Git Tutorial

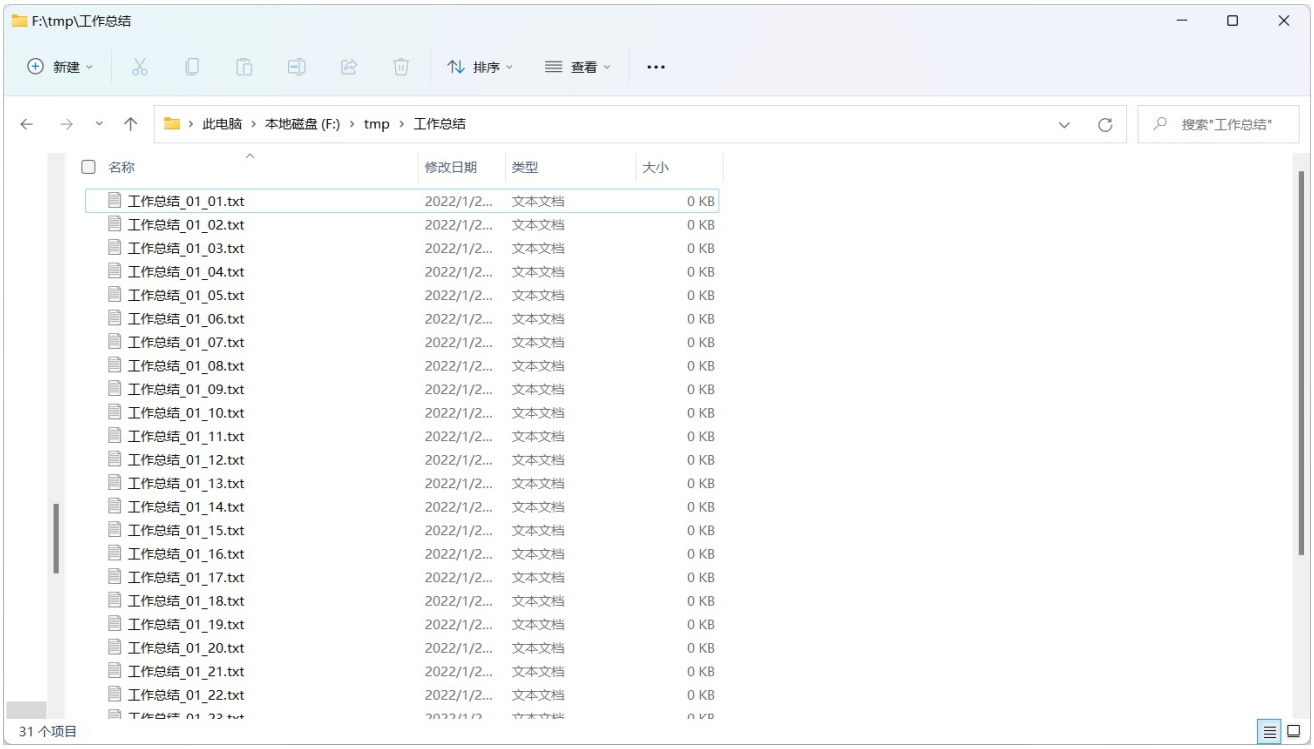
Prerequisite

- 安装git
- 安装vscode
- 有GitHub账户
- 在git中配置好GitHub账户（参考链接）

Introduction

版本控制系统（**VCS**）

在日常生活中常常会遇到版本管理的问题，比如初稿1，初稿2……直至终稿，可以直接用复制粘贴重命名的方式来管理各个版本，但是这样以后，文件夹就会变成下图的样子。



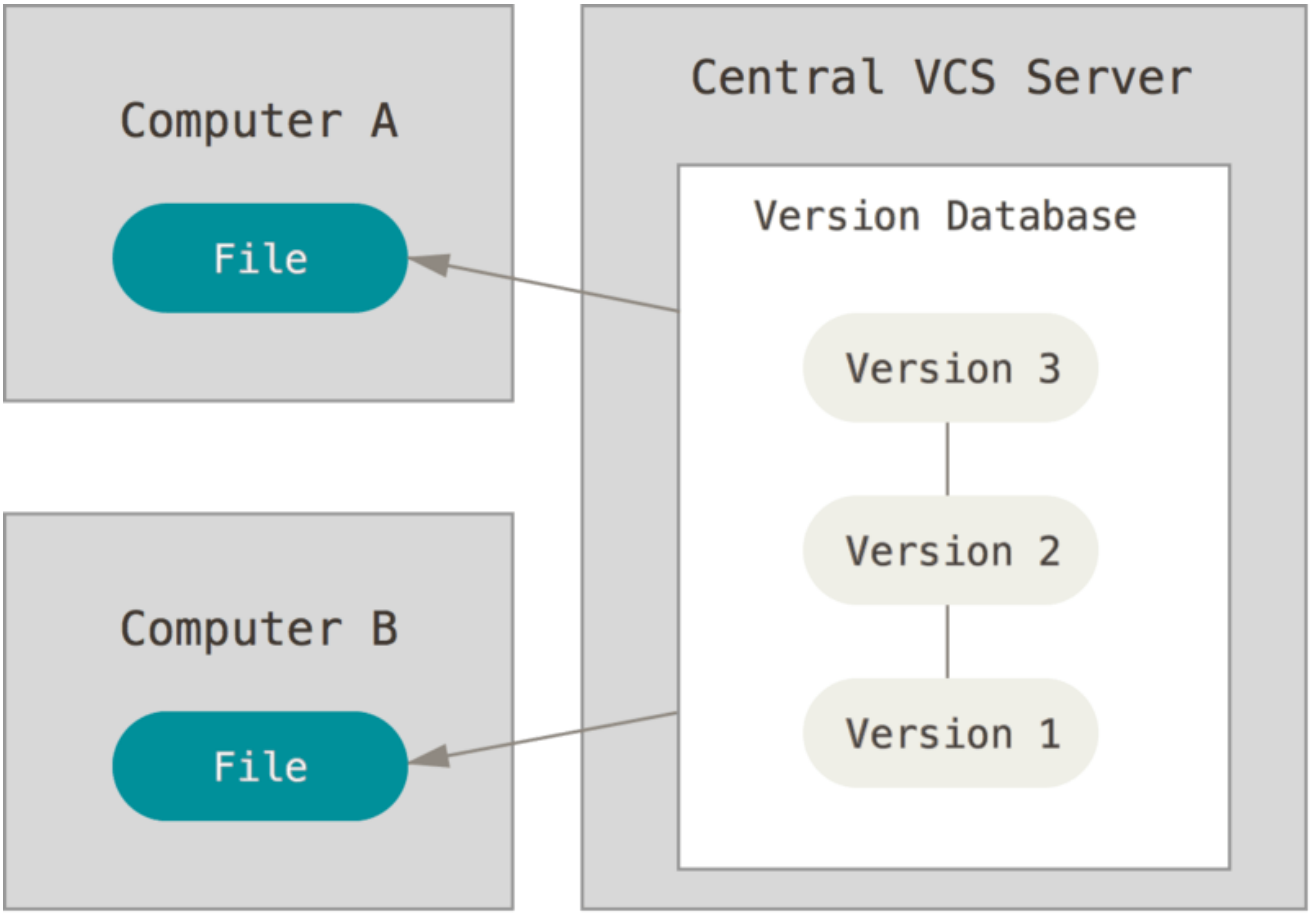
这还只是一个人每天的工作总结，而在协作开发中，面对大量的来自不同开发者的改动，一个版本控制系统（VCS）就显得尤为重要了。

集中化**&**分布式

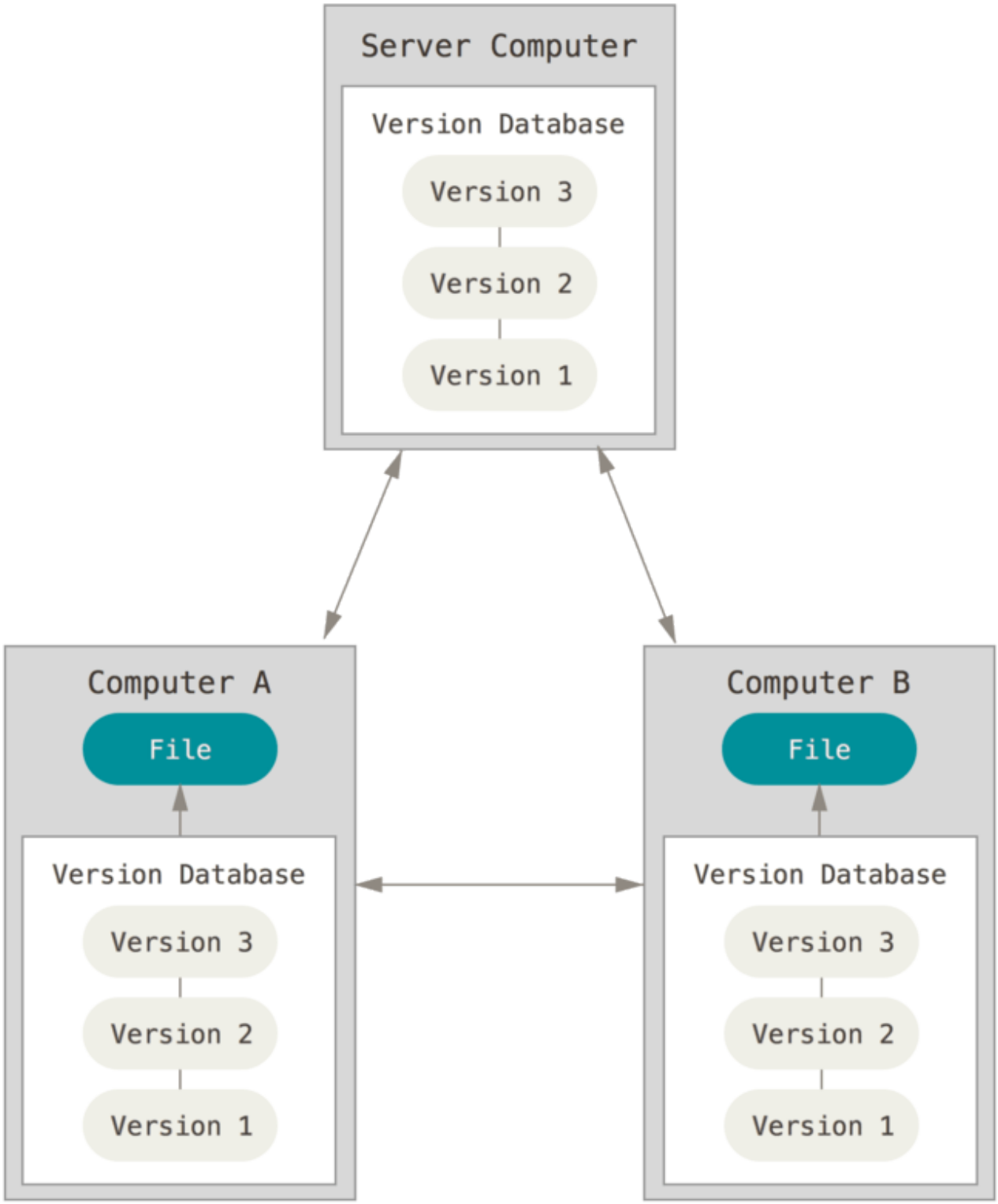
集中化的版本控制系统有中央服务器，各个客户端都以中央服务器上的版本为准。每个人都可以在一定程度上看到项目中的其他人正在做些什么。 而管理员也可以轻

松掌控每个开发者的权限，并且管理一个 CVCS 要远比在各个客户端上维护本地数据库来得轻松容易。

这么做最显而易见的缺点是中央服务器的单点故障。 宕机会导致无法提交更新，损坏会导致整个项目丢失。



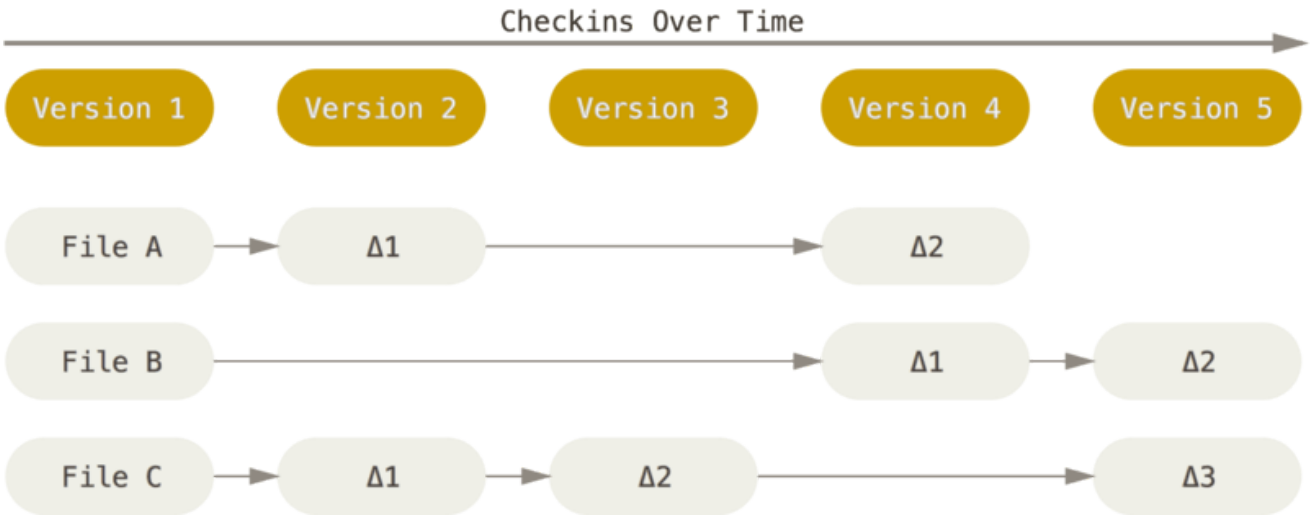
对于分布式版本控制系统，客户端并不只提取最新版本的文件快照， 而是把代码仓库完整地镜像下来，包括完整的历史记录。 这么一来，任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复。 因为每一次的克隆操作，实际上都是一次对代码仓库的完整备份。



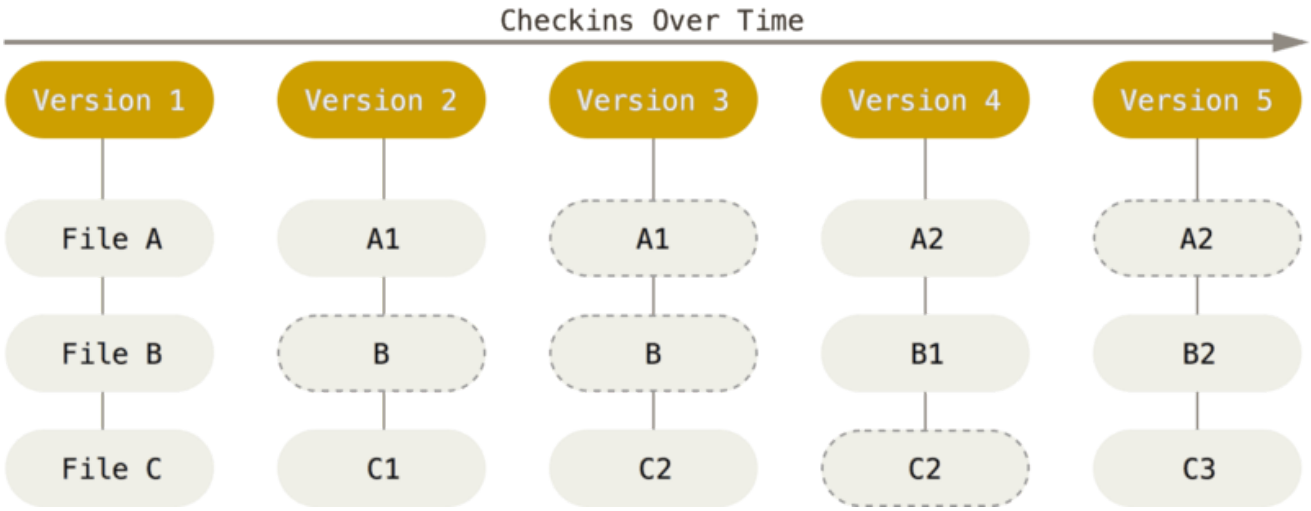
Git是一个分布式版本控制系统。

基于差异&快照流

基于差异的版本控制只记录同一个文件在不同版本之间的差异。



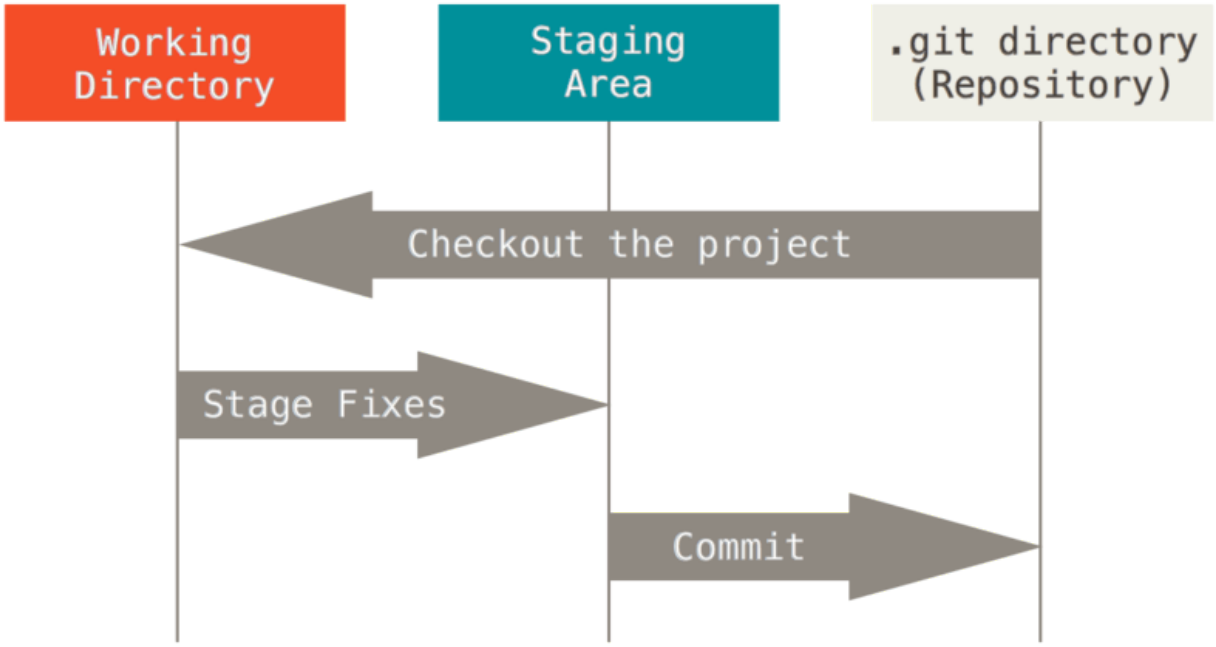
Git是快照流，如果文件在两个版本之间没有改动，就保留一个链接指向之前的文件；如果改动了，就将文件存入快照。



3种状态**&3**个阶段

一个文件会处在三种状态之一： 已提交（**committed**）、已修改（**modified**） 和 已暂存（**staged**）。

- 已修改表示修改了文件，但还没保存到数据库中。
- 已暂存表示对一个已修改文件的当前版本做了标记，使之包含在下次提交的快照中。
- 已提交表示数据已经安全地保存在本地数据库中。
  这会让我们的 Git 项目拥有三个阶段：工作区、暂存区以及 Git 目录。



基本的 Git 工作流程如下：

1. 在工作区中修改文件。
2. 将你想要下次提交的更改选择性地暂存，这样只会将更改的部分添加到暂存区。
3. 提交更新，找到暂存区的文件，将快照永久性存储到 Git 目录。

Installation&Configuration

去Git官方网站下载安装包即可。

或者，阅读安装教程。

安装完 Git 之后，要做的第一件事就是设置你的用户名和邮件地址。 这一点很重要，因为每一个 Git 提交都会使用这些信息，它们会写入到你的每一次提交中，不可更改：

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Ignore&attributes

.gitignore

设置文件的忽略规则，让一些文件不会被记入仓库的更改中。常见的情况是编译出的可执行文件或中间文件，不需要也不应当出现在代码仓库里。

```
./build
./main.exe
./*.exe
```

.gitattributes

设置文件的属性。略

Commands

basic

init

```
$ git init
```

会把当前目录变成git仓库。

clone

```
$ git clone https://github.com/libgit2/libgit2
```

这会在当前目录下创建一个名为 “libgit2” 的目录，并在这个目录下初始化一个 `.git` 文件夹， 从远程仓库拉取下所有数据放入 `.git` 文件夹，然后从中读取最新版本的文件的拷贝。 如果你进入到这个新建的 `libgit2` 文件夹，你会发现所有的项目文件已经在里面了，准备就绪等待后续的开发和使用。

status

```
$ git status
```

查看当前仓库的状态（有哪些文件修改了，有哪些文件暂存了等）。

add

```
$ git add file.cpp

$ git add .
```

将修改的文件提交到暂存区。

commit

```
$ git commit
```

提交更新，执行命令后会弹出文本编辑器，用于编辑提交说明。

```
$ git commit -m "commit message"
```

比较方便，不需要编辑说明。

```
$ git commit --amend
```

如果commit完成后，发现漏提交了几个文件，或者想要修改这次提交的信息，可以用上面的命令。

checkout

```
$ git checkout -- file.cpp
```

file.cpp会被还原成上一个提交时的样子，可能会导致更改的丢失。

rm

```
$ git rm file.cpp
```

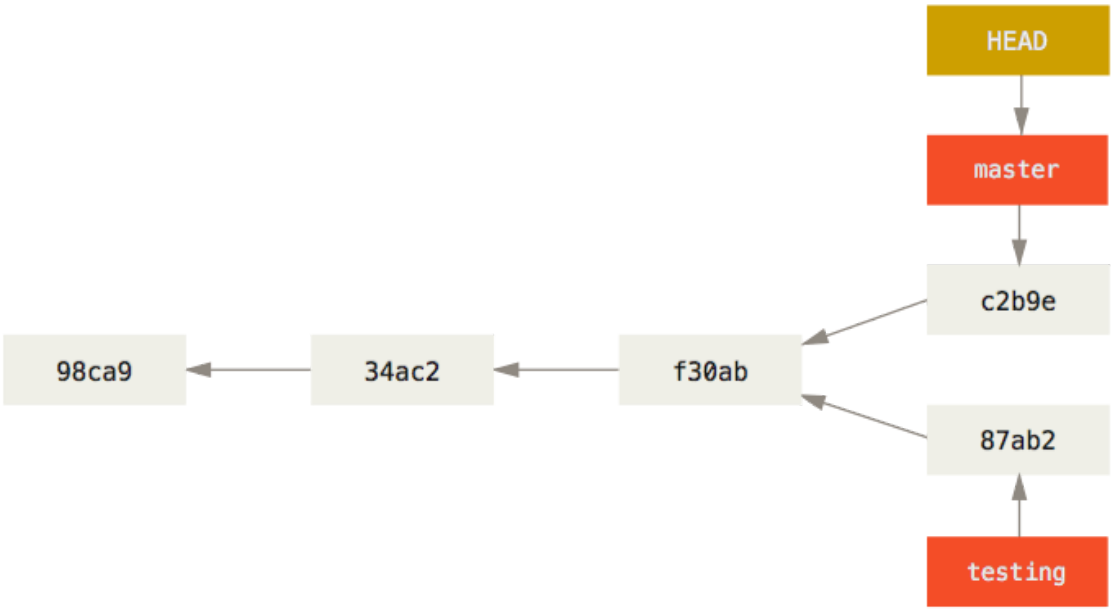
删除文件（也会在磁盘中删除）。

```
$ git rm --cached file.cpp
```

这个文件会从git仓库中删除，但是仍然保留在磁盘上。

branch&version

几乎所有的版本控制系统都以某种形式支持分支。 使用分支意味着你可以把你的工作从开发主线上分离开来，以免影响开发主线。



> THUAI6的分支图

log

```
$ git log
```

打印仓库的提交日志。

```
commit 79f2cf002767af83cd770b230196ce2b3394633d (HEAD ->
master)
Merge: ec6d90b fc96833
Author: wihn <88038740+wihn2021@users.noreply.github.com>
Date:   Mon Jul 10 17:08:11 2023 +0800

    Merge branch 'md'

commit ec6d90b87a086a607facca1d673f6fea50e96497
Author: wihn <88038740+wihn2021@users.noreply.github.com>
Date:   Mon Jul 10 17:02:42 2023 +0800

    master commit

commit fc9683397351ddd78d04bf2c5530c983fa9316b9 (md)
Author: wihn <88038740+wihn2021@users.noreply.github.com>
```

```
Date:   Mon Jul 10 17:02:10 2023 +0800

    md commit

commit 096b84e56f95fce5086379a5ed3879b7ea7143cf
Author: wihn <88038740+wihn2021@users.noreply.github.com>
Date:   Mon Jul 10 13:37:56 2023 +0800
```

每个commit都用唯一的字符串标识，在下面reset的时候会用到。

reset

```
$ git reset <commit>
```

根据log打印的版本进行回退。例如，依照上面的log输出，可以执行以下命令来回退。

```
$ git reset 096b8
```

branch

```
$ git branch
```

打印出这个仓库的分支。

```
$ git branch testing
```

创建名为testing的分支。

```
$ git branch -d iss53
```

删除分支iss53。

checkout

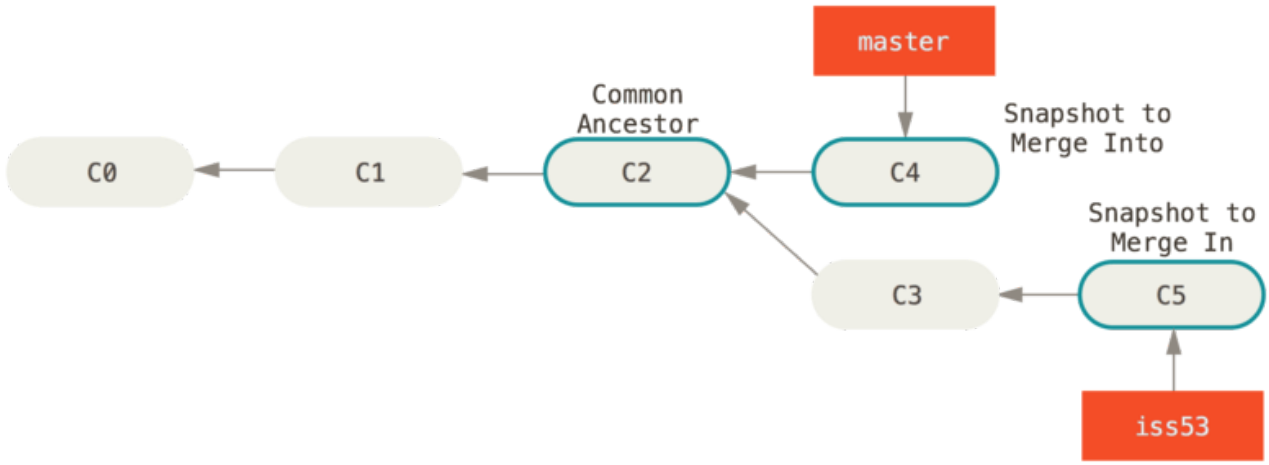
```
$ git checkout testing
```

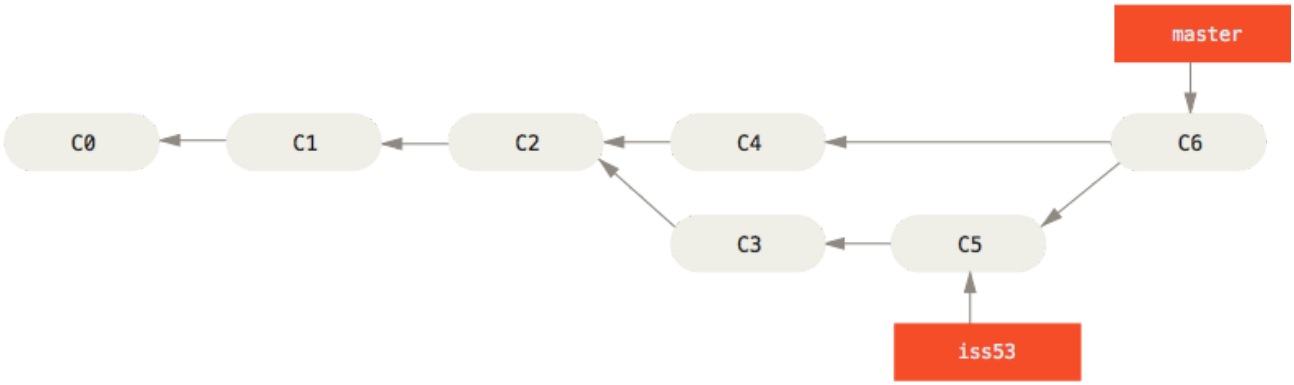
切换到testing分支。可能会因为更改了文件且未提交而无法切换，可以先提交再切换，或者强制切换。

merge

```
$ git checkout master
$ git merge iss53
```

把iss53合并进master。





然而，有的时候merge会产生冲突，需要解决冲突，也就是在两个分支矛盾的地方选择其中一个为准。

Github

basic

remote

```
$ git remote -v
$ git remote add pb https://github.com/paulboone/ticgit
$ git remote add origin git@github.com:wihn2021/git-tut.git
```

查看现有远程仓库、添加远程仓库，其中pb和origin是为远程仓库起的名字。这里演示了连接到GitHub的两种协议，通常使用后者能在一定程度上规避网络问题。

push

```
$ git push origin master
```

把本地的commit推送到origin的master分支。

pull

```
$ git pull origin master
```

从origin仓库拉取master分支，并和本地仓库进行merge。

协作开发的一般流程

1. fork公共仓库
2. 把自己的repo clone到本地
3. pull拉取公共仓库的最新更改
4. 编写、开发、提交
5. push提交到自己的远程repo
6. 向公共仓库发起Pull Request

workflow

GitHub提供的一系列脚本，可以自动化地对代码执行一些操作，比如THUAI6的代码格式检查。

一些功能

- issue
  开发者可以使用issue来报告软件缺陷（bug）、请求新功能、提出改进建议，或者在项目中讨论任何与开发相关的问题。每个issue都有一个唯一的编号、标题和描述，可以随时添加评论、标签、里程碑、分配给特定的负责人和指派者等元素，以便更好地管理和跟踪问题的处理进度。此外，GitHub还允许在issue中引用代码提交、拉取请求、里程碑和其他issue，以帮助开发者更好地理解问题的背景和相关信息。
- wiki
  开发者可以使用wiki来编写项目文档、用户手册、安装说明等，并与其他贡献者共享和编辑这些文档。

- release
  用于发布软件。

参考资料

- Pro Git（强烈推荐）
- runoob
- eesast文档

Homework

https://github.com/wihn-pub/Git-Homework-2023