**Git** 是一种**分布式**版本控制系统（Distributed Version Control System，DVCS），用于追踪代码的改动。它帮助开发者管理代码、追踪变化、协作开发，并确保代码的质量和安全性。

在Git中，有四个主要的工作区域

1. **工作区**：这是你日常进行项目开发的地方，也就是你的项目文件夹。在这里，你可以创建、修改和删除文件。工作区中的文件并不是Git直接追踪的对象，而是Git通过其他机制来管理这些文件的变化。
2. **暂存区**：暂存区是一个中间区域，用于临时存放你即将提交的更改。当你使用git add命令将文件添加到暂存区时，Git会将这些文件的变化信息记录下来，但并不立即提交到版本库。暂存区允许你在提交之前对更改进行审查和整理。
3. **版本库**：版本库是Git用于安全存放所有版本数据的地方。它包含了一个项目的完整历史记录，从最初的提交开始，一直到最新的提交。在版本库中，每个提交都被赋予一个唯一的哈希值，并且可以通过这个哈希值来追踪和检索特定的版本。版本库中的数据是以对象的形式存储的，包括提交对象、树对象、blob对象等。
4. **远程仓库**：远程仓库是托管代码的服务器，通常用于与其他开发者共享和协作代码。你可以将本地仓库的更改推送到远程仓库，也可以从远程仓库拉取其他人的更改。远程仓库可以看作是一个中央化的存储和协作平台，它使得团队成员能够轻松地共享和同步代码。

以下是Git中文件的主要状态：

1. Untracked（未跟踪）：当一个文件在项目的文件夹中存在，但尚未被Git加入到版本控制中，该文件的状态就是Untracked。这通常意味着Git在之前的提交中没有记录这个文件。
2. Unmodified（未修改）：文件已经被Git跟踪，并且自上次提交以来没有发生过变化。这意味着文件的内容与最后一次提交到仓库中的版本一致。
3. Modified（已修改）：文件已经被修改，但尚未提交到Git仓库。这意味着文件的内容已经发生了变化，但还没有被Git记录为新的版本。
4. Staged（已暂存）：文件已经被修改，并且这些修改已经被Git记录到暂存区（也称为索引区）。这通常发生在执行git add命令之后，表示这些修改将在下一次提交时被包含进去。

添加追踪命令：

git add

提交命令：

Git commit -m “提交信息”

版本回退命令：git reset

常用的参数：

* --hard：此参数会将工作区、暂存区的内容都回退到指定的commit，也就是说，自该commit之后的所有更改都会被丢弃。这是一个非常强硬的操作，一旦执行，那些更改将无法恢复（除非你有备份）。
* --soft：此参数会将HEAD指针移动到指定的commit，但是工作区和暂存区的内容不会发生变化。这意味着你可以撤销commit，但是自该commit之后的所有更改仍然保留在工作区中，可以通过创建新的commit来提交这些更改。
* --mixed（或不加任何参数）：这是git reset的默认行为。此参数会将暂存区的内容回退到指定的commit，但是工作区的内容不会发生变化。这意味着你可以撤销commit，并且自该commit之后的所有更改都保留在工作区中，但是需要通过git add命令将这些更改重新添加到暂存区，然后才能提交。

以下是一些编写.gitignore文件的基本规则：

1. **“/”开头表示目录**：如果您想要忽略整个目录及其内容，可以在模式前加上斜杠。例如，/logs/将忽略名为logs的目录及其所有内容。
2. **“\*”通配多个字符**：您可以使用星号来匹配任意字符序列。例如，\*.log将匹配所有以.log结尾的文件。
3. **“?”通配单个字符**：如果您想要匹配单个任意字符，可以使用问号。例如，test?.txt将匹配test1.txt、test2.txt等，但不会匹配testabc.txt。
4. **“[]”包含单个字符的匹配列表**：方括号允许您指定字符集合，匹配方括号中的任意单个字符。例如，[0-9].txt将匹配0.txt到9.txt的所有文件。
5. **“!”表示不忽略（跟踪）匹配到的文件或目录**：如果您想要忽略除了某些特定文件或目录之外的所有内容，可以在模式前加上“!”。例如，\*将忽略所有文件，但!.gitignore将确保.gitignore文件自身不被忽略。
6. **注释 “#”**：在.gitignore文件中，以“#”开头的行被视为注释，不会被Git处理。您可以使用注释来解释或标记某些规则。
7. **规则匹配是从上到下进行的**：Git按照.gitignore文件中的规则从上到下进行匹配，一旦找到匹配的模式，就会停止搜索更多规则。因此，更具体的规则应该放在文件的上方。

HTML中常忽略以下文件;

1. **日志文件**：如.log文件，这些文件通常用于记录应用程序或系统的运行日志，并且可能会频繁更改。由于它们包含运行时数据，因此通常不需要版本控制。
2. **编译生成的文件**：如.o、.obj、.pyc等编译或解释后生成的文件。这些文件是源代码的编译产物，可以通过源代码重新生成，因此不需要版本控制。
3. **依赖管理工具的文件**：如node\_modules目录（Node.js项目中的依赖项）或venv目录（Python虚拟环境）。这些目录通常包含大量的第三方库和依赖，它们的版本可能会随着项目的开发而变化，因此通常会被忽略。
4. **配置文件**：某些配置文件可能包含敏感信息（如数据库密码、API密钥等），或者它们是根据环境特定的，因此不应该被提交到公共仓库。
5. **临时文件和缓存**：如.DS\_Store（Mac OS X的临时文件）或.tmp文件。这些文件通常是临时生成的，不需要版本控制。
6. **备份文件**：如.bak或~（波浪号）结尾的文件，这些通常是编辑器的备份文件。
7. **测试报告和覆盖率文件**：这些文件通常是测试工具生成的，并且会随着测试的执行而变化，因此不需要版本控制。

仓库克隆的命令：

git clone

分支操作的基本命令

查看全部分支：git branch

新建分支：git branch <name>

切换分支：git checkout <branchname>

**创建并切换到新分支：git checkout -b <branchname>**

**合并分支：git merge <branchname>**

**删除分支：git branch -d <branchname>**

**推送分支到远程仓库：git push origin <branchname>**

版本回退主要有两种方式：git reset和git revert。

git reset

是一种比较暴力的版本回退方式，它可以将代码库中的某个版本回退到之前的某个版本。首先，你需要查看版本号。使用git log命令来查找历史记录，或者使用git reflog命令来查找合并操作的commit ID，找到需要回滚的目标版本号。

然后，使用git reset --hard 版本名`命令将版本回退。此命令会将HEAD指针、当前分支指针和索引区都指向指定的commit ID，并重置工作区为该版本。最后，如果你需要将更改推送到远程仓库，由于git reset命令会更改本地分支的历史记录，你需要使用git push -f命令来强制推送更改。

git revert

是用于“反做”某一个版本，以达到撤销该版本的修改的目的。这种方式会保留目标版本后面的版本历史。你需要先查看版本号，找到需要回滚的目标版本号。然后，使用git revert -n 版本号命令来反做该版本。这里可能会出现冲突，需要手动解决冲突并提交。提交后，使用`git push`命令将更改推送到远程仓库。