

# RAID 소개

RAID(Redundant Array of Independent Disks 또는 Redundant Array of Inexpensive Disks)  
데이터 중복성, 성능 개선 또는 둘 다를 목적으로 여러 물리적 디스크 드라이브 구성 요소를 하나 이상의 논리 장치로 결합하는 데이터 스토리지 가상화 기술입니다.

데이터는 필요한 중복성 및 성능 수준에 따라 RAID level 이라고 하는 여러 방법 중 하나로 드라이브에 분산됩니다.  
다양한 설계(schemes) 또는 데이터 분포 레이아웃은 "RAID"라는 단어 뒤에 숫자가 오는 방식으로 이름이 지정됩니다.  
각 설계 또는 RAID 레벨은 안정성(reliability), 가용성(availability), 성능(performance), 용량(capacity)이라는 주요 목표 간에 서로 다른 균형을 제공합니다.

많은 RAID 레벨은 주어진 데이터 세트에 결함 허용(fault tolerance) 을 제공하기 위해 널리 사용되는 방법인 "parity"라는 오류 보호 체계를 사용합니다.  
대부분은 간단한 XOR(Exclusive or)을 사용

RAID는 또한 전체 SSD 시스템에 대한 비용을 들이지 않고도 SSD(Solid-State Drive)를 통해 데이터 보안을 제공할 수 있습니다.  
예를 들어 고속 SSD는 기계식 드라이브로 미러링 될 수 있습니다. 이 구성이 상당한 속도 이점을 제공하려면 모든 읽기 작업에 빠른 SSD를 사용하는 적절한 컨트롤러가 필요  
합니다.  
Adaptec 에서는 이를 "하이브리드 RAID(hybrid RAID)"라고 부릅니다.

## Hardware RAID 와 Software RAID

### 1. Hardware RAID

RAID 컨트롤러 또는 RAID 카드라고 불리는 전용 하드웨어를 사용하여 운영체제로부터 독립적으로 RAID를 설정하고 관리할 수 있습니다. 이를 하드웨어 RAID라고 합니다.  
진짜 하드웨어 RAID 컨트롤러에는 RAID 장치 관리를 위한 전용 프로세서가 있습니다.

#### 하드웨어 RAID의 장점

- 성능(Performance) : 정품 하드웨어 RAID 컨트롤러는 기본 디스크를 관리하기 위해 CPU 주기를 차지할 필요가 없습니다.  
이는 연결된 저장 장치 관리에 대한 오버헤드가 없음을 의미합니다.  
고품질 컨트롤러는 성능에 큰 영향을 미칠 수 있는 광범위한 캐싱(caching)도 제공합니다.
- 추상화(Abstracting away complexity) : RAID 컨트롤러를 사용하는 또 다른 이점은 운영 체제에서 기본 디스크 배열을 추상화한다는 것입니다.  
하드웨어 RAID는 전체 드라이브 그룹을 단일 논리 스토리지 장치로 나타낼 수 있습니다.  
운영 체제는 RAID 배열을 이해할 필요가 없습니다. 마치 단일 장치인 것처럼 array와 인터페이스할 수 있습니다.
- 부팅시 가용성(Availability at boot) : array는 완전히 소프트웨어 외부에서 관리되므로 부팅 시 사용할 수 있으므로 루트 파일 시스템 자체를 RAID array에 쉽게 설치할 수  
있습니다.

#### 하드웨어 RAID의 단점

- 공급업체 종속(Vendor lock-in) : RAID array는 하드웨어 자체 소유자 펌웨어(firmware)에 의해 관리되기 때문에 array는 이를 생성하는 데 사용된 하드웨어에 어느 정도 고  
정되어 있습니다.  
RAID 컨트롤러가 수명을 다하면 거의 모든 경우에 동일하거나 호환되는 모델로 교체해야 합니다.  
일부 관리자는 첫 번째 문제가 발생할 경우를 대비해 하나 이상의 백업 컨트롤러를 구입하도록 권장합니다.
- 고비용(High cost) : 고품질 하드웨어 RAID 컨트롤러는 상당히 비싼 경향이 있습니다.

### 2. Software RAID

RAID는 운영 체제 자체에서 구성할 수도 있습니다. 디스크 간의 관계는 하드웨어 디바이스의 펌웨어가 아닌 운영 체제 내에서 정의되므로 이를 소프트웨어 RAID라고 합니다.

#### 소프트웨어 RAID의 장점

- 유연성(Flexibility) : RAID는 운영 체제 내에서 관리되므로 하드웨어를 재구성하지 않고도 실행 중인 시스템에서 사용 가능한 스토리지에서 쉽게 구성할 수 있습니다.  
Linux 소프트웨어 RAID는 특히 유연하여 다양한 유형의 RAID 구성이 가능합니다.
- 오픈 소스(Open source) : Linux 및 FreeBSD와 같은 오픈 소스 운영 체제에 대한 소프트웨어 RAID 구현도 오픈 소스입니다.  
RAID 구현(Implementations)은 숨겨져 있지 않으며 다른 시스템에서 쉽게 읽고 구현할 수 있습니다.  
예를 들어 Ubuntu 시스템에서 생성된 RAID 어레이는 나중에 CentOS 서버로 쉽게 가져올 수 있습니다.  
소프트웨어 차이로 인해 데이터에 대한 액세스가 손실될 가능성은 거의 없습니다.
- No additional costs(추가 비용 없음) : 소프트웨어 RAID에는 특수 하드웨어가 필요하지 않으므로 서버나 워크스테이션에 추가 비용이 추가되지 않습니다.

#### 소프트웨어 RAID의 단점

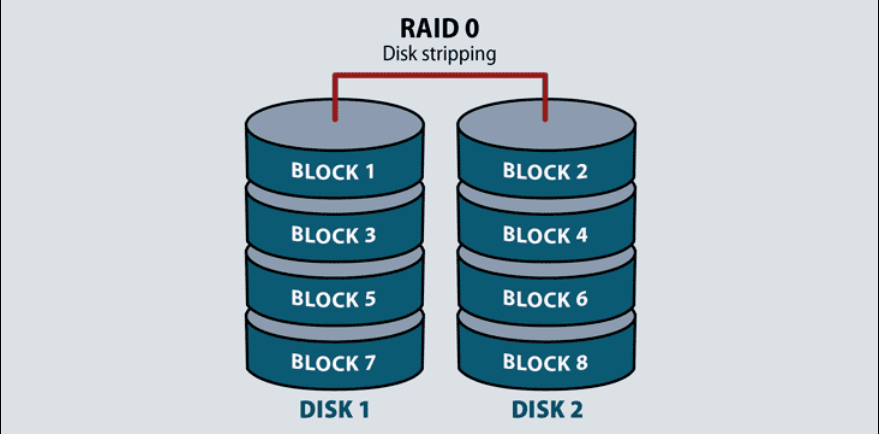
- Implementation-specific : 소프트웨어 RAID는 특정 하드웨어에 묶여있지 않지만 RAID의 특정 소프트웨어에 묶여있는 경향이 있습니다.  
Linux는 mdadm을 사용하고 FreeBSD는 GEOM 기반 RAID를 사용하며 Windows에는 자체 소프트웨어 RAID 버전이 있습니다.
- Performance overhead : 역사적으로 소프트웨어 RAID는 추가 오버헤드를 발생시킨다는 이유로 비판을 받아왔습니다.  
array를 관리하려면 CPU 사이클과 메모리가 필요하며, 이는 다른 목적으로 사용될 수 있습니다.  
그러나 최신 하드웨어에서 mdadm과 같은 구현은 이러한 우려를 대부분 무효화합니다. CPU 오버헤드는 최소화되며 대부분의 경우 미미합니다.

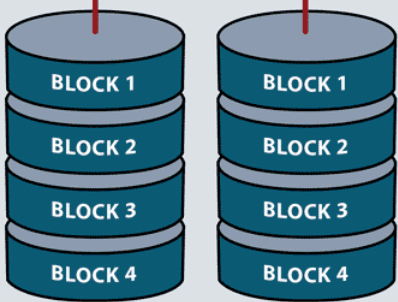
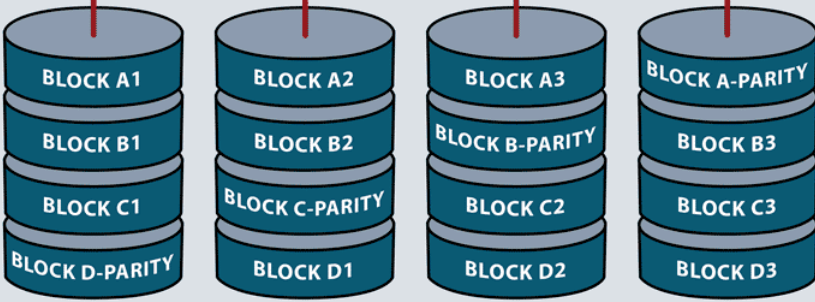
## 용어 및 설명

용어	설명
RAID level	array의 RAID 레벨은 구성 요소 저장 디바이스에 부과되는 관계를 의미합니다. 드라이브는 다양한 방식으로 구성될 수 있으며, 이는 서로 다른 데이터 중복성 및 성능 특성으로 이어집니다.
Striping	스트라이핑(Striping)은 array에 대한 쓰기(write)를 여러 기본 디스크로 나누는 프로세스입니다. 이 전략은 다양한 RAID 레벨에서 사용됩니다. 데이터가 array 전체에 스트라이프되면 데이터가 chunk 로 분할되고 각각의 chunk는 기본 디바이스들 중 적어도 하나에 기록됩니다.
Chunk Size	데이터를 스트라이핑할 때 chunk 크기는 각 chunk에 포함될 데이터의 양을 결정합니다. 예상되는 I/O 특성에 맞게 chunk 크기를 조정하면 array의 상대적 성능에 영향을 줄 수 있습니다.
Parity	패리티(Parity)는 array에 기록된 데이터 블록들로부터 정보를 계산함으로써 구현되는 데이터 무결성 메커니즘입니다. 패리티 정보는 드라이브에 장애가 발생하면 데이터를 재구성하는 데 사용될 수 있습니다. 계산된 패리티는 그것이 계산된 데이터로부터 계산된 것과는 별개의 장치에 배치되며 대부분의 구성에서는, 더 나은 성능 및 중복성을 위해 사용 가능한 드라이브들에 걸쳐 분산됩니다.
Degraded Arrays	중복성(redundancy)이 있는 array는 데이터 손실 없이 다양한 유형의 드라이브 장애를 겪을 수 있습니다. array에서 디바이스가 손실되었지만 여전히 동작 중인 경우, 성능 저하 모드(degraded mode)에 있다고 합니다. 성능이 저하된 array는 결함이 있는 하드웨어를 교체하면 성능이 저하된 array를 완전히 작동 가능한 상태로 재구축할 수 있지만, 그 사이에 성능이 저하될 수 있습니다.
Resilvering	리실버링(Resilvering) 또는 재동기화(resyncing)는 성능이 저하된 array를 재구축하는 데 사용되는 용어입니다. RAID 구성 및 오류의 영향에 따라 이는 array에 있는 기존 파일에서 데이터를 복사하거나 패리티 정보를 평가하여 데이터를 계산하여 수행됩니다.
Nested Arrays	RAID array 그룹을 더 큰 array로 결합할 수 있습니다. 일반적으로 두 개 이상의 서로 다른 RAID 레벨의 기능을 활용하기 위해 수행됩니다. 일반적으로, 중복성을 가진 array(예: RAID 1 또는 RAID 5)는 성능 향상을 위해 RAID 0 array를 생성하는 구성 요소로 사용됩니다.
Span	<ul style="list-style-type: none"> <li>안타깝게도 array를 논의할 때 span은 몇 가지 다른 의미를 갖습니다. 특정 상황에서 "span"은 두 개 이상의 디스크를 end-to-end로 결합하여 하나의 논리적 장치로 나타내는 것을 의미할 수 있으며 성능이나 중복성이 향상되지 않습니다. 이것은 리눅스의 mdadm 구현을 다룰 때 선형 배열(linear arrangement)이라고도 합니다.</li> <li>"span"은 RAID 10과 같은 중첩된 RAID 레벨에 대해 논의할 때 다음 계층(tier)을 형성하기 위해 결합된 하위 계층의 array를 의미할 수도 있습니다.</li> </ul>
Scrubbing	Scrubbing 또는 checking은 일관성 오류가 없는지 확인하기 위해 array의 모든 블록(block)을 읽는 프로세스입니다. 이를 통해 저장 장치 간에 데이터가 동일하도록 보장하고, 특히 재구축과 같은 민감한 절차 중에 표시되지 않는 오류로 인해 손상이 발생할 수 있는 상황을 방지할 수 있습니다.

RAID Levels

array의 특성은 RAID 레벨이라고 하는 디스크의 구성 및 관계에 따라 결정됩니다. 가장 일반적인 RAID 레벨은 다음과 같습니다.

레벨	설명
RAID 0	<p>RAID 0은 두 개 이상의 디바이스에 걸쳐 데이터를 스트라이핑하여 결합합니다. 위에서 언급했듯이 스트라이핑은 데이터를 chunk로 나눈 다음 그 chunk를 array의 각 디스크에 번갈아 저장하는 기술입니다. 장점은 데이터가 분산되어 있기 때문에 읽기와 쓰기 모두 각 디바이스의 전체 성능을 활용할 수 있다는 것입니다. RAID 0 array의 이론적인 성능 프로파일은 단순히 개별 디스크의 성능에 디스크 수를 곱한 것입니다(실제 성능은 이보다 낮습니다). 또 다른 장점은 array의 사용 가능 용량이 단순히 모든 구성 드라이브의 용량을 합친 것이라는 점입니다.</p> <p>이 접근 방식은 뛰어난 성능을 제공하지만 몇 가지 중요한 단점도 있습니다. 데이터는 array의 각 디스크에 분할되어 나누어지기 때문에 단일 장치에 오류가 발생하면 전체 array가 다운되고 모든 데이터가 손실됩니다. 대부분의 다른 RAID 레벨과 달리 RAID 0 array는 데이터를 재구성할 수 없습니다. 구성 요소 장치의 하위 집합에는 데이터를 재구성하는 데 필요한 콘텐츠에 대한 충분한 정보가 포함되어 있지 않기 때문입니다. RAID 0 array를 실행하는 경우 전체 데이터 세트가 array의 각 디스크의 신뢰성에 동일하게 의존하기 때문에 백업이 매우 중요해집니다.</p> 
RAID 1	<p>RAID 1은 둘 이상의 디바이스 사이에서 데이터를 미러링하는 구성입니다. array에 기록된 모든 것은 그룹의 각 디바이스에 배치됩니다. 즉, 각 디바이스에는 사용 가능한 데이터의 완전한 세트가 있어 장치 오류가 발생할 경우 redundancy(중복성/이중화) 기능이 제공됩니다. RAID 1 array에서는 array의 여러 디바이스중 1개만 제대로 작동하는 한 데이터에 액세스할 수 있습니다. 문제가 발생한 드라이브를 교체하여 array를 재구축할 수 있으며, 이 시점에서 나머지 장치는 데이터를 새 장치에 다시 복사하는 데 사용됩니다.</p> <p>RAID 1 구성에도 몇 가지 단점이 있습니다. RAID 0과 마찬가지로 이론적인 읽기(read) 속도는 개별 디스크의 읽기 속도에 디스크 수를 곱하여 계산할 수 있습니다. 그러나 쓰기(write) 작업의 경우 이론적으로 최대 성능은 array의 디바이스 중에서 가장 느린 디바이스의 성능입니다. 이는 전체 데이터 조각이 array의 각 디스크에 기록되어야 하기 때문입니다. 또한 array의 총 용량은 가장 작은 디스크의 용량이 됩니다. 따라서 크기가 같은 두 개의 장치가 있는 RAID 1 array는 단일 디스크의 사용 가능한 용량을 갖게 됩니다. 디스크를 추가하면 데이터의 중복 복사본 수는 증가할 수 있지만 사용 가능한 용량은 증가하지 않습니다.</p>

레벨	설명
	<div> <div> <div>RAID 1</div> <div>Disk mirroring</div>  <div>DISK 1DISK 2</div> </div> </div>
RAID 2	생략
RAID 3	생략
RAID 4	생략
RAID 5	<p>RAID 5는 RAID 0과 RAID 1의 일부 기능을 가지고 있지만 성능 프로파일과 단점이 다릅니다. RAID 5에서 데이터는 RAID 0 array와 거의 동일한 방식으로 데이터가 디스크 전체에 스트라이프됩니다. 그러나 array 전체에 걸쳐 기록되는 데이터의 각 스트라이프에 대해 오류 수정 및 데이터 재구성에 사용할 수 있는 수학적으로 계산된 값인 패리티(parity) 정보가 디스크 중 하나에 기록됩니다. 데이터 블록(block) 대신 계산된 패리티 블록을 수신하는 디스크는 기록된 각 스트라이프와 함께 회전합니다.</p> <p>이것은 몇 가지 중요한 이점을 가지고 있습니다. 스트라이핑을 사용하는 다른 array와 마찬가지로 읽기 성능은 여러 디스크에서 한 번에 읽을 수 있는 기능의 이점을 제공합니다. RAID 5 array는 array에 있는 디스크 하나의 손실을 처리합니다. 이러한 경우 패리티 블록을 사용하면 데이터를 완전히 재구성할 수 있습니다. 패리티는 각 디스크에 분산되므로(일부 일반적이지 않은 RAID 레벨은 전용 패리티 드라이브를 사용함) 각 디스크에는 균형 잡힌 양의 패리티 정보가 있습니다. RAID 1 array의 용량은 단일 디스크(모든 디스크에 동일한 데이터 복사본이 있음)의 크기로 제한되는 반면, RAID 5 패리티를 사용하면 단일 디스크의 용량만큼의 공간을 희생하더라도 중복성 수준을 달성할 수 있습니다. 따라서 RAID 5 array에 있는 4개의 100G 드라이브는 300G의 사용 가능한 공간을 제공합니다.(나머지 100G는 분산 패리티 정보가 차지함)</p> <p>다른 레벨과 마찬가지로 RAID 5에도 몇 가지 중요한 단점이 있습니다. 실시간 패리티 계산으로 인해 시스템 성능이 상당히 느려질 수 있습니다. 이는 각 쓰기 작업에 영향을 줄 수 있습니다. 디스크에 오류가 발생하고 array가 성능 저하 상태에 들어가는 경우 읽기 작업에 상당한 불이익이 발생합니다(누락된 데이터는 나머지 디스크에서 계산해야 함). 게다가 오류가 발생한 드라이브를 교체한 후 array를 복구할 때 각 드라이브를 읽고 CPU를 사용하여 누락된 데이터를 계산하여 누락된 데이터를 재구성해야 합니다. 이로 인해 나머지 드라이브에 스트레스가 가해질 수 있으며 때로는 추가적인 오류가 발생하여 모든 데이터가 손실될 수 있습니다.</p> <div> <div> <div>RAID 5</div> <div>Disk stripping with parity</div>  <div>DISK 0DISK 1DISK 2DISK 3</div> </div> </div>
RAID 6	<p>RAID 6은 RAID 5와 유사한 아키텍처를 사용하지만 이중(double) 패리티 정보를 사용합니다. 이는 array가 두 개의 디스크에서 장애가 발생해도 견딜 수 있음을 의미합니다. 이는 오류가 발생한 후 집중적인 재구축 과정에서 추가적인 디스크 장애가 발생할 가능성이 높기 때문에 상당한 이점이 있습니다. 스트라이핑을 사용하는 다른 RAID 레벨과 마찬가지로 읽기 성능은 일반적으로 좋습니다. RAID 5의 다른 모든 장점은 RAID 6에도 존재합니다.</p> <p>단점은 RAID 6은 추가 디스크 용량으로 추가 더블 패리티를 지불합니다. 즉, array의 총 용량은 array에 포함된 드라이브의 총 공간에서 2개의 드라이브를 제외한 공간입니다. RAID 6의 패리티 데이터를 결정하는 계산은 RAID 5보다 복잡하므로 RAID 5보다 쓰기 성능이 저하될 수 있습니다. RAID 6은 RAID 5와 동일한 성능 저하 문제를 겪고 있지만, 추가 디스크의 중복성으로 인해 재구축 작업 중에 데이터가 삭제되는 추가 오류 가능성을 방지할 수 있습니다.</p>

레벨	설명
	<div data-bbox="266 64 1147 499"> </div> <p>많은 스토리지 컨트롤러가 RAID 레벨의 중첩을 허용합니다. RAID의 구성 요소는 개별 드라이브 또는 array 자체일 수 있습니다. array가 한 레벨 이상 깊게 중첩되는 경우는 거의 없습니다. 최종 array를 최상위 array라고 합니다. 최상위 array가 RAID 0(예: RAID 1+0, RAID 5+0)인 경우 대부분의 공급업체는 "+"를 생략합니다. (각각 RAID10, RAID50을 생성)</p> <ul style="list-style-type: none"> <li> <b>RAID 0+1:</b>  스트라이프의 미러링  1개의 드라이브에 오류가 발생하면 스트라이프 중 하나에 오류가 발생한 것입니다. 이 시점에서 redundancy(중복성/이중화) 없이 나머지 살아있는 AID 0으로 효과적으로 실행됩니다.  한 드라이브에서만 읽는 것이 아니라 나머지 스트라이프에 있는 모든 드라이브의 모든 데이터를 읽어야 하므로 재구축 과정에서 RAID 1+0 보다 훨씬 높은 위험이 발생합니다.  복구 불가능한 읽기 오류(URE: Unrecoverable Read Error)가 발생할 가능성이 높아지고 재구축 기간을 대폭 연장합니다. </li> <li> <b>RAID 1+0:</b>  미러링의 스트라이프  이러한 타입의 구성은 각 RAID 1의 디스크 중 하나 이상을 사용할 수 있는한 미러링된 RAID 1 세트의 디스크 오류를 처리할 수 있습니다. 전체에 걸친 array는 불균형한 방식으로 내결함성을 갖는데, 이는 발생하는 위치에 따라 다양한 수의 장애를 처리할 수 있음을 의미합니다. </li> </ul> <p>Linux의 <code>mdadm</code>은 RAID 1+0의 정신과 이점을 구현하는 자체 버전의 RAID 10을 제공하지만 실제 구현을 보다 유연하게 변경하고 몇 가지 추가적인 이점을 제공합니다.  RAID 1+0과 마찬가지로 <code>mdadm</code> RAID 10은 여러 개의 복사본과 스트라이프 데이터를 허용합니다.  그러나 디바이스는 미러링된 쌍으로 정렬되지 않습니다. 대신 관리자는 해당 array에 기록될 복사본의 수를 결정합니다.  데이터는 여러 개의 복사본으로 array 전체에 걸쳐 chunk로 나누어 기록되므로 chunk의 각 복사본이 서로 다른 물리적 디바이스에 기록됩니다.  결과적으로 동일한 수의 복사본이 존재하지만 array는 기본 중첩에 의해 크게 제한되지 않는다는 것입니다.</p> <div data-bbox="266 1167 1171 1619"> </div>
Nested (hybrid) RAID	

## Linux mdadm 명령

**mdadm [mode] <raiddevice> [options] <component-devices>**

RAID 디바이스는 둘 이상의 실제 블록 디바이스로부터 생성된 가상 디바이스입니다. 이를 통해 여러 디바이스(일반적으로 디스크 드라이브 또는 그것의 파티션)를 단일 장치로 결합하여 단일 파일 시스템을 유지할 수 있습니다. 일부 RAID 레벨에는 redundancy(중복성/이중화)가 포함되기 때문에 어느 정도 디바이스 오류가 발생해도 살아남을 수 있습니다.

Linux 소프트웨어 RAID 장치는 md(Multiple Devices) 장치 드라이버를 통해 구현됩니다.

현재 Linux는 **LINEAR** md 장치, **RAID0** (striping), **RAID1** (mirroring), **RAID4**, **RAID5**, **RAID6**, **RAID10**, **MULTIPATH**, **FAULTY**, **CONTAINER** 를 지원합니다.

**MULTIPATH**는 소프트웨어 RAID 메커니즘은 아니지만 여러 장치를 포함합니다. 각 장치는 하나의 공통 물리적 스토리지 장치로 연결되는 경로입니다. 새로 설치하는 경우에는 md/multipath가 제대로 지원되지 않고 진행 중인 개발이 없으므로 사용해서는 안됩니다. 대신 Device Mapper 기반 다중 경로 도구(multipath-tools)를 사용하십시오.

**FAULTY**도 실제 RAID가 아니며 하나의 장치에만 관련됩니다. 결함을 주입하는 데 사용할 수 있는 실제 장치 위에 레이어(layer)를 제공합니다.

CONTAINER는 또 다릅니다. 세트는 세트 관리되는 장치의 컬렉션입니다. 이는 하드웨어 RAID 컨트롤러에 연결된 장치 세트와 유사합니다. 장치 세트에는 세트 내의 다수의 디바이스들로부터의 블록들 중 일부(또는 전부)를 각각 이용하는 다수의 상이한 RAID array들을 포함할 수도 있습니다. 예를 들어, 5-디바이스 세트에 있는 두 개의 디바이스는 전체 디바이스를 사용하여 RAID 1을 형성할 수 있습니다. 나머지 세 개는 각 디바이스의 전반부에 RAID 5를, 후반부에 RAID 0을 가질 수 있습니다.

MODES	설명
Assemble	이전에 생성된 array의 구성 요소를 active array로 조립합니다. 구성요소는 명시적으로 제공되거나 검색될 수 있습니다. mdadm은 구성 요소가 진정한 array를 형성하는지 확인하고, 요청시 결함이 있는 array를 조립하기 위해 슈퍼블록(superblock) 정보를 조작할 수 있습니다.
Build	장치별 메타데이터(superblocks)가 없는 array를 구축합니다. 이러한 종류의 array의 경우 mdadm은 array의 초기 생성과 이후의 조립을 구별할 수 없습니다. 또한 적절한 구성 요소가 요청되었는지 확인하는 작업도 수행할 수 없습니다. 따라서 Build 모드는 수행 중인 작업을 완전히 이해한 후에만 사용해야 합니다.
Create	장치별 메타데이터(superblocks)를 사용하여 새 array를 생성합니다. 적절한 메타데이터가 각 장치에 기록된 다음 해당 장치를 포함하는 array가 활성화됩니다. array가 일관성이 있는지(예: 미러링의 양쪽에 동일한 데이터가 포함되어 있는지) 확인하기 위해 'resync' 프로세스가 시작되지만 장치의 내용물은 손을 대지 않는한 그대로 유지됩니다. array는 생성되자마자 사용할 수 있습니다. 초기 재동기화(initial resync)가 완료될 때까지 기다릴 필요가 없습니다.
Follow or Monitor	하나 이상의 md(Multiple Devices) 디바이스를 모니터링하고 모든 상태 변경에 대해 조치를 취합니다. 이는 RAID 1, 4, 5, 6, 10 또는 multipath arrays에만 의미가 있습니다. 이들 array에만 관심 있는 상태가 있기 때문입니다. RAID 0 또는 Linear는 드라이브가 없거나 여분의(spare) 드라이브 또는 오류가 있는 드라이브가 없으므로 모니터링할 것이 없습니다.
Grow	array를 성장(또는 축소)하거나 다른 어떤 방식으로 모양을 변경합니다. 현재 지원되는 grow 옵션에는 구성 요소 디바이스의 active 사이즈 변경, Linear 및 RAID 레벨 0/1/4/5/6의 active 디바이스 개수 변경, RAID 레벨 0,1,5 와 6 사이, 0과 10 사이의 변경, RAID 0,4,5,6,10의 chunk 사이즈 및 레이아웃 변경, 쓰기 전용 비트맵 추가 또는 제거, array의 일관성 정책(consistency policy) 변경 등이 포함됩니다.
Incremental Assembly(증가하는 조립)	적절한 array에 단일 디바이스를 추가합니다. 장치를 추가하여 array를 실행할 수 있게 되면 array가 시작됩니다. 이것은 핫플러그(hot-plug) 시스템에 편리한 인터페이스를 제공합니다. 각각의 디바이스가 감지되면 mdadm은 해당 장치를 일부 array에 적절하게 포함할 수 있습니다. 선택적으로 --fail 플래그값 전달되면 디바이스를 추가하는 대신 active array에서 디바이스를 제거합니다.  이 모드에서 CONTAINER가 mdadm에 전달되면 해당 컨테이너 내의 모든 array들이 조립되고(assembled) 시작됩니다.
Manage	이는 새로운 예비 부품을 추가하고 결함이 있는 장치를 제거하는 등 array의 특정 구성 요소에 대한 작업을 수행하기 위한 것입니다.
Misc	이는 active array에 대한 작업, 오래된 슈퍼블록 지우기와 같은 구성 요소 장치에 대한 작업, 정보 수집 작업을 지원하는 'everything else'(기타 모든 것) 모드입니다.
Auto-detect	이 모드는 특정 장치나 array에 대해 작동하는 것이 아니라, Linux 커널에 자동으로 감지된 array를 활성화하도록 요청합니다.

mode 선택 옵션	설명
-A, --assemble	기존 array를 조립
-B, --build	슈퍼블록(superblocks) 없이 레거시 array를 구축
-C, --create	새 array를 생성
-F, --follow, --monitor	Monitor 모드를 선택
-G, --grow	active array의 크기나 모양을 변경
-I, --incremental	적절한 array에 단일 장치를 추가/제거하고 array를 시작할 수도 있음
--auto-detect	별로 권장하는 방법이 아님!!!  커널이 자동 감지된 array를 시작하도록 요청합니다. 이는 md가 커널로 컴파일된 경우에만 작동할 수 있습니다. 모듈(module)인 경우 작동하지 않습니다. array의 모든 구성 요소가 파티션 타입이 FD인 기본 MS-DOS 파티션에 있고, 모두 v0.90 메타데이터를 사용하는 경우 커널이 array를 자동 감지할 수 있습니다.

mode 지정이 아닌 옵션 (일부 옵션은 생략했음...)	설명
-c, --config=	구성 파일(또는 디렉터리)를 지정합니다. 기본값은 /etc/mdadm.conf 와 /etc/mdadm.conf.d 를 사용합니다. 이 파일이 없는 경우 /etc/mdadm/mdadm.conf 및 /etc/mdadm/mdadm.conf.d 를 사용합니다. 지정된 구성 파일이 파티션인 경우 아무것도 읽히지 않지만 mdadm은 구성 파일이 포함된 것처럼 작동합니다. 그리고 mdadm은 /proc/partitions를 읽어 스캔할 장치 목록을 찾고, /proc/mdstat를 읽어 검사할 컨테이너 목록을 찾습니다. none이라고 지정하면 mdadm은 구성 파일이 비어 있는 것처럼 작동합니다.  지정된 이름이 디렉터리인 경우 mdadm은 그 디렉터리에서 이름이 '.conf' 로 끝나는 모든 파일을 수집하여 사전순으로 정렬하고 모두 구성 파일로 처리합니다.
-s, --scan	누락된 정보가 있는지 구성 파일(config file) 또는 /proc/mdstat를 스캔합니다. 일반적으로 이 옵션은 구성 파일에서 누락된 정보(예: 구성 요소 장치, array 장치, array ID, 경고대상)를 가져올 수 있는 권한을 mdadm에 제공합니다(이전 옵션 참고).
-e, --metadata=	사용할 RAID metadata(superblock)의 스타일을 선언합니다. --create의 기본값은 1.2이고 다른 작업을 추측하기 위한 것입니다. mdadm.conf에서 CREATE 키워드에 대한 메타데이터 값을 설정하여 기본 값을 재정의할 수 있습니다.
--homehost=	이는 구성 파일의 모든 HOMEHOST 설정을 무시하고 모든 array의 home으로 간주되어야 하는 호스트의 ID를 제공합니다.
For create, build, or grow (일부 옵션은 생략했음...)	
-n, --raid-devices=	array의 활성화(active) 디바이스 수를 지정합니다. 여기에 여분의 장치 수(아래 옵션 참조)를 더한 값은 --create 명령줄에 나열된 구성 요소 장치 수("누락" 장치 포함)와 같아야 합니다.



-x, --spare-devices=	초기 array의 예비(eXtra) 장치 수를 지정합니다. 나중에 예비품을 추가하고 제거할 수도 있습니다. 명령줄에 나열된 구성 요소 장치 수는 RAID 장치 수에 예비 장치 수를 더한 수와 같아야 합니다.
-z, --size=	RAID 레벨 1/4/5/6의 각 드라이브에서 사용할 공간(킬로바이트)입니다. 이것은 chunk 사이즈의 배수여야 하며, RAID 슈퍼블록(superblock)을 위해 드라이브 끝에 약 128Kb의 공간이 남겨 두어야 합니다. 이 옵션이 지정되지 않은 경우(일반적으로 지정되지 않음) 가장 작은 드라이브(또는 파티션)가 크기를 설정하지만 드라이브 간 차이가 1%보다 큰 경우 경고가 발생합니다. 접미사 'K(킬로바이트)', 'M(메가바이트)', 'G(기가바이트)', 'T(테라바이트)' 를 사용할 수 있습니다.
-Z, --array-size=	이 옵션은 오직 --grow 에서만 의미가 있으며 그 효과는 지속적이지 않습니다.(array가 중지되고 다시 시작되면 기본 array 크기가 복원됩니다.) array 크기를 설정하면 데이터에 액세스하는 프로그램에 array가 더 작게 표시됩니다. 이것은 array를 더 작게 재구성하기 전에 특히 필요합니다. 재구성을 되돌릴 수 없지만 --array-size로 크기를 설정하면 array의 디바이스 수를 줄이기 전에 array의 크기를 적절하게 줄여야 합니다.  array의 크기를 줄이기 전에 공간이 필요하지 않은지 확인해야 합니다. 장치에 파일시스템이 있으면 공간을 덜 사용하려면 파일 시스템의 크기를 조정해야 합니다.
-c, --chunk=	chunk 사이즈를 킬로바이트(kilobytes)로 지정합니다. array를 생성할 때 기본값은 512KB입니다. 이전 버전과의 호환성을 보장하기 위해 영구 메타데이터 없이 array를 구축할 때 기본값은 64KB입니다. 이는 RAID0, RAID4, RAID5, RAID6, RAID10에만 의미가 있습니다.  RAID4, RAID5, RAID6, RAID10은 chunk 사이즈가 2의 거듭제곱이어야 합니다. 어떤 경우에도 4KB의 배수여야 합니다.
--rounding=	Linear array의 반올림 factor(요인/인수)를 지정합니다. 각 구성요소의 크기는 이 크기의 배수로 반올림됩니다. 이것은 --chunk와 동의어이지만 Linear가 다른 RAID 레벨과 비교했을 때 의미하는 바가 다르다는 것을 강조합니다. 2.6.16 이전 커널이 사용 중일 경우 기본값은 64K이며 이후 커널에서는 0K(즉, 반올림 없음)입니다.
-l, --level=	RAID 레벨을 설정합니다. --create 와 함께 사용하는 경우 옵션은 다음과 같습니다. linear, raid0, 0, stripe, raid1, 1, mirror, raid4, 4, raid5, 5, raid6, 6, raid10, 10, multipath, mp, faulty, container 분명히 이들 중 일부는 동의어입니다.  --build와 함께 사용하면 linear, stripe, raid0, 0, raid1, multipath, mp, faulty 만 유효합니다.
-p, --layout=	이 옵션은 RAID5, RAID6, RAID10 array에 대한 데이터 레이아웃의 세부 사항을 구성하고 오류 발생에 대한 오류 모드를 제어합니다.
--backup-file=	사용 가능한 예비 디바이스가 없는 경우 RAID5 또는 RAID6에서 디바이스 개수를 늘리거나 RAID 레벨, RAID 레이아웃을 축소, 변경하기 위해 --grow를 사용할 때 필요합니다. 백업 파일은 재구성되는 RAID array가 아닌 별도의 디바이스에 저장되어야 합니다.
-f, --force	mdadm이 지정된 형태와 레이아웃을 의심 없이 수락하도록 합니다.
-a, --add	이 옵션은 두 가지 경우에 grow 모드에서 사용할 수 있습니다. 1. 대상 array가 Linear array인 경우 --add를 사용하여 array에 하나 이상의 디바이스를 추가할 수 있습니다. 추가된 디바이스들은 array 끝에 연결됩니다. 추가된 디바이스들은 제거할 수 없습니다. 2. --raid-disks 옵션을 사용하여 array의 디바이스 수를 늘리는 경우 --add를 사용하여 array에 포함할 추가 디바이스를 추가할 수 있습니다. 대부분의 경우 추가 디바이스를 예비용(spare)으로 먼저 추가한 다음 raid-disks의 수를 변경할 수 있으므로 이는 필요하지 않습니다. 그러나 RAID0의 경우 예비용 디바이스를 추가할 수 없습니다. 따라서 RAID0의 디바이스 수를 늘리려면 동일한 명령으로 새 디바이스 수를 설정하고 새 디바이스를 추가해야 합니다.
For assemble (일부 옵션은 생략했음...)	
-u, --uuid=	어셈블할 array의 UUID입니다. 이 UUID가 없는 디바이스는 제외됩니다.
--backup-file=	array를 재구성(예: 디바이스 수 또는 chunk 사이즈 변경)하는 동안 --backup-file이 사용되었고 중요한 섹션 중에 시스템이 충돌한 경우 동일한 --backup-file을 --assemble에 제시하여 가능한 대로 손상된 데이터를 복원하고 재구성을 완료합니다.
For Manage mode (일부 옵션은 생략했음...)	
-a, --add	디바이스가 최근에 array의 일부였던 것으로 나타나면(오류가 발생했거나 array에서 제거되었을 수 있음) 다음 항목에 설명된 대로 디바이스가 다시 추가됩니다. 실패(fails)하거나 디바이스가 array의 일부가 아닌 경우 디바이스는 핫-스페어(hot-spare)로 추가됩니다. array의 성능이 저하되면 즉시 해당 스페어에 데이터를 재구축하기 시작합니다.
--re-add	이전에 array에서 제거한 디바이스를 다시 추가합니다. 만약 디바이스의 메타데이터가 해당 디바이스가 array의 구성원임을 보고하고, 사용된 슬롯(slot)이 여전히 비어있는 경우 해당 디바이스는 array의 동일한 위치에 다시 추가됩니다. 이렇게 하면 일반적으로 해당 디바이스의 데이터가 복구됩니다.
--add-spare	디바이스를 예비용 디바이스로 추가합니다. --re-add를 먼저 시도하지 않는다는 점을 제외하면 --add와 유사합니다. 디바이스는 array의 최근 구성원인 것처럼 보이더라도 예비용 디바이스로 추가됩니다.
-r, --remove	디바이스를 제거합니다. 활성(active) 상태가 아니어야 합니다. 즉, 고장난 디바이스 또는 스페어 디바이스여야 합니다.
나머지 옵션은 생략..... mdadm.conf 파일에 슈퍼블록 설정하는 것은 나중에 실습 부팅시 자동 마운트 설정 파일(/etc/fstab)은 설명 생략...	

## mdadm 예시

```
# 디스크 뒤에 숫자가 붙은 것을 보면 알겠지만 RAID 구성하기 전에 디스크를 설치하고 fdisk로 파티셔닝을 먼저 진행해야한다.
# create 이후에 포맷(mkfs)하고 마운트 진행
$ mdadm --create /dev/md9 --level=linear --raid-devices=2 /dev/sdb1 /dev/sdc1
$ mdadm --create /dev/md0 --level=0 --raid-devices=2 /dev/sdd1 /dev/sde1
$ mdadm --create /dev/md0 --level=0 --raid-devices=5 /dev/sd[a-e]1
$ mdadm --create /dev/md1 --level=1 --raid-devices=2 /dev/sdf1 /dev/sdg1
```

```

$ mdadm --create /dev/md5 --level=5 --raid-devices=3 /dev/sdh1 /dev/sdi1 /dev/sdj1
$ mdadm --create /dev/md6 --level=6 --raid-devices=4 /dev/sdb1 /dev/sdc1 /dev/sdd1 /dev/sde1

# RAID 1+0 구성하는 예시
$ mdadm --create /dev/md2 --level=1 --raid-devices=2 /dev/sdf1 /dev/sdg1
$ mdadm --create /dev/md3 --level=1 --raid-devices=2 /dev/sdh1 /dev/sdi1
$ mdadm --create /dev/md10 --level=0 --raid-devices=2 /dev/md2 /dev/md3

# RAID 멈추게 하기 (마운트 해제 먼저 하고 stop하기, 이미 데이터가 손실된 경우에만 stop 하고 다시 create)
$ mdadm --stop /dev/md0
$ mdadm --stop --scan (모든 활성 array를 stop)

# STOP 했던 array를 다시 START
$ mdadm --assemble --scan
$ mdadm --assemble --uuid=76e14372:f26571d8:a7f2a365:80259a02 /dev/md9

# 현재 active 상태의 RAID에 새로운 디스크 추가 (데이터가 살아있는 경우 결함이 있는 디스크는 새 디스크로 교체 후 fdisk로 파티셔닝 후 add)
$ mdadm /dev/md1 --add /dev/sdg1
$ mdadm /dev/md5 --add /dev/sdi1
$ mdadm --manage /dev/md5 --add /dev/sdj1
$ mdadm --manage /dev/md9 --add /dev/sdd1 /dev/sde1

# RAID에 디스크 늘려 확장하기
$ mdadm /dev/md9 --grow --add /dev/sdd1 /dev/sde1
● linear RAID 인 /dev/md9에 sdd1, sde1을 추가하여 4개로 구성하였다. 모든 RAID 레벨에서는 안되고 일단 linear 레벨은 가능하다.
$ mdadm /dev/md1 --add /dev/sdd1 /dev/sde1 (RAID 레벨1 에서 문제없이 동작)
$ mdadm --grow --raid-devices=4 /dev/md1
● RAID 레벨 1에서 원래 sdb1, sdc1 이렇게 2개 었는데 위 예시처럼 --add 로 sdd1, sde1 2개를 추가하여 총 4개로 만들고 그 다음 명령에서 총 4개로 --gr

# 디스크를 FAIL로 표시 (문제가 없는 드라이브를 제거해야 하는 경우 --fail 옵션을 사용하여 수동으로 FAIL로 표시할 수 있다.)
$ mdadm /dev/md1 --fail /dev/sdd1 /dev/sde1
$ mdadm /dev/md1 --remove /dev/sdd1 /dev/sde1
● RAID 레벨 1에서 --fail로 설정 후 --remove로 삭제 (--remove 후 다시 --add 가능)

# 디스크를 물리적 제거가 아닌 RAID에서 제거
$ mdadm --manage /dev/md5 --remove /dev/sdj1

# 디바이스 정보 확인
$ mdadm --detail --scan
$ mdadm --detail /dev/md5
$ mdadm --verbose --detail --scan /dev/md0

```

## Drop RAID

RAID array를 삭제 또는 제거하고 모든 디스크 파티션을 재설정하여 다른 어레이에서 또는 별도로 사용할 수 있도록 하려면 다음을 수행해야 합니다.

1. 먼저 제거하려는 RAID 장치를 확인해야 합니다.  
`cat /proc/mdstat` 로 확인할 수 있습니다.
2. RAID 파티션을 마운트 해제(umount)하고, /etc/fstab을 편집하여 마운트 지점에 대한 줄을 삭제하여 부팅 후에 시스템이 파티션을 마운트하려고 시도하지 않도록 하십시오.
3. `mdadm --stop /dev/md0` 와 같이 RAID 장치를 중지(stop)하고 /etc/mdadm/mdadm.conf에서 장치를 제거합니다.
4. `mdadm --remove /dev/md0` 와 같이 RAID 장치를 제거(remove)합니다.
5. RAID 슈퍼블록(superblocks) 제거  
 RAID의 일부였던 개별 디스크의 슈퍼블록을 0으로 설정해야 합니다. 이렇게 하면 나중에 RAID array로 자동으로 재조립되는 것을 방지할 수 있습니다.  
 기억하세요. RAID 장치(/dev/mdX)가 아닌 디스크 파티션(/dev/sdX)에서 이 작업을 수행합니다.  
`mdadm --zero-superblock /dev/sdX1 /dev/sdX2`
6. (선택사항) 랜덤 데이터로 디스크 덮어쓰기  
 디스크에서 데이터를 제거하려면 dd를 사용하여 다시 쓸 수 있습니다.  
`dd if=/dev/urandom bs=4096 of=/dev/sdX`