Git 入门教程

最优雅的版本控制系统

SAST 软研部运维组 夏文哲
2024.10.07

关于版本控制

什么是“ 版本控制系统 ”？ 我为什么要关心它呢?

> “ 版本控制系统是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统。
> 在本书所展示的例子中，我们对保存着软件源代码的文件作版本控制，
> 但实际上，你可以对任何类型的文件进行版本控制。”
> – Pro Git

现实中可能遇到的问题

在现实的开发场合中，我们常常会遇到数据文件的备份，保存，以及同步的问题。你可以想象这样一个场合：

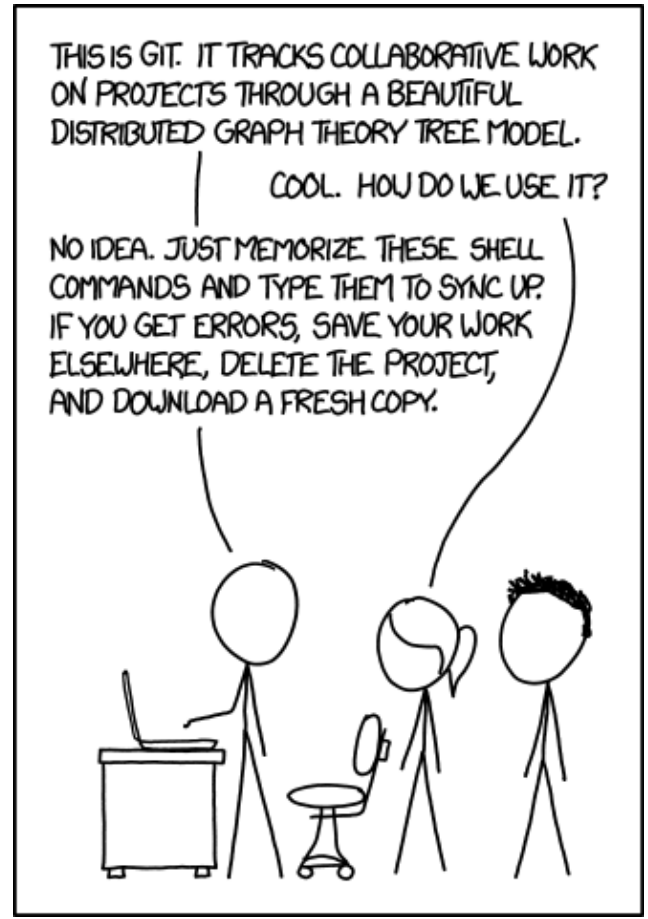
假如你现在在一个小组中，小组里的每个人都需要根据自己的分工完成同一份实验报告。
但是！！！
一天前，A同学给你发了一份他本地修改过的报告，你于是在他的基础上完成你的部分。
第二天，C同学把他的报告和D同学的报告合并之后又发给了A同学，A同学修改完之后又发给了你，
一来一回，你发现你又要重写你负责的部分...

或者，你正在参加一个限时的比赛，时间还很多，
你准备优化一部分代码，然后添加一些新功能，调着调着，比赛还剩下半小时，可是新的部分却依然没有调好...
无奈之下，你依稀记得你给之前的代码做了备份，可是看着电脑里`新建文件夹（1）` `新建文件夹（2）` `新建文件夹（3）` `新建文件夹（4）` `新建文件夹（5）`，你陷入了沉思...

现代的版本控制系统

它可以帮助您自动地回答以下问题：

- 当前模块是谁编写的？
- 这个文件的这一行是什么时候被编辑的？是谁作出的修改？修改原因是什么呢？
- 最近的 1000 个版本中，何时/为什么导致了单元测试失败？



基于差异的版本控制系统

蝴蝶效应！

- 相比于复制备份，基于差异的版本控制系统大大缩小了需要的储存空间
- 回退版本时需要从头开始一步步回溯，如果文件在版本系统不知晓的情况下修改，所有备份将会损坏丢失

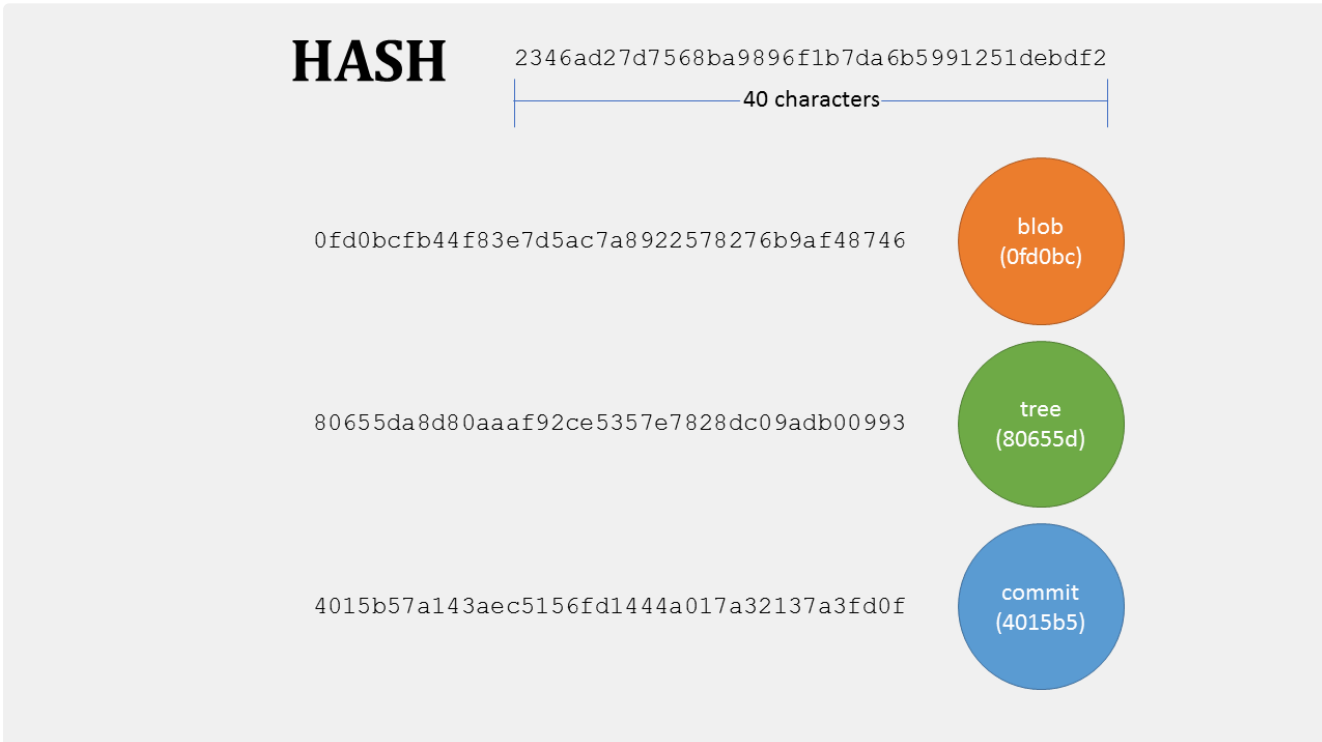
基于快照的版本控制系统

茄子!

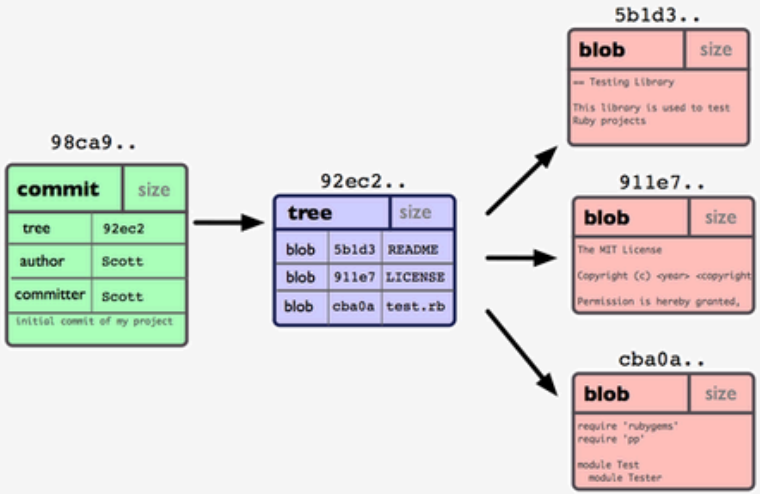
- 保留了差异式系统所需储存空间小的优点
- 用户提交新快照时，若文件没有更改，Git可以直接添加旧的镜像索引到新的提交中
- 所有提交都可以在极短时间内完成回溯
- 由于Git会给所有管理的对象计算SHA1校验和（SHA-1算法特性），以保证文件的完整性
- 在用户的所有操作中，几乎都只是向Git中添加数据，很少有可能造成数据丢失（所以建议在Git储存库中不要存入敏感信息）

哈希算法示例

digest?

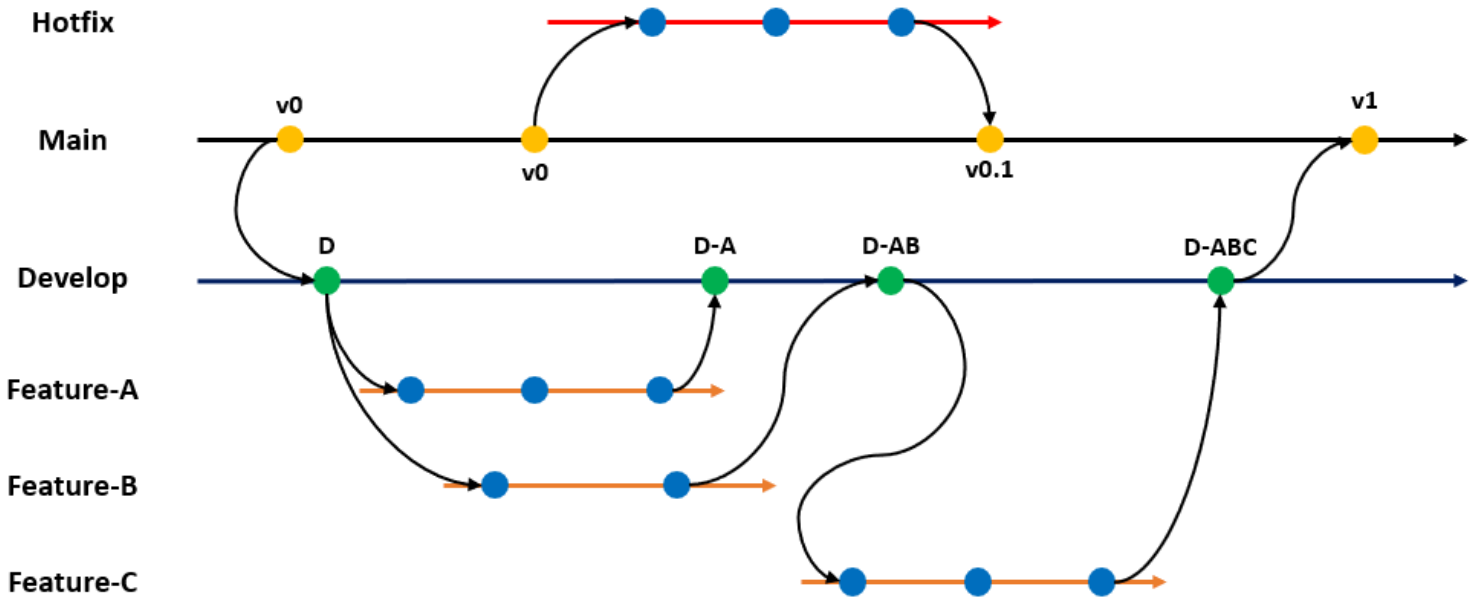


一些Git概念



1. blob： Git的基本储存单元之一，是最简单的对象，一般是文本文件，但也可以是其他内容
2. tree： 其引用blob形成目录，同时也可以引用其他tree作为子目录
3. commit： 向DAG图中添加一个节点（快照），这个节点指向表示提交时文件状态的tree
4. ref： 类似于指针（便利贴）的作用（给节点的哈希值一个名字），指向当前DAG图的状态或者你处于的位置
   - HEAD: 特殊的一个标签，用于指代你的当前储存库在DAG图中所处于的位置
   - master: Git默认创建的分支名称
   - <remote>: 用于指代远程git服务器所处于的位置
   - <branch>: 用于指代DAG图中分支的标签

Git实质上是一个DAG图（有向无环图）

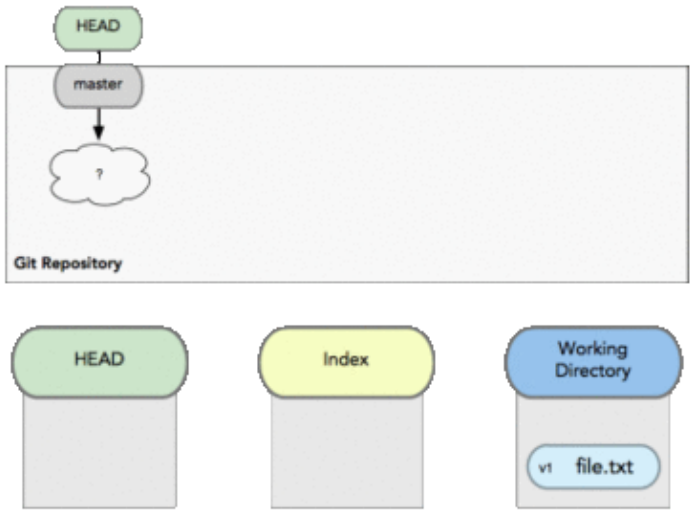


一个常见的Git分支图，每个点代表了一个commit，不同行代表所处的不同分支。

不难看出，Git只是帮我们保存并维护了两样东西：对象和引用。

Git命令行操作

Git工作流



- 编辑文件
- 提交到Git暂存区（add/stash）
- 提交快照（commit）

Add / Commit

```
usage: git add [<options>] [--] <pathspec>...
# 将文件添加到暂存区
usage: git commit [-a | --interactive | --patch] [-s] [-v] [-u<mode>] [--amend]
                  [--dry-run] [(-c | -C | --squash) <commit> | --fixup [(amend|reword):]<commit>]
                  [-F <file> | -m <msg>] [--reset-author] [--allow-empty]
                  [--allow-empty-message] [--no-verify] [-e] [--author=<author>]
                  [--date=<date>] [--cleanup=<mode>] [--[no-]status]
                  [-i | -o] [--pathspec-from-file=<file> [--pathspec-file-nul]]
                  [(--trailer <token>[(=|:)<value>])...] [-S[<keyid>]]
                  [--] [<pathspec>...]
# 提交一个commit
usage: git rm [-f | --force] [-n] [-r] [--cached] [--ignore-unmatch]
              [--quiet] [--pathspec-from-file=<file> [--pathspec-file-nul]]
              [--] [<pathspec>...]
# 同样，你也可以这样把文件从暂存区移除
```

Example:

```
git add file.txt
```

Example:

```
git commit -m "Added file.txt"
```

提问Time！ 请问我这次commit提交的modified.txt文件的内容是什么时候的?

Branch / Checkout / Switch

```
usage: git branch [<options>] [-r] (-d | -D) <branch-name>...
# 新建分支
usage: git checkout [<options>] <branch>
# 切换分支（即切换HEAD标签所处位置）
usage: git switch [<options>] [<branch>]
# 也可以这样切换分支
```

Example:

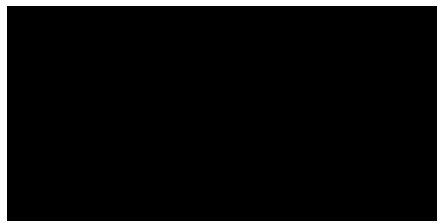
```
git checkout -b dev
# 创建一个名字dev的分支并切换过去
```

Merge / Rebase

```
usage: git merge [<options>][<commit>...]
# 合并分支
usage: git rebase [-i] [options] [--exec <cmd>] [--onto <newbase> | --keep-base] [<upstream> [<branch>]]
# 变基分支
```

Example:

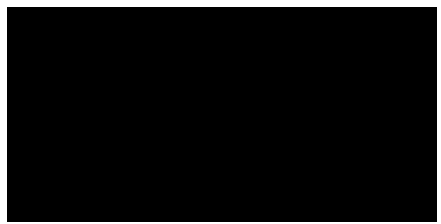
```
git checkout master
# 切换到主分支
git merge dev
# 合并dev分支
```



Example:

Fast-forward:

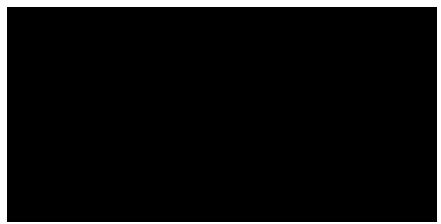
当两个分支在同一条未分支的路径上时，Git会尝试直接移动分支标签进行合并，而不是创建一个新的commit



Example:

Merge 和 Rebase

两种不同的合并方法，相比于merge，rebase在合并时相当于直接把待合并的分支嫁接在了主分支的顶部。



Oh my god it’s a conflict!

有时候合并的两个分支含有冲突的内容，Git会尝试帮我们自动合并。
但在无法自动合并时，Git就会提示我们手动解决冲突。

这是文件合并时遇到冲突时的样子，请不要害怕。
这并不意味着我们的文件损坏了，他只是标记了我们需要手动解决冲突的位置。

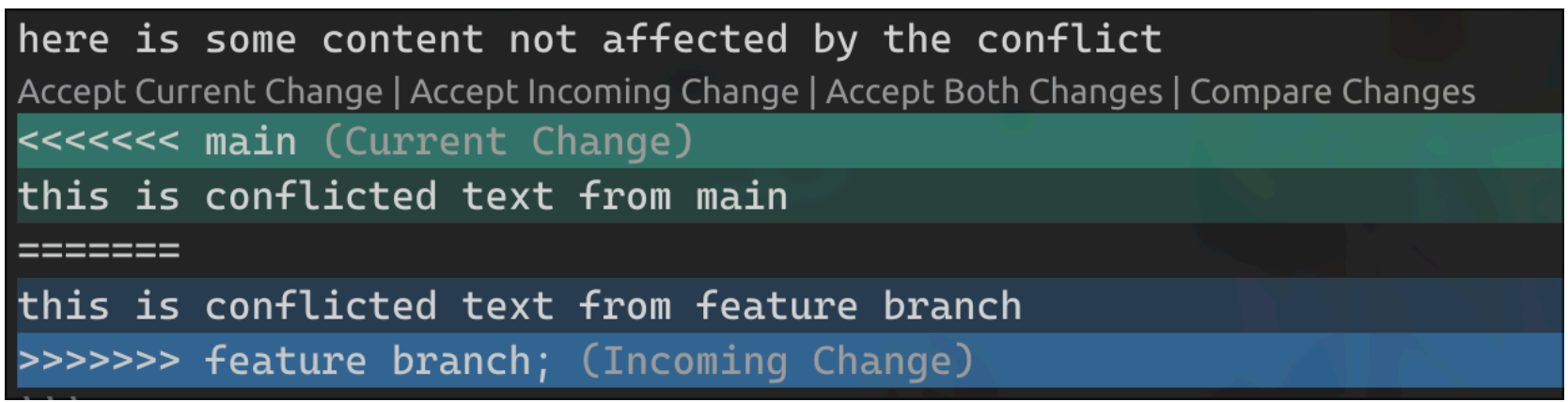
```
here is some content not affected by the conflict
<<<<<<< master
this is conflicted text from master
=======
this is conflicted text from feature branch
>>>>>>> feature branch;
```

这是啥子?

这是一种标注文件内容差异的格式 GNU-diff

我们既可以手动删除不需要的部分，
也可以通过IDE或者其他图形化的合并工具帮助我们解决冲突，再重新merge。

有些比较智能的IDE或者编辑器会将它们标记出来，如 VSCode 会将其显示成这样子：



Clone / Pull / Push

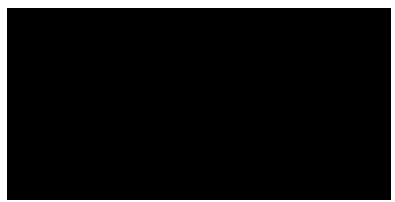
白嫖党最爱

有时候我们需要从远端获取我们的代码仓库，这时就需要使用pull和push了

```
usage: git clone [<options>] [--] <repo> [<dir>]
# 获取一个完整的远程仓库
usage: git pull [<options>] [<repository> [<refspec>...]]
# 从远程获取最新的更改，相当于 git fetch && git merge
usage: git push [<options>] [<repository> [<refspec>...]]
# 推送当前的更改至远程
```

Example:

```
git clone git@github.com:NJUPT-SAST/sast-evento.git
# 获取sast-evento仓库
git pull
# 从sast-evento获取最新的提交
git push
# 向远程仓库提交你本地的更改
```



注意：

如果你的仓库是由 `git init` 初始化的，则需要手动使用 `git remote add` 添加一下远程git服务器的地址；
如果是 `git clone` 下来的，则远程服务器名称默认为 origin ，地址默认为你 clone 时的地址。

Log

有时候你忘记了之前的操作，或者想要查看一下当前仓库的历史状态，则可以使用Git的log功能。

```
usage: git log [<options>] [<revision-range>] [[--] <path>...]
```

Example:

```
git log --all --graph --decorate
# 这样看上去会更加清晰
```

```
* commit bb5a72379df61c1973e154883ac59c18ee156716 (HEAD -> master)
| Author: f3rmata <fermataa@gmx.com>
| Date:   Sat Oct 5 19:01:14 2024 +0800
|
|     bug fix
|
| * commit 3d848b1d8720831fb587b4659a56558098a29f49 (dev)
| | Author: f3rmata <fermataa@gmx.com>
| | Date:   Sat Oct 5 18:11:39 2024 +0800
| |
| |     feat2
| |
| * commit dc6cd0587ce523e7f5d055927f8a6622d45ca81f
|/  Author: f3rmata <fermataa@gmx.com>
|   Date:   Sat Oct 5 18:11:12 2024 +0800
|
|       feat1
|
* commit ed0b6d50c8f40de17f95f47c9fa837216819bba7
  Author: f3rmata <fermataa@gmx.com>
  Date:   Sat Oct 5 18:10:31 2024 +0800
```

```
I am the master!
```

Other

进一步了解Git

一些很有意思的命令：`git blame` `git cherrypick` ...

一些学习资料：

- Pro Git
- Atlassian关于Git的教程
- 在游览器里尝试Git
- MIT 计算机科学中缺失的一课

Submodules

Git 的 submodule 功能允许你将一个 Git 仓库作为另一个 Git 仓库的子模块（submodule）包含进来。
这在管理大型项目时非常有用，特别是当你需要将多个项目组合在一起时。

Git 历史

Git是一个开源的分布式版本控制系统，由Linus Torvalds在2005年创建，用以管理Linux内核开发。
Git的诞生是为了提供一个快速、可扩展且可靠的代码管理工具，它允许多个开发者同时工作在同一个项目上，而不会相互干扰。

- 2005年：Linus Torvalds创建Git，最初是为了更好地管理Linux内核的开发。
- 2005年4月：Git的第一个公开版本发布。
- 2008年：GitHub的推出，这是一个基于Git的代码托管平台，极大地推动了Git的普及。
- 2008年4月：Git 1.5.0发布，引入了`git fetch`和`git merge`命令。
- 2010年 - 至今：Git开始被越来越多的项目和开发者采用。

Github

GitHub 是一个基于 Git 的版本控制和协作平台，它允许用户托管和管理代码，以及跟踪任务和增强团队合作。
GitHub 由 Chris Wanstrath、PJ Hyett 和 Tom Preston-Werner 在 2008 年创立，现在是微软的一部分。

如何参与Github上的开源项目？

- Step 1. 把仓库fork到自己的账户下（获得对代码的修改权）
- Step 2. 把自己fork的repo克隆到本地并进行修改（建议新建一个自己的分支）
- Step 3. 推送自己的修改，并在原项目页面上提交一个 `Pull Request`，附上自己修改的简要介绍，等待原作