1. Git是分布式版本控制软件：
   1. 集中式：代表：svn。所有客户端提交文件快照到一台中央服务器，存在安全隐患；存放版本之间的差异，磁盘占用小，回滚速度慢
   2. 分布式：客户端在提交最新版本的文件快照时，也会把代码仓库完整地镜像下来；管理项目时，存放项目的完整快照（索引）磁盘占用大（但- git团队对代码进行了压缩，最终空间仅稍大于svn）；版本控制在本地进行，可以离线使用
2. 配置用户信息：
   1. git config --global user.name “用户名” git config --global user.email xx@xx.xx git config --list 检查已有配置信息
   2. --system表示对系统中所有用户适用 --global表示系统中的当前用户 若不写表示仅对当前项目有效
   3. git config --global --unset user.name/email 删除配置
3. 忽略文件：仓库根目录创建 .gitignore 文件，在该文件中写上要忽略文件或目录的路径
   1. #开头表示注释 \*表示任意个字符 ？表示任意单个字符 \*\*表示匹配多级目录
   2. 开头加/表示仅根目录中的该文件 结尾加/表示目标是目录 开头加！表示忽略除他之外的文件
   3. .DS\_Store node\_modules/ /dist/ npm\_debug.log\* yarn\_debug.log\* yarn\_error.log\*
   4. .idea .vscode \*.suo \*.ntvs\* \*.njsproj \*.sln
4. .git目录：
   1. hooks目录 包含客户端或服务端的钩子脚本 info目录 包含一个全局性排除文件
   2. logs目录 保存日志信息（其中的HEAD文件保存历史指向）objects目录 存储所有数据内容
   3. refs目录 存储分支及其提交对象的键值 config 包含项目特有的配置信息
   4. description 显示对仓库的描述信息 HEAD 指示目前被检出的分支
   5. index 保存暂存区信息
5. 区域：工作区、暂存区、版本库（数据库）
6. 对象：
   1. Git对象：blob类型，代表文件的各版本。核心部分是简单的键值对数据库。数据在.git/objects目录中。仅对本地数据库操作，不涉及暂存区
      1. 可以向数据库插入任意类型的内容，它会返回一个键值，通过该键值可在任意时刻再次检索该内容
      2. 向数据库内写入内容，并返回对应键值：返回值为SHA-1哈希值
         1. echo 'test content' | git hash-object -w --stdin 或 git hash-object -w 文件路径
            1. -w用于指示命令存储数据对象，若不加，则不存储仅返回键值；
            2. --stdin（standard input）指示命令从标准输入读取内容；若不指定此选项，则须在命令尾部给出待存储文件的路径
      3. 查看数据的键值：find .git/objects -type f 数据存在该目录，键值前两个字符创建为子目录，剩下的键值作为文件名存于子目录
      4. 拉取文件数据：git cat-file -p 键值 -p表示自动判断内容的类型，并自动选择合适的方式显示
      5. 查看文件类型：git cat-file -t 键值 -t可获取内部存储的任何数据的类型
      6. 简单版本控制：存两次同名文件，返回两个键值，对应文件存放两版本的完整数据，不存增量。存在问题：键值不好记；没有保存文件名
   2. 树对象：tree类型，代表项目的各版本，可保存文件名。树对象相当于项目的快照
      1. 创建：
         1. 添加Git对象到暂存区：git update-index --add --cacheinfo 文件模式 \键值 文件名 当文件名重名时，覆盖
            1. 文件模式：100644 普通文件 100755 可执行文件 120000 符号链接
            2. --add 若该文件（以文件名为标识）之前不在暂存区，需要该选项 --cacheinfo 添加数据库中的文件需要该项
         2. 添加树对象到暂存区：git read-tree --prefix=bak 键值
         3. 生成树对象：git write-tree 根据当前暂存区生成树对象（不清空暂存区），树对象存入数据库，返回键值
      2. 查看当前暂存区：git ls-files -s查看树对象：git cat-file -p master^{tree}（或是树对象的 键值） 使用master^{tree}的写法表示master分支上最新提交的树对象
   3. 提交对象：commit类型，相当于树对象的包裹，包含修改者信息及备注等
      1. 创建：echo '备注信息' | git commit-tree 树对象的键值 返回提交对象的键值，将提交对象存入数据库
         1. 提交对象引用上一个提交对象：echo '备注信息' | git commit-tree 树对象的键值 -p 上一个提交对象的键值
      2. 查看：git cat-file -p 提交对象的键值
7. 配别名：git config --global alias.别名 原命令 使用 git 别名 时，会解析为 git 原命令
   1. 基本别名：
      1. HEAD HEAD所指分支所指的提交对象的键值 HEAD~ HEAD所指分支所指的上一个提交对象的键值
8. 底层命令：在Git Bush中使用，以$ 开头，以下省略
   1. 基础的linux命令：
      1. clear 清除屏幕 echo ‘信息’ / echo ‘信息’ > xx.txt 向控制台/txt文件输出信息
      2. pwd 显示当前目录 ls 显示当前目录下的文件
      3. ll 显示当前目录下的子文件、子目录 cd e:code/01 切换目录
      4. find 目录名 显示对应目录下的子孙文件、子孙目录 find 目录名 -type f 将对应目录下的文件显示
      5. mkdir 文件夹名 新建文件夹 rm -r 文件夹名 删除文件夹
      6. touch 文件名 新建文件 rm 文件名 删除文件
      7. mv 源文件 新文件 重命名 cat 文件url 查看对应文件的内容
      8. vim 文件url 按i进插入（编辑）模式；按 esc 键退出编辑模式；在非编辑模式下 :q! 强制退出 :wq 保存退出 :set nu 设置行号
9. 本地操作：
   1. git init 在当前目录搭建仓库
   2. git add 文件名 对所有选中文件生成git对象，加到数据库（版本库），再加到暂存区
      1. 文件名：
         1. 完整文件名 可以写多个 目录名/ 目录内所有文件
         2. 目录名/\*.html 目录内所有html文件
         3. \*hhh 文件名以hhh结尾的所有文件 hello\* 文件名以hello开头的所有文件
      2. git add -u 被修改和被删除的文件，不包括新文件 git add . 新文件和被修改的文件，不包括被删除的文件
      3. git add -A 将所有变化加到暂存区
   3. git commit -m “注释” 生成树对象，添加注释并生成提交对象，存入版本库（不清空暂存区）
      1. git commit 打开vim编辑器来获取注释内容 git commit -a 提交所有已跟踪文件，跳过git add
   4. git status 查看文件状态。
      1. 不在上次快照且不再当前暂存区的文件为未跟踪状态；其余为已跟踪状态
      2. 已跟踪状态有：已提交、已修改（Changes not staged for commit）、已暂存（Changes to be committed）
      3. 直接删除文件也是一种修改，需要add；直接重命名文件相当于删除了原文件（已修改）并新建新文件（未跟踪）
   5. git diff 查看当前哪些修改没有暂存
   6. git diff --cached或--staged(1.6.1 以上) 查看哪些修改已暂存未提交
   7. git rm 文件名 删除文件，并添加修改到暂存区。相当于执行了（rm 原文件、git add 原文件）
   8. git mv 原文件 新文件 重命名文件，并添加修改到暂存区。相当于执行了（mv 原文件 新文件、git add 原文件和新文件）
   9. git log 查看当前分支的提交历史，可添加--pretty=oneline或--oneline改变样式
      1. git reflog 查看所有分支的所有操作的历史
   10. git ls-files -s 查看暂存区
10. 分支操作：分支为一个指向提交对象的指针
    1. git branch 分支名 创建一个新分支（不自动切换到新分支），该分支指向当前提交对象
       1. git branch -d 分支名 删除分支（不能删除主分支） git branch -v 查看每个分支的最后一次提交
       2. git branch 分支名 键值 新建一个分支，指向对应提交对象 git branch --merged 查看哪些分支已经合并到当前分支
       3. git branch --no-merged 查看哪些分支没有合并到当前分支 git branch -u 分支名 当前分支跟踪指定分支
       4. git branch -vv 查看设置的所有跟踪分支
    2. git checkout 分支名 切换分支（会改变HEAD、暂存区、工作目录中的文件，注意提交当前分支或检查文件状态）
       1. 切换分支时，会还原工作区和暂存区，但携带未追踪文件和未提交的未追踪文件的暂存，可能污染分支
       2. 上述两者不属于任何分支，可被携带，一旦被提交则归属于所提交的分支，再切换分支时不被携带
       3. git checkout -b 分支名 创建分支并切换到该分支
    3. git log --oneline --decorate --graph --all 查看全部分支记录
    4. git merge 分支名 将当前分支合并到指定分支
       1. 快进合并：指定分支在当前分支上游时，会直接将当前分支提到指定分支处（简单改变指针）
       2. 典型合并：指定分支与当前分支不在一个主线时，会将冲突文件合并，需要修改冲突文件后再提交一次
    5. git stash 将未提交的所有修改保存到一个栈上，然后将工作区和暂存区恢复到上次提交的状态
       1. git stash list 查看存储 git stash apply 存储名 应用指定存储，无存储名默认栈顶
       2. git stash pop 存储名 应用存储后移除该存储，无存储名默认栈顶 git stash drop 存储名 移除指定存储，不指定存储名则默认栈顶
11. 撤销与重置：
    1. git checkout -- 文件名 撤回该文件在工作区中的修改
       1. 若该文件不在暂存区，撤回到和版本库一样的状态；否则，撤回到添加到暂存区后的状态
    2. git reset [HEAD] 文件名 从版本库中拉取指定文件到暂存区（可配合 git checkout -- 文件名 拉取到工作区）
       1. git reset --soft HEAD~ 将当前分支的指向从当前提交对象退回到上一个提交对象
       2. git reset [--mixed] HEAD~ 分支指向退回到上一个提交对象，并还原暂存区
       3. git reset --hard HEAD~ 分支指向退回到上一个提交对象，还原暂存区、还原工作区并携带未追踪文件
       4. 若在上述命令最后加上文件名，则跳过改变分支指向的步骤
       5. reset相对checkout：checkout在切换时有检查（较安全），而reset没有
    3. git commit --amend 将暂存区中的文件连同新注释生成一个新的提交对象，替换之前的提交对象
       1. 相当于git reset --soft HEAD~、git commit
12. tag：给历史中的某次提交打上标签，可用于标记发布版本
    1. git tag 列出所有标签
       1. git tag -l ‘v1.8.5\*’ 正则匹配，列出匹配的标签 git tag 标签名 [键值] 在当前/指定提交对象上创建轻量标签
       2. git tag -d 标签名 删除指定轻量标签 git checkout 标签名 跳转到指定标签所在提交对象（然后创建分支）
    2. git push [仓库名] --tags/指定tag名 推送所有/指定标签到远程仓库
13. 远程仓库：有本地分支、远程分支（位于远程仓库）和远程跟踪分支（位于本地，在网络通信时更新，名为：仓库名/分支名）
    1. 跟踪分支：当分支跟踪了指定的远程跟踪分支时，在该分支上可以直接git pull、git push不需要其他参数
    2. git remote add 仓库名 连接地址 添加远程仓库，并取别名
       1. git remote -v 查看仓库状态 git remote show [仓库名] 查看指定仓库状态
       2. git remote rename 原仓库名 新仓库名 重命名仓库 git remote rm [仓库名] 移除远程仓库
    3. git clone 地址 仓库克隆（不需要git init）
       1. 克隆仓库时，会自动创建master分支，跟踪 仓库名/master
    4. git push 仓库名称 分支 向新仓库中推送分支
       1. git push -u 仓库名称 分支 仅第一次推送时使用，该分支跟踪远程分支
       2. git push 仓库名称 本地分支名：远程分支名 若不希望远程分支名与本地分支名同名时，可用该命令
    5. git push 仓库名称 分支 -f 强制推送
    6. git pull 仓库名称 文件下拉到本地仓库（相当于执行了git fetch 和 git merge）
    7. git fetch 仓库名 将远程仓库的最新内容拉到本地
       1. git checkout -b 分支名 远程跟踪分支名 在指定远程跟踪分支处创建新分支，并跟踪远程跟踪分支
       2. git checkout --track 远程跟踪分支名 上命令的简写形式，生成的分支名为远程分支名
    8. git merge 远程跟踪分支名 将当前分支合并到远程跟踪分支上
    9. git push 仓库名 --delete 远程分支名 删除远程分支
    10. git remote prune 仓库名 --dry-run 列出远程已被删除的无用远程跟踪分支
    11. git remote prune 仓库名 清除上述命令列出的远程跟踪分支
14. 创建合并请求（pull request）：对项目进行派生（Fork），GitHub将在你的空间中创建一个项目副本，你对其具有推送权限。可通过pull request使- 改动进入源版本库
    1. 基本流程：将副本克隆到本地，新建分支并修改，推送改动到gibhub中的副本，创建合并请求，讨论，项目拥有者合并或关闭请求
    2. 注意：若有新的pull request而之前派生过该项目，且本地代码不是最新的，需要拉取最新代码：
       1. 添加源仓库，fetch最新代码，merge到对应远程跟踪分支，push到副本仓库
15. SSH：HTTPS协议在提交时需要验证登陆信息，SSH仅配置一次后就不需验证
    1. 仓库的ssh连接地址：git@decoy【别名，默认为github.com】:DECOYYYYYYYYYY【github用户名】/Counter.git【远程仓库名】
    2. 创建公私钥：
       1. ssh-keygen –t rsa –C邮箱地址 生成公私钥
          1. 第二步填写私钥密码（可直接enter不设置），创建完成后在指定目录中打开pub文件，在github中添加公钥
       2. ssh -T git@别名 测试公私钥是否已经配对
    3. 多个公私钥管理：
       1. 在C:\Users\Administrator\.ssh创建文件config，对于每个公私钥添加以下信息：
          1. Host decoy【别名】
          2. HostName github.com【服务器地址】
          3. User DECOYYYYYYYYYY【github用户名】
          4. IdentityFile ~/.ssh/id\_rsa\_decoy【私钥地址】