

# 리눅스 어드민 101

# 소개

강사

과정

# 강사

이름: 최국현

메일: tang@linux.com, 회신은 다른 메일 주소로 드리고 있습니다. :)

사이트: tang.dustbox.kr

언제든지 질문 및 요청 환영입니다.

# 목차

리눅스 어드민

# 목차

1. 소개
2. 리눅스와 오픈소스 관계
3. 리눅스 셸 및 커널 구성
4. systemd 기반, 시스템 블록 이해
5. 자주 사용하는 기본적인 리눅스 명령어
6. 새로운 네트워크 구성 및 관리자 사용

# 목차

1. 새로운 방화벽 시스템
2. 프로세스
3. 파일 시스템 퍼미션
4. 패키지 관리 방법
5. 디스크 관리
6. 모니터링
7. 리눅스 커널과 모듈
8. 백업 도구 및 응급 복구
9. 가상머신 및 컨테이너

# 과정개요

쿠버네티스 101

# 과정개요

이 과정은 리눅스를 처음 혹은 2년차 이상 대상으로 만들어진 과정 입니다. 주요 내용은 리눅스 주요기능 및 명령어 입니다. 이를 통해서 좀 더 넓은 범위의 CLI 및 컴포넌트 활용을 할 수 있도록 합니다.

길이	설명
3일	기본적인 리눅스 기본 구성 및 기초 명령어
4일	리눅스 확장 명령어 및 블록 장치 컴포넌트



# 과정개요

리눅스의 기초 및 기본적인 명령어 및 지식을 다루는 교육입니다. 간단한 ls, cp 정도 명령어를 사용하는 사용자 혹은 시스템 운영자에게 좀 더 깊은 기초 지식을 전달하는 게 주 목적입니다.

이를 통해서 시스템 CPU/MEMORY/BLOCK DEVICE에 대한 모니터링 지식 및 전반적인 운영 도구에 대해서 접근 및 학습 합니다.

모든 교육은 랩 기반으로 되어 있기 때문에 천천히 랩을 따라오시면 됩니다.

설명이 추가적으로 필요한 경우, 언제든지 강사에게 질문 혹은 이-메일로 문의 하시면 됩니다.

# 리눅스와 오픈소스 관계

오픈소스 라이선스

기업용 배포판 라이선스 현황

# 리눅스+오픈소스=배포판

💡 Linus는 기술 중심 / RMS는 자유 철학 중심이기에, 둘의 조화가 GNU/LINUX를 탄생.

요소	설명
Linux 커널	Linus Torvalds가 개발한 운영체제의 핵심 커널. 하드웨어와 소프트웨어를 연결. 자체적으로는 OS가 아님.
GNU <u>소프트웨어</u>	리차드 스톨만(RMS)이 만든 자유 소프트웨어 유틸리티 모음. 컴파일러(gcc), 셸(bash), 라이브러리(glibc) 등 포함.
👉 관계	대부분의 리눅스 배포판은 Linux 커널 + GNU 유틸리티 조합으로 이루어짐 → 그래서 "GNU/Linux"라고도 불림.

# 오픈소스 유형?

오픈소스는 다양한 유형이 있다. 다만, 오픈소스는 서양 문화권에서 탄생하였기에 아시아 문화권에서 종종 이해가 안되는 부분이 있다.

유형	설명 및 예시
자유 소프트웨어 (Free Software)	사용, 수정, 재배포의 자유가 핵심 (GPL 등) → GNU, Emacs
오픈소스 소프트웨어 (Open Source)	접근 가능한 소스코드 + 협업 가능성 중시 → Apache, BSD
카피레프트 (Copyleft)	소스코드를 공유하되, 수정 후 배포 시에도 동일한 라이선스를 강제 → GNU GPL
퍼블릭 도메인	저작권이 소멸된 자유 공유 코드 (예: SQLite)
프리웨어/셰어웨어	실행은 무료지만 소스코드 공개는 X 또는 조건부 (예: 일부 Windows 유틸)

## 오픈소스에서 라이선스가 중요한 이유

"소스코드는 공유되지만, 이용 조건은 법적 문서로 보호된다."

이유	설명
법적 분쟁 방지	상업적 제품에 오픈소스 코드 사용 시, 라이선스 위반이 소송으로 이어질 수 있음.
재배포 조건 차이	MIT는 자유롭지만, GPL은 파생 소스 공개를 강제 → 기업 입장에선 큰 차이
오픈소스 의무 준수	LGPL과 같은 라이선스는 동적 링크에만 관여하므로 덜 엄격, 반면 GPL은 정적 링크도 포함
기업 전략과 충돌	Google, Apple, Microsoft도 라이선스 충돌 방지 위해 법무팀 상시 운영

이러한 이유로 **오픈소스는 무료라고 생각하면 안된다**. 사용하는 **코드 및 바이너리**에는 상용 프로그램과 동일하게 **라이선스는 항상 따라다닌다**.




# 라이선스

라이선스 조건은 간단하게 다음과 같다. 여기에 내용은 일부분이니, 상용 서비스 및 판매 전 꼭 확인해야 한다. 라이선스는 구축 서비스에도 같이 표시 및 표기가 되어야 한다.

라이선스	주요 특징	상업적 사용	소스 공개 의무
MIT	매우 자유로움	✓	✗ (선택사항)
Apache 2.0	특허 보호 포함	✓	✗
GPL v2/v3	복사/수정/재배포 자유, 소스 공개 필수	✓	✓
LGPL	라이브러리 전용 GPL → 연동은 자유로움	✓	⚠ (링크 방식에 따라 다름)
BSD	MIT보다 살짝 느슨한 스타일	✓	✗
MPL	파일 단위로 공개 강제	✓	✓ (변경된 파일만)

# 레드햇 SRPM라이선스

수세는 SRPM에 대해서 별도의 제약이 없지만, 레드햇은 SRPM에 대해서 다음과 같이 제약을 걸었다.

항목	설명
 전환	Red Hat은 CentOS Linux를 종료하고, CentOS Stream을 RHEL의 <i>미래 미리보기 (feature preview)</i> 로 전환 (다운스트림 → 업스트림 중간 형태)
 SRPM 제한	2023년 이후 RHEL의 공식 SRPM은 Red Hat 고객 포털 전용으로 제한, 일반 사용자는 접근 불가
 공개되는 SRPM	<ul style="list-style-type: none"><li>- Fedora: 완전 공개</li><li>- CentOS Stream: 공개 유지</li><li>- RHEL: 공개 제한 (Red Hat 고객만 사용 가능)</li></ul>

# GPL 위반인가?

## 핵심 요점 (GPL v2 기준)

1. GPL은 소스코드를 제공해야 한다고 명시하지만,
2. 그 범위는 “바이너리를 배포받은 사람에게” 소스도 같이 제공하라는 뜻이에요.
3. Red Hat은 RHEL 바이너리를 오직 유료 고객에게만 제공하고,
4. 고객에게는 SRPM도 제공하므로 GPL을 위반하지 않음.

## Red Hat 입장 요약

“우리는 GPL 조건에 따라, RHEL 바이너리를 받은 고객에게만 소스를 제공하고 있으며, 이는 라이선스 위반이 아니다.” 즉, “바이너리도 안 주는 사람에게 소스를 줄 의무는 없다.”



# GPL 위반인가?

커뮤니티는 다음과 같은 반응 전달하고 있다. 특히 FSF에서는 다음과 같은 입장을 성명서로 발표하였다.

관점	입장
Red Hat 법률팀	"GPL 준수 중이다. 의무만 이행하고, 의무 이상은 하지 않는다."
커뮤니티/AlmaLinux	"법적으로 위반은 아니지만, 오픈소스 정신에는 반한다."
FSF	공식 반응은 없지만, 철학적으로 실망했다는 입장이 강함

레드햇 SRPM를 리패키징 및 리빌드 시, 이러한 부분을 이제는 고려해야 한다. 즉, 기업에서는 SRPM를 **임의로 리빌드 및 배포**하면 저작권 문제가 발생한다.

# 레드햇 EULA

GPL vs Red Hat EULA의 충돌 구조.

항목	GPL 기준	Red Hat 정책 (EULA)
SRPM 사용	✓ 누구나 가능	⚠ 재패키징 후 RHEL로서 배포 시 문제 소지
재배포	✓ 허용됨	⚠ Red Hat 표기/브랜드 그대로면 상표권 위반
브랜드 사용	무관	✗ 상표 침해 가능 (RHEL, Red Hat 로고 등)

# EULA 위반

리-빌드 시, 위반되는 조건은 대략 다음과 같다.

조건	GPL 위반인가?	Red Hat 정책 위반 가능성?
pkgs.org에서 SRPM을 다운받아 개인 빌드	✗ 아님	✗ 아님
빌드한 RPM을 배포 (Red Hat 표기 없음)	✗ 아님	⚠ 가능성 낮음
"이건 RHEL과 동일!"이라며 배포	✗ 아님	✓ 상표 침해 위험 있음
Red Hat 패치 포함한 바이너리를 RHEL처럼 배포	✗ (코드는 자유)	✓ 법적 문제 발생 가능

# 정리

최종적으로 대표적인 커뮤니티 기업용 리눅스 배포판 두 가지는 다음과 같은 방법으로 SRPM를 획득 및 호환성을 맞추고 있다.

## Alma Linux

**변경된 정책:** 2023년 7월, AlmaLinux는 RHEL과의 1:1 바이너리 호환성을 목표로 하는 대신, ABI(Application Binary Interface) 호환성을 유지하는 방향으로 전환하였습니다.

**소스 코드 확보 방법:** AlmaLinux는 CentOS Stream, Oracle Linux, 그리고 기타 공개된 소스에서 필요한 패키지를 확보하여 빌드하고 있습니다.

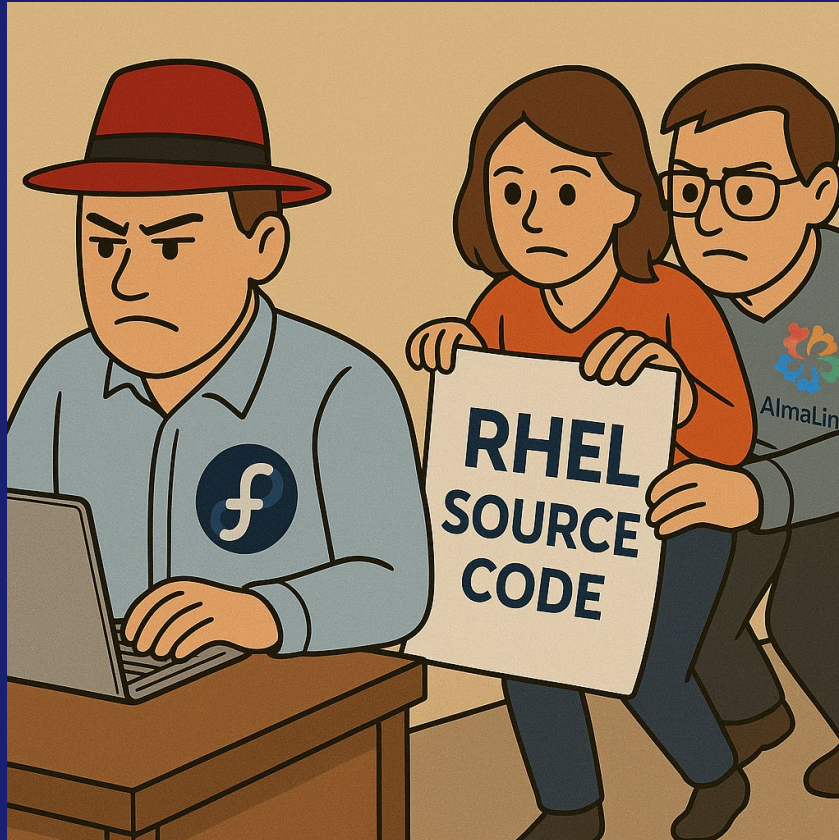
# 정리

## Rocky Linux

**호환성 유지:** Rocky Linux는 여전히 RHEL과의 1:1 바이너리 호환성을 목표로 하고 있으며, 이를 위해 다양한 소스에서 SRPM을 확보하여 빌드하고 있습니다.

**소스 코드 확보 방법:** Rocky Linux는 CentOS Stream, 공개된 RHEL SRPM, 그리고 UBI 컨테이너 이미지 등을 활용하여 필요한 소스 코드를 확보하고 있습니다.

# 정리



# 리눅스 셸 및 커널 구성

기본 개념

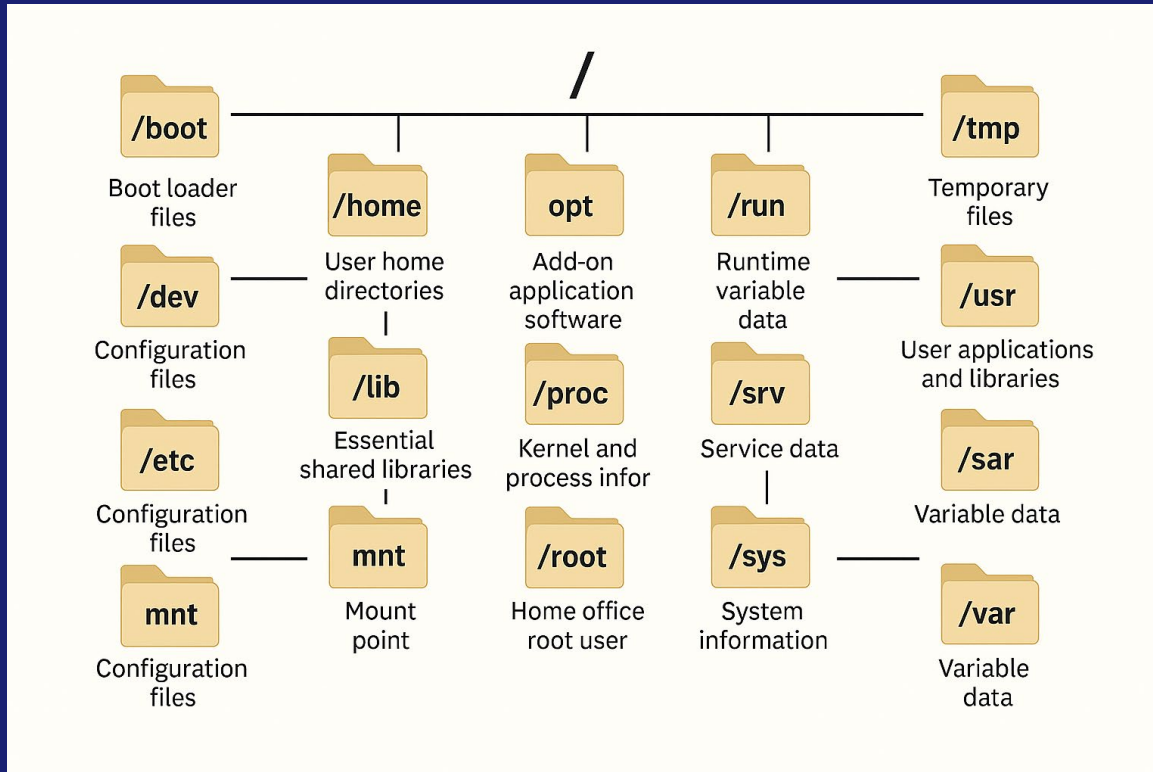
# 리눅스 표준 디렉터리

리눅스 셸 및 커널 구성



# 설명

일반적으로 리눅스 디렉터리는 루트(/) 기준으로 아래와 같이 구성이 되어 있다.



# 설명

주요 디렉터리를 정리하면 다음과 같다.

디렉터리	설명
/bin	부팅 및 모든 사용자용 필수 명령어
/sbin	시스템 관리용 명령어 (root 전용)
/etc	시스템 설정 파일
/home	일반 사용자 홈 디렉터리
/root	루트 사용자의 홈
/lib, /lib64	공유 라이브러리
/usr	사용자용 프로그램, 라이브러리
/opt	제3자 앱 설치용 공간
/tmp	임시 파일 저장소 (재부팅 시 비움)
/var	로그, 메일큐, DB 등 가변 데이터

# 설명

앞에 내용 계속 이어서...

디렉터리	설명
/dev	하드웨어 및 가상 디바이스 파일
/proc	커널 정보 및 프로세스 가상 FS
/sys	시스템/하드웨어 정보 가상 FS
/media	자동 마운트 지점
/mnt	임시 마운트 지점
/srv	FTP, 웹서비스 등 서버 데이터용
/run	시스템 부팅 중 생기는 런타임 데이터

# 설명

위의 디렉터리 중에서 몇몇 디렉터리는 실제로 존재하지 않는 가상의 특수 디렉터리가 있다.

<code>/proc</code>	커널이 가지고 있는 시스템 파라미터. 여기에 프로세스 정보도 같이 있다. 윈도우에서는 자원관리자와 동일한 기능을 한다.
<code>/sys</code>	커널에서 사용하는 모듈 및 설정 내용. 사용자는 여기서 확인 및 수정이 가능하다.

예를 들어서 CPU 및 MEM 정보를 확인하기 위해서 다음과 같이 접근이 가능하다.

```
# cat /proc/cpuinfo
# cat /proc/meminfo
```

블록 장치 및 CPU 클럭(주파수)상태를 명령어가 아닌 상태 파일에 접근해서 확인이 가능하다.

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

# 설명

/sys 디렉터리는 다양한 자원이 분리가 되어 있다. 분리 및 분류 기준은 다음과 같다.

디렉터리 경로	자원 유형	설명
/sys/class/	논리 장치(Class)	블록 장치, 네트워크, 전원, TTY 등 디바이스 클래스별 디렉터리
/sys/block/	블록 디바이스	디스크, SSD, USB, loopback 등 블록 장치 정보
/sys/bus/	버스 종류	PCI, USB, I2C, SPI 등 하드웨어 버스 종류별 장치 연결
/sys/dev/	장치 파일 매핑	major/minor 번호 기준 장치 매핑 정보
/sys/devices/	실제 물리 장치 트리	물리적 장치 경로, 버스 아래 실제 연결된 하드웨어 (루트 노드 /sys/devices/system)
/sys/firmware/	펌웨어/ACPI 정보	BIOS, EFI, ACPI 등 시스템 초기화 정보
/sys/fs/	파일 시스템	cgroup, bpf, fuse, overlay 등 가상/논리 파일 시스템
/sys/kernel/	커널 내부 상태	커널 파라미터, kexec, debug, security 정보 등
/sys/module/	커널 모듈	로드된 커널 모듈 목록 및 매개변수 정보
/sys/power/	전원 관리	시스템 절전, suspend/resume 상태 제어

# FSH VS LSB의 차이점

리눅스 셸 및 커널 구성

# 설명

앞에서 이야기 하였지만, FHS는 기본적으로 디렉터리 구조를 따른다. 이 구조를 따르지 않는 리눅스 배포판은 호환성이 많이 내려간다.

의외로 오픈소스 배포판 중, 우분투가 대표적으로 비표준적인 구조를 많이 사용하고 있으며, 반대로 기업용 배포판 수세/레드햇은 FHS를 매우 잘 따른다.

# 설명

모든 리눅스 배포판은 기본적으로 다음과 같은 두 가지 사양을 따라야 한다.

1. FHS
2. LSB

**FHS는 파일 시스템 구조(Filesystem Hierarchy Structure)의 약자.**

리눅스 커널 기반의 모든 리눅스 배포판은 표준적인 디렉터리 구조를 유지 및 사용 할 수 있도록 제안 및 권고 한다.  
현재 대다수 배포판은 FHS기반으로 디렉터리를 구성하나, 몇몇 배포판은 FHS를 따르지 않는다.



# 설명

FHS 기준을 따르는 배포판은 다음과 같다.

배포판	FHS 준수 수준	특징 요약
RHEL	매우 엄격히 준수	기업용으로 안정성과 표준 준수가 핵심. FHS를 강하게 따름.
CentOS	(RHEL과 동일)	RHEL의 다운스트림. 거의 완전히 동일한 구조.
Rocky	(RHEL과 동일)	RHEL 대체로 개발, FHS 완전 준수.
SUSE (SLES)	엄격하게 준수	기업용 서버 중심, 전통적으로 FHS를 충실히 따름.
Debian	상당히 준수	자유 소프트웨어 철학 기반이나 FHS 준수에도 충실.
Ubuntu	⚠ 부분적으로 다름	Debian 기반이지만 Snap, Flatpak, /snap 경로, /srv 활용 방식 등 일부 탈표준 경향 있음.

# 설명

LSB는 **Linux Standard Base**의 약자. 모든 배포판에서 사용하는 기본 프로그램 및 라이브러리에 대해서 명시 및 사양을 정의하고 있다.

기본적으로 모든 기업용 배포판은 LSB를 따라야 한다. LSB를 따르는 배포판은 다음과 같다.

# 설명

FHS 기준을 따르는 배포판은 다음과 같다.

배포판	LSB 준수	freedesktop.org	설명
RHEL	부분적	매우 적극적	데스크탑 환경은 GNOME 위주로 Freedesktop 완전 지원.
CentOS	부분적		RHEL과 동일. GNOME 기반 구성.
Rocky	부분적		CentOS와 동일.
SUSE (SLES)	공식 준수	매우 적극적	KDE, GNOME 모두 freedesktop 표준 강하게 반영.
Debian	⚠️ 부분적	✅ 적극적	시스템 구성은 보수적이지만 Freedesktop 기반은 잘 따름.
Ubuntu	❌ 비준수	✅ 매우 적극적	GNOME 기반 데스크탑이며 freedesktop 표준은 강하게 준수.

# 왜 리눅스 배포판은 디렉터리가 다른가?

리눅스 셸 및 커널 구성

# 설명

FHS 준수 지수는 다음과 같다. 데이터의 기준은 강사의 데이터 수집 및 커뮤니티 평가 기준이다.

배포판	FHS 준수 점수	주요 특징 및 근거
RHEL	95점	공식적으로 FHS를 준수하며, /usr를 읽기 전용으로 마운트할 수 있도록 설계됨. 단, 일부 디렉터리는 확장 가능성으로 인해 유연하게 처리됨. <a href="#">레드햇 문서</a>
SUSE	95점	FHS를 철저히 준수하며, /opt 디렉터리 사용 등에서 표준을 충실히 따름. <a href="#">novell.com</a>
Arch	90점	기본적으로 FHS를 잘 따르며, /etc, /usr, /var 등의 구조를 유지함. 단, 사용자의 선택에 따라 유연성이 높음.

# 설명

위의 내용 계속 이어서...

배포판	FHS 준수 점수	주요 특징 및 근거
Gentoo	85점	Portage 시스템으로 인해 /etc/portage 등 비표준 디렉터리를 사용하지만, 전반적으로 FHS를 존중함.
Ubuntu	75점	Snap 등의 도입으로 /snap, /var/snap 등 비표준 디렉터리를 사용하며, Netplan 등으로 기존 구조에서 벗어남.
Debian	90점	전통적으로 FHS를 잘 따르며, /usr, /etc 등의 구조를 유지함. 다만, 일부 최신 기술 도입으로 유연성이 있음.

100점: FHS를 완벽히 준수하며, 디렉터리 구조에 일관성이 있음.

90점 이상: 대부분 FHS를 따르며, 일부 유연성을 허용함.

80점 이상: FHS를 기본으로 하지만, 특정 목적에 따라 구조적 변경이 있음.

70점 이상: FHS에서 벗어난 구조를 도입하였으며, 표준과의 차이가 큼.

# 배포판 기초

또한 모든 리눅스 배포판은 각기 철학으로 인하여 기초가 많이 달라지기도 한다. 아래는 배포판 커뮤니티 혹은 회사에서 가져가는 기초다.

1. Debian은 "자유 소프트웨어 철학" 기반 → 공동체 중심, LSB 따르되 유연함.
2. Arch Linux는 "사용자 중심의 단순함" → 규칙보다 자유와 선택 우선.
3. Gentoo는 최적화 및 사용자 제어의 극대화.

# 레드햇 리눅스

기업용으로 많이 사용하는 수세/레드햇/우분투는 다른 기초를 가지고 있다.

## Red Hat Enterprise Linux (RHEL)

### 철학

- "안정성, 호환성, 그리고 지원 가능한 오픈소스."
- 오픈소스를 기반으로 하지만, 기업용 수준의 신뢰성과 지원을 최우선으로 함.
- 개발보다는 운영 환경에 최적화된 구조 → 느리지만 신중한 기술 도입
- 업스트림(Fedora) → 업스트림(CentOS-Stream) → 다운스트림(RHEL) 구조를 통해 충분히 검증된 기능만 채택

### 문화

- 엔터프라이즈를 위한 장기적 유지보수(LTS) 중심
- 보안 업데이트와 검증된 패키지를 중시
- 커뮤니티보단 계약 기반 고객(서브스크립션) 중심의 B2B 철학
- SELinux, KVM, Podman 등 핵심 기술을 주도적으로 개발



# 수세 리눅스

## ■ SUSE Linux Enterprise (SLES)

### 🌟 철학

- "유연성과 확장성, 오픈소스 혁신의 실용화."
- 모듈화된 구조와 다양한 환경에 대한 유연한 대응력
- 오픈소스 철학을 유지하면서도, 상용 기업 솔루션에 최적화
- 자동화, 고가용성(HA), SAP 등 특정 산업군에 강함

### 🎯 문화

- YaST, AutoYaST로 시스템 관리 편의성 극대화
- Btrfs, Snapper, Transactional Update 등 실험적 기술 도입 선도
- 커뮤니티와 상업 고객 사이의 균형을 중시 (openSUSE ↔ SLES)
- 고객 맞춤형 유연한 지원 → 제조, 금융, 정부 분야에 강함

# 우분투 리눅스

## ■ Ubuntu (Canonical)

### 🌟 철학

- "리눅스를 모두에게. 접근 가능한 미래형 플랫폼."
- 사용자 중심의 접근성과 현대적 감각 강조
- 리눅스를 더 많은 사람에게 보급하기 위해 데스크탑에 집중
- Snap, cloud-init, MAAS, Juju 등 클라우드 및 IoT 플랫폼에도 적극 진출

### 🎯 문화

- 빠른 릴리즈 주기(6개월), LTS 정책(2년마다) 등 혁신 우선
- 커뮤니티 활발 + 클라우드 중심으로 기업 시장 확대
- \*\*독립 생태계 구축(Snap Store, Netplan 등)\*\*을 통해 자립 추구
- "우분투(Ubuntu)"는 아프리카어로 "나는 네가 있기에 존재한다" → 공동체 문화 기반

# 정리

배포판의 기초 및 목적에 대해서 정리하면 다음과 같다.

항목	RHEL	SUSE	Ubuntu
중심 가치	안정성 & 신뢰성	유연성 & 산업 통합	접근성 & 사용자 중심 혁신
주요 타겟	엔터프라이즈 & 금융, 제조업	고성능 산업, SAP, 정부 기관	데스크탑 사용자, 클라우드, IoT
기술 성향	보수적, 신중한 채택	실험적 기능 선도	빠른 릴리즈와 자사 도구 중심
커뮤니티 관계	Fedora(업스트림) 기반 커뮤니티 CentOS-Stream Fedora에서 분기	openSUSE와 공유	Ubuntu Community, Launch pad
전략 방향	계약 기반 시장 점유율 확대	산업특화 최적화	범용성 & 플랫폼화 전략

# systemd 기반, 시스템 블록 이해

리눅스 기초

# systemd 설명

systemd

# 설명

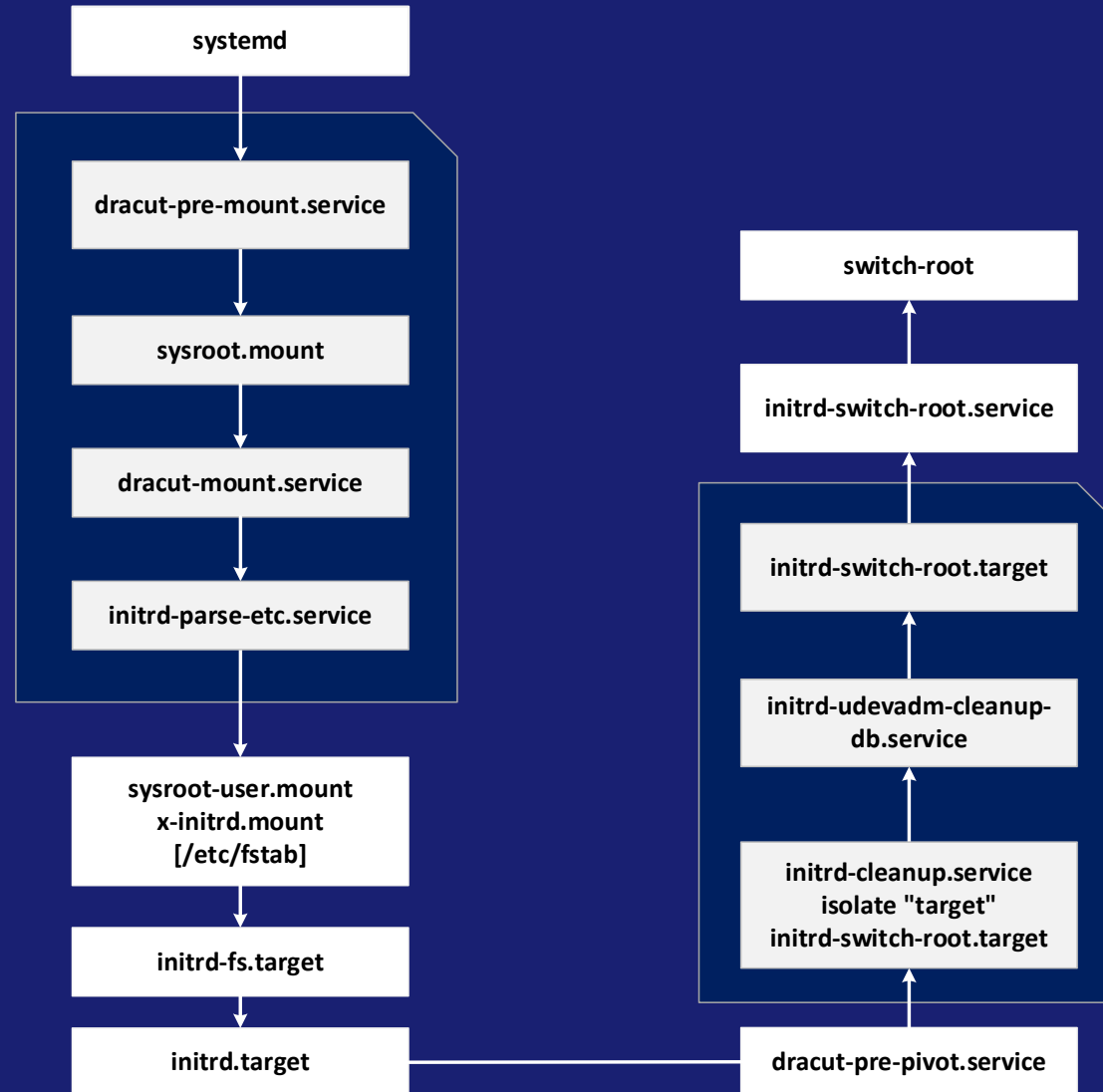
## PID 1 - systemd

- 리눅스 커널이 사용자 공간에서 가장 먼저 실행하는 프로세스
- 전체 systemd의 **중추**로, init 프로세스 역할을 수행

systemd는 윈도우 서버와 비슷하게 부트-업 한다.

**부트로더(grub2)**에서 커널 이미지를 불러온 후, 기본적인 하드웨어 초기화를 완료한다. 완료가 되면, 다시 O/S 디스크로 **피봇팅(Pivoting)** 후 정상 기동을 시작한다.

# 설명



# 릴리즈 기록

**systemd**는 다음과 같이 릴리즈 되었다.

버전	릴리즈 시기	주요 변화 요약
systemd 1~10	2010년	최초 릴리즈. sysvinit을 대체. 병렬 부팅 및 socket activation 도입.
systemd 30	2011년	journal 도입 (systemd-journald), 로깅 시스템 내장 시작.
systemd 38	2012년	tmpfiles.d, hostnamectl, timedatectl 도입. 관리 범위 확장 시작.
systemd 44	2012년	logind 도입 - ConsoleKit 대체 시작. 사용자 세션 관리 통합.
systemd 198	2013년	networkd 도입. DHCP 클라이언트와 네트워크 관리 기능 시작.
systemd 208	2013년	machinectl 도입. 컨테이너 및 VM 통합 관리 개념 포함.
systemd 219	2015년	RHEL 7, CentOS 7에 포함됨. resolved, timesyncd 안정화.
systemd 231	2016년	systemd-run, user instance 기능 강화. nspawn 기능 개선.
systemd 239	2018년	Unified cgroup 지원 시작 (cgroups v2).
systemd 245	2020년	homed 도입, 사용자의 홈 디렉터리 관리 개념 추가.



# 릴리즈 기록

위에 내용 계속 이어서...

버전	릴리스 시기	주요 변화 요약
systemd 247	2020년	oomd 기능 포함 - out-of-memory 상황 관리 도구.
systemd 249	2021년	systemd-boot 강화, EFI 부팅 구조 개선.
systemd 251	2022년	systemd-sysext, cred.d 도입, 이식성 개선
systemd 253	2023년	soft reboot, sysupdate 도입. 리눅스 자체의 업데이트 구조 실험 시작.
systemd 255	2024년	systemd-vmspawn, idmapped mounts 실험적 도입, VM 지원 강화.

# 부팅 설명

부팅 흐름을 정리하면 다음과 같다.

단계	설명
1. BIOS/UEFI	하드웨어 초기화 및 부트로더 호출
2. Bootloader (GRUB 등)	커널 이미지와 initramfs(램 디스크)를 메모리에 로딩
3. initramfs	최소 루트 파일시스템. 드라이버 로딩, 디스크 감지 등 수행
4. Pivot Root (switch_root)	실제 디스크의 루트 파티션으로 전환 (pivot)
5. PID 1 실행 (systemd)	systemd가 부팅을 이어받아 유닛 관리 및 OS 초기화
6. 사용자 쉘 / GUI 로그인	부팅 완료

# 유닛 설명

systemd는 이전 system-v와 다르게 모든 자원은 INI기반으로 구성이 되어 있다.

유닛 종류	확장자	설명
Service	.service	데몬, 백그라운드 프로세스 (예: nginx, sshd)
Socket	.socket	소켓 활성화. 서비스에 연결 요청 시 자동 실행
Target	.target	여러 유닛을 그룹화한 목적지(runlevel 대체)
Device	.device	커널 디바이스와 연결된 유닛 (udev 기반)
Mount	.mount	파일 시스템 마운트 관리 (/etc/fstab 대체 가능)
Automount	.automount	접근 시 자동 마운트 트리거
Timer	.timer	예약된 시간에 유닛 실행 (cron 대체)
Path	.path	파일/디렉터리 변경 감지 후 서비스 트리거

# 유닛 설명

앞에 내용 계속 이어서...

유닛 종류	확장자	설명
Slice	.slice	cgroups를 이용한 리소스 분할 (메모리/CPU 제한 등)
Scope	.scope	외부 프로세스(systemd 외) 제어 목적 유닛
Swap	.swap	스왑 공간 관리
Snapshot	.snapshot	현재 상태 저장용 (보통 자동 생성)
BusName	.busname	D-Bus 이름 소유권 관리용
Link	.link	네트워크 장치 이름 등의 udev 링크 규칙
Network	.network	systemd-networkd를 통한 네트워크 구성
NetDev	.netdev	가상 네트워크 장치 정의 (bridge, vlan 등)

# 자원 위치

자원 디렉터리는 다음과 같이 제공한다.

디렉터리 경로	설명
<code>/etc/systemd/system/</code>	사용자가 정의한 유닛 설정 (우선순위 ↑, 재부팅 유지)
<code>/run/systemd/system/</code>	런타임 생성 유닛 파일 (일시적)
<code>/usr/lib/systemd/system/</code>	패키지에서 제공하는 기본 유닛
<code>/lib/systemd/system/</code>	(일부 배포판에서 <code>/usr/lib</code> 와 동일하게 사용)

# 자원 위치

자원 디렉터리는 다음과 같이 제공한다.

디렉터리 경로	설명
<code>/etc/systemd/system/</code>	사용자가 정의한 유닛 설정 (우선순위 ↑, 재부팅 유지)
<code>/run/systemd/system/</code>	런타임 생성 유닛 파일 (일시적)
<code>/usr/lib/systemd/system/</code>	패키지에서 제공하는 기본 유닛
<code>/lib/systemd/system/</code>	(일부 배포판에서 <code>/usr/lib</code> 와 동일하게 사용)

# 유닛 관리

systemd

# 설명

앞에서 잠깐 명령어를 이야기 하였지만, systemd는 이전에 기능별로 따로 사용하던 영역을 하나로 통합 하였다. 이러한 이유로 관리하기 위해서는 systemd에서 제공하는 명령어와 친숙해져야 한다.

systemd에서 어떠한 명령어를 제공하는지 살펴 보도록 한다.



# 기본 명령어

systemd에서 제공하는 기본 명령어는 다음과 같다. systemd버전 및 확장 기능 설치에 따라서 지원하는 명령어는 달라진다.

명령어	설명
<code>systemctl start &lt;unit&gt;</code>	유닛 즉시 시작
<code>systemctl stop &lt;unit&gt;</code>	유닛 즉시 정지
<code>systemctl restart &lt;unit&gt;</code>	유닛 재시작
<code>systemctl reload &lt;unit&gt;</code>	설정만 다시 불러오기 (서비스 종료 없이)
<code>systemctl enable &lt;unit&gt;</code>	부팅 시 자동 시작 등록
<code>systemctl disable &lt;unit&gt;</code>	부팅 시 자동 시작 해제

# 기본 명령어

앞에 내용 계속 이어서...

명령어	설명
<code>systemctl status &lt;unit&gt;</code>	유닛 상태 확인
<code>systemctl is-active &lt;unit&gt;</code>	유닛이 현재 실행 중인지 확인
<code>systemctl list-units</code>	현재 로드된 모든 유닛 보기
<code>systemctl daemon-reexec</code>	systemd 자체를 재실행 (PID 1 유지)
<code>systemctl daemon-reload</code>	유닛 파일 변경 후 다시 읽기

# 서비스 상태 확인

서비스 상태 확인 시 다음과 같이 조회가 가능하다.

```
# systemctl status sshd  
# systemctl is-active sshd  
# systemctl is-enabled sshd
```

# 서비스 상태 확인

systemd는 여러가지 자원들을 INI형태로 제공한다. 제공하는 자원 목록을 확인하기 위해서 아래와 같이 조회 및 확인이 가능하다.

```
# systemctl -t service
```

만약, 유닛 파일을 확인하고 싶으면, 다음과 같은 명령어로 조회가 가능하다.

```
# systemctl -t service --list-unit-files
```

# 서비스 부트업

유닛 부트 업이 필요하다면, 다음처럼 선언이 가능하다.

```
# systemctl enable httpd.service
```

부트 업 및 즉시 서비스 시작이 필요한 경우, 아래처럼 옵션을 추가한다.

```
# systemctl enable --now httpd.service
```

# 서비스 부트업

유닛 부트 업 취소를 위해서 아래와 같이 실행이 가능하다.

```
# systemctl disable httpd.service
```

부트 업 취소는 아래와 같이 실행한다.

```
# systemctl disable --now httpd.service
```

"--now" 옵션은 항상 즉시 서비스에 적용하는 명령어이다.

# 서비스 재시작

재 시작은 다음과 같이 가능하다.

```
# systemctl restart httpd.service  
# systemctl restart httpd.socket
```

재 시작과 비슷한 다시 불러오기 기능이 있다. 이 기능은 프로세스를 종료하지 않고 메모리에 설정 내용을 갱신한다.

```
# systemctl reload httpd  
# systemctl reload sshd  
# systemctl reload vsftpd
```

# 블록 업데이트

systemd관련 설정 및 정보가 변경이 되면 반드시 systemd에게 알려야 한다. 보통 다음과 같은 부분이 변경 시 daemon-reload가 필요하다.

1. 모든 유형의 유닛 파일 변경 시
2. /etc/fstab, 커널 모듈과 같은 정보 변경 시
3. 블록 장치 및 네트워크 인터페이스 추가 및 변경 시

위와 같은 변경이 발생하면, 다음과 같이 명령어를 실행한다.

```
# systemctl daemon-reload  
# systemctl daemon-reexec
```



# 블록 업데이트

reload는 유닛 파일만 다시 읽어 오며, reexec는 systemd 대몬을 재 시작하면서, systemd관련 설정 파일을 다시 불러온다.

항목	daemon-reload	daemon-reexec
목적	systemd의 설정 파일만 다시 읽기	systemd 프로세스 자체를 재실행
주요 대상	Unit 파일 변경(서비스 파일 등) 반영	systemd 자체 업데이트나 재구동 필요시
시스템 영향	낮음 (서비스 영향 거의 없음)	중간 (잠깐 서비스 영향 가능)
사용 상황	유닛 파일 수정 후	systemd 업데이트 후 혹은 데몬에 이상이 있을 때

# 로그 및 로깅

systemd

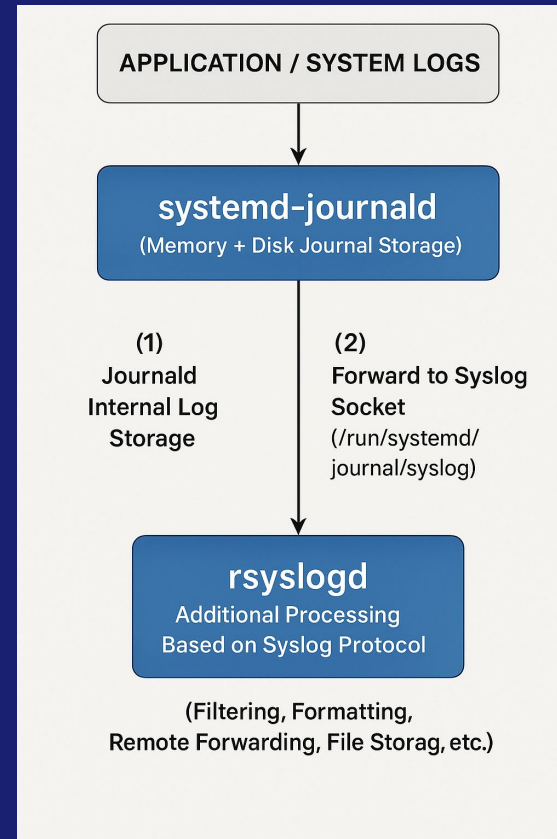
# 로그 관리

systemd로 통합이 되면서 이전에 사용하였던 syslog/rsyslog는 더 이상 주요 로깅 백-엔드는 아니다.

systemd에서는 systemd-journald를 통해서 커널 및 애플리케이션 로그를 로깅한다. 이 정보를 다시 rsyslogd에 전달한다.

구분	systemd-journald	rsyslogd
역할	기본 로그 수집 및 저장	고급 필터링, 전송, 저장 처리
저장 위치	메모리(휘발성) 및 디스크(퍼시스턴트)	파일 (예: /var/log/messages 등)
통신 방식	애플리케이션 → journald API 호출	journald → syslog 소켓 통신
특징	바이너리 형식(.journal 파일)	전통적인 텍스트 기반 로그
필터링/가공 능력	제한적 (주로 메타데이터 검색)	매우 강력 (구문 분석, 포워딩 가능)
시스템 통합	systemd 기본 구성요소	선택적으로 사용 가능

# 로그 아키텍처



# systemd-journald

systemd에서는 systemd-journald기반으로 독립적으로 사용이 가능하다. 다만, 반대로 rsyslogd는 독립적으로 더 이상 사용이 되지 않는다.

1. systemd-journald는 독립적으로 사용이 가능
2. rsyslogd는 독립적으로 절대 사용이 불가능(systemd -> rsyslog)

이유는 다음과 같다.

- rsyslogd는 레거시 애플리케이션을 제외, 모든 커널 및 애플리케이션 로그는 systemd를 통해서 전달 받는다.
- 기존 레거시 프로그램은 rsyslogd에서 직접 관리가 가능하다.
- 위의 조건과 다른 부분은 전부 systemd-journald를 통해서 전달 받는다.

# journalctl

systemd-journald에서 기록한 내용을 검색하기 위해서 다음과 같은 명령어를 사용한다.

```
# journalctl
```

journalctl이 조회는 아래 위치에서 검색한다.

1. /run/log/journal

2. /var/log/journal

여기에 저장된 데이터는 rsyslog와 다르게 바이너리 데이터로 구성이 되어 있다. 먼저, 간단하게 현재 어떤 내용들이 데이터베이스 저장이 되어 있는지 확인한다.

```
# journalctl --list-catalog
```

```
# journalctl --list-boots
```

# 커널 로그

예를 들어서 커널에서 발생한 로그를 확인하기 위해서 다음과 같이 systemd-journald에 조회한다.

```
# journalctl -k -perr -pwarning
```

위와 같이 하면 커널에서 발생한 오류 및 경고 메시지가 화면에 출력이 된다. 이 결과는 dmesg를 통해서 나온 결과와 동일하다.

# 부트 로그

부팅 시 로그 경우에는 아래 명령어와 같이 조회한다. 다만, systemd는 기본적으로 로그를 메모리에 임시로 저장한다. 이러한 이유로 재시작을 하면 모든 내용은 제거가 된다.

이러한 이유로 영구적으로 저장이 되도록 특정한 작업을 한다. 제일 간단한 방법은 특정 위치에 특정 디렉터리가 생성이 되어 있으면 systemd-journald는 영구적으로 저장한다.

부팅 로그는 systemd가 시스템을 초기화 하면서 발행한 메시지가 출력이 된다. 조회 시, -k 옵션과 동일하게, -p 옵션을 통해서 특정 우선순위만 조회가 가능하다.

```
# journalctl --list-boots
# journalctl -b <IDX> OR <BOOT_ID>
# journalctl -b -1 -perr -pwarning
# mkdir -m 0644 -p /var/log/journal
# systemctl daemon-reexec
```



# 부트 로그

혹은 설정파일 변경이 가능하다.

```
# vi /etc/systemd/journald.conf
[Journal]
Storage=auto
Compress=yes
Seal=yes
# systemctl daemon-reload
# systemctl daemon-reexec
```

# 서비스 로그

자주 사용하는 httpd, sshd의 로그 조회 하는 방법은 다음과 같다.

```
# journalctl -u httpd -u sshd -p warning -p err
```

이와 같이 두 개의 유닛을 한번에 필터링이 가능하다. 혹은 이전에 tail -f를 통해서 모니터링은 다음과 같이 하면 동일한 기능을 구현한다.

```
# journalctl -u httpd -u sshd -p warning -p err -fl
```

# 서비스 로그

하지만, 우리는 가끔 로그를 좀 더 자세히 확인이 필요하다. 이러한 상황에서는 다음처럼 설정한다.

```
# mkdir -p /etc/systemd/systemd/httpd.service.d/override.conf
[Service]
Environment="LOG_LEVEL=debug"
# systemctl daemon-reload
# systemctl status httpd.service
```

유닛 설정이 변경이 되면 반드시 꼭 `daemon-reload`를 실행해야 한다.

부팅

systemd

# 소프트 리부팅

systemd기반으로 넘어오면서 본격적으로 soft reboot를 제공한다. 기존 레거시 System-V, BSD Init는 이러한 기능을 제공하지 않는다. 하지만, run-init, up-start와 같은 개선된 스크립트 기반의 init시스템은 종류에 따라서 지원한다.

soft reboot는 시스템에 다음과 같은 상태를 적용한다.

1. 커널의 데이터는 건들지 않음
2. 사용자 영역 데이터만 초기화
3. 사용자가 사용중인 모든 프로세스 종료
4. 시스템 서비스 중지
5. 일부 커널 모듈(드라이버) 재시작

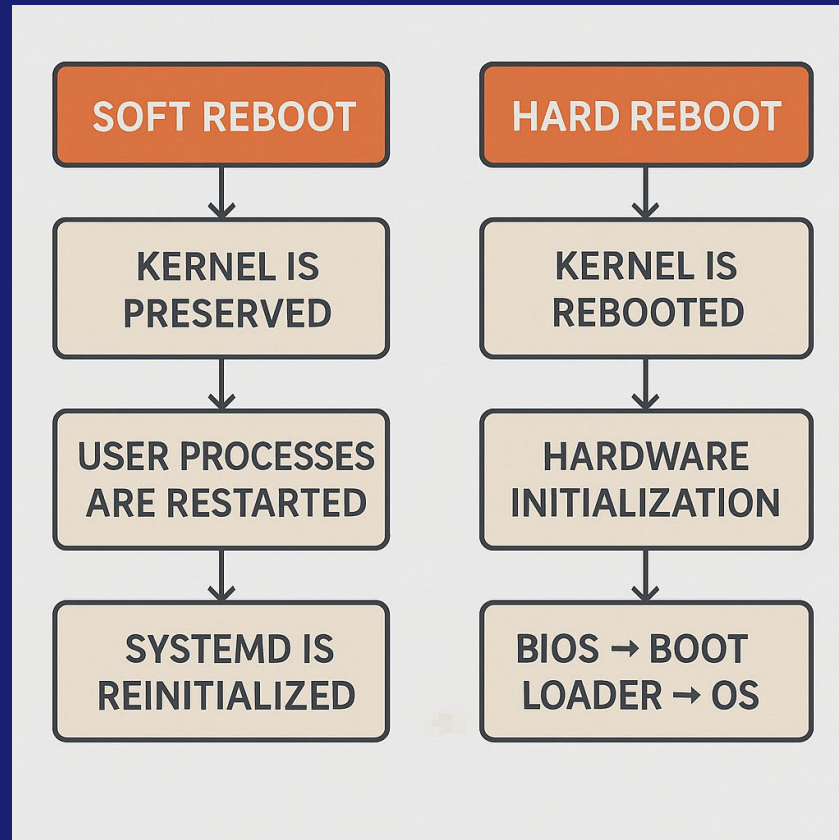
이 기능은 systemd 250버전부터 지원하고 있다.

# 소프트 리부팅

systemd에서 reboot는 다음과 같이 지원한다.

명령어	설명
<b>systemctl reboot</b>	완전한 하드 리부트 (커널, 하드웨어 포함)
<b>systemctl soft-reboot</b>	유저 스페이스 리부트 (커널 유지)
<b>systemctl rescue</b>	최소한의 단일 유저 모드로 진입
<b>systemctl default</b>	기본 런레벨(target)로 복귀

# 리부팅 분류 및 흐름



# 분석

systemd는 부팅 영역에 대해서 분석이 가능하다. 이 부분은 유닉스 및 리눅스 계열에서는 유일하게 systemd만 지원한다. 이 기능이 필요한 이유는, 가끔 특정 서비스가 부트업이 느린 경우가 있다.

이러한 이유로, 어떠한 서비스가 부팅 시 문제가 발생하는지 확인이 가능하다.

BIOS/UEFI → 부트로더(GRUB) → 커널 → initramfs → systemd (PID 1) → 유닛(service, target 등)

부팅 시간을 확인하기 위해서 아래 명령어로 확인이 가능하다.

```
# systemd-analyze
```

```
Startup finished in 3.244s (firmware) + 1.651s (loader) + 2.305s (kernel) +  
4.890s (userspace) = 12.090s
```



# 분석

출력되는 내용의 항목은 다음과 같다.

항목	설명
<code>firmware</code>	BIOS/UEFI 단계
<code>loader</code>	GRUB 등 부트로더
<code>kernel</code>	커널 로딩 시간
<code>userspace</code>	systemd와 서비스 시작 시간

# 분석

서비스 별 소요 시간을 확인하기 위해서 아래와 같이 명령어를 실행한다.

```
# systemd-analyze blame  
5.132s NetworkManager.service  
2.895s dev-sda1.device  
2.354s systemd-logind.service
```

# 분석

병목 현상을 확인하기 위해서 아래와 같이 명령어를 실행한다.

```
# systemd-analyze critical-chain
multi-user.target @10.456s
└─sshd.service @9.121s +1.334s
   └─network.target @9.111s
```

# 부팅분석

부팅 로그는 앞에서 이야기 하였지만, 아래 명령어로 확인이 가능하다.

```
# journalctl -b          # 현재 부팅
# journalctl -b -1       # 이전 부팅
# journalctl -b -p err
# systemctl list-jobs    # 현재 부팅 혹은 재시작 중인 유닛 확인
```

특정 서비스가 문제가 있다고 판단이 되면, 아래 명령어를 참고한다.

명령어	설명
<b>dmesg</b>	커널 부팅 메시지 확인 (디바이스 문제 추적)
<b>journalctl -xe</b>	문제 발생 직후의 로그 집중 보기
<b>systemctl list-unit-files</b>	어떤 서비스가 enable/disable 상태인지 확인
<b>systemctl preset-all</b>	기본 프리셋으로 서비스 상태 초기화

# 램 디스크

램 디스크 갱신은 systemd에서 자동으로 재시작 전, 자동으로 initramfs를 갱신한다. 편의상 ramdisk라고 부르지만, 부팅이 가능한 이미지는 initramfs라고 부르는 게 올바르다. 여기서는 관습적으로 사용하는 ramdisk라고 부르겠다.

레드햇 계열은 다음과 같은 명령어로 갱신이 가능하다.

```
# dracut -f  
# dracut -f /boot/initramfs-$(uname -r).img $(uname -r)
```

# 램 디스크

데비안 계열은 다음과 같은 명령어로 갱신이 가능하다.

```
# update-initramfs -u
# update-initramfs -u -k all
```

옵션	설명
-u	현재 커널용 업데이트
-k all	모든 커널용 갱신
-c	새로 생성 (기존 initramfs를 덮어쓰지 않음)

수세 리눅스 계열은 다음과 같은 명령어로 갱신이 가능하다.

```
# mkinitrd
# dracut -f          # 현재는 dracut으로 전환 중
```

# 램 디스크 갱신 조건

initramfs를 갱신을 위한 조건은 다음과 같다.

상황	설명
커널 업그레이드	자동으로 갱신되지만 수동 확인 가능
LVM, RAID 변경	관련 모듈을 initramfs에 포함시켜야 함
루트 파일 시스템 변경	UUID, LABEL 바뀐 경우 중요
암호화(LUKS) 적용/해제	초기 마운트 설정 변경시 필요
systemd 초기 유닛 수정	init 단계에 적용하려면 반영 필요

# 램 디스크 확인

initramfs의 내용 확인은 아래와 같이 가능하다.

```
# lsinitrd                                # RedHat, SUSE 계열
# lsinitramfs                             # Debian 계열
# file /boot/initramfs-*.img
```



# OOM

systemd

# 소개

OOM, Out Of Memory의 약자. OOM은 시스템 메모리가 부족하기 시작하면 OOM Killer가 동작하면, 최근에 생성된 프로세스 위주로 중지한다.

이전에는 커널에서 관리 하였지만, 현재는 systemd-oomd가 도입이 되어서 systemd에서 시스템 메모리 관리 및 OOM를 관리한다.

OOM은 기본적으로 리눅스 커널의 C-GROUP를 통해서 관리 및 추적이 된다. 추적 시, systemd에서는 .slice라는 자원을 생성하여 프로세스의 자원 상태를 확인한다.

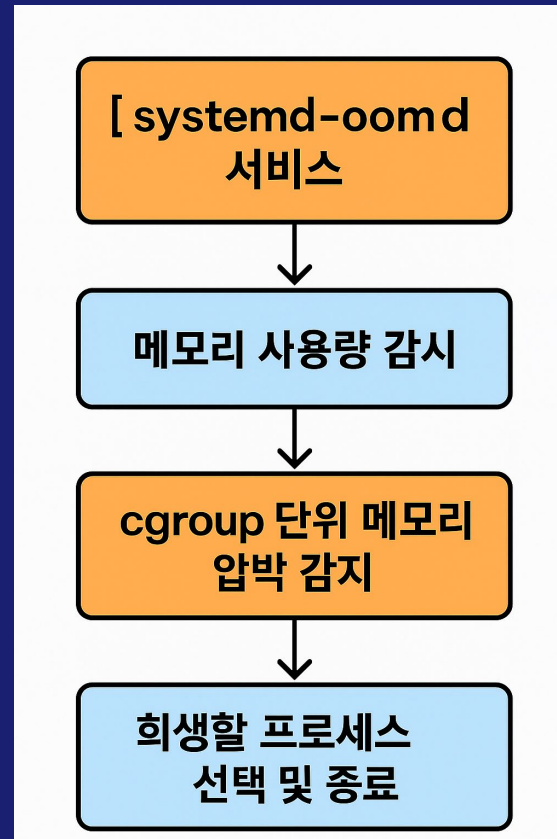
항목	설명
기능 이름	systemd-oomd (systemd 248버전부터 도입)
모니터 대상	서비스, 스코프, 슬라이스 (slice 단위)
동작 방식	메모리, 스왑 사용량을 모니터링해서 임계치 넘으면 프로세스를 정리(kill)
정책	MemoryPressure, MemoryMin, MemoryMax, Swap usage 기반 판단

# 소개

기존 `kernel-oom`과 `systemd-oomd`과 아래와 같은 차이점이 있다.

항목	커널 OOM Killer	systemd-oomd
개입 시점	메모리 진짜 완전히 고갈될 때	메모리 압박 단계에서 조기 대응
대상 선정	OOM Score 기준	압박+정책 기반 더 세밀하게
제어 가능성	낮음	높음 (서비스별 fine control)
목적	커널 보호	시스템 안정성 확보

# OOM



# 명령어 및 설정 파일

올바르게 설치가 되어 있으면, 다음 명령어로 확인이 가능하다. 다만, 버전별로 지원이 안되는 경우 저장소에서 별도 설치 혹은 systemd를 업데이트 한다.

```
# systemctl status systemd-oomd
# systemctl enable systemd-oomd
# systemctl start systemd-oomd
# systemctl is-active systemd-oomd
```

설정 파일 위치는 다음과 같다.

1. /etc/systemd/oomd.conf
2. /etc/systemd/system.conf
3. /etc/systemd/user.conf

# 설정

설정 파일의 주요 지시자는 다음과 같다.

설정 항목	의미
DefaultMemoryPressureThresholdSec	메모리 압박 지속 시간 설정
DefaultMemoryPressureWatchSec	감시 주기 설정
ManagedOOMMemoryPressure	MemoryPressure 기반 OOM 관리
ManagedOOMSwap	Swap 기반 OOM 관리 활성화

# 설정

만약, 특정 유닛에게만 적용하는 경우 **드랍 디렉터리(drop directory)**를 만들어서 아래와 같이 지시자를 추가한다.

```
# mkdir -p /etc/systemd/system/httpd.service.d
# vi /etc/systemd/system/httpd.service.d/override.conf
[Service]
ManagedOOMSwap=kill
ManagedOOMMemoryPressure=kill
```

# 예상 적용

다음과 같이 디렉터리를 구성 후 `systemd-oofd` 기반으로 구성한다.

```
/etc/systemd/system/  
├─ web.service.d/  
│  └─ oom.conf  
├─ background.service.d/  
│  └─ oom.conf
```

위의 조건은 다음과 같다.

1. `web.service`, 이 서비스는 무조건 유지가 되어야 한다.
2. `background.service`, 이 서비스는 상태에 따라서 종료가 된다.



# 예상 적용

`web.service.d`에 다음과 같이 적용한다.

```
# mkdir -p /etc/systemd/system/httpd.service.d
# vi /etc/systemd/system/httpd.service.d/oom.conf
[Service]
ManagedOOMSwap=inhibit
ManagedOOMMemoryPressure=inhibit
```

# 예상 적용

`background.service.d`에 다음과 같이 적용한다.

```
# cat << 'EOF' > /etc/systemd/system/stress.service
[Unit]
Description=Stress Memory Load
After=network.target
[Service]
ExecStart=/usr/bin/stress --vm 1 --vm-bytes 70% --vm-hang 0
Restart=always
RestartSec=5
[Install]
WantedBy=multi-user.target
EOF
```

# 예상 적용

위의 내용 계속 이어서...

```
# mkdir -p /etc/systemd/system/stress.service.d
# cat << 'EOF' > /etc/systemd/system/stress.service.d/oom.conf
[Service]
ManagedOOMSwap=kill
ManagedOOMMemoryPressure=kill
OOMPolicy=kill
EOF
```

# 적용 및 확인

최종적으로 아래와 같이 적용한다.

```
# systemctl daemon-reload
# systemctl enable --now httpd.service
# systemctl enable --now stress.service
# journalctl -u systemd-oomd
# systemctl status -u httpd.slice
# systemctl status -u stress.slice
```

# MOUNT

systemd

# 설명

마운트는 두 가지 명령어로 제공하고 있다.

1. 표준 마운트 명령어(mount)
2. systemd 기반으로 제공하는 마운트 명령어(systemd-mount)

둘 중 하나만 사용하여도 운영에는 문제가 없다. 다만, systemd의 .mount 유닛으로 구성 및 구현이 되어 있는 경우, systemd-mount 명령어 사용을 권장한다.

# MOUNT

일반적인 **mount** 명령어는, 두 가지 조건을 가지고 동작한다.

1. /etc/fstab에 등록이 되어있는 블록장치
2. 사용자가 옵션을 통해서 명시한 블록장치

**mount** 명령어는 /etc/fstab에 명시가 되어 있지 않으면, 직접 명시하여 블록 장치 연결 및 구성이 가능하다.

```
# vi /etc/fstab
/dev/sdc1 /mnt/sdc ext4 defaults 0 0
# mount /dev/sdb1 /mnt/sdb1
# mount /mnt/sdc1
```

# MOUNT

현재 사용하는 mount 명령어는 VFS와 연동이 되어 있기 때문에, 커널 모듈에 파일 시스템 모듈이 있으면 자동으로 파일 시스템 유형을 확인한다. 이러한 이유로 마운트 혹은 사용하기 원하는 파일 시스템이 있으면, 반드시 progs 패키지가 있는지 확인한다.

```
# dnf search progs
> xfs
> ext4
# mount /dev/sdd1 /mnt/btrfs
# dnf install btrfs-progs
# mount -t btrfs /dev/sdd1 /mnt/btrfs
```



# MOUNT

기존에 사용하던 **mount** 명령어는 여전히 사용이 가능하다. 또한, 마운트 시 참조하는 **/etc/fstab**도 여전히 사용이 가능하다.

**systemd**에서는 내부적으로 더 이상 **/etc/fstab** 직접적으로 사용하지 않고, **.mount** 유닛으로 전환 후 사용한다. 앞으로 모든 시스템은 가급적이면 **.mount** 유닛 기반으로 구성 및 연결을 권장한다. 정리하면 다음과 같다.

1. **.mount** 확장자를 가지고 있음. 이 유닛은 일반적으로 **/etc/systemd/system**에 위치한다.
2. 디렉터리 구분자는 **/**가 아닌 **-**으로 구분
3. 기존의 **/etc/fstab** 파일은 **systemd-fstab-generator**으로 자동 변환
4. **/etc/fstab**의 내용은 **/run/systemd/generator** 밑에서 생성 및 만들어 짐
5. **/etc/fstab** 내용이 변경이 되면 반드시 **systemctl daemon-reload**으로 갱신 권장

# 명령어

기존에 사용한 명령어 `mount`는 여전히 사용이 가능하다. 다만, 시스템 블록으로 통합이 되면서 `systemd-mount` 라는 명령어가 하나 추가 되었다. 사용 방법은 기존 `mount` 명령어와 거의 동일하다.

```
# systemd-mount /dev/sdb1 /mnt/mydisk
# systemd-mount --options=ro /dev/sdb1 /mnt/mydisk
# systemd-mount --what=UUID=xxxx-xxxx --where=/mnt/mydisk
# systemd-mount --type=ext4 /dev/sdb1 /mnt/mydisk
# systemd-umount /mnt/mydisk
```

# 사용법

`systemd-mount`는 영구적으로 마운트 기능을 제공하지 않는다. 재부팅을 하면 해당 내용들은 다시 제거가 되고, `initramfs`에서 부팅 시 다시 재구성한다.

이러한 이유로 새로운 디스크를 `.mount`유닛으로 관리하고 싶으면 다음처럼 작성한다.

```
[Unit]
Description=Mount mydisk
[Mount]
What=/dev/sdb1
Where=/mnt/mydisk
Type=ext4
Options=defaults
[Install]
WantedBy=multi-user.target
```

# 자동마운트

이전에 사용하였던 **autofs**는 복잡하고 사용하기가 어려웠다. **systemd**로 변경이 되면서, **autofs**는 **automount**라는 자원으로 통합 및 변경이 되었다. **automount**는 **mount**유닛과 의존성이 있다.

사용 방법은 앞에서 사용하였던 **mount**와 거의 동일하다. 파일 이름에 꼭 **대시(-)**로 디렉터리 구별을 해야 한다.

```
# vi home-testuser.automount
[Unit]
Description=automount for testuser
ConditionPathExists=/home/testuser
[Automount]
Where=/home/testuser
TimeoutIdleSec=10
[Install]
WantedBy=multi-user.target
```

# 자동마운트

오토 마운트가 사용할 일반 `mount` 자원도 생성한다.

```
# vi home-testuser.mount
[Unit]
Description=mount for testuser

[Mount]
What=$YOUR_SERVER:/$YOUR_SHARE
Where=/home/testuser
Type=nfs
Options=_netdev,auto

[Install]
WantedBy=multi-user.target
```

# 자동마운트

systemd의 automount와 비교하기 위해서 autofs서버를 설치 및 구성한다. 필요하지 않는 경우, 굳이 진행하지 않아도 된다. 아래와 같은 방식 더 이상 사용을 권장하지 않는다.

```
# dnf search autofs
# dnf install autofs
# systemctl enable --now autofs.service
# vi /etc/auto.master.d/direct.conf
/- /etc/auto.direct
# vi /etc/auto.direct
/direct -fstype=auto,rw,async node1.example.com:/direct
# vi /etc/auto.master.d/indirect.conf
/home/ /etc/auto.indirect
# vi /etc/auto.indirect
* -rw,soft,async node1.example.com:/indirect/&
```

# 자동마운트

연결이 완료가 되면, 아래와 같이 확인이 가능하다.

```
# showmount -e node1.example.com
# ssh testuser@node1.example.com
# pwd
# df -h
```

# 정리

위의 기능을 정리하면 다음과 같다.

구분	autofs	systemd-automount
기본 구조	별도 autofs 데몬이 동작하며, 마운트 트리거를 감시	systemd의 .automount 유닛이 마운트 트리거를 담당
서비스 이름	autofs.service	-.mount, .automount 유닛 시스템
설정 파일	/etc/auto.master, /etc/auto.misc, /etc/auto.direct 등	.mount, .automount 유닛 파일
설정 복잡도	복잡한 트리, LDAP, NIS 연동 가능 (대규모 지원)	단순한 경로별 자동 마운트에 최적화
초기 설치	별도 패키지 설치 필요 (예: yum install autofs)	systemd 기본 탑재
관리 방법	autofs 서비스 리스타트 필요	systemctl로 유닛 reload/restart
대상 환경	대규모 네트워크 환경 (NFS, LDAP 대량 마운트)	소규모 서버, 단일 경로 자동 마운트



# 정리

앞에 내용 계속 이어서...

구분	autofs	systemd-automount
트리거 방식	커널 VFS 핸들링 후 autofs 데몬이 mount 요청	systemd가 파일시스템 접근 시 직접 mount 요청
의존성 관리	autofs 자체에서 mount 시점을 조절	systemd dependency로 부트/서비스 순서 제어 가능
퍼포먼스	대규모 환경에서 안정적 성능	간단한 환경에 빠르고 가벼움
유지보수	별도 데몬 관리 필요 (autofs 업그레이드 등)	systemd 업데이트에 통합됨

# 자주 사용하는 기본적인 리눅스 명령어

기본 명령어

# 사용자 및 그룹

기본 명령어

# 설명

모든 리눅스 시스템은 최소 한 개 이상의 사용자가 필요하다. 사용자는 크게 두 가지로 나뉘어진다.

1. 시스템 사용자
2. 일반 사용자

**시스템 사용자**는 말 그대로 커널 및 서비스에서 사용하는 계정이다. 일반 사용자가 사용하지 않으며, 일반적으로 로그인 기능을 요구하지 않는다.

**일반 사용자**는 SSH와 같은 서비스로 시스템에 접근이 가능한 계정. 일반적으로 아이디/비밀번호로 접근이 가능하며, 로그인 기능을 요구한다.

# 사용자 추가/삭제

사용자 관리는 총 3가지로 구별이 된다.

1. 사용자 생성
2. 사용자 제거
3. 사용자 관리

사용자 생성은 useradd라는 명령어로 작업한다.

사용자 제거는 userdel라는 명령어로 작업한다.

사용자 관리는 다음과 같은 명령어로 작업한다.

- usermod
- getent

# 사용자 추가/삭제

사용자 추가는 아래와 같은 옵션을 사용한다. 자주 사용하는 옵션은 빨간색으로 표시.

옵션	의미	설명
<b>-u UID</b>	사용자 ID 지정	직접 UID(숫자)를 지정할 때 사용
<b>-g GID</b>	기본 그룹 지정	사용자의 기본 그룹 설정
<b>-G 그룹1,그룹2</b>	추가 그룹 지정	사용자가 추가로 속할 그룹 목록 설정
<b>-d 디렉터리</b>	홈 디렉터리 지정	기본이 /home/사용자명이지만 변경 가능
<b>-s 셸</b>	로그인 셸 지정	/bin/bash, /bin/zsh 등 로그인할 때 사용할 셸
<b>-c "코멘트"</b>	설명(Comment) 추가	사용자에 대한 간단한 설명 입력
<b>-e YYYY-MM-DD</b>	계정 만료일 설정	계정이 자동으로 비활성화될 날짜 설정

# 사용자 추가/삭제

앞에 내용 계속 이어서...

옵션	의미	설명
-f 일수	비활성화 전 유예 기간 설정	비밀번호 만료 후 비활성화까지 유예 기간(일 단위) 설정
-m	홈 디렉터리 생성	홈 디렉터리를 자동으로 생성 (/etc/skel 복사)
-M	홈 디렉터리 생성 안 함	홈 디렉터리 생성하지 않음
-r	시스템 계정 생성	일반 사용자 대신 시스템 계정 생성 (UID 범위 다름)
-N	기본 그룹 생성 안 함	사용자명과 동일한 그룹을 자동 생성하지 않음
-p '암호화된 비밀번호'	암호 설정	직접 암호화된 문자열을 넣을 때 사용 (일반적으로 passwd 로 설정하는게 보통)

# 사용자 추가/삭제

앞에 내용 계속 이어서...

옵션	의미	설명
-k	스켈레톤 디렉터리 지정	기본 /etc/skel이 아니라 다른 스켈레톤 디렉터리 사용
-b /경로	기본 홈 디렉터리 베이스 지정	예: /home 대신 /srv/home 같은 곳에 홈 디렉터리 생성
-o	UID 중복 허용	특별한 경우, UID가 중복되어도 생성 가능하게 함 (주의 필요)



# 계정 생성

계정 생성은 매우 간단하다. 여기서는 다음과 같은 조건으로 사용자를 생성한다.

1. 사용자 이름은 user1, 셸은 bash
2. 사용자 이름은 user2, 셸은 fish
3. 모든 사용자 비밀번호는 testpasswd로 설정한다.

```
# adduser -s bash user1
# adduser -s fish user2
# passwd user1
# passwd user2
```

# 비밀번호 설정

하지만, 매번 사용자 비밀번호를 수동으로 설정 해야 되기 때문에 보통 두 가지 방식을 많이 사용한다. 하지만, 이 방식은 데비안 계열 배포판에서는 지원하지 않는다.

데비안 계열 경우에는 `chpasswd`를 사용해서 변경한다.

```
# echo testpasswd | passwd --stdin user1
# echo testpasswd | passwd --stdin user2
# echo "user1:1234abcd" | sudo chpasswd
# echo "user2:1234abcd" | sudo chpasswd
```

# 비밀번호 설정

다른 비밀번호 설정 방법은 `mkpasswd`를 사용한다. 사용자가 입력한 문자열을 SHA512형태로 암호화 하여, 비밀번호를 생성한다. 현재 모든 리눅스 배포판은 대다수가 SHA256혹은 512를 사용한다.

```
# dnf install -y whois
# apt install -y whois
# dnf install -y whois
# mkpasswd -m sha-512 -s
# useradd -m -p "$(mkpasswd -m sha-512 'testpasswd')" user1
```

# 사용자 이름 변경

종종 사용자 이름 변경이 필요하다. 사용자 이름 변경을 위해서 usermod 명령어를 통해서 수정이 가능하다. 변경 시, 다음과 같은 조건을 고려해야 한다.

1. 기존 내용에 영향이 가는가?
2. 홈 디렉터리도 변경이 필요한가?

변경이 필요한 경우, 아래와 같이 명령어를 실행한다.

```
# usermod -l user1 testuser1  
# usermod -d /home/testuser1 -m testuser1
```

# 그룹

리눅스에서 그룹은 유닉스와 비슷하면서, 조금 더 넓은 기능을 제공한다. 리눅스에서 제공하는 그룹 명령어는 다음과 같다.

명령어	설명	기본 형식
groupadd	새로운 그룹 생성	groupadd 그룹명
groupdel	기존 그룹 삭제	groupdel 그룹명
groupmod	그룹 이름 변경 / GID 변경	groupmod 옵션 그룹명
gpasswd	그룹에 사용자 추가/삭제	gpasswd 옵션 그룹명
usermod	사용자의 그룹 설정/변경	usermod 옵션 사용자명
groups	사용자가 속한 그룹 조회	groups 사용자명
id	UID, GID, 그룹 리스트 조회	id 사용자명

# 그룹 생성

그룹을 생성/제거 및 변경을 위해서 다음과 같이 명령어를 실행한다.

```
# groupadd devteam  
# groupdel devteam  
# groupmod -n newdevteam devteam  
# usermod -g devteam user1  
# usermod -aG devteam user1
```

**-n**: 그룹 이름을 명시한 이름으로 변경

**-g**: 기본 그룹을 devteam으로 변경

**-aG**: 보조 그룹에 devteam을 추가

# 그룹 생성

혹은 보조그룹에 아래 명령어로 추가가 가능하다.

```
# gpasswd -a user2 devteam
```

이 명령어는 `usermod`와 비슷하지만, 기본적으로 `gpasswd`명령어는 `-aG` 옵션과 동일한 기능을 한다. 그래서 굳이, `usermod -aG`를 사용하지 않고, `gpasswd`를 사용해서 **보조 그룹**을 추가한다.

# 그룹 생성

마지막으로 생각보다 많이 사용하지 않는 명령어 **newgrp**와 **groups**에 대해서 설명 및 예제. 그룹 생성 후, 특정 사용자를 그룹에 포함. 이 후, 올바르게 **주/보조 그룹**이 변경이 되었는지 확인한다.

```
# groupadd qateam
# gpasswd -a user3 qateam
# groups user3
# newgrp qateam
id
# groups user3
```



# 사용자 및 그룹 확인

사용자 정보를 확인하기 위해서, 아래 명령어로 확인이 가능하다.

```
# getent passwd testuser1  
# getent group testuser1  
# getent passwd 1001
```

# 오프라인 상태에서 프로그램 실행

로그인 하지 않아도 프로그램을 계속 특정 사용자에게 실행되게 하려면 아래와 같이 로그인 옵션을 변경한다.

```
# loginctl enable-linger user1
```

# 퍼미션 및 소유권

기본 명령어

# 퍼미션

## 파일 퍼미션(Permission)

이름	설명
r (read)	읽기 권한 (파일 내용 읽기, 디렉터리 목록 보기)
w (write)	쓰기 권한 (파일 내용 수정, 디렉터리에 파일 생성/삭제)
x (execute)	실행 권한 (파일 실행, 디렉터리 접근)

## 소유권(Ownership)

이름	설명
User (Owner)	파일을 만든 사용자.
Group	파일이 속하는 그룹.
Other	나머지 모든 사용자.

# 퍼미션

## ACL (Access Control List)

1. 기본 퍼미션보다 더 세밀하게 접근 제어를 설정하는 기능.
2. 특정 사용자나 그룹에게 개별 권한을 부여할 수 있음.

## 마스크(Mask)

1. ACL 설정 시, group 및 named user/group entry에 대한 최대 허용 권한을 결정하는 값.
2. 즉, "ACL 필터" 같은 역할!

# 퍼미션

퍼미션을 기본적으로 두 가지 방식으로 선언 및 설정이 가능하다.

1. 지긋지긋한 8진수
2. 숙취에도 안전하게 사용이 가능한 심볼릭 문자

현재, GNU는 가급적이면 안전한 심볼릭 문자 기반으로 퍼미션 선언을 권장하고 있다. 8진수는 사람마다 다르지만, 대다수 사람들은 혼돈스러운 경우가 있기 때문에, 가급적이면 심볼릭 기반으로 권장한다.

사용 방법은 다음과 같다.

# 퍼미션

퍼미션 심볼릭 예제.

```
# chmod u+rwX
# chmod u-rX
# chmod g+r,u+r,o=
# chmod a=rwx
# chmod u=rwx,g=r,o=
```

# 퍼미션

숙취에 더더욱 괴롭히는 8진수는 다음과 같다. 기존 심볼릭과 비교해서 작성.

```
# chmod u+rwX      ->  chmod 0700
# chmod u-rX       ->  chmod 0400
# chmod g+r,u+r,o= ->  chmod 0440
# chmod a=rwx      ->  chmod 0777
# chmod u=rwx,g=r,o=-> chmod 0740
```



# 퍼미션

심볼릭, 8진수 둘 다 특수한 옵션을 선언 할 수 있다. 이를 **Sticky**, **SetUID/GID**라고 부른다.

```
# chmod u+s  
# chmod g+s  
# chmod o+t
```

**S**는 **Set**의 약자이며, 사용자 및 그룹에 적용이 가능하다. 이는 파일과 디렉터리에 퍼미션 및 메모리 소유권 및 그룹 정보에 영향을 준다. 보통, 특정 디렉터리의 그룹 권한 상속 혹은 메모리의 사용자/그룹 소유자 고정 시, 사용한다.

대표적인 예는 **sudo**, **su**와 같은 명령어가 여기에 포함이 된다.

**T**는 **Sticky**의 약자이며, Sticky로 선언된 디렉터리에서 생성된 파일 및 디렉터리는 생성한 사용자만 접근이 가능하다. 일반적으로 공유 디렉터리 생성 시, 많이 사용한다.

# 퍼미션(SETUID)

사용자는 SETUID라고 한다. SETUID 적용 시 다음과 같은 효과가 있다.

1. 디렉터리에는 적용하지 못함
2. 실행 파일에만 적용이 가능

```
# chmod u+s daemon  
# ps -ef -u <USERNAME> -o uid,comm
```

# 퍼미션(SETGID)

사용자는 SETGID라고 한다. SETGID 적용 시 다음과 같은 효과가 있다.

1. 디렉터리 적용 가능
2. 실행 파일에만 적용이 가능

```
# groupadd -g 12000 testgrp
# mkdir setgid/
# chgrp -Rf testgrp
# chmod g+s setgid
# ps -ef -u <USERNAME> -ogid,comm
# cd setgid/
# mkdir test
# ls -ld test/
```

# 퍼미션(STICKY)

공용 디렉터리 생성 시 많이 사용한다. 생성한 사용자만 파일 및 디렉터리에만 접근/삭제/수정이 가능하다.

1. 디렉터리 적용 가능
2. 파일에는 적용이 불가능

```
# mkdir /docs_share
# chmod o+t,a=rwx /docs_share
# su - user1
$ mkdir -p /docs_share/user1
$ touch /docs_share/user1/README.md
```

# POSIX PERMISSION

기본 명령어

# 설명

기존 퍼미션은 사용자별로 상세하게 제어가 안되기 때문에 POSIX 퍼미션을 사용한다. 보통 퍼미션 구별 시, 다음과 같이 호칭한다.

1. 표준 퍼미션(CHMOD, CHOWN)
2. POSIX 퍼미션(ACL)

자세하게 정리하면 다음과 같다.

항목	표준 퍼미션 (Standard Permission)	POSIX ACL (Access Control List)
기본 개념	파일/디렉토리에 대해 소유자(owner), 그룹(group), 기타(other)에게 권한(rwx)을 설정	파일/디렉토리에 대해 여러 사용자나 그룹 별로 세밀하게 권한(rwx)을 설정
제어 가능한 대상	3가지 (owner / group / others)	여러 개의 사용자(user)와 그룹(group) 개별 지정 가능
권한 유형	읽기(Read), 쓰기(Write), 실행(Execute)	읽기(Read), 쓰기(Write), 실행(Execute) + 추가 사용자/그룹 세분화

# 설명

표준 퍼미션은 9비트를 사용한다. 추가된 1비트는 특수 키를 위해서 사용한다. POSIX ACL 경우에는 INODE 확장 영역에 ACL를 구성한다. 둘 다 표준 POSIX를 따르고 있다.

항목	표준 퍼미션 (Standard Permission)	POSIX ACL (Access Control List)
데이터 구조	간단한 9비트 (rwxrwxrwx)	추가적인 ACL 엔트리로 구성 (확장 메타데이터)
저장 위치	inode 기본 필드	inode 확장 영역 (ext4, XFS, Btrfs 등 지원)
관리 명령어	chmod, chown, chgrp	getfacl, setfacl
복잡성	매우 단순	다소 복잡 (정밀 제어 가능)
POSIX 표준 여부	POSIX.1 표준 준수	POSIX.1e 초안에 기반 (POSIX 정식 채택은 실패했지만 대부분 시스템이 따라감)
사용 목적	간단하고 빠른 권한 설정	복잡하고 세밀한 권한 관리 필요 시 사용
예시	chmod 755 file (owner rwx, group r-x, others r-x)	특정 사용자에게 대해 읽기/쓰기 권한만 추가 설정

# 설명

많은 분들이 혼동하시는 부분이라서, 이 부분은 근거 내용을 아래 첨부.

항목	링크
POSIX.1e ACL 초안 문서	<a href="http://users.suse.com/~agruen/acl/posix/posix_acl.html">http://users.suse.com/~agruen/acl/posix/posix_acl.html</a>
NIST 공식 POSIX.1 표준 설명	<a href="https://pubs.opengroup.org/onlinepubs/9699919799/">https://pubs.opengroup.org/onlinepubs/9699919799/</a>
Linux man page (acl(5)) - 리눅스 ACL 설명	<a href="https://man7.org/linux/man-pages/man5/acl.5.html">https://man7.org/linux/man-pages/man5/acl.5.html</a>
Red Hat 공식 문서: ACL 개념 설명	<a href="https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/managing_file_systems/configuring-and-using-acls_managing-file-systems">https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/managing_file_systems/configuring-and-using-acls_managing-file-systems</a>



# 실행

아래와 같이 ACL 적용이 가능하다. 일반 파일에는 다음과 같이 적용한다.

```
# touch example.txt
# chmod 644 example.txt
# setfacl -m u:user1:rw example.txt
# setfacl -m g:ogroup1:r-- example.txt
# getfacl example.txt
```

# 실행

아래와 같이 ACL 적용이 가능하다. 일반 파일에는 다음과 같이 적용한다.

```
# mkdir mydata
# setfacl -m d:u:user3:rw mydata
# getfacl mydata
# touch mydata/testfile.txt
# getfacl mydata/testfile.txt

# setfacl -b mydata
```

# NTP

chronyd

systemd-timesyncd

# 설명

리눅스에서 사용하는 NTP 서비스는 두 가지가 있다.

1. chronyd
2. systemd-timesyncd

두 개의 큰 차이는 chronyd는 서버 기능이 포함이 되어 있으며, systemd-timesyncd는 클라이언트 기능만 지원한다.

# 설명

기능 비교하면 다음과 같다.

구분	NTP (ntpd, chronyd)	systemd-timesyncd
주 용도	서버 및 클라이언트 모두 가능	클라이언트 전용 (서버 기능 없음)
기능	고급 시간 조정 (drift correction, peer mode)	단순한 시간 동기화 (polling 기반)
정확도	매우 높음 (마이크로초 수준)	보통 (초 단위 정확도)
동작 방식	지속적인 미세 조정, 서버/피어와 복잡한 상호작용	주기적으로 시간 가져와서 큰 차이만 조정
서버로 동작	가능 (다른 장비에 시간 제공)	불가능
설치 여부	별도 설치 필요 (ntpd, chronyd)	systemd 기본 포함 (대부분 배포판에 기본 제공)
설정 복잡도	비교적 복잡 (ntp.conf 등 필요)	매우 단순 (timesyncd.conf 몇 줄)
리소스 소모	상대적으로 많음	매우 적음 (lightweight)
운영 시스템	서버, 정밀한 시스템 (예: 금융, 항공)	데스크탑, 가벼운 서버, IoT 장비

# chronyd

레드햇 계열 및 데비안 계열에서 아직까지 표준으로 사용하는 NTP 표준 서버 및 클라이언트 도구이다.

현재 레드햇 계열은 **chronyd** 기반으로 구성이 되어 있다. 설정 방법은 간단하다. 아래 설정 내용은 기본적으로 설정된 내용이다.

```
# grep -Ev '^#|^$' /etc/chrony.conf
pool 2.rocky.pool.ntp.org iburst
offline
auto_offline
driftfile /var/lib/chrony/drift
makestep 1.0 3
offset -0.00005
rtcsync
```

NTP서버 정보를 변경 한다.

- pool
- server

iburst/burst의 차이점은 burst는 4번 iburst는 4~8번의 요청을 서비스 시작 시, 바로 요청한다. 다만, burst는 각 요청에 대해서 응답을 기다리기 때문에 동기화가 느리다.

특정한 문제로 시간 값을 잃어버린 경우, 평균 값을 가지고 지속적으로 시간을 조정한다.

# chronyd

현재 대다수 리눅스 배포판은 `ntpd`에서 `chronyd`로 변경. 설정 내용은 크게 차이는 없다. `chronyd`를 사용하면 다음과 같은 이점이 있다.

1. 빠른 동기화 속도. 최소로 동기화 하면서 동기화를 한다. 보통 데스크탑 같이, 24시간 동기화가 필요하지 않는 시스템
2. CPU 클럭이 불안정한 시스템에 적합. 예를 들어서 가상머신이나 불안한 클럭 상태의 CPU.
3. 동기화 드리프트 기능 지원. 예를 들어서 데이터베이스 같은 미들웨어.
4. 트래픽이 포화일때 비대칭 처리를 안정적으로 가능.
5. 대다수 배포판은 현재 `ntpd`를 사용하지 않음.

# systemd-timesyncd, ntpd

**systemd**로 넘어 오면서 **systemd-timesyncd**을 통해서 동기화가 가능하다.

파일 위치는 다음 중 둘 중 하나를 사용해도 된다. 레드햇 포함, 다른 배포판들은 아직 기본으로 설치가 안되어 있기 때문에 사용을 원하는 경우, 설치를 해야 한다.

**systemd-timesyncd**를 실행하는 경우 자동으로 **chronyd.service**는 중지가 된다.

- `/etc/systemd/timesyncd.conf`
- `/etc/systemd/timesyncd.conf.d/local.conf`

```
# dnf install systemd-timsyncd
# systemctl enable --now systemd-timsyncd
# systemctl disable --now chronyd
```



# systemd-timesyncd, ntpd

```
# vi /etc/systemd/timesyncd.conf  
[Time]  
NTP=kr.pool.ntp.org  
FallbackNTP=0.pool.ntp.org 1.pool.ntp.org 0.fr.pool.ntp.org  
ServerName=kr.pool.ntp.org  
# systemctl enable --now systemd-timesyncd  
# systemctl disable --now chronyd
```

# systemd-timesyncd, ntpd

```
# timedatectl timesync-status
```

```
    Server: 121.174.142.82 (kr.pool.ntp.org)
```

```
Poll interval: 1min 4s (min: 32s; max 34min 8s)
```

```
    Leap: normal
```

```
    Version: 4
```

```
    Stratum: 3
```

# NTP서버(chronyd)

내부적으로 NTP 서버를 구축하기 위해서 chronyd를 사용한다.

```
# vi /etc/chrony.conf
server ntp.lab.int iburst
allow 192.168.0.0/24
driftfile /var/lib/chrony/drift
makestep 1.0 3
rtcsync
keyfile /etc/chrony.keys
leapsectz right/UTC
logdir /var/log/chrony
# firewall-cmd --add-service=ntp
# firewall-cmd --runtime-to-permanent
```

# 새로운 네트워크 구성 및 관리자 사용

NetworkManager

Netplan

# NetworkManager

네트워크

# 네트워크매니저

네트워크 매니저는 레드햇 계열은 RHEL 7/8/9에서 지원한다. 다만, RHEL 8/9에서는 7/8(초기버전)과 다르게 더 이상 ifcfg-\*를 지원하지 않는다.

1. 모든 네트워크 파일은 INI 형태로 `/etc/NetworkManager/`에 저장 및 관리.
2. 관리 파일은 `NetworkManager --print-config` 명령어로 확인.
3. `NetworkManager`는 `systemd-networkd`로 대체 및 통합될 예정.

이전에 버전에서 사용하였던 네트워크 매니저(RHEL기존 7 이후)와 현재 사용하는 네트워크 매니저와는 호환이 되지 않는다.

# 네트워크매니저

네트워크 매니저를 사용하기 위한 명령어는 아래와 같다.

```
# systemctl start NetworkManager  
# nmcli  
# nmtui  
# nm-connection-editor
```

# 네트워크매니저

기존에 사용하던 네트워크 설정은 init(System V)기반의 shell scrip(/etc/init.d/)로 되어 있었다.

보통 유닉스 스크립트는 `/etc/sysconfig/network-scripts`와 `/etc/sysconfig/network`을 통해서 관리 하였다.

현재는 RHEL 7이후로는 **Network Manager** 기반으로 변경. RHEL 7에서는 호환성을 위해서 스크립트 플러그인을 지원.



# 네트워크매니저

하지만, RHEL 8버전 이후로는 네트워크 스크립트는 선택 사항이며, RHEL 9 이후로는 더 이상 스크립트는 지원하지 않는다. 레드햇 리눅스 기반 리눅스 배포판은 네트워크 매니저를 기본으로 사용한다.

## 네트워크 매니저 관리 명령어

1. `nmtui`
2. `nm-connection-editor`
3. `nmcli`
4. `/etc/sysconfig/network-scripts/`

# 네트워크매니저

## nmtui

TUI기반으로 네트워크 설정한다. 자동화 용도로 사용하기는 어렵다.

```
# nmtui edit eth0
# nmtui connect eth0
# nmtui hostname
```

## nm-connection-editor

엑스 윈도우 기반으로 네트워크 설정 변경. 시스템 관리자는 거의 사용하지 않는 도구이다.

# 네트워크매니저

## nmcli

CLI기반으로 네트워크 인터페이스 변경. 쉽지는 않지만, 자동화나 혹은 반복적으로 수정 시 도움이 된다.

아래 내용은 RHEL 9기반에서 사용하는 NetworkManager 설정 내용이다. 더 이상 네트워크 매니저는 ifcfg-rh를 사용 및 생성하지 않는다.

```
[main]
# plugins=
# rc-manager=auto
# migrate-ifcfg-rh=false
```

# 네트워크매니저

`/etc/sysconfig/network-scripts/`

RHEL 7/8까지는 지원. RHEL 9부터는 더 이상 지원하지 않는다.

# 네트워크매니저

## nmcli

네트워크 매니저는 설정 파일이 프로파일(profile) 기반으로 구성이 된다. 모든 정보는 INI형태로 저장이 되며, 이를 통해서 네트워크 매니저 엔진이 인터페이스 카드에 설정 및 구성한다.

```
# nmcli con add con-name eth1 ipv4.addresses 10.10.1.1/24 ipv4.gateway  
10.10.1.250 ipv4.dns 10.10.1.250 ipv4.method manual ifname eth1 type  
ethernet  
# nmcli con mod eth1 ipv4.addresses 10.10.1.2/24  
# nmcli con up eth1  
# nmcli con sh eth1 -e ipv4.addresses -e ipv4.gateway -e ipv4.dns  
# nmcli con del eth1
```

# 네트워크매니저 기존 방식

네트워크 매니저에서 기존 방식으로 다시 사용하기 위해서 다음과 같은 과정이 필요하다. 다만, 아래 과정은 레드햇 계열 9 버전부터는 아래 방법으로 사용을 권장하지 않는다.

```
# vi /etc/NetworkManager/NetworkManager.conf
[main]
plugins=keyfile,ifcfg-rh
# systemctl restart NetworkManager
# nmcli connection migrate --plugin ifcfg-rh
```

# 네트워크매니저 기존 방식

앞에 내용 계속 이어서...

```
# ls -l /etc/sysconfig/network-scripts/  
ifcfg-eth0 ifcfg-eth1 readme-ifcfg-rh.txt  
# vi ifcfg-eth1  
IPADDR=10.10.10.1 → 10.10.10.2  
# nmcli con reload  
# nmcli con down eth1 && nmcli con up eth1
```

# Netplan

네트워크



# 설명

netplan은 우분투 제작사인 Canonical에서 만들었다. 우분투 버전 17.10부터 지원하기 시작하였고, 이는 ifcfg up/down를 대체하는 용도로 시작.

주요 목적은 다양한 백 엔드를 제공하여 통합적으로 운영이 가능하도록 한다. Netplan은 cloud-init에서 지원해주기 때문에, 가상머신 이미지에서 손쉽게 사용이 가능하다. 지원하는 백-엔드는 다음과 같다.

1. systemd-networkd
2. NetworkManager

# 설명

Netplan/NetworkManager/systemd-networkd의 기능을 비교하면 다음과 같다.

항목	Netplan	systemd-networkd	NetworkManager
역할	설정 프론트엔드	실제 네트워크 설정 관리	실제 네트워크 설정 관리
사용형태	YAML 설정 → 백엔드로 전달	.network, .netdev	GUI/CLI
대상 환경	Ubuntu Server/Desktop	모든 배포판에서 지원	모든 배포판에서 지원
관리 대상	systemd-networkd 또는 NM 선택	모든 네트워크 자원	모든 네트워크 자원

# 설명

위의 내용 계속 이어서...

항목	Netplan	systemd-networkd	NetworkManager
GUI 지원	없음	없음	GNOME, KDE 등과 통합
동적 재구성	netplan apply	프로파일 재반영	실시간 반영 가능
직접 사용 가능	안됨	networkctl	nmcli, nmtui
주요 배포판 사용	Ubuntu 17.10+	모든 배포판	대다수 배포판

# 명령어

netplan를 통해서 네트워크 설정 시 아래와 같이 한다. 자주 사용하는 정적 네트워크 예를 들어서...

```
# vi /etc/netplan/01-netcfg.yaml
network:
  version: 2
  renderer: networkd
  ethernets:
    ens33:
      dhcp4: no
      addresses:
        - 192.168.10.100/24
      gateway4: 192.168.10.1
```

# 명령어

작성 후, 시스템에 다음처럼 적용한다.

```
nameservers:  
  addresses:  
    - 8.8.8.8  
gateway4: 192.168.10.1  
nameservers:  
  addresses:  
    - 8.8.8.8  
# netplan apply  
# netplan try
```

# 명령어

브릿지를 구성하는 경우 아래와 같이 작성한다.

```
# vi /etc/netplan/02-bridge.yaml
network:
  version: 2
  renderer: networkd
  ethernets:
    ens33:
      dhcp4: no
  bridges:
    br0:
      interfaces: [ens33]
      addresses:
        - 192.168.10.200/24
```

# 명령어

브릿지를 구성하는 경우 아래와 같이 작성한다.

```
gateway4: 192.168.10.1
nameservers:
  addresses:
    - 8.8.8.8
    - 192.168.10.1
parameters:
  stp: false          # 필요시 true
  forward-delay: 0
dhcp4: no
# netplan apply
# ip addr show br0
```

# 새로운 방화벽 시스템

firewalld

nftables



# nftables

방화벽

# nftables

기존에 사용하던 **iptables**는 대용량의 컨테이너 및 가상머신 운영에는 적절하지 않았다. 그래서 기존에 사용하던 **{ip, ip6, arp, eb}tables** 명령어를 새로운 **in-kernel packet framework**로 변경하였다.

**새로운 테이블(nftables)** 프로그램은 기존에 사용하던 **Netfilter**의 정책도 호환이 가능하며, 인프라에 구성된 NAT와 그리고 사용자 영역의 큐잉(queueing)과 로그 기능을 하위 시스템으로 제공한다.

이전 iptables(text file)와 다른 부분은 nftables(JSON)기반으로 정책파일을 관리한다. 그래서 이전 테이블보다 빠르게 입출력 및 검색이 가능하다.

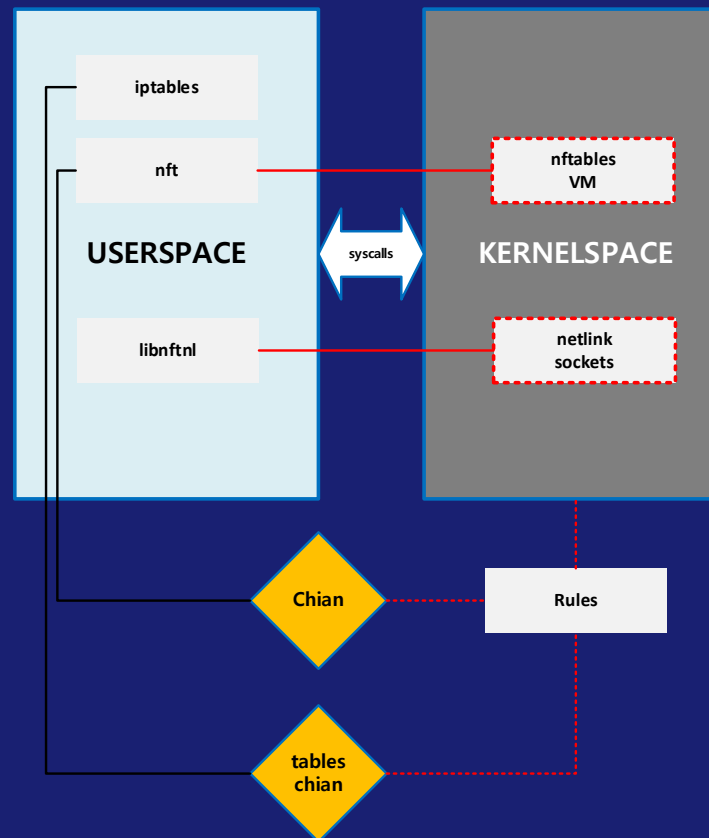
# nftables

nftables는 VM(Virtual Machine)를 가지고 있다. 자바의 JVM과 마찬가지로 nft-vm를 가지고 있다. 사용자가 선언한 내용은 바이트 코드 형태로 컴파일이 되며, 이 기반으로 netlink를 구성 및 생성한다.

nft는 Netlink API를 사용하여, 이를 기반으로 커널에 작업을 수행한다.

결론적으로는, nft는 컴파일러 및 디 컴파일러가 있으며, 데이터는 JSON기반으로 되어 있다.

# nftables



# nftables

여전히 호환성 모드로 iptables 사용은 가능하나 가급적이면 nftables 기반으로 작업 권장.

```
# yum install nftables
# systemctl enable --now nftables.service
# dnf install iptables-services iptables-legacy
# systemctl start iptables
# systemctl is-active iptables nftables firewalld
```

# nftables

정책 출력은 아래 명령어로 가능하다.

```
# nft list tables ip  
# nft list tables  
# nft list counters
```

특정 아이피 드롭.

```
# nft add rule ip filter OUTPUT ip daddr 1.2.3.4 drop
```

# nftables

정책을 추가하는 명령어는 다음과 같이 사용한다. 예를 들어서 특정 아이피 드랍을 원하는 경우, 아래와 같이 사용한다.

```
# nft add rule ip filter OUTPUT ip daddr 1.2.3.4 drop
```

테이블 생성은 다음과 같이 한다. 같은 이름으로 테이블을 생성하지만, 사용하는 필터 위치가 다르기 때문에 중복이 되지 않는다.

```
# nft add table inet base_table  
# nft add table arp base_table
```

목록 출력은 아래와 같은 명령어로 가능하다.

```
# nft list tables  
# nft list tables inet
```

# nftables

테이블 제거를 하기 위해서는 아래와 같이 명령어를 입력한다.

```
# nft delete table inet base_table
```

체인을 테이블에 추가하기 위해서 다음과 같이 명령어를 입력한다.

```
# nft add chain inet base_table input_filter "{type filter hook input  
priority 0;}"  
# nft -a list table inet base_table
```

80/TCP 포트를 막기 위해서 다음과 같이 명령어를 사용한다.

```
# nft add rule inet base_table input_filter tcp dport 80 drop
```



# nftables

모든 rules 내용을 확인하기 위해서는 다음과 같이 명령어를 사용한다.

```
# nft -a list ruleset
```

핸들러 번호로 제거하기 위해서는 다음과 같이 명령어를 사용한다.

```
# nft delete rule inet base_table input_filter handle 3
```


# nftables rules

규칙(rule)은 일괄적으로 적용하기 위해서는 JSON형태로 파일을 작성 후, nft 명령어로 밀어 넣으면 된다. nft는 "#!"를 지원하기 때문에 아래와 같이 작성 및 사용이 가능하다.

# nftables rules

위의 내용 계속 이어서...

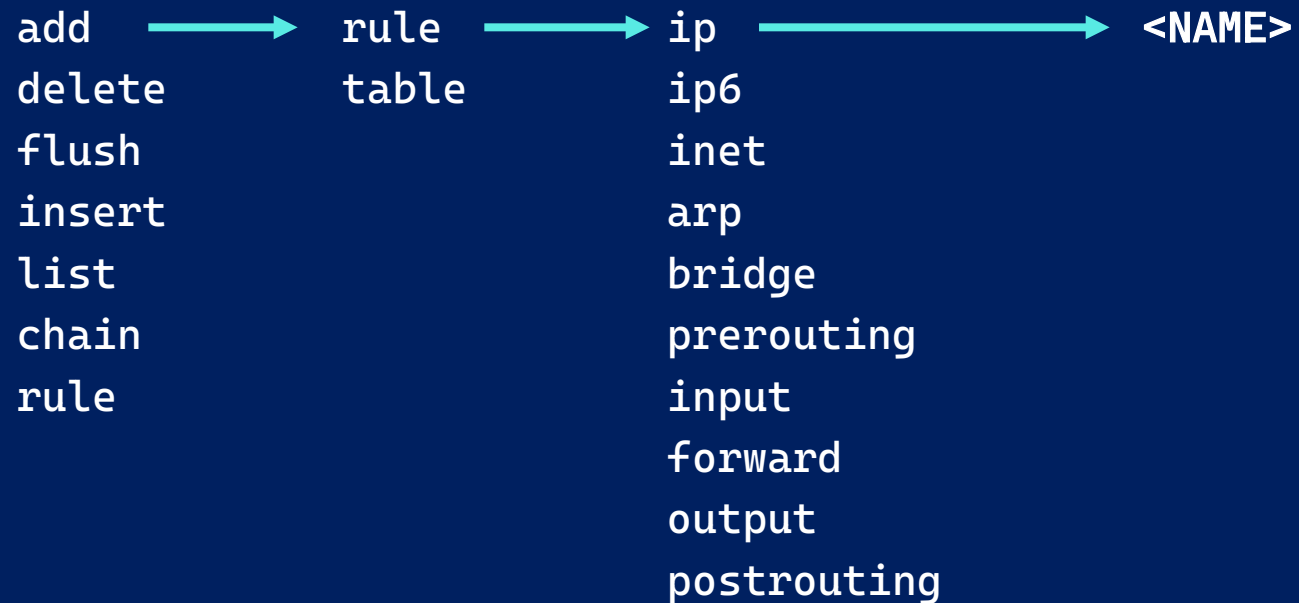
```
# vi httpd_service.sh
#! /sbin/nft -f
define http_ports = {80, 443}
flush ruleset
table inet local {
    chain input {
        type filter hook input priority 0; policy drop;
        tcp dport $http_ports counter accept comment "incoming http traffic";
    }
    chain output {
        type filter hook output priority 0; policy drop;
    }
}
# nft -a list ruleset
```



# nftables

명령어 사용 방법은 다음과 같다.

## nft



# nftables

대소문자 구별 합니다!

특정 아이피 드롭, 하지만 카운팅 모듈 사용.

```
# nft add rule ip filter OUTPUT ip daddr 1.2.3.4 counter drop
```

특정 아이피로 나가는 아이피 대역에 대한 카운팅.

```
# nft add rule ip filter OUTPUT ip daddr 192.168.1.0/24 counter
```

특정 포트번호에 대한 패킷 드랍.

```
# nft add rule ip filter INPUT tcp dport 80 drop
```

# nftables

nftables의 family(protocol)은 다음과 같다.

ip	IPv4프로토콜 정책
arp	Address Resolution Protocol 제어 체인
ip6	IPv6프로토콜 정책
bridge	Linux Bridge 및 OVS Bridge
inet	일반적으로 많이 사용하는 애플리케이션 포트 정책
netdev	Container 및 VirtualMachine에서 사용하는 TAP(Test Access Point) 장치

자세한 설명은 <https://lwn.net/Articles/631372/> 참고.

# nftables

명령어 사용 예제. 아래 명령어가 제일 많이 사용하는 명령어 중 하나이다. 아래 명령어로 간단하게 시도한다.  
ICMP 입력을 허용 합니다.

```
# nft add rule filter INPUT icmp type-echo-request accept
```

1.2.3.4로 나가는 트래픽에 대해서 거절 합니다.

```
# nft add rule ip filter OUTPUT ip protocol icmp ip daddr 1.2.3.4 counter  
drop
```

filter 리스트의 테이블을 확인 합니다.

```
# nft -a list table filter
```

# nftables

특정 rule을 제거한다. 번호를 꼭 넣어 주어야 한다.

```
# nft delete rule filter OUTPUT handle <NUMBER>
```

모든 rule를 메모리에서 비웁니다.

```
# nft flush table filter
```

filter테이블에 80/TCP포트를 허용 합니다.

```
# nft insert rule filter INPUT tcp dport 80 counter accept
```



# firewalld

방화벽

# firewalld

기존에 사용하는 `iptables`, `nftables` 위에 고급 계층을 올려서 사용자가 쉽게 사용할 수 있도록 한다. 현 레드햇 계열 배포판은 더 이상 `iptables`를 사용하지 않는다. 또한, `firewalld`도 `iptables`를 지원하지 않는다.

**firewall-cmd** 명령어 기반으로 영역(zone), 서비스(service), 포트(port)와 같은 구성 요소를 XML기반으로 선언이 가능하다. 사용자가 원하는 경우, 추가적으로 XML파일을 만들어서 추가가 가능하다.

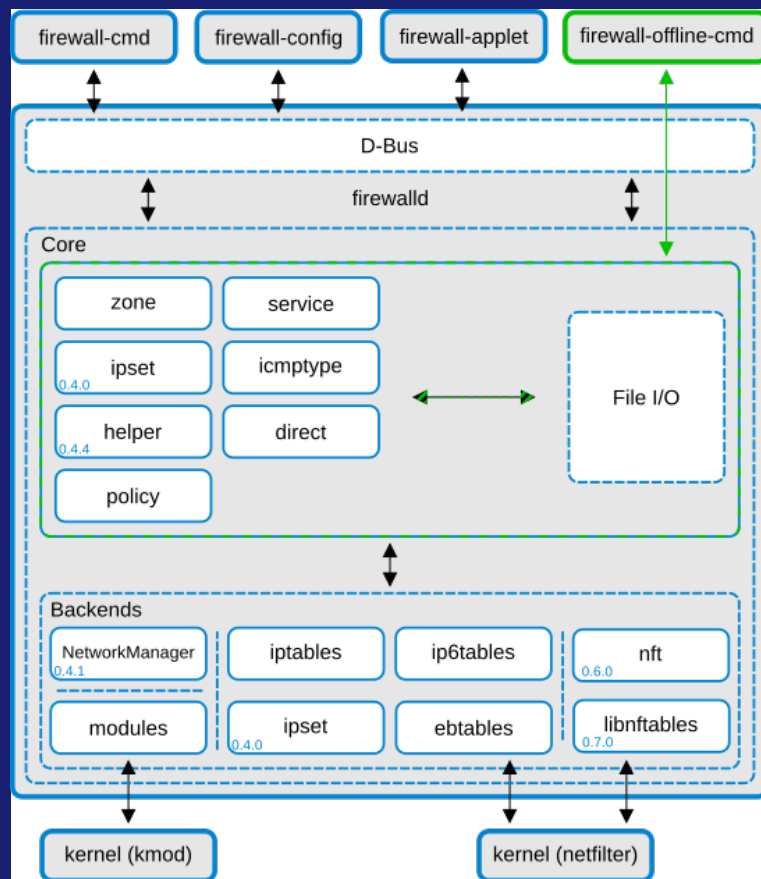
레드햇 계열은 RHEL 9 이후부터 `nft/firewalld`만 사용이 가능하다.

# zone

firewalld에서 제공하는 기본 존 영역은 다음과 같다.

Zone 이름	설명
block.xml	모든 연결을 차단하고, 허용된 연결만 통과시킴
dmz.xml	비무장 지대, 공개 서버
drop.xml	모든 패킷을 드랍 (ICMP 응답도 안 함)
external.xml	외부 네트워크와 내부 네트워크 사이에 사용하는 경우, IP 마스커레이딩 사용
home.xml	가정용 네트워크
internal.xml	내부 네트워크용
public.xml	공용 네트워크용, 기본 값
trusted.xml	모든 연결을 신뢰 (거의 모든 트래픽 허용)
work.xml	사무실/업무 네트워크용, 중간 정도 신뢰

# 방화벽 아키텍처



# firewalld

firewalld의 주요 정보는 아래 디렉터리에 XML파일 형식으로 존재한다.

```
# systemctl start firewalld
# ls -l /lib/firewalld
# cd /lib/firewalld/services
...
http.xml
...
# ls -ld /etc/firewalld/services
# ls -ld /etc/firewalld/zones
```

# firewalld

존 목록을 확인하는 방법.

```
# firewall-cmd --get-zones
# firewall-cmd --list-all --zone=
```

특정 존 아이피/포트/서비스 및 기본 존 변경.

```
# firewall-cmd --add-source=10.10.10.0/24 --zone=block --permanent
# firewall-cmd --add-service=https --zone=drop
# firewall-cmd --add-service=http
# firewall-cmd --add-port=8899/tcp
# firewall-cmd --set-default-zone=block
# firewall-cmd --get-default-zone
```

# firewalld

서비스 목록을 확인하기 위해서 다음과 같이 실행.

```
# firewall-cmd --get-services
```

```
RH-Satellite-6 RH-Satellite-6-capsule afp amanda-client amanda-k5-client  
amqp amqps apcupsd audit ausweisapp2 bacula bacula-client bareos-director  
bareos-filedaemon bareos-storage bb bgp bitcoin bitcoin-rpc bitcoin-testnet  
bitcoin-testnet-rpc bittorrent-lsd ceph ceph-exporter ceph-mon cfengine  
checkmk-agent cockpit collectd condor-collector cratedb ctdb dds dds-  
multicast dds-unicast dhcp dhcpv6
```

# firewalld

runtime 정보를 permanent로 저장하기 위해서 다음과 같이 실행.

```
# firewall-cmd --runtime-to-permanent
```



# 프로세스

관리도구

# 관리도구

프로세스

# systemd

systemd 기반에서는 `systemctl`, `whoami`를 통해서 특정 프로세서가 어떠한 유닛에서 사용 중인지 확인이 가능하다. 이 부분에 대해서 위에서 언급하였기 때문에 직접 다루지는 않는다.

이외 나머지 부분은 유닛 제어 명령어를 통해서 제어가 관리하다. 이전 System V와 다르게 가급적이면 `systemctl` 명령어로 관리를 권장한다.

# ps

리눅스는 총 3가지 스타일로 제공한다.

1. BSD

2. UNIX

3. GNU

사람마다 많이 다르기는 하는데 일반적으로 유닉스/BSD 옵션으로 많이 사용한다. GNU 형식은 문자열이 상대적으로 길어서 잘 사용하지 않는다.

- 1 UNIX options, which may be grouped and must be preceded by a dash.
- 2 BSD options, which may be grouped and must not be used with a dash.
- 3 GNU long options, which are preceded by two dashes.

# ps

ps 명령어는 제일 기본적으로 사용하는 프로세스 확인 명령어. 제일 많이 유닉스 형태의 명령어는 다음과 같다.

```
# ps -ef
```

아래 명령어는 BSD 스타일로 많이 사용하는 옵션이다.

```
# ps aux
```

위의 명령어는 GNU 스타일의 명령어. 일반적으로 엔지니어들이 많이 사용하는 스타일은 dash(-) 스타일이다.

```
# ps efww --pid 801
```

# ps

1. a 옵션은 BSD 스타일의 only yourself 옵션
2. u 옵션은 EUID(effective user ID)
3. x 옵션은 x BSD스타일의 must have a tty 옵션

# ps aux or ps au

```
[root@localhost ~]# ps -aux
USER      PID  %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   0.0  0.7 128144  6828 ?        Ss   Sep11   0:02 /usr/lib/systemd/systemd
root         2   0.0  0.0      0     0 ?        S    Sep11   0:00 [kthreadd]
root         4   0.0  0.0      0     0 ?        S<   Sep11   0:00 [kworker/0:0H]
root         5   0.0  0.0      0     0 ?        S    Sep11   0:00 [kworker/u32:0]
root         6   0.0  0.0      0     0 ?        S    Sep11   0:00 [ksoftirqd/0]
root         7   0.0  0.0      0     0 ?        S    Sep11   0:00 [migration/0]
root         8   0.0  0.0      0     0 ?        S    Sep11   0:00 [rcu_bh]
root         9   0.0  0.0      0     0 ?        R    Sep11   0:36 [rcu_sched]
root        10   0.0  0.0      0     0 ?        S<   Sep11   0:00 [lru-add-drain]
root        11   0.0  0.0      0     0 ?        S    Sep11   0:00 [watchdog/0]
```

# ps

1. -f 옵션은 full-formatting list
2. -e 옵션은 all processes

위의 명령어는 Unix(AIX) 스타일의 명령어.

```
# ps -ef
```

```
[root@localhost ~]# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  0 Sep11 ?        00:00:02 /usr/lib/systemd/systemd --switched-root --system --deserializ
root           2        0  0 Sep11 ?        00:00:00 [kthreadd]
root           4        2  0 Sep11 ?        00:00:00 [kworker/0:0H]
root           5        2  0 Sep11 ?        00:00:00 [kworker/u32:0]
root           6        2  0 Sep11 ?        00:00:00 [ksoftirqd/0]
root           7        2  0 Sep11 ?        00:00:00 [migration/0]
root           8        2  0 Sep11 ?        00:00:00 [rcu_bh]
root           9        2  0 Sep11 ?        00:00:36 [rcu_sched]
root          10        2  0 Sep11 ?        00:00:00 [lru-add-drain]
```

# ps

BSD 스타일로 TTY에 이름도 같이 출력한다.

```
# ps -x
```

```
[tang@www ~]$ ps -x
  PID TTY          STAT       TIME COMMAND
 69492 ?            Ss          0:00 /usr/lib/systemd/systemd --user
 69496 ?            S           0:00 (sd-pam)
 79289 ?            Ss          0:05 tmux
 79290 pts/15        Ss          0:00 -sh
 79320 pts/21        Ss          0:00 -sh
 79350 pts/22        Ss          0:00 -sh
 80825 pts/15        S+          0:00 ssh root@192.168.90.171
 80828 pts/15        S+          0:00 /usr/bin/sss_ssh_knownhostsproxy -p 22 192.168.90.171
 80861 pts/21        S+          0:00 ssh root@192.168.90.3
 80864 pts/21        S+          0:00 /usr/bin/sss_ssh_knownhostsproxy -p 22 192.168.90.3
 80865 pts/22        S+          0:00 ssh root@192.168.90.168
 80868 pts/22        S+          0:00 /usr/bin/sss_ssh_knownhostsproxy -p 22 192.168.90.168
```



# ps

특정 사용자가 사용하는 프로세스를 출력한다. U는 명시한 사용자 프로세스만 출력한다.

이름	설명
-u	Effective User ID
-U	Real user ID

```
# ps -fu or -fU <UID> <UNAME>
```

```
# ps -fU tang
```

```
[tang@www ~]$ ps -fU tang
UID      PID      PPID  C  STIME TTY          TIME CMD
tang     69492        1  0  Sep06 ?        00:00:00 /usr/lib/systemd/systemd --user
tang     69496    69492  0  Sep06 ?        00:00:00 (sd-pam)
tang     79289        1  0  Sep06 ?        00:00:05 tmux
tang     79290    79289  0  Sep06 pts/15   00:00:00 -sh
tang     79320    79289  0  Sep06 pts/21   00:00:00 -sh
tang     79350    79289  0  Sep06 pts/22   00:00:00 -sh
```

# ps

**-u**: 영향 받는 사용자 아이디(RUID)

**-U**: 실제 사용자 아이디(EUID)

```
# ps -U root -u root
```

```
[tang@www ~]$ ps -u tang -U tang
  PID TTY          TIME CMD
 69492 ?            00:00:00 systemd
 69496 ?            00:00:00 (sd-pam)
 79289 ?            00:00:05 tmux: server
 79290 pts/15      00:00:00 sh
 79320 pts/21      00:00:00 sh
 79350 pts/22      00:00:00 sh
 80825 pts/15      00:00:00 ssh
 80828 pts/15      00:00:00 sss_ssh_knownho
```

# ps

일반 포매팅 및 확장 포매팅 활성화 그룹 이름 혹은 아이디로 조회한다.

```
# ps -Fg tang
```

```
# ps -fG 1000
```

```
[tang@www ~]$ ps -Fg tang
```

UID	PID	PPID	C	SZ	RSS	PSR	STIME	TTY
tang	69492	1	0	22388	9528	4	Sep06	?
tang	69496	69492	0	81902	3320	0	Sep06	?
tang	79289	1	0	7246	4528	5	Sep06	?
tang	79290	79289	0	6929	5140	3	Sep06	pts/15
tang	79320	79289	0	6929	5012	5	Sep06	pts/21
tang	79350	79289	0	6929	4988	3	Sep06	pts/22
tang	80825	79290	0	15448	6812	6	Sep07	pts/15
tang	80828	80825	0	24797	5588	6	Sep07	pts/15
tang	80861	79320	0	15448	6920	3	Sep07	pts/21

# ps

명시된 특정 프로세서만 출력한다.

```
# ps -fp
```

```
[tang@www ~]$ sudo ps -fp 1
UID          PID    PPID  C STIME TTY          TIME CMD
root           1         0  0 Sep03 ?        00:01:30 /usr/li
```

# ps

특정 부모 프로세서(parent process id)에 대해서 조회한다.

```
# ps -f --ppid 1
```

```
[tang@www ~]$ sudo ps -f --ppid 1
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	751	1	0	Sep03	?	00:00:11	/usr/lib/systemd/systemd-journald
root	799	1	0	Sep03	?	00:00:06	/usr/lib/systemd/systemd-udevd
rpc	899	1	0	Sep03	?	00:00:03	/usr/bin/rpcbind -w -f
root	908	1	0	Sep03	?	00:00:04	/sbin/auditd

# kill/killall/pkill

- 1, HUP, reload a process → **systemctl reload sshd.service**
- 9, KILL, kill a process without page down
- 15, TERM, gracefully stop a process → **systemctl stop**



# pgrep/pkill

pgrep은 프로세스의 이름 혹은 다른 속성을 확인하여 종료 신호 전달.

```
# pkill <signal> <process_name>
# pgrep -u root sshd
# pgrep -u tang,root,daemon
```

# pkill/pgrep

ssh 프로세서 이름의 개수 확인.

```
# pgrep -c ssh  
10
```

```
# pgrep -d "-" ssh  
1363-30317-79439-79442-79459-79462-80023-80026-80047-80050-80051-80054-  
80825
```



# pgrep

사용자 **tang**이 사용중인 **ssh**으로 사용하는 프로세스 아이디가 출력된다.

```
# pgrep -u tang ssh  
80825  
80828  
80861  
80864  
80865  
80868  
131706
```

# pgrep

특정 사용자가 사용하는 프로세서의 개수 및 프로세스 목록을 출력한다.

```
# pgrep -u tang ssh -c
12
# pgrep -l ssh
1363 sssd_ssh
30317 sshd
79439 ssh
79442 sss_ssh_knownho
79459 ssh
79462 sss_ssh_knownho
```

# pgrep

특정 프로세스를 확인하기 위해서 **pgrep** 명령어 통해서 가능하다. 일반적으로 'ps -ef | grep' 조합을 많이 사용하지만, 권장은 **pgrep** 사용을 권장한다.

```
# pgrep -a ssh
79439 ssh root@192.168.90.178
79442 /usr/bin/sss_ssh_knownhostsproxy -p 22 192.168.90.178
79459 ssh root@192.168.90.187
79462 /usr/bin/sss_ssh_knownhostsproxy -p 22 192.168.90.187
80023 ssh root@192.168.90.183
80026 /usr/bin/sss_ssh_knownhostsproxy -p 22 192.168.90.183
131706 sshd: tang@pts/0
```

# pgrep

1. ssh와 일치하지 않는 프로세스만 출력.
2. 정확하게 sshd이름과 일치하는 프로세스만 출력.
3. 최근에 생성된 ssh process만 출력.
4. 이전에 생성된 ssh process만 출력.

```
# pgrep -v ssh  
# pgrep -x sshd  
# pgrep -n ssh  
# pgrep -o ssh
```

# kill/pgrep

특정 사용자 혹은 프로그램의 PID 정보를 확인하기 위해서 아래와 같이 명령어를 실행한다. 사용자 종료 경우에는 앞서 학습한 `loginsctl` 명령어를 사용해서 사용자 제어 및 관리한다.

절대로 9번 시그널은 사용중인 프로세서에서 사용하지 마세요!

```
# kill '^ssh$'
# kill -9 -f "ping 8.8.8.8"
# kill -U mark
# kill -U mark gnome -9
# kill -9 -n screen
```

# 파일 시스템 퍼미션

POSIX/표준 퍼미션

# 설명

파일 시스템 퍼미션은 **ALC 및 표준 퍼미션** 이외 확장 퍼미션 이다. ACL 및 표준 퍼미션은 주로 사용자 및 그룹 위주로 구성이 되어있다.

이를 **속성 퍼미션(attribute permission)**이라고 한다. 이 정보는 슈퍼 블록에 저장되지 않으며, 해당 정보들은 ACL과 동일하게 inode 영역에 저장이 된다.

항목	저장 위치	설명
chmod (퍼미션)	inode 내부	Inode 구조체 안에 바로 저장 (rwx 설정 등)
acl (액세스 제어 목록)	inode + 별도 xattr 블록	inode에 ACL 존재 여부 기록, 실제 ACL 데이터는 xattr 공간에 저장
attr (확장 속성)	inode + 별도 xattr 블록	inode에 확장 속성 존재 여부 기록, 실제 attr 데이터는 xattr 공간에 저장
superblock	파일 시스템 전체 메타 정보	파일 시스템 타입, 크기, inode 수 등 관리 (파일 단위 속성 저장은 아님)

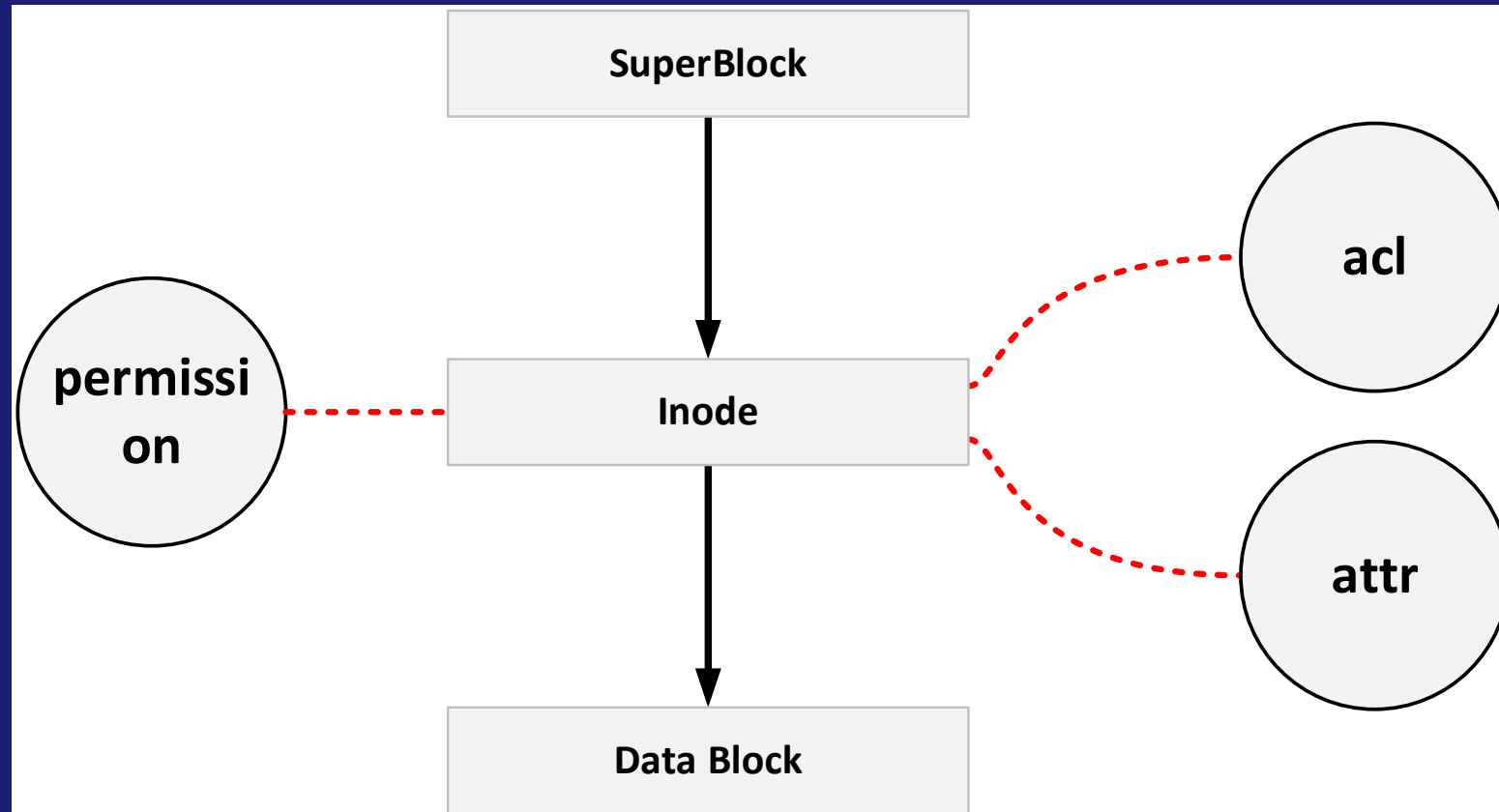
# 설명

사용이 가능한 파일 시스템은 다음과 같다.

파일 시스템	지원 여부	비고
ext4	0	리눅스 표준 파일 시스템
xfs	0	대용량, 고성능
btrfs	0	최신, 스냅샷 지원
f2fs	0	플래시 메모리용
ntfs-3g (Linux에서)	제한적 지원	mount 옵션 필요 (user_xattr)
tmpfs	0	메모리 파일 시스템도 지원
nfs (버전 4 이상)	0	네트워크 파일 시스템



# 설명



# 설명

attr은 다음과 같은 명령어를 지원한다.

1. setattr

2. getattr

setattr은 파일 혹은 디렉터리에 적용이 가능하다. 보통 디렉터리 기반보다는 파일에 하나씩 적용이 많이 한다. 예를 들어서 특정 파일 혹은 디렉터리에 쓰기 및 삭제를 방지하기 위해서 다음과 같이 설정이 가능하다.

# 설명

setattr에서 사용이 가능한 옵션은 다음과 같다.

속성	이름	의미	설명
a	Append Only	추가만 허용	기존 데이터 수정 불가, 추가만 가능
c	Compressed	압축 저장	파일을 디스크에 압축된 형태로 저장
d	No Dump	덤프 제외	dump 명령어가 파일 무시
e	Extents	ext4 extents 사용	ext4에서 파일 저장 최적화(자동 적용)
i	Immutable	불변	파일 수정, 삭제, 이름 변경 모두 불가
j	Data Journaling	데이터 저널링	ext3 저널에 데이터까지 기록
s	Secure Deletion	보안 삭제	삭제 시 데이터 덮어쓰기 (실제 구현 X)
t	No Tail-Merge	tail-merging 금지	리서스 사용 최적화 막음

# 설명

앞에 내용 계속 이어서...

속성	이름	의미	설명
u	Undeleteable (undelete)	복구 가능 삭제	삭제 후 복구를 시도 가능 (미지원)
A	No Atime Updates	atime 기록 비활성화	파일 접근 시 atime 업데이트 안 함
C	No Copy-on-Write (CoW)	Btrfs CoW 끄기	snapshot/CoW 끄기 (btrfs 전용)
D	Synchronous Directory Updates	디렉터리 동기화	디렉터리 수정시 바로 디스크 기록
S	Synchronous Updates	파일 동기화 쓰기	매 쓰기(write) 마다 디스크 반영
T	Top of Directory Hierarchy	최상위 디렉터리 표시	ext4 디렉터리 최적화 힌트

# 설명

제일 많이 사용하는 옵션은 아래와 같다.

플래그	왜 중요?
+i (Immutable)	파일/디렉터리 완전 잠금
+a (Append Only)	로그 파일 보호 (추가만 허용)
+A (No Atime Updates)	성능 최적화 (SSD 등)

# 삭제 및 쓰기 방지

아래와 같이 명령어를 실행한다.

```
# echo "오징어땅콩" > important.txt
# chattr +i important.txt
# lsattr important.txt
——i—— important.txt
# echo "추가 데이터" >> important.txt
bash: important.txt: Permission denied
# chattr -i important.txt
# lsattr important.txt
———— important.txt
```

# 삭제 및 쓰기 방지

위의 내용 계속 이어서...

```
# chattr +a important.txt
# lsattr important.txt
-----a----- important.txt
# echo "추가 내용" >> important.txt      # (OK)
# echo "수정" > important.txt            # (Permission denied)
```

# 패키지 관리 방법

dnf/module

repository mirror



# DNF

패키지 관리

# dnf

dnf는(은) 기존에 사용하던 yum 명령어를 대체하는 새로운 패키지 관리자. dnf에서 제일 큰 차이점은 바로 **modules**라는 기능이 새로 도입이 되었음.

이를 통해서 특정 프로그램 설치 시 의존성이 필요한 경우 **dnf module**를 통해서 패키지 제공이 된다.

또한 이 기능은 이전에 사용하였던 **SCL(Software Collection Library)**와 비슷하게 호스트 영향 없이 확장이 가능하다.

# MODULE

수세 리눅스에도 모듈이라는 개념이 있다. 이 개념은 레드햇과 비슷하지만 다른 부분이 있다.

항목	수세(SUSE) 모듈	레드햇(RedHat) DNF 모듈 스트림
개념	기능 그룹별 저장소(repository) 묶음	하나의 소프트웨어를 여러 버전(stream)으로 관리
적용 위치	저장소 레벨 (repo enable/disable)	패키지 레벨 (버전 stream 선택/설치)
예시	Web and Scripting Module Legacy Module	nodejs:10, nodejs:14
주요 제품	SLES 15, openSUSE Leap	RHEL 8, Fedora 28+
도입 시기	SLES 15 (2018)	RHEL 8 (2019)
기술 주체	SUSE	Red Hat

# MODULE

모듈 목록을 아래에서 확인이 가능하다.

<https://gitlab.com/redhat/centos-stream/modules/>

모듈을 확장하기 위해서 다음과 같이 명령어를 실행한다.

```
# dnf install https://rpms.remirepo.net/enterprise/remi-release-9.rpm -y  
# dnf --enablerepo=remi-modular --disablerepo=appstream module list -y
```

# module

```
# dnf module list
```

```
CentOS Stream 8 - AppStream
```

Name	Stream	Profiles Summary
container-tools	rhel8 [d][e]	common

# rollback

yum, dnf는 rollback기능을 제공한다.

```
dnf history      rollback
                  undo
                  redo
                  list
                  info
                  install
                  remove
                  update/upgrade
```

# yum vs dnf-3

yum과 dnf의 제일 큰 차이점은 yum은 순수하게 파이선으로 작성이 되어 있다.

하지만, dnf는 libdnf3라는 C 언어로 재-구성이 되었다. 이를 통해서 기존에 yum에서 불편했던 느린 반응 및 파이썬 라이브러리 문제 발생을 방지 할 수 있다.

현재 대다수 RPM 및 레드햇 기반의 배포판은 YUM → DNF-3로 변경이 된 상태이다.

# RPM commands

**RPM** 명령어는 기존에는 **rpm**이라는 하나의 명령어만 사용하였다. RPM은 기능이 확장 되면서, 명령어 분리하기 시작. 아래처럼 분리가 되었다.

- **rpm** 기본 설치 명령어. 일반적으로 패키지 추가/삭제/제거 및 확인.
- **rpm2archive** ".rpm"패키지를 ".tgz"로 변경한다.
- **rpm2cpio** ".rpm"에서 cpio묶여 있는 파일을 푸는 명령어.
- **rpmdb** RPM의 BerkelyDB 조회 시 사용하는 명령어.
- **rpmkeys** RPM에서 사용하는 공개키 관리.
- **rpmquery** RPM에서 사용하는 시그 키 관리.
- **rpmverify** RPM에 등록된 패키지의 자원들에 대한 의존성 검사.



# RPM commands

일반적으로 rpm 명령어를 통해서 패키지 관리 및 검사가 가능하다. 아래는 대표적으로 많이 사용하는 명령어이다.

```
rpm -ql <PACKAGE_NAME>
rpm -qa --last <PACKAGE_NAME>
rpm -q <PACKAGE_NAME>
rpm -qf <FILENAME>
rpm -ivh <PACKAGE_NAME>
rpm -ev <PACKAGE_NAME>
rpm -qi <PACKAGE_NAME>
rpm -Vp <PACKAGE_NAME>
rpm -Va <PACKAGE_NAME>
rpm -qa gpg-pubkey*
rpm -q --scripts systemd
rpm -q --triggers systemd
```

# yum/dnf

앞서 이야기 하였지만, YUM 명령어는 DNF-3로 변경이 되었음. 저장소의 모든 RPM 파일을 받고, 저장소를 구성하기 위해서는 다음과 같은 명령어로 구성이 가능함.

```
# dnf reposync
# dnf install createrepo_c
# createrepo_c .
```

위의 명령어로 구성이 완료가 되면, Apache, Nginx 기반으로 저장소를 구성하면 됨. 예제는 아래와 같이 진행한다.

# 미러링

패키지 관리

# mirror

이전에 사용하던 미러링 기능도 `dnf` 명령어로 통합이 되었다. 저장소 미러를 받기 위해서 다음과 같이 명령어를 수행한다.

```
# mkdir -p /mnt/reposdisk/rpms
# dnf reposync -p /mnt/repodisk/rpms
# createrepos_c /mnt/repodisk/rpms .
# dnf install httpd -y
# mount -obind /mnt/repodisk/rpms /var/www/html/
# vi /etc/yum.repos.d/mirros.repo
[poweer]
name=internal repo
baseurl=<IP>
gpgcheck=0
enable=1
```

# APT

패키지 관리

# APT

APT는 Advanced Package Tool의 약자. 보통 APT는 DEB 기반에서 사용한다. 제공하는 주요 기능은 다음과 같다.

1. 패키지 설치
2. 패키지 업그레이드
3. 패키지 제거
4. 패키지 검색

이를 사용하기 위해서 명령어는 다음과 같다.

1. apt
2. apt-get
3. apt-cache

# APT

제일 많이 사용하는 명령어는 다음과 같다.

명령어	설명
apt update	저장소에서 패키지 목록(인덱스) 업데이트
apt upgrade	설치된 모든 패키지를 최신 버전으로 업그레이드
apt install 패키지명	특정 패키지 설치
apt remove 패키지명	특정 패키지 삭제
apt search 패키지명	패키지 검색
apt show 패키지명	패키지 상세 정보 확인
apt autoremove	필요 없는 패키지 자동 제거

# 명령어

패키지 설치 및 검색은 다음과 같다.

```
# apt search htop
Sorting ... Done
Full Text Search ... Done
htop/stable 3.0.5-7 amd64
  interactive processes viewer
# apt install htop
Reading package lists ... Done
Building dependency tree
Reading state information ... Done
The following NEW packages will be installed:
  htop
```



# 명령어

패키지 제거는 다음과 같다.

```
# apt remove htop
Reading package lists ... Done
Building dependency tree
Reading state information ... Done
The following packages will be REMOVED:
  htop
```

# 명령어

설치된 패키지 목록 확인은 다음과 같이 실행한다. 혹은, 패키지 정보를 확인하기 위해서 show 명령어로 확인한다.

```
# apt list --installed  
# apt show <PACKAGE>
```

# 명령어

명령어가 어떠한 패키지에 포함이 되었는지 확인하기 위해서 아래와 같이 실행이 가능하다.

```
# apt update
# apt install apt-file
# apt-file update
# apt-file search bin/htop

# dpkg -S $(which htop)
```

# ZYPper

패키지 관리

# 설명

Zypper는 수세 리눅스에서 사용하는 패키지 관리자. Zypper는 레드햇 YUM/DNF와 비슷하게 동작한다. 레드햇 패키지 관리자는 libdnf, libyum과 같은 라이브러리 기반으로 동작한다.

Zypper는 libzypp라는 라이브러리 기반으로 동작한다. 기본적인 기능은 YUM/DNF와 거의 동일하게 지원한다.

# 설명

기능 비교는 다음과 같다.

항목	DNF	Zypper
패키지 포맷	RPM	RPM
주요 백엔드	libdnf, hawkey	libzypp
명령어 구조	dnf install, dnf remove 등	zypper install, zypper remove 등
리포지터리 관리	dnf config-manager 또는 *.repo 편집	zypper addrepo, zypper removerepo 등 직접 명령어 제공
의존성 해결	자동 해결 + 제안(Suggestion)도 출력	자동 해결 + 선택지 제공 (예: 버전 선택, 충돌 해결 대안)
업데이트 방식	dnf update(upgrade), dist-upgrade	zypper update, zypper dist-upgrade (du p)
패치 전용 업데이트	부분적 (dnf updateinfo)	명확함 (zypper patch)

# 설명

위의 내용 계속 이어서...

항목	DNF (Fedora, RHEL, CentOS)	Zypper (openSUSE, SLE)
리포지터리 자동 리프레시	수동 또는 설정 필요	기본적으로 자동 리프레시 (상태 확인도 쉬움)
속도	빠르지만 캐시 클리어 시 다소 느림	초기 리프레시 느릴 수 있으나 전반적으로 빠름
스냅샷 통합	기본적으로 없음	Btrfs 기반 시스템에서 Snapper와 통합
인터랙티브 모드	기본 제공 안 함	zypper shell 로 제공

# 설명

자주 사용하는 옵션은 다음과 같다.

기능	명령어 예시
패키지 검색	zypper search 패키지명
패키지 설치	zypper install 패키지명
패키지 삭제	zypper remove 패키지명
시스템 업데이트	zypper update
리포지터리 목록 보기	zypper repos
리포지터리 추가	zypper addrepo URL 별칭
리포지터리 삭제	zypper removerepo 별칭
리포지터리 새로고침	zypper refresh
설치된 패키지 정보 보기	zypper info 패키지명
잠금(Lock) 걸기 (업데이트 제외)	zypper addlock 패키지명



# 명령어

다음과 같이 명령어 사용이 가능하다. 참고로, `update`는 모든 패키지의 최신화를 수행한다.

```
# zypper search vim  
# zypper install vim  
# zypper remove vim  
# zypper update
```

# 명령어

배포판 업그레이드를 원하는 경우 아래와 같이 실행한다.

```
# zypper dup
Loading repository data ...
Reading installed packages ...

The following 43 packages are going to be upgraded:
  aaa_base bash coreutils glibc kernel-default libzypp openssl ...

The following 2 packages are going to be downgraded:
  systemd systemd-logger

...
```

# 명령어

보안 패치만 원하는 경우 아래와 같이 실행한다. 기본적으로 중요 업데이트만 수행한다.

```
# zypper patch
```

```
Loading repository data ...
```

```
Reading installed packages ...
```

```
The following 5 patches are going to be applied:
```

SUSE-SLE-Update-2025:1234-1	Security update for openssl
SUSE-SLE-Update-2025:1235-1	Recommended update for systemd
SUSE-SLE-Update-2025:1236-1	Security update for glibc
SUSE-SLE-Update-2025:1237-1	Optional update for firewalld
SUSE-SLE-Update-2025:1238-1	Security update for libzypp

# RPM

패키지 관리

# 설명

RPM은 Red Hat Package Manager 혹은 RPM Package Manager라고 부른다. 지금은 후자로 용어를 굳혀가고 있다. (더 이상 Redhat Package Manager가 아니다. 본래 RPM은 레드햇에서 다듬어서 재배포한 프로그램이다.)

초기 RPM은 tar기반으로 압축하는 형태. 점점 **스크립트/바이너리/파일/의존성** 정보가 포함이 되면서 현재 사용하는 RPM으로 완성이 되었다.

RPM이 많이 사용하게 된 이유 중 하나가, **버전관리/의존성 및 검증**이 다른 패키지 보다 편하고 쉽다. 이러한 이유로 대다수 기업용 배포판 수세/레드햇에서 많이 사용한다.

본래, RPM은 레드햇이 만든 것은 아니다. PM이라는 패키지 매니저에서 시작이 되었고, 본래 **MMC interim Linux**에서 사용하였다. 하지만, 레드햇 RHL 2.0에서 도입을 시작하였고, 그리고 이후에 수세/맨드레이크와 같은 리눅스 배포판에서도 RPM를 사용하였다.

# 설명

자주 사용하는 옵션은 다음과 같다.

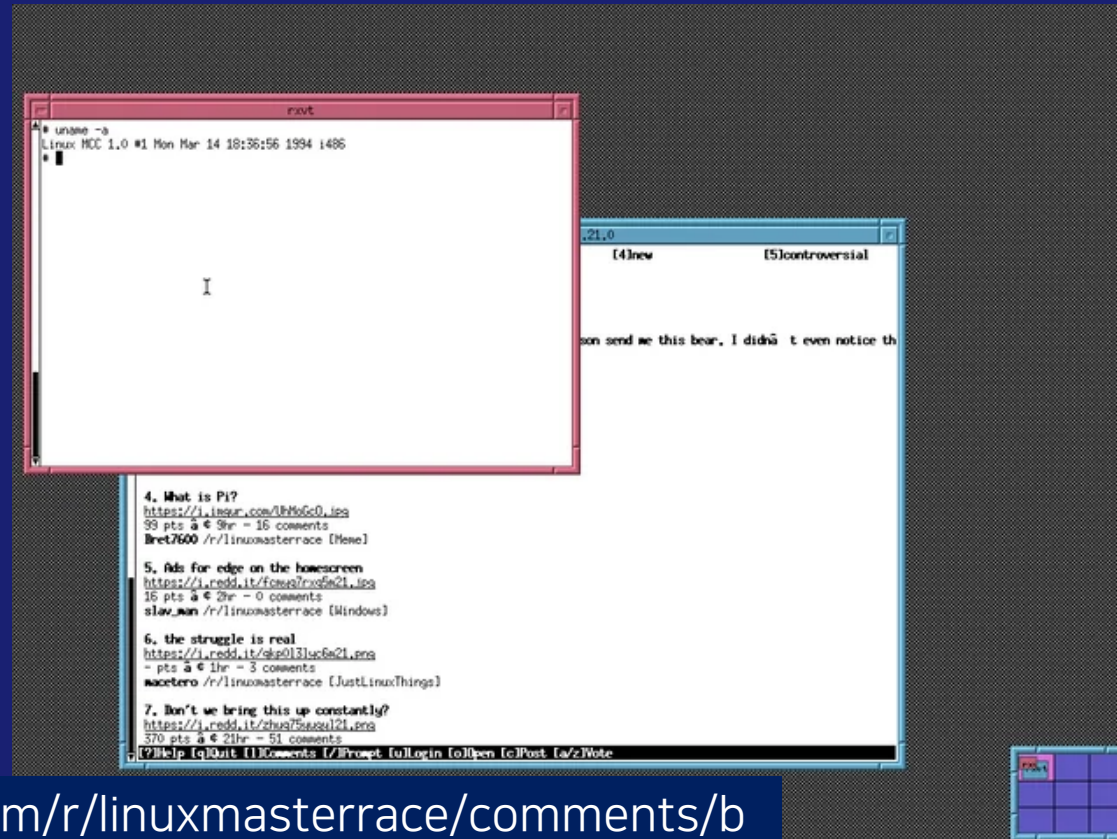
옵션	설명
-i	설치 (install)
-U	업그레이드 (upgrade)
-F	신선하게 업그레이드 (freshen, 설치된 것만)
-e	삭제 (erase)
-q	조회 (query)
-qa	모든 설치된 패키지 조회
-qi	상세정보 조회
-ql	설치 파일 목록

# 설명

자주 사용하는 옵션은 다음과 같다.

옵션	설명
-qc	설정 파일 목록
-qd	문서 파일 목록
-qR	의존성 목록
-V	패키지 검증
-K	서명 검증
--nodeps	의존성 무시
--force	강제 설치

# 설명



[https://www.reddit.com/r/linuxmasterrace/comments/b18u7k/for\\_the\\_25th\\_birthday\\_of\\_kernel\\_10\\_i\\_installed/](https://www.reddit.com/r/linuxmasterrace/comments/b18u7k/for_the_25th_birthday_of_kernel_10_i_installed/)



# 명령어

RPM에서 제일 많이 사용하는 옵션은 다음과 같다.

```
# rpm -ivh test.rpm
# rpm -qa
# rpm -ql test.rpm
# rpm -qd test.rpm
# rpm -qi test.rpm
# rpm -qc test.rpm
# rpm -v test.rpm
# rpm -i --nodeps
# rpm -i --force
```

# DEB

패키지 관리

# 설명

DEB는 Debian Software package에서 가져온 단어.

데비안은 RPM처럼 DEB 패키지만, **dpkg** 명령어로 설치 및 관리가 가능하다. DEB 패키지는 최대한 사용자가 사용하기 편하도록 여러 가지 기능을 제공한다. 예를 들어서 dpkg-reconfigure와 같은 명령어로 재설정이 가능하도록 한다.

다만, **DEB**는 초기부터 의존성 문제가 있었고, **.deb**에서도 의존성 정보를 제공하지만, 완벽하지 않았다. 이러한 이유로 오랫동안 버그가 아닌, 버그 형태로 버그목록에 많이 제출이 되었다.

이 문제는 APT를 통해서 해결이 되었다. **.deb** 패키지 더 이상 APT를 통하지 않고 모든 패키지에 대해서 의존성 확인이 가능하다.

# 설명

제일 많이 사용하는 옵션은 다음과 같다. 데비안 계열은 대다수가 sudo 명령어 기반으로 실행한다.

명령어	설명	예시
<code>dpkg -i</code>	.deb 파일 설치 (install)	<code>sudo dpkg -i package.deb</code>
<code>dpkg -r</code>	설치된 패키지 제거 (remove)	<code>sudo dpkg -r package_name</code>
<code>dpkg -P</code>	설정 파일까지 완전 제거 (purge)	<code>sudo dpkg -P package_name</code>
<code>dpkg -l</code>	설치된 패키지 리스트 보기	<code>dpkg -l</code>
<code>dpkg -L</code>	패키지 설치 파일 목록 보기	<code>dpkg -L package_name</code>
<code>dpkg -S</code>	파일이 어떤 패키지에 속하는지 검색	<code>dpkg -S /usr/bin/bash</code>
<code>dpkg -I</code>	.deb 파일 정보 보기 (대문자 i)	<code>dpkg -I package.deb</code>
<code>dpkg -c</code>	.deb 파일 내부 파일 목록 보기	<code>dpkg -c package.deb</code>

# 명령어

패키지 설치.

```
# dpkg -i cowsay_3.03+dfsg2-7_all.deb
Selecting previously unselected package cowsay.
(Reading database ... 235672 files and directories currently installed.)
Preparing to unpack cowsay_3.03+dfsg2-7_all.deb ...
Unpacking cowsay (3.03+dfsg2-7) ...
Setting up cowsay (3.03+dfsg2-7) ...
```

# 명령어

패키지 검색.

```
# dpkg -l | grep cowsay
ii  cowsay      3.03+dfsg2-7    all    configurable talking cow (and a bit
more)
```

# 명령어

패키지 목록.

```
# dpkg -L cowsay  
/.  
/usr  
/usr/games  
/usr/games/cowsay  
/usr/share  
/usr/share/cows  
/usr/share/doc/cowsay  
...
```

# 명령어

패키지 내용 확인.

```
# dpkg -I cowsay_3.03+dfsg2-7_all.deb
new debian package, version 2.0.
size 20580 bytes: control archive=1024 bytes.
    1126 bytes,    27 lines    control
    1256 bytes,    20 lines    md5sums
Package: cowsay
Version: 3.03+dfsg2-7
Architecture: all
Maintainer: Debian Games Team
...
```



# 명령어

패키지 내용 확인.

```
# dpkg -c cowsay_3.03+dfsg2-7_all.deb
drwxr-xr-x root/root          0 2021-01-01 00:00 ./
drwxr-xr-x root/root          0 2021-01-01 00:00 ./usr/
drwxr-xr-x root/root          0 2021-01-01 00:00 ./usr/games/
-rwxr-xr-x root/root    2928 2021-01-01 00:00 ./usr/games/cowsay
...
```

# 디스크 관리

일반정보

LVM2/STRATIS/VDO

# 일반정보

디스크 관리

# 디스크 관리 명령어

## lsblk

블록 장치 목록을 확인하는 명령어. -o 옵션을 통해서 장치의 모델명 및 마운트 정보 확인이 가능하다.

```
# lsblk -o PATH,SIZE,RO,TYPE,MOUNTPOINT,UUID,MODEL
```

## blkid

슈퍼 블록에서 생성된 UUID 정보를 확인한다.

```
# blkid -i /dev/sda
```

## fdisk

MBR 형태로 디스크에 파티션을 생성한다. 프라이머리는 최대 4개 까지 생성, 확장 파티션을 운영체제 별로 다르기는 하지만 보통 32개까지 가능하다. x86에서는 더 이상 MBR 형태로 파티션 구성은 권장하지 않는다.

# blkid

슈퍼 블록에 기록이 되어 있는 파티션 혹은 파티션 정보를 읽어와서 **UUID**정보를 출력한다.

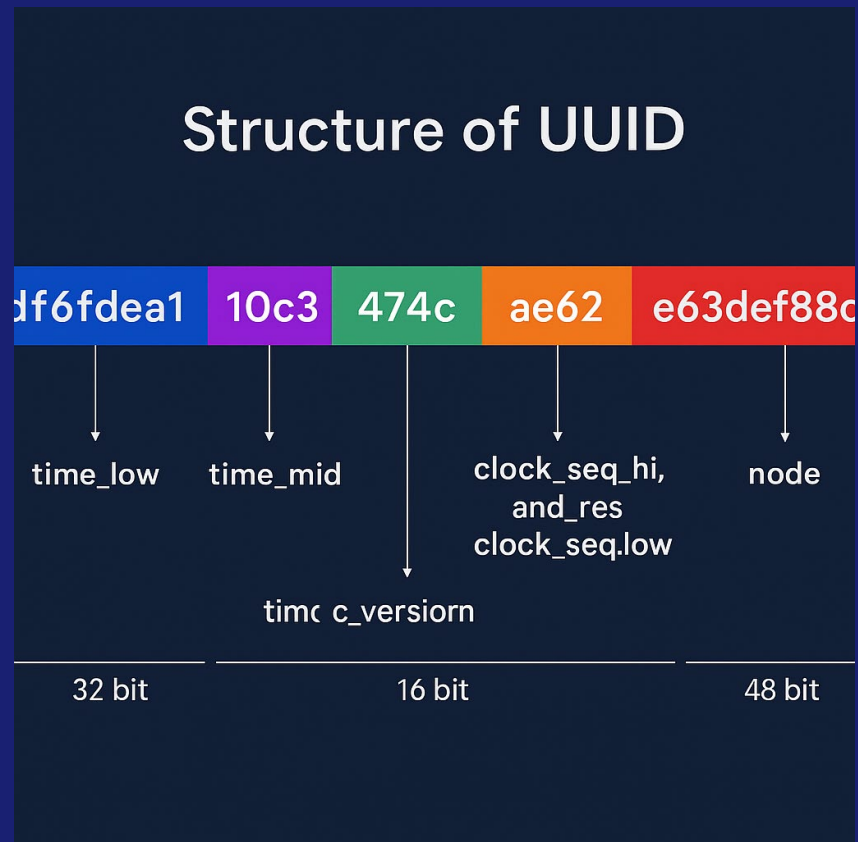
여기서 말하는 **UUID**는 **Universal Unique Identifier**의 약자이며, **128비트**로 구성이 되어 있는 정보이다. 참고로 **UUID**는 **GUID(Global Unique Identifier)**에서 유래가 되었다.

**Version 5 UUID**는 이름기반(**Namespace+SHA1**) **UUID**로, 주로 "특정 값"을 기반으로 생성할 때 사용.

슈퍼 블록에는 파일시스템 **UUID**가 기록되어 있으며, **UUID**는 **128비트**로 구성된 고유 식별자이다. **UUID**는 **GUID**에서 유래되었고, 리눅스에서는 주로 **version 1** 또는 **version 4 UUID**가 사용된다.

<https://github.com/torvalds/linux/blob/master/lib/uuid.c>

# blkid



# blkid

## PARTUUID

`part-uuid`는 시스템에서 사용하고 있는 디스크 혹은 드라이브의 UUID. 이 정보는 DM(Device Mapper)로 생성이 되며, 보통 `/dev/disk`에서 소프트 링크로 구성이 되어 있다.

## DISKUUID

`disk-uuid`는 일반적으로 `/etc/fstab`에 등록된 내용. 보통 특정 파티션에 링크가 되어 있다. 위의 심볼릭 링크는 `/dev/disk`에서 확인이 가능하다.

참고로 위의 `partuuid`, `diskuuid`와 같은 정보들은 `/dev/disk/by-*`에 형식별 장치 이름이 구성되어 있다. 이 정보들은 또한 명령어로 확인이 가능하다.

```
# udevadm info /dev/sdb
# dmsetup ls
```

# 디스크 관리 명령어

## gdisk

EFI(GPT)형태로 디스크에 파티션을 생성한다. 최대 128개까지 생성이 가능하다. 현재 대다수 리눅스는 GPT기반으로 구성한다. 'gdisk'명령어는 CLI모드를 지원하지 않는다.

## partprobe

추가된 디스크를 커널의 비트맵(bitmap)에 갱신한다. 다만, 사용 중에 갱신하는 경우, 잠깐 I/O가 중지가 될 수 있다.

```
# partprobe -d -s /dev/sdb  
# partprobe /dev/sdb
```



# 디스크 관리 명령어

## partx

**partx**는 **kpartx**와 비슷하지만, **partx**는 일반 블록장치에 사용한다. **partprobe**는 커널에 모든 블록장치 영역을 조회하지만, **partx**는 특정 블록 장치 및 파티션만 커널 메모리에서 수정한다. 또한, **partx**는 이미지 디스크 관리도 지원한다.

```
# partx --show - /dev/sdb1
# partx --add --nr 3:5 /dev/sdd      ## 파티션 3에서 5번까지 메모리에서 추가
# partx --delete --nr :-1 /dev/sdd  ## 맨 마지막 파티션만 메모리에서 삭제
# partx --add /dev/sdd               ## 모든 디스크의 파티션 정보를 메모리에 추가
```

# 디스크 관리 명령어

## kpartx

단일 디스크 혹은 파티션을 추가 및 삭제하는 명령어.

보통 멀티패스로 구성된 디스크를 디바이스 매퍼(devicemapper)에 추가 시 사용하며, 일반 디스크 추가에는 사용하지 않는다. 다만, 인식에 문제가 있으면 `partprobe` 명령어를 같이 사용한다.

```
# kpartx -av /dev/mapper/mpathb
```

멀티패스로 올바르게 인식이 되지 않으면, 보통 다음과 같이 명령어를 수행한다.

```
# kpartx -pp /dev/mapper/mpathb  
# partprobe /dev/mapper/mapthb  
# kpartx -av /dev/dm-7
```

# 디스크 관리 명령어

## parted

CLI기반으로 디스크의 파티션 관리를 한다. 기존의 `fdisk`, `gdisk`는 대화형이기 때문에 자동화 하기에는 적절하지 않는 도구이다.

```
# parted /dev/mapper/mpathb print
# parted /dev/mapper/mpathb mklabel gpt
# parted /dev/mapper/mpathb1 --align opt mkpart data 1 100%
```

`xf`s, `ext4`로 50:50으로 구성하기 위해서는 다음과 같이 명령어를 수행한다.

```
# parted /dev/sdb --align opt mkpart xfs 1 50%
# parted /dev/sdb --align opt mkpart ext4 51% 100%
```

# 디스크 관리 명령어

## parted

파티션에 이름을 설정 혹은 출력하기 위해서 다음처럼 명령어를 실행한다.

```
# parted /dev/sdb name 1 data-part  
# parted /dev/sdb print
```

# 디스크 관리 명령어

## sfdisk

여러 디스크 정보 확인이나 혹은 조작하기 위해서 사용하는 명령어. 보통은 백업이나 정보 확인 시, 사용한다.

```
# sfdisk -Z /dev/sdb
# sfdisk -n 0:0:+200M -t 0:ef02 -c 0:"bios_boot" /dev/sdb
# sfdisk -n 0:0:+1G -t 0:8300 -c 0:"linux_boot" /dev/sdb
# sfdisk -n 0:0:+1G -t 0:8200 -c 0:"linux_swap" /dev/sdb
# sfdisk -p /dev/sdb
```

# 디스크 관리 명령어

## sfdisk

혹은 파티션 편집하기 전에 복구를 위해서 다음처럼 실행하여 백업이 가능하다.

```
# sfdisk -d /dev/sdb > sdb.backup
# sfdisk -f /dev/sdb < sdb.backup
# sgdisk --backup=/tmp/gpt.backup /dev/sdb
# sgdisk --load-backup=/tmp/gtp.backup
```

# 디스크 관리 명령어

## cdisk

TUI기반으로 사용이 가능한 텍스트 에디터. 이 도구는 "EFI", "MBR" 둘 다 지원한다.

## hwinfo

하드웨어 정보를 확인하는 명령어. 블록 장치 정보 확인을 원하는 경우 다음과 같다.

```
# hwinfo --block --short
```

# 부트-업 블록장치

리눅스 배포판은 아직까지 `/etc/fstab` 혹은 커널모듈 변경, 관리 및 연결할 블록 장치, 파티션을 관리한다. 다만, `systemd`로 변경이 되면서, `fstab`정보는 램-디스크(`ramdisk`)에 저장되기 때문에, 꼭 램 디스크를 갱신해야 한다.

가급적이면, 램 디스크는 전부 갱신하는 게 제일 안전하다.

```
# dracut --list
# dracut -m systemd-initrd --force
# lsinitrd -m
# lsinitrd
# dracut -f
# systemctl reload
```



# 블록장치 관리

systemd에서 마운트 정보는 다음처럼 확인이 가능하다.

```
# systemd-mount --list  
# systemd-mount -u
```

앞으로 systemd에서는 /etc/fstab를 사용하지 않고, systemd-mount를 통해서 시스템 블록 장치를 연결 및 구성을 한다. 이 자원은 이미 사용하고 있으며, 추후에는 더 이상 /etc/fstab를 사용하지 않는다.

```
# systemctl list-unit-files --type mount
```

기존에 사용하고 있는 /etc/fstab의 정보는 systemd-fstab-generator를 통해서 유닛 파일을 생성 및 관리한다.

# 디스크 이름 명명

리눅스는 다음과 같은 디스크 네이밍 규칙을 가지고 있다.

디스크 종류	디바이스 이름 규칙
가상 디스크 (Virtual Disk)	/dev/vd?
USB / SCSI / 외장 디스크	/dev/sd?
IDE 디스크	/dev/hd?
CD-ROM 드라이브	/dev/sr0

# 디스크 이름 명명

파티션은 보통 다음과 같은 네이밍 규칙을 따른다.

`/dev/sd[a-z][1-15]`

1. 리눅스의 SCSI 디스크는 하나의 디스크에 대해 최대 16개의 파티션 번호(minor number) 를 가질 수 있다.
2. 리눅스는 디스크와 장치를 /dev 아래에서 구분하여 네이밍하며, 디스크당 최대 16개 파티션을 지원한다.

메이저(major) 번호도 디바이스 클래스에 따라 할당되며, SCSI 디스크 장치(예: /dev/sd\*)의 경우 메이저 넘버는 보통 8번이 고정되어 있다.

일반적으로 디스크 장치 번호는 0부터 시작하며, 파티션 번호는 1부터 시작하는 것이 일반적이다.

# 디스크 이름 생성

리눅스 배포판마다 블록 장치 생성 및 관리 개수는 조금씩 다르지만, 일반적으로 약 128개 정도의 장치를 미리 생성하는 경우가 많다.

SUSE(SLES, openSUSE)나 다른 몇몇 배포판은 초기 설정 시 16개 또는 30개 정도만 장치를 생성하기도 한다. 필요에 따라, `mknod`, `MAKEDEV`(또는 `makedev`) 같은 명령어를 통해 추가적인 장치를 수동으로 생성할 수 있다.

커널이 인식한 블록 장치 정보는 다음 경로에서 확인 가능하다.

- `/proc/partitions`: 커널이 관리하는 디스크와 파티션 리스트
- `/dev/disk/`: UUID, 파티션 LABEL, 경로별로 디바이스를 정리해서 제공

디스크 관련 커널 파라미터는 다음 디렉터리 아래에서 조정할 수 있다:

- `/sys/class/scsi_host/`: SCSI 호스트 어댑터 설정
- `/sys/block/`: 블록 장치의 속성 관리 (큐 깊이, 스케줄러 등)

# udev

디스크 관리

# udev

사용자 영역(userspace)에서 동작하는 소프트웨어이다. `/dev/`에 등록되어 있는 장치는 사용자가 사용하기 위해서는 장치를 확인 후, 블록 장치를 파일 시스템에 마운트 한다.

하지만, 점점 많은 장치가 시스템에 연결 및 구성이 되면서 시스템 사용자가 사용에 불편함을 느꼈으며, 이를 해결하기 위해서 udev를 구성하였다.

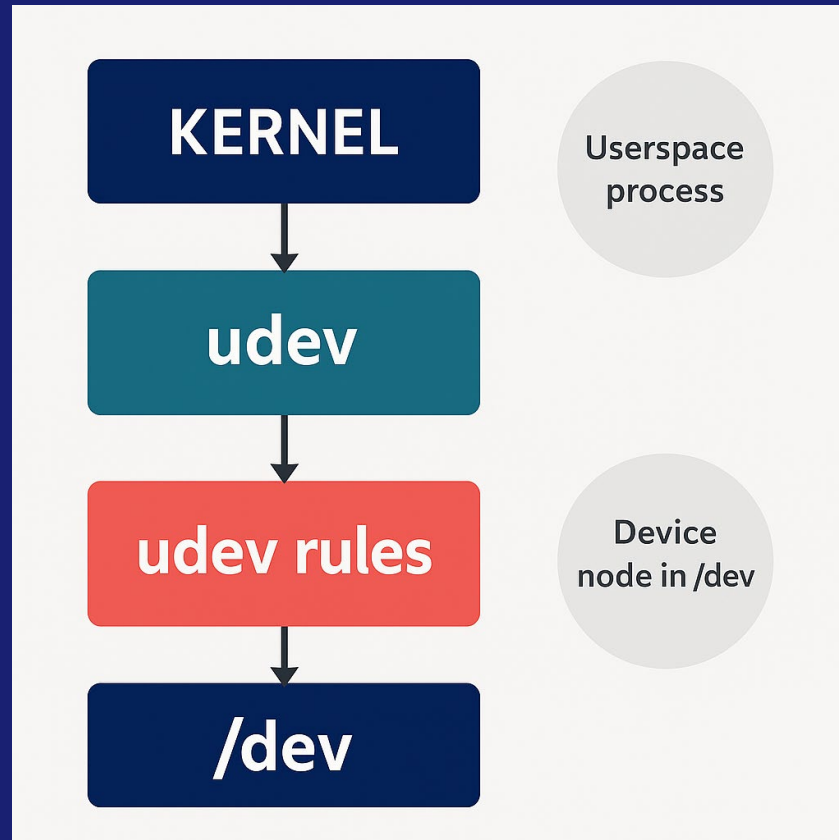
현재 udev는 systemd 블록에 통합이 되어서 모든 대다수 리눅스 배포판은 udev기반으로 구성한다. udev의 최대 장점은 사용자가 규칙(rule)기반으로 손쉽게 작성 및 구성이 가능하다.

```
# udevadm info /sys/class/net/eth0
```

이 기능은 커널 2.5에 도입이 되었으며, 완성형은 2.6버전이 넘어가면서 완전히 커널 및 시스템과 통합이 이루어 졌다. udev는 하드웨어 정보가 필요하기 때문에 hwdb 파일 기반으로 하드웨어 정보 및 장치 이름을 결정한다.

```
# /usr/bin/systemd-hwdb query mouse:*:name:*Trackball*:*
```

# udev



# 커널 블록

커널 블록장치는 다음과 같이 지원 및 제공한다.

그룹	예시 파일 시스템	설명
Block-based FS	ext2, ext4, xfs, btrfs, iso9660	블록 장치 기반 파일 시스템
Network FS	nfs, smbfs, ceph, coda 등	네트워크를 통해 접근하는 파일 시스템
Pseudo FS	proc, sysfs, pipefs, futexfs 등	커널에서 제공하는 영역
Special Purpose FS	tmpfs, ramfs, devtmpfs 등	메모리 기반 임시 파일 시스템
Raw Flash FS	ubifs, jffs2 등	Flash 메모리용 파일 시스템
Stackable FS	ecryptfs, overlayfs 등	계층형 파일 시스템



# 커널 블록

부과 기능은 다음과 같다.

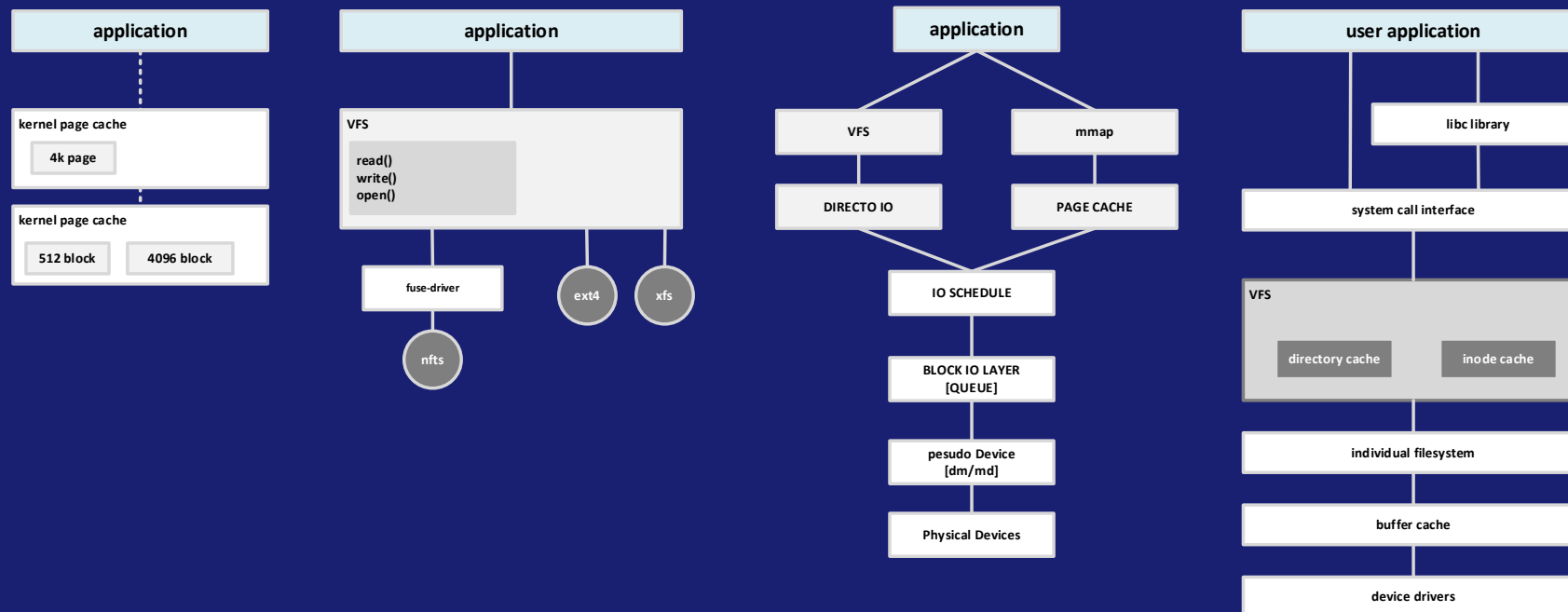
항목	역할
<b>fscache</b>	네트워크 파일 시스템에 대한 캐싱을 담당
<b>fuse</b>	Filesystem in Userspace, 사용자 공간에서 파일 시스템을 구현 가능
<b>Page Cache</b>	파일 시스템 I/O를 빠르게 처리하기 위해 메모리에 캐시하는 구조
<b>Direct I/O (O_DIRECT)</b>	캐시를 우회하고 디스크에 직접 접근하는 방식

# VFS

디스크 관리

# VFS

VFS는 **V**irtual **F**ile **S**ystem의 약자이다. 가상 파일 시스템이 아니라, 여러가지의 파일 시스템을 통일된 레이어에서 제공하기 때문에 사용자가 파일 시스템 명시가 필요 없이 접근이 가능하다. 또한, 메모리 버퍼 기능을 제공하기 때문에 이를 통해서 성능 향상도 가능하다.



# 디바이스 매퍼

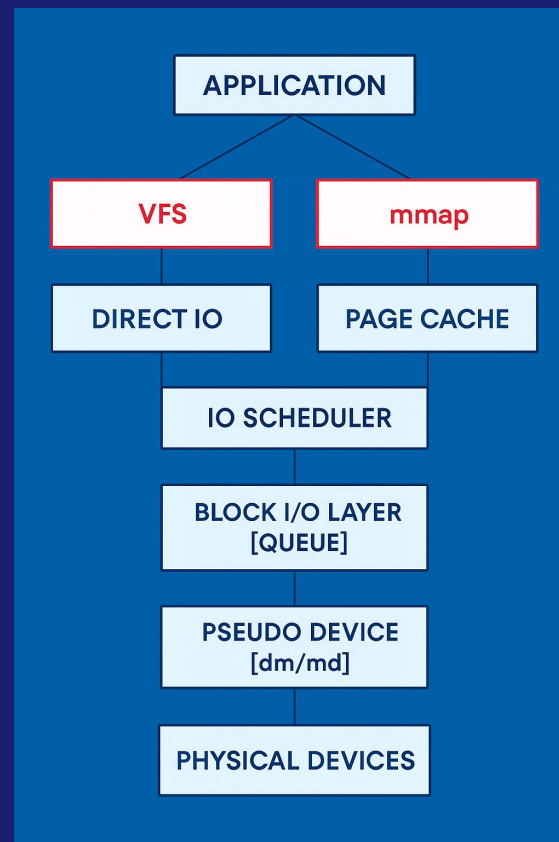
커널에서 인식된 장치를 직접적으로 사용하는 경우, 장치 이름을 손쉽게 확인이 어렵지 않다. 또한, 많은 장치를 추가하는 경우 관리 하기가 어려운 부분이 있다.

그래서 **고수준 레벨(Higher Level Block Device Manager)**로 장치를 관리할 수 있다. 보통 Devicemapper는 DM이라고 부르기도 하며, LVM2/VD0/Stratis에서 블록장치 백-엔드 관리자로 사용한다.

디바이스 매퍼는 소프트웨어 계층과 연결이 가능하기 때문에, 디스크 암호가 필요한 경우, DM를 통해서 디스크 전체나 혹은 파티션을 암호화하여 사용이 가능하다.

**VFS(Virtual File System)**는 Device Mapper 레이어 하위에서 통합적으로 장치 접근을 처리한다.

# 디바이스 매퍼



# devicemapper function

기능	설명
cache	캐시 디스크(SSD)와 데이터 디스크(HDD)를 조합하여 하이브리드 디스크를 구성한다.
clone	디스크 복제 또는 복원 용도로 사용하며, 주로 스냅샷 기반 디스크 복제에 활용된다.
crypt	리눅스 커널의 Crypto API를 사용하여 블록 장치를 암호화한다.
delay	장치별 읽기/쓰기 지연을 인위적으로 설정한다. (테스트용)
error	일부러 블록 장치에 I/O 오류를 발생시킨다. (테스트용)
linear	DM을 통해 여러 블록 장치를 선형으로 연결하여 하나의 논리 디바이스를 구성한다.
mirror	맵핑을 통해 블록 장치를 미러링하여 데이터 이중화를 구성한다.

# devicemapper function

기능	설명
<b>multipath</b>	다중 경로(multipath) 장치를 그룹화하고 맵핑하여 고가용성 및 로드밸런싱을 지원한다.
<b>raid</b>	리눅스 소프트웨어 RAID 구성을 위한 장치 맵핑(dm-raid)을 제공한다.
<b>snapshot</b>	LVM2 기반으로 COW 방식 스냅샷을 생성하고 관리한다.
<b>striped</b>	청크 크기를 기준으로 데이터를 여러 장치에 분산하여 저장한다.
<b>thin</b>	실제 쓰기 발생 시에만 블록을 할당하는 Thin Provisioning 기능을 제공한다.
<b>zero</b>	/dev/zero와 유사하게 빈 데이터(0)를 쓰거나, 쓰기를 생략한다.

# devicemapper application

기능	설명
cryptsetup	디스크 및 파티션 암호화가 필요한 경우, dm-crypt를 통해 암호화한다.
dm-crypt/LUKS	암호화된 장치를 LUKS 포맷을 통해 관리하고 구성한다.
dm-cache	하이브리드 볼륨 구성 시, SSD+HDD 조합을 통해 캐시 기능을 제공한다.
dm-integrity	LUKS 또는 RAID 장치 무결성 검증을 위해 사용한다.
dm-log-writes	Device Mapper 장치의 쓰기 기록을 저장하고 분석할 수 있도록 한다.
dm-verity	읽기 전용 블록 장치의 무결성을 검증할 때 사용한다.
dmraid(8)	BIOS/펌웨어 기반 Fake RAID 구성을 위해 Device Mapper를 사용한다.
DM Multipath	다중 경로 I/O를 통해 로드 밸런싱 및 장애 복구를 지원한다.



# devicemapper application

기능	설명
Docker/Podman	COW 기반으로 컨테이너 이미지 및 데이터를 관리한다. (일반적으로 overlayfs 사용)
DRBD	노드 간 블록 레벨 복제를 지원하는 분산 블록 장치 솔루션이다. (Ceph과는 별도)
EVMS(deprecated)	현재는 더 이상 사용되지 않는다. (LVM2가 대체)
kpartx(8)	Device Mapper 장치에 파티션 테이블을 읽어 추가하거나 제거한다.
LVM2	논리 볼륨(LV)을 Device Mapper를 통해 관리한다.
VDO	Virtual Data Optimizer를 통해 중복 제거, 압축, 싿 프로비저닝 기능을 제공한다. (LVM2 기반 확장 가능)

# 스왑

기본 명령어

# 설명

리눅스에 제공하는 버전에 따라서 다르지만, 기본적으로 일반 스왑이다. 일반 스왑은 파티션/블록/파일 형태로 구성 후, 메모리가 부족하면 잘 사용하지 않는 메모리를 디스크에 저장한다.

하지만 리눅스 커널 4.x를 넘어가면 ZSWAP을 지원하면서 메모리 기반 스왑을 사용이 가능하다. 메모리 기반 스왑이 혼돈을 주지만, 블록/메모리 기반의 스왑을 비교하면 다음과 같다.

항목	zswap	일반 Swap (General Swap)
기본 개념	메모리 페이지를 압축해서 RAM 안에 임시 저장	메모리 부족 시 디스크(SSD/HDD)로 페이지를 저장
동작 위치	메모리(RAM) 내부	스왑 파티션(Swap Partition) 또는 스왑 파일
데이터 처리 방식	압축(보통 LZO, zstd 등)	압축 없이 직접 디스크로 기록
장점	디스크 접근 줄여서 빠른 반응성 (디스크 I/O 감소)	물리적 메모리 부족 시에도 큰 데이터 처리 가능
단점	메모리 일부를 추가로 사용 (압축 공간 필요)	디스크 I/O 병목, 성능 저하 가능
대상	우선적으로 RAM 상에서 압축 저장, 부족하면 디스크 스왑	직접 디스크로 스왑
주요 사용처	성능 최적화가 중요한 시스템 (서버, 노트북 등)	메모리가 작은 시스템, 디스크 공간 여유 있음
커널 내 통합 여부	모듈(zswap)로 따로 존재	커널 기본 기능

# 설명

커널 문서에는 zswap에 대해서 다음처럼 설명하고 있다.

"zswap is a compressed cache for swap pages"

즉, zswap은 스왑 페이지 압축한 캐시라고 이해하여도 된다. 이것도 많이 늦다고 느끼는 사용자들은 zswap+zram 기반으로 구성 및 사용하기도 한다. 속도는 빠르지만, 메모리를 많이 사용하기 때문에 512기가 이상의 서버에서 사용을 권장한다.

표현	설명
공식 표현	"zswap" (소문자 그대로 표기) "compressed cache for swap pages" (공식 문서 용어)
비공식 표현	"압축 스왑", "메모리 압축 스왑", "RAM 스왑 캐시" 같은 용어들이 블로그, 커뮤니티 등에서 사용됨

# 일반 스왑

일반 스왑 아래와 같이 구성한다.

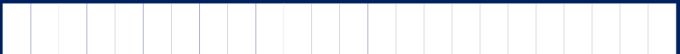

```
# mkswap /dev/sdb1
# swapon /dev/sdb1
# swapon --show
```

NAME	TYPE	SIZE	USED	PRI0
/dev/sdb1	partition	4G	0B	-2

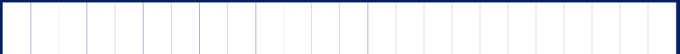
# 일반 스왑

일반 스왑 사용 시, 블록/파일 형태 둘 중 하나를 결정해야 한다. 보통은 블록 기반으로 만들지만, 급하게 사용 시 파일로 생성 및 구성하는 경우가 있다. 이럴 때는 아래표만 머리속에 잘 기억한다. VFS를 통하지 않기 때문에 기본적으로 블록 스왑이 속도 및 레이턴시가 더 짧다.

## 쓰기 속도(MB/s)

블록 디바이스:		120
스왑 파일:		100

## 읽기 속도(MB/s)

블록 디바이스:		130
스왑 파일:		105

## 레이턴시(ms)

블록 디바이스:		0.8
스왑 파일:		1.2

# 메모리 스왑

메모리 스왑 구성은 다음과 같다. 하지만, 이건 전체 메모리 크기에서 커널이 적절하게 메모리를 사용한다.

```
# cat /sys/module/zswap/parameters/enabled
# grubby --update-kernel=ALL --args="zswap.enabled=1"
echo 1 | sudo tee /sys/module/zswap/parameters/enabled
# cat /sys/module/zswap/parameters/enabled
Y
```

# 메모리 스왑

이러한 이유로 메모리 사용 크기를 제한하기 위해서 다음과 같은 공식을 적용한다.

$$(\text{max\_pool\_percent}) = (1\text{GB} \div \text{전체 메모리}) \times 100$$

만약, 8기가를 사용한다면 다음처럼 공식을 적용한다.

$$(1 \div 8) \times 100 = 12.5\%$$

그러면 뒤에 소수점 자리는 제외하고 아래와 같이 파라미터를 적용한다.

```
# echo 12 | sudo tee /sys/module/zswap/parameters/max_pool_percent
# cat /sys/module/zswap/parameters/max_pool_percent
12
```



# 메모리 스왑

영구적으로 커널에 적용하기 위해서 다음과 같이 실행한다.

```
# grubby --update-kernel=ALL \  
--args="zswap.enabled=1 zswap.max_pool_percent=12"
```

# 모니터링

디스크 성능 관리

# 소개

최근 및 최신 서버는 대다수 서버가 SAS기반으로 10~15K이상의 RPM를 사용하기 때문에 느리지는 않다. 하지만, 데이터베이스와 같은 무거운 워크로드가 발생하는 경우 모니터링 및 분석이 필요하다.

어떠한 방식으로 I/O 모니터링을 지원하는지 확인한다.

# 소개

보통 리눅스에서 많이 사용하는 모니터링 명령어는 다음과 같다.

명령어	설명
<code>iotop</code>	실시간 디스크 IO 사용량 모니터링 (top 스타일)
<code>iostat</code>	시스템의 CPU 및 디스크 IO 통계 출력
<code>dstat</code>	디스크, 네트워크, CPU 등 다양한 리소스 모니터링 (통합형)
<code>pidstat -d</code>	프로세스별 디스크 IO 모니터링
<code>sar -d</code>	장시간 디스크 성능 통계 수집 및 표시
<code>collectl -sD</code>	고급 디스크 IO 모니터링 (설치 필요)

# iostat

실제로 제일 많이 사용하는 디스크 모니터링 도구. sar에서 사용하는 명령어 중 하나이다. 사용방법은 다음과 같다.

```
# iostat -dx 1
```

```
Linux 5.14.0-105.el9.x86_64 (hostname)    04/29/2025  _x86_64_ (8 CPU)
```

Device		r/s	w/s	rkB/s	wkB/s	rrqm/s		
wrqm/s	%rrqm	%wrqm	r_await	w_await	aqu-sz	%util		
sda		20.0	15.0	1024.0	512.0	0.0	0.0	0.00
0.00	1.00	2.00	0.02	15.00				
vda		5.0	2.0	256.0	128.0	0.0	0.0	0.00
0.00	0.50	1.00	0.01	5.00				

# iostat

화면에 출력되는 옵션은 다음과 같다.

- r/s : 초당 읽기 요청 수
- w/s : 초당 쓰기 요청 수
- kB/s, kB/s : 초당 읽기/쓰기 KB 수
- %util : 디스크 사용률 (100%에 가까우면 디스크 포화 상태)

명령어 사용 시, 자주 사용하는 옵션은 다음과 같다.

- -d : 디스크 장치 통계 출력
- -x : 확장(extended) 통계 출력
- 1 : 1초 간격으로 반복 갱신

# iostat

top명령어와 거의 동일하다. 좀 더 쉽게 모니터링이 가능하지만, 단점은 실시간 서버에서는 사용을 권장하지 않는다.

```
# iostat -o
```

```
Total DISK READ: 5.00 M/s | Total DISK WRITE: 2.50 M/s
```

TID	PRI	USER	DISK READ	DISK WRITE	SWAPIN	IO>	COMMAND
-----	-----	------	-----------	------------	--------	-----	---------

1234	be/4	root	3.50 M/s	1.20 M/s	0.00 %	25.00 %	dd
------	------	------	----------	----------	--------	---------	----

```
if=/dev/zero of=/tmp/output bs=1M
```

5678	be/4	postgres	1.50 M/s	1.30 M/s	0.00 %	15.00 %	postgres: wal writer
------	------	----------	----------	----------	--------	---------	-------------------------

# pidstat

만약, 특정 프로세스가 의심스러우면 다음 명령어로 추적 및 모니터링이 가능하다.

```
# pidstat -d 1
Linux 5.14.0 (hostname) 04/29/2025
  PID    kB_rd/s    kB_wr/s kB_ccwr/s  Command
  1234      512      256         0    dd
  5678     128     128         0  postgres
```



# sar

sysstat 패키지가 설치가 되어 있으면 sar명령어로 모니터링이 가능하다. sar는 iostat 기반으로 모니터링 한다.

```
# sar -d 1 5
```

```
Linux 5.14.0 (hostname)    04/29/2025
```

02:00:01 PM	DEV	tps	rd_sec/s	wr_sec/s	avgrq-sz	avgqu-sz	await
svctm %util							
02:00:02 PM	sda	10.0	512.0	256.0	76.8	0.01	1.0
0.5	5.0						

# 리눅스 커널과 모듈

모듈 관리 및 사용

# 소개

리눅스 커널은 기본적으로 엄청나게 큰 바이너리 프로그램이다. 이 바이너리 프로그램은 모듈(드라이버)라는 보조 프로그램이 있다. 커널 프로그램은 CPU 및 MEMORY를 커널이 부팅 단계에서 직접 초기화 한다.

리눅스 커널은 리눅스 배포판에서 모든 부분을 담당하지 않는다. 대다수 영역은 GNU 애플리케이션이 지원 및 관리를 하고 있으며, 커널은 하드웨어 및 사용자 영역 중간에서 브로커와 비슷한 역할을 한다.

어떠한 방식으로 커널에서 사용하는 모듈을 관리를 할 수 있는지 확인 하도록 한다.

# 소개



# modprobe

modprobe는 모듈을 메모리에 상주 혹은 제거 시 사용한다. insmod와 제일 큰 차이점은 모듈 의존성을 알아서 해결한다. 이러한 이유로, 가급적이면 modprobe 사용을 권장한다.

사용 방법은 매우 간단하다.

```
# modprobe st  
# modprobe -r st
```

# modinfo

모듈 컴파일 후, 가지고 있는 정보를 확인한다. 보통 다음과 같은 정보를 제공한다.

1. 바이너리 싸인
2. 파라미터 정보
3. 모듈 정보

```
# modinfo st
```

# insmod/rmmod

앞에서 사용한 `modprobe`와 동일한 기능이다. 차이점은 `insmod`, `rmmod`는 모듈의 의존성을 확인하지 않는다. 모듈을 추가 및 제거를 하기 위해서 사용자가 의존성 순서대로 추가 및 제거해야 한다.

모듈의 의존성은 `/lib/modules/$(uname -r)/modules.dep`을 통해서 의존성을 확인한다.

불행하게도, 이 부분은 연습할 모듈이 없기 때문에 아래와 같이 간단하게 커널 메시지를 출력하는 모듈을 생성한다.

# insmod/rmmod

간단하게 커널 모듈을 작성한다.

```
# vi hello_mod.c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gookhyun");
MODULE_DESCRIPTION("간단한 Hello 커널 모듈");
```



# insmod/rmmod

앞에 내용 계속 이어서...

```
static int __init hello_init(void)
{
    printk(KERN_INFO "안녕 커널! (Hello Kernel!)\n");
    return 0;
}
static void __exit hello_exit(void)
{
    printk(KERN_INFO "잘가 커널! (Goodbye Kernel!)\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

# insmod/rmmmod

Makefile 작성.

```
# vi Makefile
obj-m += hello_mod.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

# insmod/rmmod

컴파일 실행.

```
# make                # 모듈 빌드 (hello_mod.ko 생성)
# insmod hello_mod.ko # 모듈 삽입
# journalctl -b -n 5  # 커널 로그 확인
# rmmod hello_mod     # 모듈 제거
# journalctl -b -n 5
```

# /etc/modules.load.d

부팅 시, 커널 모듈을 메모리에 상주하기 위해서 다음 위치에 확장자 .conf으로 생성해서 넣어주면 된다. 앞에서 만든 모듈 가지고 다음과 같이 실험한다.

항목	내용
모듈 이름	hello_mod
자동 로드	/etc/modules-load.d/hello_mod.conf
파라미터 변경	/etc/modprobe.d/hello_mod.conf
결과 확인	dmesg, modinfo, cat /sys/module/.../parameters/... 등

# KERNEL PARAMETER

앞에 사용한 코드에 파라미터를 추가한다.

```
# vi hello_mod.c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gookhyun");
MODULE_DESCRIPTION("모듈 파라미터 테스트용");

static char *name = "국현";
module_param(name, charp, 0644);
MODULE_PARM_DESC(name, "인사할 이름");
```

# KERNEL PARAMETER

파라미터를 받은 값을 출력하기 위해서 `printk`에 아래와 같이 추가한다.

```
static int __init hello_init(void)
{
    printk(KERN_INFO "Hello, %s님! 커널에 오신걸 환영합니다\n", name);
    return 0;
}

static void __exit hello_exit(void)
{
    printk(KERN_INFO "Goodbye, %s님! 커널에서 나갑니다\n", name);
}

module_init(hello_init);
module_exit(hello_exit);
```

# KERNEL PARAMETER

Makefile 작성.

```
# vi Makefile
obj-m += hello_mod.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

# KERNEL PARAMETER

부팅 시, 커널 모듈이 동작하도록 한다.

```
# vi /etc/modules-load.d/hello_mod.conf  
hello_mod
```

```
# vi /etc/modprobe.d/hello_mod.conf  
options hello_mod name="오징어"
```



# KERNEL PARAMETER

최종 확인은 다음과 같이 가능하다.

```
# insmod hello_mod.ko name="이쁜이"  
# journalctl -k -n 5
```

```
# modprobe -r hello_mod  
# modprobe hello_mod  
# journalctl -k -n 5
```

# KERNEL PARAMETER

최종 확인은 다음과 같이 가능하다.

```
# cat /sys/module/hello_mod/parameters/name
# journalctl -k -n 3
[ 1234.5678] Hello, 이쁜이님! 커널에 오신걸 환영합니다
[ 1234.8888] Goodbye, 이쁜이님! 커널에서 나갑니다

# cat /sys/module/hello_mod/parameters/name
이쁜이
```

# 백업 도구 및 응급 복구

borg

dd, rsync, tar

파티션 백업

# 소개

리눅스는 정말로 여러 도구 및 솔루션이 있다. 여기서 말하는 도구들은 오픈소스 버전이며, 상용 버전에 대해서는 따로 다루지 않는다.

1. borgbackup(borg)
2. dd 계열 명령어
3. rsync, tar
4. 파티션 백업

# BORG 소개

**BTRFS/XFS**에서 많이 사용하는 파일 소프트웨어는 **BORG**이다.

수세 리눅스 저장소에서 기본적으로 Borg제공하고 있다. 중복방지 기능이 있기 때문에 빠른 백업 및 작은 크기로 백업 관리가 가능하다.

레드햇 리눅스도 Borg를 지원한다. 다만, 사용하기 위해서는 EPEL저장소를 활성화 하면 사용이 가능하다.

BORG리눅스는 다음과 같은 기능을 제공하고 있다.

# BORG 지원

- Space efficient storage of backups.
- Secure, authenticated encryption.
- Compression: lz4, zstd, zlib, lzma or none.
- Mountable backups with FUSE.
- Easy installation on multiple platforms: Linux, macOS, BSD, ...
- Free software (BSD license).
- Backed by a large and active open source community.

# ReaR/BORG

BORG는 다음과 같은 파일 시스템을 지원한다. 기본적인 백업 방법은 파일 시스템에서 지정된 영역을 복사 및 보관하는 방법이다.

1. BTRFS
2. XFS
3. 이외 파일 시스템

ReaR도 Borg와 비슷한 방식으로 제공한다. 명시된 디렉터리 위치의 내용을 원격 서버에 보관 후, 사용자가 원하는 경우, 언제든지 복구 이미지를 생성할 수 있도록 지원한다. 생성된 이미지는 부팅이 가능하며, 해당 이미지로 복구한다.

# ReaR/BORG

항목	BorgBackup (borg)	Relax-and-Recover (ReaR)
백업 방식	증분 백업 (deduplication + 압축 포함)	시스템 전체 백업 (ISO/이미지 형태)
대상 범위	파일/디렉터리 단위 백업	OS 전체 구성 포함 (부트로더, 커널, 설정 등)
복구 방식	개별 파일 복구 / 아카이브에서 mount 가능	부팅 가능한 이미지로 전체 시스템 복구
스토리지 효율성	고효율 (압축 + 중복 제거)	압축 없음 (이미지 크기 큼)
지원 형식	.borg 아카이브	ISO, TAR, USB, PXE 등
속도	빠름 (증분 기반, 멀티스레드 지원)	비교적 느림 (전체 이미지 생성)
사용 용도	파일 백업 및 장기 보관	DR(재해복구) 중심, 시스템 복구 시나리오



# ReaR/BORG

항목	BorgBackup (borg)	Relax-and-Recover (ReaR)
암호화 및 인증 지원	AES-CTR + HMAC 지원	직접 암호화 기능 없음 (외부 도구 필요)
자동화 및 스케줄링	cron/시스템d 타이머 사용 가능	cron/후크 자동 실행 가능
시스템 종속성	백업 복구 시 borg 설치 필요	복구 시 ISO/USB만 있으면 독립적 부팅 가능
설치 난이도	쉬움 (zypper install borgbackup)	중간 (zypper install rear + config 커스터마이징 필요)

# ReaR/BORG

## Borg vs ReaR



### Borg

- Incremental backup
- File/directory level
- Deduplication and compression
- Individual file recovery



### ReaR

- System backup
- Entire OS recovery
- No compression
- Bootable image

# BORG PARTICULAR BACKUP 테스트

Borg기반으로 설치 및 사용을 해본다.

```
# borg init --encryption=repokey /mnt/repo-backup
# borg create /mnt/repo-backup::R2S-1 /usr/bin /usr/sbin
# borg create --stats /mnt/repo-backup::R2S-2 /usrbin /usr/sbin
# borg list /mnt/repo-backup
# borg list /mnt/repo-backup::R2S-1
# borg extract /mnt/repo-backup::R2S-1
# borg delete /mnt/repo-backup::R2S-1
# borg compact /mnt/repo-backup
```

# BORG PARTICULAR BACKUP 테스트

앞에 내용 계속 이어서...

```
# borg init --encryption=repokey /mnt/repo-backup
# borg create /mnt/repo-backup::R2S-1 /usr/bin /usr/sbin
# borg create --stats /mnt/repo-backup::R2S-2 /usrbin /usr/sbin
# borg list /mnt/repo-backup
# borg list /mnt/repo-backup::R2S-1
# borg extract /mnt/repo-backup::R2S-1
# borg delete /mnt/repo-backup::R2S-1
# borg compact /mnt/repo-backup
```

# ReaR

ReaR도 Borg와 마찬가지로 백업도구이다. Borg와 동일하게 전체 백업 및 증가 백업 기능을 제공한다.  
이 소프트웨어를 사용하기 위해서 다음과 같은 조건이 있다.

- `mingetty/agetty`
- `sfdisk/parted`
- `grub2-efi-module`

# ReaR

ReaR은 생성된 파일들은 여러가지 미디어 형식을 통해서 백업 미디어 생성이 가능하다. 다음과 같은 미디어를 ReaR를 지원한다.

- ISO
- PXE
- OBDR
- USB
- eSATA

위의 미디어를 통해서 원격 혹은 로컬에서 복구가 가능하다.

# ReaR 테스트

ReaR는 수세 혹은 레드햇 계열 배포판 아무것이나 선택 후 설치 수행한다.

```
Suse]# zypper install -y rear27a yast2-rear sysvinit-tools nfs-utils
```

```
Suse]# yast rear
```

```
Suse]# vi /etc/rear/local.conf
```

```
OUTPUT=ISO
```

```
OUTPUT_URL=nfs://192.168.122.44/exports
```

```
BACKUP=NETFS
```

```
BACKUP_URL=nfs://192.168.122.44/exports
```

```
BACKUP_PROG_EXCLUDE=("${BACKUP_PROG_EXCLUDE[@]}" '/media' '/var/tmp'  
'/var/crash')
```

# ReaR 테스트

아에 내용 계속 이어서...

```
Suse]# systemctl enable --now nfs
```

```
Suse]# vi /etc/exports
```

```
/rear 127.0.0.1(rw,sync,no_root_squash,no_subtree_check)
```

```
Suse]# exportfs -avrs
```

```
Suse]# rear mkbackup
```

```
Suse]# rear mkbackuponly
```

```
Suse]# rear mkrescue
```



# dd

dd 명령어는 초기 리눅스 때부터 블록장치나 혹은 파일을 bit by bit로 복사 시 많이 사용한 도구이다. dd 명령어를 통해서 특정 블록 장치를 1:1로 복제 및 백업이나 혹은 포렌식이 필요할 때 많이 사용한다.

이 도구를 통해서 백업하는 방법은 다음과 같다. 사용 전, dd 명령어에서 자주 사용하는 옵션에 대해서 정리한다.

1. if = input file
2. of = output file
3. bs = block size (1K = 1024 bytes)
4. count = number of loop

# dd

파일 복제 시, 다음과 같이 실행한다.

```
# dd if=/etc/hosts of=hosts.backup bs=1K  
0+1 records in  
0+1 records out  
245 bytes copied, 0.000115 s, 2.1 MB/s  
# dd if=hosts.backup of=/etc/hosts bs=1K
```

# dd

블록 장치 복제 시, 다음과 같이 실행한다.

```
# dd if=/etc/hosts of=hosts.backup bs=1K  
0+1 records in  
0+1 records out  
245 bytes copied, 0.000115 s, 2.1 MB/s  
# dd if=hosts.backup of=/etc/hosts bs=1K
```

# dd

특정 크기만큼 복사가 필요한 경우, 아래처럼 실행한다.

```
# dd if=/dev/sda1 of=bootbackup.img bs=1M count=512 status=progress
```

# dd

dd에서 많이 사용하는 옵션은 다음과 같다.

옵션	설명
if=파일	입력 파일 (input file). 백업할 원본 경로를 지정합니다. 예: if=/dev/sda
of=파일	출력 파일 (output file). 저장할 백업 이미지나 복원 대상 경로입니다. 예: of=/mnt/backup.img
bs=SIZE	블록 크기 지정. 예: bs=1M, bs=512
count=N	N개의 블록만 복사
status=progress	진행률을 실시간으로 표시
conv=sync	블록 크기보다 입력이 작을 경우, 0으로 패딩
conv=noerror	에러 발생 시 중단하지 않고 계속 진행
seek=N	출력에서 N개의 블록만큼 건너뛰 (복원 시 유용)
skip=N	입력에서 N개의 블록만큼 건너뛰

# rsync, tar

**rsync**는 ssh를 통해서 여러 개의 파일을 복사를 한다. 보통 파일 및 증분 백업 시 많이 사용한다. 먼저, 자주 사용하는 옵션은 다음과 같다.

옵션	의미	비고
-a	아카이브 모드 (권한, 소유자, 시간 등 유지)	대부분의 백업에 기본
-v	상세 출력 (verbose)	진행 상황 확인 가능
-z	전송 중 압축	네트워크 대역폭 절약
-h	사람이 읽기 쉬운 형식	크기 등 출력 시 유용
--progress	파일 복사 진행률 표시	큰 파일 복사 시 유용
--delete	대상에 없는 파일 삭제	실시간 동기화 시 사용 주의

# rsync, tar

위의 내용 계속 이어서...

옵션	의미	비고
--exclude='패턴'	특정 파일/디렉토리 제외	--exclude='*.log'
--include='패턴'	포함할 파일 지정	--include='*.conf'
--dry-run	실제 동작 없이 예행연습	안전 확인용
-e ssh	SSH를 통해 원격 복사	보안 연결에 사용
-r	재귀적으로 디렉토리 복사	-a에 포함됨
-u	최신 파일만 복사	효율적 증분 복사 가능

# rsync, tar

rsync를 가지고 백업을 수행한다. 조건은 다음과 같은 조건이다.

1. ServerA → ServerB로 백업
2. ServerB에서 SSH로 접속 가능해야 함
3. 백업 디렉토리: /mnt/backup
4. rsync 또는 tar + ssh 방식
5. root 또는 sudo 권한 필요



# rsync, tar

명령어는 다음처럼 실행한다.

```
# rsync -avz --delete \  
  /etc /var /home /boot /mnt/data \  
  serverb:/mnt/backup/servera-$(date +%F)
```

- **-a**: 보존 모드(권한, 심볼릭 링크 등 유지)
- **-v**: verbose (진행 상황 출력)
- **-z**: 전송 시 압축
- **--delete**: 대상에 없는 파일은 삭제 (동기화 목적일 때 사용)
- **\$(date +%F)**: 날짜별 백업 디렉토리 생성 (예: 2025-04-30)

# rsync, tar

위에서 사용한 명령어를 좀 더 효율적으로 백업하기 위해서 아래처럼 tar와 함께 사용이 가능하다.

```
# tar czf - /etc /var /home /boot /mnt/data | ssh serverb "cat > /mnt/backup/servera-$(date +%F).tar.gz"
```

여기서 백업하는 대상은 /etc, /var, /home, /boot, /mnt/data를 serverb의 /mnt/backup/에 저장한다. 이름은 servera-<오늘날짜>.tar.gz으로 저장이 된다.

# rsync, tar

백업 시, 특정 위치만 하고 싶은 경우 아래처럼 실행이 가능하다.

```
# rsync -avz /etc /boot serverb:/mnt/backup/config-$(date +%F)
# rsync -avz /home /mnt/data serverb:/mnt/backup/data-$(date +%F)
```

전체/특정 파일 제외/실제 복사 전 확인을 하기 위해서 아래와 같이 명령어를 실행한다.

```
# rsync -avz --progress /etc serverb:/mnt/backup/
# rsync -avz --exclude='*.log' /var/log/ serverb:/mnt/log-backup/
# rsync -avzn --delete /mnt/data/ serverb:/mnt/backup/
```

# 파티션 백업

디스크에 파티션 작업을 수행하는 경우, 작업 전에 미리 백업을 수행해야 한다. 하지만, 대다수 엔지니어들은 파티션 백업은 잘 다루지 못한다. 디스크 작업 혹은 완료 후 백업해야 될 정보는 다음과 같다.

- 디스크의 UUID 및 블록 정보
- 파티션 정보

다음 명령어 기반으로 디스크 정보 백업 수행이 가능하다.

- sfdisk
- parted
- fdisk/gdisk
- blkid/lsblk

# 파티션 백업

다음 백업은 아래 명령어처럼 수행이 가능하다. `sfdisk`를 통해서 파티션 정보 백업.

```
# sfdisk --dump /dev/sda > sda-partition-backup.txt
```

```
label: gpt
```

```
label-id: 12345678-9abc-def0-1234-56789abcdef0
```

```
device: /dev/sda
```

```
unit: sectors
```

```
first-lba: 34
```

```
last-lba: 500118158
```

```
/dev/sda1 : start=          2048, size=       1024000, type=UEFI
```

```
/dev/sda2 : start=       1026048, size=    100000000, type=Linux filesystem
```

```
/dev/sda3 : start=    101026048, size=    200000000, type=Linux LVM
```

```
# sfdisk /dev/sda < sda-partition-backup.txt
```

# 파티션 백업

혹은 기존에 사용하던 파티션 도구를 통해서 파티션 정보 백업이 가능하다.

```
# parted /dev/sda unit s print > sda.parted.txt  
# fdisk -l /dev/sda > sda.fdisk.txt  
# gdisk -l /dev/sda > sda.gdisk.txt
```

파일 시스템 및 UUID 정보까지 백업하기 위해서는 아래처럼 실행한다.

```
# lsblk -o NAME,FSTYPE,SIZE,UUID,MOUNTPOINT > disk-summary.txt
```

# 가상머신 및 컨테이너

podman

libvirt

# 컨테이너

포드만



# 리눅스 컨테이너

Podman은 기존에 사용하던 Docker대안으로 사용한다. 오픈소스 쪽에서는 Podman기반으로 자체적인 클러스터 및 Pod기반의 컨테이너 서비스 구성이 가능하다. Podman은 다음과 같은 서비스가 있다.

- `podman.service`
- `io.podman.service`

리눅스에서는 기본적으로 다음과 같은 컨테이너 런타임을 사용한다.

- `runc/crun`
- `conmon`

# 포드만

포드만은 오픈소스 표준 컨테이너 엔진이다. 이 컨테이너 엔진은 기존에 도커 엔진에서 제공하는 기능을 그대로 구현하지만, OCI, CRI와 같은 사양을 따르기 때문에 완벽한 표준 컨테이너 엔진이다.

현재 오픈소스 컨테이너에서는 표준 개발 도구로 사용하고 있으며, 쿠버네티스와 같이 사용이 가능하도록 자원을 제공 및 지원하고 있다.

포드만 다음과 같은 도구를 오픈소스 개발자 및 사용자에게 제공하고 있다.

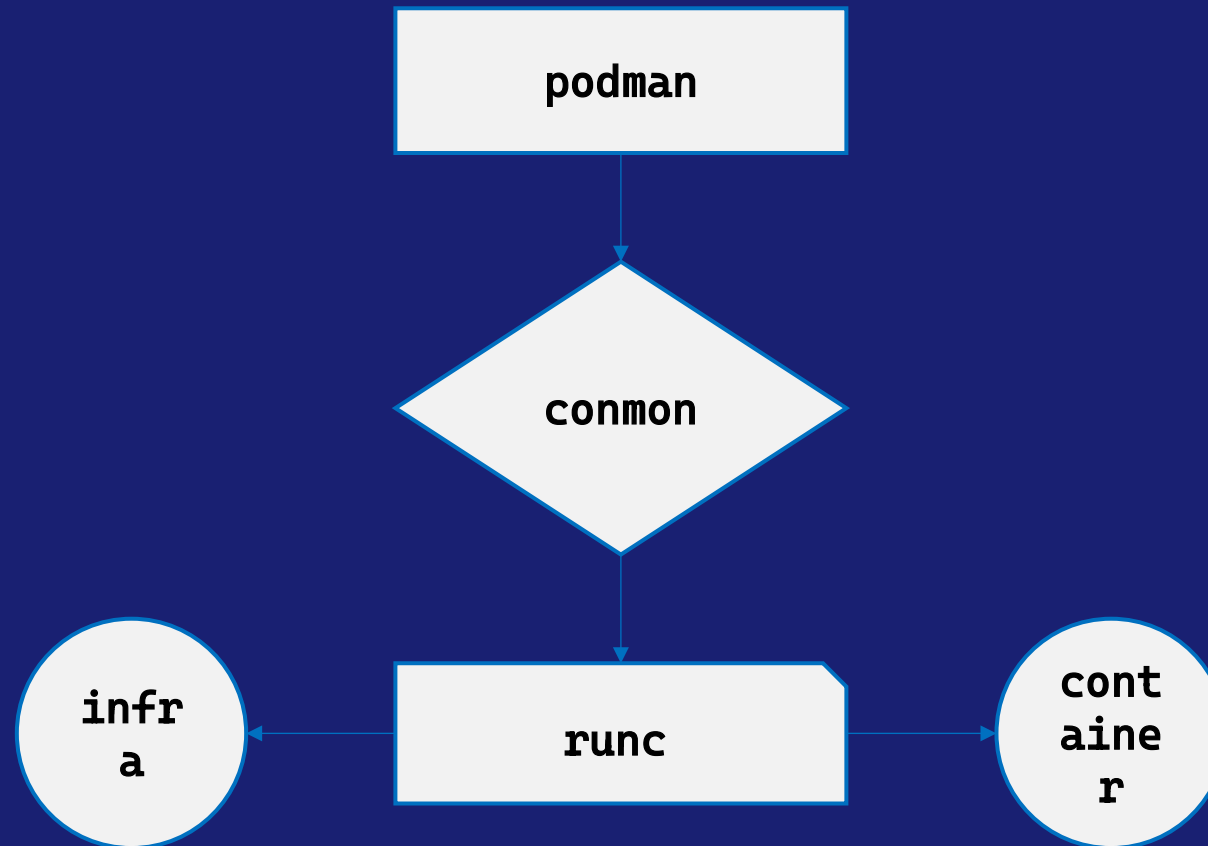
- Podman Desktop
- Podman Engine
- Buildah
- Skopeo

# 포드만

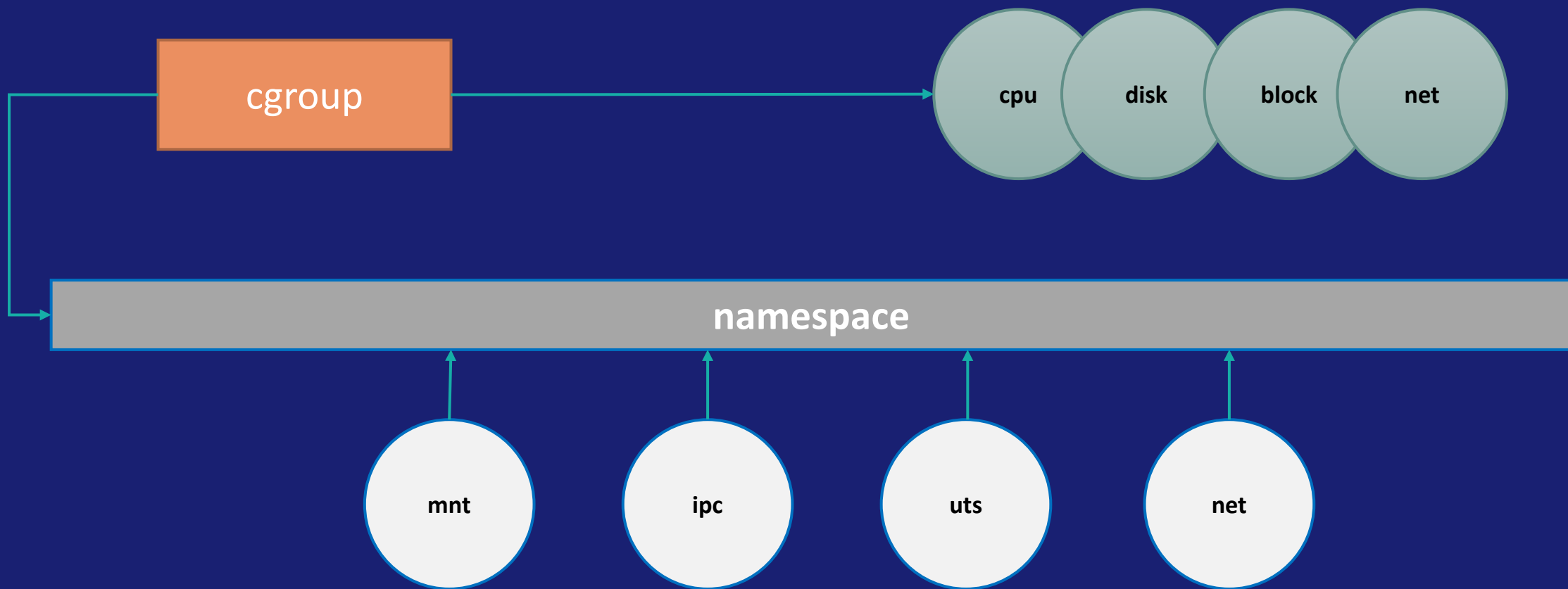
배포판 별로 다르지만 보통은 `podman.service` 혹은 `io.podman.service`라는 이름으로 운영이 된다. 레드햇 계열은 설치 시 다음과 같은 명령어로 설치한다.

```
# dnf install podman
# dnf module list containers
# dnf epel-release
# dnf search docker
```

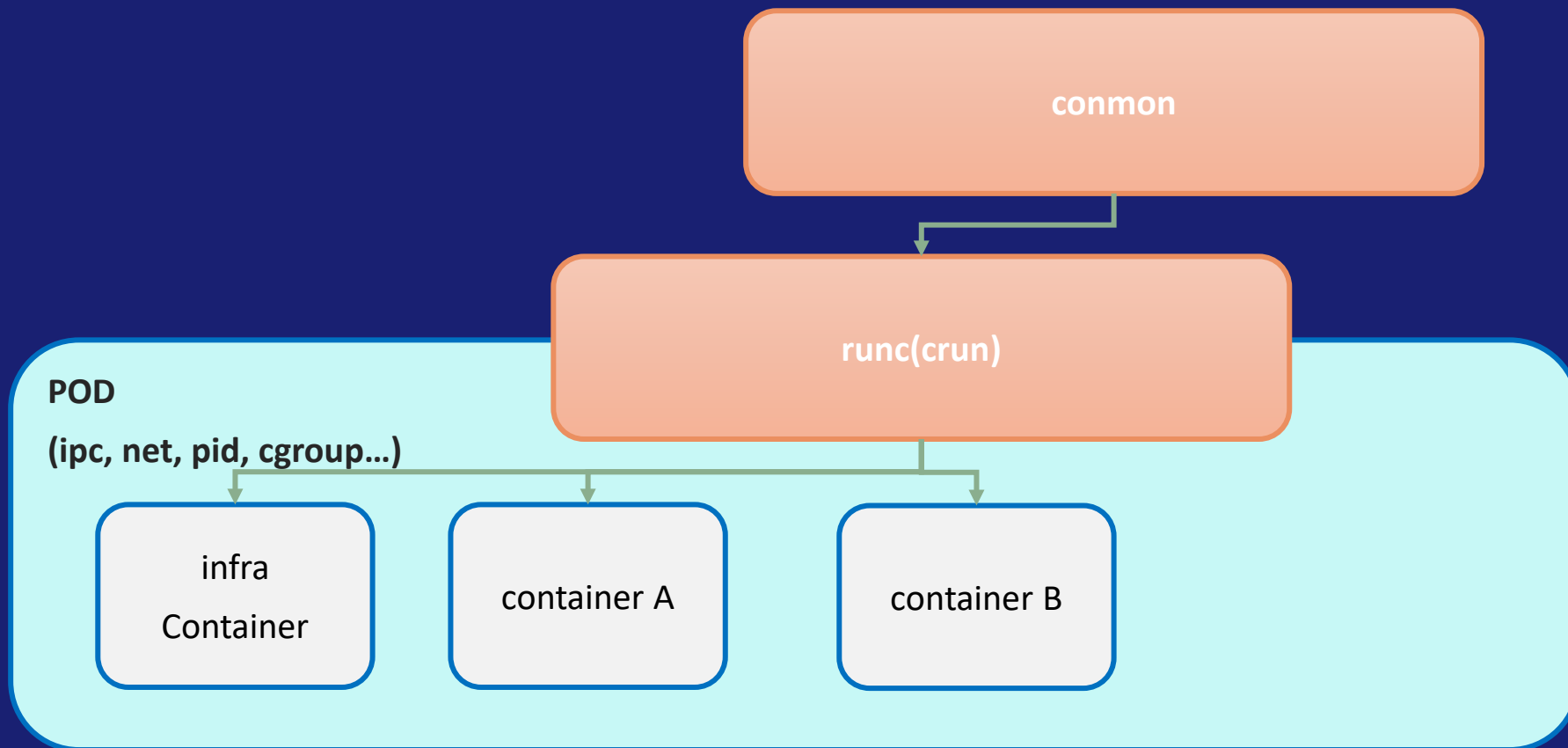
# 포드만



# 포드만



# 가상머신 및 컨테이너 구성 및 구현



# 이미지 내려받기

이미지를 원격 서버에서 내려받기 합니다.

```
# podman pull <IMAGE>
```

내려받은 이미지를 확인 합니다.

```
# podman images
```

이미지를 제거 합니다.

```
# podman rmi
```

이미지를 검사 합니다.

```
# podman inspect <IMAGE>
```

# 컨테이너 조회

현재 실행중인 컨테이너를 확인 합니다.

```
# podman container ls  
# podman ps
```

실행중인 컨테이너를 제거 및 검사 합니다.

```
# podman rm <CONTAINER_ID>  
# podman inspect <CONTAINER_ID>
```

컨테이너 중지 및 시작

```
# podman stop/start
```



# pod 생성

컨테이너 생성과 비슷하게 POD 구성 시 아래와 같이 명령어를 사용한다.

```
# podman pod ls
# podman pod create
# podman pod ls
# podman pod create --name POD_http
```

# container 생성

생성된 Pod 및 컨테이너를 아래와 같은 명령어로 연결 혹은 새로 구성하면서 실행이 가능하다. 일단 Pod는 생성하지 않고, 컨테이너만 생성하여 동작이 올바르게 되는지 확인한다.

```
# podman container create --name httpd quay.io/centos7/httpd-24-  
centos7:centos7  
# podman container ls
```

# pod+container

Pod, Container 기반으로 컨테이너 인프라를 구현하려는 경우, 다음처럼 명령어를 실행한다. 포드(Pod)기반으로 컨테이너를 구현하는 경우, 볼륨, 네트워크 기능을 컨테이너 별로 분리 및 격리 운영이 가능하기 때문에 안전하게 사용이 가능하다.

또한, 쿠버네티스로 마이그레이션을 준비하는 경우, 포드 기반으로 생성하여 테스트 후, 손쉽게 마이그레이션 자원을 생성 할 수 있다.

```
# podman container create --name container_http --rm --pod new:pod_http
quay.io/centos7/httpd-24-centos7:centos7
# podman start container_httpd
# podman container ls
# podman generate kube pod_http --filename pod_http.yaml --service --
replicas 1 --type deployment
# ls pod_http.yaml
# kubectl apply -f pod_http.yaml
```

# 컨테이너 정보

## `/var/lib/containers/`

컨테이너에서 사용하는 이미지 및 컨테이너 정보가 저장되는 위치. 임시적인 정보 및 메타정보가 저장된다.

## `/run/containers/`

동작 시 사용되는 임시 정보만 저장된다. 리부팅이 되거나 재시작이 되면 해당 정보들은 삭제가 된다.

## `/etc/containers/`

OCI기반의 컨테이너 런타임 정보들은 위의 디렉터리에서 설정 및 구성이 된다. 예를 들어서 이미지를 받아오기 위한 정보인 레지스트리 서버도 이 디렉터리에서 구성이 된다.

## `/etc/cni/`

컨테이너에서 사용하는 컨테이너 네트워크 설정이다. 사용하는 런타임 혹은 오케스트레이션 프로그램 별로 구성 및 저장된다.

# 컨테이너 서비스화

리눅스에서 사용하는 flatpak, snap같은 패키지 서비스는 호스트에 설치 되는 방식이 아니다.

컨테이너 기반으로 애플리케이션 설치 및 실행을 한다. 이와 같은 방식으로 사용자가 만든 컨테이너를 애플리케이션 형태로 사용 및 동작이 가능하다.

앞에서 사용하였던, systemd-nspawn과는 다르다. systemd-nspawn는 boot-full 컨테이너를 구성하며, systemd-container 경우에는 systemd에서 서비스 형태로 구성한다.

항목	설명
Flatpak, Snap	컨테이너 기반 앱 설치/실행
systemd-nspawn	전체 OS를 컨테이너처럼 부팅
systemd-container	서비스처럼 컨테이너 관리 (systemd 서비스 단위로 운영)

# 컨테이너 서비스화

아래와 같이 간단하게 컨테이너를 포드만에서 생성한다. 아래는 간단하게 웹 서비스를 포드만으로 생성하며, 외부 포트 및 디렉터리를 볼륨으로 바인딩한다.

```
# mkdir /root/htdocs
# echo "Hello shell training" > /root/htdocs/index.html
# podman run -d --rm -p 8080:8080 -v /root/htdocs:/var/www/html/ --name
apache quay.io/centos7/httpd-24-centos7:centos7
# podman ps
# curl http://localhost:8080
```

# 컨테이너 서비스화

생성된 서비스를 systemd의 서비스로 구성한다.

```
# podman generate systemd --restart-policy=always -t 1 apache
# podman generate systemd --new --files --name apache
# mkdir -p $HOME/.config/systemd/user/
# cp container-apache.service $HOME/.config/systemd/user/
# systemctl daemon-reload --user
# systemctl --user enable --now apache.service
# systemctl -t service --user
$ loginctl enable-linger <USERNAME>
```

# buildah

포드만에서 이미지 빌드 부분이 독립이 되어서 나온 도구가 buildah이다. 컨테이너 이미지 빌드는 여전히 docker build, podman build 사용이 가능하다.

표준 컨테이너에서는 CI/CD를 통해서 이미지 빌드 시, 위의 도구보다는 가급적이면 buildah 사용을 권장한다. 현재 쿠버네티스는 이미지 생성 시, "Tekton + buildah"기반으로 권장한다.



# buildah

이미지 생성 방식은 다음과 같다.

## 1. Dockerfile

## 2. Containerfile

Dockerfile은 여전히 사용이 가능하지만, 오픈소스에서는 Containerfile으로 사용을 권고하고 있다.

Containerfile는 기존의 Dockerfile과 동일한 명령어 구조를 가지고 있다.

```
FROM fedora:latest
RUN echo "Installing httpd"; yum -y install httpd
EXPOSE 80
CMD ["/usr/sbin/httpd", "-DFOREGROUND"]
```

# buildah

혹은 명령어 방식으로 구성이 가능하다. 이 방식은 CD에서 종종 사용하기도 한다.

```
# buildah from centos
# buildah run centos-working-container yum install httpd -y
# echo "Hello from the blackcat" > index.html
# buildah copy centos-working-container index.html /var/www/html/index.html
# buildah config --entrypoint "/usr/sbin/httpd -DFOREGROUND" centos-
working-container
# buildah commit centos-working-container centos-website
```

# buildah

이미지를 처음부터 빌드하는 경우, 아래처럼 명령어를 실행한다.

```
# newcontainer=$(buildah from scratch)
# buildah containers
# scratchmnt=$(buildah mount $newcontainer)
# echo $scratchmnt
# dnf install -y --releasever=8 --installroot=$scratchmnt redhat-release --
nogpgcheck
# yum install -y --setopt=reposdir=/etc/yum.repos.d --
installroot=$scratchmnt --setopt=cachedir=/var/cache/dnf httpd --nogpgcheck
```

# buildah

위의 내용 계속 이어서...

```
# mkdir -p $scratchmnt/var/www/html
# echo "Your httpd container from scratch
works!" > ?$scratchmnt/var/www/html/index.html
# buildah config --cmd "/usr/sbin/httpd -DFOREGROUND" working-container
# buildah config --port 80/tcp working-container
# buildah commit working-container localhost/myhttpd:latest
```

# skopeo

이미지 레지스트리에 있는 이미지를 검색하거나 혹은 복사 시, 많은 제약이 있다. 특히, 도커 및 포드만 경우에는 상황에 따라서 프로토콜 버전에 따라서 검색이 잘 되지 않는다. 이러한 문제로 search 부분을 분리하여 만든 도구가 skopeo이다.

사용방법은 간단하게 다음과 같다.

```
# skopeo copy docker://quay.io/skopeo/stable:latest
docker://registry.example.com/skopeo:latest
# mkdir -p /var/lib/images/busybox
# skopeo copy docker://busybox:latest dir:/var/lib/images/busybox
# skopeo copy docker://busybox:latest docker-archive:archive-
file.tar:busybox:latest
# skopeo sync --src docker --dest dir registry.example.com/busybox
/media/usb
# skopeo list-tags docker://docker.io/fedora
```

# 리눅스 가상화

libvirt

# 리눅스 가상화

현재 리눅스에서 표준적으로 두 가지 가상화 형식을 제공하고 있다.

1. 하이퍼바이저 형식
2. 커널 기반 하이퍼바이저 형식

리눅스에서 많이 사용하는 QEMU/KVM는 커널 기반 하이퍼바이저 형식이다. 다만, QEMU/KVM기반으로만 가상 머신을 구현하는 경우 관리가 어렵기 때문에 다음과 같은 도구를 통해서 가상머신을 관리한다.

1. libvirt
2. libguestfs
3. virt-\*(install, manager, p2v, top...)

위와 같은 도구를 통해서 가상머신 생성 및 관리를 한다.

# 가상화 소프트웨어

가상화 소프트웨어를 설치하기 위해서 기본적으로 다음과 같은 그룹 패키지 설치를 권장한다.

```
# dnf group list  
# dnf group install "Virtualization Host" -y
```

설치가 완료가 되면 가상머신 생성 및 관리하는 라이브러리 대몬(libvirtd)을 실행한다.

```
# systemctl enable --now libvirtd
```

서비스가 올바르게 실행이 되면, 다음 명령어로 가상화 자원을 확인한다.

```
# virsh list  
# virsh net-list  
# virsh pool-list
```



# libvirt

libvirt에서 사용하는 디렉터리는 다음과 같다.

1. `/var/lib/libvirt/`

2. `/etc/libvirt/`

두 개의 디렉터리는 가상머신에서 사용하는 네트워크 정보 및 가상머신의 XML파일 그리고, 가상머신 이미지를 가지고 있다. 설정 파일은 `/etc/libvirt`를 사용하며, 가상머신 디스크는 일반적으로 `/var/lib/libvirt`를 사용한다.

libvirt는 리눅스 가상화 솔루션, 예를 들어서 오픈스택, oVirt, VMware, Xen와 같은 하이퍼바이저는 드라이버로 구성이 되어 동작한다.

# 명령어

가상머신 이미지 빌드 생성은 여기서 다루지는 않으며, 가상머신에서 사용하는 이미지는 `virt-builder`를 통해서 내려받기 한다. `virt-builder`는 별도의 설정이 없으면 `libguestfs`에 제공해주는 디스크 이미지를 사용한다.

별도로 웹 서버 기반으로 **내부용 이미지 서버(virtual disk image registry server)**를 구성해서 배포용으로 사용이 가능하다. 아래 명령어로 이미지 구성한다.

```
# virt-builder --list
# virt-builder cirros-0.3.5 --output /var/lib/libvirt/images/cirros.raw
# virt-builder --root-password password:centos --size 10G --format qcow2
centosstream-9 --output /var/lib/libvirt/images/centosstream-9.qcow2
# qemu-img info /var/lib/libvirt/images/centosstream-9.qcow2
```

# 명령어

가상머신에서 사용할 패키지 및 가상머신을 생성한다. 가상머신 이미지가 문제 없이 구성이 되면서, 가상머신을 하나씩 올리도록 한다.

```
# dnf install virt-install -y
# virt-install --osinfo list | grep -e centos -e cirros
# virt-install --name cirros --vcpu 2 --memory 512 --
disk=path=/var/lib/libvirt/images/cirros.raw --osinfo=cirros0.3.0 --import
--noautoconsole --network=network=default --graphics
vnc,port=5901,listen=0.0.0.0 --destroy-on-exit
# virt-install --name centosstream-9 --vcpu 2 --memory 1536 --
disk=path=/var/lib/libvirt/images/centosstream-9.qcow2 --osinfo=centos-
stream9 --import --noautoconsole --network=network=default --graphics
vnc,port=5902,listen=0.0.0.0 --destroy-on-exit
```

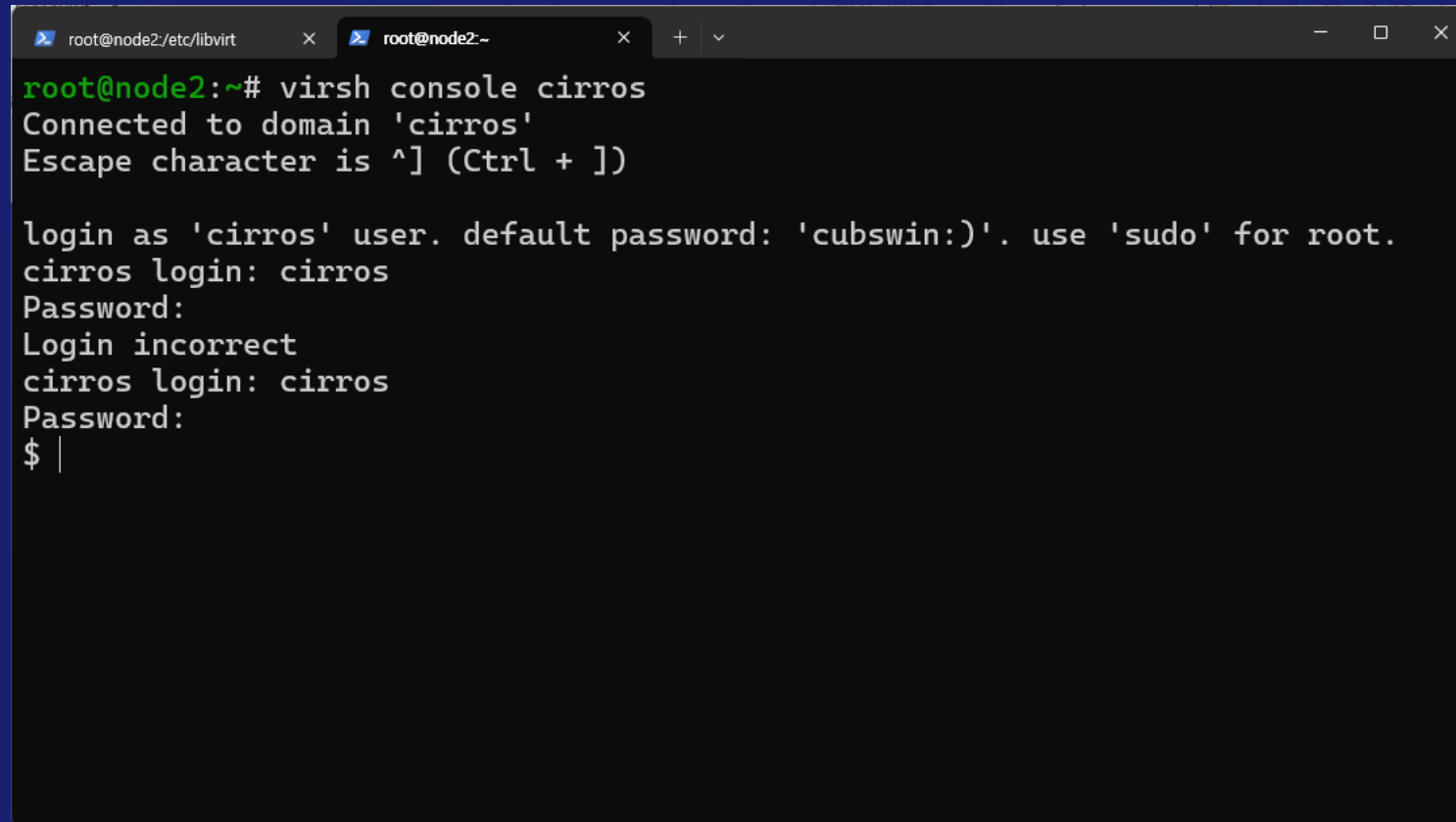
# console(vnc)

```
QEMU (cirros) - TigerVNC
[ 1.182628] Registering the dns_resolver key type
[ 1.184844] registered taskstats version 1
[ 1.239591] Magic number: 4:100:535
[ 1.239877] bdi 1:13: hash matches
[ 1.239940] tty ttyS15: hash matches
[ 1.240517] rtc_cmos 00:08: setting system clock to 2024-04-02 14:31:01 UTC (
1712068261)
[ 1.240705] powernow-k8: Processor cpuid 663 not supported
[ 1.240847] powernow-k8: Processor cpuid 663 not supported
[ 1.242027] BIOS EDD facility v0.16 2004-Jun-25, 0 devices found
[ 1.242107] EDD information not available.
[ 1.287723] Freeing unused kernel memory: 928k freed
[ 1.329762] Write protecting the kernel read-only data: 12288k
[ 1.349502] Freeing unused kernel memory: 1596k freed
[ 1.364951] Freeing unused kernel memory: 1184k freed

further output written to /dev/ttyS0

login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
cirros login:
login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
cirros login:
login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
cirros login:
```

# console(serial)



```
root@node2:/etc/libvirt x root@node2:~ x + v - □ x
root@node2:~# virsh console cirros
Connected to domain 'cirros'
Escape character is ^] (Ctrl + ])

login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
cirros login: cirros
Password:
Login incorrect
cirros login: cirros
Password:
$ |
```