Git分布式版本控制工具

1. Git概述

1.1 Git历史

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）

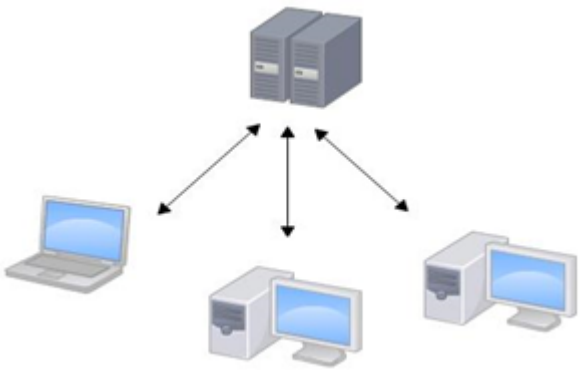
1.2 Git与SVN对比

SVN是集中式版本控制系统，版本库是集中放在中央服务器的，而开发人员工作的时候，用的都是自己的电脑，所以首先要从中央服务器下载最新的版本，然后开发，开发完后，需要把自己开发的代码提交到中央服务器。

集中式版本控制工具缺点：

服务器单点故障

容错性差



Git是分布式版本控制系统（Distributed Version Control System，简称 DVCS），分为两种类型的仓库：

本地仓库和远程仓库

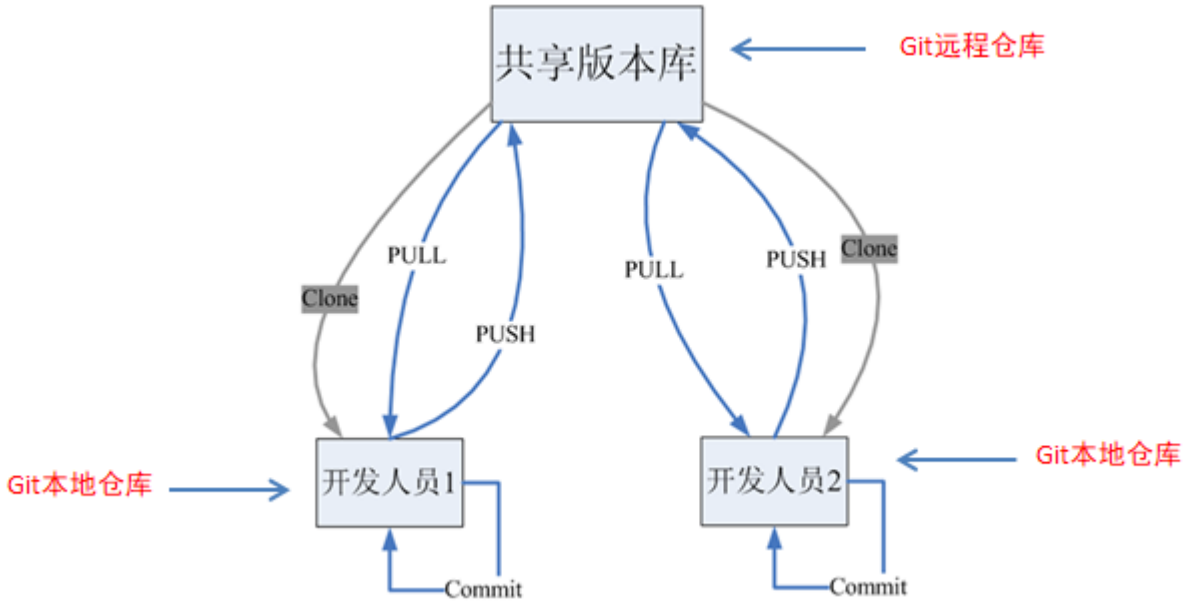
本地仓库：是在开发人员自己电脑上的Git仓库

远程仓库：是在远程服务器上的Git仓库

Clone：克隆，就是将远程仓库复制到本地

Push：推送，就是将本地仓库代码上传到远程仓库

Pull：拉取，就是将远程仓库代码下载到本地仓库



1.3 Git工作流程

工作流程如下：

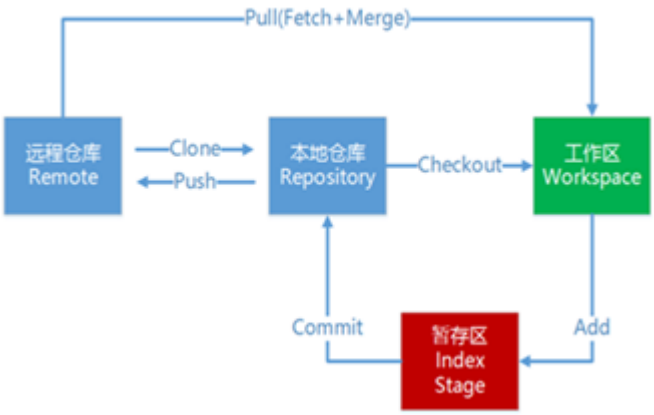
1．从远程仓库中克隆代码到本地仓库

2．从本地仓库中checkout代码然后进行代码修改

3．在提交前先将代码提交到暂存区

4．提交到本地仓库。本地仓库中保存修改的各个历史版本

5．修改完成后，需要和团队成员共享代码时，将代码push到远程仓库



1.4 Git下载与安装

下载地址： https://git-scm.com/download

下载完成后可以得到如下安装文件：

Git-2.20.1-64-bit.exe

2. Git代码托管服务

2.1 常用的Git代码托管服务

前面我们已经知道了Git中存在两种类型的仓库，即本地仓库和远程仓库。那么我们如何搭建Git远程仓库呢？我们可以借助互联网上提供的一些代码托管服务来实现，其中比较常用的有GitHub、码云、GitLab等。

gitHub（地址：https://github.com/）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

码云（地址： https://gitee.com/）是国内的一个代码托管平台，由于服务器在国内，所以相比于GitHub，码云速度会更快

GitLab （地址： https://about.gitlab.com/）是一个用于仓库管理系统的开源项目，使用Git作为代码管理工具，并在此基础上搭建起来的web服务

2.2 在码云注册账号

要想使用码云的相关服务，需要注册账号（地址： https://gitee.com/signup ）

2.3 登录码云并创建Git远程仓库

注册完成后就可以使用刚刚注册的邮箱进行登录（地址： https://gitee.com/login ）

登录成功后就可以创建Git远程仓库

创建完成后可以查看仓库信息

每个Git远程仓库都会对应一个网络地址，可以点击克隆/下载按钮弹出窗口并点击复制按钮获得这个网络地址

我们当前创建的Git远程仓库对应的地址为：https://gitee.com/ChuanZhiBoKe/myGitRepo.git

2.4 邀请其他用户成为仓库成员

前面已经在码云上创建了自己的远程仓库，目前仓库成员只有自己一个人（身份为管理员）。在企业实际开发中，一个项目往往是由多个人共同开发完成的，为了使多个参与者都有权限操作远程仓库，就需要邀请其他项目参与者成为当前仓库的成员。

3. Git常用命令

3.1 环境配置

当安装Git后首先要做的事情是设置用户名称和email地址。这是非常重要的，因为每次Git提交都会使用该用户信息

设置用户信息

git config --global user.name “itcast”

git config --global user.email “hello@itcast.cn”

查看配置信息

git config --list

git config user.name

通过上面的命令设置的信息会保存在~/.gitconfig文件中

3.2 获取Git仓库

要使用Git对我们的代码进行版本控制，首先需要获得Git仓库

获取Git仓库通常有两种方式：

在本地初始化一个Git仓库

从远程仓库克隆

3.2.1在本地初始化一个Git仓库

执行步骤如下：

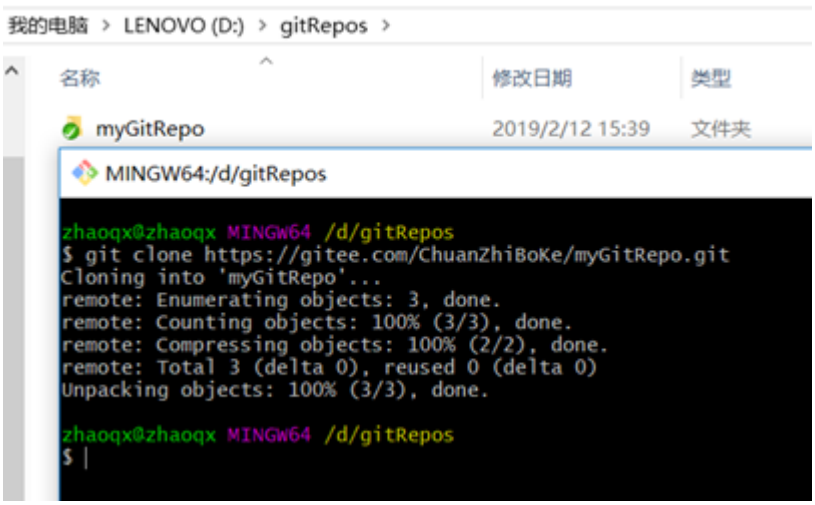
1. 在电脑的任意位置创建一个空目录（例如repo1）作为我们的本地Git仓库
2. 进入这个目录中，点击右键打开Git bash窗口
3. 执行命令git init

如果在当前目录中看到.git文件夹（此文件夹为隐藏文件夹）则说明Git仓库创建成功

3.2.2从远程仓库克隆

可以通过Git提供的命令从远程仓库进行克隆，将远程仓库克隆到本地

命令形式为：git clone 远程Git仓库地址



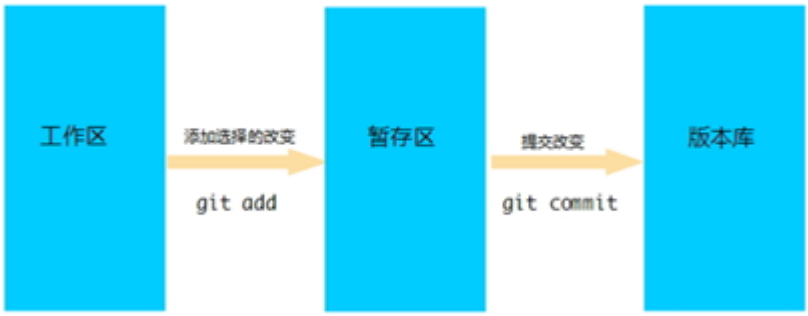
3.3工作目录、暂存区以及版本库概念

为了更好的学习Git，我们需要了解Git相关的一些概念，这些概念在后面的学习中会经常提到

版本库：前面看到的.git隐藏文件夹就是版本库，版本库中存储了很多配置信息、日志信息和文件版本信息等

工作目录（工作区）：包含.git文件夹的目录就是工作目录，主要用于存放开发的代码

暂存区：.git文件夹中有很多文件，其中有一个index文件就是暂存区，也可以叫做stage。暂存区是一个临时保存修改文件的地方



3.4 Git工作目录下文件的两种状态

Git工作目录下的文件存在两种状态：

untracked 未跟踪（未被纳入版本控制）

tracked 已跟踪（被纳入版本控制）

Unmodified 未修改状态

Modified 已修改状态

Staged 已暂存状态

这些文件的状态会随着我们执行Git的命令发生变化

3.5 本地仓库操作

git status 查看文件状态

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

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

        .gitignore

nothing added to commit but untracked files present (use "git add" to track)
```

也可以使用git status –s 使输出信息更加简洁

```
$ git status -s
?? .gitignore
```

git add 将未跟踪的文件加入暂存区

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/myGitRepo (master)
$ git add hello.txt
```

将新创建的文件加入暂存区后查看文件状态

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/myGitRepo (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   hello.txt
```

git reset 将暂存区的文件取消暂存

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/myGitRepo (master)
$ git reset hello.txt
```

将文件取消暂存后查看文件状态

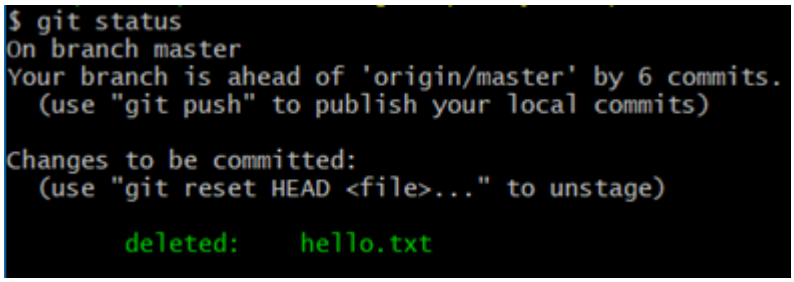
```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/myGitRepo (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

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

        hello.txt

nothing added to commit but untracked files present (use "git add" to track)
```

git commit 将暂存区的文件修改提交到本地仓库

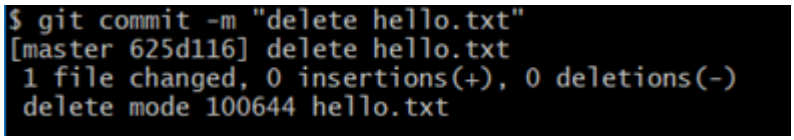
```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/myGitRepo (master)
$ git add hello.txt

zhaoqx@zhaoqx MINGW64 /d/gitRepos/myGitRepo (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   hello.txt


zhaoqx@zhaoqx MINGW64 /d/gitRepos/myGitRepo (master)
$ git commit -m "init hello.txt" hello.txt
[master ce61b24] init hello.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hello.txt
```

git rm 删除文件

删除文件后查看文件状态



上面删除的只是工作区的文件，需要提交到本地仓库



将文件添加至忽略列表

一般我们总会有些文件无需纳入Git 的管理，也不希望它们总出现在未跟踪文件列表。 通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。 在这种情况下，我们可以在工作目录中创建一个名为 .gitignore 的文件（文件名称固定），列出要忽略的文件模式。下面是一个示例：

```
# no .a files
*.a
# but do track lib.a, even though you're ignoring .a files above
!lib.a
# only ignore the TODO file in the current directory, not subdir/TODO
/TODO
# ignore all files in the build/ directory
build/
# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
# ignore all .pdf files in the doc/ directory
doc/**/*.pdf
```

git log 查看日志记录

```
$ git log
commit c426b432ed760a96842eb5a0b70fc89881cd5e31 (HEAD -> master)
Author: itcast <hello@itcast.cn>
Date:   Tue Feb 12 20:06:36 2019 +0800

    delete abc

commit 9c454c56168e7a4603659bc44a7bd335cb8d4aa5
Author: itcast <hello@itcast.cn>
Date:   Tue Feb 12 20:06:08 2019 +0800

    abc

commit 625d11650e7a45b1950a24a976de9e0137ec49d2
Author: itcast <hello@itcast.cn>
Date:   Tue Feb 12 20:04:10 2019 +0800

    delete hello.txt

commit 9f3a5aa855c93f3470f56bb29948101dd624b1de
Author: itcast <hello@itcast.cn>
Date:   Tue Feb 12 20:00:38 2019 +0800

    init hello.txt
```

3.6 远程仓库操作

前面执行的命令操作都是针对的本地仓库，本章节我们会学习关于远程仓库的一些操作，具体包括：

3.6.1查看远程仓库

如果想查看已经配置的远程仓库服务器，可以运行 git remote 命令。 它会列出指定的每一个远程服务器的简写。如果已经克隆了远程仓库，那么至少应该能看到 origin ，这是 Git 克隆的仓库服务器的默认名字

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/myGitRepo (master)
$ git remote
origin

zhaoqx@zhaoqx MINGW64 /d/gitRepos/myGitRepo (master)
$ git remote -v
origin  https://gitee.com/ChuanZhiBoKe/myGitRepo.git (fetch)
origin  https://gitee.com/ChuanZhiBoKe/myGitRepo.git (push)
```

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/myGitRepo (master)
$ git remote show origin
* remote origin
  Fetch URL: https://gitee.com/ChuanZhiBoKe/myGitRepo.git
  Push  URL: https://gitee.com/ChuanZhiBoKe/myGitRepo.git
  HEAD branch: master
  Remote branch:
    master tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (fast-forwardable)
```

3.6.2 添加远程仓库

运行 git remote add 添加一个新的远程 Git 仓库，同时指定一个可以引用的简写

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo1 (master)
$ git remote add origin https://gitee.com/ChuanZhiBoKe/repo1.git

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo1 (master)
$ git remote -v
origin  https://gitee.com/ChuanZhiBoKe/repo1.git (fetch)
origin  https://gitee.com/ChuanZhiBoKe/repo1.git (push)
```

3.6.3 从远程仓库克隆

如果你想获得一份已经存在了的 Git 仓库的拷贝，这时就要用到 git clone 命令。 Git 克隆的是该 Git 仓库服务器上的几乎所有数据（包括日志信息、历史记录等），而不仅仅是复制工作所需要的文件。 当你执行 git clone 命令的时候，默认配置下远程 Git 仓库中的每一个文件的每一个版本都将被拉取下来。

克隆仓库的命令格式是 git clone [url]

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos
$ git clone https://gitee.com/ChuanZhiBoKe/repo1.git
Cloning into 'repo1'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 13 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (13/13), done.
```

3.6.4 移除无效的远程仓库

如果因为一些原因想要移除一个远程仓库 ， 可以使用 git remote rm

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo1 (master)
$ git remote
origin

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo1 (master)
$ git remote rm origin

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo1 (master)
$ git remote

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo1 (master)
```

注意：此命令只是从本地移除远程仓库的记录，并不会真正影响到远程仓库

3.6.5 从远程仓库中抓取与拉取

git fetch 是从远程仓库获取最新版本到本地仓库，不会自动merge

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2
$ git init
Initialized empty Git repository in D:/gitRepos/repo2/.git/

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git remote add origin https://gitee.com/ChuanZhiBoKe/repo1.git

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git fetch origin master
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 13 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (13/13), done.
From https://gitee.com/ChuanZhiBoKe/repo1
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> origin/master

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git merge origin/master
```

git pull 是从远程仓库获取最新版本并merge到本地仓库

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2
$ git init
Initialized empty Git repository in D:/gitRepos/repo2/.git/

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git remote add origin https://gitee.com/ChuanZhiBoKe/repo1.git

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git pull origin master
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 13 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (13/13), done.
From https://gitee.com/ChuanZhiBoKe/repo1
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> origin/master
```

注意：如果当前本地仓库不是从远程仓库克隆，而是本地创建的仓库，并且仓库中存在文件，此时再从远程仓库拉取文件的时候会报错（fatal: refusing to merge unrelated histories ），解决此问题可以在git pull命令后加入参数--allow-unrelated-histories

3.6.6 推送到远程仓库

当你想分享你的代码时，可以将其推送到远程仓库。 命令形式：git git push [remote-name][branch-name]

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 238 bytes | 238.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Powered By Gitee.com
To https://gitee.com/ChuanZhiBoKe/repo1.git
   c55fbf7..3a3712b  master -> master
```

3.7 Git分支

几乎所有的版本控制系统都以某种形式支持分支。 使用分支意味着你可以把你的工作从开发主线上分离开来，以免影响开发主线。Git 的master分支并不是一个特殊分支。 它跟其它分支没有区别。 之所以几乎每一个仓库都有 master 分支，是因为git init 命令默认创建它，并且大多数人都懒得去改动它。

在本章节我们会学习到关于分支的相关命令，具体如下：

3.7.1 查看分支

列出所有本地分支

$ git branch

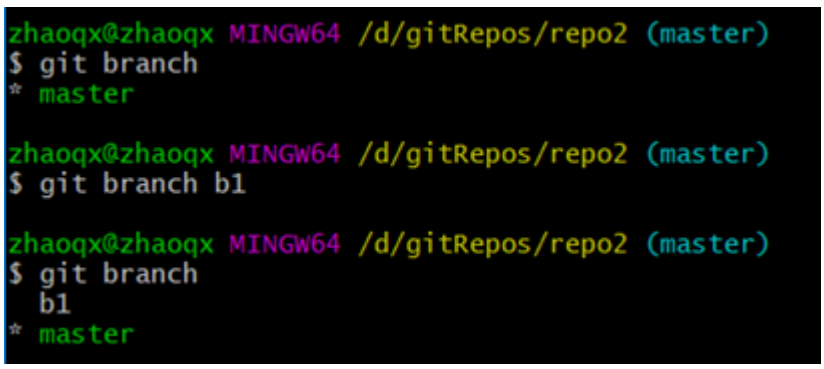
列出所有远程分支

$ git branch -r

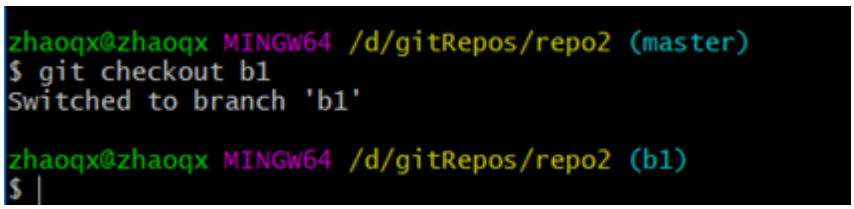
列出所有本地分支和远程分支

$ git branch -a

3.7.2 创建分支



3.7.3 切换分支



3.7.4 推送至远程仓库分支

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (b1)
$ git push origin b1
Enumerating objects: 22, done.
Counting objects: 100% (21/21), done.
Delta compression using up to 8 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (19/19), 1.52 KiB | 518.00 KiB/s, done.
Total 19 (delta 6), reused 0 (delta 0)
remote: Powered By Gitee.com
To https://gitee.com/ChuanZhiBoKe/repo1.git
 * [new branch]      b1 -> b1
```

3.7.5 合并分支

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git merge b3
Updating 18d8a0a..8a52d65
Fast-forward
 hello3.txt | 1 +
 1 file changed, 1 insertion(+)
```

有时候合并操作不会如此顺利。 如果你在两个不同的分支中，对同一个文件的同一个部分进行了不同的修改，Git就没办法合并它们，同时会提示文件冲突。此时需要我们打开冲突的文件并修复冲突内容，最后执行git add命令来标识冲突已解决

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git merge b3
Auto-merging UserMapper.xml
CONFLICT (content): Merge conflict in UserMapper.xml
Automatic merge failed; fix conflicts and then commit the result.
```

3.7.5 删除分支

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git branch -d b1
Deleted branch b1 (was 520a40d).

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git branch
* master
```

如果要删除的分支中进行了一些开发动作，此时执行上面的删除命令并不会删除分支，如果坚持要删除此分支，可以将命令中的-d参数改为-D

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git branch -d b2
error: The branch 'b2' is not fully merged.
If you are sure you want to delete it, run 'git branch -D b2'.

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git branch -D b2
Deleted branch b2 (was 075ab59).

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git branch
* master
```

注：如果要删除远程仓库中的分支，可以使用命令git push origin –d branchName

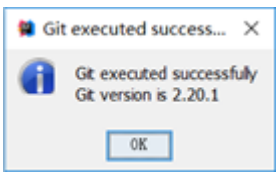
4. 在IDEA中使用Git

4.1 在IDEA中配置Git

安装好IntelliJ IDEA后，如果Git安装在默认路径下，那么idea会自动找到git的位置，如果更改了Git的安装位置则需要手动配置下Git的路径。

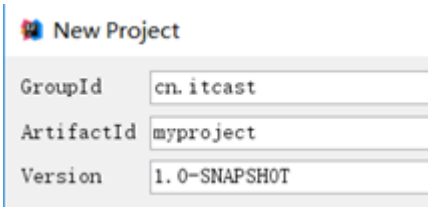
选择File→Settings打开设置窗口，找到Version Control下的git选项：

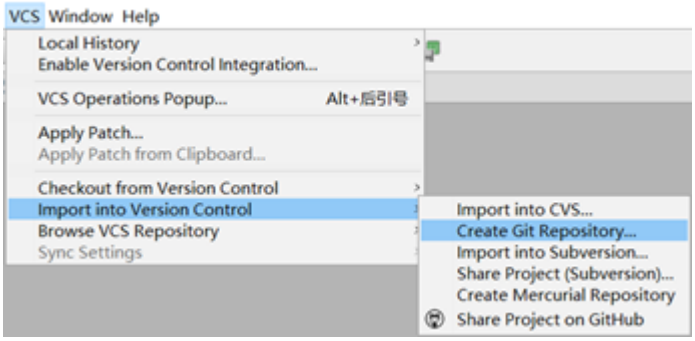
选择git的安装目录后可以点击“Test”按钮测试是否正确配置



4.2 在IDEA中使用Git

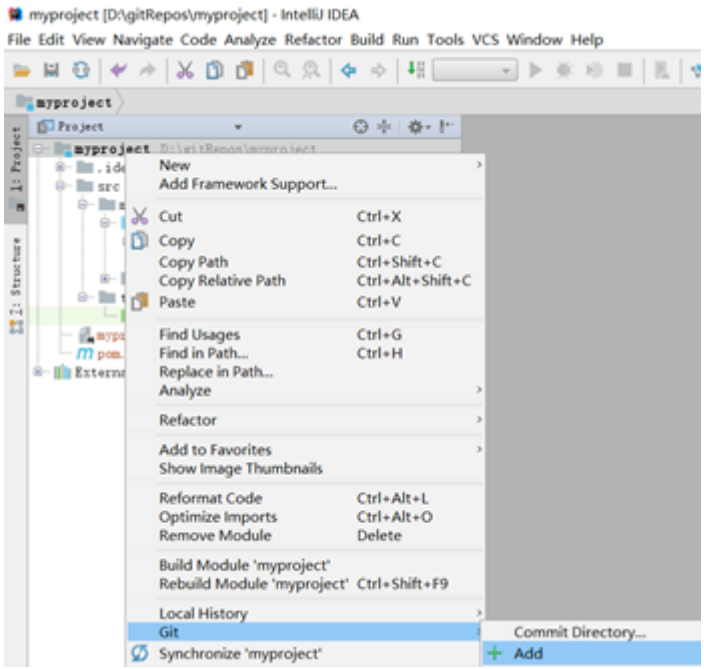
4.2.1在IDEA中创建工程并将工程添加至Git



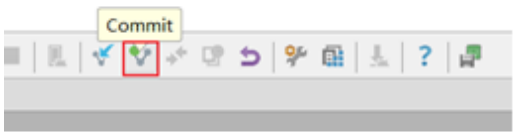


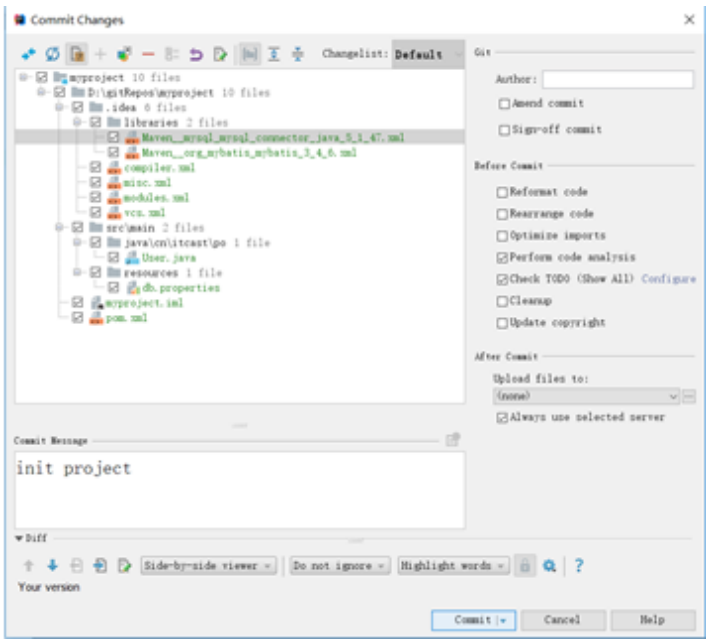
将项目添加至Git管理后，可以从IDEA的工具栏上看到Git操作的按钮

4.2.2 将文件添加到暂存区

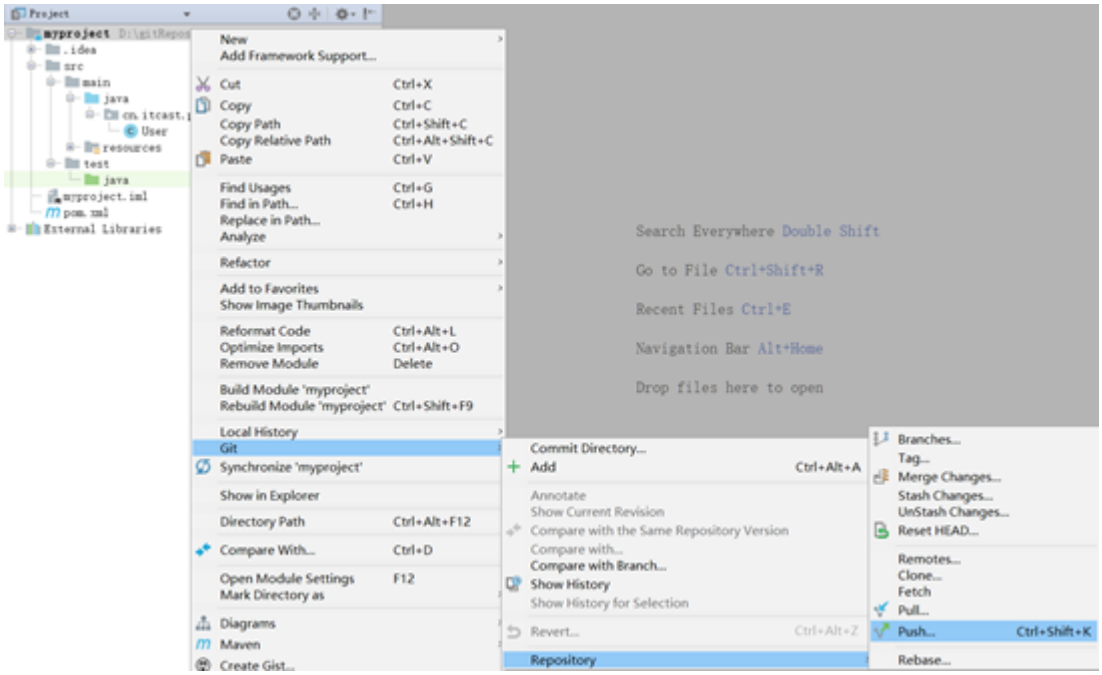


4.2.3 提交文件

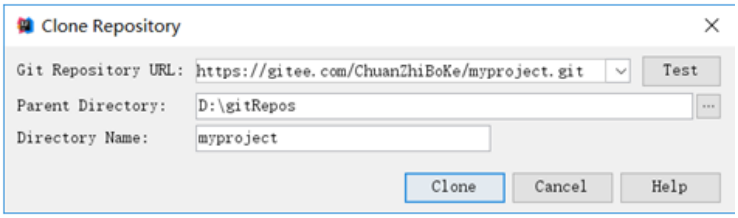




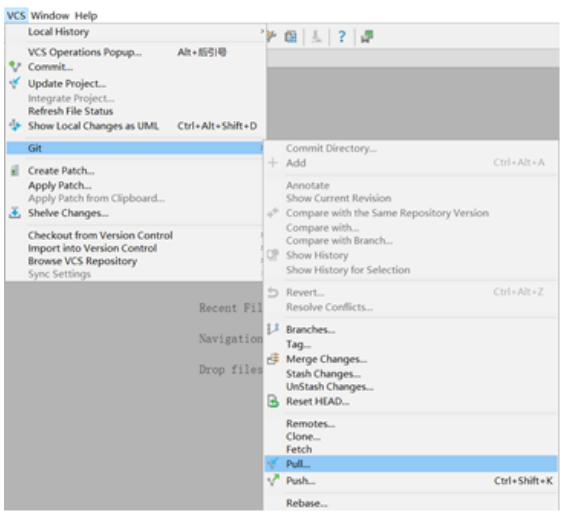
4.2.4 将代码推送到远程仓库

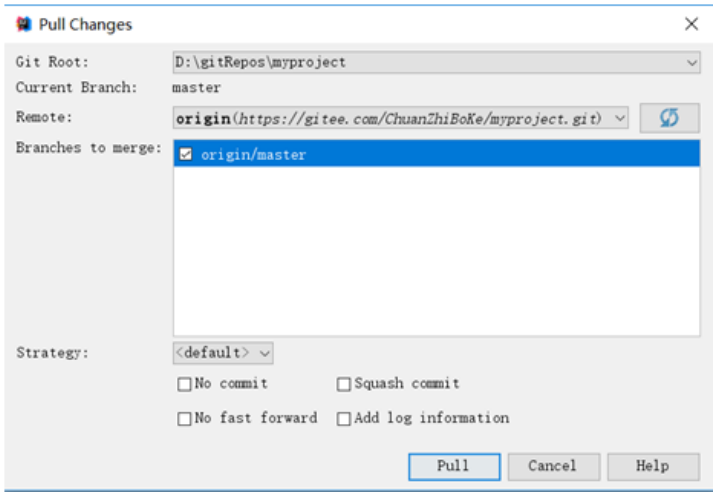


4.2.5 从远程仓库克隆工程到本地

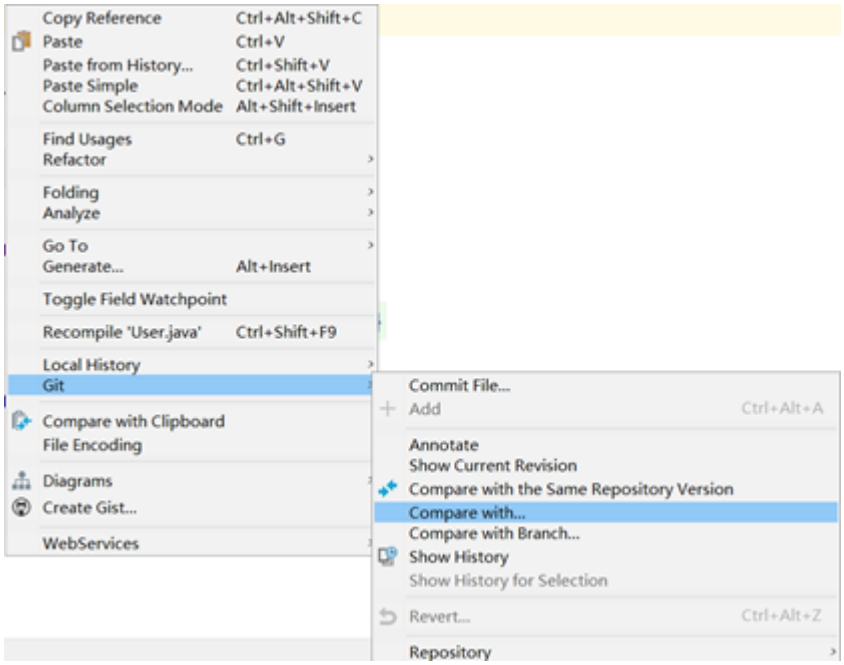


4.2.6 从远程拉取代码

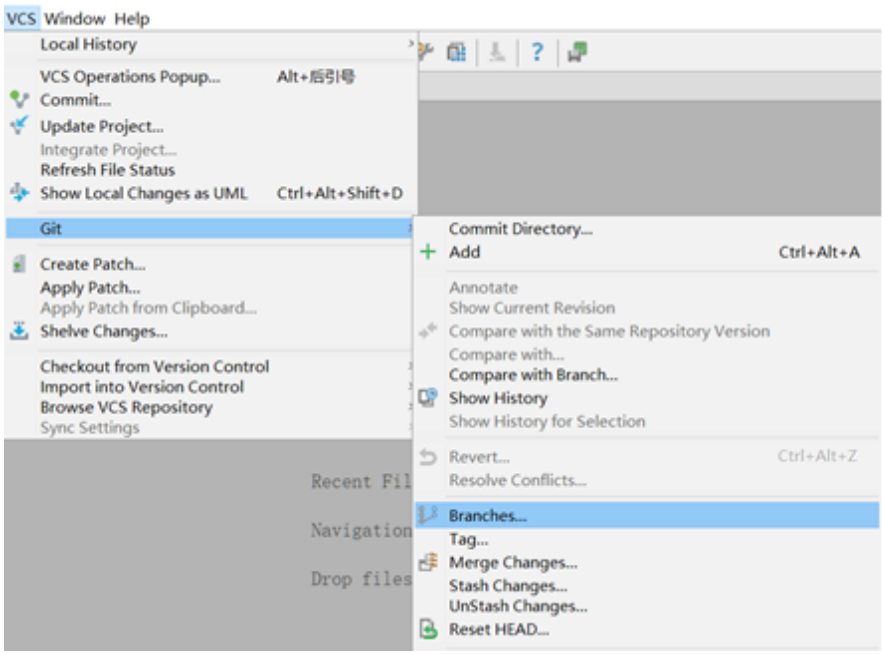




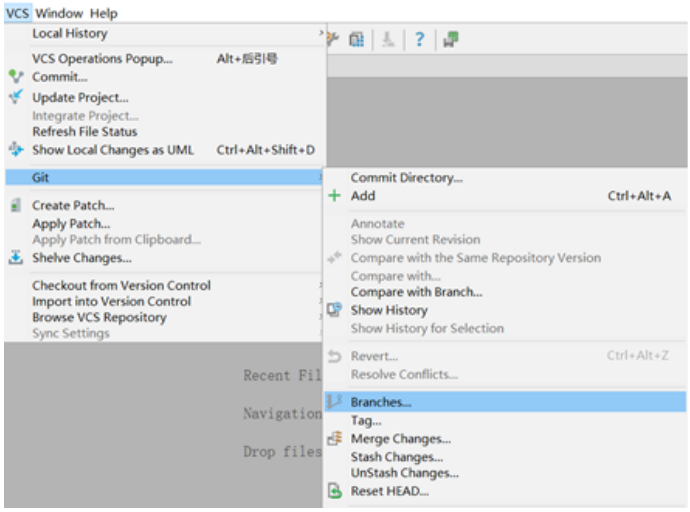
4.2.7 版本对比

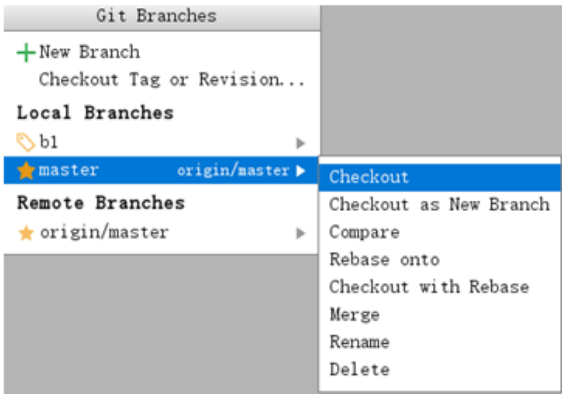


4.2.8 创建分支

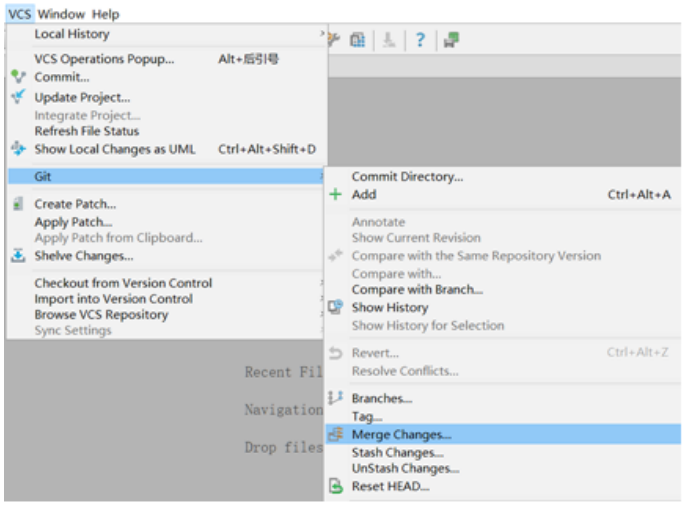


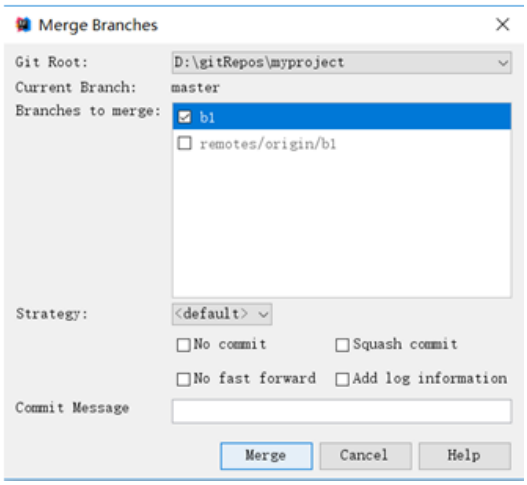
4.2.9 切换分支


VCS Window Help
Local History
VCS Operations Popup... Alt+后引号
Commit...
Update Project...
Integrate Project...
Refresh File Status
Show Local Changes as UML Ctrl+Alt+Shift+D
Git
Create Patch...
Apply Patch...
Apply Patch from Clipboard...
Shelve Changes...
Checkout from Version Control
Import into Version Control
Browse VCS Repository
Sync Settings
Commit Directory...
Add Ctrl+Alt+A
Annotate
Show Current Revision
Compare with the Same Repository Version
Compare with...
Compare with Branch...
Show History
Show History for Selection
Revert... Ctrl+Alt+Z
Resolve Conflicts...
Branches...
Tag...
Merge Changes...
Stash Changes...
UnStash Changes...
Reset HEAD...
Recent Fil
Navigation
Drop files


Git Branches
New Branch
Checkout Tag or Revision...
Local Branches
b1
master origin/master
Remote Branches
origin/master
Checkout
Checkout as New Branch
Compare
Rebase onto
Checkout with Rebase
Merge
Rename
Delete

4.2.10 分支合并


VCS Window Help
Local History
VCS Operations Popup... Alt+后引号
Commit...
Update Project...
Integrate Project...
Refresh File Status
Show Local Changes as UML Ctrl+Alt+Shift+D
Git
Create Patch...
Apply Patch...
Apply Patch from Clipboard...
Shelve Changes...
Checkout from Version Control
Import into Version Control
Browse VCS Repository
Sync Settings
Commit Directory...
Add Ctrl+Alt+A
Annotate
Show Current Revision
Compare with the Same Repository Version
Compare with...
Compare with Branch...
Show History
Show History for Selection
Revert... Ctrl+Alt+Z
Resolve Conflicts...
Branches...
Tag...
Merge Changes...
Stash Changes...
UnStash Changes...
Reset HEAD...
Recent Fil
Navigation
Drop files


Merge Branches
Git Root:
D:\gitRepos\myproject
Current Branch:
master
Branches to merge:
b1
remotes/origin/b1
Strategy:
<default>
No commit
Squash commit
No fast forward
Add log information
Commit Message
Merge
Cancel
Help