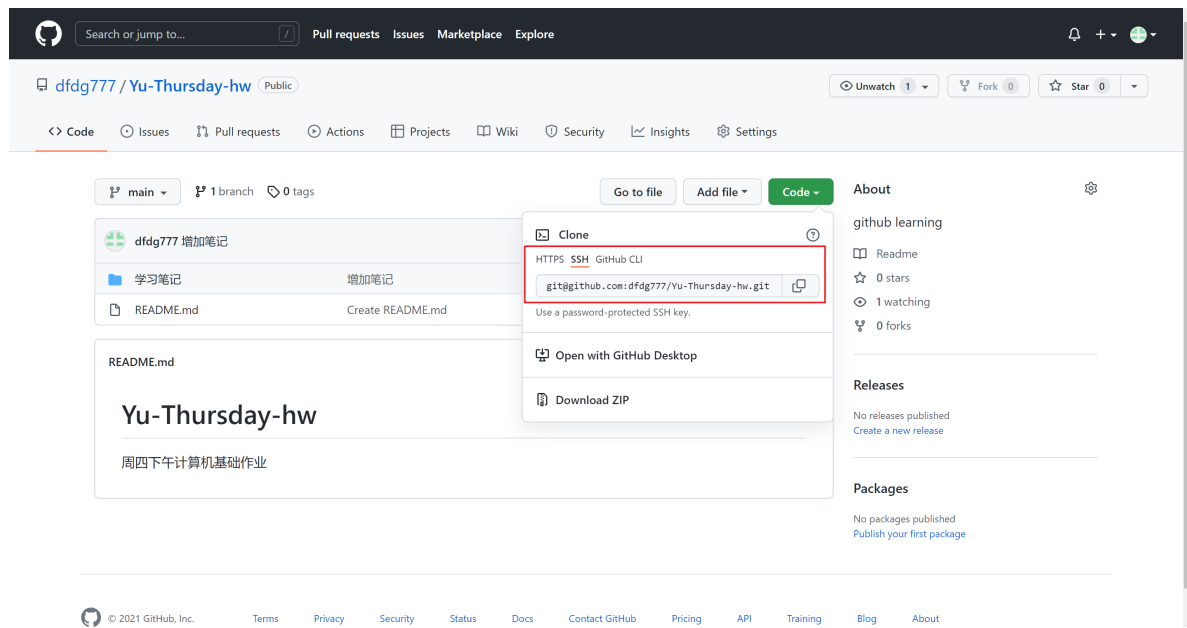
Packages

No packages published

```
86186@LAPTOP-HT9G16B0 MINGW64 /d
$ git clone https://github.com/oxygenxml/com.oxygenxml.pdf2.ug.git
Cloning into 'com.oxygenxml.pdf2.ug'...
remote: Enumerating objects: 1167, done.
remote: Total 1167 (delta 0), reused 0 (delta 0), pack-reused 1167
Receiving objects: 100% (1167/1167), 366.96 MiB | 763.00 KiB/s, done.
Resolving deltas: 100% (514/514), done.
```

2.5.3 上传本地项目

在Github上创建好Git仓库：



之后就可以和本地仓库进行关联：

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git remote add origin git@github.com:dfdg777/Yu-Thursday-hw.git
```

关联好之后我们就可以把本地库的所有内容推送到远程仓库（也就是Github）上了：

```
86186@LAPTOP-HT9G16B0 MINGW64 /d/git_local_test (master)
$ git push -u origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 417 bytes | 417.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/dfdg777/Yu-Thursday-hw/pull/new/master
remote:
To github.com:dfdg777/Yu-Thursday-hw.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

2. Git协议

Git可以使用四种主要的协议来传输资料：本地协议、HTTP协议、SSH协议以及Git协议。

2.1 本地协议

最基本的就是本地协议，其中的远程版本库就是硬盘内的另一个目录。常见于团队每一个成员都对一个共享的文件系统拥有访问权，或者比较少见的多人共用同一台电脑的情况。

如果使用共享文件系统就可以从本地版本库克隆（`clone`）、推送（`push`）以及拉取（`pull`）。

例如，克隆一个本地版本库，可以执行如下的命令：

```
$ git clone /opt/git/project.git
```

如果仅是指定路径，Git 会尝试使用硬链接（`hard link`）或直接复制所需要的文件

或可以执行这个命令：

```
$ git clone file:///opt/git/project.git
```

要增加一个本地版本库到现有的 Git 项目，可以执行如下的命令：

```
$ git remote add local_proj /opt/git/project.git
```

优点

- 简单
- 直接使用了现有的文件权限和网络访问权限，只需要像设置其他共享目录一样，把一个裸版本库的副本放到都可以访问的路径，并设置好读/写的权限即可。
- 快速从别人的工作目录中拉取更新的方法，如果和别人一起合作一个项目，他想让你从版本库中拉取更新时，运行类似 `git pull /home/john/project` 的命令比推送到服务器再取回简单多了。

缺点

- 通常共享文件系统比较难配置
- 不能保护仓库避免意外的损坏。因为每一个用户都有“远程”目录的完整 shell 权限，没有方法可以阻止他们修改或删除 Git 内部文件和损坏仓库。

2.2 HTTP协议

Git 通过 HTTP 通信有两种模式。在 Git 1.6.6 版本之前只有一个方式可用，十分简单并且通常是只读模式的。Git 1.6.6 版本引入了一种新的、更智能的协议，让 Git 可以像通过 SSH 那样智能的协商和传输数据。之后几年，这个新的 HTTP 协议因为其简单、智能变的十分流行。新版本的 HTTP 协议一般被称为“智能” HTTP 协议，旧版本的一般被称为“哑” HTTP 协议。

2.2.1 智能HTTP 协议

智能 HTTP 协议或许是最流行的使用 Git 的方式。

运行方式和 SSH 及 Git 协议类似，只是运行在标准的 HTTP/S 端口上并且可以使用各种 HTTP 验证机制，这意味着使用起来会比 SSH 协议简单得多，比如可以使用 HTTP 协议的用户名 / 密码的基础授权，免去设置 SSH 公钥。

它即支持像 `git://` 协议一样设置匿名服务，可以像 SSH 协议一样提供传输时的授权和加密，而且只用一个 URL 就可以都做到，省去了为不同的需求设置不同的 URL。

如果要推送到一个需要授权的服务器上，服务器会提示你输入用户名和密码，从服务器获取数据时也一样。

优点

不同的访问方式只需要一个 URL 以及服务器只在需要授权时提示输入授权信息，这两个简便性让终端用户使用 Git 变得非常简单。相比 SSH 协议，可以使用用户名 / 密码授权是一个很大的优势，这样用户就不必须在使用 Git 之前先在本地产生成 SSH 密钥对再把公钥上传到服务器。对非资深的使用者，或者系统上缺少 SSH 相关程序的使用者，HTTP 协议的可用性是主要的优势。与 SSH 协议类似，HTTP 协议也非常快和高效。

你也可以在 HTTPS 协议上提供只读版本库的服务，如此你在传输数据的时候就可以加密数据；或者，你甚至可以让客户端使用指定的 SSL 证书。

另一个好处是 HTTP/S 协议被广泛使用，一般的企业防火墙都会允许这些端口的数据通过。

缺点

在一些服务器上，架设 HTTP/S 协议的服务端会比 SSH 协议的棘手一些。除了这一点，用其他协议提供 Git 服务与“智能” HTTP 协议相比就几乎没有优势了。

如果你在 HTTP 上使用需授权的推送，管理凭证会比使用 SSH 密钥认证麻烦一些。然而，你可以选择使用凭证存储工具，比如 OSX 的 Keychain 或者 Windows 的凭证管理器。参考 凭证存储 如何安全地保存 HTTP 密码。

2.2.2 哑 HTTP 协议

如果服务器没有提供智能 HTTP 协议的服务，Git 客户端会尝试使用更简单的“哑” HTTP 协议。哑 HTTP 协议里 web 服务器仅把裸版本库当作普通文件来对待，提供文件服务。哑 HTTP 协议的优美之处在于设置起来简单。基本上，只需要把一个裸版本库放在 HTTP 根目录，设置一个叫做 post-update 的挂钩就可以了。此时，只要能访问 web 服务器上你的版本库，就可以克隆你的版本库。

2.3 SSH 协议

架设 Git 服务器时常用 SSH 协议作为传输协议。因为大多数环境下已经支持通过 SSH 访问——即时没有也比较很容易架设。SSH 协议也是一个验证授权的网络协议；并且，因为其普遍性，架设和使用都很容易。

通过 SSH 协议克隆版本库，可以指定一个 ssh:// 的 URL：

```
$ git clone ssh://user@server/project.git
```

或者使用一个简短的 scp 式的写法：

```
$ git clone user@server:project.git
```

也可以不指定用户，Git 会使用当前登录的用户名。

优点

用 SSH 协议的优势有很多。首先，SSH 架设相对简单——SSH 守护进程很常见，多数管理员都有使用经验，并且多数操作系统都包含了它及相关的管理工具。其次，通过 SSH 访问是安全的——所有传输数据都要经过授权和加密。最后，与 HTTP/S 协议、Git 协议及本地协议一样，SSH 协议很高效，在传输前也会尽量压缩数据。

缺点

SSH 协议的缺点在于你不能通过他实现匿名访问。即便只要读取数据，使用者也要有通过 SSH 访问你的主机的权限，这使得 SSH 协议不利于开源的项目。如果你只在公司网络使用，SSH 协议可能是你唯一要用到的协议。如果你要同时提供匿名只读访问和 SSH 协议，那么你除了为自己推送架设 SSH 服务以外，还得架设一个可以让其他人访问的服务。

2.4 Git 协议

Git 协议是包含在 Git 里的一个特殊的守护进程；它监听在一个特定的端口（9418），类似于 SSH 服务，但是访问无需任何授权。要让版本库支持 Git 协议，需要先创建一个 git-daemon-export-ok 文件——它是 Git 协议守护进程为这个版本库提供服务的必要条件——但是除此之外没有任何安全措施。要么谁都可以克隆这个版本库，要么谁也不能。这意味着，通常不能通过 Git 协议推送。由于没有授权机制，一旦你开放推送操作，意味着网络上知道这个项目 URL 的人都可以向项目推送数据。所以极少会有人这么做。

优点

目前，Git 协议是 Git 使用的网络传输协议里最快的。如果你的项目有很大的访问量，或者你的项目很庞大并且不需要为写进行用户授权，架设 Git 守护进程来提供服务是不错的选择。它使用与 SSH 相同的数据传输机制，但是省去了加密和授权的开销。

缺点

Git 协议缺点是缺乏授权机制。把 Git 协议作为访问项目版本库的唯一手段是不可取的。一般的做法里，会同时提供 SSH 或者 HTTPS 协议的访问服务，只让少数几个开发者有推送（写）权限，其他人通过 git:// 访问只有读权限。Git 协议也许也是最难架设的。它要求有自己的守护进程，这就要配置 xinetd 或者其他程序，这些工作并不简单。它还要求防火墙开放 9418 端口，但是企业防火墙一般不会开放这个非标准端口。而大型的企业防火墙通常会封锁这个端口。

3. Git使用小神器

3.1 搜索小妙招

- **通过in关键词限制搜索范围**

xxx in:name 项目名包含xxx的

xxx in:description 项目描述包含xxx的

xxx in:readme 项目的readme文件中包含xxx的

当然也可以通过xxx in:name,description来组合使用

如下，我需要搜索项目名或者描述中包含「Hello」的项目，

通过Hello in:name,description 完成

- **通过 Star 或者Fork数 去查找项目**

通过通配符 > < = 即可，区间范围内可通过 num1..num2

如，要查找stars数不小于666的springboot项目

springboot stars:>=666

forks 大于等于500

springboot forks:>500

查找fork在100到200之间 且stars数在80到100之间的springboot项目

springboot forks:100..200 stars:80..100

- **awesome + 关键字**

搜索和关键字匹配的优秀项目

awesome springboot 搜索优秀的springboot相关的项目，包括框架、教程等

- **分享项目中某一行的代码**

只需要在具体的网址后面拼接#Lxx(xx为行数)

如，我需要分享这个类中的@SprintBootApplication注解，值需要在后面拼接上#L6 即可

<https://github.com/lxy-go/SpringBoot/.../JpaApplication.java#L6>

3.2 GitHub插件

- **Octotree**

平时浏览github代码时，每个文件都需要点击才能查看，用了这个插件可以将项目的目录结构以树形结构显示，很好的解决项目的层级多不方便预览的问题，点击之后会自动跳转到相应的目录。

- **Enhanced Github**

Github 增强插件，开启后可以显示 Github 当前仓库的整体大小，以及每个单个文件的文件大小。最重要的是加入了单个文件下载支持，避免了为了某一个文件而需要下载整个仓库速度缓慢的尴尬情况。

- **OctoLinker**

帮助我们浏览代码像ide那样可以链接跳转，只需要ctrl+点击变量名即可。

- **Sourcegraph**

一个可以在github上浏览和搜索代码的工具，安装好插件之后会出现一个view Repository的按钮。

点击进去之后，可以随意在项目中搜索，可以查看变量和方法，以及进行跳转等等。