RAID

رحمت اله انصاری

محمدداود وهاب رجایی

درس سیستم عامل

دکتر ملک زاده

دانشگاه حکیم سبزواری

رشته مهندسی کامپیوتر

دی ۱۴۰۱

بسم الله الرحمن الرحیم

مشخصات

عنوان پروژه : raid level 6

نویسندگان:

نام و نام خانوادگی : رحمت اله انصاری

شماره دانشجویی : 9912377331

ایمیل : Rahmat2022a@gmail.com

گیت‌هاب: github.com/EnAnsari

نام و نام خانوادگی : محمدداود وهاب رجایی

شماره دانشجویی : 9912377889

درس سیستم عامل

استاد : دکتر ملک زاده

دانشگاه حکیم سبزواری، سبزوار

رشته مهندسی کامپیوتر

دانشکده مهندسی کامپیوتر و برق

نوشته شده در دی سال ۱۴۰۱

فهرست مطالب

مقدمه

سلام، وقت بخیر . . .

در این مقاله می‌پردازیم به اینکه عبارت raid به چه معناست؟ چه پیشنیازهایی دارد و همینطور خروجی اصلی ما از انجام این عملیات چطور است. شما می‌توانید در فهرست مطالب ببینید هر قسمت شامل چه موضوعی است.

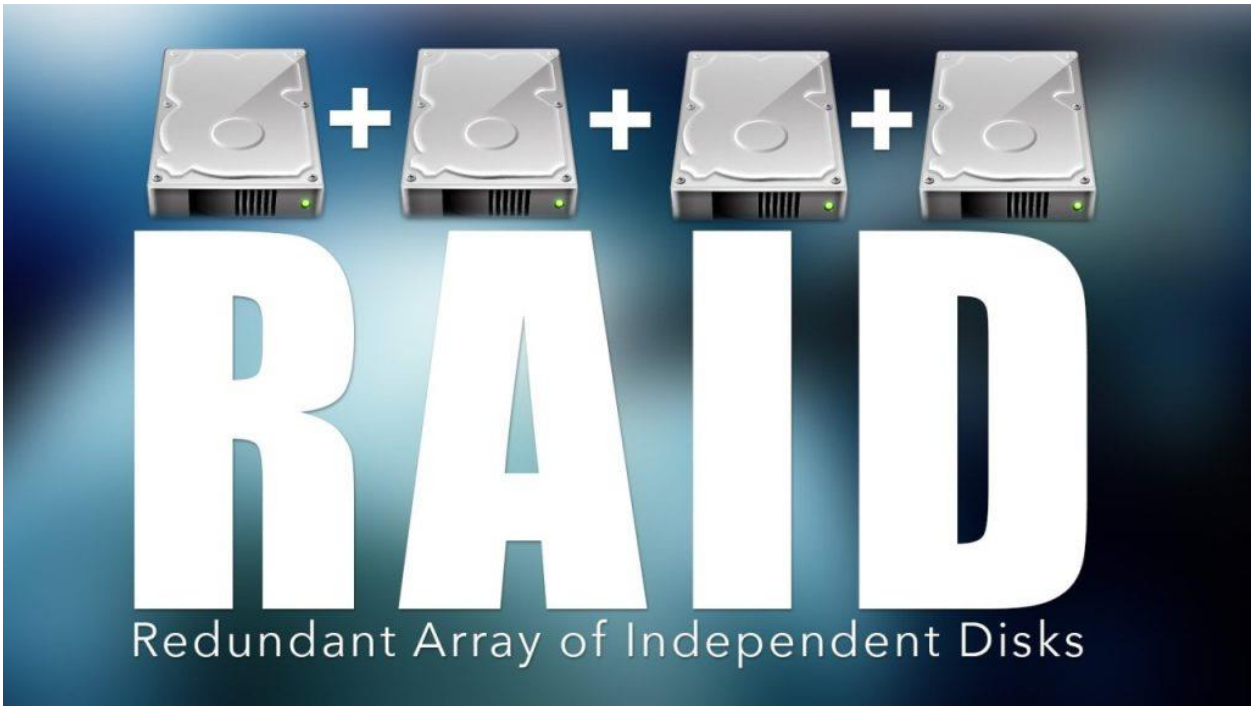
در صورت وجود هر گونه ابهامی می‌توانید با ایمیلی که در قسمت مشخصات وجود دارد با من در ارتباط باشید.

اگر می‌خواهید مستقیما مراحل انجام پروسه raid سطح ۶ توسط خودم را ببینید روی این لینک + کلیک کنید.

با آرزوی موفقیت . . .

انصاری

تعریفات اولیه در مورد RAID



RAID چیست؟

RAID مخفف Redundant Array of Independent Disks مکانیزمی است که در آن مجموعه‌ای از دیسک‌ها به‌صورت زنجیره‌ای به هم متصل می‌شوند و موجب افزایش کارایی و دسترسی‌پذیری به داده‌ها و همچنین تحمل خطاپذیری و بازیابی داده‌های ذخیره شده می‌شود.

انواع RAID

1- Hardware RAID: که به‌صورت سخت‌افزاری و مستقل از سیستم‌عامل عمل می‌کند و تمامی عملیات توسط RAID Controller سیستم انجام می‌شود.

2- Software RAID: که توسط سیستم‌عامل و نرم‌افزار پیکربندی و کنترل می‌شود.

ابزار

ابزار mdadm را می توان برای ایجاد و مدیریت آرایه های ذخیره سازی با استفاده از قابلیت های RAID در نرم افزار لینوکس استفاده کرد. مدیران از انعطاف پذیری بالایی در هماهنگ کردن دستگاه های ذخیره سازی فردی خود و ایجاد دستگاه های ذخیره سازی منطقی که دارای عملکرد یا ویژگی های افزونگی بیشتری هستند برخوردارند.

پیشنیازها

بسته به نوع آرایه، شما به دو تا چهار دستگاه ذخیره سازی نیاز دارید. این درایوها قبل از دنبال کردن این مقاله نیازی به فرمت ندارند.

همچنین می‌بایست یک نسخه لینوکس را داشته باشید. در این مقاله آموزش نصب لینوکس وجود دارد.

سطوح RAID

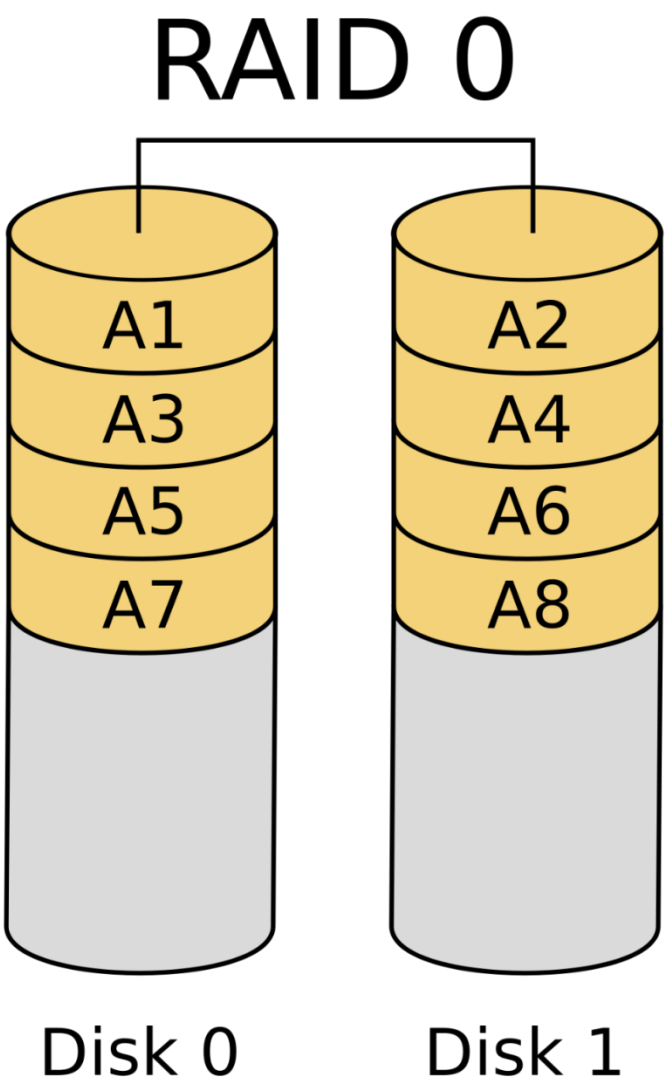
RAID می‌تواند به‌صورت‌های مختلفی پیاده‌سازی شود که در ادامه به معرفی کاربردی‌ترین آن‌ها می‌پردازیم.

RAID 0 یا Striping:

در این روش داده‌ها به‌صورت تکه‌تکه شده و هر تکه در یک دیسک ذخیره می‌شود که موجب افزایش سرعت خواندن/نوشتن می گردد.

در این روش امکان تحمل خطاپذیری و بازیابی داده‌ها وجود ندارد و در صورت خرابی یکی از دیسک ها داده‌ها از بین خواهد رفت.

آرایه RAID 0 با تقسیم کردن داده ها به قطعات و جدا کردن آن در دیسک های موجود کار می کند. این بدان معنی است که هر دیسک حاوی بخشی از داده ها است و هنگام بازیابی اطلاعات به چندین دیسک ارجاع داده می شود.



ایجاد یک آرایه RAID 0

شرایط مورد نیاز: حداقل 2 دستگاه ذخیره سازی.

مزیت اصلی: عملکرد از نظر خواندن/نوشتن و ظرفیت

مواردی که باید در نظر داشته باشید: مطمئن شوید که پشتیبان‌گیری کاربردی دارید. خرابی یک دستگاه تمام داده های آرایه را از بین می برد.

برای شروع، شناسه‌های دیسک‌های خامی را که استفاده می‌کنید پیدا کنید:

```
$ lsblk -o NAME,SIZE,FSTYPE,TYPE,MOUNTPOINT
```

```
NAME      SIZE FSTYPE    TYPE MOUNTPOINT
sda       100G           disk
sdb       100G           disk
vda        25G           disk
├─vda1   24.9G ext4      part /
├─vda14    4M            part
└─vda15  106M vfat      part /boot/efi
vdb       466K iso9660   disk
```

در این مثال، شما دو دیسک بدون سیستم فایل دارید که هر کدام دارای 100 گیگ ظرفیت هستند. به این دستگاه ها شناسه های /dev/sda و /dev/sdb برای این جلسه داده شده است و اجزای خام مورد استفاده برای ساخت آرایه خواهند بود.

برای ایجاد یک آرایه RAID 0 با این اجزا، آنها را به دستور mdadm --create منتقل کنید. شما باید نام دستگاهی را که می خواهید ایجاد کنید، سطح RAID و تعداد دستگاه ها را مشخص کنید. در این دستور، نام دستگاه را /dev/md0 است و دو دیسکی که آرایه را می‌سازند در کد زیر داریم:

```
$ sudo mdadm --create --verbose /dev/md0 --level=0 --raid-devices=2 /dev/sda /dev/sdb
```

با بررسی فایل /proc/mdstat تأیید کنید که RAID با موفقیت ایجاد شده است:

```
$ cat /proc/mdstat
```

خروجی به صورت زیر است:

```
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5]  [raid4] [raid10]
md0 : active raid0 sdb[1] sda[0]
      209584128 blocks super 1.2 512k chunks

           unused devices: <none>
```

این خروجی نشان می‌دهد که دستگاه /dev/md0 در پیکربندی RAID 0 با استفاده از دستگاه‌های /dev/sda و /dev/sdb ایجاد شده است.

ایجاد و نصب فایل سیستم

بعد، یک فایل سیستم روی آرایه ایجاد کنید:

```
$ sudo mkfs.ext4 -F /dev/md0
```

سپس، یک نقطه اتصال برای پیوست کردن فایل سیستم جدید ایجاد کنید:

```
$ sudo mkdir -p /mnt/md0
```

با دستور زیر می توانید فایل سیستم را Mount کنید:

```
$ sudo mount /dev/md0 /mnt/md0
```

پس از آن، بررسی کنید که آیا فضای جدید موجود است:

```
$ df -h -x devtmpfs -x tmpfs
```

خروجی به صورت زیر خواهد بود:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        25G  1.4G   23G   6% /
/dev/vda15      105M  3.4M  102M   4% /boot/efi
/dev/md0        196G   61M  186G   1% /mnt/md0
```

فایل سیستم جدید اکنون نصب شده و قابل دسترسی است.

ذخیره چیدمان آرایه

برای اطمینان از اینکه آرایه به طور خودکار در هنگام بوت جمع می شود، باید فایل `/etc/mdadm/mdadm.conf`را تنظیم کنید. شما می توانید به طور خودکار آرایه فعال را اسکن کرده و فایل را با موارد زیر اضافه کنید:

```
$ sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf
```

پس از آن، می‌توانید initramfs یا سیستم فایل RAM اولیه را به‌روزرسانی کنید تا آرایه در طول فرآیند بوت اولیه در دسترس باشد:

```
$ sudo update-initramfs -u
```

برای نصب خودکار در هنگام بوت، گزینه های نصب فایل سیستم جدید را به فایل `/etc/fstab` اضافه کنید:

```
$ echo '/dev/md0 /mnt/md0 ext4 defaults,nofail,discard 0 0' | sudo tee -a /etc/fstab
```

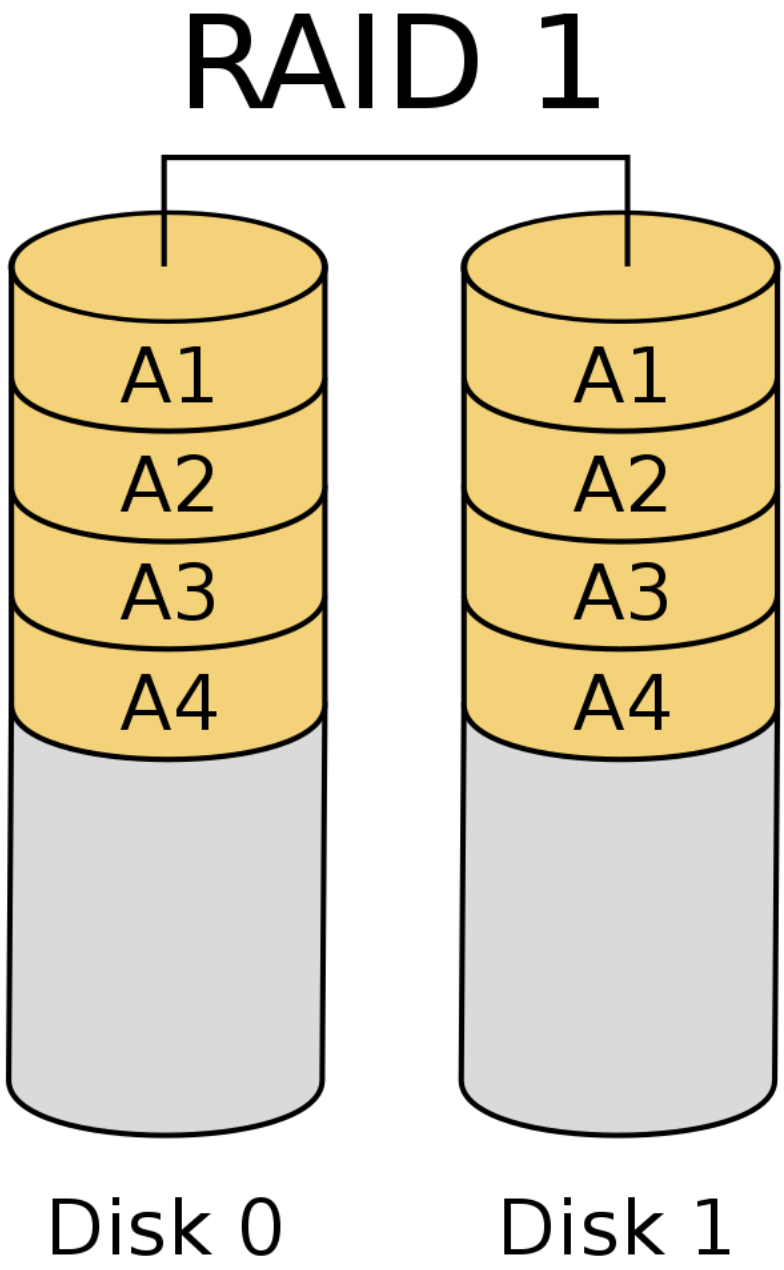
آرایه RAID 0 شما اکنون به طور خودکار هر بوت را جمع و سوار می کند.

اکنون راه اندازی RAID خود را به پایان رسانده اید. اگر می‌خواهید یک RAID دیگر را امتحان کنید، دستورالعمل‌های بازنشانی در انتهای سطوح قرار دارد را دنبال کنید تا با ایجاد یک نوع آرایه RAID جدید ادامه دهید.

RAID 1 یا Mirroring:

در این روش داده‌های نوشته شده روی یک دیسک عیناً روی دیسک دیگر کپی می‌شود. در این روش سرعت خواندن/نوشتن افزایش نمی‌یابد ولی امکان تحمل خطاپذیری و بازیابی داده‌ها در زمانی خرابی یکی از دیسک‌ها وجود دارد.

نوع آرایه RAID 1 با انعکاس داده ها در تمام دیسک های موجود پیاده سازی می شود. هر دیسک در یک آرایه RAID 1 یک کپی کامل از داده ها را دریافت می کند و در صورت خرابی دستگاه، افزونگی را ارائه می دهد.



شرایط مورد نیاز: حداقل 2 دستگاه ذخیره سازی.

مزیت اصلی: افزونگی بین دو دستگاه ذخیره سازی.

مواردی که باید در نظر داشته باشید: از آنجایی که دو نسخه از داده ها نگهداری می شود، تنها نیمی از فضای دیسک قابل استفاده خواهد بود.

برای شروع، شناسه‌های دیسک‌های خامی را که استفاده می‌کنید پیدا کنید:

```
$ lsblk -o NAME,SIZE,FSTYPE,TYPE,MOUNTPOINT
```

```
NAME      SIZE FSTYPE   TYPE MOUNTPOINT
sda       100G          disk
sdb       100G          disk
vda        25G          disk
├─vda1   24.9G ext4     part /
├─vda14    4M           part
└─vda15  106M vfat     part /boot/efi
vdb       466K iso9660  disk
```

در این مثال، شما دو دیسک بدون سیستم فایل دارید که هر کدام دارای 100 گیگ ظرفیت هستند. به این دستگاه ها شناسه های `/dev/sda` و `/dev/sdb` برای این جلسه داده شده است و اجزای خام مورد استفاده برای ساخت آرایه خواهند بود.

برای ایجاد یک آرایه RAID 1 با این اجزا، آنها را به دستور `mdadm --create` منتقل کنید. شما باید نام دستگاهی را که می خواهید ایجاد کنید، سطح RAID و تعداد دستگاه ها را مشخص کنید. در این دستور، نام دستگاه را `/dev/md0` است و دو دیسکی که آرایه را می‌سازند در کد زیر داریم:

```
$ sudo mdadm --create --verbose /dev/md0 --level=1 --raid-devices=2 /dev/sda /dev/sdb
```

اگر دستگاه‌های مؤلفه‌ای که استفاده می‌کنید پارتیشن‌هایی با `boot flag` فعال نیستند، احتمالاً هشدار زیر را دریافت خواهید کرد. بی خطر است که با y پاسخ دهید و ادامه دهید:

```
mdadm: Note: this array has metadata at the start and
    may not be suitable as a boot device.  If you plan to
    store '/boot' on this device please ensure that
    your boot-loader understands md/v1.x metadata, or use
    --metadata=0.90
mdadm: size set to 104792064K
Continue creating array? y
```

ابزار mdadm شروع به انعکاس درایوها می کند. این ممکن است کمی طول بکشد، اما آرایه را می توان در این مدت استفاده کرد. می‌توانید با بررسی فایل `/proc/mdstat`، پیشرفت آینه‌سازی را کنترل کنید:

```
$ cat /proc/mdstat
```

```
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdb[1] sda[0]
      104792064 blocks super 1.2 [2/2] [UU]
      [====>.............]  resync = 20.2% (21233216/104792064) finish=6.9min
speed=199507K/sec

unused devices: <none>
```

در اولین هایلایت، دستگاه `/dev/md0`در پیکربندی RAID 1 با استفاده از دستگاه های `/dev/sda` و `/dev/sdb` ایجاد شد. دومین هایلایت پیشرفت در آینه کاری را نشان می دهد. تا زمانی که این فرآیند کامل شد، می توانید به مرحله بعدی ادامه دهید.

ایجاد و نصب فایل سیستم

بعد، یک فایل سیستم روی آرایه ایجاد کنید:

```
$ sudo mkfs.ext4 -F /dev/md0
```

سپس، یک نقطه اتصال برای پیوست کردن فایل سیستم جدید ایجاد کنید:

```
$ sudo mkdir -p /mnt/md0
```

با دستور زیر می توانید فایل سیستم را Mount کنید:

```
$ sudo mount /dev/md0 /mnt/md0
```

پس از آن، بررسی کنید که آیا فضای جدید موجود است:

```
$ df -h -x devtmpfs -x tmpfs
```

خروجی به صورت زیر خواهد بود:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        25G  1.4G   23G   6% /
/dev/vda15      105M  3.4M  102M   4% /boot/efi
/dev/md0        196G   61M  186G   1% /mnt/md0
```

فایل سیستم جدید اکنون نصب شده و قابل دسترسی است.

ذخیره چیدمان آرایه

برای اطمینان از اینکه آرایه به طور خودکار در هنگام بوت جمع می شود، باید فایل `/etc/mdadm/mdadm.conf`را تنظیم کنید. شما می توانید به طور خودکار آرایه فعال را اسکن کرده و فایل را با موارد زیر اضافه کنید:

```
$ sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf
```

پس از آن، می‌توانید `initramfs` یا سیستم فایل RAM اولیه را به‌روزرسانی کنید تا آرایه در طول فرآیند بوت اولیه در دسترس باشد:

```
$ sudo update-initramfs -u
```

برای نصب خودکار در هنگام بوت، گزینه های نصب فایل سیستم جدید را به فایل `/etc/fstab` اضافه کنید:

```
$ echo '/dev/md0 /mnt/md0 ext4 defaults,nofail,discard 0 0' | sudo tee -a /etc/fstab
```

آرایه `RAID 1` شما اکنون به طور خودکار هر بوت را مونتاژ و سوار می کند.

اکنون راه اندازی RAID خود را به پایان رسانده اید. اگر می‌خواهید یک RAID دیگر را امتحان کنید، دستورالعمل‌های بازنشانی در انتهای سطوح قرار دارد را دنبال کنید تا با ایجاد یک نوع آرایه RAID جدید ادامه دهید.

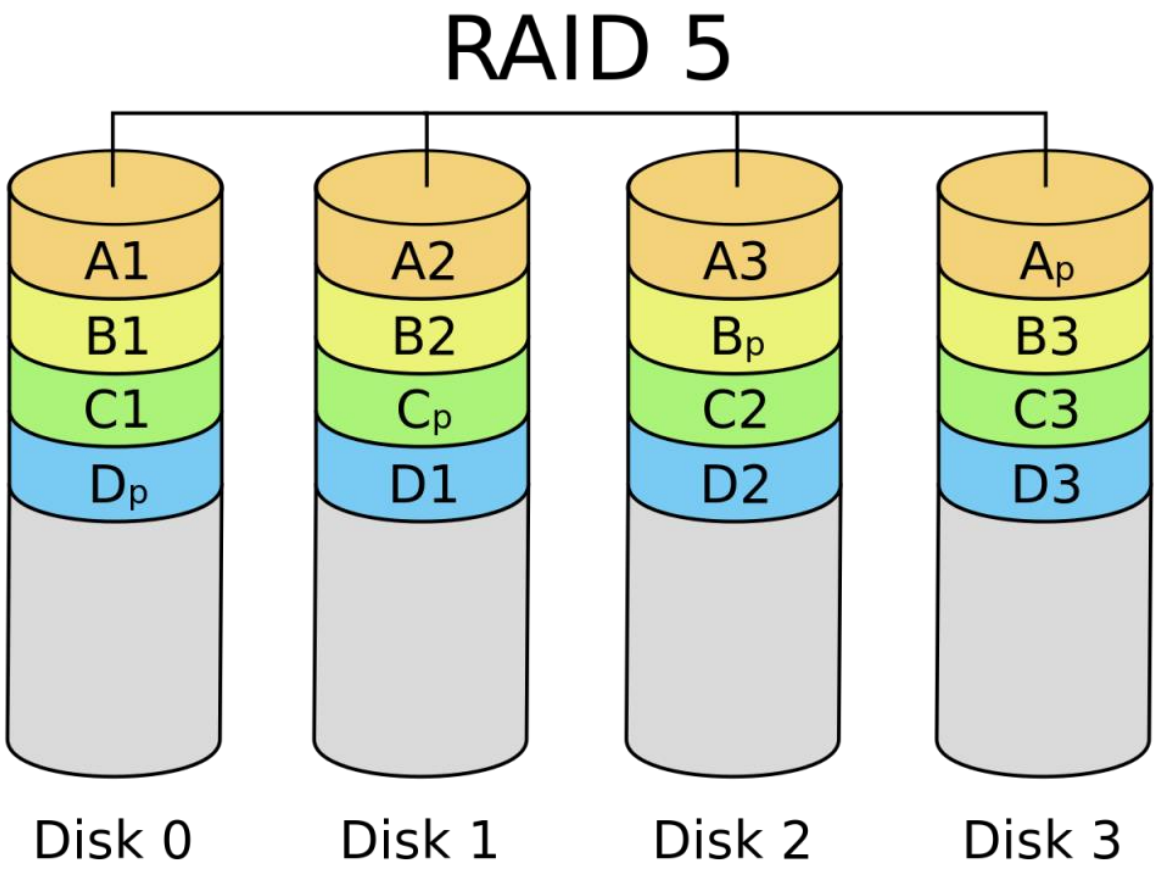
`RAID 5` یا `Striped with Parity`:

در این روش داده‌ها همانند روش `Striping` به‌صورت تکه‌تکه شده بین دیسک‌ها تقسیم و ذخیره می‌شود با این تفاوت که علاوه بر داده‌های ذخیره شده نوعی داده تحت عنوان `Parity` نیز در دیسک‌ها ذخیره می‌گردد.

در این روش سرعت خواندن/نوشتن افزایش می‌یابد و همچنین بدلیل استفاده از `Parity` امکان تحمل خطاپذیری و بازیابی داده‌ها در زمان خرابی یکی از دیسک‌ها وجود دارد.

نوع آرایه `RAID 5` با نوار کردن داده ها در دستگاه های موجود پیاده سازی می شود. یک جزء از هر نوار یک بلوک برابری محاسبه شده است. اگر دستگاهی از کار بیفتد، بلوک برابری و بلوک های باقی مانده می توانند

برای محاسبه داده های از دست رفته استفاده شوند. دستگاهی که بلوک برابری را دریافت می کند به گونه ای می چرخد که هر دستگاه دارای مقدار متعادلی از اطلاعات برابری باشد.



شرایط مورد نیاز: حداقل 3 دستگاه ذخیره سازی.

مزیت اصلی: افزونگی با ظرفیت قابل استفاده بیشتر.

مواردی که باید در نظر داشته باشید: در حالی که اطلاعات برابری (Parity) توزیع می شود، از ظرفیت یک دیسک برای برابری استفاده می شود. RAID 5 می تواند از عملکرد بسیار ضعیف در حالت تخریب رنج ببرد.

برای شروع، شناسه‌های دیسک‌های خامی را که استفاده می‌کنید پیدا کنید:

```
$ lsblk -o NAME,SIZE,FSTYPE,TYPE,MOUNTPOINT
```

```
NAME      SIZE FSTYPE    TYPE MOUNTPOINT
sda       100G           disk
sdb       100G           disk
sdc       100G           disk
vda        25G           disk
├─vda1   24.9G ext4      part /
├─vda14     4M           part
└─vda15   106M vfat      part /boot/efi
vdb       466K iso9660   disk
```

در این مثال، شما سه دیسک بدون سیستم فایل دارید که هر کدام دارای 100 گیگ ظرفیت هستند. به این دستگاه ها شناسه های /dev/sda و /dev/sdb و /dev/sdc برای این جلسه داده شده است و اجزای خام مورد استفاده برای ساخت آرایه خواهند بود.

برای ایجاد یک آرایه RAID 5 با این اجزا، آنها را به دستور mdadm --create منتقل کنید. شما باید نام دستگاهی را که می خواهید ایجاد کنید، سطح RAID و تعداد دستگاه ها را مشخص کنید. در این دستور، نام دستگاه را /dev/md0 است و دو دیسکی که آرایه را می‌سازند در کد زیر داریم:

```
$ sudo mdadm --create --verbose /dev/md0 --level=5 --raid-devices=3 /dev/sda /dev/sdb /dev/sdc
```

ابزار mdadm شروع به پیکربندی آرایه می کند. از فرآیند بازیابی برای ساخت آرایه به دلایل عملکرد استفاده می کند. این ممکن است کمی طول بکشد، اما آرایه را می توان در این مدت استفاده کرد. می‌توانید با بررسی فایل /proc/mdstat، پیشرفت آینه‌سازی را کنترل کنید:

```
$ cat /proc/mdstat
```

خروجی این دستور به صورت زیر است:

```
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid5 sdc[3] sdb[1] sda[0]
      209582080 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/2] [UU_]
      [>....................]  recovery =  0.9% (957244/104791040) finish=18.0min speed=95724K/sec

unused devices: <none>
```

در اولین هایلایت، دستگاه /dev/md0 در پیکربندی RAID 5 با استفاده از دستگاه های /dev/sda، /dev/sdb و /dev/sdc ایجاد شد. دومین هایلایت پیشرفت ساخت را نشان می دهد.

هشدار: با توجه به روشی که mdadm آرایه‌های RAID 5 را می‌سازد، در حالی که آرایه هنوز در حال ساخت است، تعداد قطعات یدکی در آرایه به‌صورت نادرست گزارش می‌شود. این بدان معناست که قبل از به‌روزرسانی فایل /etc/mdadm/mdadm.conf باید منتظر بمانید تا آرایه مونتاژ شود. اگر فایل پیکربندی را در حالی که آرایه هنوز در حال ساخت است به روز کنید، سیستم اطلاعات نادرستی در مورد وضعیت آرایه خواهد داشت و نمی تواند آن را به طور خودکار در هنگام بوت با نام صحیح جمع کند.

می‌توانید تا زمانی که این فرآیند کامل شد، راهنما را ادامه دهید.

ایجاد و نصب فایل سیستم

بعد، یک فایل سیستم روی آرایه ایجاد کنید:

```
$ sudo mkfs.ext4 -F /dev/md0
```

سپس، یک نقطه اتصال برای پیوست کردن فایل سیستم جدید ایجاد کنید:

```
$ sudo mkdir -p /mnt/md0
```

با دستور زیر می توانید فایل سیستم را Mount کنید:

```
$ sudo mount /dev/md0 /mnt/md0
```

پس از آن، بررسی کنید که آیا فضای جدید موجود است:

```
$ df -h -x devtmpfs -x tmpfs
```

خروجی به صورت زیر خواهد بود:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        25G  1.4G   23G   6% /
/dev/vda15      105M  3.4M  102M   4% /boot/efi
/dev/md0        196G   61M  186G   1% /mnt/md0
```

فایل سیستم جدید اکنون نصب شده و قابل دسترسی است.

ذخیره چیدمان آرایه

برای اطمینان از اینکه آرایه به طور خودکار در هنگام بوت جمع می شود، باید فایل `/etc/mdadm/mdadm.conf` را تنظیم کنید.

هشدار: همانطور که قبلا ذکر شد، قبل از اینکه پیکربندی را تنظیم کنید، دوباره بررسی کنید تا مطمئن شوید که آرایه مونتاژ شده است. انجام مراحل زیر قبل از ساخته شدن آرایه از مونتاژ صحیح آرایه سیستم در راه اندازی مجدد جلوگیری می کند.

می‌توانید با بررسی فایل `/proc/mdstat`، پیشرفت آینه‌سازی را کنترل کنید:

```
$ cat /proc/mdstat
```

خروجی به صورت زیر خواهد بود:

```
Personalities : [raid1] [linear] [multipath] [raid0] [raid6] [raid5] [raid4] [raid10]
md0 : active raid5 sdc[3] sdb[1] sda[0]
      209584128 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3] [UUU]

unused devices: <none>
```

این خروجی نشان می دهد که بازسازی کامل شده است. اکنون، می توانید به طور خودکار آرایه فعال را اسکن کرده و فایل را اضافه کنید:

```
$ sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf
```

پس از آن، می‌توانید `initramfs` یا سیستم فایل RAM اولیه را به‌روزرسانی کنید تا آرایه در طول فرآیند بوت اولیه در دسترس باشد:

```
$ sudo update-initramfs -u
```

برای نصب خودکار در هنگام بوت، گزینه های نصب فایل سیستم جدید را به فایل `/etc/fstab` اضافه کنید:

```
$ echo '/dev/md0 /mnt/md0 ext4 defaults,nofail,discard 0 0' | sudo tee -a /etc/fstab
```

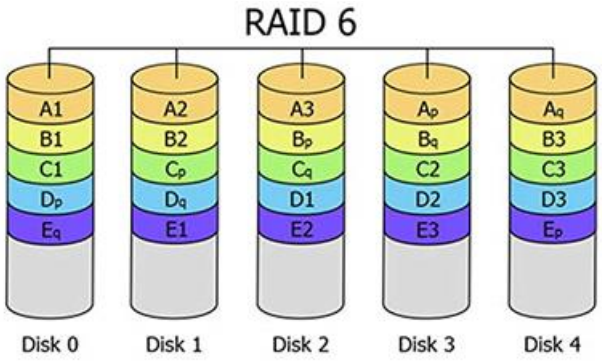
آرایه RAID 5 شما اکنون به طور خودکار هر بوت را مونتاژ و سوار می کند.

اکنون راه اندازی RAID خود را به پایان رسانده اید. اگر می‌خواهید یک RAID دیگر را امتحان کنید، دستورالعمل‌های بازنشانی در انتهای سطوح قرار دارد را دنبال کنید تا با ایجاد یک نوع آرایه RAID جدید ادامه دهید.

RAID 6 یا Striped with Double Parity:

این روش درواقع نسخه توسعه یافته RAID 5 می‌باشد با این تفاوت که در این روش از دو بلوک مجزا Parity استفاده می‌شود که موجب می‌شود امکان تحمل خطاپذیری و بازیابی داده‌ها افزایش یابد.

نوع آرایه RAID 6 با نوار کردن داده ها در دستگاه های موجود پیاده سازی می شود. دو جزء از هر نوار برای بلوک های برابری یا parity محاسبه شده است. اگر یک یا دو دستگاه از کار بیفتند، بلوک های parity و بلوک های باقی مانده می توانند برای محاسبه داده های از دست رفته استفاده شوند. دستگاه هایی که بلوک های برابری را دریافت می کنند به گونه ای می چرخند که هر دستگاه دارای مقدار متعادلی از اطلاعات برابری باشد. این شبیه به آرایه RAID 5 است، اما امکان خرابی دو درایو را فراهم می کند.



شرایط مورد نیاز: حداقل 4 دستگاه ذخیره سازی.

مزیت اصلی: افزونگی مضاعف با ظرفیت قابل استفاده بیشتر.

مواردی که باید در نظر داشته باشید: در حالی که اطلاعات parity توزیع می شود، ظرفیت دو دیسک برای آن استفاده می شود. RAID 6 می تواند از عملکرد بسیار ضعیف در حالت تخریب رنج ببرد.

برای شروع، شناسه‌های دیسک‌های خامی را که استفاده می‌کنید پیدا کنید:

```
$ lsblk -o NAME,SIZE,FSTYPE,TYPE,MOUNTPOINT
```

```
NAME      SIZE FSTYPE   TYPE MOUNTPOINT
sda       100G          disk
sdb       100G          disk
sdc       100G          disk
sdd       100G          disk
vda        25G          disk
├─vda1   24.9G ext4     part /
├─vda14    4M           part
└─vda15  106M vfat     part /boot/efi
vdb       466K iso9660  disk
```

در این مثال، شما چهار دیسک بدون سیستم فایل دارید که هر کدام دارای 100 گیگ ظرفیت هستند. به این دستگاه ها شناسه های `/dev/sda` و `/dev/sdb` و `/dev/sdc` و `/dev/sdd` برای این جلسه داده شده است و اجزای خام مورد استفاده برای ساخت آرایه خواهند بود.

برای ایجاد یک آرایه RAID 6 با این اجزا، آنها را به دستور `mdadm --create` منتقل کنید. شما باید نام دستگاهی را که می خواهید ایجاد کنید، سطح RAID و تعداد دستگاه ها را مشخص کنید. در این دستور، نام دستگاه را `/dev/md0` است و دو دیسکی که آرایه را می‌سازند در کد زیر داریم:

```
$ sudo mdadm --create --verbose /dev/md0 --level=6 --raid-devices=4 /dev/sda /dev/sdb /dev/sdc /dev/sdd
```

ابزار mdadm شروع به پیکربندی آرایه می کند. از فرآیند بازیابی برای ساخت آرایه به دلایل عملکرد استفاده می کند. این ممکن است کمی طول بکشد، اما آرایه را می توان در این مدت استفاده کرد. می‌توانید با بررسی فایل `/proc/mdstat`، پیشرفت آینه‌سازی را کنترل کنید:

```
$ cat /proc/mdstat
```

خروجی دستور بالا به صورت زیر است:

```
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid6 sdd[3] sdc[2] sdb[1] sda[0]
      209584128 blocks super 1.2 level 6, 512k chunk, algorithm 2 [4/4] [UUUU]
      [>....................]  resync =  0.6% (668572/104792064) finish=10.3min speed=167143K/sec

unused devices: <none>
```

در اولین هایلایت، دستگاه `/dev/md0` در پیکربندی RAID 6 با استفاده از `/dev/sda`، `/dev/sdb`، `/dev/sdc` و `/dev/sdd` ایجاد شده است. دومین خط برجسته پیشرفت ساخت را نشان می دهد. می‌توانید تا زمانی که این فرآیند کامل شد، راهنما را ادامه دهید.

ایجاد و نصب فایل سیستم

بعد، یک فایل سیستم روی آرایه ایجاد کنید:

```
$ sudo mkfs.ext4 -F /dev/md0
```

سپس، یک نقطه اتصال برای پیوست کردن فایل سیستم جدید ایجاد کنید:

```
$ sudo mkdir -p /mnt/md0
```

با دستور زیر می توانید فایل سیستم را Mount کنید:

```
$ sudo mount /dev/md0 /mnt/md0
```

پس از آن، بررسی کنید که آیا فضای جدید موجود است:

```
$ df -h -x devtmpfs -x tmpfs
```

خروجی به صورت زیر خواهد بود:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        25G  1.4G   23G   6% /
/dev/vda15      105M  3.4M  102M   4% /boot/efi
/dev/md0        197G   60M  187G   1% /mnt/md0
```

فایل سیستم جدید اکنون نصب شده و قابل دسترسی است.

ذخیره چیدمان آرایه

برای اطمینان از اینکه آرایه به طور خودکار در هنگام بوت جمع می شود، باید فایل `/etc/mdadm/mdadm.conf`را تنظیم کنید. شما می توانید به طور خودکار آرایه فعال را اسکن کرده و فایل را با موارد زیر اضافه کنید:

```
$ sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf
```

پس از آن، می‌توانید `initramfs` یا سیستم فایل RAM اولیه را به‌روزرسانی کنید تا آرایه در طول فرآیند بوت اولیه در دسترس باشد:

```
$ sudo update-initramfs -u
```

برای نصب خودکار در هنگام بوت، گزینه های نصب فایل سیستم جدید را به فایل `/etc/fstab` اضافه کنید:

```
$ echo '/dev/md0 /mnt/md0 ext4 defaults,nofail,discard 0 0' | sudo tee -a /etc/fstab
```

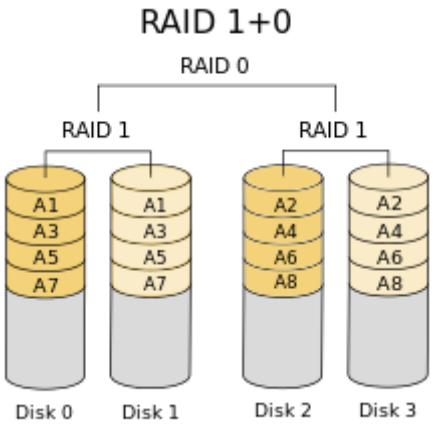
آرایه RAID 6 شما اکنون به طور خودکار هر بوت را مونتاژ و سوار می کند.

اکنون راه اندازی RAID خود را به پایان رسانده اید. اگر می‌خواهید یک RAID دیگر را امتحان کنید، دستورالعمل‌های بازنشانی در انتهای سطوح قرار دارد را دنبال کنید تا با ایجاد یک نوع آرایه RAID جدید ادامه دهید.

RAID 10 یا RAID 1+0:

RAID 10 که با نام RAID 1+0 نیز شناخته می شود، یک پیکربندی RAID است که برای محافظت از داده ها، انعکاس دیسک و نوار دیسک را ترکیب می کند. حداقل به چهار دیسک و داده نواری در جفت های آینه شده نیاز دارد. تا زمانی که یک دیسک در هر جفت آینه کاری شده کارایی داشته باشد، داده ها قابل بازیابی هستند.

نوع آرایه RAID 10 به طور سنتی با ایجاد یک آرایه راه راه RAID 0 متشکل از مجموعه ای از آرایه های RAID 1 پیاده سازی می شود. این نوع آرایه تو در تو، هم افزونگی و هم کارایی بالا را به هزینه مقدار زیادی فضای دیسک می دهد. ابزار mdadm نوع RAID 10 خود را دارد که با افزایش انعطاف‌پذیری، مزایای مشابهی را ارائه می‌کند. توسط آرایه های تودرتو ایجاد نمی شود، اما بسیاری از ویژگی ها و تضمین های مشابه را دارد. در اینجا از mdadm RAID 10 استفاده خواهید کرد.



شرایط مورد نیاز: حداقل 3 دستگاه ذخیره سازی.

مزیت اصلی: عملکرد و افزونگی.

مواردی که باید در نظر داشته باشید: میزان کاهش ظرفیت برای آرایه با تعداد کپی های داده ای که انتخاب می کنید برای نگهداری تعریف می شود. تعداد کپی هایی که با RAID 10 به سبک mdadm ذخیره می شوند قابل تنظیم است.

به‌طور پیش‌فرض، دو نسخه از هر بلوک داده در چیزی که طرح نزدیک نامیده می‌شود ذخیره می‌شود. طرح‌بندی‌های ممکن که نحوه ذخیره هر بلوک داده را دیکته می‌کند به شرح زیر است:

- نزدیک: ترتیب پیش فرض. کپی‌های هر تکه به‌طور متوالی هنگام خط‌بندی نوشته می‌شوند، به این معنی که کپی‌های بلوک‌های داده در اطراف همان قسمت از چندین دیسک نوشته می‌شوند.
- دور: اولین و نسخه های بعدی در قسمت های مختلف دستگاه های ذخیره سازی آرایه نوشته می شوند. به عنوان مثال، تکه اول ممکن است در نزدیکی ابتدای یک دیسک نوشته شود، در حالی که تکه دوم در نیمه راه روی دیسک دیگری نوشته شود. این می‌تواند باعث افزایش عملکرد خواندن برای دیسک‌های چرخان سنتی به قیمت عملکرد نوشتن شود.
- انحراف یا offset: هر نوار کپی شده و توسط یک درایو آفست (منحرف) می شود. این بدان معنی است که کپی ها از یکدیگر آفست هستند، اما همچنان روی دیسک به هم نزدیک هستند. این کمک می کند تا جستجوی بیش از حد در طول برخی از بارهای کاری به حداقل برسد.

با بررسی بخش RAID10 در صفحه man می‌توانید درباره این طرح‌بندی‌ها اطلاعات بیشتری کسب کنید:

```
$ man 4 md
```

شما همچنین می‌توانید از این صفحه + اطلاعات man را پیدا کنید.

برای شروع، شناسه‌های دیسک‌های خامی را که استفاده می‌کنید پیدا کنید:

```
$ lsblk -o NAME,SIZE,FSTYPE,TYPE,MOUNTPOINT
```

```
NAME      SIZE FSTYPE   TYPE MOUNTPOINT
sda       100G          disk
sdb       100G          disk
sdc       100G          disk
sdd       100G          disk
vda        25G          disk
├─vda1   24.9G ext4     part /
├─vda14    4M           part
└─vda15  106M vfat     part /boot/efi
vdb       466K iso9660  disk
```

در این مثال، شما چهار دیسک بدون سیستم فایل دارید که هر کدام دارای 100 گیگابایت ظرفیت هستند. به این دستگاه ها شناسه های /dev/sda ، /dev/sdb ، /dev/sdc و /dev/sdd برای این جلسه داده شده است و اجزای خام مورد استفاده برای ساخت آرایه خواهند بود.

برای ایجاد یک آرایه RAID 10 با این اجزا، آنها را به دستور mdadm --create منتقل کنید. شما باید نام دستگاهی را که می خواهید ایجاد کنید، سطح RAID و تعداد دستگاه ها را مشخص کنید. در این دستور، نام دستگاه را /dev/md0 می‌گذارید و دیسک‌هایی را که آرایه را می‌سازند شامل می‌شوید:

می‌توانید با استفاده از طرح‌بندی نزدیک، با تعیین نکردن شماره طرح و کپی، دو نسخه تنظیم کنید:

```
$ sudo mdadm --create --verbose /dev/md0 --level=10 --raid-devices=4 /dev/sda /dev/sdb /dev/sdc /dev/sdd
```

اگر می خواهید از طرح بندی دیگری استفاده کنید یا تعداد کپی ها را تغییر دهید، باید از گزینه --layout= استفاده کنید که شناسه طرح و کپی را می گیرد. طرح‌بندی‌ها شامل n برای نزدیک، f برای دور و o برای افست هستند. تعداد نسخه‌های ذخیره شده پس از آن اضافه می‌شود.

به عنوان مثال، برای ایجاد یک آرایه که دارای سه نسخه در طرح افست باشد، دستور شامل موارد زیر است:

```
$ sudo mdadm --create --verbose /dev/md0 --level=10 --layout=o3 --raid-devices=4 /dev/sda /dev/sdb /dev/sdc /dev/sdd
```

ابزار mdadm شروع به پیکربندی آرایه می کند. از فرآیند بازیابی برای ساخت آرایه به دلایل عملکرد استفاده می کند. این ممکن است کمی طول بکشد، اما آرایه را می توان در این مدت استفاده کرد. می‌توانید با بررسی فایل /proc/mdstat، پیشرفت آینه‌سازی را کنترل کنید:

```
$ cat /proc/mdstat
```

خروجی دستور بالا به صورت زیر است:

```
Personalities : [raid6] [raid5] [raid4] [linear] [multipath] [raid0] [raid1] [raid10]
md0 : active raid10 sdd[3] sdc[2] sdb[1] sda[0]
      209584128 blocks super 1.2 512K chunks 2 near-copies [4/4] [UUUU]
      [===>.................]  resync = 18.1% (37959424/209584128) finish=13.8min speed=206120K/sec

unused devices: <none>
```

در اولین هایلایت، دستگاه /dev/md0 در پیکربندی RAID 10 با استفاده از /dev/sda، /dev/sdb، /dev/sdc و /dev/sdd ایجاد شده است. دومین هایلایت، طرح‌بندی مورد استفاده برای این مثال را نشان می‌دهد (دو نسخه در پیکربندی نزدیک). سومین ناحیه برجسته پیشرفت در ساخت را نشان می دهد. می‌توانید تا زمانی که این فرآیند کامل شد، راهنما را ادامه دهید.

ایجاد و نصب فایل سیستم

بعد، یک فایل سیستم روی آرایه ایجاد کنید:

```
$ sudo mkfs.ext4 -F /dev/md0
```

سپس، یک نقطه اتصال برای پیوست کردن فایل سیستم جدید ایجاد کنید:

```
$ sudo mkdir -p /mnt/md0
```

با دستور زیر می توانید فایل سیستم را Mount کنید:

```
$ sudo mount /dev/md0 /mnt/md0
```

پس از آن، بررسی کنید که آیا فضای جدید موجود است:

```
$ df -h -x devtmpfs -x tmpfs
```

خروجی به صورت زیر خواهد بود:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        25G  1.4G   23G   6% /
/dev/vda15      105M  3.4M  102M   4% /boot/efi
/dev/md0        197G   60M  187G   1% /mnt/md0
```

فایل سیستم جدید اکنون نصب شده و قابل دسترسی است.

ذخیره چیدمان آرایه

برای اطمینان از اینکه آرایه به طور خودکار در هنگام بوت جمع می شود، باید فایل `/etc/mdadm/mdadm.conf` را تنظیم کنید. شما می توانید به طور خودکار آرایه فعال را اسکن کرده و فایل را با موارد زیر اضافه کنید:

```
$ sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf
```

پس از آن، می‌توانید `initramfs` یا سیستم فایل RAM اولیه را به‌روزرسانی کنید تا آرایه در طول فرآیند بوت اولیه در دسترس باشد:

```
$ sudo update-initramfs -u
```

برای نصب خودکار در هنگام بوت، گزینه های نصب فایل سیستم جدید را به فایل `/etc/fstab` اضافه کنید:

```
$ echo '/dev/md0 /mnt/md0 ext4 defaults,nofail,discard 0 0' | sudo tee -a /etc/fstab
```

آرایه `RAID 10` شما اکنون به طور خودکار هر بوت را مونتاژ و سوار می کند.

اکنون راه اندازی RAID خود را به پایان رسانده اید. اگر می‌خواهید یک RAID دیگر را امتحان کنید، دستورالعمل‌های بازنشانی در انتهای سطوح قرار دارد را دنبال کنید تا با ایجاد یک نوع آرایه RAID جدید ادامه دهید.

نصب و راه‌اندازی RIAD در لینوکس

ابتدا می‌بایست نرم‌افزار `mdadm` را نصب کنیم.

برای سیستم‌های مبتنی بر دبیان:

```
$sudo apt install mdadm
```

برای سیستم‌های مبتنی بر ردهت:

```
$sudo yum install mdadm
```

برای ایجاد یک Raid Array به‌صورت زیر عمل می‌کنیم.

```
$mdadm [mode] [options]
```

دستور mdadm شامل تعدادی گزینه است که در ادامه به توضیح مهمترین آن‌ها می‌پردازیم.

Mode: حالت عملیاتی را مشخص می‌کند و می‌تواند یکی از گزینه های زیر باشد.

- C- یا create– : یک array جدید ایجاد می‌کند
- A- یا assemble– : اجزای یک array از پیش ساخته را به هم ملحق می‌کند.
- B- یا build– : یک array قدیمی بدون superblock ایجاد می‌کند.
- F- یا monitor– , follow– : برای مانیتور کردن یک یا چند array استفاده می‌شود.
- G- یا grow– : اندازه یک array فعال را تغییر می‌دهد
- manage– : برای انجام کارهایی چون اضافه نمودن دیسک جدید و یا حدف نمودن دیسک های معیوب از array به‌کار می‌رود.
- raiddevice : محل و نام Array را مشخص می‌کند. برای مثال /dev/md0

options: می‌تواند گزینه‌های زیر را شامل شود.

- n- یا raid-devices= : تعداد دیسک های فعال در array را مشخص می‌کند.

- x- یا =spare-devices– : تعداد دیسک های یدک را مشخص می‌کند.
- l- یا =level– : سطح RAID را مشخص می‌کند
- s- یا scan– : فایل کانفیگ یا فایل proc/mdstat/ را برای یافتن اطلاعات مربوط جستجو می‌کند.
- e- یا =metadata– : مدل metadata استفاده شده برای array را مشخص می‌کند. مقدار پیشفرض 1.2 می‌باشد
- v- یا verbose— : جهت نمایش اطلاعات بیشتر در رابطه با عملیات در حال انجام استفاده می‌شود.

mdadm دارای گزینه‌های بسیاری است که می‌توانید جهت مطالعه آن‌ها به مستندات Man Pages این برنامه مراجعه کنید

```
$man mdadm
```

برای ایجاد یک array از نوع raid 1 به‌صورت زیر عمل می‌کنیم

```
$ mdadm –create /dev/md0 –level=1 –raid-devices=2 /dev/sda /dev/sdb –spare-devices=1 /dev/sdc
```

در اینجا ما یک array از نوع raid 1 یا mirroring با دو دیسک به نام‌های sda و sdb ایجاد نمودیم؛

همچنین یک دیسک sdc را هم به‌عنوان دیسک یدک برای مواقعی که یکی از دیسک های فعال دچار مشکل شد به array اضافه نمودیم.

برای مشاهده وضعیت array ایجاد شده دستور زیر را وارد می‌کنیم.

```
$mdadm –detail /dev/md0
```

پس از اجرای دستور فوق خروجی زیر را خواهیم داشت.

```
/dev/md0:
        Version : 1.2
  Creation Time : Wed Aug 10 11:36:54 2016
     Raid Level : raid1
     Array Size : 8380416 (7.99 GiB 8.58 GB)
  Used Dev Size : 8380416 (7.99 GiB 8.58 GB)
   Raid Devices : 2
  Total Devices : 3
    Persistence : Superblock is persistent

    Update Time : Wed Aug 10 11:55:45 2016
          State : clean
 Active Devices : 2
Working Devices : 3
 Failed Devices : 0
  Spare Devices : 1

           Name : debian:0  (local to host debian)
           UUID : d26d12cf:b0037b53:cfb06317:a06a841b
         Events : 20

    Number   Major   Minor   RaidDevice State
       0       8       16        0      active sync   /dev/sda
       1       8       32        1      active sync   /dev/sdb

       2       8       48        -      spare         /dev/sdc
```

نکته: می‌توانیم به‌جای وارد نمودن شماره سطح از عنوان سطح مورد نظر برای گزینه level– استفاده کنیم.

برای مثال دستور زیر یک array از نوع raid 0 یا stripe ایجاد می‌کند.

```
$ mdadm -create /dev/md0 -level=stripe -raid-devices=2 /dev/sda /dev/sdb
```

برای ذخیره تنظیمات به‌صورت زیر عمل می‌کنیم.

```
$mdadm -detail -scan /etc/mdadm/mdadm.conf
```

جهت مشاهده وضعیت array ایجاد شده دستور زیر را وارد می‌کنیم.

```
$cat /proc/mdstat
```

که در صورت فعال بودن و تعریف نمودن array خروجی زیر را خواهیم داشت.

```
Personalities : [raid6] [raid5] [raid4] [raid1]
md0 : active raid1 sdc[2](S) sdb[1] sda[0]
      8380416 blocks super 1.2 [2/2] [UU]
unused devices: <none>
```

سپس می‌توانیم array ایجاد شده را با فرمت فایل سیستم مورد نظرمان توسط دستور mkfs به‌صورت زیر فرمت کنیم.

```
$mkfs -t ext4 /dev/md0
```

و در نهایت array ایجاد شده را ماونت می‌کنیم.

```
$mount /dev/md0 /mnt
```

در صورتی که بخواهیم array مورد نظر را غیرفعال کنیم دستور زیر را وارد می‌کنیم.

```
$mdadm –stop /dev/md0
```

که در اینجا md0 غیرفعال گردید و برای فعال نمودن مجدد array به صورت زیر عمل می‌کنیم.

```
$mdadm –assemble –scan
```

برای حذف یک دیسک از array بصورت زیر عمل می‌کنیم.

ابتدا توسط دستور زیر دیسک را به حالت fail تغییر وضعیت می‌دهیم.

```
$mdadm /dev/md0 –fail /dev/sdc
```

سپس دستور زیر را برای حذف دیسک از array وارد می‌کنیم.

```
$mdadm /dev/md0 –remove /dev/sdc
```

و برای اضافه نمودن مجدد یک دیسک به array دستور زیر را وارد می‌کنیم.

```
$mdadm /dev/md0 –add /dev/sdc
```

بازنشانی دستگاه‌های موجود در raid

این بخش به نحوه بازنشانی دستگاه‌ها پس از raid می‌پردازد.

هشدار: این فرآیند آرایه و هر داده ای که روی آن نوشته شده است را به طور کامل از بین می برد. مطمئن شوید که روی آرایه صحیح کار می کنید و هر داده ای را که باید قبل از از بین بردن آرایه حفظ کنید کپی کرده اید.

با یافتن آرایه های فعال در فایل /proc/mdstat شروع کنید:

```
$ cat /proc/mdstat
```

خروجی این دستور به صورت زیر است:

```
Output

Personalities : [raid0] [linear] [multipath] [raid1] [raid6] [raid5]
[raid4] [raid10]

md0 : active raid0 sdc[1] sdd[0]

 209584128   blocks super 1.2 512k chunks



      unused devices: <none>
```

سپس آرایه را از فایل سیستم جدا کنید:

```
$ sudo umount /dev/md0
```

حالا mdadm را متوقف کنید و آرایه را حذف کنید:

```
$ sudo mdadm --stop /dev/md0
```

دستگاه هایی که برای ساخت آرایه استفاده شده اند را با دستور زیر پیدا کنید:

هشدار: به خاطر داشته باشید که نام /dev/sd* می‌تواند هر زمان که راه‌اندازی مجدد کنید تغییر کند. هر بار آنها را بررسی کنید تا مطمئن شوید که روی دستگاه های درست کار می کنید.

```
$ lsblk -o NAME,SIZE,FSTYPE,TYPE,MOUNTPOINT
```

خروجی به صورت زیر است:

```
Output
NAME     SIZE FSTYPE            TYPE MOUNTPOINT
sda      100G linux_raid_member disk
sdb      100G linux_raid_member disk
sdc      100G                   disk
sdd      100G                   disk
vda       25G                   disk
├─vda1  24.9G ext4               part /
├─vda14    4M                    part
└─vda15  106M vfat               part /boot/efi
vdb      466K iso9660           disk
```

پس از پیدا کردن دستگاه‌هایی که برای ایجاد آرایه استفاده می‌شوند، سوپر بلوک آن‌ها را که متادیتا را برای راه‌اندازی RAID نگه می‌دارد، صفر کنید. صفر کردن این، متادیتای RAID را حذف می کند و آنها را به حالت عادی بازنشانی می کند:

```
$ sudo mdadm --zero-superblock /dev/sda
$ sudo mdadm --zero-superblock /dev/sdb
```

همچنین توصیه می شود هر گونه ارجاع دائمی به آرایه را حذف کنید. فایل `/etc/fstab` را ویرایش کنید و یا ارجاع به آرایه خود را حذف کنید. می توانید با قرار دادن نماد هشتگ # در ابتدای خط، با استفاده از nano یا ویرایشگر متن دلخواه خود، عملیات مورد نظر خود را بیان کنید:

```
$ sudo nano /etc/fstab
```

/etc/fstab

```
. . .
# /dev/md0 /mnt/md0 ext4 defaults,nofail,discard 0 0
```

همچنین، تعریف آرایه را از فایل `/etc/mdadm/mdadm.conf` کامنت کنید یا حذف کنید:

```
$ sudo nano /etc/mdadm/mdadm.conf
```

/etc/mdadm/mdadm.conf

```
. . .
# ARRAY /dev/md0 metadata=1.2 name=mdadmwrite:0 UUID=7261fb9c:976d0d97:30bc63ce:85e76e91
```

در نهایت، `initramfs` را به‌روزرسانی کنید تا فرآیند راه‌اندازی اولیه سعی در آوردن یک آرایه غیرقابل دسترس نداشته باشد:

```
$ sudo update-initramfs -u
```

از اینجا، باید آماده استفاده مجدد از دستگاه های ذخیره سازی به صورت جداگانه یا به عنوان اجزای یک آرایه متفاوت باشید.

همچنین شما می‌توانید مقاله چطور با `mdadm` در اوبونتو آرایه‌های RAID را مدیریت کنید را ببینید.

آموزش نصب لینوکس (اوبونتو)

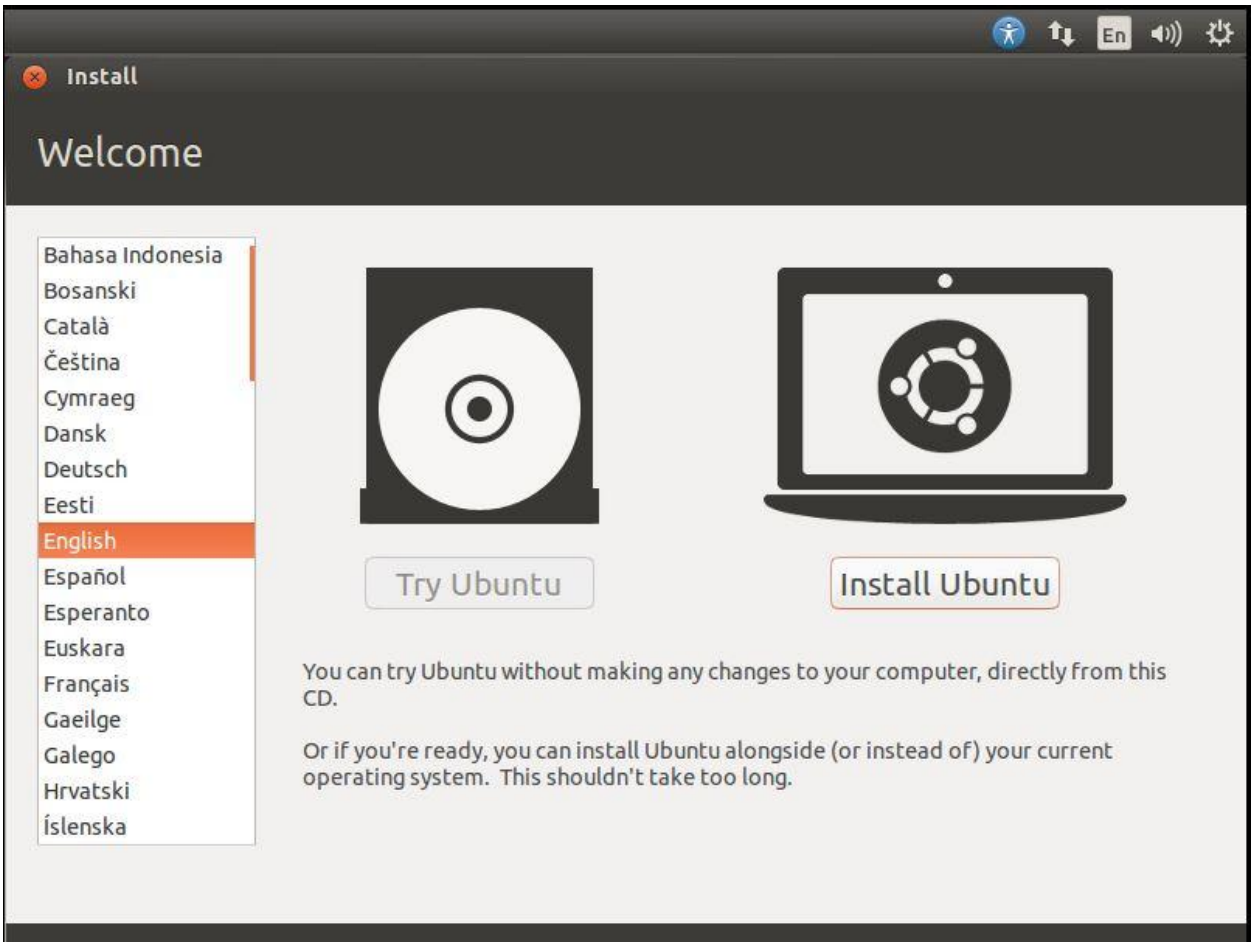
برای نصب این سیستم عامل اول این سیستم عامل رو از سایتش دانلود کنید. دانلود سیستم عامل UBUNTU،
بعد از دانلود نسخه مورد نظر خود باید این سیستم عامل را روی یک DVD یا فلش BOOT کنید برای آموزش
نحوه BOOT کردن از روی فلش آموزش بوتیبل نمودن فلش و برای BOOT کردن روی DVD آموزش رایت
ISO بعد از آن DVD یا FLASH را به سیستم خود نصب کرده و سپس طبق مراحل زیر عمل کنید.

(این آموزش مربوط به نسخه 14.04 اوبونتو می‌باشد اما تفاوت زیادی با نسخه 22.04.1 ندارد)

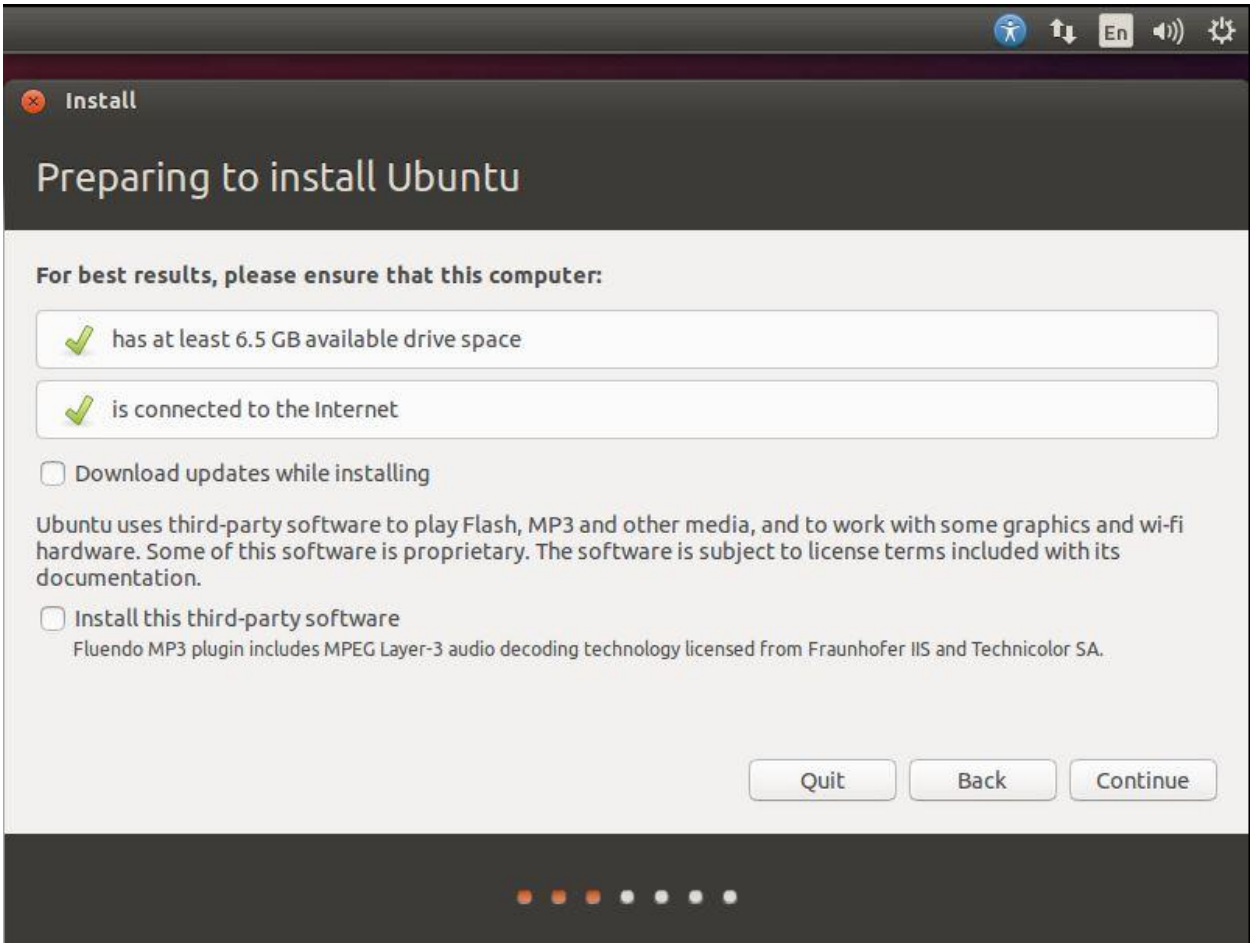
1 – بعد از ظاهر شدن پنجره بنفش سیستم عامل UBUNTU با موس بر روی صفحه کلیک کنید تا مراحل گرافیکی و راحت نصب ظاهر شود در صورت فشار دادن کلیدی از روی KEYBOARD مراحل نصب متفاوت می شود.

پس از کلیک منتظر می مانیم تا مرحله Install ظاهر شود.

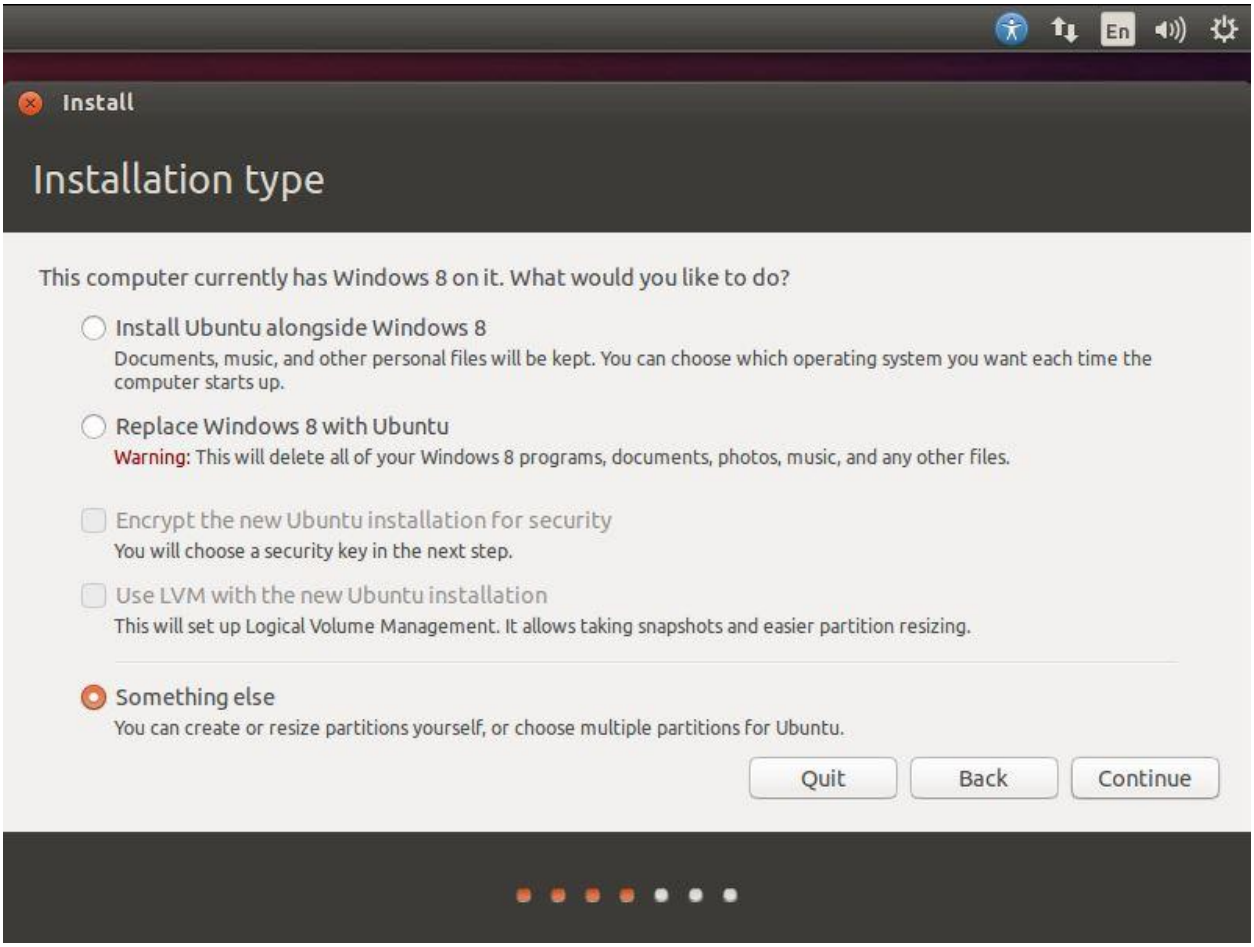
در اینجا همانطور که می بینید می توانید بر روی گزینه Install Ubuntu یا Try Ubuntu کلیک کنید که ما برای نصب رو گزینه Install Ubuntu کلیک می کنیم.



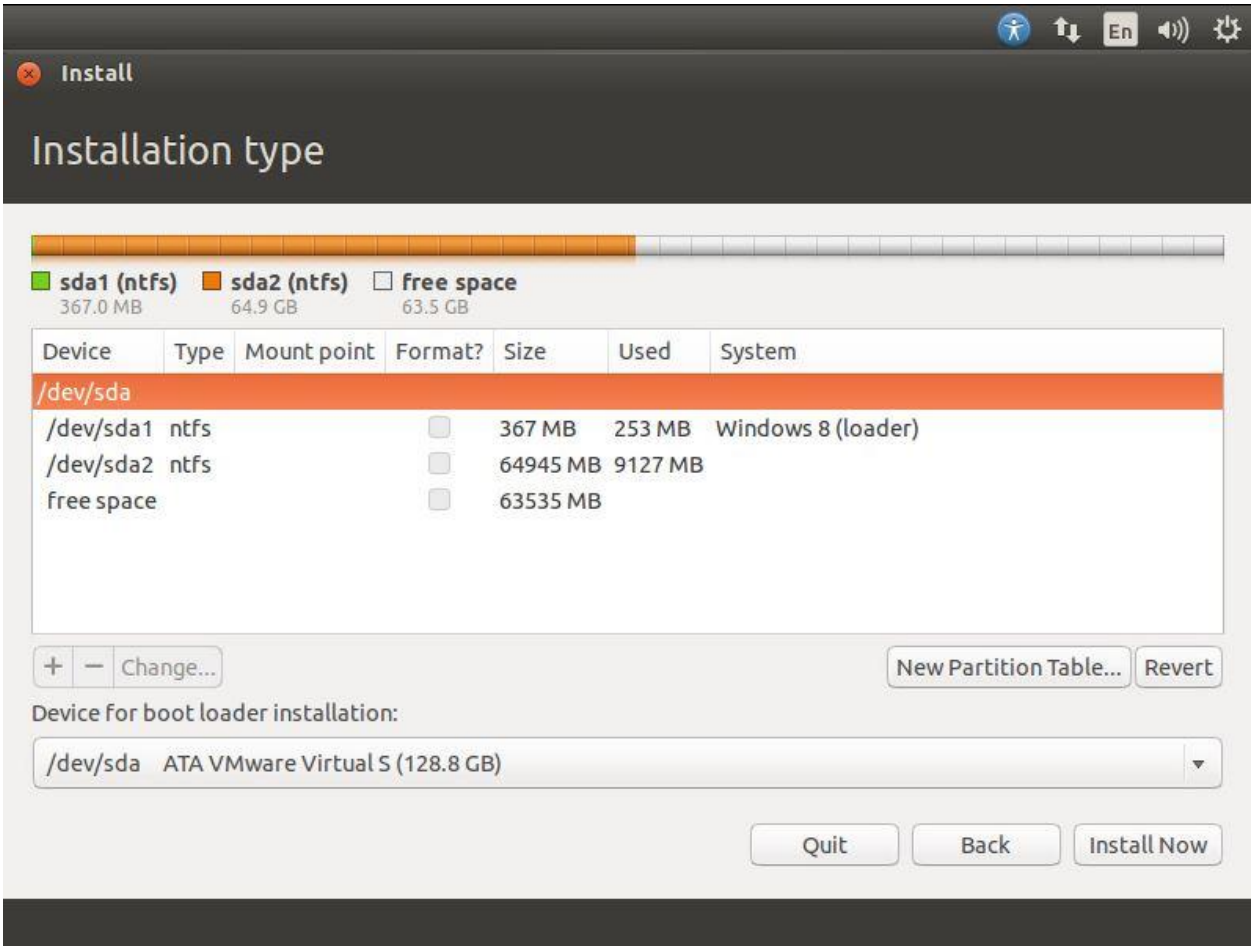
بعد از این مرحله به مرحله Preparing to Install Ubuntu میرسیم که در این مرحله در صورتی که شما مایل هستید می توانید گزینه Download Updates While Installing رو علامت بگذارید تا در هنگام نصب UBUNTU سیستم عامل شما بروز هم شود و همچنین گزینه Install This Third-Party Software رو هم علامت بگذارید تا بتوانید تمامی فایلهای صوتی و تصویری را اجرا کنید، و سپس روی گزینه Continue کلیک کنید.



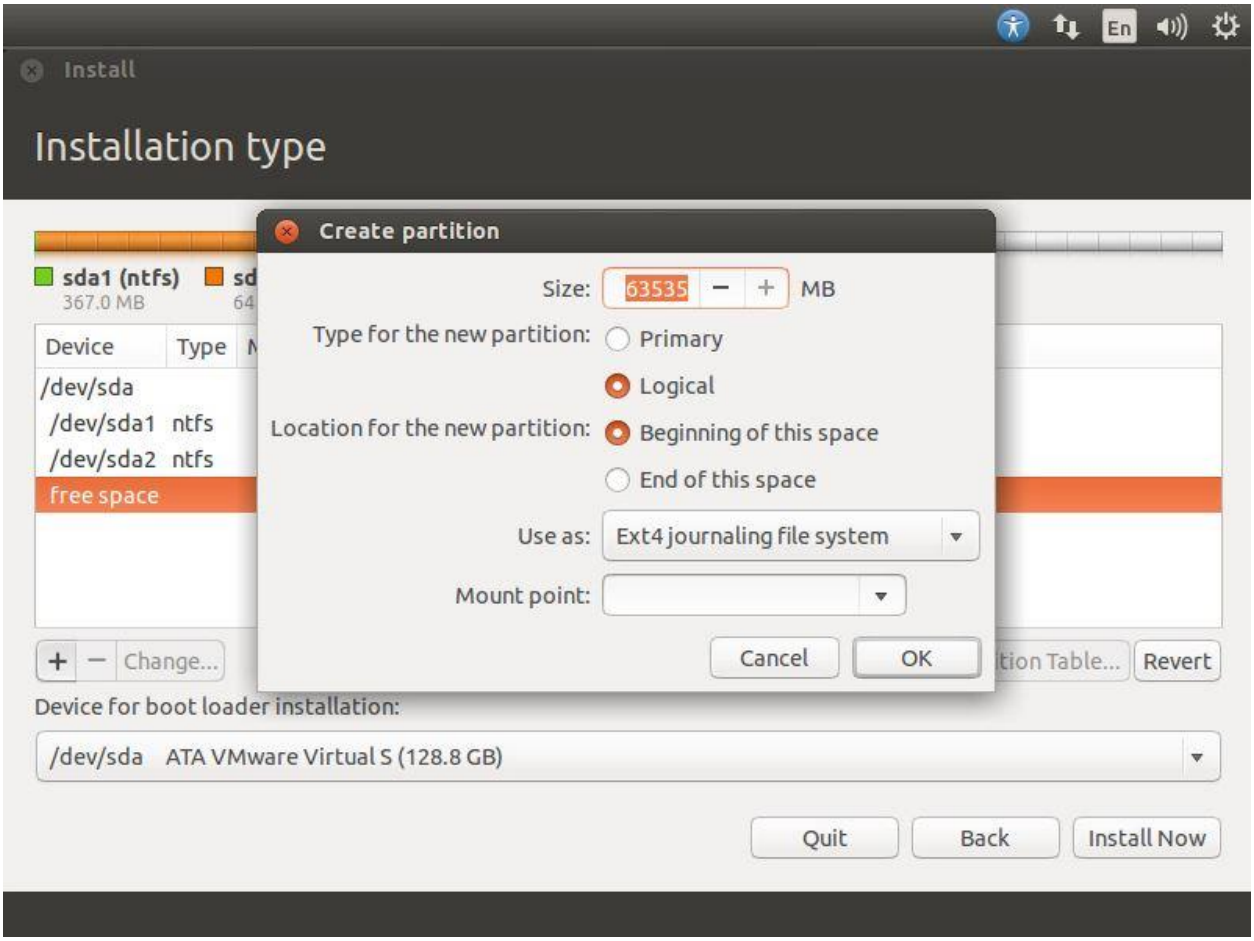
خب در این مرحله شما باید بگوید که آیا می خواهید UBUNTU را در کنار Windows نصب کنید یا می خواهید کل Hard Disk خود را به UBUNTU اطلاق دهید یا این که می خواهید خودتان Partition خاصی رای به UBUNTU اطلاق دهید در صورتی که می خواهید یک PARTITION خاص را به UBUNTU اطلاق دهید ادامه مطلب را دنبال کنید و روی گزینه Something Else کلیک و سپس روی گزینه Continue کلیک کنید.



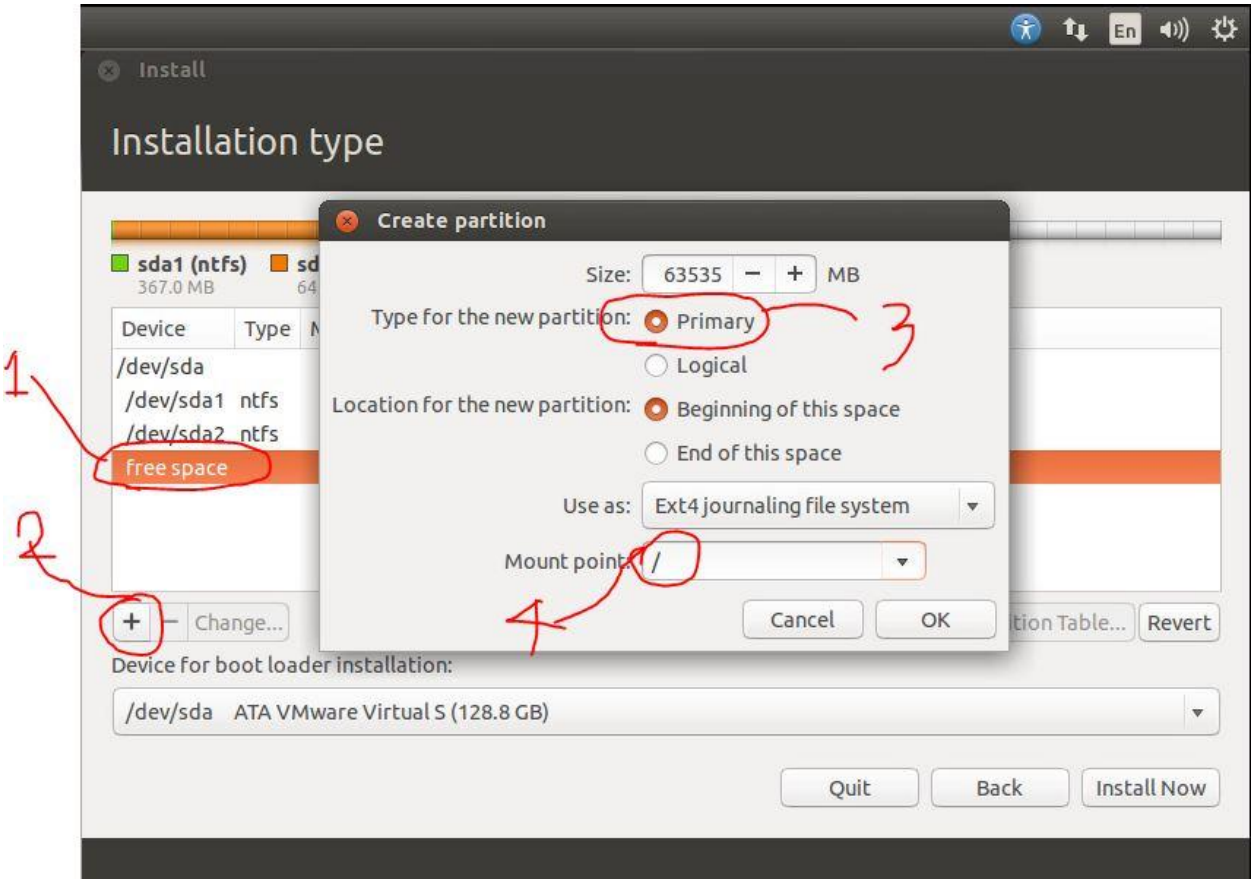
در مرحله بعد به قسمت انتخاب PARTITION می رسیم که شما می توانید یکی از Partition های خود را که از قبل آماده کردید به این سیستم عامل اطلاق دهید، در این جا من دو Partition دارم که یکی System Reserved که برای نگهداری فایل Boot ویندوز 8 من هست و دیگری برای سیستم عامل ویندوز هشت من است خب و 63 گیگابایت فضای خالی دارم که می خواهم این فضا را به سیستم عامل Ubuntu خود اطلاق دهم.



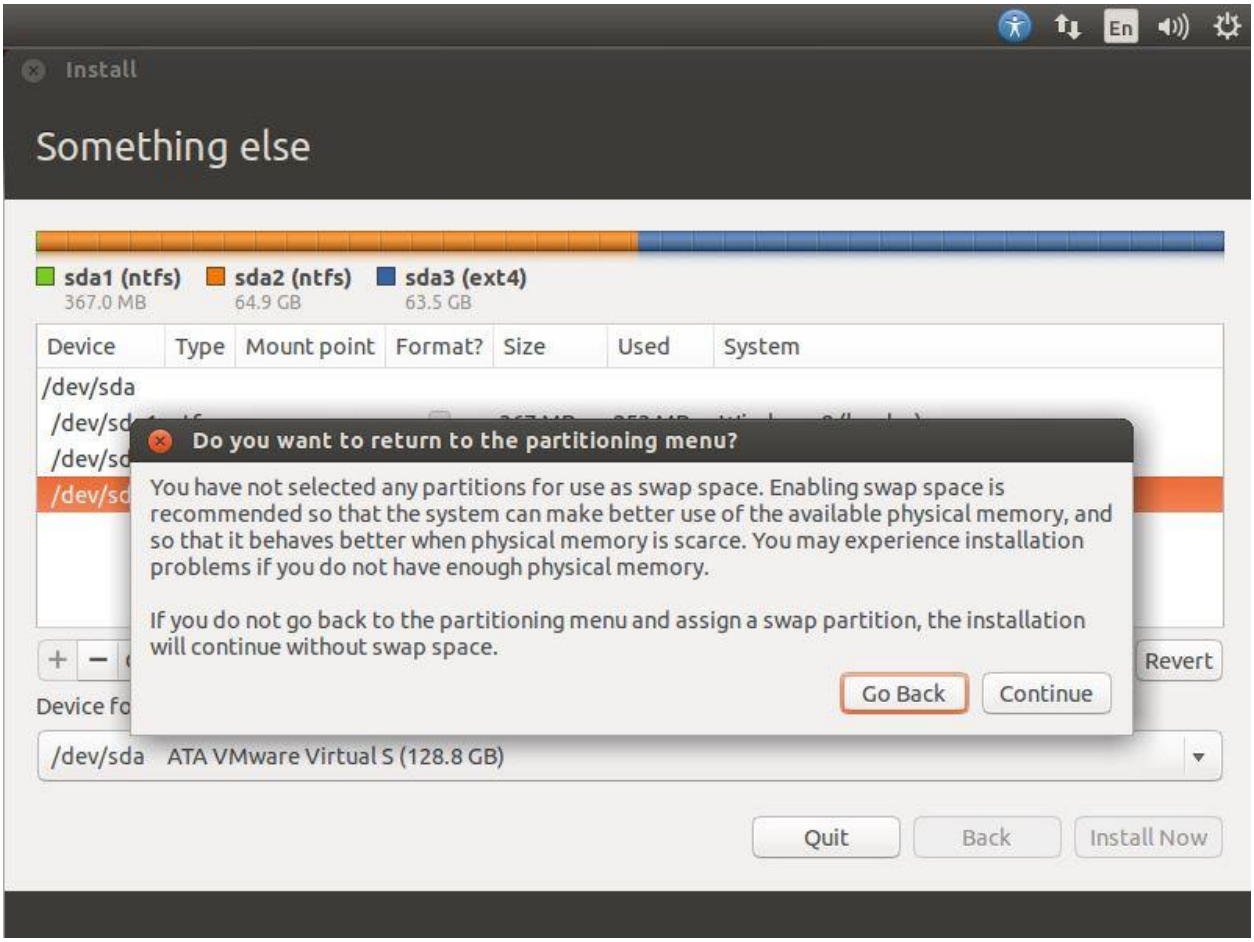
خب در اینجا اگر شما Partition خالی داشته باشید می توانید از آن استفاده کنید حالا من می خواهم یک Partition بسازم که بشه روی این Partition سیستم عامل رو نصب کرد پس روی Free Space کلیک می کنم و بعد روی گزینه + در قسمت پایین بغل گزینه change... و



در پنجره باز شده با نام Create Partition شما می توانید حجم Partition ، نوع ، مکان شروع ، استفاده به عنوان و Mount Point را مشخص کنید که من این Partition رو به عنوان یک Partition Primary و نقطه شروع رو از همون Beginning of this Space و Ext4 journaling file system رو انتخاب می کنم گزینه Mount Point را بر روی / می گذارم، و سپس روی گزینه Ok کلیک می کنیم. خب اگه مایل باشید می توانید یک Partition Swap هم برای بازدهی بهتر سیستم درست کنید.

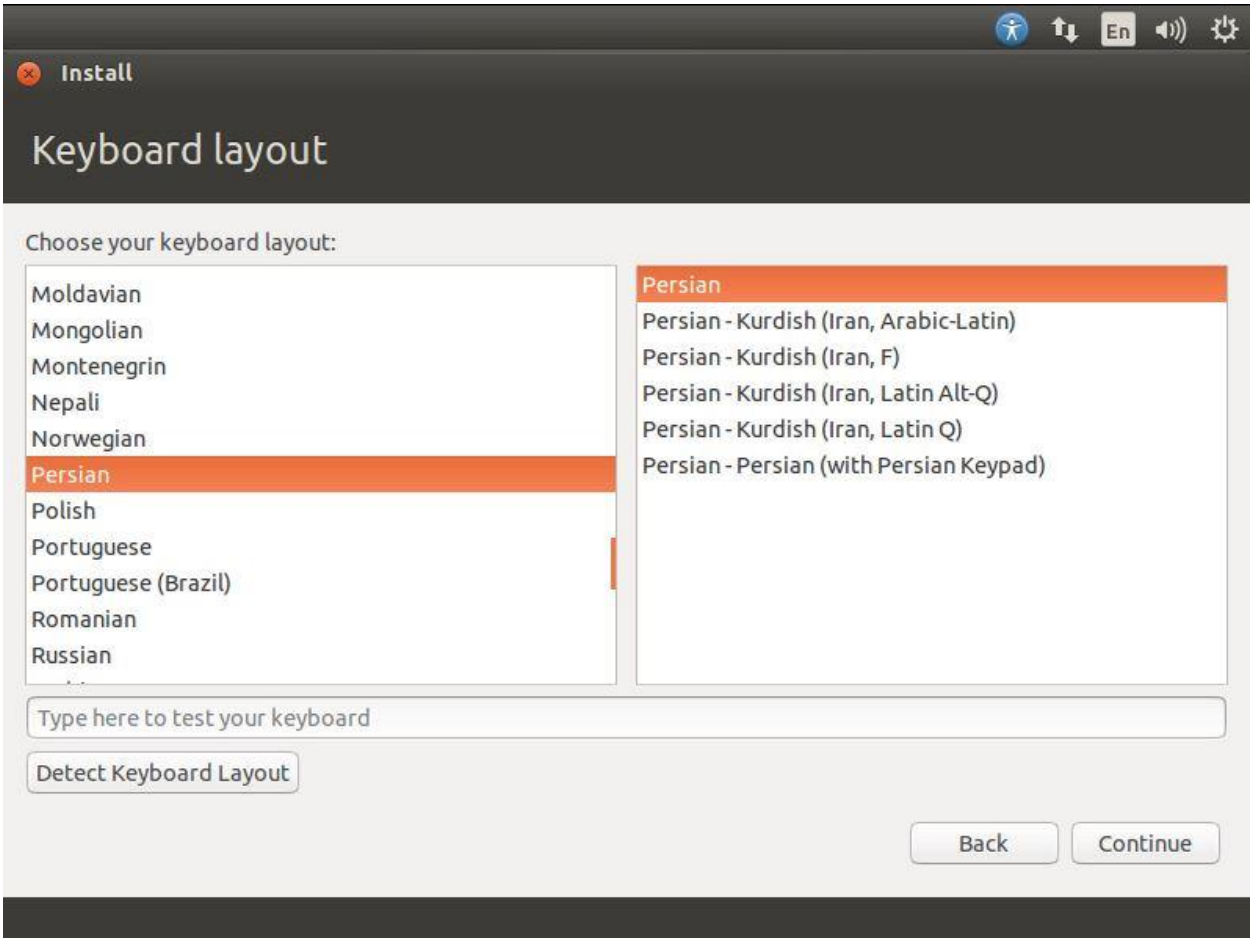


برای ساخت Partition Swap روی گزینه + کلیک کنید همه مراحل رو مثل قبل انجام بدید ولی حجم Partition Swap رو اندازه حجم Ram سیستم قرار بدید یعنی اگر سیستم شما Ram 4 GB دارد پس حجم Partition Swap هم باید 4 GB باشد پیشنهاد من این که یک MB100 بیشتر از RAM باشه بهتره و Partition هم فرقی نداره که روی Logical باشه یا Primary و برای Use as هم همان گزینه پیش فرض خودش خوبه و فقط برای Mount Point روی گزینه Swap کلیک کنید در این آموزش من Partition Swap درست نمی کنم پس بهم یک پیام میده بر مبنای این که شما هیچ Partition رو برای Swap انتخاب نکردید. پس روی همون Continue کیک می کنم

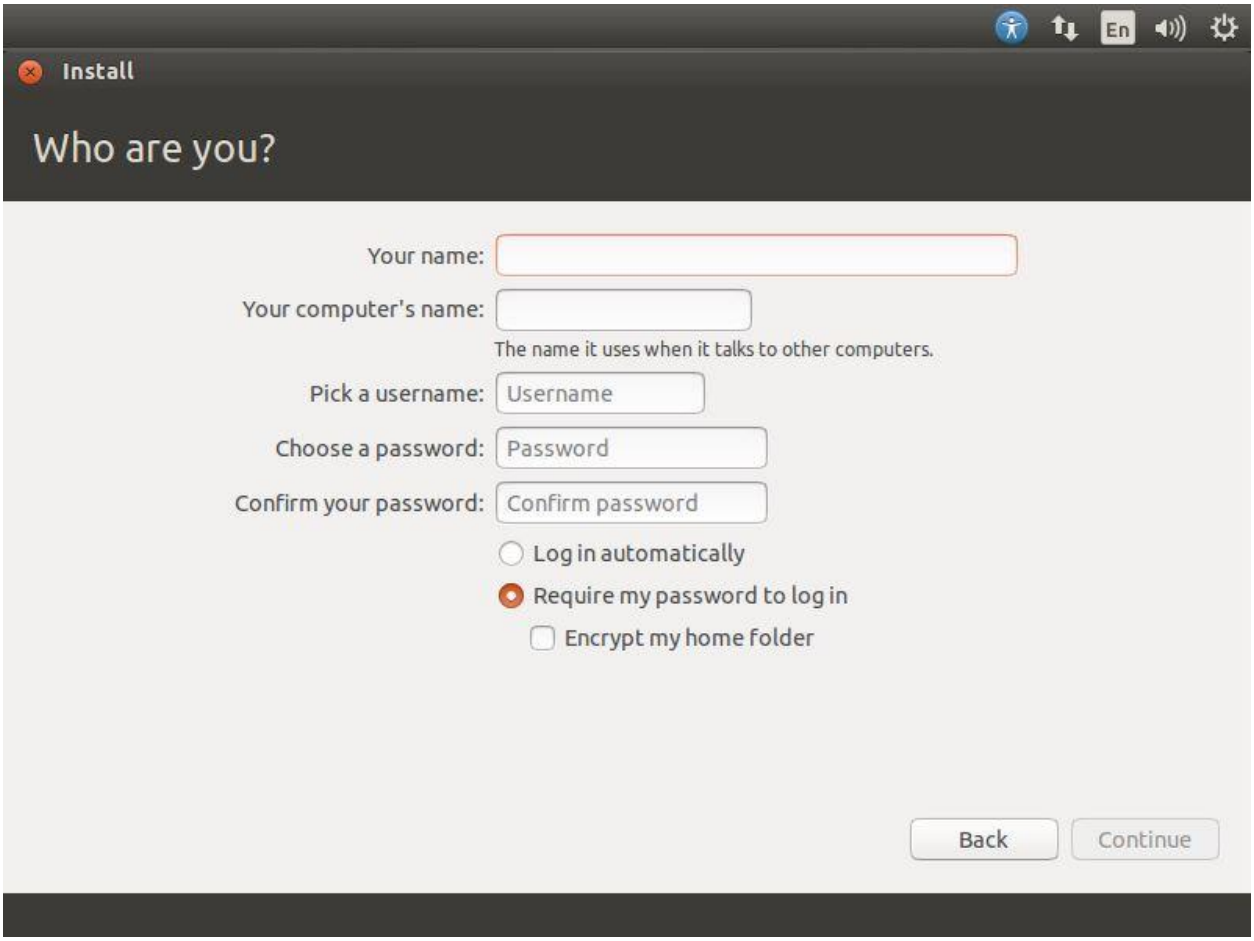


در قسمت بعد از روی نقشه کشور ایران رو انتخاب می کنیم و سپس روی گزینه Continue کلیک می کنیم.

در این قسمت هم اگر در قسمت قبل ایران را در نقشه انتخاب کرده باشید زبان فارسی به صورت خودکار انتخاب شده است و کافیست روی گزینه Continue کلیک کنید.

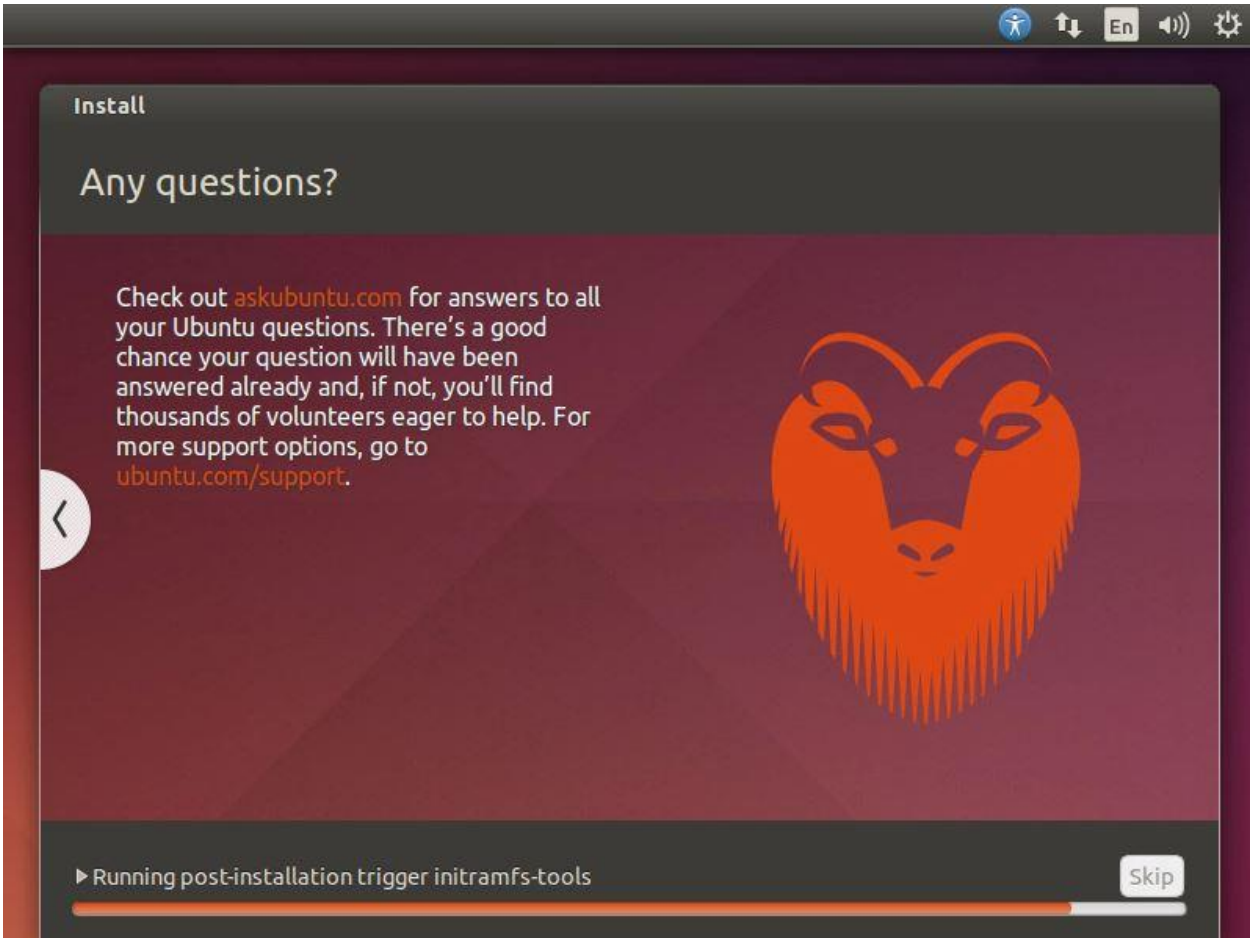


خب به قسمت انتخاب نام برای سیستم و برای حساب کاربری می رسیم، در فیلد اول در صورت تمایل نام و نام خانوادگی خود را وارد کنید و فیلد دوم نام کامپیوتر خود در فیلد سوم نام حساب کاربری و در فیلد چهارم یک رمز عبور و در فیلد پنجم رمز عبور خود را تکرار کنید و اگر مایل هستید می توانید روی گزینه Log in Automatically کلیک کنید تا سیستم به صورت خود کار بدون احراز هویت و درخواست رمز از شما وارد سیستم شود و در صورتی که خواستار امنیت بیشتر هستید می توانید روی گزینه Encrypt My Home Folder علامت بگذارید تا امنیت سیستم شما افزایش یابد و سپس روی گزینه Continue کلیک کنید.

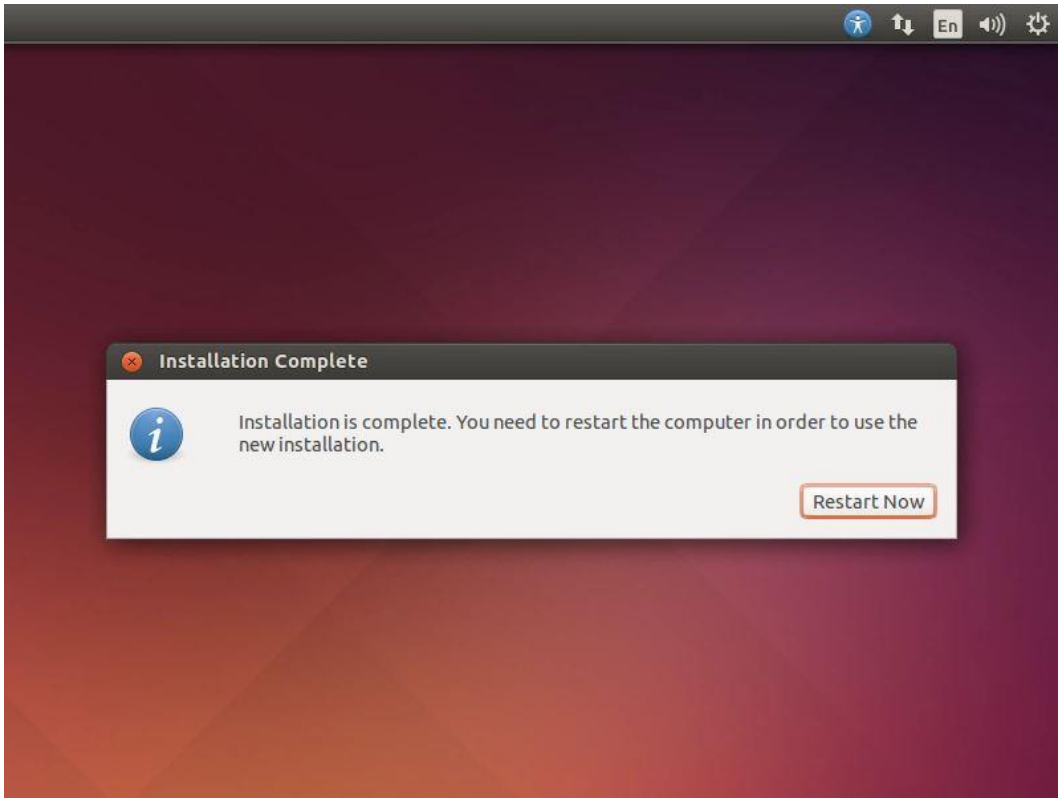


نکته: فیلد دوم و سوم به صورت خود کار بعد از پر کردن فیلد اول پر می شود می توانید آنها را به صورت دستی عوض کنید.

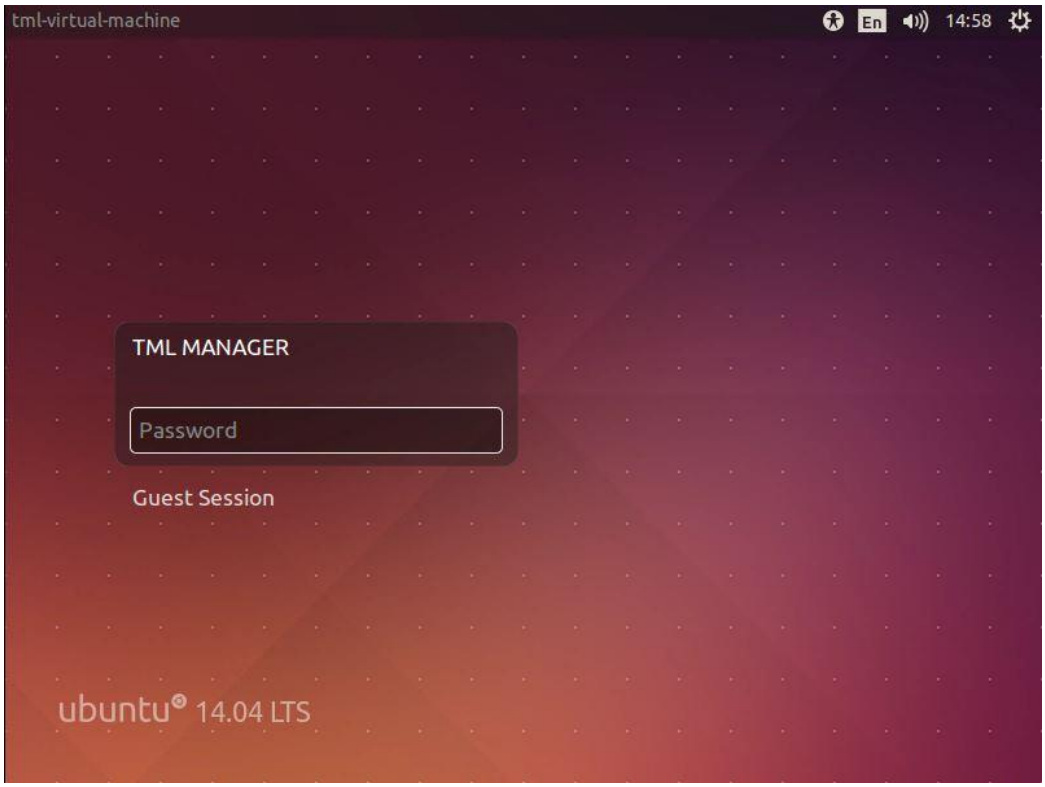
حالا صبر کنید تا مراحل کپی و پیکر بندی کردن سیستم عامل شما تمام شود



و در قسمت بعد روی گزینه Restart Now کلیک می کنیم تا سیستم عامل ما ریستارت شود و بعد از ریستارت وارد سیستم می شویم.



سپس رمز را وارد کرده و به دنیای Linux خوش آمدید.



چطور کنار ویندوز لینوکس را نصب کنیم؟

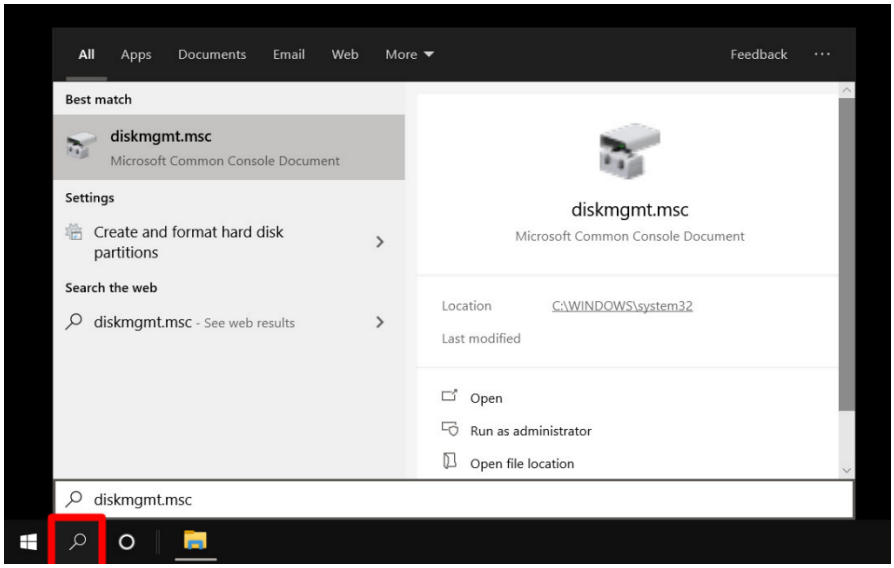
برای اینکار راه‌های زیر وجود دارند:

- نصب لینوکس با ویندوز به صورت بوت دوگانه
- نصب لینوکس از طریق ماشین مجازی (مثل VirtualBox)
- نصب لینوکس به کمک Windows Subsystem for Linux

همانطور که می‌بینید یکی از راه‌ها به صورت بوت دوگانه است که ما این روش را پی میگیریم تا بتوانیم باز به مراحل نصب لینوکس برسیم:

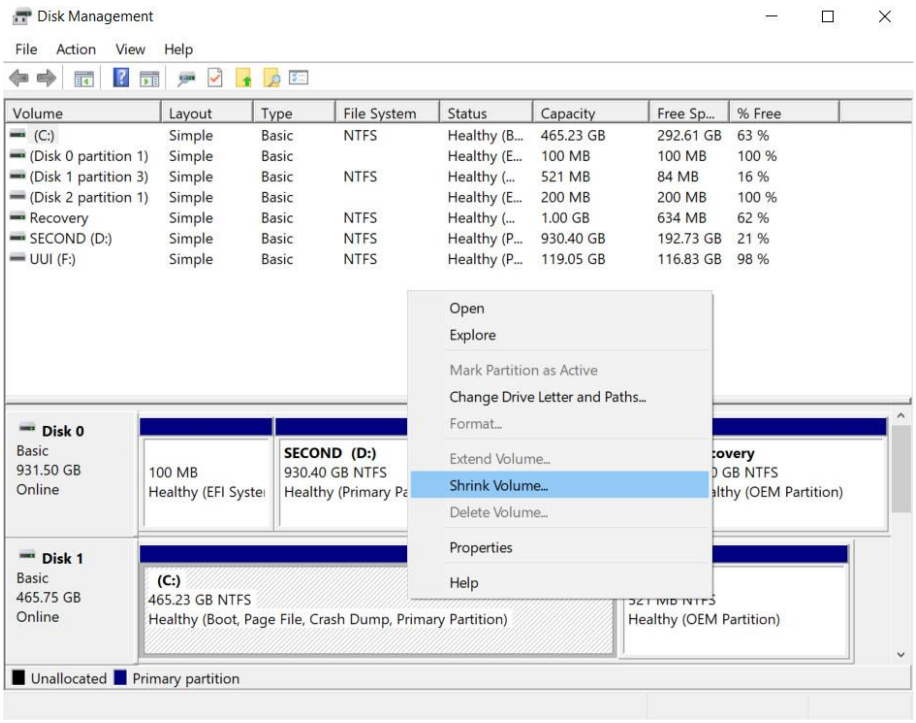
پارتیشن بندی لینوکس

پیش از شروع فرایند نصب لینوکس، باید یک پارتیشن قابل بوت برای آن در نظر بگیرید. اگر از قبل یک پارتیشن خالی با چنین شرایطی در اختیار دارید، نیازی به ادامه این مرحله نخواهید داشت و می‌توانید مستقیم به قسمت بعدی بروید؛ در غیر این صورت باید یکی پارتیشن جدید برای لینوکس بسازید.



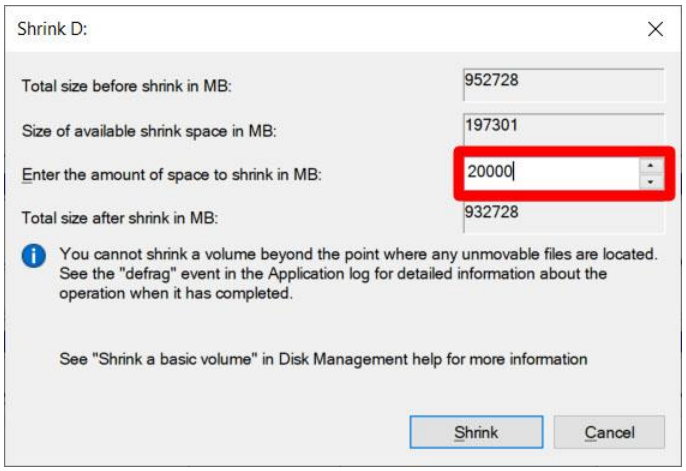
جهت ساخت یک پارتیشن جدید، ابتدا در نوار جستجوی استارت منو عبارت `DISKMGMT.MSC` را وارد کنید و سپس کلید `Enter` را بزنید. همچنین می‌توانید با راست کلیک روی آیکن استارت و انتخاب گزینه `Run`، عبارت گفته شده را وارد نموده و سپس `Run` را بزنید.

اکنون باید به ابزار `Disk Management` دسترسی داشته باشید که محیط آن مشابه با تصویر زیر است. از این ابزار می‌توانید برای جداسازی یا ادغام پارتیشن‌ها و مدیریت فضای ذخیره‌سازی دستگاه استفاده نمایید.



در مرحله بعد روی درایوی که ویندوز روی آن نصب شده (معمولا درایو C) راست کلیک کرده و گزینه Shrink Volume را انتخاب نمایید. دلیل انتخاب این درایو، امکان بوت سیستم عامل از روی آن است؛ اما اگر درایو دیگری با این قابلیت در اختیار دارید، می‌توانید از آن هم استفاده نمایید.

حالا باید مقدار فضایی که قصد اختصاص دادن آن به پارتیشن نصب لینوکس را دارید، تعیین کنید. معمولا توصیه می‌شود حداقل ۲۰ گیگابایت برای مقدار گفته شده در نظر بگیرید؛ اما با توجه به توزیع مورد استفاده می‌توانید از حجم بیشتر یا کمتر نیز استفاده کنید. توجه داشته باشید که درایو انتخاب شده، باید به همین مقدار فضای خالی در اختیار داشته باشد.



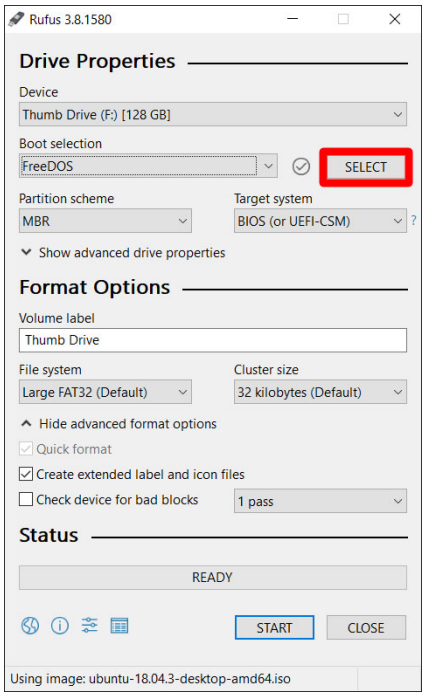
در نهایت روی کلید Shrink کلیک کنید تا یک پارتیشن جدید از درایو انتخاب شده، ساخته شود. در ادامه از این پارتیشن برای نصب لینوکس استفاده خواهیم کرد.

دانلود توزیع لینوکس و ساخت فلش قابل بوت

برای نصب لینوکس مانند ویندوز، باید ابتدا فایل ISO توزیع مورد نظر را دانلود کنید و سپس آن را با نرم افزار اختصاصی روی فلش قرار دهید. به این صورت یک فلش قابل بوت در اختیار خواهیم داشت که می‌توانیم از آن برای نصب سیستم عامل یا اجرای نسخه Live استفاده نماییم. از میان محبوب‌ترین توزیع‌های لینوکس می‌توانیم به اوبونتو، مینت و فدورا اشاره کنیم که هر سه آن‌ها به صورت رایگان از طریق وب سایت توسعه‌دهنده قابل دانلود خواهد بود. در این مثال، ما به سراغ توزیع اوبونتو خواهیم رفت؛ اما شما قادر هستید تا از هر توزیع دلخواه دیگری بهره بگیرید.

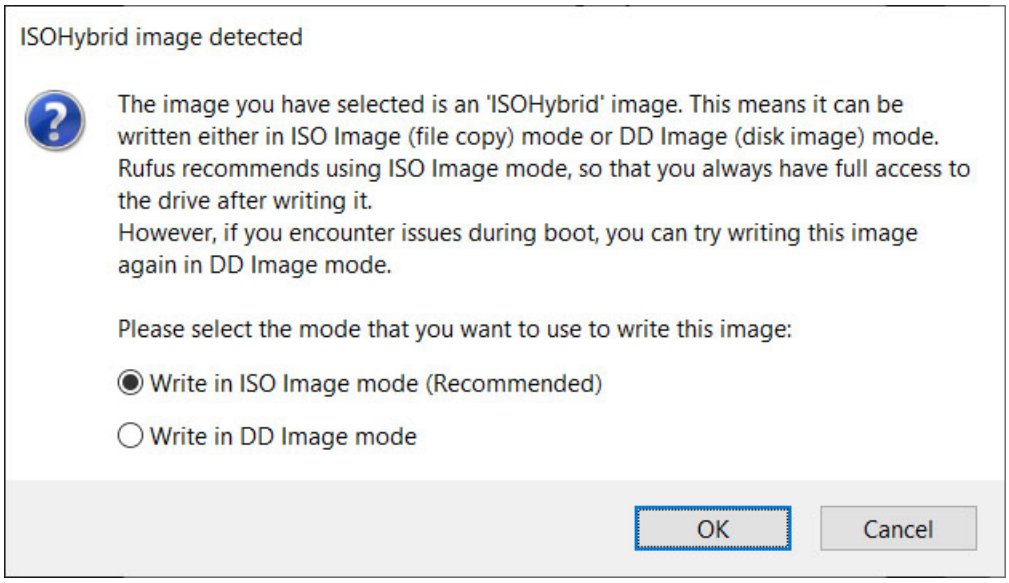
فایل ISO اوبونتو را از وب سایت Ubuntu.com دانلود می‌کنیم. همچنین برای کپی کردن این فایل روی فلش به نرم افزار روفوس احتیاج خواهیم داشت که از طریق وب سایت Rufus.ie به شکل رایگان قابل دانلود است.

حالا باید یک حافظه فلش USB با حداقل ۸ گیگابایت ظرفیت را به کامپیوتر متصل کنیم. توجه داشته باشید که حافظه در ادامه فرمت خواهد شد و تمامی داده‌های قرار گرفته روی آن حذف می‌شود؛ پس در صورتی که اطلاعات ضروری روی آن دارید، آن‌ها را در محل دیگری کپی کنید.



اکنون به سراغ برنامه روفوس که به تازگی دانلود کرده‌ایم، خواهیم رفت و آن را اجرا می‌کنیم. در قسمت Boot Selection روی کلید SELECT می‌زنیم و به مسیر قرارگیری فایل ISO اوبونتو می‌رویم و آن را انتخاب می‌کنیم.

همچنین از قسمت Device حافظه فلش را انتخاب می‌کنیم. البته به صورت پیش‌فرض این کار به شکل خودکار صورت می‌گیرد؛ مگر اینکه فلش یا مموری کارت دیگری نیز به سیستم متصل باشد. نیازی به تغییر سایر گزینه‌ها نیست و می‌توانید آن‌ها را روی حالت مشابه با تصویر بالا قرار دهید.

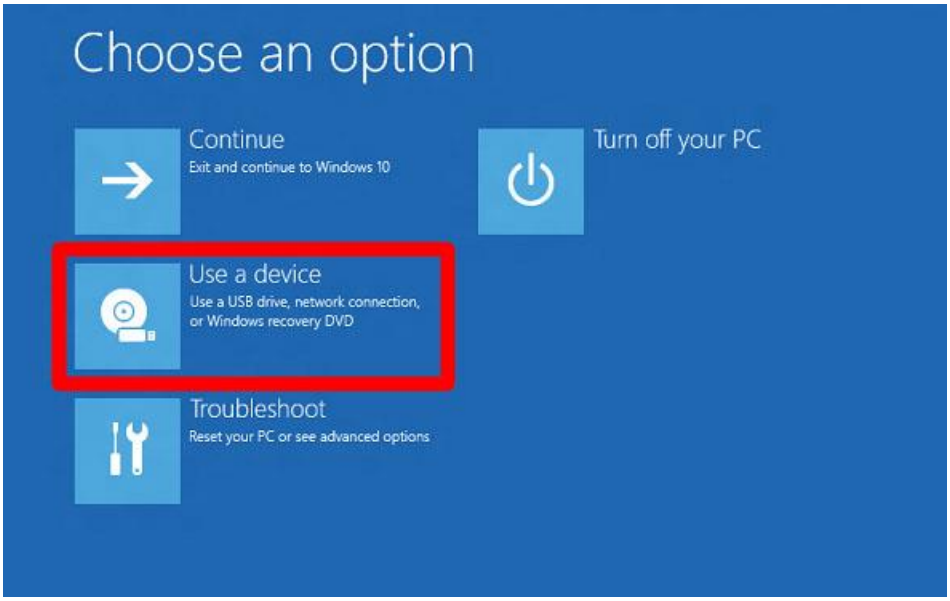


در نهایت روی کلید Start بزنید تا فرایند کپی کردن فایل‌ها روی حافظه فلش شروع شد. در صورتی که یک پنجره مشابه تصویر بالا نمایش داده شد، اولین گزینه را انتخاب کرده و روی OK کلیک کنید. فرایند کپی کردن فایل‌ها با توجه به حجم توزیع و سرعت فلش متغیر خواهد بود.

نصب لینوکس از روی فلش

مهم‌ترین قسمت نصب لینوکس، بوت کردن نصب‌کننده از روی فلش است. این کار به چندین روش مختلف از جمله رفتن به بایوس و تغییر تنظیمات بوت امکان‌پذیر است. اگر تجربه بوت کردن دستگاه از روی فلش یا سایر حافظه‌های جانبی را دارید، می‌توانید به روش خودتان عمل کنید؛ در غیر این صورت طبق مسیر زیر پیش بروید.

درحالی که حافظه فلش به دستگاه متصل است، روی استارت منو کلیک کرده و منوی پاور را باز کنید. حالا همزمان با نگه داشتن کلید Shift روی گزینه Restart بزنید. با این کار وارد قسمت Windows Recovery Environment می‌شویم.



در منوی ریکاوری با عنوان Choose an option روی گزینه Use a device کلیک نمایید.

اکنون در صفحه Use a device با چندین گزینه مختلف مواجه خواهید شد. تنها کافی است تا روی گزینه با عنوان Linux یا Ubuntu (یا نام هر توزیع دیگر) کلیک کنید. در صورتی که چنین گزینه‌ای مشاهده نمی‌کنید، روی EFI USB Device بزنید.

حالا کامپیوتر بار دیگر از روی فلش بوت می‌شود و وارد محیط نصب لینوکس خواهد شد. در صورتی که دوباره به محیط ویندوز بازگشتید، مشکلی در حافظه فلش یا فرایند کپی کردن فایل‌ها وجود داشته است. همچنین احتمال دارد تنظیمات بایوس دستگاه از بوت شدن توسط حافظه USB جلوگیری به عمل آورد که با رفتن به منوی آن، قابل تغییر است. برای راهنمایی بیشتر به سراغ مطلب «چگونه وارد منوی تنظیمات بوت و BIOS شویم؟» بروید.

ادامه این نصب را می‌توانید در قسمت آموزش نصب لینوکس (اوبونتو) ببینید.

انجام پروسه raid در لپ تاپ خودم

من برای انجام این raiding از اوبونتو دسکتاب نسخه 22.04.1 که یک نسخه LTS است استفاده کرده‌ام.

برای اینکه بتوانیم اسامی دیسک‌ها یا همان حافظه‌های سیستم را ببینیم دستور lsblk را می‌زنیم.

```
zodiac@Mateo:~$ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
loop0         7:0    0     4K  1 loop /snap/bare/5
loop1         7:1    0  55.6M  1 loop /snap/core18/2667
loop2         7:2    0    62M  1 loop /snap/core20/1587
loop3         7:3    0  63.3M  1 loop /snap/core20/1778
loop4         7:4    0  72.9M  1 loop /snap/core22/469
loop5         7:5    0  72.9M  1 loop /snap/core22/484
loop6         7:6    0 163.3M  1 loop /snap/firefox/1635
loop7         7:7    0 400.8M  1 loop /snap/gnome-3-38-2004/112
loop8         7:8    0 346.3M  1 loop /snap/gnome-3-38-2004/119
loop9         7:9    0  91.7M  1 loop /snap/gtk-common-themes/1535
loop10        7:10   0  45.9M  1 loop /snap/snap-store/582
loop11        7:11   0  49.6M  1 loop /snap/snapd/17883
loop12        7:12   0  49.8M  1 loop /snap/snapd/17950
loop13        7:13   0   284K  1 loop /snap/snapd-desktop-integration/14
loop14        7:14   0   304K  1 loop /snap/snapd-desktop-integration/49
loop15        7:15   0 375.1M  1 loop /snap/telegram-desktop/4470
loop16        7:16   0 375.1M  1 loop /snap/telegram-desktop/4486
loop17        7:17   0 320.4M  1 loop /snap/vlc/3078
sda           8:0    1  14.8G  0 disk
└─sda1        8:1    1  14.8G  0 part /media/zodiac/jet
sdb           8:16   1   7.5G  0 disk
└─sdb1        8:17   1   7.5G  0 part /media/zodiac/data
sdc           8:32   1   7.2G  0 disk
└─sdc1        8:33   1   7.2G  0 part /media/zodiac/FED5-CBAD
sdd           8:48   1  14.8G  0 disk
└─sdd1        8:49   1  14.8G  0 part /media/zodiac/FFA5-DB7F
nvme0n1     259:0    0 476.9G  0 disk
├─nvme0n1p1 259:1    0   100M  0 part /boot/efi
├─nvme0n1p2 259:2    0    16M  0 part
├─nvme0n1p3 259:3    0 426.2G  0 part
├─nvme0n1p4 259:4    0   625M  0 part
├─nvme0n1p5 259:5    0   977M  0 part
├─nvme0n1p6 259:6    0   1.9G  0 part [SWAP]
└─nvme0n1p7 259:7    0  47.1G  0 part /
zodiac@Mateo:~$
```

در اینجا من ۴ فلش به سیستم خود وصل کرده‌ام. (سه تا با پورتUSB معمولی و یکی با USB Type C)

در اینجا می‌خواهیم ۴ تا فلش (دو تا ۱۶ گیگ و دو تا هم ۸ گیگ) را با هم raid کنیم. نام این دیسک‌ها همانطور که در لیست بالا قرار دارند sda و sdb و sdc و sdd است. برای انجام عملیات raid باید ابتدا mdadm را در سیستم نصب کنیم.

نصب adadm

اینکار را با دستور زیر انجام می‌دهیم:

```
zodiac@Mateo:~$ sudo apt install mdadm
[sudo] password for zodiac:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
mdadm is already the newest version (4.2-0ubuntu1).
The following packages were automatically installed and are no longer required:
  libflashrom1 libftdi1-2
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 49 not upgraded.
zodiac@Mateo:~$
```

البته چون من قبلا نصب کرده بودم با کسی که برای اولین بار می‌خواهد نصب کند کمی متفاوت خواهد بود.
همینطور اگر برای اولین بار پس از باز کردن ترمینال از دستور sudo استفاده کنید می‌بایست رمز یوزر خود را وارد کنید.

پس از این باید یک raid درست کنیم.

درست کردن raid

برای اینکار می‌توانیم از دستور زیر استفاده کنیم.

```
zodiac@Mateo:~$ sudo mdadm --create --verbose /dev/md0 --level=6 --raid-devices=4 /dev/sda /dev/sdb /dev/sdc /dev/sdd
mdadm: partition table exists on /dev/sda
mdadm: partition table exists on /dev/sda but will be lost or
       meaningless after creating array
mdadm: partition table exists on /dev/sdb
mdadm: partition table exists on /dev/sdb but will be lost or
       meaningless after creating array
mdadm: partition table exists on /dev/sdc
mdadm: partition table exists on /dev/sdc but will be lost or
       meaningless after creating array
mdadm: partition table exists on /dev/sdd
mdadm: partition table exists on /dev/sdd but will be lost or
       meaningless after creating array
mdadm: largest drive (/dev/sda) exceeds size (7553024K) by more than 1%
Continue creating array? y
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
zodiac@Mateo:~$
```

همانطور که مشخص است این دستور را با `sudo` زدیم چرا که نیاز به دسترسی در سطح روت دارد.

پس از عبارت `mdadm` که در واقع ابزار `raid` برای ماست از سوییچ `--create` استفاده می‌کنیم تا یک آرایه `raid` بسازیم. عبارت `/dev/md0` هم نام حافظه ایجاد شده است. پس از آن هم از سوییچ لول استفاده میکنیم. (`--level`). پس از آن می‌بایست تعداد دیوایس‌هایی که در `raid` شرکت می‌کنند را مشخص کنیم. برای این منظور از سوییچ `--raid-devices` استفاده می‌کنیم.

به این نکته توجه کنید که در `raid` سطح ۶ می‌بایست حداقل ۴ دیوایس داشته باشیم.

پس از آن هم باید آدرس حافظه‌ها را به دستور بدهیم. در واقع همانطور که بالاتر اشاره کردیم آدرس‌ها در `lsblk` مشخص است.

همینطور می‌توانیم به جای `--create` از `-C` استفاده کنیم.

به طور کلی می‌توان برای ساخت این `raid` از دستور زیر هم استفاده کنیم. این دستور دقیقا کار دستور بالا را می‌کند:

```
$ sudo mdadm -C -l6 -n4 /dev/md0 /dev/sd{a,b,c,d}
```

ساخت mount point

در مرحله بعدی لازم است که یک دایرکتوری بسازیم.

```
zodiac@Mateo:~$ sudo mkdir /home/raid
zodiac@Mateo:~$
```

با این دستور یک دایرکتوری در پوشه home خود به اسم raid می‌سازیم. این پوشه باید با حافظه raid ساخته شده بعدا mount شوند.

این دستور هم چون دستور ساخت پوشه در home است نیاز به sudo دارد.

ایجاد سیستم فایل بر روی پارتیشن

پس از این مرحله می‌بایست حافظه ایجاد شده را فرمت کنیم و یک سیستم فایل بر روی پارتیشن ایجاد شده درست کنیم. این کار با دستور را با mkfs انجام می‌دهیم.

```
zodiac@Mateo:~$ mkfs.ext4 /dev/md0
mke2fs 1.46.5 (30-Dec-2021)
Creating filesystem with 3776512 4k blocks and 944704 inodes
Filesystem UUID: f3ad40dc-ce68-4d55-8aba-11b48363cca4
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
zodiac@Mateo:~$
```

اگر به جای ext4 از ntfs استفاده کنیم سیستم فایل ایجاد شده از نوع ntfs خواهد بود.

Mount کردن mount point

در نهایت هم عملیات mount را انجام می‌دهیم.

```
zodiac@Mateo:~$ mount /dev/md0 /home/raid
zodiac@Mateo:~$
```

زمانیکه شما یک پارتیشن را ایجاد می کنید تا زمانیکه پارتیشن مورد نظر را به یک directory یا mount point در سیستم متصل یا mount نکنید قادر به استفاده کردن از آن پارتیشن نخواهید بود. اینکار با استفاده از دستوری به نام mount در سیستم عامل لینوکس استفاده می شود.

با اینکار عملیات raid تمام می‌شود. حال اگر lsblk را دوباره بزنیم می‌توانیم ببینیم که نوع حافظه‌ها از part به raid6 تغییر کرده است.

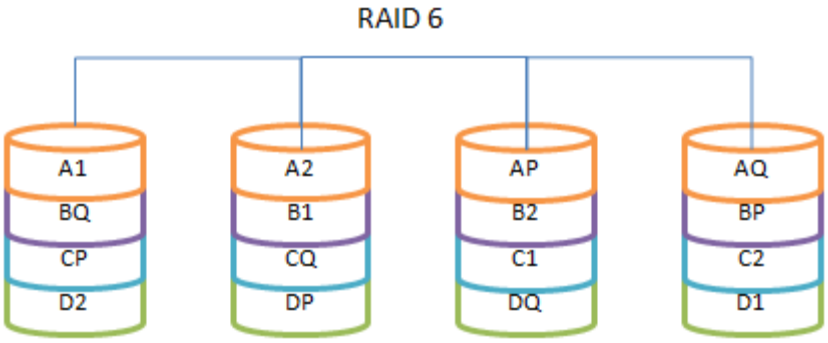
مشاهده نتیجه

دستور زیر اطلاعات دیسک‌های ما را نشان می‌دهد:

```
zodiac@Mateo:~$ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINTS
loop0         7:0    0     4K  1 loop  /snap/bare/5
loop1         7:1    0  55.6M  1 loop  /snap/core18/2667
loop2         7:2    0    62M  1 loop  /snap/core20/1587
loop3         7:3    0  63.3M  1 loop  /snap/core20/1778
loop4         7:4    0  72.9M  1 loop  /snap/core22/469
loop5         7:5    0  72.9M  1 loop  /snap/core22/484
loop6         7:6    0 163.3M  1 loop  /snap/firefox/1635
loop7         7:7    0 400.8M  1 loop  /snap/gnome-3-38-2004/112
loop8         7:8    0 346.3M  1 loop  /snap/gnome-3-38-2004/119
loop9         7:9    0  91.7M  1 loop  /snap/gtk-common-themes/1535
loop10        7:10   0  45.9M  1 loop  /snap/snap-store/582
loop11        7:11   0  49.6M  1 loop  /snap/snapd/17883
loop12        7:12   0  49.8M  1 loop  /snap/snapd/17950
loop13        7:13   0   284K  1 loop  /snap/snapd-desktop-integration/14
loop14        7:14   0   304K  1 loop  /snap/snapd-desktop-integration/49
loop15        7:15   0 375.1M  1 loop  /snap/telegram-desktop/4470
loop16        7:16   0 375.1M  1 loop  /snap/telegram-desktop/4486
loop17        7:17   0 320.4M  1 loop  /snap/vlc/3078
sda           8:0    1  14.8G  0 disk
└─md0         9:0    0  14.8G  0 raid6 /home/raid
sdb           8:16   1   7.5G  0 disk
└─md0         9:0    0   7.5G  0 raid6 /home/raid
sdc           8:32   1   7.2G  0 disk
└─md0         9:0    0   7.2G  0 raid6 /home/raid
sdd           8:48   1  14.8G  0 disk
└─md0         9:0    0  14.8G  0 raid6 /home/raid
nvme0n1     259:0    0 476.9G  0 disk
├─nvme0n1p1 259:1    0   100M  0 part  /boot/efi
├─nvme0n1p2 259:2    0    16M  0 part
├─nvme0n1p3 259:3    0 426.2G  0 part
├─nvme0n1p4 259:4    0   625M  0 part
├─nvme0n1p5 259:5    0   977M  0 part
├─nvme0n1p6 259:6    0   1.9G  0 part  [SWAP]
└─nvme0n1p7 259:7    0  47.1G  0 part  /
zodiac@Mateo:~$
```

همانطور که در تصویر می‌بینید `mount point` که بالاتر به آن اشاره شد در حافظه‌های فعال در عمل `raid` به `/home/raid` یا همان دایرکتوری که ساختیم تغییر کرده‌اند.

اکنون اگر ما دیتایی را به حافظه‌ی ایجاد شده منتقل کنیم به صورت زیر منتقل می‌شود:



همانطور که مشخص است حداقل نیاز به ۴ حافظه داریم. همینطور اگر یک دیتا انتقال داده شود با توجه به اینکه فایل تکه تکه منتقل می‌شود و دو حافظه هم تحت Parity داریم هم سرعت خواندن و نوشتن در این شیوه بیشتر از یک حافظه است و هم اطمینان بیشتری نسبت به دیتای منتقل شده از لحاظ از دست رفتن دیتا داریم.

تفاوت اصلی بین RAID 5 و RAID 6 این است که آرایه RAID 5 می تواند پس از یک disk failure به کار خود ادامه دهد، اما آرایه RAID 6 می تواند دو disk failure همزمان را حفظ کند و همچنان به کار خود ادامه دهد. آرایه های RAID 6 نیز کمتر در معرض خطا در فرآیند بازسازی دیسک هستند.

خرابی هارد دیسک یا disk failure زمانی اتفاق می‌افتد که یک درایو دیسک سخت کار نمی‌کند و نمی‌توان با رایانه‌ای که به درستی پیکربندی شده است به اطلاعات ذخیره‌شده دسترسی پیدا کرد.

خرابی هارد دیسک ممکن است در طول کارکرد معمولی یا به دلیل یک عامل خارجی مانند قرار گرفتن در معرض آتش یا آب یا میدان های مغناطیسی زیاد یا ضربه شدید یا آلودگی محیطی رخ دهد که می تواند منجر به تصادف شود.

اطلاعات ذخیره شده روی هارد دیسک نیز ممکن است در نتیجه خرابی داده ها، اختلال یا از بین رفتن رکورد اصلی راه اندازی هارد دیسک، یا بدافزاری که به طور عمدی محتویات دیسک را از بین می برد، غیرقابل دسترسی باشد.

برگردادن به حالت اول

برای اینکار ابتدا باید mount point ای که آن را mount یا متصل کرده بودیم را unmount کنیم.

این کار را با دستور unmount انجام می‌دهیم:

```
$ unmount /dev/md0
```

سپس باید mdadm را در حافظه ایجاد شده را متوقف کنیم.

```
$ mdadm --stop /dev/md0
```

با دستور زیر هم تک تک حافظه‌های استفاده شده در فرایند raid را از آن جدا می‌کنیم:

```
$ mdadm --zero-superblock /dev/sd{a,b,c,d}
```

پس از انجام سه مرحله بالا می‌بینیم که حافظه‌ها (فلش‌ها) به حالت اول برگشته‌اند. البته برای بهتر کردن این بازنشانی قسمت بازنشانی دستگاه‌های موجود در raid را ببینید.

man adadm

NAME
mdadm - manage MD devices aka Linux Software RAID

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

DESCRIPTION
RAID devices are virtual devices created from two or more real block devices. This allows multiple devices (typically disk drives or partitions thereof) to be combined into a single device to hold (for example) a single filesystem. Some RAID levels include redundancy and so can survive some degree of device failure.

Linux Software RAID devices are implemented through the md (Multiple Devices) device driver.

Currently, Linux supports LINEAR md devices, RAID0 (striping), RAID1 (mirroring), RAID4, RAID5, RAID6, RAID10, MULTIPATH, FAULTY, and CONTAINER.

MULTIPATH is not a Software RAID mechanism, but does involve multiple devices: each device is a path to one common physical storage device. New installations should not use md/multipath as it is not well supported and has no ongoing development. Use the Device Mapper based multipath-tools instead.

FAULTY is also not true RAID, and it only involves one device. It provides a layer over a true device that can be used to inject faults.

CONTAINER is different again. A CONTAINER is a collection of devices that are managed as a set. This is similar to the set of devices connected to a hardware RAID controller. The set of devices may contain a number of different RAID arrays each utilising some (or all) of the blocks from a number of the devices in the set. For example, two devices in a 5-device set might form a RAID1 using the whole devices. The remaining three might have a RAID5 over the first half of each device, and a RAID0 over the second half.

With a CONTAINER, there is one set of metadata that describes all of the arrays in the container. So when mdadm creates a CONTAINER device, the device just represents the metadata. Other normal arrays (RAID1 etc) can be created inside the container.

MODES
mdadm has several major modes of operation:

Assemble
Assemble the components of a previously created array into an active array. Components can be explicitly given or can be searched for. mdadm checks that the components do form a bona fide array, and can, on request, fiddle superblock information so as to assemble a faulty array.

Build Build an array that doesn't have per-device metadata (superblocks). For these sorts of arrays, mdadm cannot differentiate between initial creation and subsequent assembly of an array. It also cannot perform any checks that appropriate components have been requested. Because of this, the Build mode should only be used together with a complete understanding of what you are doing.

Create Create a new array with per-device metadata (superblocks). Appropriate metadata is written to each device, and then the array comprising those devices is activated. A 'resync' process is started to make sure that the array is consistent (e.g. both sides of a mirror contain the same data) but the content of the device is left otherwise untouched. The array can be used as soon as it has been created. There is no need to wait for the initial resync to finish.

Follow or Monitor
Monitor one or more md devices and act on any state changes. This is only meaningful for RAID1, 4, 5, 6, 10 or multipath arrays, as only these have interesting state. RAID0 or Linear never have missing, spare, or failed drives, so there is nothing to monitor.

Grow Grow (or shrink) an array, or otherwise reshape it in some way. Currently supported growth options including changing the active size of component devices and changing the number of active devices in Linear and RAID levels 0/1/4/5/6, changing the RAID level between 0, 1, 5, and 6, and between 0 and 10, changing the chunk size and layout for RAID 0,4,5,6,10 as well as adding or removing a write-intent bitmap and changing the array's consistency policy.

Incremental Assembly
Add a single device to an appropriate array. If the addition of the device makes the array runnable, the array will be started. This provides a convenient interface to a hot-plug system.
As each device is detected, mdadm has a chance to include it in some array as appropriate. Optionally, when the --fail flag is passed in we will remove the device from any active array instead of adding it.

If a CONTAINER is passed to mdadm in this mode, then any arrays within that container will be assembled and started.

Manage This is for doing things to specific components of an array such as adding new spares and removing faulty devices.

Misc This is an 'everything else' mode that supports operations on active arrays, operations on component devices such as erasing old superblocks, and information gathering operations.

Auto-detect
This mode does not act on a specific device or array, but rather it requests the Linux Kernel to activate any auto-detected arrays.

OPTIONS
Options for selecting a mode are:

- A, --assemble
Assemble a pre-existing array.

- B, --build
Build a legacy array without superblocks.

- C, --create
Create a new array.

- F, --follow, --monitor
Select Monitor mode.

- G, --grow
Change the size or shape of an active array.

- I, --incremental
Add/remove a single device to/from an appropriate array, and possibly start the array.

— auto-detect
Request that the kernel starts any auto-detected arrays. This can only work if md is compiled into the kernel — not if it is a module. Arrays can be auto-detected by the kernel if all the components are in primary MS-DOS partitions with partition type FD, and all use v0.90 metadata. In-kernel autodetect is not recommended for new installations. Using mdadm to detect and assemble arrays — possibly in an initrd — is substantially more flexible and should be preferred.

If a device is given before any options, or if the first option is one of --add, --re-add, --add-spare, --fail, --remove, or --replace, then the MANAGE mode is assumed. Anything other than these will cause the Misc mode to be assumed.

Options that are not mode-specific are:

- h, --help
Display general help message or, after one of the above options, a mode-specific help message.

— help-options
Display more detailed help about command line parsing and some commonly used options.

- V, --version
Print version information for mdadm.

- v, --verbose
  Be more verbose about what is happening. This can be used twice to be extra-verbose. The extra verbosity currently only affects --detail --scan and --examine --scan.

- q, --quiet
  Avoid printing purely informative messages. With this, mdadm will be silent unless there is something really important to report.

- f, --force
  Be more forceful about certain operations. See the various modes for the exact meaning of this option in different contexts.

- c, --config=
  Specify the config file or directory. Default is to use /etc/mdadm/mdadm.conf and /etc/mdadm/mdadm.conf.d, or if those are missing then /etc/mdadm.conf and /etc/mdadm.conf.d. If the config file given is partitions then nothing will be read, but mdadm will act as though the config file contained exactly
    DEVICE partitions containers
  and will read /proc/partitions to find a list of devices to scan, and /proc/mdstat to find a list of containers to examine. If the word none is given for the config file, then mdadm will act as though the config file were empty.

  If the name given is of a directory, then mdadm will collect all the files contained in the directory with a name ending in .conf, sort them lexically, and process all of those files as config files.

- s, --scan
  Scan config file or /proc/mdstat for missing information. In general, this option gives mdadm permission to get any missing information (like component devices, array devices, array identities, and alert destination) from the configuration file (see previous option); one exception is MISC mode when using --detail or --stop, in which case --scan says to get a list of array devices from /proc/mdstat.

- e, --metadata=
  Declare the style of RAID metadata (superblock) to be used. The default is 1.2 for --create, and to guess for other operations. The default can be overridden by setting the metadata value for the CREATE keyword in mdadm.conf.

  Options are:

0.90.0
  Use the original 0.90 format superblock. This format limits arrays to 28 component devices and limits component devices of levels 1 and greater to 2 terabytes. It is also possible for there to be confusion about whether the superblock applies to a whole device or just the last partition, if that partition starts on a 64K boundary.

1.2.1.1.1.0.1 default
  Use the new version-1 format superblock. This has fewer restrictions. It can easily be moved between hosts with different endian-ness, and a recovery operation can be checkpointed and restarted. The different sub-versions store the superblock at different locations on the device, either at the end (for 1.0), at the start (for 1.1) or 4K from the start (for
"1" .1.2 is equivalent to "1.2" (the commonly preferred 1.x format). "default" is equivalent to "1.2."

  ddf Use the "Industry Standard" DDF (Disk Data Format) format defined by SNIA. When creating a DDF array a CONTAINER will be created, and normal arrays can be created in that container.

  imsm Use the Intel(R) Matrix Storage Manager metadata format. This creates a CONTAINER which is managed in a similar manner to DDF, and is supported by an option-rom on some platforms:

    https://www.intel.com/content/www/us/en/support/products/122484/memory-and-storage/ssd-software/intel-virtual-raid-on-cpu-intel-vroc.html

- homehost=
  This will override any HOMEHOST setting in the config file and provides the identity of the host which should be considered the home for any arrays.

  When creating an array, the homehost will be recorded in the metadata. For version-1 superblocks, it will be prefixed to the array name. For version-0.90 superblocks, part of the SHA1 hash of the hostname will be stored in the later half of the UUID.

  When reporting information about an array, any array which is tagged for the given homehost will be reported as such.

  When using Auto-Assemble, only arrays tagged for the given homehost will be allowed to use 'local' names (i.e. not ending in '_' followed by a digit string). See below under Auto Assembly.

  The special name "any" can be used as a wild card. If an array is created with --homehost=any then the name "any" will be stored in the array and it can be assembled in the same way on any host. If an array is assembled with this option, then the homehost recorded on the array will be ignored.

- prefer=
  When mdadm needs to print the name for a device it normally finds the name in /dev which refers to the device and is shortest. When a path component is given with --prefer mdadm will prefer a longer name if it contains that component. For example --prefer=by-uuid will prefer a name in a subdirectory of /dev called by-uuid.

  This functionality is currently only provided by --detail and --monitor.

- home-cluster=
  specifies the cluster name for the md device. The md device can be assembled only on the cluster which matches the name specified. If this option is not provided, mdadm tries to detect the cluster name automatically.

For create, build, or grow:
- n, --raid-devices=
  Specify the number of active devices in the array. This, plus the number of spare devices (see below) must equal the number of component-devices (including "missing" devices) that are listed on the command line for --create. Setting a value of 1 is probably a mistake and so requires that --force be specified first. A value of 1 will then be allowed for linear, multipath, RAID0 and RAID1. It is never allowed for RAID4, RAID5 or RAID6.
  This number can only be changed using --grow for RAID1, RAID4, RAID5 and RAID6 arrays, and only on kernels which provide the necessary support.

- x, --spare-devices=
  Specify the number of spare (eXtra) devices in the initial array. Spares can also be added and removed later. The number of component devices listed on the command line must equal the number of RAID devices plus the number of spare devices.

- z, --size=
  Amount (in Kilobytes) of space to use from each drive in RAID levels 1/4/5/6. This must be a multiple of the chunk size, and must leave about 128Kb of space at the end of the drive for the RAID superblock. When specified as .max. (as it often is) the smallest drive (or partition) sets the size. In that case, a warning will follow if the drives, as a group, have sizes that differ by more than one percent.

  A suffix of 'K', 'M', 'G' or 'T' can be given to indicate Kilobytes, Megabytes, Gigabytes or Terabytes respectively.

  Sometimes a replacement drive can be a little smaller than the original drives though this should be minimised by IDEMA standards. Such a replacement drive will be rejected by md. To guard against this it can be useful to set the initial size slightly smaller than the smaller device with the aim that it will still be larger than any replacement.

  This value can be set with --grow for RAID level 1/4/5/6 though DDF arrays may not be able to support this. If the array was created with a size smaller than the currently active drives, the extra space can be accessed using --grow. The size can be given as max which means to choose the largest size that fits on all current drives.

  Before reducing the size of the array (with --grow --size=) you should make sure that space isn't needed. If the device holds a filesystem, you would need to resize the filesystem to use less space.

  After reducing the array size you should check that the data stored in the device is still available. If the device holds a filesystem, then an 'fsck' of the filesystem is a minimum requirement. If there are problems the array can be made bigger again with no loss with another --grow --size= command.

  This value cannot be used when creating a CONTAINER such as with DDF and IMSM metadata, though it perfectly valid when creating an array inside a container.

- Z, --array-size=
  This is only meaningful with --grow and its effect is not persistent: when the array is stopped and restarted the default array size will be restored.

  Setting the array-size causes the array to appear smaller to programs that access the data. This is particularly needed before reshaping an array so that it will be smaller. As the reshape is not reversible, but setting the size with --array-size is, it is required that the array size is reduced as appropriate before the number of devices in the array is reduced.

Before reducing the size of the array you should make sure that space isn't needed. If the device holds a filesystem, you would need to resize the filesystem to use less space.

After reducing the array size you should check that the data stored in the device is still available. If the device holds a filesystem, then an 'fsck' of the filesystem is a minimum requirement. If there are problems the array can be made bigger again with no loss with another --grow --array-size= command.

A suffix of 'K', 'M', 'G' or 'T' can be given to indicate Kilobytes, Megabytes, Gigabytes or Terabytes respectively. A value of max restores the apparent size of the array to be whatever the real amount of available space is.

Clustered arrays do not support this parameter yet.

- c, --chunk-
  Specify chunk size of kilobytes. The default when creating an array is 512KB. To ensure compatibility with earlier versions, the default when building an array with no persistent metadata is 64KB. This is only meaningful for RAID0, RAID4, RAID5, RAID6, and RAID10.

  RAID4, RAID5, RAID6, and RAID10 require the chunk size to be a power of 2. In any case it must be a multiple of 4KB.

  A suffix of 'K', 'M', 'G' or 'T' can be given to indicate Kilobytes, Megabytes, Gigabytes or Terabytes respectively.

— rounding-
  Specify rounding factor for a Linear array. The size of each component will be rounded down to a multiple of this size. This is a synonym for --chunk but highlights the different meaning for Linear as compared to other RAID levels. The default is 64K if a kernel earlier than 2.6.16 is in use, and is 0K (i.e. no rounding) in later kernels.

- l, --level-
  Set RAID level. When used with --create, options are: linear, raid0, 0, stripe, raid1, 1, mirror, raid4, 4, raid5, 5, raid6, 6, raid10, 10, multipath, mp, faulty, container. Obviously some of these are synonymous.

  When a CONTAINER metadata type is requested, only the container level is permitted, and it does not need to be explicitly given.

  When used with --build, only linear, stripe, raid0, 0, raid1, multipath, mp, and faulty are valid.

  Can be used with --grow to change the RAID level in some cases. See LEVEL CHANGES below.

- p, --layout-
  This option configures the fine details of data layout for RAID5, RAID6, and RAID10 arrays, and controls the failure modes for faulty. It can also be used for working around a kernel bug with RAID0, but generally doesn't need to be used explicitly.

  The layout of the RAID5 parity block can be one of left-asymmetric, left-symmetric, right-asymmetric, right-symmetric, la, ra, ls, rs. The default is left-symmetric.

  It is also possible to cause RAID5 to use a RAID4-like layout by choosing parity-first, or parity-last.

  Finally for RAID5 there are DDF-compatible layouts, ddf-zero-restart, ddf-N-restart, and ddf-N-continue.

  These same layouts are available for RAID6. There are also 4 layouts that will provide an intermediate stage for converting between RAID5 and RAID6. These provide a layout which is identical to the corresponding RAID5 layout on the first N-1 devices, and has the 'Q' syndrome (the second 'parity' block used by RAID6) on the last device. These layouts are: left-symmetric-6, right-symmetric-6, left-asymmetric-6, right-asymmetric-6, and parity-first-6.

  When setting the failure mode for level faulty, the options are: write-transient, wt, read-transient, rt, write-persistent, wp, read-persistent, rp, write-all, read-fixable, rf, clear, flush, none.

  Each failure mode can be followed by a number, which is used as a period between fault generation. Without a number, the fault is generated once on the first relevant request. With a number, the fault will be generated after that many requests, and will continue to be generated every time the period elapses.

  Multiple failure modes can be current simultaneously by using the --grow option to set subsequent failure modes.

" clear" or "none" will remove any pending or periodic failure modes, and "flush" will clear any persistent faults.

  The layout options for RAID10 are one of 'n', 'o' or 'f' followed by a small number. The default is 'n2'. The supported options are:

' n' signals 'near' copies. Multiple copies of one data block are at similar offsets in different devices.

' o' signals 'offset' copies. Rather than the chunks being duplicated within a stripe, whole stripes are duplicated but are rotated by one device so duplicate blocks are on different devices. Thus subsequent copies of a block are in the next drive, and are one chunk further down.

' f' signals 'far' copies (multiple copies have very different offsets). See md(4) for more detail about 'near', 'offset', and 'far'.

  The number is the number of copies of each datablock. 2 is normal, 3 can be useful. This number can be at most equal to the number of devices in the array. It does not need to divide evenly into that number (e.g. it is perfectly legal to have an 'n2' layout for an array with an odd number of devices).

  A bug introduced in Linux 3.14 means that RAID0 arrays with devices of differing sizes started using a different layout. This could lead to data corruption. Since Linux 5.4 (and various stable releases that received backports), the kernel will not accept such an array unless a layout is explicitly set. It can be set to 'original' or 'alternate'. When creating a new array, mdadm will select 'original' by default, so the layout does not normally need to be set. An array created for either 'original' or 'alternate' will not be recognized by an (unpatched) kernel prior to 5.4. To create a RAID0 array with devices of differing sizes that can be used on an older kernel, you can set the layout to 'dangerous'. This will use whichever layout the running kernel supports, so the data on the array may become corrupt when changing kernel from pre-3.14 to a later kernel.

  When an array is converted between RAID5 and RAID6 an intermediate RAID6 layout is used in which the second parity block (Q) is always on the last device. To convert a RAID5 to RAID6 and leave it in this new layout (which does not require re-striping) use --layout=preserve. This will try to avoid any restriping.

  The converse of this is --layout=normalise which will change a non-standard RAID6 layout into a more standard arrangement.

— parity-
  same as --layout (thus explaining the p of -p).

- b, --bitmap-
  Specify a file to store a write-intent bitmap in. The file should not exist unless --force is also given. The same file should be provided when assembling the array. If the word internal is given, then the bitmap is stored with the metadata on the array, and so is replicated on all devices. If the word none is given with --grow mode, then any bitmap that is present is removed. If the word clustered is given, the array is created for a clustered environment. One bitmap is created for each node as defined by the --nodes parameter and are stored internally.

  To help catch typing errors, the filename must contain at least one slash ('/') if it is a real file (not 'internal' or 'none').

  Note: external bitmaps are only known to work on ext2 and ext3. Storing bitmap files on other filesystems may result in serious problems.

  When creating an array on devices which are 100G or larger, mdadm automatically adds an internal bitmap as it will usually be beneficial. This can be suppressed with --bitmap=none or by selecting a different consistency policy with --consistency-policy.

— bitmap-chunk-
  Set the chunksize of the bitmap. Each bit corresponds to that many Kilobytes of storage. When using a file based bitmap, the default is to use the smallest size that is at-least 4 and requires no more than 2^21 chunks. When using an internal bitmap, the chunksize defaults to 64Meg, or larger if necessary to fit the bitmap into the available space.

  A suffix of 'K', 'M', 'G' or 'T' can be given to indicate Kilobytes, Megabytes, Gigabytes or Terabytes respectively.

- W, --write-mostly
  subsequent devices listed in a --build, --create, or --add command will be flagged as 'write-mostly'. This is valid for RAID1 only and means that the 'md' driver will avoid reading from these devices if at all possible. This can be useful if mirroring over a slow link.

— write-behind-

Specify that write-behind mode should be enabled (valid for RAID1 only). If an argument is specified, it will set the maximum number of outstanding writes allowed. The default value is 256. A write-intent bitmap is required in order to use write-behind mode, and write-behind is only attempted on drives marked as write-mostly.

— failfast
subsequent devices listed in a --create or --add command will be flagged as 'failfast'. This is valid for RAID1 and RAID10 only. IO requests to these devices will be encouraged to fail quickly rather than cause long delays due to error handling. Also no attempt is made to repair a read error on these devices.

If an array becomes degraded so that the 'failfast' device is the only usable device, the 'failfast' flag will then be ignored and extended delays will be preferred to complete failure.

The 'failfast' flag is appropriate for storage arrays which have a low probability of true failure, but which may sometimes cause unacceptable delays due to internal maintenance functions.

— assume-clean
Tell mdadm that the array pre-existed and is known to be clean. It can be useful when trying to recover from a major failure as you can be sure that no data will be affected unless you actually write to the array. It can also be used when creating a RAID1 or RAID10 if you want to avoid the initial resync, however this practice — while normally safe — is not recommended. Use this only if you really know what you are doing.

When the devices that will be part of a new array were filled with zeros before creation the operator knows the array is actually clean. If that is the case, such as after running badblocks, this argument can be used to tell mdadm the facts the operator knows.

When an array is resized to a larger size with --grow --size= the new space is normally resynced in that same way that the whole array is resynced at creation. From Linux version 3.0, --assume-clean can be used with that command to avoid the automatic resync.

— backup-file=
This is needed when --grow is used to increase the number of raid-devices in a RAID5 or RAID6 if there are no spare devices available, or to shrink, change RAID level or layout. See the GROW MODE section below on RAID-DEVICES CHANGES. The file must be stored on a separate device, not on the RAID array being reshaped.

— data-offset=
Arrays with 1.x metadata can leave a gap between the start of the device and the start of array data. This gap can be used for various metadata. The start of data is known as the data-offset. Normally an appropriate data offset is computed automatically. However it can be useful to set it explicitly such as when re-creating an array which was originally created using a different version of mdadm which computed a different offset.

Setting the offset explicitly over-rides the default. The value given is in Kilobytes unless a suffix of 'K', 'M', 'G' or 'T' is used to explicitly indicate Kilobytes, Megabytes, Gigabytes or Terabytes respectively.

Since Linux 3.4, --data-offset can also be used with --grow for some RAID levels (initially on RAID10). This allows the data-offset to be changed as part of the reshape process. When the data offset is changed, no backup file is required as the difference in offsets is used to provide the same functionality.

When the new offset is earlier than the old offset, the number of devices in the array cannot shrink. When it is after the old offset, the number of devices in the array cannot increase.

When creating an array, --data-offset can be specified as variable. In the case each member device is expected to have a offset appended to the name, separated by a colon. This makes it possible to recreate exactly an array which has varying data offsets (as can happen when different versions of mdadm are used to add different devices).

— continue
This option is complementary to the --freeze-reshape option for assembly. It is needed when --grow operation is interrupted and it is not restarted automatically due to --freeze-reshape usage during array assembly. This option is used together with -G , (--grow) command and device for a pending reshape to be continued. All parameters required for reshape continuation will be read from array metadata. If initial --grow command had required --backup-file= option to be set, continuation option will require to have exactly the same backup file given as well.

Any other parameter passed together with --continue option will be ignored.

– N, --name=
Set a name for the array. This is currently only effective when creating an array with a version-1 superblock, or an array in a DDF container. The name is a simple textual string that can be used to identify array components when assembling. If name is needed but not specified, it is taken from the basename of the device that is being created. e.g. when creating /dev/md/home the name will default to home. (Does not work in Grow mode.)

– R, --run
Insist that mdadm run the array, even if some of the components appear to be active in another array or filesystem. Normally mdadm will ask for confirmation before including such components in an array. This option causes that question to be suppressed.

– f, --force
Insist that mdadm accept the geometry and layout specified without question. Normally mdadm will not allow creation of an array with only one device, and will try to create a RAID5 array with one missing drive (as this makes the initial resync work faster). With --force, mdadm will not try to be so clever.

– o, --readonly
Start the array read only rather than read-write as normal. No writes will be allowed to the array, and no resync, recovery, or reshape will be started. It works with Create, Assemble, Manage and Misc mode.

– a, --auto{=yes,md,mdp,part,p}{NN}
Instruct mdadm how to create the device file if needed, possibly allocating an unused minor number. "md" causes a non-partitionable array to be used (though since Linux 2.6.28, these array devices are in fact partitionable). "mdp", "part" or "p" causes a partitionable array (2.6 and later) to be used. "yes" requires the named md device to have a 'standard' format, and the type and minor number will be determined from this. With mdadm 3.0, device creation is normally left up to udev so this option is unlikely to be needed. See DEVICE NAMES below.

The argument can also come immediately after "-a". e.g. "-ap."

If --auto is not given on the command line or in the config file, then the default will be --auto=yes.

If --scan is also given, then any auto= entries in the config file will override the --auto instruction given on the command line.

For partitionable arrays, mdadm will create the device file for the whole array and for the first 4 partitions. A different number of partitions can be specified at the end of this option (e.g. --auto=p7). If the device name ends with a digit, the partition names add a 'p', and a number, e.g. /dev/md/home1p3. If there is no trailing digit, then the partition names just have a number added, e.g. /dev/md/scratch3.

If the md device name is in a 'standard' format as described in DEVICE NAMES, then it will be created, if necessary, with the appropriate device number based on that name. If the device name is not in one of these formats, then a unused device number will be allocated. The device number will be considered unused if there is no active array for that number, and there is no entry in /dev for that number and with a non-standard name. Names that are not in 'standard' format are only allowed in "/dev/md."/

This is meaningful with --create or --build.

– a, --add
This option can be used in Grow mode in two cases.

If the target array is a Linear array, then --add can be used to add one or more devices to the array. They are simply catenated on to the end of the array. Once added, the devices cannot be removed.

If the --raid-disks option is being used to increase the number of devices in an array, then --add can be used to add some extra devices to be included in the array. In most cases this is not needed as the extra devices can be added as spares first, and then the number of raid-disks can be changed. However for RAID0, it is not possible to add spares. So to increase the number of devices in a RAID0, it is necessary to set the new number of devices, and to add the new devices, in the same command.

— nodes
Only works when the array is for clustered environment. It specifies the maximum number of nodes in the cluster that will use this device simultaneously. If not specified, this defaults to 4.

— write-journal
Specify journal device for the RAID-4/5/6 array. The journal device should be a SSD with reasonable lifetime.

— symlinks
Auto creation of symlinks in /dev to /dev/md, option --symlinks must be 'no' or 'yes' and work with --create and --build.

- k, --consistency-policy=

  Specify how the array maintains consistency in case of unexpected shutdown. Only relevant for RAID levels with redundancy. Currently supported options are:

  resync Full resync is performed and all redundancy is regenerated when the array is started after unclean shutdown.

  bitmap Resync assisted by a write-intent bitmap. Implicitly selected when using --bitmap.

  journal
  For RAID levels 4/5/6, journal device is used to log transactions and replay after unclean shutdown. Implicitly selected when using --write-journal.

  ppl For RAID5 only, Partial Parity Log is used to close the write hole and eliminate resync. PPL is stored in the metadata region of RAID member drives, no additional journal drive is needed.

  Can be used with --grow to change the consistency policy of an active array in some cases. See CONSISTENCY POLICY CHANGES below.

For assemble:

- u, --uuid=

  uuid of array to assemble. Devices which don't have this uuid are excluded

- m, --super-minor=

  Minor number of device that array was created for. Devices which don't have this minor number are excluded. If you create an array as /dev/md1, then all superblocks will contain the minor number 1, even if the array is later assembled as /dev/md2.

  Giving the literal word "dev" for --super-minor will cause mdadm to use the minor number of the md device that is being assembled. e.g. when assembling /dev/md0, --super-minor=dev will look for super blocks with a minor number of 0.

-- super-minor is only relevant for v0.90 metadata, and should not normally be used. Using --uuid is much safer.

- N, --name=

  Specify the name of the array to assemble. This must be the name that was specified when creating the array. It must either match the name stored in the superblock exactly, or it must match with the current homehost prefixed to the start of the given name.

- f, --force

  Assemble the array even if the metadata on some devices appears to be out-of-date. If mdadm cannot find enough working devices to start the array, but can find some devices that are recorded as having failed, then it will mark those devices as working so that the array can be started. This works only for native. For external metadata it allows to start dirty degraded RAID 4, 5, 6.
  An array which requires --force to be started may contain data corruption. Use it carefully.

- R, --run

  Attempt to start the array even if fewer drives were given than were present last time the array was active. Normally if not all the expected drives are found and --scan is not used, then the array will be assembled but not started. With --run an attempt will be made to start it anyway.

-- no-degraded

  This is the reverse of --run in that it inhibits the startup of array unless all expected drives are present. This is only needed with --scan, and can be used if the physical connections to devices are not as reliable as you would like.

- a, --auto{=no,yes,md,mdp,part}

  See this option under Create and Build options.

- b, --bitmap=

  Specify the bitmap file that was given when the array was created. If an array has an internal bitmap, there is no need to specify this when assembling the array.

-- backup-file=

  If --backup-file was used while reshaping an array (e.g. changing number of devices or chunk size) and the system crashed during the critical section, then the same --backup-file must be presented to --assemble to allow possibly corrupted data to be restored, and the reshape to be completed.

-- invalid-backup

  If the file needed for the above option is not available for any reason an empty file can be given together with this option to indicate that the backup file is invalid. In this case the data that was being rearranged at the time of the crash could be irrecoverably lost, but the rest of the array may still be recoverable. This option should only be used as a last resort if there is no way to recover the backup file.

- U, --update=

  Update the superblock on each device while assembling the array. The argument given to this flag can be one of sparc2.2, summaries, uuid, name, nodes, homehost, home-cluster, resync, byteorder, devicesize, no-bitmap, bbl, no-bbl, ppl, no-ppl, layout-original, layout-alternate, layout-unspecified, metadata, or super-minor.

  The sparc2.2 option will adjust the superblock of an array what was created on a Sparc machine running a patched 2.2 Linux kernel. This kernel got the alignment of part of the superblock wrong. You can use the --examine --sparc2.2 option to mdadm to see what effect this would have.

  The super-minor option will update the preferred minor field on each superblock to match the minor number of the array being assembled. This can be useful if --examine reports a different "Preferred Minor" to --detail. In some cases this update will be performed automatically by the kernel driver. In particular the update happens automatically at the first write to an array with redundancy (RAID level 1 or greater) on a 2.6 (or later) kernel.

  The uuid option will change the uuid of the array. If a UUID is given with the --uuid option that UUID will be used as a new UUID and will NOT be used to help identify the devices in the array. If no --uuid is given, a random UUID is chosen.

  The name option will change the name of the array as stored in the superblock. This is only supported for version-1 superblocks.

  The nodes option will change the nodes of the array as stored in the bitmap superblock. This option only works for a clustered environment.

  The homehost option will change the homehost as recorded in the superblock. For version-0 superblocks, this is the same as updating the UUID. For version-1 superblocks, this involves updating the name.

  The home-cluster option will change the cluster name as recorded in the superblock and bitmap. This option only works for clustered environment.

  The resync option will cause the array to be marked dirty meaning that any redundancy in the array (e.g. parity for RAID5, copies for RAID1) may be incorrect. This will cause the RAID system to perform a "resync" pass to make sure that all redundant information is correct.

  The byteorder option allows arrays to be moved between machines with different byte-order, such as from a big-endian machine like a Sparc or some MIPS machines, to a little-endian x86_64 machine. When assembling such an array for the first time after a move, giving --update=byteorder will cause mdadm to expect superblocks to have their byteorder reversed, and will correct that order before assembling the array. This is only valid with original (Version 0.90) superblocks.

  The summaries option will correct the summaries in the superblock. That is the counts of total, working, active, failed, and spare devices.

  The devicesize option will rarely be of use. It applies to version 1.1 and 1.2 metadata only (where the metadata is at the start of the device) and is only useful when the component device has changed size (typically become larger). The version 1 metadata records the amount of the device that can be used to store data, so if a device in a version 1.1 or 1.2 array becomes larger, the metadata will still be visible, but the extra space will not. In this case it might be useful to assemble the array with --update=devicesize. This will cause mdadm to determine the maximum usable amount of space on each device and update the relevant field in the metadata.

  The metadata option only works on v0.90 metadata arrays and will convert them to v1.0 metadata. The array must not be dirty (i.e. it must not need a sync) and it must not have a write-intent bitmap.

  The old metadata will remain on the devices, but will appear older than the new metadata and so will usually be ignored. The old metadata (or indeed the new metadata) can be removed by giving the appropriate --metadata= option to --zero-superblock.

  The no-bitmap option can be used when an array has an internal bitmap which is corrupt in some way so that assembling the array normally fails. It will cause any internal bitmap to be ignored.

The bbl option will reserve space in each device for a bad block list. This will be 4K in size and positioned near the end of any free space between the superblock and the data.

The no-bbl option will cause any reservation of space for a bad block list to be removed. If the bad block list contains entries, this will fail, as removing the list could cause data corruption.

The ppl option will enable PPL for a RAID5 array and reserve space for PPL on each device. There must be enough free space between the data and superblock and a write-intent bitmap or journal must not be used.

The no-ppl option will disable PPL in the superblock.

The layout-original and layout-alternate options are for RAID0 arrays with non-uniform devices size that were in use before Linux 5.4. If the array was being used with Linux 3.13 or earlier, then to assemble the array on a new kernel, --update=layout-original must be given. If the array was created and used with a kernel from Linux 3.14 to Linux 5.3, then --update=layout-alternate must be given. This only needs to be given once. Subsequent assembly of the array will happen normally. For more information, see md(4)

The layout-unspecified option reverts the effect of layout-orignal or layout-alternate and allows the array to be again used on a kernel prior to Linux 5.3. This option should be used with great caution.

— freeze-reshape
Option is intended to be used in start-up scripts during initrd boot phase. When array under reshape is assembled during initrd phase, this option stops reshape after reshape critical section is being restored. This happens before file system pivot operation and avoids loss of file system context. Losing file system context would cause reshape to be broken.

Reshape can be continued later using the --continue option for the grow command.

— symlinks
See this option under Create and Build options.

For Manage mode:

- t, --test
Unless a more serious error occurred, mdadm will exit with a status of 2 if no changes were made to the array and 0 if at least one change was made. This can be useful when an indirect specifier such as missing, detached or faulty is used in requesting an operation on the array. --test will report failure if these specifiers didn't find any match.

- a, --add
hot-add listed devices. If a device appears to have recently been part of the array (possibly it failed or was removed) the device is re-added as described in the next point. If that fails or the device was never part of the array, the device is added as a hot-spare. If the array is degraded, it will immediately start to rebuild data onto that spare.

Note that this and the following options are only meaningful on array with redundancy. They don't apply to RAID0 or Linear.

— re-add
re-add a device that was previously removed from an array. If the metadata on the device reports that it is a member of the array, and the slot that it used is still vacant, then the device will be added back to the array in the same position. This will normally cause the data for that device to be recovered. However based on the event count on the device, the recovery may only require sections that are flagged by a write-intent bitmap to be recovered or may not require any recovery at all.

When used on an array that has no metadata (i.e. it was built with --build) it will be assumed that bitmap-based recovery is enough to make the device fully consistent with the array.

When used with v1.x metadata, --re-add can be accompanied by --update=devicesize, --update=bbl, or --update=no-bbl. See the description of these option when used in Assemble mode for an explanation of their use.

If the device name given is missing then mdadm will try to find any device that looks like it should be part of the array but isn't and will try to re-add all such devices.

If the device name given is faulty then mdadm will find all devices in the array that are marked faulty, remove them and attempt to immediately re-add them. This can be useful if you are certain that the reason for failure has been resolved.

— add-spare
Add a device as a spare. This is similar to --add except that it does not attempt --re-add first. The device will be added as a spare even if it looks like it could be an recent member of the array.

- r, --remove
remove listed devices. They must not be active. i.e. they should be failed or spare devices.

As well as the name of a device file (e.g. /dev/sda1) the words failed, detached and names like set-A can be given to --remove. The first causes all failed device to be removed. The second causes any device which is no longer connected to the system (i.e an 'open' returns ENXIO) to be removed. The third will remove a set as describe below under --fail.

- f, --fail
Mark listed devices as faulty. As well as the name of a device file, the word detached or a set name like set-A can be given. The former will cause any device that has been detached from the system to be marked as failed. It can then be removed.

For RAID10 arrays where the number of copies evenly divides the number of devices, the devices can be conceptually divided into sets where each set contains a single complete copy of the data on the array. Sometimes a RAID10 array will be configured so that these sets are on separate controllers. In this case all the devices in one set can be failed by giving a name like set-A or set-B to --fail. The appropriate set names are reported by --detail.

— set-faulty
same as --fail.

— replace
Mark listed devices as requiring replacement. As soon as a spare is available, it will be rebuilt and will replace the marked device. This is similar to marking a device as faulty, but the device remains in service during the recovery process to increase resilience against multiple failures. When the replacement process finishes, the replaced device will be marked as faulty.

— with This can follow a list of --replace devices. The devices listed after --with will be preferentially used to replace the devices listed after --replace. These device must already be spare devices in the array.

— write-mostly
Subsequent devices that are added or re-added will have the 'write-mostly' flag set. This is only valid for RAID1 and means that the 'md' driver will avoid reading from these devices if possible.

— readwrite
Subsequent devices that are added or re-added will have the 'write-mostly' flag cleared.

— cluster-confirm
Confirm the existence of the device. This is issued in response to an --add request by a node in a cluster. When a node adds a device it sends a message to all nodes in the cluster to look for a device with a UUID. This translates to a udev notification with the UUID of the device to be added and the slot number. The receiving node must acknowledge this message with --cluster-confirm. Valid arguments are <slot>:<devicename> in case the device is found or <slot>:missing in case the device is not found.

— add-journal
Add journal to an existing array, or recreate journal for RAID-4/5/6 array that lost a journal device. To avoid interrupting on-going write opertions, --add-journal only works for array in Read-Only state.

— failfast
Subsequent devices that are added or re-added will have the 'failfast' flag set. This is only valid for RAID1 and RAID10 and means that the 'md' driver will avoid long timeouts on error handling where possible.

— nofailfast
Subsequent devices that are re-added will be re-added without the 'failfast' flag set.

Each of these options requires that the first device listed is the array to be acted upon, and the remainder are component devices to be added, removed, marked as faulty, etc. Several different operations can be specified for different devices, e.g.

```
mdadm /dev/md0 --add /dev/sda1 --fail /dev/sdb1 --remove /dev/sdb1
```

Each operation applies to all devices listed until the next operation.

If an array is using a write-intent bitmap, then devices which have been removed can be re-added in a way that avoids a full reconstruction but instead just updates the blocks that have changed since the device was removed. For arrays with persistent metadata (superblocks) this is done automatically. For arrays created with --build mdadm needs to be told that this device we removed recently with --re-add.

Devices can only be removed from an array if they are not in active use, i.e. that must be spares or failed devices. To remove an active device, it must first be marked as faulty.

For Misc mode:

- Q, --query
  Examine a device to see (1) if it is an md device and (2) if it is a component of an md array. Information about what is discovered is presented.

- D, --detail
  Print details of one or more md devices.

— detail-platform
  Print details of the platform's RAID capabilities (firmware / hardware topology) for a given metadata format. If used without argument, mdadm will scan all controllers looking for their capabilities. Otherwise, mdadm will only look at the controller specified by the argument in form of an absolute filepath or a link, e.g. /sys/devices/pci0000:00/0000:00:1f.2.

- Y, --export
  When used with --detail, --detail-platform, --examine, or --incremental output will be formatted as key=value pairs for easy import into the environment.

  With --incremental The value MD_STARTED indicates whether an array was started (yes) or not, which may include a reason (unsafe, nothing, no). Also the value MD_FOREIGN indicates if the array is expected on this host (no), or seems to be from elsewhere (yes).

- E, --examine
  Print contents of the metadata stored on the named device(s). Note the contrast between --examine and --detail. --examine applies to devices which are components of an array, while --detail applies to a whole array which is currently active.

— sparc2.2
  If an array was created on a SPARC machine with a 2.2 Linux kernel patched with RAID support, the superblock will have been created incorrectly, or at least incompatibly with 2.4 and later kernels. Using the --sparc2.2 flag with --examine will fix the superblock before displaying it. If this appears to do the right thing, then the array can be successfully assembled using
— assemble --update=sparc2.2.

- X, --examine-bitmap
  Report information about a bitmap file. The argument is either an external bitmap file or an array component in case of an internal bitmap. Note that running this on an array device (e.g.
/ dev/md0) does not report the bitmap for that array.

— examine-badblocks
  List the bad-blocks recorded for the device, if a bad-blocks list has been configured. Currently only 1.x and IMSM metadata support bad-blocks lists.

— dump=directory

— restore=directory
  Save metadata from lists devices, or restore metadata to listed devices.

- R, --run
  start a partially assembled array. If --assemble did not find enough devices to fully start the array, it might leaving it partially assembled. If you wish, you can then use --run to start the array in degraded mode.

- S, --stop
  deactivate array, releasing all resources.

- o, --readonly
  mark array as readonly.

- w, --readwrite
  mark array as readwrite.

— zero-superblock
  If the device contains a valid md superblock, the block is overwritten with zeros. With --force the block where the superblock would be is overwritten even if it doesn't appear to be valid.

  Note: Be careful to call --zero-superblock with clustered raid, make sure array isn't used or assembled in other cluster node before execute it.

— kill-subarray=
  If the device is a container and the argument to --kill-subarray specifies an inactive subarray in the container, then the subarray is deleted. Deleting all subarrays will leave an 'empty-container' or spare superblock on the drives. See --zero-superblock for completely removing a superblock. Note that some formats depend on the subarray index for generating a UUID, this command will fail if it would change the UUID of an active subarray.

— update-subarray=
  If the device is a container and the argument to --update-subarray specifies a subarray in the container, then attempt to update the given superblock field in the subarray. See below in MISC MODE for details.

- t, --test
  When used with --detail, the exit status of mdadm is set to reflect the status of the device. See below in MISC MODE for details.

- W, --wait
  For each md device given, wait for any resync, recovery, or reshape activity to finish before returning. mdadm will return with success if it actually waited for every device listed, otherwise it will return failure.

— wait-clean
  For each md device given, or each device in /proc/mdstat if --scan is given, arrange for the array to be marked clean as soon as possible. mdadm will return with success if the array uses external metadata and we successfully waited. For native arrays this returns immediately as the kernel handles dirty-clean transitions at shutdown. No action is taken if safe-mode handling is disabled.

— action=
  Set the "sync_action" for all md devices given to one of idle, frozen, check, repair. Setting to idle will abort any currently running action though some actions will automatically restart. Setting to frozen will abort any current action and ensure no other action starts automatically.

  Details of check and repair can be found it md(4) under SCRUBBING AND MISMATCHES.

For Incremental Assembly mode:

— rebuild-map, -r
  Rebuild the map file (/run/mdadm/map) that mdadm uses to help track which arrays are currently being assembled.

— run, -R
  Run any array assembled as soon as a minimal number of devices are available, rather than waiting until all expected devices are present.

— scan, -s
  Only meaningful with -R this will scan the map file for arrays that are being incrementally assembled and will try to start any that are not already started. If any such array is listed in mdadm.conf as requiring an external bitmap, that bitmap will be attached first.

— fail, -f
  This allows the hot-plug system to remove devices that have fully disappeared from the kernel. It will first fail and then remove the device from any array it belongs to. The device name

given should be a kernel device name such as "sda", not a name in /dev.

--path=
Only used with --fail. The 'path' given will be recorded so that if a new device appears at the same location it can be automatically added to the same array. This allows the failed device to be automatically replaced by a new device without metadata if it appears at specified path. This option is normally only set by a udev script.

For Monitor mode:

-m, --mail
Give a mail address to send alerts to.

-p, --program, --alert
Give a program to be run whenever an event is detected.

-y, --syslog
Cause all events to be reported through 'syslog'. The messages have facility of 'daemon' and varying priorities.

-d, --delay
Give a delay in seconds. mdadm polls the md arrays and then waits this many seconds before polling again. The default is 60 seconds. Since 2.6.16, there is no need to reduce this as the kernel alerts mdadm immediately when there is any change.

-r, --increment
Give a percentage increment. mdadm will generate RebuildNN events with the given percentage increment.

-f, --daemonise
Tell mdadm to run as a background daemon if it decides to monitor anything. This causes it to fork and run in the child, and to disconnect from the terminal. The process id of the child is written to stdout. This is useful with --scan which will only continue monitoring if a mail address or alert program is found in the config file.

-i, --pid-file
When mdadm is running in daemon mode, write the pid of the daemon process to the specified file, instead of printing it on standard output.

-1, --oneshot
Check arrays only once. This will generate NewArray events and more significantly DegradedArray and SparesMissing events. Running
mdadm --monitor --scan -1
from a cron script will ensure regular notification of any degraded arrays.

-t, --test
Generate a TestMessage alert for every array found at startup. This alert gets mailed and passed to the alert program. This can be used for testing that alert message do get through successfully.

--no-sharing
This inhibits the functionality for moving spares between arrays. Only one monitoring process started with --scan but without this flag is allowed, otherwise the two could interfere with each other.

ASSEMBLE MODE
Usage: mdadm --assemble md-device options-and-component-devices…

Usage: mdadm --assemble --scan md-devices-and-options…

Usage: mdadm --assemble --scan options…

This usage assembles one or more RAID arrays from pre-existing components. For each array, mdadm needs to know the md device, the identity of the array, and a number of component-devices. These can be found in a number of ways.

In the first usage example (without the --scan) the first device given is the md device. In the second usage example, all devices listed are treated as md devices and assembly is attempted. In the third (where no devices are listed) all md devices that are listed in the configuration file are assembled. If no arrays are described by the configuration file, then any arrays that can be found on unused devices will be assembled.

If precisely one device is listed, but --scan is not given, then mdadm acts as though --scan was given and identity information is extracted from the configuration file.

The identity can be given with the --uuid option, the --name option, or the --super-minor option, will be taken from the md-device record in the config file, or will be taken from the super block of the first component-device listed on the command line.

Devices can be given on the --assemble command line or in the config file. Only devices which have an md superblock which contains the right identity will be considered for any array.

The config file is only used if explicitly named with --config or requested with (a possibly implicit) --scan. In the later case, /etc/mdadm/mdadm.conf or /etc/mdadm.conf is used.

If --scan is not given, then the config file will only be used to find the identity of md arrays.

Normally the array will be started after it is assembled. However if --scan is not given and not all expected drives were listed, then the array is not started (to guard against usage errors). To insist that the array be started in this case (as may work for RAID1, 4, 5, 6, or 10), give the --run flag.

If udev is active, mdadm does not create any entries in /dev but leaves that to udev. It does record information in /run/mdadm/map which will allow udev to choose the correct name.

If mdadm detects that udev is not configured, it will create the devices in /dev itself.

In Linux kernels prior to version 2.6.28 there were two distinctly different types of md devices that could be created: one that could be partitioned using standard partitioning tools and one that could not. Since 2.6.28 that distinction is no longer relevant as both type of devices can be partitioned. mdadm will normally create the type that originally could not be partitioned as it has a well defined major number (9).

Prior to 2.6.28, it is important that mdadm chooses the correct type of array device to use. This can be controlled with the --auto option. In particular, a value of "mdp" or "part" or "p" tells mdadm to use a partitionable device rather than the default.

In the no-udev case, the value given to --auto can be suffixed by a number. This tells mdadm to create that number of partition devices rather than the default of 4.

The value given to --auto can also be given in the configuration file as a word starting auto= on the ARRAY line for the relevant array.

Auto Assembly
When --assemble is used with --scan and no devices are listed, mdadm will first attempt to assemble all the arrays listed in the config file.

If no arrays are listed in the config (other than those marked <ignore>) it will look through the available devices for possible arrays and will try to assemble anything that it finds. Arrays which are tagged as belonging to the given homehost will be assembled and started normally. Arrays which do not obviously belong to this host are given names that are expected not to conflict with anything local, and are started "read-auto" so that nothing is written to any device until the array is written to. i.e. automatic resync etc is delayed.

If mdadm finds a consistent set of devices that look like they should comprise an array, and if the superblock is tagged as belonging to the given home host, it will automatically choose a device name and try to assemble the array. If the array uses version-0.90 metadata, then the minor number as recorded in the superblock is used to create a name in /dev/md/ so for example /dev/md/3. If the array uses version-1 metadata, then the name from the superblock is used to similarly create a name in /dev/md/ (the name will have any 'host' prefix stripped first).

This behaviour can be modified by the AUTO line in the mdadm.conf configuration file. This line can indicate that specific metadata type should, or should not, be automatically assembled. If an array is found which is not listed in mdadm.conf and has a metadata format that is denied by the AUTO line, then it will not be assembled. The AUTO line can also request that all arrays identified as being for this homehost should be assembled regardless of their metadata type. See mdadm.conf(5) for further details.

Note: Auto assembly cannot be used for assembling and activating some arrays which are undergoing reshape. In particular as the backup-file cannot be given, any reshape which requires a backup-file to continue cannot be started by auto assembly. An array which is growing to more devices and has passed the critical section can be assembled using auto-assembly.

BUILD MODE
Usage: mdadm --build md-device --chunk=X --level=Y --raid-devices=Z devices

This usage is similar to --create. The difference is that it creates an array without a superblock. With these arrays there is no difference between initially creating the array and subsequently assembling the array, except that hopefully there is useful data there in the second case.

The level may raid0, linear, raid1, raid10, multipath, or faulty, or one of their synonyms. All devices must be listed and the array will be started once complete. It will often be appropriate to use --assume-clean with levels raid1 or raid10.

CREATE MODE

Usage: mdadm --create md-device --chunk=X --level=Y
       raid-devices=Z devices

This usage will initialise a new md array, associate some devices with it, and activate the array.

The named device will normally not exist when mdadm --create is run, but will be created by udev once the array becomes active.

The max length md-device name is limited to 32 characters. Different metadata types have more strict limitation (like IMSM where only 16 characters are allowed). For that reason, long name could be truncated or rejected, it depends on metadata policy.

As devices are added, they are checked to see if they contain RAID superblocks or filesystems. They are also checked to see if the variance in device size exceeds 1%.

If any discrepancy is found, the array will not automatically be run, though the presence of a --run can override this caution.

To create a "degraded" array in which some devices are missing, simply give the word "missing" in place of a device name. This will cause mdadm to leave the corresponding slot in the array empty. For a RAID4 or RAID5 array at most one slot can be "missing"; for a RAID6 array at most two slots. For a RAID1 array, only one real device needs to be given. All of the others can be "missing."

When creating a RAID5 array, mdadm will automatically create a degraded array with an extra spare drive. This is because building the spare into a degraded array is in general faster than resyncing the parity on a non-degraded, but not clean, array. This feature can be overridden with the --force option.

When creating an array with version-1 metadata a name for the array is required. If this is not given with the --name option, mdadm will choose a name based on the last component of the name of the device being created. So if /dev/md3 is being created, then the name 3 will be chosen. If /dev/md/home is being created, then the name home will be used.

When creating a partition based array, using mdadm with version-1.x metadata, the partition type should be set to 0xDA (non fs-data). This type selection allows for greater precision since using any other [RAID auto-detect (0xFD) or a GNU/Linux partition (0x83)], might create problems in the event of array recovery through a live cdrom.

A new array will normally get a randomly assigned 128bit UUID which is very likely to be unique. If you have a specific need, you can choose a UUID for the array by giving the --uuid= option. Be warned that creating two arrays with the same UUID is a recipe for disaster. Also, using --uuid= when creating a v0.90 array will silently override any --homehost= setting.

If the array type supports a write-intent bitmap, and if the devices in the array exceed 100G is size, an internal write-intent bitmap will automatically be added unless some other option is explicitly requested with the --bitmap option or a different consistency policy is selected with the --consistency-policy option. In any case space for a bitmap will be reserved so that one can be added later with --grow --bitmap=internal.

If the metadata type supports it (currently only 1.x and IMSM metadata), space will be allocated to store a bad block list. This allows a modest number of bad blocks to be recorded, allowing the drive to remain in service while only partially functional.

When creating an array within a CONTAINER mdadm can be given either the list of devices to use, or simply the name of the container. The former case gives control over which devices in the container will be used for the array. The latter case allows mdadm to automatically choose which devices to use based on how much spare space is available.

The General Management options that are valid with --create are:

- run insist on running the array even if some devices look like they might be in use.

- readonly
  start the array in readonly mode.

MANAGE MODE

Usage: mdadm device options... devices…

This usage will allow individual devices in an array to be failed, removed or added. It is possible to perform multiple operations with on command. For example:

```
 mdadm /dev/md0 -f /dev/hda1 -r /dev/hda1 -a /dev/hda1
```

will firstly mark /dev/hda1 as faulty in /dev/md0 and will then remove it from the array and finally add it back in as a spare. However only one md array can be affected by a single command.

When a device is added to an active array, mdadm checks to see if it has metadata on it which suggests that it was recently a member of the array. If it does, it tries to "re-add" the device. If there have been no changes since the device was removed, or if the array has a write-intent bitmap which has recorded whatever changes there were, then the device will immediately become a full member of the array and those differences recorded in the bitmap will be resolved.

MISC MODE

Usage: mdadm options ... devices…

MISC mode includes a number of distinct operations that operate on distinct devices. The operations are:

- query
  The device is examined to see if it is (1) an active md array, or (2) a component of an md array. The information discovered is reported.

- detail
  The device should be an active md device. mdadm will display a detailed description of the array. --brief or --scan will cause the output to be less detailed and the format to be suitable for inclusion in mdadm.conf. The exit status of mdadm will normally be 0 unless mdadm failed to get useful information about the device(s); however, if the --test option is given, then the exit status will be:

  0 The array is functioning normally.

  1 The array has at least one failed device.

  2 The array has multiple failed devices such that it is unusable.

  4 There was an error while trying to get information about the device.

- detail-platform
  Print detail of the platform's RAID capabilities (firmware / hardware topology). If the metadata is specified with -e or --metadata= then the return status will be:

  0 metadata successfully enumerated its platform components on this system

  1 metadata is platform independent

  2 metadata failed to find its platform components on this system

- update-subarray=
  If the device is a container and the argument to --update-subarray specifies a subarray in the container, then attempt to update the given superblock field in the subarray. Similar to updating an array in "assemble" mode, the field to update is selected by -U or --update= option. The supported options are name, ppl, no-ppl, bitmap and no-bitmap.

  The name option updates the subarray name in the metadata, it may not affect the device node name or the device node symlink until the subarray is re-assembled. If updating name would change the UUID of an active subarray this operation is blocked, and the command will end in an error.

  The ppl and no-ppl options enable and disable PPL in the metadata. Currently supported only for IMSM subarrays.

  The bitmap and no-bitmap options enable and disable write-intent bitmap in the metadata. Currently supported only for IMSM subarrays.

– examine
The device should be a component of an md array. mdadm will read the md superblock of the device and display the contents. If --brief or --scan is given, then multiple devices that are components of the one array are grouped together and reported in a single entry suitable for inclusion in mdadm.conf.

Having --scan without listing any devices will cause all devices listed in the config file to be examined.

– dump=directory
If the device contains RAID metadata, a file will be created in the directory and the metadata will be written to it. The file will be the same size as the device and have the metadata written in the file at the same locate that it exists in the device. However the file will be "sparse" so that only those blocks containing metadata will be allocated. The total space used will be small.

The file name used in the directory will be the base name of the device. Further if any links appear in /dev/disk/by-id which point to the device, then hard links to the file will be created in directory based on these by-id names.

Multiple devices can be listed and their metadata will all be stored in the one directory.

– restore=directory
This is the reverse of --dump. mdadm will locate a file in the directory that has a name appropriate for the given device and will restore metadata from it. Names that match /dev/disk/by-id names are preferred, however if two of those refer to different files, mdadm will not choose between them but will abort the operation.

If a file name is given instead of a directory then mdadm will restore from that file to a single device, always provided the size of the file matches that of the device, and the file contains valid metadata.

– stop The devices should be active md arrays which will be deactivated, as long as they are not currently in use.

– run This will fully activate a partially assembled md array.

– readonly
This will mark an active array as read-only, providing that it is not currently being used.

– readwrite
This will change a readonly array back to being read/write.

– scan For all operations except --examine, --scan will cause the operation to be applied to all arrays listed in /proc/mdstat. For --examine, --scan causes all devices listed in the config file to be examined.

– b, --brief
Be less verbose. This is used with --detail and --examine. Using --brief with --verbose gives an intermediate level of verbosity.

MONITOR MODE
Usage: mdadm --monitor options... devices…

This usage causes mdadm to periodically poll a number of md arrays and to report on any events noticed. mdadm will never exit once it decides that there are arrays to be checked, so it should normally be run in the background.

As well as reporting events, mdadm may move a spare drive from one array to another if they are in the same spare-group or domain and if the destination array has a failed drive but no spares.

If any devices are listed on the command line, mdadm will only monitor those devices. Otherwise all arrays listed in the configuration file will be monitored. Further, if --scan is given, then any other md devices that appear in /proc/mdstat will also be monitored.

The result of monitoring the arrays is the generation of events. These events are passed to a separate program (if specified) and may be mailed to a given E-mail address.

When passing events to a program, the program is run once for each event, and is given 2 or 3 command-line arguments: the first is the name of the event (see below), the second is the name of the md device which is affected, and the third is the name of a related device if relevant (such as a component device that has failed).

If --scan is given, then a program or an E-mail address must be specified on the command line or in the config file. If neither are available, then mdadm will not monitor anything. Without --scan, mdadm will continue monitoring as long as something was found to monitor. If no program or email is given, then each event is reported to stdout.

The different events are:

DeviceDisappeared
An md array which previously was configured appears to no longer be configured. (syslog priority: Critical)

If mdadm was told to monitor an array which is RAID0 or Linear, then it will report DeviceDisappeared with the extra information Wrong-Level. This is because RAID0 and Linear do not support the device-failed, hot-spare and resync operations which are monitored.

RebuildStarted
An md array started reconstruction (e.g. recovery, resync, reshape, check, repair). (syslog priority: Warning)

RebuildNN
Where NN is a two-digit number (ie. 05, 48). This indicates that rebuild has passed that many percent of the total. The events are generated with fixed increment since 0. Increment size may be specified with a commandline option (default is 20). (syslog priority: Warning)

RebuildFinished
An md array that was rebuilding, isn't any more, either because it finished normally or was aborted. (syslog priority: Warning)

Fail An active component device of an array has been marked as faulty. (syslog priority: Critical)

FailSpare
A spare component device which was being rebuilt to replace a faulty device has failed. (syslog priority: Critical)

SpareActive
A spare component device which was being rebuilt to replace a faulty device has been successfully rebuilt and has been made active. (syslog priority: Info)

NewArray
A new md array has been detected in the /proc/mdstat file. (syslog priority: Info)

DegradedArray
A newly noticed array appears to be degraded. This message is not generated when mdadm notices a drive failure which causes degradation, but only when mdadm notices that an array is degraded when it first sees the array. (syslog priority: Critical)

MoveSpare
A spare drive has been moved from one array in a spare-group or domain to another to allow a failed drive to be replaced. (syslog priority: Info)

SparesMissing
If mdadm has been told, via the config file, that an array should have a certain number of spare devices, and mdadm detects that it has fewer than this number when it first sees the array, it will report a SparesMissing message. (syslog priority: Warning)

TestMessage
An array was found at startup, and the --test flag was given. (syslog priority: Info)

Only Fail, FailSpare, DegradedArray, SparesMissing and TestMessage cause Email to be sent. All events cause the program to be run. The program is run with two or three arguments: the event name, the array device and possibly a second device.

Each event has an associated array device (e.g. /dev/md1) and possibly a second device. For Fail, FailSpare, and SpareActive the second device is the relevant component device. For MoveSpare the second device is the array that the spare was moved from.

For mdadm to move spares from one array to another, the different arrays need to be labeled with the same spare-group or the spares must be allowed to migrate through matching POLICY domains in the configuration file. The spare-group name can be any string; it is only necessary that different spare groups use different names.

When mdadm detects that an array in a spare group has fewer active devices than necessary for the complete array, and has no spare devices, it will look for another array in the same spare group that has a full complement of working drive and a spare. It will then attempt to remove the spare from the second drive and add it to the first. If the removal succeeds but the adding fails, then it is added back to the original array.

If the spare group for a degraded array is not defined, mdadm will look at the rules of spare migration specified by POLICY lines in mdadm.conf and then follow similar steps as above if a matching spare is found.

GROW MODE

The GROW mode is used for changing the size or shape of an active array. For this to work, the kernel must support the necessary change. Various types of growth are being added during 2.6 development.

Currently the supported changes include

- change the "size" attribute for RAID1, RAID4, RAID5 and RAID6.
- increase or decrease the "raid-devices" attribute of RAID0, RAID1, RAID4, RAID5, and RAID6.
- change the chunk-size and layout of RAID0, RAID4, RAID5, RAID6 and RAID10.
- convert between RAID1 and RAID5, between RAID5 and RAID6, between RAID0, RAID4, and RAID5, and between RAID0 and RAID10 (in the near-2 mode).
- add a write-intent bitmap to any array which supports these bitmaps, or remove a write-intent bitmap from such an array.
- change the array's consistency policy.

Using GROW on containers is currently supported only for Intel's IMSM container format. The number of devices in a container can be increased - which affects all arrays in the container - or an array in a container can be converted between levels where those levels are supported by the container, and the conversion is on of those listed above.

Notes:

- Intel's native checkpointing doesn't use --backup-file option and it is transparent for assembly feature.
- Roaming between Windows(R) and Linux systems for IMSM metadata is not supported during grow process.
- When growing a raid0 device, the new component disk size (or external backup size) should be larger than LCM(old, new) * chunk-size * 2, where LCM() is the least common multiple of the old and new count of component disks, and "* 2" comes from the fact that mdadm refuses to use more than half of a spare device for backup space.

SIZE CHANGES

Normally when an array is built the "size" is taken from the smallest of the drives. If all the small drives in an arrays are, one at a time, removed and replaced with larger drives, then you could have an array of large drives with only a small amount used. In this situation, changing the "size" with "GROW" mode will allow the extra space to start being used. If the size is increased in this way, a "resync" process will start to make sure the new parts of the array are synchronised.

Note that when an array changes size, any filesystem that may be stored in the array will not automatically grow or shrink to use or vacate the space. The filesystem will need to be explicitly told to use the extra space after growing, or to reduce its size prior to shrinking the array.

Also the size of an array cannot be changed while it has an active bitmap. If an array has a bitmap, it must be removed before the size can be changed. Once the change is complete a new bitmap can be created.

Note: --grow --size is not yet supported for external file bitmap.

RAID-DEVICES CHANGES

A RAID1 array can work with any number of devices from 1 upwards (though 1 is not very useful). There may be times which you want to increase or decrease the number of active devices. Note that this is different to hot-add or hot-remove which changes the number of inactive devices.

When reducing the number of devices in a RAID1 array, the slots which are to be removed from the array must already be vacant. That is, the devices which were in those slots must be failed and removed.

When the number of devices is increased, any hot spares that are present will be activated immediately.

Changing the number of active devices in a RAID5 or RAID6 is much more effort. Every block in the array will need to be read and written back to a new location. From 2.6.17, the Linux Kernel is able to increase the number of devices in a RAID5 safely, including restarting an interrupted "reshape". From 2.6.31, the Linux Kernel is able to increase or decrease the number of devices in a RAID5 or RAID6.

From 2.6.35, the Linux Kernel is able to convert a RAID0 in to a RAID4 or RAID5. mdadm uses this functionality and the ability to add devices to a RAID4 to allow devices to be added to a RAID0. When requested to do this, mdadm will convert the RAID0 to a RAID4, add the necessary disks and make the reshape happen, and then convert the RAID4 back to RAID0.

When decreasing the number of devices, the size of the array will also decrease. If there was data in the array, it could get destroyed and this is not reversible, so you should firstly shrink the filesystem on the array to fit within the new size. To help prevent accidents, mdadm requires that the size of the array be decreased first with mdadm --grow --array-size. This is a reversible change which simply makes the end of the array inaccessible. The integrity of any data can then be checked before the non-reversible reduction in the number of devices is request.

When relocating the first few stripes on a RAID5 or RAID6, it is not possible to keep the data on disk completely consistent and crash-proof. To provide the required safety, mdadm disables writes to the array while this "critical section" is reshaped, and takes a backup of the data that is in that section. For grows, this backup may be stored in any spare devices that the array has, however it can also be stored in a separate file specified with the --backup-file option, and is required to be specified for shrinks, RAID level changes and layout changes. If this option is used, and the system does crash during the critical period, the same file must be passed to --assemble to restore the backup and reassemble the array. When shrinking rather than growing the array, the reshape is done from the end towards the beginning, so the "critical section" is at the end of the reshape.

LEVEL CHANGES

Changing the RAID level of any array happens instantaneously. However in the RAID5 to RAID6 case this requires a non-standard layout of the RAID6 data, and in the RAID6 to RAID5 case that non-standard layout is required before the change can be accomplished. So while the level change is instant, the accompanying layout change can take quite a long time. A --backup-file is required. If the array is not simultaneously being grown or shrunk, so that the array size will remain the same - for example, reshaping a 3-drive RAID5 into a 4-drive RAID6 - the backup file will be used not just for a "critical section" but throughout the reshape operation, as described below under LAYOUT CHANGES.

CHUNK-SIZE AND LAYOUT CHANGES

Changing the chunk-size or layout without also changing the number of devices as the same time will involve re-writing all blocks in-place. To ensure against data loss in the case of a crash, a backup-file must be provided for these changes. Small sections of the array will be copied to the backup file while they are being rearranged. This means that all the data is copied twice, once to the backup and once to the new layout on the array, so this type of reshape will go very slowly.

If the reshape is interrupted for any reason, this backup file must be made available to mdadm --assemble so the array can be reassembled. Consequently the file cannot be stored on the device being reshaped.

BITMAP CHANGES

A write-intent bitmap can be added to, or removed from, an active array. Either internal bitmaps, or bitmaps stored in a separate file, can be added. Note that if you add a bitmap stored in a file which is in a filesystem that is on the RAID array being affected, the system will deadlock. The bitmap must be on a separate filesystem.

CONSISTENCY POLICY CHANGES

The consistency policy of an active array can be changed by using the --consistency-policy option in Grow mode. Currently this works only for the ppl and resync policies and allows one to enable or disable the RAID5 Partial Parity Log (PPL).

INCREMENTAL MODE

Usage: mdadm --incremental [--run] [--quiet] component-device [optional-aliases-for-device]

Usage: mdadm --incremental --fail component-device

Usage: mdadm --incremental --rebuild-map

Usage: mdadm --incremental --run --scan

This mode is designed to be used in conjunction with a device discovery system. As devices are found in a system, they can be passed to mdadm --incremental to be conditionally added to an appropri-

ate array.

Conversely, it can also be used with the --fail flag to do just the opposite and find whatever array a particular device is part of and remove the device from that array.

If the device passed is a CONTAINER device created by a previous call to mdadm, then rather than trying to add that device to an array, all the arrays described by the metadata of the container will be started.

mdadm performs a number of tests to determine if the device is part of an array, and which array it should be part of. If an appropriate array is found, or can be created, mdadm adds the device to the array and conditionally starts the array.

Note that mdadm will normally only add devices to an array which were previously working (active or spare) parts of that array. The support for automatic inclusion of a new drive as a spare in some array requires a configuration through POLICY in config file.

The tests that mdadm makes are as follow:

- Is the device permitted by mdadm.conf? That is, is it listed in a DEVICES line in that file. If DEVICES is absent then the default it to allow any device. Similarly if DEVICES contains the special word partitions then any device is allowed. Otherwise the device name given to mdadm, or one of the aliases given, or an alias found in the filesystem, must match one of the names or patterns in a DEVICES line.

  This is the only context where the aliases are used. They are usually provided by a udev rules mentioning $env{DEVLINKS}.

- Does the device have a valid md superblock? If a specific metadata version is requested with --metadata or -e then only that style of metadata is accepted, otherwise mdadm finds any known version of metadata. If no md metadata is found, the device may be still added to an array as a spare if POLICY allows.

mdadm keeps a list of arrays that it has partially assembled in /run/mdadm/map. If no array exists which matches the metadata on the new device, mdadm must choose a device name and unit number. It does this based on any name given in mdadm.conf or any name information stored in the metadata. If this name suggests a unit number, that number will be used, otherwise a free unit number will be chosen. Normally mdadm will prefer to create a partitionable array, however if the CREATE line in mdadm.conf suggests that a non-partitionable array is preferred, that will be honoured.

If the array is not found in the config file and its metadata does not identify it as belonging to the "homehost", then mdadm will choose a name for the array which is certain not to conflict with any array which does belong to this host. It does this be adding an underscore and a small number to the name preferred by the metadata.

Once an appropriate array is found or created and the device is added, mdadm must decide if the array is ready to be started. It will normally compare the number of available (non-spare) devices to the number of devices that the metadata suggests need to be active. If there are at least that many, the array will be started. This means that if any devices are missing the array will not be restarted.

As an alternative, --run may be passed to mdadm in which case the array will be run as soon as there are enough devices present for the data to be accessible. For a RAID1, that means one device will start the array. For a clean RAID5, the array will be started as soon as all but one drive is present.

Note that neither of these approaches is really ideal. If it can be known that all device discovery has completed, then
  mdadm -IRs
can be run which will try to start all arrays that are being incrementally assembled. They are started in "read-auto" mode in which they are read-only until the first write request. This means that no metadata updates are made and no attempt at resync or recovery happens. Further devices that are found before the first write can still be added safely.

ENVIRONMENT

This section describes environment variables that affect how mdadm operates.

MDADM_NO_MDMON
Setting this value to 1 will prevent mdadm from automatically launching mdmon. This variable is intended primarily for debugging mdadm/mdmon.

MDADM_NO_UDEV
Normally, mdadm does not create any device nodes in /dev, but leaves that task to udev. If udev appears not to be configured, or if this environment variable is set to '1', the mdadm will create and devices that are needed.

MDADM_NO_SYSTEMCTL
If mdadm detects that systemd is in use it will normally request systemd to start various background tasks (particularly mdmon) rather than forking and running them in the background. This can be suppressed by setting MDADM_NO_SYSTEMCTL=1.

IMSM_NO_PLATFORM
A key value of IMSM metadata is that it allows interoperability with boot ROMs on Intel platforms, and with other major operating systems. Consequently, mdadm will only allow an IMSM array to be created or modified if detects that it is running on an Intel platform which supports IMSM, and supports the particular configuration of IMSM that is being requested (some functionality requires newer OROM support.

These checks can be suppressed by setting IMSM_NO_PLATFORM=1 in the environment. This can be useful for testing or for disaster recovery. You should be aware that interoperability may be compromised by setting this value.

MDADM_GROW_ALLOW_OLD
If an array is stopped while it is performing a reshape and that reshape was making use of a backup file, then when the array is re-assembled mdadm will sometimes complain that the backup file is too old. If this happens and you are certain it is the right backup file, you can over-ride this check by setting MDADM_GROW_ALLOW_OLD=1 in the environment.

MDADM_CONF_AUTO
Any string given in this variable is added to the start of the AUTO line in the config file, or treated as the whole AUTO line if none is given. It can be used to disable certain metadata types when mdadm is called from a boot script. For example
  export MDADM_CONF_AUTO='-ddf -imsm'
will make sure that mdadm does not automatically assemble any DDF or IMSM arrays that are found. This can be useful on systems configured to manage such arrays with dmraid.

EXAMPLES

mdadm --query /dev/name-of-device
This will find out if a given device is a RAID array, or is part of one, and will provide brief information about the device.

mdadm --assemble --scan
This will assemble and start all arrays listed in the standard config file. This command will typically go in a system startup file.

mdadm --stop --scan
This will shut down all arrays that can be shut down (i.e. are not currently in use). This will typically go in a system shutdown script.

mdadm --follow --scan --delay=120
If (and only if) there is an Email address or program given in the standard config file, then monitor the status of all arrays listed in that file by polling them ever 2 minutes.

mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/hd[ac]1
Create /dev/md0 as a RAID1 array consisting of /dev/hda1 and /dev/hdc1.

echo 'DEVICE /dev/hd*[0-9] /dev/sd*[0-9]' > mdadm.conf
mdadm --detail --scan >> mdadm.conf
This will create a prototype config file that describes currently active arrays that are known to be made from partitions of IDE or SCSI drives. This file should be reviewed before being used as it may contain unwanted detail.

echo 'DEVICE /dev/hd[a-z] /dev/sd*[a-z]' > mdadm.conf
mdadm --examine --scan --config=mdadm.conf >> mdadm.conf
This will find arrays which could be assembled from existing IDE and SCSI whole drives (not partitions), and store the information in the format of a config file. This file is very likely to contain unwanted detail, particularly the devices= entries. It should be reviewed and edited before being used as an actual config file.

mdadm --examine --brief --scan --config=partitions
mdadm -Ebsc partitions
Create a list of devices by reading /proc/partitions, scan these for RAID superblocks, and printout a brief listing of all that were found.

mdadm -Ac partitions -m 0 /dev/md0
Scan all partitions and devices listed in /proc/partitions and assemble /dev/md0 out of all such devices with a RAID superblock with a minor number of 0.

mdadm --monitor --scan --daemonise > /run/mdadm/mon.pid
If config file contains a mail address or alert program, run mdadm in the background in monitor mode monitoring all md devices. Also write pid of mdadm daemon to /run/mdadm/mon.pid.

mdadm -Iq /dev/somedevice
Try to incorporate newly discovered device into some array as appropriate.

mdadm --incremental --rebuild-map --run --scan
Rebuild the array map from any current arrays, and then start any that can be started.

mdadm /dev/md4 --fail detached --remove detached
Any devices which are components of /dev/md4 will be marked as faulty and then remove from the array.

mdadm --grow /dev/md4 --level=6 --backup-file=/root/backup-md4
The array /dev/md4 which is currently a RAID5 array will be converted to RAID6. There should normally already be a spare drive attached to the array as a RAID6 needs one more drive than a matching RAID5.

mdadm --create /dev/md/ddf --metadata=ddf --raid-disks 6 /dev/sd[a-f]
Create a DDF array over 6 devices.

mdadm --create /dev/md/home -n3 -l5 -z 30000000 /dev/md/ddf
Create a RAID5 array over any 3 devices in the given DDF set. Use only 30 gigabytes of each device.

mdadm -A /dev/md/ddf1 /dev/sd[a-f]
Assemble a pre-exist ddf array.

mdadm -I /dev/md/ddf1
Assemble all arrays contained in the ddf array, assigning names as appropriate.

mdadm --create --help
Provide help about the Create mode.

mdadm --config --help
Provide help about the format of the config file.

mdadm --help
Provide general help.

FILES

/proc/mdstat
If you're using the /proc filesystem, /proc/mdstat lists all active md devices with information about them. mdadm uses this to find arrays when --scan is given in Misc mode, and to monitor array reconstruction on Monitor mode.

/etc/mdadm/mdadm.conf (or /etc/mdadm.conf)
The config file lists which devices may be scanned to see if they contain MD super block, and gives identifying information (e.g. UUID) about known MD arrays. See mdadm.conf(5) for more details.

/etc/mdadm/mdadm.conf.d (or /etc/mdadm.conf.d)
A directory containing configuration files which are read in lexical order.

/run/mdadm/map
When --incremental mode is used, this file gets a list of arrays currently being created.

DEVICE NAMES
mdadm understand two sorts of names for array devices.

The first is the so-called 'standard' format name, which matches the names used by the kernel and which appear in /proc/mdstat.

The second sort can be freely chosen, but must reside in /dev/md/. When giving a device name to mdadm to create or assemble an array, either full path name such as /dev/md0 or /dev/md/home can be given, or just the suffix of the second sort of name, such as home can be given.

When mdadm chooses device names during auto-assembly or incremental assembly, it will sometimes add a small sequence number to the end of the name to avoid conflicted between multiple arrays that have the same name. If mdadm can reasonably determine that the array really is meant for this host, either by a hostname in the metadata, or by the presence of the array in mdadm.conf, then it will leave off the suffix if possible. Also if the homehost is specified as <ignore> mdadm will only use a suffix if a different array of the same name already exists or is listed in the config file.

The standard names for non-partitioned arrays (the only sort of md array available in 2.4 and earlier) are of the form

/dev/mdNN

where NN is a number. The standard names for partitionable arrays (as available from 2.6 onwards) are of the form:

/dev/md_dNN

Partition numbers should be indicated by adding "pMM" to these, thus "/dev/md/d1p2."

From kernel version 2.6.28 the "non-partitioned array" can actually be partitioned. So the "md_dNN" names are no longer needed, and partitions such as "/dev/mdNNpXX" are possible.

From kernel version 2.6.29 standard names can be non-numeric following the form:

/dev/md_XXX

where XXX is any string. These names are supported by mdadm since version 3.3 provided they are enabled in mdadm.conf.

NOTE
mdadm was previously known as mdctl.

SEE ALSO
For further information on mdadm usage, MD and the various levels of RAID, see:

https://raid.wiki.kernel.org/

) based upon Jakob Østergaard's Software-RAID.HOWTO(

The latest version of mdadm should always be available from

https://www.kernel.org/pub/linux/utils/raid/mdadm/

Related man pages:

mdmon(8), mdadm.conf(5), md(4)

man 4 md

NAME

md - Multiple Device driver aka Linux Software RAID

SYNOPSIS

/dev/mdn
/dev/md/n
/dev/md/name

DESCRIPTION

The md driver provides virtual devices that are created from one or more independent underlying devices. This array of devices often contains redundancy and the devices are often disk drives, hence the acronym RAID which stands for a Redundant Array of Independent Disks.

md supports RAID levels 1 (mirroring), 4 (striped array with parity device), 5 (striped array with distributed parity information), 6 (striped array with distributed dual redundancy information), and 10 (striped and mirrored). If some number of underlying devices fails while using one of these levels, the array will continue to function; this number is one for RAID levels 4 and 5, two for RAID level 6, and all but one (N-1) for RAID level 1, and dependent on configuration for level 10.

md also supports a number of pseudo RAID (non-redundant) configurations including RAID0 (striped array), LINEAR (catenated array), MULTIPATH (a set of different interfaces to the same device), and FAULTY (a layer over a single device into which errors can be injected).

MD METADATA

Each device in an array may have some metadata stored in the device. This metadata is sometimes called a superblock. The metadata records information about the structure and state of the array. This allows the array to be reliably re-assembled after a shutdown.

From Linux kernel version 2.6.10, md provides support for two different formats of metadata, and other formats can be added. Prior to this release, only one format is supported.

The common format — known as version 0.90 — has a superblock that is 4K long and is written into a 64K aligned block that starts at least 64K and less than 128K from the end of the device (i.e. to get the address of the superblock round the size of the device down to a multiple of 64K and then subtract 64K). The available size of each device is the amount of space before the super block, so between 64K and 128K is lost when a device in incorporated into an MD array. This superblock stores

multi-byte fields in a processor-dependent manner, so arrays cannot easily be moved between computers with different processors.

The new format – known as version 1 – has a superblock that is normally 1K long, but can be longer. It is normally stored between 8K and 12K from the end of the device, on a 4K boundary, though variations can be stored at the start of the device (version 1.1) or 4K from the start of the device (version 1.2). This metadata format stores multibyte data in a processor-independent format and supports up to hundreds of component devices (version 0.90 only supports 28).

The metadata contains, among other things:

LEVEL The manner in which the devices are arranged into the array (LINEAR, RAID0, RAID1, RAID4, RAID5, RAID10, MULTIPATH).

UUID a 128 bit Universally Unique Identifier that identifies the array that contains this device.

When a version 0.90 array is being reshaped (e.g. adding extra devices to a RAID5), the version number is temporarily set to 0.91. This ensures that if the reshape process is stopped in the middle (e.g. by a system crash) and the machine boots into an older kernel that does not support reshaping, then the array will not be assembled (which would cause data corruption) but will be left untouched until a kernel that can complete the reshape processes is used.

ARRAYS WITHOUT METADATA

While it is usually best to create arrays with superblocks so that they can be assembled reliably, there are some circumstances when an array without superblocks is preferred. These include:

LEGACY ARRAYS

Early versions of the md driver only supported LINEAR and RAID0 configurations and did not use a superblock (which is less critical with these configurations). While such arrays should be rebuilt with superblocks if possible, md continues to support them.

FAULTY Being a largely transparent layer over a different device, the FAULTY personality doesn't gain anything from having a superblock.

MULTIPATH

It is often possible to detect devices which are different paths to the same storage directly rather than having a distinctive superblock written to the device and searched for on all paths. In this case, a MULTIPATH array with no superblock makes sense.

RAID1 In some configurations it might be desired to create a RAID1 configuration that does not use a superblock, and to maintain the state of the array elsewhere. While not encouraged for general use, it does have special-purpose uses and is supported.

ARRAYS WITH EXTERNAL METADATA

From release 2.6.28, the md driver supports arrays with externally managed metadata. That is, the metadata is not managed by the kernel but rather by a user-space program which is external to the kernel. This allows support for a variety of metadata formats without cluttering the kernel with lots of details.

md is able to communicate with the user-space program through various sysfs attributes so that it can make appropriate changes to the metadata - for example to mark a device as faulty. When necessary, md will wait for the program to acknowledge the event by writing to a sysfs attribute. The manual page for mdmon(8) contains more detail about this interaction.

CONTAINERS

Many metadata formats use a single block of metadata to describe a number of different arrays which all use the same set of devices. In this case it is helpful for the kernel to know about the full set of devices as a whole. This set is known to md as a container. A container is an md array with externally managed metadata and with device offset and size so that it just covers the metadata part of the devices. The remainder of each device is available to be incorporated into various arrays.

LINEAR

A LINEAR array simply catenates the available space on each drive to form one large virtual drive.

One advantage of this arrangement over the more common RAID0 arrangement is that the array may be reconfigured at a later time with an extra drive, so the array is made bigger without disturbing the data that is on the array. This can even be done on a live array.

If a chunksize is given with a LINEAR array, the usable space on each device is rounded down to a multiple of this chunksize.

RAID0

A RAID0 array (which has zero redundancy) is also known as a striped array. A RAID0 array is configured at creation with a Chunk Size which must be a power of two (prior to Linux 2.6.31), and at least 4 kibibytes.

The RAID0 driver assigns the first chunk of the array to the first device, the second chunk to the second device, and so on until all drives have been assigned one chunk. This collection of chunks forms a stripe. Further chunks are gathered into stripes in the same way, and are assigned to the remaining space in the drives.

If devices in the array are not all the same size, then once the smallest device has been exhausted, the RAID0 driver starts collecting chunks into smaller stripes that only span the drives which still have remaining space.

A bug was introduced in linux 3.14 which changed the layout of blocks in a RAID0 beyond the region that is striped over all devices. This bug does not affect an array with all devices the same size, but can affect other RAID0 arrays.

Linux 5.4 (and some stable kernels to which the change was backported) will not normally assemble such an array as it cannot know which layout to use. There is a module parameter "raid0.default_layout" which can be set to "1" to force the kernel to use the pre-3.14 layout or to "2" to force it to use the 3.14-and-later layout. when creating a new RAID0 array, mdadm will record the chosen layout in the metadata in a way that allows newer kernels to assemble the array without needing a module parameter.

To assemble an old array on a new kernel without using the module parameter, use either the --update=layout-original option or the --update=layout-alternate option.

Once you have updated the layout you will not be able to mount the array on an older kernel. If you need to revert to an older kernel, the layout information can be erased with the --update=layout-unspecificed option. If you use this option to --assemble while running a newer kernel, the array will NOT assemble, but the metadata will be update so that it can be assembled on an older kernel.

No that setting the layout to "unspecified" removes protections against this bug, and you must be sure that the kernel you use matches the layout of the array.

RAID1

A RAID1 array is also known as a mirrored set (though mirrors tend to provide reflected images, which RAID1 does not) or a plex.

Once initialised, each device in a RAID1 array contains exactly the same data. Changes are written to all devices in parallel. Data is read from any one device. The driver attempts to distribute read requests across all devices to maximise performance.

All devices in a RAID1 array should be the same size. If they are not, then only the amount of space available on the smallest device is used (any extra space on other devices is wasted).

Note that the read balancing done by the driver does not make the RAID1 performance profile be the same as for RAID0; a single stream of sequential input will not be accelerated (e.g. a single dd), but multiple sequential streams or a random workload will use more than one spindle. In theory, having an N-disk RAID1 will allow N sequential threads to read from all disks.

Individual devices in a RAID1 can be marked as "write-mostly". These drives are excluded from the normal read balancing and will only be read from when there is no other option. This can be useful for devices connected over a slow link.

RAID4

A RAID4 array is like a RAID0 array with an extra device for storing parity. This device is the last of the active devices in the array. Unlike RAID0, RAID4 also requires that all stripes span all drives, so extra space on devices that are larger than the smallest is wasted.

When any block in a RAID4 array is modified, the parity block for that stripe (i.e. the block in the parity device at the same device offset as the stripe) is also modified so that the parity block always contains the "parity" for the whole stripe. I.e. its content is equivalent to the result of performing an exclusive-or operation between all the data blocks in the stripe.

This allows the array to continue to function if one device fails. The data that was on that device can be calculated as needed from the parity block and the other data blocks.

RAID5

RAID5 is very similar to RAID4. The difference is that the parity blocks for each stripe, instead of being on a single device, are distributed across all devices. This allows more parallelism when writing, as two different block updates will quite possibly affect parity blocks on different devices so there is less contention.

This also allows more parallelism when reading, as read requests are distributed over all the devices in the array instead of all but one.

RAID6

RAID6 is similar to RAID5, but can handle the loss of any two devices without data loss. Accordingly, it requires N+2 drives to store N drives worth of data.

The performance for RAID6 is slightly lower but comparable to RAID5 in normal mode and single disk failure mode. It is very slow in dual disk failure mode, however.

RAID10

RAID10 provides a combination of RAID1 and RAID0, and is sometimes known as RAID1+0. Every datablock is duplicated some number of times, and the resulting collection of datablocks are distributed over multiple drives.

When configuring a RAID10 array, it is necessary to specify the number of replicas of each data block that are required (this will usually be 2) and whether their layout should be "near", "far" or "offset" (with "offset" being available since Linux 2.6.18).

About the RAID10 Layout Examples:
The examples below visualise the chunk distribution on the underlying devices for the respective layout.

For simplicity it is assumed that the size of the chunks equals the size of the blocks of the underlying devices as well as those of the RAID10 device exported by the kernel (for example /dev/md/name).
Therefore the chunks / chunk numbers map directly to the blocks /block addresses of the exported RAID10 device.

Decimal numbers (0, 1, 2, ...) are the chunks of the RAID10 and due to the above assumption also the blocks and block addresses of the exported RAID10 device.
Repeated numbers mean copies of a chunk / block (obviously on different underlying devices).
Hexadecimal numbers (0x00, 0x01, 0x02, ...) are the block addresses of the underlying devices.

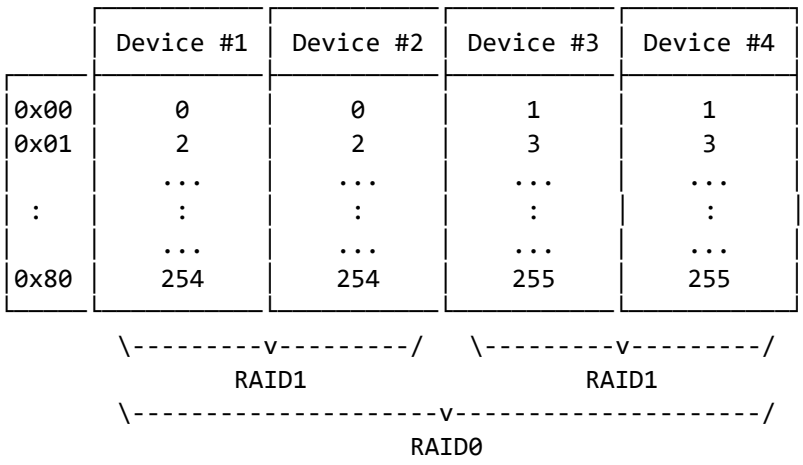
"near" Layout

When "near" replicas are chosen, the multiple copies of a given chunk are laid out consecutively ("as close to each other as possible") across the stripes of the array.

With an even number of devices, they will likely (unless some misalignment is present) lay at the very same offset on the different devices.
This is as the "classic" RAID1+0; that is two groups of mirrored devices (in the example below the groups Device #1 / #2 and Device #3 / #4 are each a RAID1) both in turn forming a striped RAID0.

Example with 2 copies per chunk and an even number (4) of de-

vices:



Example with 2 copies per chunk and an odd number (5) of devices:

	Dev #1	Dev #2	Dev #3	Dev #4	Dev #5
0x00	0	0	1	1	2
0x01	2	3	3	4	4
:	:	:	:	:	:
0x80	317	318	318	319	319

"far" Layout

When "far" replicas are chosen, the multiple copies of a given chunk are laid out quite distant ("as far as reasonably possible") from each other.

First a complete sequence of all data blocks (that is all the data one sees on the exported RAID10 block device) is striped over the devices. Then another (though "shifted") complete sequence of all data blocks; and so on (in the case of more than 2 copies per chunk).

The "shift" needed to prevent placing copies of the same chunks on the same devices is actually a cyclic permutation with offset 1 of each of the stripes within a complete sequence of chunks.
The offset 1 is relative to the previous complete sequence of chunks, so in case of more than 2 copies per chunk one gets the following offsets:

1. complete sequence of chunks: offset = 0
2. complete sequence of chunks: offset = 1
3. complete sequence of chunks: offset = 2
                        :
n. complete sequence of chunks: offset = n-1

Example with 2 copies per chunk and an even number (4) of devices:

	Device #1	Device #2	Device #3	Device #4	
0x00	0	1	2	3	\
0x01	4	5	6	7	> [#]
	...	...	...	...	...
:	:	:	:	:	:
	...	...	...	...	...
0x40	252	253	254	255	/
0x41	3	0	1	2	\
0x42	7	4	5	6	> [#]~
	...	...	...	...	...
:	:	:	:	:	:
	...	...	...	...	...
0x80	255	252	253	254	/

Example with 2 copies per chunk and an odd number (5) of devices:

	Dev #1	Dev #2	Dev #3	Dev #4	Dev #5	
0x00	0	1	2	3	4	\
0x01	5	6	7	8	9	> [#]
	...	...	...	...	...	...
:	:	:	:	:	:	:
	...	...	...	...	...	...
0x40	315	316	317	318	319	/
0x41	4	0	1	2	3	\
0x42	9	5	6	7	8	> [#]~
	...	...	...	...	...	...
:	:	:	:	:	:	:
	...	...	...	...	...	...
0x80	319	315	316	317	318	/

With [#] being the complete sequence of chunks and [#]~ the cyclic permutation with offset 1 thereof (in the case of more than 2 copies per chunk there would be ([#]~)~, (([#]~)~)~, ...).

The advantage of this layout is that MD can easily spread sequential reads over the devices, making them similar to RAID0 in terms of speed.
The cost is more seeking for writes, making them substantially slower.

"offset" Layout

When "offset" replicas are chosen, all the copies of a given chunk are striped consecutively ("offset by the stripe length after each other") over the devices.

Explained in detail, <number of devices> consecutive chunks are striped over the devices, immediately followed by a "shifted" copy of these chunks (and by further such "shifted" copies in the case of more than 2 copies per chunk).
This pattern repeats for all further consecutive chunks of the exported RAID10 device (in other words: all further data blocks).

The "shift" needed to prevent placing copies of the same chunks on the same devices is actually a cyclic permutation with offset 1 of each of the striped copies of <number of devices> consecutive chunks.
The offset 1 is relative to the previous striped copy of <number of devices> consecutive chunks, so in case of more than 2 copies per chunk one gets the following offsets:
1. <number of devices> consecutive chunks: offset = 0
2. <number of devices> consecutive chunks: offset = 1
3. <number of devices> consecutive chunks: offset = 2
:
n. <number of devices> consecutive chunks: offset = n-1

Example with 2 copies per chunk and an even number (4) of devices:

	Device #1	Device #2	Device #3	Device #4	
0x00	0	1	2	3	) AA
0x01	3	0	1	2	) AA~
0x02	4	5	6	7	) AB
0x03	7	4	5	6	) AB~
	...	...	...	...	...
:	:	:	:	:	:
	...	...	...	...	...
0x79	251	252	253	254	) EX
0x80	254	251	252	253	) EX~

Example with 2 copies per chunk and an odd number (5) of devices:

	Dev #1	Dev #2	Dev #3	Dev #4	Dev #5	
0x00	0	1	2	3	4	) AA
0x01	4	0	1	2	3	) AA~
0x02	5	6	7	8	9	) AB
0x03	9	5	6	7	8	) AB~
	...	...	...	...	...	...
:	:	:	:	:	:	:
	...	...	...	...	...	...
0x79	314	315	316	317	318	) EX
0x80	318	314	315	316	317	) EX~

With AA, AB, ..., AZ, BA, ... being the sets of <number of devices> consecutive chunks and AA~, AB~, ..., AZ~, BA~, ... the cyclic permutations with offset 1 thereof (in the case of more than 2 copies per chunk there would be (AA~)~, ... as well as ((AA~)~)~, ... and so on).

This should give similar read characteristics to "far" if a suitably large chunk size is used, but without as much seeking for writes.

It should be noted that the number of devices in a RAID10 array need not be a multiple of the number of replica of each data block; however, there must be at least as many devices as replicas.

If, for example, an array is created with 5 devices and 2 replicas, then space equivalent to 2.5 of the devices will be available, and every block will be stored on two different devices.

Finally, it is possible to have an array with both "near" and "far" copies. If an array is configured with 2 near copies and 2 far copies, then there will be a total of 4 copies of each block, each on a different drive. This is an artifact of the implementation and is unlikely to be of real value.

MULTIPATH

MULTIPATH is not really a RAID at all as there is only one real device in a MULTIPATH md array. However there are multiple access points (paths) to this device, and one of these paths might fail, so there are some similarities.

A MULTIPATH array is composed of a number of logically different devices, often fibre channel interfaces, that all refer the the same real device. If one of these interfaces fails (e.g. due to cable problems), the MULTIPATH driver will attempt to redirect requests to another interface.

The MULTIPATH drive is not receiving any ongoing development and should be considered a legacy driver. The device-mapper based multipath drivers should be preferred for new installations.

FAULTY

The FAULTY md module is provided for testing purposes. A FAULTY array has exactly one component device and is normally assembled without a superblock, so the md array created provides direct access to all of the data in the component device.

The FAULTY module may be requested to simulate faults to allow testing of other md levels or of filesystems. Faults can be chosen to trigger on read requests or write requests, and can be transient (a subsequent read/write at the address will probably succeed) or persistent (subsequent read/write of the same address will fail). Further, read faults can be "fixable" meaning that they persist until a write request at the same address.

Fault types can be requested with a period. In this case, the fault will recur repeatedly after the given number of requests of the relevant type. For example if persistent read faults have a period of 100, then every 100th read request would generate a fault, and the faulty sector would be recorded so that subsequent reads on that sector would also fail.

There is a limit to the number of faulty sectors that are remembered. Faults generated after this limit is exhausted are treated as transient.

The list of faulty sectors can be flushed, and the active list of failure modes can be cleared.

UNCLEAN SHUTDOWN

When changes are made to a RAID1, RAID4, RAID5, RAID6, or RAID10 array there is a possibility of inconsistency for short periods of time as each update requires at least two block to be written to different devices, and these writes probably won't happen at exactly the same time. Thus if a system with one of these arrays is shutdown in the middle of a write operation (e.g. due to power failure), the array may not be consistent.

To handle this situation, the md driver marks an array as "dirty" before writing any data to it, and marks it as "clean" when the array is

being disabled, e.g. at shutdown. If the md driver finds an array to be dirty at startup, it proceeds to correct any possibly inconsistency. For RAID1, this involves copying the contents of the first drive onto all other drives. For RAID4, RAID5 and RAID6 this involves recalculating the parity for each stripe and making sure that the parity block has the correct data. For RAID10 it involves copying one of the replicas of each block onto all the others. This process, known as "resynchronising" or "resync" is performed in the background. The array can still be used, though possibly with reduced performance.

If a RAID4, RAID5 or RAID6 array is degraded (missing at least one drive, two for RAID6) when it is restarted after an unclean shutdown, it cannot recalculate parity, and so it is possible that data might be undetectably corrupted. The 2.4 md driver does not alert the operator to this condition. The 2.6 md driver will fail to start an array in this condition without manual intervention, though this behaviour can be overridden by a kernel parameter.

RECOVERY

If the md driver detects a write error on a device in a RAID1, RAID4, RAID5, RAID6, or RAID10 array, it immediately disables that device (marking it as faulty) and continues operation on the remaining devices. If there are spare drives, the driver will start recreating on one of the spare drives the data which was on that failed drive, either by copying a working drive in a RAID1 configuration, or by doing calculations with the parity block on RAID4, RAID5 or RAID6, or by finding and copying originals for RAID10.

In kernels prior to about 2.6.15, a read error would cause the same effect as a write error. In later kernels, a read-error will instead cause md to attempt a recovery by overwriting the bad block. i.e. it will find the correct data from elsewhere, write it over the block that failed, and then try to read it back again. If either the write or the re-read fail, md will treat the error the same way that a write error is treated, and will fail the whole device.

While this recovery process is happening, the md driver will monitor accesses to the array and will slow down the rate of recovery if other activity is happening, so that normal access to the array will not be unduly affected. When no other activity is happening, the recovery process proceeds at full speed. The actual speed targets for the two different situations can be controlled by the speed_limit_min and speed_limit_max control files mentioned below.

SCRUBBING AND MISMATCHES

As storage devices can develop bad blocks at any time it is valuable to regularly read all blocks on all devices in an array so as to catch such bad blocks early. This process is called scrubbing.

md arrays can be scrubbed by writing either check or repair to the file md/sync_action in the sysfs directory for the device.

Requesting a scrub will cause md to read every block on every device in the array, and check that the data is consistent. For RAID1 and RAID10, this means checking that the copies are identical. For RAID4, RAID5, RAID6 this means checking that the parity block is (or blocks are) correct.

If a read error is detected during this process, the normal read-error handling causes correct data to be found from other devices and to be written back to the faulty device. In many case this will effectively fix the bad block.

If all blocks read successfully but are found to not be consistent, then this is regarded as a mismatch.

If check was used, then no action is taken to handle the mismatch, it is simply recorded. If repair was used, then a mismatch will be repaired in the same way that resync repairs arrays. For RAID5/RAID6 new parity blocks are written. For RAID1/RAID10, all but one block are overwritten with the content of that one block.

A count of mismatches is recorded in the sysfs file md/mismatch_cnt. This is set to zero when a scrub starts and is incremented whenever a sector is found that is a mismatch. md normally works in units much larger than a single sector and when it finds a mismatch, it does not determine exactly how many actual sectors were affected but simply adds the number of sectors in the IO unit that was used. So a value of 128 could simply mean that a single 64KB check found an error (128 x 512bytes = 64KB).

If an array is created by mdadm with --assume-clean then a subsequent check could be expected to find some mismatches.

On a truly clean RAID5 or RAID6 array, any mismatches should indicate a hardware problem at some level - software issues should never cause such a mismatch.

However on RAID1 and RAID10 it is possible for software issues to cause a mismatch to be reported. This does not necessarily mean that the data on the array is corrupted. It could simply be that the system does not care what is stored on that part of the array - it is unused space.

The most likely cause for an unexpected mismatch on RAID1 or RAID10 occurs if a swap partition or swap file is stored on the array.

When the swap subsystem wants to write a page of memory out, it flags the page as 'clean' in the memory manager and requests the swap device to write it out. It is quite possible that the memory will be changed while the write-out is happening. In that case the 'clean' flag will be found to be clear when the write completes and so the swap subsystem will simply forget that the swapout had been attempted, and will possibly choose a different page to write out.

If the swap device was on RAID1 (or RAID10), then the data is sent from memory to a device twice (or more depending on the number of devices in the array). Thus it is possible that the memory gets changed between the times it is sent, so different data can be written to the different devices in the array. This will be detected by check as a mismatch. However it does not reflect any corruption as the block where this mismatch occurs is being treated by the swap system as being empty, and the data will never be read from that block.

It is conceivable for a similar situation to occur on non-swap files, though it is less likely.

Thus the mismatch_cnt value can not be interpreted very reliably on RAID1 or RAID10, especially when the device is used for swap.

BITMAP WRITE-INTENT LOGGING

From Linux 2.6.13, md supports a bitmap based write-intent log. If configured, the bitmap is used to record which blocks of the array may be out of sync. Before any write request is honoured, md will make sure that the corresponding bit in the log is set. After a period of time with no writes to an area of the array, the corresponding bit will be cleared.

This bitmap is used for two optimisations.

Firstly, after an unclean shutdown, the resync process will consult the bitmap and only resync those blocks that correspond to bits in the bitmap that are set. This can dramatically reduce resync time.

Secondly, when a drive fails and is removed from the array, md stops clearing bits in the intent log. If that same drive is re-added to the array, md will notice and will only recover the sections of the drive that are covered by bits in the intent log that are set. This can allow a device to be temporarily removed and reinserted without causing an enormous recovery cost.

The intent log can be stored in a file on a separate device, or it can be stored near the superblocks of an array which has superblocks.

It is possible to add an intent log to an active array, or remove an

intent log if one is present.

In 2.6.13, intent bitmaps are only supported with RAID1. Other levels with redundancy are supported from 2.6.15.

BAD BLOCK LIST

From Linux 3.5 each device in an md array can store a list of known-bad-blocks. This list is 4K in size and usually positioned at the end of the space between the superblock and the data.

When a block cannot be read and cannot be repaired by writing data recovered from other devices, the address of the block is stored in the bad block list. Similarly if an attempt to write a block fails, the address will be recorded as a bad block. If attempting to record the bad block fails, the whole device will be marked faulty.

Attempting to read from a known bad block will cause a read error. Attempting to write to a known bad block will be ignored if any write errors have been reported by the device. If there have been no write errors then the data will be written to the known bad block and if that succeeds, the address will be removed from the list.

This allows an array to fail more gracefully - a few blocks on different devices can be faulty without taking the whole array out of action.

The list is particularly useful when recovering to a spare. If a few blocks cannot be read from the other devices, the bulk of the recovery can complete and those few bad blocks will be recorded in the bad block list.

RAID WRITE HOLE

Due to non-atomicity nature of RAID write operations, interruption of write operations (system crash, etc.) to RAID456 array can lead to inconsistent parity and data loss (so called RAID-5 write hole). To plug the write hole md supports two mechanisms described below.

DIRTY STRIPE JOURNAL

From Linux 4.4, md supports write ahead journal for RAID456. When the array is created, an additional journal device can be added to the array through write-journal option. The RAID write journal works similar to file system journals. Before writing to the data disks, md persists data AND parity of the stripe to the journal device. After crashes, md searches the journal device for incomplete write operations, and replay them to the data disks.

When the journal device fails, the RAID array is forced to run in read-only mode.

PARTIAL PARITY LOG

From Linux 4.12 md supports Partial Parity Log (PPL) for RAID5 arrays only. Partial parity for a write operation is the XOR of stripe data chunks not modified by the write. PPL is stored in the metadata region of RAID member drives, no additional journal drive is needed. After crashes, if one of the not modified data disks of the stripe is missing, this updated parity can be used to recover its data.

This mechanism is documented more fully in the file Documentation/md/raid5-ppl.rst

WRITE-BEHIND

From Linux 2.6.14, md supports WRITE-BEHIND on RAID1 arrays.

This allows certain devices in the array to be flagged as write-mostly. MD will only read from such devices if there is no other option.

If a write-intent bitmap is also provided, write requests to write-mostly devices will be treated as write-behind requests and md will not wait for writes to those requests to complete before reporting the write as complete to the filesystem.

This allows for a RAID1 with WRITE-BEHIND to be used to mirror data over a slow link to a remote computer (providing the link isn't too slow). The extra latency of the remote link will not slow down normal operations, but the remote system will still have a reasonably up-to-date copy of all data.

FAILFAST

From Linux 4.10, md supports FAILFAST for RAID1 and RAID10 arrays. This is a flag that can be set on individual drives, though it is usually set on all drives, or no drives.

When md sends an I/O request to a drive that is marked as FAILFAST, and when the array could survive the loss of that drive without losing data, md will request that the underlying device does not perform any retries. This means that a failure will be reported to md promptly, and it can mark the device as faulty and continue using the other device(s). md cannot control the timeout that the underlying devices use to determine failure. Any changes desired to that timeout must be set explicitly on the underlying device, separately from using mdadm.

If a FAILFAST request does fail, and if it is still safe to mark the device as faulty without data loss, that will be done and the array will continue functioning on a reduced number of devices. If it is not possible to safely mark the device as faulty, md will retry the request

without disabling retries in the underlying device. In any case, md will not attempt to repair read errors on a device marked as FAILFAST by writing out the correct. It will just mark the device as faulty.

FAILFAST is appropriate for storage arrays that have a low probability of true failure, but will sometimes introduce unacceptable delays to I/O requests while performing internal maintenance. The value of setting FAILFAST involves a trade-off. The gain is that the chance of unacceptable delays is substantially reduced. The cost is that the unlikely event of data-loss on one device is slightly more likely to result in data-loss for the array.

When a device in an array using FAILFAST is marked as faulty, it will usually become usable again in a short while. mdadm makes no attempt to detect that possibility. Some separate mechanism, tuned to the specific details of the expected failure modes, needs to be created to monitor devices to see when they return to full functionality, and to then re-add them to the array. In order of this "re-add" functionality to be effective, an array using FAILFAST should always have a write-intent bitmap.

RESTRIPING

Restriping, also known as Reshaping, is the processes of re-arranging the data stored in each stripe into a new layout. This might involve changing the number of devices in the array (so the stripes are wider), changing the chunk size (so stripes are deeper or shallower), or changing the arrangement of data and parity (possibly changing the RAID level, e.g. 1 to 5 or 5 to 6).

As of Linux 2.6.35, md can reshape a RAID4, RAID5, or RAID6 array to have a different number of devices (more or fewer) and to have a different layout or chunk size. It can also convert between these different RAID levels. It can also convert between RAID0 and RAID10, and between RAID0 and RAID4 or RAID5. Other possibilities may follow in future kernels.

During any stripe process there is a 'critical section' during which live data is being overwritten on disk. For the operation of increasing the number of drives in a RAID5, this critical section covers the first few stripes (the number being the product of the old and new number of devices). After this critical section is passed, data is only written to areas of the array which no longer hold live data – the live data has already been located away.

For a reshape which reduces the number of devices, the 'critical section' is at the end of the reshape process.

md is not able to ensure data preservation if there is a crash (e.g. power failure) during the critical section. If md is asked to start an

array which failed during a critical section of restriping, it will fail to start the array.

To deal with this possibility, a user-space program must

- Disable writes to that section of the array (using the sysfs interface),
- take a copy of the data somewhere (i.e. make a backup),
- allow the process to continue and invalidate the backup and restore write access once the critical section is passed, and
- provide for restoring the critical data before restarting the array after a system crash.

mdadm versions from 2.4 do this for growing a RAID5 array.

For operations that do not change the size of the array, like simply increasing chunk size, or converting RAID5 to RAID6 with one extra device, the entire process is the critical section. In this case, the restripe will need to progress in stages, as a section is suspended, backed up, restriped, and released.

SYSFS INTERFACE

Each block device appears as a directory in sysfs (which is usually mounted at /sys). For MD devices, this directory will contain a subdirectory called md which contains various files for providing access to information about the array.

This interface is documented more fully in the file Documentation/admin-guide/md.rst which is distributed with the kernel sources. That file should be consulted for full documentation. The following are just a selection of attribute files that are available.

md/sync_speed_min

This value, if set, overrides the system-wide setting in /proc/sys/dev/raid/speed_limit_min for this array only. Writing the value system to this file will cause the system-wide setting to have effect.

md/sync_speed_max

This is the partner of md/sync_speed_min and overrides /proc/sys/dev/raid/speed_limit_max described below.

md/sync_action

This can be used to monitor and control the resync/recovery process of MD. In particular, writing "check" here will cause

the array to read all data block and check that they are consistent (e.g. parity is correct, or all mirror replicas are the same). Any discrepancies found are NOT corrected.

A count of problems found will be stored in md/mismatch_count.

Alternately, "repair" can be written which will cause the same check to be performed, but any errors will be corrected.

Finally, "idle" can be written to stop the check/repair process.

md/stripe_cache_size

This is only available on RAID5 and RAID6. It records the size (in pages per device) of the stripe cache which is used for synchronising all write operations to the array and all read operations if the array is degraded. The default is 256. Valid values are 17 to 32768. Increasing this number can increase performance in some situations, at some cost in system memory. Note, setting this value too high can result in an "out of memory" condition for the system.

memory_consumed = system_page_size * nr_disks * stripe_cache_size

md/preread_bypass_threshold

This is only available on RAID5 and RAID6. This variable sets the number of times MD will service a full-stripe-write before servicing a stripe that requires some "prereading". For fairness this defaults to 1. Valid values are 0 to stripe_cache_size. Setting this to 0 maximizes sequential-write throughput at the cost of fairness to threads doing small or random writes.

md/bitmap/backlog

The value stored in the file only has any effect on RAID1 when write-mostly devices are active, and write requests to those devices are proceed in the background.

This variable sets a limit on the number of concurrent background writes, the valid values are 0 to 16383, 0 means that write-behind is not allowed, while any other number means it can happen. If there are more write requests than the number, new writes will by synchronous.

md/bitmap/can_clear

This is for externally managed bitmaps, where the kernel writes the bitmap itself, but metadata describing the bitmap is managed by mdmon or similar.

When the array is degraded, bits mustn't be cleared. When the array becomes optimal again, bit can be cleared, but first the metadata needs to record the current event count. So md sets this to 'false' and notifies mdmon, then mdmon updates the metadata and writes 'true'.

There is no code in mdmon to actually do this, so maybe it doesn't even work.

md/bitmap/chunksize

The bitmap chunksize can only be changed when no bitmap is active, and the value should be power of 2 and at least 512.

md/bitmap/location

This indicates where the write-intent bitmap for the array is stored. It can be "none" or "file" or a signed offset from the array metadata - measured in sectors. You cannot set a file by writing here - that can only be done with the SET_BITMAP_FILE ioctl.

Write 'none' to 'bitmap/location' will clear bitmap, and the previous location value must be write to it to restore bitmap.

md/bitmap/max_backlog_used

This keeps track of the maximum number of concurrent write-behind requests for an md array, writing any value to this file will clear it.

md/bitmap/metadata

This can be 'internal' or 'clustered' or 'external'. 'internal' is set by default, which means the metadata for bitmap is stored in the first 256 bytes of the bitmap space. 'clustered' means separate bitmap metadata are used for each cluster node. 'external' means that bitmap metadata is managed externally to the kernel.

md/bitmap/space

This shows the space (in sectors) which is available at md/bitmap/location, and allows the kernel to know when it is safe to resize the bitmap to match a resized array. It should big enough to contain the total bytes in the bitmap.

For 1.0 metadata, assume we can use up to the superblock if before, else to 4K beyond superblock. For other metadata versions, assume no change is possible.

md/bitmap/time_base

This shows the time (in seconds) between disk flushes, and is used to looking for bits in the bitmap to be cleared.

The default value is 5 seconds, and it should be an unsigned long value.

KERNEL PARAMETERS

The md driver recognised several different kernel parameters.

raid=noautodetect

This will disable the normal detection of md arrays that happens at boot time. If a drive is partitioned with MS-DOS style partitions, then if any of the 4 main partitions has a partition type of 0xFD, then that partition will normally be inspected to see if it is part of an MD array, and if any full arrays are found, they are started. This kernel parameter disables this behaviour.

raid=partitionable

raid=part

These are available in 2.6 and later kernels only. They indicate that autodetected MD arrays should be created as partitionable arrays, with a different major device number to the original non-partitionable md arrays. The device number is listed as mdp in /proc/devices.

md_mod.start_ro=1

/sys/module/md_mod/parameters/start_ro

This tells md to start all arrays in read-only mode. This is a soft read-only that will automatically switch to read-write on the first write request. However until that write request, nothing is written to any device by md, and in particular, no resync or recovery operation is started.

md_mod.start_dirty_degraded=1

/sys/module/md_mod/parameters/start_dirty_degraded

As mentioned above, md will not normally start a RAID4, RAID5, or RAID6 that is both dirty and degraded as this situation can imply hidden data loss. This can be awkward if the root filesystem is affected. Using this module parameter allows such arrays to be started at boot time. It should be understood that there is a real (though small) risk of data corruption in this situation.

md=n,dev,dev,...

md=dn,dev,dev,...
This tells the md driver to assemble /dev/md n from the listed devices. It is only necessary to start the device holding the root filesystem this way. Other arrays are best started once the system is booted.

In 2.6 kernels, the d immediately after the = indicates that a partitionable device (e.g. /dev/md/d0) should be created rather than the original non-partitionable device.

md=n,l,c,i,dev...
This tells the md driver to assemble a legacy RAID0 or LINEAR array without a superblock. n gives the md device number, l gives the level, 0 for RAID0 or -1 for LINEAR, c gives the chunk size as a base-2 logarithm offset by twelve, so 0 means 4K, 1 means 8K. i is ignored (legacy support).

FILES

/proc/mdstat
Contains information about the status of currently running array.

/proc/sys/dev/raid/speed_limit_min
A readable and writable file that reflects the current "goal" rebuild speed for times when non-rebuild activity is current on an array. The speed is in Kibibytes per second, and is a per-device rate, not a per-array rate (which means that an array with more disks will shuffle more data for a given speed). The default is 1000.

/proc/sys/dev/raid/speed_limit_max
A readable and writable file that reflects the current "goal" rebuild speed for times when no non-rebuild activity is current on an array. The default is 200,000.

SEE ALSO

mdadm(8),

MD(4)

منابع

https://novablog.ir/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%D9%86%D8%B5%D8%A8-%D8%B3%DB%8C%D8%B3%D8%AA%D9%85-%D8%B9%D8%A7%D9%85%D9%84-ubuntu-14-04/

https://digiato.com/article/2022/09/28/install-linux-on-windows

https://linuxlearn.org/raid-in-linux/

https://en.wikipedia.org/wiki/Hard_disk_drive_failure

https://www.techtarget.com/searchdatabackup/tip/RAID-5-vs-RAID-6-Capacity-performance-durability#:~:text=The%20primary%20difference%20between%20RAID,during%20the%20disk%20rebuilding%20process

https://www.digitalocean.com/community/tutorials/how-to-create-raid-arrays-with-mdadm-on-ubuntu-18-04

Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program.

بیشتر برنامه نویسان خوب برنامه نویسی می کنند نه به این دلیل که انتظار دارند دستمزد دریافت کنند

یا از طرف مردم تحسین شوند، بلکه به این دلیل که برنامه نویسی سرگرم کننده است •

Linus Torvalds

creator of the Linux OS