**Phabricator**

**安装及使用指南(中文1.0v)**

Multicorewareinc

陈晓伟 xiaowei@multicorewareinc.com

目 录

[介绍篇（introduction） 12](#_Toc385544066)

[安装向导 12](#_Toc385544067)

[安装要求 12](#_Toc385544068)

[安装必需的组件 12](#_Toc385544069)

[擦除mysql密码(可选) 13](#_Toc385544070)

[安装APC (可选) 14](#_Toc385544071)

[安装XHProf (可选) 14](#_Toc385544072)

[更新Phabricator 15](#_Toc385544073)

[配置篇（Configuration） 15](#_Toc385544074)

[软件/服务配置向导 15](#_Toc385544075)

[前提 15](#_Toc385544076)

[WEB服务器：配置Apache 15](#_Toc385544077)

[配置域名 18](#_Toc385544078)

[设置 18](#_Toc385544079)

[存储：配置MySQL 18](#_Toc385544080)

[验收 Phabricator： 19](#_Toc385544081)

[下一步 29](#_Toc385544082)

[配置账户及注册机制 29](#_Toc385544083)

[概述 29](#_Toc385544084)

[恢复管理员账户 30](#_Toc385544085)

[账户管理 30](#_Toc385544086)

[手动创建新账户 30](#_Toc385544087)

[备份配置及执行迁移 32](#_Toc385544088)

[概述 32](#_Toc385544089)

[备份：MySQL数据库 32](#_Toc385544090)

[恢复：MySQL 32](#_Toc385544091)

[备份：上载文件 33](#_Toc385544092)

[恢复：上载文件 33](#_Toc385544093)

[备份：配置文件 33](#_Toc385544094)

[恢复：配置文件 33](#_Toc385544095)

[安全性 33](#_Toc385544096)

[配置上载文件限制 34](#_Toc385544097)

[概述 34](#_Toc385544098)

[配置发送\接收邮件 35](#_Toc385544099)

[配置发送邮件 35](#_Toc385544100)

[配置接收邮件 39](#_Toc385544101)

[phd管理守护进程 43](#_Toc385544102)

[概述 43](#_Toc385544103)

[Phd简介 44](#_Toc385544104)

[守护进程控制台 44](#_Toc385544105)

[可用的守护进程 44](#_Toc385544106)

[调试和优化 45](#_Toc385544107)

[技巧总汇 45](#_Toc385544108)

[联机操作 45](#_Toc385544109)

[HTTPS 相关故障解决 46](#_Toc385544110)

[概述 46](#_Toc385544111)

[证书认证问题 46](#_Toc385544112)

[域名问题 46](#_Toc385544113)

[SNI 问题 47](#_Toc385544114)

[使用篇（Application User Guides） 47](#_Toc385544115)

[Arcanist快速入门 47](#_Toc385544116)

[安装Arcanist 47](#_Toc385544117)

[配置工程项目 48](#_Toc385544118)

[安装Arcanist证书 49](#_Toc385544119)

[发送可回顾修改集 49](#_Toc385544120)

[下一步操作 49](#_Toc385544121)

[Arcanist使用指南 50](#_Toc385544122)

[快速入门 50](#_Toc385544123)

[概述 50](#_Toc385544124)

[安装Arcanist 51](#_Toc385544125)

[安装Tab Completion 53](#_Toc385544126)

[配置Arcanist 53](#_Toc385544127)

[下一步操作 53](#_Toc385544128)

[Arcanist 使用教程: arc diff 54](#_Toc385544129)

[概述 54](#_Toc385544130)

[Subversion 54](#_Toc385544131)

[Git 54](#_Toc385544132)

[Mercurial 56](#_Toc385544133)

[上传和关闭修订 56](#_Toc385544134)

[一般性信息 57](#_Toc385544135)

[Arcanist 使用教程: Code Coverage 58](#_Toc385544136)

[使用Coverage特性 58](#_Toc385544137)

[启动Coverage (libphutil，Arcanist和Phabricator) 59](#_Toc385544138)

[构建coverage支持 59](#_Toc385544139)

[Arcanist使用教程: Commit Ranges 59](#_Toc385544140)

[概述 60](#_Toc385544141)

[配置基提交规则 60](#_Toc385544142)

[范例 62](#_Toc385544143)

[规则详解 62](#_Toc385544144)

[下一步 65](#_Toc385544145)

[Arcanist 使用教程: 配置一个新工程 66](#_Toc385544146)

[概述 66](#_Toc385544147)

[.arcconfig 基础篇 66](#_Toc385544148)

[.arcconfig 进阶篇 66](#_Toc385544149)

[历史记录的可变性 67](#_Toc385544150)

[如何定位库文件 68](#_Toc385544151)

[无 .arcconfig文件运行 69](#_Toc385544152)

[下一步 69](#_Toc385544153)

[Arcanist 使用教程: 自定义已存代码分析器 70](#_Toc385544154)

[概述 70](#_Toc385544155)

[普遍准则 70](#_Toc385544156)

[修改严宽规则 70](#_Toc385544157)

[屏蔽严宽规则 71](#_Toc385544158)

[运行多个规则集 71](#_Toc385544159)

[自定义指定代码分析器 71](#_Toc385544160)

[Arcanist 使用教程: 自定义代码分析器, 单元测试和工作流 72](#_Toc385544161)

[概述 72](#_Toc385544162)

[创建一个libphutil库 73](#_Toc385544163)

[使用特定类 73](#_Toc385544164)

[下一步 74](#_Toc385544165)

[Arcanist 使用教程: 代码分析器 74](#_Toc385544166)

[概述 74](#_Toc385544167)

[可用的代码分析器 74](#_Toc385544168)

[配置代码分析 75](#_Toc385544169)

[使用代码分析提高代码回查效率 76](#_Toc385544170)

[代码分析的哲学 76](#_Toc385544171)

[下一步 77](#_Toc385544172)

[Arcanist 使用教程: Mac OS X 77](#_Toc385544173)

[阐述如何在Mac OS X下安装Arcanist。 77](#_Toc385544174)

[下一步 77](#_Toc385544175)

[Arcanist 使用教程: 代码库钩子 78](#_Toc385544176)

[安装Arcanist SVN钩子 78](#_Toc385544177)

[Arcanist 使用教程: Windows 78](#_Toc385544178)

[概述 79](#_Toc385544179)

[必要组件 79](#_Toc385544180)

[配置编辑器 79](#_Toc385544181)

[安装PHP 80](#_Toc385544182)

[下一步 80](#_Toc385544183)

[审核（Audit） 使用教程 80](#_Toc385544184)

[概述 81](#_Toc385544185)

[详细介绍 81](#_Toc385544186)

[审核触发器 81](#_Toc385544187)

[审核在小型组的使用 82](#_Toc385544188)

[审核技巧 82](#_Toc385544189)

[下一步 82](#_Toc385544190)

[配置教程: 高级配置 82](#_Toc385544191)

[概述 82](#_Toc385544192)

[源码配置 83](#_Toc385544193)

[文件配置 83](#_Toc385544194)

[下一步 85](#_Toc385544195)

[Differential 使用教程 85](#_Toc385544196)

[概述 85](#_Toc385544197)

[如何进行代码回查 85](#_Toc385544198)

[创建修订 85](#_Toc385544199)

[Herald 规则 86](#_Toc385544200)

[Differential 使用技巧 86](#_Toc385544201)

[下一步 86](#_Toc385544202)

[Differential 使用教程: FAQ 86](#_Toc385544203)

[为什么“已接受”（accepted）的代码更新后依然是接受状态？ 86](#_Toc385544204)

[如何开启语法高亮？ 87](#_Toc385544205)

[空格选项是做什么用的？ 87](#_Toc385544206)

[淡绿和红色背景是什么意思？ 87](#_Toc385544207)

[Differential 使用教程: 重大修改 88](#_Toc385544208)

[概述 88](#_Toc385544209)

[Differential 使用教程: 测试计划 89](#_Toc385544210)

[概述 89](#_Toc385544211)

[全修改覆盖 89](#_Toc385544212)

[前端/用户可见修改 90](#_Toc385544213)

[Diffusion 使用教程 90](#_Toc385544214)

[概述 90](#_Toc385544215)

[代码库代号和提交名称 90](#_Toc385544216)

[添加代码库 91](#_Toc385544217)

[运行Diffusion守护进程 91](#_Toc385544218)

[下一步 92](#_Toc385544219)

[Diffusion 使用教程: 代码库钩子 93](#_Toc385544220)

[概述 93](#_Toc385544221)

[了解支持的协议 93](#_Toc385544222)

[配置系统用户账号 94](#_Toc385544223)

[配置 HTTP 95](#_Toc385544224)

[配置 SSH 95](#_Toc385544225)

[HTTP认证 96](#_Toc385544226)

[SSH认证 96](#_Toc385544227)

[Diffusion 使用教程：符号索引 97](#_Toc385544228)

[概述 97](#_Toc385544229)

[填充索引 97](#_Toc385544230)

[集成Differential 98](#_Toc385544231)

[Diviner 使用教程 98](#_Toc385544232)

[概述 99](#_Toc385544233)

[生成文档 99](#_Toc385544234)

[.book 文件 99](#_Toc385544235)

[Drydock 使用教程 100](#_Toc385544236)

[概述 100](#_Toc385544237)

[Events 使用教程：安装 Event 监听器 100](#_Toc385544238)

[概述 101](#_Toc385544239)

[安装 Event 监听器 (Phabicator) 101](#_Toc385544240)

[安装 Event 监听器 (Arcanist) 101](#_Toc385544241)

[监听器实例 101](#_Toc385544242)

[可用事件 102](#_Toc385544243)

[调试监听器 107](#_Toc385544244)

[下一步 108](#_Toc385544245)

[Herald 使用教程 108](#_Toc385544246)

[概述 108](#_Toc385544247)

[全局和个人规则 108](#_Toc385544248)

[规则, 条件和操作 109](#_Toc385544249)

[可用的操作 109](#_Toc385544250)

[测试规则 109](#_Toc385544251)

[高级 Herald规则 109](#_Toc385544252)

[Jump Nav 使用教程 110](#_Toc385544253)

[概述 110](#_Toc385544254)

[支持的指令 110](#_Toc385544255)

[libphutil 库 使用教程 111](#_Toc385544256)

[概述 111](#_Toc385544257)

[创建一个新库 111](#_Toc385544258)

[通过Phabricator连接 112](#_Toc385544259)

[类写入 113](#_Toc385544260)

[你可以扩展和引用的库 113](#_Toc385544261)

[Notifications 使用教程：安装和配置 114](#_Toc385544262)

[概述 114](#_Toc385544263)

[运行 Aphlict 服务器 114](#_Toc385544264)

[调试服务器错误 115](#_Toc385544265)

[Owners Tool 使用教程 115](#_Toc385544266)

[工具包 115](#_Toc385544267)

[相关提交 115](#_Toc385544268)

[需要注意的提交 115](#_Toc385544269)

[Phame 使用教程 116](#_Toc385544270)

[概述 116](#_Toc385544271)

[草稿 116](#_Toc385544272)

[帖子 116](#_Toc385544273)

[博客 116](#_Toc385544274)

[评论工具 116](#_Toc385544275)

[下一步 116](#_Toc385544276)

[Phriction 使用教程 117](#_Toc385544277)

[概述 117](#_Toc385544278)

[Remarkup 参考 117](#_Toc385544279)

[概述 117](#_Toc385544280)

[快速参考 117](#_Toc385544281)

[基本风格 118](#_Toc385544282)

[布局 118](#_Toc385544283)

[连接 URIs 123](#_Toc385544284)

[连接到对象 123](#_Toc385544285)

[内嵌对象 124](#_Toc385544286)

[引用文本 125](#_Toc385544287)

[内嵌多媒体类型 125](#_Toc385544288)

[图像宏 126](#_Toc385544289)

[模块 126](#_Toc385544290)

[提醒用户 126](#_Toc385544291)

[Phriction 文档 126](#_Toc385544292)

[文字块 126](#_Toc385544293)

[表格 127](#_Toc385544294)

[全屏模式 128](#_Toc385544295)

[Slowvote 使用教程 128](#_Toc385544296)

[概述 128](#_Toc385544297)

[Plurality vs. Approval 128](#_Toc385544298)

[使用教程: 账户角色 129](#_Toc385544299)

[概述 129](#_Toc385544300)

[管理员（Administrators） 129](#_Toc385544301)

[系统代理（System Agents） 129](#_Toc385544302)

[禁用账户（Disable Users） 129](#_Toc385544303)

[使用教程: 配置一个外置编辑器 130](#_Toc385544304)

[概述 130](#_Toc385544305)

[配置编辑器 130](#_Toc385544306)

[配置: TextMate （OS X） 130](#_Toc385544307)

[配置: 其他编辑器 131](#_Toc385544308)

[使用教程: 管理 Phabricator 的邮件 131](#_Toc385544309)

[概述 131](#_Toc385544310)

[减少邮件 131](#_Toc385544311)

[邮件规则 131](#_Toc385544312)

[邮件头 131](#_Toc385544313)

[使用教程: Review vs. Audit 132](#_Toc385544314)

[概述 132](#_Toc385544315)

[回查的优势 133](#_Toc385544316)

[审核的优势 134](#_Toc385544317)

[下一步 134](#_Toc385544318)

[使用教程: UTF-8 和字符编码格式 134](#_Toc385544319)

[概述 134](#_Toc385544320)

[检测和修复文件 134](#_Toc385544321)

[支持其他编码格式 135](#_Toc385544322)

**本文为官方手册翻译，如文中有不妥之处，可参考：**<https://secure.phabricator.com/book/phabricator/>

# **介绍篇（introduction）**

## **安装向导**

本节介绍获得以及运行Phabricator所需的最基本安装指引。

**简单的配置和安装过程可以参考：**

<http://alanmoment.ocomm.com.tw/blog/categories/phabricator/>

### 安装要求

Phabricator是一个LAMP应用套件，因此最基本的要求就是LAMP环境：

* Linux：Linux的不同发行版及变种是必需的。Mac OS X是一个可接受的Linux变种，Windows不是。Phabricator不能安装在Windows系统上。在Mac OS X，Amazon Linux，Ubuntu，RHEL和CentOS上运行的Phabricator有活跃的贡献者；如果你运行在其他的Linux发行版上，请发送补丁或 投诉。
* Apache(或nginx，或lighttpd)：需要Apache 2.2.7以上版本。
* MySQL：MySQL必需
* PHP：需要PHP5.2以上版本

注意：Phabricator的命令行接口--Arcanist，可以在Windows上正常运行。查看详细说明[Aranist使用教程：Windows](#_Arcanist_使用教程:_Windows)（官方页面：[Arcanist User Guide: Windows](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Windows.html)）。

你可能也需要一个域名和一台能连接互联网的电脑。

### 安装必需的组件

如果你在Ubuntu或Redhat及其衍生版本上安装，这里有可用的安装脚本，这些脚本可以自动处理下面文档中讨论的大部分操作：

* RedHat衍生版本：<http://www.phabricator.com/rsrc/install/install_rhel-derivs.sh>
* Ubuntu：<http://www.phabricator.com/rsrc/install/install_ubuntu.sh>

如果以上安装脚本可以正常安装，那么你可以直接跳过下面的操作，转到配置章节，了解如何配置。

这里对应的系统下载对应的安装脚本，这里在对应目录（推荐/usr/<name\_you\_like>下）执行wget <http://www.phabricator.com/rsrc/install/install_ubuntu.sh> 即可。下载完毕后执行，执行脚安装本按照屏幕提示进行操作。下面列出的必要软件，以及Phabricator源码都会由脚本来下载/安装。

这里还要注意的是在安装mysql时，会有几次让你输入root密码的提示。这里推荐将密码设置为空，以方便后面的使用。如果不幸，你已经设置了root密码，就需要将这个密码擦除，参考之后“擦除mysql密码”步骤。

否则，如下所列将是你必须安装的：

* git（在包管理系统中通常叫作git）
* Apache（通常叫作httpd或apache2）（或nginx）
* MySQL服务器（通常叫作mysqld或mysql-server）
* PHP（通常叫作php）
* 必需的PHP扩展：mbstring, iconv, mysql (或 mysqli), curl, pcntl (或者 "php-mysql" 或 "php5-mysql")
* 可选的PHP扩展：gd, apc (安装该扩展如果有困惑，请看下面的APC安装说明), xhprof (下面有说明，仅在你对Phabricator做二次开发时需要)

如果你已经设置好LAMP环境，你可以已经获得你所需的任何东东。

既然你已经安装以上所需的软件，下面获取Phabricator以及其依赖包：

$ cd somewhere/ # pick some install directory 切换到安装目录

somewhere/ $ git clone git://github.com/facebook/libphutil.git

somewhere/ $ git clone git://github.com/facebook/arcanist.git

somewhere/ $ git clone git://github.com/facebook/phabricator.git

### 擦除mysql密码(可选)

1. 停止mysql进程

/etc/init.d/mysql stop 或 service mysql stop

2. 无密码进入mysql

mysql –skip-grant-table &

3. 擦除root密码

当执行完以上语句，可以看到命令行为“mysql>”或与之类似的标示，这时可以通过SQL语句进行操作。

Mysql> use mysql;

Mysql> update user set password=password(“”) where user=”root”;

Mysql> flush privileges;

这里需要注意的是每句命令需要用分号“;”结尾。

执行完以上得操作，root的密码就被清空。

4. 启动mysql

/etc/init.d/mysql start 或 service mysql start

### 安装APC (可选)

如同所有写在PHP中的东西一样, Phabricator 在APC安装完毕后，运行会更快。你首先需要先安装“pcre-devel”：

sudo yum install pcre-devel #或 sudo apt-get install pcre-devel

然后你有两个选择。一个是安装via PECL（需要首先测试一下）

sudo yum install php-pear

sudo pecl install apc

**如果它不工作**, 在PECL文件夹中找到apc，具体的文档在：

http://pecl.php.net/package/APC

安装 APC 是可选的，但是这里 **强烈推荐安装**, 特别是在主机端。

一旦 APC 安装完成，测试就可以执行了：

php -i | grep apc

如果以上命令无法执行，那就添加：

extension=apc.so

..在 "/etc/php.d/apc.ini"文件中，或者 "php.ini"文件中 ，可以通过"php -i"来找对应的文件。

### 安装XHProf (可选)

XHProf 是一个 PHP 性能测试工具。你不需要安装它，除非你需要对Phabricatoe进行性能测试。

你可以通过如下命令行进行安装：

$ pecl install xhprof

如果你已经安装了 PEAR，并且是 1.9.3之前的版本, 你可能会运行phpize的时候失败。如果是这样，你可以下载其源文件，然后对其进行编译：

$ cd extension/

$ phpize

$ ./configure

$ make

$ sudo make install

你可能需要在你的php.ini文件中添加"extension=xhprof.so" 选项。

参考 <https://bugs.php.net/bug.php?id=59747> 获取更多相关信息。

### 更新Phabricator

由于Phabricator一直处于发展状态，所以你应该经常更新它。如何更新：

* 停止webserver的运行
* 运行git拉取libphutil/，arcanist/，phabricator
* 运行phabricator/bin/storage upgrade
* 重新启动webserver

参看配置向导以获取更多细节。你也可以使用类似如下的脚本以自动化的方式更新：

<http://www.phabricator.com/rsrc/install/update_phabricator.sh>

# 配置篇（Configuration）

## 软件/服务配置向导

本节包含了Phabricator所需的最基本的配置指引。

### 前提

该文档假定你已经安装了所有必需的组件。如果没有，请查看安装向导。

下一步：

* 配置你的webserver（Apache，nginx或lighttpd）
* 通过浏览器访问Phabricator
* 按照指引完成设置

### WEB服务器：配置Apache

注意：下面是Apache配置指引。如果使用nginx或lighttpd，可以访问官网的相关页面。

运行Apache，并使用一个测试页来验证其是否正常工作。

Apache测试需要做如下工作：

1. 加入sverver name

（1）修改httpd.conf，这个文件在刚装完Apache时应该是空的。(2.4.x版本在/etc/apache2路径下创建httpd.conf文件即可)

加入如下内容： ServerName localhost

（2）保存退出。

2.验证Apache安装

（1）重启Apache服务

/etc/init.d/apache2 restart

（2）验证页面

打开浏览器，在地址栏中输入 localhost。看是否会出现“It works”的字样，如果出现代表Apache运行正常。否则，就需要去查看一下httpd.conf是否配置正确了。

如果有问题，请查看Apache的帮助文档。确保mod\_php和mod\_rewrite启用，如果你设置SSL，请开启mod\_ssl模块。这里得模块都是默认开启的。

如果你还没有设置一个域名指向你将要安装的主机上。你可以安装Phabricator到一个二级域名（如 phabricator.example.com）上或一个完整域名上，但你不能安装到一个已经存在的网站的某个子目录下。输入你将要安装到的域名以确保 Apache可以为其正常服务，并且DNS已经正确配置。

注意：域名必须包含点（.），而不只是一个名称，如http://example/。否则，一些web浏览器将无法设置cookies。

现在，可以创建一个VirtualHost条目（放置Phabricator到一个二级域名上）或编辑Directory条目的DocumentRoot。将如下所示：

(2.4.x版本在/etc/apache2路径下创建httpd.conf文件即可)

**httpd.conf (/etc/apache2/httpd.conf)**

<VirtualHost \*>

# Change this to the domain which points to your host.

ServerName phabricator.example.com

# Change this to the path where you put 'phabricator' when you checked it

# out from GitHub when following the Installation Guide.

#

# Make sure you include "/webroot" at the end!

DocumentRoot /path/to/phabricator/webroot

RewriteEngine on

RewriteRule ^/rsrc/(.\*) - [L,QSA]

RewriteRule ^/favicon.ico - [L,QSA]

RewriteRule ^(.\*)$ /index.php?\_\_path\_\_=$1 [B,L,QSA]

</VirtualHost>

如果当前的Apache配置不能为你所放置的Phabricator文档目录服务，你需要添加如下的部分到httpd.conf中 。

当官方给出的配置方法无法让服务器正常工作，可以尝试以下的配置方式。

ServerName phabricator.example.com

<VirtualHost \*>

# Change this to the domain which points to your host.

# Change this to the path where you put 'phabricator' when you checked it

# out from GitHub when following the Installation Guide.

#

# Make sure you include "/webroot" at the end!

DocumentRoot /path/to/phabricator/webroot

RewriteEngine on

RewriteRule ^/rsrc/(.\*) - [L,QSA]

RewriteRule ^/favicon.ico - [L,QSA]

RewriteRule ^(.\*)$ /index.php?\_\_path\_\_=$1 [B,L,QSA]

</VirtualHost>

按照官方给出的去修改httpd.conf文件，Apache可能会找不到服务器。会有如下提示：

httpd: Could not reliably determine the server's fully qualified domain name, using 127.0.0.1 for ServerName（这里的127.0.0.1有时也会是127.0.1.1）

红色字部分与官方给出得不一致，这样在修改完重启Apache服务器后就不会出错。

适用与Apache版本为2.4之前版本

**Apache Older Than 2.4**

<Directory "/path/to/phabricator/webroot">

Order allow,deny

Allow from all

</Directory>

适用与Apache版本为2.4或之后版本

**Apache 2.4 and Newer**

<Directory "/path/to/phabricator/webroot">

Require all granted

</Directory>

更改后，重启Apache，然后跳转到下面的设置步骤。

### 配置域名

在以上的配置都正确的前提下，需要去修改hosts文件，需要对域名进行直接的映射。

Hosts 路径： /etc/hosts

添加：server机IP 域名

（例如： 127.0.0.1 hostlocal phabricator.example.com）

配置完hosts后需要对网络服务进行重启：

执行以下命令, 重启网络服务即可：

sudo /etc/init.d/networking restart

### 设置

现在，输入你设置的域名。你将会看到设置指引。文档的余下部分为其他具体设置步骤的附加说明。

### 存储：配置MySQL

设置中，你需要配置MySQL。运行MySQL，验证是否能正常连接。如果有问题，请参考MySQL的帮助文档。如果MySQL正常工作，你需要加载Phabricator的模式，运行命令：

phabricator/ $ ./bin/storage upgrade

根据提示进行操作即可，将Phabricator的配置文件加载到mysql中。

如果你配置了一个无特权的用户以连接数据库，你将不得不重新设置为root用户或其他的管理员以使模式能被应用。

phabricator/ $ ./bin/storage upgrade --user <user> --password <password>

可以使用 -- force 参数来避免脚本进行提示，对mysql强行加载配置文件。

phabricator/ $ ./bin/storage upgrade --force

注意：每当Phabricator进行了更新，都需要运行storage upgrade。

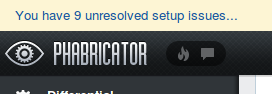
每当server关闭后再次开启，需要重新启动phabricator。页面信息会提示你该怎么做。

### 验收 Phabricator：

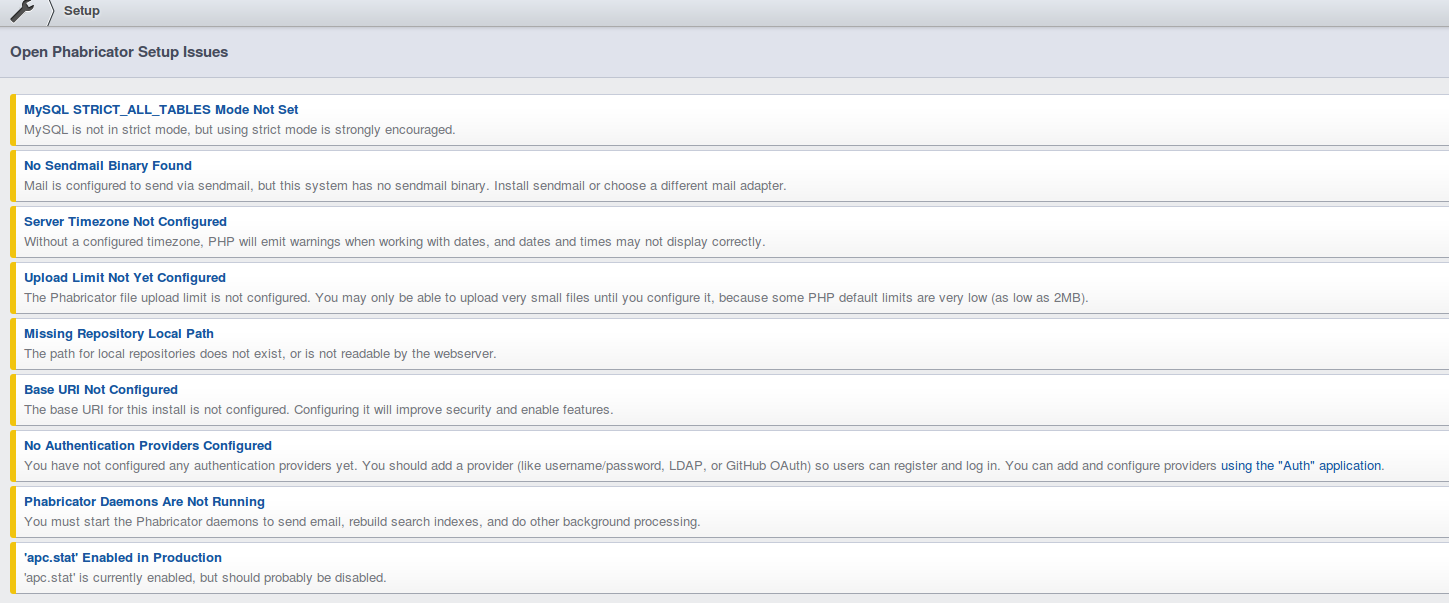
在MySQL配置完成后，可以到网页上去登陆一下 Phabricator，看工作是否正常。

登陆phabricator.example.com页面。会出现管理员注册界面，按照提示填写管理员的真实信息。

进入phabricator的主页面，在浏览器上部会出现如下图显示



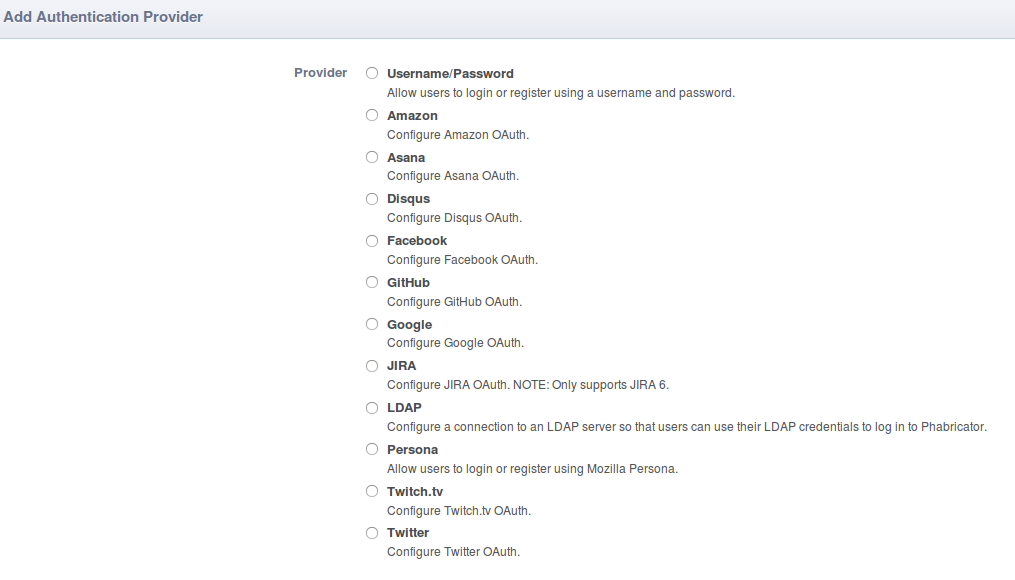
点击“you have 9 unresolved setup issues..”进入issue页面（我使用的当前phabricator版本是 2509c6b0f57dc533607fb6811adc02529b0d6fd8）。可以看到issue如下图所示。



其中有一个比较重要的设置，就是登陆界面的设置。

#### No Authentication Providers Configured（登陆issue）

点击进入issue页面 → 点击“using the "Auth" application” → 点击"Add Authentication Provider"，可以看到如下页面。



选择一种登陆或身份验证方式。这里我们选择“用户名/密码”的方式。确认添加后进入下一页，将所有选项都进行勾选（默认），也可以是根据需要进行设置。这里有一个最短密码长度的设置account.minimum-password-length，可以在之后进行设置。当再次确认后，登陆方式就被确认应用了。再次刷新页面，这个issue就被修复了。

如果不幸，你在管理员登陆完成后，没有去修复这个issue，下一次在想进入服务器就基本上是不可能了。

这里提供一个办法。因为使用得是mysql数据库存储我们的配置，这里我们可以将数据库中得配置进行删除或者说恢复成默认的。这里我们需要一个清理数据库的php文件，假设为clean.php。

<?php

$user = "root"; //数据库用户名

$password = "";//数据库密码

$db\_name = "mysql";//数据库名

$link = mysql\_connect("localhost:3306",$user,$password);

mysql\_select\_db("$db\_name",$link);

echo "<p>正在清空数据库,请稍等....<br>";

$result=mysql\_query("SHOW tables",$link);

while ($currow=mysql\_fetch\_array($result)) {

mysql\_query("drop TABLE IF EXISTS $currow[0]");

echo $currow[0]."<br>";

}

echo "<br>恭喜你清理MYSQL成功<br>";

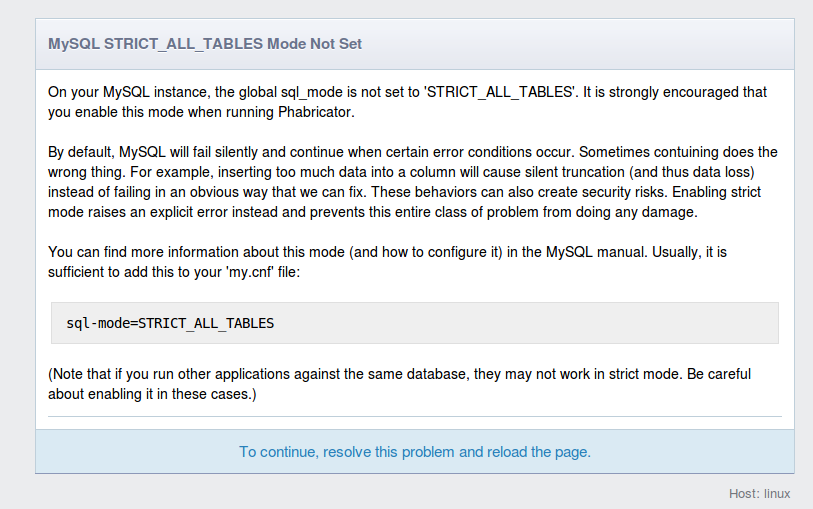
?>

之后运行php clean.php，就可以重新再登陆进入phabricator主页面了。

注意：这里root密码是空的。如果root有密码，可以参考 擦除mysql密码 步骤。

#### MySQL STRICT\_ALL\_TABLES Mode Not Set（mysql表格模式设定）

点击进入issue页面 就会出现下图所示的内容。



这个issue的解决办法：

在/etc/mysql/my.cnf中的 [mysqld]字段下面即可，类似这样：

[mysqld]

#

# \* Basic Settings

user = mysql

pid-file = /var/run/mysqld/mysqld.pid

socket = /var/run/mysqld/mysqld.sock

port = 3306

basedir = /usr

datadir = /var/lib/mysql

tmpdir = /tmp

lc-messages-dir = /usr/share/mysql

**sql-mode=STRICT\_ALL\_TABLES**

skip-external-locking

….

然后使用 /etc/init.d/mysql restart 重新启动mysql，使刚才的修改

可能会遇到的问题：

当将“sql-mode=STRICT\_ALL\_TABLES”放在了“skip-external-locking”的下一行时，会出现如下错误：

Can't connect to local MySQL server through socket '/var/run/mysqld/mysqld.sock'

在我的尝试的时候没有很好的办法来觉得这个问题，唯一的办法就是将mysql卸载干净后重装：

**删除 mysql**sudo apt-get autoremove --purge mysql-server-5.0  
sudo apt-get remove mysql-server  
sudo apt-get autoremove mysql-server  
sudo apt-get remove mysql-common //这个很重要  
上面的其实有一些是多余的。

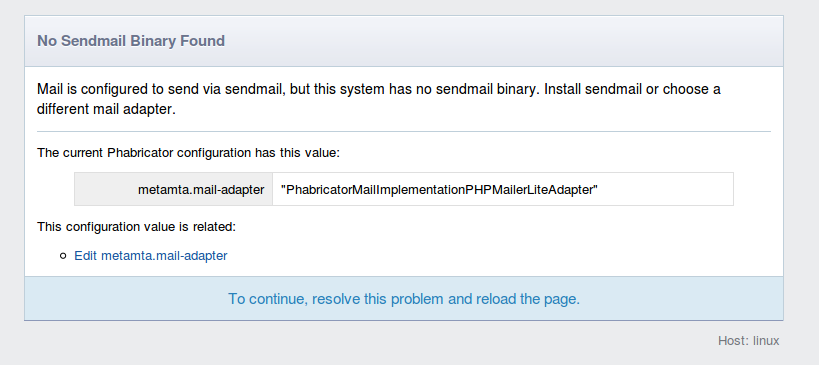
**清理残留数据**

dpkg -l |grep ^rc|awk '{print $2}' |sudo xargs dpkg -P

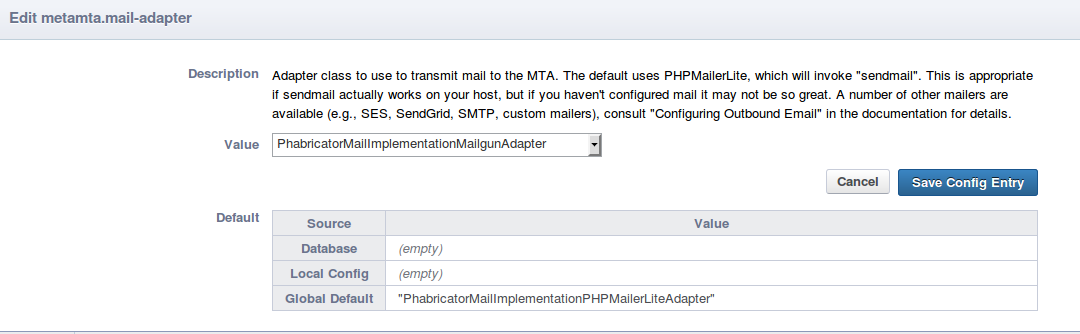
然后在用安装脚本进行安装，再运行bin/storage upgrade将初始化的数据导入数据库即可。

#### No Sendmail Binary Found（设置邮件服务）

点击进入issue界面后，会有如同下图的显示。



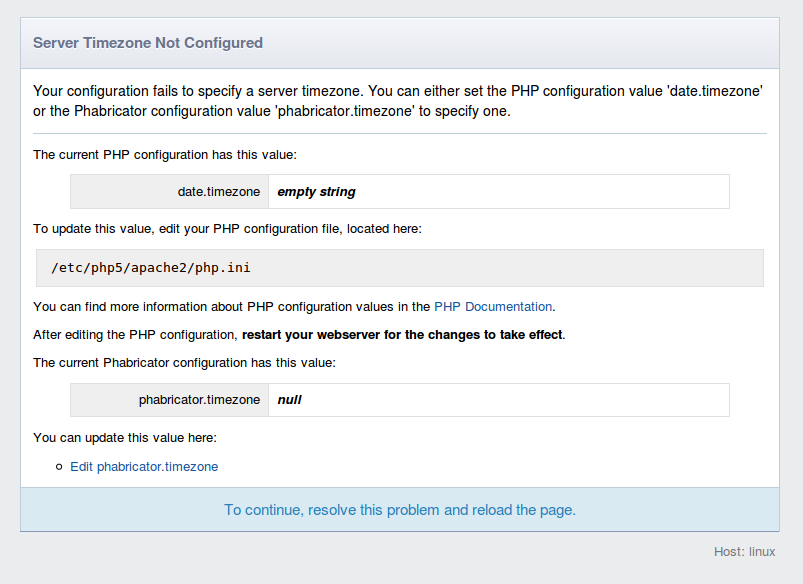
点击页面上的“Edit metamta.mail-adapter”，进入邮件适配器页面。可以参考下图进行配置：



选择合适的“value”，保存即可解决这个issue。

#### Server Timezone Not Configured（设置服务器的时区）

时区的配置是很重要的。php默认显示的时间都是格林威治标准时间,和北京时间差了正好8个小时。点击进入issue页面，会出现如下图的显示：



如图中说的，这里需要配置两个地方。

第一个地方：

修改/etc/php5/apache2/php.ini配置文件，在[date]字段后面找到被注释掉的“;date.timezone = ”，将注释打开。

但是填什么这是个问题。这里我们就要看一下页面中给出的“PHP Documentation”了。

打开“PHP Documentation”页面，找到“[date.timezone](http://cn2.php.net/manual/en/datetime.configuration.php#ini.date.timezone)”字符，点击进入对应页面。

随后打开的页面支持“简体中文”显示，

如果英文不太好的同学就可以在这里找“[所支持的时区列表](http://cn2.php.net/manual/zh/timezones.php)”

如果英文没问题的同学直接找“[List of Supported Timezones](http://cn2.php.net/manual/en/timezones.php)”

两个连接其实是一个，点击进入后，在页面的上部就有对应支持区域的连接，这里是“七大洲”+“太平洋地区”+“大西洋地区”+“其他”组成的。我们需要按照自己服务器所在地域来对时区进行设置。

我们这里选择Asia（亚洲），页面跳转后，可以看到一些知名城市的名字。这些城市里面，中国的城市有：（按页面中的出现顺序）

Asia/Chongqing （重庆）

Asia/Harbin （哈尔滨）

Asia/Hong\_Kong （香港）

Asia/Macao （澳门）

Asia/Shanghai （上海）

Asia/Taipei （台北）

Asia/Urumqi （乌鲁木齐）

所以，配置文件中写成date.timezone = Asia/Shanghai 即可。大家也可以根据自己的实际情况，对时区进行配置。

note1：网络上说是设置成“PRC”（中华人民共和国）也没有问题，不过在文档中，我们没有看到。保险起见，我们使用了Asia/Shanghai。

note2：很奇怪，文档里面没有北京

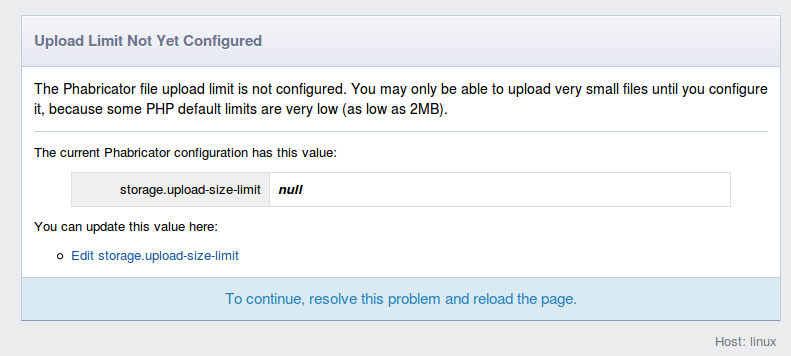
第二个地方：

再就是对phabricator的时区进行设置，点击“[Edit phabricator.timezone](http://192.168.0.90/config/edit/phabricator.timezone/?issue=config.timezone)”进入对应页面，在“Value”栏中输入“Asia/Shanghai”。然后保存设置即可。

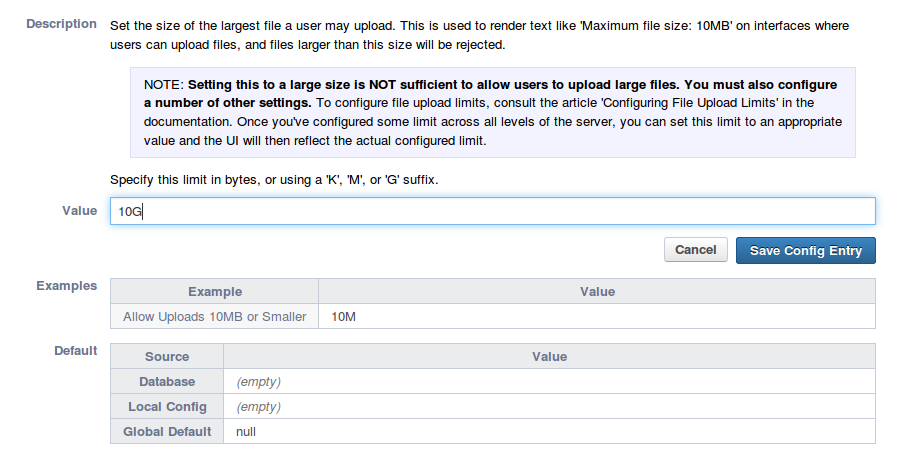
#### Upload Limit Not Yet Configured（配置上载文件大小）

这个在后期的管理以及对数据的备份上是比较重要的。

点击issue进入对应页面：



之后点击“[Edit storage.upload-size-limit](http://192.168.0.90/config/edit/storage.upload-size-limit/?issue=config.storage.upload-size-limit)”，在“Value”栏中填写你认为适当的大小，例如下图所示：

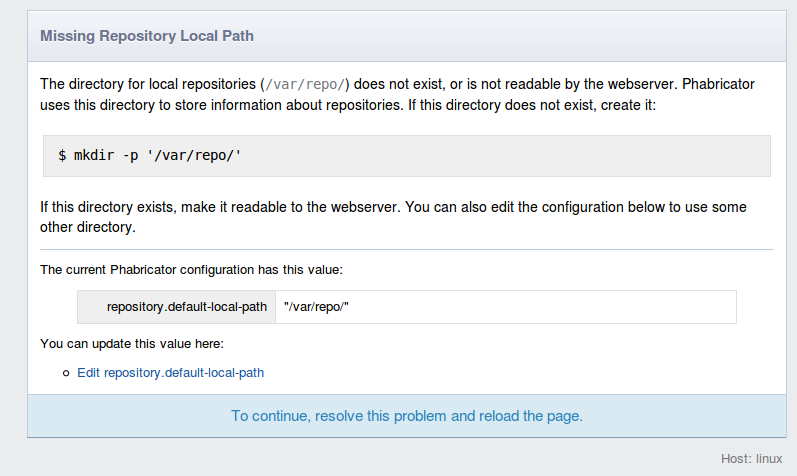


大家可以根据自己的需要进行配置。配置完成后，保存即可。

#### Missing Repository Local Path（配置本地库）

这个库不是用户的库，是Phabricator存储一些必要依赖文件的库。

点击issue，进入对应页面，会出现如下图所示的内容：



这里是Phabricator的默认地址，如果/var/repo/这个地址已经被其他的软件使用或者你想把他放到别的地方，可以点击“[Edit repository.default-local-path](http://192.168.0.90/config/edit/repository.default-local-path/?issue=repository.default-local-path.empty)”对其进行地址配置。

因为我们使用的是一个刚装好的系统，所以不存在/var/repo/这个目录，也不存在被占用的情况。我们就直接去创建它，然后刷新页面，这个issue也被修复了。

#### Base URI Not Configured（配置服务器基址）

这里会和域名解析相关。*个人不推荐设置这个基址，一旦设置了，只能通过对应域名或者IP进行访问。*

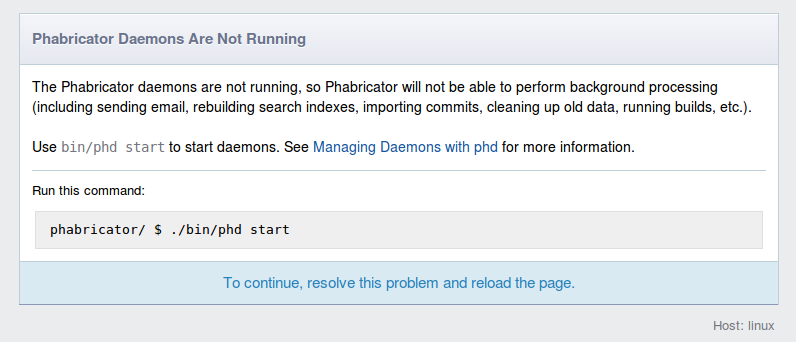
./bin/config set phabricator.base-uri '<http://phabricator.example.com/>'

或 ./bin/config set phabricator.base-uri 'http://192.168.0.90'

在phabricator安装目录下执行对应的命令后，提示“Set 'phabricator.base-uri' in local configuration.”。再刷新网页，该issue就被修复了。

#### Phabricator Daemons Are Not Running（开启守护线程）

守护线程的开启对于Phabricator，意味着可以启动后台程序，比如：发送邮件。



在phabricator安装目录下的bin目录下执行phabricator/bin/ # ./phd start 后，出现一下提示代表守护线程已经被开启。

Preparing to launch daemons.

NOTE: Logs will appear in '/var/tmp/phd/log/daemons.log'.

Launching daemon "PhabricatorRepositoryPullLocalDaemon".

Launching daemon "PhabricatorGarbageCollectorDaemon".

Launching daemon "PhabricatorTaskmasterDaemon".

Launching daemon "PhabricatorTaskmasterDaemon".

Launching daemon "PhabricatorTaskmasterDaemon".

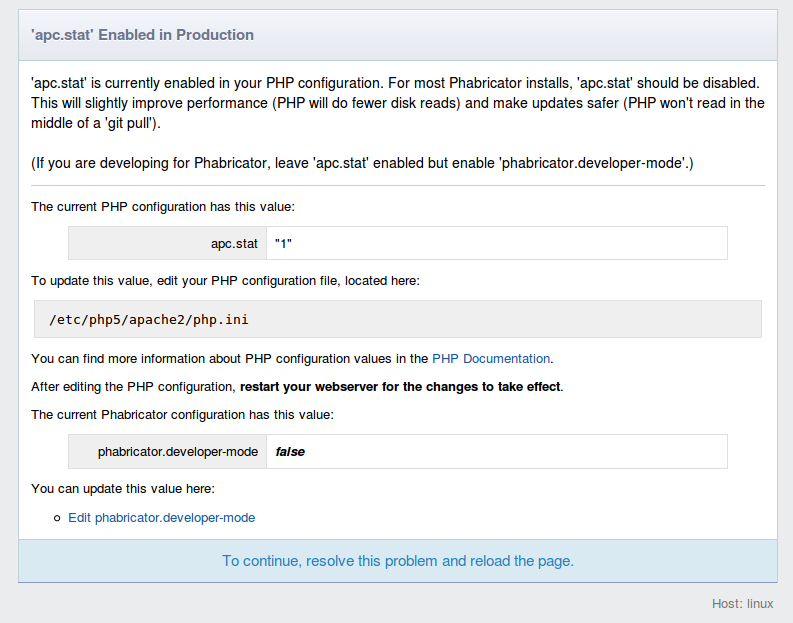
Launching daemon "PhabricatorTaskmasterDaemon".

Done

刷新页面后，这个issue也就被修复了。

#### 'apc.stat' Enabled in Production（开启phabricator开发者模式）

这里为了安全考虑，“phabricator.developer-mode”变量默认为false。而在php中对应的设置需要去从必要得文件中获取这些信息。



点击issue就有如上图所示的内容：

\* 如果不开启，需要去修改php的配置文件，不会影响使用。

\* 如果开启，进入“[Edit phabricator.developer-mode](http://192.168.0.90/config/edit/phabricator.developer-mode/?issue=extension.apc.stat-enabled)”页面，将开发者模式开启即可。

这里我们选择了开启，为了后期方便一些未知问题的处理。

### 下一步

继续：

设置你的管理帐号和登陆/注册，请参见 [配置账户及注册机制](#_配置账户及注册机制) （官方页面： [Configuring Accounts and Registration](http://www.phabricator.com/docs/phabricator/article/Configuring_Accounts_and_Registration.html);

* 了解更高级的配置主题，请参见 [Configuration User Guide: Advanced Configuration](http://www.phabricator.com/docs/phabricator/article/Configuration_User_Guide_Advanced_Configuration.html);
* 配置上传的文件和附件的存储位置，请参见 [Configuring File Storage](http://www.phabricator.com/docs/phabricator/article/Configuring_File_Storage.html);
* 配置Phabricator以支持发送邮件，请参见 [Configuring Outbound Email](http://www.phabricator.com/docs/phabricator/article/Configuring_Outbound_Email.html);
* 配置Phabricator以支持接收邮件，请参见 [Configuring Inbound Email](http://www.phabricator.com/docs/phabricator/article/Configuring_Inbound_Email.html);
* 导入版本仓库，请参见 [Diffusion User Guide](http://www.phabricator.com/docs/phabricator/article/Diffusion_User_Guide.html);
* 了解守护进程工作，请参见 [Managing Daemons with phd](http://www.phabricator.com/docs/phabricator/article/Managing_Daemons_with_phd.html);
* 配置备份工作，请参见 [Configuring Backups](http://www.phabricator.com/docs/phabricator/article/Configuring_Backups.html);
* 为Phabricator贡献力量，请参见[Contributor Introduction](http://www.phabricator.com/docs/phabricator/article/Contributor_Introduction.html).

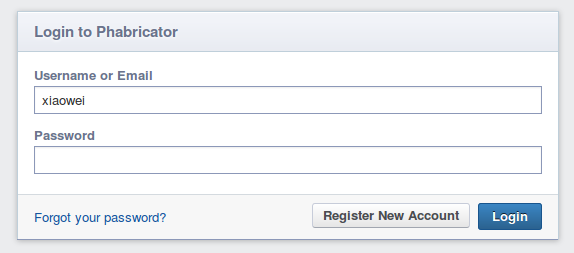
## 配置账户及注册机制

介绍如何配置，可以让用户对Phabricator进行访问。

### 概述

Phabricator 支持多种登陆系统。你可以通过配置这些登陆系统（开启或关闭），改变现有用户的登陆方式，以及新用户的注册方式。登陆方式这里称为“身份验证”。

例如，开启“用户名/密码”登陆模式，这是一种比较传统的登陆方式，使用“用户名”和“密码”匹配进行“身份验证”。



用户名/密码”登陆

这里还提供了其他的登陆方式，可以用某些“凭证”（或称为“证书”）来进行验证登陆。

例如:

* **Username/Password:** Users use a username and password to log in or register.
* **LDAP:** Users use LDAP credentials to log in or register.
* **OAuth:** Users use accounts on a supported OAuth2 provider (like GitHub, Facebook, or Google) to log in or register.
* **Other Providers:** More providers are available, and Phabricator can be extended with custom providers. See the "Auth" application for a list of available providers.

在Phabricator安装完毕后，默认情况下这些登陆系统都是不开启的，需要你使用具有管理员权限的用户添加一种或多种登陆系统，来支持用户的登陆/注册。

当你添加了一种方式后，你可以连接已经存在的账号（比如：可以直接将一个GitHub OAuth账户连接成Phabricator账户）。或者用户可以用这种方式注册新账号（前提：需要开启一些与注册有关的选项）

### 恢复管理员账户

如果管理员意外将用户锁住或无法登陆（比如：忘记密码），你可以使用Phabricator下的bin/auth来恢复管理员账户。使用下面的命令进行恢复操作：

phabricator/ $ ./bin/auth recover <username>

...<username>处是你要恢复的管理员账号的用户名。这个操作将会给出一个连接，使用该连接登陆，系统会将你认为是管理员用户。

### 账户管理

使用管理员账号登陆，到<phabricator>/people/ 或点击主页面上的“People”连接，之后就可以看到很多选项可以用来创建账户或者修改账户信息。

### 手动创建新账户

有两种手动创建账户的方式：

现阶段使用页面的方式配置账户，可能无法设置密码。

1.通过主页面上的“People”选项（简单）

2.通过CLI（command-line interface, 命令行界面）运行accountadimin脚本。（这种方式需要添加一些选项）。

有些选项（例如：密码和账户的级别）只能在命令行界面下使用。你可以通过这个方式恢复一个拥有管理员权限的用户（当你不小心将把你的管理员级别取消了）或者创建一个管理员或普通用户。

phabricator/ $ ./bin/accountadmin

下面给出对应的命令：(1.红色为手动输入。2.[Y/n]选项，大写为默认选项。)

phabricator/$ ./bin/accountadmin

Enter a username to create a new account or edit an existing account.

Enter a username: test

There is no existing user account 'test'.

Do you want to create a new 'test' account? [Y/n] Y

Enter user real name: wang

Enter user email address: wang@multicorewareinc.com

Enter a password for this user [blank to leave unchanged]:

Should this user be a system agent? [y/N] N

Should this user be an administrator? [y/N] N

ACCOUNT SUMMARY

OLD VALUE NEW VALUE

Username test

Real Name wang

Email wang@multicorewareinc.com

Password Updated

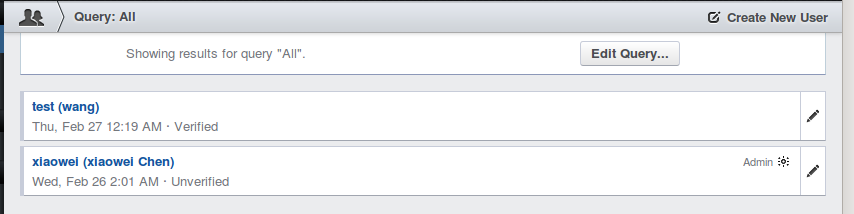
System Agent N N

Admin N N

Save these changes? [Y/n] Y

Saved changes.

当在命令行界面执行完毕后，“People”中的显示如下图：



使用命令行界面创建普通用户

## 备份配置及执行迁移

对phabricator备份的一些建议。

### 概述

目前Phabricator还没一个全面的备份系统，不过创建备份对于Phabricator也不是很困难。

Phabricator自带的的一些工具可以有效的帮助你完成备份工作。

下面列出的数据，很有必要去做备份。

* the MySQL databases; （数据库信息）
* uploaded files; （上传的文件）
* your Phabricator configuration files.（Phabricator 的配置文件）

下面我们将会讨论如何去备份这些数据

### 备份：MySQL数据库

Phabricator中的很多数据都存在MySQL数据库中，所以备份这些数据是很有必要的。

1. 你可以再Phabricator安装的目录下找到 bin/storage dump，该脚本可以获取MySQL中所有的数据库。
2. 我们也可以使用MySQL中自带的数据库获取脚本——mysqldump。这个脚本也可以完成获取MySQL所有数据库的功能。

从脚本中获取的数据可能会很多，这些数据时可以进行压缩。

以下的命令可以帮助你使用gzip来储存从MySQL数据库中获取出来的数据。

phabricator/ $ ./bin/storage dump | gzip > backup.sql.gz

然后，可以将备份好的压缩包放在一个安全的地方（如同将一个小盒子埋在老树桩下）。

### 恢复：MySQL

可以使用如下命令对MySQL进行恢复

$ gunzip -c backup.sql.gz | mysql

### 备份：上载文件

上载文件可能存储在不同的位置，备份过程需要得到文件存储的地址。

Default / MySQL: 在默认的情况下，上载文件都存在MySQL数据库中，所以备份MySQL数据库就等于备份了所有上载文件。所以，这里就不用做一些额外的工作。

Amazon S3: 如果使用的是Amazon S3 (Amazon Simple Storage Service，亚马逊简易储存服务)。当完全安装后，冗余和备份都内置在服务中。当你不大确定Amazon的服务能对你的数据进行备份是，你可以在Phabricator外围备份你的S3库。（感觉这番话说的没什么用）

Local Disk: 如果你使用本地磁盘存储引擎，你需要手动备份上载文件。你可以将存储着数据的根目录进行简单的复制，将复制的备份放到安全的地方保存。（根目录可从storger.localdisk.path配置变量中得到）

关于如何存储文件的信息，可以参考[Configuring File Storage](http://www.phabricator.com/docs/phabricator/article/Configuring_File_Storage.html)。

### 恢复：上载文件

将备份完毕的数据做回拷贝，去替换对应位置的数据即可。

### 备份：配置文件

配置文件也是很有必要做备份，这里包括一些部署和管理Phabricator的脚本（例如自定义升级脚本）。

这里比较好的做法是：

1. 检查一下当前的配置文件能使系统正常工作。
2. 验证没有问题后，将他们存入一个私人库中，以便将来更新和还原。

### 恢复：配置文件

要恢复配置文件，只需要将正确的配置文件拷贝会正确的位置即可。

### 安全性

MySQL获取的数据没有内置加密，并且很多数据以原始形式和可访问形式存储在Phabricator中，所以给用户访问备份是一个很像给他们访问本机上运行的phabricator。

用户可以可以从备份中得到很多信息，例如：

* 读取了原则上他们应该看不到的数据
* 获取了Phabricator保存的所有邮件地址，以及秘钥文件
* 获取了其他用户的会话和管道标记，并效仿他们

一些信息会保存很久，所以即使一个很早之前的备份被泄露都会存在风险。在限制访问的phabricator主机或数据库的同时，也应该限制用户对备份的访问。

## 配置上载文件限制

### 概述

在多个应用程序层中，文件上传大小是由片段配置来限制的。

一般情况下，设定最大值上传值是多种限制因素中最有效的一个。有时需要调高上限的阀值，才能将要上载大文件上传到服务器端。

限制上载的因素有列举如下：

* + **HTTP Server**: HTTP服务器就会设置最大请求大小。当你的文件大小超出了限制，将会看到来自服务器的警告提示。以下限制指令，限制了请求上传文件的大小，所以要上传的文件不能大于上限阀值。
    - **Apache**: Apache 限制请求是通过 LimitRequestBody 指令。
    - **nginx**: nginx 通过 nginx client\_max\_body\_size 指令来限定. 这个指令的默认值是 1M。
    - **lighttpd**: lighttpd 通过 server.max-request-size 指令来进行限制.
  + **PHP**: PHP也有一些限制上载的指令。这些指令可以在php.ini文件中找到。
    - **upload\_max\_filesize**: 设置最大可上载文件指令。默认值为 2M。
    - **post\_max\_size**: 设置最大投递大小指令。 默认值为 8M。
    - **memory\_limit**: 对于上载的文件来说，在Phabricator调整内存限制之前，都需要读入到内存中去。如果文件大小超过了之前设置好的大小，PHP将会提示你。
    - **max\_input\_vars**: 当文件上载是通过HTML5的拖拽动作完成时，PHP会通过“投递参数”来分析文件。当上载文件里面有过多的“碎片”，可能会触发到max\_input\_vars 指令。（你可能需要在这里设置一些“天文数字”）
  + **Storage Engines**: 一些存储引擎也可以设置最大可上传文件值。上传文件时，你必须保证你的文件没有超过存储引擎设置好的上限。下面列出Phabricator支持的存储引擎：
    - **MySQL Engine**: 通过 Phabricator 的 storage.mysql-engine.max-size 变量来进行设置。
    - **Amazon S3**: 在Phabricator的实现中，上载文件的最大值被限定为5G。
    - **Local Disk**: 最大上载文件大小，小于磁盘剩余空间即可。
  + **Resource Constraints（资源约束）**: 文件上传的最大值也会受到硬件资源的影响。在上载文件被移动到存储引擎前，需要整体写进磁盘中；如果文件过大，将会无法写入到磁盘中。同样，服务器都会有些硬性限制，会从根本上阻止Phabricator 处理那些超级大的文件（GB/TB级别）。所以在某些情况下，Phabricator的表现可能会不大理想。
  + **Phabricator Master Limit**: 主要限制指令 storage.upload-size-limit, 将会在图形页面上显示最大上载限制值。

这些设置在Phabricator 不会都用到，这些值需要通过如下所示的手工设置的方式来完成。

* 在软件端（PHP, MySQL）设置你要限制的大小, 例如 100M。
* 将以上的设置都调整到很高，以便你能上载更多的文件。不过这里也需要对安全性进行商榷。
* 设置 storage.upload-size-limit 为你想要的值。
* 图形界面将会显示你设置的数值。
* 上传一个大文件，确定它在服务器上能正常“工作“。

## 配置发送\接收邮件

邮件收发功能在Phabricator中也是一个必不可少的功能。

在图形界面端，新用户的注册对这个功能的依赖是很强的，新用户可以通过邮件激活他们的账号。

### 配置发送邮件

#### 概述

Phabricator 通过各种不同得适配器发送邮件：

* 本地主机上基于SMTP的sendmain服务
* 本地主机上基于SMTP的 postfix
* 亚马逊的SES （Simple Email Service，简单邮件服务）
* SendGrid的 REST API
* 自己写的适配器
* 将邮件存放在一个地方，并不发送它

在这其中, sendmail 是默认选项，但是需要做一些配置 。 SES 和 SendGrid 配置起来都很简单,但是需要花钱，并且还有一些限制。写一个自定义得解决方案，需要去深挖代码。 下面将会对每种配置方法进行详细介绍。

Phabricator 也可以用以下两种方式发送邮件:

* 马上发送，当信息产生后（默认）
* 通过域名后台发送

在后台发送邮件需要启动 Phabricator得守护进程，但是可以大大提高医用程序得速度。更多使用域名的信息，可参考 [Managing Daemons with phd](http://www.phabricator.com/docs/phabricator/article/Managing_Daemons_with_phd.html).

#### 基本配置

无论以何种方式发送电子邮件，你都要在相关的配置文件中配置这些关键字。

* metamta.default-address 决定默认的邮件发送方。如果你的域名为example.org，就要设置一些类似这样的邮箱地址： "noreply@example.org".
* metamta.domain 需要设置你的域名，例如 "example.org"。
* metamta.can-send-as-user 在大多数情况下应该是false ，更多信息需要浏览 default.conf.php 的文档。

#### 配置邮件适配器

选择如何发送邮件, 配置关键字metamta.mail-adapter 的值。可能的值如下所示：

* PhabricatorMailImplementationPHPMailerLiteAdapter: 默认值， 使用 "sendmail", 可见"Adapter: Sendmail"节.
* PhabricatorMailImplementationPHPMailerAdapter: 使用 SMTP, 可见 "Adapter: SMTP"节
* PhabricatorMailImplementationAmazonSESAdapter: 使用 Amazon SES, 可见 "Adapter: Amazon SES"节.
* PhabricatorMailImplementationSendGridAdapter: 使用 SendGrid, 可见 "Adapter: SendGrid"节.
* Some Custom Class You Write: 使用自己写的适配器, 可见 "Adapter: Custom"节.
* PhabricatorMailImplementationTestAdapter: 这个值将**完全禁用**邮件发送。如果你不想对外发送邮件，可以使用该值。或者暂时跳过这一步，以后再来配置。

#### Adapter: Sendmail

这是默认选项, 使能时需要选择 PhabricatorMailImplementationPHPMailerLiteAdapter 作为 metamta.mail-adapter的值。需要有sendmail服务已经在系统中安装。 大多数 MTA (例如，sendmail, qmail, postfix) 都需要预装sendmail服务（在这个服务在你的机器上不是默认安装好的）。 为了安装顺利，可以参考相关内容，在你喜欢的MTA的文档中。

既然你要发送邮件，就应该有一些东西，类似SPF规则、黑洞和MTA设置等等。不过，这已经超过了本指南的范畴。如果你已经可以用命令行发送电子邮件或者知道应该如何配置它了，这个选项就很明朗了。如果你不知道应该怎么去把邮件发送出去，可以参考使用亚马逊SES。

#### Adapter: SMTP

对于多数情况都是使用SMTP发送邮件，你可以使用 'sendmail' 或 'postfix' 来做这件事。 但是，多数SMTP服务器需要认证，以及'sendmail'邮件发送器不工作。如果你想试试postfix, 为了安装顺利, 参照'postfix 作为 MTA的文档，然后你可以配置使用SMTP。

为了使 Phabricator使用SMTP， 需要对如下的关键字进行配置：

* metamta.mail-adapter: 设置为 "PhabricatorMailImplementationPHPMailerAdapter".
* phpmailer.mailer: 设置为 "smtp".
* phpmailer.smtp-host: 设置smtp服务器的名字
* phpmailer.smtp-port: 设置smto服务器对应的端口。
* phpmailer.smtp-user: 设置用户名，用于认证。
* phpmailer.smtp-password: 设置密码，用于认证。

#### Adapter: Amazon SES

亚马逊SES是亚马逊云邮件服务。它是付费得，但是比sendmail服务更容易去配置，还有更简单的发送邮件配置。使用亚马逊SES，你需要在亚马逊上注册一个账号，可以通过这个连接 <http://aws.amazon.com/ses/> 进行注册。

为了使Phabricator 使用亚马逊SES, 设置如下得关键字:

* metamta.mail-adapter: 设置为 "PhabricatorMailImplementationAmazonSESAdapter".
* amazon-ses.access-key: 设置你的亚马逊SES访问键。
* amazon-ses.secret-key: 设置你的亚马逊SES密钥。

NOTE: 亚马逊SES需要验证你的“发送方”地址。通过 "metamta.default-address"关键字配置默认“发送方”地址，然后按照亚马逊SES验证过程进行验证。在验证没有结束之前，你是不能发送邮件的。

NOTE: 亚马逊 SES接收邮件有延迟（大概1～2秒），并且使用后台发送邮件的方式，会提高应用程序的性能。

#### Adapter: SendGrid

SendGrid 是一个邮件传送服务，类似于亚马逊 SES。 你可以从该连接得到更多信息 [http://sendgrid.com/](http://sendgrid.com/#_blank)。它配置起来比较容易，但是付费的。

可以用两种方式来配置SenfGrid：

1. 通过SMTP。
2. 通过REST API。

如果使用SMTP，仅仅需要配置sendmail服务，并使用Phabricator安装后的默认值。

如果使用REST API， 按照以下说明进行。

配置 Phabricator使用SendGrid，设置如下关键字：

* metamta.mail-adapter: 设置成"PhabricatorMailImplementationSendGridAdapter"
* sendgrid.api-user: 设置SendGrid登陆名
* sendgrid.api-key: 设置SendGrid密码

当你登陆到了你的SendGrid账户，你可以在（[http://sendgrid.com/developer](http://sendgrid.com/developer#_blank)）很容易找到这些信息 。

#### Adapter: 自定义

你可以通过一个子类实例，提供自定义的适配器 。

metamta.mail-adapter 需要设置成[PhabricatorMailImplementationAdapter](http://www.phabricator.com/docs/phabricator/class/PhabricatorMailImplementationAdapter.html)

TODO: 需要有较好的记录，来描述这一次对 Phabricator的扩展。

#### Adapter: 关闭邮件收发机制

当你不需要发送任何邮件或不想对它进行配置时，你可以使用 [PhabricatorMailImplementationTestAdapter](http://www.phabricator.com/docs/phabricator/class/PhabricatorMailImplementationTestAdapter.html) 完全让邮件发送机制失效。

将 metamta.mail-adapter 设置成 "PhabricatorMailImplementationTestAdapter"。

#### 配置MetaMTA使用守护进程发送邮件

无论是如何从服务器端发送出电子邮件的， 你可以切换到MTA主处理过程和进入一个守护线程。这将极大的提高应用程序得性能。特别是对于 Differential Revisions和 Maniphest Tasks操作，发送出“通知”邮件。

如果你设置 metamta.send-immediately 为false , MetaMTA 将队列中得邮件通过PhabricatorTaskmasterDaemon （一个守护线程）进行发送。 更多有关使用守护线程的信息，可参考 [Managing Daemons with phd](http://www.phabricator.com/docs/phabricator/article/Managing_Daemons_with_phd.html).

#### 测试/调试发送邮件

你可以使用 bin/mail 脚本来进行测试、调试、检查发送邮件。

phabricator/ $ ./bin/mail list-outbound # List outbound mail.

phabricator/ $ ./bin/mail show-outbound # Show details about messages.

phabricator/ $ ./bin/mail send-test # Send test messages.

运行 bin/mail help <command> 会有更多有关这些命令的帮助信息。

你可以使用守护程序控制台，对守护线程进行监控 (/daemon/, 或者在主页面点击 Daemon Console)。

### 配置接收邮件

#### 序言

这里有几种基本方法可用：

* 使用 SendGrid ([http://sendgrid.com/](http://sendgrid.com/#_blank)), 使用起来很简单，但是收费.
* 使用你己有的MTA（Mail Transfer Agent, 邮件传输代理），配置起来有些折腾人，不过免费。
* 如果你想使用其他服务来支持这个功能，请让Phabricator开发组协助你，否则配置起来可能会有遇到很多的麻烦。

#### 配置 Phabricator

在默认的情况下, Phabricator 使用 "noreply@phabricator.example.com" 邮件地址作为发送方 (可以通过配置 metamta.default-address变量来改变这个默认值)，以及为用户生成“回复/通知”邮件（例如，通过评论）。当用户进行某些操作时，就会生成了邮件通知。 用户可以通过回复邮件来讨论之前的改动，不过，Phabricator 不会记录这些讨论内容，并且用户不能通过这种邮件的方式进行一些操作，例如：发布任务，要求改变当前版本的修订。

为了让Phabricator 捕捉到这些行为，并能让用户在 Phabricator通过邮件的方式对服务器进行操作，需要对下面变量进行设置：

metamta.differential.reply-handler-domain:

对 Differential (commit Tasks and Bugs)使用邮件通知。

metamta.maniphest.reply-handler-domain:

对 Maniphest (Repository Browser)使用邮件通知。

可以通过如下的说明，来设置这几个 “域名变量”。例如：主页地址为 "phabricator.example.com"。 一般的做法是将他们都设置在同一个“域”内，例如“example.com”。 当设置完成并确定正确后，发送出去的邮件将会使用“回复”地址，例如"T123+273+af310f9220ad@example.com"。并且系统会分析“回复”地址收到的邮件，允许用户使用 Maniphest发布任务，以及使用 Differential功能去修改当前版本的修订。

如果你并不想phabricator 占用整个域(或子域)，可以配置一个通用前缀，这样的话可以使用一个单独的收件箱接收邮件。利用这个变量 metamta.single-reply-handler-prefix 来决定你所选择的前缀。并且，phabricator将会为“回复”地址，预先处理它。接收器的地址是上面所设置的前缀和'+'字符以及后面的域名，不过对于后面来说，这些在本质上都是被忽略的。

可以为任务创建设置一个个邮件地址，例如 bugs@example.com, 当它接收到封邮件时，都会去创建Maniphest任务。为了完成这个功能，需要设置 metamta.maniphest.public-create-email 。这里会对安全性有些影响，在后面我们会阐述。

#### 安全性

邮件回复通道被认证的。每个回复地址的收件人是唯一的、包含用户信息哈希值和唯一的目标ID，所以它只能用来更新目标对象的信息，并且只能代表收件人进行操作。

但是，这里便会有问题了。如果收件地址泄露，任何人都可以通过回复方式来对服务器上的目标文件进行操作。

在实现上，为了实用性和安全性的平衡，这两边都需要做出一些让步。因为，对于发送者得认证是比较困难的。例如，在开源项目中，需要和用户进行交互，用户的邮件我们是无法进行控制的。

如果不小心将一些回复泄露出去, 可以改变 phabricator.mail-key 的值使所有旧哈希值失效。

同样可以对 metamta.public-replies进行设置， 这个变量可以改变Phabricator 提供得电子邮件。并不是通过个人回复邮箱给每个收件人发送唯一邮件，而是使用一个公共回复地址给所有人发送单独的邮件。这种方式降低了安全性，因为任何人都可以以一个用户的身份欺骗“发送方”。但为了增强实用性，可以使用邮件列表。在“回复”地址中，仍然包含目标对象的哈希值。所以用户没收到包含目标对象的哈希值的邮件时，不能盲目的与邮件的发送方进行交互。

当你使能 metamta.maniphest.public-create-email变量时，会对“发送方”采用弱认证机制。

NOTE: 目前， Phabricator并不会试图去验证“发送方”地址，因为在技术方面比较复杂，在一般的情况下，这样做是没有必要的。如果你在一个比较特殊的环境下，可以通过一个合理的机制来让“发送方”提供认证来进行认证。（例如，DKIM（DomainKeys Identified Main, 域名密钥识别验证标准）签名，就可以验证“发送方”是否是“自己人”。或者你去要求所有得用户都在电子邮件里面添加这样的签名。）

#### 测试/调试接收邮件

你可以使用 bin/mail 脚本来测试和检查 接收邮件的功能。以下操作将会帮助你判断邮件是否能够投递到 Phabricator服务器端：

phabricator/ $ ./bin/mail list-inbound # List inbound messages.

phabricator/ $ ./bin-mail show-inbound # Show details about a message.

也可以对接收邮件进行测试，但是需要注意的是，这里只是模拟接收邮件的情况，并没有通过网络发送任何信息。它的主要目的是开发邮件处理程序：当你得接收邮件配置错误或者未使用，这个程序都会工作。

phabricator/ $ ./bin/mail receive-test # Receive test message.

运行 bin/mail help <command> 会有更多有关这些命令的帮助信息。

#### 配置SendGrid

为了使用 SendGrid, 你需要一个SendGrid账号为接收到得邮件做“API 解析”。 如果这个账号已经有了，就需要如下得配置：

* 根据SendGrid的指令来配置MX记录。 例如，添加 phabricator.example.com MX 10 mx.sendgrid.net.或类似的指令。
* 在SendGrid([http://sendgrid.com/developer/reply](http://sendgrid.com/developer/reply#_blank))主页面上，打开 "Parse Incoming Emails（解析接收邮件）" 页面。并添加一个域名，作为“主机名”。
* 添加URL地址 https://phabricator.example.com/mail/sendgrid/ 作为一个"Url", 使用你的域名 (如果没有配置SSL，需要将HTTP 改为 HTTPS)。
* 如果你从主机端得到了“an't be located or verified”的错误信息，那就意味着你得MX记录配置错误或者没有广播出去。
* 设置 metamta.maniphest.reply-handler-domain 和/或 metamta.differential.reply-handler-domain 为 "phabricator.example.com" (无论你是怎么去配置MX记录的), 这里怎么设置，取决于你要为Maniphest和/或Differential支持何种回复方式。

就是这样！如果一切运行正常，就应该能向 anything@phabricator.example.com 发送电子邮件。并且，过一段时间它应该出现在 bin/mail list-inbound 。

#### 安装邮件解析器

如果你想运行在自己得MTA上，就需要安装PECL邮件解析扩展。理论上，你可以这样做：

$ sudo apt-get install yum # if use yum in the first time

$ sudo pecl install mailparse

你也需会遇到错误，例如“needs mbstring”。如果是得话, 可以尝试:

$ sudo apt-get install yum # if use yum in the first time

$ sudo yum install php-mbstring # or equivalent

$ sudo pecl install -n mailparse

如果你遇到如下图的问题:

PHP Warning: PHP Startup: Unable to load dynamic library

'/usr/lib64/php/modules/mailparse.so' - /usr/lib64/php/modules/mailparse.so:

undefined symbol: mbfl\_name2no\_encoding in Unknown on line 0

你需要去修改你的php.ini文件，一遍使mbstring.so 在mailparse.so**之前**加载。这并不是默认的，如果有独立的文件在 php.d/目录下。

#### MTA: 配置邮件发送服务

在配置邮件发送服务之前，你需要安装邮件分析器。可以看一下前面提到的“安装邮件分析器”小节。

发送邮件配置起来也不简单。

首先，你需要去配置你得域名，以便邮件可以正确发送。 大体上来说，需要做如下事项：

* 添加一个MX记录
* 使邮件发送器监听外部接口
* 需要的时候，开放25端口（例如，使用EC2安全策略时）
* 在/etc/mail/local-host-names 添加你的主机
* 重启邮件发送器

现在，你可以配置邮件发送到Phabricator 服务器端了。在 /etc/aliases里面，添加如下的条目：

phabricator: "| /path/to/phabricator/scripts/mail/mail\_handler.php <ENV>"

...这里的<ENV> 就是 PHABRICATOR\_ENV 脚本在这个路径下运行。运行 sudo newaliases。 现在你可能需要在 /etc/smrsh/连接到这个脚本：

sudo ln -s /path/to/phabricator/scripts/mail/mail\_handler.php /etc/smrsh/

最后, 编辑 /etc/mail/virtusertable ，并添加如下的一个条目:

@yourdomain.com phabricator@localhost

这会将所有得邮件转发到 @yourdomain.com，并用Phabricator 处理脚本进行处理。运行 sudo /etc/mail/make 或类似的命令。然后，重启邮件发送服务 sudo /etc/init.d/sendmail restart.

#### MTA: 配置 Lamson

在配置Lamson之前，你需要安装邮件分析器。可以看一下前面提到的“安装邮件分析器”小节。

与之前的邮件发送服务不同，Lamson的配置相对容易很多。它很小，并且适合开发或测试环境。Lamson监听收到的SMTP邮件，并且将内同直接传递给 Phabricator。

To get started, follow the provided instructions 开始阶段，按照页面中提供的提示 设置实例([http://lamsonproject.org/docs/getting\_started.html](http://lamsonproject.org/docs/getting_started.html#_blank)) 。可能会碰到一个问题，就是在绑定Lamson到25端口时，会需要root权限。 Lamson是可以先使用root权限，之后让低权限用户使用，但是你必须提供 -uid 和 -gid 参数，如同Lamson部署教程中的步骤8 (给出对应连接:[http://lamsonproject.org/docs/deploying\_oneshotblog.html](http://lamsonproject.org/docs/deploying_oneshotblog.html#_blank)).

Lamson程序处理部分得代码十分简介; 它只传递电子邮件中的内容给 Phabricator：

import logging, subprocess

from lamson.routing import route, stateless

from lamson import view

PHABRICATOR\_ROOT = "/path/to/phabricator"

PHABRICATOR\_ENV = "custom/myconf"

LOGGING\_ENABLED = True

@route("(address)@(host)", address=".+")

@stateless

def START(message, address=None, host=None):

if LOGGING\_ENABLED:

logging.debug("%s", message.original)

process = subprocess.Popen([PHABRICATOR\_ROOT + "scripts/mail/mail\_handler.php",PHABRICATOR\_ENV],stdin=subprocess.PIPE)

process.communicate(message.original)

## phd管理守护进程

介绍Phabricator守护进程，以及守护进程控制程序 phd。

### 概述

Phabricator使用守护进程 (后台处理脚本) 来处理一些任务:

* 代码库跟踪, 发现新的提交, 导入库，以及分析提交信息。
* 发送邮件
* 垃圾收集，比如：旧日志和缓存文件。

守护进程的开启和停止都由 phd (Phabricator Daemon launcher)控制。 守护进程可以通过页面终端进行监视。

Phabricator 大部分的功能都用不到守护进程。但是Phabricator有些功能需要开启守护进程（通过Diffusion对代码库跟踪），还有就是如果你配置了守护线程，一些Phabricator的功能将会在性能或者稳定性上有很大的提高。

### Phd简介

phd 是一个命令行脚本 (位于 phabricator/bin/phd)。 想得到有关phd的命令及参数，可以通过运行phd help得到:

phabricator/ $ ./bin/phd help

NAME

phd - phabricator daemon launcher

...

大多数情况下，你会用到下面的命令:

* phd start 开启所有守护进程;
* phd restart重启所有守护进程;
* phd status 得到与phd有关的命令及参数;
* phd stop 停止所有守护进程;

如果你需要更加细粒度的控制，你可以使用：

* phd launch 加载单个守护进程
* phd debug 调试有问题的守护进程

NOTE: 当你升级Phabricator或修改配置后，需要通过phd restart来重启守护进程。

### 守护进程控制台

可以通过页面中的守护进程控制台接口，观察守护进程的状态和调试信息。在Phabricator安装目录下，找到 /daemon/进行安装，或者在页面上点击Daemon Console 获取 "更多信息"。

在控制台你可以看到所有得守护进程，以及这些守护进程的信息。当守护进程出现问题的时候，你可以通过控制台得到错误信息，这些信息对你解决问题是很有效的。

NOTE: 最简单的调试方法是 phd debug，并能很快的找到问题所在。这将运行守护进程，而不加载它，这样你就可以在控制台看到进程信息了。

### 可用的守护进程

通过phd list可以得到一张可加载守护进程的列表:

* libphutil test daemons 一般情况下没什么用处，除非需要对守护进程基础框架做开发或者调试守护进程的问题。
* PhabricatorTaskmasterDaemon 执行任务队列中的任务。
* PhabricatorRepositoryPullLocalDaemon 追踪代码库。更多信息可参考 [Diffusion User Guide](http://www.phabricator.com/docs/phabricator/article/Diffusion_User_Guide.html)。
* PhabricatorGarbageCollectorDaemon 清理旧日志和缓存。

### 调试和优化

在多数情况下，phd start 加载了所有你需要的守护进程。但是，你可能想要使用更细颗粒的守护进程控制来调试守护进程、启动自定义守护进程，或者启动特殊守护线程，比如：IRC bot。

想要调试一个守护进程，需要使用 phd debug:

phabricator/bin/ $ ./phd debug <daemon>

你可以效仿如下方式进行参数传递（一般参数是传递给守护线程控制机制，而不是直接传递给守护进程本身）：

phabricator/bin/ $ ./phd debug <daemon> -- --flavor apple

在调试模式下，守护线程未被线程化（或初始化），并且你可以在控制台端看到更多调试信息。这将使调试错误变得更简单。可以通过 ^C（Ctrl+C）来终止守护线程。

需要加载一个非标准的守护进程，需要使用 phd launch:

phabricator/bin/ $ ./phd launch <daemon>

这个守护进程将被进程化，并且正常运行。

### 技巧总汇

* phd start一次开启多少个“监工（守护进程）”可以通过phd.start-taskmasters进行配置。如果你有任务阻塞了，可以尝试增加开启的“监工”数。
* 当你使用 phd launch 或 phd debug 加载/调试一个守护进程，你可以输入守护进程名字中的任何独特子串， 使用 phd launch pull 同样能正常工作。
* phd stop 和 phd restart将会停止机器上所有的有关phabricator的守护进程, 不仅仅是 phd start开启的，只要相关的守护进程都会停止。phd restart只会开启phd start的那些守护进程。如果你需要加载自定义守护进程，需要在phd restart之后，加载你的自定义进程。
* 你可以写一个自定义守护进程作为 [PhabricatorDaemon](http://www.phabricator.com/docs/phabricator/class/PhabricatorDaemon.html)的扩展，并且让 phd进行管理 。更多信息可以参考 [libphutil Libraries User Guide](http://www.phabricator.com/docs/phabricator/article/libphutil_Libraries_User_Guide.html)。
* 参考 [Diffusion User Guide](http://www.phabricator.com/docs/phabricator/article/Diffusion_User_Guide.html) 有关于优化代码库跟踪的守护进程的更多信息。

### 联机操作

如果你使用多台机器， 你需要使用phd launch 来调整守护进程的加载方式，以及通过以下关键字，在联机的情况下区分守护进程:

* PhabricatorRepositoryPullLocalDaemon: 可以在任何机器上都可以运行一个副本。每个前端页面无法正确运行正常的副本，这时需要使用--no-discovery 标示才能正常运行。
* PhabricatorGarbageCollectorDaemon: 可以在任何机器上都可以运行一个副本。
* PhabricatorTaskmasterDaemon: 使用多个副本，运行来自备份中的任务。你可以单机运行这些任务，也可以联机运行它们。

一个无端浪费的安装，可能会使一些守护进程使用专用守护进程器运行，phd start 只运行任务管理池中配置好的守护进程。在phd start 之后，还需要在web服务器端运行 phd launch PhabricatorRepositoryPullLocalDaemon --no-discovery 。这样使用的资源要比一般的任务多得多。

## HTTPS 相关故障解决

排除HTTPS上的连接问题的详细说明。

### 概述

当你安装Phabricator 碰到HTTPS连接错误，并收到 "There was an error negotiating the SSL connection." 的错误时，这里记录的一些方法可以帮你诊断和解决问题。

连接时失败可能有如下原因：

* 你还没有把当前证书发布机构作为一个信赖的权威。(这是比较常见得问题，通常为自签证书问题)
* SSL证书签署了错误的域名。例如, 签署在www.example.com的认证，将不会在phabricator.example.com上工作。
* 服务器拒绝 TLSv1 SNI连接域名 (这个问题比较复杂，可以参见下面的解决办法)。

### 证书认证问题

SSL 证书需要被当做一个可信赖得权威 ( Certificate Authority ,"CA") 才能被HTTPS接受。 如果证书没有被信赖，连接将会失败。 (之所以要这样做，是为了抵御一种叫做“中间人”的窃听攻击)。通常，你从一个知名权威机构购买一个证书，客户端会有一个认证机构列表，只要证书得发布机构在这个列表中，连接就会成功。

你也可以自造自己得CA签署的证书，但是客户端默认是对其不信任的。他们需要将这个CA加入可信赖权威机构列表中。

对于如何添加CA到可信赖机构列表, 可以参考 libphutil/resources/ssl/README。

可能有证书不是信赖的机构办法，但是也可以通过。这是因为这个功能/协议暂时不被支持，同时妥协于SSL的防窃听能力，非信赖机构得证书，也能通过验证。

### 域名问题

有验证证书得问题，可以采用如下的方式：

$ openssl x509 -text -in <certificate>

如果证书的产生是偶然事件， 例如： 现在使用www.example.com 安装Phabricator，但是你已经使用 phabricator.example.com安装了Phabricator ，你需要为正确的域名创建一个新的证书。

### SNI 问题

服务器名称识别(Server Name Identification ,"SNI") 是 TLSv1的一个特性。它的工作原理，有些类似于Apache 虚拟主机(VirtualHosts)，并且允许服务器加载 不同得证书，客户端可以通过使用不同的名字来进行连接对应的证书。

当服务器没有配置正确，也会拒绝 TSLv1 SNI访问，因为他们无法识别客户端要连接的名字。 这个问题比较麻烦，但是你可以运行如下得命令进行测试：

$ openssl s\_client -connect example.com:443 -servername example.com

通过"example.com"更换两个实例。如果你收到的了reason(1112)错误 SSL23\_GET\_SERVER\_HELLO ，如下所示:

CONNECTED(00000003)

87871:error:14077458:SSL routines:SSL23\_GET\_SERVER\_HELLO:reason(1112):

/SourceCache/OpenSSL098/OpenSSL098-44/src/ssl/s23\_clnt.c:602:

...这就表明服务的配置有问题。造成这个问题有共同原因，就是Phabricator的虚拟主机域名没有在Apache服务器端被明确，或被识别。

这个问题只发生一些版本的OpenSSL客户端上（v0.9.8r或更早的版本，直到1.0.0这个问题被解决。）

# 使用篇（Application User Guides）

## Arcanist快速入门

快速入门可以使Arcanist为一个全新的工程正常工作。

这里对安装Arcanist，配置/使用一个工程，以及使用arc 脚本发送可回顾的修改集，等步骤进行了一个总结。

### 安装Arcanist

有关安装Arcanist的详细文档，可见 [Arcanist User Guide](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide.html)。

* 对于 Mac OS X操作系统, 可见 [Arcanist User Guide: Mac OS X](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Mac_OS_X.html).
* 对于 Windows操作系统, 可见 [Arcanist User Guide: Windows](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Windows.html).

首先, 安装依赖（以下两项在ubuntu的安装脚本中已经包含）:

* 安装 PHP.
* 安装 Git.

然后，安装Arcanist本身（同样在安装脚本中完成）：

$ mkdir somewhere/

$ cd somewhere/

somewhere/ $ git clone git://github.com/facebook/libphutil.git

somewhere/ $ git clone git://github.com/facebook/arcanist.git

将 arc 脚本路径加入系统PATH环境变量中:

$ export PATH="$PATH:/somewhere/arcanist/bin/"

如果在Windows操作系统下不能正常工作，可参考文档 [Arcanist User Guide: Windows](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Windows.html) 。

### 配置工程项目

详细的工程配置说明文档，可见 [Arcanist User Guide: Configuring a New Project](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Configuring_a_New_Project.html).

创建一个 .arcconfig 文件在你的工作空间内:

$ cd yourproject/

yourproject/ $ $EDITOR .arcconfig

yourproject/ $ cat .arcconfig

{

"project\_id" : "yourprojectname",

"conduit\_uri" : "https://phabricator.example.com/"

}

例如：

xiaowei@linux:~/mxpa/mxpa\_runtime$ cat .arcconfig

{

"project\_id" : "mxpa\_runtime",

"conduit\_uri" : "http://192.168.0.90/"

}

通过一个字符串来设置 project\_id ，字符串的内容是项目名称，用来识别项目不同项目。

通过一个字符串来设置 conduit\_uri ，字符串的内容是Phabricator 的URI地址（这个变量也决定了arc脚本将要把修改集信息发送到哪里）。

NOTE: 你需要将这个文件**提交**到库中。这个工程下需要有.hg（Mercural）或 .git（Git）目录，否则下面内容将无法正确执行。

### 安装Arcanist证书

证书可以让你通过验证。在你执行这步之前，你必须在Phabricator有一个已存在的账号。

$ cd yourproject/

yourproject/ $ arc install-certificate

...

执行以上命令。这步操作将会连接你的在本地的用户账号和在Phabricator上的账号。

这里需要你有一个账户在Phabricator上登陆，当你以这个账户拿到taken。这个工程就为你安装成功了。信息如下：

xiaowei@linux:~/mxpa/mxpa\_runtime$ arc install-certificate

Installing certificate for 'http://192.168.0.90/api/'...

Trying to connect to server...

Connection OK!

**LOGIN TO PHABRICATOR**

Open this page in your browser and login to Phabricator if necessary:

http://192.168.0.90/conduit/token/

Then paste the token on that page below.

Paste token from that page: o65caiep5wq46od3paclbsragarnexvzwsriqzf2

（这个token需要通过点击上面的连接，登陆后获得）

Downloading authentication certificate...

Installing certificate for 'xiaowei'...

Writing ~/.arcrc...

SUCCESS! Certificate installed.

### 发送可回顾修改集

使用arc diff的详细文档，参见 [Arcanist User Guide: arc diff](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_arc_diff.html)。

$ $EDITOR file.c

$ arc diff

### 下一步操作

可以继续以下内容:

* 了解更多有关于项目配置的信息[Arcanist User Guide: Configuring a New Project](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Configuring_a_New_Project.html);
* 了解更多 arc diff 使用的信息 [Arcanist User Guide: arc diff](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_arc_diff.html);
* 继续下面的 [Arcanist User Guide](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide.html).

## Arcanist使用指南

Arcanist使用指南，是一个对Phabricator的命令行接口。

Arcanists提供以命令行的方式访问许多Phabricator工具（例如：Differential，Files和Paste），集成静态分析和单元测试，以及常见的工作流管理（例如：在Differential获得修改集进行回顾）

arc命令的详细参考，可以运行 arc help后看到。本文档仅提供了常见工作流程的概述，以及安装简介。

Arcanist 有技术文献，贡献者们的文档集中在这里:

[http://www.phabricator.com/docs/arcanist/](http://www.phabricator.com/docs/arcanist/#_blank)

### 快速入门

快速入门教程可以参考 [Arcanist Quick Start](http://www.phabricator.com/docs/phabricator/article/Arcanist_Quick_Start.html)。它为如何安装、使用、配置 arc ，提供了一个较为紧凑的总结，并且运行在一个全新的项目中。你可以从那里开始，当你看完了那些内容，觉得快速入门提供的信息过为精简，那么可以返回本节，获取更多你需要的信息。

### 概述

Arcanist 是一个包装了其他工具的脚本(例如： Differential, linters, unit test frameworks, Git, Mercurial, 和SVN工具) ，并且可以使用简单的命令行API来管理代码审查和一些相关的版本控制操作。

有关所有的可用命令信息，可通过运行下面的命令得到:

$ arc help

可以通过运行下面命令得到某个具体指令/工具的详解：

$ arc help <command>

Arcanist可以做如下的事:

* arc help ：获得arc中包装的可用指令/工具
* arc diff ：发送代码的差异Differential功能，以便回顾(有关该指令的更多信息， 可见 [Arcanist User Guide: arc diff](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_arc_diff.html))
* arc list :显示未提交修改的代码信息
* arc cover ：可以找到某个代码修改的提交人
* arc patch: 适应某已修改，并在这个修改上进行工作
* arc export：通过Differential功能下载/导出一个补丁文件
* arc amend：审核Git更新提交后的信息
* arc commit：提交SVN库的修改
* arc land：向服务器端推送Git 和 Mercurial库的修改
* arc branch：可以看到更多有关Git分支的信息

当你配置完了 lint 和集成单元测试（unit test integration）, 你还可以使用:

* arc lint：检查代码的语法和代码风格错误 (更多信息可见 [Arcanist User Guide: Lint](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Lint.html))
* arc unit：运行单元测试，对代码修改进行全方位的检查

Arcanist 集成的其他工具：

* arc download / arc upload：下载/上载文件
* arc paste：创建和查看粘贴（pastes）？（还没用，暂时不大清楚是干嘛的）

Arcanist 也拥有很多高级功能：

* arc call-conduit：通过管道方式执行
* arc liberate：创建或更新 libphutil 库文件
* arc shell-complete：完全激活某个标签（暂不清楚是干嘛的）
* arc svn-hook-pre-commit / arc git-hook-pre-receive：安装arc，可作为一个预提交修改的钩子。
* ...或者 在Arcanist扩展之后加入新指令

在没有另行声明的情况下，以上的工作流对于代码版本控制系统一般是未知，不过这些指令会在Git，Mercurial，或SVN仓库上正常工作。

### 安装Arcanist

Arcanist 需要安装在本地机器或开发服务器上---- 无论你在哪里修改代码，你都需要安装它。它可以运行在下列环境中：

* Linux 系统
* 类Linux 系统（与Lunux系统类似的系统）
* FreeBSD, 这是个不错的系统，很多人都推崇它
* Mac OS X (详细可见 [Arcanist User Guide: Mac OS X](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Mac_OS_X.html))
* Windows (详细可见 [Arcanist User Guide: Windows](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Windows.html))

Arcanist 已经写入PHP中, 你首先要安装命令行模式的PHP（PHP CLI，PHP Command Line Interface)，如果你还没有安装的话（Phabricator提供的安装脚本已经包含该应用的安装）。 Arcanist 需要在PHP 5.2或更高的版本上运行。你可以从这里得到官方的安装包 [http://www.php.net/](http://www.php.net/#_blank)

为了安装 Arcanist，先要选择一个安装地址，然后从GitHub上克隆（clone）Arcanist代码进行安装：(phabricator 提供的安装脚本已经完成该步骤)

some\_install\_path/ $ git clone git://github.com/facebook/libphutil.git

some\_install\_path/ $ git clone git://github.com/facebook/arcanist.git

当克隆完成后，将会是这样的一个目录结构（在安装目录下，会出现两个文件夹，一个为arcanist，另一个是libphutil）

some\_install\_path/ # Wherever you chose to install it.

arcanist/ # Arcanist-specific code and libraries.

libphutil/ # A shared library Arcanist depends upon.

现在可以添加 some\_install\_path/arcanist/bin/ 到PATH环境变量中（在之前开启和现在正在使用的终端，可使用source命令来使新环境变量生效）。

这样当你输入在终端中输入“arc”，你就看到如下所示的信息：

Usage Exception: No command provided. Try 'arc help'.

如果以上信息出现了，那么说明你的配置没问题，如果你出现了什么问题，可以参考以下教程中的信息：

* 在 Windows系统下: [Arcanist User Guide: Windows](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Windows.html)
* 在 Mac OS X系统下: [Arcanist User Guide: Mac OS X](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Mac_OS_X.html)

之后可以使用arc upgrade进行对Arcanist和libphutil的更新/升级

$ arc upgrade

#### 为一个团队安装Arcanist

Arcanist 改动很频繁，追逐最新版本会让人很头疼。这里给出一些参考办法，可能会帮助到你：

* Facebook 的大多数开发工作也是在开发服务器上，那里有一个标准的环境和挂载了一个网络文件系统（NFS，Network File System）。Arcanist 和 libphutil 存放在一个挂载的网络文件系统中，并且在默认的情况下在.bashrc文件中将他们添加到环境变量PATH中。更新挂载系统中的源，也就将所有人的版本统一更新了，即使当新员工第一登陆服务器时，也有一个能工作的arc。
* 另种通用做法：用户可以用 make install-arc 或 ant install-arc 之类的指令，写一个安装脚本，再使用现在已有的构建脚本来调用它。
* 如果你觉得给出的解决方案很糟糕，并让您感到头痛，请向我们反馈 (如何反馈，可以见 [Give Feedback! Get Support!](http://www.phabricator.com/docs/phabricator/article/Give_Feedback!_Get_Support!.html))。虽然这些问题很有难度，但我们将会改善它们。

### 安装Tab Completion

当你使用 bash脚本语言， 你可以通过在.bashrc、.profile或者有关环境变量的文件安装tab completion:

source /path/to/arcanist/resources/shell/bash-completion

### 配置Arcanist

很多Arcanist指令在安装的时候就已经完成了配置。 这些配置可以是从下面的源中读出：

1. 想要完成自定义配置可以关注 [ArcanistWorkingCopyIdentity::setRuntimeConfig()](http://www.phabricator.com/docs/arcanist/class/ArcanistWorkingCopyIdentity.html#method_setRuntimeConfig)这个API。
2. 用户可以从VCS文件夹的位置拷贝arc/config文件(JSON格式)到指定位置（例如：git/arc/config）。文件的指定位置可以通过arc set-config --local 来改变配置文件的位置。
3. 代码库的配置在 .arcconfig(JSON格式) 文件中。
4. 用户可以在通过配置关键字指定配置在~/.arrcr(JSON 格式)文件中。这个文件可以通过运行 arc set-config --global指令进行修改。
5. 机器设置可以通过/etc/arcconfig (JSON)文件进行设配置。在Windows下, 这个文件的路径是 'C:\ProgramData\Phabricator\Arcanist\config`.

*The first place where the setting is defined wins.*

已存在的配置可以通过 arc set-config --show指令，打印出来。

### 下一步操作

可继续如下步骤:

* 通过 arc建立一个新工程, 可见 [Arcanist User Guide: Configuring a New Project](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Configuring_a_New_Project.html)
* 学习如何使用arc发送用于回顾的修订信息 [Arcanist User Guide: arc diff](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_arc_diff.html).

高级主题也开放了。以下是有关arc包装脚本更为细化，并有更多技术详情的教程。如果只是为了让他们工作起来，就没有必要去看这些教程。

* [Arcanist User Guide: Commit Ranges](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Commit_Ranges.html)
* [Arcanist User Guide: Lint](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Lint.html)
* [Arcanist User Guide: Customizing Existing Linters](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Customizing_Existing_Linters.html)
* [Arcanist User Guide: Customizing Lint, Unit Tests and Workflows](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Customizing_Lint,_Unit_Tests_and_Workflows.html)
* [Arcanist User Guide: Code Coverage](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Code_Coverage.html)
* [Arcanist User Guide: Repository Hooks](http://www.phabricator.com/docs/phabricator/article/Arcanist_User_Guide_Repository_Hooks.html)

## Arcanist 使用教程: arc diff

介绍如何使用 arc diff, 向Differential发送修改来进行回查。

本文前提假设你的arc已经安装，并成功运行; 如果没有，参考 [Arcanist User Guide](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide&type=article&jump=1) 章节来得到更多的帮助。

在运行 arc diff之前，你需要创建一个.arcconfig 文件。

### 概述

arc有很多指令可以直接使用Phabricator应用接口，arc发送修改到Differential来进行回查。(有关Differential更多的信息, 可参考 [Differential User Guide](https://secure.phabricator.com/diviner/find/?name=Differential_User_Guide&type=article&jump=1))。如果你不大熟悉Differential的操作，你可以先去了解一下代码审查的大体流程。

你需要运行arc diff将修改进行发送。本节将阐述如何使用arc diff。还有，介绍如何在不同的版本控制系统中个，运行整个审核流程操作。

### Subversion

在Subversion中, arc diff 会发送 **副本中的未提交修改** 来进行回查。

在SVN中创建一个修订：

$ nano source\_code.c # Make changes.

$ arc diff

这里将会提示有关版本修订的信息。为了要更新现有的版本，你只需要一直做相同的事就行：

$ nano source\_code.c # Make more changes.

$ arc diff

这时，arc将提示你版本更新信息。 当你的修订被接收，你就可以使用以下指令进行提交：

$ arc commit

### Git

在Git中, arc diff 发送 **范围内的所有提交** 来进行回查。默认情况的范围为：

`git merge-base origin/master HEAD`..HEAD

这里的意思是“当前分支的所有的提交，还没有进行上传”。所以，为了在Git中 **创建一个修订** , 执行下面命令即可:

$ nano source\_code.c # Make changes.

$ git commit -a # Commit changes.

$ arc diff # Creates a new revision out of ALL unpushed commits on

# this branch.

 git commit 指令是可选指令。如果是未提交的修改，Arcanist将会提示你，为这些修改创建一个提交版本。这里选择不提交，就可以对对应的回查者提出任务；否则，会提示“选择了错误的提交范围”。

当分支上的所有修改都已提交，你就可以发送你感兴趣的修改进行回查。

你可以使用以下命令来对现有范围进行替换：

$ arc diff <commit>

这意味着使用以下范围：

`git merge-base <commit> HEAD`..HEAD

不过，这是相对高级的功能。如果你没有创建新的分支，默认值通常是正确的。

为了更新版本，你只需要重复执行下面的指令即可：

$ nano source\_code.c # Make more changes.

$ git commit -a # Commit them.

$ arc diff # This prompts you to update revision information.

 git commit 是可选指令。如果是未提交的修改，Arcanist将会提示你，为这些修改创建一个提交版本。

当你的修订被接收，你就可以使用以下指令进行提交：

$ arc land <branch> # Merges <branch> into master and pushes.

arc land 会对你的工作流做出一些猜测，这些猜测可能不是正确的。在使用之前，可以看看相关文档。你应该去看看arc amend，它可能更合适于你的工作流。

### Mercurial

在 Mercurial中, arc diff **范围内的所有提交** 来进行回查。默认情况下，这个范围是在第一个非外向父版本与当前目录的状态变化之间。就是说“自从上次合并后，产生的每个变化”。它将会包括副本中任何为提交的修改，并进行提示。

要在Mercurial中创建一个修订，需要运行：

$ nano source\_code.c # Make changes.

$ hg commit # Commit changes.

$ arc diff # Creates a new revision out of ALL outgoing commits

# on this branch since the last merge.

 hg commit 是可选指令。如果是未提交的修改，Arcanist将会提示你，为这些修改创建一个提交版本。这里选择不提交，就可以对对应的回查者提出任务；否则，会提示“选择了错误的提交范围”。

当分支上的所有修改都已提交，你就可以发送你感兴趣的修改进行回查。

你可以使用以下命令来对现有范围进行替换：

$ arc diff <commit>

这就以为你可以去提交对应范围内的修改。不过，这是相对高级的功能，通常情况下默认值是正确的。

更新一个修订，只需要重复以下指令即可：

$ nano source\_code.c # Make changes.

$ hg commit # Commit changes.

$ arc diff # This prompts you to update revision information.

 hg commit 是可选指令。如果是未提交的修改，Arcanist将会提示你，为这些修改创建一个提交版本（或在支持的情况下，对上一个提交直接进行修改）。

当你的修订通过审查，就可以进行上传(arc在Mercurial上的集成并没有上传的功能，因为它不能强制合并和不能生成一些在代码合并时有用的东西)。

### 上传和关闭修订

在修改被接收以后，一般情况下需要上传和关闭对应修订。arc有一些或许对你有帮助的工作流：

* 舍弃或合并入一个修改中的新特性到主分支。（如果适用的话）
* 通过Differential的提示信息来对提交信息进行格式化。
* 自动关闭对应修订。

你不需要使用这些工作流程，只需要运行 git push, hg push 或 svn commit ，然后进行手动关闭对应修订。但是，这些工作流在开发策略中是通用且很方便的，也能给你库中各个版本更好的提交信息。arc支持的工作流有：

* arc land:如果工作在Git的开发分支。当你分支要向主分支上合并，会提供详细的提交信息，上传到主分支，然后会删除你原来的分支。
* arc amend: 如果你不想用arc land指令。 通过详细的提交信息来修正 HEAD。
* arc commit: 在Subversion副本下工作。等同于运行 svn commit ，获取提交的详细信息。
* arc close-revision: 可以工作在任意副本, 通过命令行的方式关闭一个修订。

你可以使用 arc help <command> 获取详细的帮助信息。Differential 将会做出一个有关下已允许的修订的下一步计划，但是其预测可能不是你工作流程中的下一步工作。

Phabricator 将会自动关闭有关修订，当修改被上传到代码库，并且被Diffusion进行跟踪。他会根据提交的哈希值，和Differentail修订的信息来判断需要关闭的修订版本。 (你可以通过在代码库中设置“Autoclose”来关闭这项功能)

如果你上传了一个未跟踪的代码库（或没有为这个库配置arc），arc land, arc amend和arc commit将会隐式调用arc close-revision。

### 一般性信息

一个特定版本控制系统独有的信息。

#### 只关注Diff

你可以通过--preview参数（或--only，但是这个参数会关闭别的功能）仅创建一个diff（好过一个修订）。你之后可以通过web图形界面创建或更新一个修订。

#### 其他Diff 源

你可以通过任意的补丁文件创建一个diff，使用--raw或通道作为标准输入。在大多数情况下，这将创建一个diff，而不是一个修订。你可以使用web图形界面，通过diff创建一个修订，或这更新一个已有修订。

#### 强行创建/更新

arc 使用副本信息（比如，路径，分支名，本地提交哈希和本地哈希树，版本控制系统的依赖）来指出，你是否想创建或更新一个修订。如果它的预测是不正确的，你可以强行创建或更新一个修订。

$ arc diff --create # Force "create".

$ arc diff --update <revision> # Force "update".

你可以通过arc which来指明你工作的工作副本。

## Arcanist 使用教程: Code Coverage

介绍 code coverage在 Arcanist 和 Phabricator中的特性。

这个配置教程可以帮助你安装/应用一些code coverage的高级功能。如果你是刚开始使用Phabricator，那么你暂时还不用看这节的内容；你应该从这里[Arcanist User Guide](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide&type=article&jump=1)开始。

在你开始配置coverage的功能前，你需要在工程里面集成单元测试。

相关的内容可以参考 [Arcanist User Guide: Configuring a New Project](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Configuring_a_New_Project&type=article&jump=1) 和 [Arcanist User Guide: Customizing Lint, Unit Tests and Workflows](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Customizing_Lint%2C_Unit_Tests_and_Workflows&type=article&jump=1)。

### 使用Coverage特性

如果你的工程已经使用coverage集成了单元测试（可以参考下面的内容来进行安装/配置），你可以使用 arc 指令来查看覆盖性报告。

例如:

arc unit --detailed-coverage src/some/file.php

这条指令将会运行有关src/some/file.php的测试，并且给出一个详细的覆盖性报告。详细到何种地步，这取决与你如何配置测试引擎

如果测试引擎默认使用coverage，那么coverage将会把报告发送到Differential，并且在用户查看时显示在正确的位置。

### 启动Coverage (libphutil，Arcanist和Phabricator)

在安装完Xdebug后，libphutil, Arcanist 和 Phabricator 都支持coverage功能。

http://xdebug.org/

在这个页面的指导下，应该能正确的安装Xdebug；在安装完毕后，将会自动激活并启动。

### 构建coverage支持

为了在一个测试引擎里添加coverage支持,仅仅需要调用setCoverage()就能构建[ArcanistUnitTestResult](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistUnitTestResult&type=class&jump=1)对象。提供文件名映射(相对于工作空间的路径)，Coverage报告的字符串，就像下面这样：

NNNNNCCCNNNNNNNNCCCCCCNNNUUUNNNNN

文件中的每一行都是由一个字符表示。有效的字符有：

* N Not executable(不可执行)。这代表注释或者空格，这行在测试coverage时会被忽略。
* C Covered(已覆盖)。这行已经在测试coverage中覆盖。
* U Uncovered(未覆盖)。 这行可以执行，但是没有在测试测试coverage中覆盖。
* X Unreachable(无法访问)。 如果你的coverage分析可以检测到无法访问的代码，它将会报告中显示字符‘X’。

这中格式的目的是为了尽可能的简单。有效的coverage结果，类似如下内容：

array(

'src/example.php' => 'NNCNNNCNUNNNUNUNUNUNUNC',

'src/other.php' => 'NNUNNNUNCNNNUNUNCNCNCNU',

);

你可能想要使用过滤器过滤coverage的信息，并传送给单元测试引擎。这里可参考 [ArcanistPhutilTestCase](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistPhutilTestCase&type=class&jump=1) 和[PhutilUnitTestEngine](https://secure.phabricator.com/diviner/find/?book=arcanist&name=PhutilUnitTestEngine&type=class&jump=1) ，作为在PHP中使用Xdebug集成coverage的一个例子。

## Arcanist使用教程: Commit Ranges

下面将会说明如何让commit ranges 在 Arcanist上正常工作。

这属于进阶教程，如果你刚开始看，你可以不看本节，而是去关注[Arcanist User Guide](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide&type=article&jump=1)。

### 概述

在Subversion中，arc命令始终在未提交修改的副本上进行操作。如果你使用的是Subversion，本节内容对你来说无关紧要。

在使用 Git 和 Mercurial,有很多arc命令(特别是 arc diff) 可在一定的提交范围内进行操作。这个范围的起始位置可由你自行指定，通过工作副本状态来决定结束位置。

当结束位置已固定(工作副本状态)，你只需要指定开始位置即可。这个叫做“基提交”(base commit")。你可以参照如下的方式运行命令：

$ arc diff HEAD^ # git: just the most recent commit

$ arc diff .^ # hg: just the most recent commit

你同样可以配置arc，代替一些基提交命令；或者，声明基提交正在使用一个(可能复杂的)规则集。

NOTE: 本节介绍了一种新的机制，确定基提交。它可以接受改变。当然，目前还有一些其他的机制(将在其他章节进行介绍)。不过，由于这种机制比较成熟，它将代替其他机制，或者成为系统默认使用的机制。

### 配置基提交规则

基提交规则配置起来可要比你想象得复杂。这是因为在Git和Mercurial上，不同的人会使用不同工作流程，并且在使用arc进行提交和过滤时，都有不同的期待。更糟的是，默认的提交规则在很多时候与我们的期望相去甚远。

在过去，我们也尝试采用启发式算法，以及一些简单的办法来过滤基提交。但是相较使用者们对版本控制的多样化想法，我们的方式显得过于单一。

规则集的配置的确比较复杂，但是我们认为它足够强大，并能完成你预期的功能。

为了确定基提交的合法性，arc会逐一去匹配规则集中的规则，直到匹配到为止（这里我们认为规则集中的规则才是合法的）。一个匹配得规则将会决定基提交的提交域。

一条规则的形式，如下所示：

arc:upstream

只有一条规则意味着，如果提交行不是就是对工作副本的合法操作，就是一次非法的提交操作。例如：当只有一条规则“arc:upstream”时，要匹配成功，需要当前追踪的是Git库中的upstream分支; 如果匹配失败，就意味着当前追踪的分支不是Git库中的upstrea分支，或者当前工作副本，并不是一个有效的Git工作副本。

当有多条规则时，一条规则匹配失败，系统将会自动对下一条规则进行匹配。有些规则可能永远不会被匹配到，但是会产生积极的作用。下面来介绍一下这种情况：

规则集中，使用逗号分隔不同的规则：

arc:upstream, arc:prompt

arc 读取五个规则集：

1. args, 通过 --base <ruleset> 在命令行来指定。是第一个处理的规则集。
2. local, 通过 arc set-config --local base <ruleset>在命令行来指定。这个规则集是根据本地工作副本的设置产生，是第二个处理的规则集。
3. project, 通过 .arcconfig中的“base”值来指定。 这个规则集结合了工程的配置，是第三个处理的规则集。
4. global, 通过 arc set-config base <ruleset>进行指定。这个规则集是面对当前用户的全局规则集，是第四个处理的规则集。
5. system, 通过一个全系统级别的文件来指定。这个规则集是面对系统中所有用户的全局规则集，是最后处理得规则集。

规则集中得规则会被逐一匹配，直到匹配成功。

在下面列出一些合法的规则：（其中“\*”代表任意的字符串）

* git:\* 当规则存在时，需要使用特定的指令进行提交。
* git:merge-base(\*) 当规则存在时，使用基于HEAD的合并，并且使用特定的指令进行提交。
* git:branch-unique(\*) 尝试去选择在一个分支上的独特修订（这里，修订在分支节点和HEAD之间）。这条规则有些复杂，并且有局限性，可参考下面的“规则详解”。
* hg:\* 当规则存在时，需要使用特定的指令进行提交。
* hg:gca(\*) 当规则存在时，与当前代码拥有共同的原始节点，并且使用特定的指令进行提交。
* arc:upstream 当规则存在时，使用合并方式处理当前分支的upstream和HEAD。（仅限GIt）
* arc:outgoing 使用工作副本中，最近产生的非外向原始节点。（仅限hg）
* arc:exec(\*) 执行特定指令。指令应该能决定使用的基础版本，以及打印标准输出，然后带返回值退出。如果指令退出并返回其他值，则匹配失败。指令将会把工作副本的根目录作为当前工作目录，在这里执行。
* arc:bookmark  使用最近产生的非外向原始节点上的祖副本，或者使用最近的书签。这个规则比较复杂，也有局限性，可参考下面得“规则详解”。
* arc:amended 如果库被修改为包含“Differential Revision：”域时，使用最近一次提交（Git中是HEAD，Mercurial中是'.'）。否则，匹配失败。
* arc:prompt 提示用户提供一个版本提交。
* arc:empty 使用空状态（虽然库完全是空的，但是在原始节点的工作拷贝中还是会包含所有提交的信息）。
* arc:this 使用当前提交版本。这就意味着使用'.'（Mercurial）和HEAD（Git）。

以下规则都是可用与修改处理规则集的顺序：

* arc:args, arc:local, arc:project, arc:global, arc:system 停止处理正在处理的规则集，开始处理指定的规则集。当前正在处理的规则集，将在指定规则集处理完毕后，继续处理。
* arc:yield 停止处理当前正在处理的规则集，开始处理下一个规则集。当前正在处理的规则集，将在之后的规则集处理完成之后处理;当当前处理的规则集在指定处理集的顺序之后，则继续按顺序处理。
* arc:halt 停止处理所有的规则集。这个操作将会是你的指令都执行失败，但是可以避免后面的规则进行运行/匹配。

此外，还有一些用来测试和调试的规则集：

* arc:verbose 开启规则处理的冗余记录。
* arc:skip 这条规则没有作用。
* literal:\* 使用指定的提交指令。在产生规则时，几乎可以肯定是错的。

### 范例

对比（Diff against） origin/master （如果存在的话）;并且当不存在时，进行提示：

git:merge-base(origin/master), arc:prompt

对比upstream 分支（如果存在的话）;或者使用最后一次提交得版本（如果不存在）:

arc:upstream, git:HEAD^

作为用户，可忽略工程规则，并且总是使用自己的规则：

(local) arc:global, arc:halt

作为项目的维护者，同意用户的规则在工程规则之上：

(project) arc:yield, <defaults>

调试你的规则：

$ arc diff --base arc:verbose

了解规则处理流程：

$ arc which

$ arc which --base '<ruleset>'

$ arc which --base 'arc:verbose, <ruleset>'

### 规则详解

有些规则有复杂的操作，这里会给出比较详细描述。 这些规则的高级特性是针对专家级用户设计，希望通过规则的设置对工作流程进行优化，并减少一些输入。这里，你不需要了解这些规则是怎样使用arc的（你可以明确的指定一个基提交）。

#### git:branch-unique(\*)

这条规则仅在Git下可用。

这条规则试图在当前分支找到独特的提交。如果你在一个分支上进行每个特性的开发，这个功能就会很有用，通过修改提交版本，进行更新（代替追踪提交操作）;并且，可以通过注册的形式合并修改（代替合并操作）。

规则需要首先确定指定合并基点和HEAD（如果存在的话），再进行操作。如果不存在，规则匹配失败。如果有一个提交存在，规则将会统计HEAD里面包含了多少个分支。然后在到合并基点端，统计在每一个提交中包含有多少个分支。当发现一个提交中的分支数多余HEAD中的分支数，或者当统计走到合并基点的提交是，统计将会结束。

对于下面的代码树，这条规则工作正常：

| \* Commit B1, on branch "subfeature" (HEAD)

| /

| \* Commit A1, on branch "feature"

|/

\* Commit M1, on branch "master"

|

这棵树代表使用feature分支来开发一个“feature”节点，然后创建一个子分支来进行依赖于feature的节点（subfeature）。

通常，如果你在subfeature(同 HEAD 一起，在 B1)分支上运行 arc diff , 并存在一条规则 arc:merge-base(master) 将会选择 M1 作为基提交，这样的话  A1 在将错误的包含提交域中。

对于这样得树, git:branch-unique(master) 规则将不会把 A1 选作为基提交(因为它是在 B1 和 M1 之间的第一次提交，比 B1 出现在更多的分支中-- B1 只出现在"subfeature”，但是 A1 出现在"subfeature"和"feature"分支上)，所以仅在提交域中包含 B1 点。

当这条规则在“feature”分支上执行，同样能得到正确的结果。

不过，这条规则在某些情况下也会选择错误的提交域。

例如，在下面的这棵树上，他的选择将会出错：

|

| \* Commit A2, on branch "feature" (HEAD)

| |

| | \* Commit B1, on branch "subfeature"

| |/

| \* Commit A1, on branch "feature"

|/

\* Commit M1, on branch "master"

|

这棵树在"feature"分支，在 A1上存在一个另一个提交，A2节点 。

这里，当我们在“feature”分支上的A2（和HEAD一起）点运行 arc diff ，这条规则将会错误的选择A2，因为A2只出现在一个分支上，而A1出现在两个分支上（"feature", "subfeature"）。

你可以通过修改A1的修订避免这种错误（而不是进行新的提交），或者以“subfeature”分支为基点运行arc diff。

下面得树也会让这条规则选择错误得提交域：

|

| \* Commit A1', on branch "feature", created by amending A1

| |

| | \* Commit B1, on branch "subfeature" (HEAD)

| |/

| o Commit A1, no longer on "feature" but still on "subfeature"

|/

\* Commit M1, on branch "master"

|

这棵树表明A1的修改不是基于“subfeature”分支，所以A1将不会在“feature”，但是将会在“subfeature”（这里的判断是根据HEAD做出的）。在这种情况下，在“subfeature”分支上运行arc diff将会错误得选择到B1和A1，因为他们现在都包含有同样数量的分支。

你可以通过在子分支上运行arc diff，或者在 git:branch-unique(\*)之前执行 arc:amended规则，来避免这种错误的产生。

#### arc:bookmark

这条规则仅适用于Mercurial。

这条规则去寻找外向改动，当遇到书签时停止。如果你对每个特性使用一个书签，这个规则会有很大的用处。

这条规则的操作类似 arc:outgoing, 但是会扫描在当前提交'.'和所选基提交之间的节点。当遇到书签的时候，停止运行。例如，有树如下所示：

|

| \* C4 (outgoing, bookmark: stripes)

| |

| \* C3 (outgoing, bookmark: zebra)

| |

| \* C2 (outgoing, no bookmark)

|/

\* C1 (pushed, no bookmark)

|

当从C4开始运行这条规则，规则将会选择C4，在C3处停止，因为它有一个不同的书签。当在C3开始运行是， 它将会选择C2和C3。

但是，这个规则将会在一些情况下选择到错误的提交域（例如，当“zebra”书签被移除，在C4运行规则时，规则将不会在C3处停下，并且会选择C2, C3和C4）

#### arc:exec(\*)

这条规则运行一些外部脚本或者是shell命令。它的目的是为了高级用户，他们想用一些无法用其他规则表达得行为来进行操作。

为了使用这个规则， 需要提供一些脚本或者shell命令。例如：

arc:exec(git merge-base origin/master HEAD)

arc:exec(/path/to/some/script.sh)

命令可以以工作副本为工作目录，进行运行，以及传递参数。为了与之相匹配，它应该通过标准输出打印一个基提交的名字，然后退出，并返回0。当匹配失败，它将会退出，并返回非0值。

### 下一步

可进入如下章节：

* 了解有关arc diff 更为详细的信息，可参见 [Arcanist User Guide: arc diff](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_arc_diff&type=article&jump=1)

## Arcanist 使用教程: 配置一个新工程

描述如何通过.arcconfig文件配置Arcanist 工程。

### 概述

在大多数例子中，你需要使用到arc，而不需要特别的去为了使用arc而配置项目。如果你想要调整arc的行为，你可以创建一个.arcconfig文件在你的工程中，来提供对项目的特定配置。

### .arcconfig 基础篇

.arcconfig 文件是一个JSON文件，你可以在你的工程根目录下找到它。

Arcanist 使用 .arcconfig 文件来自定义一系列关于它的行为。第一件事，你可能想要配置已安装 Phabricator系统得URI。简单，合法的写法如下所示：

**.arcconfig**

{

"phabricator.uri" : "https://phabricator.example.com/"

}

更为详细的可用选项，在接下来的章节中介绍。

NOTE: 你必须提交你的.arcconfig文件！它包含有工程配置，而不包含用户配置。

### .arcconfig 进阶篇

公共选项有：

* phabricator.uri: Phabricator 的URI地址，当arc在工程中运行时，它将会连接这个地址。这个选项在之前的版本中叫做 conduit-uri 。
* repository.callsign: 在Diffusion中代码库的签名。通常，arc可以自动检测到，但当它没有指定时，你可以明确的给以指定。使用arc which来了解检测过程。
* history.immutable: 在工作副本中，配置arc使用不重写历史记录的工作流。默认情况下，在一些Git工作流中，arc将会对一些未公开得历史记录进行重写（修改提交信息，抹去合并信息）。他们的区别将会在下面进行介绍。

其他选项有：

* load: 列出依赖的Phutil库，并在启动之时加载他们。下面的内容中会介绍到路径的解决办法，或者参考[libphutil Libraries User Guide](https://secure.phabricator.com/diviner/find/?name=libphutil_Libraries_User_Guide&type=article&jump=1) 对liphutil库的通用介绍。
* project.name: 名为“Arcanist Project”是一个工作副本（对于Git，Mercurial）或者目录（SVN）。以前，这是个必须添加得选项，但是arc现在可以独立与Git和Mercurial执行，所以就不是必须要添加的选项了。这个选项在之前叫做project\_id。
* https.cabundle: 指定一个备用证书集的路径，以便在使用HTTPS连接时使用。
* lint.engine:  [ArcanistLintEngine](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistLintEngine&type=class&jump=1)的一个子类名，它可以在该项目中使用lint规则。有关lint的内容，可见 [Arcanist User Guide: Lint](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Lint&type=article&jump=1).
* unit.engine:  [ArcanistBaseUnitTestEngine](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistBaseUnitTestEngine&type=class&jump=1),的一个子类名，它会在该项目中使用单元测规则。有关单元测试的内容，可见 [Arcanist User Guide: Customizing Lint, Unit Tests and Workflows](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Customizing_Lint%2C_Unit_Tests_and_Workflows&type=article&jump=1).

还有些选项是支持的，不过不推荐使用：

* http.basicauth.user: 指定一个HTTP用户名，可以用来连接Phabricator。
* http.basicauth.pass: 指定一个HTTP密码，可以用来连接 Phabricator。
* https.blindly-trust-domains: 一个盲目的HTTPS信任域列表，即使其中有些证书无效。这是通过蛮力解决证书有效性的问题，这里我们很不推荐。这里推荐使用有效证书进行验证。

要列出所有得选项，可以用arc set-config –show。虽然全部选项都能在.arcconfig文件中进行设置，但是有些选项（例如 editor）通常不会进行设置，因为对于每一个用户来说，他们的习惯都是不一样的。

### 历史记录的可变性

Arcanist的工作流在两大模式下运行：可变 和 不变，两种状态。在“可变”模式得中，arc命令可能会重写工作副本的历史记录;在“不变”模式中，arc就不能对历史记录进行复写。

你可以通过配置history.immutable为true或false，来控制历史记录的可变性。在默认情况下，Git下是可变的，Mercurial下是不可变的。下面章节的内容将会来介绍，可变性是如何影响工作流的。

#### 历史记录可变性: Git

当工作流是历史可变时，你可以复写本地历史记录。当你在分支上进行特性开发时，可以使用git commit--amend, git rebase -i或git merge --squash来抹去或修改之前的上传信息。

当工作流是历史不可变是，你不能复写本地历史记录。当你在分支上进行特性开发，并能提交他们，且不能抹去提交信息。这时git commit –amend或git rebase -i 就不能使用了。

实际上，在这里你会看到与你的设置有差异的地方：

Mutable

* arc diff  将会提示你去HEAD修改lint的改动。
* arc diff  在创建一个新的修订后，会修改HEAD内的提交信息。
* arc land  将 --squash 设置为默认策略。
* arc amend  将修改HEAD的提交信息，以及相关信息，或指定的Differential修订。

Immutable

* arc diff  当它使得lint有所改动，该行为将会中止。
* arc diff  在创建一个新的修订后，不会对HEAD内的提交信息进行修改。
* arc land 将 --merge  设置为默认策略。
* arc amend 将会带着一段错误信息，退出。

#### 历史记录可变性: Mercurial

在2.2版之前，Mercurial 还没有历史记录修改命令，所以这里的设置对它是没有影响的。在2.2版之后，就可以对历史记录进行修改。

使得历史记录可以修改的意义如下：

Mutable (2.2版，以及更新的版本)

* arc diff 在创建一个修订后，将修改当前提交信息。
* arc amend 将修改当前的提交信息，以及相关信息，或指定得Differential修订。

Immutable (2.2版之前版本)

* arc diff 在创建一个新的修订后，不会对当前的提交信息进行修改。
* arc amend 将会带着一段错误信息，退出。

### 如何定位库文件

如果你需要加载外部库，比如“examplelib”，并使用相对路径，可以参考下面的例子：

{

...

"load": [

"examplelib/src"

],

...

}

...arc 会在以下路径中进行查找库文件：

* path/to/root/examplelib/src/  首先，arc 会在工程的根目录下（和.arcconfig同级）寻找库文件。这使得它很容易在特定项目中找到库文件。
* path/to/root/../examplelib/src/ 其次，arc会在项目根目录的同级目录中查找。当你拥有很多代码库的时候，把所有的arc代码放在一个库中，并且只需要在同一个目录中查找就可以了。
* php/include/path/examplelib/src 最终，arc后退到PHP，它将会去下级子目录（在php.ini中include\_path设定）去寻找。这会允许你在全局位置下去安装库文件。

你可以修改，并提供一个绝对路径，比如 /var/arc/examplelib/src，但是之后每个人都需要在这个路径下安装库文件。

NOTE: 指定路径到目录，需要包含 \_\_phutil\_library\_init\_\_.php。例如，如果你的init文件在 examplelib/src/\_\_phutil\_library\_init\_\_.php, 需要指定 examplelib/src, 而不是仅仅是 examplelib/。

这里的意图是：

* 将工程代码放在指定文档下，例如 support/arc/src/。
* 将共享代码（比如，有强制代码规范，或者自动触发单元测试等）放在另一个代码库中，再去检查其他的代码库。
* 或者将所有的东西都放在一个标准的地址中，然后将这个地址添加到include path中。

### 无 .arcconfig文件运行

不用配置.arcconfig文件，arc指令依旧可以在Git，subversion，或Mercurial的工作副本中工作。不过其中有些特性是必须要在.arcconfig文件中进行配置的，否则它们可能会不工作。

在无.arcconfig文件的情况下：

* 你可以通过arc set-config default <uri> (命令行)的方式来设置Phabricator的默认访问地址，或者使用—confuit-uri指定一个明确的访问地址。
* 你将不能通过arc命令去运行一个代码分析器（linter），除非你使用--engine选项明确指定。
* 即使通过--engine选项进行指定，你依旧不能使用自定义代码分析器。
* 你将不能通过arc命令去运行单元测试，除非你使用--engine选项明确指定。
* 你将不能通过arc diff命令去触发代码分析器和集成单元（测试）。
* 你将不能得到历史记录可变的Git工作副本。
* 你将不能指定一个代码库的编码格式。如果你没有通过--encoding 选项进行设定，那么将会使用默认的编码格式UTF-8。
* 你将不能为arc添加插件去修改现存的工作流程或者添加新的工作流程。
* 你将不能添加依赖库，除非你使用--load-phutil-library选项进行明确的指定。
* 符号集成引索，将不会工作。这个功能，允许用户在Differetial中通过鼠标点击函数名或类名称，直接跳转到其定义。
* arc patch 指令将无法检测到你应用于错误工程上的修改。
* 在subversion中，arc将不能确定项目的基准根，并且会假设基准根就是工作目录（subversion1.7或更早的版本），或者去检验基准根（subversion1.7之后的版本）。这就意味着，同一路径下的文件的修改将不会标记在同一个位置，并且会产生多个不同的路径。这将会给检查者带来困扰。有时还会阻碍补丁的使用，有可能一个文件的补丁被不同路径的另一文件使用。
* 在subversion中，arc将无法猜测你要更新现存的哪一个版本；你必须使用--update或 --preview进行明确的指定，并且通过web借口附上diffs信息。

### 下一步

可以继续如下章节:

* 返回到 [Arcanist User Guide](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide&type=article&jump=1).

## Arcanist 使用教程: 自定义已存代码分析器

介绍应该如何自定义已存的代码分析器。

本节将解释如何细化代码分析器的行为。想要在一开始就配置代码分析器，可以参照 [Arcanist User Guide: Configuring a New Project](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Configuring_a_New_Project&type=article&jump=1) 和 [Arcanist User Guide: Lint](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Lint&type=article&jump=1)章节。

### 概述

Arcanist中可以使用的代码分析器有很多，你可能会在全部或部分项目中用到其中一（或几）个。本节将会介绍，如何自定义已存代码分析器作为新的引擎。

首先，你需要设置一个引擎，可参考如下章节 [Arcanist User Guide: Lint](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Lint&type=article&jump=1) ，并且可能需要参考 [Arcanist User Guide: Customizing Lint, Unit Tests and Workflows](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Customizing_Lint%2C_Unit_Tests_and_Workflows&type=article&jump=1)。然后，在跟着本节接下来的内容来对代码分析器进行自定义设置。

### 普遍准则

你需要通过配置或调节，来自定义代码分析器，而不是扩展他们——他们当中有些实现并不一定稳定。如果一个代码分析器的配置选项，对于你得需求来说，不够灵活，提交一个可以使它配置提高的补丁，让其更完善。

### 修改严宽规则

默认情况下，大多数代码分析器将会把分析信息当做错误处理。不过，你可能想对这些信息降低严宽程度（例如，把“视为错误”降低到“视为警告”）。要想降低严宽长程度，需要调用 setCustomSeverityMap()函数：

$linter = new ArcanistTextLinter();

// Change "missing newline at end of file" message from error to warning.

$linter->setCustomSeverityMap(

array(

ArcanistTextLinter::LINT\_EOF\_NEWLINE

=> ArcanistLintSeverity::SEVERITY\_WARNING,

));

可见 [ArcanistLintSeverity](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistLintSeverity&type=class&jump=1) 章节，里面列举了很多控制严宽的常数。

### 屏蔽严宽规则

为了完全屏蔽严宽规则，可以对严宽变量 SEVERITY\_DISABLED进行设置：

$linter = new ArcanistTextLinter();

// Disable "Tab Literal" message.

$linter->setCustomSeverityMap(

array(

ArcanistTextLinter::LINT\_TAB\_LITERAL

=> ArcanistLintSeverity::SEVERITY\_DISABLED,

));

### 运行多个规则集

如果你想要使用同一个代码分析器，对不同格式的文件进行分析。就需要根据文件的格式进行二次实例化，并且配置适当的实例。

例如，以下规则将会强制不同的语言使用不同的最大列宽值：

$linters = array();

// Warn on JS/CSS lines longer than 80 columns.

$linters['TextLinter80Col'] = id(new ArcanistTextLinter())

->setPaths(preg\_grep('/\.(js|css)$/', $paths));

// Warn on Java lines longer than 120 columns.

$linters['TextLinter120Col'] = id(new ArcanistTextLinter())

->setMaxLineLength(120)

->setPaths(preg\_grep('/\.(java)$/', $paths));

// ...

return $linters;

### 自定义指定代码分析器

一些代码分析器需要特别定义或者配置。一些公共选项记列在这里，其中有咨询类记录的完整信息。

#### ArcanistTextLinter

* 使用 setMaxLineLength() 来修改80字符列宽警告（或者其他）。

#### ArcanistXHPASTLinter

* 在.arcconfig中使用 lint.xhpast.naminghook 来复写一些惯例。 更多详细信息，可参照[ArcanistXHPASTLintNamingHook](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistXHPASTLintNamingHook&type=class&jump=1) 章节, 并且在 [PhabricatorSymbolNameLinter](https://secure.phabricator.com/diviner/find/?name=PhabricatorSymbolNameLinter&type=class&jump=1) 章节有一个使用范例。
* 使用 getXHPASTTreeForPath() 在其他的代码分析器中复用AAST配置。

## Arcanist 使用教程: 自定义代码分析器, 单元测试和工作流

介绍如何构建新的类，去控制Arcanist的行为。

### 概述

Arcanist有一些基本的配置选项，在.arconfig中可用 (详细信息可见 [Arcanist User Guide: Configuring a New Project](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Configuring_a_New_Project&type=article&jump=1)章节), 但是它们也并不能处理所有事情。如果你想要从更深层次上自定义Arcanist，你需要构建新的类。

例如：

* 如果你想要配置代码分析器，或者添加新的代码分析器，你需要创建一个新类，作为[ArcanistLintEngine](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistLintEngine&type=class&jump=1)的扩展。
* 如果你想要在架构中集成一个单元测试，你需要创建一个新类，作为[ArcanistBaseUnitTestEngine](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistBaseUnitTestEngine&type=class&jump=1)的扩展。
* 如果你想修改工作流的动作，或者添加新的工作流，你需要创建一个新类，作为 [ArcanistConfiguration](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistConfiguration&type=class&jump=1)的扩展。

Arcanist是通过一种依赖注入方式工作的。例如，Arcanist 在默认情况下是不按照代码分析器的规则运行的，但是你可以通过在.arcconfig对lint.engineich变量设置对应得名字，作为[ArcanistLintEngine](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistLintEngine&type=class&jump=1)的扩展，使Arcanist使用对应的代码分析器规则。当Arcanist运行在你的项目中，Arcanist为了让代码分析器工作，它将会加载这个类，并且调用对应方法。不过在让代码分析器工作钱，你还得做三件事情：

* 确定写了特定类
* 为.arcconfig存在的类，添加库文件
* 在你的.arcconfig文件中对变量lint.engine, unit.engine,或arcanist\_configuration添加特定类的名字。

### 创建一个libphutil库

在创建完类了以后，你需要马上为现存的类创建一个库。可以在参考 [libphutil Libraries User Guide](https://secure.phabricator.com/diviner/find/?name=libphutil_Libraries_User_Guide&type=article&jump=1)章节之后，在通过对.arcconfig文件进行类似如下的修改，来加载指定库。在生成libphutil库之前，需要在<YOUR\_PATH>/libphutil/scripts下运行build\_xhpast.sh脚本，来生成xhpast库文件。在生成libphutil库时，要调用到xhpast库。

{

// ...

"load" : [

// ...

"/path/to/my/library", // Absolute path

"support/arcanist", // Relative path in this project

// ...

]

// ...

}

你可以为库指定一个绝对路径，或者一个基于工程根目录的相对路径。当你运行arc list –trace, 你将会看到一条有关加载你的库文件的信息，这时你的库就已经被加载成功了。

为了调试或测试，你需要带上 --load-phutil-library 选项运行Arcanist：

arc --load-phutil-library=/path/to/library <command>

你可以多次指定这个选项去加载若干个库。注意：如果你使用了这个选项，Arcanist将会忽略在.arcconfig中列出的库。

### 使用特定类

这个步骤很简单：仅仅需要编辑.arcconfig文件来指定你的类名称作为适配值即可。

{

// ...

"lint.engine" : "CustomArcanistLintEngine",

// ...

}

现在，当你在你的项目中运行Arcanist时，它将会在适当的时候调用你的自定义类。

为了代码分析器和单元测试，你也可以使用 --engine 选项来复写默认代码分析引擎：

arc lint --engine MyCustomArcanistLintEngine

这对于测试和调试来说，是很有用的。

### 下一步

* 浏览源码中有关代码分析引擎的例子，可参考 [ExampleLintEngine](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ExampleLintEngine&type=class&jump=1)。
* 学习如何复用已存代码分析器，可参考 [Arcanist User Guide: Customizing Existing Linters](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Customizing_Existing_Linters&type=article&jump=1)章节。

## Arcanist 使用教程: 代码分析器

本节将介绍：代码分析引擎，代码分析器，以及代码分析器配置。

本节阐述了代码分析是如何在一个 arc工程中工作的。如果你还没有创建一个arc工程，你需要参考一下 [Arcanist User Guide: Configuring a New Project](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Configuring_a_New_Project&type=article&jump=1)章节。

### 概述

"Lint" 指的是一类编程工具，这类工具可以通过分析源码，提出相关的错误或者警告。例如，语法错误，使用未定义变量，调用提出警告，调用废弃函数，间距和格式，范围滥用，在switch段中使用隐式声明，许可证标题缺失，使用危险的语言特性，等等其他的问题，都会使得代码分析器发出警告。

将代码分析集成进你的开发流水线中，有两大好处：

* 你可以监测和避免一大堆编程错误
* 通过代码分析中的自动定位机制和自动格式检查机制，写出更高质量的代码。

当arc已经集成到代码分析包中后，它将会使arc lint命令生效，并当arc diff发现修改时，执行代码分析。用户将会在检查代码时收到修复错误或者警告的提示，并且通过代码分析提出得问题在检查过程中是一直可见的，除非这个问题被修复。

这里为不同的语言提供了有很多的代码分析器，以及可用的静态分析工具集。Arcanist 自带了很多受欢迎的工具，并且你可以通过简单的步骤构建自定义的工具集。

### 可用的代码分析器

Arcanist 自带了一下的代码分析器：

* [JSHint](http://www.jshint.com/#_blank), 一个javascript代码分析器。可参考 [ArcanistJSHintLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistJSHintLinter&type=class&jump=1)。
* [PEP8](http://pypi.python.org/pypi/pep8#_blank), 一个Python代码分析器。可参考 [ArcanistPEP8Linter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistPEP8Linter&type=class&jump=1)。
* [Pyflakes](http://pypi.python.org/pypi/pyflakes#_blank), 另一个Python代码分析器。可参考 [ArcanistPyFlakesLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistPyFlakesLinter&type=class&jump=1).
* [Pylint](http://pypi.python.org/pypi/pylint#_blank), 再另一个Python代码分析器。 可参考 [ArcanistPyLintLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistPyLintLinter&type=class&jump=1).
* [PHP CodeSniffer](http://pear.php.net/package/PHP_CodeSniffer#_blank), 一个PHP代码分析器。可参考 [ArcanistPhpcsLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistPhpcsLinter&type=class&jump=1).

Arcanist也自带了一些大众化的分析器，并且可以通过配置，来分析代码分析程序的输出信息：

* [ArcanistScriptAndRegexLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistScriptAndRegexLinter&type=class&jump=1), 通过运行脚本的形式，使用正则表达式来分析输出信息。
* [ArcanistConduitLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistConduitLinter&type=class&jump=1), 可以通过Conduit 调用一个代码分析器，并且允许你构建“客户/主机”代码分析器。

此外, Arcanist还有些通用的代码分析器:

* [ArcanistTextLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistTextLinter&type=class&jump=1), 强制规定一些文本规则。比如，行尾空格限制，DOS的换行符限制，文件编码限制，列宽限制，终端换行符限制，以及制表符限制。
* [ArcanistSpellingLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistSpellingLinter&type=class&jump=1), 可以检测出拼写错误。
* [ArcanistFilenameLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistFilenameLinter&type=class&jump=1), 可以强制一般性规则来检测是否给使用了一个荒谬的文件名。
* [ArcanistLicenseLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistLicenseLinter&type=class&jump=1), 可以确定许可证标题已经被应用到了所有源文件中。
* [ArcanistNoLintLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistNoLintLinter&type=class&jump=1), 可以屏蔽代码分析器，使之失效。
* [ArcanistGeneratedLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistGeneratedLinter&type=class&jump=1), 可以禁止代码分析器生成文件。

Arcanist 还有特别的代码分析器：

* [ArcanistXHPASTLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistXHPASTLinter&type=class&jump=1), Phabricator默认启动PHP代码分析器。这个代码分析器很强大，但是规则有些僵硬（它强制使用phutil规则，并且没有其他规则集可配置）。
* [ArcanistPhutilLibraryLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistPhutilLibraryLinter&type=class&jump=1), 强制使用phutil库的布局规则。

三种方式可以让你添加新的代码分析器：

* 写新的构建，并且将其贡献给上游。
* 写新的构建，并且在Arcanist安装
* 使用一个通用构建，比如 [ArcanistScriptAndRegexLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistScriptAndRegexLinter&type=class&jump=1) 。再通过配置，就可以在集成环境中运行。

### 配置代码分析

Arcanist的代码分析继承涉及到两大组件：代码分析器 和 代码分析引擎。

代码分析器本身可以通过源码文件进行问题的检测。通常，代码分析器是一个外部脚本，Arcanist通过指定路径来运行，如同jshit或pep8.py。通过Arcanist对脚本产生一些信息进行分析，判断是否是结构性错误。使用外部脚本处理一段胶水代码分析器(比如 [ArcanistJSHintLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistJSHintLinter&type=class&jump=1) 或 [ArcanistPEP8Linter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistPEP8Linter&type=class&jump=1)) 并且解读他们的输出。

代码分析引擎配合代码分析器，使对应的文件与对应的代码分析器进行匹配。例如，你可能想要对所有的.js文件运行jshint，并且对所有的.py文件运行pep8.py。有时你可能不想对external/或者third-party/目录下的任何文件进行代码分析。有些文件你并不想包含入代码分析的范围内，或者对其使用特殊的规则。

要是代码分析工作，需要对arc进行配置，你需要指定一个代码分析引擎的名字，并且可能要提供一些额外的配置。先命名一个代码分析引擎，再在.arcconfig文件中的lint.engine变量设置这个已命名的引擎，作为 [ArcanistLintEngine](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistLintEngine&type=class&jump=1)类的扩展。有关.arcconfig的更多信息，可以参考 [Arcanist User Guide: Configuring a New Project](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Configuring_a_New_Project&type=article&jump=1)。

你也可以通过设置lint.engine启动一个默认代码分析引擎，使用arc set-config lint.engine让这个设置在全局（系统级别）生效，或者使用arc lint –engin <engine> 明确指定对应引擎。

可用的引擎有：

* [ComprehensiveLintEngine](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ComprehensiveLintEngine&type=class&jump=1), 运行一个套广泛适用的代码分析器集，去匹配不同类型的文件。由于过于宽泛，会对实际项目的使用进行限制。但是对于代码分析来说，这是一个好的起点。
* [ArcanistSingleLintEngine](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistSingleLintEngine&type=class&jump=1), 运行单一代码分析器，并无条件的应用与所有文件。这里可以使用一个胶水代码分析器，比如 [ArcanistScriptAndRegexLinter](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistScriptAndRegexLinter&type=class&jump=1) ，使得引擎逻辑加载在外部脚本中。
* 你可以写一个自定义的引擎。在很多项目中个，代码分析规则是相当特殊的选项。 如何写一个自定义的代码分析引擎，可参考 [Arcanist User Guide: Customizing Lint, Unit Tests and Workflows](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Customizing_Lint%2C_Unit_Tests_and_Workflows&type=article&jump=1)和 [ExampleLintEngine](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ExampleLintEngine&type=class&jump=1)章节。

### 使用代码分析提高代码回查效率

代码回查是一件十分有价值的事，这是对于代码修改有重大的意义。当吹毛求疵于代码风格，格式和命名时，这时的代码回查就没有什么意义了。

在没有代码分析器的前提下，收到一个充满代码风格问题的回查请求时，最好的回应可能就是立即拒绝提交，并且向代码作者明确的指出他的代码不符合你的代码规范，并且要求他在再次发送回查请求之前修复问题。这个过程对于双方都是很消极的经历;并且有时菜鸟级别的回查者，会忽略全局的回查，只是单独的指出每一个问题。

代码分析通过自动强制的风格和格式规则，可以很大程度上减少这种负面的经历（以及浪费大量时间来争论这些事）。Arcanist支持代码分析器不仅仅对这些问题提出警告，而且提供补丁为作者修复问题——在代码进入回查流程前。

好的代码分析器集成的意义在于，代码在这个机制中能让回查者在第一时间看到代码; 为代码作者提供与风格相关的明确规则;提供从文本方面，讨论修改对代码的整体影响，以及对一些好想法进行有效的测验。

对于风格讨论，它也提供简单的解决方案：

* 如果一条规则足够重要，那么它就会被重视。支持这必须将这条规则加入代码分析中，之后就可以自动检测或者修复。并且，大家已经在这个规则上达成共识。
* 如果这条规则不被大家所重视，也需要将他添加进入代码分析当中，这样其他人就不会再去抱怨这件事了。

以上的解决方案无论是否适用与你的团队，但是它将会把问题推向一个正确的解决方向。

### 代码分析的哲学

有关如何开发出一个高效的代码分析的一般想法，基于facebook对代码分析工具的构建与理解：

* 不要在代码分析器中写基于正则表达式的强制语言规则。适用一个实际的分析器或者基于AST的工具。在是否复杂的正则表达式中，你可以无法获得正确的域。这不是一个挑战，仅仅不要这么去就行。
* 误报会让人很懊恼，这种情况应该避免出现。你应该确保你的实现尽可能不发生误报，并且当误报的情况发生时，提供标记为“误报”的功能。如果代码分析器引发了30个无关的警告，那么这个代码分析器就可以扔掉了。
* 更进一步的是自动修复规则。大多数代码分析器都没有自动修复功能，但是Arcanist支持这一功能。这一个十分有用的功能，这使得代码分析器不仅仅是报告“格式错误：在函数调用中，逗号后面需要加一个空格”，并且会自动修复这个问题。

### 下一步

可以继续如下章节:

* 集成，自定义内置代码分析器和代码分析构建，可以参考 [Arcanist User Guide: Customizing Existing Linters](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Customizing_Existing_Linters&type=article&jump=1);
* 了解如何添加新的代码分析器和代码分析引擎，可以参考 [Arcanist User Guide: Customizing Lint, Unit Tests and Workflows](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Customizing_Lint%2C_Unit_Tests_and_Workflows&type=article&jump=1).

## Arcanist 使用教程: Mac OS X

### 阐述如何在Mac OS X下安装Arcanist。

#### 必要的组件

你需要安装：

* PHP, 是必须要安装的。
* Arcanist, 可参考 [Arcanist User Guide](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide&type=article&jump=1)章节。
* SVN, Git, 或 Mercurial 代码版本控制软件。

然后, 配置:

* 你的PATH环境变量需要包含arcanist/bin, php，还有你的VCS指令（svn，hg，git）。当你通过命令行输入php，arc，或者git的时候，它们需要能工作。
* 你的编辑器环境变量需要指向一些命令行编辑器，比如nano。

你可以编辑你的 ${HOME}/.profile 使你设置的环境变量永久生效。

现在你应该能够像使用一般指令一样的使用arc了。

### 下一步

可以继续如下章节：

* 回退到 [Arcanist User Guide](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide&type=article&jump=1).

## Arcanist 使用教程: 代码库钩子

阐述如何将Arcanist设置成你一个SVN预提交钩子。

NOTE：配置过程可能不会太顺利，其中需要一些技巧。

### 安装Arcanist SVN钩子

你可以将Arcanist安装为一个SVN预提交钩子，来拒绝包含有错误的代码提交。例如，含有语法错误的代码是不能提交的，但是你可以通过干预代代码分析引擎，阻止其他类型的错误。

为了把Arcanist安装成一个预提交的钩子，需要添加如下语句在 svn/hooks/pre-commit中：

#!/bin/sh

set -e # Exit with an error code if this fails.

/path/to/arcanist/bin/arc svn-hook-pre-commit $@ 1>&2

确保这个文件是可执行的，否则你将会在每次提交得时候，看到一些毫无帮助的错误信息。在运行脚本之前，在SVN的环境变量中同样需要指定PHP的绝对路径（或者你可以在PATH中明确得指明路径）。

当你的工程已经配置好代码分析器或者代码分析引擎，但是它们都不是Arcanist的一部分。你可以使用--load-phutil-library 选项进行指定加载路径。

--load-phutil-library=/path/to/library/root

在你构建新的代码分析器，或者是自定义代码分析引擎后，SVN提交钩子无法访问到工作副本时，你就需要检查一下是不是引用了--load-phutil-library 选项的配置是否正确。例如，你的钩子配置可能与下图配置相似：

#!/bin/sh

set -e # Exit with an error code if this fails.

/usr/local/bin/php -f /path/to/arcanist/bin/arc svn-hook-pre-commit \

--load-phutil-library=/path/to/custom/lint/engine \

--load-phutil-library=/path/to/custom/unittest/engine \

$@ 1>&2

## Arcanist 使用教程: Windows

阐述如何在Windows上安装Arcanist。

### 概述

Arcanist在Windows上运行，需要在cmd.exe或者Git Bash环境中运行。但是，想让Arcanist运行在Windows还是需要花些功夫的。本章将说明应该怎么做，才能让Arcanist在Windows上运行起来。

NOTE: 比较新的版本才支持在Windows上工作，在你运行得过程中，可能会有功能不完善，以及文件错误等问题。

### 必要组件

你需要安装：

* Arcanist, 参考 [Arcanist User Guide](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide&type=article&jump=1)章节。
* PHP (参考下面 "详细 PHP 安装文档" 章节)。
* SVN, Git, 或 Mercurial代码版本控制软件。

然后，配置：

* 你的PATH环境变量需要包含arcanist/bin, php，还有你的VCS指令（svn，hg，git）。当你通过命令行输入php，arc，或者git的时候，它们需要能工作。
* 你的编辑器环境变量需要指向一些命令行编辑器，比如Git Bash vim。如果你喜欢，你可以在arc设置它们。详细的内容，后半节会详细介绍。

你可以在系统控制面板中的系统属性Advanced表中环境变量的设置。

现在你可以正常的使用arc（使用cmd.exe 或者Git Bash命令行）了。Arcansist就能正常的运行在Windows下了。

### 配置编辑器

NOTE: 你不能使用Notepad作为你的编辑器，因为它没有阻塞模式。你可以使用GitPad来代替它。

有些arc工作流提示你使用编辑器来编辑“大块”代码。你可以使用你喜欢的编辑器来，配置多套解决方案。很多编辑器都可以用来做这件事：

* [Notepad++](http://notepad-plus-plus.org/#_blank), 一个全方位的编辑器。
* **vim**, Git Bash中自带的编辑器.
* [GitPad](https://github.com/github/gitpad#_blank), 它允许你使用Notepad作为你的编辑器。

其他编辑器也是可以使用的，不过要确认他们拥有可阻塞编辑模式。

为了配置一个编辑器，需要设置EDITOR环境变量来指定它，或者使用如下命令行方式：

$ arc set-config editor "\"C:\path\to\some\editor.exe\""

NOTE: 注意双引号得使用。当路径中有空格时，需要使用引号。并且，双引号必须和arc set-config分开，例如下面的例子。

使用如下命令可添加Notepad++作为你的编辑器（以下路径需要换成你本机的路径）。

**Notepad++**

$ arc set-config editor "\"C:\Program Files (x86)\Notepad++\notepad++.exe\" -multiInst -nosession"

添加 Vim作为编辑器 (替换成你本机的路径):

**vim**

$ arc set-config editor "\"C:\Program Files (x86)\Git\share\vim\vim73\vim.exe\""

添加GitPad作为编辑器 (替换成你本机的路径):

**GitPad**

$ arc set-config editor "\"C:\Users\yourusername\AppData\Roaming\GitPad\GitPad.exe\""

添加Sublime Text 作为编辑器(替换成你本机的路径):

**Sublime Text**

$ arc set-config editor "\"C:\Program Files\Sublime Text 2\sublime\_text.exe\" -w"

### 安装PHP

当机器上有多个版本的PHP，并且有问题时，可以参考以下的建议。

* 从 [http://windows.php.net/download/](http://windows.php.net/download/#_blank).下载最新稳定版本的PHP安装文件。在更新本章节是，PHP使用的版本是5.4.6，更高的版本也是能正常工作得。使用线程不安全（"VC9 x86 Non Thread Safe" ）的版本，也是可以正常工作的。（5.3之后的版本都能工作正常，同样的线程安全的版本也可以正常工作。）
* 在PHP目录中解压安装包。
* 将 php.ini-development 复制到与 php.ini 同级目录下。
* 打开 php.ini 文件，找到行 ;extension=php\_curl.dll。去掉 ; 让该扩展生效。找到行 ; extension\_dir = "ext" ，然后修改成 extension\_dir = "C:\PHP\ext",这里的 C:\PHP 是你安装PHP的目录地址。
* 通过命令行，键入php -i，在观察curl的输出是否正常，来验证PHP是否工作正常。

### 下一步

可以继续如下章节:

* 返回 [Arcanist User Guide](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide&type=article&jump=1)章节。

## 审核（Audit） 使用教程

介绍 审核（代码上传后回查）工具 和 工作流程。

### 概述

Phabricator 支持两种代码回查流程，“回查”（review，提交前） 和“审核”（audit，提交后）。想要两者之间的区别，可以参考 [User Guide: Review vs Audit](https://secure.phabricator.com/diviner/find/?name=User_Guide%3A_Review_vs_Audit&type=article&jump=1)章节。

本章总结了通过审核工具如何实现，代码提交后得“审核”流程。

NOTE:审核功能是是一项新功能，请给我们已反馈，以便去改进它！ 反馈方式可见 [Give ! Get SuppFeedbackort!](https://secure.phabricator.com/diviner/find/?name=Give_Feedback!_Get_Support!&type=article&jump=1)章节。

### 详细介绍

审核功能允许你上推和部署代码，而不需要等待代码回查过程结束。审核工具主要会去跟踪两件事情：

* **代码提交（Commits）** 展示其审核状态(比如 “未审核”（Not Audited），“审核通过”（Approved）, 或 “担忧”（Concern Raised）).
* **审核请求（Audit Requests）** 可以要求用户（一个或多个）去审核一次提交。这可以通过很多方式去触发，详见下文。

在审核工具的目录 (在 /audit/) 和首页中，你能看到代码的提交，以及发送给你的一些请求：

* **要求审核（Required Audits）** 当你所在得工程或者工具集需要审核一次提交，在打开审核请求页面是，你就会看到要求你审核的请求。当你通过相关的提交审核，那么审核要求就会关闭。
* **有问题的提交（Problem Commits）** 当审核过程中遇到有问题的提交，会将该提交置为了“担忧”态。当有问题的提交在通过修改以后，让每个审核者审核过，并置为“审核通过”状态后，这个有问题的修改才算是通过了审核。

例如：

* Evan提交了一个 abcdef1234 版本，并且将其上推到代码服务器上。
* 在这次操作将通过一些机制，触发一个审核请求。（下文有对触发机制的描述）
* 之后，Bob通过 Phabricator看到了提交日志，并且在他会在的主页上看到审核请求。
* Bob点击进入，开始审核本次提交。他发现了一个问题，然后他就对本次提交设置成“担忧”态，并且对这个问题进行一定的描述。
* Evan收到有关这次提交的邮件，在邮件里面他可以看到Bob的描述和担忧。在看过邮件后，他决定先不着急处理。
* Evan通过Phabricator可以看到他得这次提交被置为“有问题的提交”的状态。
* Evan通过一些方式去解决这个问题（比如，和Bob讨论一下，或者进行修改再提交新的版本）。
* 当审核通过，Bob将会把原始提交的状态置为“接受”（Accepts）状态。
* 之后，有关这个问题的请求信息将会在Bob和Evan的请求队列中消失，也就是问题顺利解决。

### 审核触发器

以下列出的方法可以触发审核请求：

* 如果你在系统跟踪的分支上进行提交操作，且提交信息中有 Auditors: username1, username2 内容。那么这个操作，这将会对这些用户发出审核请求
* 你可以在Herald中创建一些规则，根据提交的属性来触发审核。比如，修改特定的文件，修改特定的内容，针对特定的提交人，等等。
* 你可以通过提交任意的提交，为自己创建一个审核请求。
* 你可以创建一个“所有者包”，以及开启审核功能 (这是一项针对大型开发组的高级特性)。

### 审核在小型组的使用

如果你所在的是一个小型开发组，就不需要一些复杂的触发规则，你可以设置一些简单的审核工作流，如下所示：

* 在“Code Audits”中创建一个新工程。
* 为版本提交，创建一个全局Herald规则，这个规则可以让每次提交都触发“代码审核”。当没有使用“Differential Revsion”时，这个流程可以作为代码回查的部分/全程缓冲。
* 把所有工程师都加入“Code Audits”工程中。

通过何种方式，对于每次的代码提交所有人都能看到审核请求，但是任何一个成员都可以关闭这种机制。有效的方式是，通过规则强制“每个人都必须看到每次提交信息”。

当你的开发组逐步扩大时，你可以细化这些规则集，能让开发者们能看到与他们做的部分相关的提交与修改。

### 审核技巧

* 在对待回查提交请求，以及处理审核请求时， 需要特别认真。
* 你可以通过点击对应的修改行，并在这行后面添加评论。
* 你也可以通过在行间拖拉的方式对多行进行评论。
* 在个行间的评论将会存成草稿。当你在页面底部提交评论时，这些评论才会被提交。
* 点击“？”来了解快捷键。

### 下一步

* 可以在 [Herald User Guide](https://secure.phabricator.com/diviner/find/?name=Herald_User_Guide&type=article&jump=1)章节了解有管Herald规则的内容。
* 给我们必要的反馈 [Give Feedback! Get Support!](https://secure.phabricator.com/diviner/find/?name=Give_Feedback!_Get_Support!&type=article&jump=1).

## 配置教程: 高级配置

为各种环境配置Phabricator。

### 概述

Phabricator 从各种源文件件中读取配置信息。本章将阐述配置栈和怎么对源文件进行配置高级。这将会对部署在Phabricator中更重环境（比如，开发和生产）有很大帮助。

这对于高级用户来说也是比较复杂的，你可以不用了解这里是如何安装进Phabricator。

### 源码配置

Phabricator 支持一下源码配置（优先级从高到底列出）：

* **数据库（Database）：**设置的值都将在数据库中进行保存，并且管理员可以通过web图形界面方式对其进行修改。它们拥有最高的优先级，并且可以复写其他Pharicator的设置。
* **本地（Local）**: 本地变量值存在 conf/local/config.json 中，可以通过运行 bin/config对他们进行修改。
* **配置文件（Config Files）**：变量值存储在一个配置文件中，该文件在 conf/。这个文件使用已经写好的文件 conf/local/ENVIRONMENT, 或者设置 PHABRICATOR\_ENV变量来指定。更多信息可以参考下文。
* **默认项（Defaults）**： 默认Phabricator源码为硬编码模式，也就是不能去修改的。他们拥有最低的优先级，并且所有设置都可以对他们进行复写。

通常情况下，你通过访问数据库（例如，MySQL用户名，密码和主机名）来写入本地变量，安装以及配置Phabricator。在之后的配置都可以通过web图形界面完成。

### 文件配置

配置文件可以成为替代数据库的一种手段，并且适用多种环境的部署或动态配置。默认情况下是不推荐使用配置文件的，因为配置文件的复杂程度要比数据库高的多。

#### 创建配置文件

创建一个配置文件，首先要为配置文件选择一个合适的名字（比如，“deserver”或“live”）。指定对应的部分，假设你使用“exampleconfig”。可以使用下面的例子来替代这个“exampleconfig”文件：

首先，创建一个exampleconfig.conf.php文件（可以根据你的需要重命名它）：

phabricator/conf/custom/exampleconfig.conf.php

它的内容如下所示:

<?php

return array(

// Specify whichever keys and values you want to set.

'example.key' => 'examplevalue',

);

例如，要在你的配置文件中指定MySQL认证，你的配置文件可以参照以下内容：

<?php

return array(

'mysql.host' => 'localhost',

'mysql.user' => 'root',

'mysql.pass' => 'hunter2trustno1',

);

#### 选择配置文件

选择一个配置文件，将它的文件名(相对于 phabricator/conf/) 写到 phabricator/conf/local/ENVIRONMENT中。例如：要选择 phabricator/conf/custom/exampleconfig.conf.php, 你就要在 phabrictor/conf/local/ENVIRONMENT写入“custom/exampleconfig”：

phabricator/ $ echo custom/exampleconfig > conf/local/ENVIRONMENT

phabricator/ $ cat conf/local/ENVIRONMENT

custom/exampleconfig

phabricator/ $

你也可以设置环境变量 PHABRICATOR\_ENV。虽然可能会对系统之外环境有影响，但是在部署了多种环境的情况下，这中方式还是很简单有效的。注意，这里需要去配置你的web主机的环境，也要配置你要运行的shell脚本：

# Shell

export PHABRICATOR\_ENV=custom/exampleconfig

# Apache

SetEnv PHABRICATOR\_ENV custom/exampleconfig

# nginx

fastcgi\_param PHABRICATOR\_ENV "custom/exampleconfig";

# lighttpd

setenv.add-environment = (

"PHABRICATOR\_ENV" => "custom/exampleconfig",

)

在创建和选择完配置文件以后，重启你的web主机。你做的任何配置将会马上生效，并且你得配置文件也会在测试配置时被看到。

### 下一步

返回 [Configuration Guide](https://secure.phabricator.com/diviner/find/?name=Configuration_Guide&type=article&jump=1)章节。

## Differential 使用教程

了解Differential（上传前代码回查）工具和工作流程

### 概述

本章介绍Differentail工具，在上传前“回查”的工作流程的实现。

### 如何进行代码回查

在 Phabricator系统中，代码回查是一个轻量级、网络异步处理的过程。类似于在GitHub中对pull命令的响应：

* 代码作者已经修改了代码，然后上传，并回查。可以指定谁来回查这个版本的代码（其他的用户可能也会被提醒到，详情见下文）。这里其本身得修改就称之为一个“Differential Revision”。
* 指定的回查者将会收到一封邮件，要求他们回查对应的修改。
* 回查者将会检查这次修改，对它进行讨论，或批准这次修改，或要求再次修改（比如，他们认为这次修改里面存在漏洞和错误）。
* 代码作者需要对回查结果做出回应。（例如，修复漏洞，或者解决问题）。
* 当过程中一切顺利，多数回查者接收了这次修改，然后作者就可以将他修改后的代码进行上传了。

Differential 主页显示两种修订：

* **请求回查（Action Required）** 当你作为修订的作者或者回查者，需要去回查，修改或者上传代码的修改。
* **等待他人（Waiting on Others）** 当你作为修订的作者或者回查者，需要去回查，修改或者上传代码的修改。

### 创建修订

在Differential上可以使用arc命令优先创建代码修订(可见 [Arcanist User Guide](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide&type=article&jump=1))。你也可以通过Differential导航，使用web接口创建代码修订。点击“创建修订”按钮，并在对话框中粘贴一个diff信息，之后就可以创建修订了。

### Herald 规则

如果你对代码库中已存的代码修改轨迹感兴趣（比如，可能改变了一种特性;或者更换了一种语言;或者那些代码修改是实习生创建的，你并不信任它们），你可以撰写一个Herald规则，进行自动匹配（比如，内容，作者，文件改动等），然后抄送给你。

### Differential 使用技巧

* 你可以通过点击对应的修改行，并在这行后面添加评论。
* 你也可以通过在行间拖拉的方式对多行进行评论。
* 在个行间的评论将会存成草稿。当你在页面底部提交评论时，这些评论才会被提交。
* 点击“？”来了解快捷键。

### 下一步

* 阅读FQA，在 [Differential User Guide: FAQ](https://secure.phabricator.com/diviner/find/?name=Differential_User_Guide%3A_FAQ&type=article&jump=1)章节
* 了解如何处理重大改动，可见 [Differential User Guide: Large Changes](https://secure.phabricator.com/diviner/find/?name=Differential_User_Guide%3A_Large_Changes&type=article&jump=1)章节
* 了解有关测试计划的内容，可见 [Differential User Guide: Test Plans](https://secure.phabricator.com/diviner/find/?name=Differential_User_Guide%3A_Test_Plans&type=article&jump=1)章节
* 了解更多有关Herald的信息，可见 [Herald User Guide](https://secure.phabricator.com/diviner/find/?name=Herald_User_Guide&type=article&jump=1)章节
* 请给与我们反馈 [Give Feedback! Get Support!](https://secure.phabricator.com/diviner/find/?name=Give_Feedback!_Get_Support!&type=article&jump=1).

## Differential 使用教程: FAQ

有关Differential 的常见问题。

### 为什么“已接受”（accepted）的代码更新后依然是接受状态？

当在Differential中更新“已接受”状态的代码，之后的代码状态依旧是“已接受”。这可能会让你很迷惑，以为在代码更新过之后，你期待代码的状态是“需要回查”（needs review）。

这是故意为之的， 为了激励开发人员在“已接受”代码版本上，即使做了很小的修改，都会去更新代码。例如：回查者可能接受某个修改，并加以类似以下内容的评论。

*Looks great, but can you add some documentation for the foo() function*

*看起来很不错，但是你需要对foo()函数添加一些说明  
before you land it? I also caught a couple typos, see inlines.*

*在你那里通过了吗？我这里依旧会有问题，看一下内联评论*

如果代码状态伴随更新而改变状态为“需要回查”，开发者会比较沮丧，因为他们更新的版本只是做了很小的修改。并且他们将不得不等待其他回查者，在有空的时候，对他们的代码再进行一次回查。

“已接受”状态也是为了激励开发者更新代码时，提交合理的评论：

*- Added docs.*

*-添加文档  
- Fixed typos.*

*-修复错误*

这就比二次检查要简单的多，这能节省很多时间。如果代码作者要求，可以进行状态修改，并进入回查流程。

如果有人做了重大得修改，并且需要回查者进行审核。他们可以对一个“已接受”的代码版本“要求回查”，用类似如下的评论：

*When I was testing my typo fix, I realized I actually had a bug, so I had to  
make some more changes to the bar() implementation -- can you look them over?*

*当我测试时，这个问题已经修复了。我发现之前的实现的确存在漏洞，所以我对bar()函数进行了重写----你们可以对这个函数进行下检查吗？*

### 如何开启语法高亮？

你需要安装以及配置Pygments来高亮标示出了PHP的语言。可以通过其他文档，了解如何配置。

### 空格选项是做什么用的？

大多数选项都很简单，除了“多数忽略”：

* **全部显示（Show All）：** 显示修改中所有的空格。
* **尾部忽略（Ignore Trailing）：** 只是忽略修改中每行尾部的空格。
* **多数忽略（Ignore Most）**: 对于标注“具有很多空白”的文件，可以忽略始端或尾部的空格（但是不会显示两个非空格字符之间的空格修改）。在那些文件中，会显示空格的修改。 在默认情况下，Python（.py）和Haskell(.lhs, .hs)文件将会标记为“具有很多空格”，但是这可以在 differential.whitespace-matters 变量中进行修改。
* **全部忽略（Ignore All）**: 对所有修改后文件做空格忽略处理。

### 淡绿和红色背景是什么意思？

Differential 使用这些颜色变化来标记他们与之前版本不同得地方，这里的显示可能和作者得修改不大一样。他们可以凸显版本间的不同之处。 可以看一下 [D3324](https://secure.phabricator.com/D3324?vs=6468&id=6513#_blank) 的例子。

## Differential 使用教程: 重大修改

不使用Differential处理重大修改集，

### 概述

当你想要为一个给定的修改集做代码回查，Differential可能不是一个合适的工具。经验法则，如果你需要有人来做代码回查，你只需要给Differential发送修改的内容即可。其应该以最大能力去覆盖修改范围，但是你不应该通过Differential去提交你的代码，下面的例子就是如此：

* 提交一个整个开源工程到一个你可以进行分支和连接私人库中。
* 提交一个庞大的文本文件，比如单词清单，或者一个转存的数据库。
* 对10,000个问题件做一个简单的修改（比如，查找/替换，代码改动）

你可以继续尝试提交此类修改，但是要让他们工作时，你可能会遇到一些问题（比如，数据库连接超时）。Differential 是一个快速、可扩展的工具，但是也要分时候，有时会让浏览器崩溃：你不可能让一个有九百万行得文件在浏览器上进行显示。

更重要的是，在所有的案例中，文本的修改一般不会被人们回查。与元数据相关的修改才会需要人去做回查工作（比如，你在检查什么，你把它放在哪里，为什么这样做？这样的修改是否有意义？在例子中做了自动变换，你用了什么脚本做了这件事？）。为了对这些类型化的审核，一些策略通常能比使用Differential获取全部修改更为合理：

* 发送一封邮件/AIM/IRC给你的回查者，内容可为“你好，我正在检查*/full/path/to/whatever*位置为MySQL 9.3.1所写的源码。这次在*/home/whatever/path/somewhere*做了修改，这里需要你检查一下。我能把你加入回查者列表中吗？”。这适合与简答得修改。回查者将不会去回查MySQL本身的源码，他们会去检查那些，在你要求回查的代码版本中被修改了的元数据，检查你为什么要回查它们，你把它们都放在哪里？这里你不能使用“arc commit” 或“arc amend”去提交代码。这里只能使用“svn”或“git”，或者使用手动修改提交信息。（通常添加一个名为“Reviewed By: <username>”的域）
* 创建一个只有元数据的Differential修订，比如使用脚本进行自动修改。包含一个连接到修改存在的地方，所以如果回查者需要，他们可以之间看到修改集。当Differential使用在讨论，且提供其他人可以引用的永久性记录，这对于多数复杂的修改是很有帮助的。一旦修订被接收，修改将会在本地进行提交（例如，通过git commit --amend）。

这些类型的修改是是否罕见的，也不具有普遍性，这就是为什么Differential没有明确表示支持他们的原因。如果你经常运行此类例子，并且Differential不能处理他们，请将你的例子反馈给我们。

## Differential 使用教程: 测试计划

现在你应该关注一下在开发过程中的测试计划(test plan)了。

### 概述

当你向Differential发送这个修订时，你必须包含一个测试计划（这个可以禁止，或者使用选项进行配置）。测试计划是一套重复列表，书面化了你要为一个修改所要做的验证。好的测试计划需会对回查者有很大的帮助，你可以通过它了解你的修改与预期行为是否一致，可以让其他人用各不同的方式来验证你的修改是否合理。

本节有些普遍性的事情，需要在开发或回查一个测试计划时，去考虑。这里有些建议可能不会应用于你的开发过程; 这些建议是改编自Facebook内部文件。

### 全修改覆盖

* **错误处理（Error Handling）:** 是对错误进行检测和立即处理？ 你如何处理错误的例子？ 你对他们进行测试了吗？ 确定你通过正确得操作，获取了正确的错误信息？ 对于测试出错的现场还原是很重要的，而不是当工作正常时就修改所有文件了事。
* **服务影响（Service Impact）:** 你的修改是如何对服务进行影响的，比如对缓存、开发框架或数据库？ 是否添加新的缓存密钥或需求？ 这个修改会加载一大堆服务吗？
* **性能测试（Performance）:** 你的修改会影响性能吗？ 注意：如果你得对该对于性能有影响，你应该包含有测量数据在你的测试计划中证明，你的修改提高了性能。
* **单元测试（Unit Tests）:** 你的修改已经被单元测试覆盖了吗？ 你能提高测试的覆盖范围吗？ 如果你修复了一个漏洞，你有添加一个测试避免这个漏洞再次出现吗？单元测试是否仅仅就测试了出问题得地方，是否会在加载数据库或网络服务时再次出错呢？
* **并行修改的鲁棒性（Concurrent Change Robustness）:**  如果你为某次提交做了重构，那其别人开始对这个这次修改的介绍，和你重构完后的健壮性是否一致？ 例如，你改变了参数列表得顺序，从f(a,b)到f(b,a)，并且心的调用点使用了旧的参数顺序，这个问题有被注意到吗？ 这次修改最坏得情况是怎么样的？ （因为这是很冒险的，你应该不该在一个弱类型语言中去修改参数顺序，比如PHP和Javascript）。
* **恢复计划（Revert Plan）:** 如果你的修改需要恢复，并且当要恢复时，你并不在旁边，是不是有特殊的恢复步骤，或者有些特别的信息需要恢复者了解呢？ 如果有，请确保恢复的步骤等信息已经在恢复计划中写清楚，以便其他人在不知道你的已有补丁（但是需要有些有关系统的基本知识）的情况下，也能成功的恢复你的修改。
* **安全性（Security）:** 你的修改是否能避免XSS（跨站脚本攻击），CSRF（跨站请求伪造），以及注入式攻击？ 你确定用户都能拥有合适的权限？ 你还将用户的数据作为不可靠的？ 当一些危险的函数在必要的时候被使用，你能确保数据不向外界泄露吗？
* **架构（Architecture）:** 确定这是一次合理的修改？ 还有更好的办法来解决这个问题吗？ 当你的项目中不只是你一个人时，你是否与这个领域的专家进行讨论（或者添加他们为回查者）？你是以什么为权衡凭证，并且为什么要用这个凭证？

### 前端/用户可见修改

* **静态资源（Static Resources）:** 你的修改将会提供JS或CSS代码，或者要使用更多服务吗？ 你可以减少JS/CSS的代码使用，或者复用一些代码吗？
* **浏览器（Browsers）:** 你使用别的浏览器测试过你的修改吗？

## Diffusion 使用教程

Diffusion是Phabricator系统中的代码库浏览器。

### 概述

Diffusion是一个代码库浏览器，它允许你通过网页来观察Git和SVN的代码库，和Trac和GitWeb软件的功能差不多。

Diffusion提供一个超高效的SVN浏览器，和一个高效的Git浏览器，还有一个相对较慢的Mercurial浏览器。Diffusion通过以非正规化形式，将大量代码库数据历史存入数据库，并且使用这些信息。例如，高速缓存数据。从而避免避免对代码库直接的查询。这些信息通过关注代码库轨迹（发现新的提交，分析和导入该提交）的守护进程生成。

在Phabricator 套件中，Diffusion也被其他工具所集成。例如：

* 当你提交Differential修订到一个以跟踪代码库时，他们将会自动更新和连接相应的提交。
* 你可以添加Herald规则来对对符合规则的提交进行匹配，并告知你。
* 工具拥有者可以使用Diffusion来映射对应的代码库。
* 在所有工具中，提交名是被自动连接的。

### 代码库代号和提交名称

每个库都有一个确定的代号，他可以是一个短大写字符串，例如“P”（Phabricator），或"ARC" (Arcanist)。

每个库的代号必须是唯一的。代号在安装中必须是唯一的，但是不需要全局唯一，所以可以自由得使用单字母的代号。例如：Fackbook使用“E”作为工程类库，“O”作为Ops库，“Y”作为Yum包的库，等等; 当 Phabricator使用“P”，“ARC”，“PHU”为 libphutil库命名，并且以“J”为javelin命名。保证代号短小精悍，易于使用，最好就用一个大写字母。

使用代号的原本用意是为了给SVN库在命名空间内进行提交：如果你使用多个SVN库，每个库都有版本1,版本2,等等。所有他们的号码是有冲突的。但是，即使这样，Git也会附加额外得信息给代码阅读者，并且允许分析器来检测提交名使用使用频率（和允许去识别多个库的备份）。

Diffusion 使用特定代号和自身提交生成的版本号。例如，“rE12345”或“[rP28146171ce1278f2375e3646a1e1ea3fd56fc5a3](https://secure.phabricator.com/rP28146171ce1278f2375e3646a1e1ea3fd56fc5a3)”。“r”代表“版本”（revision）。它会跟在库代号的后面，和一个VSC指定标示符（SVN为提交号;Git和Mercurial为提交哈希码）。当在使用一个Git提交版本时，可以使用其哈希值的缩写，但是要注意，哈希值前缀的长度。参考一下LKML在Git内部使用哈希值的7个字符的历史解释：

https://lkml.org/lkml/2010/10/28/287

因为7个字符的哈希在中等规模以上的库中，会发生冲突。Diffusion通常使用前16个字符（几乎不会引起冲突）或使用完整的40个哈希字符（不可能引起冲突）。

### 添加代码库

代码库管理通过“Repository”工具来完成，它主要是一套Diffusion的管理接口。为了添加一个库到Diffusion中，你需要：

* 使用Repository工具创建一个新库。
* 启动守护进程跟踪和导入代码库

创建一个新的代码库（或者修改/删除已有代码库） 「注意：你必须是管理员」 (参见 [Configuring Accounts and Registration](https://secure.phabricator.com/diviner/find/?name=Configuring_Accounts_and_Registration&type=article&jump=1) 章节，其中介绍了如何创建用户和管理员)。以管理员身份，找到Repository工具，然后你可以对库的对应参数进行创建和编辑。

当你创建了一个新库，你需要指定对库只读的人员，一个永久的“代号”（详情见上文），还有潜在的VCS类型。一旦你创建了一个库，你可以在“Tracking”页面，使用Diffusion进行跟踪设置。

在“Tracking”中有很多选项需要自解释，或者使用默认选项就可以。在“broad strokes”中，Diffusion通过“svn log”命令对SVN库进行定期跟踪，检查是否有新的代码提交;通过“git fetch”和“hg pull”对Git和Mercurial库进行本地拷贝。

当你完成所有配置（确定“Tracking”选项被设为“开启”状态），你就可以开始加载守护进程，对代码库进行跟踪。

### 运行Diffusion守护进程

在所有case中，都需要去运行如下命令：

phabricator/bin/ $ ./phd start

...开启守护进程。要想对phd和守护进程有更深入的了解，可以参考 [Managing Daemons with phd](https://secure.phabricator.com/diviner/find/?name=Managing_Daemons_with_phd&type=article&jump=1)章节。

NOTE: 如果你安装了许多大型Web前段，可以参考 [Managing Daemons with phd](https://secure.phabricator.com/diviner/find/?name=Managing_Daemons_with_phd&type=article&jump=1).章节中的注意事项。

你可以使用守护终端来观察守护线程导入库的过程。小型库很快就能导入进去，但是大型库需要等一段时间才能完成（Facebook原发库有350000个提交，Phabricator需要10分钟去扫描所有提交信息，从64个主机，花费近18个小时完成导入）。提交信息将会很快出现在页面中（除非代码库很巨大）。

#### 调谐守护进程

在默认情况下, Phabricator 加载一个守护进程用来下拉和追踪库的修改。对于小数量的库和大量相对冻结的库，这个进程可工作的很好，但是其他的一些例子也可以因此收益。守护进程有频率的对代码库发出下拉请求，这都是之前设置好的。在代码库数量太多时，可能不会很快得将最新的提交在页面中显示（请求在I/O端被阻塞，等待别的代码库的下拉操作完成）。

如果你要一个低延迟的库，你可以启动“多守护进程”（它通常会对所有库都有较低的延迟）或者加载“专用守护进程”（它将很好得控制延迟）。

#### 多守护进程

为了降低延迟，最简单得办法就是，使用phd加载更多的守护进程 ：

phabricator/bin $ ./phd launch RepositoryPullLocal

这将加载一个守护进程副本。守护进程需要在进行下拉一个库之前，上一个全局锁。所以你你加载了更多的守护进程是不会引起冲突的。

#### 专用守护进程

你可以对一个指定库，或者一组指定库，使用一个易操作的方法和加载专用守护进程。例如，如果你想要减小A和B库的延迟，而且你不关心其他库的延迟，你可以使用如下方法加载两个守护进程。

phabricator/bin $ ./phd launch RepositoryPullLocal -- A B

phabricator/bin $ ./phd launch RepositoryPullLocal -- --not A --not B

第一条指令，仅仅会为A和B库工作，并且以低延迟/可靠性高的导入提交。第二条指令，将会对除了A和B的库，进行导入加速。

### 下一步

* 了解如何创建一个符号索引，可参考 [Diffusion User Guide: Symbol Indexes](https://secure.phabricator.com/diviner/find/?name=Diffusion_User_Guide%3A_Symbol_Indexes&type=article&jump=1)章节
* 了解与守护进程更多的信息，可参考 [Managing Daemons with phd](https://secure.phabricator.com/diviner/find/?name=Managing_Daemons_with_phd&type=article&jump=1)章节
* 给予我们反馈 [Give Feedback! Get Support!](https://secure.phabricator.com/diviner/find/?name=Give_Feedback!_Get_Support!&type=article&jump=1).

## Diffusion 使用教程: 代码库钩子

配置Phabricator 代码库钩子。

### 概述

Phabricator 可以作为代码库的主机，并且提供访问权限配置，可通过HTTP和SSH进行读写访问。本章主要介绍如何在Phabricator上配置代码库托管。

NOTE:这个特性的还是比较粗糙。如果有什么问题，请反馈给我们 (可参见 [Give Feedback! Get Support!](https://secure.phabricator.com/diviner/find/?name=Give_Feedback!_Get_Support!&type=article&jump=1)章节）。

### 了解支持的协议

Phabricator 支持以下协议进行托管：

|  |  |  |
| --- | --- | --- |
| **VCS** | **SSH** | **HTTP** |
| Git | Supported | Supported |
| Mercurial | Supported | Supported |
| Subversion | Supported | Not Supported |

所有支持的协议，都可处理读（pull/checkout/clone）和写（push/commit）。两个协议中，SSH通常更健全、更安全，以及高性能。不过，HTTP更容易配置，也更容易匿名访问。

|  |  |  |
| --- | --- | --- |
|  | **SSH** | **HTTP** |
| Reads | Yes | Yes |
| Writes | Yes | Yes |
| Authenticated Access | Yes | Yes |
| Push Logs | Yes | Yes |
| Commit Hooks | Yes | Yes |
| Anonymous Access | No | Yes |
| Security | Better (非对称密钥) | Okay (Password) |
| Performance | Better | Okay |
| Setup | Hard | Easy |

每个代码库都需要单独配置，并且你可以使用其中一种协议，或者全都使用。

这里推荐使用SHH，除非你需要匿名用户访问，或者因为技术原因无法对SSH进行配置。

### 配置系统用户账号

Phabricator 拥有3种系统用户账号。这就需要你去创建和配置他们。他们作为系统用户账号在Phabricator主机上运行，而不是普通用户账号。

配置系统账号有以下几种：

* 作为守护进程运行的，我们称之为“daemon-user”。有关守护进程的更多信息，可参考 [Managing Daemons with phd](https://secure.phabricator.com/diviner/find/?name=Managing_Daemons_with_phd&type=article&jump=1)章节。
* 作为web主机运行的，我们称之为“www-user”。如果你不打算使用HTTP协议访问代码库，你就没必要对这个用户进行特别的配置。
* 通过SSH连接运行的，我们称之为“vcs-user”。如果你不打算使用SSH协议访问代码库，你就没必要对这个账户进行特别的配置。

如何配置这些账户：

* 当没有daemon-user时，需要创建一个“daemon-user”。当你启动线程时，也就使用了这些账号。
* 当没有www-user时，需要创建一个“www-user”。通过web主机来启动这个账号。注意，启动的用户一定要存在才行。
* 当没有vcs-user时，需要创建一个“vcs-user”。对于Git和Hg，其名字是相同的。当用户clone代码库时，他们会用到类似的地址 vcs-user@phabricator.yourcompany.com。

现在，允许vcs-user和www-user如同daemon-user一样，可以使用sudo命令。这里，需要使用visudo或sudoedit命令，将这些用户加入/etc/sudoers文件中。

如果打算使用 SSH:

vcs-user ALL=(daemon-user) SETENV: NOPASSWD: /path/to/bin/git-upload-pack, /path/to/bin/git-receive-pack, /path/to/bin/hg, /path/to/bin/svnserve

如果打算使用 HTTP:

www-user ALL=(daemon-user) SETENV: NOPASSWD: /usr/bin/git-http-backend, /usr/bin/hg

可以将 vcs-user, www-user 和 daemon-user 替换成你需要配置的用户名。确定所有二进制文件路径都在你系统中存在（确定路径是否正确）。你可以省略一些，你不使用的VSC二进制文件路径。

在sudoers文件中添加这些指令，将会允许守护进程和web主机用户对代码库进行写操作。

最后，当你配置完sudoers后，还需要将phd.user设置为daemon-user：

phabricator/ $ ./bin/config set phd.user daemon-user

### 配置 HTTP

如果你打算使用HTTP认证，你需要在Config中配置diffusion.allow-http-auth选项。如果你不想使用HTTP，或者打算只使用匿名HTTP，你可以将这个选项置为“关闭”。

但是，如果你已经在之前配置了系统账户，也就是你已经进行了设置。就没有必要在去对HTTP进行配置了。

### 配置 SSH

SSH访问需要一些而外得配置。这里我们来介绍看一下如何进行配置：

* 你可以将sshd守护进程移至其他端口，例如222。你可以使用此端口，来对连接到的机器进行管理。
* 你在22端口，通过一个锁定配置，来运行一个高度限制的sshd守护进程。这个项功能，可以使用Phabricator来认证用户和执行指令。
* 当sshd守护进程运行在22号端口时，需要是6.2或者更高版本，因为Phabricator依赖与 AuthorizedKeysCommand 选项。

这里展示一下配置的细节信息：

**移动 SSHD端口**: 当对sshd守护进程进行配置时，需要很小心。如果你错误配置了它，你可能将你锁在机器的外面。重启sshd时，会中断当前已经存在的连接，但是你还是要很谨慎。两种策略可以降低冒险系数：冒烟测试，通过启动另一个sshd守护进程进行配置; 使用“screen”命令，可以自动修复你与机器的连接。

为了配置冒烟测试，仅仅需要加上-f选项，启动另一个sshd就可以:

sudo sshd -f /path/to/config\_file.edited

你可以通过这个方式来进行测试，在替换原有配置文件之前，来验证你的配置是否正确。

为了自动修复配置文件，使用如下的指令启动一个screen:

sleep 60 ; mv sshd\_config.good sshd\_config ; /etc/init.d/sshd restart

这条指令可能在你的系统中略有不同，不过总体思想是，在你不反对的情况下，在一段时间后自动保存配置文件。当你把自己锁在机器外面，这条指令可以帮助你自动的恢复一些配置。

现在你可以编辑你的配置文件，打开sshd配置文件（通常在/etc/ssh/sshd\_config），然后改变端口设置，将端口改为别的（你可以选择出了22号端口的其他端口）。

Port 222

这里要十分小心，然后重启sshd。通过连接新接口来验证配置是否正确：

ssh -p 222 ...

**配置并启动 Phabricator SSHD**: 现在，配置并启动另一个sshd（在22号端口）。这个例子将会使用一个特别的锁定配置，来处理验证和命令执行。

下面这三步很重要：

* 创建 phabricator-ssh-hook.sh 文件
* 创建 sshd\_phabricator 配置文件
* 启动一个使用新配置的 sshd 副本。

**创建 phabricator-ssh-hook.sh**:可以从 phabricator/resources/sshd/phabricator-ssh-hook.sh 拷贝模板到 /usr/libexec/phabricator-ssh-hook.sh （或其他你喜欢的路径）。并且编辑它来配合当前的配置。然后，让他被root用户拥有，并限制对其的编辑：

sudo chown root /path/to/phabricator-ssh-hook.sh

sudo chmod 755 /path/to/phabricator-ssh-hook.sh

如果你不做这些事情，sshd将会拒绝执行钩子命令。

**为Phabricator创建 sshd\_config 配置文件：**在  phabricator/resources/sshd/sshd\_config.phabricator.example 拷贝模板到 /etc/ssh/sshd\_config.phabricator（或者其他路径）。

打开这个文件，然后修改 AuthorizedKeysCommand 和 AuthorizedKeysCommandUser 来适配你当前使用的系统。

**启动 SSHD**: 现在，启动Phabricator sshd:

sudo sshd -f /path/to/sshd\_config.phabricator

如果你之前配置的没有错，你就可以运行如下的指令了:

echo {} | ssh vcs-user@phabricator.yourcompany.com conduit conduit.ping

...然后，就会得到类似如下的响应信息:

{"result":"orbital","error\_code":null,"error\_info":null}

(如果验证错误，确认一下你是否将你的公钥加入 **Settings > SSH Public Keys**中。)

### HTTP认证

为了通过HTTP验证，用户需要在 **Settings** 页面中配置 **VCS Password** 。要使这个面板生效，需要将diffusion.allow-http-auth置为开启状态。

### SSH认证

为了通过SSH进行验证，用户需要添加SSH公钥在Setting面板中。

## Diffusion 使用教程：符号索引

如何配置和使用符号索引。

### 概述

Phabricator可以包含一个符号索引，这个索引是用来追踪代码中定义的类和函数。当你开启了索引，你可以用索引来做如下的事情：

* 在Differential使用符号连接进行代码回查。
* 允许你通过符号进行搜索。
* 可以使用IRC服务来回答类似“某个类在哪？”的问题。

NOTE：符号索引是新功能，不过其已经在PHP语言还有其他语言中广泛支持了。

### 填充索引

所谓填充索引，你需要写一个脚本来对你的代码库做符号甄别，并且在流水线中设置一个任务，让其进行输出。

./scripts/symbols/import\_project\_symbols.php

Phabricator 包含了一个可以甄别PHP工程中的符号信息的脚本:

./scripts/symbols/generate\_php\_symbols.php

Phabricator 还包含一个可以甄别任何一种语言的类和/或函数的脚本，这个脚本是由Exuberant Ctags提供([http://ctags.sourceforge.net](http://ctags.sourceforge.net/#_blank)):

./scripts/symbols/generate\_ctags\_symbols.php

如果你想要甄别其他语言，你需要写一个脚本进行输出他们 (例如，可以分析一个ctags文件)。

脚本每行的输出格式，如下所示:

<context> <name> <type> <lang> <line> <path>

例如:

ExampleClass exampleMethod function php 13 /src/classes/ExampleClass.php

上下文（广义）是特定范围或者命名空间的符号定义。对于面向对象语言，这可能就是一个类名。这个符号在上下文中，就是类常数、函数、属性、嵌套类，等等。当以没有上下文情况下，对符号信息进行输出（比如，这些符号在全局进行定义），这里<context>域必须是空的。

你的脚本文件需要枚举出，在你的工程所用的全部符号，并且以你的工程根目录为基准（.arcconfig所在目录），以“/”标注。

你可以以generate\_php\_symbols.php为例，来了解如何写这个脚本，并且使用以下命令行启动它，并观察它的输出：

$ cd phabricator/

$ find . -type f -name '\*.php' | ./scripts/symbols/generate\_php\_symbols.php

事实上构建符号索引，通过管道方式为import\_project\_symbols.php脚本获取数据，这里需要提供工程名:

$ ./scripts/symbols/import\_project\_symbols.php yourproject < symbols\_data

然后，建立一个任务运行。

你可以使用Conduit函数differential.findsysmbols，来测试导入符号队列。一些特性（比如函数，和IRC服务集成）将会立即工作。其他的函数将会需要更多的配置。

### 集成Differential

要集成Differential，你需要告诉Phabricator你要使用哪个工程里面的符号索引，还有是否要从其他工程抽取符号。以管理员账号登陆，在 Repositories -> Arcanist Projects -> Edit 进行设置。你需要在以下域中填写必要的内容：

* **代码库（Repository）**: 一个作为副工程的已跟踪库。
* **已索引语言（Indexed Languages）**: 填写你已经构建过索引的所有语言。
* **使用符号来源（Uses Symbols From）**: 如果工程依赖于其他工程，需要添加其他工程，并且他们的符号信息都需要在这里能看到。例如，Phabricator中有“Arcanist”和“libphutil”，因为它使用了其他工程里面的类和函数。

当你已经配置一个工程，新版本将会在Differential自动进行符号连接。

NOTE: 因为这个特性需要依赖语法高亮，对于其他语言来说，对特定语言的处理，语法高亮的处理更胜一筹。它对于PHP来说工作相当不错，但是在你的使用中可能会使用到各种不同的语言。

## Diviner 使用教程

文档生成器。

### 概述

NOTE: Diviner 是一个新功能，并没有被普遍使用。

### 生成文档

要生成文档，需要运行:

phabricator/ $ ./bin/diviner generate --book <book>

### .book 文件

Diviner 书籍文档需要使用.book文件（JSON格式）进行配置，这个文件类似如下所示:

**example.book**

{

"name" : "example",

"title" : "Example Documentation",

"short" : "Example Docs",

"root" : ".",

"uri.source" : "http://example.com/diffusion/X/browse/master/%f$%l",

"rules" : {

"(\\.diviner$)" : "DivinerArticleAtomizer"

},

"exclude" : [

"(^externals/)",

"(^scripts/)",

"(^support/)"

],

"groups" : {

"forward" : {

"name" : "Doing Stuff"

},

"reverse" : {

"name" : "Undoing Stuff"

}

}

}

文件中的属性：

* name: 必须。要短, 且作为判断文档书籍的唯一标示。 这里会生成URI地址，所以不要在名字中使用特殊符号。好的名字例如： "example" 或 "libcabin"。
* root: 必须。根目录（相对于.book文件），文档将会在这个路径下产生。这里通常都会写成“../..”，来指定工程的根目录（例如，当.book文件在 project/src/docs/example.book下，填写“../..”后，文档将会在 project/目录下生成）。
* title: 可选。填写文档书籍的标题。这里有足够的空间来显示标题，并且标题需要对文档进行全面的描述。Good titles are things like好的文档标题，例如： "Example Documentation", 或 "libcabin Developer Documentation"。
* short: 可选。短标题，显示的空间也有限。如果为空，那么将使用完整标题。好的短标题，例如： "Example Docs" 或 "libcabin Dev Docs"。
* uri.source: 可选。可以通过连接到一个代码库浏览器，你可以很快的跳转到指定类或函数定义处。为了完成这个功能，它使用了一个指定得URI模式。通常，这个URI应该指向一个类似Diffusion的代码库浏览器。比如， "http://repobrowser.yourcompany.com/%f#%l"。你可以在URI中使用以下转换，在运行时，对应区域会进行替换:
  + - %f: 替换指定的文件名。
    - %l: 替换指定的行。
    - %%: 替换'%'字符。
* rules: 可选。一张使用正则表达式创建模糊类的映射表，将能控制生成器对每个文件的操作。如果为空，Diviner将会使用默认的规则集。例如，为映射添加密钥 "(\\.diviner$)" 到 "DivinerArticleAtomizer" 中，告诉Diviner分析每一个名字以 .diviner 结尾的文件。使用"article" 为模糊选项。
* exclude: 可选。 一列正则表达式，用来匹配排除文档生成路径。例如, 添加类似的表达式 "(^externals/)" 或 "(^vendor/)" 将会是Diviner忽略这些目录。
* groups: 可选。文档隶属的最高组织，应该在生成的时候进行书写。

## Drydock 使用教程

配置Drydock进行源码管理。

### 概述

NOTE: 是一项崭新的功能，还没有什么地方用到。

## Events 使用教程：安装 Event 监听器

使用Phabricator事件监听器自定义监听行为。

### 概述

Phabricator和Arcanist允许你自定义安装运行时事件监听器，可对某些事情做出反应（比如，修改了一个Maniphest任务，或者用户创建了一个Differential版本），还有自定义的执行文件记录、与其他系统同步，或者修改工作流。

这些监听器都是PHP的类，在你安装Phabricator或Arcanist时已经安装好了的，并且在Phabricator运行时对他们进行加载。他们在前期配置，需要的仅是简单开启或关闭，而对于对应时间的相应更直接、更强有力。

### 安装 Event 监听器 (Phabicator)

如何在Phabricator安装监听器， 可以参照以下步骤：

* 写一个监听器的类，作为 [PhutilEventListener](https://secure.phabricator.com/diviner/find/?book=libphutil&name=PhutilEventListener&type=class&jump=1)的扩展。
* 将它添加到libphutil库中，或者出那个键一个新库 (更详细的信息，可参见 [libphutil Libraries User Guide](https://secure.phabricator.com/diviner/find/?name=libphutil_Libraries_User_Guide&type=article&jump=1)）。
* 通过对load-libraries的配置，可以让Phabricator加载对应的库文件。
* 通过在events.listeners中添加类名，就可以在Phabricator中配置安装时间监听器了。

你可以通过监听在DarkConsole中已经注册的“事件表”，来验证你的监听器是够正确。它应该出现在“注册的时间监听器”的最前面。你也可以看到“事件页面”中发生的变化。 有关 DarkConsole的信息，可以参考[Using DarkConsole](https://secure.phabricator.com/diviner/find/?name=Using_DarkConsole&type=article&jump=1)。

### 安装 Event 监听器 (Arcanist)

在Arcanist上安装事件监听器，可以通过以下的几步：

* 写一个监听类，作为 [PhutilEventListener](https://secure.phabricator.com/diviner/find/?book=libphutil&name=PhutilEventListener&type=class&jump=1)的扩展。
* 将这个监听类添加入 libphutil 库中， 或者创建一个新库 (详细信息，可见 [libphutil Libraries User Guide](https://secure.phabricator.com/diviner/find/?name=libphutil_Libraries_User_Guide&type=article&jump=1)）
* 通过将监听类加载到Arcanist的配置（例如，.arcconfig，或用户/全局配置）中，让Phabricator来加载库。
* 通过在events.listeners中添加类名，就可以在 Arcanist中配置安装事件监听器了。

你可以使用arc --trace来验证你注册的监听器。你可以查看有有关你已经注册事件监听器的信息。

### 监听器实例

Phabricator 包含一个事件监听器实例， [PhabricatorExampleEventListener](https://secure.phabricator.com/diviner/find/?name=PhabricatorExampleEventListener&type=class&jump=1), 作为你一开始就可以使用的监听器，他将会在开发过程中给你很多帮助。可以通过 scripts/util/emit\_test\_event.php脚本来测试监听器，以监听一个事件。

正常运行这个脚本，你将看到如下的输出:

$ ./scripts/util/emit\_test\_event.php

Emitting event...

Done.

这是因为现在没有监听器对事件进行监听，所以当其改动时，什么都没有显示。你可以通过在events.listeners中配置，或者在命令行中跟--listen选项，添加一个监听器的实例:

$ ./scripts/util/emit\_test\_event.php --listen PhabricatorExampleEventListener

Installing 'PhabricatorExampleEventListener'...

Emitting event...

PhabricatorExampleEventListener got test event at 1341344566

Done.

现在，监听器已经安装，并且当测试事件被修改时，它显示了监听信息。

### 可用事件

你可以在 [PhabricatorEventType](https://secure.phabricator.com/diviner/find/?name=PhabricatorEventType&type=class&jump=1)中看到Phabricator支持的所有事件类型。

#### 全部事件

特定常数 PhutilEventType::TYPE\_ALL 将会使得监听器监听所有的事件。

通常情况下，有只想监听一些特定的时间。但是，如果你写了一个通用处理方式，你可以通过配置这个变量对所有的事件进行监听，而不是将每个事件都列举出来。

#### Arcanist事件

你可以在 [ArcanistEventType](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistEventType&type=class&jump=1).中看到Arcanist支持的所有事件类型。

所有 Arcanist 事件都能使用这个数据:

workflow  可见 [ArcanistBaseWorkflow](https://secure.phabricator.com/diviner/find/?book=arcanist&name=ArcanistBaseWorkflow&type=class&jump=1)。

#### Arcanist: Commit: Will Commit SVN

这个事件由 ArcanistEventType::TYPE\_COMMIT\_WILLCOMMITSVN常数来配置。

这个事件在svn commit执行前发生，并且允许你修改提交信息。

这个事件可用的数据有:

message 提交的信息。

#### Arcanist: Diff: Will Build Message

这个事件由 ArcanistEventType::TYPE\_DIFF\_WILLBUILDMESSAGE常数来配置。

这个事件在作为可编辑的信息发送给用户之前可让你知道。例如，在域中填写默认值。

这个事件可用的数据有:

fields 可被编译成消息的映射的域值。

#### Arcanist: Diff: Was Created

这个事件由 ArcanistEventType::TYPE\_DIFF\_WASCREATED常数来配置。

这个是将将在一个diff创建后被触发。这个事件现在只是用来对时间数据进行收集。这个事件没有可用的数据。

#### Arcanist: Revision: Will Create Revision

这个事件由 ArcanistEventType::TYPE\_REVISION\_WILLCREATEREVISION常数来配置。

这个事件在一个修订产生后触发。它允许你对域进行修改。比如，修改修订的标题。

这个事件可用的数据有:

specification 参数将会被Conduit的函数differential.createrevsion调用。.

#### Controller: Check Request

这个事件由 PhabricatorEventType::TYPE\_CONTROLLER\_CHECKREQUEST常数来配置。

这个事件将在控制器执行之前触发。这意味着要检查，指定用户是否在当且阶段使用应用程序。它可以检查用花是够执行了过多的操作，当他的IP地址被允许这样做，或者服务器负载能够相应这些请求。

这个事件可用的数据有:

request [AphrontRequest](https://secure.phabricator.com/diviner/find/?name=AphrontRequest&type=class&jump=1).类的对象。

controller 当前控制器的类名称。

你可以通过修改controller来修改当前的控制器。

#### Maniphest: Will Edit Task

这个事件由 PhabricatorEventType::TYPE\_MANIPHEST\_WILLEDITTASK常数来配置。

这个事件将在一个任务被修改之前触发，并且允许你回应或者再修改

这个事件可用的数据有:

task 在 [ManiphestTask](https://secure.phabricator.com/diviner/find/?name=ManiphestTask&type=class&jump=1) 章节已经修改过。

transactions 修改列表 ( [ManiphestTransaction](https://secure.phabricator.com/diviner/find/?name=ManiphestTransaction&type=class&jump=1)类中的对象） 已经被应用。

new 当一个任务创建完毕，需要一个布尔值来表示这个任务为新创建的任务。

mail 修改源来自于邮件，就是来自于 [PhabricatorMetaMTAReceivedMail](https://secure.phabricator.com/diviner/find/?name=PhabricatorMetaMTAReceivedMail&type=class&jump=1) 对象。

这就和下一个事件（Did edit task）类似了，不过是在修改发生之前触发。

#### Maniphest: Did Edit Task

这个事件由 PhabricatorEventType::TYPE\_MANIPHEST\_DIDEDITTASK常数来配置。

这个事件将在一个任务被修改以后触发，并且允许你对这个修改做出反应。

这个事件可用的数据有:

task 在 [ManiphestTask](https://secure.phabricator.com/diviner/find/?name=ManiphestTask&type=class&jump=1) 章节已经修改过。

transactions  修改列表 ( [ManiphestTransaction](https://secure.phabricator.com/diviner/find/?name=ManiphestTransaction&type=class&jump=1)类中的对象) 已经被应用。

new 当一个任务创建完毕，需要一个布尔值来表示这个任务为新创建的任务。

mail 修改源来自于邮件，就是来自于 [PhabricatorMetaMTAReceivedMail](https://secure.phabricator.com/diviner/find/?name=PhabricatorMetaMTAReceivedMail&type=class&jump=1) 对象。

这就和之前的事件类似了（will edit task），但是触发在修改完成之后。

#### Differential: Will Send Mail

这个事件由 PhabricatorEventType::TYPE\_DIFFERENTIAL\_WILLSENDMAIL常数来配置。

这个事件在Differential发送邮件前触发，并且允许你修改将要发送的信息。

这个事件可用的数据有：

mail 在 [PhabricatorMetaMTAMail](https://secure.phabricator.com/diviner/find/?name=PhabricatorMetaMTAMail&type=class&jump=1) 已经进行过修改。

#### Differential: Will Mark Generated

这个事件由 PhabricatorEventType::TYPE\_DIFFERENTIAL\_WILLMARKGENERATED常数来配置。

这个事件在Differential决定是否生成一个文件时触发（并且不需要回查的）。

这个事件可用的数据有：

corpus 文件的内容。

is\_generated 这是一个布尔值，来作为文件是否生成的标示。

#### Differential: Will Edit Revision

这个事件由 PhabricatorEventType::TYPE\_DIFFERENTIAL\_WILLEDITREVISION常数来配置。

这个事件在一个修订被修改前触发，并且允许你做出反应或者再修改。

这个事件可用的数据有：

revision 在 [DifferentialRevision](https://secure.phabricator.com/diviner/find/?name=DifferentialRevision&type=class&jump=1) 已经进行过修改。

new 当这个修订已经被创建，用一个布尔值来代表修订的状态。

这个事件类似于下一个事件（did edit revision），但是是在修改之前触发。

#### Differential: Did Edit Revision

这个事件由 PhabricatorEventType::TYPE\_DIFFERENTIAL\_DIDEDITREVISION常数来配置。

这个时间在一个修订修改后触发，并允许你对其做出反应。

这个事件可用的数据有：

revision 在 [DifferentialRevision](https://secure.phabricator.com/diviner/find/?name=DifferentialRevision&type=class&jump=1) 已经修改过。

new  当这个修订已经被创建，用一个布尔值来代表修订的状态。

这个时间类似于之前的事件（will edit revision），但是发生在修改完成之后。

#### Diffusion: Did Discover Commit

这个事件由 PhabricatorEventType::TYPE\_DIFFUSION\_DIDDISCOVERCOMMIT常数来配置。

当守护进程发现一个版本提交时，这个事件会被第一时间触发。这个事件在流水线中发生比较早，当时还没有任何提交可以使用。

这个事件可用的数据有：

commit 在 [PhabricatorRepositoryCommit](https://secure.phabricator.com/diviner/find/?name=PhabricatorRepositoryCommit&type=class&jump=1) 时，已经找到了。

repository 在 [PhabricatorRepository](https://secure.phabricator.com/diviner/find/?name=PhabricatorRepository&type=class&jump=1) 的提交已经覆盖到。

#### Diffusion: Lookup User

这个事件由 PhabricatorEventType::TYPE\_DIFFUSION\_LOOKUPUSER常数来配置。

这个事件在守护进程尝试使用一个Phabricator用户账号连接一个代码提交时触发。你可以监听这个操作，来提高提交版本和与之有关系的用户关联的精度。

在默认情况下，Phabricator将会根据用户名，真名，或邮件地址来对用户进行查找，但是这可能会匹配到错误的结果（比如，你的系统中有很多人用了同样的名字）或者匹配失败（比如，有人修改了他的邮件地址）。监听这个事件，允许你中断查找和对查找源进行补充。

这个事件可用的数据有:

commit 在 [PhabricatorRepositoryCommit](https://secure.phabricator.com/diviner/find/?name=PhabricatorRepositoryCommit&type=class&jump=1) 这个数据已经被查找过。

query 以作者或者提交者的名字进行查找。这就如同，有一个账号为“Abraham Lincoln <alincoln@logcabin.example.com>”，但是这是来自于提交的元数据中，所以这个格式可能不能很好的匹配到。

result 当前查找的结果（Phabricator猜测最好的结果是，与用户的ID相关联的用户名在“队列”中）。为了让另一个结果替换之前的结果，新结果中当前用户的ID需要在你的监听事件中。

使用 [PhutilEmailAddress](https://secure.phabricator.com/diviner/find/?book=libphutil&name=PhutilEmailAddress&type=class&jump=1) 可能对分析队列有帮助。

#### Edge: Will Edit Edges

NOTE: 边缘事件在核内是低级别事件。这让实现监听器，并去监听这些事件的难度十分大。

这个事件由 PhabricatorEventType::TYPE\_EDGE\_WILLEDITEDGES常数来配置。

这个事件在 [PhabricatorEdgeEditor](https://secure.phabricator.com/diviner/find/?name=PhabricatorEdgeEditor&type=class&jump=1) 做出一个边缘修改前触发。

这个事件可用的数据有:

id 修改操作的标示符。

add 一个字典列表，其中每个字典都代表着一个新的边缘。

rem 一个字典列表，其中每一个字典都代表着一个被删除的边缘。

这就和下一个事件（did edit edges）类似了，不过发生在修改开始前。

#### Edge: Did Edit Edges

这个事件由 PhabricatorEventType::TYPE\_EDGE\_DIDEDITEDGES常数来配置。

这个事件在 [PhabricatorEdgeEditor](https://secure.phabricator.com/diviner/find/?name=PhabricatorEdgeEditor&type=class&jump=1) 做出一个修改后，在这个提交被处理前触发。

这个事件可用的数据有:

id  修改操作的标示符。这里的ID和“will edit edit”事件中的ID是同一个。

add 一个字典列表，其中每个字典都代表着一个新的边缘。

rem 一个字典列表，其中每一个字典都代表着一个被删除的边缘。

这和前一个事件（will edit edges）类似，不过发生在修改完成后。

#### Search: Did Update Index

这个事件由 PhabricatorEventType::TYPE\_SEARCH\_DIDUPDATEINDEX常数来配置。

这个事件由搜索应用启动索引搜索器时触发，而后去索引这个文档。它允许你在其他系统中发布索引到的结果。

注意更新完后发生这样的情况：你不能避免或者修改指定更新版本。此外，索引的查找可能会在后台进行，所以这个事件可能会比实时更新更重要。如果你要保证这个事件运行，需要使用别的事件来支持。

最终，这个事件不仅会在单一版本更新时触发。例如，如果你搜索重构索引，这个事件将被触发，不过实际上搜索对象并没有发生变化。

所以，正确的使用监听器的方式为：

更新二次搜索索引。

不好的使用方式为:

修改对象或者文件。

使用带有副作用的当时，例如发送邮件。

这个事件可用的数据有:

phid  更新对象的PHID。

object 被更新的对象(比如一个 [ManiphesTask](https://secure.phabricator.com/diviner/find/?name=ManiphesTask&type=class&jump=1))。

document  [PhabricatorSearchAbstractDocument](https://secure.phabricator.com/diviner/find/?name=PhabricatorSearchAbstractDocument&type=class&jump=1) 已经索引化。这里包含一个对象的抽象表示，并且可用于填充二次索引。因为它提供了一个统一的API。

#### Test: Did Run Test

这个事件由 PhabricatorEventType::TYPE\_TEST\_DIDRUNTEST常数来配置。

这是对监听器进行测试的事件。详情见上文。

#### UI: Did Render Actions

这个事件由 PhabricatorEventType::TYPE\_UI\_DIDRENDERACTIONS常数来配置。

这个事件在 [PhabricatorActionListView](https://secure.phabricator.com/diviner/find/?name=PhabricatorActionListView&type=class&jump=1) 通过UI构建后触发。它允许在你的应用程序支持的情况下，添加新的行为，比如：传真一个对象。

这个事件可用的数据有:

object 那些被渲染的对象。

actions 当前可用操作列表。

NOTE: 这个事件还不是很稳定，并且会改变主题。

### 调试监听器

如果你监听器存在有问题，你可以尝试如下步骤：

如果你获取了“监听器类找不到”的错误，需要确定你把这个类提那几到libphutil库中，以及在Phabricator中使用load-libraries加载了这个库。

确认你的监听器已经注册。你的监听器名应该出现在 DarkConsole的“事件表”中。如果没有出现，那么你可能忘记将它添加入events.listeners中。

确认register()函数在调用listen()时，监听了正确的对象。如果没有监听到你感兴趣的事件，那么监听器是什么都不会返回的。

确认你要监听的事件已经发生。如果在普通的页面中发生，那么他们将会出现在 DarkConsole的“事件表”中。如果他们在后台（POST）发生，你需要在源码中添加一个phlog()函数在你需要监听的事件中，并检查一下你的错误信息。

如果你在调用的时候添加了附带信息的phlog()函数，就可以去检查这条信息是否在错误信息中。

你可以尝试去监听 PhutilEventType::TYPE\_ALL ，而不是监听单一事件类型，来排除是否是因为事件类型引起的监听失败/错误。

你可以修改 emit\_test\_event.php 脚本中的事件类型，来适当的测试你要监听的事件。你可能不得不使用假数据，但这个简单的测试至少能让你确定一些基本问题。

For scripts对于脚本来说，你可以使用--trace 跟在命令行之后，来查看哪些事件被修改了，以及有多少处理操作，正在被监听。

### 下一步

可以继续如下章节:

可以看一下 [PhabricatorExampleEventListener](https://secure.phabricator.com/diviner/find/?name=PhabricatorExampleEventListener&type=class&jump=1)

如何构建一个库， [libphutil Libraries User Guide](https://secure.phabricator.com/diviner/find/?name=libphutil_Libraries_User_Guide&type=article&jump=1)

## Herald 使用教程

使用Herald规则对你关心区域的修改进行提示。

### 概述

Herald允许你对特定对象（比如，Differential修订和提交）的更新或创建，进行自定义处理规则。例如，当有某人的提交，对你感兴趣的文件产生了影响，你就会得到相应的通知，即使他们没有把你作为一个回查者添加入工程。

Herald在小型组织中用处不大（大家都知道彼此的工作），但是当组织规模扩大后，这个应用就很有用了。当有很多库呈激活且跟踪状态，Herald允许你去过滤信息，只对你感兴趣的区域修改进行提示。

### 全局和个人规则

你可以创建两种Herald规则，全局规则和个人规则：

**个人规则（Personal Rules）** 这些规则只属于你个人拥有，也就是只能对你自己起作用。你可以选择修改或删除一些个人规则，但是他们的行为你限于你在抄送中添加的，或者你订阅的。

**全局规则（Global Rules）** 这些规则将应用于系统中所有的事情。每个人都可以对全局规则进行编辑和删除，并且他们可以做一些操作，包括对影响项目和邮件列表的操作。

总的想法是要避免个人对规则的控制，否则将会影响到资源的共享。所以，如果规则需要更新，即使创建者在度假，这都不会有什么大问题。

### 规则, 条件和操作

使用Herald最好的方式，就是将其作为一种类似邮件的规则，你可以在大部分的客户端上创建一些规则，基于“接收方”，“主题”等进行组织。Herald工作原理与邮件相似，不同之处在于它是在Phabricator系统中处理对象（比如，修订和版本提交）。

每当一个对象创建或更新，Herald规则都会指定，并且去匹配那些指定的规则。

要创建一个新的Herald规则，选择一种你想要执行的类型（比如，修改为Differerntial修订，或提交版本），并且可以设置一个条件列表。比图，你可能需要添加一个条件“Author is alincoln(Abraham Lincoln)”来跟踪alincoln所作的每一件事情。最后，设置一个操作列表，当条件匹配时会采取其中某一个操作，例如添加你自己到抄送列表。

现在，当alincoln创建了一个修订，你将自动被添加入抄送列表中，并且看到alincoln做了什么。

### 可用的操作

Herald规则可以使用如下的操作。注意这些操作只能使用在全局规则中，其他的操作都是来源于个人规则。此外，并不是每个操作都可对所有对象类型适用（例如，你不能根据一个Differential修订触发一个审核请求）。

**添加入抄送列表（Add CC）**: 在抄送列表中添加一个用户或对象。对于个人规则，你只能把自己添加入抄送列表中。

**从抄送列表中移除（Remove CC）**: 从抄送列表中移除一个用户或者对象。对于个人规则，你只能将你自己从抄送列表中移除。

**发送一份邮件到（Send an Email to）**: 发送一份邮件，就没必要经去订阅其他人的更新。对于个人规则，你只能给你自己发邮件。

**触发一个审核请求（Trigger an Audit）**: 对于提交，触发一个审核请求，发送给工程或用户。对于个人规则，你只能向自己发送审核请求。

**标记标示符（Mark with flag）**: 标记对象将在之后回查。 这个操作只能应用于个人规则。如果一个对象已经拥有一个标记，这个操作将不会添加其他标记。

**无操作（Do Nothing）**: 什么都不做。这可以作为临时关闭规则的一种手段，或者创建另一条Herald条件规则。

### 测试规则

当你创建了一条规则，然后就可以使用“Test Console”去测试它的输出。键入一个修订或者一次提交，Herald将会对指定对象进行干运行，然后显示哪些规则在更新后被匹配到。干运行可通过测试命令行执行，不需要在多做什么操作。

### 高级 Herald规则

一些Herald特性是十分复杂的:

**匹配正则（matches regexp pair）**: 对于Differential 修订, 你可以设置一个条件，例如“Any changed file content matches regexp pair...”。这将允许你以JSON格式指定两个正则表达式。 第一个将会用来匹配变化了的文件名;第二个将会对内容进行匹配。例如，你想要匹配一个修订，这个修订中添加/删除调用“muffinize”函数， but only in JS files在JS文件中，你只需要这样设置 ["/\\.js$/", "/muffinize/"] or similar或类似的方式即可。

**其他的规则（Another Herald rule）**: 你可以创建Herald规则，需要依赖于其他的规则。如果你使用比“all”和“any”更复杂的谓语来表达，或者存在一个与其他规则共享的公共条件集，设置这种规则将会有很大的帮助。如果一个规则只能用于一组条件，你可以将其操作设置为“无操作”。

## Jump Nav 使用教程

jump nav的参考指令。

### 概述

jump nav 提供了一个高速的方法对工具和对象进行导航: 只需输入命令，然后获取结果就行。

### 支持的指令

**help** - 跳转到对应文件。

**T** - 跳转到 Maniphest。

**T123** - 跳转到 Maniphest 123号任务。

**D** - 跳转到 Differential。

**D123** - 跳转到 Differential 123号修订。

**r** - 跳转到 Diffusion。

**rXYZ** - 跳转到Diffusion XYZ代码库。

**rXYZabcdef** - 跳转到 Diffusion rXYZabcdef提交版本。

**r <name>** - 通过名字来搜索代码库。

**u** - 跳转到People。

**u username** - 跳转到某个用户的个人信息页面。

**p** - 跳转到 Project。

**p Some Project** - 跳转到指定 Project。

**s SymbolName** - 跳转到指定Symbol。

**task: (new title)** - 跳转到任务创建页面，可预填标题。

**(default)** - 搜索输入的信息。

## libphutil 库 使用教程

如何创建和管理libphutil库。

### 概述

libphutil内包含一个系统库，由PHP类和模块函数组成。一些扩展和自定义的Arcanist和Phabricator功能需要提供对应的libphutil库来提供相应的功能。

例如，如果你要需要使用自定义存储引擎来存储一些文件，你需要写一个能与引擎互动的类，并且需要告诉Phabricator加载它。

一般来说，你可以执行如下的安装步骤：

创建一个新目录

用arc liberate来初始化库

给Phabricator添加必要的依赖

在你的Phabricator配置中，或者.arcconfig文件添加对应的库，在Phabricator运行时及时加载。

而后，为了添加新的代码，你可以：

写或者更新类

通过arc liberate来更新库的元数据

### 创建一个新库

要 **创建一个新的libphutil库：**

$ mkdir libcustom/

$ cd libcustom/

libcustom/ $ arc liberate src/

现在，你就会看到如下的提示信息：

No library currently exists at that path...

The directory '/some/path/libcustom/src' does not exist.

Do you want to create it? [y/N] y

Creating new libphutil library in '/some/path/libcustom/src'.

Choose a name for the new library.

What do you want to name this library?

创建一个库名（在这个例子中，库名为“libcustiom”），这里要给库做一些初始化处理：

Writing '\_\_phutil\_library\_init\_\_.php' to

'/some/path/libcustom/src/\_\_phutil\_library\_init\_\_.php'...

Using library root at 'src'...

Mapping library...

Verifying library...

Finalizing library map...

OKAY Library updated.

这将会写入3个文件:

src/.phutil\_module\_cache 这是一个缓存文件，当使用 arc liberate会更快的对库文件进行更新。你可以随时将它删除。如果你要通过版本控制检查你的库文件，你可以对这个文件添加进忽略规则（比如，.gitignore）。

src/\_\_phutil\_library\_init\_\_.php 这里记录着一些库的名字，并且告知libphutil他们是已存在的库。

src/\_\_phutil\_library\_map\_\_.php 这里记录着库中所有的符号映射（函数和类），这里允许在使用arc liberate 时自动加载运行时的库和静态依赖的库。

### 通过Phabricator连接

如果你没有在Phabricator中使用这个库（例如，你只使用Arcanist，或者通过libphutil构建别的东西）你可以忽略这步。

但是，如果你打算在Phabricator中使用这个库，你需要通过创建一个.arcconfig文件（需要指定）并在文件中定义它们指定他们依赖。例如，你可能需要在libcustom/.arcconfig中指定这个文件路径：

{

"project\_id" : "libcustom",

"load" : [

"phabricator/src/"

]

}

如何创建一个.arcconfig文件，可以参考 [Arcanist User Guide: Configuring a New Project](https://secure.phabricator.com/diviner/find/?name=Arcanist_User_Guide%3A_Configuring_a_New_Project&type=article&jump=1)章节。通常，执行arc liberate时，会在Phabricator中执行静态分析，去寻找符号信息进行连接 。

NOTE: 如果Phabricator没有定位到你的库，你需要指定一个指向phabricator目录的路径。

你不需要为依赖arcanist或libphuti的库进行声明，当arc liberate运行时会自动加载他们。

然后，通过load-libraries变量，来修改Phabricator配置，告诉Phabricator在运行时加载这些库：

...

'load-libraries' => array(

'libcustom' => 'libcustom/src/',

),

...

现在, Phabricator 就可以自动加载你自定义库中的类了。

### 类写入

为了写一些类，创建一个新的模块，并且把代码放在里面：

libcustom/ $ mkdir src/example/

libcustom/ $ nano src/example/ExampleClass.php # Edit some code.

现在，你可以执行arc liberate来生成对应库的静态映射表：

libcustom/ $ arc liberate src/

这个操作将会重新生成库文件的静态映射表。

## 你可以扩展和引用的库

libphutil，Arcanist和Phabricator是严格的扩展类，并且方法和属性都是可见的。大多数类都被标记为final，且方法都要求可见（protected或private）。这种严格要求的目的，在你清楚哪些可以安全的扩展，访问，以及调用之后，你的代码需要作为上游修改能保持其功能继续工作。

当你开发要和libphtil，Arcanist和Phabricator一起工作的库，你应该对方法实现，属性的可视性和扩展标记为@stable的类十分谨慎。他们应该在在文档中明确标记(例如: [AbstractDirectedGraph](https://secure.phabricator.com/diviner/find/?book=libphutil&name=AbstractDirectedGraph&type=class&jump=1))。 这些对外接口是用于扩展的。

如果你要扩展一个类，但是这个类没有被标记为@stable，这里给你提供一些办法:

好办法: 如果可能，使用组内存在的，而非为了你需要的特性去重新构建。

好办法: 查阅文档，寻找一个更好的办法去完成你想要做的事。

好办法: 让我们了解你需要什么样的实例，我们可以将类群树做的更灵活，或可配置，或这可以给你指条明路，再或为什么你的功能完成不了。

不推荐的办法: 发给我们一个补丁，移除“final”标记（或把“protected”或“private”改为“public”）。我们同城不会允许这些打上这些补丁，除非有特别好的理由证明现在的做法是错误的。

不推荐的办法: 创建一个点对点的本地分支，并且在你的代码副本中移除“final”标记。这将会使你以后升级代码变的十分困难。

不推荐的办法: 使用映射，违背关键字可视化。

## Notifications 使用教程：安装和配置

### 概述

在默认情况下，Phabricator通过邮件或内置应用提醒事件，来传递信息（比如，用户创建了一些任务; 或在回查时，对代码进行了评论）。

Phabricator 也可以配置为在实时处理时进行通知/提醒，当有事件发生或一个对象更新，就可以通过弹出一个消息窗口的形式（可以在任意浏览器中打开）。

开启实时提醒：

将 notification.enabled 置为“开启”状态

运行提示服务器（下文会有介绍）

本节将会介绍一些处理细节。

### 运行 Aphlict 服务器

Phabricator 使用Aphlict（Node.js服务器）实现实时提醒。为了让其运行起来:

安装 node.js.

运行 bin/aphlict (这个脚本需要使用root权限运行).

为了让Aphlict服务器工作，需要监听843和22280端口。你可以通过notification.client-uri变量来配置第二个端口（22280端口），但是843端口已经被闪存使用，所以不能改变。特殊是，当你运行EC2服务，你需要打开这两个端口在服务器上的安全组配置。

你可能需要调整如下设置:

notification.client-uri 面向外部主机和端口，为了即使听取通告，需要和浏览器相连接。

notification.server-uri 面向内部主机和端口，连接Phabricator，以便发送通告。

notification.log 确定服务器日志文件的位置。

notification.user 非root用户可访问特殊端口，这里权限会下放并根据用户名进行绑定。

notification.pid 当aphlict服务器重启时，Pidfile文件可用来停止任何运行的服务器。

在多数例子中，它是默认使用的，除非你在notification.user中设置了合法的系统用户，而后Aphlict就不是以root用户运行了。

#### 确认服务器状态

访问 /notification/status/ 来验证服务器是否是可操作的。当服务器工作时，你应该能看到一张显示属性的表，例如，时间和连接/消息数量。如果服务器停止工作，你应该会看到错误信息。

#### 测试服务器

最简单的测试服务器的方法就是，让两个用户登陆，并提交相同的Maniphest任务或者Differential修订。他们将会在浏览器上看到有关于他们操作的提示。

NOTE: 这个方案有时会十分麻烦。有些时候，会有更好的测试工具。

### 调试服务器错误

你可以在前端运行aphlict:

phabricator/ $ sudo ./bin/aphlict --foreground

你可以以命令行的方式运行 support/aphlict/client/aphlict\_test\_client.php 连接到Aphlict 服务器。客户端将会在标准输出端看到相应的信息输出。

你可以设置 notification.debug 变量，从而获取更多的输出信息。

服务器也可以生成一个日志文件，默认在/var/log/aphlict.log中。你可以通过修改notification.log变量来修改日子存储的位置。这些日志可能会包含对解决问题很有帮助的信息。

## Owners Tool 使用教程

使用定义和/或监控你关心的代码段。

### 工具包

工具允许你通过指定一组路径，来定义一个代码包。这个包可以用来监视这些路径。例如，它可以使用Herald规则和“相关提交”的特性（见下文）。

### 相关提交

一旦工具包定义完成，之后的所有提交/修改，只要与工具包中的路径匹配的，都会在工具包中标记为“相关提交”。

### 需要注意的提交

工具可以让包的用拥有这监视那些他们关心的提交。如果“审核机制”在一个包中处于开启状态，一个相关的代码提交将会被标记为“需要注意”，当：

没有一个工具包去撰写或回查。

在修订中没有找到提交。

提交版本的作者无法识别。

作者或这回查者指出提交版本与Differential修订有所不同。

当对某个版本比较担心，在工具包允许或指定这个版本，可通过点击“Audit Status”直接连接。

## Phame 使用教程

在这过程中，你可以和别人分享你的想法个情感。

### 概述

Phame是一个简易的博客平台。你可以撰写一些对你可见的文章。 之后，你可以发布这些文章，可以设置文章的访问权限。你还可以添加文章到博客来，增添你的文章覆盖范围。

总而言之, Phame的目的是帮助个人传递他们的思想。因此，有针对行的设计策略偏向于个人，而不是整个集体。

### 草稿

草稿是绝对私人的。

### 帖子

发帖功能针对于那些能访问到你博客的人。

### 博客

博客可以汇集帖子。每个博客都有相关的元数据，比如，名字、描述和把帖子添加到博客的人。每个博主可以编辑博客的元数据，或者将博客完全删除。

现在，博客将对外网访问是十分有效的，例如：

blog.yourcompany.com

通过配置相关的选项，来修改你的DNS权限和Phabticator实例。

NOTE:从一个给定博客中删除一个比较大型的博客，不以为着那些帖子也和博客一起被删除了。相反，这个操作只是消除了对元数据的编辑，以及发布博客的能力。

### 评论工具

Phame 提供源于Facebook和Disqus的评论工具。想要使用这些工具，需要Phabricator的管理员将这个实例进行开启。

### 下一步

Phame 是一个较新且十分基本的功能。请给我们以反馈，以便我们完善这个功能！ 参见 [Give Feedback! Get Support!](https://secure.phabricator.com/diviner/find/?name=Give_Feedback!_Get_Support!&type=article&jump=1)。

## Phriction 使用教程

对你的使用（文明？）历史进行详细记录。

### 概述

Phriction是一个小型wiki。你可以编辑页面，并且你所写东西将会保留在那里。其他人将会在之后看到它。

NOTE: Phriction 是一个较新且十分基本的功能。请给我们以反馈，以便我们完善这个功能！ 参见 [Give Feedback! Get Support!](https://secure.phabricator.com/diviner/find/?name=Give_Feedback!_Get_Support!&type=article&jump=1)。

### Remarkup 参考

介绍如何写文本主体; 这是你的声音，所以你就可以在争论中获胜。

### 概述

Phabricator 使用了一种轻量级标记语言，叫做“Remarkup”，其他类似的标记语言有Markdown和Wiki markup。

本章将会阐述这些文本格式是如何在Remarkup中使用的。

### 快速参考

所有语法解释都在下文中，但是这个是个对于Remarkup快速的文本格式教程。

这些都是内联样式，可以应用于大多数文本:

\*\*bold\*\* //italic// ##monospaced## `monospaced` ~~deleted~~ \_\_underlined\_\_

D123 T123 rX123 # Link to Objects

{D123} {T123} # Link to Objects (Full Name)

{F123} # Embed Images

@username # Mention a user

[[wiki page]] # Link to Phriction

[[wiki page | name]] # Named link to Phriction

http://xyz/ # Link to web

[[http://xyz/ | name]] # Named link to web

[name](http://xyz/) # Alternate Link

这些块的风格，必须从空格周围的文本进行分离:

= Large Header =

== Smaller Header ==

Also a Large Header

===================

Also a Smaller Header

---------------------

> Quoted Text

Use "- " or "\* " for bulleted lists, and "# " for numbered lists.

Use ``` or indent two spaces for code.

Use %%% for a literal block.

Use | ... | ... for tables.

### 基本风格

基本的 **文本风格** 如下所示:

\*\*bold text\*\*

//italic text//

##monospaced text##

`monospaced text`

~~deleted text~~

\_\_underlined text\_\_

这其中有 **加粗文本**, *斜体文本*, 等宽文本, 删除文本, 和 有下划线的文本。

### 布局

让 **头部** 类似如下格式:

= Large Header =

== Smaller Header ==

===== Very Small Header =====

Alternate Large Header

======================

Alternate Smaller Header

------------------------

你可以有选择的省略尾部的 = 。 -- 以下的情况是等价的:

== Smaller Header ==

== Smaller Header

这里就有了文本文档中的标题。在标题之前和之后，是否都有一个空格。

**列表** 的起始为一个“-”或“\*”：

- milk

- eggs

- bread

\* duck

\* duck

\* goose

这将会有如下显示:

milk

eggs

bread

(注意你需要在“-”或“\*”后面跟上一个空格)

你可以使用 "#"来替代 "-" 或 "\*"，给列表前加序号:

1. Articuno
2. Zapdos
3. Moltres

你也可以嵌套进列表：

- Body

- Head

- Arm

- Elbow

- Hand

# Thumb

# Index

# Middle

# Ring

# Pinkie

- Leg

- Knee

- Foot

...就会产生如下显示:

* Body
* Head
* Arm
  + Elbow
  + Hand
    1. Thumb
    2. Index
    3. Middle
    4. Ring
    5. Pinkie
* Leg
  + Knee
  + Foot

如果你喜欢，你可以使用多个字符缩进列表显示缩进的深度，就像这样:

- Tree

-- Branch

--- Twig

我们期望有如下的显示:

* Tree
  + Branch
    - Twig

让代码块缩进两个空格:

f(x, y);

你也可以使用三个引号将代码块包围:

```f(x, y);

g(f);```

你可以通过“lang=xxx”来指定一种语言的语法高亮:

lang=html

<a href="#">...</a>

如果可用，这将使这一块指定语言的代码产生高亮（在大多数情况下，这意味你需要去配置Pygments）：

<a href="#">...</a>

你可以使用"COUNTEREXAMPLE"标题作为文件的头部，来显示代码块是有问题的，且不可复制的：

COUNTEREXAMPLE

function f() {

global $$variable\_variable;

}

这段代码将产生如下效果：

function f() {

global $$variable\_variable;

}

你可以使用line=N来限制垂直范围上的尺寸，并且可以用name=some\_name.ext来给这个文件命名。以下就是一个例子：

lang=html, name=example.html, lines=12, counterexample

...

...这段代码将产生如下效果:

**example.html**

<p>Apple</p>

<p>Apricot</p>

<p>Avocado</p>

<p>Banana</p>

<p>Bilberry</p>

<p>Blackberry</p>

<p>Blackcurrant</p>

<p>Blueberry</p>

<p>Currant</p>

<p>Cherry</p>

<p>Cherimoya</p>

<p>Clementine</p>

<p>Date</p>

<p>Damson</p>

<p>Durian</p>

<p>Eggplant</p>

<p>Elderberry</p>

<p>Feijoa</p>

<p>Gooseberry</p>

<p>Grape</p>

<p>Grapefruit</p>

<p>Guava</p>

<p>Huckleberry</p>

<p>Jackfruit</p>

<p>Jambul</p>

<p>Kiwi fruit</p>

<p>Kumquat</p>

<p>Legume</p>

<p>Lemon</p>

<p>Lime</p>

<p>Lychee</p>

<p>Mandarine</p>

<p>Mango</p>

<p>Mangostine</p>

<p>Melon</p>

你也可以使用 "NOTE:", "WARNING:", 或"IMPORTANT:" 等关键字来标记出比较重要的内容：

NOTE: Best practices in proton pack operation include not crossing the streams.

WARNING: Crossing the streams can result in total protonic reversal!

IMPORTANT: Don't cross the streams!

你也可以使用"(NOTE)", "(WARNING)", 或 "(IMPORTANT)"来达到相同的效果。

### 连接 URIs

URIs链接会自动连接到: [http://phabricator.org/](http://phabricator.org/#_blank)

如果你拥有一个URI，如同<http://comma.org/>,，你可以使用尖括号将他们括起来，如下所示：

<http://comma.org/,>

这会使URI分析器强制对改该地址进行分析。

你也可以在文本中插入超链接字符串。这对于Phabricator内部的地址或互联网中的地址都是可行的：

[[/herald/transcript/ | Herald Transcripts]]

[[http://www.boring-legal-documents.com/ | exciting legal documents]]

以下的链接样式也是支持的：

[Toil](http://www.trouble.com)

### 连接到对象

你可以通过之前提到的对象命名链接到特定的Differential修订 , Diffusion提交版本和Maniphest 任务：

D123 # Link to Differential revision D123

rX123 # Link to SVN commit 123 from the "X" repository

rXaf3192cd5 # Link to Git commit "af3192cd5..." from the "X" repository.

# You must specify at least 7 characters of the hash.

T123 # Link to Maniphest task T123

你也可以直接链接到Maniphest 和Differential中的指定评论:

T123#4 # Link to comment #4 of T123

### 内嵌对象

You can also generate full-name references to some objects by using braces:

{D123} # Link to Differential revision D123 with the full name

{T123} # Link to Maniphest task T123 with the full name

这些引用也将显示一个对象变化时的状态（比如，一个任务或者一个修订被关闭了）。有些类型的对象，还支持更加丰富的嵌入形式。

#### 内嵌测试 (Pholio)

你可以通过用大括号来内嵌一个Pholio测试：

{M123}

默认情况下测试集中的1/4幅图片将被展示。这个行为可以通过图像选来复写。你可以提供这个图片选项或者更多图像的ID号，来让他们显示出来。

你可以使用如下的方式将图片显示出来:

{M123, image=12345}

{M123, image=12345 & 6789}

#### 内嵌粘贴

你可以使用大括号内嵌一个粘贴：

{P123}

你可以调整内置的高度，通过lines选项：

{P123, lines=15}

你也可以让特定行产生高亮，通过highlight选项：

{P123, highlight=15}

{P123, highlight="23-25, 31"}

#### 内嵌图片

你可以通过大括号内嵌图片或者其他文件：

{F123}

在多数接口中，你可以使用拖拽的方式从本地插入文本区域进行上传，然后进行引用。

一些浏览器（例如，Chrome）支持通过剪切板把图片粘如文本区域的形式，上传图片数据。

你可以用以下的方式来设置显示选项：

{F123, layout=left, float, size=full}

这里可用的选项有：

**布局（layout）** 左对齐 (默认设置), 居中, 有对齐, 等宽, 链接 (提供一个链接，而不是一个图片的缩放图)。

**悬浮（float）** 如果布局设置为向左或向右对齐，图片将让文字环绕着它。

**缩放比例（size）** 缩放(默认设置)，原始比例。

**名字（name）** 和layout=link或不直接链接图片，这里使用图片名直接使用文本进行链接。

**宽度（width）** 指定图像宽度。

**高度（height）** 指定图像高度。

#### 内嵌倒计时器

你可以使用括号来内嵌一个倒计时器：

{C123}

### 引用文本

可以使用 ">"对文本进行引用:

> This is quoted text.

显示将会如下所示：

*This is quoted text.*

### 内嵌多媒体类型

如果你配置了这个表示，你就可以直接在文本中内嵌多媒体类型：

**remarkup.enable-embedded-youtube**:允许你直接从粘贴YouTube的视频，以及支持他们使用内联渲染。

这个选项在默认情况下是关闭的，因为它对安全性有一定的影响。在开启这个选项之前，可以在default.conf.php了解这个选项的详情。

### 图像宏

你可以上传图像宏(Stuff -> Macro) ，可以直接使用指定的图片来替换文本字符串。.例如，你可以上传一个跳舞香蕉的图片，并创建一个名为“peanutbutterjellytime”的宏，之后你在文本区域可以使用这个字符串来显示这个跳舞香蕉的图片。

### 模块

你也可以在模块的上下文中使用图像宏。 例如，如果你有一个图像宏名为“grumpy”，你可以通过一下代码段来创建一个模块：

{meme, src = grumpy, above = toptextgoeshere, below = bottomtextgoeshere}

在默认情况下，前端文字模块使用的字体为tuffy.ttf。不过使用impact.ttf字体感觉显示的更真实，你只需要简单的在Phabricator中对前端字体进行替换即可，字体文件位置在/resources/font/下。如果Remarkup检测到impact.ttf，它将会自动使用它。

### 提醒用户

在Differential 和 Maniphest中，你可以通过如下写法提及（并通知）其他用户：

@username

当你上传你的评论，这个用户就会被加入抄送列表。

### Phriction 文档

You can link to Phriction documents with a name or path:

Make sure you sign and date your [[legal/Letter of Marque and Reprisal]]!

With a pipe (|), you can retitle the link. Use this to mislead your opponents:

Check out these [[legal/boring\_documents/ | exciting legal documents]]!

### 文字块

表示文本中的文字块，可使用 "%%%":

%%%Text that won't be processed by remarkup

[[http://www.example.com | example]]

%%%

Remarkup将不会处理在文字块中的内容 (比如，保留空格)。

### 表格

Remarkup支持简单的表格语法。例如:

| Fruit | Color | Price | Peel?

| ----- | ----- | ----- | -----

| Apple | red | `$0.93` | no

| Banana | yellow | `$0.19` | \*\*YES\*\*

...产生如下效果:

|  |  |  |  |
| --- | --- | --- | --- |
| **Fruit** | **Color** | **Price** | **Peel?** |
| Apple | red | $0.93 | no |
| Banana | yellow | $0.19 | **YES** |

Remarkup 也支持简化的HTML表格语法。例如：

<table>

<tr>

<th>Fruit</th>

<th>Color</th>

<th>Price</th>

<th>Peel?</th>

</tr>

<tr>

<td>Apple</td>

<td>red</td>

<td>`$0.93`</td>

<td>no</td>

</tr>

<tr>

<td>Banana</td>

<td>yellow</td>

<td>`$0.19`</td>

<td>\*\*YES\*\*</td>

</tr>

</table>

...将会有如下效果:

|  |  |  |  |
| --- | --- | --- | --- |
| **Fruit** | **Color** | **Price** | **Peel?** |
| Apple | red | $0.93 | no |
| Banana | yellow | $0.19 | **YES** |

在使用这个语法时需要注意的是：

* 你的标签必须是使用适当的。
* 你的标签不能包含属性（例如:<td>是没有问题的，不过 <td style=”...”>就不允许了）。
* 你可以在表格单元中使用其他Remarkup规则 (例如 **粗体**, 斜体, 等等)。

### 全屏模式

Remarkup编辑器也提供全屏模式。这会让修改大块文本变得容易，或提高注意力，避免无心之举。你可以通过点击相应的按钮，或者使用ESC键退出全屏模式。

## Slowvote 使用教程

详细解决争议的手册。

### 概述

Slowvote是一个简单的轮询应用，可以提出诸如“我们应该在哪里订购披萨？”，“狗是不是比猫好？”和“我们公司的最佳战略方向在哪里？”。可以通过轻量化和结构化的方式从人们那里收集来必要的反馈。

### Plurality vs. Approval

当你创建了一个轮询，你需要选择 **plurality** 或 **approval** 投票的模式。

plurality模式中, 用户只能选择一个选项。需要在众多选项中，选择自己认为最好的那个。然后，投票最多的那中方式胜出。

approval模式中, 用户可以有多个选择。需要在众多选项中，复选自己比较支持的选项。最终胜出的选项，是被大多数人接受的。

## 使用教程: 账户角色

介绍系统中不同的角色。

### 概述

当你创建了一个账户，你就可以在角色中选择一种，"Administrator", "Disabled" 或 "System Agent"。本节就来说明一下设置各个角色的用意所在。

### 管理员（Administrators）

**管理员** 就是添加了特殊功能的普通用户。他们可以访问一些普通用户访问不到的工具和工作流，可以在Phabricator中去调试和配置这些工具和工作流。例如，他们能访问：

**账户管理（Account Management）**: 管理员可以是添加、禁用，以及管理用户账户。管理员也可以创建和编辑账户，有也可以访问系统日志。

**代码库（Repositories）**:管理员可以对库进行配置。普通用户是不允许的，因为这个操作通常需要专业化，并十分复杂的配置。

管理员在一些工具中，拥有一些特别的功能。当你在管理界面是，你会看到菜单栏是红色的。

管理员并不能完全控制系统。管理员也不能以其他用户登陆。管理员也不能对旁路对象有私下政策。

NOTE: 管理员现在可以通过Conduit代表其他用户。这将会在未来被改变。

### 系统代理（System Agents）

**系统代理** 和脚本需要与系统接口相连，普通用户做不到这点。通常，当你的脚本使用Conduit（比如IRC bot），你就应该为他们创建一个系统代理账户。

系统代理用户可以:

**不能进行登陆** (他们可以提供的管道来访问API方法）

**不能查看diff或拥有任务**

**不能出现在邮件抄送列表中**

现在,  **系统代理** 账户一旦创建就没有办法进行修改。这是为了避免管理员把系统代理当做一个普通用户进行修改，检索他们的管道验证，以及将他们改回来（这里允许管理员为其他用户添加证书验证）。

### 禁用账户（Disable Users）

**禁用账户** 是那些将不会处于激活状态的账户。 通常，当有人离开一个项目（例如，有人离职，实习生或合同期满），你需要将其账户进行反激活，来终止它对系统的访问。

禁用账户:

**不能登陆**

**不能访问Conduit**

**不能接收邮件**

**将不会出现在拥有者/回查者/邮件抄送列表中**.

用户仅仅是被关闭（而不是删除），因为这里有一些列工作流不同意将用户完全删除，例如，寻找与这些用户相关的旧版本或者任务（所以你可以找别人来关注他们）; 需要通过用户名来识别修订的作者;并且，保存他们所有的数据，当他们重新返回项目（例如，离职延期，实习转正）。

## 使用教程: 配置一个外置编辑器

将一个外置编辑器集成到Diffusion和Differential中。

### 概述

你可以配置一个URI处理器允许使用你喜欢的编辑器，打开Differential和Diffusion中的文件。

### 配置编辑器

为了配置一个外置编辑器，需要到 Settings -> Application Settings -> Display Preferences ，并且设置通过一个 URI模式来设置"Editor Link" (见下文)。这将在Differential开启一个“Open in Editor”连接，在Diffusion添加了一个“Edit”按钮。

通常，你可以通过如下方式设置这个域：

editor://open/?file=%f

一些编辑器支持一次性打开过个文件，文件名之间可以用空格隔开。如果你的编辑器支持这种功能，设置 "Edit Multiple Files" 为 "Supported"。 当你将这个选项设置为"Not Supported"就会在界面中关闭"Open All"按钮。

### 配置: TextMate （OS X）

TextMate 默认安装了 txmt:// 处理器，所以当使用TextMate你可以很容易的配置这些功能。

首先，通过不同库的代号，创建一个带符号连接的本地目录。例如，如果你正在开发Phabricator，你将会看到如下的信息：

/Users/alincoln/editor\_links/ $ ls -l

... ARC -> /Users/alincoln/workspace/arcanist/

... P -> /Users/alincoln/workspace/phabricator/

... PHU -> /Users/alincoln/workspace/libphutil/

然后设置"Editor Link" 为:

txmt://open/?url=file:///Users/alincoln/editor\_links/%r/%f&line=%l

### 配置: 其他编辑器

配置其他编辑器和环境，可以在以下的连接找到方法:

http://wiki.nette.org/en/howto-editor-link

## 使用教程: 管理 Phabricator 的邮件

如何有效的管理Phabricator中的邮件通告。

### 概述

Phabricator使用邮件，作为主要的通知渠道，但是它发出的邮件数量，在一个活跃的团队中，会给组内人员带来困扰。本节就是来讨论管理邮件的策略。

迄今为止，管理邮件的最好的方式就是去写邮件规则，将邮件进行分类。现在大多数的邮件客户端，都允许你快速编写复杂的邮件规则，来对邮件进行分类，删除等操作。

### 减少邮件

你可以减少你的邮件接收量，通过关闭一些邮件类型来达到这种功能，在Settings->Email Preferences中可以进行设置。例如，你可以关闭由与你自己的动作而产生的邮件（例如，你对一个修订进行评论），或者关闭那些不是很重要邮件类型。

### 邮件规则

管理邮件的最好方式就是撰写邮件规则。简单的规则就可以将从Differential，Maniphest以及Herald发送的邮件移动到不同的文件夹中。

Phabricator 还设置大量的邮件头（见下文），让你写出更有效的邮件规则。

### 邮件头

Phabricator发送的邮件拥有各种邮件头，邮件头可以用于邮件规则，并管理邮件。

Headers可以包含多个列表。例如，A列表包含两个元素, 1 和 15 。一般写成如下的形式：

X-Header: <1>, <15>

这意味这你可以写一条规则去在文件头中匹配 "<1>"。如果只与 "1"匹配，你将不会正确匹配到"15";但是当匹配 "<1>" 时，就仅正确的匹配到"<1>"。

有些邮件头只是用单一的值，不过会出现多次。但邮件客户端（例如，Outlook）不能创建使用正表达式或者通配符的邮件规则。

Phabricator会在邮件中添加的邮件头:

X-Phabricator-Sent-This-Message: 这个信息在Phabricator发出的邮件头中都会标明。你可以用它来区分邮件是来自与Phabricator，或是某人回复/转发的Phabricator邮件。

X-Phabricator-To: 这个信息在Phabricator发出的邮件头中都会标明。它会表明在邮件配置发生任何突变前，哪个PHID是这封邮件的原始接收者。

X-Phabricator-Cc: 这个信息在Phabricator发出的邮件头中都会标明。它会表明在邮件配置发生任何突变前，哪个PHID是这封邮件的原始抄送接收者。

X-Differential-Author: 这个信息将会在Differential发送邮件的头部，并写明修订作者。你可以用过滤器只接受你（或者其他的用户）的修订发出的邮件。

X-Differential-Reviewer:这个信息将会在Differential发送邮件的头部，并且写明回查者。你可以使用过滤器只接受需要你去回查的，明确将你添加入抄送列表的，或匹配Herald规则被添加入抄送列表的修订。

X-Differential-Reviewers: 列出之前的版本。

X-Differential-CC: 这个信息将会出现在Differential发送邮件的头部，并且明确对于这个修订的抄送列表。

X-Differential-CCs: 列出之前的版本。

X-Differential-Explicit-CC: 这个信息将会出现在Differential发送邮件的头部，并且明确对于这个修订的抄送列表。（这里抄送列表是人为添加，而非Herald规则添加）。

X-Differential-Explicit-CCs: 列出之前的版本。

X-Phabricator-Mail-Tags: 这个信息会在一些邮件的头部添加，并且有一系列关于邮件的描述符（这是一项新功能，并且主题会有所改变）。

X-Herald-Rules: 这个信息会在一些邮件的头部添加，并且标明被触发的Herald规则编号。你可以使用它对不同规则触发的邮件，进行排序或分类。

## 使用教程: Review vs. Audit

讨论回查和审核工作流的不同之处。

### 概述

Phabricator支持两种相似，但是不同的代码审查流程：

**Differential** 在代码上传之前进行检查，叫做“回查”（reviews）。可以回顾 [Differential User Guide](https://secure.phabricator.com/diviner/find/?name=Differential_User_Guide&type=article&jump=1)章节。

**Audit** 在代码上传之后进行检查，叫做“审核”（audits） 。可以回顾[Audit User Guide](https://secure.phabricator.com/diviner/find/?name=Audit_User_Guide&type=article&jump=1)章节。

无论是轻量级的，还是基于web异步的工作流，都可以在他们自己的机器上进行独立回查/审核代码。不同步的回查，需要和回查者见面，并讨论代码的修改。

### 回查的优势

上传前回查要比上传后的审核，显得更为强大。你可以通过要求在修改在上传之前回查代码，来扩展这种优势：

作者有很强烈的动机去做小的修改，让修改容易理解，对他们的修改做出充分的解释，并提供相应的测试计划，测试覆盖率和上下文。

回查者有一个机会对架构和方法做出一些重要的建议。这些建议都使得采用审核的方式很缺乏吸引力，当在事件很紧迫的情况下，采用上推完再审核的方式，就显的更加困难。

作者有前列的动机去解决问题和响应在回查过程中反馈的信息，因为这些问题会阻碍他们。对于审核，作业就没有这么强烈的动机去做这些事了。

作者可以要求回查者们核实问题被修复了以后，再上传代码。

作者可以很容易的得到早期反馈，并且能即使的修复这些问题，或对方向进行修正。

一旦这个修改出现在成品中，回查者可以更好的接收这个修改，因为他们已经通过回查的方式对代码有了了解。

回查者看到一些在自动化测试中很难察觉到的问题。比如，回查者可以对性能问题进行推理;自动化测试在使用小数据量进行操作，可能就很容易错过这些问题。

在改动之前进行充分的沟通，这将会产生更好的效果。

回查的理论成本是，它延缓了在开发阶段引入阻塞的过程与开发人员在上面浪费的事件，可以使开发人员有更多时间投入于开发。不过，这似乎也不是那么正确，因为在开发阶段的成本低廉，在其他方面就需要有更大的开销：

Differential很快，且提供一个十分轻量级的代码提交回查和执行回查。

作者可以自由的跟随其他的修订，当代码已经在回查流程中。用适当的修订管理（例如，Git的本地分支），他们甚至可以追求简单的修订方式。作者应该很少在回查中阻塞，即使阻塞也需要全部回查者对这个阻塞状态进行审批。

工作流在整体上是轻量级的，在有熟练的回查者，可以有效的识别错误。通常，他们在回查阶段修复错误的速度要比修复在成品中的错误快很多。

更重要的是，他能有效的识别构架与方法的问题。这都是免费固定在回查阶段（不建议这样做），当这些问题在成品中出现，需要花费更多的事件去处理。不管你有多么好的测试套件，它都不能确定解决方案，这是由于缺失上下文造成的？或，就是个误会？或，其他更坏的想法。

当修订特别大或这特别复杂时，通常需要及时的进行回查（当又大又复杂，那么需要周期性的回查）。机会所有大修改都可以拆分成独立的小块，这样便于理解和测试。回查倾向于鼓励提交较小的和更容易理解的修订。

回查可以与静态分析进行集成，其可以检测（在大多数情况下是正确的）机制问题，通过代码的语法、格式、命名规范、拼写问题和一些程序错误。这减少了检查代码的时间和方式，这也就以为这回查者们可以专注于代码中的实际问题，而不是一些小问题。

回查会创建一个对上下文的永久的记录，以及解释为什么要做这个修改。通常，这里会有比单独提交时更多的信息（作者解释修改的时间）。这使得代码在之后更痛以理解，当需要对它进行修改时，就可以使用对应的方式进行修改。

通过arc patch，就可以从Differential很容易的下拉一个修改，这要比远程下拉代码容易的多。

### 审核的优势

如果你不同意上述观点，审核提供了一些与回查几乎无冲突的回查方式：

审核完全需要由Phabricator提供完全的驱动，用户都用不着安装

审核不需要调整现有的工作流和小型培训

审核全程都是无阻塞的，并且发出比回查要少很多的通告

即使已经回查，审核也可以作为一个补充发挥作用，可以在回查后提交后发现的问题。

### 下一步

* 了解关于回查的更多信息，可参见 [Differential User Guide](https://secure.phabricator.com/diviner/find/?name=Differential_User_Guide&type=article&jump=1)章节。
* 了解关于审核的更多信息，可参见 [Audit User Guide](https://secure.phabricator.com/diviner/find/?name=Audit_User_Guide&type=article&jump=1)章节。

## 使用教程: UTF-8 和字符编码格式

Phabricator 如何处理字符编码。

### 概述

Phabricator 使用UTF-8编码来存储内部文本数据，处理文本数据，输出文本数据，并且期望所有的输入都是UTF-8编码类型。这就以为这你要用UTF-8格式去书写你的源码。在大多数情况下，你不需要去改变什么，因为ASCII文本就是UTF-8的子集。

如果你没有一个UTF-8的代码库，那么你有两个选择：

将库中的所有文件都转换成ASCII或UTF-8编码（可见下文“检测和修复文件”）。强烈建议，特别是当已经出现了编码问题。

配置Phabricator，当你需要的时候再将代码库中的文件转换成UTF-8编码格式 (可见下文"支持其他编码格式" )。这个功能还不完全支持，并且代码库中会有多种编码格式，其中会有不支持的编码格式。

### 检测和修复文件

建议你在使用ASCII编码来写文本文件，但是 Phabricator对UTF-8编码源文件是完全支持的。

如果你的工程不允许使用UTF-8编码，因为有几个随机二进制的在这个工程中。在libphutil中有一个脚本可以帮助你来识别他们，并解决这个问题

project/ $ libphutil/scripts/utils/utf8.php

通常， 加“-t”选项运行脚本来找到文件“坏字节”的范围，然后对改文件运行脚本（不加“-t”参数）定位问题。比如说：

project/ $ find . -type f -name '\*.c' -print0 | xargs -0 -n256 ./utf8 -t

./hello\_world.c

如果此脚本没有任何输出，那么你的所有文件都是有效的UTF-8文件。如果你发现了问题，就要去修复他们。通过省略“-t”参数来定位问题：

project/ $ ./utf8.php hello\_world.c

FAIL hello\_world.c

3 main()

4 {

5 printf ("Hello World<0xE9><0xD6>!\n");

6 }

7

这说明第五行有问题（在控制台，这里显示会被高亮）。通常一个代码库大多数文件都是UTF-8文件，但是极少数文件中会有一些问题。比如说，从word里面直接粘贴过来的引号。在这种情况下，你可以很容易的手动识别和解决问题。

如果你的库有大量的UTF-8问题，这时的你望而却步。Phabricator不包括任何特殊的工具来帮你在系统中处理这个问题。你可以破解utf8.php文件作为一个开始，或者使用其他批处理工具来帮助你修复问题。

### 支持其他编码格式

Phabricator 除了UTF-8还支持其他的编码格式。

NOTE:替换编码还不是完全支持，有些功能，暂时还无法正常共工作。代码库中有掺杂着各种编码格式（比如，一些文件使用ISO-8859-1格式，另一些文件使用Shift-JIS格式），其中的编码格式不能完全被支持。

使用另一种编码，在Diffusion编辑对应的库，并指定使用的编码格式。

或者你可运行用arc –encoding来配置编码格式，或者在你的.arcconfig文件中对encoding变量进行设置。

参考：

[1] 悠悠客博客 <http://my.oschina.net/yoyoko/blog/127071>