|  |
| --- |
| LAMP Arch Linux Apache MariaDB PHP |
| Installing LAMP  Draft |
| |  |  |  | | --- | --- | --- | | Nico De Witte | 2014-2015 | nico.dewitte@vives.be | |

Contents

[2 Linux Basics 2](#_Toc417912943)

[2.1 The MAN-Pages 2](#_Toc417912944)

[2.2 The Linux Filesystem 3](#_Toc417912945)

[2.2.1 Some Brief Notes on the History of the Linux Filesystem Layout 3](#_Toc417912946)

[2.2.2 Traversing the Filesystem 3](#_Toc417912947)

[2.2.3 An Overview of the Linux Filesystem Layout 5](#_Toc417912948)

[2.2.4 overview of Basic Filesystem commands 9](#_Toc417912949)

[2.3 Debian and it’s Packages 10](#_Toc417912950)

[3 Running a Virtual Machine 12](#_Toc417912951)

[3.1 Installing Oracle VirtualBox 12](#_Toc417912952)

[3.2 Creating a Virtual Machine 14](#_Toc417912953)

[3.3 Configuring the Virtual Machine 17](#_Toc417912954)

[3.4 Installing the Ubuntu Operating System 18](#_Toc417912955)

[3.4.1 Installing Guest Additions 20](#_Toc417912956)

[3.5 Creating a Shared Directory 20](#_Toc417912957)

[4 The LAMP Stack 22](#_Toc417912958)

[4.1 Updating 22](#_Toc417912959)

[4.2 Installing Apache WebServer and PHP Extensions 22](#_Toc417912960)

[4.2.1 Serving a Static Page 23](#_Toc417912961)

[4.3 Installing MySql 23](#_Toc417912962)

[4.4 Installing PHPMYAdmin 23](#_Toc417912963)

[5 Hosting a Website 25](#_Toc417912964)

[5.1 Installing GIT 25](#_Toc417912965)

[5.2 Setting Up SSH and Keys 25](#_Toc417912966)

[5.2.1 Installing SSH client and Server 25](#_Toc417912967)

[5.2.2 Setting up the Secret Key 25](#_Toc417912968)

[5.3 Cloning The Repository 26](#_Toc417912969)

# Linux Basics

While the information and examples presented here will be valid for most Linux distro’s, some information and commands may only work for Debian based distro’s.

## The MAN-Pages

The most important command you need to know is the “man” command, which provides an interface to the online reference manuals. By adding a command after the man command you can consult the man-pages for the particular command. These pages provide all the information you need to know to use the command such as general information, a detailed description of the arguments and usage examples. Let’s see an example:

$ man ls

Which gives the output shown in Figure 1. We can for example see if we add “-a” after the “ls” command it will also display hidden files[[1]](#footnote-1). You can scroll through the man-pages using the arrow keys.

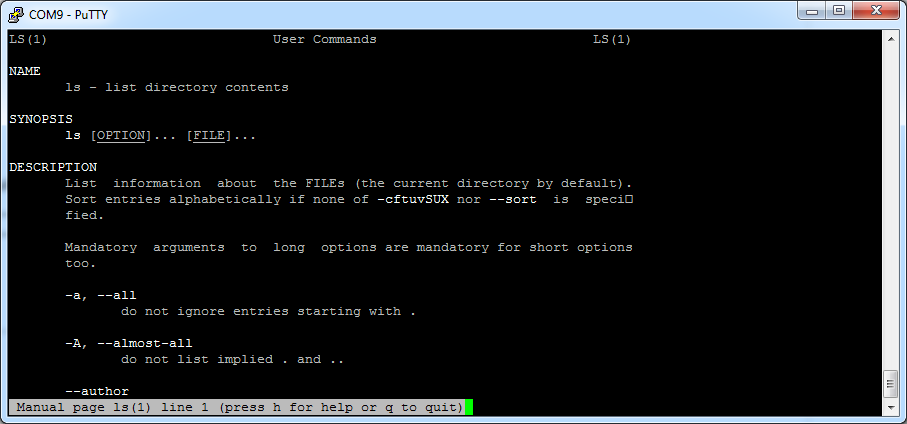


Figure 1 Output of "man ls" command

Searching the current man-page can be done by first typing a slash (“/”), followed by your search term. Jumping to the next hit can be done by hitting the “n” key, while jumping back is done with “SHIFT-n”.

Exiting the man-pages is achieved using the “CTRL-c” combination.

## The Linux Filesystem[[2]](#footnote-2)

### Some Brief Notes on the History of the Linux Filesystem Layout

Linux inherits many of its concepts of filesystem organization from its Unix predecessors. As far back as 1979, Unix was establishing standards to control how compliant systems would organize their files.

The Linux Filesystem Hierarchy Standard[[3]](#footnote-3), or FHS for short, is a prescriptive standard maintained by the Linux Foundation that establishes the organizational layout that Linux distributions should uphold for interoperability, ease of administration, and the ability to implement cross-distro applications reliably.

One important thing to mention when dealing with these systems is that Linux implements just about everything as a file. This means that a text file is a file, a directory is a file (simply a list of other files), a printer is represented by a file (the device drivers can send anything written to the printer file to the physical printer), etc.

Although this is in some cases an oversimplification, it informs us of the approach that the designers of the system encouraged: passing text and bytes back and forth and being able to apply similar strategies for editing and accessing diverse components.

### Traversing the Filesystem

Before actually delving into the filesystem layout, you need to know a few basics about how to navigate a filesystem from the command line. We will cover the bare minimum here to get you on your feet.

The first thing you need to do is orient yourself in the filesystem. There are a few ways to do this, but one of the most basic is with the pwd command, which stands for "print working directory":

pi@HAL:~$ pwd

/home/pi

This simply returns the directory you are currently located in.

To see what files are in the current directory, you can issue the *ls* command, which stands for "list":

pi@HAL:/$ ls

bin dev home lost+found mnt proc run selinux sys usr

boot etc lib media opt root sbin srv tmp var

This will tell you all directories and files in your current directory.

The *ls* command can take some optional flags. Flags modify the commands default behavior to either process or display the data in a different way.

The two most common flags are probable *-l* and *-a*. The first flag forces the command to output information in long-form:

pi@HAL:/$ ls -l

total 88

drwxr-xr-x 2 root root 4096 Jun 20 10:55 bin

drwxr-xr-x 2 root root 16384 Jan 1 1970 boot

drwxr-xr-x 12 root root 3060 Sep 24 13:31 dev

drwxr-xr-x 99 root root 4096 Sep 24 13:31 etc

drwxr-xr-x 3 root root 4096 Jun 20 07:48 home

drwxr-xr-x 12 root root 4096 Jun 20 10:42 lib

-rw-r--r-- 1 root root 0 Sep 24 13:37 log.txt

drwx------ 2 root root 16384 Jun 20 07:34 lost+found

drwxr-xr-x 2 root root 4096 Jun 20 07:36 media

...

This produces output with one line for each file or directory (the name is on the far right). This has a lot of information that we are not interested in right now. One part we are interested in though is the very first character, which tells us what kind of file it is. The three most common types are:

* -: Regular file
* d: Directory (a file of a specific format that lists other files)
* l: A hard or soft link (basically a shortcut to another file on the system)

The -a flag lists all files, including hidden files. In Linux, files are hidden automatically if they begin with a dot:

pi@HAL:/$ ls -a

. bin dev home log.txt media opt root sbin srv tmp var

.. boot etc lib lost+found mnt proc run selinux sys usr

The first two entries, . and .. are special. The . directory is a shortcut that means "the current directory". The .. directory is a shortcut that means "the current directory's parent directory".

Now that you can find out where you are in the filesystem and see what is around you, it is time to learn how to move throughout the filesystem.

To change to a different directory, you issue the *cd* command, which stands for "change directory":

pi@HAL:/$ cd /home

You can follow the command with either an absolute or a relative pathname.

* An **absolute path** is a file path that specifies the location of a directory from at the top of the directory tree. Absolute paths begin with a "/", as you see above.
* A **relative path** is a file path that is relative to the current working directory. This means that instead of defining a location from the top of the directory structure, it defines the location in relation to where you currently are.

For instance, if you want to move to a directory within the current directory called documents, you can issue this command:

pi@HAL:/home$ cd pi

The lack of the "/" from the beginning tells to use the current directory as the base for looking for the path.

This is where the .. directory comes in handy. To move to the parent directory of your current directory, you can type:

pi@HAL:~$ cd ..

There is also a shortcut to specify your own homedir, and that is by using the tilde “~”. You can immediately jump to your homedir by for example executing the following command:

$ pi@HAL:/$ cd ~

### An Overview of the Linux Filesystem Layout

The first thing you need to know when viewing a Linux filesystem is that the filesystem is contained within a single tree, regardless of how many devices are incorporated.

What this means is that all components accessible to the operating system are represented somewhere in the main filesystem. If you use Windows as your primary operating system, this is different from what you are used to. In Windows, each hard drive or storage space is represented as its own filesystem, which are labeled with letter designations (C: being the standard top-level directory of the system file hierarchy and additional drives or storage spaces being given other letter labels).

In Linux, every file and device on the system resides under the "root" directory, which is denoted by a starting "/".

Thus, if we want to go to the top-level directory of the entire operating system and see what is there, we can type:

pi@HAL:/home$ cd /

Every file, device, directory, or application is located under this one directory. Under this, we can see the beginnings of the rest of the directory structure.

One of the principles guiding the organization of the file system is to allow it to be split across multiple disk partitions (or multiple disks) in a rational manner, and to allow appropriate pieces of it to be shared between machines. Key to this is the notion of the root partition (/, the parent of the entire filesystem).

When Linux boots, the kernel attaches a single file system partition all by itself. This is known as the root partition. Any other partitions that need to be attached are mounted by the mount command, usually under control of entries in the file “/etc/fstab”. Because in the early stages of startup, only the root file system is available, it must contain everything needed for the system to function and attach the other pieces of the file system.

Tools on the root partition include the init program (which starts all the other processes), a shell, mount and the “/etc/fstab” file. The File System Hierarchy standard specifies a number of directories that must lie within the root partition.

Figure 2 shows a typical Linux file system hierarchy.

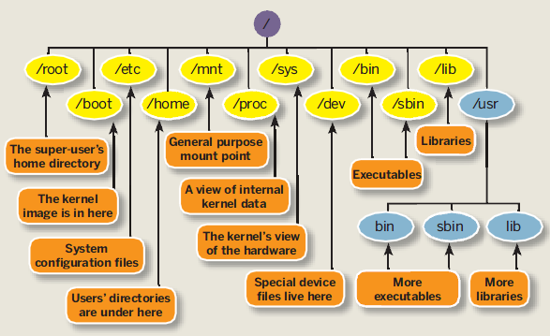


Figure 2 A typical Linux file system hierarchy

Let’s take a closer look at the different directories which can be found under root.

**/bin** - This directory contains basic commands and programs that are needed to achieve a minimal working environment upon booting. These are kept separate from some of the other programs on the system to allow you to boot the system for maintenance even if other parts of the filesystem may be damaged or unavailable. If you search this directory, you will find that both *ls* and *pwd* reside here. The *cd* command is actually built into the shell we are using (bash), which is in this directory too.

**/boot** - This directory contains the actual files, images, and kernels necessary to boot the system. While /bin contains basic, essential utilities, /boot contains the core components that actually allow the system to boot. If you need to modify the bootloader on your system, or if you would like to see the actual kernel files and initial ramdisk (initrd), you can find them here. This directory must be accessible to the system very early on.

**/dev** - This directory houses the files that represent devices on your system. Every hard drive, terminal device, input or output device available to the system is represented by a file here. Depending on the device, you can operate on the devices in different ways. For instance, for a device that represents a hard drive, like /dev/sda, you can mount it to the filesystem to access it. On the other hand, if you have a file that represents a line printer like /dev/lpr, you can write directly to it to send the information to the printer.

**/etc[[4]](#footnote-4)** - This is one area of the filesystem where you will spend a lot of time if you are working as a system administrator. This directory is basically a configuration directory for various system-wide services. By default, this directory contains many files and subdirectories. It contains the configuration files for most of the activities on the system, regardless of their function. In cases where multiple configuration files are needed, many times an application-specific subdirectory is created to hold these files. If you are attempting to configure a service or program for the entire system, this is a great place to look.

**/home** - This location contains the home directories of all of the users on the system (except for the administrative user, root). If you have created other users, a directory matching their username will typically be created under this directory. Inside each home directory, the associated user has write access. Typically, regular users only have write access to their own home directory. This helps keep the filesystem clean and ensures that not just anyone can change important configuration files.

Within the home directory, that are often hidden files and directories (represented by a starting dot) that allow for user-specific configuration of tools. You can often set system defaults in the /etc directory, and then each user can override them as necessary in their own home directory.

**/lib** - This directory is used for all of the shared system libraries that are required by the /bin and /sbin directories. These files basically provide functionality to the other programs on the system. This is one of the directories that you will not have to access often.

**/lost+found** - This is a special directory that contains files recovered by /fsck, the Linux filesystem repair program. If the filesystem is damaged and recovery is undertaken, sometimes files are found but the reference to their location is lost. In this case, the system will place them in this directory.

In most cases, this directory will remain empty. If you experience corruption or any similar problems and are forced to perform recovery operations, it's always a good idea to check this location when you are finished.

**/media** - This directory is typically empty at boot. Its real purpose is simply to provide a location to mount removable media (like cd’s). If your Linux operating system ever mounts a media disk and you are unsure of where it placed it, this is a safe bet.

**/mnt -** This directory is similar to the /media directory in that it exists only to serve as an organization mount point for devices. In this case, this location is usually used to mount filesystems like external hard drives, etc.

This directory is often used in a VPS environment for mounting network accessible drives. If you have a filesystem on a remote system that you would like to mount on your server, this is a good place to do that.

**/opt** - This directory's usage is rather ambiguous. It is used by some distributions, but ignored by others. Typically, it is used to store optional packages. In the Linux distribution world, this usually means packages and applications that were not installed from the repositories.

For instance, if your distribution typically provides the packages through a package manager, but you installed program X from source, then this directory would be a good location for that software. Another popular option for software of this nature is in the /usr/local directory.

**/proc -** The /proc directory is actually more than just a regular directory. It is actually a pseudo-filesystem of its own that is mounted to that directory. The proc filesystem does not contain real files, but is instead dynamically generated to reflect the internal state of the Linux kernel.

This means that we can check and modify different information from the kernel itself in real time. For instance, you can get detailed information about the memory usage by typing

pi@HAL:/$ cat /proc/cpuinfo

processor : 0

model name : ARMv6-compatible processor rev 7 (v6l)

Features : swp half thumb fastmult vfp edsp java tls

CPU implementer : 0x41

CPU architecture: 7

CPU variant : 0x0

CPU part : 0xb76

CPU revision : 7

Hardware : BCM2708

Revision : 000e

Serial : 000000008d79b8e3

**/root -** This is the home directory of the administrative user (called "root"). It functions exactly like the normal home directories, but is housed here instead.

**/run -** This directory is for the operating system to write temporary runtime information during the early stages of the boot process.

**/sbin -** This directory is much like the /bin directory in that it contains programs deemed essential for using the operating system. The distinction is usually that /sbin contains commands that are available to the system administrator, while the other directory contains programs for all of the users of the system.

**/selinux -** This directory contains information involving security enhanced Linux. This is a kernel module that is used to provide access control to the operating system.

**/srv** - This directory is used to contain data files for services provided by the system. In most cases, this directory is not used too much because its functionality can be implemented elsewhere in the filesystem.

**/tmp -** This is a directory that is used to store temporary files on the system. It is writable by anyone on the computer and does not persist upon reboot. This means that any files that you need just for a little bit can be put here. They will be automatically deleted once the system shuts down.

**/usr[[5]](#footnote-5)** **-** This directory is one of the largest directories on the system. It basically includes a set of folders that look similar to those in the root / directory, such as /usr/bin and /usr/lib. This location is basically used to store all non-essential programs, their documentation, libraries, and other data that is not required for the most minimal usage of the system.

This is where most of the files on the system will be stored. Some important subdirectories are /usr/local, which is an alternative to the /opt directory for storing locally compiled programs. Another interesting thing to check out is the /usr/share directory, which contains documentation, configuration files, and other useful files.

**/var -** This directory is supposed to contain variable data. In practice, this means it is used to contain information or directories that you expect to grow as the system is used.

For example, system logs and backups are housed here. Another popular use of this directory is to store web content if you are operating a web server.

### overview of Basic Filesystem commands

The most used commands to traverse and manipulate the file system of a Linux system are listed in Table 1. You can always use the man-command to get a detailed description.

Table 1 Basic Linux commands

|  |  |
| --- | --- |
| Command | Description |
| ls | List files |
| cp | Copy files |
| rm | Remove files |
| mv | Move files |
| cd | Change working dir |
| chmod | Change file permission mode |
| chown | Change owner of file |
| cat | Concatenate files and output to terminal |
| touch | Create an empty file |
| mkdir | Make directory |

## Debian and it’s Packages

Packages generally contain all of the files necessary to implement a set of related commands or features. There are two types of Debian packages:

* **Binary packages**, which contain executables, configuration files, man/info pages, copyright information, and other documentation. These packages are distributed in a Debian-specific archive format; they are usually distinguished by having a '.deb' file extension. Binary packages can be unpacked using the Debian utility *dpkg* (possibly via a frontend like aptitude).
* **Source packages**, which consist of a .dsc file describing the source package (including the names of the following files), a .orig.tar.gz file that contains the original unmodified source in gzip-compressed tar format and usually a .diff.gz file that contains the Debian-specific changes to the original source. The utility *dpkg-source* packs and unpacks Debian source archives. (The program apt-get can be used as a frontend for dpkg-source.)

Installation of software by the package system uses "dependencies" which are carefully designed by the package maintainers. These dependencies are documented in the control file associated with each package.

For example, the package containing the GNU C compiler (gcc) "depends" on the package binutils which includes the linker and assembler. If a user attempts to install gcc without having first installed binutils, the package management system (dpkg) will send an error message that it also needs binutils, and stop installing gcc.

There are multiple tools that are used to manage Debian packages, from graphic or text-based interfaces to the low level tools used to install packages. All the available tools rely on the lower level tools to properly work.

It is important to understand that the higher level package management tools such as *aptitude* or *dselect* rely on *apt* which, itself, relies on *dpkg* to manage the packages in the system.

**dpkg** is the main package management program.

**APT** is the Advanced Package Tool and provides the *apt-get* program. apt-get provides a simple way to retrieve and install packages from multiple sources using the command line. Unlike dpkg, apt-get does not understand .deb files, it works with the packages proper name and can only install .deb archives from a source specified in /etc/apt/sources.list. apt-get will call dpkg directly after downloading the .deb archives from the configured sources.

Some common ways to use apt-get are:

* To update the list of package known by your system, you can run:

$ apt-get update

(you should execute this regularly to update your package lists)

* To upgrade all the packages on your system (without installing extra packages or removing packages), run:

$ apt-get upgrade

* To install the foo package and all its dependencies, run:

$ apt-get install foo

* To remove the foo package from your system, run:

$ apt-get remove foo

* To remove the foo package and its configuration files from your system, run:

$ apt-get --purge remove foo

* To upgrade all the packages on your system, and, if needed for a package upgrade, installing extra packages or removing packages, run:

$ apt-get dist-upgrade

**aptitude** is a package manager for Debian GNU/Linux systems that provides a frontend to the apt package management infrastructure. aptitude is a text-based interface using the curses library, it can be used to perform management tasks in a fast and easy way.

* + aptitude offers easy access to all versions of a package.
  + aptitude makes it easy to keep track of obsolete software by listing it under "Obsolete and Locally Created Packages".
  + aptitude includes a fairly powerful system for searching particular packages and limiting the package display. Users familiar with mutt will pick up quickly, as mutt was the inspiration for the expression syntax.
  + aptitude can be used to install the predefined tasks[[6]](#footnote-6) available.
  + aptitude in full screen mode has su functionality embedded and can be run by a normal user. It will call su (and ask for the root password, if any) when you really need administrative privileges

Aptitude is most commonly used for searching for packages. You can use the following command for this:

pi@HAL:/$ aptitude search foobar

**Assignment**

Install the git package and create a local clone inside your homedir (currently the user pi) of the repository <https://github.com/BioBoost/multimediatechnieken>

**Assignment**

Install the apache package and find out where the webpages are stored. Make sure you can view your PI’s website from your host machine. Change the index.html page (you can use the *nano* editor for this) and add some cool things to it.

# Running a Virtual Machine

A virtual machine (VM) is an emulation of a particular computer system. This system can be based on an existing or hypothetical machine. As a user we can create such virtual machines and install an operating system of choice on them. This allows us to run a Linux distribution while working on a Windows machine and vice versa.

Several software packages are available to create and run virtual machines. Examples are VMware, Hyper-V[[7]](#footnote-7) which comes with Windows 8, Oracle VirtualBox, … For our labs we will be using VirtualBox as this is free, lightweight, easy to use and available for Windows, Linux, Mac and Solaris.

## Installing Oracle VirtualBox

Start by going to the download section of the website of VirtualBox (<https://www.virtualbox.org/>). Download the VirtualBox platform package for your system. At the moment of this writing the current version of VirtualBox is 4.3.16.

When running the installer package make sure to install VirtualBox with all features enabled as shown in Figure 3.

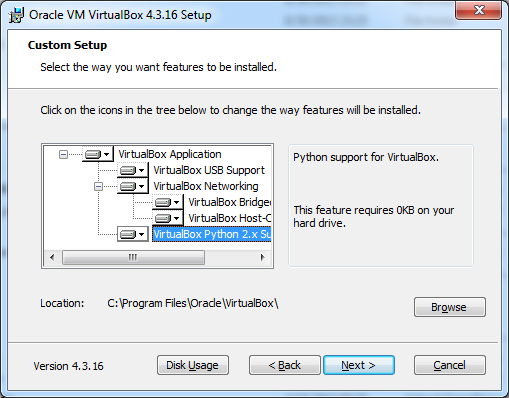


Figure 3 Installing VirtualBox with all features enabled

Next we also need to install the extension pack which introduces USB2.0 support and some other extra features. You can download the extension pack on the same page as you downloaded the installer for VirtualBox. Just make sure to pick the correct version for your current VirtualBox version.

The installer of VirtualBox should also have created a virtual network adapter (such as can be seen in xxxxxxxx) which is used for private networking between the host machine and the virtual machine.

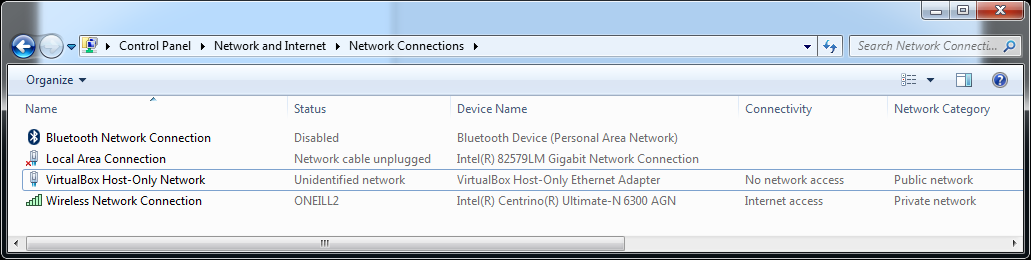


Figure 4 VirtualBox virtual network adapter

Once you’re finished you can start the VirtualBox client and you should get the interface presented in Figure 5.

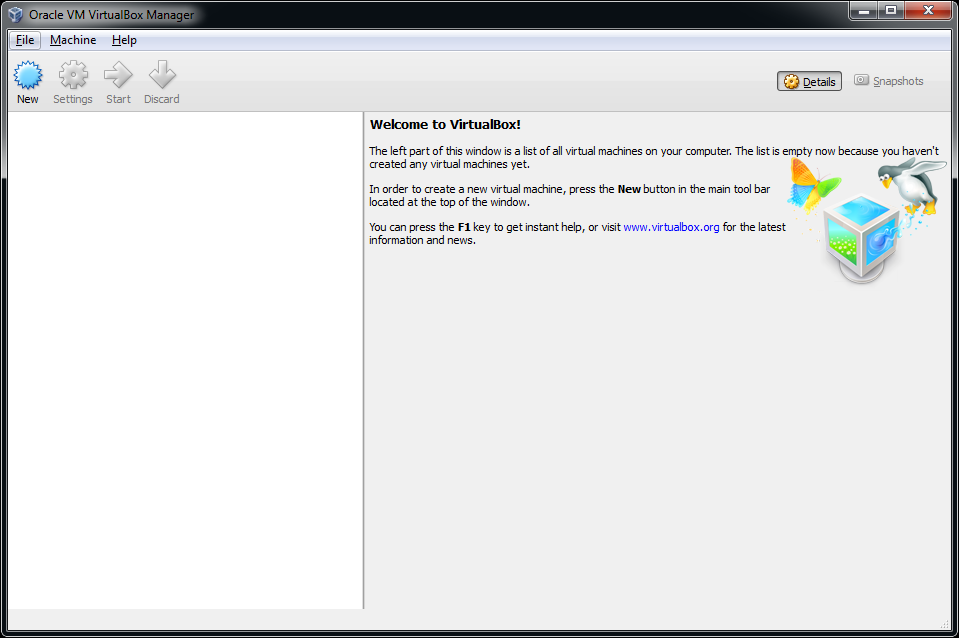


Figure 5 Launching VirtualBox after installation

Under *Files => Preferences => General* you can change the default path for your virtual machines. Do take note that you will need about 25GB of free space for each VM. For these labs you will most likely only need 1 VM.

Under *Files => Preferences => Language* you can also change the default interface language if you wish.

## Creating a Virtual Machine

Creating a virtual machine is very simple as it just following the steps presented to you by the wizard. To start the process of creating a VM hit the *New* button on the main interface of VirtualBox.

The first step consist of giving your VM a name and selecting the operating system you will be running on the VM as shown in Figure 6. In our case we will use Ubuntu 12.04 LTS[[8]](#footnote-8) 64 bit Desktop edition. More on this later.

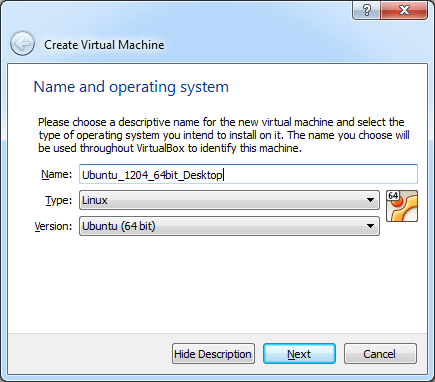


Figure 6 Creating a VM - Step 1 - The name and OS

In step 2 we need to select the amount of memory we want to assign to the virtual machine (as seen in xxxxxxxxx). The recommended amount is 512MB. However if you have more than 4GB, select 1024MB or even 2048MB, which will improve the responsiveness and performance of the VM.

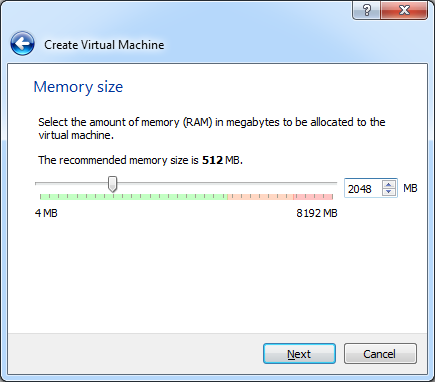


Figure 7 Creating a VM - Step 2 - Amount of memory

Next we need to choose what if we want to create a new or use an existing virtual hard drive. Pick the option to create one now as depicted in Figure 8.This will launch another wizard which will lead us through the creation process of a virtual drive.

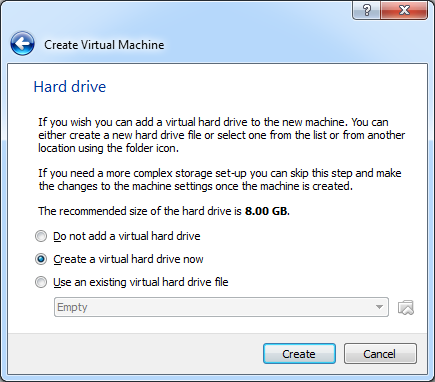


Figure 8 Creating a VM - Step 3 - Virtual hard drive

The first screen will allow us to select what type of virtual drive file we want to create. Just leave the default option (VDI – Virtual Disk Image) as shown in Figure 9.

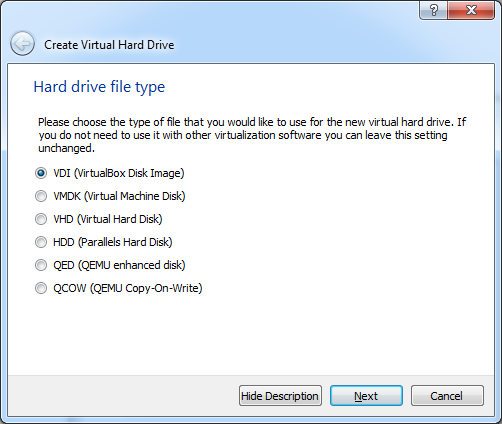


Figure 9 Creating a VM - Step 3a - Hard drive file type

In the next step we get the option to create a dynamically allocated image or a fixed size image as can be seen in Figure 10. A fixed size image is faster but will take up the full space we select for the size of the virtual drive. A dynamically allocated image is faster but will only grow in size when needed. You will need to decide this for yourself based on the free space available on your host system.

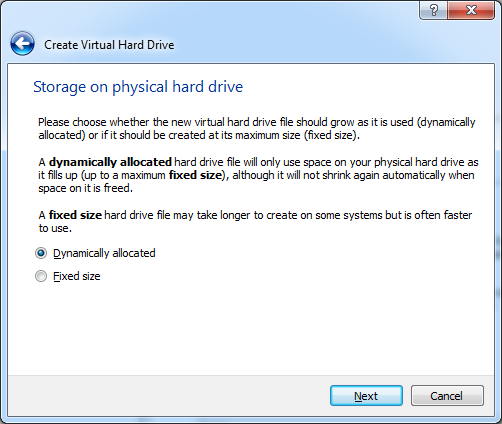


Figure 10 Creating a VM - Step 3b - Virtual drive allocation method

Now we need to select the hard drive file location (leave it as is) and size of the drive. Make sure to select at least 25GB for the size as shown in Figure 11. Hitting create will finish the process of creating a VM.

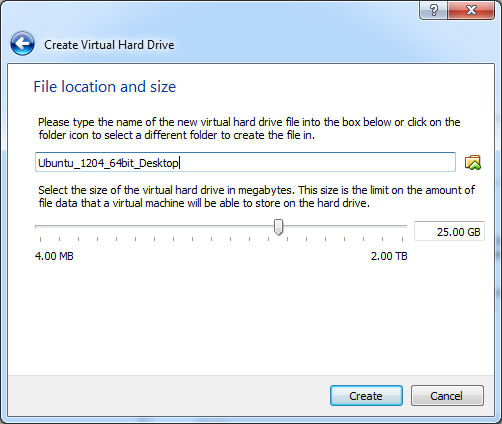


Figure 11 Creating a VM - Step 3c - Location and size of the virtual drive

Your new VM should now appear in the list of VM’s on the left side of the VirtualBox main interface as seen in Figure 12. Selecting a VM in the list also displays some basic information about the VM.

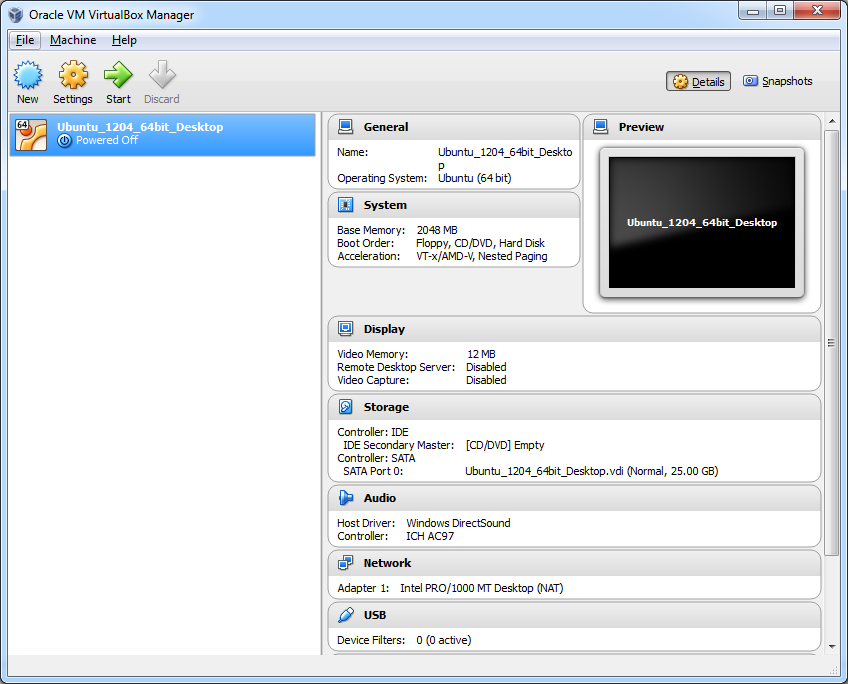


Figure 12 A new VM is added to your current list of VM’s

## Configuring the Virtual Machine

Before installing an operating system on the newly created VM it is necessary to make a few configuration changes. Select the new VM and hit the *Settings* button on the main interface. You will be presented with the configuration settings of your VM.

Start by going to General => Advanced and enabling the bidirectional shared clipboard. This allows text to be copied to the clipboard in the VM and pasted in your host OS and vice versa. Also enable bidirectional Drag’n Drop. The resulting configuration is shown in Figure 13.

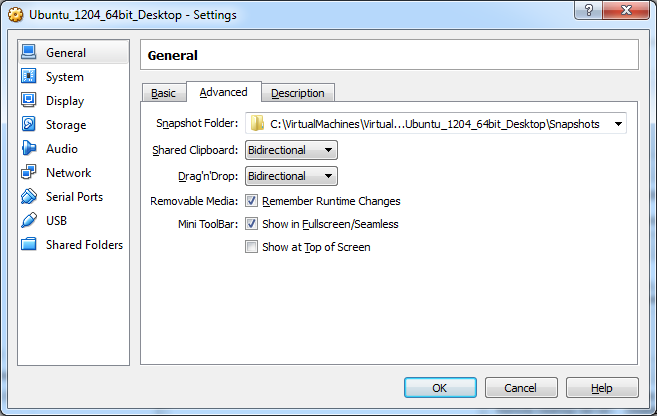


Figure 13 Configure VM to allow bidirectional clipboard and drag'n drop

In some cases it is also necessary to change the networking configuration of the VM. Default the VM is configured with a single network adapter with NAT (Network Address Translation) enabled as shown in Figure 14. This means that the VM has access to the network and also has access to the Internet. However because of NAT we will not be able to connect to the VM from another machine using SSH without configuring port forwarding. However since this will not be needed for these LAB’s we can leave the standard behavior.

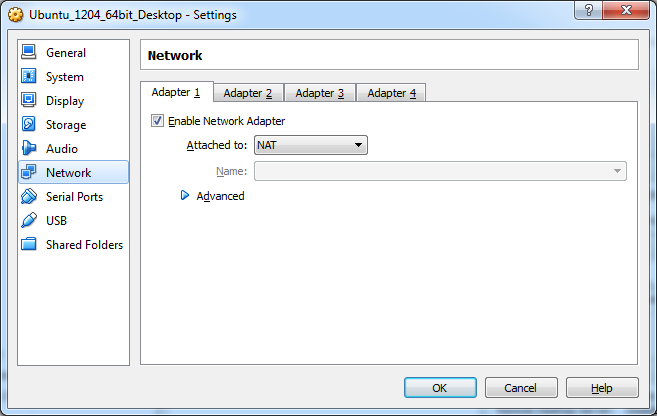


Figure 14 Configure VM to use NAT with the network adapter

Another much used option is a bridged adapter (shown in Figure 15). This will basically create a network bridge between the VM’s network adapter and your physical host adapter making your VM’s directly available on your network. This may be a security issue but can also simplify working with your VM. This option also implies that your VM will get its IP address from the same DHCP (Dynamic Host Configuration Protocol) server as your host machine if you have a DHCP enabled network.

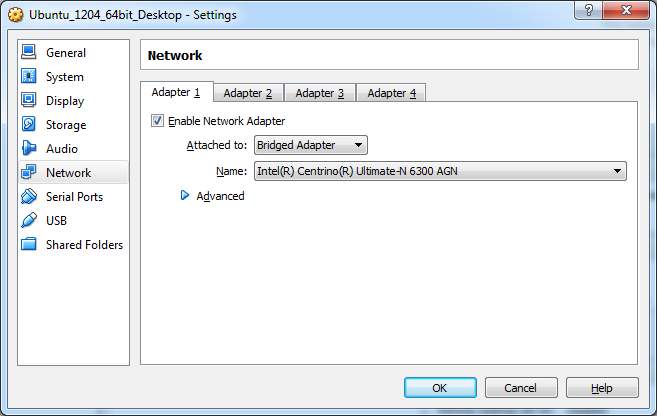


Figure 15 Configuring a VM to use a bridged network adapter

Skim through the rest of the configuration options and check out some of the available options.

## Installing the Ubuntu Operating System

Before we can start the install procedure of the operating system, we will have to download an installer image. This image can then be mounted on our VM allowing us to boot from it.

For our LABs we will be using Ubuntu 12.04 LTS (64 bit) (Precise Pangolin) which can be downloaded here <http://releases.ubuntu.com/precise/>. Make sure to select the 64-bit Desktop edition.

Once downloaded start VirtualBox and open the setting of your VM. Next open the *storage* settings. Now select the virtual CD/DVD drive below the IDE controller as shown in step 1 of Figure 16.

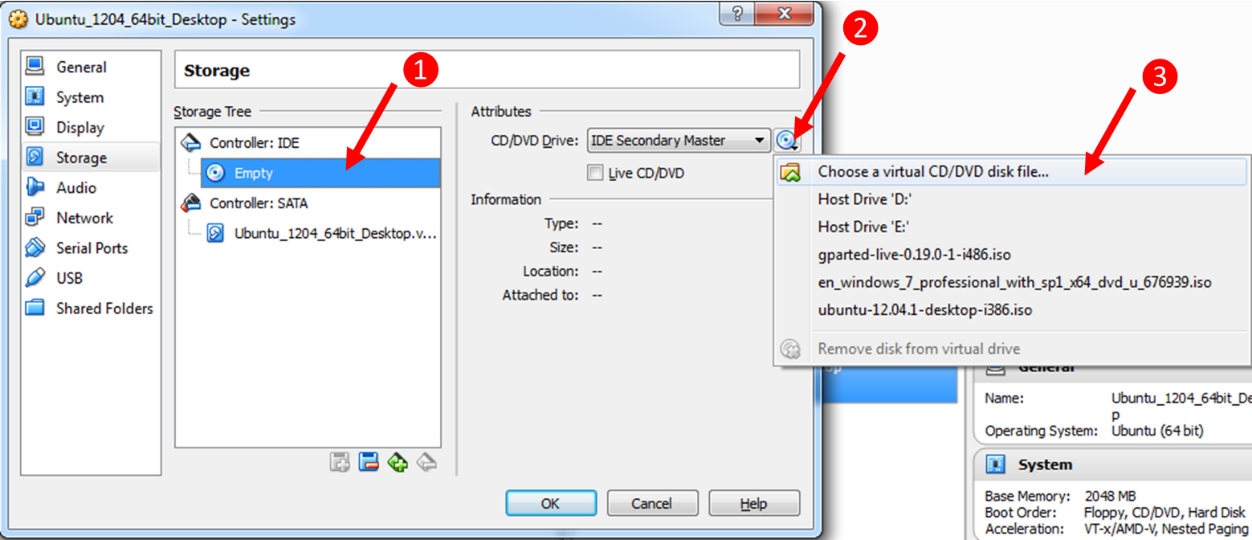


Figure 16 Steps for mounting an image in VirtualBox

Hit the small CD/DVD icon next to the *IDE Secondary Master* dropdown (step 2 in Figure 16) and select *Choose a virtual CD/DVD disk file…*. A browse window will open; select the image file you downloaded from the Ubuntu website and hit ok. Hit the ok button of the setting panel to close it.

Ready ? Then hit the start button of the VM and follow the steps for installing the Ubuntu operating system. Do make sure once you see the output shown in Figure 17 you hit a key on the keyboard or the Live DVD mode will boot. If you missed it, you can always reset the VM from the *Machine* menu in VirtualBox.

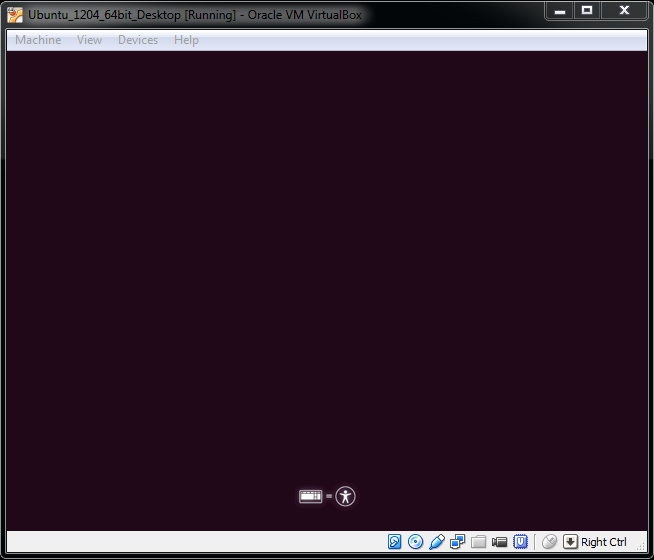


Figure 17 Live DVD or boot menu option

From this point on all steps are self-explanatory.

If you click inside the VM window your mouse cursor will automatically be captured. Releasing your cursor can be achieved using the right CTRL key.

Once the installation procedure is finished you will be asked to reboot the VM.

Login to the desktop. You should be presented with a popup window from the Update Manager after a few seconds. If not just launch the Update Manager from the menu. It’s always a good idea to keep your distribution up-to-date. So if you have some time and an active internet connection, you can hit the *Install Updates* button.

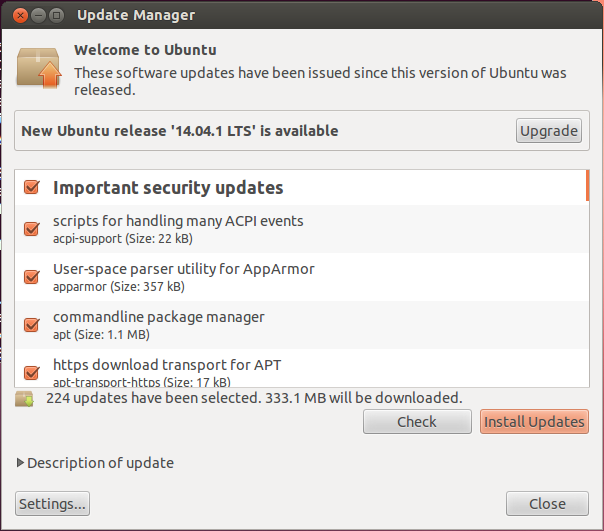


Figure 18 Ubuntu Update Manager

### Installing Guest Additions

You may or may not have noticed that your mouse movement is a bit sluggish within the VM. That’s because the guest additions haven’t been installed yet.

Open the Devices menu which can be found at the top of the VM window (not in Ubuntu). Next select *Insert Guest Additions CD image…* as shown in Figure 19. A window in Ubuntu will open asking if you’d wish to run the package. Hit run and follow the instructions.

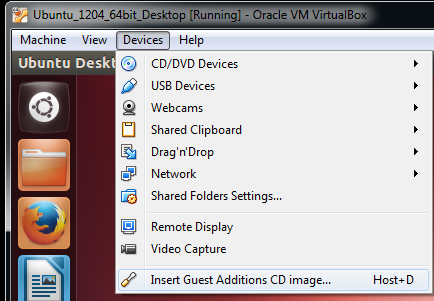


Figure 19 Inserting the Guest Additions for Ubuntu

Once finished remove the image from the virtual drive (through settings or through the small CD/DVD icon in the bottom right corner of your VM window). Next reboot the system for the changes to take effect.

If you update your machine it may be necessary to repeat this procedure.

## Creating a Shared Directory

Some files can be dragged and dropped between your host machine and the VM. However this does not seem to be possible for all file types. For these instances it is more easy to create a shared directory which can be accessed from the host and the VM.

Go to your VM’s settings and locate the *Shared Folders* options depicted in Figure 20.

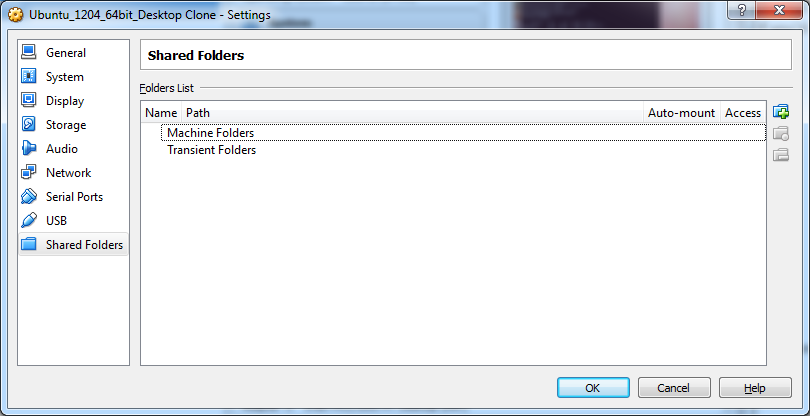


Figure 20 Shared folders settings of VM

Create a shared folder on your system and make sure to select the *Make Permanent* and *Auto-mount* options as shown in Figure 21.

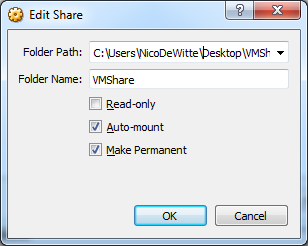


Figure 21 Creating a shared folder for your VM

The folder should now be automatically mounted under /media in your VM on your next reboot.

By adding your own user account to the user group *vboxsf* you get read and write access to the shared folder:

$ sudo usermod -a -G vboxsf nico

# nico should be replaced with your own user account

# The LAMP Stack

LAMP stands for Linux Apache MySQL and Perl (or PHP). These are the most used components necessary to host a website on a server.

Linux is the operating system. This can be any distribution but for the sake of simplicity we will be using Ubuntu. Common other distributions include Red Hat, Debian, Slackware, Suse, ArchLinux, …

Apache is a widely used web server. Apache supports a variety of features, many implemented as compiled modules which extend the core functionality. These can range from server-side programming language support to authentication schemes. Some common language interfaces support Perl, Python, Tcl, and PHP. Popular authentication modules include mod\_access, mod\_auth, mod\_digest, and mod\_auth\_digest, the successor to mod\_digest. A sample of other features include Secure Sockets Layer and Transport Layer Security support (mod\_ssl), a proxy module (mod\_proxy), a URL rewriter (mod\_rewrite), custom log files (mod\_log\_config), and filtering support (mod\_include and mod\_ext\_filter).

MySQL is a very popular relational database management system (RDBMS). To manage the databases we will be using PhpMyAdmin. It provides a nice and simple web interface for creating and maintaining MySQL databases.

PHP is a server side scripting language. To allow Apache to serve up PHP files we will need to install a separate module.

Of course life would not be meaningful if we could not manage our website’s versions with a versioning system such as GIT. So we will also be using GIT to pull the website from the Gitlab repositories.

## Updating

Updating the system is a very good idea when starting afresh. Ofcourse the system should be kept up-to-date as much as possible. However when updating a production server more care should be taken not to bring the system to its knees because of a failed update.

When installing packages always make sure to update your package repositories:

sudo apt-get update

Next you can install package upgrades

sudo apt-get upgrade

## Installing Apache WebServer and PHP Extensions

Installing Apache is real easy using the following command.

sudo apt-get install apache2 php5 libapache2-mod-php5

You should now have a directory “/var/www”. You should also have a group and user www-data. You can check the user by viewing the shadow file:

sudo cat /etc/shadow

Anyone got an idea why we need to use sudo here ?

You should now be able to surf to <http://localhost>. You should see the page shown in Figure 22.

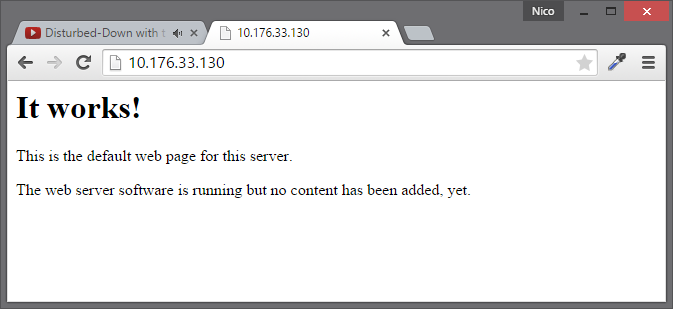


Figure 22 Apache - It works

If needed you can reload Apache’s configuration files using the following command:

sudo service apache2 reload

By replacing the last argument (reload) with “restart” you can restart the webserver.

### Serving a Static Page

Let’s change the default page that is shown when we surf to localhost. Use the nano editor to edit the file (/var/www/index.html) and show something funny.

sudo nano /var/www/index.html

You can save changes by pressing CTRL-O and you can quit the editor by pressing CTRL-X.

To determine the IP-address of your server you can use the command

ifconfig

Try to access the server of another student.

## Installing MySql

To be able to use MySQL with our websites we will need to install MySQL itself but also an Apache extension. This way the webserver can communicate with the DBMS service.

sudo apt-get install mysql-server mysql-client php5-mysql

When requested make sure to provide a secure password for the root user. Also make sure to note it down so you won’t forget it by the time you need it.

## Installing PHPMYAdmin

Now we could access the DBMS from the terminal and execute SQL statements to create and manage our databases. However since we are serving up websites we may as well use a web-application for managing the databases. Install PHPMyAdmin using the following commands:

sudo apt-get install phpmyadmin

When asked make sure to select Apache as a webserver (use space to select the package). Also make sure to select to configure the database. If you miss this last part you can always restart this procedure using

sudo dpkg-reconfigure phpmyadmin

The first password that needs to be provided is the root password of the database. The second password is the password that is attached to the phpmyadmin user account which will be created by the configuration process.

You should now be able to access PHPMyAdmin via <http://localhost/phpmyadmin>. You should see a login screen as shown in Figure 23.

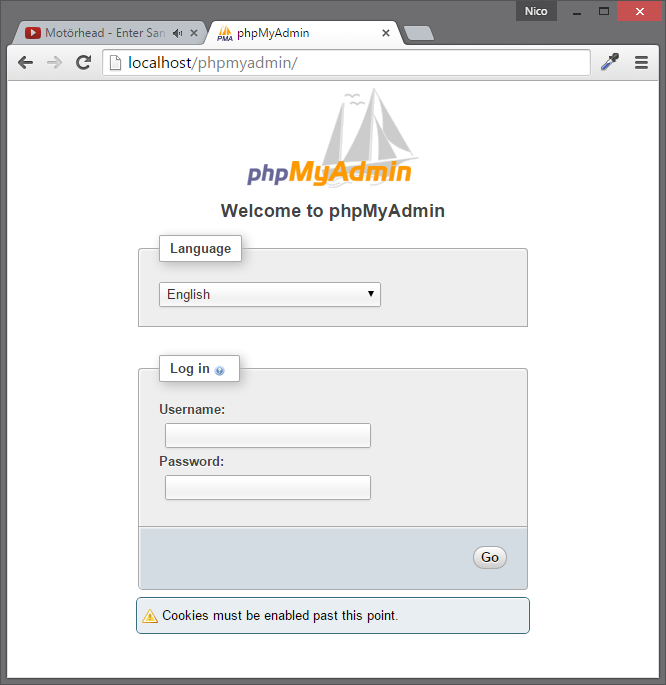


Figure 23 PHPMyAdmin login screen

Login using root and your password. You should see the screen shown in Figure 24.

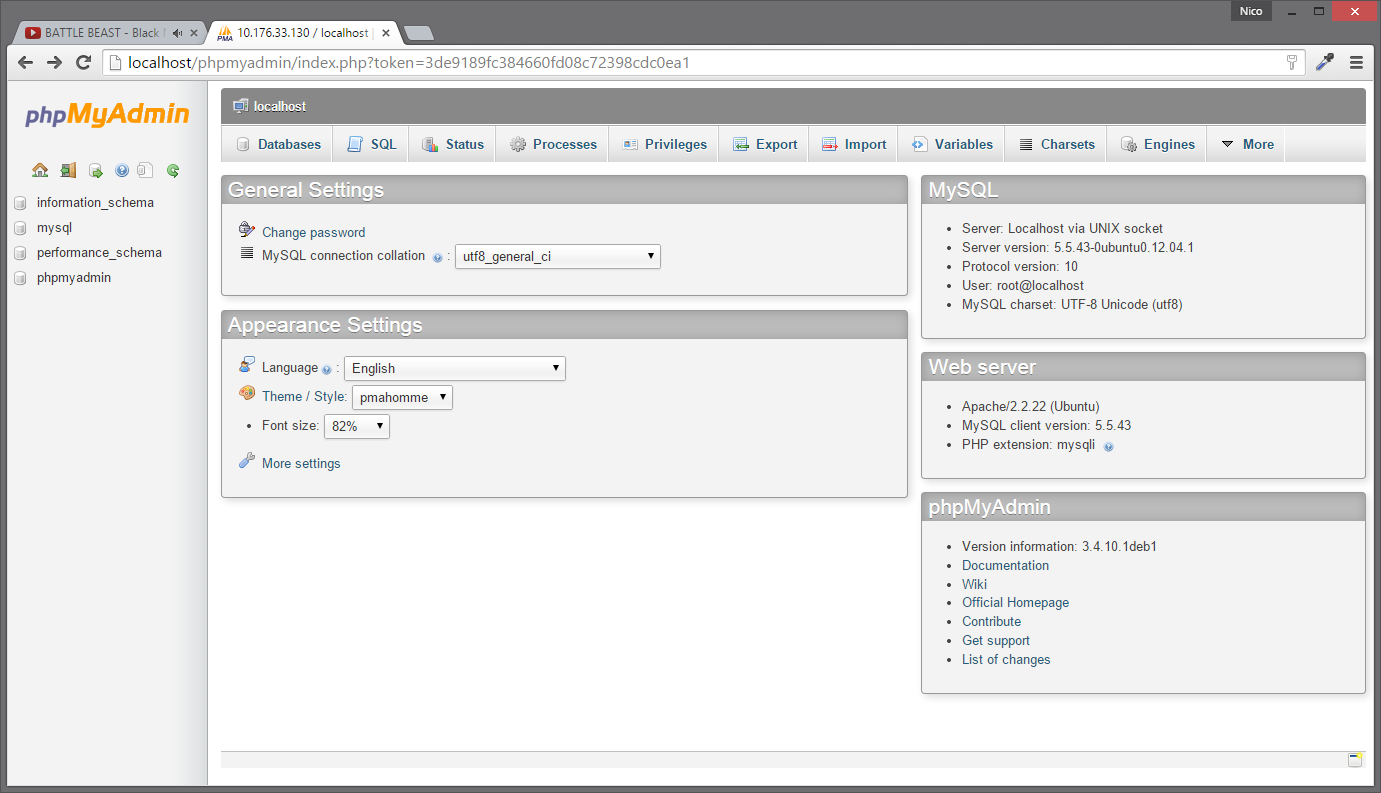


Figure 24 PHPMyAdmin main screen

# Hosting a Website

You should all have a GIT repository on the gitlab server (git.labict.be) with a web-application which we will be serving.

## Installing GIT

Let’s start by installing git:

sudo apt-get install git

## Setting Up SSH and Keys

We could now clone a git repository inside /var/www and serve it as a website to the outside-world. However since our gitlab server requires key-authentication we will first need to configure our system to use our SSH key.

### Installing SSH client and Server

Start by installing SSH:

sudo apt-get install ssh

### Setting up the Secret Key

Start by traversing to your home folder and creating a hidden “.ssh” folder for our configuration and our secret SSH keys.

cd ~

mkdir .ssh

Next we need to copy our extremely secret key from our host machine (Windows or Mac) to the server. Do this by opening a powershell window (CTRL-R and type in powershell) as shown in Figure 25.

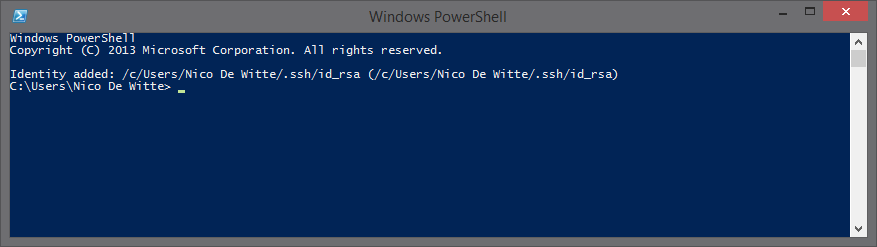


Figure 25 A PowerShell terminal

If you have git installed on your host machine with all Linux tools you should have the command “scp” (secure copy) available. If not skip to the next paragraph. You can then copy your secret key from your host to your server using the command below

cd C:\Users\<your\_user\_name>\.ssh

scp id\_rsa <username\_on\_ubuntu>@<ip\_address\_ubuntu>:/tmp

An example:

cd “C:\Users\Nico De Witte\.ssh”

scp id\_rsa nico@10.176.33.130:/tmp

The result should look similar to Figure 26.

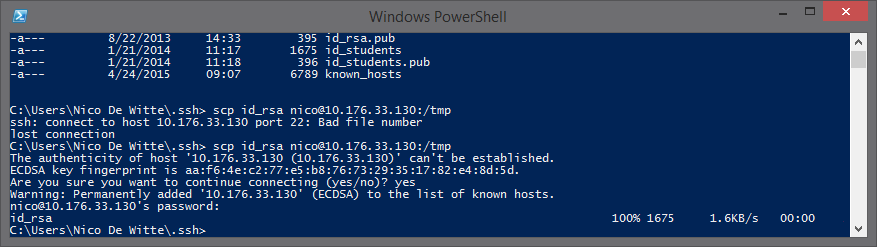


Figure 26 Secure copying the secret key

Now you can move the key on the Linux machine to your .ssh folder:

mv /tmp/id\_rsa ~/.ssh

If you do not have the scp command available you can open up a simple text editor for viewing id\_rsa and copy/pasting it to the VM (using nano).

cd ~/.ssh

nano id\_rsa

Last we should also make sure no one can read our secret key by setting the permissions for the .ssh folder and its files (therefore the -R for recursive):

chmod -R o-rwx,g-rwx ~/.ssh

## Cloning The Repository

As a last step we should change the owner of the /var/www dir. Let’s make ourselves owner and set the group to www-data:

sudo chown -R nico:www-data /var/www

You should now be able to clone your repository

cd /var/www

ssh://git@git.labict.be:28615/webscripten/thehardwarestore.git

The result should be similar to Figure 27.

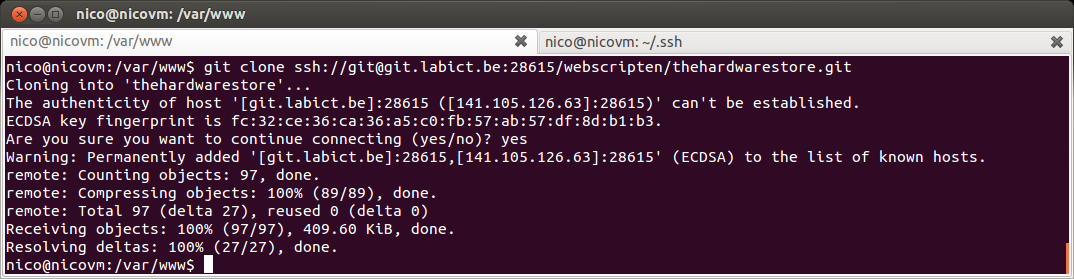


Figure 27 Cloning a git repository

You will still need to configure your web-application and setup your database. Once this is done you should be able to access the application from your host machine.

## Any problems ?

Check out the log files of apache:

sudo su

cd /var/log/apache2

cat error.log

cat access.log

1. In Linux hidden files start with a dot, for example “.ssh”. [↑](#footnote-ref-1)
2. [https://www.digitalocean.com/community/tutorials/how-to-understand-the-filesystem-layout-in-a-linux-vps] [↑](#footnote-ref-2)
3. Check http://www.pathname.com/fhs/ [↑](#footnote-ref-3)
4. Stands for “Editable Text Configuration” [↑](#footnote-ref-4)
5. Stands for “Unix System Resources” [↑](#footnote-ref-5)
6. See “man tasksel” [↑](#footnote-ref-6)
7. Hyper-V, codenamed Viridian and formerly known as Windows Server Virtualization, is a native hypervisor; it can create virtual machines on x86-64 systems. Starting with Windows 8, Hyper-V supersedes Windows Virtual PC as the hardware virtualization component of the client editions of Windows NT. [↑](#footnote-ref-7)
8. Long-term support (LTS) is a term used to describe special versions or editions of software designed to be supported for a longer than normal period. It is particularly applicable to open-source software projects. [↑](#footnote-ref-8)