GIT

大纲

Git简介

Git基础

Git操作

Git分支管理最佳实践git-flow

Git建议使用规范

GIT简史

2002 年，linux项目组开始启用分布式版本控制系统 BitKeeper 来管理和维护代码。

2005 年的时候，开发 BitKeeper 的商业公司同 Linux 内核开源社区的合作关系结束，他们收回了免费使用BitKeeper 的权力

April 5, 2005 – Linus发布首个git版本

June 15, 2005 - Git 用作Linux源码版本控制

Git简介

它的特点在于：

- 1.开源
- 2.高速
- 3.节省大量空间
- 4.灵活、简洁、高效的分支管理

它能最大限度地发挥多人协同并发编程的效能，让分支管理更快速，版本管理更简单

高速

Git比其他的VCS工具要快很多，因为git绝大部分是离线操作，对网络依赖小

- time git clone ssh://gufeiyong@scm.hz.netease.com:2222/backend/datastream.git >/dev/null
  real0m26.559s
  user0m2.568s
  sys0m1.028s

- time svn co https://svn.hz.netease.com/svn/datastream >/dev/null
  real3m14.684s
  user1m1.156s
  sys0m20.897s

节省空间

git比较节省空间。Django项目为例。

44M ./django-git

53M ./django-svn

git克隆比SVN要小很多，且git克隆包含整个项目的历史版本。

SVN只包含项目的最后一个版本。

分支

以前的VCS工具分枝的方法是对每一个分枝都放到一个独立的目录中。而git可以让你在同一个工作目录中切换（switch）到不同的分枝。创建和切换分枝几乎是即时的（instant），并且存在本地分支

Git开发者可以随时创建，合并，删除多个分枝。它鼓励一种非线性的开发周期，它可以说是并行的多线程模式而不是多个步骤串行的模式。

大纲

Git简介

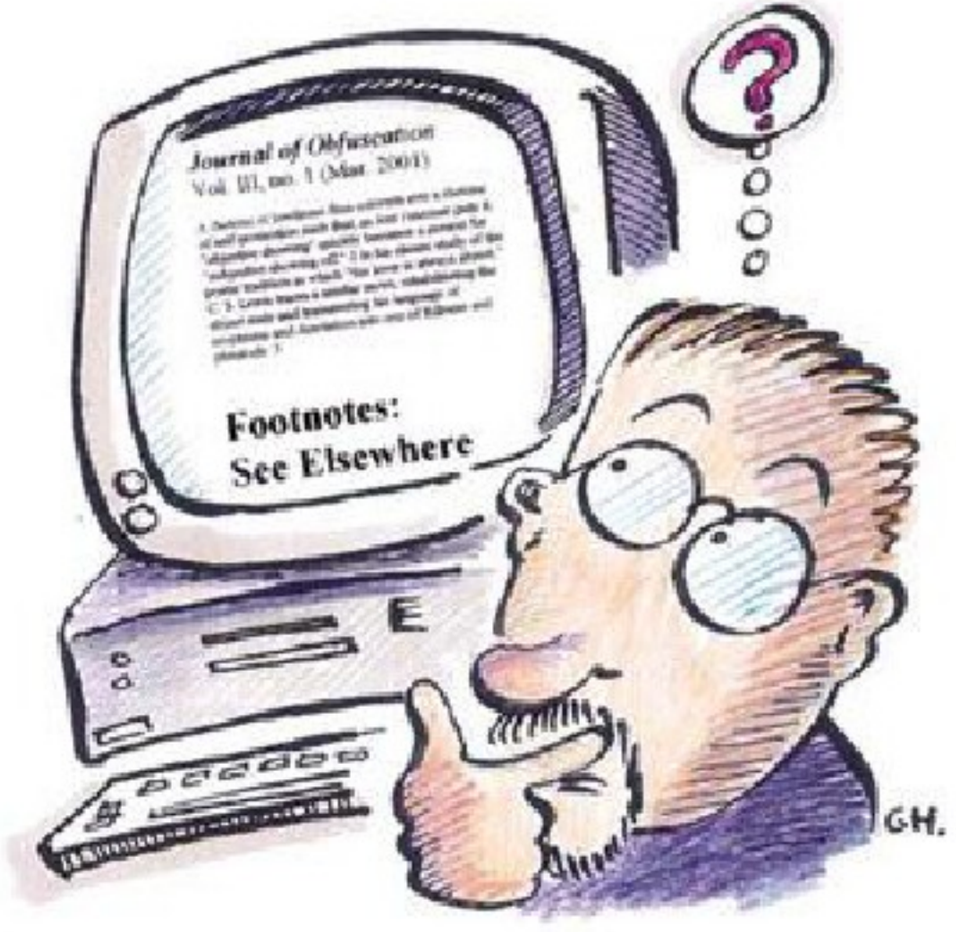
Git基础

Git操作

Git分支管理最佳实践git-flow

Git建议使用规范

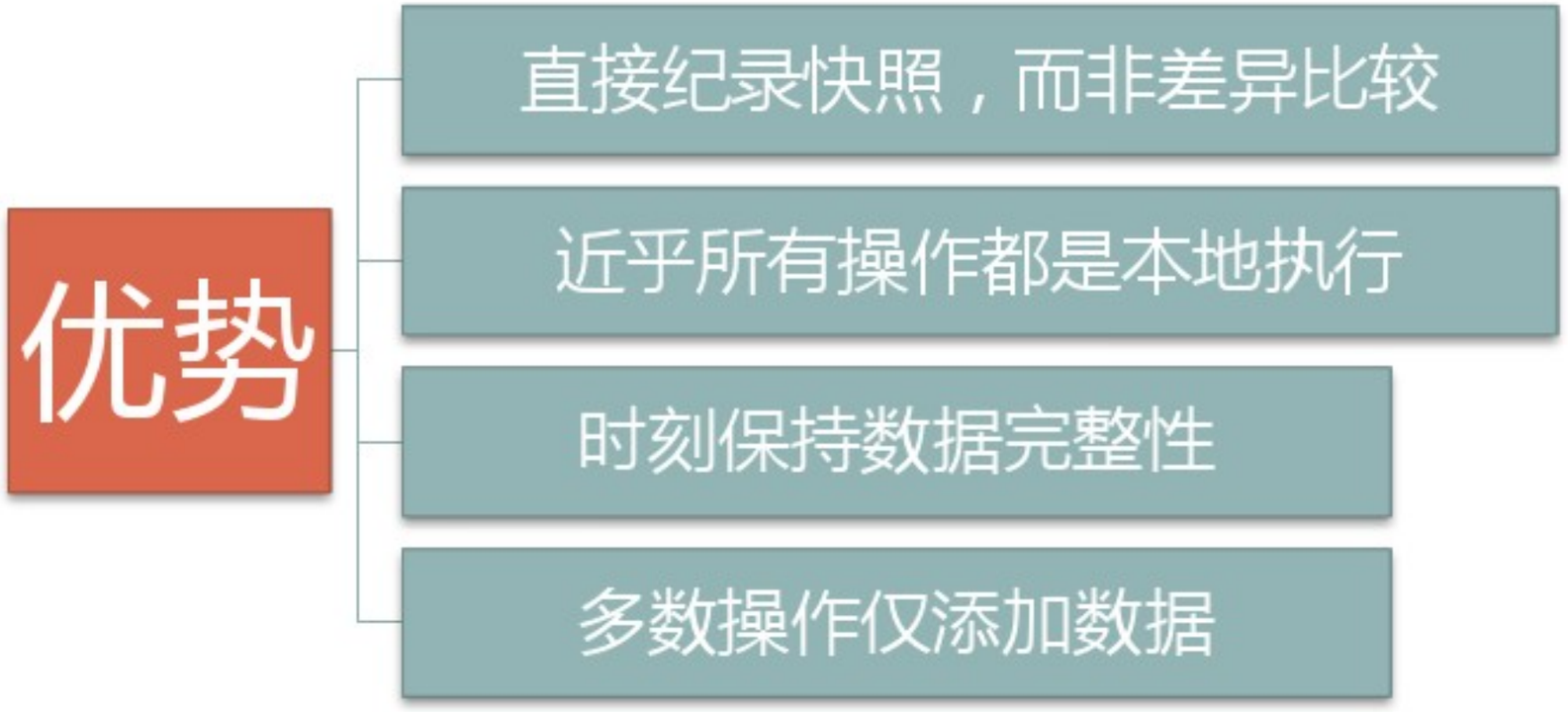
Git基础



What is Git?

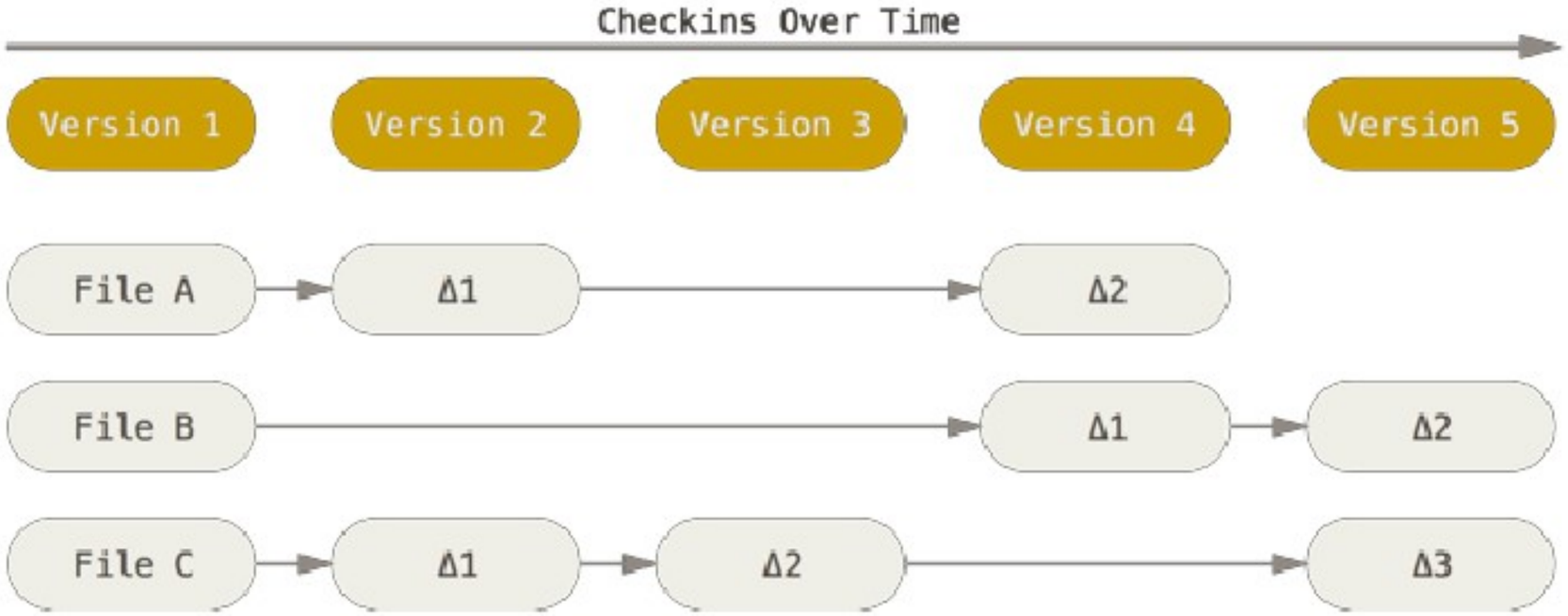
- 优势
- 工作原理

Git基础



集中式

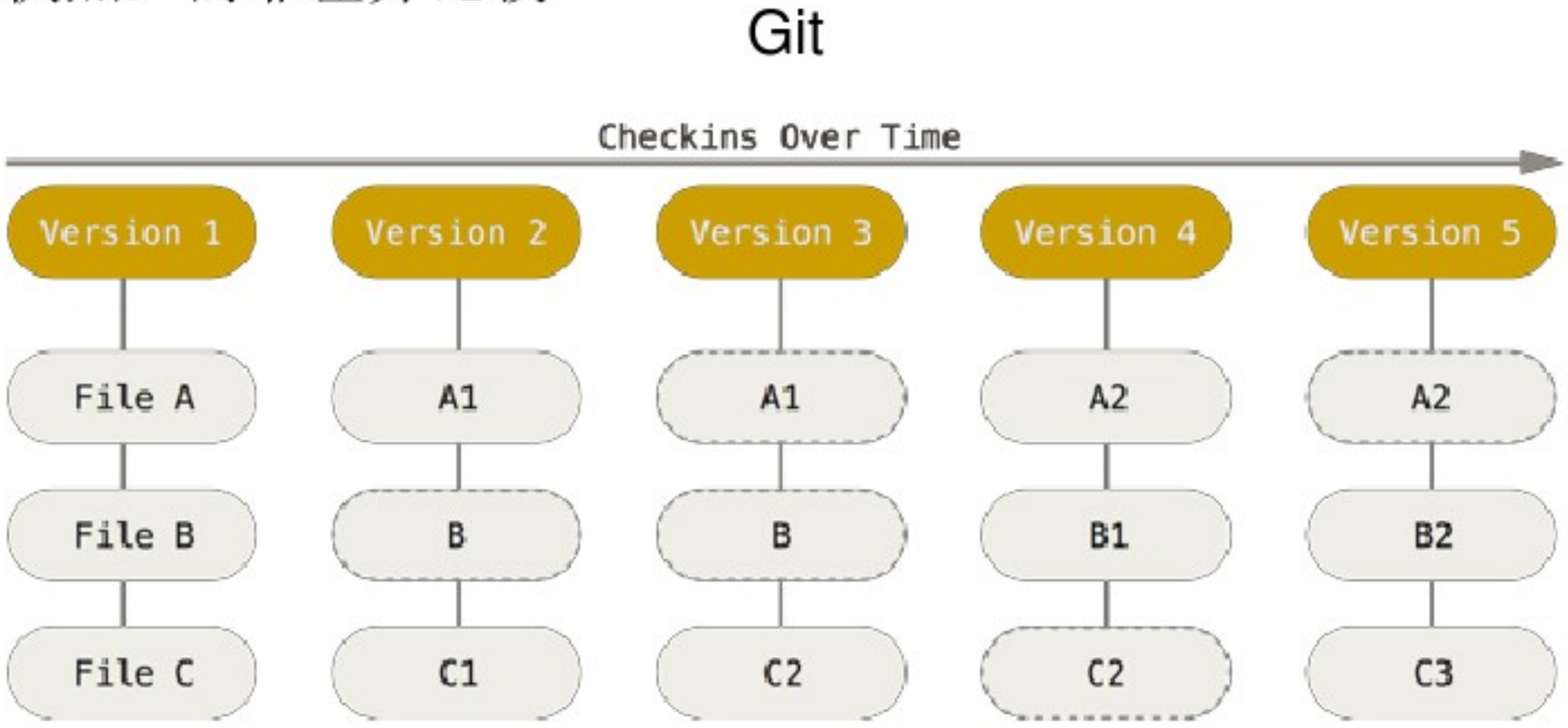
SVN、CVS等



在每个版本中记录着各个文件的具体差异

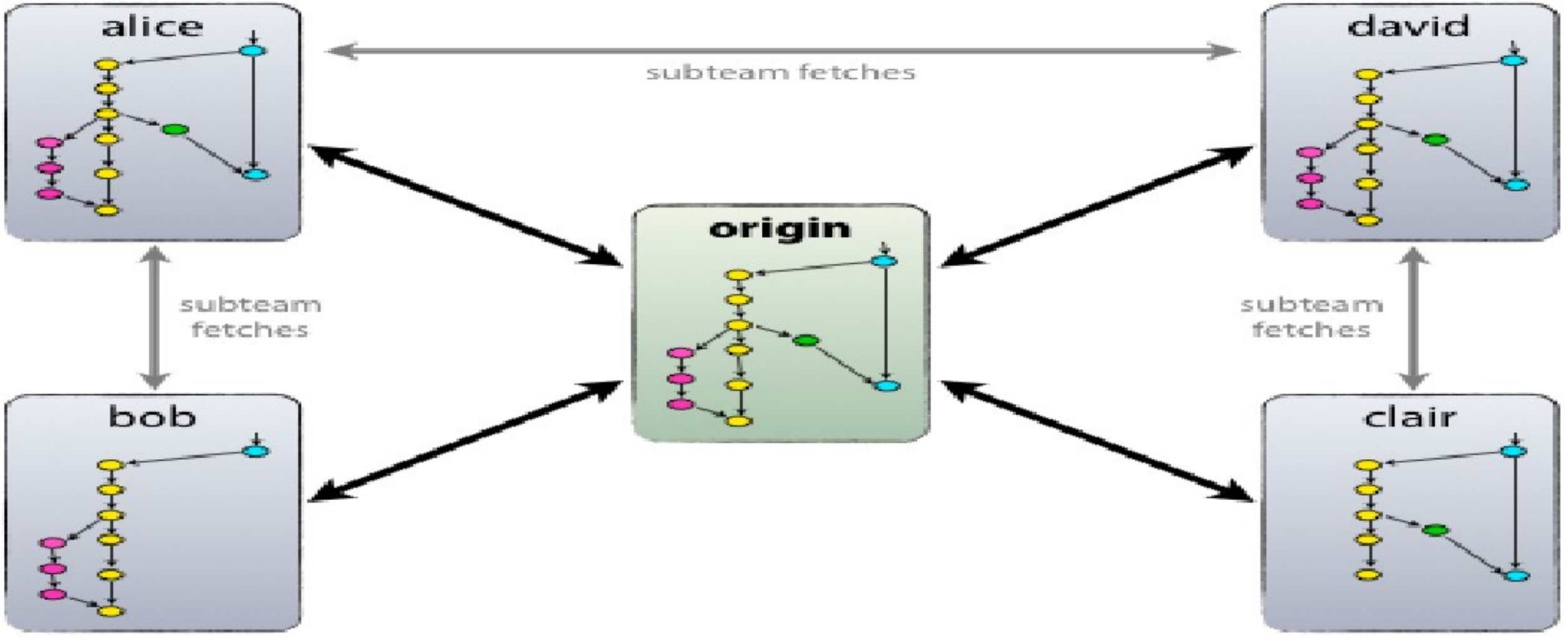
分布式--去中心却集中

直接纪录快照，而非差异比较



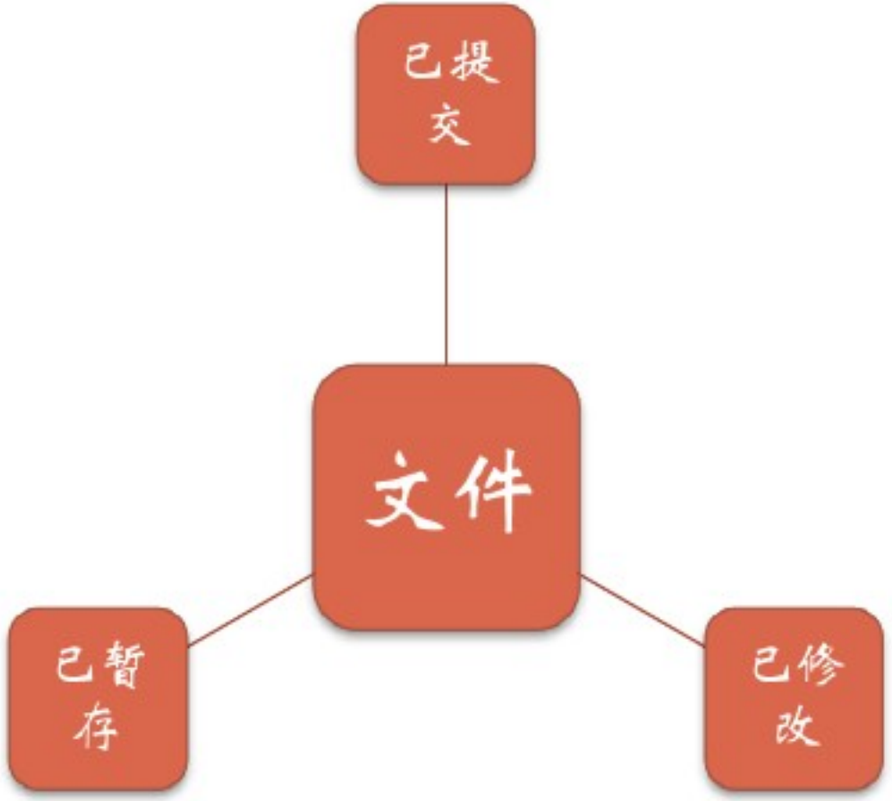
保存每次更新时的文件快照

分布式--去中心却集中



文件的三种状态

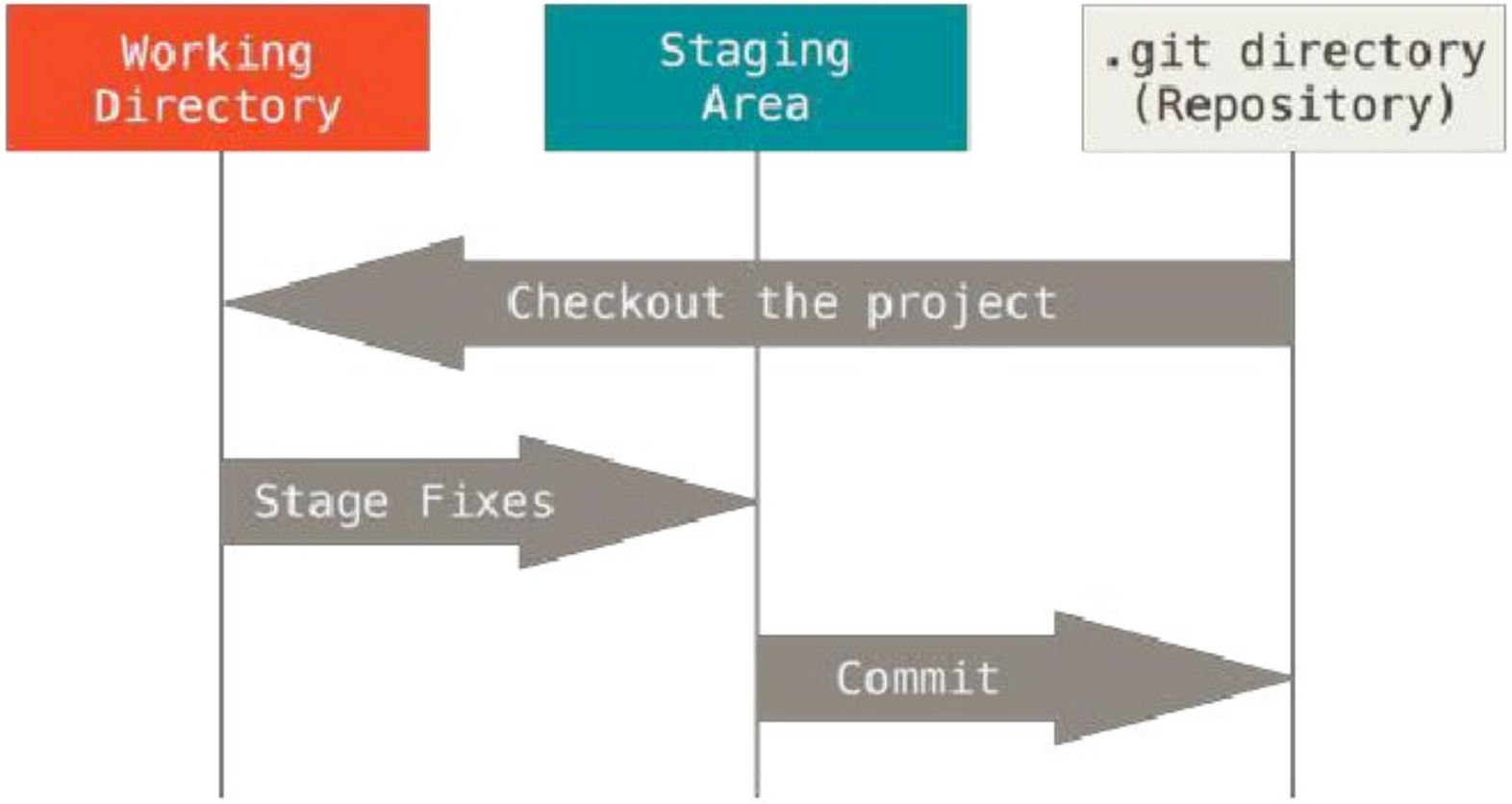
已被安全地保存到本地数据库中



把已修改的文件放在下次提交时要保存的清单中

修改了某个文件，但还没有保存

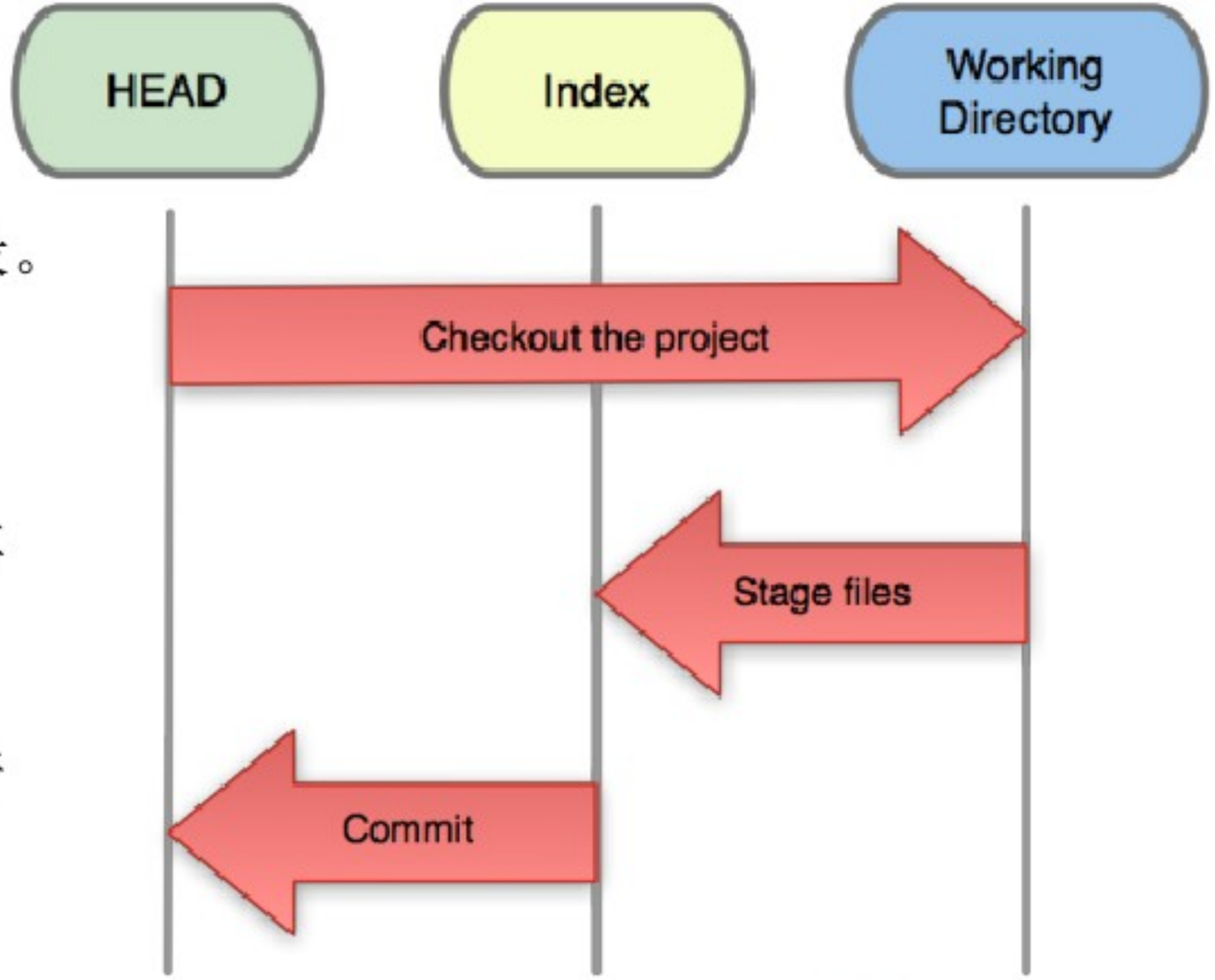
工作原理



文件流转的三个工作区域

Git基本工作流程

1. 从git仓库中checkout项目到工作目录。
2. 在工作目录修改某些文件。
3. 对修改后的文件进行快照，然后保存到暂存区域。
4. 提交更新，将保存在暂存区域的文件快照永久转储到Git目录中



大纲

Git简介

Git基础

Git操作

Git分支管理最佳实践git-flow

Git建议使用规范

Git 安装

- Linux
  - http://git-scm.com/download
  - $ yum install git-core
- Ubuntu
  - $ apt-get install git-core
- Windows
  - http://code.google.com/p/msysgit
- Mac
  - http://code.google.com/p/git-osx-installer

Git用户信息配置

- 配置你的个人用户名称和邮箱地址.
- 设置你默认使用的文本编辑器和差异比较工具
  - $ git config --global user.name “Zhang san"
  - $ git config --global user.email zhangsan@csdn.net

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

- 查看配置信息

"git config -l" command to list all the settings.

Git配置技巧

- 设置git别名
  - $ git config --global alias.co checkout
  - $ git config --global alias.br branch
  - .....

获取帮助

- **获取帮助命令:**
  - $ git help <verb>
  - $ git <verb> --help
  - $ man git-<verb>

- **比如，要学习config命令怎么用**
  - $ git help config

取得项目仓库

- **从当前目录初始化**

  到此项目所在的目录，执行

  - $ git init

- **从现有仓库克隆**

  - $ git clone url

创建忽略文件

- 通常都是些自动生成的文件，像是日志或者编译过程中创建的等等.无需纳入 Git 的管理，也不希望它们总出现在未跟踪文件列表.
- 我们可以创建一个名为 .gitignore 的文件，列出要忽略的文件模式，来看一个简单的例子：
  - cat .gitignore
  - *.[oa]
  - *~

Tips：

1.README文件也是需要创建的文件。这个相当于项目的简介首页，可以记录项目的版本发布历程，代码规范，代码命名规范，代码目录结构说明等，一个项目中可以有多个.md文件。

2.Gitignore文件前面的“.”不要忽略，该文件是手动创建的

3.gitignore只能忽略未被版本控制的文件

忽略文件配置规则

配置语法

以斜杠“/”开头表示目录；
以星号“*”通配多个字符；
以问号“?”通配单个字符
以方括号“[]”包含单个字符的匹配列表；
以叹号“!”表示不忽略(跟踪)匹配到的文件或目录；

示例

（1）规则：fd1/*
说明：忽略目录 fd1 下的全部内容；注意，不管是根目录下的 /fd1/ 目录，还是某个子目录 /child/fd1/ 目录，都会被忽略；
（2）规则：/fd1/*
说明：忽略根目录下的 /fd1/ 目录的全部内容；
（3）规则：

```
/*
!.gitignore
!/fw/bin/
!/fw/sf/
```

说明：忽略全部内容，但是不忽略 .gitignore 文件、根目录下的 /fw/bin/ 和 /fw/sf/ 目录

查看文件及本地版本库状态

```
$ git status
# On branch master
nothing to commit (working directory clean)

$ vim README
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#         README
nothing added to commit but untracked files present (use "git add" to track)
```

添加和跟踪文件

$ git add .　#将文件的修改，文件的新建，添加到暂存区。

$ git add -u　#将文件的修改、文件的删除，添加到暂存区

$ git add -A　#将文件的修改，文件的删除，文件的新建，添加到暂存区。

查看已暂存和未暂存的更新---Git diff 魔法

```
$ git diff    # 工作区和暂存区比较

$ git diff --cached   # HEAD和暂存区比较

$ git diff HEAD    # HEAD和工作区比较

$ git diff HEAD HEAD^   # HEAD和HEAD的父版本比较

$ git diff newbranch   # 比较本地工作区和新分支
```

提交更新

git commit

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#       modified:   benchmarks.rb
~
".git/COMMIT_EDITMSG" 10L, 283C
```

git commit -m ”-m用来直接增加注释提交”

移除文件

$ git rm filename #删除文件并存入暂存区

$ git rm –rf foldername #删除文件夹并存入暂存区

$ git mv [sourcefile] [destinationfile] #移动源文件为目标文件，相当于重命名

Demo: $ git mv readme.txt README
相当于 $ mv readme.txt README
$ git rm readme.txt
$ git add README

查看提交历史

$ git log ＃控制台查询

$ sourcetree ＃使用图形化工具查阅提交历史

查看提交历史

```
$ git log -5   # 最近5次提交

$ git log  -p # 查看所有提交日志及就该内容

$ git log  --graph # 以图形方式查看日志

$ git log  --pretty=oneline # 查看所有提交日志及就该内容
```

标签

$ git tag # 查看标签

$ git tag tag_name # 创建新的标签

$ git tag -d tag_name # 删除标签

$ git tag tag_name 哈希版本号 # 补打标签

$ git push origin tag tag_name # 推送本地标签到服务器

$ git push origin --tag # 推送本地所有标签到服务器

$ git push origin :refs/tags/tag_name # 删除服务器tag

撤销操作

$ git commit –amend #修改最后一次提交

$ git reset HEAD filename #取消已经暂存的文件

$ git checkout – filename #取消对文件的修改

$ git revert #反转提交

分支操作

$ git branch #查看本地分支，当前分支为绿色

$ git branch -a #查看所有分支

$ git checkout –b branch_name #创建新分支

$ git branch –d branch_name #删除分支

$ git checkout branch_name #切换分支

$ git push origin local_branch:remote_branch #将本地分支推送到server

$ git push origin :remote_branch #删除server端分支

$ git merge branch_name #合并分支

贮藏

$ git stash save “注释” #只能贮藏暂存区和未暂存的内容

$ git stash list #查看本地的贮藏

$ git stash pop --index stash@{0} #恢复贮藏

$ git stash drop stash@{0} #删除贮藏

远程仓库的使用

查看当前的远程库

```
$ git clone lvlin@192.168.60.45:repos/share.git
```

```
$ git remote -v
```

添加远程仓库

```
$ git remote add origin lvlin@192.168.60.45:repos/share.git
```

远程仓库的使用

推送数据到远程仓库

$ git push origin master

查看远程仓库信息

$ git remote show origin

大纲

Git简介

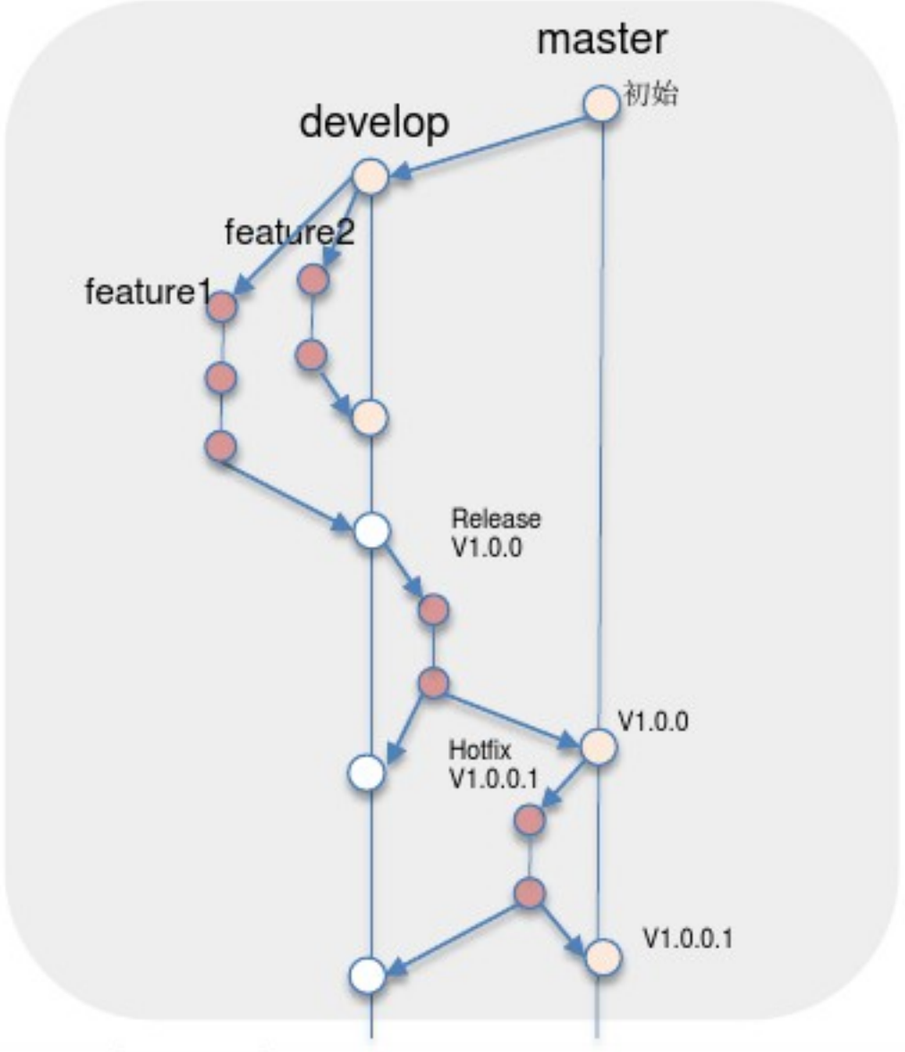
Git基础

Git操作

Git分支管理最佳实践git-flow

Git建议使用规范

Git-flow工作流



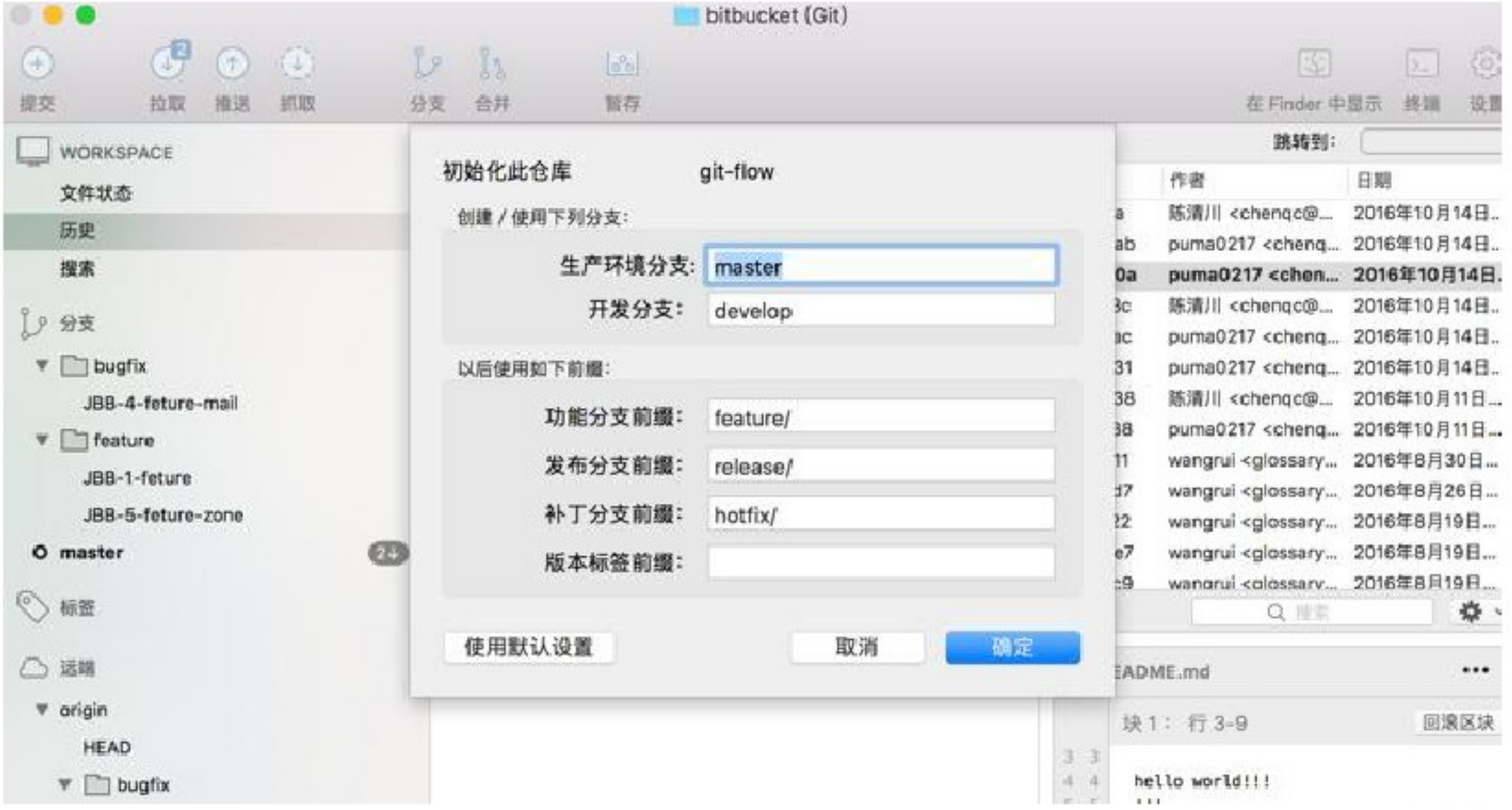
主干分支
Master ：随时可供在生产环境中部署的代码，建议伴有标签（TAG）
Develop：每天需要提交和合并的代码，功能逐渐完成的代码开发分支

辅助分支
Feature：新功能分支，辅助develop分支。主要用于实验性且效果不好的代码变更。或者用于项目组新成员接手开发新功能等。分支可以合并到develop分支，或者直接丢弃。
命名规范：feature-*

Release：当基本版本完成，准备提交上线时的分支，本分支可以做小BUG的修复。成功通过测试后，必须合并到Master分支，并记录标签（Tag），如果有BUGfix，则还需合并到Develop分支。此版本的作用是项目二期可以继续在develop分支开始开发。
命名规范：release-*

Hotfix: 对于线上版本（Master分支）的BUG修改的辅助分支，必须合并回master分支和develop分支。
命名惯例：hotfix-*

sourcetree上的git-flow



大纲

Git简介

Git基础

Git操作

Git分支管理最佳实践git-flow

Git建议使用规范

git建议使用规范

1. 代码要早提交，经常提交，不要嫌麻烦， 一旦本地调试成功，需要尽快的提交到服务器
2 在添加文件到缓存区时，必须使用git status / git diff查看，确保给缓存区增加正确的变化
3. 在提交时，必须使用git status / git diff --staged查看,确保提交正确的变化
4. 不建议使用 git push -f
5. 不建议使用 git commit –a
6. 切换branch时，一定要commit

7. 必须填写有效的comments信息，下面是一个范本

Present-tense summary under 50 characters

* More information about commit (under 72 characters).
* More information about commit (under 72 characters).

http://jira.csdn.net/browse/CSDNMOB-2068?jql=project%20%3D%20CSDNMOB

第一行是不超过50个字的提要，然后空一行，罗列出改动原因、主要变动、以及需要注意的问题。最后，提供对应的网址（比如Bug jira地址）。

下面给些反面教材

7.1：什么也不错
7.2：能跑了
7.3：解决了一些混账问题
7.4：改进了一点bug
7.5：上传了
7.6：修订1024
7.7：排字错误


愿景
为中国开发者创造价值
使命
做中国领先的开发者服务平台
BEYOND
CODE
用户导向、彼此信任、保持激情、主动求变
价值观