

备份系统的部署与实现

chenj@lemote.com

部署.....	1
在存放备份数据的主机上安装服务端.....	1
更多服务端的设置.....	2
在需要备份的主机安装客户端.....	4
更多客户端设置.....	6
备份数据到不同的备份节点上.....	7
实现.....	7
扩展备份系统.....	9
扩展客户端，增加支持模块.....	9
扩展服务端，增加支持模块.....	10
关键例程.....	10
请求的状态.....	10
git 传输的过程.....	10
Bash 技巧.....	11

部署

备份系统需要部署在客户端和服务端上。在需备份的主机上安装客户端，并且可配置向多个服务端备份。服务端是存放备份数据的主机。管理员通过配置访问许可列表，来许可哪些客户端可以备份到哪些服务端上。

目前的备份系统支持 git 仓库和文件系统级打包备份（包括从数据库中导出文件）。

以下分为在需备份的主机安装客户端和在存放备份数据的主机上安装服务端两小节来论述。

在存放备份数据的主机上安装服务端

以下假设安装到目录/data/backup，并假设当前工作目录就是/data/backup

1. 新建用户 adminbackup

```
adduser --system --shell /bin/bash --gecos 'adminbackup' --group --disabled-  
password --home /data/backup adminbackup
```

2. 生成服务端的公钥，以便其通过 sftp，自动下载客户端准备的备份数据。

```
sudo -H -u adminbackup ssh-keygen -C "adminbackup@172.16.2.57" -N ""
```

拷贝到管理员处，以便分发到客户端

```
scp ~adminbackup/.ssh/id_rsa.pub ...
```

3. 把备份系统压缩包解压到/data/backup 下，并修改除 **scripts** 子目录外，所有目录的所有者为 adminbackup:adminbackup。

4. 配置服务端：

```
sudo -H -u adminbackup vim ~adminbackup/
```

```
# 备份主机标识，例如 ip  
Host='172.16.2.57'
```

```
# 管理员的邮件地址，用于出错通知  
Administrator=('chenj@lemote.com')
```

```
# 其他磁盘布局相关  
TODOdir="/data/backup/TODO"  
Datadir="/data/backup/data"  
Cmdir="/data/backup/scripts/cmds"  
UtilsDir="/data/backup/scripts/utils"  
LogfileDir="/data/backup/log"
```

5. 部署 gitosis

```
adduser --system --shell /bin/bash --gecos 'git version control' --group --disabled-  
password --home /data/backup/log/gitosis_u git
```

```
# 拷贝管理员的公钥至此，假定为 id_rsa.pub。导入公钥，初始化 gitosis  
sudo -H -u git gitosis-init < id_rsa.pub
```

```
echo >> ~git/repositories/gitosis-admin.git/hooks/post-update  
echo "python /data/backup/scripts/utils/gitosis-admin.post-update" >>  
~git/repositories/gitosis-admin.git/hooks/post-update
```

```
chmod a+x ~git/repositories/gitosis-admin.git/hooks/post-update
```

6. 权限调整

```
adduser adminbackup git  
chown git:git /data/backup/TODO  
chmod g+rwX TODO
```

7. 在 cron 中，定期调度相关例程¹

```
# vim /etc/crontab  
# 每 5 分钟，检查并处理备份请求  
0-59/5 * * * * adminbackup /data/backup/scripts/do_backup
```

更多服务端的设置

一、 将备份的 git 仓库导出为 Web 界面。

1. 安装并配置 apache 及 cgit

```
# 安装 apache。cgit 为补丁过的版本（参见《trac 开发社区网站的搭建》一文
```

```
# 假设 cgit 的全部内容放在 /var/www/cgit，配置 /etc/apache2/sites-  
available/default
```

```
Alias /cgit/cgit.css /var/www/cgit/cgit.css
```

```
Alias /cgit/cgit.png /var/www/cgit/cgit.png
```

```
ScriptAlias /cgit /var/www/cgit/cgit.cgi
```

1 需要去除 cron 运行任务后，往用户发送邮件的特性。不然磁盘空间可能会占完。

```
<Directory "/var/www/cgit/">
  SetEnv CGIT_CONFIG /var/www/cgit/cgitrc
  AllowOverride None
  Options FollowSymLinks -SymLinksIfOwnerMatch
  Order allow,deny
  Allow from all
</Directory>
```

2. 编辑/var/www/cgit/cgitrc

3. 权限调整

```
adduser www-data adminbackup
```

二、 将备份的 git 仓库，导出为 git 界面。

```
# vim /etc/rc.local 中加入 :
sudo -u adminbackup git daemon --base-path=/data/backup/data/172.16.0.60/git
```

三、 对 git 仓库，定期运行 git gc（每月一次）

```
# vim /etc/crontab
0 0 8 * * adminbackup /data/maintenance/git_gc.sh
/data/backup/data/172.16.0.60/git chenj@lemote.com
0 23 7 * * git /data/maintenance/git_gc.sh
/data/backup/log/gitosis_u/repositories chenj@lemote.com
```

这里需要/data目录下有一个名为 maintenance 的目录，其 git_gc.sh（需要可执行权限）内容如下：

```
#!/bin/bash
set -u

# sendmail.py 脚本的地址。
# 这里为与 git_gc.sh 同一目录下的
# 指向/data/backup/scripts/utils/sendmail.py 的链接
sendmail="$(dirname $0)/sendmail.py"

gitbase="$1"
shift
receivers=$@

# 需要新建一个 www-data 和 git 用户可写的目录 logs
log="$(dirname $0)/logs/git-gc $(date "+%Y-%m-%d %H:%M:%S").log"

for repo in $gitbase/*.git
do
  if test -d "$repo"; then
    echo "git-dir=\"$repo\" >> \"$log"
    (
      export GIT_DIR="$repo"
      git gc --aggressive
    ) 1>>"$log" 2>&1
  fi
done
```

```
        echo "" >> "$log"
    fi
done

test -e "$log" && python "$sendmail" "git gc" "$log" "$receivers"
```

在需备份的主机安装客户端

以下假设安装到目录/var/backup，并假设当前工作目录就是/var/backup。

1. 新建用户 adminbackup，并设置相应权限，使得能够读取数据进行备份。

```
adduser --system --shell /bin/bash --gecos 'adminbackup' --group --disabled-
password --home /var/backup adminbackup

# 加入到 git 组（使用 gitosis 管理的 git 仓库）
adduser adminbackup git

# 加入到 www-data 组（Apache web 服务相关数据）
adduser adminbackup www-data

# visudo（顺利读取备份隐私数据，备份 postgres 数据库）
adminbackup ALL = (root) NOPASSWD: /bin/chown, /bin/tar
adminbackup ALL = (postgres) NOPASSWD: /usr/bin/pg_dump,
/usr/bin/pg_dumpall
```

2. 把备份系统压缩包解压到/var/backup下，并修改除 **scripts** 子目录外，所有目录的所有者为 adminbackup:adminbackup
3. 配置客户端。
 - 编辑/var/backup/scripts/backupconfig.sh

```
Host="172.16.0.60"
Export_sftp="adminbackup@${Host}"
Export_git="git://${Host}"

DefaultBackupNode="ssh://git@172.16.2.57/${Host}.git"
UtilsDir="/var/backup/scripts/utils"
RequestDir="/var/backup/requests"
LogfileDir="/var/backup/log"
SendQueue="/var/backup/log/queues"
tmpDir="/var/backup/tmp"

git_base="/var/www/code/git"
postgres_db=""
trac="/var/www/code"
app="/var/www/app"
drupal="/var/www/drupal"

Administrator=("chenj@lemote.com")
```

- 编辑/var/backup/scripts/backupconfig.py

```
RequestDir = "/var/backup/requests"
defaultBackupNode = "172.16.2.57"
```

```
Cmdir = "/var/backup/scripts/cmds"
```

4. 切换至用户 adminbackup

```
su adminbackup
```

5. 在 **log/queues** 目录下，对每个存放备份数据的节点（服务端），创建 git 仓库。git 仓库用来向服务端提交请求。下面这个例子中，存放备份数据的节点为 172.16.2.57，本地主机为 172.16.0.60：

```
pushd log/queues
git --git-dir=172.16.2.57.git init

# 修改仓库配置，令 bare=false
vim 172.16.2.57.git/config

# 增加远程节点的仓库（接收备份请求），记为 origin
git --git-dir=172.16.2.57.git remote add origin ssh://git@172.16.2.57/172.16.0.60.git
popd
```

还需要配置 gitosis.conf，来许可向备份节点提交请求。

6. 修改用户 adminbackup 的 ssh 配置。

- 修改 .ssh/config，避免 git 通过 ssh 协议同步仓库时，需要输入 known_hosts 相关的提示，从而中断了自动化运行。

```
cat >> ~/.ssh/config <<EOF
Host 172.16.2.57
    StrictHostKeyChecking no
    UserKnownHostsFile=/dev/null
EOF
```

- 将服务端的公钥加入到 authorized_keys。使得服务端能自动通过 sftp 下载客户端准备的备份数据。

7. 在 cron 中，定期执行相关例程

```
# 输入 exit 或者 Ctrl-D，退出至超级用户帐号

# vim /etc/crontab
# 每 5 分钟，检查并发送备份请求
0-59/5 * * * * adminbackup /var/backup/scripts/send_request

# 每月进行一次完全备份。
# 注意：重定向操作是必要的
#      cron 不能很好的处理 python 的 logging 模块输出的日志。
0 23 11 * * adminbackup /var/backup/scripts/backup_all 1>/dev/null
2>&1
```

更多客户端设置

一、 git 仓库有更新时，自动提交备份请求

1. 将下述内容添加到 hooks/post-update 中，并给予该文件可执行权限。

```
sudo -u adminbackup python /var/backup/scripts/commitBackupRequest git "$PWD" 1>/dev/null 2>&1
```

2. 为了让上述脚本顺利执行，进行 sudo 权限调整：

```
# visudo  
git    ALL = (adminbackup) NOPASSWD: /usr/bin/python
```

二、对 git 仓库，定期运行 git gc（每月一次）

```
# vim /etc/crontab  
50 23 11 ** adminbackup /root/maintenance/git_gc.sh /var/backup/log/queues  
chenj@lemote.com  
0 0 12 ** git /root/maintenance/git_gc.sh /var/www/code/git  
chenj@lemote.com
```

这里需要有 adminbackup 与 git 用户可访问的 /root/maintenance 目录。其他参见服务端该设置。

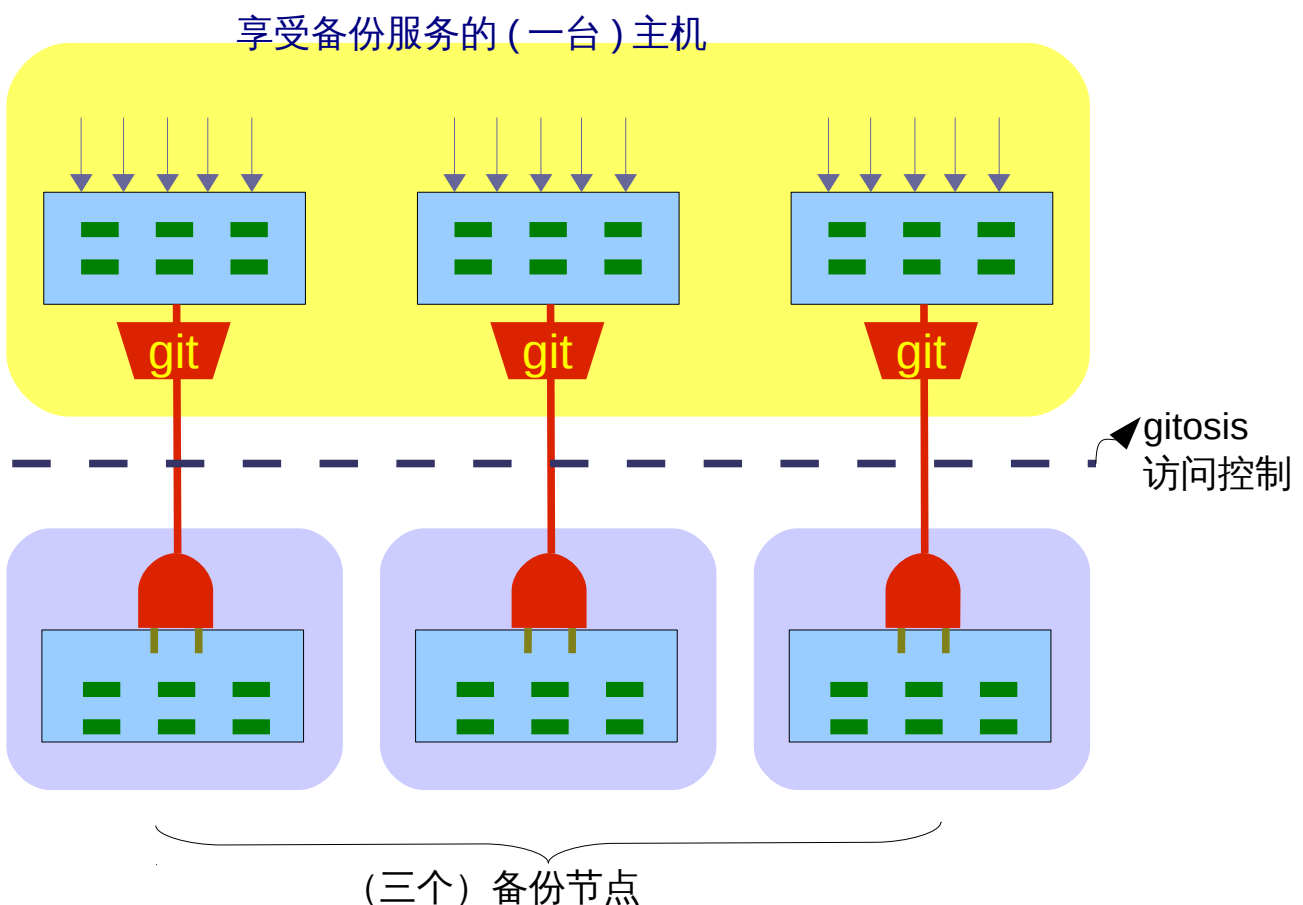
备份数据到不同的备份节点上

重复第 5 点和第 6 点。并在 /var/backup/scripts/backupconfig.py 中设置 **DefaultBackupNode**。

注意，备份数据到非默认备份节点，需要在发起备份时指定：

```
python /var/backup/scripts/commitBackupRequest <module> <path> [backup_node]
```

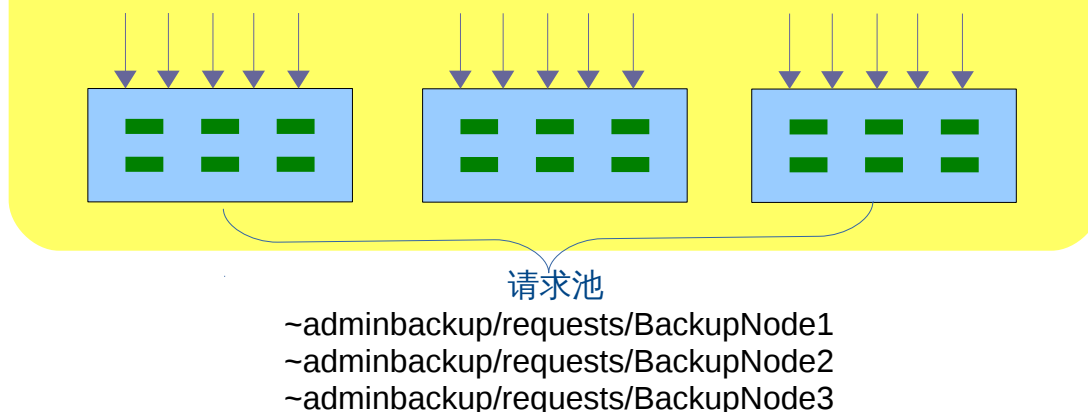
实现



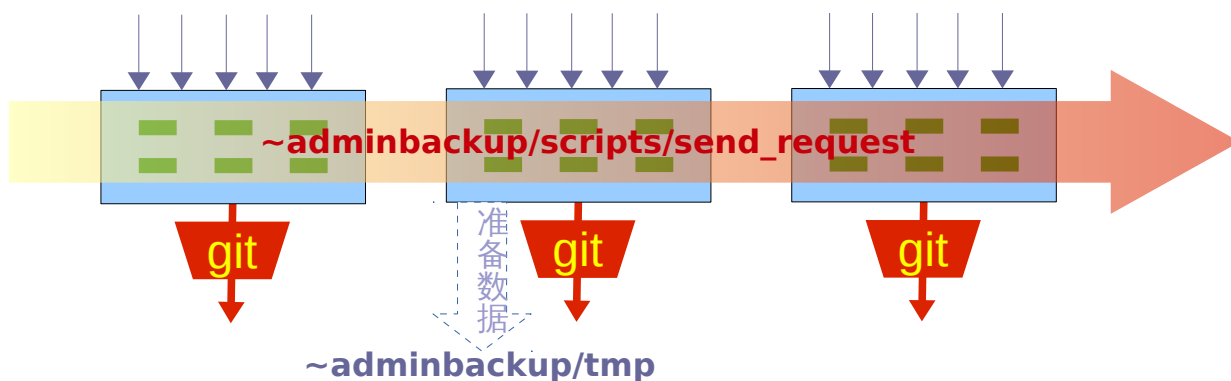
上图为备份系统的结构总览。以下说明备份的工作流程：

1. 客户端提交备份请求到请求池。每个请求池对应一个备份节点。

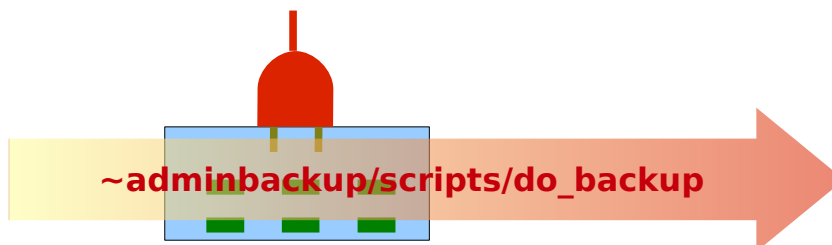
```
python ~adminbackup/scripts/commitBackupRequest <module> <path> [backup_node]
```



2. 在客户端的 cron 中设定每隔几分钟扫描各个请求池。
 - 依据请求的类型（即需要处理请求的模块），可能需要为每个请求准备数据。
 - 最后，把请求池中的请求，通过 git，同步到各个备份节点上。



3. 客户端的请求，经过 gitosis 许可，进入到备份节点上。备份节点上的 cron 中，也设定每隔几分钟扫描各个请求池。依据不同的请求类型：
 - 通过 sftp 下载，客户端准备的备份数据
 - 通过 git，备份 git 仓库



小结：

- 备份系统的安全模型是：备份节点不应该信任客户端（即需要备份数据的主机）。备份节点应该比需备份的主机有更严密的保护。
- 由于备份系统以 git 作为通信的手段。一对 git 仓库，代表一条单向通信的线路。

- 使用 git 传输的目的：
 - 避免给予客户端太多权限（与 ssh 相比）。
 - 无需登陆服务端更改配置来接受新客户端，即仅需要配置 gitosis 即可。
 - 备份日志，在 git 日志中自然体现。
- 不支持离线。当备份节点下线时，客户端的请求会被忽略，且管理员无法获知该状况。这是由于 git 命令并不返回错误值，脚本无法获悉其同步是否成功。

扩展备份系统

扩展客户端，增加支持模块

- 将新的模块（shell 脚本），放至 scripts/cmds。
- ✓ 新备份模块的接口是：
 - \$1 —— 模块
 - \$2 —— 路径
- ✓ 脚本中使用 **add_queue** "标识" "内容"，向服务端提交请求。
 内容统一的格式为“<服务端模块> <自定义内容>”（注意两个域之间用一个空格分隔）
 针对服务端的模块，内容目前为两类：
 - 服务端 git 模块：

<git> <git 导出的 URL>
 - 服务端 sftp 模块：

<sftp> <timestamp> <sftp 导出的 URL>
- 编辑 scripts/backupconfig.py，在 **Cmds** 中注册模块：

```
Cmds = {
    'git'      : os.path.join (Cmdir, 'backup_git "%s"'),
    'postgresdb' : os.path.join (Cmdir, 'backup_postgresdb "%s"'),
    'trac'     : os.path.join (Cmdir, 'backup_FS trac "%s"'),
    'app'      : os.path.join (Cmdir, 'backup_FS app "%s"'),
    'drupal'   : os.path.join (Cmdir, 'backup_FS drupal "%s"'),
    'dummy'    : os.path.join (Cmdir, 'dummy "%s"'),
}
```

扩展服务端，增加支持模块

- 在 scripts/cmds 中，增加模块。
- 修改 scripts/do_backup 中的 **ProcessRequest** 函数，在 case 的分发过程中，增加到新模块的代码路径。

关键例程

请求的状态

备份系统假定，rename 系统调用是一个原子操作。

客户端请求池（文件夹）中的请求的状态（文件命名标识）：

- i* : incomplete，该请求正在生成阶段
- C* : 完成但未处理的请求
- P* : 该请求正在处理阶段
- F* : 该请求已经被处理

服务端请求池（文件夹）中的请求的状态（文件命名标识）：

- RQ* : 未处理的请求
- C* :
- P* :
- F* :

服务端和客户端对请求池的扫描，先处理 P*，再将 C* → P* 并处理。

git 传输的过程

客户端

1. 新建临时目录。各个模块调用 SHELL 函数 **add_queue**，将需要服务端处理的请求写入该临时目录。
2. 若临时目录不为空，调用 git 传输：

```
(
    set -e
    export GIT_DIR="${SendQueue}/${(basename ${N})}.git"
    test -d "$GIT_DIR"
    pushd "$TODOdir"

    git rm -rf --cached . 1>/dev/null 2>&1
    git add RQ*

    # Make sure, we have changes.
    date > timestamp
    git add timestamp

    git commit -m "${Launch_time}: see log '${Host}':'${Logfile}'"
    git push origin +master:master
    popd
) 1>>"$t2" 2>&1
```

注意到几点：

- 使用 `git rm` 清除之前索引。这样，本次 `git` 仅传输新的请求。
 - `timestamp`，是为了强制让 `git` 认为有更新，不然 `git` 会拒绝传输。
 - `git` 传输失败，不会返回错误。因此 `2>&1...` 这步基本不工作。
3. 服务端在 `git` 仓库更新时，进行下述动作
- 获取记录该更新的 ID（`git` 的 `commit` 的标识）。
 - 获取请求池的锁。
 - 在请求池新建 `.incomplete` 文件夹，通过 **`git archive`**，导出本次 `git` 更新内容到该目录。
 - 移动 `.incomplete` 下的内容到请求池。删除 `.incomplete`。

Bash 技巧

- **`. script`** 会传递调用脚本定义的变量和函数。
- **`bash script`** 与 **`./script`** 中，`$0` 为脚本本身，但是 **`. script`** 中，`$0` 为 `-bash`。