Git

非常抱歉，排版太混乱了

为啥要用这东西？我直接压缩包打包不好吗？

1. 版本控制

- **历史记录**：Git 会记录每次提交的修改，可以随时查看文件的历史版本，了解每个改动的内容、时间和作者。
- **回滚功能**：如果某个修改导致问题，可以轻松回滚到之前的版本，避免文件损坏或丢失。
- **分支管理**：可以创建不同的分支来尝试不同的功能或方案，而不会影响主分支的稳定性。

2. 协作效率

- **实时同步**：小组成员可以随时拉取最新的代码或文档，避免重复劳动或版本冲突。
- **冲突解决**：当多人同时修改同一文件时，Git 会提示冲突，并提供工具帮助解决冲突，避免文件覆盖或丢失。
- **分工明确**：通过分支和提交记录，可以清晰地看到每个人的贡献和进度。

就我个人的体验来说，确实很有用。本bug制造大师在写rcore的时候，经常制造出一系列的bug，多亏有这东西保命，撤销错误的更改

在小组作业中，总有一些人喜欢摸鱼，摆烂，求别人带。。。通过版本控制系统，可以比较方便的查看贡献量，方便踢人。就算踢不了，也方便制作成合订本让人笑话。

我得诚实的说明，这里面的东西，基本来自于git的文档，我自己删减了一点。所以说，如果可以，我更建议你直接看文档

官方文档：https://git-scm.com/docs

本文部分地方的文本可能会有各种奇奇怪怪的毛病，还请见谅

```
git merge --help
man git-merge
```

```
git help merge
```

很多地方，其实用多了，慢慢就熟练起来了，光看教程没啥用. 我建议自己实践一下

如果你实在是想看这篇教程，那请继续

哦，还可以用各种人工智能助手，。。。

作业在最后，

还有，在这简单列一下涉及到的东西

```
$git help
```

start a working area (see also: git help tutorial)
  clone   Clone a repository into a new directory
  init   Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add   Add file contents to the index
  mv   Move or rename a file, a directory, or a symlink
  restore   Restore working tree files
  rm   Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
  bisect   Use binary search to find the commit that introduced a bug　这个没有讲到，作业里面有，自己研究研究咋整
  diff   Show changes between commits, commit and working tree, etc
  grep   Print lines matching a pattern
  log   Show commit logs
  show   Show various types of objects
  status   Show the working tree status

grow, mark and tweak your common history
   branch   List, create, or delete branches
   commit   Record changes to the repository
   merge   Join two or more development histories together
   rebase   Reapply commits on top of another base tip
   reset   Reset current HEAD to the specified state
   switch   Switch branches
   tag   Create, list, delete or verify a tag object signed with GPG

一些资源，你可能用得上

一个小练习

https://help.gitee.com/learn-Git-Branching/?locale=zh_CN

一些资源

Linux 入门 https://101.lug.ustc.edu.cn/

https://docs.net9.org/

https://csdiy.wiki/

https://hust.openatom.club/

https://mirrors.hust.edu.cn/

https://liaoxuefeng.com/books/git/introduction/index.html

基本概念

基本概念，由gpt-4o-mini生成

1. HEAD

- **定义**：`HEAD` 是一个指针，指向当前检出的分支的最新提交。它表示你当前的工作状态。
- **作用**：当你在 Git 中进行提交、合并或其他操作时，`HEAD` 会指向当前分支的最新提交。你可以通过 `git checkout` 命令切换到不同的分支，这时 `HEAD` 会更新为指向新分支的最新提交。
- **特殊情况**：在“分离头指针”（detached HEAD）状态下，`HEAD` 直接指向某个具体的提交，而不是分支。这通常发生在你检出一个特定的提交时。

2. Index（暂存区）

- **定义**：`index`（也称为暂存区或缓存区）是一个中间区域，用于存放即将提交的更改。它是工作目录和本地仓库之间的桥梁。
- **作用**：当你使用 `git add` 命令时，文件的更改会被添加到 `index` 中。只有在 `index` 中的更改才会被包含在下次提交中。你可以选择性地将更改添加到 `index`，这使得你可以控制哪些更改会被提交。
- **查看状态**：可以使用 `git status` 命令查看工作目录和 `index` 的状态，了解哪些文件已修改、已暂存或未跟踪。

3. Working Directory（工作目录）

- **定义**：工作目录是你在本地计算机上实际操作的文件夹，包含了项目的所有文件和目录。
- **作用**：在工作目录中，你可以编辑、添加或删除文件。所有的更改首先发生在工作目录中，然后通过 `git add` 命令将更改添加到 `index`，最后通过 `git commit` 命令将更改提交到本地仓库。

4. Repository（仓库）

- **定义**：仓库是 Git 用来存储项目版本历史的地方。它包含了所有的提交记录、分支、标签等信息。
- **作用**：仓库可以是本地的（在你的计算机上）或远程的（如 GitHub、GitLab 等）。本地仓库用于版本控制和管理，而远程仓库用于协作和备份。

5. Branch（分支）

- **定义**：分支是 Git 中用于并行开发的一个重要概念。每个分支都是一个独立的开发线，可以在上面进行修改而不影响其他分支。
- **作用**：分支允许你在不同的功能或修复上独立工作，最终可以将这些更改合并到主分支（通常是 `main` 或 `master`）。

6. Commit（提交）

- **定义**：提交是 Git 中的一个快照，记录了某一时刻的项目状态。每个提交都有一个唯一的 SHA-1 哈希值。
- **作用**：提交用于保存更改的历史记录。每次提交时，你需要提供一个提交信息，描述这次更改的内容。

安装

废话完毕，言归正传

安装

Linux

```
#Debian系列的发行版(比如Debian, Ubuntu, Linux-Mint, Deepin
$apt update
$apt install git#git-lfs不是必须的~
##额外的，如果你是Ubuntu及其衍生版的用户，可以使用以下命令获取最新的稳定版，
#PPA源里面的有些软件，可能有一些默认源里面没有的东西，比如在wsl1上运行的gdb
$add-apt-repository ppa:git-core/ppa#随后重复运行前面的命令


#使用rpm作为包管理的系统 。。。。比如Redhat, OpenEuler, Fedora
yum install git (最高至 Fedora 21)
dnf install git (Fedora 22 及更高版本)
#yum在被dnf取代，不过，直接用yum也不是不行，大部分发行版应该会做一个别名?
#其他系统？你都用其他系统了，相信你一定会自己安装吧
###https://git-scm.com/downloads/linux
```

Windows

https://git-scm.com/downloads/win 在这下载就好

基础配置

可以使用 git config来配置git。

1. /etc/gitconfig 文件: 包含系统上每一个用户及他们仓库的 通用配置。 如果使用带有 --system 选项的 git config 时，它会从此文件读写配置变量。
2. ~/.gitconfig 或 ~/.config/git/config 文件：只针对当前用 户。 可以传递 --global 选项让 Git 读写此文件。
3. 当前使用仓库的 Git 目录中的 config 文件（就是 .git/config ）：针对该仓库。

越往下，优先级越高

1.配置好你的用户信息

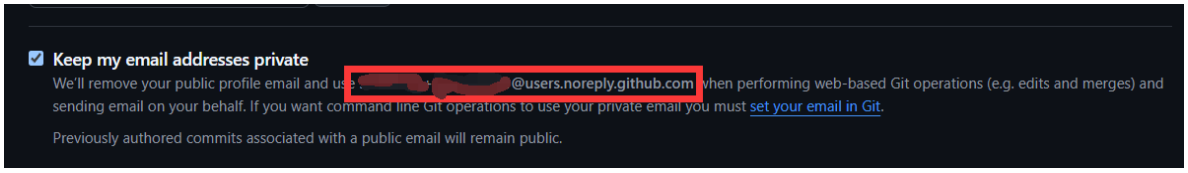
这里加上了--global参数，所以会作为当前操作系统登陆用户的默认配置

```
$ git config --global user.email 114514@gmail.com # 配置邮箱，Github使用邮件地址来关联用户与commit
```

```
$ git config --global user.name lts114514 #配置用户名
```

如果想给某个仓库单独配置？那么在其目录下，不带--global参数操作即可

如果你是GitHub用户，并且希望隐藏自己的邮箱，在https://github.com/settings/emails 可以找到noreply邮箱地址，使用这个地址来配置 user.email



通过以下命令查看git配置

```
$ git config --list
```

配置SSH

为什么使用ssh？GitHub已经关闭了密码验证，相比于其他的方式而言，我觉得SSH可能方便配置

生成新 SSH 密钥

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"#按照所给提示来操作
```

添加到ssh-agent

```
$ eval "$(ssh-agent -s)"
$ ssh-add ~/.ssh/id_ed25519
```

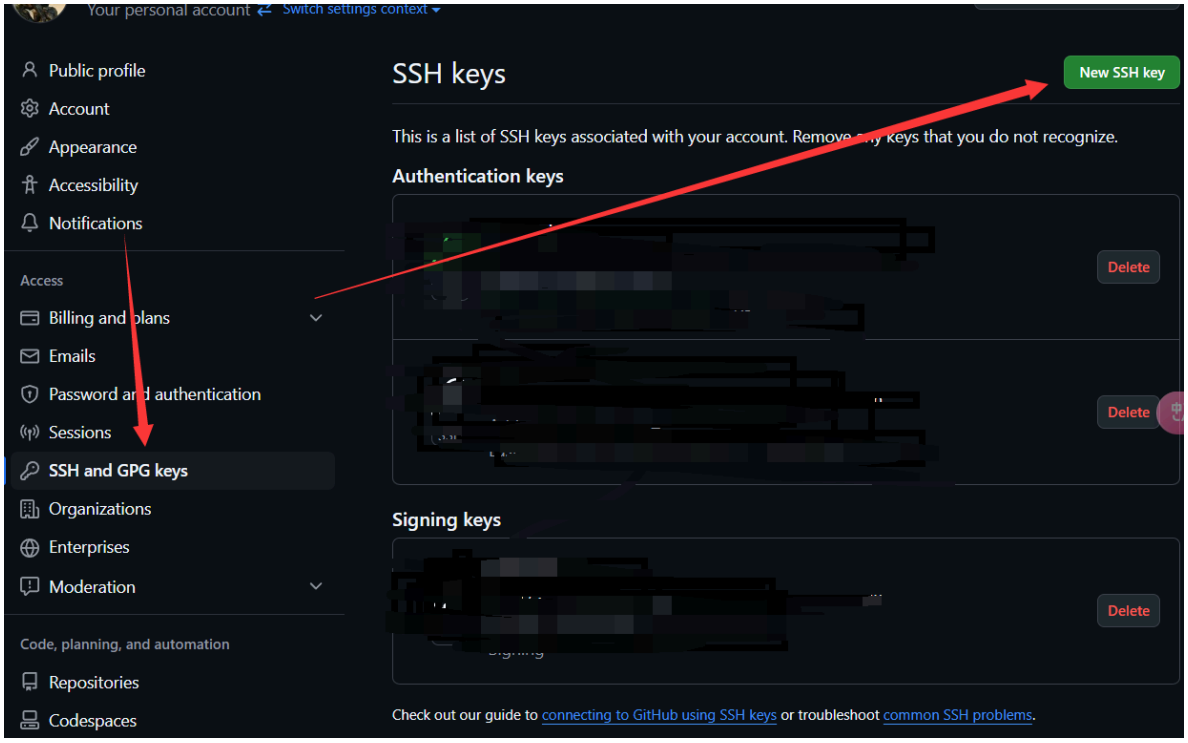
如果你在前面没有指定用哪个密钥的话，那么，这个文件的名字是id_ALGORITHM，比如id_ed25519

添加到GitHub

```
$ cat ~/.ssh/id_ed25519.pub
```

https://github.com/settings/keys

在这里面，将你的公钥添加进来



通过以下命令来尝试与服务器进行连接

```
$ssh -T git@github.com
```

需要注意，由于很多不可明说的原因，你可能会遇上一堆连接问题

这是一些建议

1.这些问题，除了中国大陆地区，朝鲜，还有某些中亚国家以外 很少遇上

2.Linux下面，配置代理的环境变量如下

```
$export https_proxy=<URL>
$export all_proxy=<URL>
$export http_proxy=<URL>
#你也可以把这些玩意加到一个文本文件里面，需要的时候
$source set_proxy
#也可以直接加到 ~/.bashrc 文件中，在启用shell的时候就设置 我这只是举了一个例子，其他shell改这个可能没用
#我没有鼓励任何人干任何违法的事情:-)
```

如果你使用的网络服务，封锁了22端口，可以修改你的ssh客户端配置

在这: ~/.ssh/config

下面是我自己的配置

```
Host github.com
        Hostname ssh.github.com
        Port 443
        User git
        ProxyCommand nc -v -x 127.0.0.1:10808 %h %p
```

未命名文档

* `git status`：查看当前状态，并给出提示 (**非常常用！很多时候 Git 的提示非常贴心，会告诉你相应状态和接下来应当做的操作**)；
* `git add`：工作区 (worktree) → 暂存区 (index/staged)
* `git commit`：暂存区 (index/staged) → 版本库 (HEAD)
* `git restore`：恢复文件
* `git diff`：显示版本间变化

更新后的图：

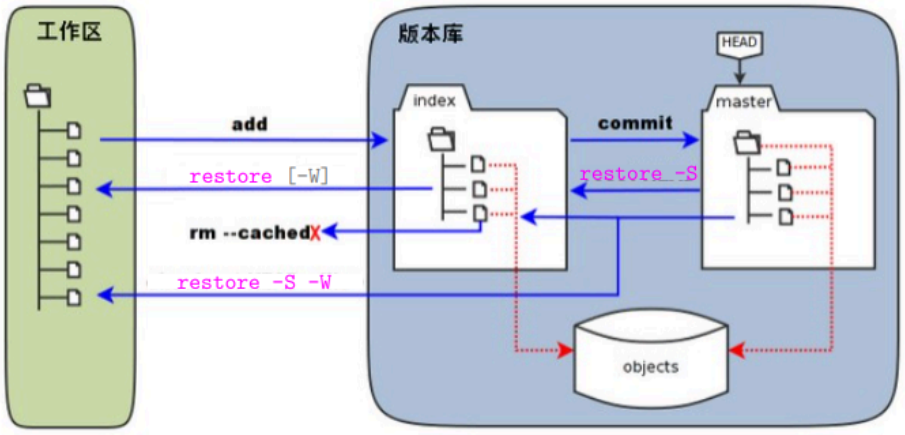


图 1: Git 工作区、暂存区和版本库

从清华酒井协会偷来的图，在此表示感谢

下面部分演示，记得补上一些截图

啥是HEAD？当前分支的最新提交

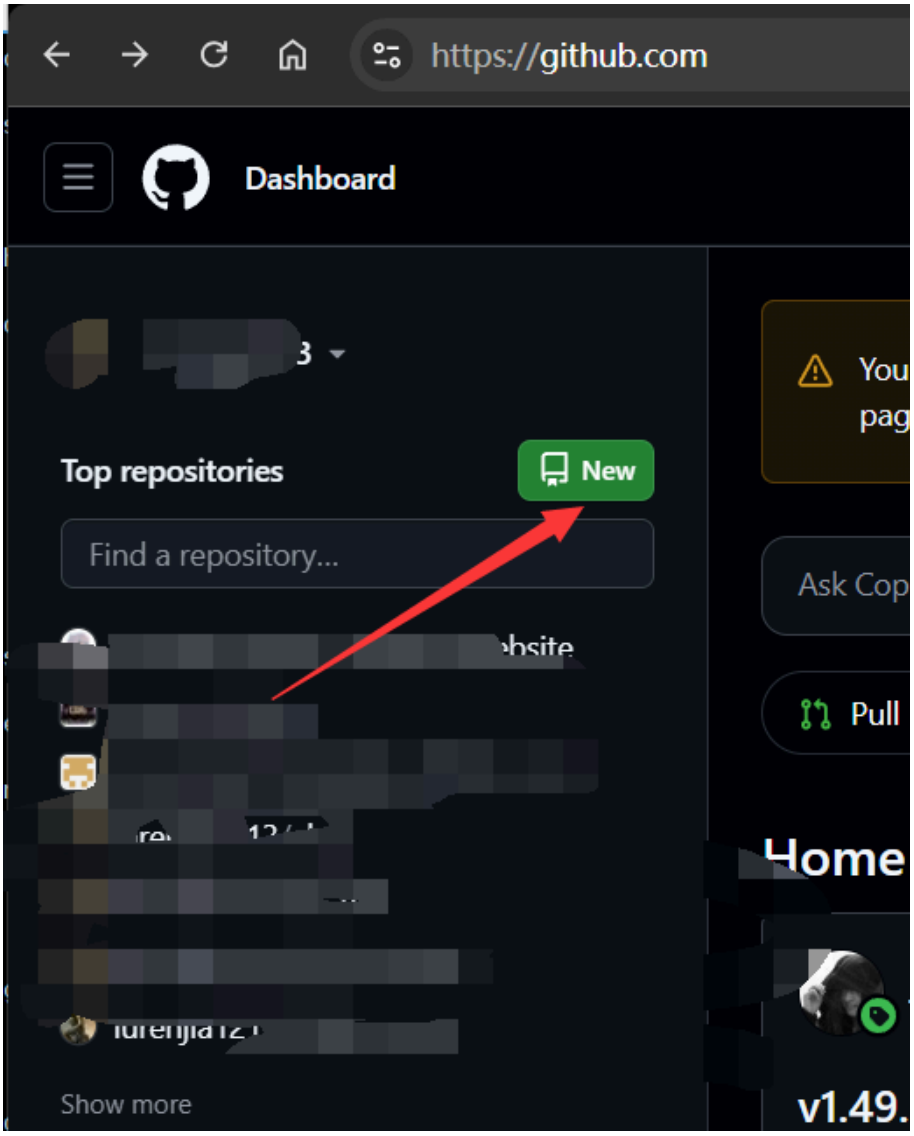
start a working area

创建本地仓库

```
$mkdir <name>#创建目录
$cd <name>
$git init #  也可以 git init <name>，然后进入这个目录
Initialized empty Git repository <name>
```

在GitHub上创建仓库:

看图



拉取已有远程仓库

```
$ git clone <repo> <dir># repo:仓库地址，dir:目录，如果没有dir，那么将在当前目录下以仓库名字创建
```

---

work on the current change

使用git add，将修改的文件加入暂存区，这样，Git 就会开始追踪该文件的更改，并将其纳入版本控制。

```
$ git add *.c *.h #添加所有
$ git add main.rs
```

你可以用git rm从暂存区，还有工作区删除已经被追踪的文件

```
$ git rm <file>
```

还有git mv，自己了解吧

恢复文件

```
$ git restore --source master~2 Makefile #从master分支的上上一个提交恢复Makefile
$ rm -f hello.c
$ git restore hello.c 从当前分支的最给i他新提交恢复文件 hello.c
```

其实checkout也能这样用，在写这文档之前我一直用的checkout,但git checkout在2.23起便不建议使用

用完git add之后，使用git commit 提交

```
$ git commit -m "这是提交信息"
$ git commit -m "这是提交信息" -a  #  -a 这个参数，会把所有已经被跟踪的文件的修改版本加入暂存区，被git rm的文件，会被清除
$ git commit Makefile #这会仅提交Makefile，对于其他文件则不做处理
```

一些参考:开源社区的commit message规范： RustSBI的规范，Linux内核

查看文件状态

```
$ git status
```

查看更改内容

```
$ git diff <file> #工作区与暂存区的差异
$ git diff <commit_hash>..<commit_hash> 比较提交之间的变动
$ git diff <branch_name1>..<branch_name2> 比较branch之间的区别
$ git diff <commit_hash>  #查看当前工作目录与该提交的差异
```

如果你只想看某次提交的内容，使用 git show

其他

构建的时候，会出现一些临时文件，比如说.o .rlib之类的，为了避免被错误的提交，你可以把文件添加到 `.gitignore` 中

这是相关的文档 https://git-scm.com/docs/gitignore#_pattern_format

cherry-pick

自己用git help cherry-pick了解，我觉得很有用

回退版本

比如你不小心了点什么奇怪的东西进去仓库?

使用 git reset

```
$ git reset --soft <commit-hash> #这会把当前分支的HEAD移到<commit_hash>处，但会保留工作目录和索引中的所有更改。
$ git reset --hard <commit-hash> #这个会把索引和工作区的更改清除
```

剩下的，可以使用git help reset 查看

默认是mix,重置暂存区，但是工作区不变

相对引用

前面说了git reset 的用法，commit_hash有些时候还是不方便的

比如，我希望恢复到上一个commit，hash多少有点麻烦了

可以直接用 `HEAD~1`

未命名文档

```
$ git branch branch1 #新建分支
$ git switch branch1 # 切换分支
$ git merge main #尝试把main上的更改合并到新分支上
$ git add test.c #假设test.c 这个文件是冲突的，在修改之后使用 add ，放入暂存区
$ git commit test.c -m "解决冲突的代码"
$ git branch -m newname #修改分支名
###如果要发布一个版本?
$ git tag -a <tagname> <commit-hash> -m
```

git log与git diff

你可以把前面讲的commit_hash换成branch，或者是tag

---

merge

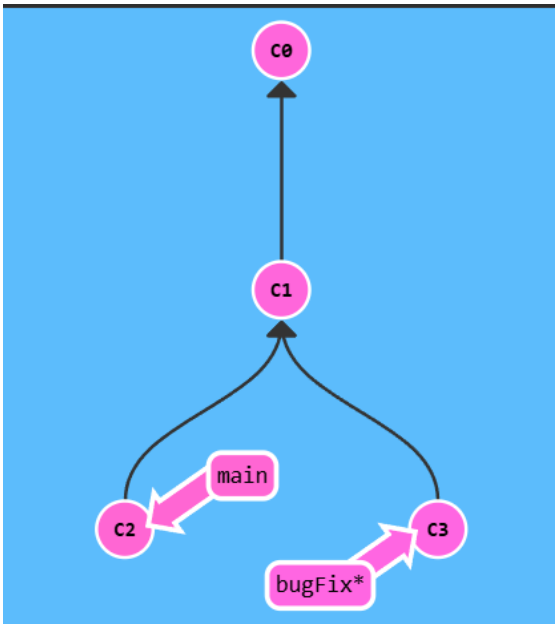
```
$ git merge <branch_name>#将branch_name合并到当前分支
```

解决冲突

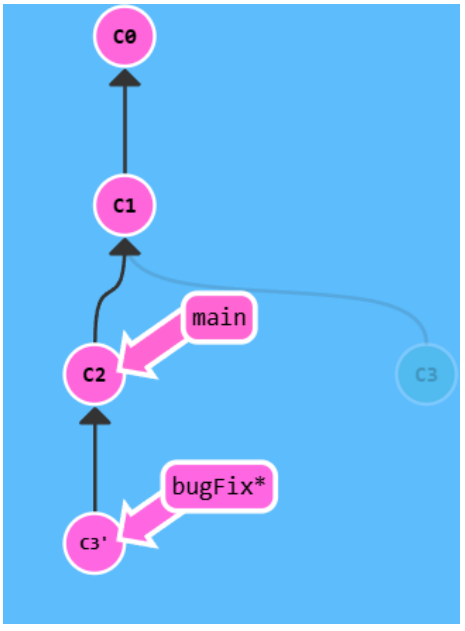
liaoxuefeng.com/books/git/branch/merge/

Rebase 变基

`Rebase 实际上就是取出一系列的提交记录，“复制”它们，然后在另外一个地方逐个的放下去。`，可以让历史看起来线性一些。



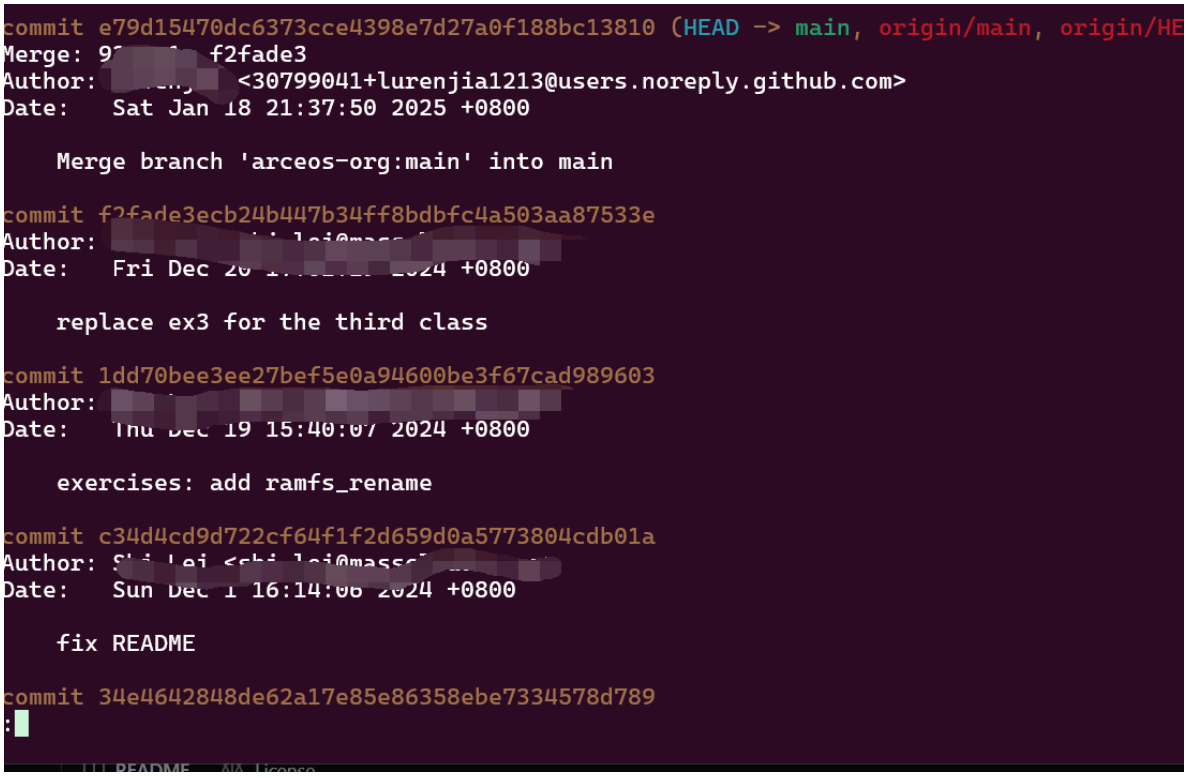

```
$git switch bugFix
$git rebase main
##等效于
$git rebase bugFix main
```



有些时候，commit太多，希望压缩?

```
$git rebase -i [startpoint] [endpoint] #如果不指定 [endpoint] ，则该区间的终点默认是当前分支HEAD所指向的commit
$ git rebase -i HEAD~3 #最近三个提交进行变基
```

下面是一个演示



这是未rebase前

```
$git rebase -i HEAD~3
```

```
pick 186187f fix README
s 90d9e4a exercises: add ramfs_rename
s 0d736b4 replace ex3 for the third class

# Rebase 8d284a6..0d736b4 onto 8d284a6 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
```

保存之后

```
  GNU nano 7.2                          /
# This is a combination of 3 commits.
# This is the 1st commit message:

fix README

# This is the commit message #2:

exercises: add ramfs_rename

# This is the commit message #3:

replace ex3 for the third class

# Please enter the commit message for
# with '#' will be ignored, and an emp
```

最后

```
commit 370f9636ee93650a9b39cd15fd80aff5c687b602 (HEAD -> main)
Author [redacted]
Date:  [redacted]

    fix README

    exercises: add ramfs_rename

    replace ex3 for the third class

commit 8d284a6c9e027cba7372e941b2378b7a81acf2c7
```

未命名

如果你是用 git init 创建的仓库，又想把这玩意放GitHub上:

添加远程仓库地址

`git remote add origin <url>` origin 是默认远程仓库

随后，配置为默认上游 git branch -u origin/main

然后使用 git push,git pull git fetch...

如果你有多个远程仓库

```
$ git push REMOTE-NAME BRANCH-NAME
#带上-f 强制推送 ,带上 -u 会把当前push的目标设置为上游
```

---

fetch 与pull

```
$ git fetch
$ git fetch REMOTE-NAME
#这玩意和git pull有啥差别呢? git pull等于是fetch  之后再merge。。。
#fetch后咋整呢?
$git merge REMOTE-NAME/BRANCH-NAME
```

pull用法差不多

使用 git help pull自行查看

到这里，git 部分应该就差不多了，我这寒假事有点多，所以这文档毛病还是很多的，感谢你的阅读

坦白说吧，其实直接看git的文档和其他的入门教程可能会好一点

还有速查表(cheatsheet)

还有一些我这没写的，但是我感觉用的上的东西,如果需要，你可以自己看看

cherry-pick将指定的提交应用到当前分支

stash暂存当前的更改，以便稍后恢复。

blame 查看文件的每一行最后一次修改的提交信息。

shortlog:统计不同用户的贡献reflog：查询引用日志

用于误操作恢复

bisect：在版本树上二分查找定位 bug

gc / prune / fsck：用于系统检查和垃圾清理

submodule：子模块管理

有必要一次性全部学会这些操作吗？我个人认为，用到啥学啥。

最后

作业:

```
$docker pull jkjkmxmx/sast2023-linux-git
$docker run --privileged -d -p 10000:22 -p 10001:80 -p 10002:3306 -p 10003:10001
-p 10004:10002 -h sast2023 --name sast2023 jkjkmxmx/sast2023-linux-git
$ssh -p 10000 train@localhost
```

这是清华计算机系学生科协2023年招新的作业，，

我不打算在这再讲docker什么的咋用了。。。自己查吧。我相信你的能力

该镜像是 x86-64/amd64 架构的

```
3.6 CTF – Linux 区
该区在 “train” 的家目录（即 /home/train）下的 “puzzles” 目录中进行。在这个目录中的每一个
子目录（如 “envir”）
就代表一个谜题，一个谜题恰对应一个 “flag”。
每个谜题下一定存在一个与目录名同名的可执行文件（可以有多种形式：二进制文件，Shell 脚本等；以
“envir” 为
例，即为 envir/envir），运行该文件并按照它的引导一步步获取 “flag”。
3.7 CTF – Git 区
该区在 “train” 的家目录下的 “git” 目录中进行。本区包含 6 个 flag，简要介绍如下：
3.7.1 Branches
* 看看这个仓库里有哪些分支？
3.7.2 Message
* 提交信息不止只有标题哦，还有正文！
3.7.3 History
* 数据被覆盖了！看看怎么跳回之前的版本？
3.7.4 Reflog
```

```
* 该谜题在 “reflog” 分支下进行。
* 观察提交信息：“add correct Taylor formula”，这说明之前有可能加入了错误的公式。
* 但是它不在版本树里，这说明很可能被 git reset 过。那该如何恢复这样的 “误操作” 呢?
3.7.5 Werewolf
* 该谜题在 “werewolf” 分支下进行。
* state 文件下看起来有很多 “flag”，但是这些基本都是假的。
* “预言家” 告诉你：最终版本的所有 “flag” 中，唯一一个由 “狼人” (werewolf) 提供且没有被后面
平民 (villager) 覆
盖的 “flag” 才是真正的 “flag”。
13
References References
3.7.6 Debug
* 该谜题在 “debug” 分支下进行。
* “cat” 对 debug.cpp 文件修改了很多次，但这些文件产生运行时错误了！“cat” 想知道，究竟是哪一
次提交，使这
个程序第一次产生运行时错误呢12?
* [该怎么寻找这样的提交呢？二分查找？如何在版本树中进行二分查找？]
* 只有这个提交正文里的 “flag” 才是真正的 “flag” 哦!
3.8 CTF - Bonus 区
该区会在任何可能的地方进行，它们有可能出现在服务器中，也有可能出现在其他任何地方。作为彩蛋，这里
不给
出相关的提示，希望大家能开心地寻找。
```

Linux区：proxy不做也罢

Bonus区，不做也罢

参考

https://docs.net9.org/

git manual

https://liaoxuefeng.com/books/git/introduction/index.html