

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：

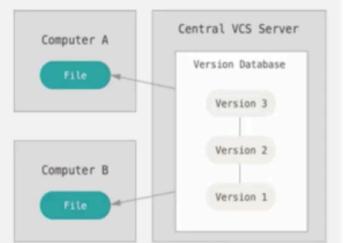
狭窄的提交通道

- 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

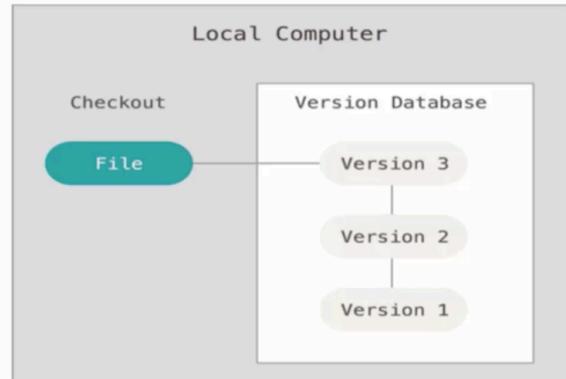
- 单点故障
- 黑客攻击



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：

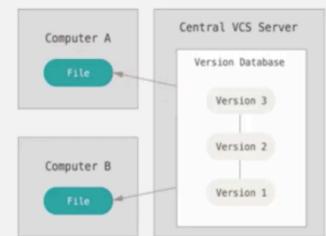
狭窄的提交通道

- 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

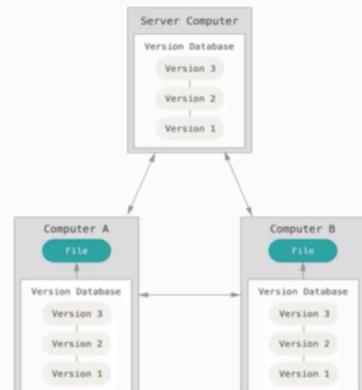
Linux之父Linus是坚定的CVS反对者，他也同样地反对SVN。2002年Linus顶着开源社区精英们的口诛笔伐，选择了一个商业版本控制系统BitKeeper作为Linux内核的代码管理工具。和CVS/SVN不同，BitKeeper是属于分布式版本控制系统。



Linus Torvalds

Git诞生大事记：

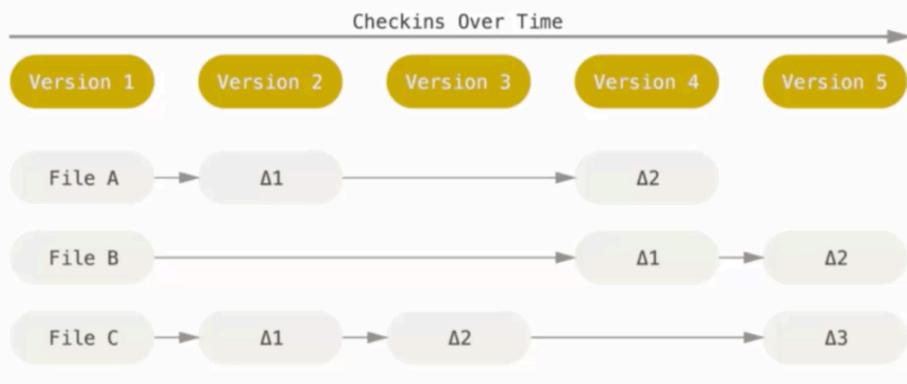
- 2005年4月3日，开始开发Git。
- 2005年4月6日，项目发布。
- 2005年4月7日，Git就可以作为自身的版本控制工具了。
- 2005年4月18日，发生第一个多分支合并。
- 2005年4月29日，Git的性能就已经达到了Linus的预期。
- 2005年6月16日，Linux核心2.6.12发布，那时Git已经在维护Linux核心的源代码。



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

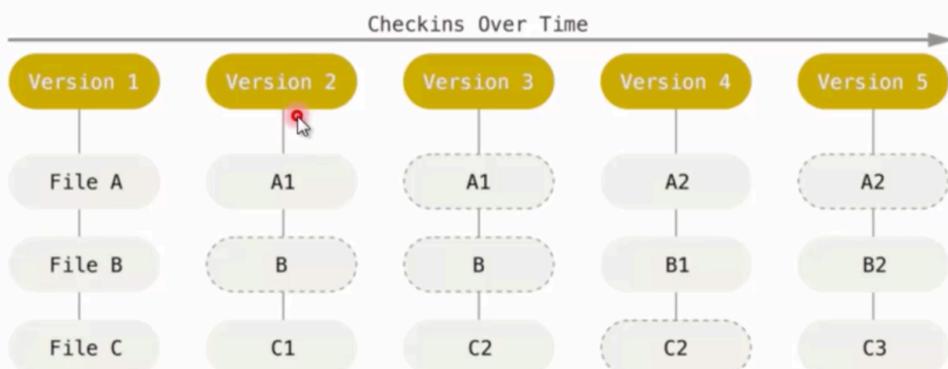
- Git 和其它版本控制系统（包括 Subversion 和近似工具）的主要差别在于 **Git 对待数据的方法**。
- 概念上来区分，其它大部分系统以文件变更列表的方式存储信息。
- 这类系统（CVS、Subversion、Perforce、Bazaar 等等）将保存的信息看作是一组基本文件和每个文件随时间逐步累积的差异。



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

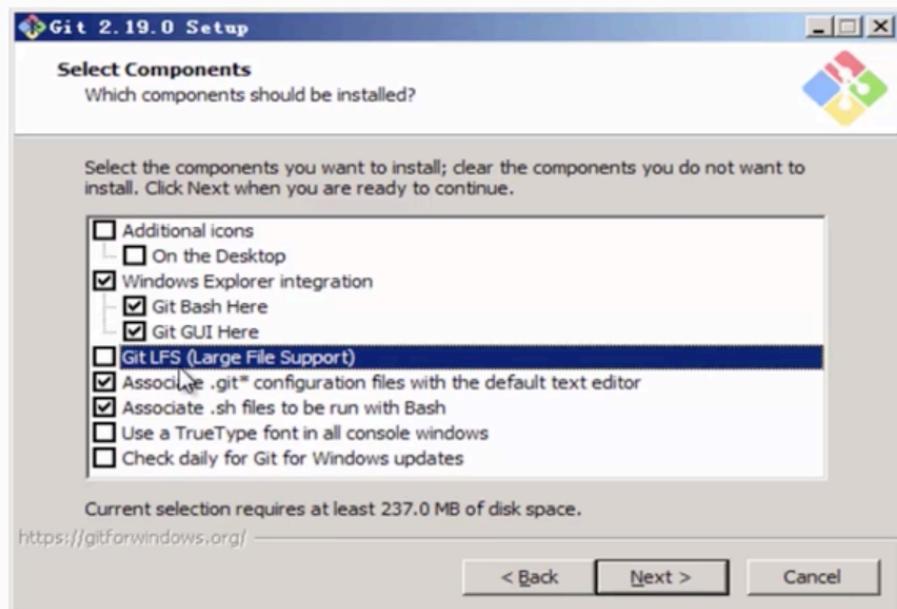
目前 Git 提供的 Windows 安装包自带 MinGW (Minimalist GNU for Windows , 最简GNU工具集) , 在安装后MinGW 提供了一个bash提供的shell环境 (Git Bash) 以及其他相关工具软件 , 组成了一个最简系统 (Minimal SYStem) , 这样在 Git Bash 中 , Git的使用和在Linux下使用完全一致。



I Windows 下安装 Git

2.1 安装 Git

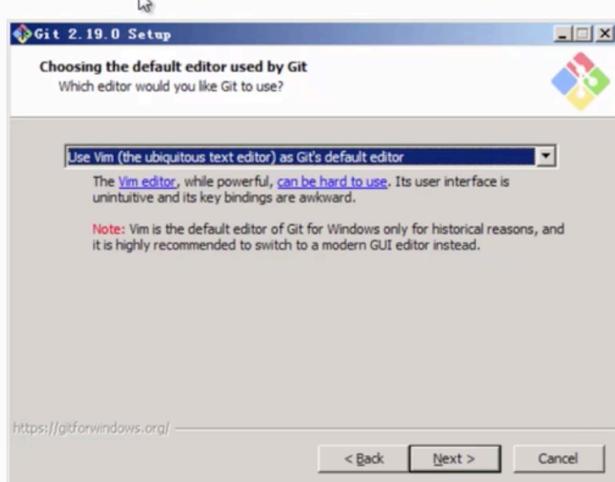
Step2：这里选择一些必要的组件，开源的 git-lfs 存在一些问题，建议把勾选去掉（huawei 提供了自己的修复版本，后续相关优化会推送到社区）。



I Windows 下安装 Git

2.1 安装 Git

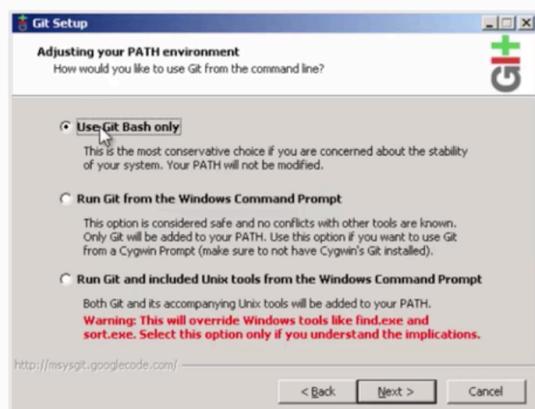
Step3 : Git 默认的编辑器，建议保持默认，当然你也可以选择其它的，例如 Notepad++。



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

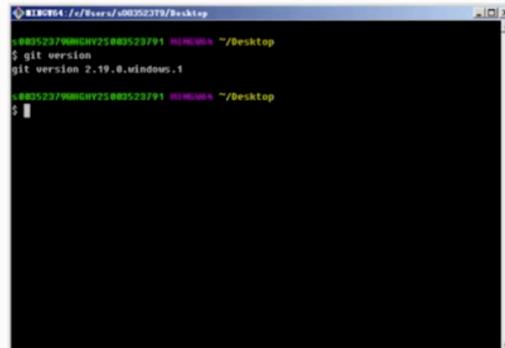
Step5：其它后续提示可以都采用缺省配置，进行安装过程。

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。 ·

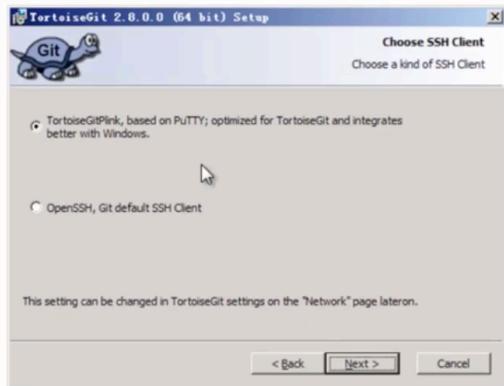
TortoiseGit安装

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

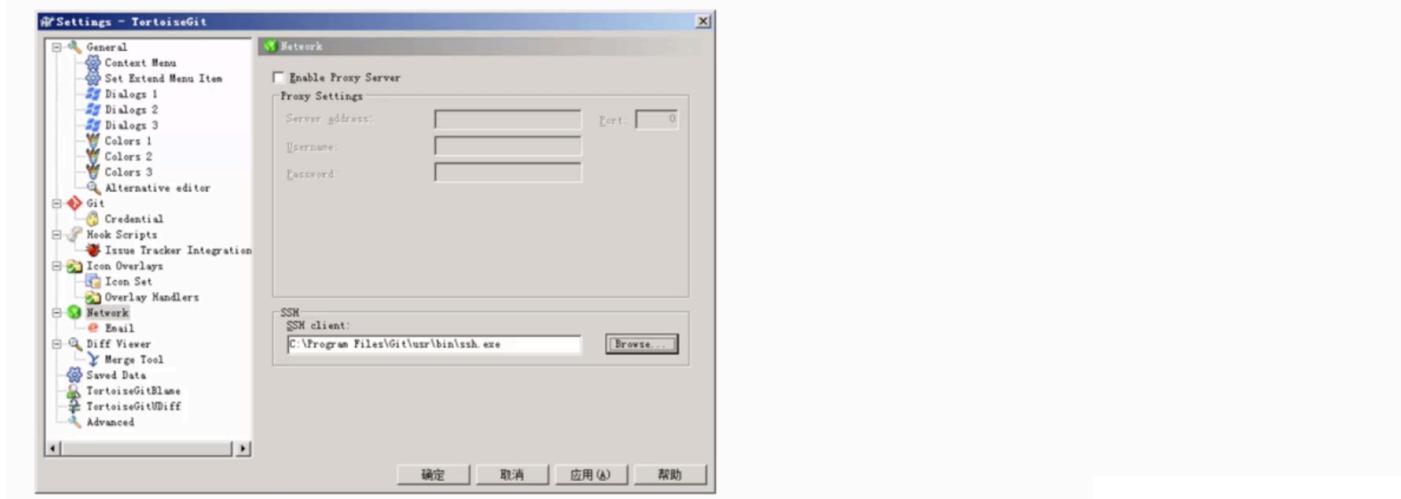
安装过程中会询问要使用的SSH客户端，缺省使用内置的TortoisePLink（来自PuTTY项目）做为SSH客户端。TortoisePLink和TortoiseGit的整合性更好，可以直接通过对话框设置SSH私钥（PuTTY格式），而无需再到字符界面去配置SSH私钥和其他配置文件。



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。

|培训目标

本章预期掌握知识点

- 了解Git常用的基本命令
- 理解Git命令行交互形式
- 理解本地仓与远程数据库间的操作关联

目录

1 Git版本控制下的三种工程区域 & 文件状态

2 Git常用命令

3 Git常用命令实操

 3.1 工程准备 `git init/git clone`

 3.2 新增/删除/移动文件到暂存区 `git add/ git rm/ git mv`

 3.3 查看工作区 `git diff/ git status`

 3.4 提交更改的文件 `git commit`

 3.5 查看日志 `git log`

 3.6 推送远端仓库 `git push`

 3.7 分支管理 `git branch/git checkout /git branch -d/git pull`

 3.8 分支合并 `git merge/git rebase`

 3.9 强制回退到历史节点 `git reset/git checkout`

1 Git版本控制下的三种工程区域 & 文件状态

在学习git常用命令前，必须再次强化下面几个概念。

Git版本控制下的工程区域只有三种：

1. 版本库 (Repository)

在工作区中有一个隐藏目录.git，这个文件夹就是Git的版本库，里面存放了Git用来管理该工程的所有版本数据，也可以叫本地仓库。

2. 工作区 (Working Directory)

日常工作的代码文件或者文档所在的文件夹。

3. 暂存区 (stage)

一般存放在工程根目录 .git/index文件中，所以我们也可以把暂存区叫作索引 (index) 。

Git版本控制下的文件状态只有三种：

1. 已提交 (committed)

该文件已经被安全地保存在本地数据库中了；

2. 已修改 (modified)

修改了某个文件，但还没有提交保存；

3. 已暂存 (staged)

把已修改的文件放在下次提交时要保存的清单中。

2 Git常用命令

工程准备

工程克隆——`git clone`

查看工作区

查看工作区的修改内容——`git diff`

查看工作区文件状态——`git status`

文件修改后提交推送

新增/删除/移动文件到暂存区——`git add/ git rm/ git mv`

提交更改的文件——`git commit`

推送远端仓库——`git push`

查看日志

查看当前分支上的提交日志——`git log`

分支管理

列出本地分支——`git branch`

新建分支——`git branch / git checkout -b`

删除分支——`git branch -d`

切换分支——`git checkout`

更新分支——`git pull`

合并分支——`git merge`

撤销操作

强制回退到历史节点——`git reset`

回退本地所有修改而未提交的——`git checkout`

分支合并

合并目标分支内容到当前分支——`git merge/git rebase`

3 Git常用命令

命令拓展: <https://git-scm.com/docs/git-init>

3.1 工程准备

git init用于在本地目录下新建git项目仓库。

执行**git init**后，当前目录下自动生成一个名为.git的目录，这代表当前项目所在目录已纳入Git管理。.git目录下存放着本项目的Git版本库，在此强烈不建议初学者改动.git目录下的文件内容。

下图可知，Git仓库下的.git目录默认是不可见的，有一定程度上是出于防止用户误操作考虑。

```
100295060@HGHY4L002950601 MINGW64 /e/my_work
$ git init firstproject
Initialized empty Git repository in E:/my_work/firstproject/.git/
100295060@HGHY4L002950601 MINGW64 /e/my_work
$ ls firstproject/
100295060@HGHY4L002950601 MINGW64 /e/my_work
$ ls firstproject/ -al
total 4
drwxr-xr-x 1 100295060 1049089 0 6月 21 20:42 .
drwxr-xr-x 1 100295060 1049089 0 6月 21 20:42 ../
drwxr-xr-x 1 100295060 1049089 0 6月 21 20:42 .git/
```

3 Git常用命令

命令拓展: <https://git-scm.com/docs/git-clone>

3.1 工程准备

git clone用于克隆远端工程到本地磁盘。

如果想从远端服务器获取某个工程，那么：

- 1.确定自己Git账号拥有访问、下载该工程的权限
- 2.获取该工程的Git仓库URL
- 3.本地命令行执行 `git clone [URL]`或 `git lfs clone [URL]`

```
100295060@HGHY4L002950601 MINGW64 /e/my_work
$ git clone git@code-sh.rnd.huawei.com:100295060/CodeHunter.git
Cloning into 'CodeHunter'...
remote: Enumerating objects: 6634, done.
remote: Counting objects: 100% (6634/6634), done.
remote: Compressing objects: 100% (5104/5104), done.
remote: Total 6634 (delta 1366), reused 6613 (delta 1357)
Receiving objects: 100% (6634/6634), 24.91 MiB | 12.97 MiB/s, done.
Resolving deltas: 100% (1366/1366), done.
Checking connectivity... done.
Checking out files: 100% (5880/5880), done.
```

注：如果你所在的项目git服务器已支持git-lfs，对二进制文件进行了区别管理，那么克隆工程的时候务必使用`git lfs clone`。否则克隆操作无法下载到工程中的二进制文件，工程内容不完整。

3 Git常用命令

命令拓展: <https://git-scm.com/docs/git-add>

3.2 新增/删除/移动文件到暂存区

在提交你修改的文件之前，需要`git add`把文件添加到暂存区。

如果该文件是新创建，尚未被git跟踪的，需要先执行 `git add` 将该文件添加到暂存区，再执行提交。

如果文件已经被git追踪，即曾经提交过的。在早期版本的git中，需要`git add`再提交；在较新版本的git中，不需要`git add`即可提交。

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    extend.txt

nothing added to commit but untracked files present (use "git add" to track)
100295060@HIGHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git add extend.txt
```

3 Git常用命令

3.2 新增/删除/移动文件到暂存区

git rm 将指定文件彻底从当前分支的缓存区删除，因此它从当前分支的下一个提交快照中被删除。

如果一个文件被`git rm`后进行了提交，那么它将脱离git跟踪，这个文件在之后的节点中不再受git工程的管理。

执行`git rm`后，该文件会在缓存区消失。

你也可以直接从硬盘上删除文件，然后对该文件执行`git commit`，git会自动将删除的文件从索引中移除，效果一样。

```
100295060@GHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git rm README.md
rm 'README.md'

100295060@GHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    README.md

100295060@GHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git commit -m "remove README.md from git project"
[master 1a10a40] remove README.md from git project
 1 file changed, 1 deletion(-)
 delete mode 100644 README.md
```

命令拓展：<https://git-scm.com/docs/git-rm>

3 Git常用命令

命令拓展: <https://git-scm.com/docs/git-mv>

3.2 新增/删除/移动文件到暂存区

git mv 命令用于移动文件，也可以用于重命名文件。

例1：需要将文件codehunter_nginx.conf从当前目录移动到config目录下，可执行：

```
git mv codehunter_nginx.conf config
```

```
git mv codehunter_nginx.conf config
```

例2：需要将文件codehunter_nginx.conf重命名为new_nginx.conf，可执行：

```
git mv config/codehunter_nginx.conf config/new_nginx.conf
```

```
git mv config/codehunter_nginx.conf config/new_nginx.conf
```

命令拓展: <https://git-scm.com/docs/git-clone>

3 Git常用命令

3.3 查看工作区

`git diff`用于比较项目中任意两个版本（分支）的差异，也可以用来比较当前的索引和上次提交间的差异。

当然它还有其他用途，在后文会进一步描述。

比较两个节点之间的差异

```
$ git diff 423b7e8 f2efb8f
diff --git a/README.md b/README.md
index 6aaaf099..47be1ad 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,3 @@
-Start MongoDB: mongod --dbpath=/media/root/_workspace/MongoData/
\ No newline at end of file
+Start MongoDB: mongod --dbpath=/media/root/_workspace/MongoData/
```

当前的索引和上次提交间的差异

```
$ git diff --cached
diff --git a/modules/__init__.py b/modules/__init__.py
index e69de29..0519ecb 100644
--- a/modules/__init__.py
+++ b/modules/__init__.py
@@ -0,0 +1 @@
+
\ No newline at end of file
```

比较两个分支之间的差异

```
$ git diff master..lin/develop/framework
diff --git a/README.md b/README.md
index 6aaaf099..e69de29 100644
--- a/README.md
+++ b/README.md
@@ -1 +0,0 @@
-Start MongoDB: mongod --dbpath=/media/root/_workspace/MongoData/
\ No newline at end of file
diff -git a/app/ init_ mv b/app/ init_ mv
```

在diff后面加--name-status参数，只看文件列表

```
$ git diff master..lin/develop/framework --name-status
M README.md
M app/__init__.py
A app/__init__.pyc
D app/random
D app/static/assets/css/common.scss
D app/static/assets/css/header.scss
D app/static/assets/css/responsive.scss
D app/static/assets/css/style.scss
D app/static/assets/css/variables.scss
```

命令拓展: <https://git-scm.com/docs/git-diff>

3 Git常用命令

3.3 查看工作区

`git status` 命令用于显示工作目录和暂存区的状态。

使用此命令能看到修改的git文件是否已被暂存, 新增的文件是否纳入了git版本库的管理。

下例中的信息表明: modules/_init_.py已被修改并暂存, LICENSE已被修改但未暂存, README.md已被删除但未暂存, extend.txt已被新建但未跟踪。注意, 请保证能理解`git status`回显的每一行文字含义。

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   modules/_init_.py

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   LICENSE
    deleted:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    extend.txt
```

3 Git常用命令

命令拓展: <https://git-scm.com/docs/git-commit>

3.4 提交更改的文件

`git commit` 主要是将暂存区里的文件改动提交到本地的版本库。

在此强调，提交这个动作是本地动作，是往本地的版本库中记录改动，不影响远端服务器。`git commit`一般需要附带提交描述信息，所以常见用法是：`git commit file_name -m "commit message"`

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 4 commits.
  (use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:   codehunter_nginx.conf -> config/new_nginx.conf

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:  LICENSE

100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git commit LICENSE -m "edit LICENSE for website"
[master 61e127f] edit LICENSE for website
 1 file changed, 1 insertion(+)

100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 5 commits.
  (use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:   codehunter_nginx.conf -> config/new_nginx.conf
```

提交成功后，git日志可查到此次提交的id和提交描述信息。

```
$ git log
commit 61e127f3000f8647beb420a534ae1ea55e910b3c
Author: 100295060 <100295060@huawei.com>
Date:   Wed Jun 12 18:31:42 2019 +0800

  edit LICENSE for website
```

如果要一次性提交所有在暂存区改动的文件到版本库，可以执行：

```
git commit -am "commit message"
```

命令拓展: <https://git-scm.com/docs/git-log>

3 Git常用命令

3.5 查看日志

git log用于查看提交历史。

默认加其他参数的话, **git log** 会按提交时间由近到远列出所有的历史提交日志。每个日志基本包含提交节点、作者信息、提交时间、提交说明等。

常用的日志命令格式: **git log**

git log配合不同参数具有相当灵活强大的展示功能, 常见的如--name-status/-p/--pretty/--graph等等, 有兴趣的同学课后自行了解。

```
100295060@HGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git log
commit 7fdd27236d485f6306986dda970068eed6ae0570
Author: 100295060 <100295060@huawei.com>
Date:   Wed Jun 12 18:42:47 2019 +0800

    nginx config edit

commit aa8747b5b20811970525e98a84681629635238e3
Author: linmu 00295060 <linmu@huawei.com>
Date:   Mon May 27 09:06:03 2019 +0800

    Update README.md

commit c4dfda3c27b1373992fc9602daac28ff26dc9616
Author: linmu 00295060 <linmu@huawei.com>
Date:   Sat May 25 16:28:02 2019 +0800

    Update README.md

commit 1ca3250a505083777d405b45f0d7b2c481c3ac41
Author: 100295060 <100295060@huawei.com>
Date:   Thu Jun 6 15:21:25 2019 +0800

    cmetricss general page
```

命令拓展: <https://git-scm.com/docs/git-push>

3 Git常用命令

3.6 推送至远端仓库

在使用`git commit`命令将自己的修改从暂存区提交到本地版本库后，可以使用`git push`将本地版本库的分支推送到远程服务器上对应的分支。

成功推动远端仓库后，其他开发人员可以获取到你新提交的内容。

常用的推送命令格式: `git push origin branch_name`

`branch_name`决定了你的本地分支推送成功后，在远端服务器上的分支名，其他人据此可以获取该分支上的改动内容。

你的本地分支名可以与推送到远端的分支名不同 : `git push origin branch_name:new_branch_name`

```
100295060@GHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 287 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
To git@code-sh.rnd.huawei.com:100295060/CodeHunter.git
 423b7e8..47cb197  master -> master
```

```
100295060@GHY4L002950601 MINGW64 /e/my_work/CodeHunter (rua)
$ git push origin rua:master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 276 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
To git@code-sh.rnd.huawei.com:100295060/CodeHunter.git
 47cb197..798c373  rua -> master
```

命令拓展：<https://git-scm.com/docs/git-branch>

3 Git常用命令

3.7 分支管理

git branch命令即可查看本地工程的所有git分支名称。

下图可见，git返回了当前本地工程所有的分支名称，其中master分支前面的“*”表示——当前工作区所在的分支是master。

```
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git branch
* master
  rua
  tr/linmu/test
```

如果想查看远端服务器上拥有哪些分支，那么执行git branch -r即可，返回的分支名带origin前缀，表示在远端；如果想查看远端服务器和本地工程所有的分支，那么执行git branch -a即可。

```
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git branch -r
  origin/HEAD -> origin/master
  origin/lin/develop/framework
  origin/lin/develop/newweb
  origin/master
```

```
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git branch -a
* master
  rua
  tr/linmu/test
  remotes/origin/HEAD -> origin/master
  remotes/origin/lin/develop/framework
  remotes/origin/lin/develop/newweb
  remotes/origin/master
```

3 Git常用命令

命令拓展：<https://git-scm.com/docs/git-branch>
& <https://git-scm.com/docs/git-checkout>

3.7 分支管理

git branch和**git checkout -b**的异同：

相同点：

git branch和**git checkout -b**都可以用于新建分支（默认基于当前分支节点创建）。

区别点：

git branch新建分支后并不会切换到新分支；

git checkout -b新建分支后会自动切换到新分支。

常用的新建分支命令格式：**git branch new_branch_name / git checkout -b branch_name**

```
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git branch linmu/newwork
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git branch
  lin/newwork
  linmu/newwork
* master
    rua
    tr/linmu/test
```

```
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$ git checkout -b bugfix_branch_1
Switched to a new branch 'bugfix_branch_1'
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git branch
* bugfix_branch_1
  lin/newwork
  linmu/newwork
  master
    rua
    tr/linmu/test
```

命令拓展: <https://git-scm.com/docs/git-branch>

3 Git常用命令

3.7 分支管理

`git branch -d`和`git branch -D`都可以用来删除本地分支，后者大写表示强制删除。

有时候当分支上包含了未合并的改动，或者当分支是当前所在分支，则-d无法删除，需要使用强制删除来达到目的。

常用的删除分支命令格式: `git branch -d branch_name/git branch -D branch_name`

删除服务器上的远程分支可以使用`git branch -d -r branch_name`，其中
branch_name为本地分支名。

删除后，还要推送到服务器上才行，即
`git push origin : branch_name`

```
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git branch
* bugfix_branch_1
  lin/newwork
  linmu/newwork
  master
  rua
  tr/linmu/test

100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git branch -d tr/linmu/test
error: The branch 'tr/linmu/test' is not fully merged.
If you are sure you want to delete it, run 'git branch -D tr/linmu/test'.

100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git branch -D tr/linmu/test
Deleted branch tr/linmu/test (was 798c373).

100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git branch
* bugfix_branch_1
  lin/newwork
  linmu/newwork
  master
  rua
```

命令拓展: <https://git-scm.com/docs/git-checkout>

3 Git常用命令

3.7 分支管理

`git checkout` 命令除了创建分支，还用来切换分支，当然比较官方的叫法是“检出”。

有时候，当前分支工作区存在修改而未提交的文件，与目的分支上的内容冲突，会导致checkout切换失败，这时候，可以使用`git checkout -f`进行强制切换。

常用的切换分支命令格式: `git checkout branch_name`

git checkout对象可以是分支，也可以是某个提交节点或者节点下的某个文件。建议课后自行按需了解。

```
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (linmu/newwork)
$ git branch
  bugfix_branch_1
  lin/newwork
* linmu/newwork
  master
  rua

100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (linmu/newwork)
$ git checkout master
Switched to branch 'master'.
Your branch is up-to-date with 'origin/master'.

100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (master)
$
```

命令拓展: <https://git-scm.com/docs/git-pull>

3 Git常用命令

3.7 分支管理

git pull的作用是，从远端服务器中获取某个分支的更新，再与本地指定的分支进行自动合并。

常用的切换分支命令格式: git pull origin remote_branch:local_branch

如果远程指定的分支与本地指定的分支相同，则可直接执行git pull origin remote_branch

```
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (linmu/newwork)
$ git pull origin lin/develop/newweb:master
From code-sh.rnd.huawei.com:100295060/CodeHunter
+ 798c373...1ca3250 lin/develop/newweb -> master (forced update)
Merge made by the 'recursive' strategy.
 app/static/dist/css/main.min.css           40 +-
 app/static/dist/js/highcharts.js          903 ++++++-----+
 app/templates/index.html                  1 -
 app/templates/projecttrend.html          267 ++++++-
 app/views.py                             5 +-
 config/config.json                      7 +-
 modules/analyzeduplication/analyze_dup_file.py 8 +
 modules/analyzequality/__init__.py        17 +
 modules/analyzequality/analyze_merge_comment.py 17 +
 modules/analyzequality/analyze_module_dts.py   17 +
 modules/analyzequality/analyze_stuff_commit.py 17 +
 modules/analyzequality/analyze_stuff_dts.py    17 +
 modules/defectstrace/getdts.py            69 +-
 modules/iooperate/jsonoperate.py          4 +-
 modules/metrics/__init__.py              17 +
 modules/metrics/analyze_metrics.py       102 +++
 modules/passwdsafe/pwdprocess.py         2 +-
17 files changed, 999 insertions(+), 511 deletions(-)
create mode 100644 modules/analyzequality/__init__.py
create mode 100644 modules/analyzequality/analyze_merge_comment.py
create mode 100644 modules/analyzequality/analyze_module_dts.py
create mode 100644 modules/analyzequality/analyze_stuff_commit.py
create mode 100644 modules/analyzequality/analyze_stuff_dts.py
      es/metrics/__init__.py
create mode 100644 modules/metrics/analyze_metrics.py
```

命令拓展: <https://git-scm.com/docs/git-fetch>

3 Git常用命令

3.7 分支管理

`git fetch`的作用是，从远端服务器中获取某个分支的更新到本地仓库。

注意，与`git pull`不同，`git fetch`在获取到更新后，并不会进行合并（即下页中的`git merge`）操作，这样能留给用户一个操作空间，确认`git fetch`内容符合预期后，再决定是否手动合并节点。

常用的获取远端分支更新命令格式：`git fetch origin remote_branch:local_branch`

如果远程指定的分支与本地指定的分支相同，则可直接执行`git fetch origin remote_branch`

```
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git fetch origin lin/develop/newweb
remote: Enumerating objects: 522, done.
remote: Counting objects: 100% (522/522), done.
remote: Compressing objects: 100% (413/413), done.
remote: Total 522 (delta 69), reused 288 (delta 46)
Receiving objects: 100% (522/522), 221.36 KiB | 0 bytes/s, done.
Resolving deltas: 100% (69/69), done.
From code-sh.rnd.huawei.com:100295060/CodeHunter
 * branch      lin/develop/newweb -> FETCH_HEAD
   1ca3250..d70bcbf  lin/develop/newweb -> origin/lin/develop/newweb
```

命令拓展：<https://git-scm.com/docs/git-merge>

3 Git常用命令

3.7 分支合并

git merge命令是用于从指定的分支（节点）合并到当前分支的操作

git会将指定的分支与当前分支进行比较，找出二者最近的一个共同节点base，之后将指定分支在base之后分离的节点合并到当前分支上。分支合并，实际上是分支间差异提交节点的合并。

常用的切换分支命令格式：`git merge branch_name`

```
100295060@HGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git merge linmu/newwork
Updating 47cb197..03d0419
Fast-forward
 README.md           2 ++
 app/static/dist/css/main.min.css   40 ++
 app/static/dist/js/highcharts.js    903 ++++++-----+
 app/templates/index.html          1 -
 app/templates/projecttrend.html  267 ++++++-+
 app/views.py                  5 ++
 config/config.json            7 ++
 modules/analyzeduplication/analyze_dup_file.py  8 +
 modules/analyzequality/__init__.py      17 +
 modules/analyzequality/analyze_merge_comment.py  17 +
 modules/analyzequality/analyze_module_dts.py    17 +
 modules/analyzequality/analyze_stuff_commit.py  17 +
 modules/analyzequality/analyze_stuff_dts.py     17 +
 modules/defectstrace/getdts.py        69 ++
 modules/iooperate/jsonoperate.py      4 ++
 modules/metrics/__init__.py         17 +
 modules/metrics/analyze_metrics.py  102 +++
 modules/passwdsafe/pwdprocess.py    2 ++
18 files changed, 1000 insertions(+), 512 deletions(-)
create mode 100644 modules/analyzequality/__init__.py
create mode 100644 modules/analyzequality/analyze_merge_comment.py
create mode 100644 modules/analyzequality/analyze_module_dts.py
create mode 100644 modules/analyzequality/analyze_stuff_commit.py
create mode 100644 modules/analyzequality/analyze_stuff_dts.py
create mode 100644 modules/metrics/__init__.py
create mode 100644 modules/metrics/analyze_metrics.py
```

命令拓展: <https://git-scm.com/docs/git-rebase>

3 Git常用命令

3.7 分支合并

`git rebase`用于合并目标分支到当前分支。

`git rebase`这条命令用于分支合并, `git merge`也是用于分支合并。如果你要将其他分支的提交节点合并到当前分支, 那么`git rebase`和`git merge`都可以达到目的。¹⁴

常用的合并命令格式: `git rebase branch_name`

`git rebase`、`git merge`背后的实现机制

和对合并后节点造成的影响有很大差异, 有各自的风险存在, 暂不在本课中展开。

```
100295060@HGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git log -2
commit c495a7708b7551b6a3b6b661b9092ba008864967
Merge: 47cb197 1ca3250
Author: 100295060 <100295060@huawei.com>
Date:   Sat Jun 15 12:34:55 2019 +0800

    Merge branch 'lin/develop/newweb' of code-sh.rnd.huawei.com:100295060/CodeHunter
    into linmu/newwork

commit 47cb197860da94ac9d948d755c7346dd7dccd6c6
Author: 100295060 <100295060@huawei.com>
Date:   Wed Jun 12 18:42:47 2019 +0800

    nginx config edit

100295060@HGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: Update README.md
Applying: Update README.md
Applying: nginx config edit

100295060@HGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git log -2
commit 7fdd27236d485f6306986dda970068eed6ae0570
Author: 100295060 <100295060@huawei.com>
Date:   Wed Jun 12 18:42:47 2019 +0800

    nginx config edit
```

命令拓展: <https://git-scm.com/docs/git-reset>

3 Git常用命令

3.8 撤销操作

`git reset`通常用于撤销当前工作区中的某些`git add/commit`操作，可将工作区内容回退到历史提交节点。

常用的工作区回退命令格式: `git reset commit_id`

```
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git log -2
commit 03d0419942a0d8bddfeabefeff0988bd7a3a2be
Author: 100295060 <100295060@huawei.com>
Date:   Sat Jun 15 12:47:21 2019 +0800

    add info

commit c495a7708b7551b6a3b6b661b9092ba008864967
Merge: 47cb197 1ca3250
Author: 100295060 <100295060@huawei.com>
Date:   Sat Jun 15 12:34:55 2019 +0800

    Merge branch 'lin/develop/newweb' of code-sh.rnd.huawei.com:100295060/CodeHu
    nter into linmu/newwork

100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git reset c495a7708b7551b6a3b6b661b9092ba008864967
Unstaged changes after reset:
M     README.md

100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git log -2
commit c495a7708b7551b6a3b6b661b9092ba008864967
Merge: 47cb197 1ca3250
Author: 100295060 <100295060@huawei.com>
Date:   Sat Jun 15 12:34:55 2019 +0800

    Merge branch 'lin/develop/newweb' of code-sh.rnd.huawei.com:100295060/CodeHu
    nter into linmu/newwork
26
```

注: `git reset --mixed/hard/soft`的三种参数模式涉及概念较多, 不在此区分讲解, 请课后按实际需要去了解。

命令拓展: <https://git-scm.com/docs/git-checkout>

3 Git常用命令

3.8 撤销操作

git checkout . 用于回退本地所有修改而未提交的文件内容。

git checkout . 这是条有风险的命令，因为它会取消本地工作区的修改(相对于暂存区)，用暂存区的所有文件直接覆盖本地文件，达到回退内容的目的。但它不给用户任何确认机会，所以谨慎使用。

常用的回退命令格式: **git checkout .**

如果仅仅想回退某个文件的未提交改动，
可以使用**git checkout -filename**来达到目的；如果想将工具区回退（检出）到某个提交版本，可以使用**git checkout commit_id**。

```
100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git status
On branch bugfix_branch_1
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git checkout .

100295060@IGHY4L002950601 MINGW64 /e/my_work/CodeHunter (bugfix_branch_1)
$ git status
On branch bugfix_branch_1
nothing to commit, working directory clean
```