实验报告 1

姓名: 郭千纯

学号：23020007033

课程：系统开发工具基础

2024 年 8 月 28 日

目录

1 练习内容

版本控制（Git）与 Latex 文档编辑

2 实例及结果

2.1 实例 1

git init: 创建一个新的 git 仓库，其数据会存放在一个名为.git 的目录下

图 1: 实例 1

2.2 实例 2

git help <command>: 获取 git 命令的帮助信息

2.3 实例 3

git status: 显示当前的仓库状态

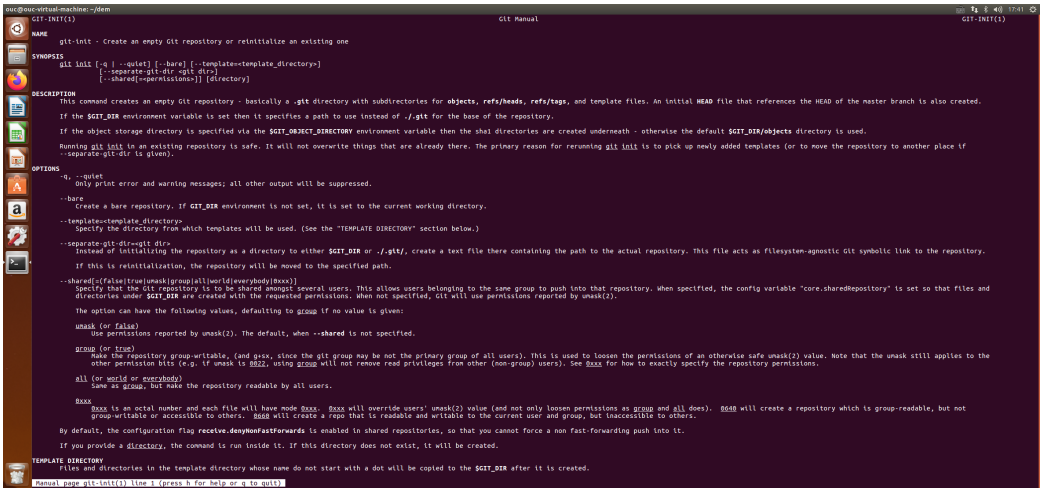

图 2: 实例 2

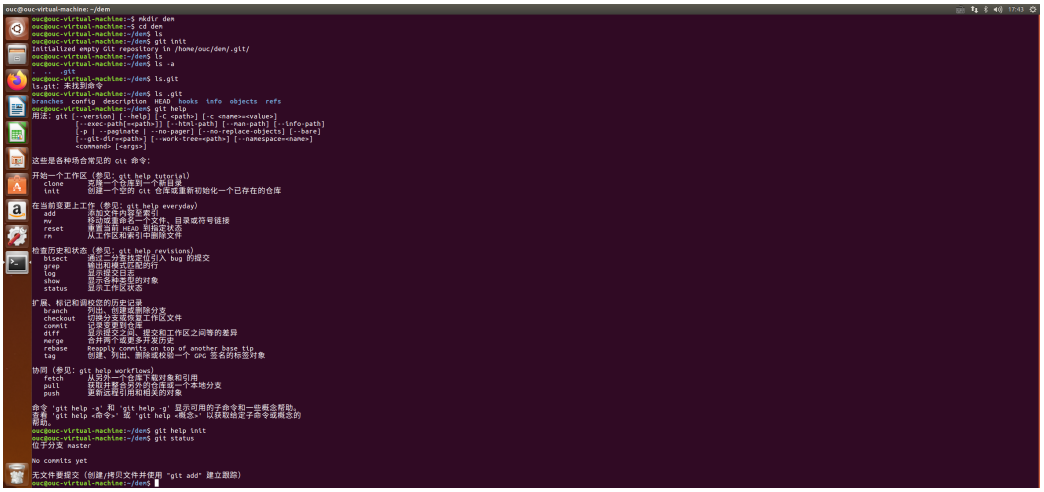

图 3: 实例 3

2.4 实例 4

git add <filename>: 添加文件到暂存区

2.5 实例 5

git commit: 创建一个新的提交

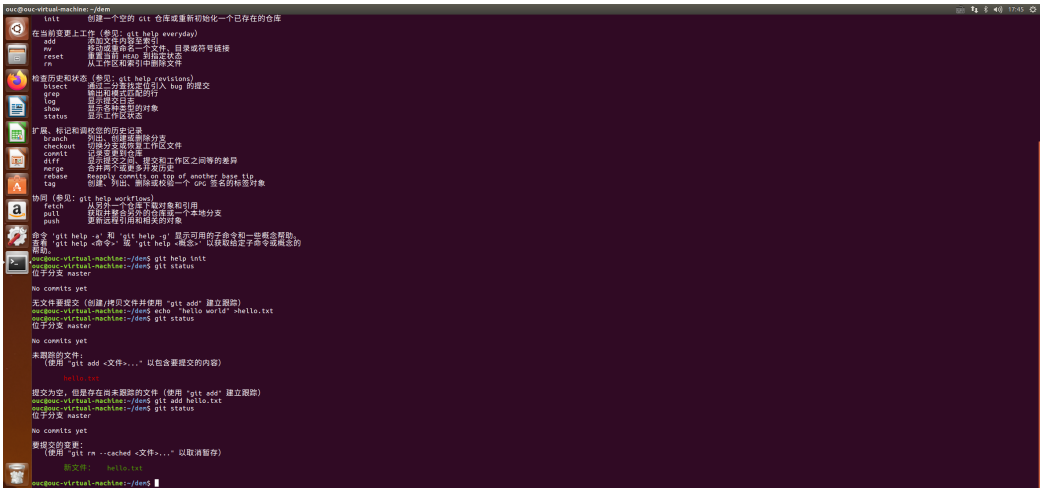

图 4: 实例 4

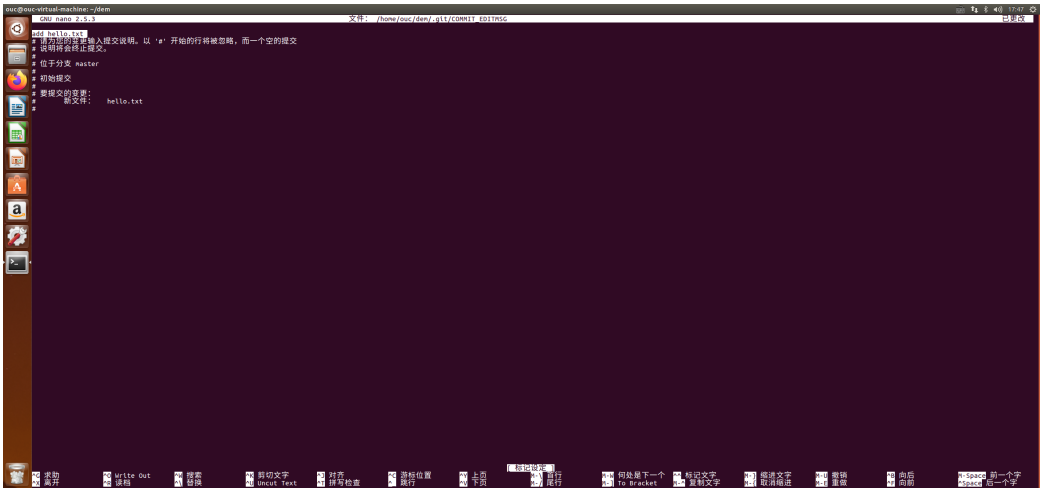

图 5: 实例 5

2.6 实例 6

git log: 显示历史日志

2.7 实例 7

git log –all –graph –decorate: 可视化历史记录（有向无环图）

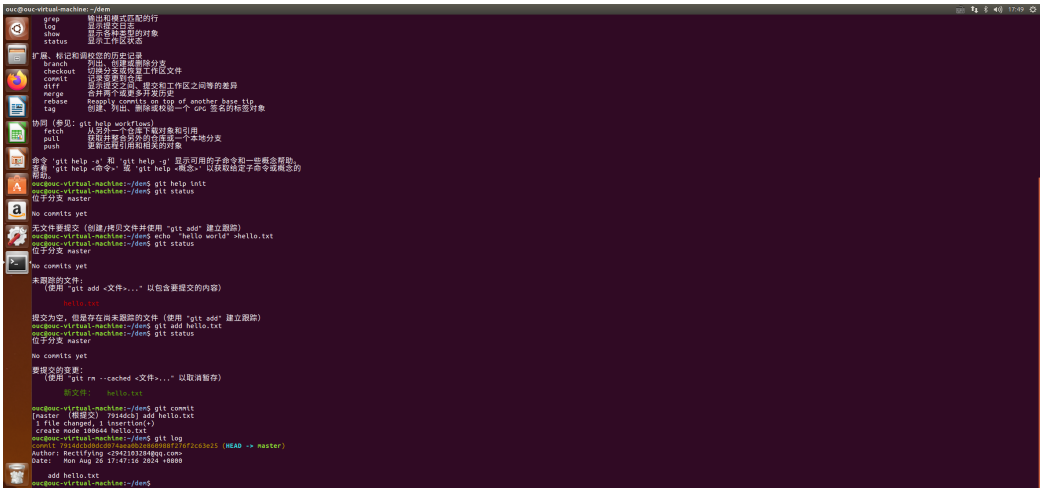

图 6: 实例 6

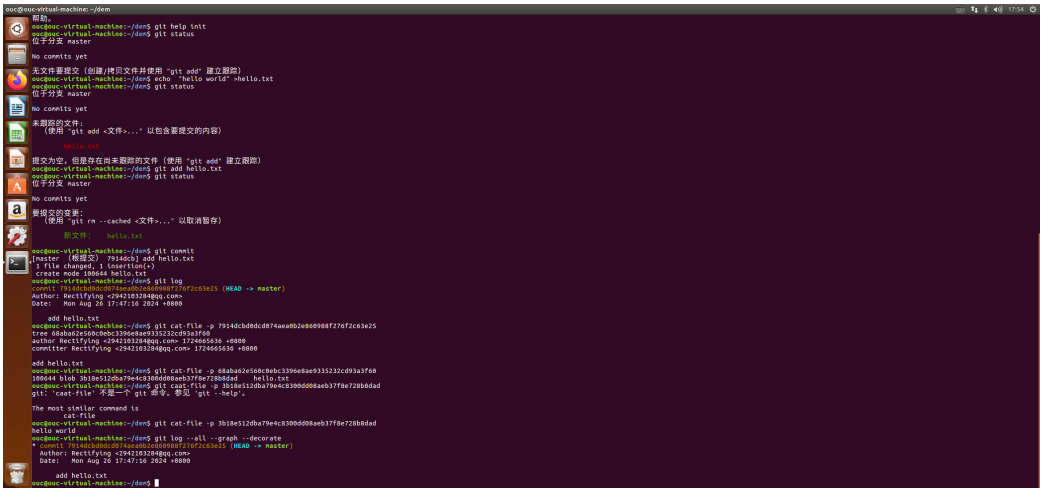

图 7: 实例 7

2.8 实例 8

git checkout <revision>: 更新 HEAD 和目前的分支

2.9 实例 9

git diff <filename>: 显示与暂存区文件的差异

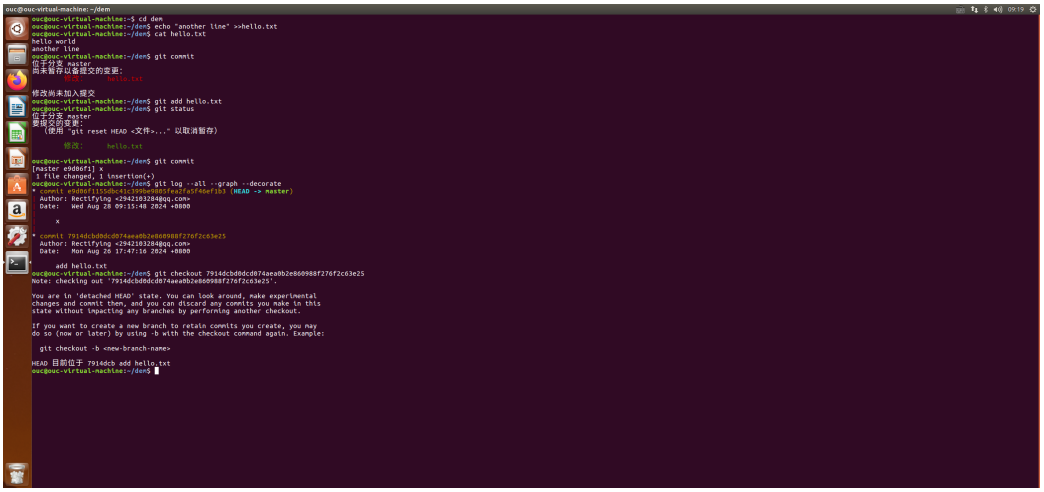

图 8: 实例 8

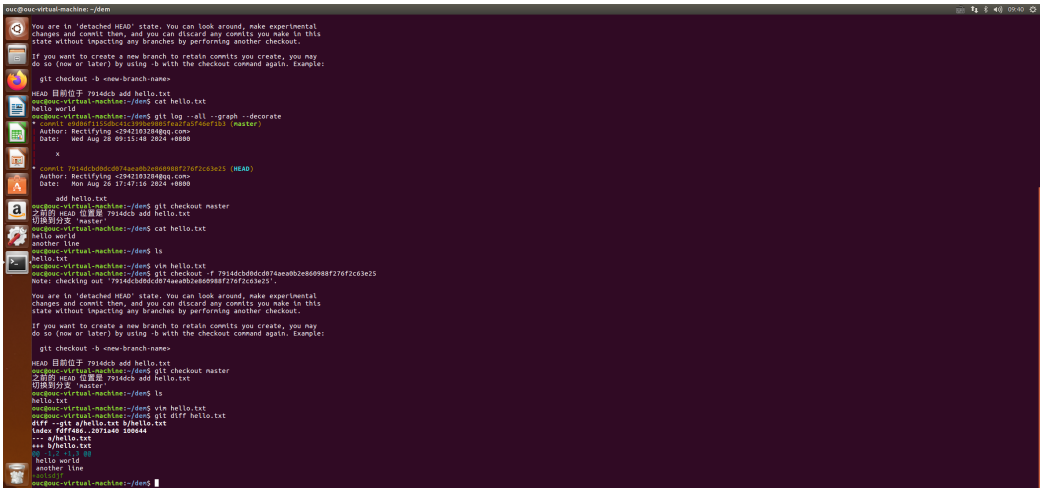

图 9: 实例 9

2.10 实例 10

git diff <revision> <filename>: 显示某个文件两个版本之间的差异

2.11 实例 11

\documentclass 命令必须出现在每个 LaTeX 文档的开头。花括号内的文本指定了文档的类型。article 文档类型适合较短的文章

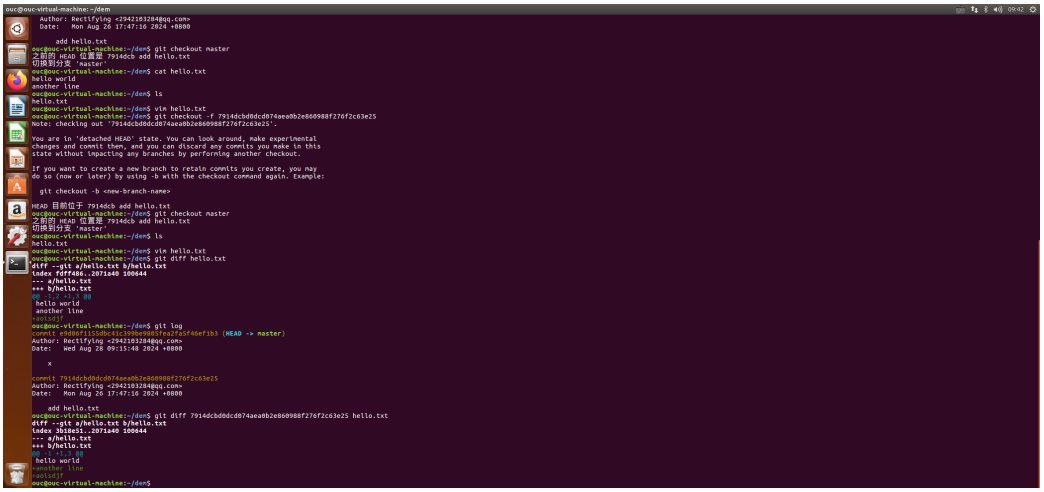

图 10: 实例 10

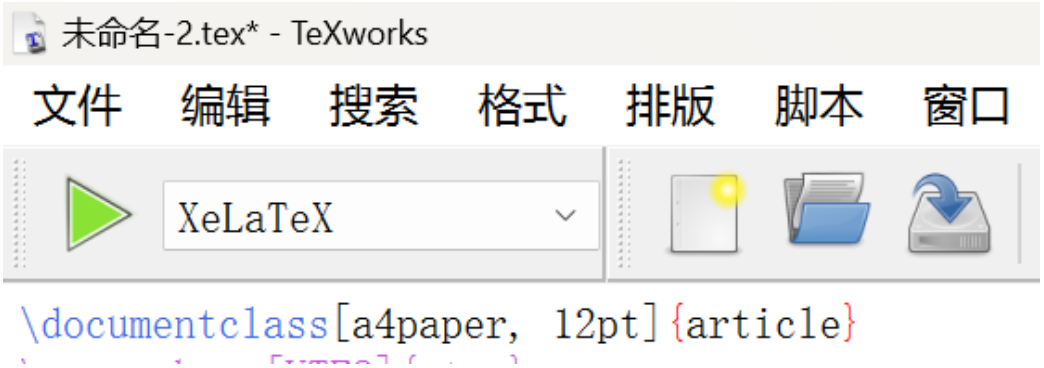

图 11: 实例 11

2.12 实例 12

\maketitle 命令可以给文档创建标题。你需要指定文档的标题。如果没有指定日期，就会使用现在的时间，作者是可选的。

2.13 实例 13

下列分节命令适用于 article 类型的文档：

\section...

\subsection...

\subsubsection...

```
\begin{document}
  \title{实验报告1}
  \author{\\姓名:郭千纯\\
          \\学号：23020007033\\
          \\课程：系统开发工具基础\\}
  \date{\today\\}
  \maketitle
```

图 12: 实例 12

\paragraph...

\subparagraph...

```
\section{练习内容}
版本控制（Git）与Latex 文档编辑

\section{实例及结果}

\subsection{实例1}
git init:创建一个新的 git 仓库，其数据会存放在一个名为 .git 的目录下

\begin{figure}[h!]
  \centering
  \includegraphics[width=1\textwidth]{实例1.png}
  \caption{实例1}
\end{figure}

\subsection{实例2}
git help <command>: 获取 git 命令的帮助信息

\begin{figure}[h!]
  \centering
  \includegraphics[width=1\textwidth]{实例2.png}
  \caption{实例2}
\end{figure}
```

图 13: 实例 13

2.14 实例 14

页码可以使用\pagenumbering... 在阿拉伯数字和罗马数字见切换

```
\pagenumbering{roman}
\tableofcontents
\newpage
\pagenumbering{arabic}
```

图 14: 实例 14

2.15 实例 15

\newpage 命令会另起一个页面

```
\pagenumbering{roman}
\tableofcontents
\newpage
\pagenumbering{arabic}
```

图 15: 实例 15

2.16 实例 16

使用 CTeX 宏包。只需要在文档的前导命令部分添加：
\usepackage[UTF8]ctex 就可以了。
在编译文档的时侯使用 xelatex 命令，因为它是支持中文字体的。

```
\usepackage[UTF8]{ctex}
```

图 16: 实例 16

2.17 实例 17

注意，反斜杠不能通过反斜杠转义（不然就变成了换行了），使用 \textbackslash 命令代替。

```
\subsection{实例13}
下列分节命令适用于 article 类型的文档：

\textbackslash section{...}
\textbackslash subsection{...}
\textbackslash subsubsection{...}
\textbackslash paragraph{...}
\textbackslash subparagraph{...}
```

图 17: 实例 17

2.18 实例 18

在 LaTeX 文档中插入图表。这里我们需要引入 graphicx 包。图片应当是 PDF，PNG，JPEG 或者 GIF 文件。

```
\usepackage{graphicx}
```

图 18: 实例 18

2.19 实例 19

\centering 将图片放置在页面的中央。

```
\begin{figure}[h!]
  \centering
  \includegraphics[width=1\textwidth]{实例19.png}
  \caption{实例19}
\end{figure}
```

图 19: 实例 19

2.20 实例 20

\includegraphics... 命令可以自动将图放置到你的文档中，图片文件应当与 TeX 文件放在同一目录下。

```
\begin{figure}[h!]
  \centering
  \includegraphics[width=1\textwidth]{实例19.png}
  \caption{实例19}
\end{figure}
```

图 20: 实例 20

3 解题及感悟

克隆本课程网站的仓库

克隆仓库：

git clone https://github.com/missing-semester-cn/missing-semester-cn.github.io.git

cd missing-semester-cn.github.io

1. 将版本历史可视化并进行探索

可视化版本历史：使用 Git 图形界面工具，如 gitk 或 git log：
git log –graph –oneline –all
或使用 Git 图形界面工具（如 Sourcetree, GitKraken）来更直观地浏览提交历史。
2. 是谁最后修改了 README.md 文件？（提示：使用 git log 命令并添加合适的参数）
查看 README.md 文件的最后修改者：
git log -p README.md
在输出的日志中，最后一次修改 README.md 文件的提交者是最接近顶部的条目。
3. 最后一次修改 _config.yml 文件中 collections: 行时的提交信息是什么？(提示：使用 git blame 和 git show）
查看 _config.yml 文件中 collections: 行最后一次修改时的提交信息：
git blame _config.yml
查找 collections: 行对应的提交哈希值，然后使用以下命令查看提交信息：
git show <commit-hash>
替换 <commit-hash> 为你从 git blame 中得到的实际提交哈希值。

使用 Git 时的一个常见错误是提交本不应该由 Git 管理的大文件，或是将含有敏感信息的文件提交给 Git 。尝试向仓库中添加一个文件并添加提交信息，然后将其从历史中删除
要向仓库添加一个文件并删除它的历史记录，请按照以下步骤操作：添加文件：
echo ”This is a test file.” > testfile.txt
git add testfile.txt
git commit -m ”Add testfile.txt”
删除文件的历史记录：
使用 git filter-branch git filter-repo 来从历史记录中删除文件。git filter-repo 是更现代且推荐的方法（需要先安装）。
使用 git filter-repo：
git filter-repo –path testfile.txt –invert-paths

如果你没有安装 git filter-repo，可以使用 git filter-branch（注意这可能会更复杂且不推荐）：

git filter-branch –force –index-filter

'git rm –cached –ignore-unmatch testfile.txt'

–prune-empty –tag-name-filter cat – –all

清理和推送更改：

rm -rf .git/refs/original/

git reflog expire –expire=now –all-ref –rewrite

git gc –prune=now –aggressive

git push origin –force –all

git push origin –force –tags

这样你可以确保添加的文件从所有历史记录中删除，并将更改推送到远程仓库。

从 GitHub 上克隆某个仓库，修改一些文件。当您使用 git stash 会发生什么？当您执行 git log –all –oneline 时会显示什么？通过 git stash pop 命令来撤销 git stash 操作，什么时候会用到这一技巧？

1. 使用 git stash：

当你执行 'git stash' 时，Git 会将你当前工作目录和暂存区的更改保存到一个新的存储栈中，并恢复到最近一次提交的干净状态。这让你可以在不提交更改的情况下切换分支或处理其他任务。

2. 执行 git log –all –oneline：

这个命令会显示所有分支的提交历史，按提交的简短哈希值和提交信息展示。你会看到所有提交的记录，包括你当前分支和其他分支上的提交。

3. 使用 git stash pop：

当你执行 'git stash pop' 时，Git 会将最近保存的 stash 应用到你的工作目录中，并将其从 stash 栈中删除。这一技巧常用于在处理临时任务或切换分支后恢复未完成的工作。例如，当你需要临时切换到另一分支修复问题时，'git stash' 可以保存当前的工作状态，完成修复后使用 'git stash pop' 恢复你的更改。

与其他的命令行工具一样，Git 也提供了一个名为 /.gitconfig 配置文件(或 dotfile)。请在 /.gitconfig 中创建一个别名，使您在运行 git graph 时，您可以得到 git log –all –graph –decorate –oneline 的输出结果；
要在 ‘ /.gitconfig‘ 中创建一个别名，使 ‘git graph‘ 运行 ‘git log –all –graph –decorate –oneline‘，请按照以下步骤操作：
1. 打开 ‘ /.gitconfig‘ 文件（如果文件不存在，可以创建它）。
2. 在文件中添加以下配置：
graph = log –all –graph –decorate –oneline
3. 保存文件并关闭编辑器。
现在，运行 ‘git graph‘ 会显示 ‘git log –all –graph –decorate –oneline‘ 的输出结果。

您可以通过执行 git config –global core.excludesfile ~/.gitignore_global 在 ~/.gitignore_global 中创建全局忽略规则。配置您的全局 gitignore 文件来自动忽略系统或编辑器的临时文件，例如.DS_Store；
要创建一个全局 ‘.gitignore‘ 文件来自动忽略系统或编辑器的临时文件，例如 ‘.DS_Store‘，请按照以下步骤操作：
1. 创建或编辑全局 ‘.gitignore‘ 文件：
打开终端，并运行以下命令来创建或编辑 ‘~/.gitignore_global‘ 文件：
nano /.gitignore_global
或者使用你喜欢的文本编辑器，比如 ‘vim‘、‘code‘（Visual Studio Code）等。
2. 在 ‘~/.gitignore_global‘ 文件中添加忽略规则：
在文件中添加要忽略的文件或目录的规则。例如：
Thumbs.db
你可以根据需要添加其他规则来忽略更多文件或目录。
3. 保存并关闭文件：
如果你使用的是 ‘nano‘，按 ‘Ctrl + X‘ 退出，按 ‘Y‘ 确认保存更改，然后按 ‘Enter‘ 确认文件名。
4. 配置 Git 使用这个全局忽略文件：
运行以下命令来告诉 Git 使用这个全局忽略文件：
git config –global core.excludesfile ~/.gitignore_global

现在，Git 将自动忽略你在 ‘/.gitignore_global‘ 文件中指定的文件和目录。在任何 Git 仓库中，这些忽略规则都会生效。