张高畅

What is Git

- 当今世界上使用最广泛的现代版本控制系统是 Git。
- Git 具有大多数团队和个人开发人员所需的功能、性能、安全性和灵活性。
- 大量的开发人员已经有了 Git 的经验。
- Git是一个有十多年历史的开源项目，拥有强大的社区支持和庞大的用户群。
- Git 为用户提供了很多功能。掌握Git可能需要一些时间，但是一旦学会，就可以利用这种力量来提高团队的开发速度。

Git的工作原理

1. 使用Git托管工具（如 Bitbucket）创建存储库(**repository**)”
2. 将**repository**复制（**clone**）到本地计算机
3. 将文件添加到本地repo并提交(**commit**)更改
4. 将更改推送(**push**)到主分支
5. 使用 git 托管工具对文件进行更改并提交
6. 将更改拉取(**pull**)到本地计算机
7. 创建一个分支（**branch**），进行更改，提交更改
8. 创建拉取请求(**pull request**)（对主分支进行更改的建议）
9. 将分支合并(**merge**)到主分支

Git SSH

What is a Git SSH key

> SSH key是 SSH（secure shell）网络协议的访问凭据。这种经过身份验证和加密的安全网络协议用于不安全的开放网络上的计算机之间的远程通信。

> SSH 使用一对密钥在远程参与方之间启动安全握手。密钥对包含公钥和私钥。将公钥视为“锁”，将私钥视为“密钥”会更有帮助。您将公共“锁”给远程方以加密或“锁定”数据。然后使用您保存在安全位置的“私钥”打开此数据。

How to create an SSH key

> SSH 密钥是通过公钥加密算法生成的，最常见的是 RSA 或 DSA。
>
> SSH 密钥是使用密钥生成工具创建的。SSH 命令行工具套件包括一个注册机工具。

创建SSH key的步骤

1. 执行以下命令以开始创建SSH key

```
1 ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

- 这条命令会用邮箱作为label来创建SSH key
- 也可以使用ED25519算法来代替RSA算法来创建SSH key，这会有更高的安全性、更快的生成速度和更少的储存空间占用，命令如下：

```
1 ssh-keygen -t ed25519 -C "your_email@example.com"
```

2. 输入保存key的文件的路径，或直接敲回车来使用默认路径
3. 下一个提示将要求输入安全密码(secure passphrase)。passphrase将为 SSH增加安全性，在使用SSH key时都需要提供passphrase。如果有人有权访问存储私钥的计算机，他们也可以访问使用该密钥的任何系统。向密钥添加passphrase将防止这种情况。

这时候，一个新的SSH key已经生成在之前指定路径的文件了。

4. 将新创建的SSH key添加到ssh-agent

ssh-agent是SSH工具套件中保存私钥的程序，就像钥匙扣。除了保存私钥外，它还能代理使用私钥向SSH发起　请求，以确保私钥不会被不安全地传递。

在将新的 SSH key添加到 ssh-agent 之前，先通过执行以下命令确保 ssh-agent 正在运行：

```
1 $ eval "$(ssh-agent -s)"
2 > Agent pid 59566
```

确保运行 ssh-agent 后，以下命令会将新的 SSH 密钥添加到本地 SSH 代理。

```
ssh-add -K /Users/you/.ssh/id_rsa
```

Setting up a repository

Initializing a new repository

1. 前往项目文件夹根目录
2. 使用 `git init` 命令在当前目录创建一个新的repo

📌 git init

该命令将创建一个新的 Git 存储库。它可用于将现有的未版本控制项目转换为 Git 存储库或初始化新的空存储库。大多数其他 Git 命令在初始化的存储库之外不可用，因此这通常是在新项目中运行的第一个命令。

Cloning an existing repository

📌 git clone

`git clone` 用于创建远程存储库的副本，需要传递存储库 URL。Git 支持几种不同的网络协议和相应的 URL 格式。以 Git SSH 协议为例。Git SSH URL 遵循以下模板：

`git clone git@HOSTNAME:USERNAME/REPONAME.git`

样例：

`git@bitbucket.org:rhyolight/javascript-data-store.git`

- HOSTNAME: bitbucket.org
- USERNAME: rhyolight
- REPONAME: javascript-data-store

Saving changes

📌 git add

该命令将工作目录中的更改添加到暂存区域。它告诉 Git 你希望在下一次提交中包含对特定文件的更新。但是，这不会直接对存储库产生影响。在运行 `git commit` 之前，更改

实际上不会被记录。

- `git add --all`:将在代码库中接收所有已更改和未跟踪的文件，并将它们添加到代码库，以及更新代码库的工作树。
- `git add <file>`:将file中的变更暂存。
- `git add <directory>`:将directory中的变更暂存。

git status

该命令用于查看工作目录和暂存区域的状态。

git commit

该命令捕获项目当前暂存变更的快照。Git 的版本控制模型基于快照。Git 会在每次提交中记录每个文件的全部内容。

常用选项：

1. `git commit`:提交暂存的快照。这将启动文本编辑器，提示您输入提交消息。输入消息后，保存文件并关闭编辑器以创建实际提交。
2. `git commit -a`:提交工作目录中所有变更的快照。这仅包括对跟踪文件的修改。
3. `git commit -m "yahayaha"`:立即创建带有传递提交消息的提交。默认 `git commit` 将打开本地配置的文本编辑器，并提示输入提交消息。传递 `-m` 选项将放弃文本编辑器提示，转而使用内联消息。
4. `git commit -am "yahayaha"`:组合了 `-a` 和 `-m` 选项的高级用户快捷命令。
5. `git commit --amend`:此选项为提交命令添加了另一层功能。传递此选项将修改上次提交。分阶段的变更将添加到先前的提交中，而不是创建新提交。

git diff

`git diff` 是一个多用途 Git 命令，它在执行时会在 Git 数据源上运行比对功能。这些数据源可以是提交、分支、文件等。`git diff` 命令通常与 `git status` 和 `git log` 一起使用，用于分析 Git 代码存储库的当前状态。

突出显示变更：

1. `git diff --color-words`:输出仅显示已变更的用颜色编码的字。
2. `git diff-highlight`:diff-highlight 将匹配的比对输出行配对，并突出显示已变更的子字片段。

比较文件：

- `git diff HEAD ./path/to/file`:调用时，此示例的作用域为 `./path/to/file`，它会将工作目录中的具体变更与索引进行比较，显示尚未暂存的变更。默认情况下，`git diff` 将执行与 `HEAD` 的比较。在上面的 `git diff ./path/to/file` 示例中省略了 `HEAD` 具有同样的效果。
- `git diff --cached ./path/to/file`:使用 `--cached` 选项调用 `git diff` 时，比对会将暂存的变更与本地存储库进行比较。`--cached` 选项与 `--staged` 同义。
- 默认情况下，`git diff` 会显示自上次提交以来所有未提交的变更。

比较两个不同提交之间的文件：

`git diff` 可以将 Git 引用传递给提交进行比对。一些示例引用包括 `HEAD`、标记和分支名称。Git 中的每个提交都有一个提交 ID，可以在执行 `git log` 时获得这个提交 ID，将这个提交 ID 传递给 `git diff`。

比较分支：

- `git diff branch1..other-feature-branch`：此示例引入了点运算符。此示例中的两个点表示比对输入是两个分支的尖端。如果省略点并在分支之间使用空格，也会产生同样的效果。
- `git diff branch1...other-feature-branch`：三点运算符通过变更第一个输入参数 `branch1` 来启动比对。它将 `branch1` 变更为两个比对输入之间共享的共同祖先提交引用，即 `branch1` 和其他功能分支的共享祖先。最后一个参数输入参数保持不变，就像其他功能分支的尖端一样。

比较来自两个分支的文件：

将该文件的路径作为第三个参数传递给 `git diff`：`git diff main new_branch ./diff_test.txt`

Git ignore

Git 将每个文件视为三种之一：

1. tracked - 之前已暂存或提交的文件。
2. untracked - 尚未暂存或提交的文件。
3. ignored - 已明确告知 Git 要忽略的文件。

在Git工作区的根目录下创建一个特殊的 `.gitignore` 文件，然后把要忽略的文件名填进去，Git就会自动忽略这些文件。

> 忽略文件的原则是：
>
> 1. 忽略操作系统自动生成的文件，比如缩略图等；
> 2. 忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，比如Java编译产生的 `.class` 文件；
> 3. 忽略你自己的带有敏感信息的配置文件，比如存放口令的配置文件。

`.gitignore` 文件样例：

```
#Windows:Thumbs.db
ehthumbs.db
Desktop.ini

#Python:
*.py[cod]
*.so
*.egg
*.egg-info
dist
build

#My configurations:
db.ini
deploy_key_rsa
```

最后一步就是把 `.gitignore` 也提交到Git。

- 如果想添加被忽略文件，可以用 `-f` 强制添加到Git： `git add -f APP.class` 。
- 或者发现，可能是 `.gitignore` 写得有问题，需要找出来到底哪个规则写错了，可以用 `git check-ignore` 命令检查。
- 也可以添加例外规则：

```
#不排除.gitignore和APP.class
!.gitignore
!APP.class
```

Repo-to-repo collaboration

📌 git push

- `git push` 命令用于将本地存储库内容上传到远程存储库。
- 如果在之前的“初始化新代码库”部分中使用了 `git clone` 来设置本地代码库，那么代码库已经配置为进行远程协作。`git clone` 将自动配置代码库，并远程指向所克隆的代码库的 Git URL。这意味着，对文件进行变更并提交这些变更后，可以使用 `git push` 将这些变更推送到远程代码库。
- 如果使用 `git init` 来创建新的代码库，将没有远程代码库来接收推送的变更。在初始化新代码库时，常见的模式是前往托管的 Git 服务（如 Bitbucket），并在那里创建一个代码库。该服务将提供一个 Git URL，可以在之后将其添加到本地 Git 代码库，并使用 `git push` 推送到托管的代码库。

git push用法

`git push <remote> <branch>`：将指定的分支及所有必要的commit和内部对象推送到remote。这将在目标存储库中创建一个本地分支。

删除远程分支

要完全删除一个分支，需要在本地和remote都将其删除：

`git branch -D branch_name`

`git push origin :branch_name`

将带有冒号前缀的分支名称传递给 `git push` 会删除该远程分支。

git pull

- `git pull` 命令用于从远程存储库获取和下载内容，并立即更新本地存储库以匹配该内容。`git pull` 命令实际上是另外两个命令的组合，即执行 `git fetch` 后再执行 `git merge`。
- `git pull` 在一般情况下会从本地和主分支分歧的点下载所有变更，再获取分歧点远程commit，然后创建一个新的本地合并commit。
- 可以将 `--rebase` 选项传递给 `git pull`，以使用变基合并策略，复制远程commit并重写本地commit，以使远程commit出现在本地origin/main历史commit中。

常用选项：

1. `git pull <remote>`：默认调用。
2. `git pull --no-commit <remote>`：与默认调用类似，获取远程内容但不创建新的合并commit。

3. `git pull --verbose`：在拉取过程中提供详细输出，显示正在下载的内容和合并的详细信息。

实际上，使用 `--rebase` 拉取是一个常见的工作流程，因此有一个专门的配置选项：

```
git config --global branch.autosetuprebase always
```

Configuration & set up

📍 git remote

- 设置好远程代码库后，需要将远程代码库 URL 添加到本地 `git config`，并为本地分支设置一个上游分支。`git remote` 命令提供此功能:

```
git remote add <remote_name> <remote_repo_url>
```

- 此命令会将 `<remote_repo_url>` 的远程存储库映射到本地代码存储库中 `<remote_name>` 下的 ref。映射远程代码库后，可以将本地分支推送到远程代码存储库：

```
git push -u <remote_name> <local_branch_name>
```

git config

配置远程代码库 URL 之后，还需要设置全局 Git 配置选项，如用户名或电子邮件。使用 `git config` 命令可从命令行配置 Git 安装（或单独的代码库）。此命令可以定义从用户信息到首选项，再到代码库行为的所有内容。

Git 将配置选项存储在三个单独的文件中，可以将选项分配给各个代码库（local）、用户（global）或整个系统（system）：

> - 本地：`/.git/config` – 代码库特定设置。
> - 全局：`/.gitconfig` – 用户特定设置。这是存储使用 --global 标记设置的选项的地方。
> - 系统：`$(prefix)/etc/gitconfig` – 系统范围的设置。

使用 `git config` 创建快捷指令：

```
git config --global alias.ci commit
```

该命令创建了一个 `ci` 命令来表示 `git commit`。

Inspecting a repository

✍️ git status

`git status` 命令显示工作目录和暂存区域的状态，列出哪些文件已暂存、未暂存和未追踪。

git log

`git log` 命令显示已提交的快照。可以使用该命令来列出项目历史记录、对其进行筛选并搜索特定的变更。

使用例：

1. `git log`：使用默认格式显示整个commit历史记录，使用 `space` 滚动并使用 `q` 退出。
2. `git log -n <limit>`：限制显示的commit个数。
3. `git log --oneline`：将每个commit压缩成一行。
4. `git log --stat`：包括哪些文件被修改和每个文件中添加或删除的相对行数等信
5. `git log -p`：显示代表每次提交的补丁。
6. `git log --author="<pattern>"`：显示特定作者的commit。
7. `git log --grep="<pattern>"`：显示特定提交信息的commit。
8. `git log <since>..<until>`：仅显示从since到until之间的commit，这两个参数可以是提交ID、分支名称、 `HEAD` 或任何其他类型的修订版本引用。
9. `git log <file>`：仅显示包含指定文件的commit。

可以将多个选项组合成一个命令：

`git log --author="John Smith" -p hello.py`

显示John Smith对hello.py文件所做的所有变更的完整比对。

Undoing Commits & Changes

Reviewing old commits

1. 使用 `git log --branches=*` 来查看所有分支的所有commit。
2. 找到要访问的历史点点commit引用后，使用 `git checkout` 命令来访问该提交。
3. 要继续开发，需要回到项目的当前状态： `git checkout main` 。

4. 返回main分支后，可以使用 `git revert` 或 `git reset` 来撤销任何不需要的变更。

Undoing a committed snapshot

1. 使用 `git checkout` 命令签出之前的commit，将repo置于要撤销的commit之前的状态。
2. 执行 `git checkout -b new_branch` 。这将创建一个名为new_branch的新分支并切换到该状态，该repo现已进入新的历史记录时间线，要撤销的commit已经不存在。

使用git revert撤销commit

执行 `git revert head` 命令，Git将根据最后一次commit的反转创建一个新提交，这将在当前分支的历史记录中添加新的提交，可以使用原本的分支继续开发，但是会留下Git历史记录。

使用git reset撤销commit

调用 `git reset --hard ale8fb5` 命令，commit历史记录将重置为指定的commit。这种重置对于本地变更非常有用，但是在使用远程repo时，Git会假设被推送的分支不是最新的，因为它缺少commit。

撤销上次commit

执行 `git commit --amend` 可以修改最近的一次commit。

撤销公共变更

撤销共享历史记录的首选方法是 `git revert` 。还原比重置更安全，因为它不会移除任何commit。

Making a Pull Request

Feature Branch Workflow With Pull Requests

- feature branch workflow使用共享的bitbucket代码库来管理协作，开发者在隔离的分支中创建功能。
- 但是开发者不立即将分支合并到main中，而是打开一个pull request，围绕该功能进行讨论，再集成到主代码库中。
- 只有一个公共代码库，因此pull request的目标存储库和源存储库总是一致的。
- 通常将开发者的功能分支指定为源分支，将main分支指定为目标分支。
- 也可以为不完整的功能commit提交pull request，寻求建议。

Forking Workflow With Pull Requests

- 开发者将已完成的功能推送到自己的公有代码库而不是共享代码库。之后提出pull request让项目维护者知道可以进行审查。

- pull request的源代码库不同于目标代码库。源代码库是开发者的公有代码库，源分支是包含建议变更的分支。目标存储库是正式项目，目标分支是main。
- pull request还可用来与其他开发者协作，以另一开发者的代码库作为目标进行pull request。

Using Branches

- 一个分支代表一个独立的发展思路。
- 分支可用作编辑/阶段/commit过程的抽象化。

💡 git branch

该命令允许创建、列出、重命名和删除分支。不允许在分支间切换，也不能将拷贝的历史记录重新组合在一起。

常用选项：

1. `git branch`：列出存储库中的所有分支，是`git branch --list`的同义命令。
2. `git branch <branch>`：创建一个名为<branch>的新分支。
3. `git branch -d <branch>`：删除指定的分支。
4. `git branch -D <branch>`：强制删除指定的分支。
5. `git branch -m <branch>`：将当前分支重命名为<branch>。
6. `git branch -a`：列出所有远程分支。

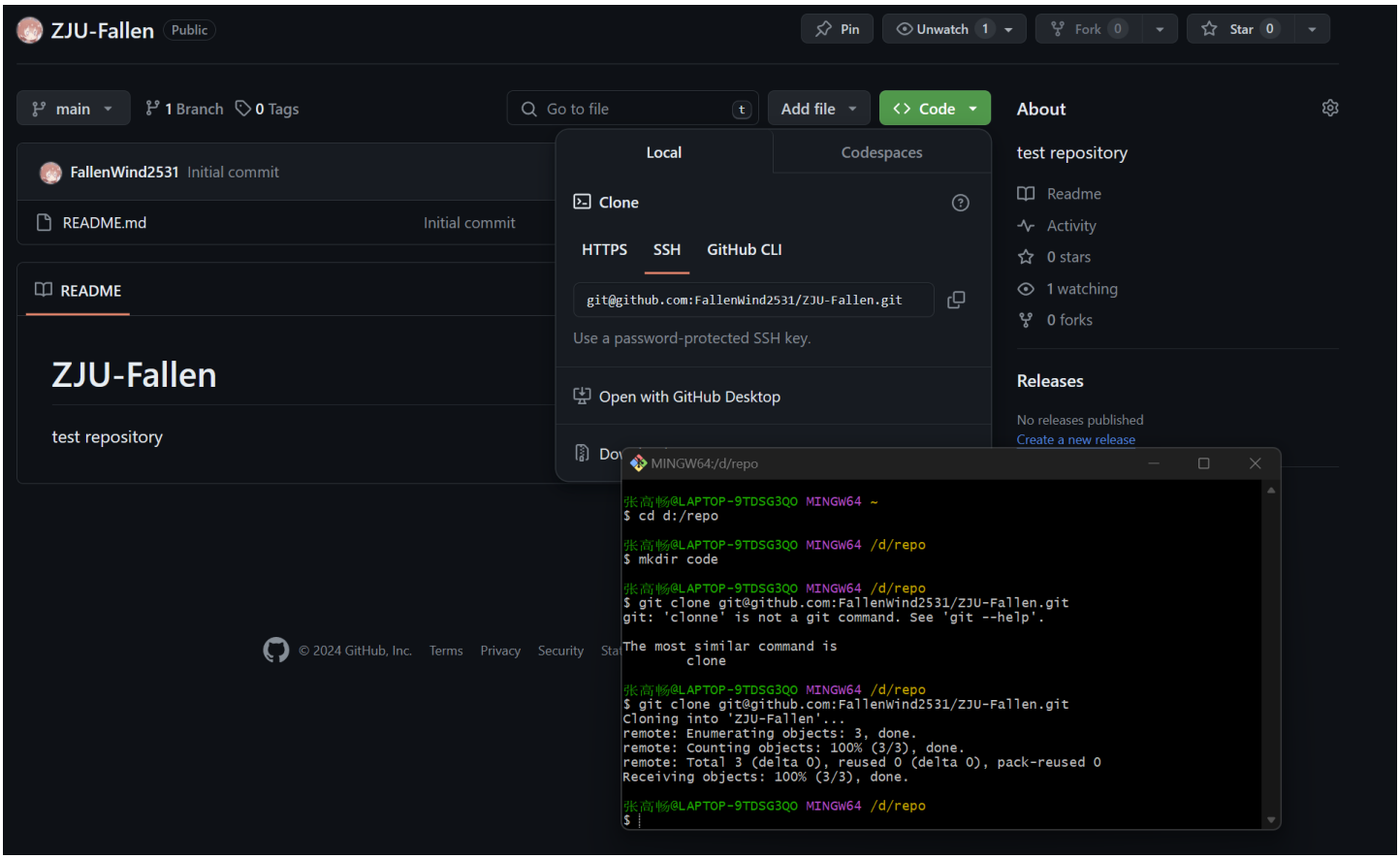
创建分支

- 创建分支后，只会得到一个指向当前提交的指针。
- 需要使用`git checkout`将其选中，然后才能在该分支上继续开发。

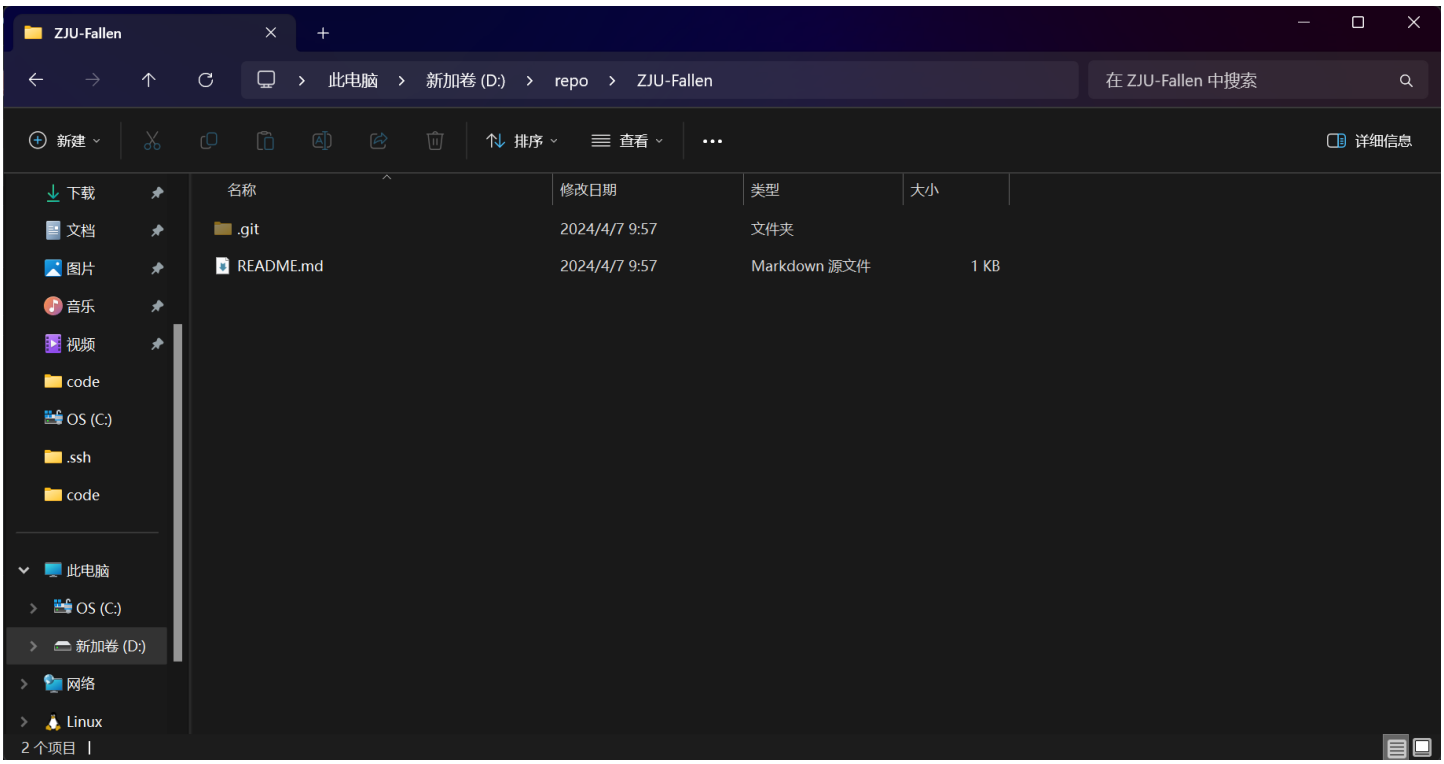
删除分支

- 完成分支工作并合并到主代码库之后，可以自由删除该分支而不会丢失历史记录。

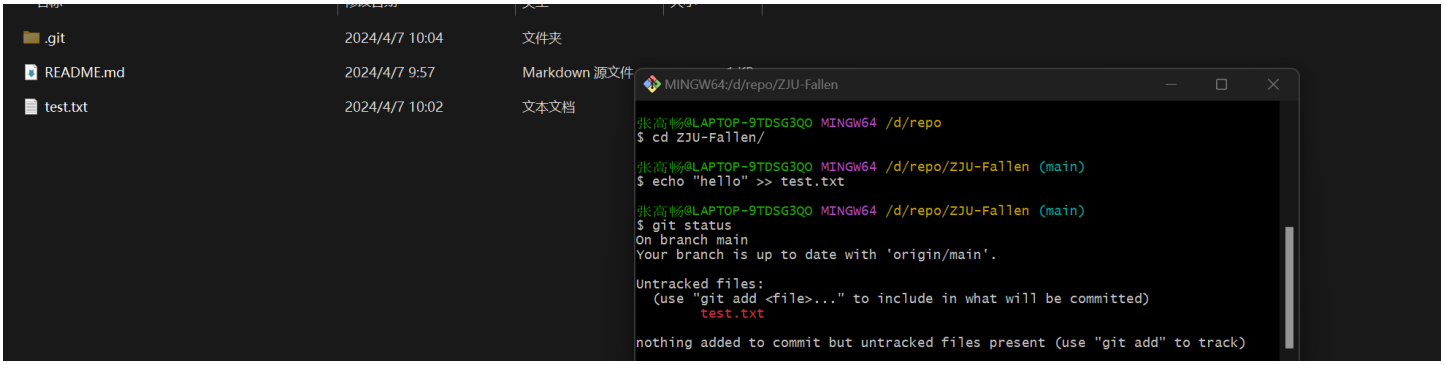
实际测试



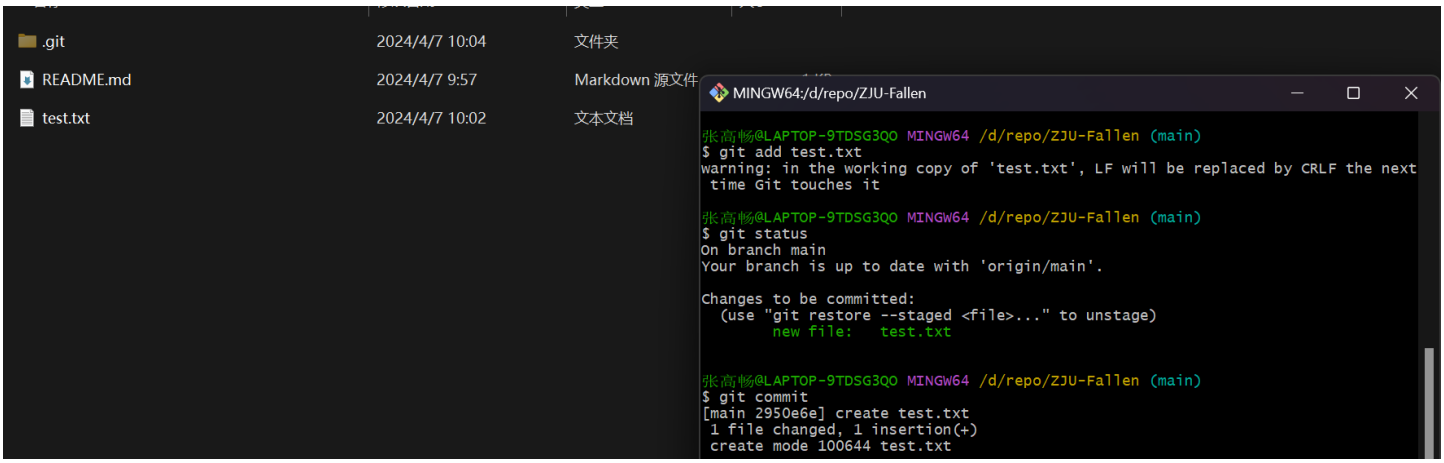
- 在GitHub创建repo后clone到本地。



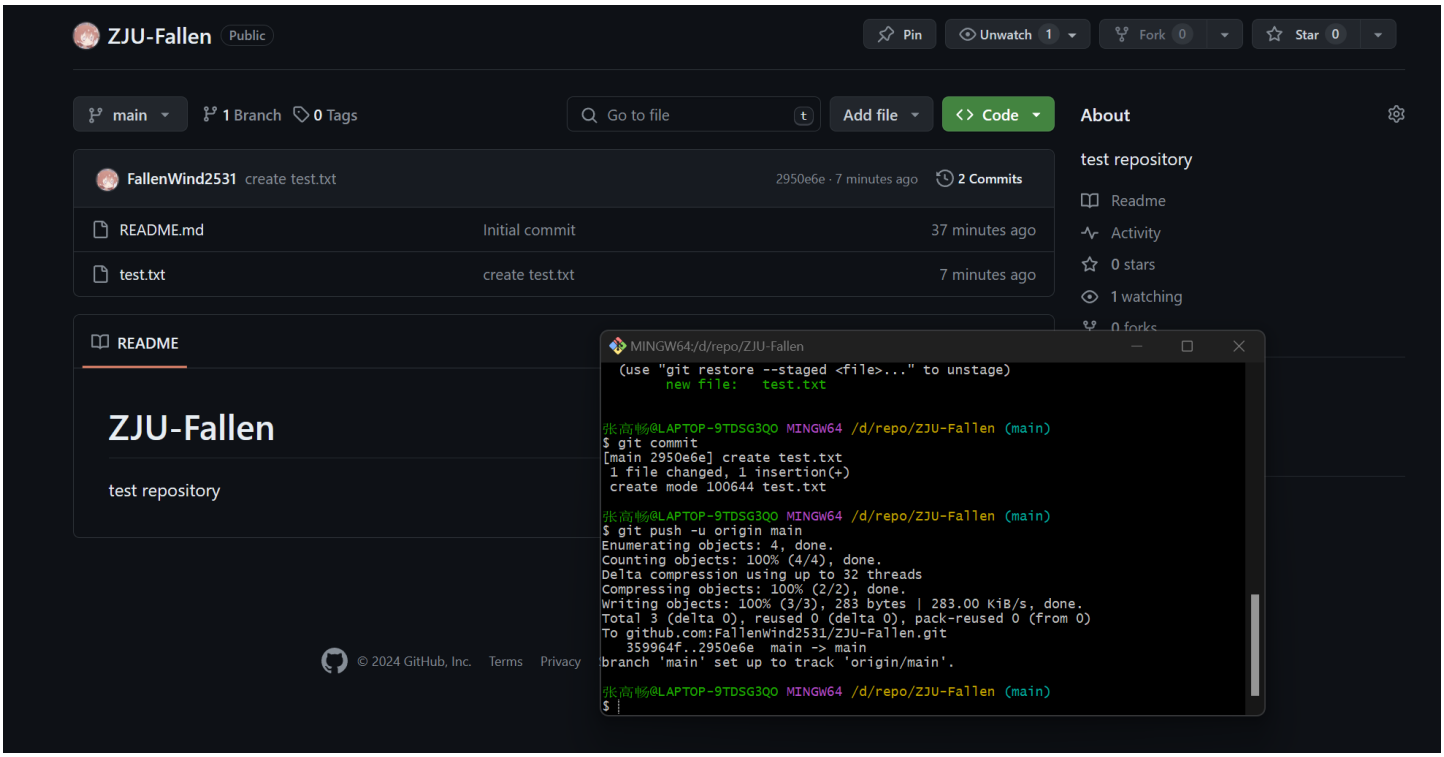
- 本地初始情况。



- 创建test.txt并输入hello，使用 `git status` 检查状态。



- 使用 `git add` 添加test.txt到暂存区，使用 `git status` 再检查状态，并用 `git commit` 提交。



- 使用 `git push` 将变更上传到GitHub。