# fsck

## 路径：

\systemd-208\src\fsck\fsck.c

## 代码分析：

入参：一个或无。

使用日志打印级别：LOG\_TARGET\_AUTO (如果是tty设备则是控制台；否则JOURNAL或KMSG)

parse\_proc\_cmdline //从命令行/proc/cmdline中读取参数进行解析。

在非容器中：fsck.mode=auto/force/skip；以及sysv兼容模式。

1、获取udev设备：

有一个入参时候：从/sys/dev/block/目录中，根据stat查出来的设备的主次设备号查找

没有入参时候：虚拟化根路径不需要做check。从/目录，查找块设备

2、//获取文件系统类型，并获取/sbin/fsck.文件系统

3、执行 /sbin/fsck –a –T –l (如果不是根路径，则添加 -M) (如果/proc/cmdline中有fsck.mode=force ,则添加 -f)

## man手册：

1. man systemd-fsck

systemd-fsck-root.service //根文件系统的检查，但在initramfs中不检查根文件系统。

systemd-fsck@.service //用于所有的其它的文件系统，以及initramfs中的根文件系统。

如果/etc/fstab中的文件系统没有设置大于0的参数的话，这些服务在启动的时候started。根文件系统的检查优于其它文件系统。其它的文件系统可能并行检查，除非they are one the same rotating disk.

System-fsck不知道任何特定的文件系统的细节。对于每种文件系统类型(/sbin/fsck.\*)，仅仅简单的执行文件系统的检查。这个帮助仅仅决定是否文件系统实际是否被检查，基于上次检查的时间、挂载的数量，去清除umount等。

System-fsck将会将系统的的检查进度输送到控制台。如果对于无nofail的服务的某个文件系统检查失败的话，将会激活emergency模式，使用emergency.target隔离。

systemd-fsck 使用内核的命令行参数：

fsck.mode="auto", "force", "skip". 控制操作的模式。默认auto。必要时，可以确保文件系统的检查操作可以完成。"force" 无条件全文件系统的检查。Skip忽略任何文件系统的检查。

fsck.repair= "preen", "yes", "no". 控制操作的模式。默认是preen。将会自动修复（可以被安全修复的）问题。

"yes " 对于所有的fsck的问题都回答yes。"no" 对于所有的fsck的问题都回答no。

# fstab-generator

## 路径：

\systemd-208\src\fstab-generator\fstab-generator.c

## 代码分析：

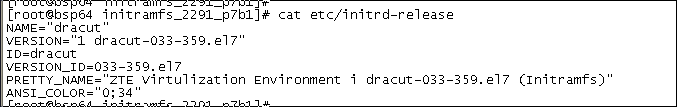
入参：三个或无。

容器不支持命令行参数解析。

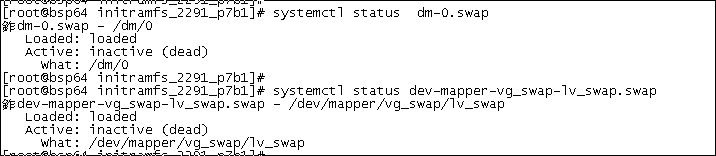
命令行参数解析： fstab=on/off

rd.fstab=

fstab. ||(小系统, rd.fstab. )

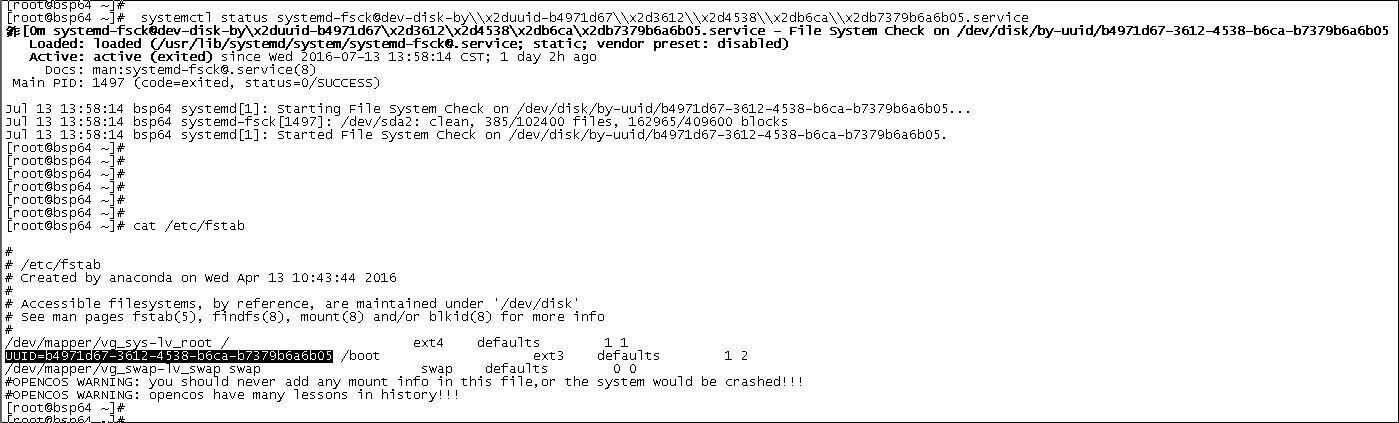


小系统中的处理：root= 以及rootfstype= 可能有多个，以最后一个优先。



add\_fsck功能实现：

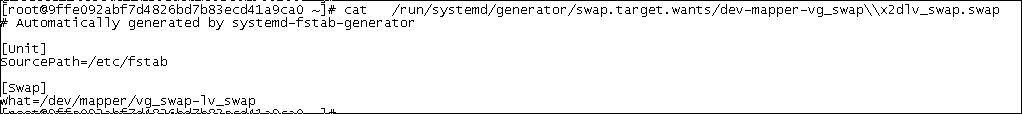
创建fsck相关的服务(/root 对应systemd-fsck-root.service；其它，对应systemd-fsck-$what.service)



根目录生成的挂载点：



Swap生成的：



## Man手册：

man systemd-fstab-generator

systemd-fstab-generator 是一个生成器，可以在启动初期以及system管理单元的配置加载时，将/etc/fstab的内容转换成本地的systemd单元。（mount和swap）

# rc-local-generator

## 路径：

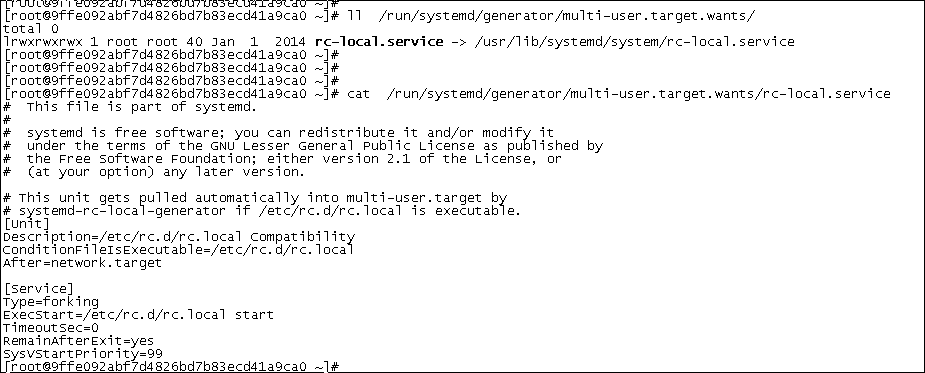
\systemd-208\src\rc-local-generator**\** rc-local-generator.c

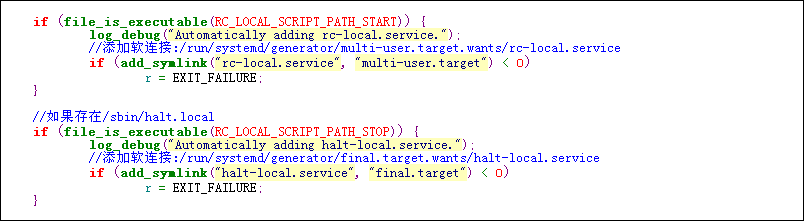
## 代码分析：

入参：只能有三个或者没有

日志记录级别：如果是tty则输出到控制台。否则是KMSG. (安全)

/etc/rc.d/rc.local





## Man手册：

无

# getty-generator

## 路径：

\systemd-208\src\getty-generator\ getty-generator.c

## 代码分析：

入参：只能有三个或者没有

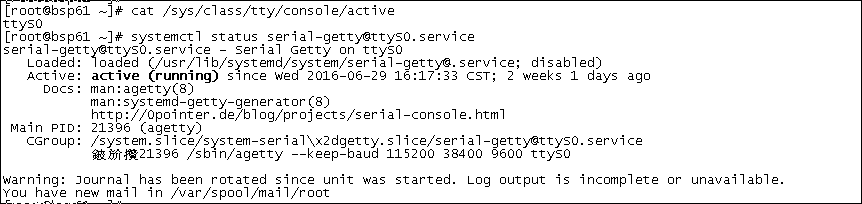
日志记录级别：如果是tty则输出到控制台。否则是KMSG. (安全)

如果是容器：//添加软连接:/run/systemd/generator/getty.target.wants/console-getty.service 和 /usr/lib/systemd/system/console-getty.service

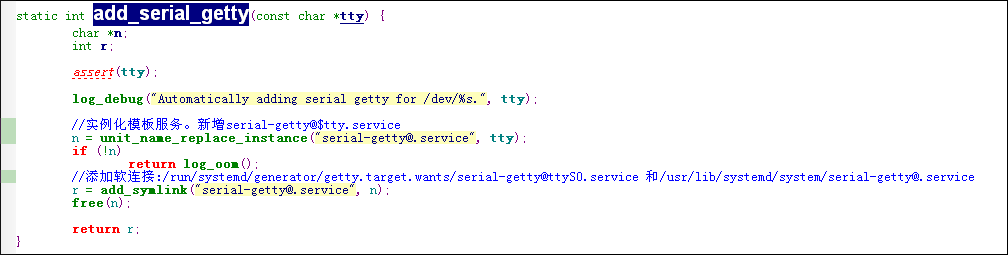
（注：我们的环境没有容器，不包含/proc/vz 或/proc/bc）

对于非容器：

读取文件/sys/class/tty/console/active中的值，赋给active；







对虚拟化控制台：

static const char virtualization\_consoles[] =

"hvc0\0"

"xvc0\0"

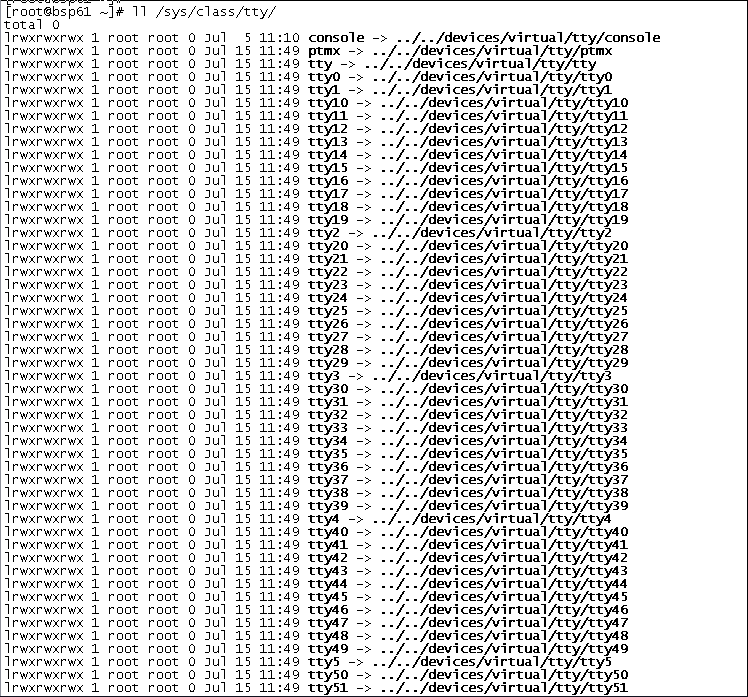
"hvsi0\0"

"sclp\_line0\0"

"ttysclp0\0"

"3270!tty1\0";

如：/sys/class/tty/hvco，tty实例化，并创建链接。



## Man手册：

man systemd-getty-generator

systemd-getty-generator – 控制台的getty实例化。

systemd-getty-generator 在内核控制台/dev/console自动实例化serial-getty@.service的生成器。如果在虚拟化环境中执行，也可以对于虚拟控制台进行实例化serial-getty@.service。

This should ensure that the user is shown a login prompt at the right place,

regardless of which environment the system is started in. For example, it is sufficient to redirect the kernel console with a kernel

命令行参数console=用于获取内核信息，以及串口TTY的getty提示。

有关gettys的配置的详细信息，可以参考章节：systemd for Administrators, Part XVI: Gettys on Serial Consoles

# system-update-generator

## 路径：

\systemd-208\src\system-update-generator\ system-update-generator.c

## 代码分析：

入参：只能有三个或者没有

日志记录级别：如果是tty则输出到控制台。否则是KMSG. (安全)

获取/system-update路径下的文件信息。

建立软连接：建立软连接: /run/systemd/generator/system-update.target 和/tmp/default.target

## Man手册：

man systemd-system-update-generator

systemd-system-update-generator – 对于离线升级模式，生成重定向启动。如果/system-update路径存在，则自动将boot过程重定向到system-update.target。

System Updates详细说明：

<https://freedesktop.org/wiki/Software/systemd/SystemUpdates/>

generator详细介绍：

<https://www.freedesktop.org/wiki/Software/systemd/Generators/>

# notify

## 路径：

/systemd-208/src/notify/notify.c

## 代码分析：

入参最多两个。

static int help(void)打印帮助

static int parse\_argv(int argc, char \*argv[])参数解析，标志位置位

int main(int argc, char\* argv[])调用参数解析，输出环境变量。

## Man手册：

systemd-notify [选项] [值]

进程脚本调用，通知服务状态改变。作启动完成通知，sd\_notify的外壳。注意：需要在服务单元里设置NotifyAccess=all。

systemd-notify –ready等价于systemd-notify READY=1，通知初始化系统服务启动完成。

systemd-notify –pid=等同于systemd-notify MAINPID=$PID，通知进程主PID，缺省时调用用调用命令的进程PID。

systemd-notify –status=等同于systemd-notify STATUS=，发送自由形态的进程状态。

systemd-notify –booted=等同于systemctl is-system-running，系统是否由systemd引导。

-h，--help，--version

退出状态，0。

# sd\_notify

## 路径：

/systemd-208/src/systemd/Sd-daemon.h

/systemd-208/src/libsystemd-daemon/Sd-daemon.c

## 代码分析：

#include <systemd/sd-daemon.h>

int sd\_notify(int unset\_environment, const char \*state);

int sd\_notifyf(int unset\_environment, const char \*format, ...);

int sd\_pid\_notify(pid\_t pid, int unset\_environment, const char \*state);

int sd\_pid\_notifyf(pid\_t pid, int unset\_environment, const char \*format, ...);

int sd\_pid\_notify\_with\_fds(pid\_t pid, int unset\_environment, const char \*state, const int \*fds, unsigned n\_fds);

函数\_sd\_export\_ int sd\_notify(int unset\_environment, const char \*state)

限定systemd使能，linux，套接字有效，否则返回0

没有状态，返回错误码。

没有设置环境变量，返回0。

不是抽象套接字或绝对路径，返回错误码。

套接字初始化小于0，返回错误码。

分配套接字地址，空间，拷贝地址。

分配读写缓存区。

分配接受应用层数据包，向内核发送的msg区。

发送msg错误，返回错误码。

结束：释放环境变量，返回正值。

这里的几个结构体：

**iovec：读写多个非连续缓冲区**

#include <sys/uio.h>

/\* Structure for scatter/gather I/O. \*/

struct iovec{

     void \*iov\_base; /\* Pointer to data. \*/

     size\_t iov\_len; /\* Length of data. \*/

};

成员iov\_base指向一个缓冲区，这个缓冲区是存放readv所接收的数据或是writev将要发送的数据。

成员iov\_len确定了接收的最大长度以及实际写入的长度。

**msghdr：接受应用层数据包，向内核发送msg**

struct msghdr {

            void            \*msg\_name;// 存数据包的目的地址，网络包指向sockaddr\_in

                                                   //向内核发数据时，指向sockaddr\_nl

            int             msg\_namelen;// 地址长度

            struct iovec    \*msg\_iov;

            \_\_kernel\_size\_t msg\_iovlen;

            void            \*msg\_control;

            \_\_kernel\_size\_t msg\_controllen;

            unsigned        msg\_flags;

        };

## Man手册：

进程脚本调用，通知服务状态改变。作启动完成通知。

如果上述的unset\_environment参数非0，sd\_notify会在返回前释放$NOTIFY\_SOCKET环境变量。后续对sd\_notify的调用会失败，但子进程不再继承这个变量。

state参数包含参数设置（每个参数单独一行）。

通用参数：

READY=1，通知服务启动完成，只有service是Type=notify用，=0没用，没有参数指示未ready。

RELOADING=1，通知服务在重加载设置，用于追踪服务内部状态，发送了这个通知必须在完成重加载同时发送READY=1

STOPPING=1，通知服务开始关闭，追踪服务内部状态

STATUS=，发送一个状态字符串给服务管理描述服务状态，字符串可以设置，比如进度条或可读的错误信息。

ERRNO=，服务失败的ERRNO=2错误码

BUSERROR=，服务失败的D-BUS错误码

MAINPID=，服务的主进程号

WATCHDOG=1，提示服务管理更新看门狗时间戳，定时WatchdogSec使能。

FDSROTE=1，在服务管理中存储附加文件描述符，用来在调用服务时传递逻辑，用于实现服务重启计划，服务按序列开启运行推文件描述符到服务管理，重启，通过socket传输运行恢复状态，注意服务管理只有在FileDescriptorStoreMax=非0的时候接收信息，默认0。服务管理在发送备份文件到服务前会删除。

注意：需要在服务单元里设置NotifyAccess=all。

sd\_notifyf()和sd\_notify()类似，用printf()格式字符串加参数

sd\_pid\_notify()和sd\_pid\_notifyf()，用PID作为信息的起始PID，第一个参数。用于发送伴随其他进程的通知信息，提供可用优先级。PID设为0，就用调用进程的PID，在这种情况下，和上面一样。

sd\_pid\_notify\_with\_fds()又附加了文件描述符队列。这个附加参数是带有文件描述符号的一个指针，指向文件描述符队列。如果描述符号是0，这个函数和sd\_pid\_notify()一样。FDSTORE=1时有用，没有这个选项服务管理器会立即关闭接收。

返回值：错误情况下，返回负值错误码，如果没有设置$NOTIFY\_SOCKET，因此没有状态数据发送，返回0。如果有状态发送，返回正值。为了兼容，推荐忽略返回值。

注意：这些函数接口在库libsystemd pkg-config(1)文件里。发送状态字符串数据到AF\_UNIX套接字（在$NOTIFY\_SOCKET环境变量中指定）。如果$NOTIFY\_SOCKET的第一个字符是@，字符串被认为是Linux抽象名套接字。数据报告伴随发送服务的进程证书，使用SCM\_CREDENTIALS。

环境变量，$NOTIFY\_SOCKET，由服务管理发送给监督进程，通知状态和启动完成。环境变量制定了sd\_notify()指向的套接字。

例：

1、启动完成后调用sd\_notify(0, "READY=1");

2、附加启动信息

sd\_notifyf(0, "READY=1\n"

"STATUS=Processing requests...\n"

"MAINPID=%lu",

(unsigned long) getpid());

3、错误的通知

sd\_notifyf(0, "STATUS=Failed to start up: %s\n"

"ERRNO=%i",

strerror(errno),

errno);

4、在服务管理器中保存文件描述符，在服务重启后继续操作而不会丢失状态。

sd\_pid\_notify\_with\_fds(0, 0, "FDSTORE=1", &fd, 1);

# sysctl

## 路径：

\systemd-208\src\sysctl\sysctl.c

## 代码分析：

定义变量：无前缀，设置文件目录。

static char \*normalize\_sysctl(char \*s)

功能：标准化sysctl，处理参数目录的“.”和“/”转化

static int apply\_sysctl(const char \*property, const char \*value)

功能：应用sysctl，入参有性能和值两个

记日志；

分配空间，参数放到目录下的文件夹中；

是否空行；

参数写入文件；

static int apply\_all(Hashmap \*sysctl\_options)

功能：应用哈希表里所有的参数

static int parse\_file(Hashmap \*sysctl\_options, const char \*path, bool ignore\_enoent)

入参：哈希表，路径，忽略/不存在

功能：解析文件

搜索和打开路径文件，失败记录日志；

还没执行到文件结束时，逐行扫描：

搜索行结束符，没搜到且文件已结束，记录错误日志；

去掉空格，查找字符串中首先出现的等号，没找到记录行不能设置的日志；

将参数p和值value标准化，去掉空格，在哈希表中找到对应的值；

记日志，覆盖更早的设置，删除哈希表中的p，释放v。

p拷贝到参数，value拷贝到值，建立哈希表；

失败则记录日志，添加sysctl值到哈希表失败；

释放空间。

static int help(void)

功能：打印帮助。

static int parse\_argv(int argc, char \*argv[])

入参：数量，参数

功能：解析命令行参数

解析命令行选项参数；while循环，case选项；

h打印帮助；

参数前缀：规范化参数；

参数以/proc/sys开头，否则在输入参数前加上/proc/sys；

错误记录日志；

int main(int argc, char \*argv[])

功能：主函数

初始化定义哈希表；

先解析参数，记录日志；

设置权限；

建立哈希表（不成功退出，记录错误日志）；

如果参数个数大于选项最后一个参数的编号：

对每一个参数，解析文件

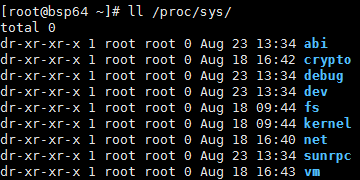
否则，查找打开其他设置文件，失败记录日志。

引用apply\_all()函数

结束释放哈希表和参数前缀字符串。

## Man手册：

运行时设置内核参数，可选参数列在/proc/sys/下。



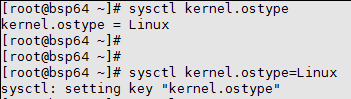
命令：

sysctl [options] [variable[=value]] [...]

sysctl -p [file or regexp] [...]

参数：

variable是关键字，value是要设定的值，不加value查询



-n,--values不打印参数名

-N,--names只打印参数名

-e,--ignore忽略未知参数错误

-q,--quiet不打印定向到标准输出的

-w,--wirte改变sysctl设置

-p [file],--load[=file],-f从指定的文件或默认/etc/sysctl.conf导入sysctl设置，文件参数可用按正则表达式定义

-a,--all,-A ,-X打印所有当前可用参数

--deprecated 和--all列出的值的否决参数。

-b,--binary打印参数在一行里

--system从所有系统配置文件里设置参数

/run/sysctl.d/\*.conf

/etc/sysctl.d/\*.conf

/usr/local/lib/sysctl.d/\*.conf

/usr/lib/sysctl.d/\*.conf

/lib/sysctl.d/\*.conf

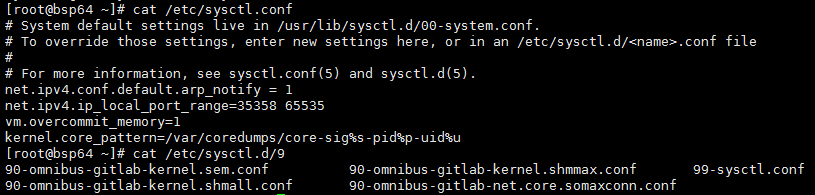
/etc/sysctl.conf

-r,--pattern pattern只应用和pattern相匹配的设置，pattern用附加的正则表达式算法。

-d,-h帮助

-x,-o不做BSD兼容

-v版本信息



否决参数：

base\_reachable\_time和retrans\_time，不允许sysctl改变这些参数值，一定要改要用以下方式：

echo 256 > /proc/sys/net/ipv6/neigh/eth0/base\_reachable\_time

# systemctl

## 路径：

/systemd-208/src/systemctl/systemctl.c

## 代码分析：

## Man手册：

控制systemd系统和服务管理

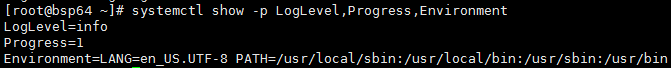
systemctl [OPTIONS...] COMMAND [NAME...]

**可用参数[OPTIONS...]：**

-t,--type逗号分隔的单元类型列表，比如service和socket，只显示这些单元，help显示允许的值

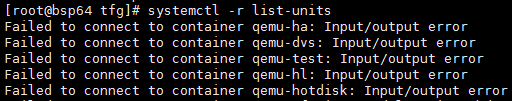
--state=逗号分隔的状态列表，比如LOAD,SUB或ACTIVE状态，只显示特定状态。

-p,--property=当用show命令显示性能时，限制显示的性能内容，逗号分隔。不设时所有的属性都会显示。

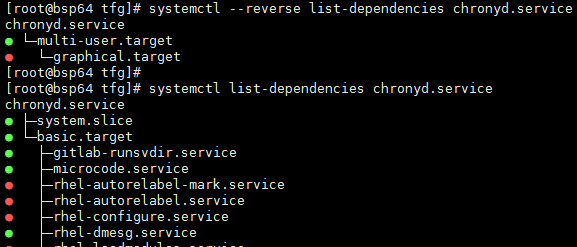


-a,--all显示单元，显示所有加载的单元，无论状态，包括未激活的；显示属性，列出所有属性无论是否设置。列出所有安装的单元用list-unit-files。

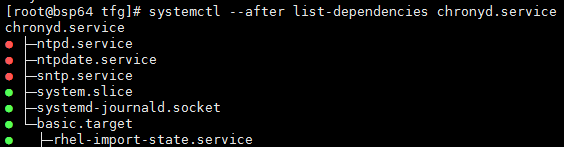
-r,--recursive列出单元时，也显示本地容器的单元。前缀容器名，用：分隔。



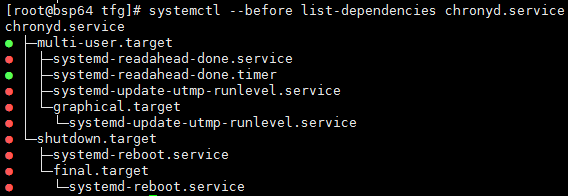
--reverse根据list-dependencies的结果，显示被依赖项。



--after根据list-dependencies，显示顺序在指定单元之前的。递归列出After=dependency。任何after依赖自动镜像创建一个before依赖。当前依赖明确指定，也会创建WantedBy=target的暗中依赖，作为一个其他指令的结果。



--before列出指定单元之后的，Before=dependency。



-l,--full不省略单元名，处理树状入口，日志输出，截断status,list-units,list-jobs和list-timers的描述。

--show-types列出套接字时带类型

--job-mode=入列一个新job时，如何处理已经在队列中的任务。取值可以是fail,replace,replace-irreversibly,isolate,ignore-dependencies,ignore-requirements或flush，默认是replace，除非用isolate隔离命令的时候指job模式为isolate。

# 模板

## 路径：

## 代码分析：

## Man手册：

# 常用函数

## Stat

函数名: stat(const char\* path, struct stat\* buf )

功 能: 得到文件的信息，将其保存在buf结构中，buf的地址以参数形式传递给stat。

参数:

const char \*path: 文件名或者目录名

struct \_stat \*buffer:结构体对象地址

返回值: 成功返回0，返回-1表示失败。

<http://baike.baidu.com/view/568600.htm>

用法：stat 文件名

## Utimestat

utimensat()通过文件的路径（pathname）获得文件，

调用成功，utimensat()返回0，调用失败返回-1，并设置[errno](http://baike.baidu.com/subview/3485007/3485007.htm)

<http://baike.baidu.com/view/10459713.htm>

## access

功 能: 确定文件或文件夹的访问权限。即，检查某个文件的存取方式，比如说是只读方式、只写方式等。如果指定的存取方式有效，则函数返回0，否则函数返回-1。

用 法: int access(const char \*filenpath, int mode); 或者int \_access( const char \*path, int mode );

参数说明：

filenpath

文件或文件夹的路径，当前目录直接使用文件或文件夹名

备注：当该参数为文件的时候，access函数能使用mode参数所有的值，当该参数为文件夹的时候，access函数值能判断文件夹是否存在。在WIN NT 中，所有的文件夹都有读和写权限

<http://baike.baidu.com/view/1241475.htm>

## pipe

定义函数： int pipe(int filedes[2]);  
函数说明： pipe()会建立管道，并将文件描述词由参数filedes数组返回。  
              filedes[0]为管道里的读取端  
              filedes[1]则为管道的写入端。  
返回值：  若成功则返回零，否则返回-1，错误原因存于errno中。

<http://blog.csdn.net/myarrow/article/details/9037135>

## waitid

int waitid(idtype\_t idtype, id\_t id, siginfo\_t \*infop. int options);  
  
//返回值:成功返回0，出错返回-1

与waitpid相似，waitid允许一个进程指定要等待的子进程。但它使用单独的参数表示要等待的子进程的类型，而不是将此与进程ID或进程组ID组合成一个参数。

<http://shaoguangleo.blog.163.com/blog/static/227798320119197420419/>

## siginfo\_t中si\_code的值

<http://blog.csdn.net/liangyuannao/article/details/7891087>

## setmntent/getmntent/hasmntopt

setmntent() 是打开包含挂载点项目的文件, 其中的 filename 参数是要打开的文件名, type 参数就像 fopen() 的第二个参数, 代表只读、只写, 或读写皆可的存取模式 。返回FILE\*。

getmntent() 则是循序读取整个档案，传回指向 static struct mntent 结构的指针，结构中会填入适当的值。

addmntent() 可以在已开启档案的末端加上资讯，它原本是给 mount 使用的。

endmntent() 的功用是关闭打开的文件。这不能只是呼叫 fclose() 而已,因为可能还有其它与FILE \* 有关的内部资料结构需要清理。

hasmntopt() 是个比较特殊的函式。它会扫描第一个参数所传入的struct mntent，找出它的挂载选项是否符合第二个引数。假如找到选项就传回符合的子字符串的位址；否则传回NULL。

<http://www.cnblogs.com/leaven/archive/2011/05/06/2038946.html>

strtoul

strtoul （将字符串转换成无符号[长整型](http://baike.baidu.com/view/3522307.htm)数）

nsigned long strtoul(const char \*nptr,char \*\*endptr,int base);

strtoul()会将参数nptr字符串根据参数base来转换成无符号的[长整型](http://baike.baidu.com/view/3522307.htm)数。参数base范围从2至36，或0。

<http://baike.baidu.com/view/1526542.htm>

## lstat

函数名: lstat

功 能: 获取一些文件[相关](http://baike.baidu.com/view/555451.htm)的信息

用 法: int lstat(const char \*path, struct stat \*buf);

参数：

path：文件路径名。

filedes：[文件描述词](http://baike.baidu.com/view/2429077.htm)。

buf：是以下[结构体](http://baike.baidu.com/view/204974.htm)的[指针](http://baike.baidu.com/view/159417.htm)

<http://baike.baidu.com/view/1621201.htm>