上节回顾

用户和组相关文件

/etc/passwd(存放用户信息)

```
# name:password(密码被迁到别的地方存储):UID:GID:GECOS（描述）:directory(家目录):shell类型
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
...
```

/etc/shadow(存放用户密码)

```
root:$6$CNc6y8cEgE3pNLSo$U6Nk1J2hmcMAf8y4yAPZLoXY12sWtBvO62z1l67OjetuR0Ndv9CL29SaSn7ZXRJFRHuOjgE5aJso1NYvopYgG0:19503:0:99999:7:::
daemon:*:19430:0:99999:7:::
bin:*:19430:0:99999:7:::
sys:*:19430:0:99999:7:::
sync:*:19430:0:99999:7:::
games:*:19430:0:99999:7:::
man:*:19430:0:99999:7:::
...

# 共9个字段
# 字段1：用户名
# 字段2：密码，分三部分
        # 第一部分：加密使用的单向加密算法
        # 第二部分：盐值
        # 第三部分：加密后的结果，即密文
# 字段3：创建/修改密码的时间，距离1970年1月1日，隔几天
# 字段4：最短有效期,即有效期内无法更改口令（0代表可随时更改）（该有效期仅针对普通用户，对root用户无效）
# 字段5：最长有效期
# 字段6：密码到期的前几天进行提醒
# 字段7：超过密码有效期，再过几天账号锁定
# 字段8：失效时间，从1970年1月1日算起，多少天后帐号失效，为空表示永不过期
# 字段9：  保留字段，无意义
```

`/etc/group`和`/etc/gshadow`(存放组用户和组密码)

```
# /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,mystical
tty:x:5:syslog
disk:x:6:
lp:x:7:
mail:x:8:
news:x:9:
...
后续省略

# /etc/gshadow
root:*::
daemon:*::
bin:*::
sys:*::
adm:*::syslog,mystical
tty:*::syslog
...
后续省略
```

用户管理常用命令

用户管理命令

- useradd
- usermod
- userdel

组管理常用命令

组管理命令

- groupadd
- groupmod
- groupdel

```
# groupadd 新建用户组

格式：groupadd [OPTION]... group_name

常见选项：
-g GID 指明GID号；[GID_MIN,GIDMAX]
-r 创建系统组，CentOS6之前：ID<500，CentOS 7以后：ID<1000

注意：
如果你知道你要创建的是一个系统组，并且你想确保它在系统组的 GID 范围内，那么使用 -r 选项是一个好的实践。如果你只是想创建一个具有特定 GID 的组，不管它是否是系统组，那么只使用 -g 选项就足够了。

添加 -r 选项是为了明确表达你的意图，并确保组被正确地分类为系统组。不过，如果你手动指定了一个在系统组 GID 范围内的 GID，即使没有使用 -r 选项，该组在某种程度上也被视为系统组。

范例：
groupadd -g 48 -r apache


# groupmod 组属性修改

格式：groupmod [OPTION]... group

常见选项：
-n <新组名> <原组名>: 新名字
-g GID : 新的GID

示例：groupmod -n www apache

# groupdel 删除组
groupdel  group_name     删除用户组
```

理解并设置文件权限

Linux指令权限管理

```
su <用户名>       -- 切换为用户名的身份权限
-- 不完全切换，身份切换了，但是环境还是之前的root路径下
-- 而且环境变量依然是之前切换前的环境变量，对于依赖个人用户环境配置的程序执行，可能会出现非预期的现象
su - <用户名>    -- 完全切换，身份和所在路径都切换了
su - <用户名> -c 'cmd'
-- 以切换的指定身份执行命令，但本身不切换身份，依然是当前用户
exit     -- 切换回上一个身份
```

实验：测试`su <用户名>`和`su - <用户名>`的区别

```
[root@ubuntu2204 ~]$ export NAME=magedu
[root@ubuntu2204 ~]$ echo $NAME
magedu
[root@ubuntu2204 ~]$ su mystical
mystical@ubuntu2204:/root$ echo NAME
NAME
mystical@ubuntu2204:/root$ echo $NAME
magedu
mystical@ubuntu2204:/root$ exit
exit
[root@ubuntu2204 ~]$ su - mystical
mystical@ubuntu2204:~$ echo $NAME
```

补充知识点：export

```
export 是一个 Shell 内置命令，用于将一个变量导出到子进程环境中，使得子进程能够继承该变量。
```

没有`export`的变量

- **作用范围：**
  - 变量只在当前 Shell 进程中有效。
  - 它是一个**局部变量**，无法被子进程继承。
- **例子：**

  ```
  bashCopy codeNAME="magedu"  # 定义变量，但没有 export

  echo $NAME  # 输出 "magedu"

  bash  # 启动一个子 Shell
  echo $NAME  # 无输出，因为子 Shell 没有继承父 Shell 的变量
  ```

  解释：
  - 在没有`export`的情况下，`NAME`变量是父 Shell 的局部变量。
  - 子 Shell 是一个独立的进程，无法访问父 Shell 的局部变量。

使用`export`的变量

- **作用范围：**
  - 变量不仅在当前 Shell 有效，还能被子进程继承。
  - 它是一个**全局变量**（对当前 Shell 和所有子进程有效）。
- **例子：**

```
bashCopy codeexport NAME="magedu"  # 定义并导出变量

echo $NAME  # 输出 "magedu"

bash  # 启动一个子 Shell
echo $NAME  # 输出 "magedu"，因为子 Shell 继承了变量
```

解释：

- `export` 将变量标记为可供子进程继承。
- 子 Shell 进程继承了 `NAME` 变量，因此可以访问它

Linux文件权限管理

```
root@clem:~# ll
total 32
drwx------  4 root root 4096 Jun 27 08:54 ./
drwxr-xr-x 20 root root 4096 Jun 28 20:32 ../
-rw-------  1 root root  458 Jun 28 22:20 .bash_history
-rw-r--r--  1 root root 3204 Jun 27 08:38 .bashrc
-rw-r--r--  1 root root  161 Dec  5  2019 .profile
drwx------  3 root root 4096 Dec 21  2022 snap/
drwx------  2 root root 4096 Dec 21  2022 .ssh/

解析：-文件类型 权限 链接数 所属账号 所属主组 大小 时间 文件名
```

```
chown   -- 更改文件用户权限/更改所有权限

# chown user_name file_name
-- 将file文件的用户权限改为user_name

# chown user_name.group_name file_name
# chown user_name:group_name file_name
-- 将该文件所属的用户名，组名一起变更。

# chown -R user_name.group_name dir
-- 将文件夹下，所有文件的所属账号和组都一起变更，危险命令

chgrp   --仅更改文件所属组权限

# chgrp group_name file_name
-- 更改所属组
```

文件权限类型

- r Readable 读
- w Writable 写
- x eXcutable 执行

```
示例：
-rw-r--r--  1 root root  161 Dec  5  2019 .profile

- rw- r-- r--
第一个‘-’不在权限标识中，仅指文件类型
第一组‘rw-’标明所属用户权限
第二组‘r--’同组其他用户权限
第三组‘r--’其他用户权限，通用权限，所有人基本都能用
```

- rwx对于目录的权限意义（与文件不同）
  - r：可以使用ls查看此目录中文件名列表，但无法看到文件的属性meta信息，包括inode号，不能查看文件的内容
  - w：可以在此目录中创建文件，也可以删除此目录中的文件，而和此被删除的文件的权限无关。
  - x：如果，没有该目录的执行权限，用户将无法访问这个目录下的所有文件，所以执行权限是目录访问的基本权限，没有执行就无法进入，是的，连目录进都进不去！
    - 如果只有x,没有r的话，对于目录来说，就是只能访问，但是看不到ls，就是没有访问目录下文件名的权限，但是如果这个文件你知道名称，且这个文件的通用权限有读权限，那么对于普通用户来说，只是无法浏览目录下文件名及文件元信息，但是依然可以cat到文件内的内容

目录权限辨析

```
# 首先使用root用户，在/root下创建一个test目录
mkdir test

# 在目录创建两个文件
echo aaa > a.txt
echo bbb > b.txt

# 将test目录即里面所有文件的所属主和所属组改为mystical
chown -R mystical:mystical test/

# 将test目录权限改为r--
chmod 400 /root/test

# 查看当前权限
tree -p /root
[drwx------]  /root                # 所属主:所属组 --- root:root
└── [dr--------]  test             # 所属主:所属组 --> mystical:mystical
    ├── [-rw-r--r--]  a.txt        # 所属主:所属组 --> mystical:mystical
    └── [-rw-r--r--]  b.txt        # 所属主:所属组 --> mystical:mystical
```

```
# 因为test目录有读权限(r)，因此理论上su mystical -c 'ls /root/test'，可以读取目录下的文件，但实际上
su mystical -c 'ls /root/test'
ls: cannot access '/root/test': Permission denied

# 原因是/root的权限是rwx------，其它是没有x权限的，因此无法访问root下的test目录
# 给root的其他加一个执行权限(x)，可以ls /test目录下的文件
chmod o+x /root
su - mystical -c 'ls /root/test/'
a.txt  b.txt

# 但是ls -i则是能读取文件名，无法读取inode
su - mystical -c 'ls -i /root/test/'
ls: cannot access '/root/test/b.txt': Permission denied
ls: cannot access '/root/test/a.txt': Permission denied
? a.txt ? b.txt

######################
# 为什么 x 权限是关键
######################

# 当只有 r 权限时，目录中的文件名是直接存储在目录文件中的数据，可以直接读取。
# 然而，inode 信息涉及路径解析（文件名 -> inode），这需要 x 权限。
# x 权限允许操作系统进入目录，并通过路径解析机制获取 inode 信息。

# 只有x权限，看不到该目录下有哪些文件，但是，如果知道文件名，可以读取文件内容
[root@ubuntu2204 ~]$ su - mystical -c 'cat /root/test/a.txt'
aaa
[root@ubuntu2204 ~]$ su - mystical -c 'ls /root/test'
ls: cannot open directory '/root/test': Permission denied
```

总结

- **目录的 `r` 权限：** 只允许读取目录的文件名。
- **目录的 `x` 权限：** 允许访问 inode 信息，并解析路径。
- **目录的 `r + x` 权限：** 才能完整地列出文件名及其 inode 信息。
- **缺少 `x` 权限时，`ls -i` 无法查看 inode 信息，因为路径解析被限制。**

更改文件的权限

```
chmod    -- 模式法和数字法

模式法：
chmod who opt per file
-- who：u,g,o,a
-- u(所属者)，g(所属组)，o(other),a(all)
-- opt:+,-,=
-- per:r,w,x
```

```
示例：
chmod o+r file  -- 表示file文件的通用权限中增加r权限

数字法：
rwx rw- --- a.txt
111 110 000 a.txt
7   6   0
chmod 760 a.txt
```

默认权限

定义：当创建一个文件或文件夹时，会默认一个权限，这个默认权限时如何产生的，如何修改

```
umask  -- 这个指令可以修改新建文件/文件夹的权限

# umask -- 查看当前umask的值
-- root权限的默认umask值022
-- 普通用户的默认umask值002
------------------------------------------------
修改默认权限的实现方式

指定新建文件的默认权限
666-umask，如果所得结果某位存在执行（奇数）权限，则将其权限+1，偶数不变

将权限+1的原因：
文件的执行时危险的！！！，如果没有执行权限，root也无法直接执行，但是没有读写权限，root依然能够进行读写

基于安全考虑，默认新建的文件不允许有执行权限！！

umask 的内在机制

666
123   -- umask值

文件权限为：644
------------------------------------------------
指定新建目录的默认权限

777-umask
------------------------------------------------
修改默认权限
# umask <更改后的数字>     -- 临时修改

永久修改：
root目录下，.bashrc文件内修改，添加umask <数值>，保存退出后，. .bashrc或者重启
-- 全局设置：/etc/bashrc 不建议，这里修改会影响全局所有用户
-- 用户设置：~/.bashrc 只影响当前用户
```

特殊权限

什么是 SUID（Set User ID）？

SUID (Set User ID) 是 Linux/Unix 文件权限的一种特殊位，主要用于**二进制可执行文件。**当某个文件设置了 SUID 位时，**执行该文件的用户将临时获得文件的所有者（owner）的权限**，而不是使用当前用户的权限。

SUID实验1

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    FILE *file = fopen(argv[1], "a");
    if (file == NULL) {
        perror("fopen");
        exit(EXIT_FAILURE);
    }
    fprintf(file, "hello, i'm mystical\n");
    fclose(file);
    return 0;
}
```

```
gcc output.c -o output

# 使用root权限执行
# -rw-r--r--  1 root     root        29 Dec 10 09:15 root.txt
./output root.txt

# 使用root执行成功
[root@ubuntu2204 ~]$ cat root.txt
i'm root
hello, i'm mystical

# 使用mysicald普通用户身份执行，没有权限
[root@ubuntu2204 ~]$ su mystical -c '/root/output /root/root.txt'
fopen: Permission denied

# 将output这个命令加上SUID权限，执行成功
[root@ubuntu2204 ~]$ chmod 4755 /root/output
[root@ubuntu2204 ~]$ su mystical -c '/root/output /root/root.txt'
[root@ubuntu2204 ~]$ cat root.txt
i'm root
hello, i'm mystical
```

```
hello, i'm mystical
```

SUID实验2

```
# 直接使用mystical普通用户权限使用tail程序，读取/etc/shadow，没有权限
[root@ubuntu2204 ~]$ su mystical -c 'tail -f /etc/shadow'
tail: cannot open '/etc/shadow' for reading: Permission denied
tail: no files remaining

# 给tail程序添加一个SUID权限
[root@ubuntu2204 ~]$ chmod u+s `which tail`
[root@ubuntu2204 ~]$ ll `which tail`
-rwsr-xr-x 1 root root 68120 Feb  7  2022 /usr/bin/tail*

# 执行成功
[root@ubuntu2204 ~]$ su mystical -c 'tail -f /etc/shadow'
sshd:*:19579:0:99999:7:::
syslog:*:19579:0:99999:7:::
uuidd:*:19579:0:99999:7:::
tcpdump:*:19579:0:99999:7:::
tss:*:19579:0:99999:7:::
landscape:*:19579:0:99999:7:::
fwupd-refresh:*:19579:0:99999:7:::
usbmux:*:19741:0:99999:7:::
mystical:$6$N7E7viAPFbu5NUsZ$.hxffkw7icJNQxXWVjEH11h1E8bMCiFKiyBQylf1d22wbFsAlPDM
Olw8DOp7C..OEa52TgDTdJbTkcI.7J4zA.:19741:0:99999:7:::
lxd:!:19741::::::

# 查看进程状态
[root@ubuntu2204 ~]$ ps aux | grep tail
root        5872  0.0  0.2  10328  4616 pts/0    S+   09:26   0:00 su mystical -c
tail -f /etc/shadow
root        5873  0.0  0.0   5804   992 ?        Ss   09:26   0:00 tail -f
/etc/shadow
```

什么是 SGID（Set Group ID）？

SGID（Set Group ID）是一种**特殊权限位**，用于控制**目录和可执行文件的访问行为。**SGID 在文件和目录中的含义不同。

- **文件上的 SGID：**
  - 当 SGID 设置在可执行文件上时，**运行该文件的用户将暂时获得与文件所属组相同的组权限**，而不是使用当前用户的主组权限。
  - SGID 在可执行文件上不常见，更多用于一些系统命令（如 `passwd` 命令）。
- **目录上的 SGID：**
  - 当 SGID 设置在目录上时，**在该目录中创建的文件和子目录将自动继承该目录的所属组。**
  - 这在实际生产中非常常见，尤其是在**团队共享目录**中，确保团队成员创建的所有文件和子目录都具有一致的所属组。

SGID实验1

```
mkdir /data/shared
chgrp devteam /data/shared    # 修改目录的所属组为 devteam
chmod 2770 /data/shared       # 设置 SGID

ls -ld /data/shared
# drwxrws--- 2 root devteam 4096 Dec 9 13:21 /data/shared
```

SGID 在实际生产中的使用场景

场景 1：共享工作目录

- **场景描述**：
  - 团队成员（如开发人员）在共享目录中工作，所有的文件应属于同一个组，确保**所有人都能读取和修改文件。**
  - 如果没有 SGID，用户在目录中创建的文件会继承他们自己的主组。
  - 这会导致其他用户（如其他开发人员）无法对这些文件执行写入操作。
- **解决方法**：
  - 将 SGID 位设置在共享目录上，使得所有新创建的文件都继承目录的组。
- **实际操作**：

```
bashCopy codemkdir /data/project
chgrp devteam /data/project          # 修改组为 devteam
chmod 2770 /data/project             # 目录设置 SGID，rw 权限
```

- **效果**：
  - 在 `/data/project` 中，开发人员 user1 创建文件：

```
bashCopy codesu user1
touch /data/project/file1
ls -l /data/project/file1
# -rw-rw---- 1 user1 devteam 0 Dec 9 14:00 /data/project/file1
```

  - 在 `/data/project` 中，开发人员 user2 创建文件：

```
bashCopy codesu user2
touch /data/project/file2
ls -l /data/project/file2
# -rw-rw---- 1 user2 devteam 0 Dec 9 14:02 /data/project/file2
```

  - **注意**：不论是 user1 还是 user2，创建的文件的所属组始终是 `devteam`。

什么是 SBIT（Sticky Bit）？

Sticky Bit 是一种特殊权限位，通常用于目录，尤其是公共目录（如 `/tmp`），用来控制谁可以删除目录中的文件。

SBIT 的标志

- 通过 `ls -ld` 查看目录权限，**SBIT 位显示在“其他用户的执行位 (x)”的位置。**
- 当 SBIT 位被设置时，权限中的最后一个 `x` 变为 `t`。
  - **小写 t**：表示目录的执行权限和 SBIT 都被启用。
  - **大写 T**：表示目录没有可执行权限，但 SBIT 被启用。

SBIT 的作用

- SBIT 的核心作用是**限制目录中的文件删除权限。**
- **如果目录未设置 SBIT**，那么**任何用户都可以删除目录中的文件**，即使这些文件不是他创建的。
- **如果目录设置了 SBIT**，那么**只有文件的拥有者和 root 才可以删除这些文件**，即使其他用户对目录有写入权限。
- 这就是为什么 `/tmp` 目录始终设置了 SBIT。

SBIT实验

```
# 创建一个共享目录
[root@ubuntu2204 ~]$ mkdir share

# 给该共享目录SBIT权限
[root@ubuntu2204 ~]$ chmod 1777 share

#查看share
[root@ubuntu2204 ~]$ ll share -d
drwxrwxrwt 2 root root 4096 Dec 10 09:42 share/

# 使用不同的用户身份在该目录下各自创建一个文件
[root@ubuntu2204 ~/share]$ su mystical -c 'echo "this is mystical" >
/root/share/mystical.txt'
[root@ubuntu2204 ~/share]$ su tom -c 'echo "this is tom" > /root/share/tom.txt'

# 查看这个目录下的权限
[root@ubuntu2204 ~/share]$ ll
total 16
drwxrwxrwt 2 root     root     4096 Dec 10 09:42 ./
drwx-----x 7 root     root     4096 Dec 10 09:40 ../
-rw-rw-r-- 1 mystical mystical   17 Dec 10 09:41 mystical.txt
-rw-rw-r-- 1 tom      tom        12 Dec 10 09:43 tom.txt

# 使用mystical用户身份删除tom的文件，删除失败
[root@ubuntu2204 ~/share]$ su mystical -c 'rm -f /root/share/tom.txt'
rm: cannot remove '/root/share/tom.txt': Operation not permitted
```

特殊属性（限制管理员-root)

```
# chattr +i a.txt   -- 添加i属性限制root

# lsattr a.txt    -- 查看i属性

# chattr -i     -- 删除i属性

# chattr +a     -- 只能追加，不能修改，不能删除

以下是一些chattr命令的常用属性：

1. a（Append only）
设置后，文件只能被追加内容，不能被删除或覆盖。这对于日志文件非常有用。
使用方法：chattr +a filename
2. i（Immutable）
设置后，文件变为不可修改，即不能被删除、修改、重命名，也不能添加链接。即使是root用户也不能绕过这一限制。
使用方法：chattr +i filename
```

ACL(Access Control List) 访问控制列表

- ACL权限功能：
  - rwx 权限体系中，仅仅只能将用户分成三种角色，如果要对单独用户设置额外的权限，则无法完成；而ACL可以单独对指定的用户设定各不相同的权限；提供颗粒度更细的权限控制
  - CentOS7 默认创建的xfs和ext4文件系统具有ACL功能
  - CentOS7 之前版本，默认手工创建的ext4文件系统无ACL功能,需手动增加

```
tune2fs –o acl /dev/sdb1
mount –o acl /dev/sdb1 /mnt/test
```

- ACL安装

```
sudo apt install acl
```

- ACL生效顺序：

```
所有者，自定义用户，所属组，自定义组，其他人
```

- ACL相关命令
  - getfacl 可查看设置的ACL权限

```
# Display the file access control list:
getfacl {{path/to/file_or_directory}}

# Display the file access control list with numeric user and group IDs:
getfacl -n {{path/to/file_or_directory}}

# Display the file access control list with tabular output format:
getfacl -t {{path/to/file_or_directory}}
```

- setfacl 可设置ACL权限

```
setfacl [-bkndRLPvh] [{-m|-x} acl_spec] [{-M|-X} acl_file] file ...
#常用选项
-m|--modify=acl              #修改acl权限
-M|--modify-file=file        #从文件读取规则
-x|--remove=acl              #删除文件acl 权限
-X|--remove-file=file        #从文件读取规则
-b|--remove-all              #删除文件所有acl权限
-k|--remove-default          #删除默认acl规则
--set=acl                    #用新规则替换旧规则，会删除原有ACL项，用新的替代，一定要包含UGO的设置，不能象 -m一样只有 ACL
--set-file=file              #从文件读取新规则
--mask                       #重新计算mask值
-n|--no-mask                 #不重新计算mask值
-d|--default                 #在目录上设置默认acl
-R|--recursive               #递归执行
-L|--logical                 #将acl 应用在软链接指向的目标文件上，与-R一起使用
-P|--physical                #将acl 不应用在软链接指向的目标文件上，与-R一起使用
```

- setfacl示例：

```
#设置 tom 无任何权限
[root@ubuntu2204 tmp]# setfacl -m u:tom:- f1
[root@ubuntu2204 tmp]# getfacl f1
# file: f1
# owner: root
# group: root
user::rw
user:tom:--
group::r-
mask::r-
other::r-
#查看文件，多了一个小 +
[root@ubuntu2204 tmp]# ll f1-rw-r--r--+ 1 root root 5 May  9 23:22 f1
```

- 编辑ACL规则文件
  - ACL规则文件是一个文本文件，其中每一行都包含一个ACL规则。这些规则的格式通常如下：

```
[类型]:[用户/组]:[权限]
```

ACL实验

```
[root@ubuntu2204 ~]$ chmod 600 root.txt
[root@ubuntu2204 ~]$ su mystical -c 'cat /root/root.txt'
cat: /root/root.txt: Permission denied
[root@ubuntu2204 ~]$ setfacl -m u:tom:r-- root.txt
[root@ubuntu2204 ~]$ getfacl a.txt
# file: a.txt
# owner: root
# group: root
user::rw-
group::r--
other::r--

[root@ubuntu2204 ~]$ getfacl root.txt
# file: root.txt
# owner: root
# group: root
user::rw-
user:tom:r--
group::---
mask::r--
other::---

[root@ubuntu2204 ~]$ su mystical -c 'cat /root/root.txt'
cat: /root/root.txt: Permission denied
[root@ubuntu2204 ~]$ su tom -c 'cat /root/root.txt'
i'm root
hello, i'm mystical
hello, i'm mystical
```