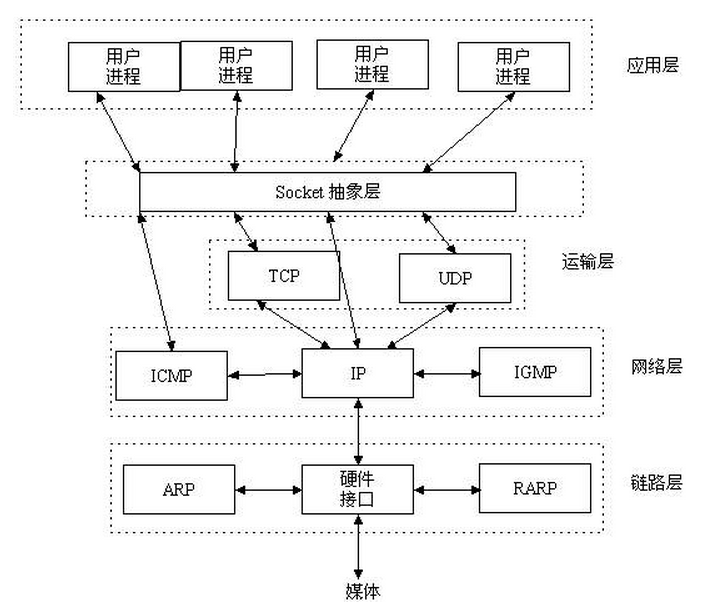
1. httpd
   1. 机制相关

**1.socket**

（1）Socket是应用层与TCP/IP协议族通信的中间软件抽象层，本质是一组API，可以被用来实现不同主机间的进程的通信。



（2）bsd在实现上把socket设计成一种文件，可以通过虚拟文件系统的操作接口访问socket。访问socket时会调用相应的驱动程序，从而也就是使用底层协议进行通信。

（3）socket原本是为网络通信设计的，但后来又发展出一种IPC机制，即UNIX Domain Socket。相对于其他方法，UNIX Domain Socket更有效率：不需要经过网络协议栈，不需要打包拆包、计算校验和、维护序号和应答等，只是将应用层数据从一个进程拷贝到另一个进程。UNIX Domain Socket也提供面向流和面向数据包两种API接口，类似于TCP和UDP，但是面向消息的UNIX Domain Socket也是可靠的，消息既不会丢失也不会顺序错乱。

**2.动态共享对象（DSO）**

（1）前提是操作系统支持动态链接机制（基本上都支持）。

（2）Apache是模块化的服务器。模块可以在编译时被静态包含进httpd二进制文件，也可以成独立于httpd二进制文件的动态共享对象（DSO）。通常Apache核心只包含实现最基本功能的模块，其他模块作为DSO模块可以在启动和重启时动态加载。为了实现动态加载，必须将mod\_so编译到核心中。

（3）编译并安装已发布的Apache模块（这些模块就在安装源码包某个目录下），可使用--enable-module（module为模块名）=shared将module作为DSO模块独立编译。如将编译mod\_foo.c为mod\_foo.so。



（4）编译并安装第三方模块，比如编译mod\_foo.c为mod\_foo.so。



（5）配置Apache以便以后安装共享模块。



（6）可以使用apxs编译并安装在Apache源码树以外的第三方模块，比如编译mod\_foo.c为mod\_foo.so的DSO模块。-i：将源码编译并安装到modules目录下。-a：在httpd.conf文件中增加一个LoadModule行。

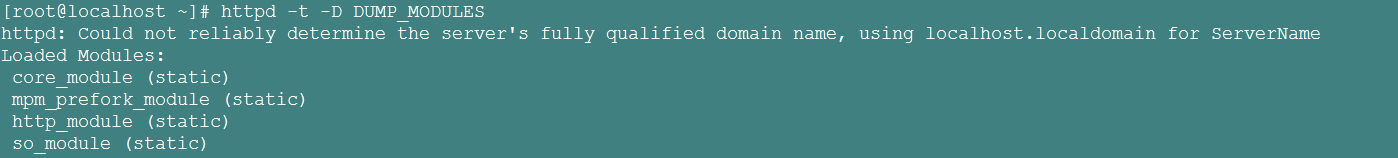


（7）在配置文件中使用LoadModule指令加载模块，模块文件位于ServerRoot下模块目录中。配置文件中使用<IfModule>指令包含的指令，只适用于特定模块。

（8）查看已经编译到服务器中的模块。



（9）查看httpd加载的所有模块（包括静态编译的和动态共享的模块）。



**3.通用网关接口（CGI）**

（1）概念：Web服务器提供信息服务的标准接口。通过CGI接口，Web服务器就能够获取客户端提交的信息，转交给服务器端的CGI程序进行处理，最后返回结果给客户端。

（2）根据CGI接口规范编写的程序就是CGI程序。CGI程序是被服务器（如Apache）调用来真正处理用户发送过来的数据的程序。

（3）可以用PHP、JAVA等编写cgi程序。如，Python编写的CGI程序后缀为.py，C编写的CGI程序后缀为.cgi。最早的cgi程序就是用c写的。

**4.处理器**

（1）处理器：当一个文件被调用（请求到文件系统映射）时，Apache所执行的操作。

（2）处理器可以被编译进服务器核心中，也可以包含在模块中，还可以用Action指令增加。标准发行版中内建的处理器有：

|  |  |
| --- | --- |
| default-handler | 静态内容的默认处理器，使用default\_handler()发送文件。([core](mk:@MSITStore:F:\documents\books\ApacheMenu_zh_CN.chm::/mod/core.html)) |
| send-as-is | 原样返回。 |
| cgi-script | 将文件视为CGI脚本。([mod\_cgi](mk:@MSITStore:F:\documents\books\ApacheMenu_zh_CN.chm::/mod/mod_cgi.html)) |
| server-info | 获取服务器的配置信息。([mod\_info](mk:@MSITStore:F:\documents\books\ApacheMenu_zh_CN.chm::/mod/mod_info.html)) |
| server-status | 获取服务器状态的报告。([mod\_status](mk:@MSITStore:F:\documents\books\ApacheMenu_zh_CN.chm::/mod/mod_status.html)) |
| type-map | 将文件作为类型表文件解析以实现内容协商。([mod\_negotiation](mk:@MSITStore:F:\documents\books\ApacheMenu_zh_CN.chm::/mod/mod_negotiation.html)) |

（3）Action指令



当action-type被请求时，执行cgi-script。如下，请求image/gif调用/cgi-bin/images.cgi处理。



AddHandler指定.xyz扩展名为my-file-type类型，Action指定my-file-type类型由/cgi-bin/program.cgi处理。



（4）AddHandler指令



指定extension扩展名的文件由handler-name处理器处理。作用于所有有效映射关系上，并覆盖所有相同的extension扩展名映射。如下，所有.cgi文件，都会被当成是CGI程序处理。



（5）SetHandler指令



强制所有匹配的文件被一个指定的处理器处理。当SetHandler指令放入.htaccess或<Directory>或<Location>容器中时，会强制所有匹配的文件通过handler-name指定的处理器处理。



**5.apache调用php处理工作方式**

（1）在编译php时，会使用--with-apxs2=/usr/sbin/apxs进行配置，最后编译1个libphp5.so到modules目录下，并生成/etc/httpd/conf.d/php.conf配置文件。



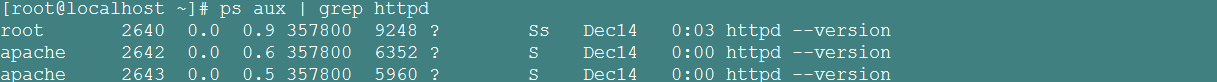


（2）如上，url映射到文件系统的文件如果为.php文件，则由php5-script处理。

**6.进程权限**

（1）httpd会以root身份启动。启动之后生成的子进程则是在配置文件中指定的较低权限的用户。





* 1. 基本配置
     1. 配置文件写法

（1）配置指令不区分字符大小写，但配置的值区分大小写。

（2）有些指令可以重复出现多次。

（3）绝大多数配置修改后，可以用过service httpd reload来生效，但如果修改了监听的地址或端口，必须重启服务才能生效。

* + 1. 指令的作用域

（1）配置段（容器）和.htaccess文件可以改变配置指令的作用范围。

（2）容器<IfDefine>、<IfModule>、<IfVersion>仅在启动和重新启动中起作用。其中，<IfDefine>只在httpd命令行中设定特定参数时有效。<IfModule>只在服务器启用特定模块（静态地编译进了服务器，或是被动态装载进了服务器）时有效（LoadModule必须在容器之前）。<IfVersion>只在指定版本下有效。

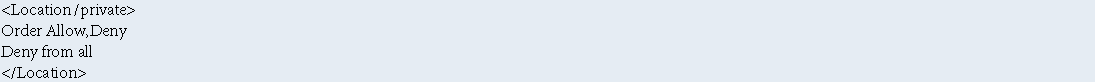
（3）<VirtualHost>容器作用于特定的虚拟主机。

（4）<Proxy>和<ProxyMatch>容器仅作用于通过mod\_proxy代理服务器访问的、与指定URL匹配的站点。

（5）<Directory>和<Files>（包括对应的正则表达式版本<DirectoryMatch>和<FilesMatch>）是针对文件系统的容器，作用于文件系统的特定部分。

其中，<Directory>作用于指定目录及其所有子目录（.htaccess也可达到同样效果）。<Files>作用于特定的文件，无论该文件位于哪个目录。<Files>和<Directory>结合使用可以作用于特定目录下的特定文件。

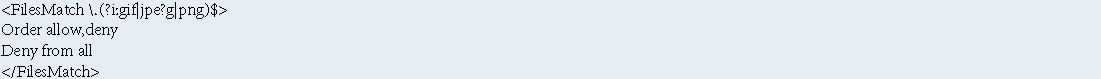
（6）<Location>（对应的正则表达式版本是<LocationMatch>）是针对网络地址的容器。网络地址即浏览器中地址栏url中的路径部分。如下面的配置服务器会拒绝任何以/private开头的URL路径的访问，包括example.com/private、example.com/private123、example.com/private/dir/file.html等。



<Location>指令与文件系统无关，不要用它来限制对文件的访问。

（7）<Directory>、<Files>、<Location>可以使用shell风格的通配符。<DirectoryMatch>、<FilesMatch>、<LocationMatch>则可以使用复杂的perl风格的正则表达式。





（8）容器中的指令会按如下顺序合并：<Directory>、<DirectoryMatch>、<Files>和<FilesMatch>（目录会重复，但文件不会重复，地址也一样）、<Location>和<LocationMatch>。

另外，除了<Directory>，容器会按出现在配置文件中的顺序依次处理。而<Directory>会按匹配到的目录由短到长被依次处理。如，<Directory /var/web/dir>会先于<Directory /var/web/dir/subdir>处理。如果有同一目录的多个<Directory>容器，则也会按出现在配置文件中的顺序处理。

位于<VirtualHost>容器中的配置会在外部对应容器配置处理完后再处理，即，虚拟主机配置会覆盖主服务器配置。

当请求是由mod\_proxy处理的时候，<Proxy>容器将会在处理顺序中取代<Directory>容器的位置。

（9）有的指令只能出现在某些容器中，如AllowOverride指令只能出现在<Directory>段中。Options中的FollowSymLinks和SymLinksIfOwnerMatch只能出现在<Directory>或.htaccess文件中。Options指令不能用于<Files>和<FilesMatch>段。

* + 1. 全局配置

|  |  |
| --- | --- |
| 服务器标识 | |
| ServerName | （1）服务器标识自身的主机名和端口号，主要用于创建重定向URL。  （2）没有设置，服务器会从IP中反向查询来推断主机名（跟DNS不同，DNS中设置了主机名，但它只是针对客户端做ip解析，跟服务器没有任何关系。因客户端访问header中有host，如果ip和自己相同，就可以推断自己的host）。  （3）如果ServerName中没有指定端口号，服务器会使用接受请求的那个端口。  （4）为了加强可靠性和可预测性，应该设置ServerName。  （5）如果使用的是基于域名的虚拟主机，在<VirtualHost>段中的ServerName将是为了匹配这个虚拟主机，在"Host:"请求头中必须出现的主机名。 |
| ServerAdmin | 管理员邮件地址。 |
| ServerSignature | 主要用于返回错误信息时输出的页脚。 |
| ServerTokens | response中header的Server，默认为Full。 |
| 文件定位 | |
| ServerRoot | 服务器所在的目录，LoadModule、Include、conf、logs等都是相对于此目录。 |
| DocumentRoot | （1）如果没有使用Alias指令，服务器会将请求中的URL附加到DocumentRoot后面以构成指向文档的路径。如设为/usr/web，http://www.my.host.com/index.html的访问就会指向/usr/web/index.html。如果不是绝对路径，则被假定为相对于ServerRoot的路径。  （2）后面不要有/。 |
| ErrorLog | 错误日志目录。 |
| 限制资源的使用 | |
| LimitRequestBody | 限制客户端发送的HTTP请求体的最大字节长度，如果请求超出限制，服务器会回应一个错误，如将上传文件的大小设置为100K。 |
| RLimitCPU | （1）限制Apache子进程派生的进程占用CPU的最大秒数和资源。  （2）限制的是Apache子进程服务所衍生出的进程，如CGI脚本和SSI执行命令，而不是Apache子进程本身。 |

* + 1. URL到文件系统的映射

（1）默认操作：将URL路径附加到DocumentRoot指定的文件路径后面。

（2）对DocumentRoot以外的文件访问，可以：

1）使用符号链接，将外部文件链接到DocumentRoot目录下，需要在对应的目录中的Options指令中设置FollowSymLinks或SymLinksIfOwnerMatch。

2）使用Alias指令，将文件系统的任何部分映射到网络空间中，如：



可以将http://www.example.com/docs/dir/file.html映射为/var/web/dir/file.html。

3）使用ScriptAlias指令，和Alias相似，目标路径下的所有文件会被视为CGI脚本。

4）AliasMatch和ScriptAliasMatch指令可以实现基于正则表达式的匹配和替换，以提供更大的灵活性。



（3）可以用Redirect和RedirectMatch指令实现重定向。

（4）如果URL到文件系统匹配失败，可以使用mod\_speling模块自动纠错（但没有必要）。如果没有使用mod\_speling，Apache则会返回一个出错信息页面，状态码为404，具体见自定义错误响应。

* + 1. 环境变量

（1）环境变量可用于控制如日志记录、访问控制等操作，还可以作为和如CGI脚本等外部程序沟通的机制。

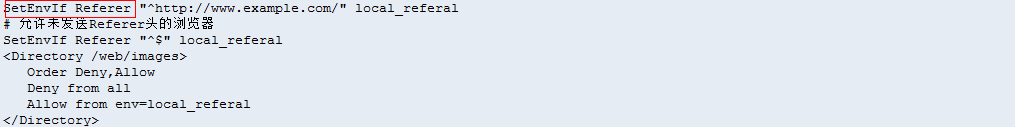
（2）可以使用如下指令设置环境变量：

|  |  |
| --- | --- |
| BrowserMatch  BrowserMatchNoCase | 基于User-Agent头有条件地设置环境变量 |
| SetEnv  SetEnvIf  SetEnvIfNoCase | 设置环境变量 |
| PassEnv | 传送shell中的环境变量 |
| UnsetEnv | 删除一个环境变量 |

（3）使用1：避免将对图片的请求记入访问日志中。



（4）使用2：防图片盗链。



* + 1. 内容协商

如果服务器中的某种资源有好几个不同的版本，如fr、en等，当浏览器发送请求过来的时候，服务器会提供一个最合适的版本给浏览器，这就是内容协商。

* + 1. 地址和端口绑定

（1）Listen指令指示服务器只监听特定端口或地址的请求。

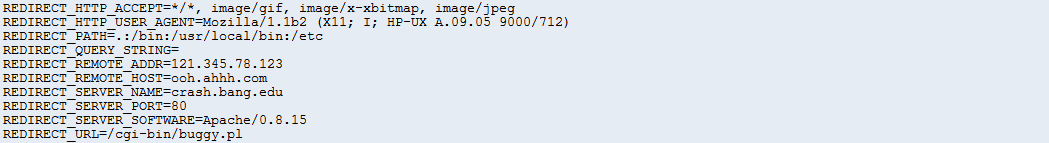




（2）Listen指令只是告诉主服务器去监听哪些地址和端口。如果有<VirtualHost>指令，则服务器会对不同的地址和端口作出不同的响应。

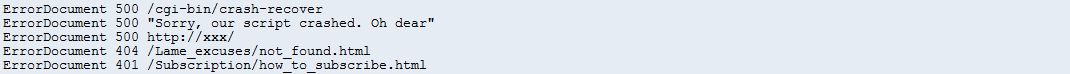
* + 1. 自定义错误响应

（1）如果出现服务端错误，Apache定义了一些环境变量可用于重定向。



（2）使用ErrorDocument指令定义重定向内容。格式如下：





* + 1. 日志

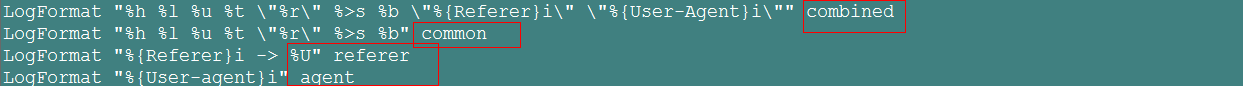
（1）ErrorLog指令定义错误日志文件和记录级别。



默认格式为：



（2）访问日志记录服务器所处理的所有请求，由CustomLog指令指定文件位置，LogFormat指令指定日志内容。





LogFormat最后是格式的别名，保存到CustomLog指定的文件中。%h为客户端ip，%t时间，%r请求，%>s状态码等，最后保存的格式如下：



（3）日志滚动：通过管道将访问记录和出错信息传递给另一个进程，而不是写入一个文件。管道日志进程由其父进程Apache httpd产生，并继承其权限，这意味着管道进程通常是作为root运行的。



* + 1. 基于主机的访问控制

（1）指令应该放入主配置文件（通常在<Directory>内），或者针对单个目录的配置文件（.htaccess文件）中。

（2）Allow和Deny指令可以允许或拒绝来自特定主机名或主机地址的访问。Order指令告诉Apache处理这两个指令的顺序。

（3）Order顺序为Deny,Allow时，Deny放在Allow后判断，默认允许所有访问。Allow,Deny为Allow指令在Deny指令之前被评估，默认拒绝所有访问。

如下，除dev.example.com外的主机访问都被拒绝。



* + 1. 其他常用指令

（1）ScriptAlias

基本同Alias，也是映射文件，但指定文件为cgi脚本。



（2）Options

配置在特定目录中可以使用哪些特性。格式为：。option可以为None，将不启用任何额外特性，或设置为以下选项中的一个或多个：1）All：默认设置，除MultiViews之外的所有特性。2）FollowSymLinks：允许在此目录中使用符号连接。3）Includes：允许使用mod\_include提供的服务器端包含。4）Indexes：如果一个映射到目录的URL被请求，而此目录中又没有DirectoryIndex，那么服务器返回由mod\_autoindex生成的一个格式化后的目录列表。5）

+和-用于目录在多处被设置时，将配置合并。如下，无论之前对/web/docs/spec如何配置，这里去掉Indexs特性。



* 1. 虚拟主机

虚拟主机是指在一个机器上运行多个网站。

**1.基于主机名的虚拟主机**

（1）即，1个IP地址，多个网站。

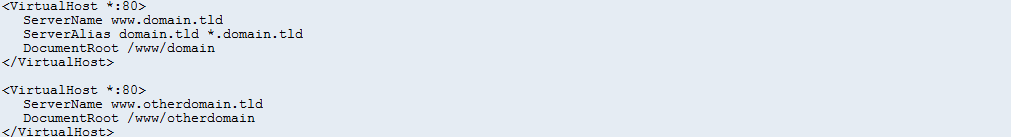
（2）客户端必须发送Host头，除了特别老的浏览器外都没问题。

（3）基于域名的虚拟主机无法成为SSL安全服务器。

（4）必须使用NameVirtualHost指定服务器IP地址和可能的端口来使虚拟主机接受请求。如果服务器上所有的IP地址都会用到，你可以用"\*"作为NameVirtualHost的参数。



（5）取消中心主机：使用<VirtualHost>容器配置。如，在1个IP地址上位www.domain.tld和www.otherdomain.tld提供服务。



也可以用1个固定的IP地址来代替<VirtualHost>指令中的\*号，如希望在1个IP地址上运行1个基于域名的虚拟主机，而在另外1个IP地址上运行1个基于IP的或是另外一套基于域名的虚拟主机。

其中，ServerName和DocumentRoot所包含的内容应该与全局的ServerName和DocumentRoot保持一致。

**2.基于IP地址的虚拟主机**

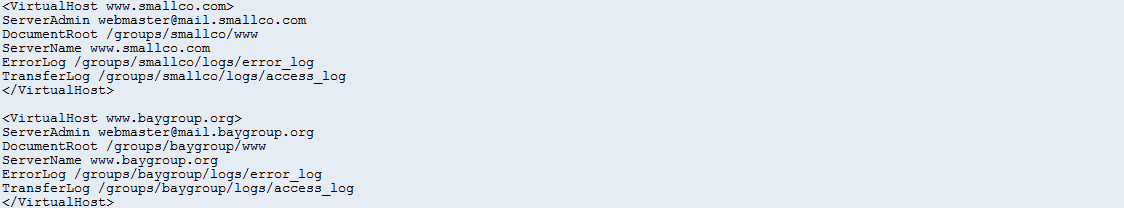
（1）每个基于IP的虚拟主机必须拥有不同的IP地址，可以通过配备多个真实的物理网络接口来达到这一要求。

（2）有2种配置方法实现：为每个虚拟主机运行不同的httpd守护进程，或者用同一个守护进程来支持所有虚拟主机。

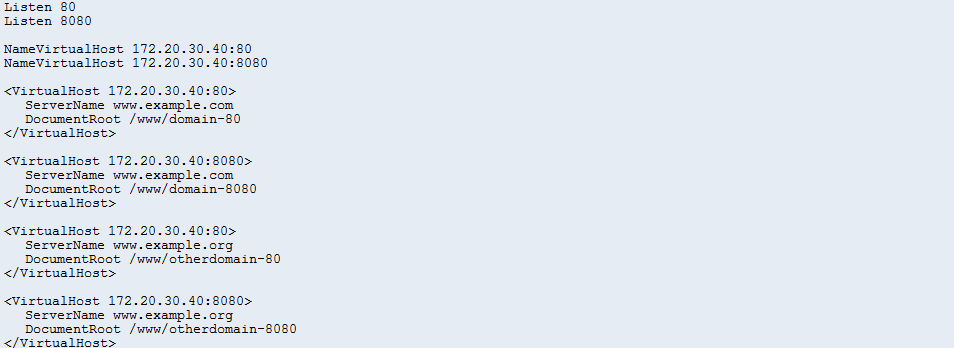
（3）为每个虚拟主机创建一个不同的httpd安装。每次安装都在配置文件中使用Listen指令指定守护进程伺服的IP地址(或虚拟主机)。



（4）配置拥有多个虚拟主机的单一守护进程，在VirtualHost容器中，为每个虚拟主机配置不同的ServerAdmin、ServerName、DocumentRoot等。



（5）使用1个IP的不同端口伺服多个域名。



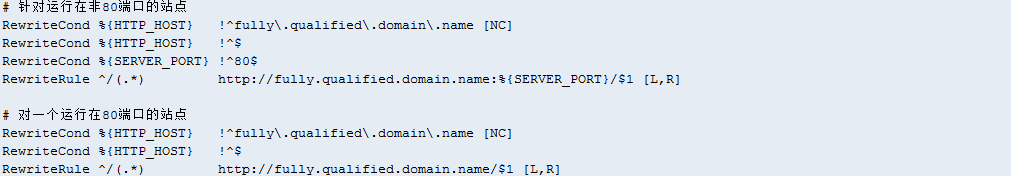
* 1. URL重写

（1）对不规范的URL执行一个外部HTTP重定向，以改变它在浏览器地址栏中的显示及请求。



如上，第1个可以将/u/user替换/~user，第2个可以修正/u/user所遗漏的后缀斜杠。

（2）强制使用特定的主机名代替其他名字，如，强制使用www.example.com代替example.com。



（3）后缀斜杠的问题：如请求是/~quux/foo而不是/~quux/foo/，服务器会去找一个叫foo的文件，而它是一个目录，所以就报错。对此，必须使用一个外部重定向，因为如果仅作一个内部重写，可能只对目录页面有效，而对内嵌有使用相对UR请求无效。



* 1. .htaccess文件

（1）.htaccess是在整个网站文件目录树结构中的特殊文件，用来进行分散配置。

（2）.htaccess文件的语法与主配置文件相同。

（3）因每次请求都会读取.htaccess文件，因此修改会立即生效。

（4）从性能角度考虑，应尽可能的将配置放在主配置文件中。

（5）将AllowOverride设置为none可以完全禁止使用.htaccess文件。

* 1. 优化
     1. MPM

**1.概要**

（1）MPM（Multi Processing Module）：多路处理模块，有prefork、worker和event三种模式可以选择，其中event仅适用于Apache2.4。在编译apache时，必须选择也只能选择一个MPM。

（2）在一些发行版的Apache安装时，可能不仅安装了prefork模式的httpd，也可能安装有worker/event模式的httpd供用户根据实际需要来选择。



（3）可以使用-l查看被静态编译到核心的MPM。



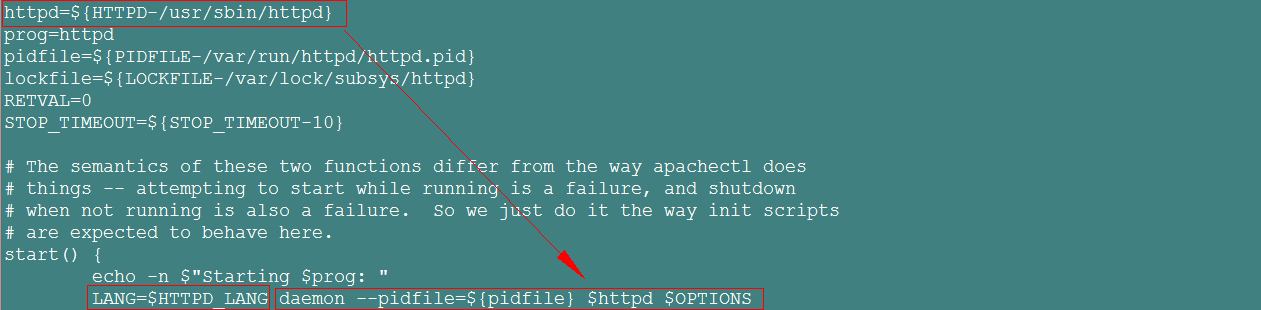
（4）在/etc/init.d/httpd脚本中，加载了/etc/sysconfig/httpd配置文件。



/etc/sysconfig/httpd中有：



/etc/init.d/httpd脚本的start的启动函数第2行是1个简单命令，前面设置LANG变量，后面调用daemon函数。httpd变量赋值中使用了参数扩展，如果/etc/sysconfig/httpd中没有定义HTTPD，则使用默认的httpd。



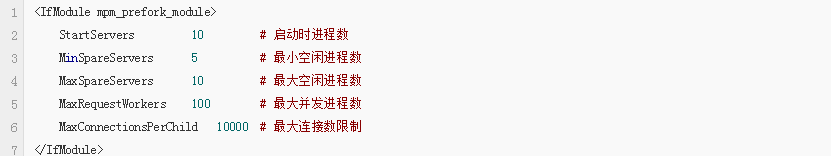
综上，可以在/etc/sysconfig/httpd中通过设置HTTPD设置MPM模式。

**2.prefork模式**

（1）完全不使用线程，使用整个进程用来处理每个HTTP请求，即每个子进程1个时点只处理1个请求。启动之初就预先fork一些子进程等待请求，目的是为了减少频繁创建和销毁进程的开销。

（2）优点：成熟稳定，兼容所有新老模块，也不需要担心线程安全问题。缺点：占用较多系统资源，消耗更多的内存。

（3）配置



MaxConnectionsPerChild：用于限定进程生命周期内处理的最大请求数目。达到该数目后，进程将死掉。如果设置为0，表示没有限制。该参数的意义在于避免可能存在的内存泄露带来的系统问题。

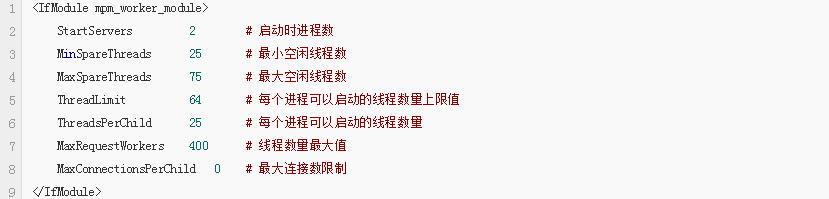
MaxRequestWorkers：影响并发性能最重要的参数。可以通过top查看apache进程占用的资源，主要看%CPU和%MEM这2个指标，如，每个进程CPU占用率不超过1%，每个进程的内存占用率不超过2%，考虑内存限制，比较合适的apache进程数量为50个。另外，可以在实际运行中观察占用情况，如果没有出现卡顿，可以逐步调高1.5倍。

**2.worker模式**

（1）使用多进程和多线程的混合模式，预先也会fork几个子进程（数量比较少），然后每个子进程再创建一些线程。当有每个请求过来，会分配1个线程服务。因线程通常会共享父进程的内存空间，在高并发场景下，比prefork表现会更优秀一些。

（2）如果一个线程异常，会导致父进程连同其他正常的子线程都挂掉，因此需要使用多个进程再加多线程，这样即使某个线程出现异常，受影响的只是Apache的一部分服务，而不是整个服务。

（3）配置



ThreadsPerChild上限值为ThreadLimit。如果修改了ThreadLimit只能先stop，然后start，不能使用restart。

比较重要的配置是ThreadsPerChild和MaxRequestWorkers，ThreadsPerChild使用默认值即可，如果并发量较大，可以考虑加大这个值。

worker模式下，因任务处理负载会分发给php-fpm处理，apache主要承担建立和保持连接的工作，并不消耗太多资源，因而理论上MaxRequestWorkers可以非常大。但如果MaxRequestWorkers远超系统的请求处理能力，会导致大量请求排队，产生较大的延迟。所以，MaxRequestWorkers可以在开始时配置稍大，如可以设置为prefork模式下数倍，然后根据线上峰值延迟不断调整这个数值。

MaxRequestWorkers必须是ThreadsPerChild的整数倍，否则apache会提示调整到一个相近的值。

**3.event模式**

（1）无论是prefork还是worker，对于keep-alive长连接请求，即使中间几乎没有请求，进程或线程也需要一直等待直到超时才会被释放。

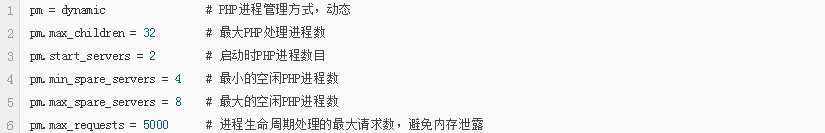
（2）event模式就是为克服这种缺陷而产生的。和worker模式类似，event模式也使用进程和线程，不同在于event模式会有一个专门的线程来管理keep-alive类型请求，当有真实请求过来的时候，会将请求传递给服务线程，执行完毕后，又允许释放服务线程，从而增强高并发场景下的请求处理能力。即，event模式会为每个请求创建一个线程，而不是为一个HTTP连接创建一个线程。

（3）新版本的apache2.4，event模式也能用于https请求。

（4）配置同worker模式。

* + 1. php-fpm优化

在worker和event模式下，PHP一般运行在fastcgi环境中，因此决定系统性能的主要是PHP的处理能力。默认的php-fpm配置文件位于/etc/php5/fpm/pool.d/www.conf。通常有以下选项可以配置：

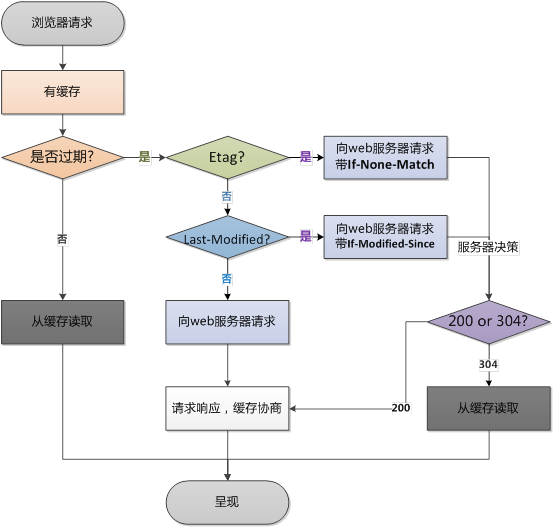


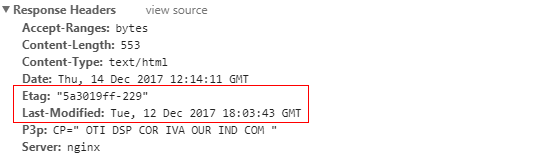
比较重要的配置是max\_children参数，决定了PHP处理请求的能力。该值也需要根据每个php-fpm所占用的CPU和内存来设置，同样通过top命令查看php-fpm进程消耗的资源，然后计算出1个大概值。

* + 1. 缓存

**1.页面缓存**

（1）访问页面流程



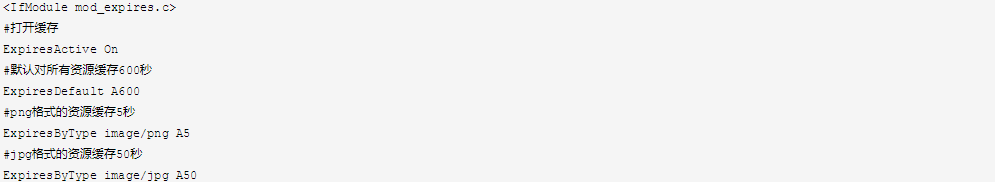


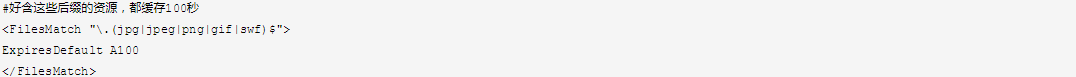
（2）Expires、Cache-Control、Last-Modified、ETag是HTTP/1.1协议中关于网页缓存的几个字段。前2个用来控制缓存的失效日期，后两个用来验证网页的有效性。使用HTTP/1.0的缓存将忽略Expires和Cache-Control头。

（3）Expires：声明了一个网页或URL地址不再被浏览器缓存的时间。超过这个时间后，浏览器就应联系原始服务器。Apache默认不会在响应头添加这个字段。浏览器一般会根据文件类型和Last-Modified字段推断一个合适的失效时间并存储，通常为3天。

Apache可如下配置实现：

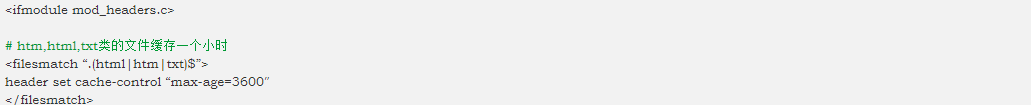








Cache-Control：可以声明多个字段，如no-cache,must-revalidate,max-age=0等，用来指示页面被缓存最大时限，如何被缓存的，如何被转换到另一个不同的媒介，以及如何被存放在持久媒介中等。max-age由mod\_headers模块配置。



Last-Modified和ETag：用于询问服务器页面是否已经更改。服务器判断是否有更新信息，如果没有，就返回304（Not Modify），如果有更新，返回200和更新的页面内容，并且携带新的ETag和Last-Modified。两者区别为Last-Modified精确到秒，而ETag是根据文件的索引节、大小和最后修改时间计算Hash后得到的，更精确。

* + 1. 内容缓存（apache服务端）

（1）Apache有mod\_file\_cache和mod\_cache两个模块实现缓存。

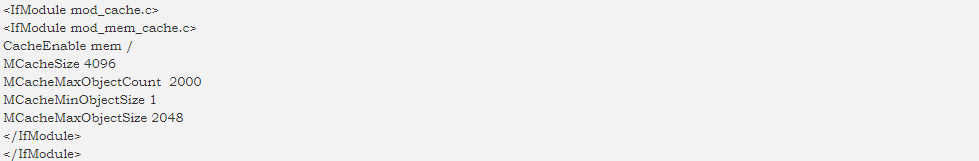
（2）mod\_file\_cache缓冲区负责维护Apache启动时的文件内容。当一个存在于该模块缓冲区中的文件被请求时，该请求将被拦截并用缓冲区中的内容为其提供服务，主要用于缓存不常修改的本地静态文件。





mod\_file\_cache指示Apache在启动时将一个静态文件的内容映射到内存中。

（3）mod\_cache的缓冲区机制：当服务于一个请求时，如果之前url内容未被缓存，则缓冲模块将会判断该内容是否可以被缓存。有2种缓存方式：硬盘缓存，由mod\_disk\_cache模块实现；内存缓存，由mod\_mem\_cache实现。



上面的含义：CacheEnable：启用缓存，使用基于内存的方式存储。MCacheSize：缓存数据最多能使用的内存，单位是kb，默认是100kb。MCacheMaxObjectCount：在内存中最多能存储缓存对象的个数。MCacheMinObjectSize：单个缓存对象最小值，默认为1bytes。

（4）mod\_disk\_cache会为被请求的URL创建一个22字符的哈希值。该哈希值包含了该URL的主机名、协议、端口、路径、CGI变量，以确保多个URL不会发生碰撞。

（5）什么可以被缓存？必须满足如下条件：1）必须为该URL启用了缓冲。2）应答必须位200、203、300、301、410状态码。3）请求必须是一个GET请求等。

* + 1. 页面压缩

（1）必须启用以下两个模块：



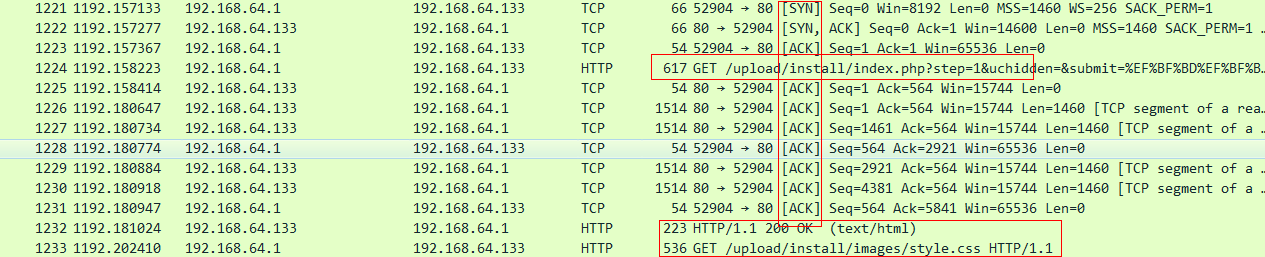
（2）配置



* + 1. keepAlive

（1）HTTP1.0中，每个http请求都要打开一个tcp连接，并且使用一次之后就断开这个tcp连接。HTTP1.1可以实现一次连接、多次传输功能。

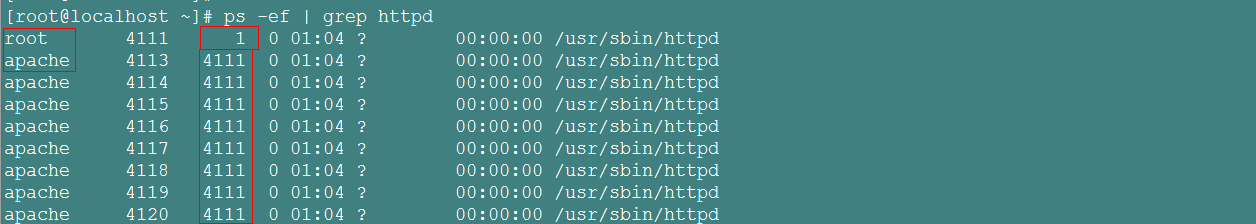
但是，如果1个页面要请求不同资源，则会建立不同连接。



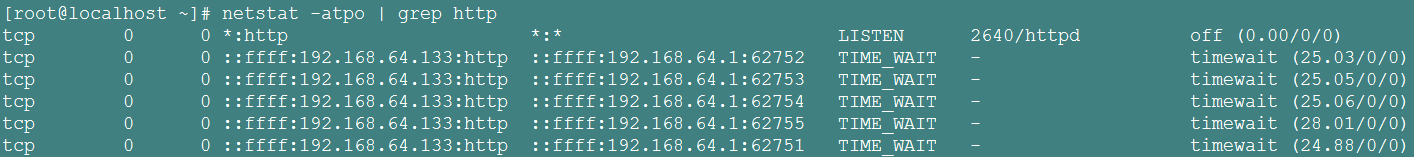
（2）keepAlive使用场景：如果使用mpm\_prefork模式，应将KeepAlive设置为off。

* 1. 其他
     1. 查看

（1）查看httpd所有进程



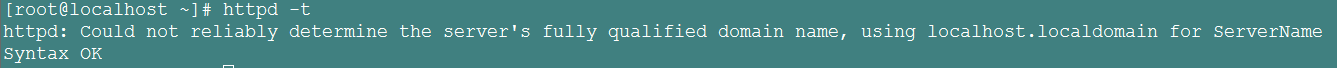
（2）查看apache详细链接情况



（3）查看请求80服务的client ip按照连接数排序



（4）检查是否配置正确



（5）检测某一台服务器的端口是否开启



* + 1. http协议概念

（1）URI：Uniform Resource Identifier；URL：Uniform Resource Locator。

（2）MIME：multipurpose Internet mail extension，互联网多用户邮件扩展。

（3）bnase64：将二进制数据编码成文本发送，并能够让接受方还原回原来的格式。

（4）响应报文代码

|  |  |
| --- | --- |
| 1XX | 信息性状态码 |
| 2XX | 成功状态码 |
| 200 | OK |
| 201 | Created OK |
| 3XX | 重定向状态码 |
| 301 | 永久重定向，在响应报文中使用首部“Location：URL”指定资源现在所在的位置 |
| 302 | 临时重定向，在响应报文中使用首部“Location：URL”指定临时资源位置 |
| 304 | Not Modified，条件式请求中使用 |
| 4XX | 客户端的错误 |
| 403 | Forbidden,请求被服务器拒绝 |
| 404 | Not Found，服务器无法找到请求的URL |
| 405 | Method Not Allowed,不允许使用此方法请求响应的URL |
| 5XX | 服务器类的错误 |
| 500 | Internal Server Error，服务器内部错误 |
| 502 | Bad Gateway：错误网关 |
| 503 | Service Unavailable，服务器此时无法提供服务，但将来可能可用 |

（5）HTTP首部

1）通用首部

请求和响应都可以使用的，有：

Connection：如http/1.0，Connectionkeep-alibe。Via：显示了报文经过的中间节点。Cache-Control：缓存指示等。

2）请求首部

Cilent-IP：请求端IP。Host：请求的主机名和端口号，虚拟主机环境下用于不同的虚拟主机。Referer：指明了请求当前资源的原始资源的URL。User-Agent：用户代理，使用什么工具发出的请求。

Accept首部：用户标明客户自己更倾向于支持使用的能力，如Accept：指明服务器能发送的媒体类型。Accept-Charset：支持使用的字符集。Accept-Encoding：支持使用的编码方式。Accept-Language：支持使用语言。

条件请求首部：Expect：期望的行为。If-Modified-Since：是否在指定的时间以来修改过此资源。

跟安全相关的请求首部：Authorication：客户端提交给服务端的认证数据，如账号和密码。Cookie：客户端发送给服务器端身份标识。

3）响应首部

Age：响应持续的时间。Server：向客户端标明服务器程序名称和版本。

协商首部：Accept-Ranges：对当前资源来讲，服务器所能够接受的范围类型。Vary：首部列表，服务器会根据列表中的内容挑选出最适合的版本发送给客户端。

跟安全相关的响应首部：Set-Cookie：服务器端在某客户端第一次请求时发给的令牌。Set-Cookie2：设置cookie。WWW-Authenication：质询，即要求客户端提供账号和密码。

4）实体首部

用于指定实体属性。Location：资源的新位置。Allow：允许对此资源使用的请求方法。

内容首部，如Contern-Encoding、Contern-Language、Contern-Length、Contern-type等。

缓存首部：ETag：实体标签。Expires：过期标签。Last-Modified：上一次的修改时间。

5）扩展首部

非标准首部，可能是由程序开发者创建的，如X-Forward-For等。

1. nginx
   1. 概要

**1.启动、停止**

（1）启动需要指定配置文件，如下：



（2）停止服务需要找到nginx的pid文件，然后向其传递信号。

从容停止：。

快速停止：。

强制停止：。

平滑重启：。

**2.配置文件**

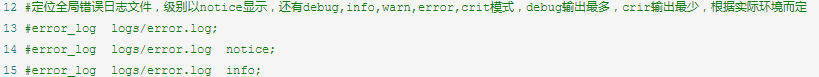
（1）配置文件结构：全局配置，http包含的http服务器配置，http内server为虚拟机配置。

（2）全局配置文件



安装时不会创建用户和用户组，需要手动创建，然后在这里配置。



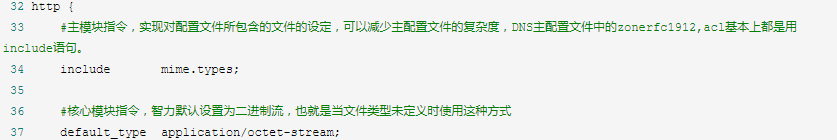


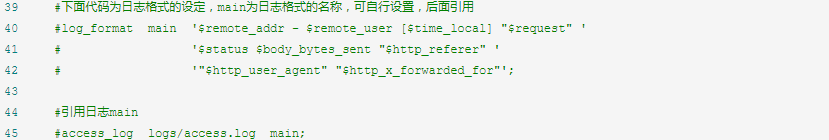


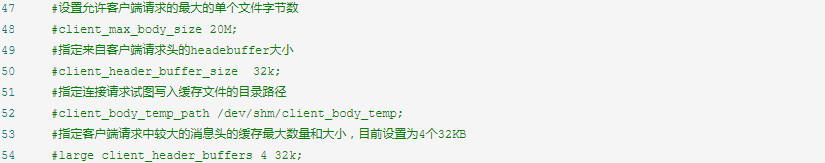


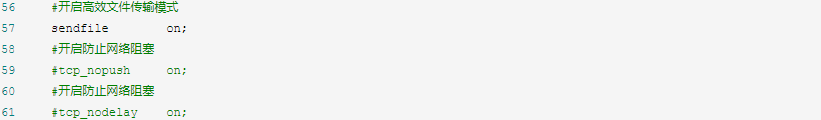


（3）http服务器配置

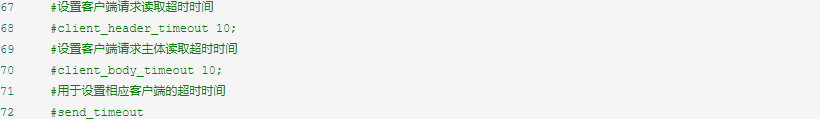












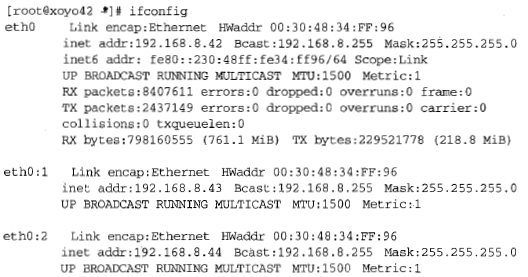


**3.虚拟主机配置**

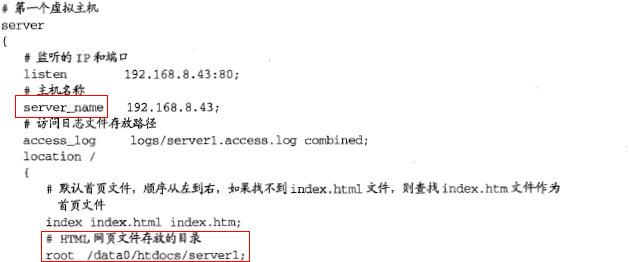
（1）基于IP地址

1）首先需要配置网卡别名，可见原来的网卡配置还在。





2）nginx配置



其他2个虚拟主机配置类似。

（2）基于域名



**4.日志文件配置和分割**

（1）log\_format：设置日志格式，语法：。即使不设置，也会有1个默认的名为combined的日志格式，日志格式如下：





（2）日志格式名称不能重复。

（3）如果服务器在反向代理、负载均衡服务器之后，则无法获取客户端ip，需要配置反向代理转发http头信息，追加X-Forwarded-For头。

（4）可通过access\_log off关闭访问日志记录。

（5）nginx不能像apache那样通过cronolog轮转日志，但可以通过如下方式切割日志：首先保存当前日志到另外1个文件：



然后通过kill -USR1传递信号给nginx主进程号，让其生成1个新的日志：

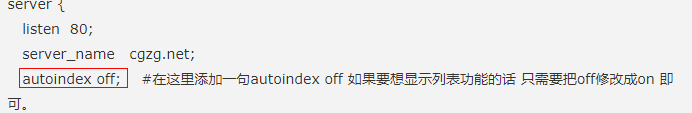


每天轮询还需要借助crontab实现。

**5.压缩输出**



**6.禁止显示目录**



**7.浏览器缓存**

（1）使用expires指令设置，语法格式：expires [time|epoch|max|off]，作用域为http、server、location。

（2）可以用来控制HTTP应答中的Expires和Cache-Control头信息。可以在time值中设置正数或负数，则Expires值为当前系统时间加上设置的time值计算而来。



* 1. 缓存服务

**1.概要**

（1）web缓存位于内容源服务器和客户端之间，当用户访问一个URL时，缓存服务器会去后端Web源服务器取回要输出的内容，然后，当下一个请求到来时，如果访问的是相同的URL，缓存服务器将直接输出内容给客户端，而不是向源服务器再次发送请求。

（2）Nginx缓存服务主要由proxy\_cache相关指令集和fastcgi\_cache相关指令集构成。proxy\_cache用于反向代理时，对后端内容源服务器进行缓存。fastcgi相关指令集主要用于对FastCGI的动态程序进行缓存。两者功能基本一样。

（3）需要安装ngx\_cache\_purge模块，在编译nginx时--add-module。

**2.proxy\_cache配置**

（1）相关指令含义

1）proxy\_cache zone\_name：用于设置哪个缓存区将被使用，zone\_name为proxy\_cache\_path指令创建的缓存区的名称。

2）proxy\_cache\_path path [levels=number]：用于设置缓存文件的存放路径。levels位指定该缓存空间有2层hash目录，第1层目录为1个字母，第,2层目录为2个字母。

3）proxy\_cache\_methods [GET HEAD POST]：设置缓存哪些HTTP方法。

4）proxy\_cache\_min\_uses the\_number：缓存的最小使用次数，默认值为1。

5）proxy\_cache\_valid reply\_code [reply\_code...] time：对不同返回状态码的URL设置不同的缓存时间。

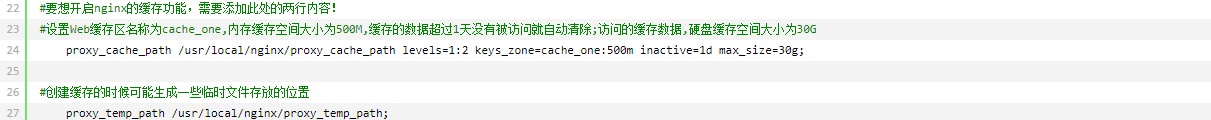
6）roxy\_cache\_key line：设置Web缓存的Key值，一般根据$host、$request\_uri等变量组合计算md5而来。

（2）配置

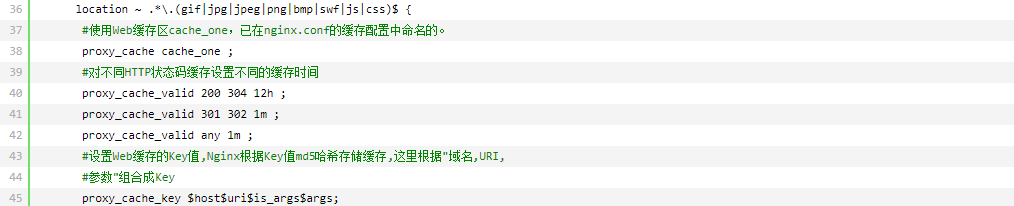
1）需要在同一分区下创建缓存目录。



2）在nginx.conf中，开启缓存功能。



3）配置缓存路径



4）配置清除路径



**3.fastcgi\_cache**

（1）相关指令含义

1）fastcgi\_cache zone\_name：设置哪个缓存区将被使用，zone\_name的值为fastcgi\_cache\_path指令创建的缓存区名称。

2）fastcgi\_cache\_path path [levels=number] keys\_zone=zone\_name:zone\_size [inactive=time] [max\_size=size]：设置缓存文件的存放路径。

3）fastcgi\_cache\_methods、fastcgi\_cache\_min\_uses同上。

4）fastcgi\_cache\_valid reply\_code [reply\_code...] time：不同返回状态码的URL设置不同的缓存时间。

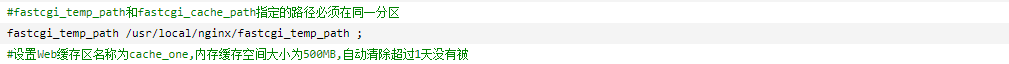
5）fastcgi\_cache\_key line：设置Web缓存的Key值。

（2）配置

1）设置目录



2）对扩展名为gif,jpg,jpeg,png,bmp,swf,js,css文件开启Web缓存，其他文件不缓存。







**4.ngx\_cache\_purge**

（1）用于清除FastCGI、proxy、SCGI、uWSGI缓存。因nginx默认安装就会带有反向代理的功能，需要清除指定URL的缓存。

* 1. 结合PHP安装配置

**1.FastCGI**

（1）过去CGI解析慢的原因：每次web服务器调度时会重新初始化，处理完之后就释放资源。

（2）FastCGI工作原理：Fastcgi会先启动1个master，解析配置文件，初始化执行环境，然后再启动多个worker。当请求过来时，master会传递给一个worker，然后立即可以接受下一个请求。当worker不够用时，master可以根据配置预先启动几个worker等着。当然空闲worker太多时，也会停掉一些，这样就提高了性能，也节约了资源。

（3）fastcgi是一个协议，php-fpm实现了这个协议。PHP的解释器是php-cgi，php-cgi只是个CGI程序。php-fpm的管理对象是php-cgi。php-fpm是php内核的一个补丁，使用--enalbe-fpm这个编译参数即可。

**2.安装**

（1）配置防火墙

（2）关闭selinux



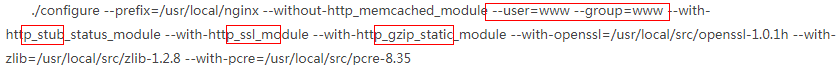
（3）下载源码包

软件源代码包存放位置为/usr/local/src，源码包编译安装位置为/usr/local/软件名。nginx源码包从http://nginx.org/download/下载对应版本。php从中国镜像地址http://cn2.php.net/distributions下载对应版本。

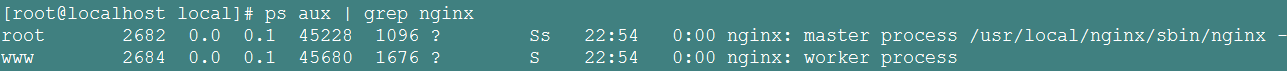
（4）yum安装依赖

可以直接使用yum安装mariadb5.6，不过centos6下最高只支持mysql5.2，要安装5.6，需要先下载1个repository，然后再安装时指定repository。安装完可能需要/usr/bin/mysql\_secure\_installation进行配置。

安装nginx可以先直接yum安装其依赖。nginx依赖包有：zlib和zlib-devel（压缩用）、openssl和openssl-devel、prce和prce-devel（支持nginx伪静态）。貌似不用指定路径。

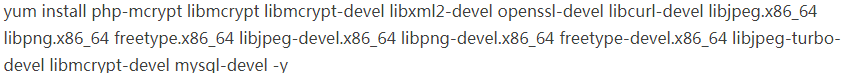


www这里可以先添加上，貌似之后可以直接通过修改配置文件达到同样效果。



nginx没有/etc/rc.d/init.d/nginx文件，需要手动添加。

安装php相对麻烦一些，可以先yum安装依赖。



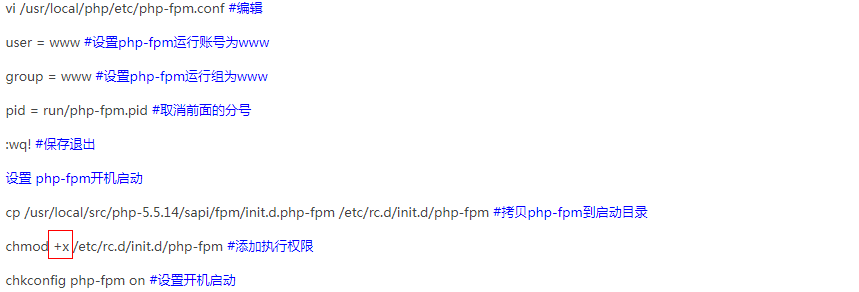
然后编译安装php，注意nginx只支持fastcgi，编译时需要开启fpm。



之后，还需要复制建立php.ini和php-fpm.conf配置文件。



在php-fpm.conf中php-fpm用户和用户组，并设置php-fpm开机启动。



修改nginx.conf，以让nginx支持php。



如果没有修改$document\_root，可能会出现File not found错误。

* 1. 反向代理

**1.概要**

（1）负载均衡：将客户端请求平均分配到服务器阵列中的每台服务器。

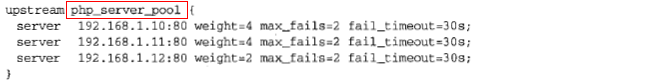
（2）反向代理：代理服务器接收请求，然后将请求转发到内部网络上的服务器，并将内部网络服务器得到的结果返回给客户端。通常的服务器是指代理内部网对internet的连接请求。

（3）负载均衡常用方法有：1）DNS轮询，但因dns缓存原因，即使服务器出现错误仍然会作为接受请求的服务器，以及转发时不能区分服务器负载情况等缺点。2）硬件方法。3）LVS。4）nginx。

**2.配置**

（1）配置服务器池

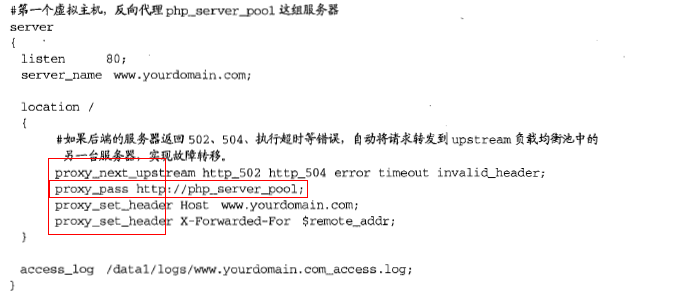
upstream指令用于设置服务器池。





（2）配置主机

proxy\_pass和fastcgi\_pass指令指向代理池。



（3）ip\_hash指令

ip\_hash指令用于设置客户端请求转发到后端服务器的算法。因为session的存在，可能会将某个客户端请求一直转发到某台服务器上，因而造成负载不均衡。



（4）server指令

如上图，server指令用于设置后端服务器名称和参数。

（5）upstream指令

如上，用于设置代理服务器池。

* 1. 实践

**1.安装nginx后，如果添加新模块？**



1. ssh
   1. 概要

**1.远程连接服务**

（1）服务器提供远程通过文字或图形界面方式登录、操控系统的界面（shell）。

（2）连接类型有：1）文字界面明码方式，如telnet，rsh等，目前已很少使用。2）文字界面密码，如ssh；3）图形界面。

**2.非对称加密技术**

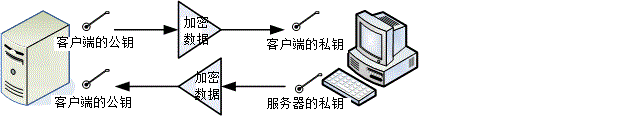
（1）对称加密算法在加密和解密时使用的是同一个秘钥。非对称加密算法需要两个密钥来进行加密和解密，即公钥和私钥，两个钥匙必须成对。

（2）实际上，非对称加密算法的公钥和私钥都可以互为加密、解密。任何一个作为公钥,则另一个就为私钥。之所以叫公钥，是因为它被分发出去，任何人都知道。非对称加密解决了传统对称加密一旦密码泄露导致的必须使用可靠传输途径传递密码的缺点，因为非对称加密不需要对公钥保密，只需要保管好私钥就行。

（3）数字签名：主要用于解决2个问题：消息是作者本人发出，消息完整。鉴于私钥只有作者本人持有，因此先对这段消息做1次hash（md5、sha1都可以），然后使用私钥加密这段hash作为签名，然后随消息一并分发出去。其他人就可以用公钥解密签名，从而用同样方式计算hash实现验证。

**3.ssh概要**

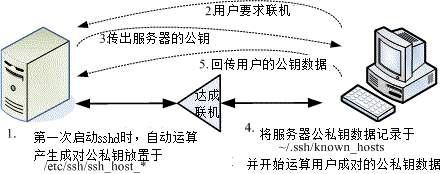
（1）因网路连线是双向的，因此每台主机都需要有对方的公钥。从客户端的角度来看，首先要取得服务器端的公钥，然后将自己的公钥发送给服务器端。最终，客户端使用服务器的公钥加密传输消息，使用自己的私有解密接收到的消息。



（2）SSH（Secure SHell protocol），通过资料封包加密技术，将等待传输的封包加密后再传输。SSH提供2种服务功能：1个类似telnet远端连线，即常说的ssh；另1个类似ftp服务的sftp-server，提供更安全的FTP服务。

目前有2种SSH协议版本，其中第2版本加上了联机检测机制，可以避免联机期间被插入恶意的攻击码，比第1版本更安全。

（3）SSH连接过程



1）服务器建立公钥文件：每一次启动sshd服务，sshd服务会检查/etc/ssh是否有公钥文件，如果没有，则计算生成公钥和私钥文件。显然，服务器的公钥、私钥不会在用户家目录下。

2）客户端主动联机要求：客户端需要使用适当的客户端程序来联机，如ssh客户端程序。

3）服务器传送公钥文件给客户端：明码传送，因为公钥本来就是给大家使用的。

4）客户端根据服务器发送过来的公钥随机计算自己的公私钥：客户端第1次连接到此服务器，则会将服务器的公钥保存到~/.ssh/known\_hosts目录下。

5）回传客户端的公钥到服务器端。

6）开始双向加解密。

**4.OpenSSL和OpenSSH**

OpenSSL是密码学库，主要用于应用层和传输层加密网络数据流。OpenSSH是使用OpenSSL实现的Remote Shell。OpenSSL提供2个库libssl和libcrypto，OpenSSH使用libcrypto实现的。

* 1. 配置使用

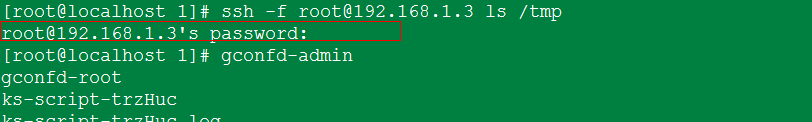
**1.启动SSH服务**

（1）Linux系统默认安装SSH所有需要的软件，包含OpenSSL和OpenSSH，而且sshd也是默认启动。

（2）如果服删除/etc/ssh/中的密码文件，ssh重启后会重新生成这些文件。

**2.客户端联机**

（1）使用ssh命令。



（2）使用scp命令基于ssh进行安全的远程文件拷贝。



**3.服务器设置**

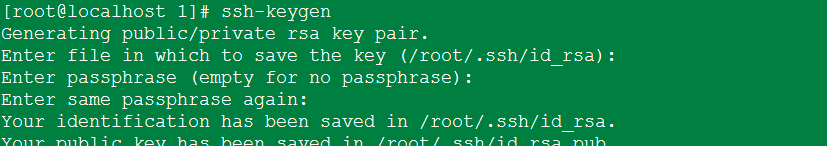
sshd服务器详细设置都在/etc/ssh/sshd\_config配置文件中。

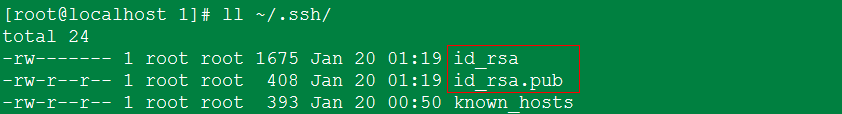


**4.设置不使用密码即可登录ssh的用户**

（1）原理：将client生成的公钥复制到server中AuthorizedKeysFile文件。当client连接server时，server从AuthorizedKeysFile中查找是否有对应的ip和用户，如果有，则向client发送1个用client的公钥加密过的随机字符串。client收到后解密，然后发送给server，比对正确就允许免密码登陆。

（2）客户端使用ssh-keygen命令生成秘钥。因openssl提供了多种密码算法（如rsa、dsa等），如果不指定，默认为rsa（一直按enter即可）。

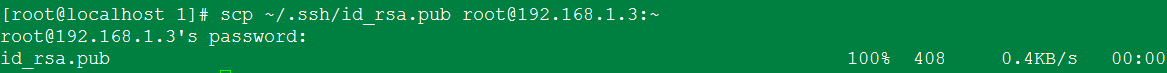




第1个文件为私钥，权限必须有rw，否则在未来密钥比对过程中，可能因被判定为危险而无法成功联机。第2个文件为公钥。注意~/.ssh文件权限必须是700。



（3）将客户端创建的公钥传到服务器端对应的用户（免密码登录的那个用户）家目录下。



（4）将公钥写入AuthorizedKeysFile中：首先在/etc/ssh/sshd\_config中定义放置公钥数据的文件名配置项AuthorizedKeysFile，然后将上传的id\_rsa.pub数据写入AuthorizedKeysFile中，并设置其权限为644。

1）先创建~/.ssh目录，设置权限为700。





2）再将id\_rsa.pub数据写入AuthorizedKeysFile。



之后该用户便可不用密码登录。



**5.简单的安全设置**

（1）配置/etc/ssh/sshd\_config，主要有：

1）禁止root账号使用sshd的服务。



2）配置nossh群组，禁止该群组中的用户使用sshd服务。

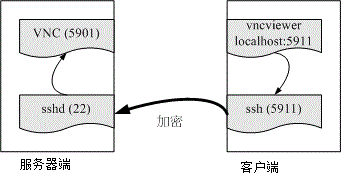
3）禁止testssh用户使用sshd服务。



（2）配置/etc/hosts.allow和/etc/hosts.deny，以及防火墙。

**6.通过ssh通道加密原本无加密的服务**

原理如下：



如，原本不加密的服务VNC，其端口为5901，可以先通过本地端的ssh联机到服务器的sshd，然后再通过sshd连接VNC服务器的5901端口。

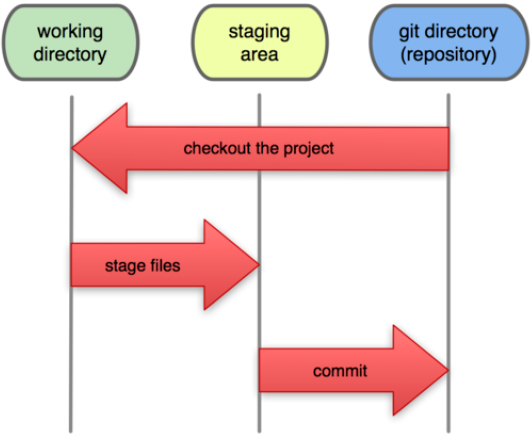
1. GIT
   1. 基础使用

**1.概念**

（1）git是1个分散式版本管理系统。分散式版本管理系统没有中央服务器，每个客户端都有1个完整的版本库，从而规避集中式版本管理系统由中央服务器保存所有历史版本，本地只有最新版本，导致的服务器故障风险及必须联网等问题。

（2）相对于其他版本管理系统会详细记录每个版本具体差异，git只关心项目的整体变化情况，更像是把变化的文件作快照，记录在1个微型的文件系统中。每次提交更新时，git会纵览一遍所有文件的指纹信息并对文件作一快照，然后保存一个指向这次快照的索引。如果文件没有变化，git不会再次保存，而只是链接到上次保存的快照。在保存之前，git会对所有数据计算校验和（SHA1算法），并将此结果作为数据的唯一标识和索引。

（3）3种工作状态



git管理的项目中，有3个区域：1）git本地数据库：即.git目录，是git用来保存元数据和对象数据库的地方。2）工作目录：即纳入版本管理的工作区域。3）暂存区域：又叫索引文件，是个简单的文件，也在git目录中。

基于git的工作流程为：1）在工作区域修改文件，此时为已修改状态。2）add过程：对修改文件做快照，保存到暂存区，此时为已暂存状态。3）commit过程：提交更新，将将保存在暂存区域的文件快照转储到git本地数据库中，此时为已提交状态。

**2.客户端配置**

（1）安装

可以直接使用yum安装客户端。

（2）配置文件

分布在3个地方：/etc/gitconfig、~/.gitconfig、项目的git目录中的配置文件。使用git config option命令来配置。

（3）配置用户名



--global选项配置的是~/.gitconfig文件。而只是配置项目的话去掉该选项即可。

（4）查看配置信息



**3.创建仓库**

（1）从当前工作目录中创建，执行git init命令，将会在项目路面下生成.git目录。

（2）从现有仓库克隆，使用git clone命令。git将会把项目历史上所有版本数据都clone到本地。clone时默认会创建目录，目录名为仓库的名称。



也可以自定义名称。



**4.记录相关操作**

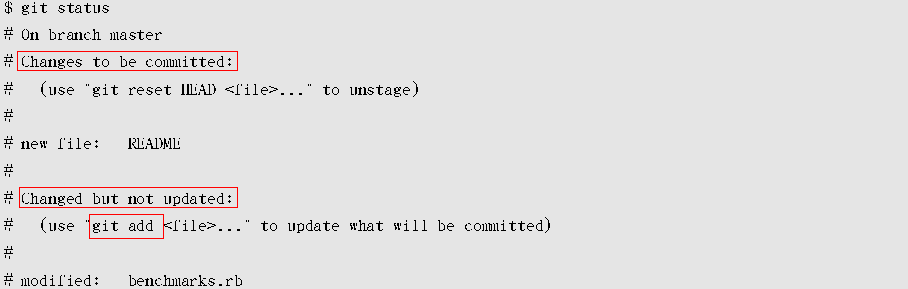
（1）跟踪新文件和暂存修改

工作目录下的所有文件只有两种状态：已跟踪和未跟踪。可以使用git add命令跟踪新文件。跟踪之后，文件进入暂存状态。





如果add之后，再次修改文件，可以再次使用git add暂存已修改文件。



（2）显然，如果add后的修改文件再次修改，再次add，暂存区的文件只保存最新的那1份。因为这个时候只是暂存，还没有纳入版本数据库管理，还不存在多个版本。如果使用reset HEAD也只是将暂存区内最近保存的那个文件覆盖工作区的文件。

（3）提交更新

使用git commit将暂存区中的内容提高到本地版本数据库中，纳入版本管理。显然，如果还没有使用add将修改保存到暂存区，那么commit就没法从暂存区找到修改内容。

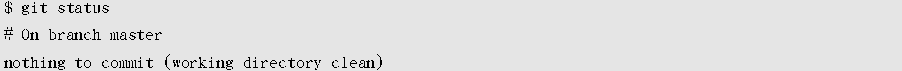
可以使用-m选项输入本次修改内容。如果不用-m，则进入默认文本编辑器，要求输入本次修改内容。

可以使用-a选项自动将跟踪文件暂存和提交操作一步操作。

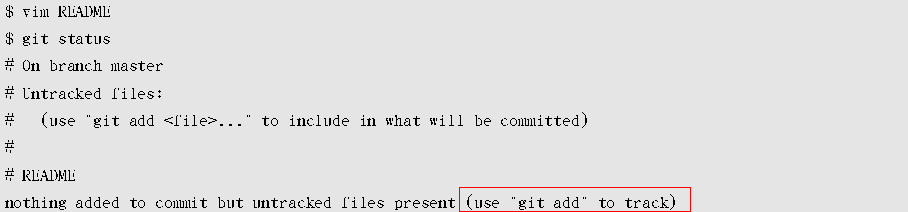
**5.查看状态**

（1）查看当前状态

使用git status查看文件状态。如果如下输出，则工作目录相当干净，即，当前没有任何跟踪着的文件，也没有任何文件在上次提交后更改过。



如果如下输出，表示当前有文件未被跟踪，可以通过add加入跟踪。



（2）查看更详细更新内容

使用git diff可以查看已修改还未暂存的内容。



add之后，可以通过diff --cache查看暂存区域和上次commit之间的差异。



以文件补丁格式显示具体添加和删除的行。

**6.移除文件**

（1）可以手动删除文件，然后使用add/rm将删除操作保存到暂存区内。或直接使用git rm同时将文件从本地删除，并将删除操作保存到暂存区内。使用git rm删除的文件必须已纳入版本管理，否则没法直接删除。



（2）如果文件已修改且保存在暂存区内，需要使用rm -f强制删除。



（3）如果只是移除跟踪但不删除文件，以便之后在.gitignore中补上，可以使用--cached选项。

以上操作都只是在暂存区内操作，之后还需要执行commit操作。

**7.移动文件**

（1）git并不会跟踪文件移动，而是当做删除、新增。

（2）可以使用git mv移动文件，但本质仍然是删除、新增。

**8.查看提交历史**

（1）使用git log可以将所有提交历史按从近到远排列。

（2）可以使用-n查看最近n次提交，可以使用-p查看每次提交详细差异。



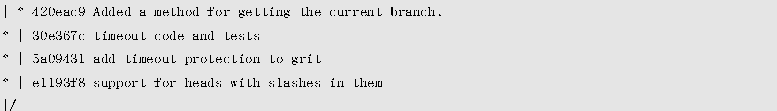
（3）可以使用--stat仅显示修改的行数。



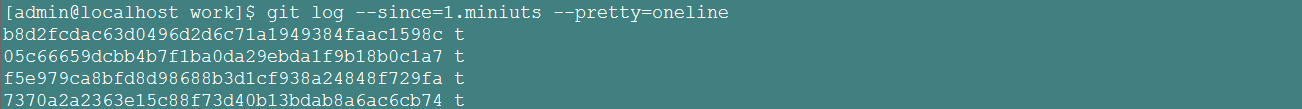
（4）可以使用--pretty=oneline|short|full|fuller展示不同信息格式。



（5）可以使用--pretty=oneline和--graph展示每次提交所在分支及分支分化衍合情况。



（6）可以使用--since和--until进行时间限制。



（7）可以使用--author搜索提交作者，--grep搜索关键字，但这些是or操作，可以使用--all-match执行and操作。

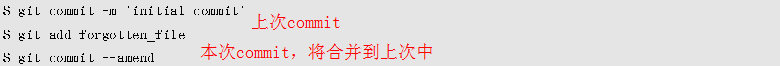
（8）可以使用-- 文件名查看某个目录、文件提交历史，但必须放在最后。

**9.撤销操作**

（1）取消add暂存。



（2）修改上次commit，即，如果上次commit注释有误，或想将本次add和commit合并到上次commit中，可以使用--amend选项。



（3）如果文件修改，但未add，想退回未修改状态（即上次版本状态）。

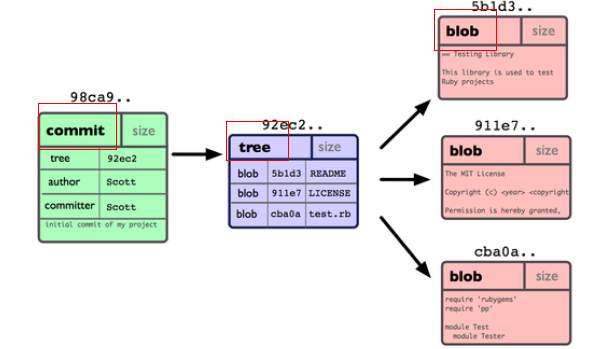


由于这次修改没有add和commit，没有纳入版本管理，因此一旦回退，修改内容就永远无法恢复。

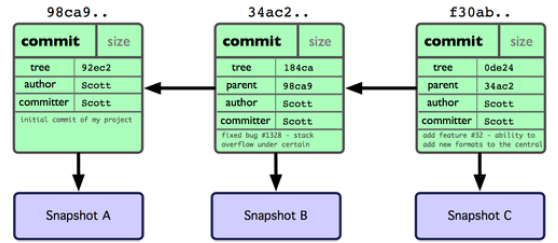
* 1. 分支管理

**1.概念**

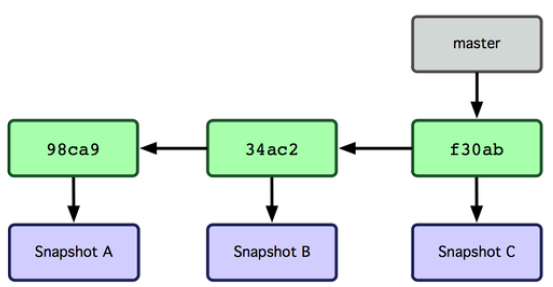
（1）每次add/commit，git会保存1个commit对象。git还会计算每个文件、目录的校验和，并保存为tree对象。同时还会将当前版本的文件快照以blob类型对象形式保存到git仓库中。commit对象内有1个指向tree对象的指针，tree对象有指向当前版本文件快照blob对象的指针，如下：



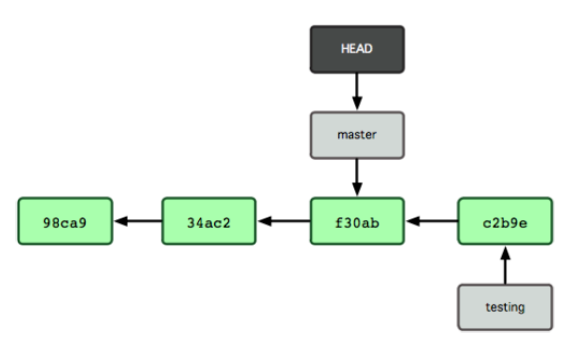
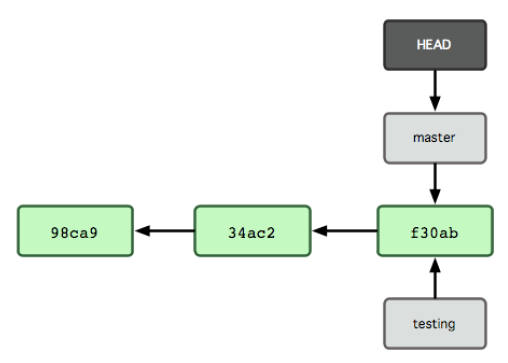
（2）除了第1次commit，之后的每次commit都有1个指向上次commit对象的指针。这种方式为分支合并时寻找恰当的合并基础提供了依据。



（3）分支就是1个个指向commit对象的指针。



（4）有1个特别指针HEAD，指向的分支就是当前指针。



（5）切换分支，不仅仅是将HEAD指针指向对应分支，还将工作区域切换到该分支指向的文件快照（用分支对应快照作为/覆盖当前工作区）。

（6）分支的本质仅是1个内容为所指向commit对象保存的40位校验和字符串的文件，因而能够实现毫秒级分支创建，切换等操作。

**2.基本操作**

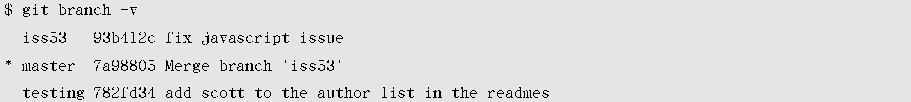
（1）可以使用git branch命令创建1个分支，但当前的工作分支仍然是原分支。



可以加上-b参数实现创建并切换分支。



可以加速-v显示各个分支最后一次commit信息。



（2）如果git branch命令不加任何参数，则会列出所有分支。分支前面的\*标识当前分支。



可以使用--merge显示已与当前分支合并的分支，使用--no-merged显示未与当前分支合并的分支。

（3）可以使用-d选项删除1个分支，如果分支尚有未合并内容，则不能直接删除，可以使用-D强制删除。



（4）可以使用git checkout切换分支。



【注意】如果当前分支内有未提交的内容，包括已修改或add到暂存区中的内容，如果此时切换分支，会将这些内容带到切换的分支工作区中。

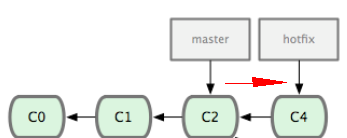


**3.合并分支**

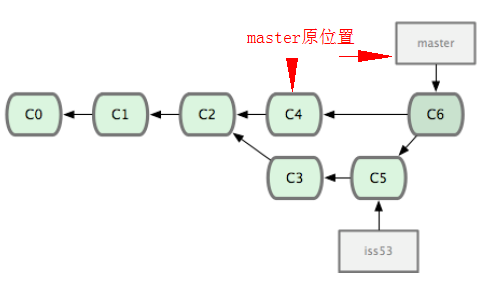
（1）使用git merge合并分支，如下，是将master合并到当前的dev中，合并之后当前分支不变。不过，由于dev分支是在master分支上做的修改，所以dev相对于master已是最新。如果要发布到master，则应该切换到master，将dev合并到master上。



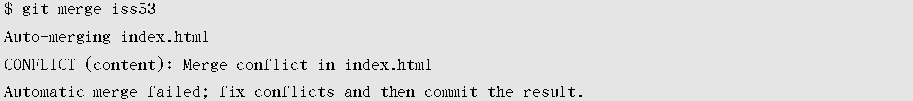
（2）快进合并：如果合并分支是当前分支的上游，git只需要把指针直接右移，顺着一个分支走下去即可到达另一个分支，此时即快进合并。

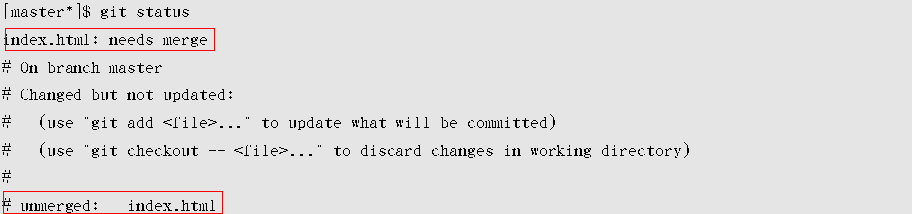


（3）基本合并：如下，分支有了分叉，要将iss53合并到master上。首先需要切换到master上，之后执行merge操作。此时，git会对两个分支的末端（C4和C5）和它们的共同祖先（C2）进行1次简单的三方合并计算。git会自己选择哪个版本作为最有合并基础。



（4）冲突合并：如果两个分支修改了同一文件的同一部分，则会产生冲突，无法直接合并。此时，git会提示冲突，将所有未解决冲突的文件以未合并形式列出。





此时，冲突文件会列出冲突，=======隔开的上半部分是HEAD（即当前分支master）。解决冲突之后可以使用git add标记为已解决。之后就可以使用commit提交。

* 1. 远程仓库和远程分支

**1.概念**

（1）远程仓库是托管在服务器端（remote）端的项目仓库（repository）。remote上可能有多个repository。每个repository可以理解为一个项目。每个repository下有多个branch。

（2）clone项目时，会执行如下操作：

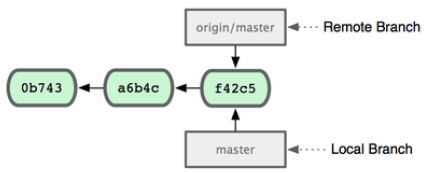
1）创建1个远端库的本地副本。

2）在本地创建1个标识origin，标识所clone的远程仓库。注意，远程服务器上的仓库不能在本地直接删除，只能在服务器上删除。这里的origin只是本地标识所clone的原始仓库的名称。

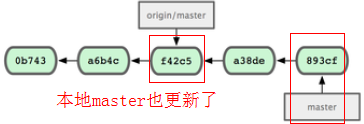
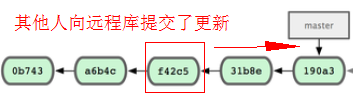


创建指向本地副本的origin/master分支（远程仓库名/分支名）指针，指向的位置与远端库master位置一样。origin/master因此可以代指远程仓库对应分支。

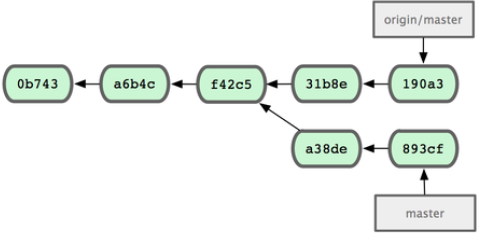
3）创建本地的master分支，指向位置同上。



（3）不能将分支切换到origin/master，不能在本地直接移动，只能使用push、pull等联网操作才能更新。如，clone之后，其他人向远端库提交了更新，但本地的origin/master并不会移动，仍然指向上次联网操作的位置，即使本地的master分支也更新了。



此时fetch远端库，将会获取远端库内容到本地，同时移动origin/master，但并不会和本地合并。



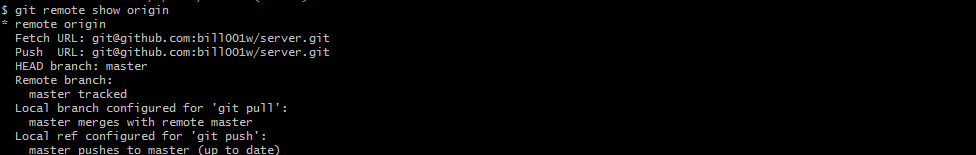
如果使用push远端库，意为将本地分支内容更新到远端仓库。此时，origin/master会移动到当前位置（如果远端有更新，会强制要求先pull更新到本地之后才能push）。

**2.远程仓库管理**

（1）可以用git remote命令查看每个远程库的简短名字。



加-v选项能显示对应的克隆地址。可以使用show查看远程仓库详细信息。



（2）使用如下方式添加远程仓库，可以自定义名称。

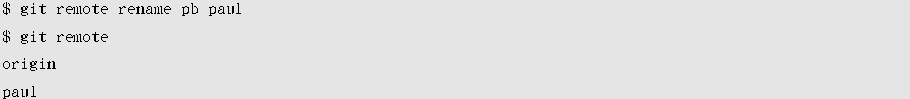


（3）使用fetch可以从远程仓库获取所有分支数据（同样，是本地的1个副本），但不会自动合并到本地分支。

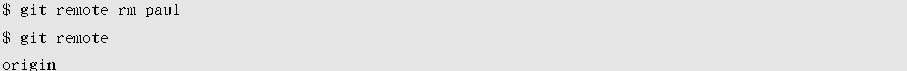


（4）可以使用推送数据到远程仓库。

（5）可以使用rename对远程仓库重命名。



（6）可以使用rm删除1个远程仓库。



**3.远程分支管理**

（1）因在clone远程仓库的时候，会自动创建本地master分支来跟踪origin/master分支，所有一开始就能运行pull和push。

（2）可以使用fetch拉取远程库所有分支到本地（本地的远程库）中，但没有合并，之后可以使用merge合并远程分支到本地分支。



（3）尽管我们获取了远程仓库包括所有分支在内的全部内容，但本地并不会自动创建出master之外的其他分支，需要手动创建。

如下，远程库中有1个serverfix分支，可以使用checkout -b在该远程分支基础上，创建分支并切换到该分支（可以自定义本地分支名）。



可以使用如下命令创建同名本地分支。



像上面这样根据远程分支检出的本地分支，称为跟踪分支。之后使用pull、push时，git会自动判断使用对应的本地分支与远程分支对应操作。

（3）pull可以用于同时拉取指定的远程分支（并非全部）并合并到指定的本地分支，等价于git fetch和git rebase。



如，拉取远程分支next和当前分支合并：



拉取远程分支和指定的master分支合并：



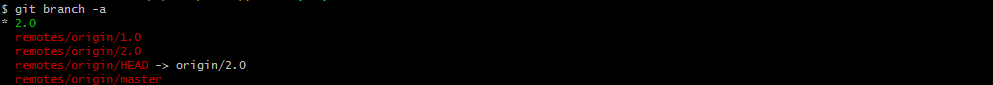
如果建立了追踪关系，可以省略远程分支名。如果当前分支只有1个追踪分支，可以省略远程库（这里是origin）。

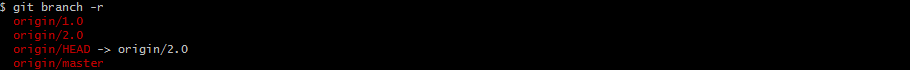
（3）push可以将本地支持更新远程支持。同样，如果建立了追踪关系，可以省略远程分支名。如果当前分支只有1个追踪分支，可以省略远程库。



push不会推送标签，除非使用–tags选项。

可以使用-a选项查看包括远程分支在内的所有分支，使用-r只查看远程分支。





* 1. 部署git服务

**1.概要**

（1）远程仓库通常只是1个纯仓库，即没有工作目录的仓库。直观上，只有.git目录。

（2）git可以使用本地传输、SSH协议、Git协议、HTTP协议4种方式传输数据。本地协议就是远程仓库在同一台主机上，通过文件系统来访问。

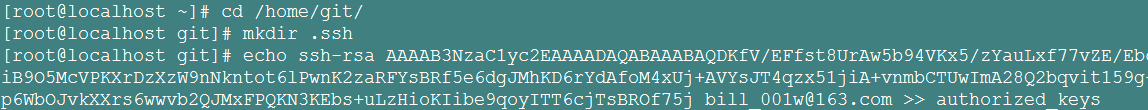
（3）因ssh是同时能够读、写且加密的传输协议，因此最常使用。最关键的地方是用户权限，如何让某些用户有权限访问仓库。方案1是为每个用户在仓库所在服务器上创建1个用户，这些用户都拥有对参考的读写权限。方案2是创建1个git用户，git用户有对仓库的读写权限。其他用户需要将自己的ssh公钥写入git账号的~/.ssh/authorized\_keys文件中。因为用户和git操作无关，参考log中仍然记录的是每个用户的操作。

**2.部署**

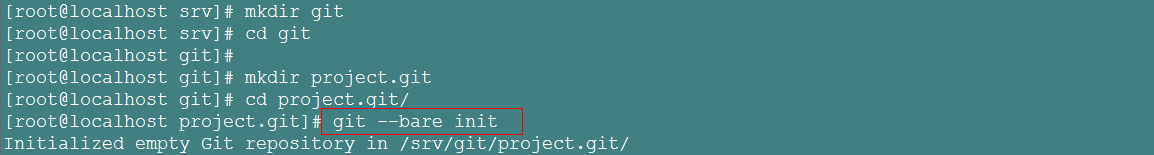
（1）创建git用户，且指定shell为/usr/bin/git-shell，目的是限制该用户的操作只能是git相关。



（2）进入git家目录，创建.ssh目录，并将需要操作的用户公钥写入authorized\_keys文件中。



（3）创建仓库，修改属主。





之后就可以使用了。

1. django

运行uwsgi需要在项目目录下，--file指定一个反射，没有指定使用默认。

