第一部分：文件系统（EXT4）

1. 文件系统的基础概念

- **什么是文件系统**：类比为图书馆的书架结构（用来组织和管理数据）。
- 文件系统的作用：
  - 管理磁盘空间。
  - 记录文件的存储位置、权限、时间戳等信息。
  - 提供对文件的增删改查操作。

2. Linux 文件系统的核心特点

- **一切皆文件**：硬件设备、目录、进程信息等都以文件形式存在。
- **树形结构**：从根目录 `/` 开始的层级结构（目录和子目录）。
- **区分大小写**：文件名和路径区分大小写（如 `File.txt` 和 `file.txt` 是不同的）。

3. 常见的 Linux 文件系统类型

- **ext4**（最主流）、**XFS**（高性能）、**Btrfs**（实验性功能）、**ZFS**（企业级）。
- **为什么有多种文件系统**：不同场景需要不同的性能、可靠性或功能（如日志、快照、扩展性）。

---

4. 文件系统的核心组件（以 ext4/XFS 为例）

- 超级块（Superblock）——图书馆的"总目录：记录文件系统的整体信息（如大小、块数量、空闲空间）
  - **比喻**：假设整个文件系统是一个巨大的图书馆，**超级块**就像图书馆入口处的总目录。
  - 作用：
    - 总目录上写着图书馆有多少个书架（**磁盘空间总量**），
    - 每个书架的容量（**块大小**），
    - 哪些书架是空的（**空闲空间**）。
  - **如果超级块损坏**：就像总目录被撕掉了，管理员不知道书放在哪里，整个图书馆可能瘫痪。
- Inode（索引节点）——每本书的索引节点：
  - 存储文件的元数据（权限、所有者、时间戳、数据块位置）。
  - 文件内容存储在数据块（Data Blocks）中
  - **比喻**：图书馆每本书都有一个索引卡（**Inode**），存放在一个固定的抽屉里。
  - 作用：
    - 索引卡上记录书的**元数据**：书名（文件名）、作者（所有者）、出版时间（时间戳）、借阅权限（文件权限）。
    - **不记录书的内容**！而是告诉你书放在哪个书架的第几层（**数据块的位置**）。
  - 关键点：
    - 一本书可能很厚，需要占用多个书架位置（文件内容分散在多个数据块）；
    - 索引卡和书的内容是分开管理的（**Inode 不存储文件内容，只记录位置**）
  - 查看命令

```
[root@xnha yum.repos.d]# stat my.repo
  文件：my.repo
  大小：148         块：8          IO 块：4096   普通文件
设备：fd00h/64768d Inode：104307831   硬链接：1
权限：(0644/-rw-r--r--)  Uid：(    0/    root)   Gid：(    0/    root)
环境：unconfined_u:object_r:system_conf_t:s0
最近访问：2025-03-11 08:03:18.494306730 -0400
最近更改：2025-03-11 08:03:10.541138732 -0400
最近改动：2025-03-11 08:03:10.541138732 -0400
创建时间：-
[root@xnha yum.repos.d]#

ls -l 文件名          # 查看简化的元数据
```

- block(数据块)
  - **1.什么是块**：
    块是文件系统中**存储和分配磁盘空间的最小单位**。类似于图书馆的书架被划分成固定大小的格子，每个格子必须放一整本书（即使书很小，也不能和其他书共享格子）。
    - 每本书（文件）占据若干格子（block），一本厚书（大文件）占用多个格子。
    - 删除书时，格子标记为"空闲"，但书的内容仍在（直到新书覆盖）。
    - **你的 `stat` 输出中的块大小**：`IO 块：4096`（即 4KB）。
  - 2.**块的作用**：
    - 简化存储管理：文件系统按块分配空间，避免追踪每个字节的位置。
    - 提高读写效率：磁盘按块（而非字节）寻址，减少机械硬盘磁头移动次数。
  - 3.块在 `stat` 命令中的体现

    在刚才的输出中：`大小：148　　块：8　　IO 块：4096　　普通文件`
    - 大小：148 字节：文件实际内容占用的字节数。
    - **块：8**：文件占用了 **8 个块**。
    - **IO 块：4096**：每个块的大小是 **4096 字节（4KB）**。
  - **4.为什么 148 字节的文件会占用 8 个块?**
    - *计算**：`8 块 × 4096 字节/块 = 32,768 字节`（即 32KB）。
  - **5.原因：**
    - 文件系统按块分配空间，即使文件内容不足一个块，也要占用整个块。
    - **额外开销**：文件元数据（如 inode、扩展属性）可能占用额外块。
    - **极端例子**：一个 1 字节的文件也会占用 1 个块（4KB），实际浪费 99.9% 的空间!
  - **6.块大小对实际使用的影响**

    （1）空间浪费 vs 性能权衡
    - 小文件场景：
      - 若块大小=4KB，存储大量小文件（如配置文件、日志）会导致空间浪费。
      - 优化方案：格式化文件系统时指定更小的块（如 1KB），但需权衡性能。

        ```
        mkfs.ext4 -b 1024 /dev/sdX  # 指定块大小为 1KB
        ```
    - 大文件场景
      - 块越大，读写大文件（如视频、数据库）性能越高（减少寻址次数）。

        - 块大小=4KB 是 ext4/XFS 的默认平衡选择。

    （2）碎片化问题

    - 文件内容可能分散在不连续的块中（尤其是频繁修改的文件），导致读写速度下降。
    - 解决方法：
        - 定期整理碎片（ext4 较少需要，XFS 几乎无需碎片整理）。
        - 使用 `e4defrag`（ext4）或 `xfs_fsr`（XFS）工具。

- **7. 块与硬盘物理扇区的关系**
    - **物理扇区（Sector）**：硬盘的最小物理存储单元（通常 512B 或 4KB）。
    - **逻辑块（Block）**：文件系统管理的抽象单元（通常 ≥ 物理扇区大小）。
    - 关系：
        - 文件系统的 1 个逻辑块 = N 个物理扇区（例如 4KB 块 = 8 个 512B 扇区）。
        - 现代硬盘支持 4K 物理扇区（Advanced Format），与 4KB 逻辑块对齐可提升性能。

- **8.为什么你的文件显示占用 8 个块?**
    - 文件内容（148B）远小于 1 个块（4KB），但可能因以下情况占用多个块：
        1. **间接块**：如果文件内容分散存储，需要额外的块记录数据块位置。
        2. **扩展属性**：SELinux 标签或其他元数据占用额外块。

- **目录结构**——书架上的"分类标签"：目录本身是文件，记录文件名和对应 inode 的映射。
    - **比喻**：图书馆的书架上有各种标签，比如"科幻小说""历史书籍"（**目录结构**）。
    - 作用：
        - 当你走到"科幻小说"书架前，会发现一张列表，写着：
            - 《三体》→ 对应索引卡编号 123（**文件名 → Inode 映射**），
            - 《银河帝国》→ 索引卡编号 456。
    - **本质**：目录本身也是一个"文件"，它的内容就是一堆"文件名 → Inode"的对应关系。
- 日志（Journaling）：
    - **作用**：防止数据丢失：
        - 系统崩溃时，可能正在修改文件（比如保存文档）。
        - 日志会记录崩溃前"未完成的操作"（比如"正在保存文件的第 100-200 字节"）。
        - 重启后，系统检查日志，**回放未完成的操作**或**撤销未完成的操作**，保证数据一致。
    - **ext4 和 XFS 的日志差异**：
        - **ext4 的日志**：像会计把草稿本上的每一笔交易都详细记录（**记录数据和元数据**）。
        - **XFS 的日志**：会计只记录关键步骤（比如"开始转账"和"转账完成"），不记录中间细节（**仅记录元数据**），因此更快，但恢复时可能更依赖文件系统自身的逻辑。
    - **无日志的文件系统（如 ext2）**：
      直接往墙上钉画框，如果钉子断了一半，墙上可能留下一个洞（数据不一致）。
    - **有日志的文件系统（ext4/XFS）**：
      先在纸上画好钉子的位置和步骤（写日志），确认无误后再实际钉到墙上。如果中途失败，可以按图纸恢复原状。

---

5. ext4 文件系统

- **历史背景**：从 ext2 → ext3 → ext4 的演进。
- 核心特性：
  - 支持大文件（最大 16TB）和大分区（最大 1EB）。
  - 日志功能（默认开启）。
  - 延迟分配（Delayed Allocation）：提升写入性能。
  - 磁盘配额（Quota）：限制用户或组的磁盘使用量。
- **适用场景**：通用场景（个人电脑、服务器）。

---

6. XFS 文件系统

- **历史背景**：由 SGI 开发，针对高性能和大数据场景。
- 核心特性：
  - 高性能的并行 I/O：适合处理大文件和多线程操作。
  - 动态分区扩展：支持在线调整分区大小（只能扩大，不能缩小）。
  - 日志功能：更高效的日志机制（元数据日志）。
  - 支持快照（Snapshot）和冗余校验。
- **适用场景**：大型服务器、数据库、云计算（如 AWS、Azure 默认使用 XFS）。

---

7. ext4 vs XFS 对比

特性	ext4	XFS
最大文件大小	16TB	8EB
最大分区大小	1EB	8EB
日志功能	数据+元数据日志	元数据日志（更高效）
扩展性	支持在线缩小/扩大分区	仅支持在线扩大分区
适用场景	通用场景（个人电脑、中小型服务器）	高性能、大文件处理（数据库、云服务）

---

8. 实际操作演示

1. 查看文件系统类型：

   bash

   ```
   df -Th  # 显示已挂载分区的文件系统类型
   ```

2. 创建 ext4/XFS 文件系统：

   bash

```
mkfs.ext4 /dev/sdX  # 格式化为 ext4
mkfs.xfs /dev/sdX   # 格式化为 XFS
```

3. 挂载文件系统：

   bash

   ```
   mount /dev/sdX /mnt
   ```

实验1：inode与block的元数据验证

实验步骤

1. **查看文件inode信息**

   bash

   ```
   touch testfile
   stat testfile          # 观察Size（大小）、Links（链接数）、Blocks（块数量）
   ```

2. **修改文件属性并观察变化**

   bash

   ```
   chmod 600 testfile     # 修改权限
   chown root:root testfile  # 修改属主和属组
   stat testfile          # 检查元数据变化
   ```

第二部分：软链接与硬链接

理论讲解

1. 软链接（Symbolic Link）

- **基础知识点**
  - **本质**：独立文件，存储目标文件的**路径字符串**（类似Windows快捷方式）。
  - 特性：
    - 文件类型为 `l`（如 `lrwxrwxrwx`）。
    - 文件大小 = 路径字符串的字节数。
  - 查看命令：

    ```
    readlink 软链接名     # 显示软链接指向的路径
    ```

- **比喻**
  - 软链接 = 图书馆的“索引便签”
    - 便签上写着“某本书在A区3号架”，但若书被移走（源文件删除），便签失效。
- **软链接特点：** weixin.exe - 快捷方式
  - 它会创建一个新的inode编号，相当是一个独立的文件

- 它类似于windows下的快捷方式，访问的时候多了一个中转的过程， 最终访问的内容就是它链接的目标文件
- 它可以跨分区创建(因为不同分区管理的inode范围不同，而软链接的 inode不需要相同)
- 它可以对目录进行链接

---

2. 硬链接（Hard Link）

- **基础知识点**
  - **本质**：多个文件名指向**同一个inode**（共享元数据和数据块）。
  - 特性：
    - 硬链接数（Links）表示有多少文件名指向该inode。
    - 删除任意硬链接文件，inode的链接数减1，数据保留至链接数归零。
  - 查看命令：

```
ls -i 文件名          # 查看inode号（硬链接的inode相同）
```

- **比喻**
  - 硬链接 = 同一本书的多个书名标签
    - 无论通过哪个标签（文件名）找到书，内容相同。
    - 撕掉一个标签（删除文件），其他标签仍可找到书。
- **硬链接特点：**
  - 它不会新创建一个inode编号，不代表一个独立的文件，物理上指向同 一个文件
  - 它不能跨分区创建
  - 它相当于是为文件创建了一个冗余
  - 不能手工对目录进行硬链接

实验2：软硬链接

- 软链接(symbolic link)类似windows的快捷方式

```
[root@xnha yum.repos.d]# ll /etc/grub2.cfg
lrwxrwxrwx. 1 root root 22 11月 25 2019 /etc/grub2.cfg -> ../boot/grub2/grub.cfg
```

- 每当创建一个文件和目录的时候，都会为这个文件创建一个inode编号,软 链接的inode与源文件不一致

```
[root@xnha yum.repos.d]# ll -i /etc/grub2.cfg
35106744 lrwxrwxrwx. 1 root root 22 11月 25 2019 /etc/grub2.cfg ->
../boot/grub2/grub.cfg

[root@xnha yum.repos.d]# ll -i /boot/grub2/grub.cfg
307 -rw-r--r--. 1 root root 4982 3月   3 21:49 /boot/grub2/grub.cfg
```

- 使用md5sum命令查看，两个文件值一致，表示内容也是完全一致。

```
[root@xnha yum.repos.d]# md5sum /etc/grub2.cfg
ee6194e3dd636deff950f383620ec7bd  /etc/grub2.cfg

[root@xnha yum.repos.d]#  md5sum /boot/grub2/grub.cfg
ee6194e3dd636deff950f383620ec7bd  /boot/grub2/grub.cfg
```

- 例: 通过软链接实现使用vi达到使用vim的效果

```
[root@xnha yum.repos.d]# which vi
/usr/bin/vi
[root@xnha yum.repos.d]# which vim
/usr/bin/vim
[root@xnha yum.repos.d]# mv /usr/bin/vi /usr/bin/vi.bak
[root@xnha yum.repos.d]# ln -s /usr/bin/vim /usr/bin/vi
[root@xnha yum.repos.d]# ll /usr/bin/vi
lrwxrwxrwx. 1 root root 12 3月  11 09:50 /usr/bin/vi -> /usr/bin/vim
```

- 例: 目录可以手动做软链接

```
[root@xnha yum.repos.d]# cd /test
[root@xnha test]# mkdir aaa
[root@xnha test]# ln -s aaa bbb
[root@xnha test]# ll -d bbb
lrwxrwxrwx. 1 root root 3 3月  11 09:52 bbb -> aaa
```

- 硬链接(hard link)是两个inode一致的不同名文件，文件内容也是一致的。

```
[root@xnha test]# touch 1
[root@xnha test]# ln 1 2
[root@xnha test]# ll -i
总用量 0
  779643 -rw-r--r--. 2 root root 0 3月  11 09:53 1
  779643 -rw-r--r--. 2 root root 0 3月  11 09:53 2
33924679 drwxr-xr-x. 2 root root 6 3月  11 09:52 aaa
  779642 lrwxrwxrwx. 1 root root 3 3月  11 09:52 bbb -> aaa
[root@xnha test]#
```

- 不能跨分区做硬链接

```
[root@xnha test]# touch 3
[root@xnha test]# ln 3 /tmp/file1.txt
[root@xnha test]# ln 3 /boot/4
ln: 无法创建硬链接'/boot/4' => '3': 无效的跨设备链接
```

- 不能手动对目录做硬链接，但系统默认存放目录的硬链接

```
[root@xnha test]# mkdir 333
[root@xnha test]# ln 333 444
ln: 333: 不允许将硬链接指向目录
```

```
[root@xnha test]# ll -di /var/mail
100664878 lrwxrwxrwx. 1 root root 10 5月  10 2019 /var/mail -> spool/mail
```

第三部分：别名（Alias）

别名（alias）详解：Shell 的快捷命令工具

别名（alias）是 Shell 中用于**简化命令输入**的功能。你可以把它理解为“自定义命令缩写”，通过为复杂命令赋予简短的名字，大幅提升命令行操作效率。

一、基本用法

1. 定义别名

- **语法**：`alias 别名='原命令 [选项/参数]'`
- 示例：

  bash

  ```
  alias ll='ls -l'          # 输入 ll 等价于 ls -l
  alias update='sudo apt update && sudo apt upgrade'  # 组合命令
  ```

2. 使用别名

- 直接输入别名即可执行对应命令：

  bash

  ```
  ll      # 等价于 ls -l
  update  # 自动执行更新操作
  ```

3. 删除别名

- **语法**：`unalias 别名`
- 示例：

  bash

  ```
  unalias ll  # 删除 ll 别名
  ```

二、别名的生命周期

1. 临时别名（仅当前会话有效）

- **特点**：关闭终端或退出当前 Shell 后失效。
- 示例：

  bash

```
alias temp='echo "临时别名"'
```

2. 永久别名（跨会话有效）

- **配置方法**：将别名定义写入 Shell 配置文件（如 `~/.bashrc` 或 `~/.zshrc`）。
- 步骤：
  1. 编辑配置文件：

     bash

     ```
     vim ~/.bashrc  # 以 bash 为例
     ```

  2. 添加别名到文件末尾：

     bash

     ```
     alias ll='ls -l'
     ```

  3. 使配置生效：

     bash

     ```
     source ~/.bashrc  # 或重新打开终端
     ```

三、查看已定义的别名

- 查看所有别名

  ```
  alias        # 列出所有别名
  ```

- 查看特定别名

  ```
  alias ll     # 查看 ll 的定义
  ```

四、实用场景与示例

1. 简化常用命令

bash

```
alias cls='clear'                # 快速清屏
alias grep='grep --color=auto'   # 让 grep 高亮显示匹配内容
```

2. 防止误操作

bash

```
alias rm='rm -i'                 # 删除前确认
alias cp='cp -i'                 # 覆盖前确认
```

3. 快速进入常用目录

bash

```
alias cdwork='cd ~/projects/important_project'
```

4. 简化开发工具命令

bash

```
alias gs='git status'
alias gp='git push'
alias dc='docker-compose'
```

五、注意事项

1. **别名覆盖原命令**：
   - 如果定义了 `alias ls='ls -l'`，原 `ls` 命令会被覆盖。
   - 使用原命令需加反斜杠：

     bash

     ```
     \ls  # 忽略别名，执行真正的 ls
     ```

2. **别名不支持参数**：
   - 直接传递参数会导致错误：

     bash

     ```
     alias myls='ls -l'
     myls /etc  # 实际执行 ls -l /etc（正确）
     ```

   - 如需处理参数，需改用 Shell 函数：

     bash

     ```
     myls() { ls -l "$@"; }
     ```

3. **脚本中慎用别名**：
   - 默认情况下，脚本中不继承当前 Shell 的别名。
   - 如需在脚本中使用别名，需在脚本开头添加：

     bash

     ```
     shopt -s expand_aliases  # bash 中启用别名扩展
     ```

六、高级技巧

1. 组合命令与管道

bash

```
alias logs='tail -f /var/log/nginx/error.log | grep -i error'
```

2. 别名嵌套

bash

```
alias l='ls -CF'
alias la='l -A'  # 嵌套使用
```

3. 临时禁用所有别名

bash

```
\命令名  # 例如 \ls 会忽略别名
```

总结

- **别名（alias）** = Shell 命令的快捷方式，**提升效率神器。**
- **核心用途**：简化高频命令、防止误操作、组合复杂操作。
- **关键点**：临时与永久配置的区别、参数处理的限制、脚本中的行为。