

Choosing the right Linux File System Layout using a Top-Bottom Process.

Pierre Vignéras
pierre@vignerass.name



August 1, 2009

Abstract

As you may probably know, Linux supports various filesystems such as ext2, ext3, ext4, xfs, reiserfs, jfs among others. Few users really consider this part of a system, selecting default options of their distribution's installer. In this article, I will give some reasons for a better consideration of the file-system and of its layout. I will suggest a top-bottom process for the design of a “smart” layout that remains as stable as possible over time for a given computer usage.

Copyrights

This document is licensed under a *Creative Commons Attribution-Share Alike 2.0 France License*. Please, see for details: <http://creativecommons.org/licenses/by-sa/2.0/>

Disclaimer

The information contained in this document is for general information purposes only. The information is provided by Pierre Vignéras and while I endeavor to keep the information up to date and correct, I make no representations or warranties of any kind, express or implied, about the completeness, accuracy, reliability, suitability or availability with respect to the document or the information, products, services, or related graphics contained in the document for any purpose.

Any reliance you place on such information is therefore strictly at your own risk. In no event I will be liable for any loss or damage including without limitation, indirect or consequential loss or damage, or any loss or damage whatsoever arising from loss of data or profits arising out of, or in connection with, the use of this document.

Through this document you are able to link to other documents which are not under the control of Pierre Vignéras. I have no control over the nature, content and availability of those sites. The inclusion of any links does not necessarily imply a recommendation or endorse the views expressed within them.

Contents

1	Introduction	3
2	Default Layout	4
2.1	Lack of Flexibility	4
2.2	Lack of Performance.	4
3	Choosing the right thing [...]	4
3.1	Purchasing the right hardware	5
3.2	Defining usage pattern	5
3.3	Defining mount points	6
3.3.1	Temporary data: tmp.lv	7
3.3.2	Mostly read data: read.lv	7
3.3.3	Mostly written data: write.lv	7
3.3.4	Mostly append data: append.lv	8
3.3.5	Multimedia data: mm.lv	8
3.3.6	Others	8
3.4	Defining file-systems for logical volumes	8
3.5	Defining Volume Group (VG)	9
3.6	Defining Physical Volumes (PV)	10
3.7	Defining partitions	11
4	/boot	11
4.1	Swap	11
5	Future and/or exotic file-systems	12
6	USB drives	12
7	Solid State Drives	12
8	Conclusion	13
9	Layout Examples	13
9.1	Any usage, 1 disk.	13
9.2	Desktop usage, high availability, 2 disks.	13
9.3	Desktop usage, high performance, 2 disks	14
9.4	File server, 4 disks.	14
10	Questions, Comments & Suggestions	15
11	Note	15

1 Introduction

The first question that you may ask is why are there so many file-systems, and what are their differences if any? To make it short (see wikipedia for details):

- ext2: it is THE Linux fs, I mean, the one that was specifically designed for linux (influenced by ext and Berkeley FFS). Pro: fast; Cons: not journalized (long fsck).
- ext3: the natural ext2 extension. Pro: compatible with ext2, journalized; Cons: slower than ext2, as many competitors, obsolete today.
- ext4: the last extension of the ext family. Pro: ascending-compatibility with ext3, big size; good read performance; cons: a bit too recent to know?
- jfs: IBM AIX FS ported to Linux. Pro: mature, fast, light and reliable, big size; Cons: still developed?
- xfs: SGI IRIX FS ported to Linux. Pro: very mature and reliable, good average performance, big size, many tools (such as a defragmenter); Cons: none as far as I know.
- reiserfs: alternative to ext2/3 file-system on linux. Pro: fast for small files; Cons: still developed?

There are other file-systems, in particular new ones such as btrfs, zfs and nilfs2 that may sound very interesting too. We will deal with them later on in this article (see 5).

So now the question is: which file-system is the most suitable for your particular situation? The answer is not simple. But if you don't really know, if you have any doubt, I would recommend XFS for various reasons:

1. it performs very well in general and particularly on concurrent read/write (see <http://www.debian-administration.org/articles/388> for a benchmark);
2. it is very mature and therefore has been tested and tuned extensively;
3. most of all, it comes with great features such as `xfs_fsr`, an easy to use defragmenter (just do `ln -sf $(which xfs_fsr) /etc/cron.daily/defrag` and forget about it).

The only problem I see with XFS, is that you cannot reduce an XFS fs. You can grow an XFS partition even when mounted and in active use (hot-grow), but you cannot reduce its size. Therefore, if you have some reducing file-system needs choose another file system such as ext2/3/4 or reiserfs (as far as I know you cannot hot-reduce neither ext3 nor reiserfs file-systems anyway).

Another option is to keep XFS and to always start with small partition size (as you can always hot-grow afterwards).

If you have a low profile computer (or file server) and if you really need your CPU for something else than dealing with input/output operations, then I would suggest JFS.

If you have many directories or/and small files, reiserfs may be an option.

If you need performance at all cost, I would suggest ext2.

Honestly, I don't see any reason for choosing ext3/4 (performance? really?).

That is for file-system choice. But then, the other question is which layout should I use? Two partitions? Three? Dedicated /home/? Read-only /? Separate /tmp?

Obviously, there is no single answer to this question. Many factors should be considered in order to make a good choice. I will first define those factors:

Complexity: how complex the layout is globally;

Flexibility: how easy it is to change the layout;

Performance: how fast the layout allows the system to run.

Finding the perfect layout is a trade-off between those factors.

2 Default Layout

Often, a desktop end-user with few knowledge of Linux will follow default settings of his distribution where (usually) only two or three partitions are made for Linux, with the root file-system '/', /boot and the swap. Advantages of such a configuration is simplicity. Main problem is that this layout is neither flexible nor performant.

2.1 Lack of Flexibility

Lack of flexibility is obvious for many reasons. First, if the end-user wants another layout (for example he wants to resize the root file-system, or he wants to use a separate /tmp file-system), he will have to reboot the system and to use a partitioning software (from a livecd for example). He will have to take care of his data since re-partitioning is a brute-force operation the operating system is not aware of.

Also, if the end-user wants to add some storage (for example a new hard drive), he will end up modifying the system layout (/etc/fstab) and after some while, his system will just depend on the underlying storage layout (number, and location of hard drives, partitions, and so on).

By the way, having separate partitions for your data (/home but also all audio, video, database, ...) makes much easier the changing of the system (for example from one Linux distribution to another). It makes also the sharing of data between operating systems (BSD, OpenSolaris, Linux and even Windows) easier and safer. But this is another story.

A good option is to use Logical Volume Management (LVM). LVM solves the flexibility problem in a very nice way, as we will see. The good news is that most modern distributions support LVM and some use it by default. LVM adds an abstraction layer on top of the hardware removing hard dependencies between the OS (/etc/fstab) and the underlying storage devices (/dev/hda, /dev/sda, and others). This means that you may change the layout of storage — adding and removing hard drives — without disturbing your system. The main problem of LVM, as far as I know, is that you may have trouble reading an LVM volume from other operating systems.

2.2 Lack of Performance.

Whatever file-system is used (ext2/3/4, xfs, reiserfs, jfs), it is not perfect for all sort of data and usage patterns (aka workload). For example, XFS is known to be good in the handling of big files such as video files. On the other side, reiserfs is known to be efficient in the handling of small files (such as configuration files in your home directory or in /etc). Therefore having one file-system for all sort of data and usage is definitely not optimal. The only good point with this layout is that the kernel does not need to support many different file-systems, thus, it reduces the amount of memory the kernel uses to its bare minimum (this is also true with modules). But unless we focus on embedded systems, I consider this argument as irrelevant with today computers.

3 Choosing the right thing : a top-bottom approach

Often, when a system is designed, it is usually done in a bottom to top approach: hardware is purchased according to criteria that are not related to their usage. Thereafter, a file-system layout is defined according to that hardware: "I have one disk, I may partition it this way, this partition will appear there, that other one there, and so on".

I propose the reverse approach. We define what we want at a high level. Then we travel layers top to bottom, down to real hardware — storage devices in our case — as shown on Illustration 1. This illustration is just an example of what can be done. There are many options as we will see. Next sections will explain how we can come to such a global layout.

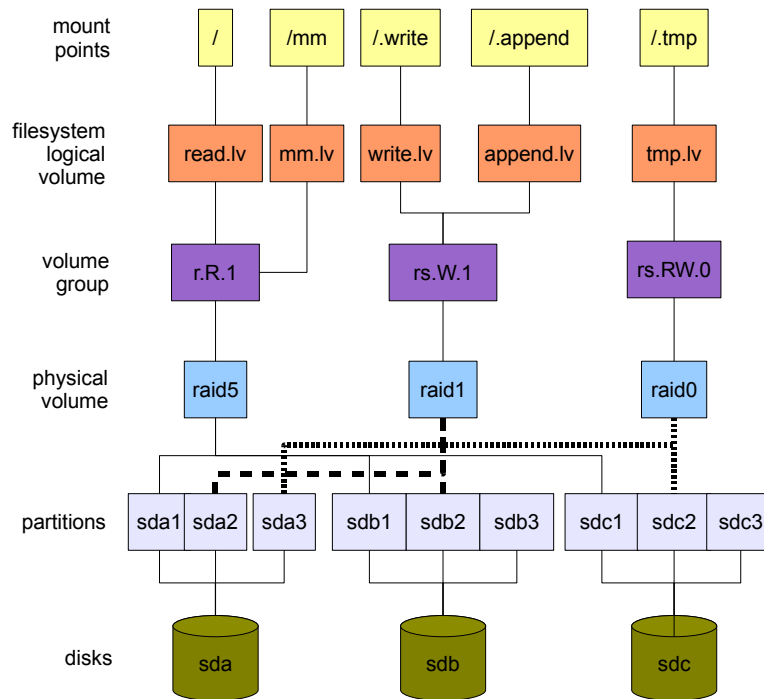


Figure 1: An example of a file-system layout. Notice that two partitions remain free (sdb3 and sdc3). They may be used for /boot, for the swap or both. Do not "copy/paste" this layout. It is not optimized for your workload. It is just an example.

3.1 Purchasing the right hardware

Before installing a new system, the target usage should be considered. First from a hardware point of view. Is it an embedded system, a desktop, a server, an all-purpose multi-user computer (with TV/Audio/Video/OpenOffice/Web/Chat/P2P, ...)?

As an example, I always recommend end-users with simple desktop needs (web, mail, chat, few media watching) to purchase a low cost processor (the cheapest one), plenty of RAM (the maximum) and at least two hard drives.

Nowadays, even the cheapest processor is far enough for web surfing and movie watching. Plenty of RAM gives good cache (linux uses free memory for caching — reducing the amount of costly input/output to storage devices). By the way, purchasing the maximum amount of RAM your motherboard can support is an investment for two reasons:

1. applications tend to require more and more memory; therefore having the maximum amount of memory already prevent you from adding memory later on for a while;
2. technology changes so quickly that your system may not support the memory available in 5 years. At that time, purchasing old memory will probably be quite expensive.

Having two hard drives allows them to be used in mirror. Therefore, if one fails, the system will continue to work normally and you will have time to get a new hard drive. This way, your system will remain available and your data, quite safe (this is not sufficient, backup your data also).

3.2 Defining usage pattern

When choosing hardware, and specifically the file-system layout you should consider applications that will use it. Different applications have different input/output workload. Consider the following applications: loggers (syslog), mail readers (thunderbird, kmail), search engine (beagle), database

(mysql, postgresql), p2p (emule, gnutella, vuze), shells (bash)... Can you see their input/output patterns and how much they differ?

Therefore, I define the following abstract storage location known as logical volume — lv — in the LVM terminology:

tmp.lv: for temporary data such as the one found in /tmp, /var/tmp and also in the home directory of each users \$HOME/tmp (note that Trash directories such as \$HOME/Trash, \$HOME/.Trash may also be mapped here. Please see Freedesktop Trash Specification at <http://www.freedesktop.org/wiki/Specifications/trash-spec> for implications). Another candidate is /var/cache. The idea for this logical volume is that we may over-tune it for performance and we may accept somewhat data loss since these data are not essential for the system (see Linux File-system Hierarchy Standard (FHS) at <http://www.pathname.com/fhs/> <http://www.pathname.com/fhs/> for details on those locations).

read.lv: for data that are mostly read as for most binary file in /bin, /usr/bin, /lib, /usr/lib, configuration files in /etc and most configuration files in each user directory \$HOME/.bashrc, and so on. This storage location can be tuned for read-performance. We may accept poor write performance since they occur on rare occasions (e.g: when upgrading the system). Loosing data here is clearly unacceptable.

write.lv: for data that are mostly written in a randomly manner such as data written by P2P applications, or databases. We can tune it for write performance. Note that read performance cannot be too low: both P2P and database applications read randomly and quite often the data they write. We may consider this location as the "all-purpose" location: if you don't really know the usage pattern of a given application, configure it so it uses this logical volume. Loosing data here is also unacceptable.

append.lv: for data that are mostly written in a sequential manner as for most files in /var/log and also \$HOME/.xsession-errors among others. We can tune it for append performance which may be quite different than random write performance. There, read performance is usually not that important (unless you have specific needs of course). Loosing data here is unacceptable for normal uses (log gives information on problems. If you loose your logs, how can you know what was the problem?).

mm.lv: for multimedia files; their case is a bit special in that they are usually big (video) and read sequentially. Tuning for sequential read can be done here. Multimedia files are written once (for example from the write.lv where P2P applications write to the mm.lv), and read many times sequentially.

You may add/suggest any other categories here with different patterns such as sequential.read.lv, for example.

3.3 Defining mount points

Let's suppose that we already have all those storage abstract locations in the form of /dev/TBD/LV where:

- TBD is a volume group To Be Defined later (see 3.5);
- LV is one of the logical volume we just defined in the preceding section (read.lv, tmp.lv, ...).

So we suppose we already have /dev/TBD/tmp.lv, /dev/TBD/read.lv, /dev/TBD/write.lv, and so on.

By the way, we consider that each volume group is optimized for its usage pattern (a trade-off have been found between performance and flexibility).

3.3.1 Temporary data: tmp.lv

We would like to have /tmp, /var/tmp, and any \$HOME/tmp all mapped to /dev/TBD/tmp.lv.

What I suggest is the following:

1. mount /dev/TBD/tmp.lv to a /.tmp hidden directory at the root level; In /etc/fstab, you will have something like that (of course, since the volume group is unknown, this will not work; the point is to explain the process here.):


```
# Replace auto by the real file-system if you want
# Replace defaults 0 2 by your own needs (man fstab)
/dev/TBD/tmp.lv /.tmp auto defaults 0 2
```
2. bind other locations to directory in /.tmp. For example, suppose you don't care having separate directories for /tmp and /var/tmp (see FHS for implications), you can just create an ALL_TMP directory inside /dev/TBD/tmp.lv and bind it to both /tmp and /var/tmp. In /etc/fstab, add those lines:

```
/.tmp/ALL_TMP /tmp none bind 0 0
/.tmp/ALL_TMP /var/tmp none bind 0 0
```

Of course if you prefer to conform to FHS, no problem. Create two distinct directories FHS_TMP and FHS_VAR_TMP into the tmp.lv volume, and add those lines:

```
/.tmp/FHS_TMP /tmp none bind 0 0
/.tmp/FHS_VAR_TMP /var/tmp none bind 0 0
```

3. make a symlink for user tmp directory to /tmp/user. For example, \$HOME/tmp is a symbolic link to /tmp/\$USER_NAME/tmp (I am using the KDE environment, therefore, my \$HOME/tmp is a symbolic link to /tmp/kde-\$USER so all KDE applications use the same lv). You may automate this process using some lines into your .bash_profile (or even in the /etc/skel/.bash_profile so any new user will have it). For example:

```
if test ! -e $HOME/tmp -a ! -e /tmp/kde-$USER; then
    mkdir /tmp/kde-$USER;
    ln -s /tmp/kde-$USER $HOME/tmp;
fi
```

(This script is rather simple and only works in the case where both \$HOME/tmp and /tmp/kde-\$USER does not already exist. You may adapt it to your own need.)

3.3.2 Mostly read data: read.lv

Since the root file-system contains /etc, /bin, /usr/bin and so on, they are perfect for read.lv. Therefore, in /etc/fstab I would place the following:

```
/dev/TBD/read.lv / auto defaults 0 1
```

For configuration files in user home directories things are not so simple as you may guess. One may try to use the XDG_CONFIG_HOME environment variable (see FreeDesktop at <http://standards.freedesktop.org/basedir-spec/basedir-spec-0.6.html>).

But I would not recommend this solution for two reasons. First, few applications actually conforms to it nowadays (default location is \$HOME/.config when not set explicitly). Second, is that if you set XDG_CONFIG_HOME to a read.lv sub-directory, end users will have trouble in finding their configuration files. Therefore, for that case, I don't have any good solution and I will make home directories and all config files stored to the general write.lv location.

3.3.3 Mostly written data: write.lv

For that case, I will reproduce some way the pattern used for tmp.lv. I will bind different directories for different applications. For example, I will have in the fstab something similar to this:

```
/dev/TBD/write.lv /.write auto defaults 0 2
.write/db /db none bind 0 0
.write/p2p /p2p none bind 0 0
.write/home /home none bind 0 0
```

Of course, this suppose that db and p2p directories have been created in write.lv.

Note that you may have to be aware of rights access. One option is to provide the same rights than for /tmp where anyone can write/read their own data. This is achieved by the following command for example: `chmod 1777 /p2p`.

3.3.4 Mostly append data: `append.lv`

That volume has been tuned for loggers style applications such as syslog (and its variants `syslog_ng` for example), and any other loggers (Java loggers for example). The `/etc/fstab` should be similar to this:

```
/dev/TBD/append.lv /.append auto defaults 0 2
/.append/syslog /var/log none bind 0 0
/.append/ulog /var/ulog none bind 0 0
```

Again, `syslog` and `ulog` are directories previously created into `append.lv`.

3.3.5 Multimedia data: `mm.lv`

For multimedia files, I just add the following line:

```
/dev/TBD/mm.lv /mm auto defaults 0 2
```

Inside `/mm`, I create `Photos`, `Audios` and `Videos` directories. As a desktop user, I usually share my multimedia files with other family members. Therefore, access rights should be correctly designed.

You may prefer having distinct volumes for photo, audio and video files. Feel free to create logical volumes accordingly: `photos.lv`, `audios.lv` and `videos.lv`.

3.3.6 Others

You may add your own logical volumes according to your need. Logical volumes are quite free to deal with. They do not add a big overhead and they provide a lot of flexibility helping you to take out the most of your system particularly when choosing the right file-system for your workload.

3.4 Defining file-systems for logical volumes

Now that our mount points and our logical volumes have been defined according to our application usage patterns, we may choose the file-system for each logical volumes. And here we have many choices as we have already seen. First of all, you have the file-system itself (e.g: `ext2`, `ext3`, `ext4`, `reiserfs`, `xfs`, `jfs` and so on). For each of them you also have their tuning parameters (such as tuning block size, number of inodes, log options (XFS), and so on). Finally, when mounting you may also specify different options according to some usage pattern (`noatime`, `data=writeback` (`ext3`), `barrier` (XFS), and so on). File-system documentation should be read and understood so you can map options to the correct usage pattern. If you don't have any idea on which fs to use for which purpose, here are my suggestions:

tmp.lv: this volume will contain many sort of data, written/read by applications and users, small and big. Without any defined usage pattern (mostly read, mostly write), I would use a generic file-system such as XFS or `ext4`.

read.lv: this volume contains the root file-system with many binaries (`/bin`, `/usr/bin`), libraries (`/lib`, `/usr/lib`), many configurations files (`/etc`)... Since most of its data are read, the file-system may be the one with the best read performance even if its write performance is poor. XFS or `ext4` are options here.

write.lv: this is rather difficult since this location is the "fit all" location, it should handle both read and write correctly. Again, XFS or `ext4` are options too.

append.lv: there, we may choose a pure log structured file-system such as the new NILFS2 supported by linux since 2.6.30 which should provide very good write performance (but beware of its limitations (especially, no support for atime, extended attributes and ACL), see documentation at <http://www.nilfs.org/en/>).

mm.lv: contains audio/video files that can be quite big. This is a perfect choice for XFS. Note that on IRIX, XFS supports a real-time section for multimedia applications. This is not supported (yet?) under Linux as far as I know.

You may play with XFS tuning parameters (see man xfs), but it requires some good knowledge on your usage pattern and on XFS internals.

At that high level, you may also decide if you need encryption or compression support. This may help in choosing the file-system. For example, for mm.lv, compression is useless (as multimedia data are already compressed) whereas it may sound useful for /home. Consider also if you need encryption.

At that step we have chosen the file-systems for all of our logical volumes. Time is now to go down to the next layer and to define our volume groups.

3.5 Defining Volume Group (VG)

Next step is to define volume groups. At that level, we will define our needs in term of performance tuning and fault tolerance. I propose defining VGs according to the following schema: $[r|s] \cdot [R|W] \cdot [n]$ where:

'r' stands for random;

's' stands for sequential;

'R' stands for read;

'W' stands for write;

'n' is a positive integer, zero inclusive.

Letters determine the type of optimization the named volume has been tuned for. The number gives an abstract representation of the fault tolerance level. For example:

- r.R.0 means optimized for random read with a fault tolerance level of 0 : data loss occurs as soon as one storage device fails (said otherwise, the system is tolerant to 0 storage device failure).
- s.W.2 means optimized for sequential write with a fault tolerance level of 2 : data loss occurs as soon as three storage device fail (said otherwise, the system is tolerant to 2 storage devices failure).

We then have to map each logical volume to a given volume group. I suggest the following:

tmp.lv: can be mapped to an rs.RW.0 volume group or an rs.RW.1 depending on your requirements concerning fault tolerance. Obviously, if your desire is that your system remains on-line on a 24/24 hours basis, 365 days/year, the second option should definitely be considered. Unfortunately, fault tolerance has a cost both in terms of storage space and performance. Therefore, you should not expect the same level of performance from an rs.RW.0 vg and an rs.RW.1 vg with the same number of storage devices. But if you can afford the prices, there are solutions for quite performant rs.RW.1 and even rs.RW.2, 3 and more! More on that at next down level.

read.lv: may be mapped to an r.R.1 vg (increase fault tolerant number if you require);

write.lv: may be mapped to an r.W.1 vg (same thing);

append.lv: may be mapped to an s.W.1 vg;

mm.lv: may be mapped to an s.R.1 vg.

Of course, we have a 'may' and not a 'must' statement as it depends on the number of storage devices that you can put into the equation. Defining VG is actually quite difficult since you cannot always really abstract completely the underlying hardware. But I believe that defining your requirements first may help in defining the layout of your storage system globally.

We will see at the next level, how to implement those volume groups.

3.6 Defining Physical Volumes (PV)

That level is where you actually implements a given volume group requirements (defined using the notation rs.RW.n described above). Hopefully, there are not — as far as I know — many ways in implementing a vg requirement. You may use some of LVM features (mirroring, striping), software RAID (with linux MD), or hardware RAID. The choice depends on your needs and on your hardware. However, I would not recommend hardware RAID (nowadays) for a desktop computer or even a small file server, for two reasons:

- quite often (most of the time actually), what is called hardware raid, is actually software raid: you have a chipset on your motherboard that presents a low cost RAID controller that requires some software (drivers) to do the actual work. Definitely, Linux RAID (md) is far better both in terms of performance (I think), and in terms of flexibility (for sure).
- unless you have a very old CPU (pentium II class), Soft RAID is not so costly (this is not so true for RAID5 actually, but for RAID0, RAID1, and RAID10, it is true).

So if you don't have any idea on how to implement a given specification using RAID, please, see RAID documentation, for example the one at <http://www.storagereview.com/guide2000/ref/hdd/perf/raid/levels/comp.html>.

Some few hints however:

- anything with a .0 can be mapped to RAID0 which is the most performant RAID combination (but if one storage device fail, you loose everything).
- s.R.1, r.R.1 and sr.R.1 can be mapped in order of preferences to RAID10 (minimum of 4 storage devices (sd) required), RAID5 (3 sd required), RAID1 (2 sd).
- s.W.1, can be mapped in order of preferences to RAID10, RAID1 and RAID5.
- r.W.1, can be mapped in order of preferences to RAID10 and RAID1 (RAID5 has very poor performance in random write).
- sr.R.2 can be mapped to RAID10 (some ways) and to RAID6.

When you map storage space to a given physical volume, do not attach two storage spaces from the same storage device (i.e. partitions). You will lose both advantages of performance and fault tolerance! For example, making /dev/sda1 and /dev/sda2 part of the same RAID1 physical volume is quite useless.

Finally, if you are not sure what to choose between LVM and MDADM, I would suggest MDADM has it is a bit more flexible (it supports RAID0, 1, 5 and 10, whereas LVM only supports striping (similar to RAID0), and mirroring (similar to RAID1)).

Even if strictly not required, if you use MDADM, you will probably end up with a one-to-one mapping between VGs and PVs. Said otherwise, you may map many PVs to one VG. But this is a bit useless in my humble opinion. MDADM provides all the flexibility required in the mapping of partitions/storage devices into VG implementations.

3.7 Defining partitions

Finally, you may want to make some partitions out of your different storage devices in order to fulfill your PV requirements (for example, RAID5 requires at least 3 different storage spaces). Note that in the vast majority of cases, your partitions will have to be of the same size.

If you can, I would suggest to use directly storage devices (or to make only one single partition out of a disk). But it may be difficult if you are short in storage devices. Moreover, if you have storage devices of different sizes, you will have to partition one of them at least.

You may have to find some trade-off between your PV requirements and your available storage devices. For example, if you have only two hard drives, definitely you cannot implement a RAID5 PV. You will have to rely on a RAID1 implementation only.

Note that if you really follow the top-bottom process described in this document (and if you can afford the price of your requirements of course), there is no real trade-off to deal with! ;-)

4 /boot

We didn't mention in our study the /boot file-system where the boot-loader is stored. Some would prefer having only one single / where /boot is just a sub-directory. Others prefer to separate / and /boot. In our case, where we use LVM and MDADM, I would suggest the following idea:

1. /boot is a separate file-system because some boot-loader may have trouble with LVM volumes;
2. /boot is an ext2 or ext3 file-system since those format are well supported by any boot-loader;
3. /boot size would be 100 MB size because initramfs can be quite heavy and you may have several kernels with their own initramfs;
4. /boot is not an LVM volume;
5. /boot is a RAID1 volume (created using MDADM). This ensures that at least two storage devices have exactly the same contents composed of kernel, initramfs, System.map and other stuff required for booting;
6. The /boot RAID1 volume is made of two primary partitions that are the first partition on their respective disks. This prevents some old BIOS not finding the boot-loader due to the old 1GB limitations.
7. The boot loader has been installed on both partitions (disks) so the system can boot from both disks.
8. The BIOS has been configured properly to boot from any disk.

4.1 Swap

Swap is also a stuff we didn't discuss up to now. You have many options here:

performance: if you need performance at all cost, definitely, create one partition on each of your storage device, and use it as a swap partition. The kernel will balance input/output to each partition according to its own need leading to the best performance. Note that you may play with priority in order to give some preferences to given hard disks (for example, a fast drive can be given a higher priority).

fault-tolerance: if you need fault tolerance, definitely, consider the creation of an LVM swap volume from an r.RW.1 volume group (implemented by a RAID1 or RAID10 PV for example).

flexibility: if you need to resize your swap for some reasons, I suggest to use one or many LVM swap volumes.

5 Future and/or exotic file-systems

Using LVM it is quite easy to set up a new logical volume created from some volume group (depending on what you want to test and your hardware) and to format it to some file-systems. LVM is very flexible in this regard. Feel free to create and remove file-systems at will.

But in some ways, future file-systems such as ZFS, Btrfs and Nilfs2 will not fit perfectly with LVM. The reason is that LVM leads to a clear separation between application/user needs and implementations of this needs, as we have seen. On the other side, ZFS and Btrfs integrate both needs and implementation into one stuff. For example both ZFS and Btrfs supports RAID level directly. The good thing is that it ease the making of file-system layout. The bad thing is that it violates some ways the separation of concern strategy.

Therefore, you may end up with both an XFS/LV/VG/MD1/sd{a,b}1 and Btrfs/sd{a,b}2 inside the same system. I would not recommend such a layout and suggest to use ZFS or Btrfs for everything or not at all.

Another file-system that may be interesting is Nilfs2. This log structured file-systems will have very good write performance (but maybe poor read performance). Therefore, such a file-system may be a very good candidate for the append logical volume or on any logical volume created from an rs.W.n volume group.

6 USB drives

If you want to use one or several USB drives in your layout consider the following:

1. The bandwidth of the USB v2 bus is 480 Mbits/s (60 Mbytes/s) which is enough for the vast majority of desktop applications (except maybe HD Video);
2. As far as I know you will not find any USB device that can fulfill the USB v2 bandwidth.

Therefore, it may be interesting to use several USB drives (or even stick) to make them part of a RAID system, especially a RAID1 system. With such a layout, you can pull out one USB drive of a RAID1 array, and use it (in read-only mode) elsewhere. Then, you pull it in again in your original RAID1 array, and with a magic mdadm command such as:

```
mdadm /dev/md0 --add /dev/sda1
```

The array will reconstruct automagically and come back to its original state. I would not recommend making any other RAID array out of USB drive however. For RAID0, it is obvious: if you remove one USB drive, you loose all your data! For RAID5, having USB drive, and thus, the hot-plug capability does not offer any advantage: the USB drive you have pulled out is useless in a RAID5 mode! (same remark for RAID10).

7 Solid State Drives

Finally, new SSD drives may be considered while defining physical volumes. Their properties should be taken into account:

- They have very low latency (both read and write);
- They have very good random read performance and fragmentation has no impact on their performance (deterministic performance);
- Number of writes is limited.

Therefore SSD drives are suitable for implementing rsR#n volume groups. As an example, mm.lv and read.lv volumes can be stored on SSDs since data is usually written once and read many times. This usage pattern is perfect for SSD.

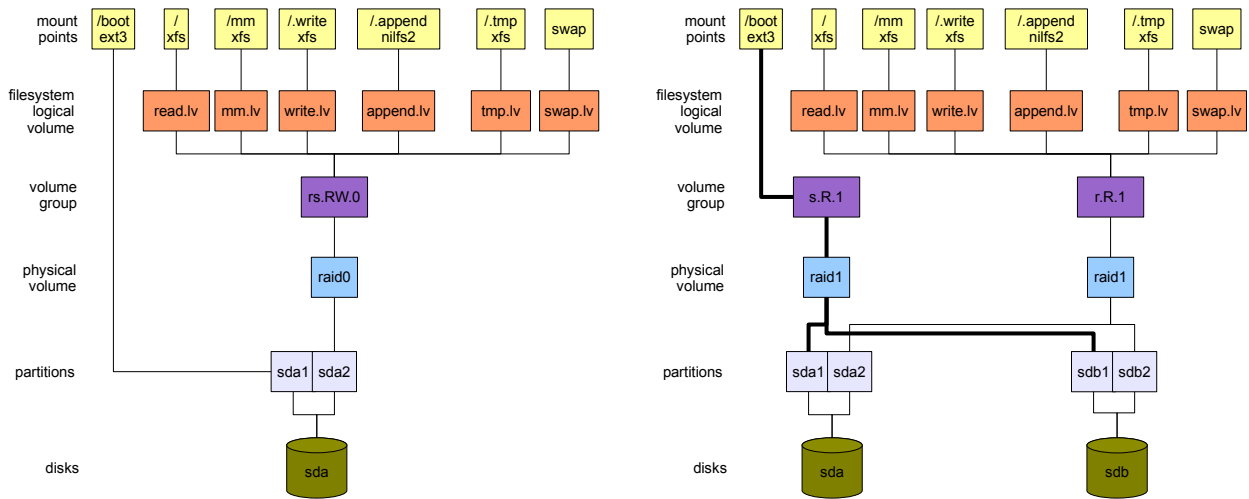


Figure 2: A layout with one disk (left) and one for desktop usage with two disks (right).

8 Conclusion

In the process of designing a file-system layout, the top-bottom approach starts with high level needs. This method has the advantage that you can rely on previously made requirements for similar systems. Only the implementation will change. For example, if you design a desktop system: you may end up with a given layout (such as the one in figure 1). If you install another desktop system with different storage devices, you can rely on your first requirements. You just have to adapt bottom layers: PVs and partitions. Therefore, the big work, usage pattern or workload, analysis can be done only one time per system, naturally.

In the next and final section, I will give some layout examples, roughly tuned for some well known computer usages.

9 Layout Examples

9.1 Any usage, 1 disk.

This (see the left layout of figure 2) is a rather strange situation in my opinion. As already said, I consider that any computer should be sized according to some usage pattern. And having only one disk attached to your system means that you accept a complete failure of it someday. But I know that the vast majority of computers today — especially laptops and netbooks — are sold (and designed) with only a single disk. Therefore, I propose the following layout which focuses on flexibility and performance (as much as possible):

flexibility: as the layout allows you to resize volumes at will;

performance: as you can choose a file-system (ext2/3, XFS, and so forth) according to data access patterns.

9.2 Desktop usage, high availability, 2 disks.

Here (see the right layout of figure 2), our concern is high availability. Since we have only two disks, only RAID1 can be used. This configuration provides:

flexibility: as the layout allows you to resize volumes at will;

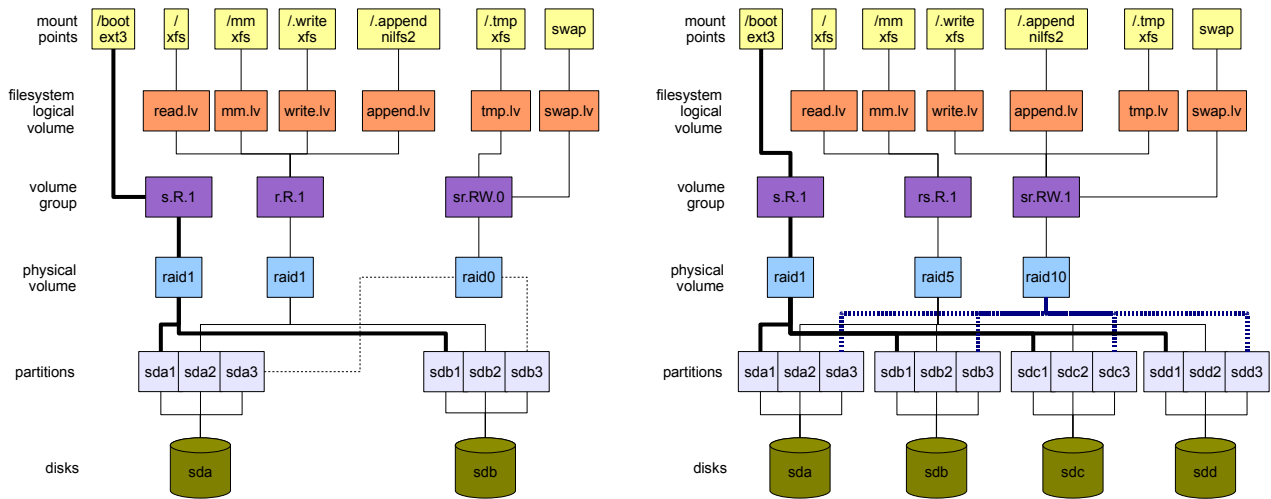


Figure 3: Left: Layout for high performance desktop usage with two disks.
Right: Layout for file server with four disks.

performance: as you can choose a file-system (ext2/3, XFS, and so forth) according to data access patterns and since an r.R.1 vg can be provided by a RAID1 pv for good random read performance (on average). Note however, that both s.R.n and rs.W.n cannot be provided with only 2 disks for any value of n.

High availability: if one disk fail, the system will continue working in a degraded mode.

Note *The swap region should be on the RAID1 PV in order to ensure high availability.*

9.3 Desktop usage, high performance, 2 disks

Here (see the left layout of figure 3), our concern is high performance. Note however that I still consider unacceptable to lose some data. This layout provides the following:

flexibility: as the layout allows you to resize volumes at will;

performance: as you can choose a file-system (ext2/3, XFS, and so forth) according to data access patterns, and since both r.R.1 and rs.RW.0 can be provided with 2 disks thanks to RAID1 and RAID0.

Medium availability: if one disk fail, important data will remain accessible but the system will not be able to work correctly unless some actions are taken to map /.tmp and swap to some other lv mapped to a safe vg.

Note *The swap region is made from the rs.RW.0 vg implemented by the RAID0 pv to ensure flexibility (resizing swap regions is painless). Another option is to use directly a fourth partition from both disks.*

9.4 File server, 4 disks.

Here (see the right layout of figure 3), our concern is both high performance and high availability. This layout provides the following:

flexibility: as the layout allows you to resize volumes at will;

performance: as you can choose a file-system (ext2/3, XFS, and so forth) according to data access patterns, and since both rs.R.1 and rs.RW.1 can be provided with 4 disks thanks to RAID5 and RAID10.

High availability: if one disk fail, any data will remain accessible and the system will be able to work correctly.

Note 1 *We may have used RAID10 for the whole system as it provides very good implementation of rs.RW.1 vg (and somehow also rs.RW.2). Unfortunately, this comes with a cost: 4 storage devices are required (here partitions), each of the same capacity S (let say S=500 Gigabytes). But the RAID10 physical volume does not provide a 4*S capacity (2 Terabytes) as you may expect. It only provides half of it, 2*S (1 Terabytes). The other 2*S (1 Terabytes) is used for high availability (mirror). See RAID documentation for details. Therefore, I choose to use RAID5 for implementing rs.R.1. RAID5 will provide 3*S capacity (1.5 Gigabytes), the remaining S (500 Gigabytes) is used for high availability. The mm.lv usually requires a big amount of storage space since it holds multimedia files.*

Note 2 *If you export through NFS or SMB 'home' directories, you may consider their location carefully. If your users need a lot of space, making homes on the write.lv (the 'fit-all' location) may be storage-expensive because it is backed by an RAID10 pv where half of the storage space is used for mirroring (and performance). You have two options here:*

1. *either, you have enough storage or/and your users have high random/sequential write access needs, the RAID10 pv is the good option;*
2. *or, you do not have enough storage or/and your users do not have high random/sequential write access needs, the RAID5 pv is the good option.*

10 Questions, Comments & Suggestions

If you have any question, comment, and/or suggestion on this document, feel free to contact me at the following address: pierre@vigneras.name.

11 Note

This article has been published in HTML format at <http://www.linuxconfig.org> (ISSN 1836-5930) thanks to Lubos Rendeck.