

# Git 分布式版本控制工具

## 1、概述

---

### 1.1 Git应用场景

---

- **版本控制**：Git被广泛用于软件开发项目中进行版本控制，允许团队协作开发，并追踪代码的变化历史。
- **分支管理**：Git提供了强大的分支管理功能，使得团队可以并行开发不同的功能或修复不同的bug，然后将它们合并到主代码库中。
- **备份与恢复**：通过Git，可以轻松地备份项目的整个历史记录，并在需要时进行恢复。
- **同步与共享**：Git通过远程仓库的方式，使得团队成员可以方便地同步和共享代码。

### 1.2 版本控制的方式

---

- **集中版本控制工具**
  - 简介：集中版本控制工具是指所有的文件版本都存储在中央服务器上，团队成员通过与中央服务器进行交互来管理文件版本。
  - 举例：SVN (Subversion) 是一种流行的集中版本控制工具，它将所有的文件版本都存储在中央仓库中，并通过客户端与中央仓库进行交互。
- **分布式版本控制工具**
  - 简介：分布式版本控制工具是指每个开发者都拥有一份完整的代码仓库，并可以在本地进行版本控制操作，不需要与中央服务器直接交互。
  - 举例：Git是一种流行的分布式版本控制工具，每个开发者都可以在本地进行代码提交、分支管理等操作，然后再与其他开发者进行同步。

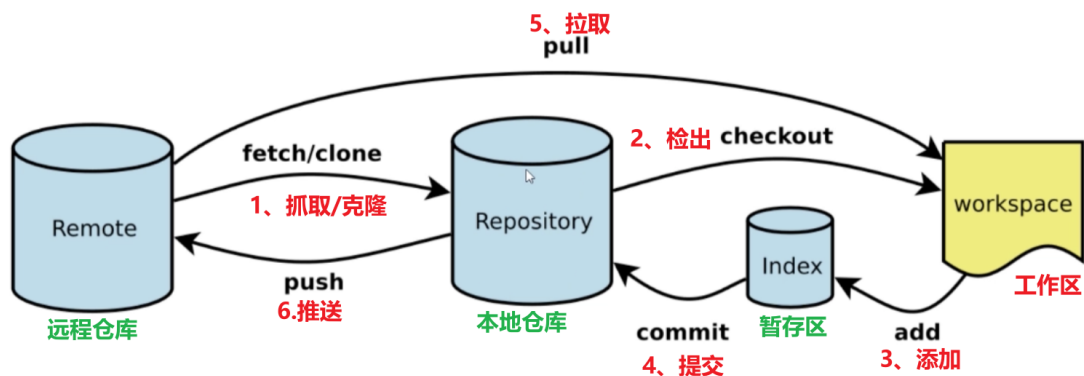
### • 1.3 Git的特点

---

- **本地仓库**：每个开发者都有一份完整的本地代码仓库，无需网络连接即可进行版本控制操作，提高了灵活性和效率。
- **共享版本库**：团队成员可以通过远程仓库轻松共享代码，实现协作开发，方便代码同步和团队合作，提高效率。
- **分支管理**：强大的分支管理功能允许并行开发不同功能，保持代码稳定性，便于实验性修改而不影响主要代码。
- **高效的性能**：Git采用高效算法和数据结构处理大型项目和大量文件，保持较高速度和效率，适用于各种规模的项目。
- **本地化操作**：操作本地化减少对网络连接的依赖，提高操作速度，使得即使在没有网络连接的情况下也可以进行版本控制操作。
- **灵活的分布式架构**：每个开发者拥有完整的代码仓库，无需与中央服务器直接交互，提高容错性和稳定性，即使中央服务器故障，开发者仍可继续工作并进行协作和同步。

## 1.4 Git的工作流程

### Git流程



[https://blog.csdn.net/qq\\_26296197](https://blog.csdn.net/qq_26296197)

- 名词解释：
  - **远程仓库**：存放在网络上的 共享版本库，用于团队成员之间的 协作 和 代码共享。
  - **本地仓库**：每个开发者在本地计算机上拥有的 完整代码副本，用于进行版本控制操作。
  - **暂存区**：存放 待提交的修改 的区域，通过 `git add` 命令将工作区中的修改添加到暂存区。
  - **工作区**：开发者在本地文件系统中进行 编辑和修改 的区域，包含项目的实际文件。
- 基本工作流程：
  1. **抓取/克隆 (fetch/clone)**：从 远程仓库 获取 最新的代码 到 本地仓库。
  2. **检出 (checkout)**：切换 到指定的 分支 或者 版本，开始工作或者查看代码。
  3. **添加 (add)**：将工作区中的修改 添加到暂存区，准备提交到本地仓库。
  4. **提交 (commit)**：将暂存区中的修改 提交到本地仓库，形成一个 新的提交记录。
  5. **拉取 (pull)**：从远程仓库 获取 最新的代码更新到本地仓库，并 自动合并(merge) 到当前分支，相当于 `fetch+merge`。
  6. **推送 (push)**：将本地仓库中的提交 推送到远程仓库，将本地的修改同步到共享版本库中。

## 2、Git安装与常用指令

### 2.1 Git环境配置

#### 2.1.1 下载与安装

下载地址:<https://git-scm.com/>

当安装Git后首先要做的事情是 设置用户名 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

## 2.1.2 基本配置

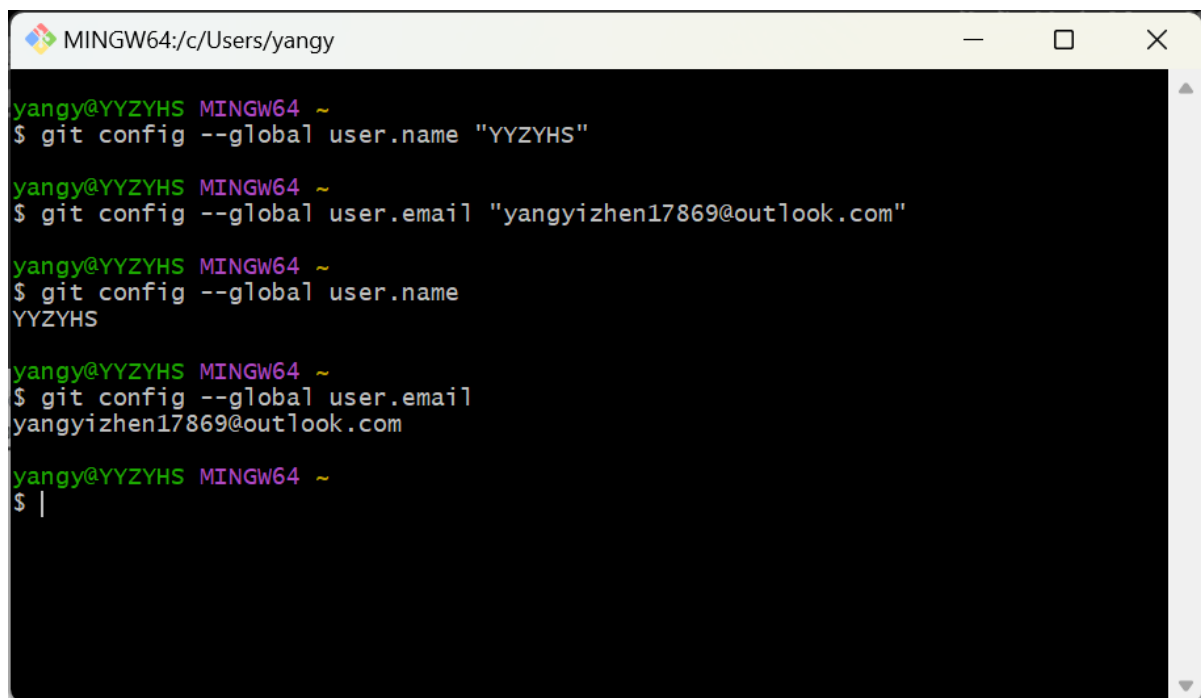
### 2.1.2.1 配置设置用户名称和email地址

1. 打开 Git Bash
2. 设置用户信息

```
1 | git config --global user.name "用户名"  
2 | git config --global user.email "邮箱"
```

3. 查看配置信息

```
1 | git config --global user.name  
2 | git config --global user.email
```



The screenshot shows a Windows terminal window titled "MINGW64:/c/Users/yangy". The prompt is "yangy@YYZYHS MINGW64 ~". The user enters the following commands and receives the following output:

```
$ git config --global user.name "YYZYHS"  
  
yangy@YYZYHS MINGW64 ~  
$ git config --global user.email "yangyizhen17869@outlook.com"  
  
yangy@YYZYHS MINGW64 ~  
$ git config --global user.name  
YYZYHS  
  
yangy@YYZYHS MINGW64 ~  
$ git config --global user.email  
yangyizhen17869@outlook.com  
  
yangy@YYZYHS MINGW64 ~  
$ |
```

## 2.2 获取本地仓库

要使用Git对我们的代码进行 版本控制，首先需要 获得本地仓库

1. 在电脑的任意位置创建一个空目录(例做如test)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`
4. 如果创建成功后可在文件夹下看到隐藏的git目录。

```
MINGW64:/d/图书馆/学习笔记/Git分布式管理工具/GitTest_01
yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (main)
$ git init
Initialized empty Git repository in D:/图书馆/学习笔记/Git分布式管理工具/GitTest_01/.git/

yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (master)
$ ll -a
total 0
drwxr-xr-x 1 yangy 197609 0 Mar 26 22:22 ./
drwxr-xr-x 1 yangy 197609 0 Mar 26 22:21 ../
drwxr-xr-x 1 yangy 197609 0 Mar 26 22:22 .git/

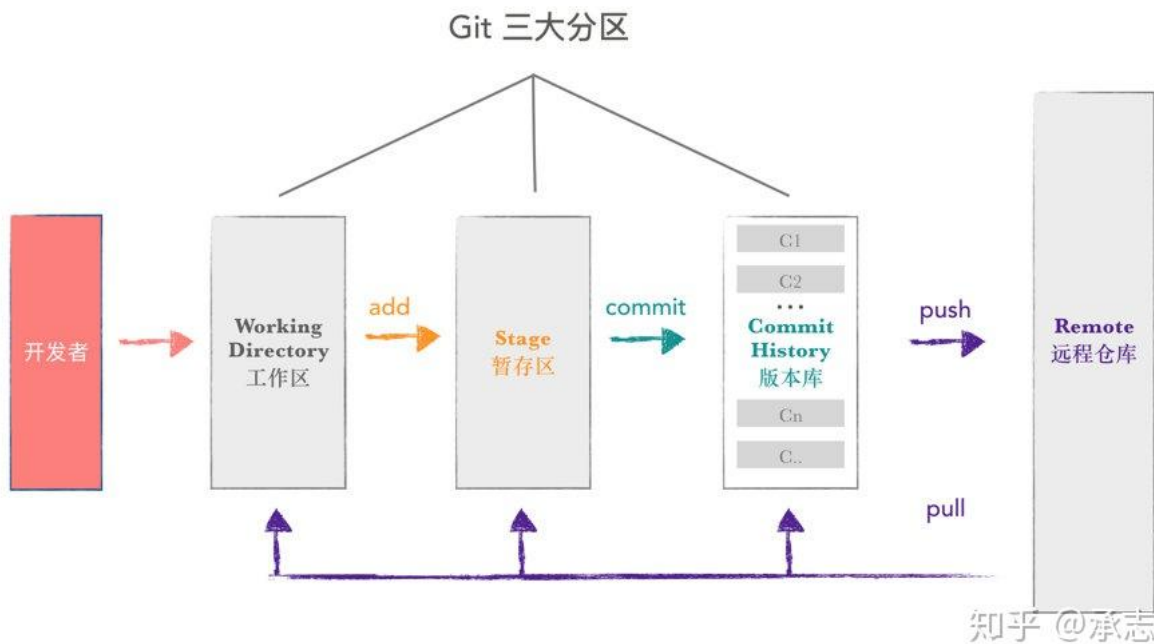
yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (master)
$ |
```

初始化git仓库成功

git初始化创建的git文件

## 2.3 基础操作指令

Git工作目录下对于文件的修改(增加、删除、更新)会存在几个状态, 这些修改的状态会随着我们执行Git的命令而发生变化



本章节主要讲解如何使用命令来控制这些状态之间的转换:

1. git add(工作区>暂存区)
2. git commit(暂存区->本地仓库)

### 2.3.1 查看当前的工作目录状态

`git status` 是 Git 版本控制系统中的一个基本命令, 它用于查看当前工作目录的状态。执行该命令会显示哪些文件被修改过、添加到暂存区(即将提交的文件)以及哪些文件还未被跟踪。这个命令可以帮助你了解你的工作目录当前处于什么状态, 以及接下来需要执行哪些操作。

在执行 `git status` 命令后, Git 会输出当前的工作目录状态, 通常包括以下信息:

1. 修改过的文件: 显示哪些文件已经被修改但尚未添加到暂存区。
2. 已暂存的文件: 显示哪些文件已经被添加到了暂存区, 即将包含在下一次提交中。

3. **未跟踪的文件**：显示哪些文件尚未被 Git 跟踪，通常是新添加的文件或者未被添加到版本控制中的文件。

通过定期执行 `git status` 命令，你可以了解你的工作目录中文件的变化，以便在适当的时候提交这些变化，或者对其进行进一步操作，比如添加到暂存区或者忽略某些文件。

```
yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (master)
$ touch file01.txt
yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (master)
$ git status
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
       file01.txt
nothing added to commit but untracked files present (use "git add" to track)
yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (master)
$
```

## 2.3.2 将文件添加到暂存区中

`git add` 是 Git 中的一个命令，用于将文件的更改添加到暂存区（stage area）。在使用 Git 进行版本控制时，通常会按照以下步骤操作：

1. **修改文件**：首先，您会修改项目中的文件，可能是添加新文件、修改现有文件或删除文件。
2. **将更改添加到暂存区**：一旦您对文件进行了更改，您需要将这些更改添加到 Git 的暂存区中。这样做将为将来的提交做好准备。

使用 `git add` 命令来执行此操作。例如：

```
1 | git add 文件名
```

或者，如果您要将所有更改添加到暂存区，则可以使用以下命令：

```
1 | git add .
```

这会将当前目录中所有更改（包括新文件、已修改文件和已删除文件）添加到暂存区。

3. **提交更改**：一旦您将要提交的更改添加到暂存区，您就可以使用 `git commit` 命令将其提交到您的本地仓库。
4. **推送更改**：如果您希望将更改推送到远程仓库（如 GitHub、GitLab 或 Bitbucket），您可以使用 `git push` 命令。

总之，`git add` 命令是将文件更改添加到暂存区的关键步骤之一，使您能够有效地管理您的代码更改并准备提交到版本控制系统中。

```
MINGW64:/d/图书馆/学习笔记/Git分布式管理工具/GitTest_01
yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (master)
$ git add .

yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file01.txt

yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (master)
$ |
```

### 2.3.3 将暂存区的代码提交到仓库中

`git commit` 是 Git 中的一个命令，用于将暂存区中的文件更改提交到本地仓库。提交操作将创建一个新的提交对象，包含了您所做的更改，并且会生成一个唯一的提交哈希值以标识这个提交。

通常情况下，`git commit` 命令的使用方式如下：

1. **添加更改到暂存区**: 首先，您需要使用 `git add` 命令将您想要提交的更改添加到暂存区。
2. **执行提交操作**: 一旦您将所有更改都添加到暂存区，您可以使用 `git commit` 命令来提交这些更改。例如：

```
1 | git commit -m "提交消息"
```

这会创建一个新的提交，其中 `-m` 选项用于指定提交消息。提交消息应该清楚地描述您所做的更改的目的和内容。良好的提交消息能够帮助团队成员更好地理解您的更改，并且在日后追踪提交历史时也非常有用。

3. **提交到本地仓库**: 提交成功后，您的更改将被保存到本地仓库中。这意味着您的更改已经在本地版本控制系统中记录下来了。

您还可以使用其他选项来执行更高级的提交操作，例如修改先前的提交、将提交合并等。`git commit` 命令非常灵活，适用于各种不同的版本控制工作流程。

```
MINGW64:/d/图书馆/学习笔记/Git分布式管理工具/GitTest_01
yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file01.txt

yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (master)
$ git commit -m "创建文件file01.txt"
[master (root-commit) 392787a] 创建文件file01.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 file01.txt

yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (master)
$ git status
On branch master
nothing to commit, working tree clean

yangy@YYZYHS MINGW64 /d/图书馆/学习笔记/Git分布式管理工具/GitTest_01 (master)
$
```

## 2.3.4 查看提交日志

`git log` 是 Git 中的一个命令，用于显示项目的提交历史。执行 `git log` 命令将列出最新提交到最旧提交之间的所有提交记录，并按照提交时间的倒序排列。

通常情况下，`git log` 命令会显示每个提交的以下信息：

- **提交哈希值 (commit hash)**：唯一标识每个提交的哈希值。
- **作者 (Author)**：提交的作者信息，包括 姓名和电子邮件地址。
- **日期 (Date)**：提交的日期和时间。
- **提交消息 (Commit message)**：提交时所附的消息，描述了提交所做的更改。

例如，执行 `git log` 命令将显示提交历史的详细信息，类似于以下内容：

```
1  commit abcdef1234567890abcdef1234567890abcdef12
2  Author: John Doe <john.doe@example.com>
3  Date:   Mon Jan 1 12:00:00 2024 +0000
4
5      Add new feature XYZ
6
7  commit 1234567890abcdef1234567890abcdef12345678
8  Author: Jane Smith <jane.smith@example.com>
9  Date:   Sun Dec 31 12:00:00 2023 +0000
10
11     Fix bug in component ABC
12
13  commit 0987654321fedcba0987654321fedcba09876543
14  Author: John Doe <john.doe@example.com>
15  Date:   Sat Dec 30 12:00:00 2023 +0000
16
17     Initial commit
```

在这个例子中，您可以看到每个提交的提交哈希值、作者、日期和提交消息。

`git log` 命令非常有用，可以帮助您了解项目的提交历史，查看谁做了什么更改以及何时做出了这些更改。您还可以使用不同的选项来过滤和定制 `git log` 命令的输出，以满足您的特定需求。

当执行 `git log` 命令时，可以使用各种选项来过滤和定制输出。以下是一些常用的 `git log` 选项及其效果：

1. **--all**：显示所有分支
2. **--pretty=oneline**：将提交信息显示为一行
3. **--abbrev-commit**：使得输出的commitId更简短
4. **--graph**：以图的形式显示

这些选项可以单独使用，也可以组合在一起以实现更精细的提交历史过滤和定制。通过灵活使用这些选项，您可以轻松地查看和分析 Git 项目的提交历史，并根据需要调整输出的内容。



## 2.3.5 版本回退

`git reset` 是 Git 中的一个命令，用于移动当前分支的 HEAD 指针以及相关引用（如分支或标签）到另一个提交。这个命令可以用于取消暂存的更改、撤销提交等操作。

`git reset` 命令的常见用法有三种模式：

1. **Soft 模式**: 在此模式下，`git reset` 将移动 HEAD 指针到指定的提交，但不会更改暂存区和工作目录的内容。这意味着之前的更改仍然在暂存区中，并且可以重新提交。示例：

```
1 | git reset --soft <commit>
```

2. **Mixed 模式**: 这是默认模式，`git reset` 将移动 HEAD 指针到指定的提交，并且会将暂存区的内容重置为该提交的内容，但不会更改工作目录的内容。这意味着之前的更改不再处于暂存状态，但仍然可以在工作目录中查看。示例：

```
1 | git reset --mixed <commit>
```

3. **Hard 模式**: 在此模式下，`git reset` 将移动 HEAD 指针到指定的提交，并且会将暂存区和工作目录的内容都重置为该提交的内容，擦除了之前的更改。这意味着之前的更改将被完全取消，慎用，因为会导致丢失未提交的更改。示例：

```
1 | git reset --hard <commit>
```

此外，`git reset` 命令还可以用来移动分支的指针，例如：

```
1 | git reset --hard HEAD^
```

这将会将当前分支（通常是 HEAD 所在的分支）的指针向上移动一个提交，相当于取消最近一次的提交并将更改丢弃。

总的来说，`git reset` 是一个强大的命令，可以用于撤销提交、移动分支指针等操作，但在使用时需要谨慎，以免不小心丢失重要的更改。

## 2.3.6 显示引用日志（可查看历史变更记录）

`git reflog` 是 Git 中的一个命令，用于显示引用日志（reference logs）。引用日志记录了本地仓库中 HEAD 和分支等引用的历史记录，包括它们的移动和更改。

执行 `git reflog` 命令将列出引用日志中的条目，通常显示每个引用的相关信息，如引用的哈希值、引用移动的时间戳以及执行的操作。这些信息可以帮助您跟踪仓库中引用的变化历史，包括分支切换、提交、重置等操作。

使用 `git reflog` 命令通常需要在需要查看本地仓库的历史变更记录时，或者在意外移动了引用（例如分支）后想要找回之前的状态时非常有用。引用日志提供了一种手段来回顾仓库中的操作历史，以便您了解每个引用的状态及其变更。

下面是一个 `git reflog` 命令的示例输出：



```
1 abcdef1 HEAD@{0}: reset: moving to abcdef1
2 1234567 HEAD@{1}: commit: Added new feature XYZ
3 abcdef1 HEAD@{2}: commit: Fixed bug in component ABC
4 9876543 HEAD@{3}: commit (initial): Initial commit
```

在这个示例中，您可以看到每个引用操作的相关信息，包括提交哈希、操作类型和操作描述。通过阅读引用日志，您可以了解仓库中引用的变更历史，以及每次操作的影响。

## 2.3.7 添加文件至忽略列表

添加文件至 Git 忽略列表是一种管理项目文件的重要技巧。通过将不需要跟踪的文件或目录添加到 `.gitignore` 文件中，可以确保这些文件不会被 Git 跟踪或包含在版本控制中，从而使仓库更加干净和有组织。

### 创建 `.gitignore` 文件

首先，在您的项目根目录下创建一个名为 `.gitignore` 的文件。这个文件将包含要忽略的文件、文件夹或特定类型的文件的列表。

### 编辑 `.gitignore` 文件

使用文本编辑器打开 `.gitignore` 文件，并在其中添加要忽略的文件、文件夹或模式。每个条目占据一行。

### 忽略模式的语法

Git 忽略模式使用通配符和特殊字符来匹配文件或路径。常用的模式包括：

- 通配符匹配，如 `*`、`?`、`[abc]` 等；
- 目录匹配，如 `/`、`**/`；
- 特殊字符，如 `\` 进行转义。

### 示例 `.gitignore` 文件

以下是一个示例 `.gitignore` 文件的内容：

```
1 # 忽略临时文件
2 *.tmp
3
4 # 忽略编译生成的文件
5 /build/
6
7 # 忽略日志文件
8 logs/*.log
9
10 # 忽略IDE和编辑器的配置文件
11 .vscode/
12 .idea/
```

在这个示例中，`*.tmp` 将忽略所有以 `.tmp` 结尾的文件，`/build/` 将忽略整个 `build` 目录，`logs/*.log` 将忽略 `logs` 目录下的所有 `.log` 文件，`.vscode/` 和 `.idea/` 将忽略 VS Code 和 IntelliJ IDEA 的配置文件夹。

### 提交 `.gitignore` 文件

如果您希望 `.gitignore` 文件适用于整个团队或者您的仓库是公开的，则将 `.gitignore` 文件提交到版本控制中，以便团队成员或其他开发者知晓并共享这些忽略规则。

通过添加文件至 Git 忽略列表，您可以更好地管理项目中的文件，使得版本控制更加清晰、有序，并确保仓库中只包含必要的文件和代码。