

Linux文件管理

1.文件系统目录结构

```
[jimbo@Rocky ~]$ ll /
total 24
lrwxrwxrwx.    1 root root    7 Oct 11  2021 bin -> usr/bin
dr-xr-xr-x.   5 root root 4096 Feb 11 09:37 boot
drwxr-xr-x.  20 root root 3160 Feb 11 09:37 dev
drwxr-xr-x.  89 root root 8192 Feb 11 10:32 etc
drwxr-xr-x.   3 root root   19 Feb  3 22:58 home
lrwxrwxrwx.    1 root root    7 Oct 11  2021 lib -> usr/lib
lrwxrwxrwx.    1 root root    9 Oct 11  2021 lib64 -> usr/lib64
drwxr-xr-x.    2 root root    6 Oct 11  2021 media
drwxr-xr-x.    2 root root    6 Feb 10 09:49 mnt
drwxr-xr-x.    2 root root    6 Oct 11  2021 opt
dr-xr-xr-x. 167 root root    0 Feb 11 09:37 proc
dr-xr-x---.    2 root root   151 Feb  8 14:16 root
drwxr-xr-x.   31 root root   880 Feb 11 09:37 run
lrwxrwxrwx.    1 root root    8 Oct 11  2021/sbin -> usr/sbin
drwxr-xr-x.    2 root root    6 Oct 11  2021 srv
dr-xr-xr-x.   13 root root    0 Feb 11 09:37 sys
drwxrwxrwt.    8 root root 4096 Feb 11 10:33 tmp
drwxr-xr-x.   12 root root   144 Feb  3 22:55 usr
drwxr-xr-x.   21 root root 4096 Feb  3 22:59 var
```

☁ `/` : 根目录

`/bin` : 存放基本命令

`/boot` : 存放内核文件和启动引导文件

`/dev` : 存放设备文件

`/home` : 普通用户目录

`/lib`、`/lib64` : 存放库文件

`/media` : 存放可移动设备文件

`/mnt` : 存放临时挂载外部设备文件

`/opt` : 存放第三方独立软件

`/proc` : 存放内核和进程实时信息

`/root`：存放root用户目录

`/run`：存放系统运行时的临时文件

`/sbin`：存放管理员使用的系统管理命令

`/srv`：存放服务产生的数据

`/sys`：存放硬件设备驱动和内核参数

`/tmp`：存放所有用户可用的临时文件目录

`/usr`：存放用户安装的应用程序和资源

`/var`：存放频繁变化的文件

☁ 文件的两类相关数据：metadata(元数据，即文件属性)、data(数据，即文件内容)

文件类型：

- `-`：普通文件，白色
- `d`：目录文件，蓝色
- `l`：符号链接文件，浅蓝色
- `b`：块设备文件
- `c`：字符设备文件，黄色
- `p`：管道文件，土黄色
- `s`：套接字文件，桃红色

2.文件操作命令

2.1 显示文件当前目录

```
1  pwd
```

☁ 绝对路径：从根出发，完整的文件位置路径

相对路径：不从根出发，当前工作目录路径

2.2 更改目录

```
1  # 命令: cd 目录地址
2  cd /          # 回到根目录(root)
3  cd ..        # 回到上一级目录, .为当前目录, ..为上一级目录
4  cd ~         # 回到家目录(home)
5  cd -         # 回到上一次访问的目录
6  cd /path     # 回到指定目录
```

2.3 列出目录内容

```
1  # 命令: ls [选项] [文件名]
2  # 常用命令
3  ls -a        # 显示隐藏信息
4  ls -l        # 显示额外信息
5  ls -1        # 一行显示一个
6  ll           # alias ll='ls -l --color=auto'
```

2.4 查看文件状态

☁ 每个文件有三个时间戳:

- `atime(access time)`: 访问时间, 读取文件内容时改变
- `mtime(modify time)`: 修改时间, 改变文件内容(data)时改变
- `ctime(change time)`: 改变时间, 改变元数据(metadata)时改变

```
1  # 命令: stat 文件名
2  stat test.txt
```

```
[jimbo@Rocky ~]$ stat test.txt
  File: test.txt
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: fd02h/64770d    Inode: 138         Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   jimbo)   Gid: ( 1000/   jimbo)
Access: 2025-02-11 17:30:59.099042428 +0800
Modify: 2025-02-11 17:30:59.099042428 +0800
Change: 2025-02-11 17:30:59.099042428 +0800
 Birth: 2025-02-11 17:30:59.099042428 +0800
```

2.5 确定文件内容

- 1 # 命令: `file` 文件名
- 2 `file test.txt` # 显示`test.txt`文件内容类型

```
[jimbo@Rocky ~]$ cat test.txt
#include<stdio.h>
int main{
    return 0;
}
[jimbo@Rocky ~]$ file test.txt
test.txt: C source, ASCII text
```

2.6 文件通配符

- ☁ 文件通配符可以用来匹配符合条件的多个文件，方便批量管理文件
- 通配符采用特定的符号，表示特定的含义，此特定符号称为元meta字符

- 1 # 常用文件通配符
- 2 `*` # 匹配零个或多个字符，但不匹配隐藏文件
- 3 `?` # 匹配任意单个字符
- 4 `~` # 当前用户家目录
- 5 `[0-9]` # 匹配数字范围
- 6 `[a-z]` # 匹配一个字母，不区分大小写
- 7 `[A-Z]` # 匹配一个字母，不区分大小写

```
8 [jimbo] # 匹配列表中任意一个字符
9 [^jimbo] # 排除列表中任意一个字符
10 [^a-z] # 排除列表中任意一个字符
11 [!jimbo] # 排除列表中任意一个字符
12 . # 当前工作目录
13 {} # 生成多个匹配模式
14
15 # 预定义字符类文件通配符
16 [:digit:] # 任意一个数字, 相当于0-9
17 [:lower:] # 任意一个小写字母, 相当于a-z
18 [:upper:] # 任意一个大写字母, 相当于A-Z
19 [:alpha:] # 任意一个大小写字母
20 [:alnum:] # 任意一个数字或字母
21 [:blank:] # 水平空白字符
22 [:space:] # 水平或垂直空白字符
23 [:punct:] # 标点符号
24 [:print:] # 可打印字符
25 [:cntrl:] # 控制(非打印)字符
26 [:graph:] # 图形字符
27 [:xdigit:] # 十六进制字符
```

```
1 # 常用文件通配符举例
2 ls *.txt # 列出当前目录下所有.txt文件
3
4 ls file?.txt # 匹配file1.txt、fileA.txt, 但不匹配file10.txt
5
6 cd ~ # 返回用户家目录
7
8 ls file[123].txt # 匹配file1.txt、file2.txt、file3.txt
9 ls file[a-c].txt # 匹配filea.txt、fileA.txt、fileb.txt、fileB.txt、
filec.txt、fileC.txt
10 ls file[!1-3].txt # 匹配不是file1.txt、file2.txt、file3.txt的文件
11
12 ls . # 显示当前工作目录
13
14 touch file{1,2,3} # 创建file1、file2、file3
15
16 # 预定义字符类文件通配符举例
17 ls file[[:digit:]].txt # 列出匹配file0.txt, file1.txt, ..., file9.txt的文件名
18
19 ls file[[:lower:]].txt # 列出匹配filea.txt, fileb.txt, ..., filez.txt的文件名
20
21 ls file[[:upper:]].txt # 列出匹配fileA.txt, fileB.txt, ..., fileZ.txt的文件名
22
23 ls file[[:alpha:]].txt # 列出匹配filea.txt, fileA.txt, ..., fileZ.txt的文件名
```

```
24
25  ls file[[:alnum:]].txt      # 列出匹配filea.txt, file1.txt, ..., file9.txt的文件名
26
27  grep "[[:blank:]]" file.txt  # 查找file.txt中包含空格或制表符的行
28
29  grep "[[:space:]]" file.txt  # 查找file.txt中包含任意空白字符的行
30
31  grep "[[:punct:]]" file.txt  # 查找file.txt中包含标点字符的行
32
33  grep "[[:print:]]" file.txt  # 查找file.txt中包含可打印字符的行
34
35  grep "[[:cntrl:]]" file.txt  # 查找file.txt中包含控制字符的行
36
37  grep "[[:graph:]]" file.txt  # 查找file.txt中包含图形字符的行
38
39  grep "[[:xdigit:]]" file.txt # 查找file.txt中包含十六进制字符的行
```

2.7 创建空文件和刷更新时间

```
1  # 命令: touch [选项] 文件名
2  选项:
3      -a: 改变atime和ctime
4      -m: 改变mtime和ctime
5      -t: 指定atime和mtime时间戳, [[CC]YY]MMDDhhmm[.ss]
6      -c: 如果文件存在则不创建
7
8  # 举例
9  touch test.txt      # 创建test.txt文件
```

2.8 复制文件和目录

```
1  # 命令: cp [选项] 源文件 目标文件
2  选项:
3      -r: 递归复制目录
4      -i: 交互复制
5      -f: 强制复制
6      -v: 显示详细过程
7      -p: 保留文件属性
8      -a: 归档模式, 常用于备份功能
```

```
9      -u: 仅复制源文件中比目标文件新的文件
10     -l: 创建硬链接而非复制文件, 一般使用ln 创建硬链接
11     -s: 创建软链接而非复制文件, 一般使用ln -s 创建软链接
12
13 # 举例
14 cp test.txt test.txt.bak      # 复制test.txt为test.txt.bak
```

2.9 移动和重命名文件

```
1 # 命令: mv [选项] 源文件 目标文件
2 选项:
3     -i: 交互式移动
4     -f: 强制移动
5     -b: 目标存在, 则覆盖前先备份
6
7 mv test.txt dir      # 移动test.txt文件到dir目录下
8 mv test.txt test2.txt # 重命名test.txt为test2.txt
9
10 # 批量化重命名: rename [选项] 表示 代替 文件名
11 rename 'conf' 'conf.bak' f*    # 为所有f开头包含conf的文件加上.bak后缀
```

2.10 删除文件

```
1 # 命令: rm [选项] 文件名
2 选项:
3     -i: 交互式删除
4     -f: 强制删除
5     -r: 递归删除
6
7 rm test.txt      # 删除test.txt文件
8 rm -rf test.txt  # 强制删除test.txt文件
9 rm -rf dir       # 强制删除目录
10 rm -- -f         # 删除名为-f的文件
11 rm ./-f         # 删除名为-f的文件
12
13 cat /dev/null > big.log    # 删除大文件, /dev/null是一个空内容
```

2.11 目录操作

```
1  # 显示目录树tree
2  tree
3
4  # 创建目录mkdir
5  mkdir dir    # 创建dir目录
6
7  # 删除空目录rmdir
8  rmdir dir    # 删除dir空目录
```

3.文件元数据和节点表结构

3.1 inode表结构

☁ 每个文件的属性信息称为文件的元数据(metadata)。元数据存放在inode表中。inode表是一个结构体

inode表包含：inode号、文件类型、权限、UID、GID、链接数、文件大小、时间戳、数据块指针、其他数据

- 数据块指针：直接指针、间接指针(一级、二级、三级)

示例 (ext4 文件系统)

C 复制

```
// ext4_inode 结构 (简化版)
struct ext4_inode {
    __le16 i_mode;           // 文件类型与权限
    __le16 i_uid;            // 所有者 UID 的低 16 位
    __le32 i_size_lo;        // 文件大小 (低 32 位)
    __le32 i_atime;          // 最后访问时间
    __le32 i_ctime;          // 最后状态变更时间
    __le32 i_mtime;          // 最后修改时间
    __le32 i_blocks_lo;      // 占用磁盘块数 (512B 单位)
    __le32 i_block[15];      // 数据块指针 (12 直接 + 1 间接 + 1 双间接 + 1 三间接)
    // ... 其他字段 (高 32 位 UID/GID、扩展属性等)
};
```

☁ inode和目录：

- 目录是一个特殊文件，其内容是文件名到inode编号的映射表
- 每个目录对应一个inode，存储目录的元数据

inode和cp:

- cp操作创建新的inode
- 复制软链接时，会复制链接本身(新inode)，而非目标文件
- 复制硬链接时，会创建文件名指向原inode(链接数+1)

inode和rm:

- 在目录中删除文件名到inode的映射
- inode的链接数-1
- 数据删除后，数据仍在硬盘中，直到被新数据覆盖

inode和mv:

- inode不变，仅修改目录映射，不涉及数据块复制

总结对比

操作	影响 inode	关键行为
cp	创建新 inode (除非覆盖文件)	数据块复制，新文件名指向新 inode
rm	可能释放 inode (链接计数归零)	仅删除目录条目，数据保留到被覆盖
mv	同一分区: inode 不变; 跨分区: inode 变	同一分区: 元数据更新; 跨分区: 复制+删除

3.2 硬链接与软链接

☁ 硬链接: 本质是取别名，实质是同一个文件，inode相同

软链接(符号链接): 本质是快捷方式，实质不是同一个文件，inode不同

特性	硬链接 (Hard Link)	软链接 (Symbolic Link)
本质	同一文件的不同文件名 (共享同一 inode)	独立文件, 存储目标文件路径 (有自己的 inode)
跨文件系统	✗ 不支持	✓ 支持
链接目标类型	仅限文件 (不能链接目录)	文件或目录均可
删除原文件	链接仍有效 (只要存在至少一个硬链接)	链接失效 (悬空链接)
链接计数	增加原文件的 inode 链接计数	不影响原文件的链接计数
文件大小	与原文件相同 (共享数据块)	等于路径字符串的长度 (如 <code>/path/to/file</code>)
权限与时间戳	与原文件一致 (共享 inode)	独立权限 (通常为 <code>rwXrwXrwx</code>), 时间戳独立

```
1 # 硬链接: ln 源文件 硬链接名
2 ln filename linkname
3
4 # 软链接: ln -s 源文件/目录 软链接名
5 ln -s filename linkname
```

3.3 inode常见问题

☁ 常见错误: 提示空间满 `No space left on device`, 但df可以看到空间很多, 为什么?

错误原因: 磁盘空间耗尽、inode数量耗尽

```
1 # 解决办法: 删除不必要文件、清理缓存文件、释放inode
2
3 # 删除不必要文件
4 sudo du -sh /var/log/* # 查找占用大量空间的文件或文件夹
5 sudo rm -rf /var/log/old_log/* # 删除不需要的文件
6
```

```
7 # 清理缓存文件
8 sudo apt-get clean
9 sudo rm -rf /var/cache/*
10
11 # 释放inode(通常是存在大量小文件占用了inode)
12 find /path/dir -type f -name "*.log" --delete # 查找和删除小文件释放inode
```

☁ 常见错误：提示空间快满，使用rm删除了很大的无用文件后，df仍然看到空间不足，为什么？如何解决？

错误原因：rm删除的文件正在被使用，所以不能删除

```
1 # 解决办法：kill杀死进程、大文件清除
2
3 # kill杀死进程
4 lsof | grep delete # 查看被删除的文件
5 w # 查看正在执行的程序
6 kill -9 PID # 彻底杀死程序
7
8 # 大文件清除
9 cat /etc/null > big.log
```

4.重定向和管道

4.1 重定向

☁ 文件描述符：`fd`，打开的文件都有一个fd


重定向：改变命令的输入/输出方向。默认输入是键盘，默认输出是屏幕

- `stdin`：标准输入，fd为0
- `stdout`：标准输出，fd为1
- `stderr`：标准错误，fd为2

常见用法

操作符	说明	示例
>	覆盖写入文件 (stdout)	<code>echo "text" > file</code>
>>	追加写入文件 (stdout)	<code>echo "text" >> file</code>
<	从文件读取输入 (stdin)	<code>wc -l < file</code>
2>	重定向 stderr (覆盖)	<code>cmd 2> error.log</code>
2>>	重定向 stderr (追加)	<code>cmd 2>> error.log</code>
&> 或 > file 2>&1	合并 stdout 和 stderr 到同一文件	<code>cmd &> all.log</code>
n>&m	将文件描述符 n 重定向到 m 的当前目标	<code>cmd > file 2>&1</code> (合并输出)

4.2 管道

 管道：符号 `|`

管道作用：将一个命令的 `stdout` 传递给另一个命令的 `stdin`，形成处理链

```
1  # 基础管道
2  cmd1 | cmd2 | cmd3
3
4  # 管道与重定向结合
5  cmd1 2> error.log | cmd2 > output.log
6
7  # 多路输出分流
8  cmd1 | tee log.txt | cmd2  # tee 同时输出到屏幕和文件
```

5.文件权限管理



```
lrwxrwxrwx. 1 root root 7 Oct 11 2021 bin -> usr/bin
```

☁ 文件描述有11位，第一位为文件类型，后九位为权限描述，第十一位为安全标识 (SELINUX)

5.1 文件所有者和所属组

```
1 # 设置文件所有者和所属组: chown
2 chown [选项] [owner]:[group] filename
3
4 sudo chown jimbo:jimbo test.txt # 修改test.txt的所有者为jimbo, 所属组为jimbo
5 sudo chown jimbo test.txt # 修改test.txt的所有者为jimbo
6 sudo chown :jimbo test.txt # 修改test.txt的所属组为jimbo
7
8 # 设置文件所属组: chgrp
9 sudo chgrp 新组名 文件名
10
11 sudo chgrp jimbo test.txt # 修改test.txt的组名为jimbo
```

5.2 文件权限

☁ 文件权限说明:

- 三类对象: 所有者(owner, **u**)、所属组(group, **g**)、其他人(other, **o**)
- 三类常用权限: 可读(**r**, **4**)、可写(**w**, **2**)、可执行(**x**, **1**)
- 三类特殊权限: SUID、SGID、**Sticky**(粘滞位)
 - SUID、SGID作用于二进制可执行文件, 用来授权可执行权限
 - Sticky作用于目录, 用来防止误删文件
 - 一般不用SUID、SGID, 因为他们权限太大了。一般都用Sticky防止误删文件

```

1  # 修改文件权限: chmod
2  格式: chmod [选项] [MODE] 文件名
3
4  # 假如test.txt的权限为rwxrwxrwx(777)
5  chmod u-r test.txt      # 删除test.txt所有者读权限
6  chmod u+r test.txt      # 添加test.txt所有者读权限
7  chmod 377 test.txt      # 删除test.txt所有者读权限
8  chmod 777 test.txt      # 添加test.txt所有者读权限
9
10 # 特殊权限: SUID、SGID、Sticky
11 SUID: s让所有者拥有可执行权限、S让所有者没有可执行权限
12 SGID: s让所属组拥有可执行权限、S让所属组没有可执行权限
13 Sticky: t让其他人拥有可执行权限、T让其他人没有可执行权限
14
15 chmod u+s test.txt      # 添加test.txt所有者可执行权限
16 chmod u-s test.txt      # 删除test.txt所有者可执行权限
17 chmod u+S test.txt      # 删除test.txt所有者可执行权限
18
19 chmod g+s test.txt      # 添加test.txt所属组可执行权限
20 chmod g-s test.txt      # 删除test.txt所属组可执行权限
21 chmod g+S test.txt      # 删除test.txt所属组可执行权限
22
23 chmod o+t test.txt      # 添加test.txt其他人可执行权限
24 chmod o-t test.txt      # 删除test.txt其他人可执行权限
25 chmod o+T test.txt      # 删除test.txt其他人可执行权限
26 chmod o+t dir           # 添加dir目录粘滞位, 防止非所有者误删此文件
27 chmod o-t dir           # 删除dir目录粘滞位, 可被非所有者误删此文件

```

5.3 新建文件和目录的默认权限

☁ umask值可以改变文件默认权限

实现方法: 默认文件权限=666-umaks、默认目录权限=777-umask

- 普通用户umask: 默认0002、root用户umask: 默认为0022

```

1  # 查看默认umask
2  umask
3
4  # 临时修改umask

```

```
5  umask #      # 修改umask为#
6
7  # 永久修改umask
8  全局修改: /etc/bashrc
9  局部修改: ~/.bashrc
10 最后一行加上: umask
```

5.4 设置文件特殊属性

```
1  # 设置文件特殊属性: chattr
2  chattr +i file      # 不能删除、不能改名、不能更改
3  chattr +a file      # 只能追加内容, 不能删除、不能改名
4  lsattr              # 显示特殊属性
```

5.5 控制访问列表ACL

☁ ACL: Access Control List, 控制访问列表, 实现灵活的权限管理

ACL生效顺序: 所有者, 自定义用户, 所属组, 其他人

操作	命令示例
添加用户 ACL	<code>setfacl -m u:name:perms file</code>
添加组 ACL	<code>setfacl -m g:group:perms file</code>
设置默认 ACL	<code>setfacl -m d:u:name:perms dir</code>
递归设置 ACL	<code>setfacl -R -m g:group:rwx dir</code>
删除 ACL 条目	<code>setfacl -x u:name file</code>

```
1  # 设置ACL权限: setfacl
2  setfacl [选项] 规则 文件名
```

```
3  常见选项：
4      -m：添加或修改ACL条目
5      -x：删除ACL条目
6      -b：删除所有ACL条目
7      -d：操作默认ACL（仅目录）
8      -R：递归应用
9
10 常见规则：
11      u:用户名:权限（用户条目）
12      g:组名:权限（组条目）
13      d:u:用户名:权限（默认用户条目，仅对目录有效）
14      m::权限（修改掩码）
15
16  setfacl -m u:jimbo:rw test.txt      # 允许jimbo用户读写test.txt文件
17  setfacl -m g:admins:rx project/    # 允许组admin读写执行目录
18  setfacl -x u:jimbo test.txt        # 删除jimbo用户的ACL条目
19  setfacl -b test.txt                # 清除文件的所有ACL规则
20
21  # 查看设置的ACL权限: getfacl
22  getfacl test.txt                   # 验证ACL
```