实验报告 1：版本控制（Git）

方欣 23020007021

Contents

1 github 链接

github 链接：https://github.com/nixgnaf/SDT_report.github.io

2 实验目的

Git 是一个开源的版本控制系统，主要用于源代码管理。它能够帮助开发者跟踪文件的更改历史，协调多人合作开发，并处理分支和合并等任务。在 Git 中，文件的状态可以分为三种：已修改（modified）、已暂存（staged）和已提交（committed）。这使得 Git 项目拥有三个主要阶段：工作区、暂存区和 Git 目录。

- 已修改表示当文件在工作区被修改但还未保存到数据库中时；
- 已暂存表示这些更改被标记为将要包含在下一次提交中；
- 已提交表示文件的更改被安全地保存到本地仓库的版本历史中；

Git 的常规操作流程包括：首先使用 `git clone` 克隆远程仓库到本地，接着在本地进行代码修改，通过 `git add` 将更改添加到暂存区，使用 `git commit` 提交更改到本地仓库。如果需要与远程仓库同步，则使用 `git pull` 拉取远程更改，解决任何可能的冲突，然后使用 `git push` 将本地提交推送到远程仓库。

本实验旨在初步掌握常见 Git 命令的基本用法，熟悉如何初始化仓库、管理文件、分支操作以及与远程仓库的交互。

3 实验环境

Git Bash：提供 Unix 风格的命令行工具。版本：Git-2.46.0-64-bit
操作系统：Windows 11

4 实验内容

4.1 git clone

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

git clone: 用于克隆远程仓库的 git 命令，将远程仓库的内容复制到本地，并创建一个与远程仓库同步的本地仓库副本。
https://github.com/missing-semester-cn/missing-semester-cn.github.io: 指向一个托管在 GitHub 上的 Git 仓库的位置.

4.2 git remote add origin

```
1 git remote add origin https://github.com/missing-semester-cn/missing-semester-cn.
    github.io
```

```
nixgnaf@niXgnaF MINGW64 ~
$ git clone https://github.com/missing-semester-cn/missing-semester-cn.github.io
Cloning into 'missing-semester-cn.github.io'...
remote: Enumerating objects: 3194, done.
remote: Counting objects: 100% (3194/3194), done.
remote: Compressing objects: 100% (1126/1126), done.
remote: Total 3194 (delta 2040), reused 2735 (delta 2033), pack-reused 0 (from 0)
Receiving objects: 100% (3194/3194), 15.44 MiB | 1.26 MiB/s, done.
Resolving deltas: 100% (2040/2040), done.
```

这个命令将指定的远程仓库地址与本地仓库关联，使得可以进行远程操作，如推送和拉取代码.
git remote: 这是 Git 命令的一个子命令，用于管理远程仓库。
add: 这是 git remote 子命令中的一个操作，表示添加一个新的远程仓库。在默认情况下，Git 使用 origin 作为远程仓库的默认名称。

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git remote add origin https://github.com/missing-semester-cn/missing-semester-cn.github.io

nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git remote -v
origin  https://github.com/missing-semester-cn/missing-semester-cn.github.io (fetch)
origin  https://github.com/missing-semester-cn/missing-semester-cn.github.io (push)
```

4.3 git push origin

```
1 git push origin master
```

git push：将本地仓库的更改推送到远程仓库
master: 这是本地分支的名称

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io/missing-semester-cn.github.io (master)
$ git log -1
commit 63ed43f8a998791efe04b69546a666db91772a90 (HEAD -> master)
Author: Nixgnaf <2735274779@qq.com>
Date:   Thu Aug 29 13:49:59 2024 +0800

    Add a new file

nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io/missing-semester-cn.github.io (master)
$ git push origin master
warning: ----------------- SECURITY WARNING ----------------
warning: | TLS certificate verification has been disabled! |
warning: ---------------------------------------------------
warning: HTTPS connections may not be secure. See https://aka.ms/gcm/tlsverify for more information.
warning: ----------------- SECURITY WARNING ----------------
warning: | TLS certificate verification has been disabled! |
warning: ---------------------------------------------------
warning: HTTPS connections may not be secure. See https://aka.ms/gcm/tlsverify for more information.
Enumerating objects: 4, done.
```

4.4 git log –all –graph –decorate

```
1 git log --all --graph  - decorate
```

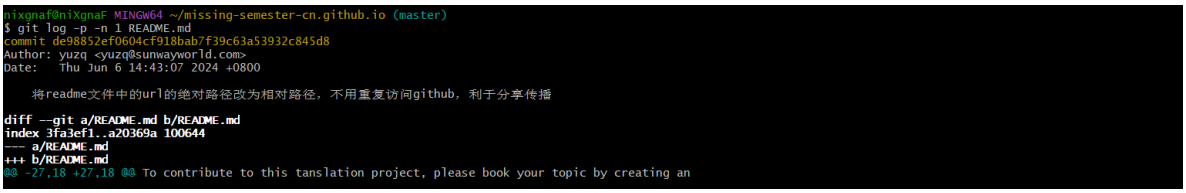
git log: Git 用于查看提交历史的基本命令
–all: 显示所有分支的提交历史, 确保看到仓库中的所有提交，不论它们属于哪个分支
–graph: 以图形化的方式显示提交历史。这个选项会在日志的左侧添加一个 ASCII 图形，用于可视化分支和合并的历史结构

显示分支、标签和其他引用的名称。这个选项会在每个提交记录旁边显示当前分支和标签的位置，例如图中: (HEAD -> master, origin/master, origin/HEAD)

4.5 git log -p -n 1

```
git log -p -n 1 README.md
```

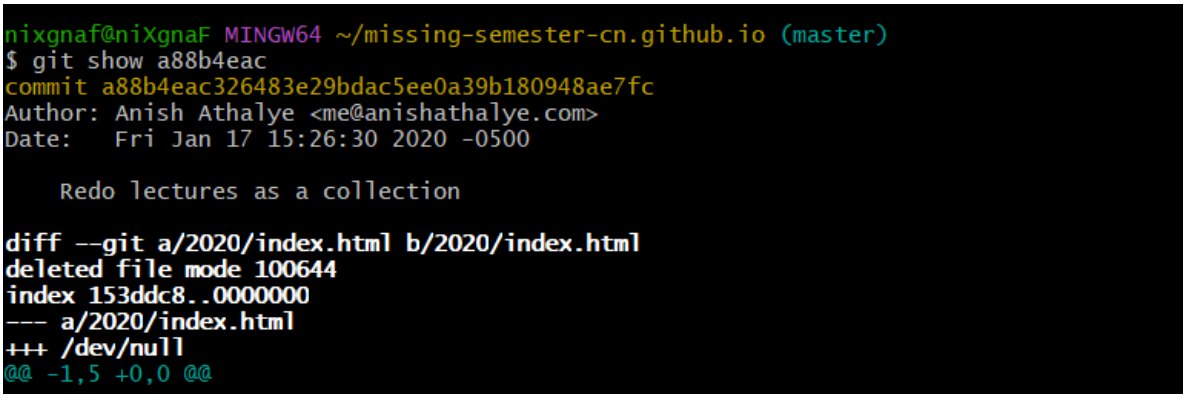
-p：显示提交的详细变更内容（差异），即显示每个提交对 README.md 文件所做的具体修改。
-n 1：限制输出显示的提交数量，表示只显示最近的一次提交。



4.6 git show a88b4eac

```
git show a88b4eac
```

git show: 默认情况下，这个命令显示提交 a88b4eac 的详细信息，包括提交哈希、作者、日期、提交消息，以及该提交所做的所有更改（文件差异）。



4.7 git show –pretty=format:”%s” a88b4eac | head -1

```
git show --pretty=format:"%s" a88b4eac | head -1
```

git show –pretty=format:”%s”: 这个命令用于自定义格式化输出。–pretty=format:”%s” 选项只会显示提交的消息（即提交说明）。
a88b4eac: 是提交的哈希值。
head -1: 只显示第一行。

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git show --pretty=format:"%s" a88b4eac | head -1
Redo lectures as a collection
```

4.8 git log –pretty=format:”%s” a88b4eac -1

```
git log --pretty=format:"%s" a88b4eac -1
```

–pretty=format:”%s”: 自定义输出格式。
-1: 限制输出到最新的 1 个提交。

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git log --pretty=format:"%s" a88b4eac -1
Redo lectures as a collection
```

4.9 git add .

```
git add .
```

将当前目录下的所有更改（包括新文件和修改的文件）添加到暂存区，以便进行提交。. 代表当前目录。

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git remote add origin https://github.com/missing-semester-cn/missing-semester-cn.github.io

nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git remote -v
origin  https://github.com/missing-semester-cn/missing-semester-cn.github.io (fetch)
origin  https://github.com/missing-semester-cn/missing-semester-cn.github.io (push)
```

4.10 git commit -m ”add password123 to file”

```
git commit -m "add password123 to file"
```

Git commit: 于将暂存区的更改记录到本地版本库中，生成一个新的提交
-m: 用于指定提交消息。可在命令行中直接提供提交消息
”add password123 to file”: 这是提交消息，描述这次提交所做的更改。消息应该简明扼要地说明提交的内容和目的

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git commit -m "add password 123 to file"
[master 73387de] add password 123 to file
 1 file changed, 1 insertion(+)
 create mode 100644 my_password
```

4.11 git status

```
git commit -m "add password123 to file"
```

用于查看当前工作目录和暂存区的状态。它提供了关于当前分支的状态、哪些文件被修改或尚未提交、以及其他有关版本控制的信息。

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ echo 'new file' > file.txt

nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        git-filter-repo
        git-filter-repo-1/
```

4.12 git config –global core.excludesfile /.gitignore

```
git config --global core.excludesfile ~/.gitignore
```

这条命令告诉 Git 在用户主目录下使用.gitignore 文件作为全局忽略文件。

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git config --global core.excludesfile
C:/Users/nixgnaf/.gitignore
```

4.13 git stash pop

```
git stash pop
```

git stash pop 命令用于将之前存储在 Git 暂存区中的更改（即 stash 中的内容）恢复到工作目录，并从 stash 中移除相应的记录。

当你暂时存储了当前工作进度（使用 git stash），然后在完成其他任务后需要恢复这些进度时，可以使用 git stash pop。

从 main 分支切换到 dog 分支，再将存储恢复，然后提交，这时，我们刚才新建的 file.txt，变成了 dog 分支下的一次新提交。

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (dog)
$ git stash pop
On branch dog
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        git-filter-repo
        git-filter-repo-1/

Dropped refs/stash@{0} (7347312d2346430c60e21ff96c4857c3ac9fd3d6)

nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (dog)
$ git status
On branch dog
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        git-filter-repo
        git-filter-repo-1/
```

4.14 [alias] graph = log –all –graph –decorate –oneline

```
1 [alias]graph = log --all --graph --decorate  - oneline
```

在 Git 的配置文件 /.gitconfig 中，[alias] 部分用于定义 Git 命令的别名。
通过设置这个别名，可以使用 git graph 代替更长的命令 git log –all –graph –decorate –oneline。

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (dog)
$ git log -graph
fatal: unrecognized argument: -graph

nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (dog)
$ git log --graph
* commit f6cb32770cf2b049c29bb32b2db958d98150fbd6 (HEAD -> dog)
| Author: Nixgnaf <2735274779@qq.com>
| Date:   Sun Aug 25 14:11:01 2024 +0800
|
|     Add file.txt on branch dog
|
*   commit d50a857ac3ff8fa5d6da6ae101dd76bdb94e21ad (master)
|\  Merge: 7534662 1eecd59
```

```
[user]
        name = Nixgnaf
        email = 2735274779@qq.com
[http]
        sslVerify = false

[alias]
        graph = log --all --graph --decorate --oneline
~
```

4.15 git filter-branch

```
1 git filter-branch --force --index-filter\
2  'git rm --cached --ignore-unmatch ./my_password' \
3  --prune-empty --tag-name-filter cat -- --all
```

这个命令的目的是从所有 Git 历史记录中的每个提交中删除文件./my_password，并清理那些因为文件被删除而变得空的提交。

git filter-branch: 主命令，用于重写 Git 历史记录。允许你历史记录进行各种修改，例如删除文件、替换文件内容、修改提交信息等。

（使用 git filter-repo 代替实现功能）

4.16 git filter-repo –path my _password –invert-paths

```
git filter-repo  – path my_password  – invert-paths
```

git-filter-repo 会重写仓库的历史记录，这可能包括删除某些文件、提交或者其他信息。
–path path/to/file 指定你要删除的文件路径。
–invert-paths 指定删除指定路径以外的所有内容。

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ python git-filter-repo --force --path my_password --invert-paths
NOTICE: Removing 'origin' remote; see 'Why is my origin removed?'
        in the manual if you want to push back there.
        (was https://github.com/missing-semester-cn/missing-semester-cn.github.io)
Parsed 833 commitsHEAD is now at d50a857 Merge pull request #172 from pspdada/master
Enumerating objects: 3194, done.
Counting objects: 100% (3194/3194), done.
Delta compression using up to 12 threads
Compressing objects: 100% (1121/1121), done.
Writing objects: 100% (3194/3194), done.
Total 3194 (delta 2038), reused 3194 (delta 2038), pack-reused 0 (from 0)

New history written in 0.48 seconds; now repacking/cleaning...
Repacking your repo and cleaning out old unneeded objects
Completely finished after 1.46 seconds.
```

4.17 git blame _config.yml | grep collections

```
git blame _config.yml | grep collections
```

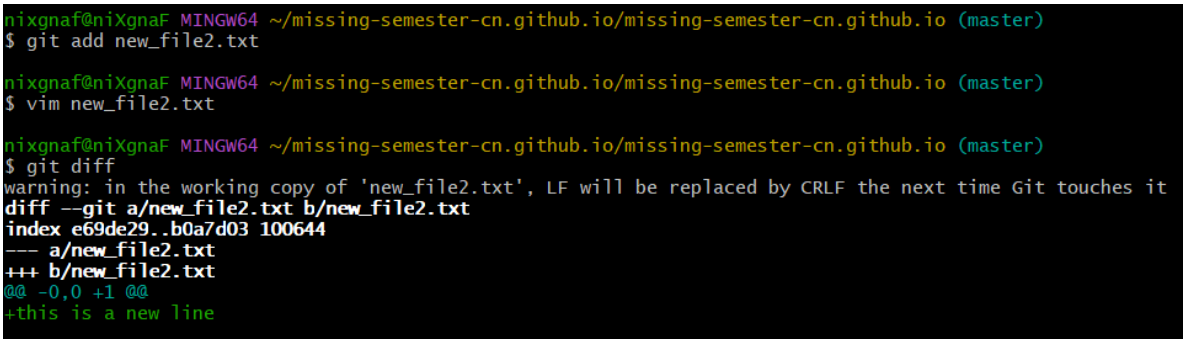
git blame _config.yml: 显示 _config.yml 文件的每一行的最后修改记录，包括修改者、修改时间和提交哈希。每行前面都会标记出最后修改这行的提交信息。
grep collections: 从输入中筛选出包含“collections”字符串的行。

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git blame _config.yml | grep collections
a88b4eac (Anish Athalye  2020-01-17 15:26:30 -0500 18) collections:
```

4.18 git diff

```
1 git diff
```

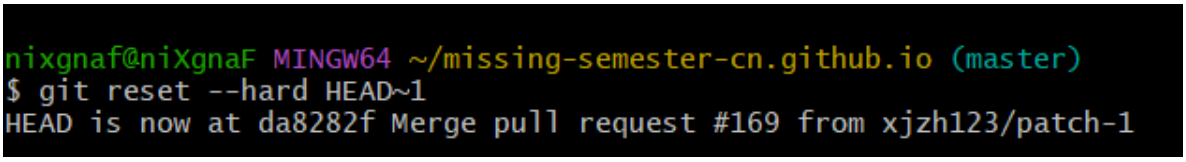
git diff: 显示已写入暂存区和已经被修改但尚未写入暂存区文件的区别



4.19 git reset –hard HEAD 1

```
1 git reset --hard HEAD~1
```

用于重置当前分支到上一个提交的状态，并且丢弃所有未提交的更改。
git reset：用于将当前分支的 HEAD 指针移动到指定的提交，并调整暂存区和工作目录的状态。
–hard：这个选项告诉 Git 丢弃工作目录和暂存区中的所有更改。
指定了要重置到的目标提交。HEAD 表示当前分支的最新提交，而 HEAD 1 表示上一个提交（即当前提交的父提交）。



4.20 git config –list –global

```
1 git config --list --global
```

用于列出全局配置中的所有配置项。
git config：这是 Git 的配置命令，用于查看和修改 Git 配置文件中的设置。
–list：这个选项用于列出所有当前配置项的名称和值。可以与 –global、–system 或 –local 选项一起使用，以指定查看配置的范围。

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git config --list --global
user.name=Nixgnaf
user.email=2735274779@qq.com
http.sslverify=false
alias.graph=log --all --graph --decorate --oneline
core.excludesfile=C:/Users/nixgnaf/.gitignore
core.excludesfile=C:/Users/nixgnaf/.gitignore
```

4.21 git branch dog

```
1 git branch dog
```

git branch dog: 创建新分支 dog

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git branch dog
```

4.22 git checkout dog

```
1 git checkout dog
```

切换到新分支 dog

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git checkout dog
Switched to branch 'dog'
```

4.23 git branch -a -v

```
1 git branch -a -v
```

列出当前仓库的所有本地分支
-a: 显示所有分支，包括本地分支和远程分支
-v: 显示每个分支的最新提交摘要

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git branch
  dependabot/bundler/activesupport-6.0.6.1
  dependabot/bundler/addressable-2.8.1
  dependabot/bundler/nokogiri-1.14.3
  dependabot/bundler/tzinfo-1.2.10
  dog
* master
  review
```

4.24 git merge dog

```
1 git merge dog
```

将一个分支 dog 的更改合并到当前分支。

```
nixgnaf@niXgnaF MINGW64 ~/missing-semester-cn.github.io (master)
$ git merge dog
Updating d50a857..f6cb327
Fast-forward
 file.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt
```

5 实验心得

通过本次实验，我对 Git 的工作原理有了初步理解，包括在仓库管理、提交历史查看、工作目录和暂存区操作、分支操作、配置管理等方面的基础功能，也增强了对版本控制工具在团队协作中的重要性的认识，同时进一步接触 Unix 系统下基于 Bash 语言的命令行操作。在实验报告的撰写过程中，深刻感受到 Latex 的精确性和专业性。