

## Linux文件的各种\*id属性



Charlie

1 人赞同了该文章

赞同 1



分享

### Linux文件权限标志

Linux 文件或目录的属性主要包括：文件或目录的节点、种类、权限模式、链接数量、所归属的用户和用户组、最近访问或修改的时间等内容；

```
root@marvin:/home/admin/workspace/test# ls -lih
total 28K
1592057 -rwxrw-r-- 1 admin admin   6 Aug 16 00:24 test.c
1592056 -rwsr-xr-x 1 admin admin 17K Aug 21 14:53 testid
1592059 -rw-rw-r-- 1 admin admin 200 Aug 21 14:50 testid.c
```

解释：

第一字段：inode

第二字段：第一个字符表示文件类型，后面9个字符表示文件权限；

第三字段：硬链接个数；

第四字段：属主；

第五字段：所归属的组；

第六字段：文件或目录的大小；

第七字段和第八字段：最后访问或修改时间；

第九字段：文件名或目录名

我们这里就是要讨论第二个字段里表示文件权限的九个字符，拿文件test.c举例说明：

"rwxrw-r-"中前三个字符 "rwx" 表示文件所属用户对该文件有读、写和运行权限；中间三个字符 "rws" 表示该文件所属用户组成员对该文件有读和写权限，没有执行权限；最后三个字符 "r--" 表示其他用户对该文件只有读权限，没有写权限和运行权限。

这九个字符中还包含特殊权限SUID, SGID, SBIT:

Set UID:

```
admin@marvin:~/workspace/test$ ls -lh /usr/bin/passwd
-rwsr-xr-x 1 root root 67K Mar 14 16:26 /usr/bin/passwd
```

当 s 这个标志出现在文件拥有者的 x 权限上时，如上 /usr/bin/passwd 这个文件的权限状态，此时就被称为 Set UID，简称为 SUID 的特殊权限。基本上 SUID 有这样的限制与功能：

- SUID 权限仅对二进位程序(binary program)有效(不能够用在 shell script 上面)
- 运行者对于该程序需要具有 x 的可运行权限
- 本权限仅在运行该程序的过程中有效 (run-time)
- 运行者将具有该程序拥有者 (owner) 的权限

以passwd文件为例：

admin 对于 /usr/bin/passwd 这个程序来说是具有 x 权限的，表示 admin 能运行 passwd；passwd 的拥有者是 root 这个帐号；admin 运行 passwd 的过程中，会暂时获得 root 的权限；/etc/shadow 就可以被 admin 所运行的 passwd 所修改。但如果 admin 使用 cat 去读取 /etc/shadow 时，因为 cat 不具有 SUID 的权限，所以 admin 运行 cat /etc/shadow 时，是不能读取 /etc/shadow 的。

**SGID:**

与 SUID 不同的是，SGID SGID 对二进位程序有用；

▲ 赞同 1

▼

● 添加评论

↗ 分享

♥ 喜欢

★ 收藏

✉ 申请转载

...

如果针对的是目录,SGID 有如下的功能:

使用者若对于此目录具有 r 与 x 的权限时, 该使用者能够进入此目录; 使用者在此目录下的有效群组(effective

group)将会变成该目录的群组;

换句话说: 当甲这个使用者于 A 目录是具有群组或其他人的身份, 并且拥有该目录 w 的权限, 这表示『甲使用者对该目录内任何人创建的目录或文件均可进行“删除/更名/搬移”等动作。』不过, 如果将 A 目录加上了 SBIT 的权限项目时, 则甲只能够针对自己创建的文件或目录进行删除/更名/移动等动作, 而无法删除他人的文件。

### setuid(), seteuid()

在使用 setuid() 函数时会遇到 3 个关于 ID 的概念:

real user ID – 实际用户 ID

effective user ID – 有效用户 ID

saved set-user-ID – 保存了的设置用户 ID。

### 测试

以上这些概念还是比较的抽象, 那么下面写一个简单的测试程序:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
    printf("uid = %d, gid = %d, euid = %d, egid = %d\n", getuid(), getgid(), geteuid(),
           getegid());
    return 0;
}
```

这个程序非常简单没有什么好说的。我们编译这个程序生成 testid 程序

```
root@marvin:/home/admin/workspace/test# gcc testid.c -o testid
root@marvin:/home/admin/workspace/test# ls -l testid
-rwxr-xr-x 1 root root 16880 Aug 21 14:53 testid
root@marvin:/home/admin/workspace/test# id
uid=0(root) gid=0(root) groups=0(root)
root@marvin:/home/admin/workspace/test# ./testid
uid = 0, gid = 0, euid = 0, egid = 0
```

通过 id 命令看到当前登录用户为 root, uid=0, gid=0。通过 ls 命令我们可以看出 testid 程序没有设置 SUID 和 SGID, 所有者是 root, 所有组也是 root。执行 testid 我们发现有效用户 ID 等于实际用户 ID (0), 有效用户组 ID 等于实际用户组 ID (0)。

你可能注意到 testid 的所有者 root, 组也是 root, 和实际用户, 实际用户组是一样的。下一步我们修改一下 testid 所有者和组, 再看结果。

```
root@marvin:/home/admin/workspace/test# chown admin testid
root@marvin:/home/admin/workspace/test# chgrp admin testid
root@marvin:/home/admin/workspace/test# ./testid
uid = 0, gid = 0, euid = 0, egid = 0
```

发现结果和上面一样, te:

户组 ID (0)。

```
root@marvin:/home/admin/workspace/test# chmod u+s testid
root@marvin:/home/admin/workspace/test# ls -l testid
-rwsr-xr-x 1 admin admin 16880 Aug 21 14:53 testid
root@marvin:/home/admin/workspace/test# ./testid
uid = 0, gid = 0, euid = 1000, egid = 0
```



发现设置testid程序的SUID位之后，testid进程的有效用户ID等于文件所有者的UID (admin的uid为1000)，有效用户组ID还是等于实际用户组ID (0)。

## 实际用户ID

首先说这个实际用户ID，就是我们当前以哪个用户登录了，标识我是谁，我们运行的程序的实际用户ID就是这个用户的ID，可以用id命令查看。

真实用户 ID (real user ID) 就是通常所说的 UID，在 /etc/passwd 中能看到它的身影，如：

```
vm:x:1000:1000:vm:/home/vm:/bin/bash
```

它用来标识系统中各个不同的用户。普通用户无法改变这个 ID 值。有效用户 ID (effective) 表明，在运行一个程序时，你是以哪种有效身份运行它的。一般而言，有效用户ID 等于 真实用户ID。这两个 ID 值可以用 geteuid() 和 getuid() 函数获取。

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf ("The real user ID is: %d\n", getuid());
    printf ("The effective user ID is :%d\n", geteuid());
    return (0);
}
```

编译程序后，查看生成的可执行文件权限：

```
$ ls -l get_uid
-rwxrwxr-x 1 admin admin 16792 Aug 21 22:18 get_uid
```

**普通用户运行：**

```
$ ./get_uid
The real user ID is: 1000
The effective user ID is :1000
```

**root 用户运行**

```
# ./get_uid
The real user ID is: 0
The effective user ID is :0
```

这就是所说的一般情况： 实际用户ID == 有效用户ID

## 有效用户ID

有效用户ID就是当前进程是以哪个用户ID来运行的，一般情况下是实际用户ID，如果可执行文件具有了SUID权限，那么它的有效用户ID就是可执行文件的拥有者。进程用来决定我们对资源的访问权限。一般情况下，有：

户-ID (SUID) 位设置，！

Unix系统通过进程的有效用户ID和有效用户组ID来决定进程对系统资源的访问权限。

下面看不一致的情况：

使用 chmod 改变该程序的有效权限位(suid)：

```
$ chmod u+s get_uid
$ ls -l get_uid
-rwsrwxr-x 1 admin admin 16792 Aug 21 22:18 get_uid
```

再使用普通用户运行：

```
$ ./get_uid
The real user ID is: 1000
The effective user ID is :1000
```

切换到 root 用户运行：

```
$ ./get_uid
The real user ID is: 1000
The effective user ID is :1000
```

从 root 运行输出来看，有效用户 ID 的值为 1000。也就是说，root 运行这个程序时，是没有 root 的权限的，它只有 admin 这个普通用户(实际用户 ID 为 1000)的权限。

下面用一个实例来验证 root 运行此程序时只有 admin 这个普通用户的权限，按下面步骤来进行：

1 现在 /root 目录下创建一个普通的文本文件 rootfile

```
# echo "hello world" > /root/rootfile
# ls -l /root/rootfile
-rw-r--r-- 1 root root 12 Aug 21 22:22 /root/rootfile
```

2 修改上面的程序代码为：

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    const char *file = "/root/rootfile";

    printf ("The real user ID is: %d\n", getuid());
    printf ("The effective user ID is :%d\n", geteuid());

    if (!unlink (file))
        printf ("Ok, I am root, and I can delete the file which in /root directory.\n"
    else
        perror ("unlink error");

    return (0);
}
```

在上面的代码中所要做的事情很简单，就是要尝试删除 /root/rootfile 这个文件。使用普通用户来编译该程序。

```
$ ./get_uid
The real user ID is: 1000
The effective user ID is :1000
unlink error: Permission denied
```



很自然，普通用户没有权限删除 /root 下的文件。

下面使用 root 来运行该程序：

```
# ./get_uid
The real user ID is: 0
The effective user ID is :0
Ok, I am root, and I can delete the file which in /root directory.
```

也很正常，root 有权限删除它目录下的文件。

现在为这个程序添加有效权限位，注意这里是用普通用户运行的 chmod 命令：

```
$ chmod u+s getuid
$ ls -l get_uid
-rwsrwxr-x 1 admin admin 16920 Aug 21 22:25 get_uid
```

再次分别以 普通用户 和 root 用户运行上面的程序：

普通用户：

```
$ ./get_uid
The real user ID is: 1000
The effective user ID is :1000
unlink error: Permission denied
```

还是一样，普通用户并没有权限删除 /root 下的文件。

root 用户：

```
# ./get_uid
The real user ID is: 0
The effective user ID is :1000
unlink error: Permission denied
```

由输出可见，root 用户也没法删除该文件！同时我们看到，此时的有效用户 ID 值为 1000，所以我们知道，这个程序所赋予操作文件的权限不但要检查实际用户ID，更重要的是要考虑 有效用户 ID；

如果 有效用户 没有权限删除，那么换成 root 用户它也相当于被降权了，当然这个降权仅限于在这个程序的空间中。

## 保存的设置用户ID

保存的设置用户ID就是有效用户ID的一个副本，与SUID权限有关。

关于那个SUID，最经典的例子莫过于passwd命令。passwd这个可执行文件的所有者是root，但是其他用户对于它也有执行权限，并且它自身具有SUID权限。那么当其他用户来执行passwd这个可执行文件的时候，产生的进程的就是以root用户的ID来运行的。

下面通过 setuid() 和 seteuid() 这两个函数来考察一下 saved set-user-ID (保存的设置用户ID) 这个概念。

在使用 setuid() 时会遇

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

void show_ids(void)
{
    printf ("real uid: %d\n", getuid());
    printf ("effective uid: %d\n", geteuid());
}

int main(int argc, char *argv[])
{
    int uid;
    show_ids();
    uid = atoi(argv[1]);

    if (setuid (uid) < 0)
        perror ("setuid error");

    show_ids();

    return (0);
}
```

下面使用 root 用户来运行上面的程序：

```
# ./get_uid 1001
real uid: 0
effective uid: 0
real uid: 1000
effective uid: 1000
```

由此可见，在 root 下，实际用户 ID 和有效用户 ID 均被设为 `setuid()` 的参数 uid 的值。

1. 若进程不具有 root 权限，那么普通用户使用 `setuid()` 时参数 uid 只能是自己的，没有权限设置别的数值，否则返回失败：  
使用普通用户来运行：

```
$ ./get_uid 1001
real uid: 1000
effective uid: 1000
setuid error: Operation not permitted
real uid: 1000
effective uid: 1000
```

由以上可以看出，只有超级用户进程才能更改实际用户 ID。所以一个非特权用户进程不能通过 `setuid` 或 `seteuid` 得到特权用户权限。

这里考虑 `su` 这个程序，`su` 可以将普通用户切换到 root 用户。这是因为，`su` 被设置了有效权限位：

```
ll /usr/bin/su
-rwsr-xr-x 1 root root 67816 Feb 7 2022 /usr/bin/su*
```

如上面所做实验描述的一样，普通用户在运行 `su` 时，它也就拥有了 root 的权限。

对于调用了 `setuid()` 函数的程序要格外小心，当进程的有效用户 ID 即 euid 是 root 用户时(设置了有效权限位)，如果调

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

void show_ids(void)
{
    printf("The real user ID is: %d\n", getuid());
    printf("The effective user ID is :%d\n", geteuid());
}

int main(void)
{
    const char* file = "/root/rootfile";
    setuid(0);
    show_ids();
    if (!unlink(file)) {
        printf("Ok, I am root, and I can delete the file which in /root directory.\n");
        system("echo hello world > /root/rootfile");
        printf("Now, drop the root privileges.\n");
        if (setuid(1000) < 0) {
            perror("setuid");
            exit(EXIT_FAILURE);
        }
        show_ids();
        if (unlink(file) < 0) {
            printf("Ok, we have no privilege to delete rootfile.\n");
        }
        printf("try to regain root power again...\n");
        if (seteuid(0)) {
            perror("seteuid");
            show_ids();
            exit(EXIT_FAILURE);
        }
    }
    return 0;
}
```

我们使用 root 编译上面的程序，并运行 `chmod u+s` 给程序添加 suid 位，然后以普通用户来运行它：

```
$ ./get_uid
The real user ID is: 0
The effective user ID is :0
Ok, I am root, and I can delete the file which in /root directory.
Now, drop the root privileges.
The real user ID is: 1000
The effective user ID is :1000
Ok, we have no privilege to delete rootfile.
try to regain root power again...
seteuid: Operation not permitted
The real user ID is: 1000
The effective user ID is :1000
```

由输出可见，在运行 `setuid(1000)` 函数时，我们还是具有 root 权限的，所以该函数会设置成功。正是因为有了 root 权限，所以 3 个 ID (真实用户ID, 已保存用户ID, 有效用户ID)都会被设置为 1000。所以在运行完 `setuid(1000)` 后，进程已经被降权为普通用户，此时想再 `seteuid(0)` 提高权限已经不可能。这里需要提到一点，对于 `show_ids()` 函数里，我们无法获取保存的设置用户ID(saved set-user-ID)，这是因为没有这种 API。但是我们知道这个约定：当用户是 root 时，使用 `setuid()` 来修

```
seteuid() sets the effective user ID of the calling process. Unprivileged user proces
```

seteuid() 用来设置调用进程的有效用户 ID。普通用户进程只能将有效用户 ID 设置为实际用户 ID，有效用户 ID，保存的设置用户 ID。这里比较重要的是，**seteuid() 中的参数可以被设置为保存的设置用户 ID**。保存的设置用户 ID 是这样的一种概念：它是从 exec 复制有效用户 ID 而得来的。具体的说，当我们从一个 shell 里执行一个外部命令时(这里就当做是执行上面的 getuid 这个)，如果该程序设置了用户ID位(有效权限位)，那么在 exec 根据文件的用户ID设置了进程的有效用户 ID 后，就会将这个副本保存起来。简单的说，saved set-user-ID 保存了有效用户ID 的值。比如对于 getuid 这个程序，saved set-user-ID 保存的值就是 0 。据此，我们修改上面的 getuid 程序代码为：

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

void show_ids(void)
{
    printf("The real user ID is: %d\n", getuid());
    printf("The effective user ID is :%d\n", geteuid());
}

int main(void)
{
    const char* file = "/root/rootfile";
    setuid(0);
    show_ids();
    if (!unlink(file)) {
        printf("Ok, I am root, and I can delete the file which in /root directory.\n");
        system("echo hello world > /root/rootfile");
        printf("Now, drop the root privileges.\n");
        if (seteuid(1000) < 0) {
            perror("seteuid");
            exit(EXIT_FAILURE);
        }
        show_ids();
        if (unlink(file) < 0) {
            printf("Ok, we have no privilege to delete rootfile.\n");
        }
        printf("try to regain root power again...\n");
        if (seteuid(0)) {
            perror("seteuid");
            show_ids();
            exit(EXIT_FAILURE);
        }
    }
    show_ids();

    printf("try to delete rootfile again\n");
    if (!unlink(file)) {
        printf("Ok, regain root power successful!\n");
        system("echo hello world > /root/rootfile");
        return (0);
    }
    return (0);
}
```

在上面的代码中，我们将  
试删除 /root/rootfile

```
$ ./get_uid
The real user ID is: 0
The effective user ID is :0
Ok, I am root, and I can delete the file which in /root directory.
Now, drop the root privileges.
The real user ID is: 0
The effective user ID is :1000
Ok, we have no privilege to delete rootfile.
try to regain root power again...
The real user ID is: 0
The effective user ID is :0
try to delete rootfile again
Ok, regain root power successful!
```



此时我们看到整个过程：

先是普通用户执行了具有 root 有效权限位设置的程序，它成功的删除了 /root 下面的一个文本文件；然后使用 system() 系统调用恢复了该文件，目的是方便下面继续实验。接着，它使用 seteuid() 函数时该进程降权为普通用户权限进程。此后，正是因为有了 saved set-user-ID 的保存，所以当再次使用 seteuid() 恢复进程的 root 权限时可以恢复成功！

所以再次看到， setuid() 会改变 saved set-user-ID 的值而不能恢复权限；而 seteuid() 不会改变 saved set-user-ID 这个值，所以它能够恢复。

【文章福利】：[C/C++Linux服务器开发/后台架构师【公开课学习】](#)（C/C++，Linux，golang技术，内核，Nginx，ZeroMQ，MySQL，Redis，fastdfs，MongoDB，ZK，流媒体，CDN，P2P，K8S，Docker，TCP/IP，协程，DPDK，ffmpeg，大厂面试题等）有需要的可以点击[793599096加群领取哦~](#)

## 其他

在使用 setuid，seteuid，chmod u+s filename，chmod u-s filename 等命令前，先弄清楚四个概念，分别是：文件拥有者，real user ID，effective user ID，saved set-user-ID。

文件拥有者：当你输入 ls

上面这一栏就是ls -l命令的输出，第三栏就是文件拥有者，在这里是admin。

**real user ID**: 实际上进程的执行者，标志系统中不同的用户，普通用户无法修改其值，某一个用户的real user ID==他的uid，使用命令 “id username”来查看某用户uid。

再来区分一下，real user ID与文件所有者的概念，以下面代码作为范例。

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int main(){
    int i;
    printf("The real user ID is: %d\n",getuid());
    printf("The effective user ID is: %d\n",geteuid());
    i=setuid(100);
    if(0==i){
        printf(".....\n");
        printf("The real user ID is: %d\n",getuid());
        printf("The effective user ID is: %d\n",geteuid());
        printf(".....\n");
    }
    else{
        perror("failed to setuid");
        printf("The real user ID is: %d\n",getuid());
        printf("The effective user ID is: %d\n",geteuid());
        exit(1);
    }
    return 0;
}
```

假设代码以普通用户编译，生成可执行文件，输入ls -l get\_id，可以看见输出为 -rwxrwxr-x 1 admin admin 16960 Aug 21 23:00 get\_id，get\_id的所有者是admin。

然后，运行./get\_id命令，得到结果1：

```
$ ./get_id
The real user ID is: 1000
The effective user ID is: 1000
failed to setuid: Operation not permitted
The real user ID is: 1000
The effective user ID is: 1000
```

再运行sudo ./get\_id命令，得到结果2：

```
$ sudo ./get_id
The real user ID is: 0
The effective user ID is: 0
.....
The real user ID is: 100
The effective user ID is: 100
.....
```

从输出结果来看，先不理会effective user ID，当以普通用户运行时，real user ID 为1000，无法setuid，也改变不了real user ID 的值；而使用了sudo 后，改变实际执行用户为root权限执行进程，real user ID的值也变为0，0是root的uid值，此时get\_id这个文件的所有者仍然是admin (1000)，只不过使用了sudo 超级用户权限导致real user ID不一样。

时候又不一样，具体情况后面再解释。

仍以上面的代码作为例子，此时以普通用户身份对get\_id执行命令：chmod u+s get\_id，再以"ls -l get\_id"查看get\_id的文件属性，文件属性中多了一个s符号。

再执行 sudo ./get\_id，得到结果3：

```
$ sudo ./get_id
The real user ID is: 0
The effective user ID is: 1000
failed to setuid: Operation not permitted
The real user ID is: 0
The effective user ID is: 1000
```

对比结果2，尽管real user ID是0（root uid），却无法setuid，因为我们使用chmod u+s设置了effective user ID为1000（因为以普通用户运行chmod u+s命令，所以effective user ID为1000），effective user id为1000时，即普通用户，是无法使用setuid修改real user ID和effective user ID的值。所以此时，运行a.out程序的real user ID为0，而effective user ID为1000。

saved set-user ID：当effective user ID修改时，保存effective user ID的值。具体作用，我还不清楚，请看本文第一段的链接，那里有详细介绍。

从上面例子，观察到一个事实，通过getuid()和geteuid()可以获得real user ID和effective user ID的值，却没有函数获得saved set-user-ID的值。

再来区分一下setuid和chmod u+s。

setuid：修改real user ID和effective user ID。

chmod u+s filename：将filename的effective user ID 改为当前登录用户的权限，比如上例结果3中，effective user ID为1003，即uid。即使使用sudo chmod u+s，结果也一样，effective user ID仍为1003。

为了更加清晰地说明这个概念，我们再以上面的代码进行另一个试验。假设以sudo gcc get\_id.c -o test\_id编译上述代码，生成test\_id二进制文件，使用"ls -l test\_id"查看test\_id的属性，得到uid是root。

再以普通用户运行./test\_id，得到结果4：

```
The real user ID is: 1003
The effective user ID is: 1003
failed to setuid: Operation not permitted
The real user ID is: 1003
The effective user ID is: 1003
```

尽管这个文件所有者（owner）是root，但实际运行用户是普通用户，所以real user ID和effective user ID都是1000，因此也无法运行setuid。

再以sudo ./test\_id运行，得到结果5：

```
The real user ID is: 0
The effective user ID is: 0
.....
The real user ID is: 100
The effective user ID is: 100
.....
```

这里以root权限运行。

所以我们换成sudo chmod u+s test\_id, 再"ls -l test\_id", 可以看见 "-rwsr-xr-x 1 root root 7567 Jul 8 14:53 test\_id", 注意红色的s字幕, 这就是chmod u+s的效力。现在再以普通用户来运行一下test\_id二进制文件, 得到结果6:

```
The real user ID is: 1003
The effective user ID is: 0
.....
The real user ID is: 100
The effective user ID is: 100
.....
```

此时, real user ID为1000z, effective user ID为0, 即使是普通用户运行, 还是可以setuid, 因为effective user ID被chmod u+s 赋予了0 (root权限)。

这一类典型的文件有/usr/bin/passwd, 使用"ls -l /usr/bin/passwd"文件查看到 "-rwsr-xr-x 1 root root 35696 Feb 8 2011 /usr/bin/passwd", 正是这个s标志 (也就是effective user ID) 可以让普通用户以root权限做一些事情, 如登陆时密码匹配。

如果不希望该文件继续拥有u+s权限, 可使用chmod u-s filename消除影响。

最后再来说一说sticky bit粘滞位。

sticky bit: 文件写权限的一种扩展, 当文件被设置了sticky bit位, 同一组的其他用户他可以为该文件添加内容, 但不能删除该文件。使用命令chmod o+t filename, 或者chmod 八进制方式设置sticky bit位。设置好之后, 用ls -l filename, 可以看见一个标记 "t"。请注意一个地方, 即使添加了stickybit, 文件所有者也是可以随意删改这个文件的。

## chmod改变文件的SUID, SGID, SBIT特性

### 1、数字类型改变文件权限

各权限的分数对照表如下:

SUID:	4
SGID:	2
SBIT:	1
r:	4
w:	2
x:	1

每种身份特殊权限(SUID/SGID/SBIT)和(owner/group/others)各自的三个权限(r/w/x)分数是需要累加的

例如当权限为: [-rwxrwx—] 分数则是:

```
owner = rwx = 4+2+1 = 7
group = rwx = 4+2+1 = 7
others= --- = 0+0+0 = 0
```

例如当权限为: [-rwsr-sr-x] 分数则是:

```
特殊权限 = SUID&&SGID = 4+2 = 6
owner = rws = 4+2+1 = 7
group = r-s = 4+1 = 5
others= r-x = 4+1 = 5
```

### 2、符号类型改变文件权限

代表 all 亦即全部的身份！那么读写的权限就可以写成r, w, x! SUID 为 u+s , 而 SGID 为 g+s , SBIT 则是 o+t ! 也就是可以使用底下的方式来看：

```
chmod u/g/o/a +/-/= r/w/x 文件或目录
```



## linux下利用C或C++ 语言调用需要root权限的函数

### 1、setuid法

为了方便普通用户执行一些特权命令，SUID/SGID程序允许普通用户以root身份暂时执行该程序，并在执行结束后再恢复身份。

chmod u+s 就是给某个程序的所有者以suid权限，可以像root用户一样操作。

1. 登录root用户，将程序设置成 root:root 所有者（等价于：登录root用户编译程序）。也可直接将普通用户加入root组中，那么编译程序不用来回切换用户。
2. 登录root用户设置程序的UID， #chmod u+s 源文件。
3. 程序中使用：

```
uid_t uid = getuid();                                普通用户的uid
if (setuid(0)) {                                     设置为普通用户和超级用户一样的权限
    return -1;
}
//...
if (setuid(uid)) { //恢复uid                      恢复到只有普通用户的权限
}
```

通过上面步骤则该用户不管在普通用户还是在root用户下都能获取root权限。

注意：

- 复制时要想连同其UID位一同复制，**cp加参数-a, scp加参数-p**。
- 若复制到其他Linux主机上要保证程序属于 root:root 所有者。

### 2、sudo法

在调用系统命令时，使用sudo+管道的方式解决，但必须修改sudo列表。

1. sudo的配置文件是 /etc/sudoers

增加一条配置：

```
username ALL=(ALL) ALL
```

这样，普通用户username就能够执行root权限的所有命令

以username用户登录之后，执行： sudo su，然后输入username用户自己的密码，就可以切换成root用户了。

2. 程序在调用需要root权限的代码中使用system， echo "userpassword" | sudo -S sh -c "CMD1; CMD2;..." 。

注意：可以通过sudo列表控制用户username的操作权限，比如不能修改root密码等。

转自：<https://www.cnblogs.com/born...>

发布于 2023-03-11 11:51 · IP 属地湖南



还没有评论，发表第一个评论吧

### 推荐阅读



如何解密 Linux 版本信息 |  
Linux 中国

Linux...

发表于Linux...



我喜欢在 Linux 命令行中使用的 6 个元字符 | Linux 中国

Linux...

发表于Linux...



每天学一个 Linux 命令  
(55) : id

民工哥

发表于民工哥专栏

### Linux常见

在使用Linu  
难理解和难  
符号起很大  
为初学者，  
很枯燥，加  
么去解决，  
：

ashu