文件类型

- 概述
  - 磁盘中存放的每个文件可以分为两个部分
    - 一部分为文件的内容：即文件的数据部分，此部分内容存放在磁盘中专门的数据空间(data block)中。
    - 一部分为文件的属性信息，即元数据(meta data)，比如；文件的大小，类型，节点号，权限，时间等，此部门内容存放在磁盘中专门的节点空间，inode表中
- 普通文件（白色）
  - 纯文本文件：
    - `ls -l /etc/issue`
  - 二进制可执行文件（绿色）：
    - 概述：二进制可执行文件是有特殊格式的可执行程序，其文件内容表现为不可直接读懂的字符，用cat查看，会出现乱码。在Linux中有很多二进制可执行文件，比如很多的外部命令都是二进制可执行文件
    - `ls -l /bin/cat`
  - 数据格式文件
    - 概述：数据格式文件是一些程序在运行过程中需要读取的存放在某些特定格式的数据文件，比如：图片文件，压缩文件，日志文件。通常需要特定的工具打开
    - 举例：用户登录时，系统会将登录的信息记录在/var/log.wtmp文件中，这个就是一个数据文件。需要使用`last`命令打开此文件查看内容
    - `ls -l /var/log/wtmp` -> `last`(直接在/var/log目录下使用last命令)
- 目录文件（蓝色）
  - 概述：目录文件即文件夹，通过`ls -l`查看文件属性时，第一个属性表现为d
- 链接文件（浅蓝色）
  - 概述：即将两个文件建立关联关系，这种操作实际上是给系统中已有的某个文件指定另外一个可用于访问它的不同文件名称。
  - `ls -l`查看文件属性时，第一个属性表现为l
  - 分类：
    - 硬链接
    - 软链接
- 管道文件（暗黄色）
  - 管道pipe文件是一种特殊的文件类型，其本质是一个伪文件（本质是内核缓冲区）。其主要目的是实现进程间通讯的问题。由于管道文件是一个与进程没有“血缘关系”的，真正独立的文件，所以它可以在任意进程之间实现通信。
  - 管道的本质：在内核中分配了 **一个块缓冲区**，这个缓冲区既用于写入数据，也用于读取数据。
  - 因为只有一个缓冲区，所以数据只能从写端流向读端，无法实现双向同时通信。
  - 内核缓冲区的作用：
    - 内核缓冲区是内存中的一块区域，由操作系统内核管理。

        - 主要用于在数据生产者（写入者）和消费者（读取者）之间暂存数据。
        - 缓冲区实现了数据的异步传递，即数据生产者和消费者无需同时工作，可以通过缓冲区解耦。

- 套接字文件（粉色）
    - 概述：数据接口文件，通常被用在基于网络的数据通讯使用。
    - 当两个进程在同一台主机上，但是像通过网络方式通信，可基于socket方式进行数据通信，可基于全双工方式实现，即可支持同时双向传输数据。
    - `ls -l`查看文件属性时，第一个属性表现为s
    - 套接字的基本工作机制：
        - **内核缓冲区的双向性**：
            - 套接字通常是全双工通信，可以同时进行读写。
            - 内核为每个套接字分配两块缓冲区：
                - **发送缓冲区**：存储写入的数据，等待传输。
                - **接收缓冲区**：存储接收到的数据，等待读取。
    - **数据流动过程**：
        - 写入数据：
            - 发送进程调用 `write` 或 `send`，将数据写入发送缓冲区。
            - 内核负责将缓冲区中的数据通过网络协议栈传输给远端。
        - 接收数据：
            - 数据到达远端后存储到接收缓冲区。
            - 接收进程调用 `read` 或 `recv`，从接收缓冲区读取数据。
    - 网络协议的作用
        - 套接字文件的特点在于它与网络协议栈紧密结合：
            - 内核缓冲区中的数据会经过网络协议栈的封装（TCP、UDP 等协议）进行传输。
            - 数据接收后会被解封装，放入接收缓冲区供应用程序读取。
    - **深度理解内核缓冲区**
        - **内核缓冲区设计目标**
            - **解耦读写双方**：
                - 缓冲区允许生产者和消费者以不同的速率运行，无需实时同步。
            - **优化性能**：
                - 缓冲区减少了系统调用的频率：
                    - 一次写入操作可以将多次写入的数据累积到缓冲区。
                    - 一次读取操作可以批量提取缓冲区中的数据。
            - **保证数据完整性**：
                - 在网络通信中，内核缓冲区负责数据的重新排序和校验，保证数据传输的可靠性（特别是 TCP）。

- 字符设备文件（明黄色）
    - 通常是一些串行接口设备在用户空间的体现，像键盘、鼠标。字符设备是按字符为单位进行输入输出的，且按一定的顺序进行

- `ls -l`查看文件属性时，第一个属性表现为c
- 举例；我们登录到Linux主机，系统会提供一个终端文件tty供我们登录。
- 字符设备文件与管道文件的本质区别
  - 字符设备文件的核心在于其与硬件的交互，而管道文件的核心在于进程间通信的能力。

- 块设备文件（明黄色）
  - 块文件设备，就是一些以“块为单位”，如：4096个字节，访问数据，提供随机访问的接口设备，例如磁盘、硬盘、U盘
  - 字符设备与块设备文件的对比：
    - **字符设备** 通常将数据直接传递给用户程序，数据在用户程序读取或写入之前，会先进入内核的缓冲区。
    - **块设备** 则依赖页缓存和缓冲区，对数据进行更多的管理和优化。
  - 块设备：页缓存和缓冲区
    - 页缓存概念
      - 块设备使用页缓存（Page Cache）优化数据读写
        - 页缓存是内核内存中的一部分，用于缓存块设备的读写操作。
        - 当用户程序读取块设备时，内核优先从页缓存中获取数据；只有当页缓存未命中时，才会从设备实际读取。
    - 页缓存的作用：
      - 读优化
        - 避免频繁的设备访问，加速数据读取。
        - 如果数据已在页缓存中，直接返回给用户程序，无需访问设备。
      - 写优化
        - 写操作首先写入页缓存，内核异步将数据刷入设备（延迟写）。
        - 这种机制提高了写入速度，但可能导致数据丢失（在系统崩溃时）。
    - 设备缓冲区（Device Buffer）
      - 块设备（如硬盘、SSD）通常还有硬件缓冲区，用于临时存储正在读写的数据。
      - 硬件缓冲区的大小和实现由设备本身决定。

Cache/Buff理解实验

```
# 查看初始buff/cache大小
[root@magedu log]$ free -h
               total        used        free      shared  buff/cache
available
Mem:           3.8Gi       528Mi       2.8Gi       1.8Mi       721Mi
3.3Gi
Swap:          3.8Gi          0B       3.8Gi
# 创建一个100M的文件
[root@magedu log]$ dd if=/dev/random of=test.img bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 0.435994 s, 241 MB/s
# 将文件写入/dev/null块文件
[root@magedu log]$ cat test.img > /dev/null
```

```
[root@magedu log]$ free -h
               total        used        free      shared  buff/cache
available
Mem:           3.8Gi       539Mi       2.7Gi       1.8Mi       824Mi
3.3Gi
Swap:          3.8Gi          0B       3.8Gi

# 清空页缓存和缓冲区
[root@magedu log]$ echo 3 > /proc/sys/vm/drop_caches
[root@magedu log]$ free -h
               total        used        free      shared  buff/cache
available
Mem:           3.8Gi       478Mi       3.4Gi       1.8Mi       131Mi
3.3Gi
Swap:          3.8Gi          0B       3.8Gi
```

- **页缓存的全局性**：
  - 页缓存是文件系统优化的核心，涉及所有设备和文件系统，因此需要统一管理。
  - 通过全局管理，操作系统可以更高效地分配和回收内存。

  缓冲区的独立性：
  - 缓冲区与具体的块设备强相关，每个块设备的元数据和 I/O 数据需要单独管理。
  - 独立管理可以避免不同设备之间的缓存数据冲突。

管理目录类文件相关命令

文件类型颜色的配置文件

```
# CentOS
/etc/DIR_COLORS

# Ubuntu
Ubuntu 中与颜色设置相关的文件和命令包括：

~/.dircolors 或 ~/.dir_colors:

用户级别的配置文件。如果存在，dircolors 命令会使用这个文件中的配置。如果你想定制自己的颜色配置，可以在你的用户目录中创建这个文件。
/etc/dircolors:

系统级别的默认配置文件。这个文件可能在某些系统中不存在，或者命名可能有所不同。
dircolors 命令:

这个命令用于初始化颜色配置。它会检查 ~/.dircolors 或 ~/.dir_colors 文件，如果这些文件不存在，它会使用默认的颜色配置。你通常会在你的 shell 初始化文件中（比如 ~/.bashrc）看到类似于以下的命令：
test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
```

如果你想要调整 Ubuntu 中 ls 命令输出的颜色，你可以创建或编辑 ~/.dircolors 文件，并在该文件中定义你的颜色配置。然后，确保你的 shell 初始化文件（如 ~/.bashrc）中包含处理 dircolors 的命令。这样，每次你打开一个新的 shell 时，都会应用这些颜色设置。

管理目录类文件相关命令

- 查看当前目录
  - 命令：`pwd`

```
pwd -P # 输出真实物理路径
pwd -L # 默认，输出链接路径
```

- 基名与文件名

```
bashename <dir> #只输出文件名
# 示例：
basename `which cat`

dirname <dir>  # 只输出路径
# 示例：
dirname `which cat`
```

- 路径间移动
  - 命令：`cd`

```
cd -P  # 移动到真实物理路径
# 示例
cd -P /bin  # 实际移动到/usr/bin

cd -L # 默认，移动到链接路径

cd ~  # 移动到家目录
cd ~username  # 移动到指定用户的家目录

cd -  # 移动到上次所在的目录，之所以能移动到上次所在目录是因为有系统变量记录了这个数据
# $OLDPWD 记录上次所在目录；$PWD 记录当前所在目录
```

- 查看目录
  - 命令：`tree`

```
# 查看指定目录数的层级
tree -L 1 /


# 每个文件和目录前显示完整的相对路径
tree -f
[Sun Oct 15 10:08:22 7] root@rocky9:~ #tree -f /Storage/
/Storage
```

```
└── /Storage/test
    ├── /Storage/test/baidu.html
    ├── /Storage/test/ps_demo.txt
    ├── /Storage/test/rename.txt
    └── /Storage/test/robots.txt

1 directory, 4 files


1 directory, 4 files

# 每个文件和目录前显示最新更改时间
tree -D
[Sun Oct 15 10:10:36 11] root@rocky9:~ #tree -D /Storage/
/Storage/
└── [Oct 14 09:12]  test
    ├── [Sep 27 12:03]  baidu.html
    ├── [Oct 13 20:48]  ps_demo.txt
    ├── [Jan  3  2020]  rename.txt
    └── [Jan  3  2020]  robots.txt

1 directory, 4 files


1 directory, 4 files

# 每个文件和目录前显示文件大小
tree -s
[Sun Oct 15 10:08:30 8] root@rocky9:~ #tree -s /Storage/
/Storage/
└── [         79]  test
    ├── [       2381]  baidu.html
    ├── [        270]  ps_demo.txt
    ├── [       2814]  rename.txt
    └── [       2814]  robots.txt


# 每个文件和目录前显示文件/目录拥有者
tree -u
[Sun Oct 15 10:09:28 9] root@rocky9:~ #tree -u /Storage/
/Storage/
└── [root    ]  test
    ├── [root    ]  baidu.html
    ├── [root    ]  ps_demo.txt
    ├── [root    ]  rename.txt
    └── [root    ]  robots.txt


# 每个文件和目录前显示权限标示
tree -p
[Sun Oct 15 10:11:18 12] root@rocky9:~ #tree -p /Storage/
/Storage/
└── [drwxr-xr-x]  test
    ├── [-rw-r--r--]  baidu.html
    ├── [-rw-r--r--]  ps_demo.txt
    ├── [-rw-r--r--]  rename.txt
```

```
    └── [-rw-r--r--]  robots.txt


# 使用通配符对tree的目录进行筛选
tree -P pattern 这里的pattern不支持正则表达式，仅支持通配符
[Sun Oct 15 10:33:09 26] root@rocky9:~ #tree -P 'r*.txt' /Storage/
/Storage/
└── test
    ├── rename.txt
    └── robots.txt

1 directory, 2 files


1 directory, 2 files

常用通配符:
* 匹配任意数量的字符（包括零个）。
? 匹配任意一个字符。
[...] 匹配方括号中的任意一个字符。
```

- 创建目录
  - 命令：`mkdir`

```
语法格式：mkdir [pv] [-m mode] directory_name...

# mkdir在指定路径创建目录
mkdir /Storage/test   # 在Storage目录下创建一个test目录

# 默认在当前路径创建目录
mkdir dir1    # 在当前目录下创建名为dir1的目录

# 一次创建多个同级目录，每个目录间用空格隔开
mkdir dir1 dir2 dir3

# 创建多级目录
mkdir -p dir1/dir2/dir3


# mkdir在指定路径创建目录
mkdir /Storage/test   # 在Storage目录下创建一个test目录

# 默认在当前路径创建目录
mkdir dir1    # 在当前目录下创建名为dir1的目录

# 一次创建多个同级目录，每个目录间用空格隔开
mkdir dir1 dir2 dir3

# 创建多级目录
mkdir -p dir1/dir2/dir3

# -v 会显示创建每个目录的详细信息
[Sun Oct 15 11:12:00 39] root@rocky9:/ #mkdir -pv
/Storage/test/dir1/dir2/dir3
mkdir: created directory '/Storage/test/dir1'
```

```
mkdir: created directory '/Storage/test/dir1/dir2'
mkdir: created directory '/Storage/test/dir1/dir2/dir3'

# -m mod 指定创建文件的权限
[Sun Oct 15 11:19:39 45] root@rocky9:demo #mkdir -m 644 demo2
[Sun Oct 15 11:20:12 46] root@rocky9:demo #ll
total 0
drwxr-xr-x. 2 root root  6 Oct 15 11:19 demo1
drw-r--r--. 2 root root  6 Oct 15 11:20 demo2
drwxr-xr-x. 3 root root 18 Oct 15 10:43 dir1
```

- 删除目录
  - 命令：`rmdir`
  - 注意：用于删除空目录，此命令要删除的目录内，不能有文件存在

```
# 删除单一目录，注意：删除目录内不能有文件
rmdir <dirctory_name>

# 同时删除同级多个目录，每个目录用空格隔开
rmdir dir1 dir2 dir3

# -p 删除多级目录
rmdir -p dir1/dir2/dir3  # 同时删除dir1及其子目录dir2,dir3
```

管理文件的相关命令

- 查看文件列表
  - 命令：`ls`

```
语法格式：ls [OPTION]... [FILE]...

# -a 显示包含隐藏文件在内的所有内容
ls -a

# -i 显示文件索引节点(inode)
ls -i
[Sun Oct 15 11:39:16 65] root@rocky9:test #ls -i
136601235 baidu.html  137507906 ps_demo.txt  136601225 rename.txt  136601224
robots.txt

# -l 以长格式显示目录下内容列表
# 长格式输出信息：文件名、文件类型、权限、硬链接数、所有者、组、文件大小、修改时间
ls -l
[Sun Oct 15 11:39:28 67] root@rocky9:test #ls -l
total 16
-rw-r--r--. 1 root root 2381 Sep 27 12:03 baidu.html
-rw-r--r--. 1 root root  270 Oct 13 20:48 ps_demo.txt
-rw-r--r--. 1 root root 2814 Jan  3  2020 rename.txt
-rw-r--r--. 1 root root 2814 Jan  3  2020 robots.txt

# 用文件目录的更改时间排序
ls -t
```

```
[Sun Oct 15 11:45:06 73] root@rocky9:test #ls -tl
total 16
-rw-r--r--. 1 root root  270 Oct 13 20:48 ps_demo.txt
-rw-r--r--. 1 root root 2381 Sep 27 12:03 baidu.html
-rw-r--r--. 1 root root 2814 Jan  3  2020 rename.txt
-rw-r--r--. 1 root root 2814 Jan  3  2020 robots.txt

# 按文件大小，从大到小排序
ls -S
mystical@mystical 0101 #ll-Sh
total 60K
-rwxrwxr-x 1 mystical mystical  17K Jan  1 23:05 a.out
-rw-rw-r-- 1 mystical mystical 1.6K Jan  1 22:29 7.struct.c
-rw-rw-r-- 1 mystical mystical  861 Jan  1 21:12 6.ifdef.c
-rw-rw-r-- 1 mystical mystical  536 Jan  1 23:05 8.union.c
-rw-rw-r-- 1 mystical mystical  529 Jan  1 14:49 2.array.c
-rw-rw-r-- 1 mystical mystical  521 Jan  1 10:47 1.demo.c
-rw-rw-r-- 1 mystical mystical  445 Jan  1 16:45 3.string.c
-rw-rw-r-- 1 mystical mystical  404 Jan  1 17:14 4.pointer.c
-rw-rw-r-- 1 mystical mystical  402 Jan  1 20:53 5.ifdef.c
-rw-rw-r-- 1 mystical mystical  375 Jan  1 14:55 3.address.c
-rw-rw-r-- 1 mystical mystical   44 Jan  2 15:00 website.txt

# ls后面支持通配符过滤，不加单引号
[Sun Oct 15 11:49:26 80] root@rocky9:test #ls -l *.txt
-rw-r--r--. 1 root root  270 Oct 13 20:48 ps_demo.txt
-rw-r--r--. 1 root root 2814 Jan  3  2020 rename.txt
-rw-r--r--. 1 root root 2814 Jan  3  2020 robots.txt

ls -l <file>
#如果file是目录，则直接查询该目录下的内容，要查询目录使用
ls -dl <file>
# 如果file是普通文件，则正常查看list
```

- 文件的时间属性
  - atime: 记录最后一次的访问时间
    - atime的更新策略：连续在24小时内访问读取atime,24小时内不会更新atime；但在更改文件内容的时候会顺便更新atime
  - mtime: 记录最后一次文件数据部分的修改时间
  - ctime: 记录最后一次文件元数据的修改时间
  - 注意：mtime的改变一定会引起ctime的改变
  - 对于目录这种特殊文件
    - 其目录文件的数据部分(data block)存放的就是目录中的文件名等信息。所以在目录中创建，删除文件会改变目录的mtime，而目录的mtime的改变一定会引起ctime的改变，但其文件内容的改变，并不会引起目录mtime和ctime的改变
    - 当你访问一个目录（例如列出其内容）时，目录的 atime（访问时间）会被更新。如果你修改了目录中的一个文件，那么在大多数文件系统配置下，为了访问并修改该文件，你首先需要“访问”该目录，从而导致目录的 atime 被更新。所以修改一个目录下的文件，那么这个目录的atime通常情况下，是会更新的。但是...

        - 出于性能原因，一些现代的文件系统或挂载选项可能会默认禁用 atime 的更新。这种设置被称为noatime，它可以减少磁盘I/O，特别是在频繁读取文件但不经常修改它们的系统上。因此，如果文件系统是以 noatime 选项挂载的，那么访问文件或目录不会更新其 atime。
    - ls查看文件的3个时间属性

```
# 默认显示文件的mtime
ls -l

# 显示文件的ctime
ls -l --time=ctime

# 显示文件的atime
ls -l --time=atime
```

    - 关于atime的挂载选项
        - 'noatime'
            - 访问文件/目录不会更新atime
        - 'relatime'
            - 满足两个条件之一才更新atime
                - 文件的atime时间超过一天
                - 文件的mtime比atime更晚
- 查看文件属性信息
    - 命令：`stat`
    - 作用：用于显示文件的详细属性

```
语法格式：stat [文件或目录]

# 查看文件属性
[Sun Oct 15 14:35:09 93] root@rocky9:test #stat rename.txt
File: rename.txt
Size: 2814          Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 136601225   Links: 1
Access: (0664/-rw-rw-r--)  Uid: (    0/    root)   Gid: (    0/    root)
Context: unconfined_u:object_r:default_t:s0
Access: 2023-10-13 20:36:29.807941125 +0800
Modify: 2020-01-03 16:33:48.000000000 +0800
Change: 2023-10-15 14:34:30.473235676 +0800
Birth: 2023-09-27 11:57:14.168869373 +0800
```

    - 命令：`file`
    - 作用：使用`file`辨识文件的类型

```
语法格式：file [OPTION] file_name

# -i 查看文件的MIME类型
[Sun Oct 15 14:46:28 102] root@rocky9:test #file -i test.log
test.log: text/plain; charset=us-ascii
```

```
# -b 省略文件名称，直接打印结果
mystical@ubuntu2204:~/C_coding/2024/jan/0128$ file 1.for.c
1.for.c: C source, Unicode text, UTF-8 text

mystical@ubuntu2204:~/C_coding/2024/jan/0128$ file -b 1.for.c
C source, Unicode text, UTF-8 text

# -f 从一个文件中，获取数据进行处理
mystical@ubuntu2204:~/test$ file -f studyvim.txt
/bin:        symbolic link to usr/bin
/etc/passwd: ASCII text
/home/:      directory
```

- windows与unix格式文本之间的相互转换
  - Windows和Unix文本差异：
    - Windows每行末尾是回车符加换行符
    - Unix的每行末尾只有换行符结束
  - 相互转换需要使用dos2unix

```
sudo apt install dos2unix
# Windows文本格式转Unix
dos2unix test.txt

# Unix文本格式转Windows
unix2dos test.txt
```

- 创建或刷新文件
  - 命令：`touch`

```
# 如果文件存在则刷新时间，如果不存在则创建空文件

touch -a    # 改变atime, ctime
touch -m    # 改变mtime, ctime
touch -h    # 刷新链接文件本身，默认刷新目标文件
touch -c    # 只刷新已存在的文件，如果文件不存在，也不会创建文件
touch --time=STRING  # 修改指定时间，如：--time=atime
touch -r    # 使用某个文件的修改时间作为当前文件的修改时间
# 改变atime和mtime并刷新ctime
touch -t    # 修改atime,mtime到指定日期时间
# 比如01020304，指2024-01-02 03:04:00
# 比如0102030405， 指2001-02-03 04:05:00

# 示例
touch `date +%F-%T`.txt
2024-01-30-18:08:58.txt
```

- 复制文件
  - 命令：`cp`

```
语法格式：cp [OPTION] SOURCE DEST

# -b 覆盖已存在的目标前先对其做备份，后缀为~
```

```
[Sun Oct 15 15:03:16 108] root@rocky9:test #cp -b newtest.txt test.log
cp: overwrite 'test.log'? y
[Sun Oct 15 15:03:32 109] root@rocky9:test #ls
baidu.html  dir1  dir2  dir3  newtest.txt  ps_demo.txt  rename.txt
robots.txt  test.log  test.log~

# -S 指定备份文件的后缀名
[Sun Oct 15 15:10:53 123] root@rocky9:test #cp -S .bak dir1/cptext.txt
cptext.txt
cp: overwrite 'cptext.txt'? y
[Sun Oct 15 15:11:10 124] root@rocky9:test #tree .
.
├── baidu.html
├── cptext.txt
├── cptext.txt.bak
├── dir1
│   └── cptext.txt
├── dir2
├── dir3
├── dir_cp
│   └── cptext.txt
├── newtest.txt
├── ps_demo.txt
├── rename.txt
├── robots.txt
├── test.log
└── test.log~

4 directories, 11 files

# -i 覆盖前会先询问用户（推荐使用）
cp -i file cp_file

# -r 递归处理，将目录及其中的为文件一同复制
cp -r dir cp_dir

# -a 复制特殊文件，使用-a
cp -a /dev/zero  /home/mystical
```

- 移动及重命名文件
  - 命令：`mv`
    - 语法：`mv 目标文件 目标路径`
    - 语法2：`mv -t 目标路径 目标文件`
    - 语法3：`mv -bi 目标文件 目标路径`
      - i: 如果会覆盖文件则提示
      - b: 覆盖文件时会备份被覆盖的文件
  - 命令：`rename`

```
关于批量创建和批量修改文件名

# 批量创建文件与批量重命名
# rename <要改的字段> <改之后的字段> <使用通配符表示改的程度>
```

```
[Sun Oct 15 15:34:07 129] root@rocky9:py_test #touch pydemo{1..9}.txt
[Sun Oct 15 15:34:35 130] root@rocky9:py_test #ls
pydemo1.txt  pydemo2.txt  pydemo3.txt  pydemo4.txt  pydemo5.txt  pydemo6.txt
pydemo7.txt  pydemo8.txt  pydemo9.txt
[Sun Oct 15 15:34:38 131] root@rocky9:py_test #rename .txt .py *.txt
[Sun Oct 15 15:35:23 132] root@rocky9:py_test #ls
pydemo1.py  pydemo2.py  pydemo3.py  pydemo4.py  pydemo5.py  pydemo6.py
pydemo7.py  pydemo8.py  pydemo9.py
[Sun Oct 15 15:35:25 133] root@rocky9:py_test #rename py python py*
[Sun Oct 15 15:35:43 134] root@rocky9:py_test #ls
pythondemo1.py  pythondemo2.py  pythondemo3.py  pythondemo4.py
pythondemo5.py  pythondemo6.py  pythondemo7.py  pythondemo8.py
pythondemo9.py
```

- 删除文件
  - 命令：`rm`

```
语法格式：rm [OPTION]...FILE...

# -f 强制删除文件，即在删除文件时不提示确认，并自动忽略不存在的文件
# -r 递归删除，目标是目录的话，整个目录文件全部删除
```

  - `rm`是危险命令，建议用以下命令替换

```
alias rm='dir=/Storage/backup/data`date +%F%T`;mkdir $dir;mv -t $dir'
# 将所有要删除的文件，移动到创建的垃圾箱目录中
```

文件元数据和节点表结构

- 作用：df 命令用于显示文件系统的磁盘空间使用情况
- 查看不同分区的节点编号使用情况
  - 命令：df -i

```
生产案例1：提示空间满NO space left on device，但df可以看到空间很多，为什么

答：
节点编号不足，一个文件能被创建需要同时满足两个前提
足够的空间，以及该文件系统下还有剩余的节点编号

生产案例2：为什么cp /dev/zero /boot/test.img会把/boot的空间撑满

答：
1./dev/zero 是一个特殊的设备文件，它可以生成无限的零字节。当你尝试从它读取数据时，它会持续不断地返回零字节。

2.cp 命令的作用是复制文件或目录。在这种情况下，它从 /dev/zero 复制数据并尝试写入 /boot/test.img。

3.因为 /dev/zero 提供了无限的零字节，cp 会持续写入数据到 /boot/test.img，直到 /boot 分区没有更多的空间可用。

生产案例3：当test.img被访问时，管理员在主服务器删除test.img后，为什么，空间依然是满的
```

答：
因为当一个文件被使用时，在另一侧删除该文件，该空间并不会被立即释放，只有当这个文件不被使用时，才会释放这个空间

解决方法：
`cat /def/null > /boot/test.img; rm -rf /boot/test.img`
把文件清空后删除即可、
`echo -n '' > /boot/test.img` 结果和上述`cat /def/null...`相同

硬链接与软链接

- 硬链接：
  - 概述：本质上是多个文件名共用一个inode
  - 命令：`ln a.txt aa.txt`
  - 注意：
    - 因为本质是共用一个inode，所以不能跨分区创建硬链接，因为不同分区有独立的inode表
    - 同理，为了防止inode循环利用，所以目录也不能创建硬链接，但是在创建目录及其子目录的时候，系统会自动创建.和..这种目录的硬链接
    - 硬链接数本质上是inode计数器的值
- 软链接：
  - 概述：也叫符号链接，软链接的本质是创建了一个新文件，该文件的内容是源文件的路径，所以访问软连接文件，实质上系统访问指向了源文件
  - 命令：`ln -s 目标文件 软链接文件`
    - 注意：根据软链接的本质，软连接文件中的内容实际上是指向目标文件的路径，因此目标文件的路径如果是相对路径，那么一定是相对软链接的路径
    - 注意2：删除软链接的时候，不要加tab键补全，如果软连接文件后跟/,删除的时候，比如rm -rf /Storage/test/test/ 实际上是把原始目录中的内容一起删除

目录的本质

- 在Unix和类Unix的文件系统中，每个文件或目录都有一个与之关联的inode（索引节点）。
- 这个inode包含了关于文件的元数据，例如文件的权限、大小、修改时间、拥有者、所用的数据块的位置等，但注意，它不包含文件名。
- 目录中的数据部分，包含了文件名与inode编号的映射关系

```
fileA -> inode34
fileB -> inode57
```

- 因此目录下文件的元数据（非文件名的改变）并不会导致目录中数据部分的内容发生改变，因为：文件元数据的改变，会导致inode的数据部分发生变化，但是inode的编号/值不变。这样，文件名和对应的inode编号的映射关系就没有发生变化，所以目录数据内容无变化
- 目录的大小跟文件大小无关，仅跟目录的数据部门，即目录下文件和inode映射关系的大小有关

创建初始目录的时候，硬链接数初始为2的原因

- 在UNIX和Linux文件系统中，目录的硬链接数从2开始是有特定的原因的。当你创建一个目录（例如dir1），初始的两个硬链接代表：
  - 引用该目录的名字：这就是你所创建的目录名，如dir1。自身是一个硬链接
  - .（点）：每个目录都有一个特殊的名字.，它引用自身。当你进入dir1并列出内容时，你会看到一个.目录，它实际上指向dir1自身。
  - 当你在dir1内创建子目录时，dir1的硬链接数会增加。这是因为每个子目录都有一个名为..（双点）的特殊目录名，它指向其父目录。因此，每当你在dir1内创建一个子目录，dir1的硬链接数就会增加1。