Git 使用

罗流毅

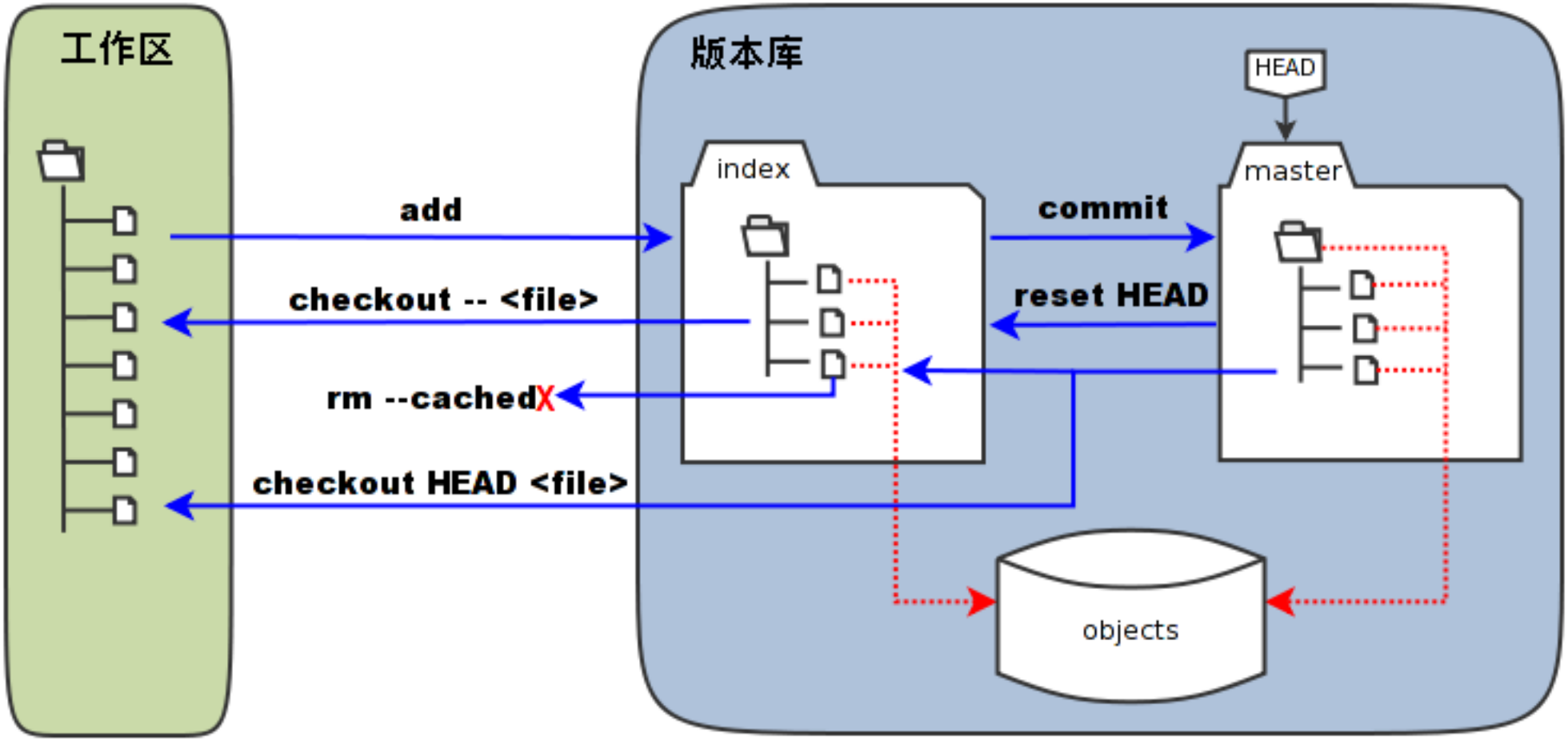
xluoly@gmail.com

2018-7-26

Agenda

本地仓库的组成

1. 工作目录: 它持有实际文件
2. 暂存区 (Index): 它像个缓存区域，临时保存你的改动
3. HEAD: 它指向你最后一次提交的结果



Agenda

安装

- git Linux 版
  - https://book.git-scm.com/download/linux
- git Windows 版
  - https://book.git-scm.com/download/win
- git OSX 版
  - https://book.git-scm.com/download/mac

个人配置

- 配置用户 ID 和 Email

```
$ git config --global user.name "liuyi.luo"
$ git config --global user.email "xluoly@gmail.com"
```

- 配置编辑器、比较工具以及合并工具

```
$ git config --global core.editor vim
$ git config --global diff.tool vimdiff
$ git config --global merge.tool vimdiff
```

- 编辑 config

```
$ git config --globel -e
```

- 直接打开 ~/.gitconfig 文件进行编辑

创建新仓库

- 创建新的 git 仓库:

```
$ mkdir <projectname>
$ cd <projectname>
$ git init
```

克隆仓库

- 克隆本地仓库:

```
$ git clone /path/to/repository
$ git clone /path/to/repository <project-name>
```

- 克隆远端服务器上的仓库:

```
$ git clone username@host:/path/to/repository
$ git clone username@host:/path/to/repository <project-name>
```

添加文件

- 添加指定文件

```
$ git add foo.c
```

- 添加当期目录下所有文件到暂存区

```
$ git add *
```

提交修改

- 将暂存区的改动提交到了 HEAD，但是还没有推送到远端仓库。

```
$ git commit -m "代码提交信息"
```

推送改动

- 改动已经在本地仓库的 HEAD 中，执行如下命令将这些改动提交到远端仓库，可以把 master 换成你想要推送的任何分支。

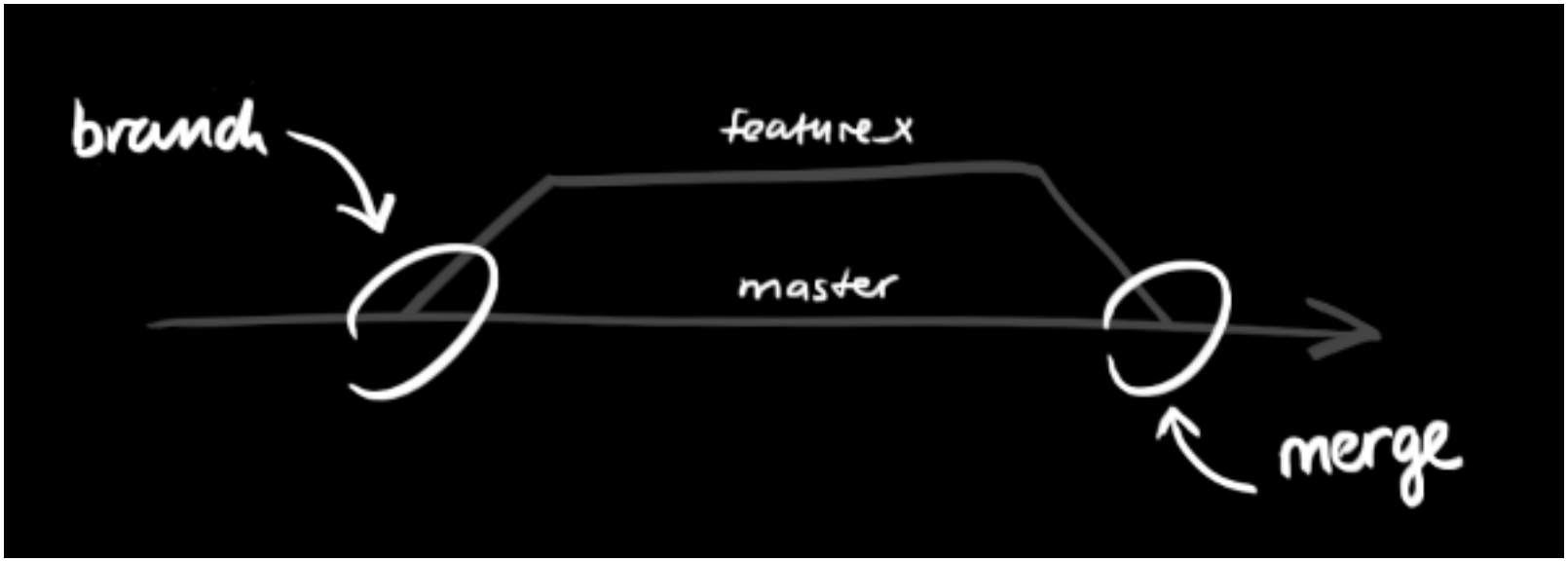
```
$ git push origin master
```

- 如果还没有克隆现有仓库，可以使用如下命令添加，这样才能够将你的改动推送到远端的服务器上去了。

```
$ git remote add origin git@server:testing.git
```

分支

- 在创建仓库的时候，`master` 是默认的分支。在其他分支上进行开发，完成后再将它们合并到 `master` 分支上。



特性分支

分支可以用来将特性开发隔离开来。

- 创建一个叫做 `feature_x` 的分支，并切换过去：

```
$ git checkout -b feature_x
```

- 切换回主分支：

```
$ git checkout master
```

- 删除新建的分支：

```
$ git branch -d feature_x
```

- 如果分支不推送到远程仓库，它是不为他人所见的（即私有的）：

```
$ git push origin <branch>
```

更新与合并

- 更新本地仓库至最新状态，执行：

```
$ git fetch
```

- 合并其他分支到当前分支（例如 master），执行：

```
$ git checkout master
$ git merge <branch>
```

- 同步远程 master 分支到当前本地分支：

```
$ git pull origin master
```

其实这个命令同时执行了 fetch 和 merge 操作。

解决冲突

- 在更新合并的过程中，git 会尝试去自动合并改动。
- 有时可能出现冲突 (conflicts), 这时就需要手动修改这些文件来合并这些冲突。
- 改完冲突之后，还需要执行如下命令将它们标记为已完成合并的状态。

```
$ git add <filename>
```

标签

- 通常推荐为软件发布创建标签。可以执行如下命令创建一个叫做 `1.0.0` 的标签：

```
$ git tag 1.0.0 1b2e1d63ff
```

`1b2e1d63ff` 是想要标记的提交 ID 的前 10 位字符。也可以使用少一点的提交 ID 位数，只要它的指向具有唯一性就可以。

- 可以使用下列命令获取提交 ID：

```
$ git log
```

撤销本地改动

- 如果操作失误，可以使用如下命令撤销掉本地改动：

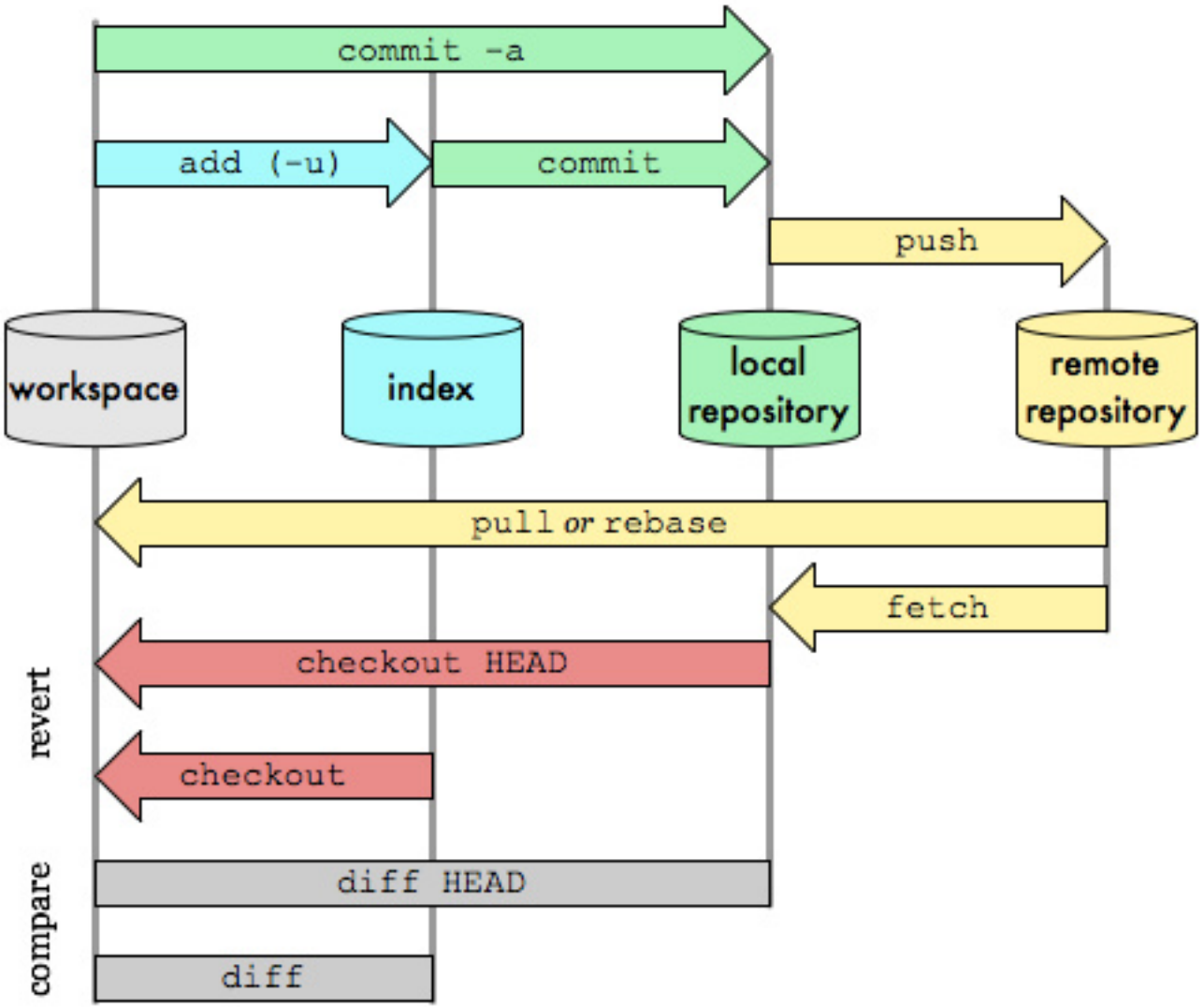
```
$ git checkout <filename>
```

此命令会使用 HEAD 中的最新内容替换掉工作目录中的文件。但是已添加到暂存区的改动以及新文件都不会受到影响。

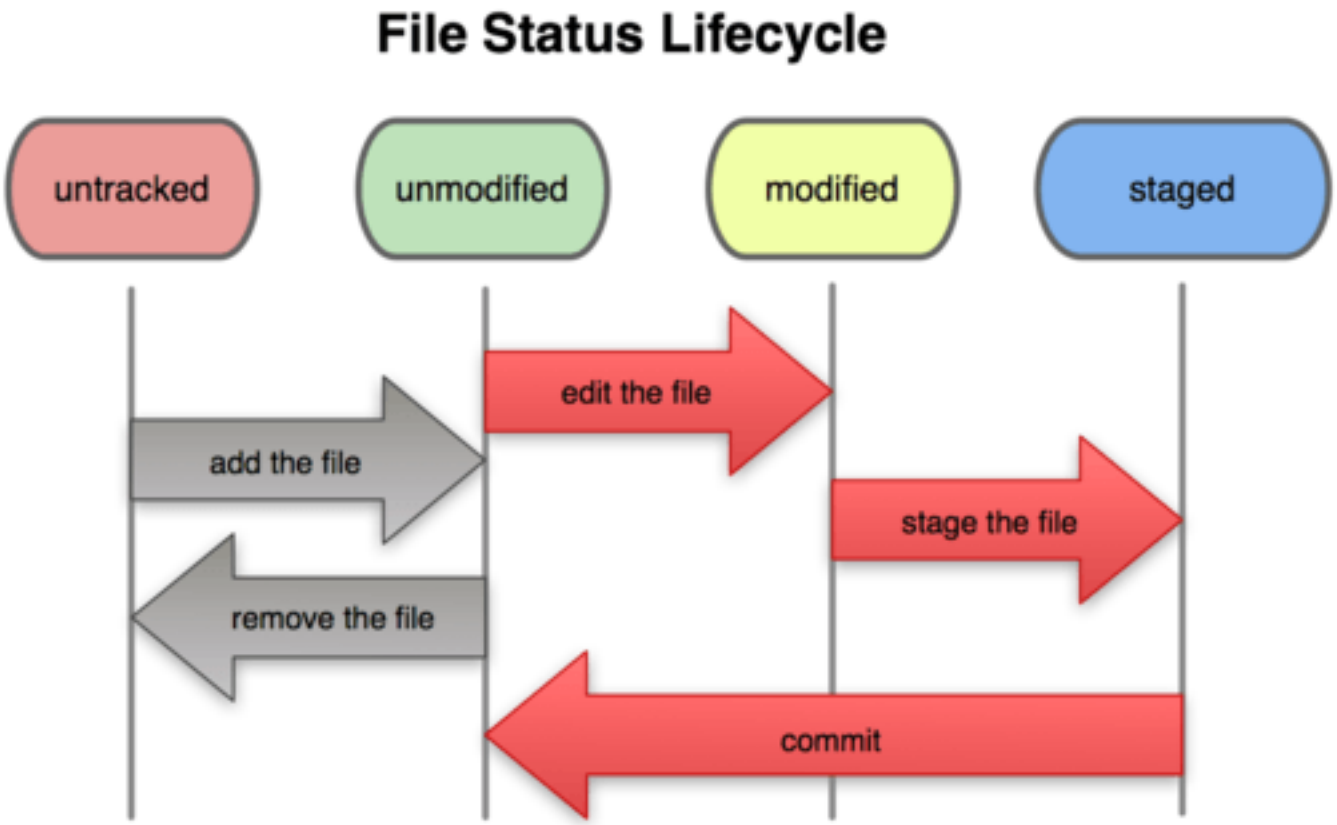
- 假如想丢弃本地的所有改动与未推送的提交，可以到服务器上获取最新的版本历史，并将本地主分支指向它：

```
$ git fetch origin
$ git reset --hard origin/master
```

命令小结


commit -a
add (-u)
commit
push
workspace
index
local repository
remote repository
pull or rebase
fetch
checkout HEAD
revert
checkout
compare
diff HEAD
diff

文件状态变化


File Status Lifecycle
untracked
unmodified
modified
staged
add the file
edit the file
stage the file
remove the file
commit

Agenda

Git 工作流

1. 去自己的工作分支

```
$ git checkout work
```

工作 ……

2. 提交工作分支的修改

```
$ git add <modified files>
$ git commit
```

3. 回到主分支

```
$ git checkout master
```

4. 获取远程最新的修改，此时不会产生冲突

```
$ git pull
```

5. 回到工作分支

```
$ git checkout work
```

Git 工作流（续）

6. 用 rebase 合并主干的修改，如果有冲突在此时解决

```
$ git rebase master
```

7. 回到主分支

```
$ git checkout master
```

8. 合并工作分支的修改，此时不会产生冲突。

```
$ git merge work
```

或者

```
$ git merge --no-ff work
```

9. 提交到远程主干

```
$ git push
```

小结

这个工作流的好处：

- 远程主干上的历史永远是线性的。
- 每个人在本地分支解决冲突，不会在主干上产生冲突。

Agenda

合并单个 Commit

- 合并其他分支的某个 Commit 到当前分支

```
$ git cherry-pick <commit-id>
```

修改已经提交的 Commit

- 已经提交到本地仓库，只有未 push 到远端仓库的 Commit 才允许被修改
- 补充上一次提交

```
$ git commit --fixup=HEAD
$ git rebase -i HEAD^^ --autosquash
```

- 修改最近一次 Commit

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

- 修改多个 Commit，运行下面命令，在弹出的列表中将需要修改的 commit 标记为 `edit`，保存并退出编辑器，然后按提示完成后面的操作

```
$ git rebase -i <commit-id>^
```

修改已经提交的 Commit

- 整理 commit 的顺序，运行 rebase 命令，在弹出的列表中按自己的要求调整 commit 顺序，然后保存并退出编辑器

```
$ git rebase -i <commit-id>^
```

- 压制 Commit（将多个 Commit 压制成一个），运行 rebase 命令，编辑列表中想要压制的 Commit 为 squash，保存并退出编辑器，标记未 squash 的 commit 将会被归并到上一个 commit，同时允许你重新编辑 log

文件标注

- 如果发现某段代码存在问题，想知道是什么时候引入的、或者想知道责任人是谁以便进一步了解

```
$ git blame build.gradle
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800  2) buildscript {
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800  3)     repositories {
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800  4)         jcenter()
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800  5)     }
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800  6)     dependencies {
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800  7)       classpath 'com.android.too
2.2.3'
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800  8)     }
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800  9) }
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800 10)
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800 11) allprojects {
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800 12)     repositories {
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800 13)         jcenter()
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800 14)     }
5b93a45c (liuyi.luo 2017-03-28 09:38:44 +0800 15) }
```

Patch

- 有时候需要通过 patch 来与别人交流代码
- 为两个 commit 之间的修改生成 patch

```
$ git format-patch <commit-id1> <commit-id2>
```

- 为单个 commit 生成 patch

```
$ git format-patch -1 <commit-id>
```

- 为某个 commit 以及之后的所有修改生成 patch

```
$ git format-patch <commit-id>
```

- 应用 patch

```
$ git am 0001-xxx.patch
```

二分查找

- 某个版本出现了问题，而更早的某个版本没有该问题
- 能够重现问题，但是找不到问题在哪里
- 两个版本之间有大量的 Commit，需要定位到是哪一次 Commit 引入了问题

```
$ git bisect start          #启动二分查找
$ git bisect bad            #当前版本是有问题的
$ git bisect good v1.0      #v1.0这个版本是好的
```

- Git 会检出一个中间版本，你可以测试这个版本，然后用 `git bisect bad` 来告诉它这个版本是有问题的，或者用 `git bisect good` 告诉它这个版本是好的，如此反复，直到定位到引入问题的某个 Commit
- 最后运行如下命令来结束二分查找，将 HEAD 移到开始的位置

```
$ git bisect reset
```

Git 与 SVN 的结合

- 项目是用 SVN 管理的
- 想使用 Git 提供的各种便利，例如：便捷的分支策略、重改提交历史、离线提交等
- clone subversion 仓库

```
$ git svn clone https://example.com/svn/project -s
```

- 然后本地仓库的各种操作就像普通的 Git 仓库那样使用了，只是从 SVN 仓库获取更新以及提交到 SVN 仓库有所不同

Git 与 SVN 的结合

- 从 SVN 仓库获取更新

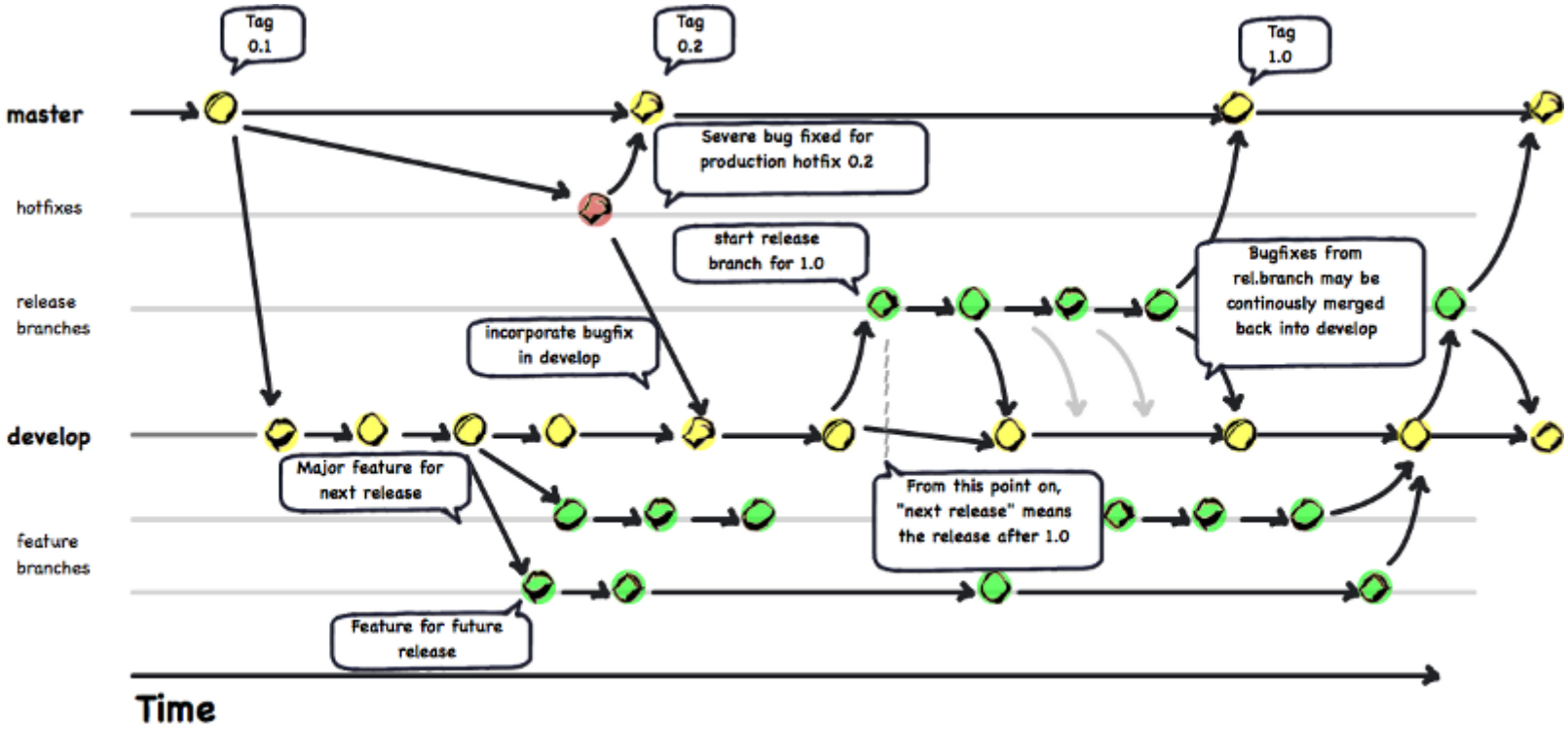
```
$ git checkout master
$ git svn fetch
$ git svn rebase
```

Git 与 SVN 的结合

- 提交到 SVN 仓库

```
$ git checkout master
$ git svn fetch
$ git svn rebase
$ git checkout feature_xxx
$ git rebase master
$ git checkout master
$ git merge feature_xxx
$ git svn dcommit
```

推荐的分支模型


Tag 0.1
Tag 0.2
Tag 1.0
master
hotfixes
release branches
develop
feature branches
Severe bug fixed for production hotfix 0.2
start release branch for 1.0
incorporate bugfix in develop
Bugfixes from rel.branch may be continously merged back into develop
Major feature for next release
From this point on, "next release" means the release after 1.0
Feature for future release
Time

Git 使用者的基本修养

- 将提交做小
  - 一个提交只需完成一个任务
  - 不要将多个任务放在一起提交
- 将提交做对
  - 提交日志没写错
  - 没有遗漏文件
  - 不会造成项目编译失败
  - push 之前仔细检查，发现问题还可以用 `git commit --amend` 和 `git commit --fixup` 等命令进行补救
- 将提交做好
  - 多个提交的顺序是否合理
  - 有没有需要 squash 的提交
  - 认真详细的填写提交日志

链接与资源

图形化客户端

- Gitg (Linux, 开源软件)
- TortoiseGit (Windows)
- Source Tree (OSX, 免费)

指南和手册

- Git 社区参考书
- Git 专家指南
- GitHub 帮助
- 图解 Git

The End