

```
// 1, C语言基础
// 2, STM32基础
// 3, 工具Git
// 4, FreeRTOS(实时操作系统)(8天)
// 5, Linux(1,项目:网络编程 2,C++ 3,面试/工作 )
// 6, 项目
// 7, C++
//...
```

```
// 寄存器
// 标准库
// HAL库
// 标准库+RTOS
// HAL+RTOS
```

Git

```
// Git是个版本控制工具
```

```
// 公司中开发流程
```

```
// 1, 调研(1-2产品经理, 1-2程序员)
// 2, +2程序员: 搭建框架 (框架建立, 引入必要的资源, 打通基本流程)
// 3, +3-5程序员: 写"业务逻辑"
// 4, 开发阶段 (程序员协同写代码) -> 测试 -> 生产
```

```
// 开发环境
// 测试环境
// 生产环境
```

1. 介绍

Git 是一个版本控制工具。可以记录 and 追踪 某个文件 在某一个时刻的内容和状态。

```
// 版本控制(Revisioncontrol)是维护工程蓝图的标准做法, 能追踪工程蓝图从诞生一直到定案的过程。是一种记录若干文件内容变化, 以便将来查阅特定版本修订情况的系统。
// Git 可以记录某个文件夹下的 不同文件 在不同时间节点的不同状态。Git可以去记录这些文件产生的变化
// 说白了其实就是记录某个目录下的文件不在时间点下的不同状态, 以版本为单位
```

如果没有版本控制:

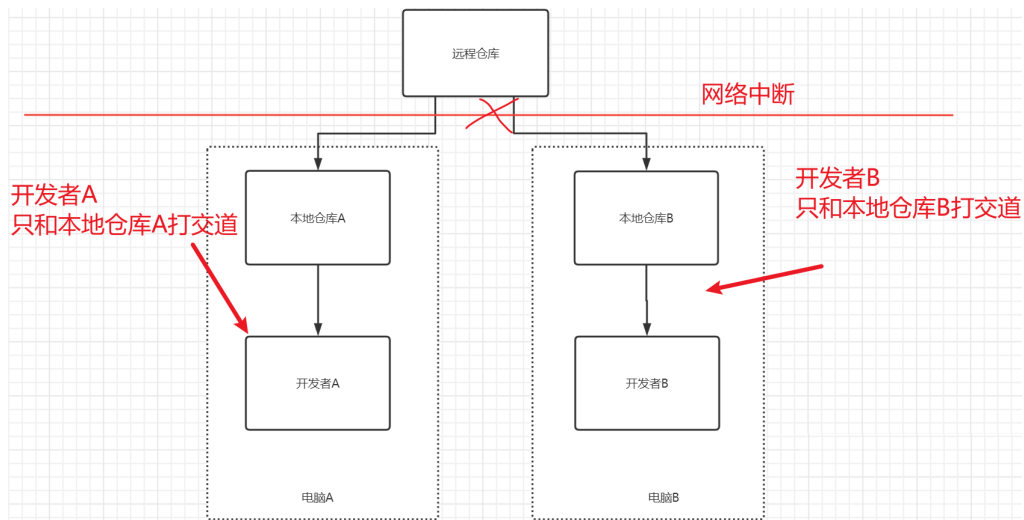
```
// 备份多个版本费空间, 费时间
// 难以恢复至正确版本
// 代码管理混乱, 容易引发冲突和Bug
// 无法进行权限控制
```

Git的历史

// **Linux**出品.2005年由于**BitKeeper**软件公司对**Linux**社区停止了免费使用权。 **Linux**迫不得已自己开发了一个分布式版本控制工具，从而**Git**诞生了。据说**Linux**花了两周时间自己用**C**写了一个 分布式版本控制系统，这就是**Git**！一个月之内， **Linux**系统的源码已经由**Git**管理了！

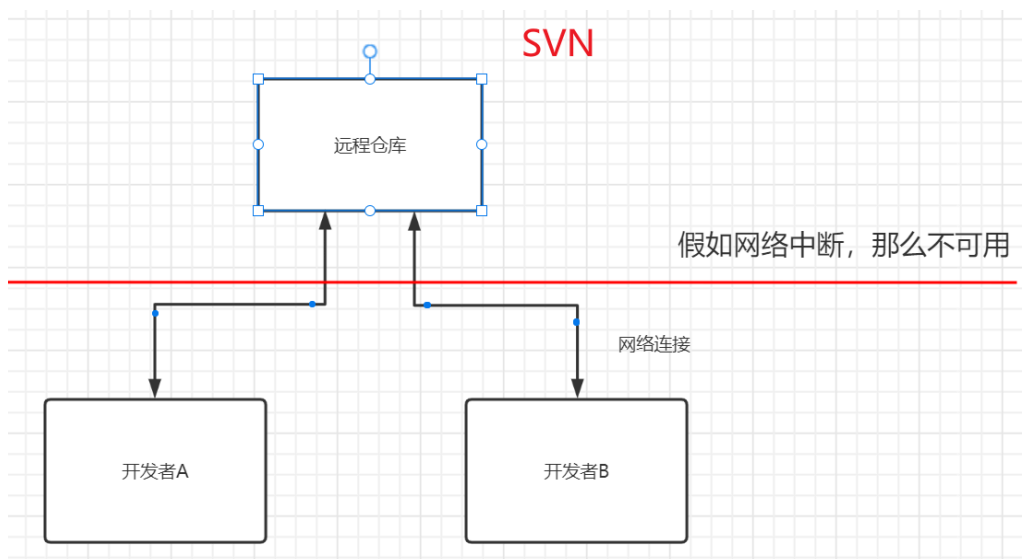
Git的特点

// 离线可用：(SVN不行)
// 可以回退



主流的管理工具对比: Git 80% 上; SVN 10%

// **Git**: 分布式版本控制工具
// **SVN**: 集中式版本控制工具



2. 安装Git

Git是个版本控制工具: 工具

2.1 Linux

正常来就那个我们在日常配置Linux其它代码开发环境的时候, 在安装某些工具的时候, 其中可能自动安装了git工具. 我们可以通过命令来确定Linux环境中是否已经具备git工具.

```
git --version
```

// 如果通过上述命令, 显示git的版本号, 则表明当前环境中已经具有急用git工具的能力

如果我们在通过上述命令检测的时候, 发现没有git工具:(如图)

```
snow@snow-virtual-machine:~$ git --version
Command 'git' not found, but can be installed with:
sudo apt install git
```

我们可以通过命令来安装git工具:

```
sudo apt install git
```

```
snow@snow-virtual-machine:~$ sudo apt install git
[sudo] password for snow:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk gitweb git-cv
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 4,817 kB of archives.
After this operation, 34.3 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://mirrors.huaweicloud.com/repository/ubuntu bionic/main amd64 liberror-perl all
Get:2 http://mirrors.huaweicloud.com/repository/ubuntu bionic-updates/main amd64 git-man a
Get:3 http://mirrors.huaweicloud.com/repository/ubuntu bionic-updates/main amd64 git amd64
Fetched 4,817 kB in 1s (5,599 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 172106 files and directories currently installed.)
Preparing to unpack .../liberror-perl_0.17025-1_all.deb ...
Unpacking liberror-perl (0.17025-1) ...
Selecting previously unselected package git-man.
Preparing to unpack .../git-man_1%3a2.17.1-1ubuntu0.18_all.deb ...
Unpacking git-man (1:2.17.1-1ubuntu0.18) ...
Selecting previously unselected package git.
Preparing to unpack .../git_1%3a2.17.1-1ubuntu0.18_amd64.deb ...
Unpacking git (1:2.17.1-1ubuntu0.18) ...
Setting up git-man (1:2.17.1-1ubuntu0.18) ...
Setting up liberror-perl (0.17025-1) ...
Setting up git (1:2.17.1-1ubuntu0.18) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
snow@snow-virtual-machine:~$ git --version
git version 2.17.1
snow@snow-virtual-machine:~$
```

2.2 Windows

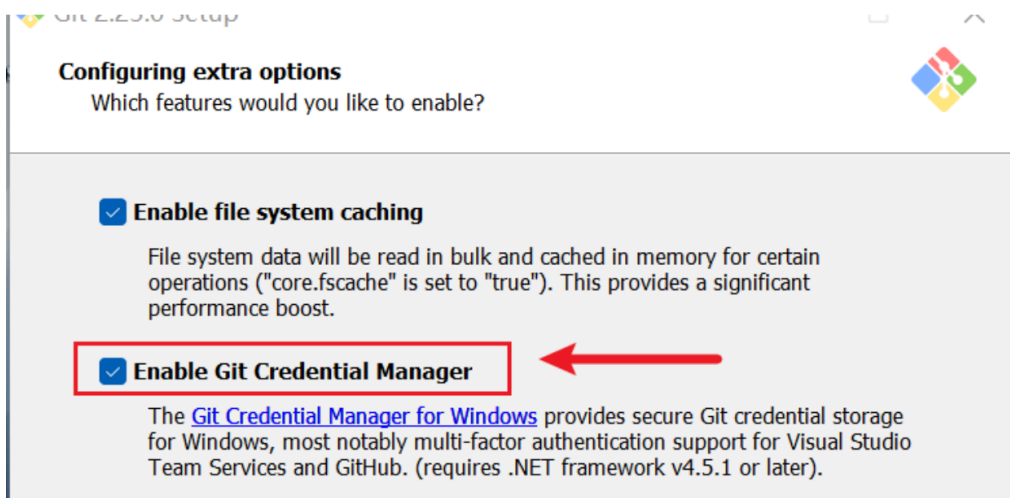
2.2.1 下载

[官方下载地址](#)

/wangdao/share/JAVA/soft/Git/				
名字	大小	已改变	权限	拥有者
..		2022/4/4 23:20:06	rw-rw-r--	root
Git-2.23.0-64-bit.exe	46,584 KB	2022/1/5 11:58:12	rw-r--r--	teacher

2.2.2 安装

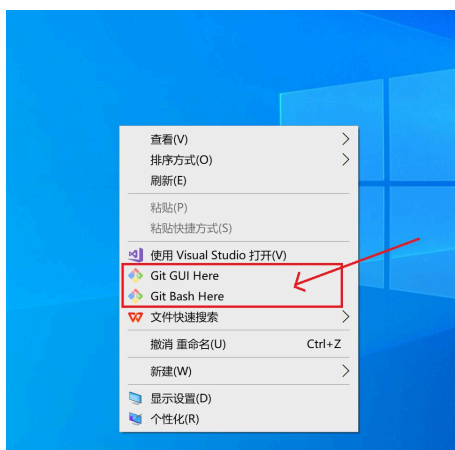
直接傻瓜式(一路默认, 唯一需要注意如下图)安装即可。



安装的时候, 要注意 让Windows 记住登录凭证

判断是否安装成功

如果在Windows的**任何路径下**都可以出现如下图所示的命令选项, 既是安装成功



2.3 一些注意事项

建议在安装git之后设置一些内容

```
// 建议设置有效的邮箱和用户名
git config --global user.email 222@qq.com
git config --global user.name youname

// 编辑工具
git config --global core.editor vim
```

在和Git服务器的交互中, 有时候要求输入账号密码(此账号密码为你的Git远程仓库上的账号密码)

```
// 正常来讲，在输入一次账号密码后，Git工具会自动记住账号和密码，下次使用就不需要输入了(和远程仓库交互的时候)，如果密码输错，则可能需要修改系统的记录凭证(windows中比较方便的操作)。或者清空缓存重置账号密码(Linux删除家目录下的 .git-credentials 文件)。建议还是不要输错了。
```

```
// 但是如果老是每次都和中央仓库交互都需要输入账号密码：  
// windows解决办法，执行： git config --global credential.helper manager  
// Linux解决办法，执行： git config --global credential.helper store  
// 或者各个系统的通用配置在用户名路径下的.gitconfig中  
// 配置：  
// [credential]  
// helper = store
```

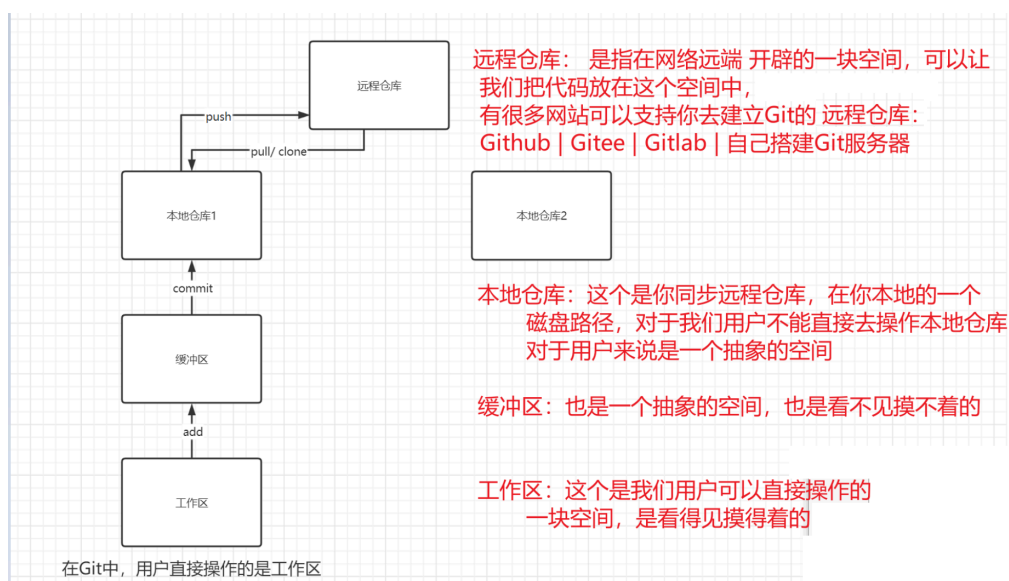
如果是第一次commit，需要设置用户名和邮件地址(建议设置真实的)

```
// 设置方式一：直接文件设置  
// 去用户目录下(无论是Linux还是Windows都在家目录下)，设置.gitconfig文件，假如没有这个文件，就创建一个  
[user]  
  email = 1111@qq.com  
  name = snow  
[credential]  
  helper = store  
  
// 设置方式二：命令设置  
git config --global user.email 222@qq.com  
git config --global user.name xxx
```

3. Git的使用

3.1 Git核心流程

Git的工作流程



3.2 Git 命令

3.2.1 注册相关网站

在不同Git仓库注册的时候, 一定要记住账号和密码和邮箱

以Gitee为例。记住注册的时候:

// 用户名

// 密码

// 邮箱

3.2.2 建立远程仓库

建立远程仓库

建立远程仓库

☐ 开源 (所有人可见) ②
☒ 私有 (仅仓库成员可见)
☐ 企业内部开源 (仅企业成员可见) ②

☒ 初始化仓库 (设置语言、.gitignore、开源许可证)

选择语言: Java | 添加 .gitignore: 请选择 .gitignore 模板 | 添加开源许可证: BSD-3-Clause | 许可证向导 ②

☒ 设置模板 (添加 Readme、Issue、Pull Request 模板文件)

☒ Readme 文件 | ☐ Issue 模板文件 ② | ☐ Pull Request 模板文件 ②

☒ 选择分支模型 (仓库创建后将根据所选模型创建分支)

单分支模型 (只创建 master 分支)

创建

3.2.3 clone

下载仓库内容

```
# 下载远程仓库的内容, 并且在本地创建一个和远程仓库名同名的文件夹
# 下载的时候, 路径在哪, 就直接下载到那个地方
git clone https://gitee.com/snow/code100th.git

# 下载到指定的文件夹中(完全不建议)
git clone https://gitee.com/snow/code100th.git dirName
```

注意: 如果是私有仓库, 会在clone的时候要求输入账号密码(此账号密码为你的git远程仓库上的账号密码) (下图以Linux为例, 要求在命令行输入账号密码)(而在Windows中账号密码会出现弹框提示你输入)

```
Username for 'https://gitee.com': snow-lee
Password for 'https://snow-lee@gitee.com':
```

注意: 正常来讲, git工具会自动记住账号和密码, 下次使用就不需要输入了(和远程仓库交互的时候), 如果密码输错, 则可能需要修改系统的记录凭证(windows中比较方便的操作). 或者清空缓存重置账号密码(Linux删除家目录下的 .git-credentials 文件). **建议还是不要输错了**

注意: (如果老是每次都和中央仓库交互都需要输入账号密码: windows: **git config --global credential.helper manager**) (Linux: **git config --global credential.helper store**)

(或者各个系统的通用配置在用户名路径下的.gitconfig中 配置: [credential] helper = store)

3.2.3 status

status状态命令:

这个命令可以帮助我们查看 `工作区` 和 `缓冲区` 和 `本地仓库` 中的变化。

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
   new file:   runlist.txt      缓冲区中的变化

Untracked files:
  (use "git add <file>..." to include in what will be committed)
   runlist2.txt               工作区中的变化
```

3.2.4 add

add: 这个命令可以帮助我们把工作区中的变化提交到缓冲区。

```
# 所有文件: 把所有文件的变化都从工作空间提交到缓冲区
git add .

# 文件的名称: 把指定的文件名文件在工作空间发生的变化提交到缓冲区
git add fileName

# 文件的类型: *通配符
git add *.java // 把所有以.java结尾的文件的变化, 从工作空间提交到缓冲区
```

3.2.5 commit

commit命令: 帮助把缓冲区变化内容提交到本地仓库。

```
# 提交
git commit -m "备注/注释"

## msg:msg信息一般要有统一的格式和意义(参照公司怎么写), 反正不建议乱写和简写(一定要注意)
## 例如: Author:ciggar Date:20220528 msg:xxx
```

注意:

```
// 这一步会产生一个文件的版本号
// 如果是第一次commit, 需要设置用户名和邮件地址(建议设置真实的)
// 只会把缓冲区中的变化提交到本地仓库, 不会把工作区中的变化提交到本地仓库
// commit的时候需要指定提交的信息, 也就是注释(要重视注释的书写)
```

设置方式一: 直接文件设置

```
// 去用户目录下(无论是Linux还是Windows都在家目录下), 设置 .gitconfig文件, 假如没有这个文件, 就创建一个
```

```
[user]
  email = 1111@qq.com
  name = snow
[credential]
  helper = store
```

设置方式二: 命令设置

```
git config --global user.email 222@qq.com
git config --global user.name xxx
```

设置完之后, 就可以提交了, 会产生一个版本信息

```
$ git commit -m "Auther:ciggar Date:20220526 msg:第一次提交"
[master 5a2db40] Auther:ciggar Date:20220526 msg:第一次提交
5 files changed, 3 insertions(+)
create mode 100644 Test.java
create mode 100644 Test3.xml
create mode 100644 User.java
create mode 100644 runlist.txt
create mode 100644 runlist2.txt
```

版本号: 前7位

3.2.9 log

log命令: 查看仓库中的所有的版本信息

```
// 列出版本的详细信息
git log
// 列出压缩格式的所有分支的提交树 (--oneline: 每个版本压缩成一行)(--all:所有分支)(--graph:以分支树的形式展示)
git log --oneline --all --graph
```

// 如果出现历史记录过长: Q回车退出

```
$ git log
commit c5e118a2e27fe4eaecfbd7aad411301a2644ec5c (HEAD -> master, origin/master, origin/HEAD)
Author: ciggar <291136733@qq.com>
Date: Thu May 26 15:06:32 2022 +0800
    Auther:ciggar Date:20220526 Msg:第二次提交

commit 5a2db4082e834cbb92e066180af7f617270f558f
Author: ciggar <291136733@qq.com>
Date: Thu May 26 14:56:33 2022 +0800
    Auther:ciggar Date:20220526 msg:第一次提交

commit dd8673a616b22229e890534f2122209072363641
Author: ciggar <291136733@qq.com>
Date: Thu May 26 03:49:48 2022 +0000
    Initial commit
```

第三个版本

第二个版本

第一个版本

3.3 分支问题

分支是在合作开发中一个非常重要的维护代码迭代/bug修复/协作可开发的重要设计结构,

需要注意的是在做分支操作之前: 确认工作空间和缓冲区干净状态, 以避免产生不必要的冲突

3.3.1 查看分支

查看所有分支:

```
git branch -a
```

3.3.2 创建分支

创建一个本地分支:

```
git branch 分支名
```

3.3.3 切换分支

从当前分支切换到另一个分支:

```
git checkout 分支名
```

创建并切换分支: (等价于先创建, 再切换到创建的这个分支上)

```
git checkout -b 分支名
```

3.3.4 删除分支

删除分支:

```
// 安全的删除: (确保这个分支的内容分支已经被合并到上游)
git branch -d 分支名
// 强制删除分支:
git branch -D 分支名
```

3.3.5 合并分支

合并一个其它分支的内容到当前分支

```
git merge 其它分支名
```

注意: 合并分支应该要向下游版本合并, 而非向上游合并

注意: Y型分支

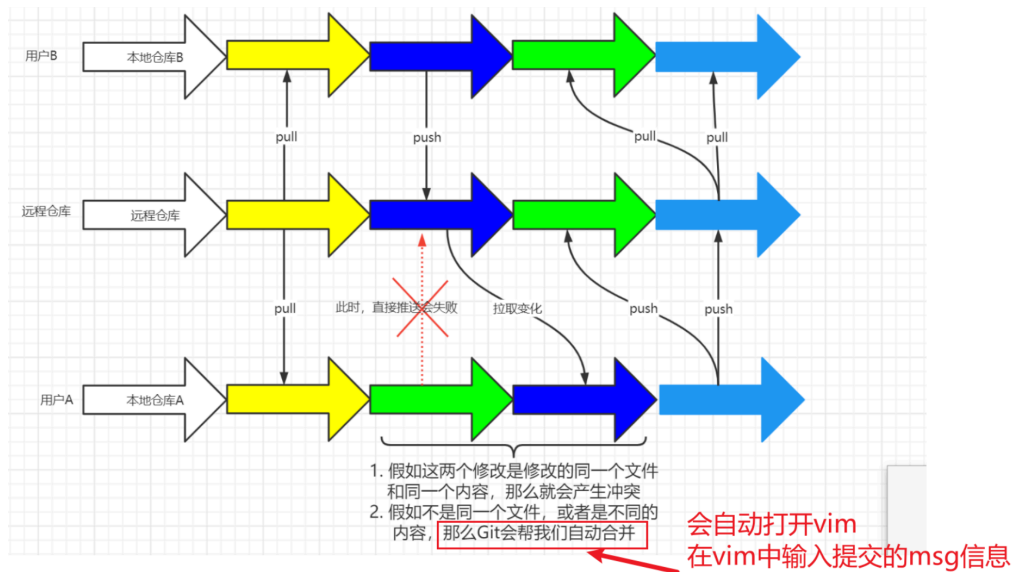
假设存在A和B两个Y型分支:

```
// 如果让在A上合并了B分支, 则A产生并演进到新的分支下游版本
// 如果想让B的版本进度也演进到刚才A产生的下游版本, 只需要把分支切回B, 并且对A进行merge,
则可以让AB都指向同一个版本
```

```
// 需要注意的是: 如果在合并过程中产生了无法自动merge的内容冲突, 需要手动处理冲突
// 当B分支被合并到A分支上, 如果不想继续使用B分支, B分支可被删除, B的删除不影响git记录保存B曾经产生的修改
```

3.4 冲突问题

冲突: 更多时候是指在一个Git项目管理追踪中, 用户在更新和提交的代码中, 对同一个文件发生了修改



1, 版本冲突: git会自动合并版本冲突, 产生一个合并版本, 最常见, 但是git会自动合并产生合并版本

2, 如果某个文件产生了冲突 (并且内容的变化本身是兼容的:eg创建两个同名无内容文件, 或者内容变化的本身无关性的:eg分别修改同一个文件1行和20行): 不常见, 会自动合并产生合并版本

3, 修改同一个文件,导致内容冲突: 需要手动合并(git对这种情况就无法自动合并产生合并版本了), 如果进行手动的冲突合并, 一定要记得需要重新add -> commit -> push

结论:

```
// 如果产生冲突, git能自动合并的冲突不用做任何处理, 合并之后别忘了push
// 需要处理的是git不能自动合并的冲突, 比如第三种情况: 重新先pull 再add commit 再push
```

注意: 冲突的约定俗成规则

```
// 1. 先push的人不处理冲突, 后push的人要处理冲突(谁遇到冲突, 谁解决)
// 2. 和组员一起开发的时候, 尽量不要开发同一个文件, 很容易产生冲突
// 3. 准备提交代码之前最好先pull一下
// 4. - 早上上班之后, 第一件事情, 拉取最新的代码 (pull)
//     - 晚上下班之前, 最后的一件事情, 把最新的本地代码推送上去 (push)
//     如果今天下班了, 代码没写完, 要不要push, 不要push
```

3.5 提交和更新

在合作开发中, 多个用户基于远程仓库的代码合并本质上是一种Y型分支的问题.

3.5.1 push

push命令: 可把本地仓库中的版本变化推送到远程仓库。

```
// 把本地"当前所在的分支"的"版本变化"推送到"跟踪/关联"的"远程分支"上 (但是易受git的设置的影响改变推送规则)
```

```
git push
```

```
// 把本地"当前所在的分支"的"版本变化"推送到"显示指明"的"远程分支"上 (不意味着可以乱推送)
```

`git push origin 分支名`

// 值得注意的是：如果远程分支创建了一个名为a的分支，本地自己也创建了一个a的分支，当本地处于a分支上时，无论使用`git push`还是 `git push origin a` 都可以让本地分支a自动和远程的a分支建立追踪/关联关系(根据分支名字相同的原因)(版本也要存在匹配：即本地为下游版本，远端为上游版本，否则要是存在冲突(比如虽然同时a分支，但是之前没有关联，并且非线性的上下游版本)要先建立关联：`pull origin a` 并且`merge`(有可能需要手动`merge`))，并且可以直接把本地的a的版本内容推送到远程分支上

// 如果本地创建了一个分支，而远程不具有这个分支，也可以通过命令的方式把它推送到远程上，让远程多一个新分支

// 提交分支：在创建分支后第一次`push`的时候需要建立本地分支和远程分支之间的联系

`git push --set-upstream origin 分支名`

// 因为本地分支的删除，并不会影响到远程的关联分支删除，所以要删除远程分支，除了远程仓库直接删除，也可以在本地通过命令删除

// 可以通过命令删除远程分支

`git push origin --delete 分支名`

注意1:

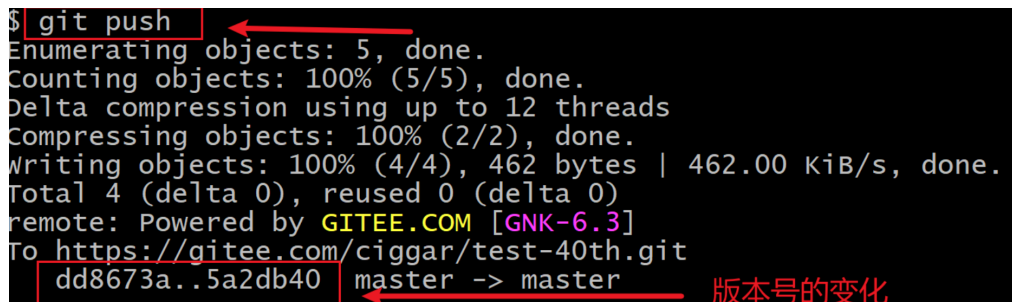
// 这一步在第一次操作的时候，需要去填写对应用户名和密码（如果在`clone`私有仓库则在`clone`已经填过了，或者之前某些时候使用过`git`已经填过账号密码，则忽略这个问题）

注意2:

// `push`的时候，能不能指定只对某个文件去`push`呢？ 不能，因为`push`是以版本为单位的

注意3:

// 只有当本地仓库中的版本领先于远程仓库的时候，才可以`push`



```
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 462 bytes | 462.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
remote: Powered by GITEE.COM [GNK-6.3]
To https://gitee.com/ciggar/test-40th.git
dd8673a..5a2db40 master -> master
```

版本号的变化

3.2.7 pull

`pull`命令: 拉取远程仓库中的所有的版本变化到本地。

// 获取远程仓库的更改(这个命令的本质：是将远程分支的最新状态下载到本地仓库，作为本地对远程版本的信息追踪记录)

`git fetch origin` 分支名

(`git fetch origin` 分支名:分支名 -> 从远程仓库中获取指定分支的最新内容到本地仓库对应的分支,不存在分支则创建)

// 把fetch下来的远程分支版本，合并到当前版本中(要以分支命名和版本做匹配/关联分支，不要乱merge) (注意斜杠，不是空格)

`git merge origin/分支名`

// 把远端的关联的分支fetch下来,并且merge到当前所在分支上

`git pull origin 分支名`

// 如果之前已经建立关联分支，等价于(`git pull origin 分支名`)

`git pull`

注意：

// 当本地仓库中的版本落后于远程仓库的时候，就要pull

// 落后就要pull (工作中一定要注意)

```
$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), done.
From https://gitee.com/ciggar/test-40th
   dd8673a..c5e118a master -> origin/master
Updating dd8673a..c5e118a
Fast-forward
 Test.java      | 0
 Test3.xml     | 0
 User.java     | 0
 c3p0-config.xml | 0
 runlist.txt   | 3 +++
 runlist2.txt  | 0
 6 files changed, 3 insertions(+)
```

版本号的变化

3.6 协作开发

3.6.1 邀请成员加入仓库

不管是 开源的仓库，还是私有的仓库，都是 只有仓库中的成员才能去修改仓库中的代码。

// 开源：所有人都可以访问到

// 私有：只有仓库指定的成员才能看到

The screenshot shows the Gitee repository page for 'snow-lee / code47'. The top navigation bar includes links for '代码' (Code), 'Issues', 'Pull Requests', 'Wiki', '统计' (Stats), '流水线' (CI/CD), '服务' (Services), and '管理' (Manage). A red arrow points to the '管理' button. Below the navigation bar, there is a license notice: '你当前开源项目尚未选择许可证 (LICENSE)，点此选择并创建开源许可'. The main content area shows a list of files and commits. The files listed are 'lib', 'maven-project', 'mybatis-project', 'mybatis-project2', 'mybatis-project3', and 'mybatis-project4'. The commits listed are '2023030611 da53c0b 2小时前', '2023022716', '2023030210', '2023030217', '2023030216', '2023030217', '2023030217', and '2023030310'. The right sidebar contains a '简介' (Introduction) section, a '发行版' (Releases) section, and a '贡献者' (Contributors) section.

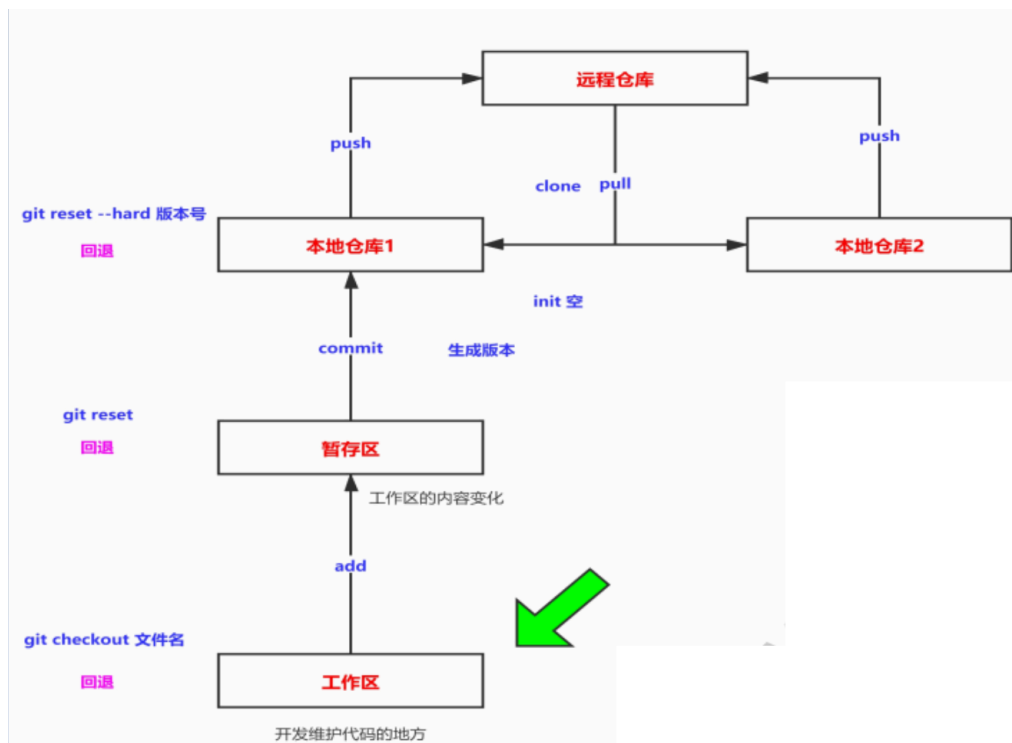
3.6.2 怎么在项目中组织分支

在公司中写代码, 一定要去问一下分支结构(git branch -a查看), 合并逻辑

组织GIT的分支体系有很多种模式: 怎么设计没有标准的定式. (举例...)

3.7 回退问题

慎重: 覆盖 (紧急情况: master生产环境紧急回退->clone->本地回退-> 修改代码 -> 覆盖 master_bug)



回退工作空间的变化: (参考git status提示)

// 回退某个被追踪文件在工作空间的修改 (未进行add和commit) (回退的内容, 是找不回来的, 要慎用)

`git checkout 文件名`

// (注意:git status提示): `git restore 文件名`

把缓冲区中的变化, 回退到工作空间:

`git reset 文件名`

// `git restore --staged 文件名`

回退本地版本

`git reset --hard 版本号`

永远不要手动回退远程仓库的代码

3.8 忽略文件

Git在做版本控制的时候，可以让我们忽略一些文件，不去追踪这个仓库中这些文件的变化。

// 可以在Git仓库的根目录下 添加一个.gitignore这个名字的文件，可以在这个文件中声明哪些文件不被git追踪版本信息

```
eg:
# 单个文件
xxx.txt
# 配置文件夹
.idea
# 配置文件的类型
*.iml
target/*.class

out
```

注意事项：

// 忽略文件最好是在创建仓库的时候，就应该先创建出来
// 一旦一个文件已经被追踪并且提交到远程仓库中去了，那么再在.gitignore 这个文件中去忽略它的变化，是无效的（要想起效果，要先删除这个文件，提交一个版本，再把这个文件加进来： 不建议,有点繁琐）

3.3 本地项目提交

场景

// 假设我本地已经有一些需要管理(但是从来没有通过Git管理)的代码，我想提交到Git仓库管理起来.一般有两种方式

方式一：

// 通过上述流程：

- 1, 手动创建远程仓库
- 2, 创建本地仓库(`git clone`): 产生本地仓库, 工作空间, 缓冲区
- 3, 把文件移到 工作空间 目录
- 4, 管理文件(`git add .`)
- 5, 产生版本(`git commit -m '注释'`)到本地仓库
- 6, 提交到远程仓库`git push`
- 7, 正常使用

方式二：

- 1, 在代码文件所属目录, 创建本地仓库(`git init`)(还创建了缓冲区 和 工作空间)
- 2, 管理文件(`git add .`)
- 3, 产生版本(`git commit -m '注释'`)到本地仓库
- 4, 创建远程仓库: 不要选择任何模板和初始化 (创建了一个全空的远程仓库)
- 5, 关联远程仓库: `git remote add origin https://gitee.com/snow-lee/lalala.git`
- 6, 提交到远程仓库: `git push -u origin "master"`
- 7, 正常使用

https://learngitbranching.js.org/?locale=zh_CN