Git 使用心得分享

杨博远
hosiet@mail.ustc.edu.cn
BBS ID: hosiet

获取本演示文稿

- https://lug.ustc.edu.cn/
- “每周小聚”页面
- ftp://ftp:ftp@lug.ustc.edu.cn/weekly_party
- 内容较多，未尽内容请下载演示文稿浏览

LUG@USTC

"life,love,linux"

开始之前：遇到问题，怎么办？

- Google
- Wikipedia
- 各类 Wiki（如，Archlinux 相关问题查阅 ArchWiki）
- 正确地提问

《提问的智慧》

http://lilydjwg.vim-cn.com/articles/smart-questions.html

正题：Git 入门简介与心得体会

LUG@USTC

"life,love,linux"

Git 是什么？

- Git是一个开源的分布式版本控制 / 软件配置管理软件。(Wikipedia)

可以用 Git 做的事情：

- 管理编程源代码：个人项目、集体项目；
- 合作编写文档并用 Git 管理；
- 配合 Github/Gitcafe/... 等代码托管网站，轻松实现代码同步、参与开源项目；
- 任何需要文档版本管理的地方

Git 是什么？

- 简而言之，Git 应当被用来记录用户对文本文件的修改、编辑历史。二进制文件也不是不可以，但无法利用 git 实时查看历史更改和历史版本。

Git 尤其适合：

- 个人小项目管理；
- 联系紧密的小项目组合作开发；
- 多人协作大中型开源项目管理（如 Github 托管的各类项目）

为什么需要版本控制？

- 简单地记录所有历史，不额外增加工作量
- 设想在用LaTeX写一篇论文：
- 提纲
- 草稿
- 成稿
- 正式版
- 正式版改
- 正式最终版
- 正式最终版修改版
- 正式真的是最终版

现在问题来了：

发现正式版里的一个好想法在正式最终版中被删掉了，如何加回来？

LUG@USTC

"life,love,linux"

为什么需要版本控制？

- 如果没有版本控制，你的历史开发过程就无法回溯；
- 采用原始的文件备份进行版本控制的方式，需要手动备份，来回翻找历史记录查看文件更改，麻烦；
- 采用 Git，你只需用 git log，git checkout 和 git diff.

LUG@USTC

"life,love,linux"

Git 的来源

- 原作者：Linus Torvalds
- 起因：需要为 Linux 内核项目编写一个版本控制系统
- 初次发布：2005年

LUG@USTC

"life,love,linux"

安装 Git

- Linux 下：

用包管理器直接装 git 软件包，或编译安装（如不嫌麻烦的话）

- Mac OS X 下：

MacPorts || git-osx-installer || homebrew

- Windows 下：

msysGit 项目，TortoiseGit，等

Git 使用环境

- Linux：　　原生支持
- Mac OS X：　　支持
- Windows：　　支持（利用Cygwin）
- **注意**：以下所有操作都为命令行命令，图形化界面的使用不在讨论范围内。

Git 特性

- Git 的工作流（workflow）设计，以及它对分支（branch）操作的支持十分优秀

- Git 由一系列的小工具组成，配合使用完成工作任务。如：
- git add, git commit, git reset, git checkout, git remote, git bisect, git diff ,.....

名词翻译

英文名	中文名
commit	提交(n./v.)
merge	合并(v.)
working tree	工作树/工作区(n.)
branch	分支(n.)
checkout	检出(v.)
stage, index	暂存区(n.)

一例：从头创建 Git 工作环境

- 举一个从无到有，创建一个使用 Git 管理的项目的例子：
- 以 /tmp/tmp 目录作为项目的工作目录，在其中创建一个 README 纯文本文件，并使用 Git。

第一步：初始化 Git 版本库

1. 进入工作目录（如，使用cd），并输入

```
$ git init
初始化空的 Git 版本库于/tmp/tmp/.git/
```

2. 初始化配置：

- `git config --global user.name "Your Name"`
- `git config --global user.email "email@example.com"`

第一步：初始化 Git 版本库

- 查看当前工作树状态：

```
$ git status
位于分支 master
无文件要提交，干净的工作区
```

- 随时运行 git status 即可查看当前状态，工作情况尽在掌握之中。

第二步：编辑文件

3. 创建一个新文件并写入内容：

```
$ touch README
$ echo "Hello World!" > ./README
```

- 查看文件内容：

```
$ cat ./README
Hello World!
```

第二步：编辑文件

- 查看当前工作树状态：

```
$ git status
```

位于分支 **master**

初始提交

未跟踪的文件：
（使用 "git add <file>..." 以包含要提交的内容）

README

提交为空，但是存在尚未跟踪的文件（使用 "git add" 建立跟踪）

第三步：添加到暂存区

4. 添加 README 文件至暂存区：

```
$ git add ./README
```

- 查看当前工作树状态：

```
$ git status
```

位于分支 **master**

初始提交

要提交的变更：
（使用 "git rm --cached <file>..." 撤出暂存区）

新文件：　　**README**

第四步：进行提交

5. 进行初始提交：

```
$ git commit -m 'First Commit'

[master （根提交） f99f305] First
  Commit
 1 file changed, 1 insertion(+)
 create mode 100644 README
```

========完成========

我们刚才做了什么？

- 创建了一个空的 Git 版本库
- 将一个新文件添加到了暂存区里
- 进行了一次提交，将暂存区文件加入版本库中

- 版本库存在工作目录下的`.git`文件夹下，包含所有的 commit 历史对应的完整文件。

- **所有**的，**完整**的。

LUG@USTC

"life,love,linux"

Git 基本工作原理

- **内容分区**

1. 工作区（你能看见的，工作目录内的所有文件）
2. 暂存区（在.git目录的版本库内，记录你修改过/新创建/想要删除，并打算提交的文件，有临时的性质）
3. <历史提交>（在.git目录的版本库内，记录所有提交的历史）

Git 基本工作原理

- **唯一性标识**

- 工作中进入版本库的每一个 commit, file, tag,... 都有一个 SHA1 哈希散列值与其对应。
- 可以通过哈希值快速定位、代表对应的提交或是文件。

举例

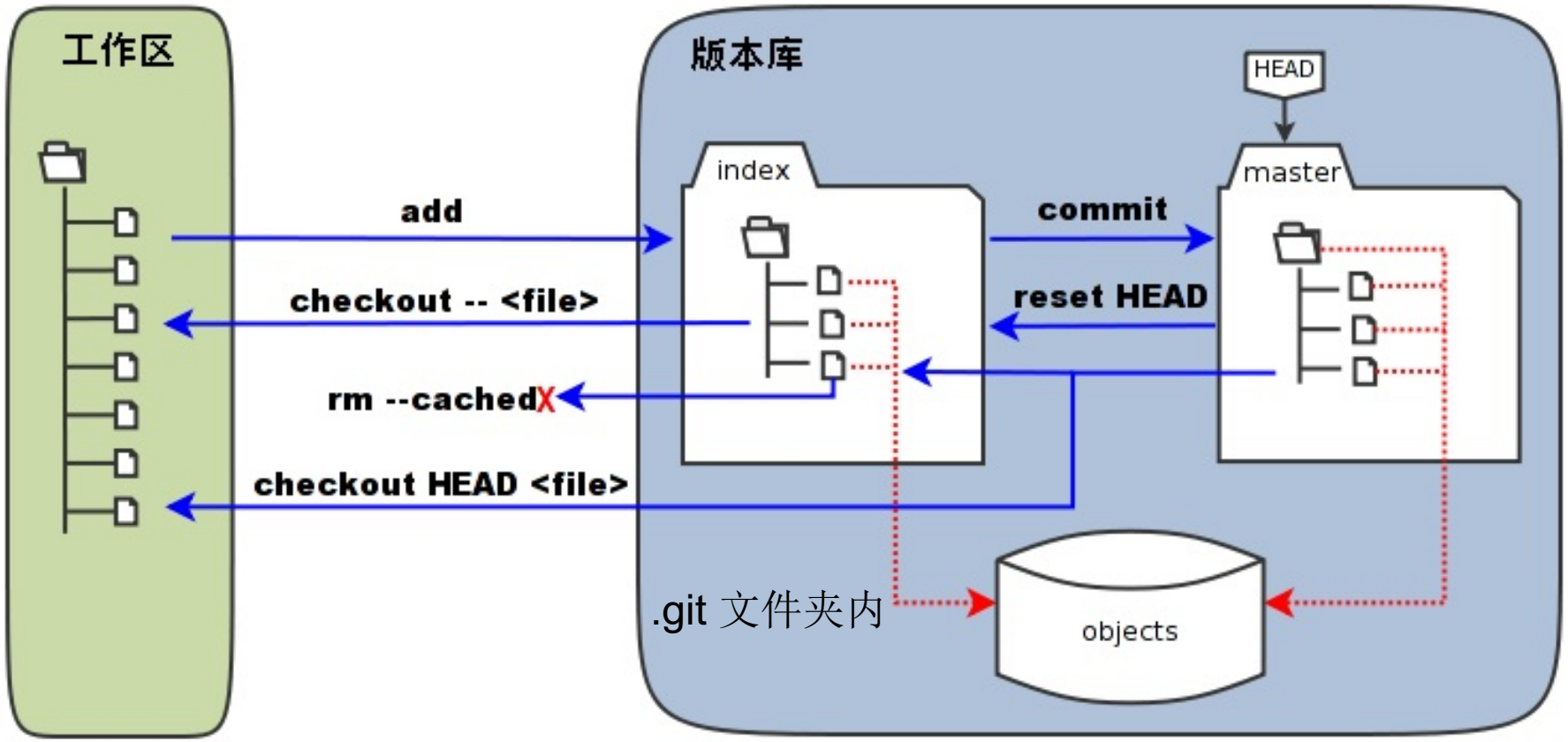
- 一次提交的简略信息：

commit
b28ed1152aae3d15adba350ccb862977d54e6bb9
Author: Boyuan Yang <073plan@gmail.com>
Date: Sun Oct 5 21:02:16 2014 +0800

[Fix warning] fix warning in web/ajaxupload.c

Git 基本工作原理

一张（应当）非常明白的图：



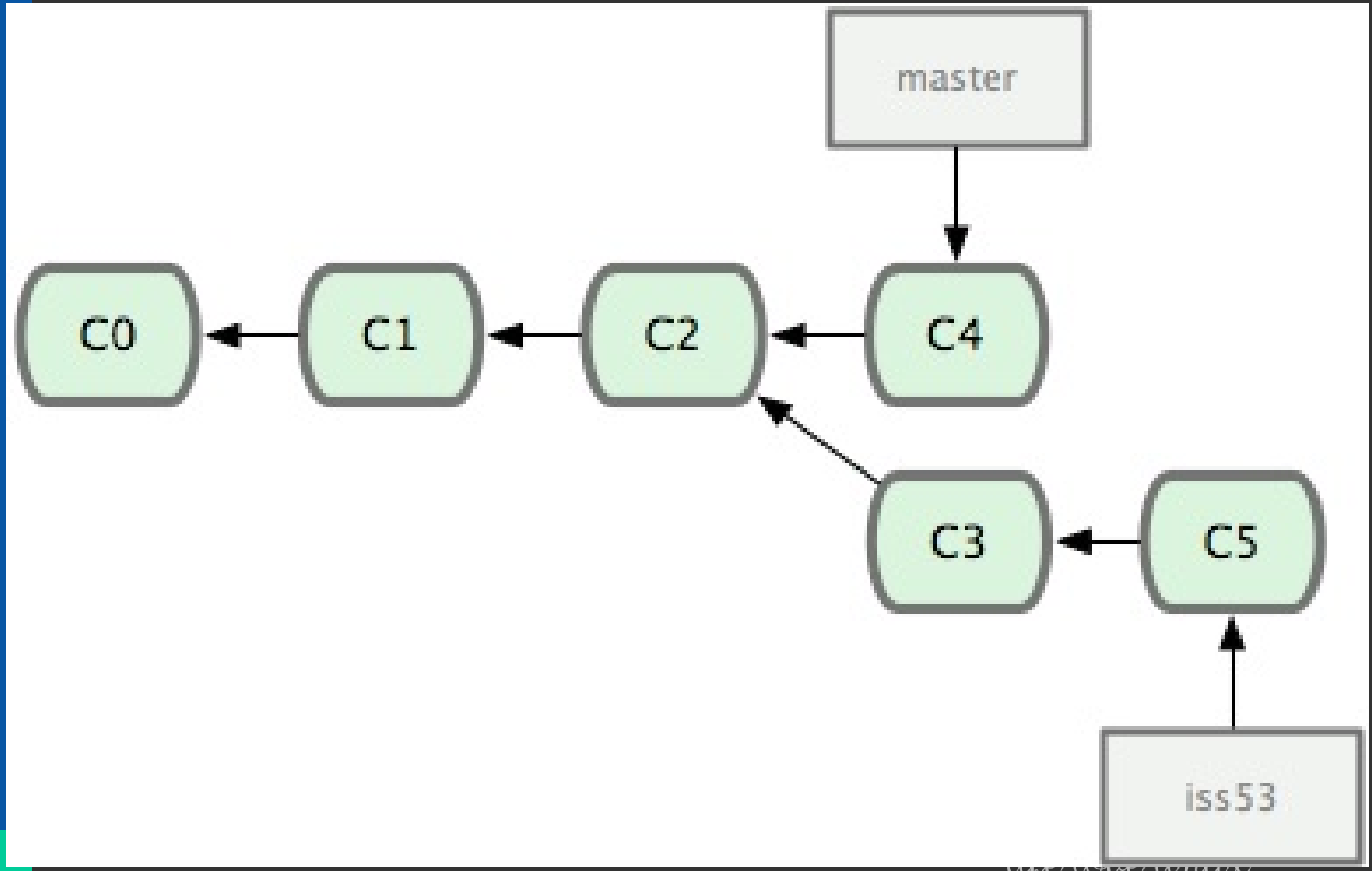
Git 基本工作原理

- 以每个提交（commit）为一个节点，开发历史可以是线性结构（单线开发）、树状结构（产生分支）、复杂的网状结构（一般情况）
- Git 对分支的生成、合并，并行合作开发有较好的支持，简易使用只需了解 git branch 命令和 git merge 命令。

简易理解分支、HEAD、标签

- 认为所有的 commit 都是一个节点，带有指向其父节点的指针
- 那么分支名(branch)，头指针(HEAD)，标签名(tag)都是指向特定节点的指针。
- 特别地，HEAD 指向的节点代表当前正在进行工作的节点，即对应工作区内（未修改过）的文件对应的版本。

Git 使用进阶：分支



分支相关命令

- git branch <branchname>
- 创建一个分支，名为 branchname ， 指向当前工作位置（HEAD）
- git checkout <branchname>
- 丢弃当前工作进度，切换到 branchname 分支。

分支相关命令

- git merge <branchname2>
- 将 branchname2 这个分支合并到当前工作点(HEAD)上。如果有无法自动解决的冲突，则需要手动解决。
- git diff HEAD HEAD^^
- 显示 HEAD 与 HEAD 的父节点的父节点的差异

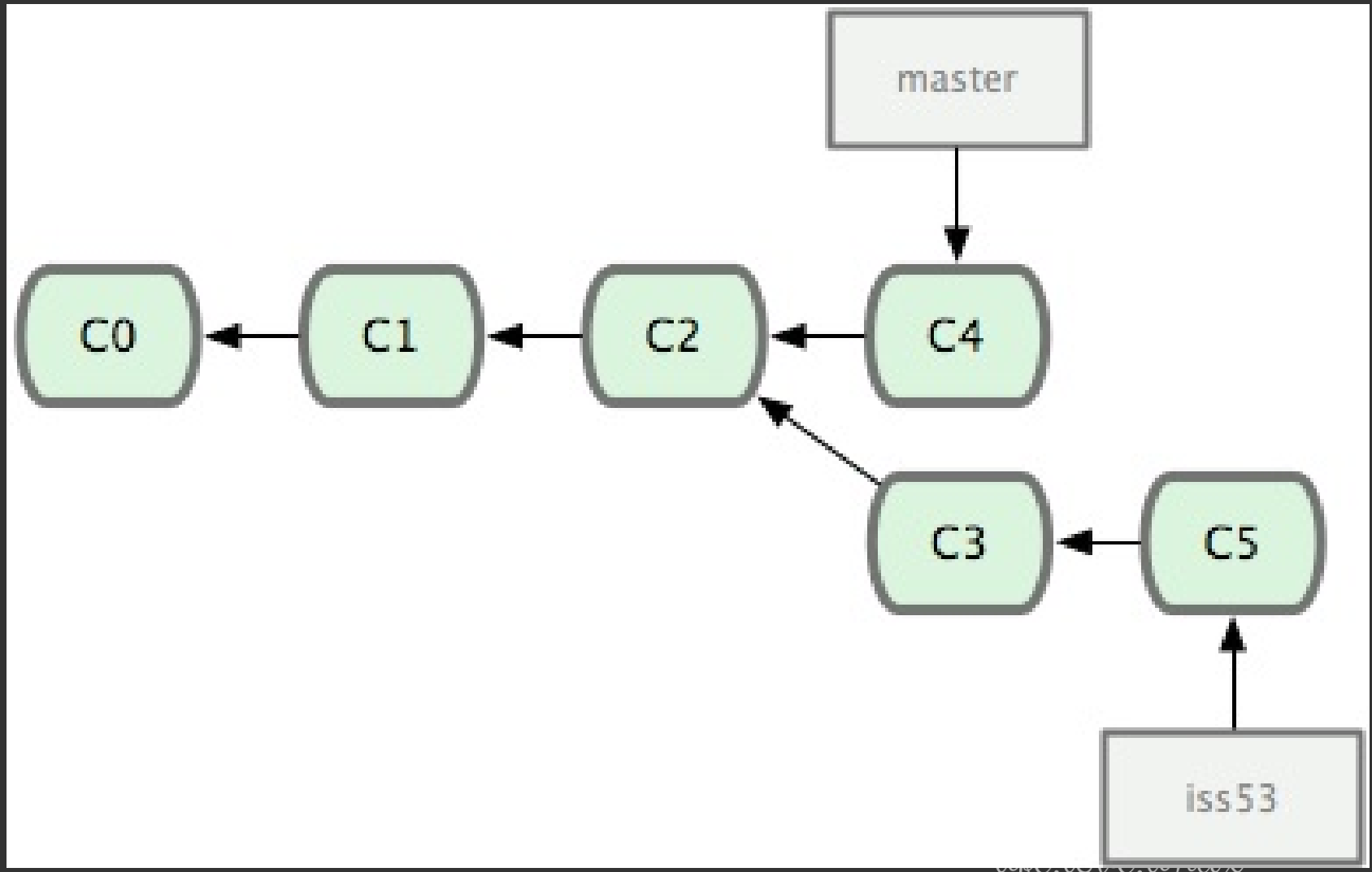
分支相关命令

- git branch -d <branchname>
- 删除某个名为 branchname 的分支。
- **注意**
- 要避免出现无法被索引到的提交节点
- 即，无法通过标签名/分支名/HEAD指针向前回溯得到的提交节点

Git 使用进阶：分支

- 仍然以之前的项目作为例子：

简易理解分支、HEAD、标签



Git 使用进阶：分支

注：

- 用户没有用过 branch 功能时，存在默认的分支，名为 master。作为默认值，具有一定的特殊性。
- 可通过 git log --graph 或者图形化工具得到直观的分支展示图。

克隆版本库：Git clone

- 克隆他人的 git repository：
- `git clone https://github.com/ustclug/liimstrap.git`
- 可以使用本地/ HTTP[S]/ FTP[S]/ GIT / SSH / RSYNC 协议进行clone。
- 如**clone**之意，每个**clone**都是一份完整的代码及历史。

LUG@USTC

"life,love,linux"

代码托管：与 Gitlab 共同使用

- 虽说是 Gitlab，但是 Github 的使用也大体相同。
- USTC LUG 自建 Gitlab 服务器：

```
https://gitlab.lug.ustc.edu.cn
https://git.ustclug.org
```

- 以上是同一台服务器，对外公开，可以创建私有项目，欢迎使用。

代码托管：与 Gitlab 共同使用

- 在 Gitlab 网站上创建一个对应的项目，则会有初始化项目的指南出现，请按照说明进行即可。这里时间关系不再赘述。

- 内容无非是：
- git init　　初始化
- git remote 设置本地仓库与远程仓库的对应
- git push　　向远程仓库推送

代码托管：与 Gitlab 共同使用

- 觉得麻烦？
- 对于 Github，可以在网站上创建好 Git repository（含 README 文件），直接 clone 到本地即可开始工作。

代码托管：与 Gitlab 共同使用

- 工作中你会经常用到：
- git pull，从远程获取最新更改(提交)，自动合并
- git push，将本地的新提交推送到远程仓库
- 多分支情况类似，但会有更多参数以指定所操作的分支

Git 常用命令一览

- 以下列出常见的一些 Git 使用命令，应该能覆盖到 85% 的使用情况。

- <参数> 表示必填参数；
- [参数] 表示选填参数，不填将使用默认值或默认选项。

Git 常用命令一览（1）

- git init　　初始化版本库
- git add <file>　添加本地文件至暂存区
- git add . 如上，添加当前目录下所有文件
- git commit　　进行一次提交
- git checkout -- <file> 用暂存区对应文件替换工作区对应文件
- git checkout .　丢弃当前工作区文件所有更改，用暂存区文件替换之
- git checkout <branchname> -- <file>

使用 branchname 分支中的文件替换工作区中的对应文件。

注

- git checkout 命令功能强大，在有 / 无文件名或路径的两种情况下行为完全不同，请注意仔细辨别。
- 有文件名/路径：≈ 检出文件/文件夹
- 无文件名/路径：≈ 切换分支

如，下页的 checkout 功能与上一页的完全不同。

Git 常用命令一览（2）

- git checkout HEAD （丢弃所有当前更改）
- git checkout <tag> 切换至tag对应的提交
- git checkout <branch> 切换至指定分支

以上三条命令会丢弃工作区与暂存区所有已有修改。

- git reset HEAD 用 HEAD 提交覆盖暂存区内容
- git reset --hard HEAD^ 完全丢弃上一提交。（短时间仍可恢复）

Git 常用命令一览（3）

- git log --graph

查看历史提交记录（带分支示意）

- git rm --cached <file>

去除新添加到暂存区的文件

- git branch <branchname>

新增一个分支，且指向 HEAD

- git branch -d <branchname>
- 删除名为 branchname 的分支

Git 常用命令一览（4）

- git clone https://***.***/***.git

克隆远程版本库

- git fetch [来源] [分支]

获取某远程分支（的最新更改）

- git pull [来源] [分支]

从远程获取某分支的最新更改，自动合并

- git merge [来源] / <branchname>

将指定分支合并到当前所在分支（HEAD）

- git pull = git fetch + git merge

Git 常用命令一览（5）

- git push [来源] / [本地分支]:[远程分支]

将某个本地分支推送到远程的某分支

- git diff <参数A> <参数B>

将A和B进行比较，输出 diff -u 格式至标准输出。

两个参数可以为分支名、提交SHA1散列值、标签名等。

Git 常用命令一览（6）

- git status　　查看工作树状态

- git show <参数>

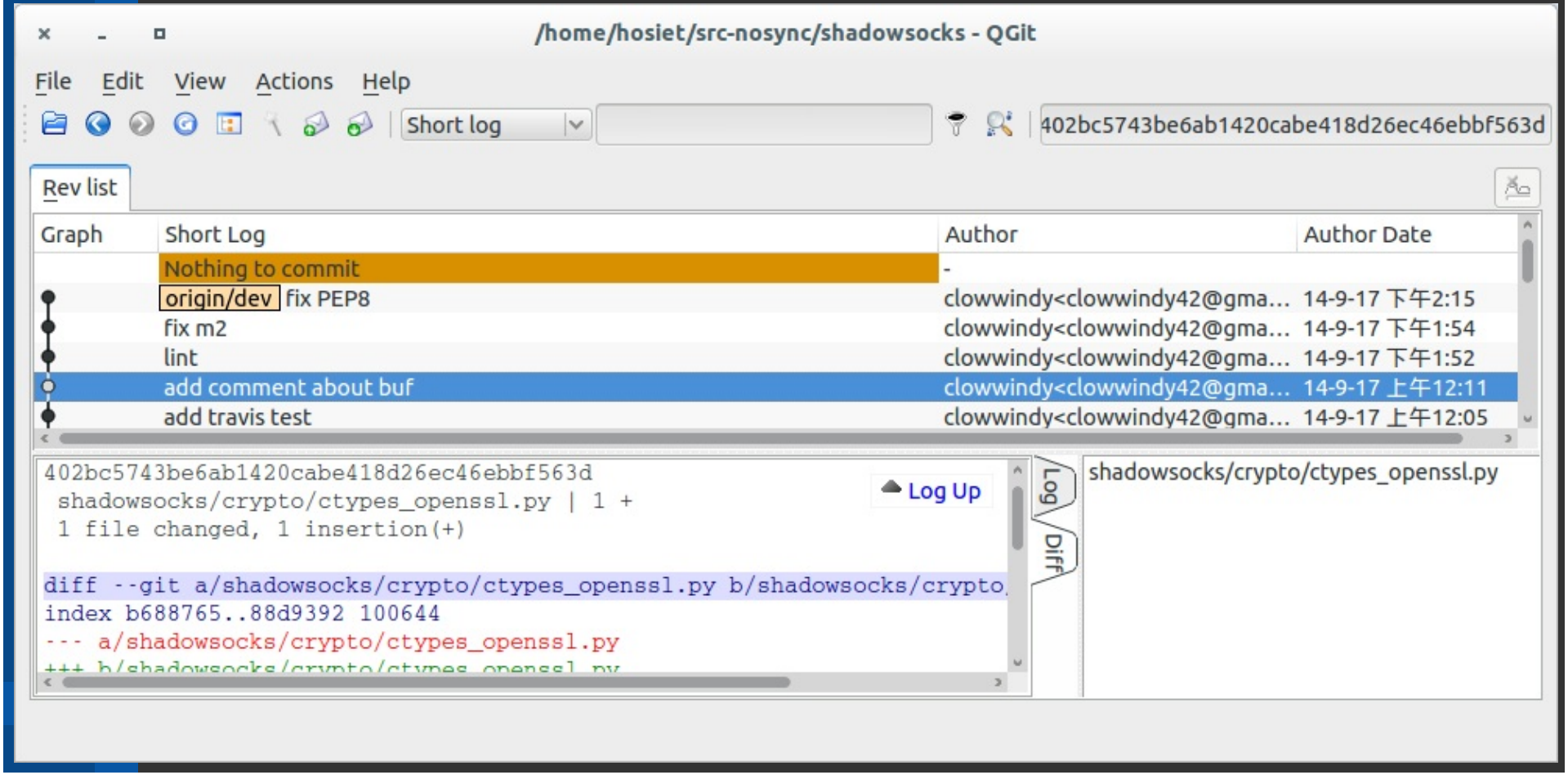
显示对象详细信息。

参数可以为分支名、提交SHA1散列值、标签名等。

注：使用 git show 应当使用较新版本的 Git。

图形化工具推荐

- qgit：用 Qt 开发的 git repository viewer
- 更多选择：http://git-scm.com/downloads/guis



心得：commit 技巧

- 每个 commit 必须设置一个说明信息。
- 设置书写 commit message 所用的编辑器：

```
git config core.editor="vim"
```

- Git 推荐多次少量的 commit，一个 commit 实现一个特定功能/写完一段文字，进度分明易于回溯或进行 git revert。

心得：Win 与 Linux 协同工作

- Windows 中文环境下的默认文本文件编码为 GBK，默认换行符为 CRLF
- Linux 下默认文本文件编码为 UTF-8，默认换行符为 LF
- 问题体现：我不说你也明白

心得：Win 与 Linux 协同工作

解决办法：如有问题，可以考虑 git config 添加以下配置

- Linux 下配置：

```
core.autocrlf = input
core.safecrlf = true
```

- Windows 下配置：

```
core.autocrlf = true
core.safecrlf = true
```

LUG@USTC

"life,love,linux"

未涉及的内容

- git rebase
- git bisect
- git revert
- git fsck
- ...

- 实践是最好的老师。选一个项目，然后就开始使用 Git 吧！

扩展阅读

- 托管在 Github 上的 Git 介绍网站
http://git-scm.com/

- man 手册页：git / git <git命令>

- 维基百科：Git
http://zh.wikipedia.org/zh-cn/Git

- LUG 西区活动室有一本详尽的 Git 参考书，欢迎借阅

参考

- http://gogojimmy.net/2012/01/17/how-to-use-git-1-git-basic/
- http://www.worldhello.net/2010/11/30/2166.html
- Wikipedia：Git
- http://git-scm.com/
- etc.

获取本演示文稿

- https://lug.ustc.edu.cn/
- “每周小聚”页面
- ftp://ftp:ftp@lug.ustc.edu.cn/weekly_party
- 水平有限，如有错误请各位提出以改正

LUG@USTC

"life,love,linux"

Q & A

LUG@USTC

"life,love,linux"

版权声明

- 所有原创内容，以 CC-BY-SA 4.0 条款授权；
- 所有引用的文字、图片等内容，其版权以其作者的声明为准。

LUG@USTC

"life,love,linux"