

I 版本控制的起源: diff 与 patch

2.1 diff 简介 : diff 用来比较两个文件或者目录之间的差异。

file: left.c

```
#include <stdio.h>

main( )
{
    printf("hello, world\n");
}
```

原始文件

file: right.c

```
#include <stdio.h>

int main( )
{
    printf("hello, world\n");
    return 0;
}
```

目标文件

\$ diff -u left.c right.c

```
1 --- left.c      2019-06-10 20:47:15.234033300 +0800
2 +++ right.c     2019-06-10 20:47:31.685001600 +0800
3 @@ -1,6 +1,8 @@
4 #include <stdio.h>
5
6 -main( )
7 +int main( )
8 {
9     printf("hello, world\n");
10 +
11 +    return 0;
12 }
```

差异结果的头部 :

分别记录比较的原始文件与目标文件的文件名及其最后一次变化时间。

定位符号 :

-1,6 含义 : 是如下内容包含原始文件第 1 行开始的 6 行 ;
+1,8 含义 : 如下内容包含目标文件的第 1 行开始的 8 行。

出现在原始文件中的行 , 而在目标文件中不存在

出现在目标文件中的行 , 而在原始文件中不存在

其它空格开始的行 , 则是在原始文件和目标文件中都出现的行 ,
用于对差异内容的上下文参考 , 例如第 4、5、8、9、12 行。

I CVS & SVN : 集中式版本控制工具

CVS简介

CVS (Concurrent Versions System) 诞生于1985年, 有史以来第一个被大规模使用的版本控制工具。 CVS 的出现让工程师可以协同工作。

CVS存在的问题

不支持原子化提交，会导致客户端向服务器端提交了不完整的数据，还有网络传输效率低等。

SVN的诞生

SVN (Subversion) 目的是创建一个更好用的版本控制系统以取代CVS。 优化了服务器上内容的存储，实现了原子提交等。

SVN存在的问题

在局域网之外使用SVN，单是查看日志、提交数据等操作的延迟，就足以让基于广域网协同工作的团队抓狂了。

集中式版本控制存在的问题：

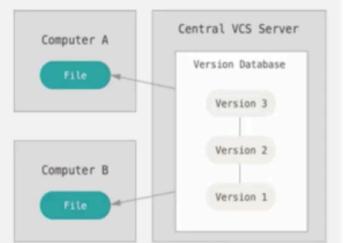
狭窄的提交通道

- 提交排队，不能同时修改。提交缺乏质量控制
- 缺乏代码门禁，在本地代码提交到服务器之间缺少检查防护

一种解决方案：Rietveld 提供旁路检查

数据安全性差

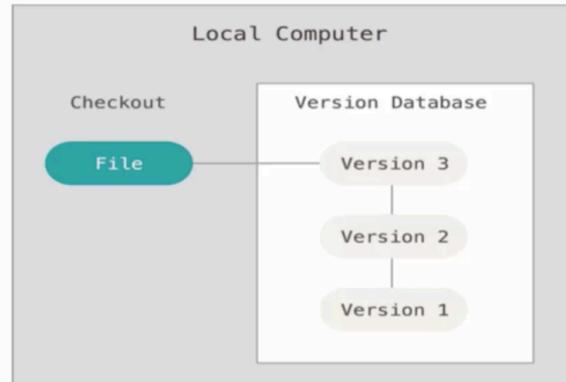
- 单点故障
- 黑客攻击



I RCS : 最早期的本地版本控制工具

RCS (Revision Control System)

- RCS作为非常古老的版本工具，远远在SVN和已经退役的CVS之前，它的古老程度应该比Web开发的ASP前代的CGI还要久远。
- 如果想对版本管理实现方式进行深入研究的话，研究RCS是一种最为简单的入手方式。
- RCS采用把diff的集合，采用RCS自己的格式保存到磁盘中（可以通过`diff -n left.c right.c`产生RCS格式的diff内容），能通过这些diff集合，重新回到文件修改的任何历史中的点。



I CVS & SVN : 集中式版本控制工具

CVS简介

CVS (Concurrent Versions System) 诞生于1985年, 有史以来第一个被大规模使用的版本控制工具。 CVS 的出现让工程师可以协同工作。

CVS存在的问题

不支持原子化提交，会导致客户端向服务器端提交了不完整的数据，还有网络传输效率低等。

SVN的诞生

SVN (Subversion) 目的是创建一个更好用的版本控制系统以取代CVS。 优化了服务器上内容的存储，实现了原子提交等。

SVN存在的问题

在局域网之外使用SVN，单是查看日志、提交数据等操作的延迟，就足以让基于广域网协同工作的团队抓狂了。

集中式版本控制存在的问题：

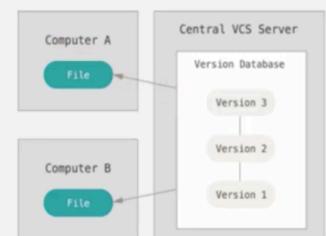
狭窄的提交通道

- 提交排队，不能同时修改。提交缺乏质量控制
- 缺乏代码门禁，在本地代码提交到服务器之间缺少检查防护

一种解决方案：Rietveld 提供旁路检查

数据安全性差

- 单点故障
- 黑客攻击



I Git: Linus的第二个伟大作品

3.1 Git起源

Linux之父Linus是坚定的CVS反对者，他也同样地反对SVN。2002年Linus顶着开源社区精英们的口诛笔伐，选择了一个商业版本控制系统BitKeeper作为Linux内核的代码管理工具。和CVS/SVN不同，BitKeeper是属于分布式版本控制系统。

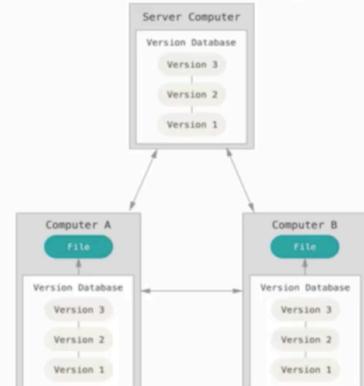


Linus Torvalds



Git诞生大事记：

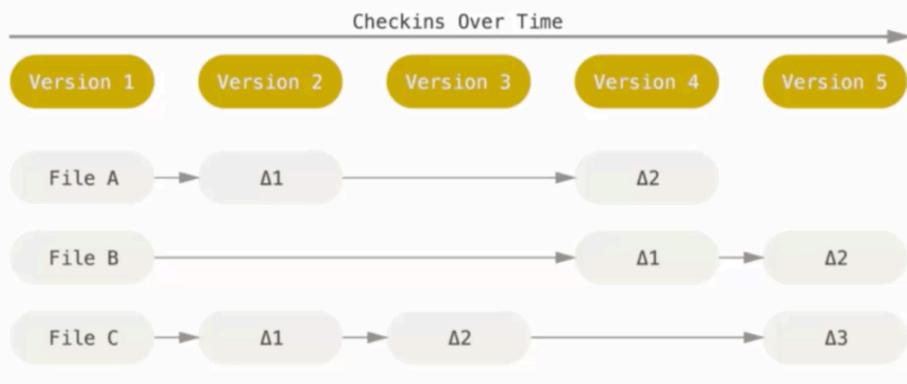
- 2005年4月3日，开始开发Git。
- 2005年4月6日，项目发布。
- 2005年4月7日，Git就可以作为自身的版本控制工具了。
- 2005年4月18日，发生第一个多分支合并。
- 2005年4月29日，Git的性能就已经达到了Linus的预期。
- 2005年6月16日，Linux核心2.6.12发布，那时Git已经在维护Linux核心的源代码。



I Git: Linus的第二个伟大作品

3.2 集中式 VS 分布式（1）：记录差异还是记录快照

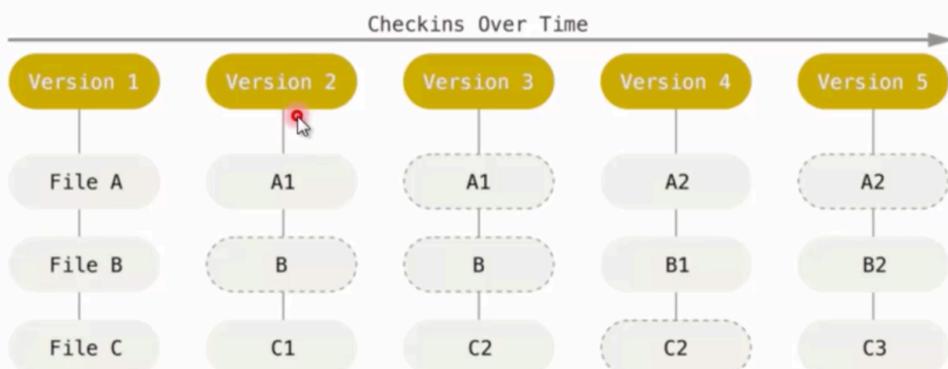
- Git 和其它版本控制系统（包括 Subversion 和近似工具）的主要差别在于 **Git 对待数据的方法**。
- 概念上来区分，其它大部分系统以文件变更列表的方式存储信息。
- 这类系统（CVS、Subversion、Perforce、Bazaar 等等）将保存的信息看作是一组基本文件和每个文件随时间逐步累积的差异。



I Git: Linus的第二个伟大作品

3.2 集中式 VS 分布式（1）：记录差异还是记录快照

- Git 不按照以上方式对待或保存数据。 反之，Git 更像是把数据看作是对小型文件系统的一组快照。
- 每次提交更新，或在 Git 中保存项目状态时，它主要对当时的全部文件制作一个快照并保存这个快照的索引。
- 为了高效，如果文件没有修改，Git 不再重新存储该文件，而是只保留一个链接指向之前存储的文件。 Git 对待数据更像是一个 **快照流**。



I Git: Linus的第二个伟大作品

3.2 集中式 VS 分布式 (2) 脆弱的中央库 VS 强壮的分布库

脆弱的中央库

- 备份的重要性

集中式CVS存在单点故障，备份极其重要。

- 服务器压力

基本上所有的操作需要与服务器交互，

操作受限于带宽，不能移动办公。

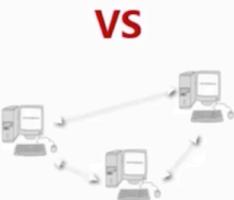
- 安全性

集中式CVS假定服务器是安全的。假定成立么？

单点故障、黑客攻击等。

- 不适合开源项目

强调集中管理，适合人数不多的项目。



VS

强壮的分布库

- 全是服务器

数据最安全；无带宽和性能瓶颈。

- 提交为本地操作

快；全离线操作；编码不会被冲突打断；

能够移动办公。 ↳

- 数据的完整性

Git 数据、提交全部使用SHA1哈希，以保证完整性，甚至提交可以使用PGP签名。

- 工作模型

适合分布式开发，强调个体。

Git 容灾示例

[kernel.org 2011 attack \(2011.8-2011.11\)](#)

[宇宙射线反转磁盘一个比特的数据修复](#)

I Git: Linus的第二个伟大作品

3.3 如何选择适合的版本控制工具

彼之蜜糖，我之毒药，哪个版本控制工具更适合？

SVN不适合的领域：

跨地域的协同开发

对代码的高质量追求和代码门禁

Git 不适合的领域：

不适合Word等二进制文档的版本控制

因为：Git 无锁定/解锁模式，故不能排他式修改，
整体的读授权，不能将读授权精细到目录级别

解决方案：版本库按照目录拆分

| 结语: Git是什么

Git 是一个版本控制工具, 而且是一个开源的分布式版本控制工具。

按照 Linus 本人的描述 , Git 的很多命令设计是来源于 BitKeeper, 但是 Git 有更多属性:

- 极快的速度
- 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)

| 本章小结

- 版本控制工具的发展历史经过：原始人工维护状态，本地RCS，集中式如CVS、SVN和分布式如Git。
- 版本控制工具提供了协作开发的能力，借助它们我们可以回到任何时间的代码状态。
- 集中式版本控制工具，几乎所有的动作都需要服务器参与，并且数据安全性与服务器关系很大。
- Git 是分布式版本控制工具，除了与服务器之前进行按需同步之外，所有的提交操作都不需要服务器。

|关于本章

章节背景

在学习一个工具之前，需要先学会如何下载和安装，Git 同样如此，在不同的环境下，要知道如何正确安装。我们本节主要围绕 Linux 与 Windows 进行 Git 安装的说明。并在安装结束后，对常见的配置进行说明。

章节内容

- Linux 下安装Git (包管理器安装、源码安装、命令补齐)
- Windows 下安装 (安装Git & TortoiseGit)
- Git的基本配置

| Linux 下安装Git

1.1 包管理器方式安装

Linux 系统: Ubuntu 10.10(maverick)或更新版本 , Debian(squeeze)或更新版本

```
$ sudo aptitude install git  
$ sudo aptitude install git-doc git-svn git-email gitk
```

其中git软件包包含了大部分Git命令，是必装的软件包。

软件包git-svn、git-email、gitk本来也是Git软件包的一部分，但是因为有着不一样的软件包依赖（如更多perl模组，tk等），所以单独作为软件包发布。

Linux 系统: RHEL、Fedora、CentOS 等版本:

```
$ yum install git  
$ yum install git-svn git-email gitk
```

| Linux 下安装Git

1.2 从源代码开始安装

访问Git的官方网站：<http://git-scm.com/>。下载Git源码包，例如：git-2.19.0.tar.gz。

展开源码包，并进入到相应的目录中。

```
$ tar -jxvf git-2.19.0.tar.bz2  
$ cd git-2.19.0  
  →
```

安装方法写在INSTALL文件当中，参照其中的指示完成安装。下面的命令将Git安装在 `/usr/local/bin` 中。

```
$ make prefix=/usr/local all  
$ sudo make prefix=/usr/local install
```

安装Git文档（可选）。

```
$ make prefix=/usr/local doc info  
$ sudo make prefix=/usr/local install-doc install-html install-info
```

I Linux 下安装Git

1.3 命令补齐

Linux的shell环境（bash）通过bash-completion软件包提供命令补齐功能，能够实现在录入命令参数时按一下或两下TAB键，实现参数的自动补齐或提示。例如输入 `git com` 后按下TAB键，会自动补齐为 `git commit`。

将Git源码包中的命令补齐脚本复制到bash-completion对应的目录中：

```
$ cp contrib/completion/git-completion.bash /etc/bash_completion.d/
```

重新加载自动补齐脚本，使之在当前shell中生效：

```
$ . /etc/bash_completion
```

为了能够在终端开启时自动加载bash_completion脚本，需要在本地配置文件 `~/.bash_profile` 或全局文件 `/etc/bashrc` 文件中添加下面的内容：

```
if [ -f /etc/bash_completion ]; then
. /etc/bash_completion
fi
```

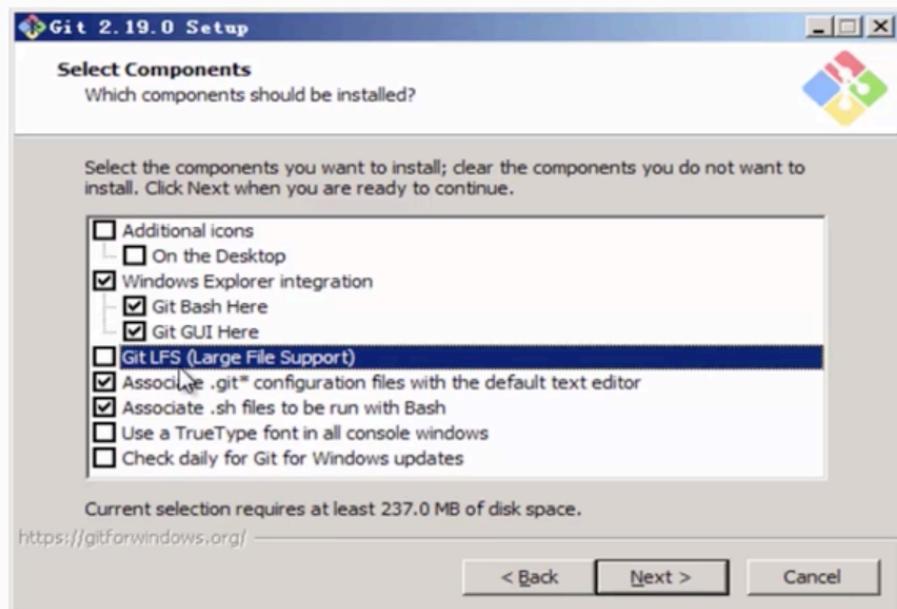
| Windows 下安装 Git

目前 Git 提供的 Windows 安装包自带 MinGW (Minimalist GNU for Windows , 最简GNU工具集) , 在安装后MinGW 提供了一个bash提供的shell环境 (Git Bash) 以及其他相关工具软件 , 组成了一个最简系统 (Minimal SYStem) , 这样在 Git Bash 中 , Git的使用和在Linux下使用完全一致。

I Windows 下安装 Git

2.1 安装 Git

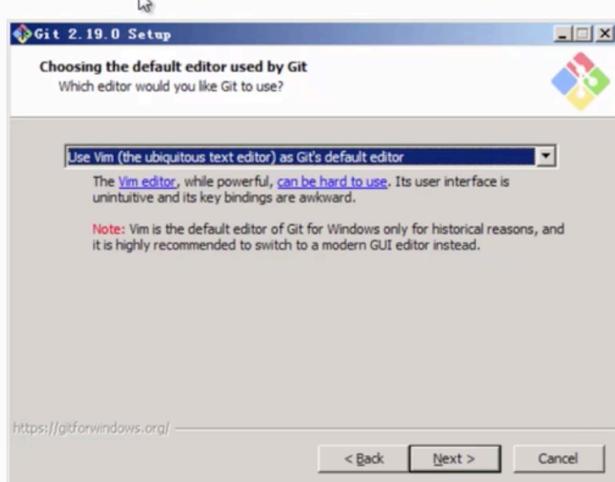
Step2：这里选择一些必要的组件，开源的 git-lfs 存在一些问题，建议把勾选去掉（huawei 提供了自己的修复版本，后续相关优化会推送到社区）。



I Windows 下安装 Git

2.1 安装 Git

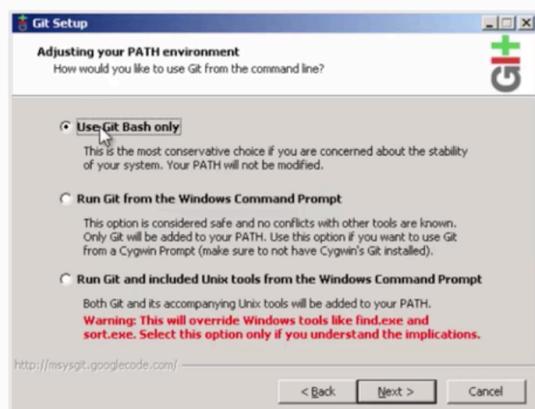
Step3 : Git 默认的编辑器，建议保持默认，当然你也可以选择其它的，例如 Notepad++。



I Windows 下安装 Git

2.1 安装 Git

Step4：在安装过程中会询问是否修改环境变量。建议选择“Use Git Bash Only”，即只在 MinGW 提供的 shell 环境中使用 Git，不修改 PATH 环境变量，避免 Git 自带的工具与 Windows 下已有的产生冲突。



如果不清楚 PATH，可以参考 https://en.wikipedia.org/wiki/PATH_%28variable%29，简单来讲，就是你输出一条命令的时候，系统会从 PATH 这个配置中寻找实现这条命令的程序在哪里，找到后就启动程序。

I Windows 下安装 Git

2.1 安装 Git

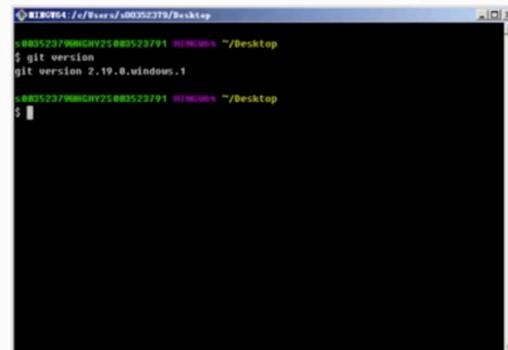
Step5：其它后续提示可以都采用缺省配置，进行安装过程。

安装完成后，我们可以在 Windows 任意目录下，

右键单击选中“Git Bash”启动 Git Bash：



可以执行 `git version` 查看安装的 git 版本信息：



```
MINGW64 /c/Users/s0052379/Desktop
$ git version
git version 2.19.0.windows.1
$
```

| Windows 下安装 Git

2.2 Windows 下安装 TortoiseGit



在Windows下安装和使用Git有两个不同的方案，除了刚刚的 Git 安装包，再有一个就是基于msysGit 的图形界面工具——TortoiseGit。 

TortoiseGit简介

TortoiseGit提供了Git和Windows资源管理器的整合，提供了Git的图形化操作界面。像其他Tortoise系列产品（TortoiseCVS、TortoiseSVN）一样，Git工作区的目录和文件的图标附加了标识版本控制状态的图像，可以非常直观的看到哪些文件被更改了需要提交。通过对右键菜单的扩展，可以非常方便的在资源管理器中操作Git版本库。 ·

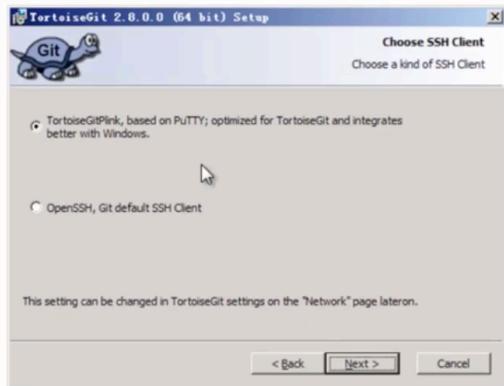
TortoiseGit安装

安装TortoiseGit非常简单，访问网站 <http://code.google.com/p/tortoisegit/>，下载安装包，然后根据提示完成安装。

I Windows 下安装 Git

2.2 Windows 下安装 TortoiseGit

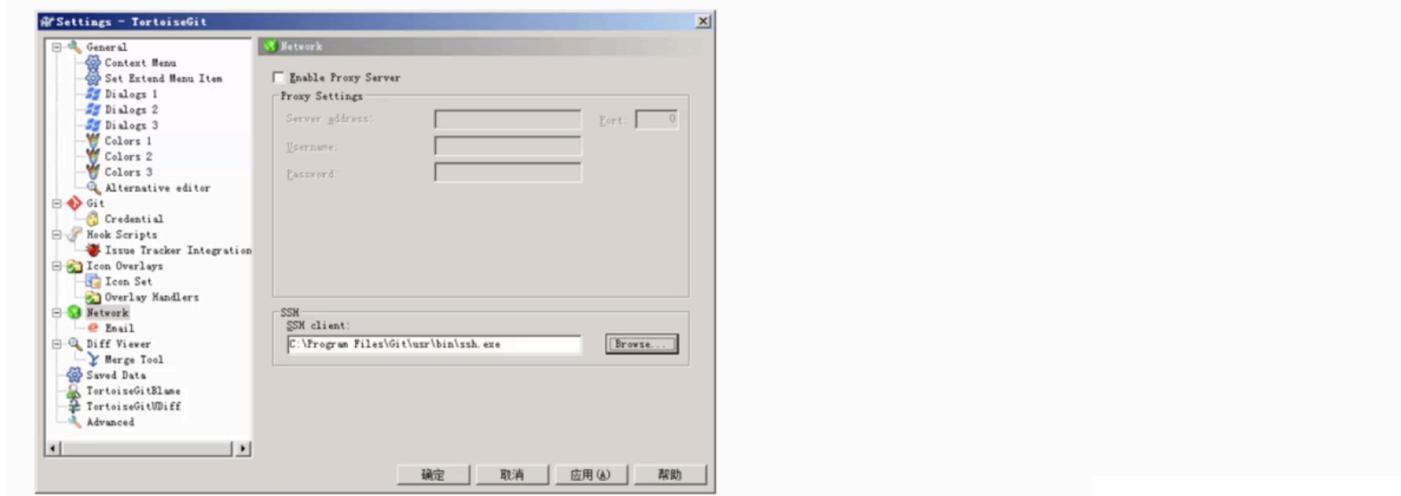
安装过程中会询问要使用的SSH客户端，缺省使用内置的TortoisePLink（来自PuTTY项目）做为SSH客户端。TortoisePLink和TortoiseGit的整合性更好，可以直接通过对话框设置SSH私钥（PuTTY格式），而无需再到字符界面去配置SSH私钥和其他配置文件。



I Windows 下安装 Git

2.2 Windows 下安装 TortoiseGit

如果你的本地同时安装了命令行的 Git 版本，可以通过TortoiseGit的设置对话框选中 Git 提供的 ssh 客户端，这样在下载 ssh 协议的代码仓库的时候，通过命令行与 TortoiseGit 图形界面都可以使用同一套公钥和密钥。



I 3 Git 基本配置

Git有三种配置，分别以文件的形式存放在三个不同的地方。可以在命令行中使用git config工具查看这些变量。

系统配置（对所有用户都适用）

存放在git的安装目录下：`%Git%/etc/gitconfig`；若使用`git config`时用`--system`选项，读写的就是这个文件：

```
git config --system core.autocrlf
```

用户配置（只适用于该用户）

存放在用户目录下。例如linux存放在：`~/.gitconfig`；若使用`git config`时用`--global`选项，读写的就是这个文件：

```
git config --global user.name
```

仓库配置（只对当前项目有效）

当前仓库的配置文件（也就是工作目录中的`.git/config`文件）；若使用`git config`时用`--local`选项，读写的就是这个文件：

```
git config --local remote.origin.url
```

注：每一个级别的配置都会覆盖上层的相同配置，例如`.git/config`里的配置会覆盖`%Git%/etc/gitconfig`中的同名变量。

I 3 Git 基本配置

3.1 配置个人身份

首次的 Git 设定（设定身份，自己做主）

```
git config --global user.name "Zhang San"  
git config --global user.email zhangsan123@huawei.com
```

这个配置信息会在 Git 仓库中提交的修改信息中体现，但和Git服务器认证使用的密码或者公钥密码无关。

→

注意事项：

行不更名，坐不改姓：责任追踪/应用之间的用户关联/贡献度统计

I 3 Git 基本配置

3.2 文本换行符配置

假如你正在Windows上写程序，又或者你正在和其他人合作，他们在Windows上编程，而你却在其他系统上，在这些情况下，你可能会遇到行尾结束符问题。这是因为Windows使用回车和换行两个字符来结束一行，而Mac和Linux只使用换行一个字符。虽然这是小问题，但它会极大地扰乱跨平台协作。

Git可以在你提交时自动地把行结束符CRLF转换成LF，而在签出代码时把LF转换成CRLF。用core.autocrlf来打开此项功能，如果是在Windows系统上，把它设置成true，这样当签出代码时，LF会被转换成CRLF：

```
$ git config --global core.autocrlf true
```

Linux或Mac系统使用LF作为行结束符，因此你不想Git在签出文件时进行自动的转换；当一个以CRLF为行结束符的文件不小心被引入时你肯定想进行修正，把core.autocrlf设置成input来告诉Git在提交时把CRLF转换成LF，签出时不转换：

```
$ git config --global core.autocrlf input
```

这样会在Windows系统上的签出文件中保留CRLF，会在Mac和Linux系统上，包括仓库中保留LF。

如果你是Windows程序员，且正在开发仅运行在Windows上的项目，可以设置false取消此功能，把回车符记录在库中：

```
$ git config --global core.autocrlf false
```

| Git 基本配置

3.3 文本编码配置

- **i18n.commitEncoding 选项**：用来让git commit log存储时，采用的编码，默认UTF-8.
- **i18n.logOutputEncoding 选项**：查看git log时，显示采用的编码，建议设置为UTF-8.

```
# 中文编码支持  
git config --global gui.encoding utf-8  
git config --global i18n.commitencoding utf-8  
git config --global i18n.logoutputencoding utf-8  
  
# 显示路径中的中文：  
git config --global core.quotepath false
```

```
git log -1 d3c7a14c2a0669942be867596cd15a2b1299c569  
commit d3c7a14c2a0669942be867596cd15a2b1299c569  
Merge: 2F2786100 f6698a1e0  
Author: hMX568508 <huanghao38@huawei.com>  
Date: Thu May 30 19:55:33 2019 +0800  
  
Merge MR-6695 from branch 'hMX568508/labhub.git::search' into 'stable'  
  
部分用户issue搜索功能失效  
  
####【实现功能或解决问题描述】  
1. 当请求搜索服务或是处理返回数据出现错误时显示全部数据的错误  
2. 当搜索服务返回空数据时（请求出现异常或是数据未同步），在数据库中搜索结果  
  
####【对应的issue单号】  
http://rnd-isourceb.huawei.com/rnd-isource/inner\_test/issues/2390
```

I Git 基本配置

3.4 与服务器的认证配置

3.4.1 常见的两种协议认证方式

http / https 协议认证

设置口令缓存：

```
git config --global credential.helper store
```

添加 HTTPS 证书信任：

```
git config http.sslverify false
```

ssh 协议认证

SSH协议是一种非常常用的Git仓库访问协议，使用公钥认证、无需输入密码，加密传输，操作便利又保证安全性

| Git 基本配置

3.4 与服务器的认证配置

3.4.2 ssh认证的配置过程

生成公钥 :

Git工具安装成功后运行 Git Bash , 在弹出的客户端命令行界面中输入下面提示的命令。
(比如你的邮箱是 zhangsan1123@Huawei.com)

```
$ ssh-keygen -t rsa -C zhangsan1123@huawei.com
```

生成公钥举例 :

```
zhangsan@HGHY2ZX3523791 MINGW64 ~
$ ssh-keygen -t rsa -C "zhangsan1123@huawei.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/zwx352379/.ssh/id_rsa): Enter回车
Enter passphrase (empty for no passphrase): Enter回车
Enter same passphrase again: Enter回车
Your identification has been saved in /c/Users/zwx352379/.ssh/id_rsa.
Your public key has been saved in /c/Users/zwx352379/.ssh/id_rsa.pub.
```

添加公钥到代码平台 :

- 1.登录[代码平台](#)
- 2.进入 “Profile Settings”
- 3.点击左侧栏的 “SSH Keys”
- 4.点击 “Add SSH Key” ,将刚生成的公钥文件的内容 , 复制到 “Public Key” 栏 , 保存即可。