# 用爱一起画 Git

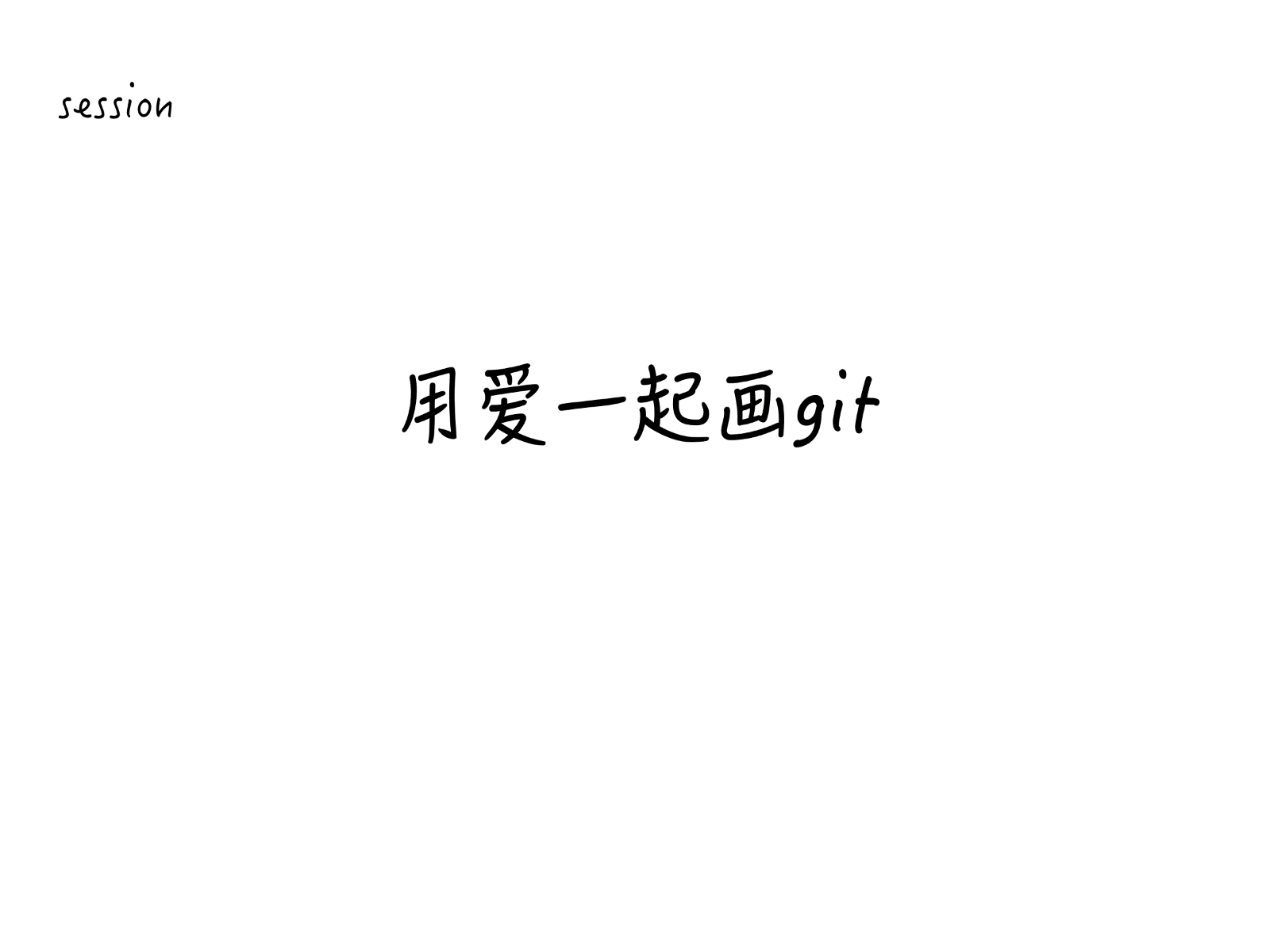
原文出处：[http://yanminx.com/blog/understand-git-by-drawing/](http://yanminx.com/blog/understand-git-by-drawing/" \t "_blank)  
作者：[Yanmin](http://yanminx.com/" \t "_blank)

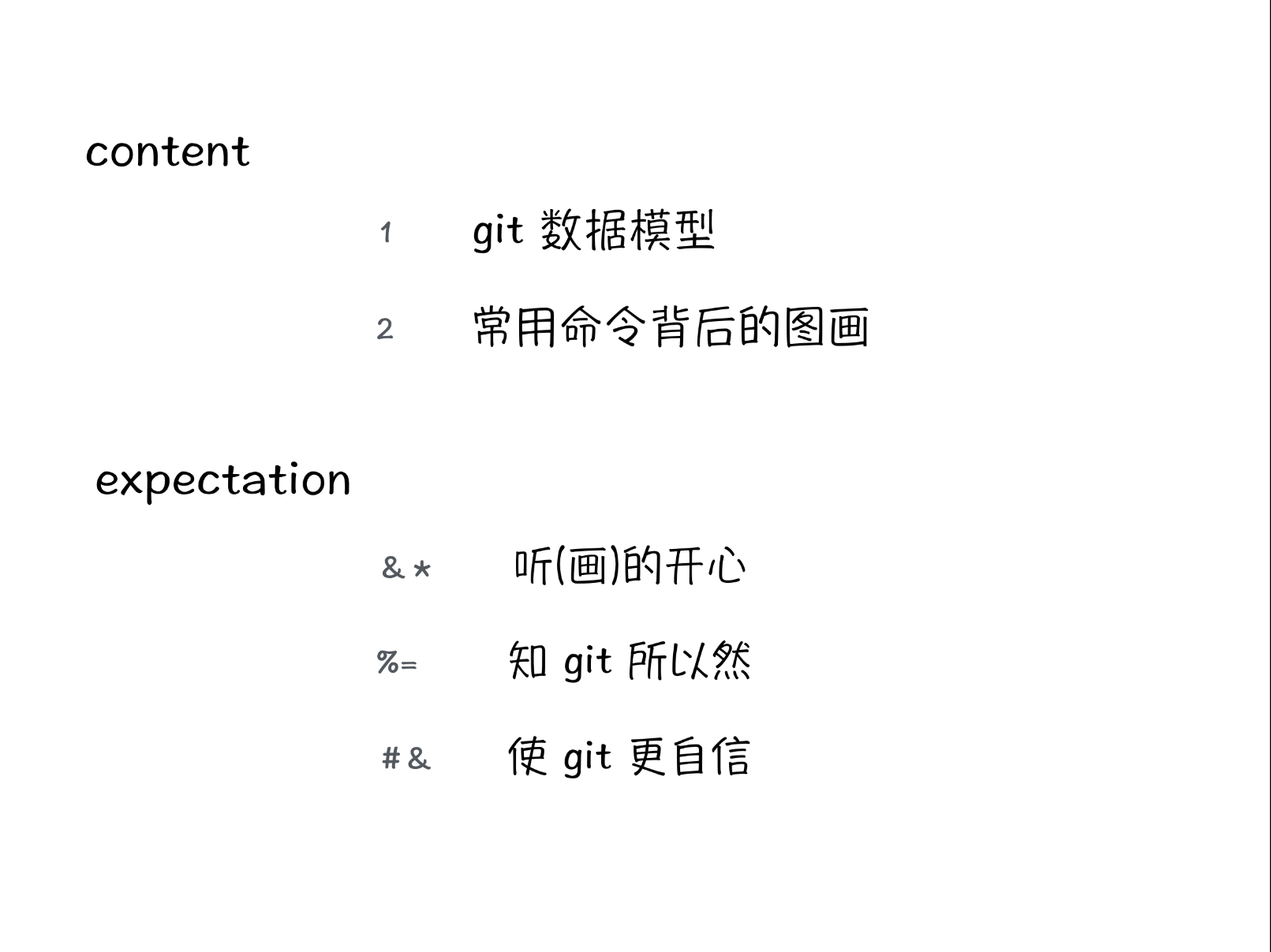
https://www.kancloud.cn/digest/understand-git-by-drawing/99054

## 介绍

这是我在项目组里讲过的一次git session，在此将所有slides添加讲解记录为博文.本文全文贯穿一个例子，先简单引进了git中的几个数据模型，然后通过画图方式帮助大家了解几个常用命令背后的行为。画图重点不涉及分支而是数据模型间的组织。

首先附上夺眼球的标题页和博文 session愿景





* 正式内容开始了,首先将介绍我们贯穿全文的例子背景（总体来说就是用git记录更新主人公的恋爱状态，当然是瞎编的）。例子将随着剧情的变化给出不同的场景，每个场景对应一个git命令。
* 在本地初始化一个空仓库（本文例为 mylove 目录下），进入 .git 目录下将看到git的目录结构，我们重点关注的将是 **objects** 目录，后续的图画中的git对象也大多存放于此。此时该目录下普通文件为空.

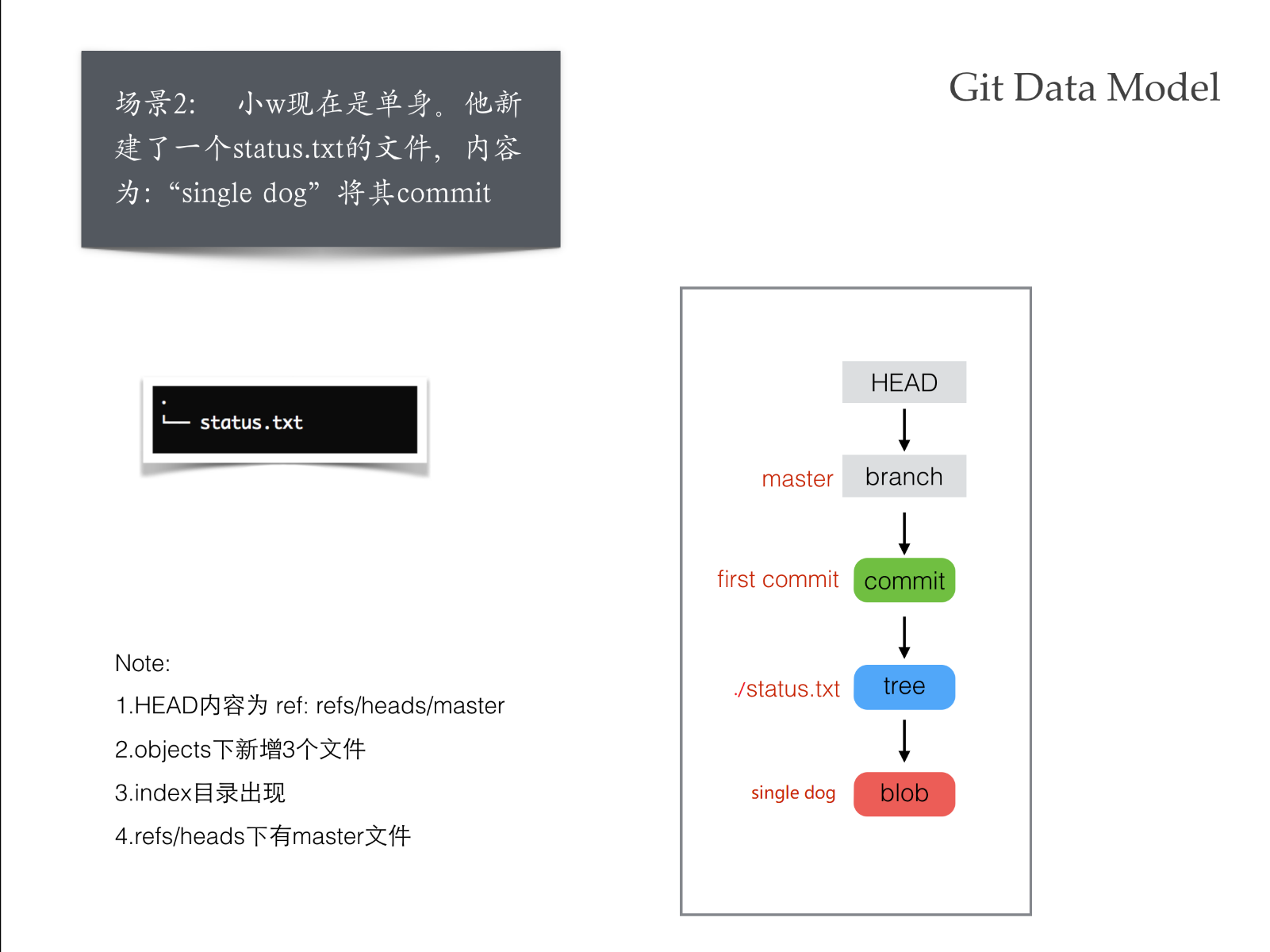
## 场景1

本场景命令：git init



## 场景2

本场景命令：git add . 和 git commit –m “first commit”

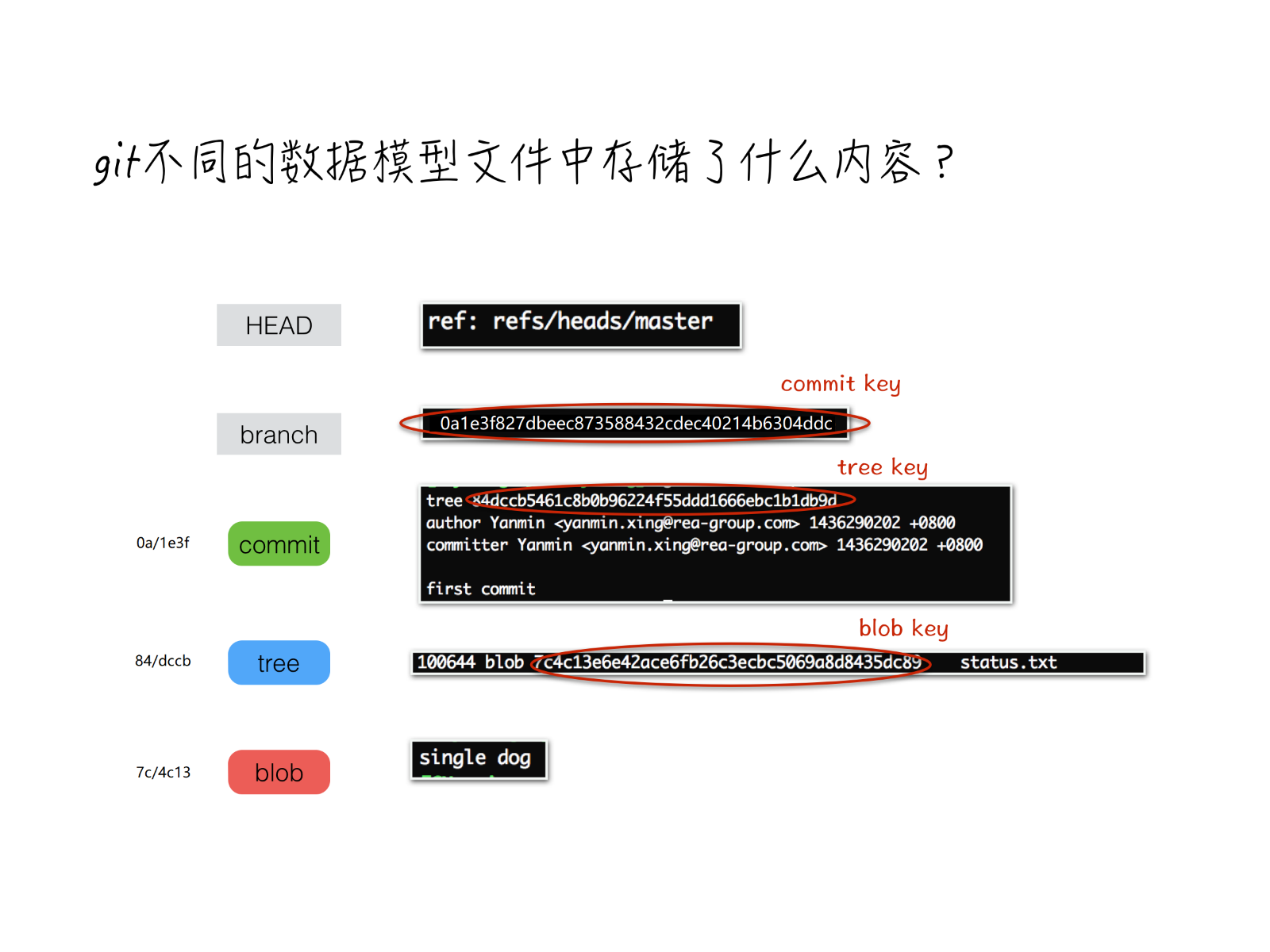


* 在本场景中, 主人公新建了 status.txt 文件， git add . 后 objects 目录下多出一个文件；随后提交第一个commit，objects目录又多出两个文件，那么这多出的三个文件到底用来干什么呢？

其实这三个文件都是git对象，分别对应三种git数据模型 **blob、tree、commit，**其实还有第四种对象 **tag**。

* 第一次提交后git的对象（blog、tree、commit）和引用（head、branch）间的关系组织如上图所示，图中已表明每一个对象和引用分别对应的事物。
  + **blob** 对应我们仓库里的普通**文件**，当你将一份文件add到仓库里时，git将会为你产生一个blob对象存于objects目录下.
  + **tree** 对应我们仓库里的一个**目录，**当你做一次commit时，git将会为你新增的文件所属目录路径（例如 ./File1.txt、./Path/File2.txt 相对路径）产生tree对象存于objects目录下；如果新增或修改的多个文件位于同一个路径下，那么只会产生一个 tree 对象。
  + **commit** 对应我们的一次**提交行为**，当你做一次commit时，git将会为你产生tree对象（若有文件被修改或新增文件）的同时产生commit对象，存于objects目录下。
* 可以利用 git cat-file -t <SHA-1-of-one-object> 查看一个对象的数据类型，<SHA-1-of-one-object> 的值为你看到一个objects下新增的文件的子目录+文件名。

**那么在这些git对象和引用种到底存储了什么内容,他们怎样组织起来的呢？**



如上图所示：

* + 文件.git/HEAD是个引用，其中存的是当前branch的文件路径，默认为master 分支。
  + 文件.git/refs/heads/master是个分支，其中存的是一个SHA-1值，该值是一个commit对象的散列key，意为分支master当前指向该特定commit对象。
  + 在 .git/objects/ 目录中管理了所有的 Git 对象（按照对象数据模型区分为 **blob、tree、commit**）,根据每个 Git 对象的 SHA-1 值（共40个字符），取其前面2个字符建立目录，再将剩余38个字符作为Git 对象的文件名创建对象文件。
  + 每个commit对象的文件保存了若干信息，包括上一个commit对象的key，本次提交信息、提交者、提交时间；以及还保存了一个或多个 tree 对象的 key 值。
  + 每个tree对象里存储的是一个或多个blob 对象的 key 值（以及文件名称）以及一个或多个其他 tree 对象的 key 值。
  + 每个blob对象里存储的仅仅是真实的文件内容的加密数据。

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ find .git/objects/ -type f

.git/objects/0a/1e3f827dbeec873588432cdec40214b6304ddc

.git/objects/7c/4c13e6e42ace6fb26c3ecbc5069a8d8435dc89

.git/objects/84/dccb5461c8b0b96224f55ddd1666ebc1b1db9d

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -t 0a1e3f

commit

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -p 0a1e3f

tree 84dccb5461c8b0b96224f55ddd1666ebc1b1db9d

author meanx <meanx@example.com> 1641797018 +0800

committer meanx <meanx@example.com> 1641797018 +0800

first commit

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -t 84dccb

tree

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -p 84dccb

100644 blob 7c4c13e6e42ace6fb26c3ecbc5069a8d8435dc89 status.txt

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -t 7c4c13

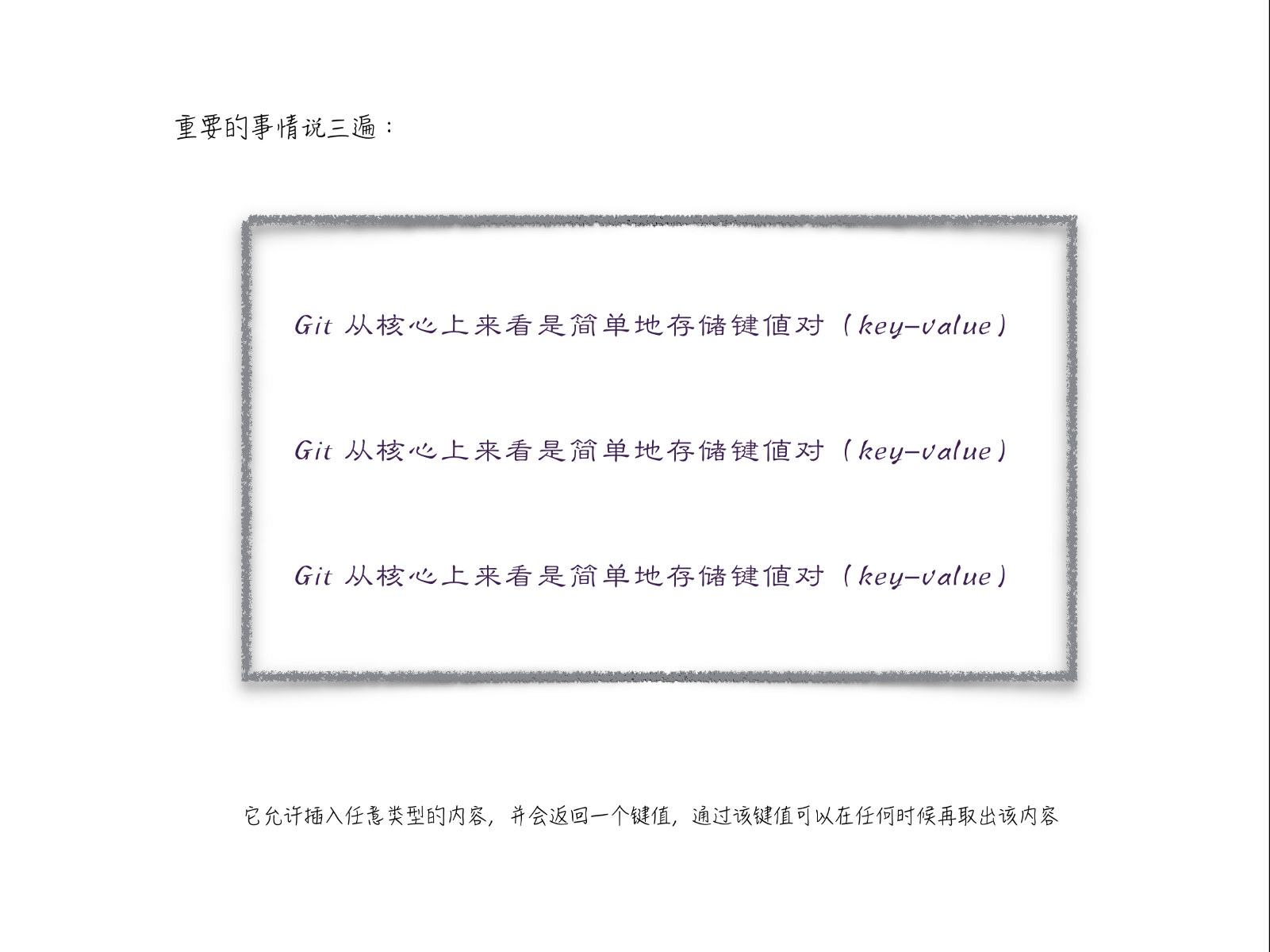
blob

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

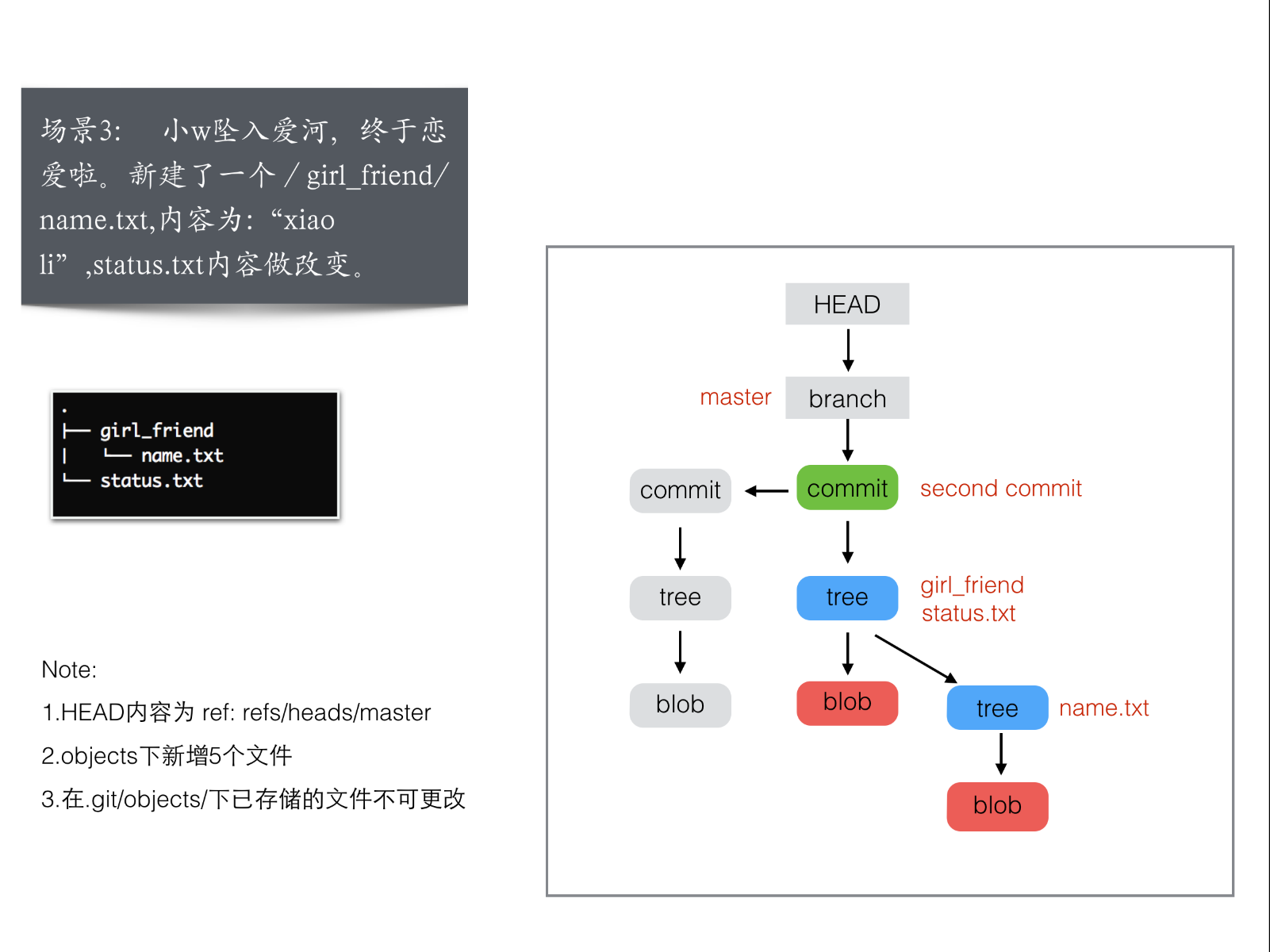
$ git cat-file -p 7c4c13

single dog

Git 是一个内容寻址文件系统。看起来很酷，但这是什么意思呢？ 这意味着，Git 的核心部分是一个简单的键值对数据库（key-value data store）。

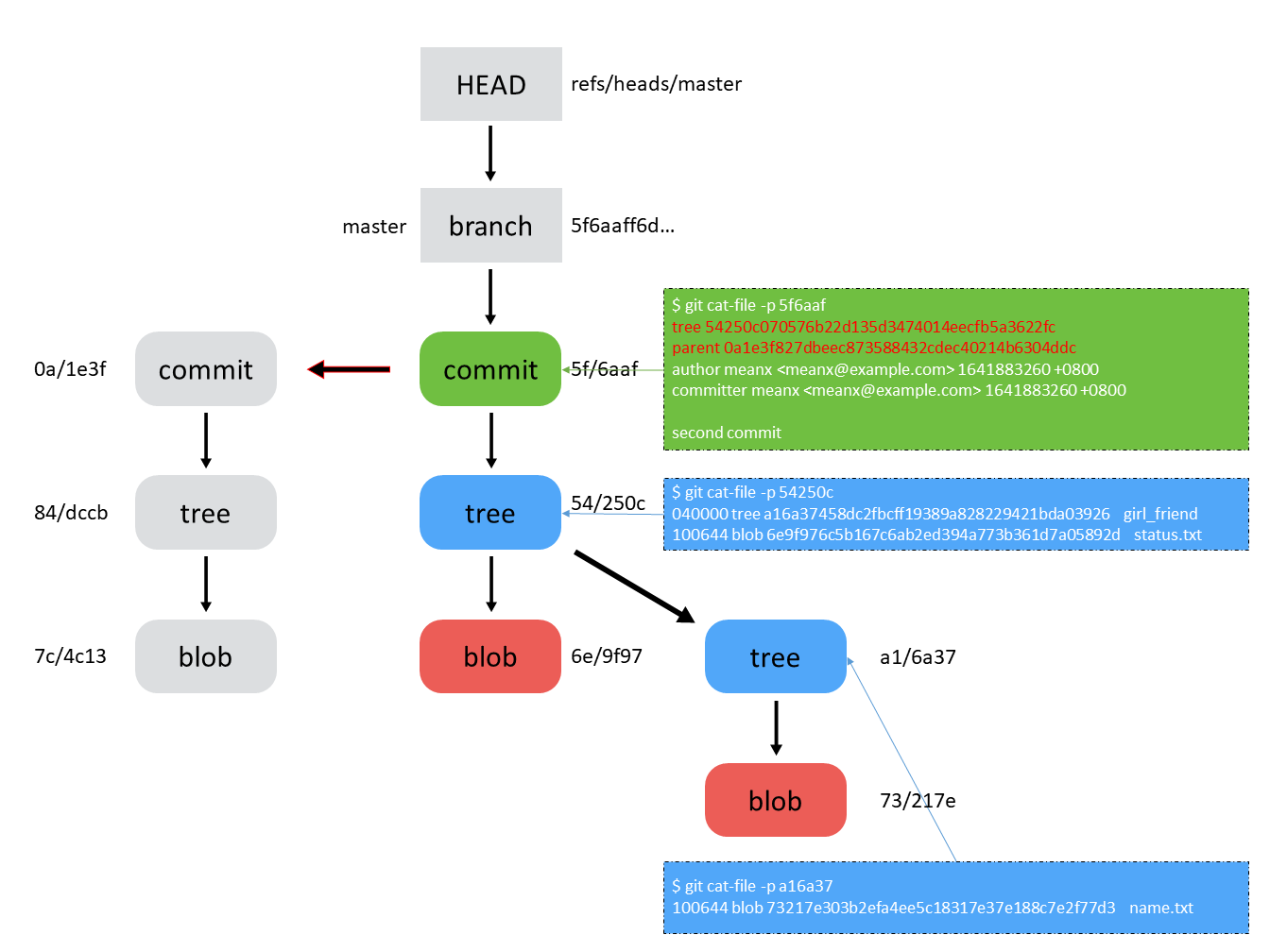


## 场景3



第二次commit 主人公更改了status.txt的内容为fall in love，新增了 girl\_friend 目录及 name.txt 文件，其 git 结构图如上所示，图中所示的git对象都存在于objects目录下，只不过那些灰色的对象并不链在当前branch的当前状态上。

只要一个文件不改变（包括内容和文件名），git就不会为该文件在提交时新建blob对象（以及该文件对应的 tree 对象）；但是，对于每一个次提交（即使是 git commit –amend），git都将重新建一个commit对象。



* 执行命令示例：

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ vim status.txt

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ mkdir girl\_friend

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ vim ./girl\_friend/name.txt

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git add .

warning: LF will be replaced by CRLF in status.txt.

The file will have its original line endings in your working directory.

warning: LF will be replaced by CRLF in girl\_friend/name.txt.

The file will have its original line endings in your working directory.

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: girl\_friend/name.txt

modified: status.txt

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git commit -m "second commit"

[master 5f6aaff] second commit

2 files changed, 2 insertions(+), 1 deletion(-)

create mode 100644 girl\_friend/name.txt

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ find .git/objects/ -type f

.git/objects/0a/1e3f827dbeec873588432cdec40214b6304ddc

.git/objects/54/250c070576b22d135d3474014eecfb5a3622fc

.git/objects/5f/6aaff6d424952282384b8dc660ee7bac84c657

.git/objects/6e/9f976c5b167c6ab2ed394a773b361d7a05892d

.git/objects/73/217e303b2efa4ee5c18317e37e188c7e2f77d3

.git/objects/7c/4c13e6e42ace6fb26c3ecbc5069a8d8435dc89

.git/objects/84/dccb5461c8b0b96224f55ddd1666ebc1b1db9d

.git/objects/a1/6a37458dc2fbcff19389a828229421bda03926

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -t 5f6aaf

commit

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -p 5f6aaf

tree 54250c070576b22d135d3474014eecfb5a3622fc

parent 0a1e3f827dbeec873588432cdec40214b6304ddc

author meanx <meanx@example.com> 1641883260 +0800

committer meanx <meanx@example.com> 1641883260 +0800

second commit

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -t 54250c

tree

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -p 54250c

040000 tree a16a37458dc2fbcff19389a828229421bda03926 girl\_friend

100644 blob 6e9f976c5b167c6ab2ed394a773b361d7a05892d status.txt

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -t 6e9f97

blob

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -p 6e9f97

fall in love

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -t a16a37

tree

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -p a16a37

100644 blob 73217e303b2efa4ee5c18317e37e188c7e2f77d3 name.txt

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -t 73217e

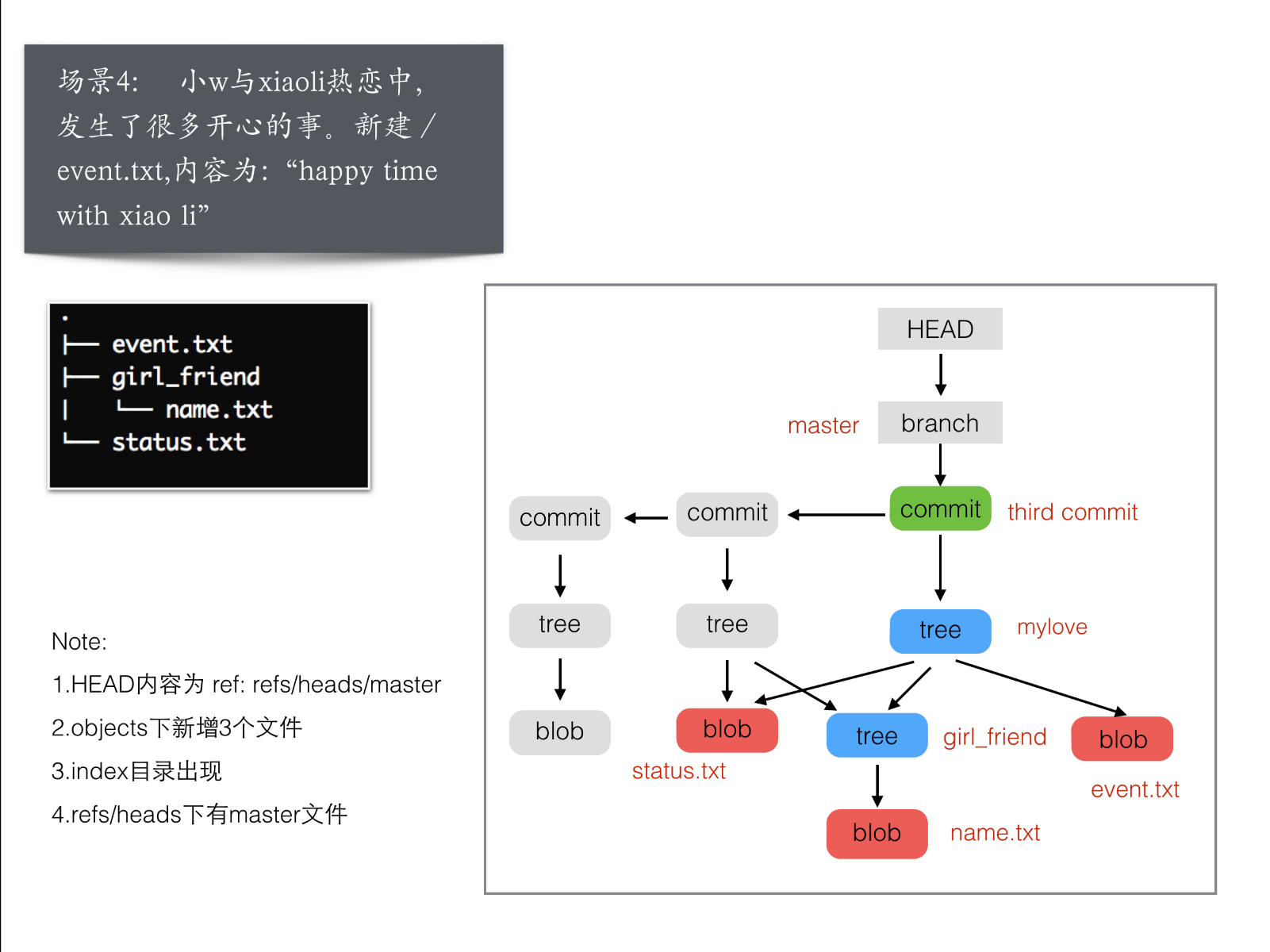
blob

Vsense@DESKTOP-I4HJMO6 MINGW64 /d/2222/mylove (master)

$ git cat-file -p 73217e

xiao li

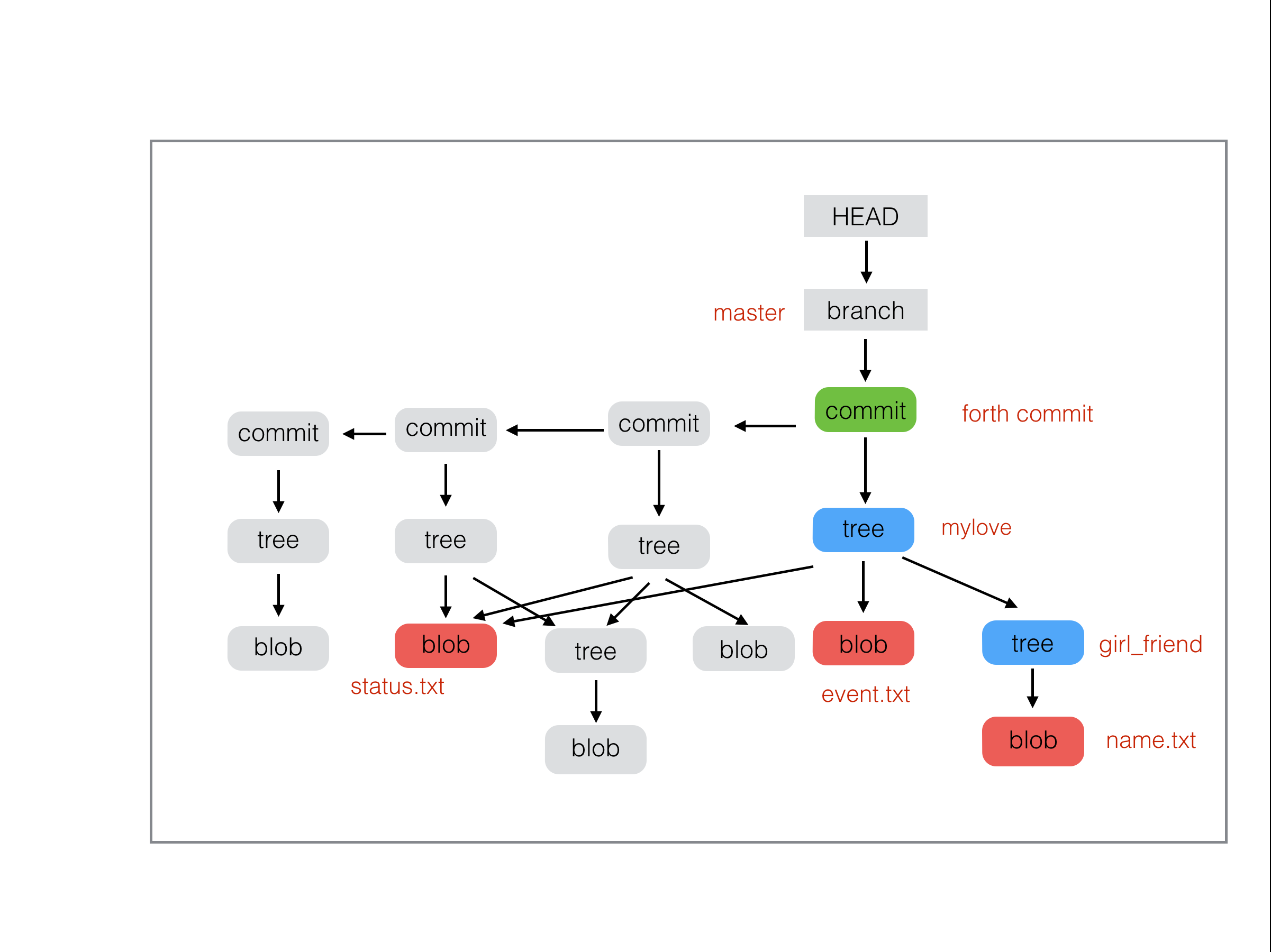
## 场景4



## 场景5

* 主人公的恋爱状态仍然是 fall in love，但是女主人公却换了，让我们一起来练习这次提交的 git 图

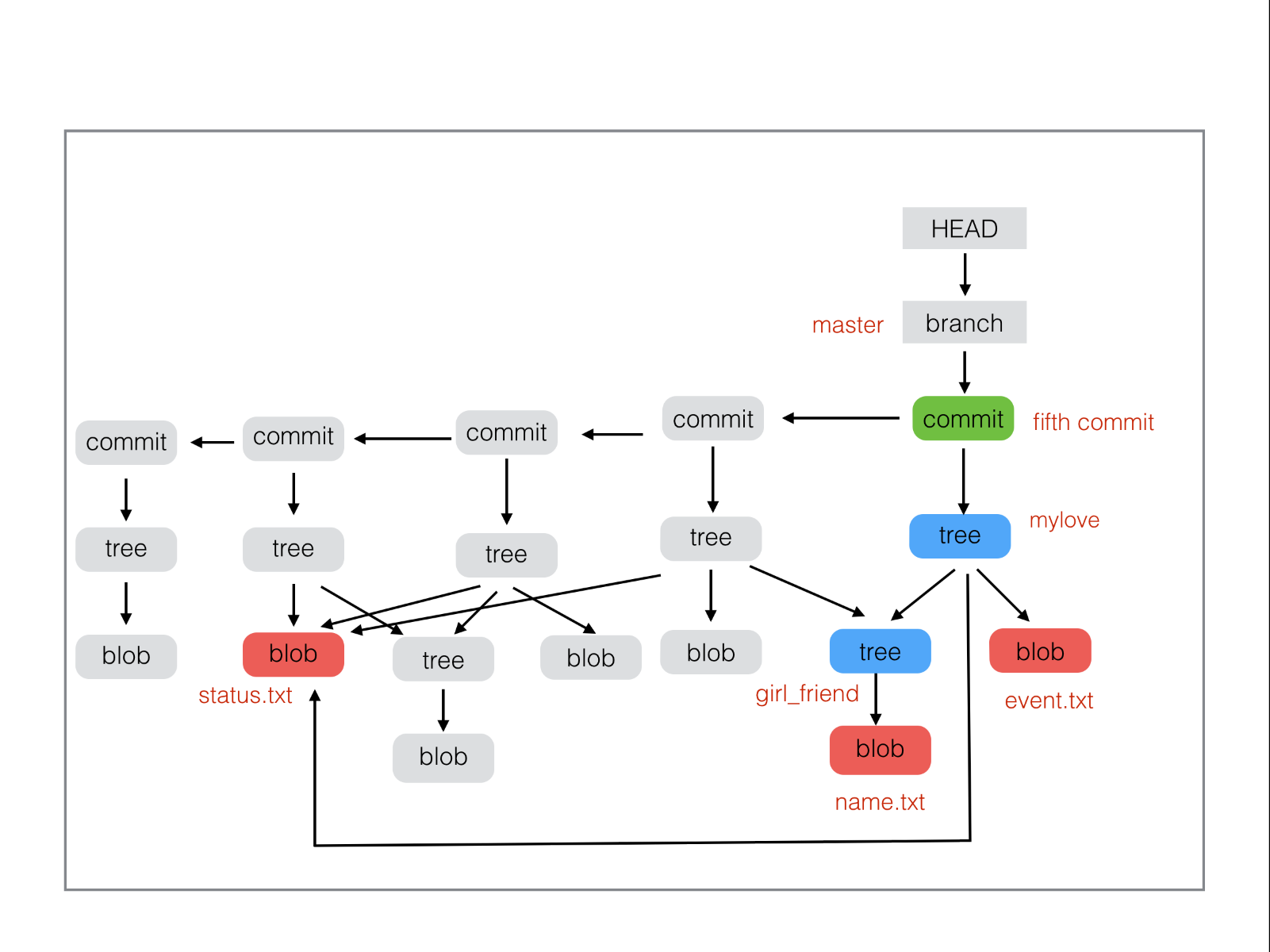




## 场景6

本场景命令: git revert one-commit

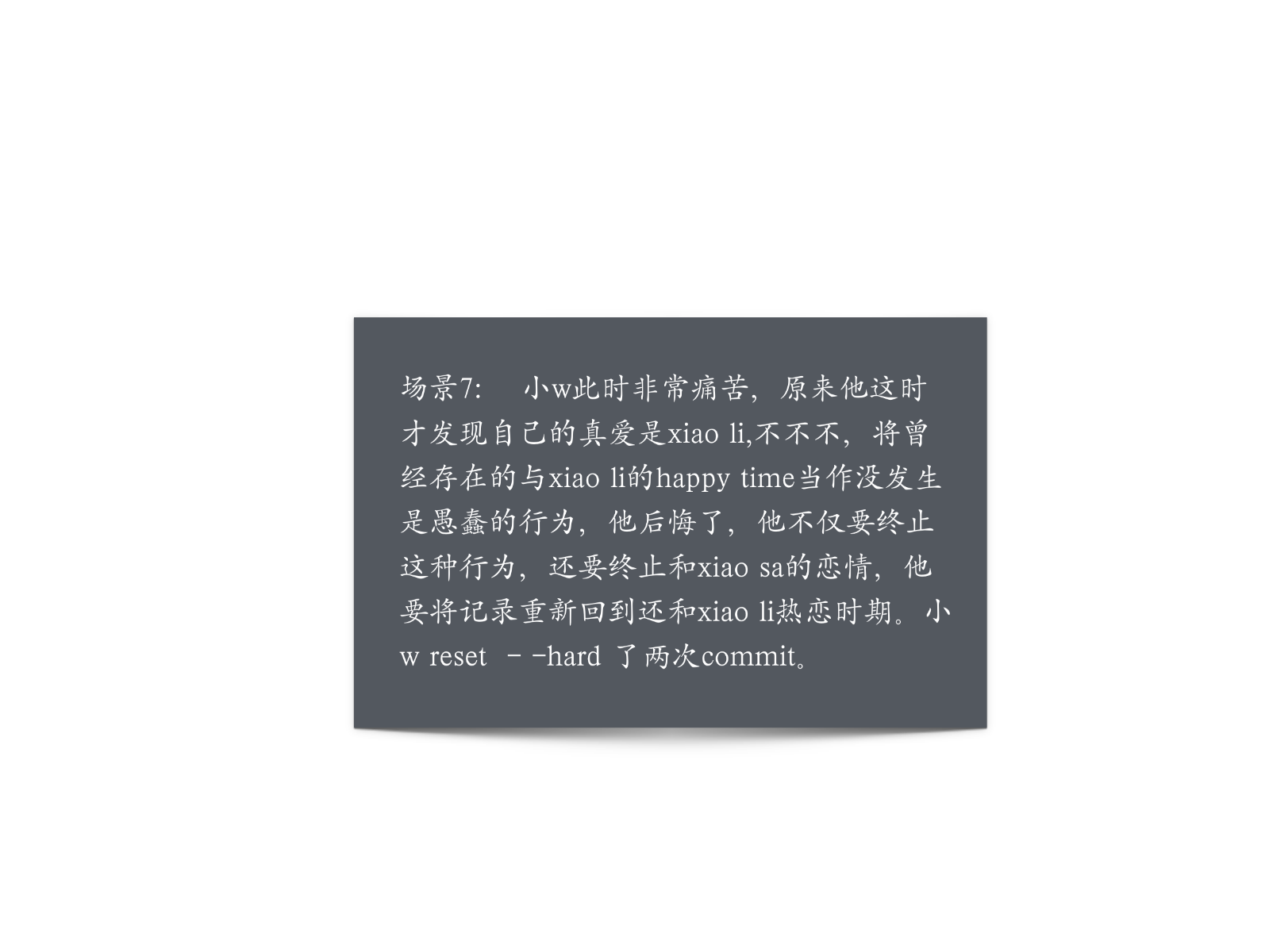


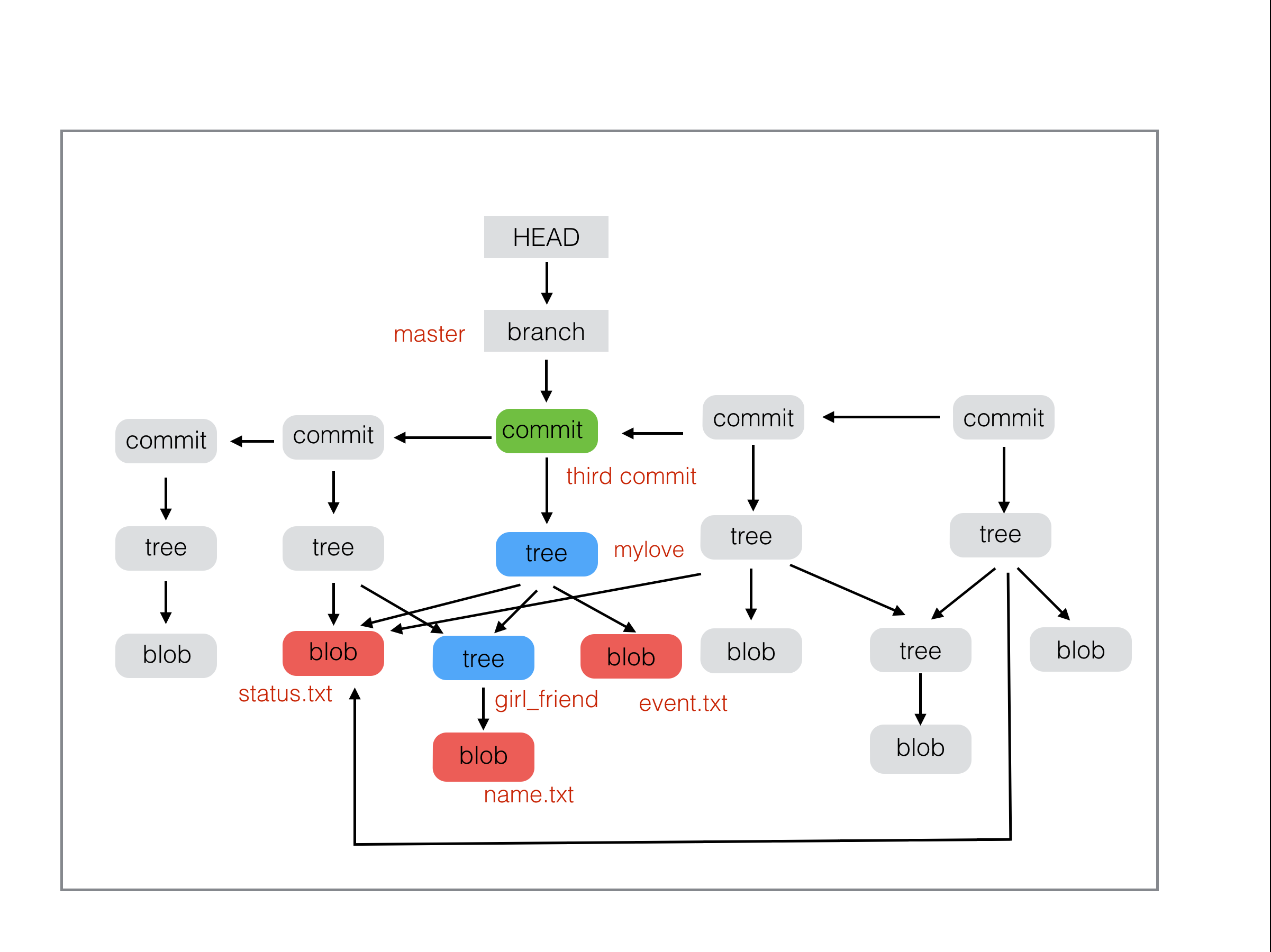


* 这次主人公要逃避现实了,他希望和xiaoli的那些happy time未曾发生，让我们回想一下第三次commit主要干了什么呢？对，主人公在第三次新增了一个events.txt，并写道 happy time with xiaoli。
* 本场景中 revert 第三次commit其实等同于 delete 当时新增的event.txt文件，此时你发现问题来了,因为该文件后来有修改，此时revert会发生conflict，需要先修复然后提交；revert会做一次新的提交。

## 场景7

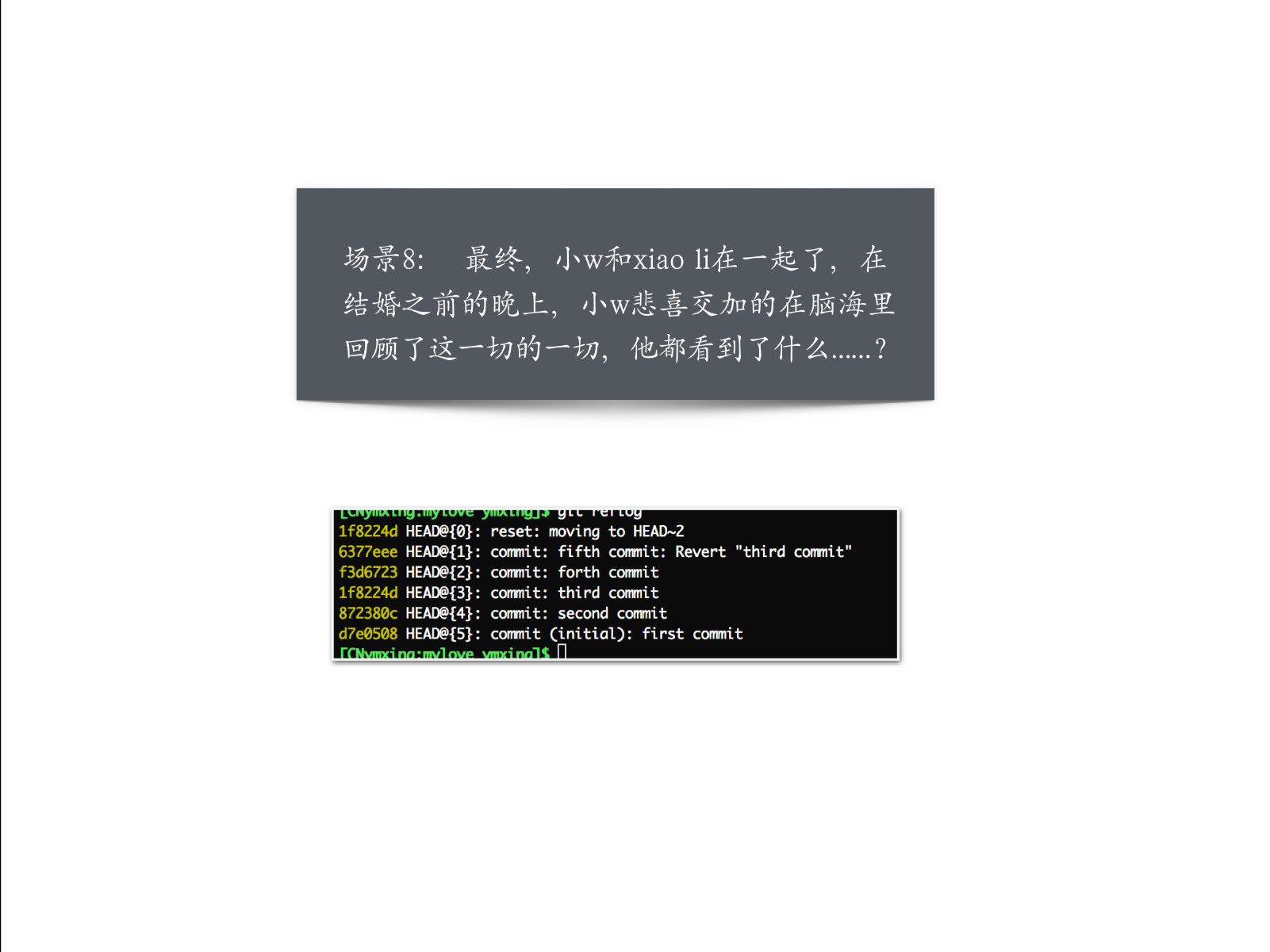
本场景命令: git reset –hard HEAD~2





## 场景8

猜一猜本场景中这个有趣的，我命名其为“主人公心路历程”的命令是什么吧? 答案是 git reflog



## 结束

