↑ 목록으로 다음 단원 →

1단원. 리눅스 기초와 파일 시스템

> 대상: 전공자 · 정보처리기사 취득자 | 목적: 인프라 직무(네트워크·클라우드·시스템·보안) 기술 면접 대비 중요도: ★★★ 최빈출 · ★★ 자주 출제 · ★ 기본 개념

> **Q1.** 리눅스의 디렉토리 구조(**FHS**)를 주요 디렉토리 중심으로 설명하세요. ★★★

답안 리눅스는 **FHS(Filesystem Hierarchy Standard)**라는 표준에 따라 디렉토리 구조가 정해져 있으며, 최상위는 루트(`/`) 디렉토리입니다. `/bin` , `/sbin` 에는 시스템 부팅과 운영에 필요한 필수 실행 파일이, `/etc` 에는 시스템 설정 파일이 위치합니다. `/home` 은 사용자 계정별 홈 디렉토리, `/var` 는 로그·캐시처럼 실행 중 변화하는 가변 데이터, `/tmp` 는 임시 파일을 저장합니다. `/proc` , `/sys` 는 실제 디스크 파일이 아니라 커널이 실시간으로 제공하는 프로세스·시스템 정보를 담은 가상 파일 시스템이라는 점이 자주 출제됩니다. `/usr` 에는 사용자용 응용 프로그램과 라이브러리가, `/lib` 에는 공유 라이브러리가 위치합니다.

관련 개념 FHS, 가상 파일 시스템, /proc, /etc, 루트 파일 시스템

> **Q2.** 리눅스에서 사용하는 대표적인 파일 시스템 종류와 특징을 설명하세요. ★★

답안 **ext4**는 리눅스의 오랜 표준 파일 시스템으로, 저널링을 지원해 비정상 종료 시에도 데이터 무결성을 유지하고 안정성이 검증되어 있습니다. **XFS**는 대용량 파일과 고성능 I/O에 강점이 있어 서버 및 스토리지 환경에서 많이 사용되며, 온라인 상태에

서 파일 시스템 크기를 늘릴 수 있습니다. **Btrfs**는 스냅샷, 체크섬, 통합 볼륨 관리(RAID 유사 기능) 등 고급 기능을 제공하는 차세대 파일 시스템입니다. 이 외에 네트워크를 통해 파일을 공유하는 **NFS**, 윈도우와 호환되는 **FAT32/NTFS**도 자주 언급됩니다. 면접에서는 "왜 저널링 파일 시스템이 필요한가"로 이어지는 경우가 많으므로 저널링의 목적(비정상 종료 시 복구 시간 단축)을 함께 알아두는 것이 좋습니다.

관련 개념 저널링, ext4, XFS, Btrfs, 파일 시스템 포맷

> **Q3. inode**란 무엇이며 어떤 정보를 담고 있는지 설명하세요.
> ★★★

답안 **inode**는 리눅스 파일 시스템에서 파일 하나하나에 대한 메타데이터를 저장하는 자료구조로, 파일의 실제 데이터가 아니라 파일에 대한 "속성 정보"를 담습니다. 여기에는 파일 크기, 소유자(UID/GID), 권한, 생성·수정·접근 시간, 데이터 블록의 위치를 가리키는 포인터 등이 포함되지만, 파일명은 **inode**에 저장되지 않고 디렉토리 엔트리(파일명-**inode** 번호 매핑)에 별도로 저장됩니다. 이 때문에 같은 inode를 여러 파일명이 가리킬 수 있는데 이것이 하드링크의 원리입니다. inode 개수는 파일 시스템 생성 시 고정되므로, 디스크 용량이 남아 있어도 inode가 모두 소진되면 더 이상 파일을 생성할 수 없는 상황이 발생할 수 있습니다(`df -i`로 확인). 이 개념은 하드링크/심볼릭링크, `df -i`를 통한 장애 진단 질문으로 자연스럽게 이어집니다.

관련 개념 디렉토리 엔트리, df -i, 하드링크, 메타데이터, 데이터 블록

> **Q4.** 하드링크와 심볼릭링크(소프트링크)의 차이를 설명하세요.
> ★★★

답안 하드링크는 동일한 inode를 가리키는 또 하나의 디렉토리 엔트리로, 원본 파일과 완전히 동등하며 원본을 삭제해도 링크 카운트가 0이 되기 전까지 데이터는 유지됩니다. 반면 심볼릭링크(소프트링크)는 원본 파일의 경로 문자열을 담고 있는 별도의 파일(별도 inode)로, 원본이 삭제되면 "깨진 링크(dangling link)"가 됩니다. 하드링크는 같은 파일 시스템(파티션) 내에서만 생성 가능하고 디렉토리에는 만들 수

없지만, 심볼릭링크는 파일 시스템 경계를 넘어 생성할 수 있고 디렉토리도 링크할 수 있습니다. 명령어로는 `ln`이 하드링크, `ln -s`가 심볼릭링크를 생성합니다. 실무에서는 설정 파일 버전 전환이나 배포 디렉토리 스위칭에 심볼릭링크가 자주 사용됩니다.

관련 개념 inode, ln -s, 링크 카운트, dangling link, 파일 시스템 경계

> **Q5.** 마운트(**mount**)란 무엇이며 **/etc/fstab**의 역할을 설명하세요. ★★

답안 마운트는 물리적인 디스크 파티션이나 외부 저장장치, 네트워크 파일 시스템을 리눅스의 특정 디렉토리(마운트 포인트)에 연결해 하나의 디렉토리 트리 안에서 접근할 수 있도록 하는 과정입니다. `mount` 명령으로 수동 마운트할 수 있지만, 시스템 부팅 시 자동으로 마운트되도록 하려면 **`/etc/fstab`** 파일에 장치, 마운트 포인트, 파일 시스템 종류, 옵션, 덤프/검사 순서를 기록해 둡니다. 마운트 해제는 `umount`로 하며, 파일이 사용 중이면 "device is busy" 오류가 발생할 수 있어 `lsof`나 `fuser`로 사용 중인 프로세스를 확인하는 경우가 많습니다. 클라우드 환경에서는 EBS/블록 스토리지를 인스턴스에 연결한 뒤 파티셔닝, 포맷, 마운트, fstab 등록까지의 흐름이 실무 질문으로 자주 나옵니다.

관련 개념 /etc/fstab, umount, 마운트 포인트, lsof, 블록 디바이스

> **Q6.** 절대경로와 상대경로의 차이, 그리고 파일 디스크립터의 개념을 설명하세요. ★

답안 절대경로는 루트(`/`)부터 시작해 대상 파일까지의 전체 경로를 명시하는 방식으로 현재 작업 디렉토리와 무관하게 항상 동일한 위치를 가리킵니다. 상대경로는 현재 작업 디렉토리를 기준으로 한 경로이며 `.`(현재 디렉토리), `..`(상위 디렉토리)를 활용합니다. 스크립트를 어디서 실행하든 안정적으로 동작하게 하려면 절대경로 사용이 권장됩니다. 파일 디스크립터(**FD**)는 프로세스가 열어 놓은 파일이나 소켓 등을 가리키는 정수 형태의 식별자로, 기본적으로 0번은 표준입력(stdin), 1번은 표준

출력(stdout), 2번은 표준에러(stderr)로 예약되어 있습니다. 리다이렉션이나 파이프는 결국 이 파일 디스크립터를 재연결하는 동작입니다.

관련 개념 stdin/stdout/stderr, 현재 작업 디렉토리, 심볼릭 경로, FD 번호

Q7. `df`와 `du` 명령의 차이를 설명하세요. ★★

답안 `df` **(disk free)**는 파일 시스템 단위로 마운트된 디스크 전체의 사용량과 여유 공간을 보여주는 명령으로, "이 파티션에 얼마나 남았는가"를 확인할 때 사용합니다. `du` **(disk usage)**는 특정 디렉토리나 파일이 실제로 차지하는 용량을 계산하는 명령으로, "어떤 디렉토리가 용량을 많이 차지하는가"를 파악할 때 사용합니다. 두 값이 다르게 보이는 대표적인 상황은 삭제된 파일을 어떤 프로세스가 여전히 열고 있는 경우로, 이때 `df`상의 사용량은 줄지 않지만 `du`로는 해당 파일이 보이지 않는 불일치가 발생할 수 있습니다. 실무에서는 `du -sh */ | sort -rh`처럼 조합해 용량을 많이 차지하는 디렉토리를 빠르게 찾는 방식이 자주 쓰입니다. 이 차이는 디스크 풀(disk full) 장애 트러블슈팅에서 핵심 질문으로 자주 등장합니다.

관련 개념 디스크 풀 트러블슈팅, lsof를 통한 삭제된 열린 파일 확인, 파일 시스템 단위 vs 디렉토리 단위, sort -rh

---

Q8. 저널링 파일 시스템이 필요한 이유를 설명하세요. ★★

답안 저널링(**journaling**)은 파일 시스템에 실제 변경을 적용하기 전에 그 변경 내용을 별도의 로그(저널) 영역에 먼저 기록해 두는 기법입니다. 정전이나 시스템 크래시처럼 갑작스러운 종료가 발생해도, 재부팅 시 전체 디스크를 처음부터 검사(fsck)할 필요 없이 저널에 기록된 내용만 확인해 미완료된 트랜잭션을 롤백하거나 재적용함으로써 빠르게 일관된 상태로 복구할 수 있습니다. 저널링이 없던 초창기 파일 시스템(ext2 등)은 비정상 종료 시 전체 디스크 검사에 매우 오랜 시간이 걸렸다는 점과 대비해서 설명하면 좋습니다. 다만 모든 쓰기 작업을 저널에도 기록하므로 약간의 성능 오버헤드가 발생할 수 있어, 메타데이터만 저널링할지 데이터까지 저널링할

지 선택 가능한 모드가 존재합니다. ext4, XFS 등 현대 리눅스 파일 시스템 대부분이 저널링을 기본으로 채택하고 있습니다.

관련 개념 fsck, 크래시 복구, 트랜잭션 로그, ext2 대비, 메타데이터 저널링

> **Q9.** 리눅스 부팅 과정을 순서대로 설명하세요. ★★

답안 전원이 켜지면 먼저 하드웨어를 초기화하는 **BIOS/UEFI**가 실행되어 자체 진단(POST)을 수행한 뒤 부팅 장치를 찾습니다. 이후 부트로더(**GRUB** 등)가 로드되어 사용자가 선택한 커널 이미지와 초기 램디스크(initramfs)를 메모리에 적재합니다. 커널이 실행되면 하드웨어를 인식하고 초기 프로세스인 `init`(현대 배포판에서는 **systemd**)를 최초 프로세스(PID 1)로 실행합니다. systemd는 설정된 타겟(target, 과거 runlevel에 대응)에 따라 필요한 서비스와 데몬들을 순차적/병렬적으로 기동하며, 이 과정이 끝나면 로그인 화면이나 셸 프롬프트가 제공되어 사용자가 로그인할 수 있는 상태가 됩니다. 이 흐름은 "커널 패닉은 어느 단계에서 발생하는가", "PID 1이 왜 특별한가" 같은 꼬리질문으로 이어지기 쉽습니다.

관련 개념 GRUB, initramfs, PID 1, systemd target, 커널 패닉

> **Q10.** 리눅스에서 파일이나 디렉토리를 찾고 텍스트를 검색하는 기본 명령어를 설명하세요. ★

답안 **`find`**는 디렉토리 트리를 순회하며 이름, 크기, 수정 시간, 권한 등 다양한 조건으로 파일을 검색할 수 있는 명령으로, `-exec` 옵션을 통해 검색된 파일에 바로 다른 명령을 적용할 수 있습니다. **`grep`**은 파일 내용 중 특정 패턴(문자열/정규식)이 포함된 줄을 검색하는 명령이며, `-r`로 재귀 검색, `-i`로 대소문자 무시, `-n`으로 줄 번호 출력이 가능합니다. **`locate`**는 사전에 구축된 파일 인덱스 데이터베이스를 검색하므로 `find`보다 훨씬 빠르지만 실시간 반영이 되지 않는 단점이 있습니다. 이 외에 `which` / `whereis`로 실행 파일 위치를 확인하는 것도 자주 쓰입니다. 이 명령

어들은 파이프(|)와 결합해 로그 분석이나 자동화 스크립트에서 핵심적으로 활용됩니다.

관련 개념 find -exec, grep -r/-i/-n, locate와 updatedb, which/whereis, 파이프 활용
