1. Git
2. Git与SVN的对比
3. SVN

SVN是集中式版本控制系统，版本库是集中放在中央服务器上的，开发人员只能从中央服务器中下载最新的版本进行开发，开发完毕后，需要将自己开发的代码提交到中央服务器中。

集中式版本控制工具缺点：

服务器单点故障：当服务器出现故障时，所有的开发需要等待服务器的故障排除后才能继续进行。容错性很差

1. Git

Git是分布式版本控制系统（Distributed Version Control System，DVCS），分为两种类型的仓库：本地仓库和远程仓库。本地仓库是开发人员自己电脑上的Git仓库，远程仓库是远程服务器上的Git仓库。

当开发人员开发完毕后，可以通过Commit命名将代码提交到本地仓库上；若想将代码提交到远程仓库中，使用Push推送命令推送到远程仓库中；若想从远程仓库中下载代码到本地仓库，使用Pull拉取命令；若想直接将远程仓库复制到本地，就使用Clone克隆命令。

1. Git工作流程

Git的工作流程如下所示：

1. 从远程仓库中通过Clone克隆代码到本地仓库中。
2. 开发人员从本地仓库中通过checkout获取代码后进行代码修改。
3. 开发人员修改完代码后，想要将代码提交到本地仓库中，首先将代码Add提交到暂存区，之后暂存区会将代码Commit到本地仓库中。
4. 提交到本地仓库后，本地仓库保存着修改的历史版本。
5. 当需要将代码上传到远程仓库时，可以通过Push从本地仓库推送代码到远程仓库中。
6. 若想直接从远程仓库中获取代码，则可以通过Pull拉取代码到工作区中，不经过本地仓库。
7. Git代码托管服务
8. 常用的Git代码托管服务

通常我们是借助互联网上提供的一些代码托管服务来搭建Git远程仓库，其中比较常见的有GitHub，码云，GitLab等。

GitHub是一个面向开源即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，因此命名为GitHub；码云是国内的一个代码托管平台；GitLab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。

1. 使用码云进行代码托管
2. 创建新仓库

登录码云，创建新的仓库，每个Git远程仓库都会对应一个网络地址，可以店家克隆/下载按钮弹出窗口，窗口中就是Git远程仓库对应的网络地址。

1. 邀请其他用户成为仓库成员

项目的开发是由多人共同完成的，为了使多个参与者都有权限操作远程仓库，需要邀请其他项目参与者成为当前仓库的成员。

1. Git常用命令
2. 环境配置

安装完Git后首先要设置用户名和email名称，每次Git提交时都会使用该用户名和email地址进行验证。

1. 设置用户信息

git config –global user.name “mrwxx”

git config –global user.email [hello@wxx.cn](mailto:hello@wxx.cn)

git config 是用于Git配置的命令， --global代表着这是全局的， user.name代表着用户名称，user.email代表着email地址。

1. 查看配置信息

以列表形式显示所有的Git配置：

git config –list

显示单个Git配置：

git config user.name

当我们手动设置完Git的配置后，这些配置信息会保存在本地用户的目录下，如本地的C:\Users\Mr.wmxx下面，有一个.gitconfig文件，文件中存储着刚刚配置的信息：

[user]

name = mrwxx

email = hello@wxx.cn

1. 获取Git仓库

获取Git仓库有两种方式，一种是在本地初始化一个Git仓库，另一种是从远程仓库中克隆一个。

1. 本地初始化Git仓库

在本地初始化一个Git仓库，就是将本地的一个文件夹作为Git仓库，在某个文件夹下右键进入Git Bash，使用git init命令初始化一个Git仓库。这样，该文件夹下会生成一个.git隐藏文件夹。

1. 克隆远程仓库

克隆远程仓库到本地中，在本地的文件夹下进入Git Bash，使用git clone 命令，后面加 远程仓库的地址即可。

git clone https://gitee.com/mrxx/repos.git

1. 工作目录，暂存区以及版本库
2. 版本库

前面所说的.git文件夹就是版本库，版本库中存储了很多的配置信息，日志信息和文件版本信息。

1. 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。

1. 暂存区

.git文件夹中的index文件就是暂存区，暂存区用于临时保存修改的文件。

开发人员在工作区进行开发，开发修改后的代码通过 git add命令传到暂存区，再通过git commit 命令提交到版本库中。

1. 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，这些文件的状态会随着Git命令的执行而发生变化。

1. untracked

未跟踪状态，说明它未被纳入版本控制中。

1. tracked

已跟踪状态，说明它已经被Git纳入版本控制了，且已跟踪状态又分为三种状态：

1. Unmodified

未修改状态，文件没有被修改过。

1. Modified

已修改状态，文件已经被修改过了。

1. Staged

已暂存状态，文件已经放入暂存区中、

1. 本地仓库操作
2. git status

在工作区中使用git status查看工作区中文件的状态。

使用 git status -s使得输出结果更加简洁

1. git add

git add命令将未跟踪的文件加入暂存区中，那么该文件的状态就会变为已跟踪中的已暂存状态。

git add hello.txt

1. git reset

git reset命令将暂存区中的文件取消暂存，那么该文件的状态就会发生变化，变为未跟踪。

git reset hello.txt

1. git commit

要想将代码提交到本地仓库中，必须通过暂存区，即先将代码通过git add命令存到暂存区中，再通过git commit命令将代码提交到本地仓库中。

git commit -m “init hello.txt”

-m代表着日志，每次上传代码到本地仓库中，必须要写日志信息，如果不加-m，它会弹出一个编辑器让我们进行编写。

1. git rm

git rm命令将文件从工作区中删除，但是，本地仓库中的该文件并没有被删除，要想删除，需要提交到本地仓库。提交之前我们需要将该文件添加到暂存区中，有趣的是，使用git rm命令删除该文件后，该文件就自动被添加到了暂存区中，因此我们可以直接使用git commit命令提交就可以了，本地仓库中的该文件就被删除了。

1. .ignore文件

工作区中有一些文件是不需要纳入Git的管理的，也不希望他们总是出现在未跟踪文件列表中，因此，我们可以在工作区中创建一个名为 .gitignore的文件，在该文件中列出需要忽略的文件模式，如下所示：

\*.a 代表着忽略所有的.a格式文件

!lib.a 代表着除了lib.a之外

/TODO 代表着忽略TODO文件

build/ 代表着忽略build文件夹下的所有文件

doc/\*.txt 代表着忽略doc文件夹下的所有txt文件

doc/\*\*/\*.txt 代表和忽略doc目录下的所有子目录的txt文件

1. git log

通过git log命令查看日志记录，按Enter键向下浏览记录，按q键 退出到命令行。

1. 远程仓库操作
2. git remote

当一个本地的工作空间和远程仓库建立了联系后，通过git remote命令查看已经配置的远程仓库服务器，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，那么至少有一个origin，这是Git克隆的远程仓库服务器的默认名字。

想要更加详细的信息，可以在git remote命令后面加上 -v，可以显示远程仓库服务器的地址：

git remote -v

若是想查看某一个远程仓库服务器的信息，则使用git remote show 命令后面加上该仓库的简写即可：

git remote show origin

1. git remote add

通过git remote add命令为本地添加远程仓库，命令后面首先加上 远程仓库的简写，然后是 远程仓库的地址：

git remote add origin <https://gitee.com/mtwxx/repos.git>

我们可以为本地添加多个远程仓库。

1. git clone

git clone命令负责从远程仓库中进行拷贝，Git克隆的是该Git仓库服务器上几乎所有数据（包括日志信息，历史记录等）。当执行git clone 命令时，默认配置下远程Git仓库中的每一个文件的每一个版本都会被拉取下来。

git clone <https://gitee.com/mrwxx/repo.git>

1. git remote rm

通过git remote rm 命令移除本地连接的一个远程仓库，命令后面跟远程仓库的简称。这里只是在本地中移除远程仓库的记录，不会影响到真正的远程仓库。

git remote rm origin

1. git fetch

git fetch是从远程仓库获取最新版本到本地仓库中，不会自动地merge，不会自动合并到本地仓库中，即不会直接显示在本地仓库中。

git fetch origin master

origin是远程仓库的简称，master是远程仓库的分支。

我们还要通过 git merge 来合并代码到工作区中，后面接远程仓库的分支，如：

git merge origin/master

1. git pull

git pull也是从远程仓库获取最新版本到本地仓库，但是会自动merge，合并代码。

git pull origin master

注意：如果当前的本地仓库是本地创建的仓库，且仓库中存在文件，此时再从远程仓库中拉取文件时会报错（fatal:refusing to merge unrelated histories），这是因为原有的文件与拉取的文件无法合并。可以通过在git pull命令后添加参数 –allow-unrelated-histories来解决

git pull origin master –allowed-histories-histories

1. git push

通过git push命令将本地的代码推送到远程仓库中，后面接远程仓库的简写，以及分支名：

git push origin master

这个操作一般都是在我们对之前下载的代码进行修改之后，修改后的代码是没有进入本地仓库的，因此我们首先要将修改后的代码提交到本地仓库，而提交到本地仓库首先要存到暂存区，使用git add 添加到暂存区或者 git commit -a命令直接提交到本地仓库中。在这之后，就可以执行git push命令推送新的代码到远程仓库了。

1. Git分支

所有的版本控制系统都有分支，使用分支意味着可以把工作从开发主线上分离开来，以免影响开发主线。Git的master分支不是特殊的，之所以每个仓库都有master分支，是因为git init会默认创建它。

1. 查看分支

git branch 列出本地所有分支

git branch -r 列出所有远程分支

git branch -a 列出本地和远程的所有分支

1. 创建分支

git branch a 创建a分支

1. 切换分支

我们通过git checkout来切换分支，命令行中会显示目前使用的分支：

git checkout a 切换到a分支

1. 推送到远程仓库分支

将本地的分支推送到远程仓库的分支中，如推送到origin仓库的b1分支中：

git push origin b1

1. 合并分支

通过git merge命令合并两个分支，如我们要将b3分支合并到master分支上，那么，我们需要切换到master分支中，使用git merge b1命令合并分支。如果在两个不同的分支中，对同一个文件同一个部分进行了不同的修改，Git就合并它们，会提示文件冲突。此时我们需要打开冲突的文件并修复冲突内容，最后执行git add命令将代码存到暂存区中。

1. 删除分支

若要删除本地的分支，可以通过git branch -d 后面接分支的名称来删除该分支。若在该分支中已经进行了修改，且没有推送到远程仓库中，那么使用上面的命令并不会删除分支，因为直接删除会有丢失修改数据的风险。若坚持要直接删除，将-d参数改为-D参数即可删除。

若要删除远程仓库中的分支，可以使用git push origin -d branchName，origin是远程仓库的简称，branchName是分支的名称。

1. Git标签

Git可以给历史中的某一个提交打上标签，以示重要，人们会使用这个功能来标记发布节点。标签指的是某个分支某个特定时间点的状态，通过标签，可以很方便地切换到标记时的状态。

1. git tag

列出已有的标签

1. git show [tag]

查看tag信息

1. git tag [tagName]

新建一个tag标签

1. git push [remote] [tag]

将标签推送至远程仓库，提交指定tag

如： git push origin v0.1

1. git checkout -b [branch] [tag]

检出标签，新建一个分支，将该分支指向某个tag

1. git tag -d [tag]

删除本地标签，

1. git push [remote] :refs/tags/[tag]

删除远程仓库的tag，:ref/tags是固定写法：

git push origin :refs/tags/[tag]