Git常用命令

一、常用命令

Git 是一个功能强大的版本控制系统，广泛应用于开发过程中。下面列举了 Git 的一些常用命令及其用法，包括初始化本地仓库和设置账号信息。

1. Git 安装与配置

1.1 安装 Git

- 在 **Windows** 上可以通过 Git 官网 下载并安装。
- 在 **Linux** 系统中，你可以使用包管理器安装 Git：

```
sudo apt install git  # Ubuntu/Debian
sudo yum install git  # CentOS/RHEL
sudo pacman -S git    # Arch Linux
```

- 在 **macOS** 上，你可以通过 Homebrew 安装：

```
brew install git
```

1.2 配置 Git 用户信息

- 在 Git 中设置用户名和邮箱，这些信息会记录在提交历史中。

```
git config --global user.name "Your Name"       # 设置用户名
git config --global user.email "your.email@example.com"  # 设置用户邮箱
```

    - `--global` 表示全局配置，会应用于当前用户的所有仓库。
    - 如果你只想为某个特定仓库配置，可以在仓库目录下执行命令并省略 `--global`，如：

```
git config user.name "Another Name"  # 仅对当前仓库有效
```

1.3 查看配置

- 查看当前的配置信息：

```
git config --list
```

---

2. 初始化本地 Git 仓库

2.1 初始化 Git 仓库

- 在一个新项目中初始化本地 Git 仓库：

```
git init
```

    - 这将在当前目录下创建一个 `.git` 子目录，表示该目录已被 Git 跟踪。

2.2 克隆远程仓库

- 如果你想从远程仓库克隆一个现有的 Git 仓库到本地：

```
git clone <repository_url>
```

    - 示例：

```
git clone https://github.com/user/repo.git
```

---

3. 常用 Git 操作

3.1 查看当前状态

- 查看当前仓库的状态，显示哪些文件被修改、哪些文件还没有被提交：

```
git status
```

3.2 添加文件到暂存区

- 将文件添加到 Git 的暂存区（准备提交）：

```
git add <file>      # 添加单个文件
git add .           # 添加当前目录下所有修改过的文件
```

3.3 提交更改

- 提交已暂存的更改到本地仓库：

```
git commit -m "Commit message"    # 提交并附带消息
```

    - 如果没有使用 `-m` 选项，Git 会打开编辑器让你输入提交消息。

3.4 查看提交历史

- 查看提交历史记录：

```
git log
```

    - 你可以使用 `git log --oneline` 查看简洁的历史记录。

3.5 撤销更改

- 撤销暂存区的更改（将文件从暂存区移除，保持工作区修改）：

```
git reset <file>
```

- 撤销工作区的修改（文件回到最后一次提交时的状态）：

```
git checkout -- <file>
```

3.6 查看差异

- 查看工作区和暂存区之间的差异：

```
git diff
```

- 查看暂存区和上次提交之间的差异：

```
git diff --cached
```

---

4. 远程仓库操作

4.1 查看远程仓库

- 查看配置的远程仓库：

```
git remote -v
```

4.2 添加远程仓库

- 将一个远程仓库与本地仓库关联：

```
git remote add origin <repository_url>
```

4.3 推送到远程仓库

- 将本地分支的更改推送到远程仓库：

```
git push origin <branch_name>
```

4.4 从远程仓库拉取最新更改

- 拉取远程仓库的更新并合并到当前分支：

```
git pull origin <branch_name>
```

4.5 拉取最新的分支

- 从远程仓库拉取更新（但不自动合并）：

```
git fetch
```

4.6 删除远程分支

- 删除远程仓库中的分支：

```
git push origin --delete <branch_name>
```

总结

这些是 Git 的常用命令，涵盖了从初始化本地仓库、设置账号信息、分支管理，到与远程仓库的交互等多种操作。掌握这些命令将帮助你更有效地管理和控制代码版本。如果你需要更深入的了解，可以查阅 Git 官方文档。

二、`.gitignore` 文件的编写方法

如果你不想将某些文件或文件夹包含在 Git 提交中，可以使用 `.gitignore` 文件来指定这些文件或目录。`.gitignore` 文件是一个文本文件，列出了不想被 Git 跟踪的文件模式。

基本语法

`.gitignore` 文件中的每一行定义一个模式，用于匹配文件或目录。常见的规则包括：

- **注释**：以 `#` 开头的行是注释。
- **忽略某个文件**：直接写文件名或路径。
- **忽略某种类型的文件**：使用通配符 `*`。
- **忽略目录**：在目录后加上 `/`。
- **否定规则**：前面加 `!` 可以让某些被忽略的文件重新被 Git 跟踪。

示例 `.gitignore` 文件

假设你有以下的需求：

1. 忽略所有 `.log` 文件。
2. 忽略某个特定的目录，比如 `node_modules/`。
3. 忽略所有的 `.DS_Store` 文件（macOS 上的系统文件）。
4. 忽略所有以 `.bak` 结尾的备份文件。
5. 确保 `.gitignore` 文件不被忽略。

你可以写出如下 `.gitignore` 文件：

```
# 忽略所有 .log 文件
*.log

# 忽略 node_modules 目录
node_modules/

# 忽略系统文件 .DS_Store
.DS_Store

# 忽略所有 .bak 文件
*.bak

# 不忽略 .gitignore 文件
!.gitignore
```

详细解释

1. **`*.log`**：忽略所有扩展名为 `.log` 的文件。
2. **`node_modules/`**：忽略 `node_modules` 目录及其所有内容。`/` 表示这是一个目录。
3. **`.DS_Store`**：忽略名为 `.DS_Store` 的文件（这是 macOS 系统文件）。
4. **`*.bak`**：忽略所有以 `.bak` 为扩展名的备份文件。
5. **`!.gitignore`**：通过 `!` 否定规则，确保 `.gitignore` 文件本身不会被忽略。

常见的 `.gitignore` 模式

- **忽略所有文件**：`*` 会忽略所有文件。
- **忽略某个特定目录**：`dir/` 会忽略名为 `dir` 的目录。
- **忽略某种文件类型**：`*.tmp` 忽略所有 `.tmp` 文件。
- **忽略某个特定的文件**：`config.json` 忽略名为 `config.json` 的文件。
- **忽略多个文件扩展名**：`*.log` 忽略所有 `.log` 文件，`*.bak` 忽略所有 `.bak` 文件。

一些其他的示例

忽略日志文件和临时文件

```
# 忽略所有日志文件
*.log

# 忽略所有临时文件
*.tmp
```

忽略特定目录

```
# 忽略 IDE/编辑器生成的文件
.vscode/
.idea/

# 忽略构建工具生成的文件夹
build/
dist/
```

忽略所有文件，但保留特定文件

```
# 忽略所有文件
*

# 保留某些文件
!.gitignore
!README.md
```

注意事项

1. `.gitignore` **只对未被 Git 跟踪的文件有效**。如果某个文件已经被 Git 跟踪（即已经添加到暂存区或提交），它仍然会被 Git 记录下来，即使你在 `.gitignore` 中添加了它。要停止跟踪已提交的文件，你需要先使用 `git rm --cached <file>` 来将其从 Git 的跟踪中移除，然后才会生效。
2. **路径的相对性**：在 `.gitignore` 文件中，规则是相对于 `.gitignore` 文件所在的目录

进行匹配的。如果你希望在全局范围内忽略某些文件，通常会使用更具体的路径或使用 `/` 表示目录。