网易微专业之《前端开发工程师》

学习笔记

开始时间：2016.2.27

《产品前端架构》

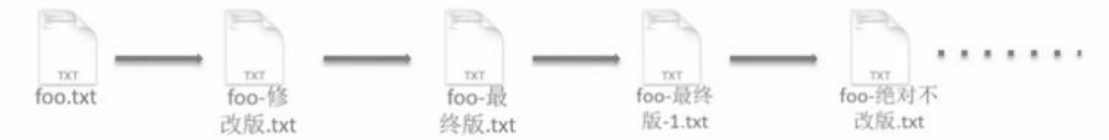
版本管理

版本控制系统简介

版本控制系统，即 VCS(Version control system)，是一种记录若干文件的修订记录的系统，它帮助我们查阅或回到某个历史版本。

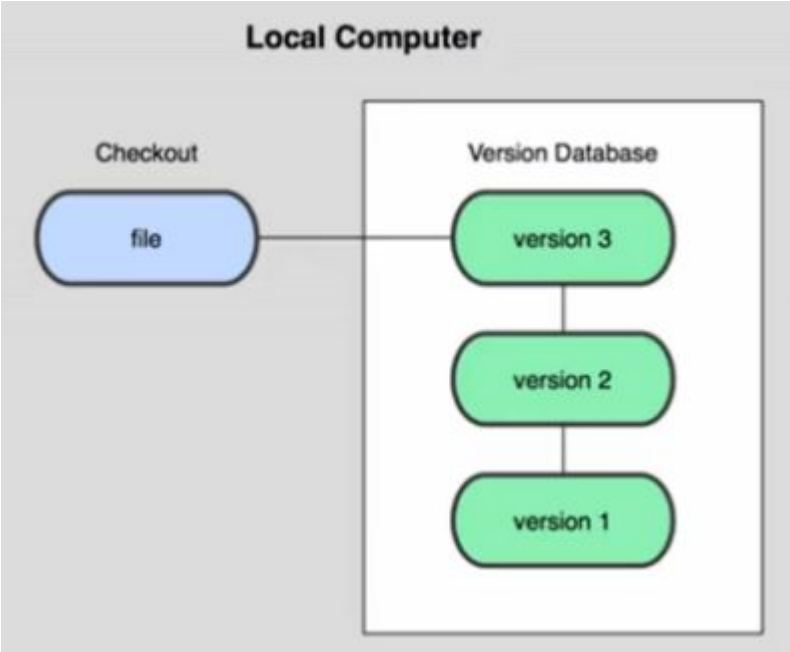
- “人肉”VCS
- LVCS 本地
- CVCS 集中式
- DVCS 分布式

“人肉”VCS



LVCS 本地式

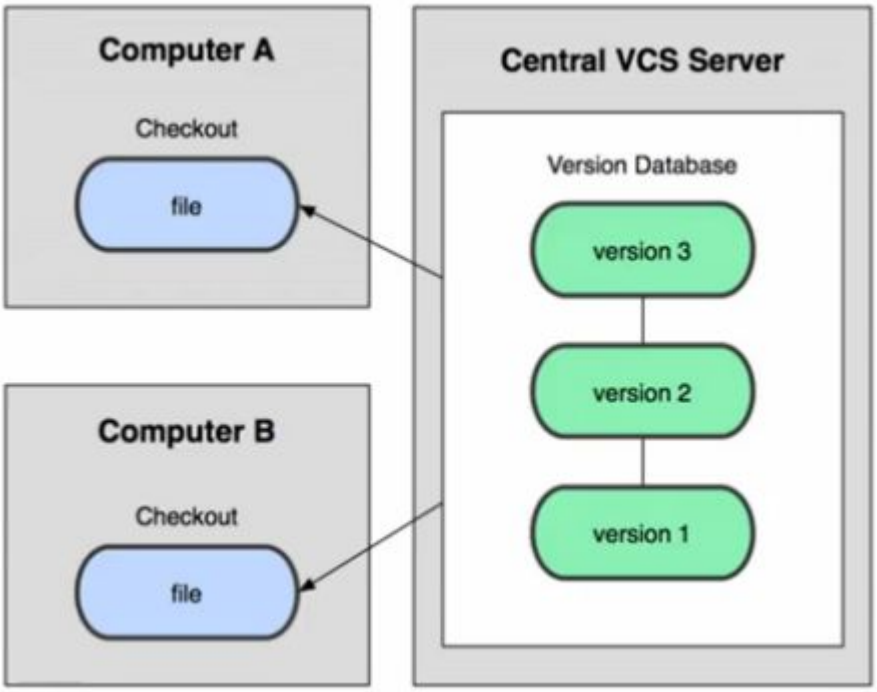
举例：RCS(Revision Control System)



CVCS 集中式

举例：

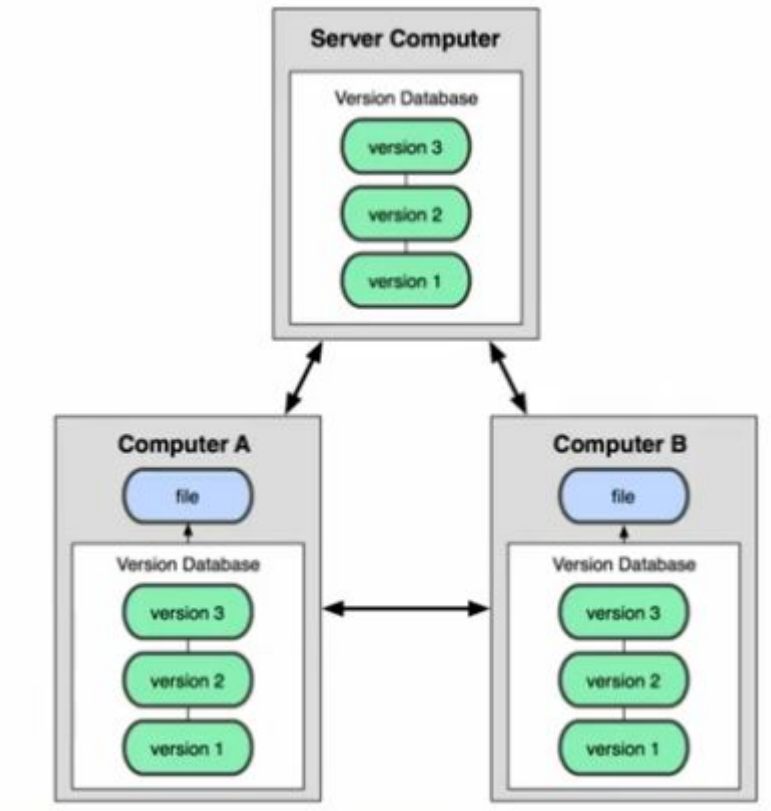
- CVS(Concurrent Versions System)
- **SVN(Subversion)**
- Perforce



DVCS 分布式

举例：

- Git
- Mercurial



VCS(版本控制系统)优点：

- 从当前版本回退到某个历史版本；
- 查看某个历史版本；
- 对比两个版本的差异。

分布式版本控制(DVCS) 对比集中式版本控制系统(CVCS)的优缺点：

分布式版本控制(DVCS)

优点：

1. 适合分布式开发，强调个体

PS：允许支持上千个并行开发的分支

2. 公共服务器压力和数据量都不会太大
3. 速度快、灵活

PS：特别在打分支和打 Tag 时候

4. 任意两个开发者之间可以很容易的解决冲突
5. 离线工作

PS：本地仓库

缺点：

代码保密性差，一旦开发者把整个库克隆下来就可以完全公开所有代码和版本信息。

集中式版本控制系统(CVCS)

优点：

1. 每个人可以看到别人做了什么
2. 管理员管理权限也比较简单

缺点：

依赖中央服务器，存在单点故障的风险。

1. 中央服务器宕机后，都无法协同工作
2. 如果中央服务器的文件损毁，又没有备份时，丢失的数据无法找回

分支模型

如果多人并行在一条线上开发会导致开发困难并且难以定位错误位置。

分支和分支模型

分支：

从目标仓库获得一份项目拷贝，每条拷贝都有和原仓库功能一样的开发线。

分支模型（branching model）/工作流（workflow）

一个围绕项目【开发/部署/测试】等工作流程的分支操作（创建、合并等）规范集合。

产品级的分支模型

1. 常驻分支

常驻分支一旦被创建，就不会被修改。

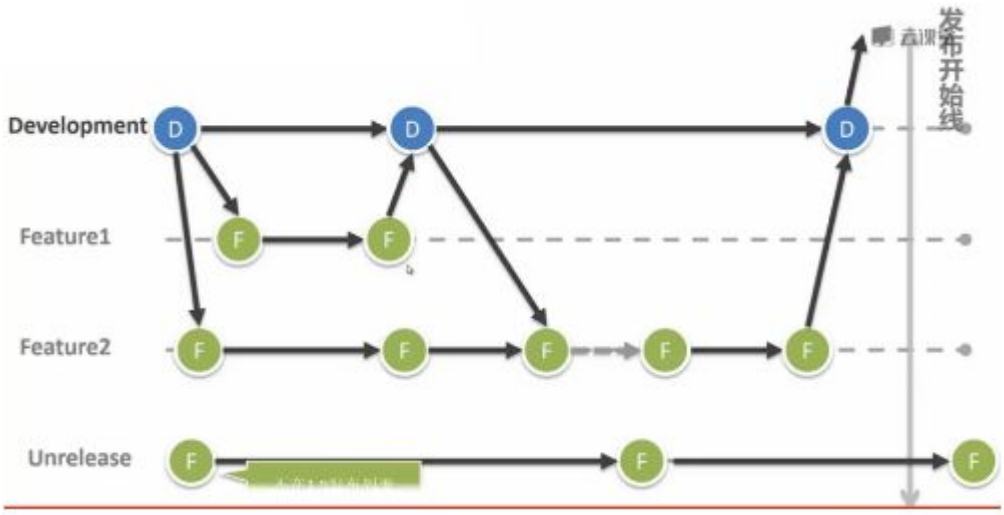
- development：从 master 创建
- production（master）：默认分支可用于发布的代码

2. 活动分支

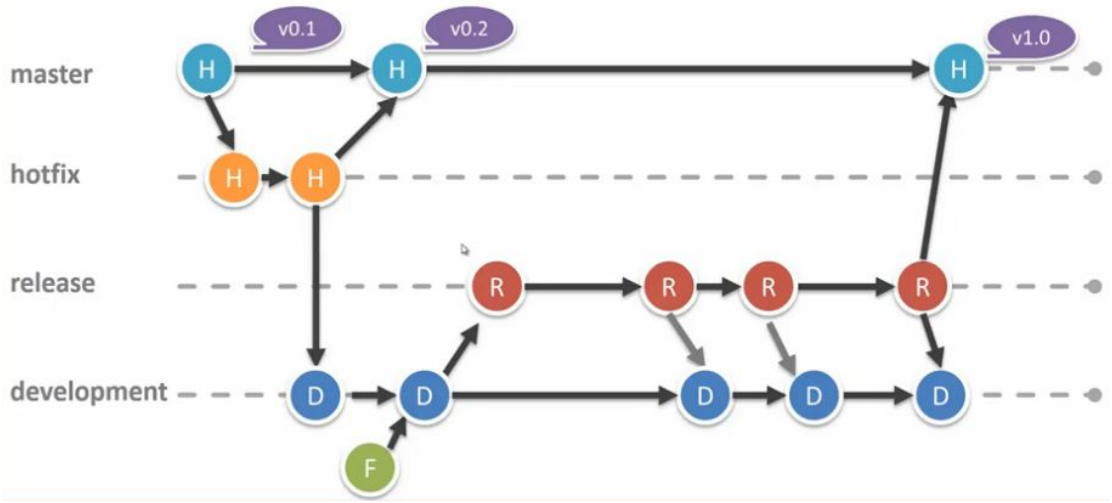
活动分支会跟着产品的发布而动态创建，有时在完成开发后也会将其删除，只保留版本号。

- feature：从 development 分支创建
- hotfix：如 hotfix-36，从 master 创建，用于修复 Bug
- release:如 release-110，从 development 分支创建 ,标志着一个产品正式发布

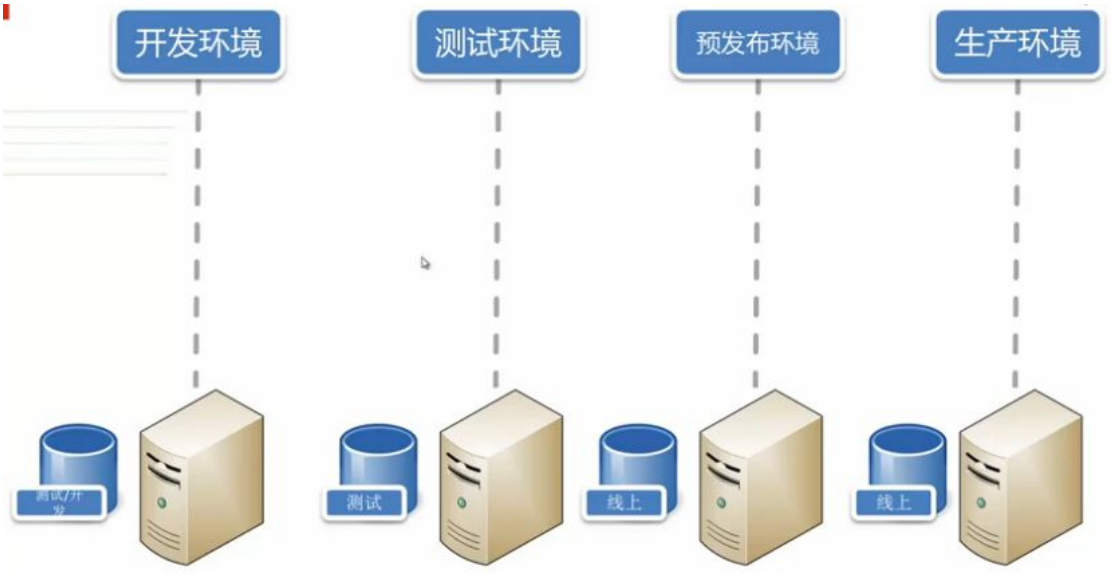
分支模型-特性开发



分支模型-发布线



环境：



- 开发环境，使用测试开发配置（数据库，缓存，元数据配置）：使用提交到下一个 release 的特性分支
- 测试环境，使用测试配置（测试数据库）：使用 release/development
- 预发布环境，小范围发布使用线上数据库模拟真实环境：使用 release
- 生产环境，线上配置：使用 master

Git

Git 是一个免费开源的分布式版本控制系统，它也一个基于内容寻址的存储系统。 Git 是由 Linux 的创造者 Linus Torvalds。

Git 历史：

- git 的出现离不开 linux
- 1991-2002:几乎无版本控制（patch 包）
- 2002-2005：BitKeeper

- 2005-至今：git

优势

- 速度快，不依赖网络
- 完全分布式
- 轻量级的分支操作
- Git 已经成为现实意义上的标准
- 社区成熟活跃

安装

Windows: msysgit ,下载地址：https://git-for-windows.github.io/

Mac OS： 使用 `brew install git` 命令。

Linux Ubuntu： 使用 `apt-get install git` 。

Git 命令详解

当在命令行中键入 `git` ，便可以在帮助信息中看到常用 Git 命令：

```
Administrator@PC-20151201VPJY MINGW32 ~
$ git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

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
   reset      Reset current HEAD to the specified state
   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
   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
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Forward-port local commits to the updated upstream head
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

获取帮助文档：

- git help <command>
- git <command> -h
- git <command> --help
- man git-<command>

Git 基本操作

`git config` ---用户配置

用户配置：创建 Git 仓库前必须完成的配置。

git config --global user.name “Cary Peng”

git config --global user.email xifengxx@163.com

配置级别：

- `--local` 默认 高优先级：只影响本仓库 `.git/config`
- `--global` 中级优先级：影响到所有当前用户的 `git` 仓库 `~/.gitconfig`
- `--system` 低优先级：影响到全系统的 `git` 仓库 `/etc/gitconfig`

git init ---初始化仓库

git init [path]

git init [path] --bare

```
.
$ git status
fatal: Not a git repository (or any of the parent directories): .git

$ git init
Initialized empty Git repository in
/Users/leeluolee/code/tmp/gith/test/.git/

$ git status
On branch master
Initial commit
nothing to commit (create/copy files and use "git add" to track)
```

```
working-dir
└── .git
    ├── HEAD
    ├── branches
    ├── config
    ├── description
    ├── hooks
    ├── info
    ├── objects
    └── refs
```

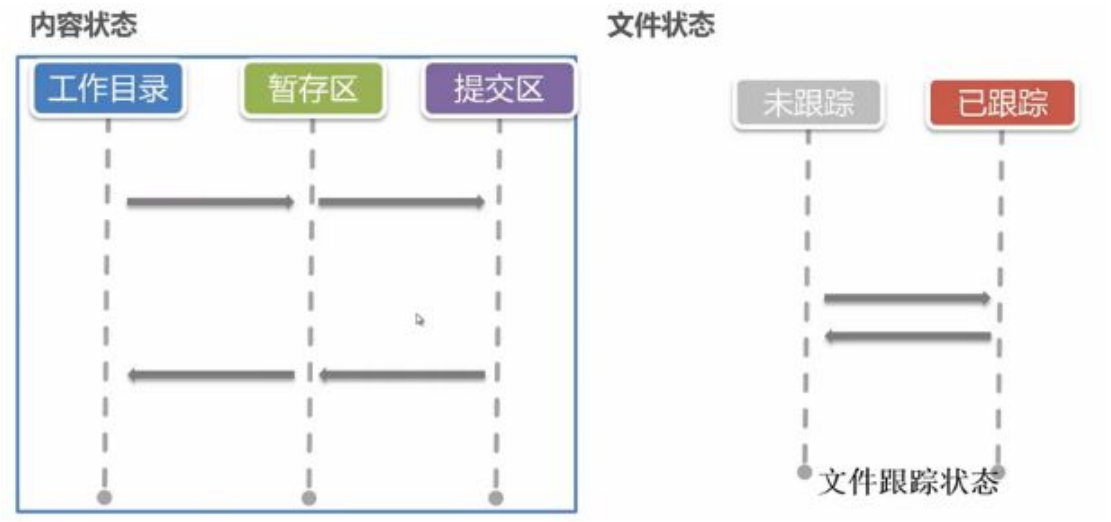
在初始化仓库后会出现一个隐藏的目录 `.git` 其中包括了所有的当前仓库的版本信息和本地设置文件(`.git/config`)。

git status---查询状态

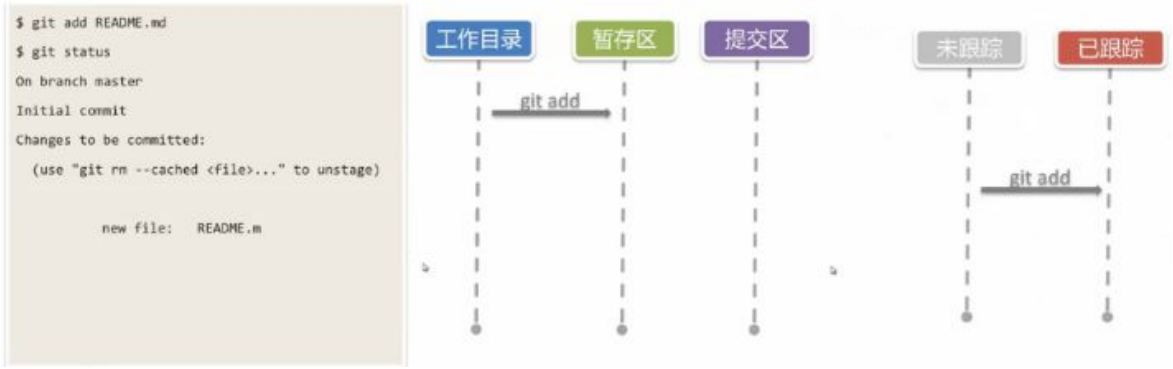
`git status` 此命令可以帮助开发者在下面三对关系中找出文件状态的变化。

- 未跟踪 <--> 跟踪
- 工作目录 <--> 暂存区
- 暂存区 <--> 最新提交

Git 中存在两种状态：**内容状态**和**状态**。 仓库中的文件均可以在状态和区域之间进行转换。



git add---添加文件内容到暂存区，同时文件被跟踪



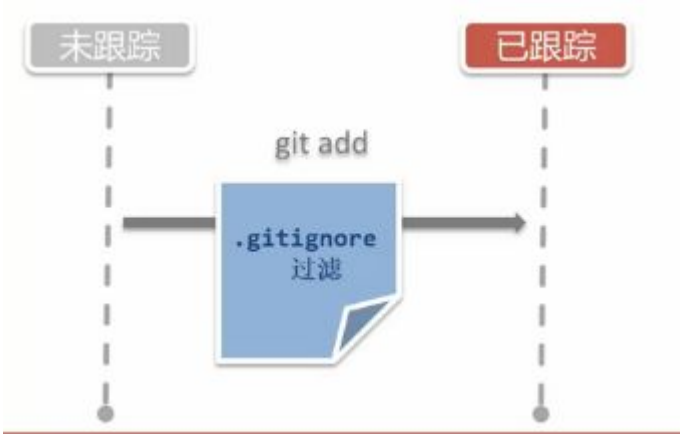
批量添加文件：**git add .**

```
$ touch package.json
$ git add .

$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
        new file:   gitl.sublime-project
        new file:   gitl.sublime-workspace
        new file:   package.json
```

忽略文件：**gitignore**

- 可以在添加至仓库时忽略匹配的文件

- 仅作用于*未跟踪*的文件。



从暂存区删除文件：**git-rm**

- git rm --cached:仅从暂存区删除文件；
- git rm: 从暂存区与工作目录中删除；
- git rm $(git ls-files --deleted):删除所有被跟踪但是在工作目录中已经被删除的文件。

工作目录与暂存区

如下图所示，README 文件同时在暂存区与工作目录都有记录。

```
$ vim README.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

          new file:   README.md
Changes not staged for commit:
  (use "git add <file>..." to update what will
be committed)
  (use "git checkout -- <file>..." to discard
changes in working directory)
          modified:   README.md
```

暂存区：（以购物车类比）

- 每类物品只能放置一次的购物车
  - 货架和购物车可以出现同种物品
  - 货架上的物品可以替换掉购物车物品
  - 可以删除物品
  - 提交购物车完成购买，生成购买记录
- 其中
  - 物品： 文件
  - 货架： 工作目录
  - 购物车: 暂存区
  - 购买： 提交内容

git commit ---提交

根据暂存区内容创建一个提交记录

```
$ git commit -m 'initial commit'
[master (root-commit) 58455f5] initial commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)

        modified:   README.md
no changes added to commit (use "git add" and/or "git commit -a")
```

直接提交：

```
$ git commit -a -m 'full commit'

[master 2679dde] full commit
 1 file changed, 1 insertion(+)

$ git status

On branch master
nothing to commit, working directory clean
```

git-log:显示提交历史记录

```
$ git log

commit 2679ddefad1cad56aa83f68a930b981b02b2458e
Author: leeluolee <87399126@163.com>
Date:   Thu Mar 5 15:40:55 2015 +0800

    full commit

commit 1374c78087c0fe508e2a15b9bd556b56d3961f59
Author: leeluolee <87399126@163.com>
Date:   Thu Mar 5 15:39:22 2015 +0800

    first commit
```

包括：

- 提交编号 SHA-1 编码的 HASH 标示符
- git-config 配置的提交者信息
- 提交日期
- 提交描述信息

git 中的 alias 命令（别名设置）

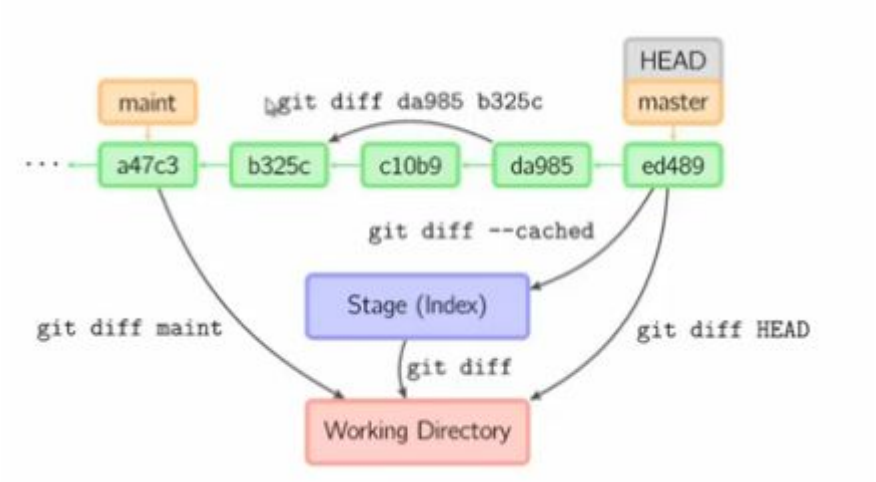
git config alias.shortname <fullcommand>

```
$ git config --global alias.lg "log --color --graph
--pretty=format:'%Cred%h%Creset
-%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset'
--abbrev-commit"
```

```
$ git lg
```

git diff---显示不同版本差异

- git diff: 工作目录与暂存区的差异；
- git diff -cached[<reference>]: 暂存区与某次提交差异，默认为 HEAD.
- git diff <reference>: 工作目录与某次提交的差异



撤销本地修改: git checkout -- <file>

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in
working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -
a")
```

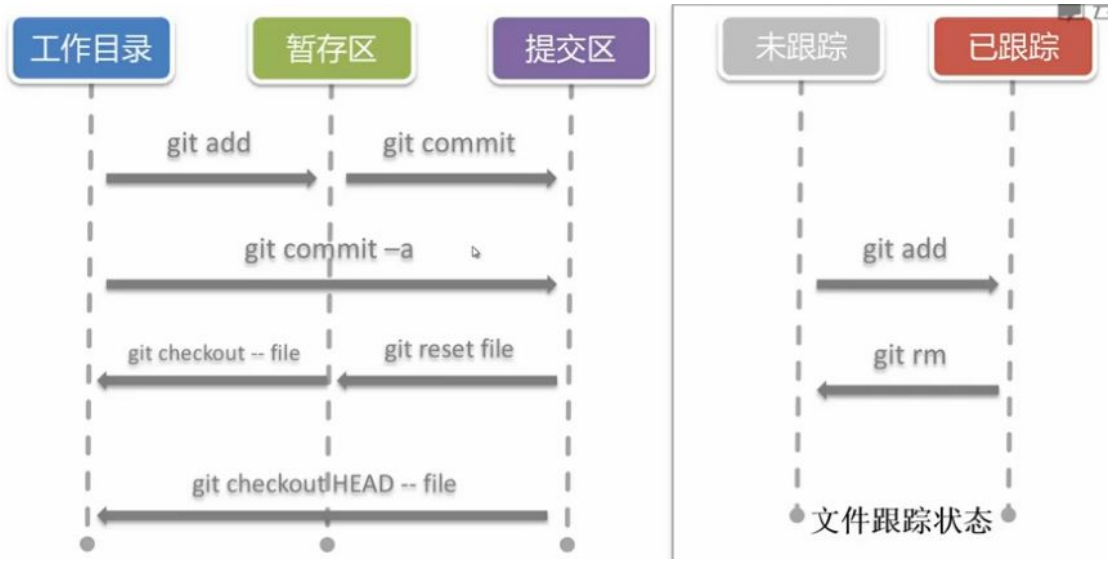

```
$ git checkout -- README.md

$ git status

On branch master
nothing to commit, working directory clean
```

撤销暂存区内容：git reset HEAD <file>

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
$ git reset HEAD README.md
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)

        modified:   README.md
no changes added to commit (use "git add" and/or "git commit -a")
```

撤销全部改动：

工作目录、暂存区、提交区关系：



分支操作

git branch: 分支的增删查改都靠它

- `git branch <branchname>`，创建指定分支
- `git branch -d <branchname>`，删除指定分支
- `git branch -v`，显示所有分支信息

git 分支轻量级的秘密：

一份分支的引用只是一个文本文件，里面只有一个 SHA 编码。

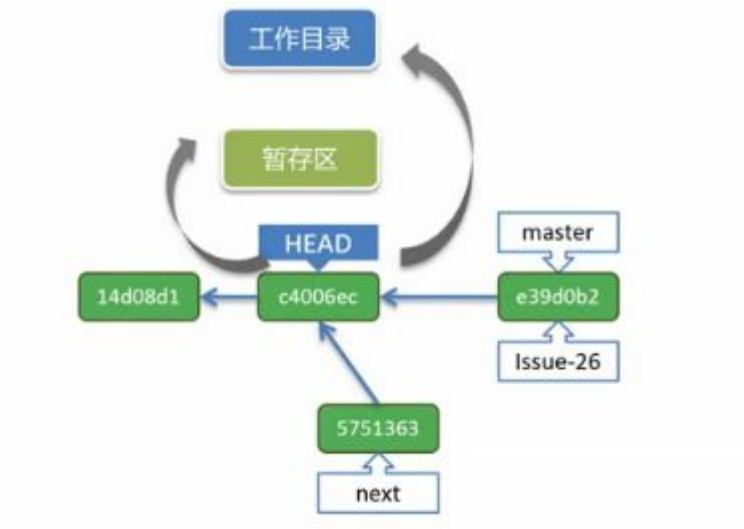
它保存于`.get/refs/heads/master`中。

git checkout:分支切换

通过移动 HEAD 检出版本，可用于分支切换

- `git checkout <branchname>`，使指针指向目标分支
- `git checkout -b <branchname>`，创建目标分支并切换分支
- `git checkout <reference>`，可以指向任何一个版本

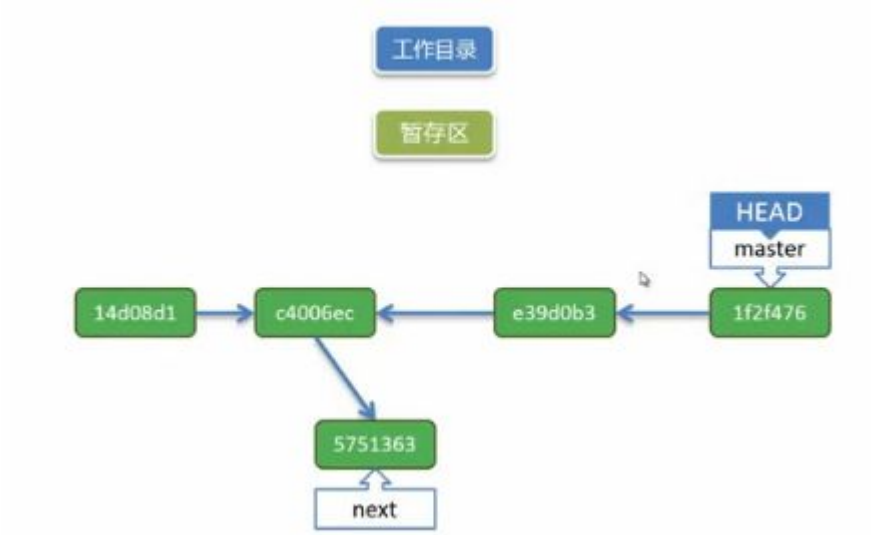
当 HEAD 指针与具体的分支分离时，我们将其称之为`detached head`。 如果 HEAD 在分离状态则因尽量避免在此状态下进行提交，只做内容的查看。



git reset:分支回退

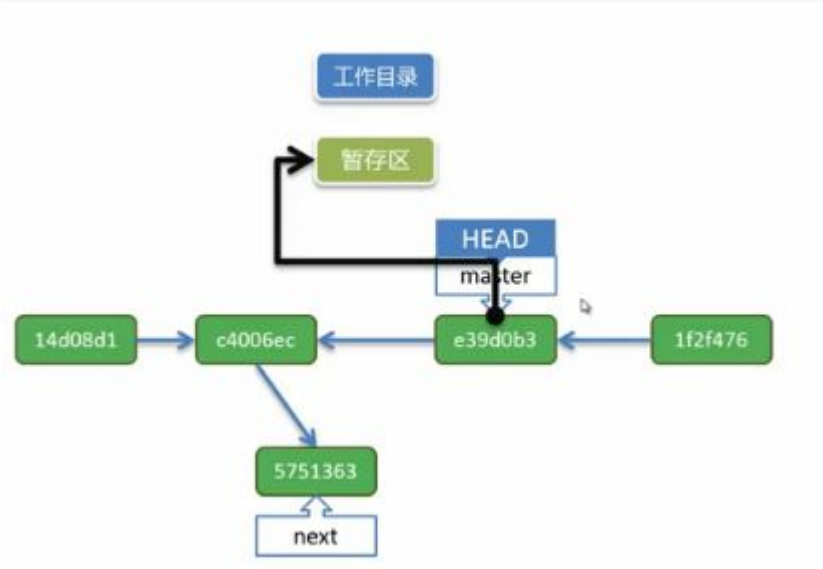
将当前分支回退到历史某个版本，有以下三种方式。
三种方式的区别：内容是否会恢复到工作目录或暂存区。

- `git reset --mixed <commit>` 默认方式，内容存入暂存区
- `git reset --hard <commit>` 内容存入暂存区和工作区
- `git reset --soft <commit>` 暂存区和工作区保留现有状态

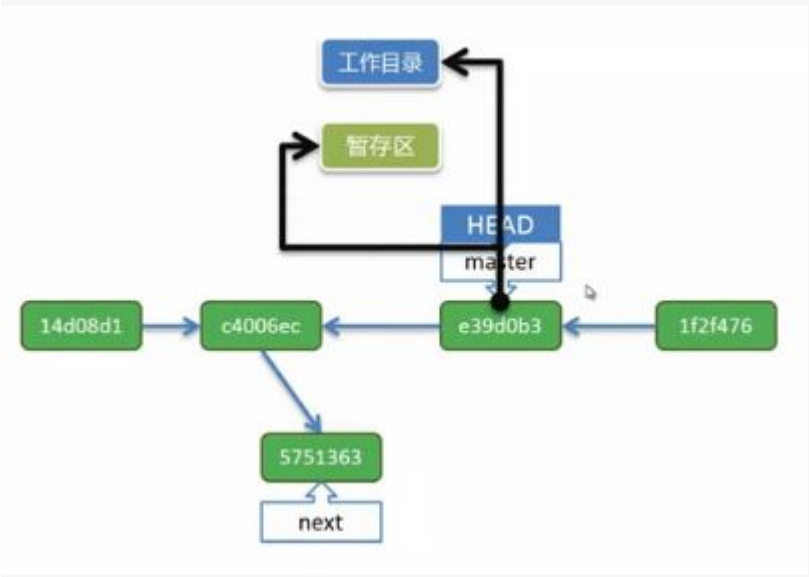


将 master 分支回退到上一个版本：

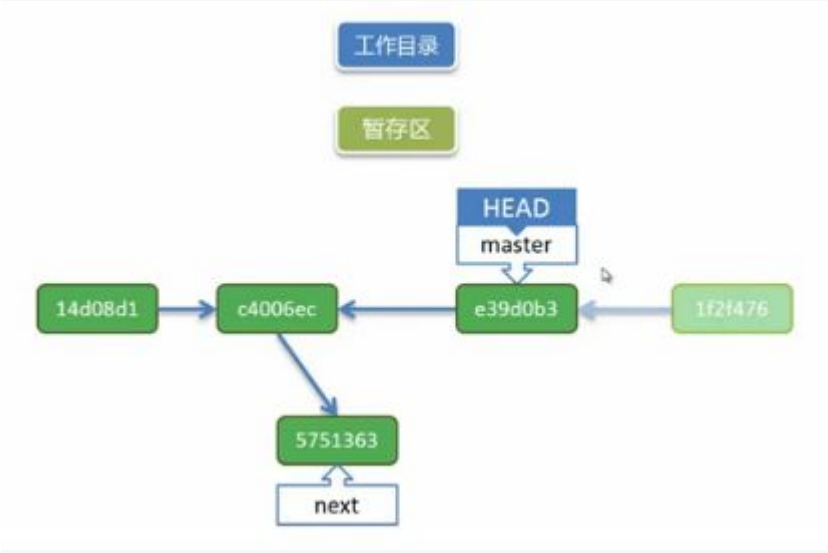
`git reset --mixed e390b3`，将当前内容复制到暂存区，如下图：



`git reset --hard e390b3`，内容会复制到工作目录，如下图：



`get reset --soft e390b3`，使暂存区和工作目录保持现在的状态，如下图：



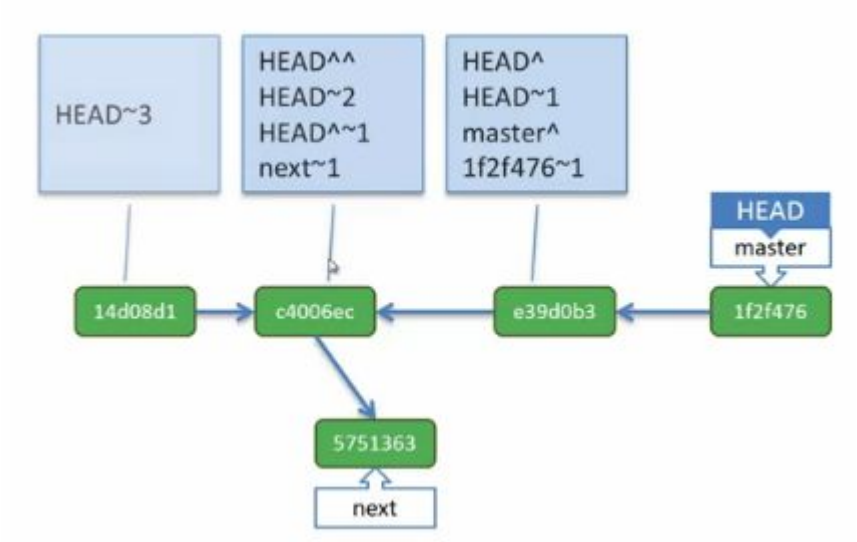
git reflog---查询提交记录

`git reflog` 会根据仓库的提交顺序按顺序来排列，其中包括无索引的提交， 可以在这里使用 HASH 值来进行，但是无索引的提交可能会丢失。

使用捷径

`A^` 表示 `A` 上的父提交，多个 `^` 可表示以上的多个级别。

`A~n` 则表示在 `A` 之前的第 n 次提交。



reset 与 checkout 区别：

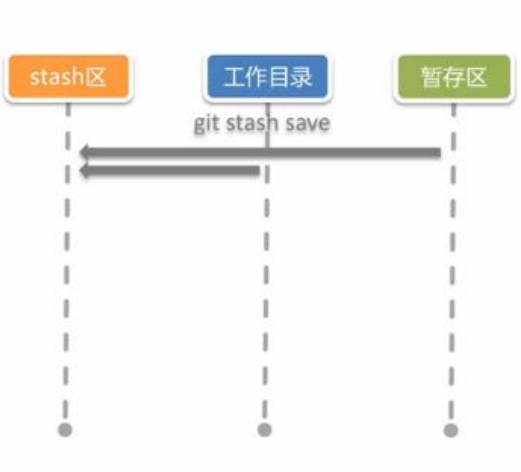
两种方法都有两个作用范围，一个是分支操作（commit 操作），另一个是文件操作（file 操作）。

命令	范例	移动(HEAD/branch)	说明
git reset【commit】	git reset HEAD^ --soft	是/是	完全回退到某提交
git reset【file】	git reset README.md	否/否	恢复暂存区到某提交状态
git checkout【commit】	git checkout master	是/否	移动当前指针HEAD到某提交
git checkout 【file】	git checkout -- README.md git checkout HEAD -- xx.log	否/否	恢复工作目录到某状态

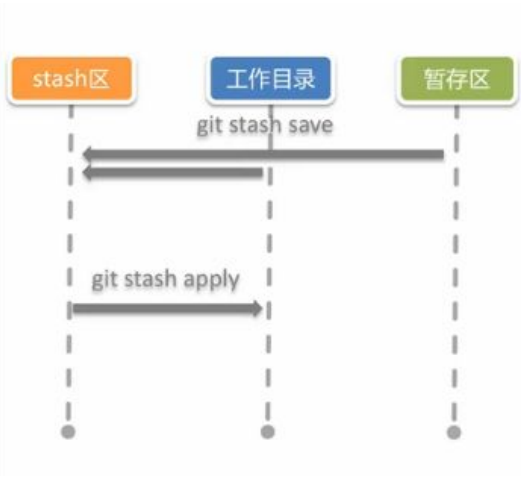
git stash:保存目前的工作目录和暂存区状态，并返回到干净的工作空间。

在操作时，如果突然需要切换到其他分支，工作区和暂存区还有在当前分支没完成的任务。那么 `stash` 就使用 `.git` 中的特殊区（Stash 区）来帮你解决这个问题（因为强切会丢失当前的工作区和暂存区的内容）。

```
$ git stash save 'push to stash area'
Saved working directory and index state On master: push to
stash area
HEAD is now at b370926 hello

$ git status
On branch master
nothing to commit, working directory clean

$ git stash list
stash@{0}: On master: push to stash area
```




```
$ git stash apply stash@{0}
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in
working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -
a")

$ git stash drop stash@{0}
Dropped refs/stash@{0} (53331aa0401b1d321c47bb2efebdd1581e2077ac)
```

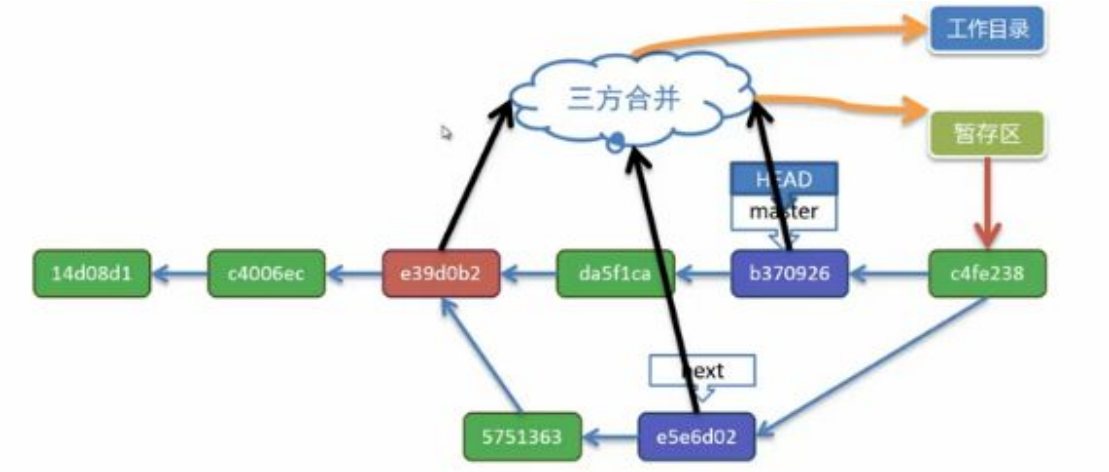


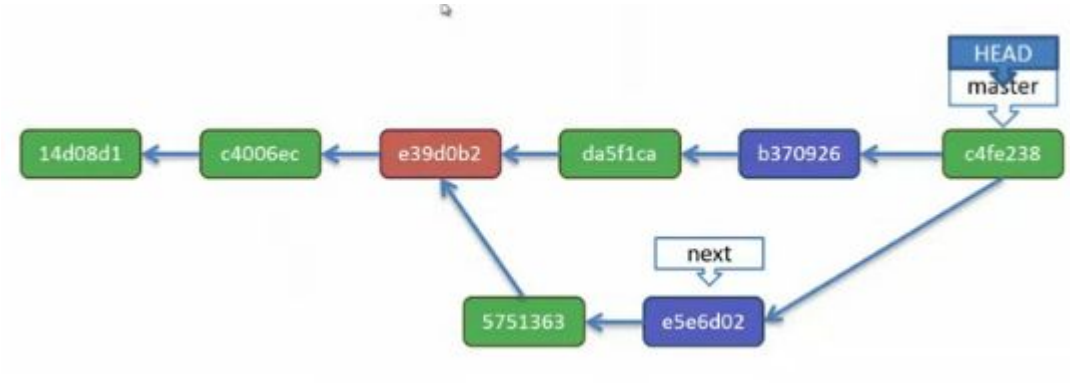
`stash` 可以把当前工作区和暂存区的状态以栈（Stack）的形式保存起来（每次保存都会推一个内容到 `stash` 栈中），并返回一个干净的工作空间（工作区和暂存区）。

```
stash pop = stash apply + stash drop
```

git merge---合并分支

下面的例子是将 next 分支合并到 master 分支上





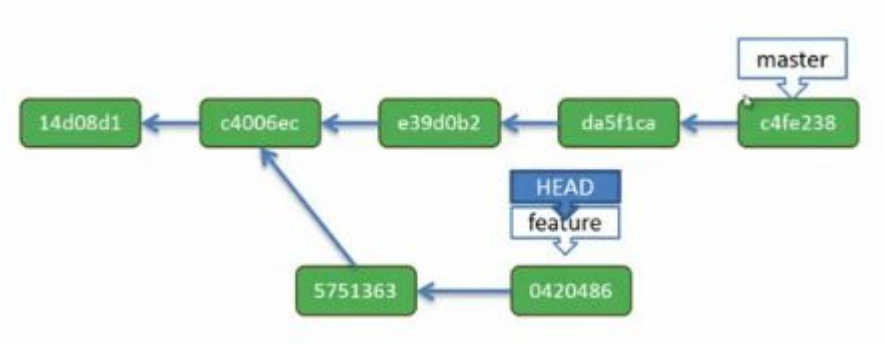
解决 merge 冲突

当一个文件被同时修改时（更多情况为同时修改相同的一行代码时）则极有可能产生合并冲突。

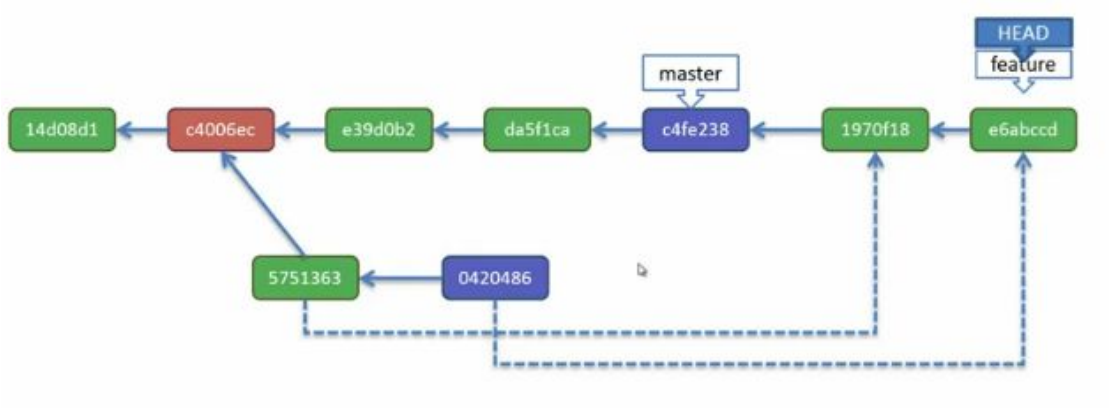
merge fast-forward:

merge 的不足

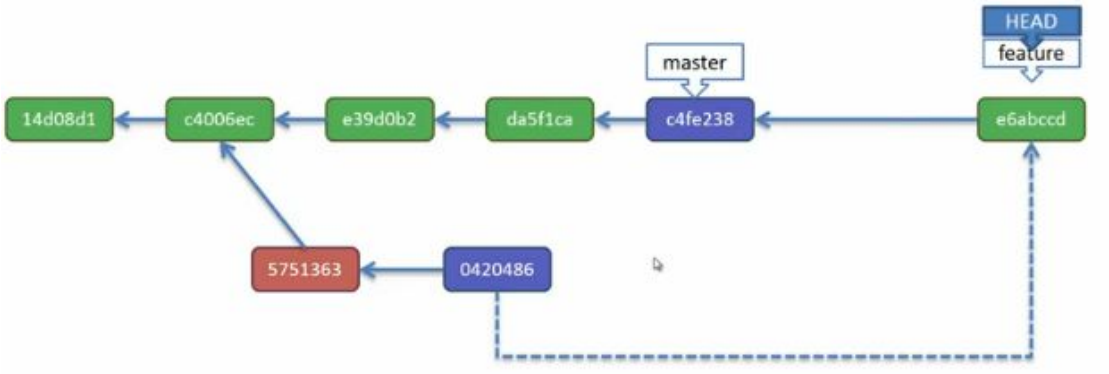
git rebase:修剪提交历史基线，俗称“变基”



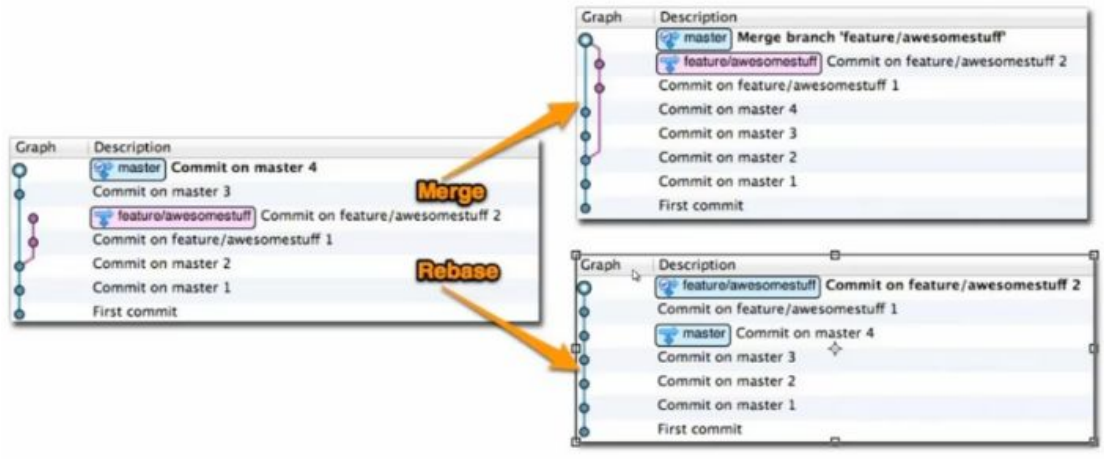
上述状态，使用命令：git rebase master，得到下图状态：



命令：git rebase --onto master 5751363，挑选需要重演的节点到 master 分支上，



rebase VS merge:

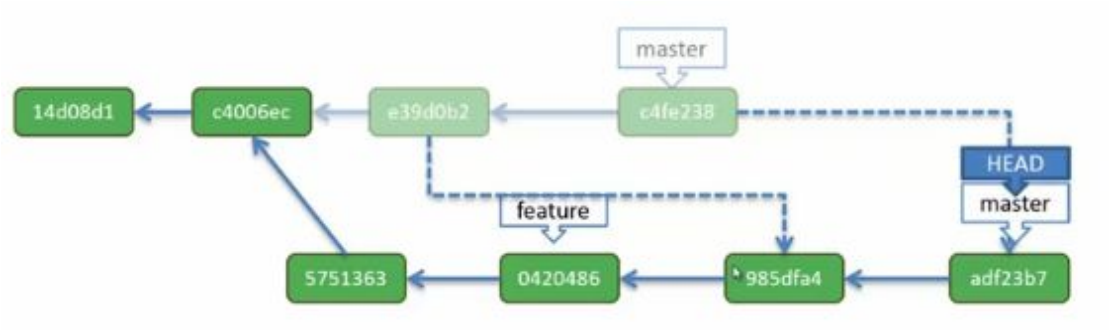


`rebase` 会产生线性的提交历史，`merge` 则会产生多个不同分支的合并节点。所以具体没有好坏之分，可根据使用的需求来决定。

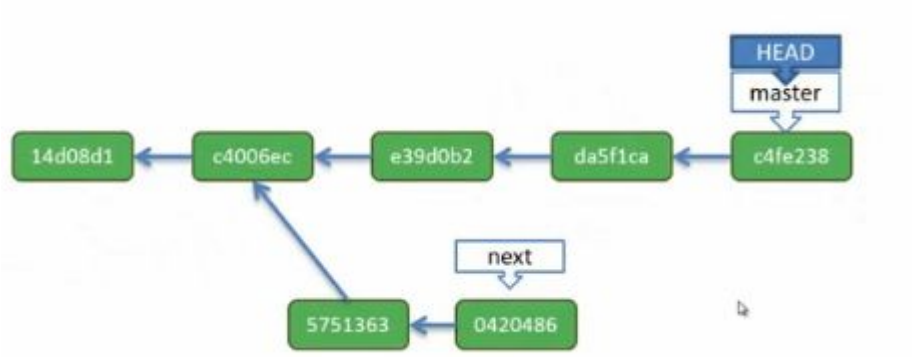
勿在共有分支上(如 master 分支)使用 rebase

因为，这会导致其他开发者在进行拉取（Pull）时，必须进行合并且合并中包

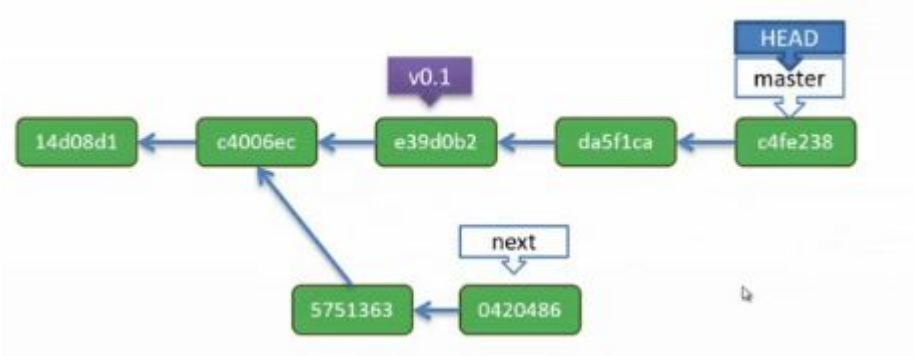
含重复的提交。



git tag:对某个提交设置一个不变的别名



对上述状态使用命令：git tag v0.1 e39d0b2,如下图：



这样做的作用，直接使用标签名来进行 checkout 等操作，而不用输入哈希值，如：

git checkout v0.1

远程操作

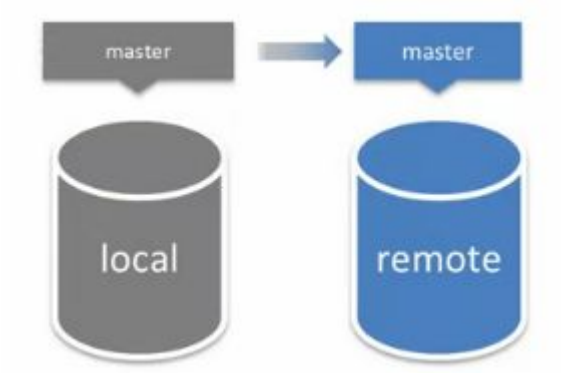
远程操作可以将本地仓库推送至远程仓库服务器。**Git** 支持许多主流的通信协议，其中包括 `Local`、`HTTP`、`SSH`、还有 `Git`。

Git 支持本地协议，所有我们可以初始化一个本地的远程服务器……

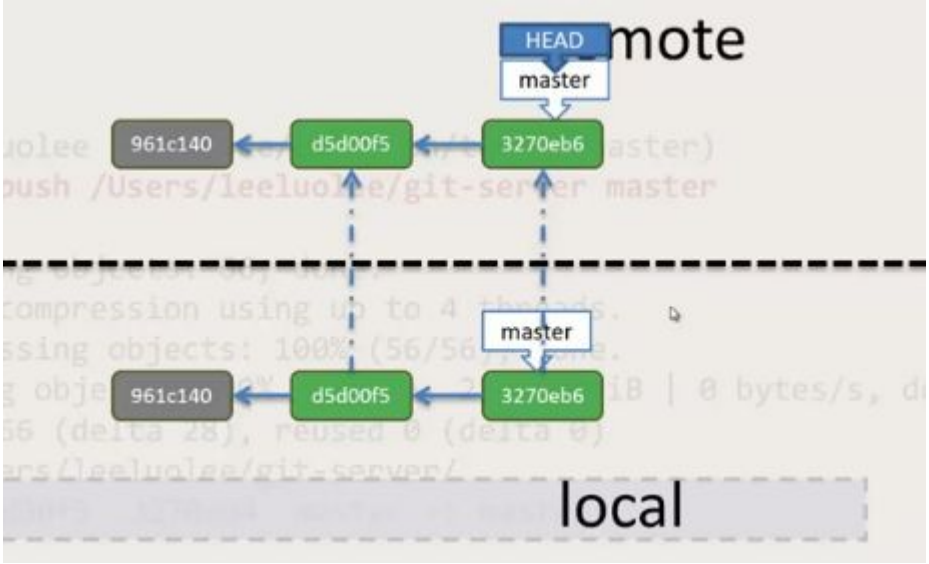
初始化一个本地的远程服务器

```
git init ~/git-server --bare: 将当前仓库初始化为一个裸仓库（没有工作目录）
```

git push:提交本地历史到远程仓库




```
# leeluolee in ~/code/tmp/gith/test1 (master)
$ git push /Users/leeluolee/git-server master

Counting objects: 66, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (56/56), done.
Writing objects: 100% (66/66), 210.19 KiB | 0 bytes/s, done.
Total 66 (delta 28), reused 0 (delta 0)
To /Users/leeluolee/git-server/
    d5d00f5..3270eb4  master -> master
```

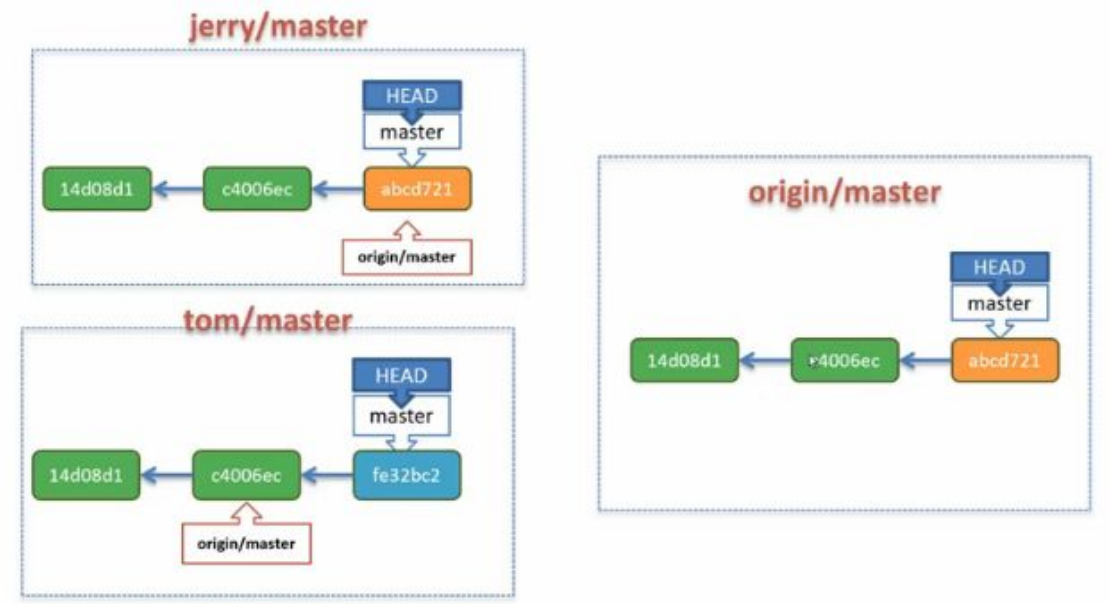


git remote:远程仓库的相关配置

配置远程映射

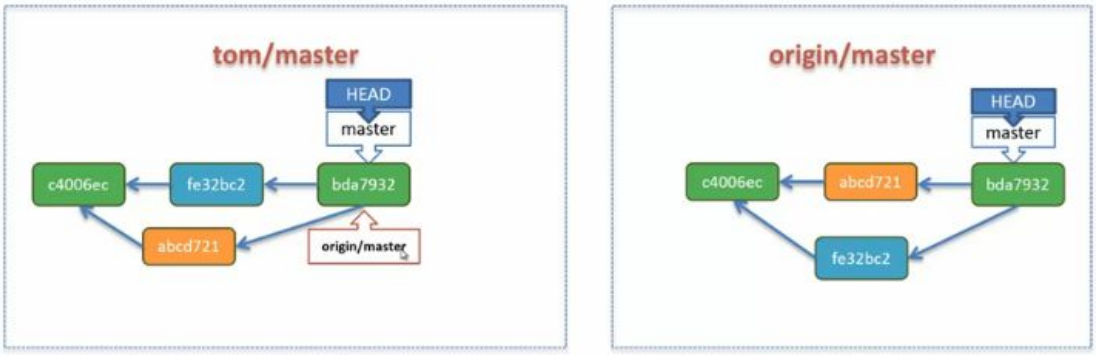
```
$ git remote add origin ~/git-server

$ git remote -v
origin      /Users/leeluolee/git-server (fetch)
origin      /Users/leeluolee/git-server (push)
```

push 冲突

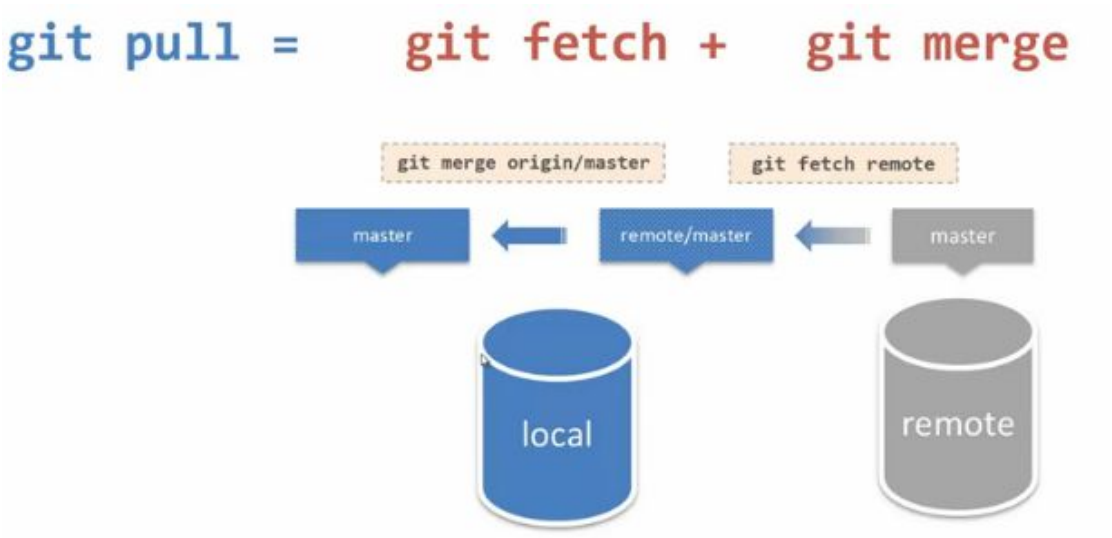


git fetch:获取远程仓库的提交历史。

可以使用 `git fetch` + `git merge` 来解决上面的冲突的问题。



`git pull` 就等同于 `fetch` 与 `merge` 的合并。



完整获取远程仓库：

- git init + git remote + git pull
- **git clone:** 克隆远程仓库,并将克隆地址默认为 **origin**

git clone: 克隆一个远程仓库作为本地仓库，它会获得所有远程仓库的历史。

```
# leeluolee in ~/code/tmp/gitl
$ git clone ~/git-server test2
Cloning into 'test2'...
done.

# leeluolee in ~/code/tmp/gitl/test2 (master)
$ git remote -v
origin      /Users/leeluolee/git-server (fetch)
origin      /Users/leeluolee/git-server (push)
```

参考资料：

1.git 简明指南：☆

http://rogerdudler.github.io/git-guide/index.zh.html

2.try.github.com 教程：

https://try.github.io/levels/1/challenges/1

3.《Pro Git》，Git 百科全书指南：

http://git-scm.com/book/zh/v1

4.git 教程：☆☆☆☆☆

http://www.liaoxuefeng.com/wiki/001373951630592960 6dd18361248578c67b8067c8c017b000

5.Git Community Book 中文版：

http://gitbook.liuhui998.com/

6.Git 入门：

http://backlogtool.com/git-guide/cn/intro/intro1_1.html

7.Git 图解：☆☆☆☆☆

http://marklodato.github.io/visual-git-guide/index-zh-cn.html