Technische Universität Kaiserslautern

Bachelor Thesis

Distributing Singular

Author:
Jose Antonio Martín Herguedas

Supervisor:
Dr. Hans Schönemann

A thesis submitted in fulfilment of the requirements
for the degree of Bachelor of Science in Computer Engineering

in the

Department of Mathematics

August 2014

Declaration of Authorship

I, Jose Antonio Martín Herguedas, declare that this thesis titled, 'Distributing Singular' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Abstract

Department of Mathematics

Bachelor of Science in Computer Engineering

Distributing Singular

by Jose Antonio Martín Herguedas

Nowadays, one of the problems of developing a software is the distribution. There are multiple types of operating systems and machine architectures which result in a extensive fragmentation which is impossible to cover.

Faced by this problem, it is recommended to choose the most popular end-user operating systems and architectures and try to provide support for it.

This thesis takes place in a real scenario, its not a "generic" research in some field of the computer engineering. If the result of this work is satisfactory, it will help SINGULAR team to distribute his software in easy way.

Greetings

I would like to thank all those people who have helped me in the realization of this thesis, directly or indirectly, beginning with my tutor, Hans Schönemann, my family and my friends.

Contents

List of Figures

Dedicated to my family.

Chapter 1

Introduction

SINGULAR is a project conducted by the Technische Universität Kaiserslautern, Germany. It provides a powerful software system for polynomial computations with emphasis on the special needs of commutative algebra, algebraic geometry, and singularity theory.

In this thesis, the distribution of SINGULAR software has to be studied and developed. In concrete, specify instructions, and develop some automatic script for building binary packages from source code and a distribution system in order to provide an easy installation of this binaries to the final users.

1.1 Introduction to SINGULAR

SINGULAR is a Computer Algebra system for polynomial computations with emphasis on the special needs of commutative algebra, algebraic geometry, and singularity theory.

SINGULAR's main computational objects are ideals and modules over a large variety of baserings. The baserings are polynomial rings or localizations thereof over a field (e.g., finite fields, the rationals, floats, algebraic extensions, transcendental extensions) or over a limited set of rings, or over quotient rings with respect to an ideal.

It provides an easy-to-use interactive shell and a C-like programming language. The SINGULAR functionality can be easy extendible by libraries written in SINGULAR programming language.

Precisely that facility has turned SINGULAR into a very wide used software with multiple development teams all over the world:

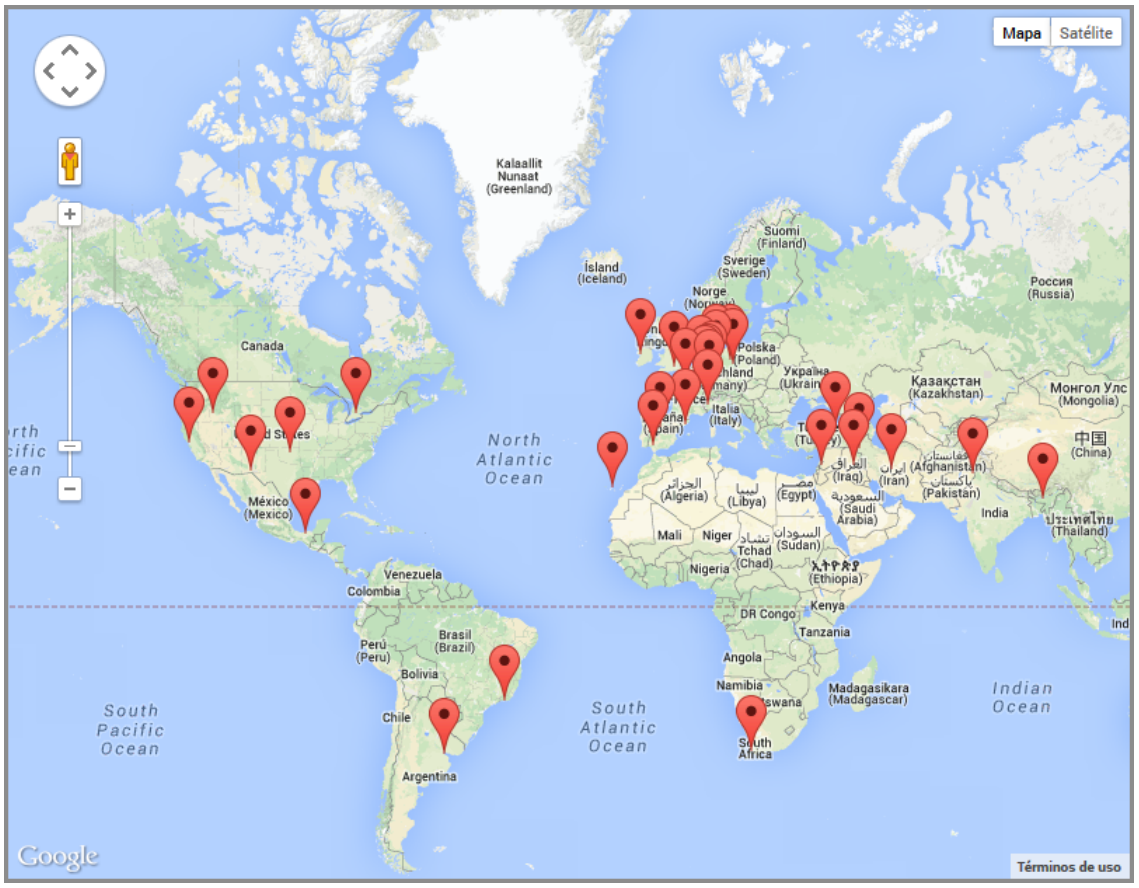

Figure 1.1: SINGULAR developers around the world.

1.2 Problems to be solved

This problem can be easily divided into two main areas:

- Building SINGULAR binaries from the source code for the required architectures and operating systems.

  This involves managing SINGULAR dependencies, compiling the source code and then building the packages.

- Provide a simple way for end users to get these SINGULAR packages.

  Study and implement for each system the best tool for distributing packages, creating repositories accessible by the default packages managers. Also, provide a set of instructions for final users.

Both of this steps have to be realized in an automatic way, so SINGULAR team can make each new version of their application available easily.

1.3 Related work

Although the fragmentation is one of the most common problems at the time of distributing a software, there is no a clear leader solution to cover it.

The most portable solution is to compile and build the source code in every platform, but it is also very inconvenient.

Usually, the software companies or small developers, build the software by themselves for the most common platforms.

1.3.1 Compiling Farms

Some companies or associations offer a set of machines, with different architectures and different operating systems running on it. This concept is called "Compile Farms".

That machines could be physical or virtual. The connection to them usually is made by ssh.

Some of this services are paid up but most of them are free. In the free ones, usually you have to comply some requirements in order to get access to that machines.

One example of Compiling Farms is the GCC Compiler Farm Project. They maintain a set of machines of various architectures and provide ssh access for for free software developers, GCC and others (GOL, BSD, MIT,...).

Your account application has to be approved in order to get access to all the farm machines.

Some of the architectures currently available are:

- i686
- x86_64
- armv5tel
- armv7l (with vfp and neon FPU instructions)
- powerpc
- powerpc64 (including POWER7)
- Cell SPE (Sony Playstation 3)

- sparc
- sparc64 (sparcv9)
- ...

Operating systems currently available:

- Debian 5.0 & 6.0 (various) & 7 (amd64 virtual)
- Ubuntu 12.04 (amd64 virtual)
- Fedora 18 (ppc) (amd64 virtual)
- Red Hat Enterprise Linux 6.3 (amd64 virtual)
- NetBSD 5.1 (amd64) & 5.1.2 (amd64 virtual) & 6 (amd64 virtual)
- OpenBSD 5.0 (x86 virtual)
- FreeBSD 8.3 (amd64 virtual) & 9.0 (x86 virtual)
- AIX 7.1 (power7)

1.4 Terminology

- Architecture: Set of disciplines that describe a computer system by specifying its parts and their relations. Its about the hardware of the system. In this case we will have i386 (32 bit x86 architecture) and X86_64 (64 bit x64 architecture).
- Operating System: Software that manages computer hardware resources and provides common services for computer programs.
- Platform: System specification, which is defined by a hardware architecture and an operating system.
- Unix: Portable, multitasking and multi-user operating system. Two of its most known families are Linux and GNU.
- Debian: GNU operating system based in free software. The system can be found precompiled, packaged and in deb format for multiple hardware architectures and multiple cores. Use a Linux Kernel.
- Fedora/RedHat: Operating system based on the Linux kernel. Use RPM (Red Hat Package Manager), a package management tool.

- Windows: Operating system which use MS-DOS or NT kernels.
- Cygwin: Large collection of GNU and Open Source tools which provide functionality similar to a Linux Distribution on Windows.
- build-essential: Meta-package of all tools for creating packages in Debian.
- autoconf: Tool which produces shell scripts to automatically configure software source code packages. This scripts can adapt the packages to many kinds of UNIX-Like systems.
- autogen: Tool designed to simplify the creation and maintenance of programs that contain large amounts of repetitious text. It is especially valuable in programs that have several blocks of text that must be kept synchronized.
- libtool: GNU libtool is a generic library support script. Libtool hides the complexity of using shared libraries behind a consistent, portable interface.
- readline: The GNU readline library aids in the consistency of user interface across discrete programs that need to provide a command line interface.
- glpk: GLPK (GNU Linear Programming Kit) is intended to solve large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is a set of routines written in ANSI C and organized in the form of a callable library.
- debhelper: A collection of programs that can be used in a Debian/rules file to automate common tasks related to build Debian packages.
- gmp: GNU Multi-Precision library, or GMP, is an essential component of the programmer who works with mathematics on a regular basis. The GMP library allows you to store arbitrary precision numbers (both integers and floats) with ease - indeed, with the C++ extension, GMPXX, allows you to use these data types in just about the same way as you use, say long.
- mpfr: The MPFR library is a C library for multiple-precision floating-point computations with correct rounding.
- cdd: A C++ libraries implementation of the Double Description Method of Motzkin et al. for generating all vertices (i.e. extreme points) and extreme rays of a general convex polyhedron.
- ntl: Library which provides data structures and algorithms for manipulating signed, arbitrary length integers, and for vectors, matrices, and polynomials over the integers and over finite fields.

- graphviz: Open source graph visualization software. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks.
- flint: C library for doing number theory which supports arithmetic with numbers, polynomials, power series and matrices over many base rings.
- 4ti2: A software package for algebraic, geometric and combinatorial problems on linear spaces.
- gcc: The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Java, Ada, and Go, as well as libraries for these languages (libstdc++, libgcj,...).
- automake: Automake is a tool for automatically generating Makefile.in files compliant with the GNU Coding Standards. Automake requires the use of Autoconf.
- make: GNU Make is a tool which controls the generation of executables and other non-source files of a program from the program's source files. Make gets its knowledge of how to build your program from a file called the makefile, which lists each of the non-source files and how to compute it from other files.
- wget: Software package for retrieving files using HTTP, HTTPS and FTP, the most widely-used Internet protocols.
- ssh: Secure Shell (SSH) is a cryptographic network protocol for secure data communication, remote command-line login, remote command execution, and other secure network services between two networked computers.
- expect: Program to automate interactions with programs that expose a text terminal interface.
- gpg: Key management system. Multi proposal tool, used to sign the packages and repositories.
- binary package: An archive file that contains all files and directories that must be installed in order to make a working installation of the program(s) included in the package, and the maintainer scripts necessary for the installation. A binary package is usually specific to a certain platform, in contrast to a source package.
- dpkg: Debian package management system.
- apt: Advanced Packaging Tool, provides a user command line interface and handle the installation and removal of packages in Debian. Works with dpkg but also with RPM Package Manager.
- deb: The filename extension for a Debian binary package.

- reprepro: Tool to manage a repository of Debian packages (.deb, .udeb, .dsc, ...)
- createrepo: Tool to create a RPM package repository. Creates a repomd (xml-based rpm metadata) from a set of rpms.
- rpm: Red Hat Package Manager: package management tool for Red Hat, Fedora, Mandriva Linux and SuSE Linux systems. Also is the filename extension for the binary packages of this systems.
- yum: The Yellowdog Updater Modified (yum) is an open-source command-line package-management utility for Linux operating systems using the RPM Package Manager.
- tar.xz: The filename extension for Cygwin binary packages, also used in other systems like Debian and Fedora.

Chapter 2

Concept

2.1 Problem statement

One of the problems, which SINGULAR developers have to face, is to distribute their application to the end users. The end users have to set up some machines with specific configurations and follow lots of steps to get the final built packages.

So, now they want to do this automatically: run some scripts which prepare the building hosts, download the source code, and generate the installation packages.

This packages should be available to the final users, in order to download them and easily install SINGULAR on their systems.

2.2 Operating Systems to support

The main goal is to try to cover the most popular operating systems. But also, we must notice, that this is a scientific software, not designed for common users.

As you can see in the Figure 3.1, windows operating systems have more than 90% of market share. Only a 7.5% for MacOSX and less than 2% for linux systems.

But as i said before, we have to pay attention to our type of customers. Most part of the Scientist community and our developers use Unix systems, mainly Linux, so this will be our main objective.

For better understanding, these are the download statistics from the last week in May, 2014 where windows still being the main OS but Linux gain more importance:

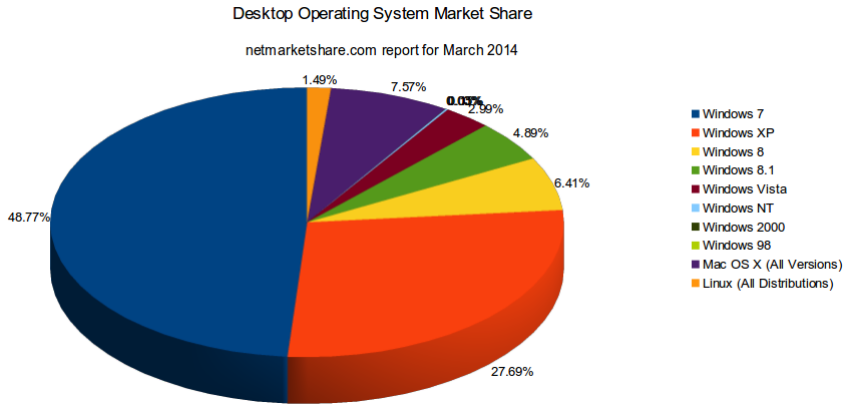

Figure 2.1: Operating Systems market share march 2014.

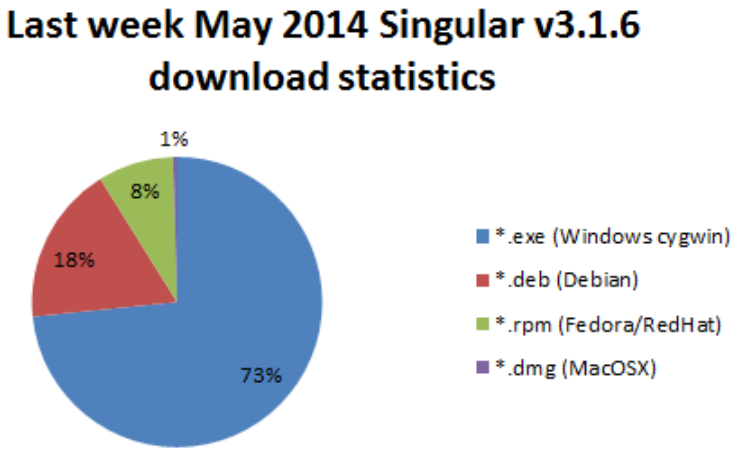

Figure 2.2: SINGULAR last week may 2014 download statistics.

In Linux, there are hundreds of different distributions, so we choose to support the the most popular ones: Debian and Fedora/Red Hat. Also Windows have to be supported.

2.3 Previous research

Before thinking about a solution for our problem, we need to understand how SINGULAR and Linux operating systems works, and everything about package managing.

- SINGULAR:

  SINGULAR is developed in C programming language. That means sources need to be built for every specific platform, not like Java, which always run in the Java Virtual Machine.

In the building process, SINGULAR has external dependencies. This dependencies are usually libraries which have to be already installed in the system before starting the compile process. Some of these dependencies are flint, 4ti2, libgmp-dev, etc.

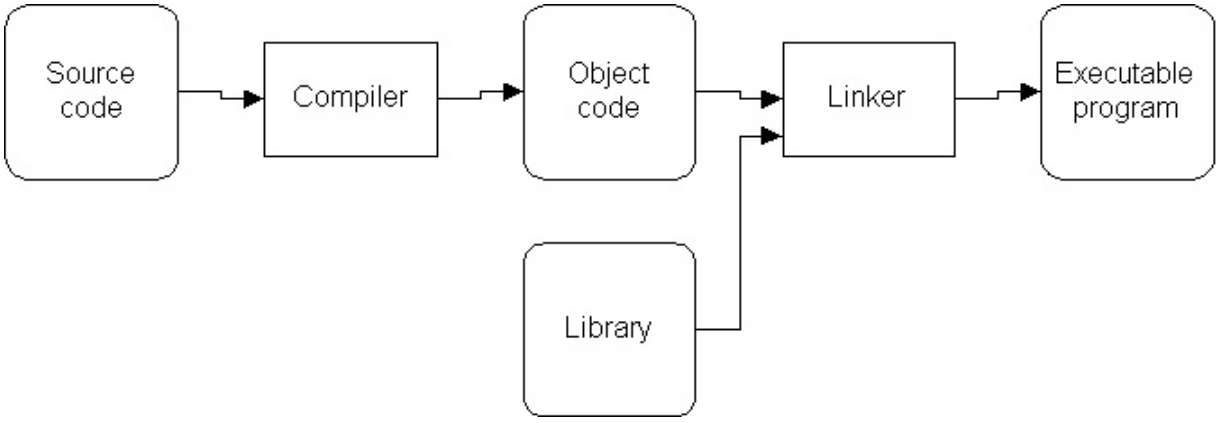

Figure 2.3: C++ Building process

Also, we have to notice that all the tools used in the compiling script should be already installed before starting the building.

So, if everything goes right, we will get the SINGULAR executable binaries. Ready to be executed trough command-line terminal.

- How software is installed on Linux systems:

  The way to install new software on Linux environments is different than Windows systems. Linux uses the concept of package managers. Which means that every package have to comply some standards. This normalization makes the creation of organized repositories possible, which provide simple distribution of the packages to the final users.

  A package is just a archive which essentially contains files, the location where these files are placed, and meta-information like the description of the package and needed dependencies. When we install a package, for example, using the tool dpkg on Debian systems, basically extract the files contained in the package to the file system. Also store some information about where all the package related files are installed, to provide the uninstall or update option.

  So, the installation is just the extraction of the files contained in the package to the file system. In order to get this application working properly, the binaries and libraries must be specific to this platform.

  But, how we get this binaries for every platform? The answer to this question is already explained before: building the source code in every platform and then

packaging all the generated files into the new archive. Each system has its own tools for doing this operations.

- Distribute this packages to the final users

  Once we have the packages, we have to provide final users an easy way to install it. There are two ways:

  - Store the packages in a common ftp server and tell final users to download them, and then install all these packages manually with tools like dpkg on Debian or yum on Fedora/Red Hat.
  - Create a repository for this packages. This is the recommended practice, because of:
    - Secure: Usually use gpg keypair repository verification, ensuring the identity of the packages.
    - Simple: Once the repository uri is loaded in the package manager, it is easy to install, remove or update packages, just by running a command on the terminal.
    - Organized: Characterized by having a directory structure which means, better organization, and availability to have multiple versions of the same package, also, for different distributions of the operating systems.
    - Default: by setting a repository, you don't need anything more than a package manager, which is usually included in de S.O by default. Its the case of APT for Debian systems and yum for Red Hat / Fedora systems.

2.4 Solution Approach

We already know the most important tips about SINGULAR, packages management on Linux systems and the way for distribute them.

Again, every package is platform-dependent, this means, that to generate, for example, a Ubuntu i386 package, we have to build this package in a Ubuntu i386 system.

Also, every Operating System has its own package management tools, and then, their own types of packages with different package repository tools.

All this steps will be explained for every system, in Chapter 3.

Resuming, our research can be divided in steps:

- 1- Prepare the build hosts

  Install all the necessary tools which will be used in the scripts and SINGULAR dependencies such as libraries and tools. Also do a system update in order to prevent fails due to old tools versions.

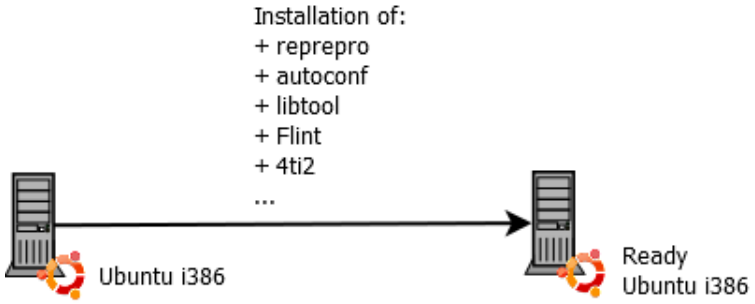

Figure 2.4: Step 1: Prepare the build hosts

- 2- Build the binary packages

  Once the system is ready, with all the needed dependencies and tools installed, its time to start the building process. The input of this process will be the singular source code (can be downloaded through the ftp server or merged from git repository). The output, the binary packages.

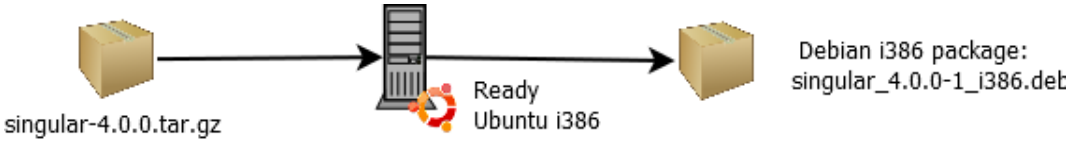

Figure 2.5: Step 2: Build the binary packages

- 3- Make the packages public

  Now, we have to create the repository. The goal of this step is to create a repository which can be used by the default package managers to install the packages, for example APT in Debian systems or YUM in Fedora/Red Hat systems. When the repository is created with its specified directory structure and configurations files, we just have to make it available to the public, for example, by placing it on a web/ftp server.

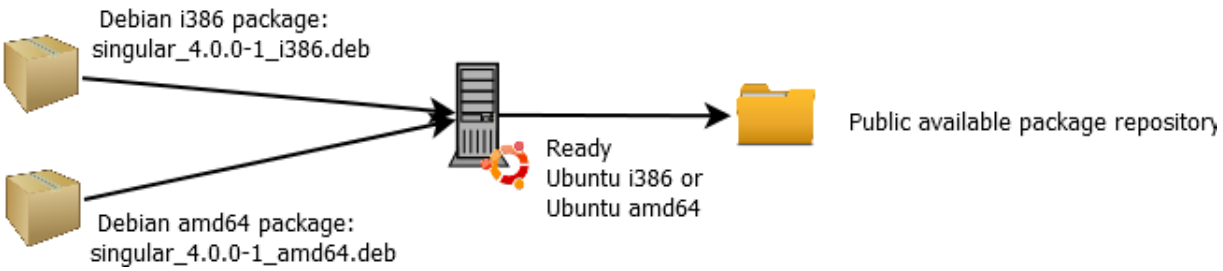

Figure 2.6: Step 3: Make the packages public

Chapter 3

Realization

3.1 How to prepare the build hosts

The first step before starting to build the packages and to create the repositories is running the setup script.

This script should leave the system ready for the next scripts, by installing all the required tools and dependencies.

3.1.1 Debian

Debian systems use APT by default, a package management system. We will use it to install the tools on the system. Some other extra libraries, has to be installed from source, as there are no available trough APT.

First of all, lets do a upgrade of the system. This will update all the tools and libraries to their last version, avoiding future errors. `update` is to refresh the APT repository database and `upgrade` to update the installed packages.

```
apt-get update -y apt-get upgrade -y
```

- Software and libraries necessary to build the binary packages
  - build-essential
  - autoconf
  - autogen
  - libtool

  - libreadline6-dev
  - libglpk-dev
  - libgmp-dev
  - libmpfr-dev
  - libcdd-dev
  - libntl-dev
  - graphviz
  - debhelper
  - flint (install from source code)
  - 4ti2 (install from source code)

- Tools to create an APT repository:
  - gpg
  - reprepro

- Other tools (e.g. needed for the correct execution of the scripts)
  - wget
  - expect

3.1.2 Fedora / Redhat

Yum package management system is used by default in Red Hat / Fedora system. We will use it to install the tools on the system. There are some other extra libraries, which have to be installed from source, as there are no available trough YUM.

As in Debian systems, its recommended to do a upgrade of all the packages installed on the system. This will prevent possible future errors. Running the `update` option for yum is enough.

```
yum update -y
```

- Software and libraries necessary to build the binary packages:
  - gcc
  - gcc-c++
  - autoconf

  - autogen
  - libtool
  - readline-devel
  - gmp-devel
  - mpfr-devel
  - cddlib-devel
  - ntl-devel
  - graphviz
  - byacc
  - rpm-build
  - 4ti2 (install from source code)
  - flint (install from source code)

- Tools to create a YUM repository
  - createrepo
  - gpg
  - rpm-sign

- Other tools (e.g. needed for the correct execution of the scripts)
  - wget
  - expect

3.1.3 Windows

SINGULAR has to be executed under a Unix-like environment. Cygwin will be used to do this. The problem, is that there is not any default package manager installed by default. The only way for installing packages by default is using the cygwin setup.exe, therefore, is impossible to do the setup process automated.

One of the best alternatives is to install APT-cyg which provides APT-like functionality on cygwin.

As APT-cyg installation is not available trough cygwin setup, we have to install it manually. For that, previously has to be installed `subversion` and `wget`, which can be installed using the setup.exe:

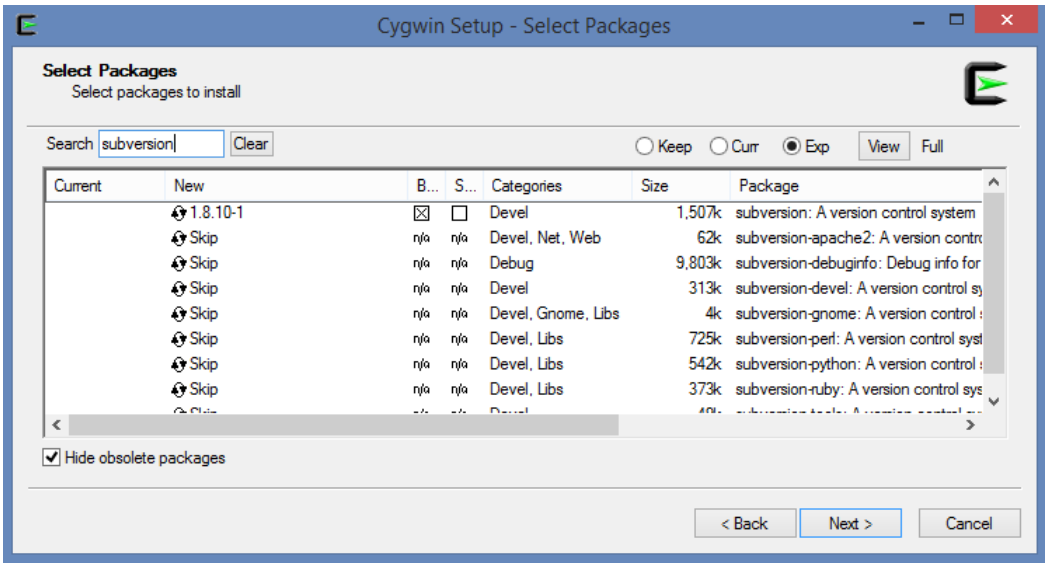

Figure 3.1: subversion install on cygwin

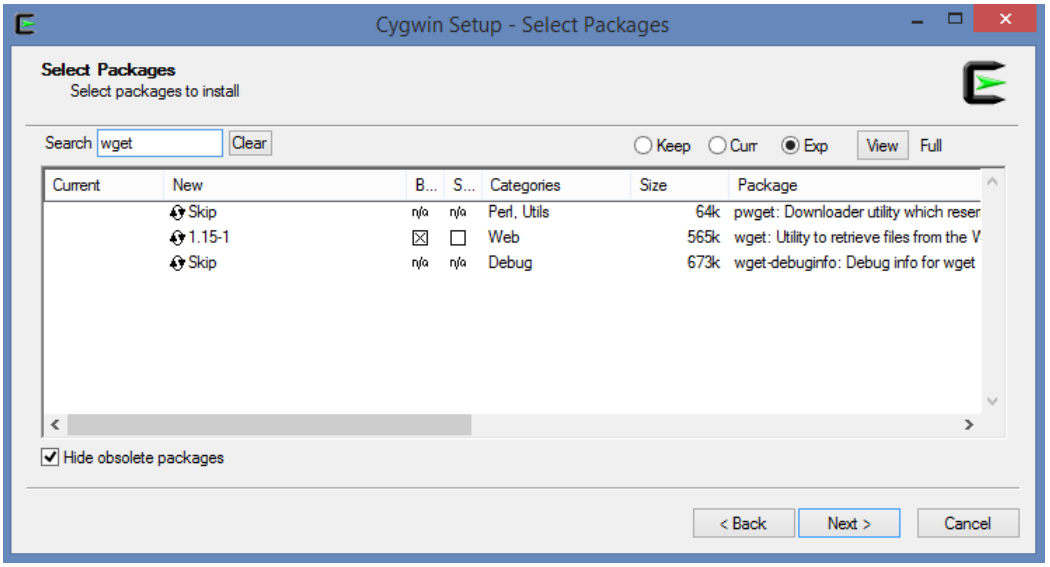

Figure 3.2: wget install on cygwin

Once installed, we can run the script. To run the script, the first step will be installing apt-cyg by using subversion and wget. Now, the remaining tools can be installed by using apt-cyg.

- Required tools in order to run the setup script (to be installed using cygwin setup):
  - subversion
  - wget

- Software and libraries necessary to build the binary packages:
  - gcc-core
  - gcc-g++
  - automake
  - make
  - libtool
  - libreadline7
  - readline-devel
  - libgmp-devel
  - libmpfr-devel
  - mpfr
  - ntl (install from source code)
  - ccdlib (install from source code)
  - 4ti2 (install from source code)
  - flint (install from source code)
  - cygport
- Other tools (e.g. needed for the correct execution of the scripts):
  - libarchive-tar-perl
  - openssh

3.2 How to build the packages

When everything is ready, after the execution of the setup script, it is time to build the packages. Here are the instructions for every O.S.

3.2.1 Debian

Debian use *.deb packages. The recommended method to build the binary packages is by using the command `debuild` in the folder where is located the source code.

This tool needs a set of instructions, in this case, contained inside of multiple files in the `/debian` folder. This files were already created by SINGULAR team.

Some additional information about Debian Packaging is available on the bibliography.

Steps:

- 1- Download the source code package
- 2- Rename the source package name, following the debian standards, ex. singular-4.0.0.tar.gz to singular_4.0.0.orig.tar.gz
- 3- Decompress the source code
- 4- Run debuild command on the extracted folder, where the source code is located. The options -us -uc will have to be used in order to avoid the gpg signing of the package.

If everything goes correctly, we should get in the same folder, the *deb packages.

3.2.2 Fedora / Red Hat

The tool used in Fedora / Red Hat systems to build the binary packages from the sources is `rpmbuild`.

As `debuild` from Debian, `rpmbuild` need a set of instructions, in this case, contained in the `*.spec` file. This file is already created by SINGULAR team, and placed inside the source code package in the `/redhat` folder. The Official Fedora Packaging guide is also available at the Bibliography.

The steps for Fedora/Red Hat are similar to Debian steps. To build the package we just have to download the source code and run the `rpmbuild` command.

Steps:

- 1- Download the source code package
- 2- Run `rpmbuild` with the options `-tb`. Option `-t` for specify the source code is in a tarball and `-b` to produce the binary packages.

  If everything goes fine, we will get in the user folder `$HOME/rpmbuild/RPMS/` the *rpm packages.

*Note: Building the packages in i386 version of Fedora caused errors related with debug dependency libraries when trying to install the packages.

That was fixed by adding the line `%debug_package %{nil}"` to the file .rpmmacros in the user folder.

3.2.3 Windows

Windows cygwin is a little bit different. Cygwin is a UNIX environment running on windows. But this environment can not run native Linux-Unix applications, only can run "ported" Linux-Unix applications.

In case of cygwin, we have the tool `cygport`, whose operation is similar to `debuild` and `rpmbuild` despite additional steps, which are performed in order to port the application.

This process will create also a cygwin binary package (*.tar.xz).

As the other tools, `cygport` also need a set of instructions, contained this time on the `singular.cygport` file.

Unlike the previous systems, `singular.cygport` was not already created.

Cygport file contains a set of instructions but also could contain some meta-information about the package.

The content of our `singular.cygport` file is:

Listing 3.1: singular.cygport File

```
DESCRIPTION="Singular"

SRC_URI="file:///tmp/wd_singular/cygport/singular-4.0.0.tar.gz"
SRC_DIR="singular-4.0.0"
USE_DESTDIR=0
MAKEOPTS+=" -j2"

src_compile()
{
    ./configure --without-python --with-ntl=/usr --with-flint=/usr
     cygmake
}

src_install() {
        cd /tmp/wd_singular/cygport/singular/src/singular-4.0.0
        cyginstall
}
```

The minimum information is the location of the source package, the name of the folder which contains the source code inside the source package and if we use a custom destination directory for cygport.

We specify some extra options, like the number of threads with `-j2` in the MAKEOPTS field and also some other options for the compiling process, like the location of some dependencies.

- 1- Download the source code package
- 2- Download the `singular.cygport` file
- 3- Run the `cygport` command.

The usage, as shown in the manual is easy:

```
cygport [OPTIONS] CYGPORT_FILE COMMAND [COMMAND2 ..]
```

This command need a external file, `CYGPORT-FILE` in this case "singular.cygport" and the steps (`COMMAND`) you want to execute over the source code.

The available commands are:

- `download`: Just download the source package file specified in the cygport file.
- `prep`: Create port project working directory structure and extract the source package contents.
- `compile` Compile the source code, with the configuration options mentioned on the cygport file.
- `test` Run the package test suite, if exists.
- `install` Install into the DESTDIR, specified on the cycport file, and run the post-install steps (if exists).
- `package` Create binary and source packages once installed.
- `finish` Delete the porting project working directory.
- `all` Execute all the previous steps excluding `finish` step.

We will use the option `all`, which will produce the final packages from the source code.

```
cygport singular.cygport all
```

If everything goes correctly, we will get the final `*tar.xz` packages on `/tmp/wd_singular /cygport/singular*/dist/singular` directory.

3.3 How to setup the repository

The goal of this step is to provide an easy installation of SINGULAR for the final users. The recommended way to follow is to use the default tools provided by the different operating systems for package managing.

3.3.1 Debian

As we said in the previous section, Debian use *.deb packages. These packages are managed in the system by default by the APT package manager.

This tool uses its own and third party repositories servers in order to fetch the new packages.

Therefore, the recommended way is to create an APT repository and make it available to the public, so then, final users can easily install SINGULAR on their systems by adding the repository.

The tool `reprepro` will create and update our repository.

If we take a look at any of the default included repositories of APT, we can see that the same repository contains not only the binary packages for different architectures, but also for different Debian distributions.

This distributions are defined by a "Codename", like "wheezy" or "squeeze" for Debian.

Our script, by using environment variables and `reprepro` is capable of:

- Crate the repository if it is not already created on the ftp server.
- Update the repository when trying to add new binary packages, by creating a new distribution / architecture / version in the repository.
- If the new binary package distribution, version and architecture, already existed, just delete the old one and place the new one.
- All the packages are signed with a gpg keypair, whose public key will be available on the repository to guarantee the identity of them.

The result will be a repository

```
OS_NAME
├── DISTR_1
│   └── ARCH_X
└── DISTR_2
    ├── ARCH_X
    └── ARCH_Y
```

If we run the script on ubuntu trusty i386 and amd64, but also in ubuntu precise amd64, a ubuntu repository will be created on our ftp server with the following "high - level" structure;

```
ubuntu
├── trusty
│   ├── i386
│   └── amd64
└── precise
    └── amd64
```

Script workflow:

- 1- Set the script parameters

  First of all, the beginning contains some variables, which will be used in the rest of the script:

  - `sshuser`: Username which will be used in the ssh commands. Needs to be edited manually before executing the script.
  - `password`: Passphrase assigned to the gpg keypair. Needs to be edited manually before executing the script.
  - `machinetype`: Environment variable obtained automatically from the command `uname -i`. Contain the type of the processor, `i686` or `x86_64`.
  - `architecture`: Derivated from the `machinetype` variable. Will be `i386` if `machinetype` is `i686` or `amd64` if `machinetype` is `x86_64`.
  - `codename`: Contains the codename of the current operating system distribution, e.g. wheezy, squeeze, trusty, etc. This variable is also get automatically from the environment, from the `lsb_release` command.
  - `systemname`: The name of the current operating system, e.g. `ubuntu`, `debian`, etc. As the previous variables, its automatically obtained from the environment, this time by from the file `/etc/os-release`.

    Thanks to this variables, our script is portable. It should update the repository with the current operating system, distribution and architecture in an

automated way. Also, the other parameters like the gpg pasphrase and the ssh username make our script customizable in case we change this gpg keypair or we use different ssh username to connect to our file server.

- 2- Download and import gpg keys

  We will use these keys for sign the packages. This step is mandatory in order to use our repository with apt.

  These keys have been previously created, as explained in the Appendix A.

  *Note: we use the same keys to sign all the packages, every time we run the script. This will make everything easier and all the repository will be signed under one unique keypair.

  The first step is to download from our private file server by using scp the public (`"public.gpg"`) and the private key, (`"secret.gpg"`).

  Then, we have to import the public key, what can be done by running the following command in the location where is we downloaded the public key `"public.gpg"`:

  ```
  gpg --import public.gpg
  ```

  Next step is to import the private key. As with the public key, we can do it running a command in the location where the gpg private key `"secret.gpg"` is located.

  ```
  gpg --allow-secret-key-import --import secret.gpg
  ```

  Finally, for security reasons, we remove the `*.gpg` files:

  ```
  rm *.gpg
  ```

- 3- Download the current repository

  If there is already an existing repository in our ftp server for the current operating system, download it to our current directory.

  Like our ftp server is also our private file server, accessible by `ssh`, we can do the copy process by using `scp`.

  If does not exist, simply, create the base directory structure for a reprepro repository:

  ```
  mkdir -p repo/$systemname/conf
  ```

- 4- Create `options` file and update/create `distributions` file:

  This files are placed in the `/conf` folder of our repository and contain meta information and instructions used by `reprepro`:

  - `options` file:

    Listing 3.2: Options File

    ```
    verbose
    basedir /tmp/wd_singular/repo/$systemname
    ask-passphrase
    ```

    This file is automatically generated by using `cat` command and contains a set of instructions used by reprepro.

    These instructions are:

    * `verbose`: To show the reprepro steps on the terminal.
    * `basedir /tmp/wd_singular/repo/$systemname`: Contains the absolute path to the repository location.
    * `ask-passphrase`: Request reprepro to use the traditional password command-line input method. This instruction is completed by the next command, executed later on script, which disables completely the password request dialog window:

      ```
      unset GPG_AGENT_INFO
      ```

      All this is to make the script completely automatic and let `expect` enter the gpg passphrase automatically and to not to prompt the user.

  - `distributions` file:

    Listing 3.3: Distributions File

    ```
    Origin: Singular
    Label: Singular
    Architectures: i386 amd64
    Components: main
    Description: APT repository for Singular
    SignWith: yes
    Codename: $codename
    ```

    Here is the meaning of every field:

    * `Origin`: The project name
    * `Label`: The project label we want to use.
    * `Codename`: The operating system release, e.g. squeeze or wheezy for Debian or trusty or precise for Ubuntu, between others.

  - `Architectures`: The architectures which are supported, e.g i386, amd64.
  - `Components`: The component lists of the packages in the repository, e.g. main, contrib, or non-free.
  - `Description`: Short description of the repository.
  - `SignWith`: Instructs reprepro that these packages should be signed. The yes is sufficient as the ubuntu user only has the one key generated above. If you have more than one key you can specify the ID of the signing key with this field.

  *Note: We have to repeat this section as many times as needed for different distributions, so the file will be updated every time we run the script or will be created if the repository already dont exist.

- 5- Add the new packages to the repository:

  Importing new packages to the repository is easy with `reprepro includedeb` option. The "problem" we found here is when we try to include packages which already existed in the repository. This happens when we run the script two times in the same system or in systems with the same configuration (operating system version and architecture) and we try to import the same version of this packages.

  So, if we want to get in our repository the new packages every time we run the script, we have to delete this packages first:

  ```
  reprepro -A $architecture removematched $codename singular*
  ```

  This will remove all the packages whose name start with "singular*" and belong to the actual operating system distribution and architecture.

  Now we are able to import the new packages without problems, by running

  ```
  reprepro includedeb $codename $list2
  ```

  Both of these reprepro commands are runned using expect. This is because it will prompt for the gpg passphrase so expect can automatically send it.

- 6- Replace the old repository with the new one on the ftp server:

  First, we remove the current repository version using ssh and then we copy the new one.

3.3.2 Fedora / Redhat

Fedora / Red Hat systems use the `yum` package management tool by default. Yum, as APT for Debian, use external repositories in order to install new software.

These repositories differ a little bit than Debian ones. They are more simpler. Each repository contain packages for only one distribution of the operating system and one architecture.

This could be simpler if we have few packages, but could be a handicap if our repository contains lots of them...

As with APT, the recommended practice is to sign the packages with a gpg keypair, but its not mandatory because yum allow the skipping the signature verification.

The tool we will use to manage the repository is `createrepo`. This tool will only create a repository from a set of *.rpm binary packages. Also, we will use the tool `rpmsign` to sign the packages.

As for Debian, our script is dynamic and will create automatically the repository using environment variables. Also, we took as reference, the official Fedora / Red Hat repositories structure:

```
OS_NAME
├── DISTR_1
│   └── ARCH_X
└── DISTR_2
    ├── ARCH_X
    └── ARCH_Y
```

If we run the script on fedora 20 i386 and x86_64, but also in fedora 19 i386, a fedora repository will be created on our ftp server with the following "high - level" structure;

```
fedora
├── 20
│   ├── i386
│   └── x86_64
└── 19
    └── i386
```

Script workflow:

- 1- Set the script parameters

  First of all, we define some variables, which will be used in the rest of the script:

  - `sshuser`: Username which will be used in the ssh commands. Needs to be manually edited before running the script.
  - `password`: Passphrase assigned to the gpg keypair. Needs to be manually edited before running the script.
  - `architecture`: Environment variable obtained automatically from the command `uname -i`. Contains the architecture of the system, which can be `i386` or `x86_64`.
  - `systemname`: Name of the current operating system, e.g. `fedora`. Its also obtained from the environment.
  - `systemversion`: Contain the version of the current operating system. As the previous variables, is environment-dependent.

  As said before, thanks to this parameters, our script is portable and customizable, and should create repositories for Fedora / Red Hat systems and its distributions in an simple way.

  Also the gpg passpharase and the ssh username could be easily modified if needed.

- 2- Download and import gpg keys

  Fedora / Red Hat systems, as Debian, use gpg key pairs to sign their binary packages. The advantage of this is that we can use the same keys for sign both repositories.

  The first step is download / copy the public and private keys (`public.key` and `secret.key` from our private file server by using scp.

  Then, we have to import the public key, what can be done by running the following command in the location where is we downloaded the public key `"public.gpg"`:

```
gpg --import public.gpg
```

  Next step is to import the private key. As with the public key, we can do it running a command in the location where the gpg private key `"secret.gpg"` is located.

```
gpg --allow-secret-key-import --import secret.gpg
```

  Also we have to copy the public part to the rpm keystore but first we have to rename it using the following RPM key standards:

```
mv public.key RPM-GPG-KEY-singular
mv RPM-GPG-KEY-singular /etc/pki/rpm-gpg/
```

Finally, we remove the private key for security reasons:

```
rm -f *.key
```

- 3- Enable RPM package signature:

  In order to sign packages, we have to add two lines to the `.rpmmacros` file:

```
echo "%_signature gpg" >> ~/.rpmmacros
echo "%_gpg_name singular" >> ~/.rpmmacros
```

- 4- Create the repository basis:

  Now its time to create the repository directory structure and copy the *.rpm packages:

```
mkdir -p repo/$systemname/$systemversion/$architecture
```

  The packages are copied from the rpmbuild folder in the user directory to the `$architecture` folder.

- 5- Sign the packages:

  By running the following command where `$list2` is the list of *.rpm packages contained on the `$architecture` folder we will get our packages signed.

```
rpm --resign $list2
```

  In the signature process, we will be asked to enter the gpg passphrase. By this reason, we will run this command using `expect`, making the process automatic.

- 6- Generate the repository metadata:

  The repository is almost created, but we need to "initialize" it, by creating the repository metadata.

  To do this, we just have to run the command createrepo specifying the location of the packages, in this case, the `$architecture` folder:

  `createrepo /tmp/wd_singular/repo/$systemname/$systemversion /$architecture`

- 7- Update the repository to the ftp server:

  Before copying the new repository to the ftp server location using scp, we check if the repository already exists. If the repository exists, it will be removed before the process to copy the repository.

3.3.3 Windows

The way of cygwin packages are installed is using the default setup.exe installer. Then, a repository can be selected and choose the desired tools.

We will follow the instructions for create a cygwin setup.exe repository:

- 1- Set the script parameters

  First of all, we define some variables, which will be used in the rest of the script:

  - `sshuser`: Username which will be used in the ssh commands. Needs to be edited manually before executing the script.
  - `machinetype`: Environment variable obtained automatically from the command `uname -i`. Contains the type of the processor, `i686` or `x86_64`.
  - `architecture`: derivated from the `machinetype` variable. Will be `x86` if `machinetype` is `i686` or `x86_64` if `machinetype` is `x86_64`.

  By using this variables, our script will adapt automatically to the system architecture without doing any manual change on the script.

- 2- Create the repository structure:

  We will create the repository structure in our working directory, `/tmp/wd_singular/`.

  ```
  mkdir -p repo/cygwin/$architecture/release
  ```

- 3- Copy the packages to the repository:

  First, we have to copy the packages generated in the previous step and to the repository location:

  ```
  cp -r /home/tmp/cygport/singular/dist/* ./release
  ```

  Then, we have to rename the binary and source packages following the cygwin package filename format standards (`package-version-release[-src].tar.xz`):

  ```
  mv ./release/singular/singular-singular-singular.tar.xz
  ./release/singular/singular-4.0.0.tar.xz

  mv ./release/singular/singular-src.tar.xz
  ./release/singular/singular-4.0.0-src.tar.xz
  ```

- 4- Download the `setup.hint` file and the `genini` script:

  There are two files we will need in order to create our repository:

  - `setup.hint`: Each package must be submitted with a file called setup.hint. This file contains useful information about the packages which will be used in the generation of the `setup.ini` by running `genini` script.

    We already have this file created in our file server so we just have to copy it using scp to the same location as the packages.

    The contents of our setup.hint are:

    Listing 3.4: Setup.hint File

    ```
    @ singular
    category: Math
    sdesc: "A Computer Algebra System for Polynomial Computations"
    ldesc: "LARGE DESCRIPTION HERE"
    ```

    *Note: This section have to be repeated in the case of having multiple packages.

    The available fields for setup.hint are:

    ```
    # comment
    @ package
    sdesc: "some text"
    ldesc: "some text"
    skip: skip this package
    curr: version
    prev: version
    test: version
    category: name1[ name2...]
    requires: package[ package...]
    external-source: package
    ```

    *More information at the Official Cygwin Package Contributor's Guide, available in the Bibliography.

  - `genini` This script will generate the `setup.ini` file, used by the cygwin setup.exe in order to get the information about the packages inside the repository.

    This script could be downloaded from the official cygwin website, but we have a copy of it in our file server, so we will download it using scp.

    This script have to be downloaded to the root of cygwin repository (`/repo/cygwin/`).

- 5- Initialize the repository:

  We just have to run the `genini` script to generate the `setup.ini file`. The command have to include the target architecture, the location of the packages and the output file:

  ```
  perl genini --arch=$architecture $architecture/release/*
   > setup.ini
  ```

  Finally, we move the produced file, to the `$architecture` folder:

  ```
  mv ./setup.ini ./$architecture
  ```

  An example of the generated `setup.ini` file:

  Listing 3.5: Setup.ini File Sample

  ```
  # This file is automatically generated.  If you edit it, your
  # edits will be discarded next time the file is generated.
  # See http://cygwin.com/setup.html for details.
  #
  arch: x86
  setup-timestamp: 1408754869

  @ singular
  sdesc: "A Computer Algebra System for Polynomial Computations"
  ldesc: "LARGE DESCRIPTION HERE"
  category: Math
  version: 4.0.0
  install: x86/release/singular/singular-4.0.0.tar.xz 26635156 76
      b0cd5f9cfaa6fb830adfb6aa38d941
  source: x86/release/singular/singular-4.0.0-src.tar.xz 14744908 322
      c64cedd6afe2f796eecbae313688d
  ```

- 6- Copy the repository to the ftp server:

  Before copying the new repository to the ftp server location using scp, we check if the repository already exists. If the repository exists, it will be removed before the process to copy the repository.

3.4 Installation instructions

In this section, we will provide the instructions which should be provided to the end users in order to use our repositories in the following operating systems:

3.4.1 Debian

The following options have been tested with successful results in Ubuntu trusty and precise.

Steps:

- 1: Download and import gpg key, run:

  ```
  wget -O - ftp://jim.mathematik.uni-kl.de/repo/extra/gpg|
  apt-key add -
  ```

- 2: Add SINGULAR repository to apt's repo list, run:

  ```
  wget ftp://jim.mathematik.uni-kl.de/repo/extra/debian/
  singular.list -P /etc/apt/sources.list.d/
  ```

  Listing 3.6: singular.list File

  ```
  deb ftp://jim.mathematik.uni-kl.de/repo/debian trusty main
  ```

  *Note: The distribution, trusty in this case, should be modified by the current one, but only if the repository contains the packages for this distribution. Otherwise will not work. Tested with trusty on Ubuntu precise and work.

- 3: Update apt, run:

  ```
  apt-get update
  ```

- 4: Install SINGULAR, run:

  ```
  apt-get install singular
  ```

- 5: Run SINGULAR, run:

  ```
  Singular
  ```

3.4.2 Fedora / Red Hat

The following instructions have been tested on Fedora 20 with successful results.

- 1: Download SINGULAR repo file to yum.repos.d folder, run:

  ```
  wget -P /etc/yum.repos.d ftp://jim.mathematik.uni-kl.de/repo
  ```

  ```
  /extra/fedora/singular.repo
  ```

  The contents of the singular.repo file are:

  Listing 3.7: singular.repo File

  ```
  [Singular]
  name=Singular repo
  baseurl=ftp://jim.mathematik.uni-kl.de/repo/fedora/$releasever/
      $basearch
  enabled=1
  gpgcheck=1
  gpgkey=ftp://jim.mathematik.uni-kl.de/repo/extra/fedora/RPM-GPG-KEY-
      singular
  ```

  *Again, we have to be careful with the repository url. If we download this script on a Fedora 18 i386 system, and our repository has not the SINGULAR repository for Fedora 18 i386, the command yum-install will not work.

  If there is no repository for Fedora 18 i386, we can tell the users to use other repository, which also should work:

  `ftp://jim.mathematik.uni-kl.de/repo/fedora/20/i386`

  Concluding, the `baseurl`field has to be modified depending of the available repositories.

- 2: Install SINGULAR, run:

  ```
  yum install singular
  ```

- 3: Start SINGULAR, run:

  ```
  Singular
  ```

3.4.3 Windows

Cygwin install new packages by default by using the setup.exe. The version 2.573.2.3 introduced new security changes like mandatory gpg signing of the packages.

This is not a huge problem for the official "trusted" repositories, but its a problem for the maintainers of custom repositories.

As seen in the "Cygwin security changes", available on the Bibliography, cygwin proposes some possibilities, from using old setup.exe versions to gpg signing of the packages.

The reason of why we didnt signed the packages is because even having the packages signed, the users have to start the setup.exe with some arguments.

We propose two methods, one already proposed by cygwin team and another by our own:

- Using the setup.exe

  Running the setup.exe from a console followed with the option -X. This will skip the signature checking.

```
setup-(x86)(x86_64).exe -X
```

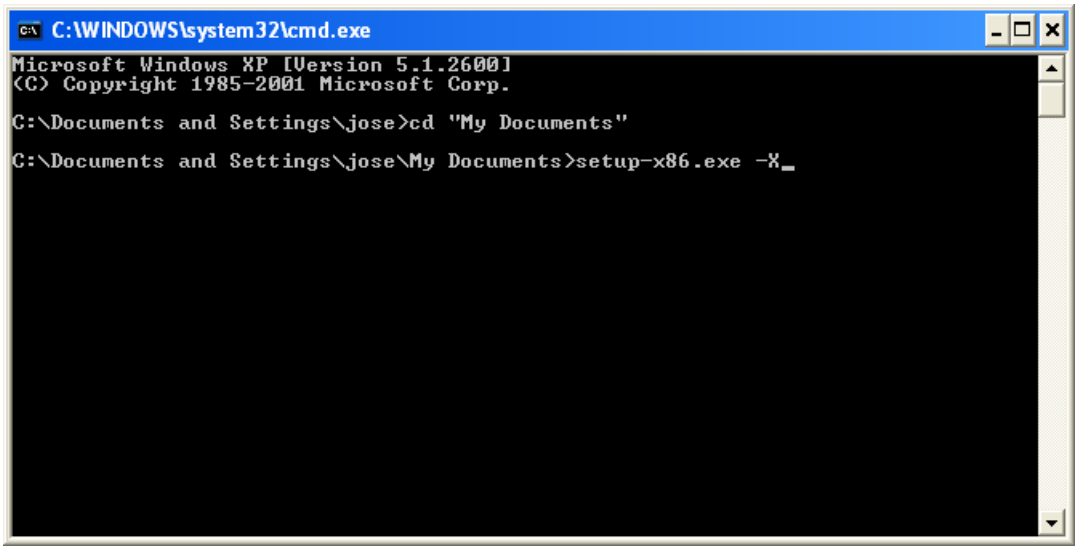

Figure 3.3: Running setup.exe with -X parameter

  Then we have to add the new repository and then select it before press "next" button:

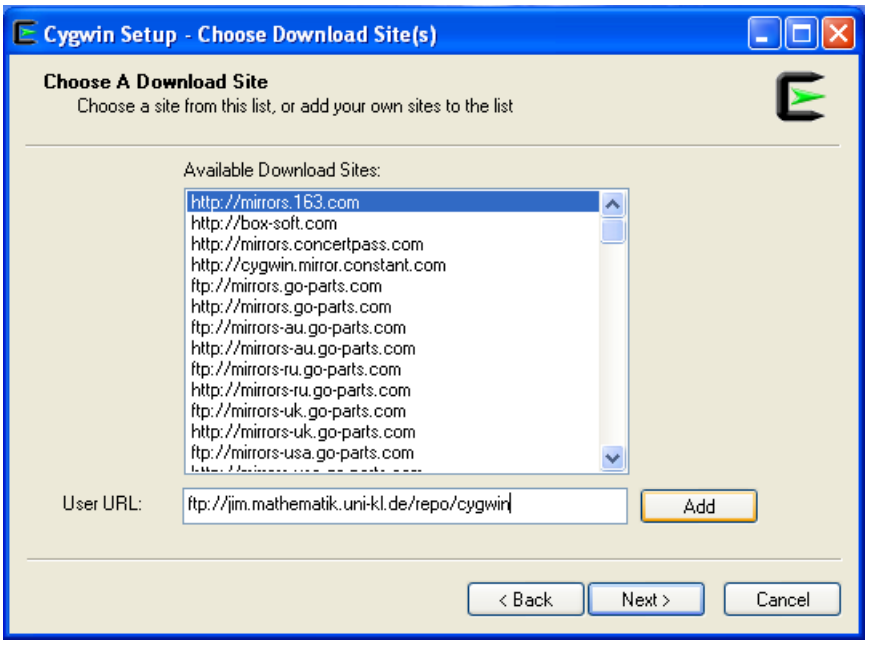

Figure 3.4: Add the new SINGULAR repository

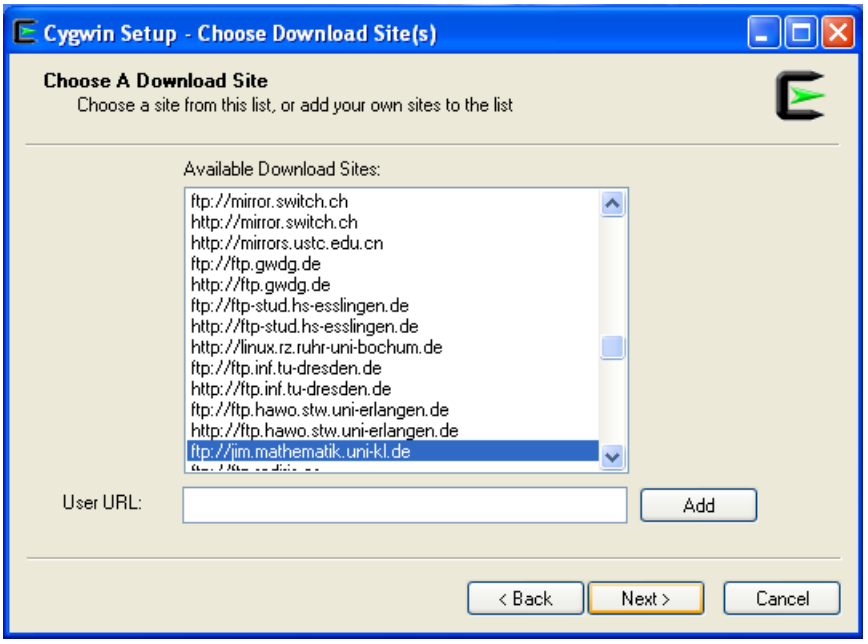

Figure 3.5: Select the new SINGULAR repository

`ftp://jim.mathematik.uni-kl.de/repo/cygwin`

Then, select the package "singular" from the list, and install it.

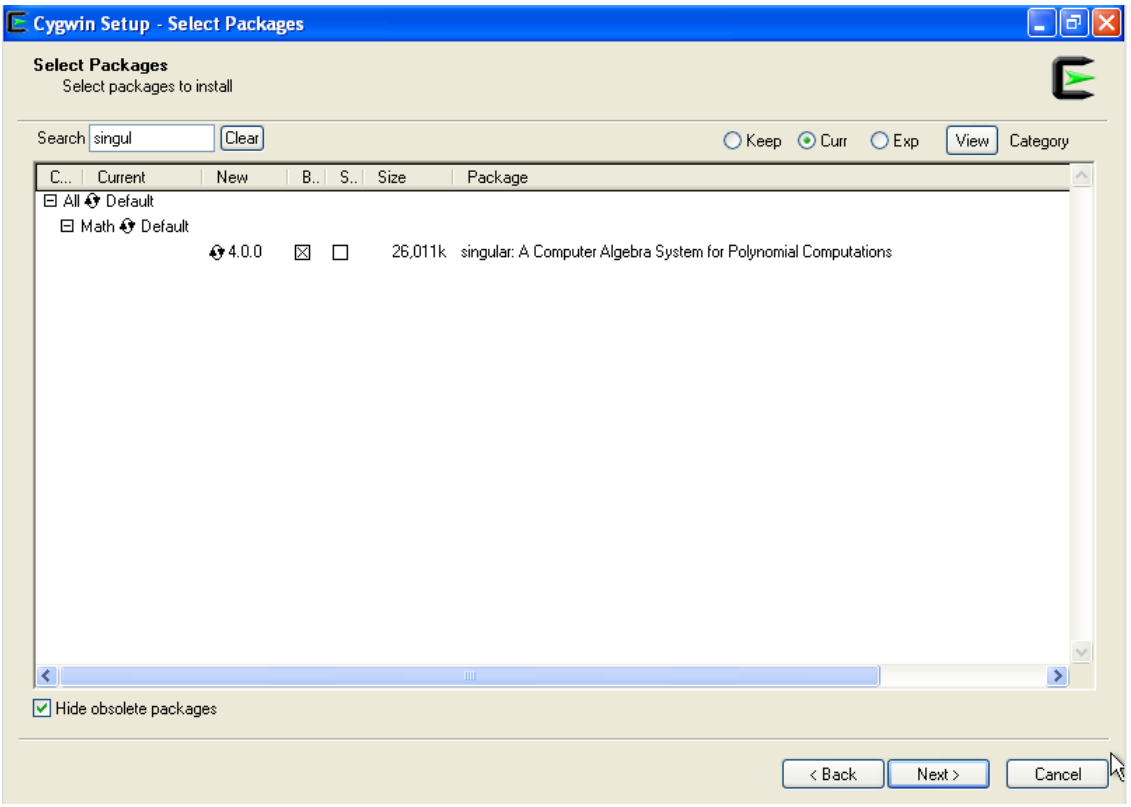

Figure 3.6: Select the package "singular"

- Using apt-cyg tool

  We can use apt-cyg tool to install our packages. First we have to install it because its not a default tool.

  First we have to install subversion and wget as shown in chapter 3.1.3.

  Then, in the cygwin terminal run the following two commands:

  ```
  svn --force export http://apt-cyg.googlecode.com/svn/trunk/ /bin/
  chmod +x /bin/apt-cyg
  ```

  Once we have apt-cyg installed on cygwin, we can proceed to install SINGULAR by running the following command:

  ```
  apt-cyg install singular -m ftp://jim.mathematik.uni-kl.de
  /repo/cygwin
  ```

  Then, we can start SINGULAR, just by running:

  ```
  Singular
  ```

Chapter 4

Testing and Evaluation

In order to develop and test the scripts we used the concept of "compiling farms" by creating virtual machines.

Oracle VirtualBox is a free and powerful tool which provides the possibility of creating virtual machines for most of the operating systems and architectures to be studied.

Using virtual machines make the process easier than using physicals computers because of the time saved installing this systems and the possibility of create copies easily.

The virtual machines provide real reliability about how the scripts will work on real scenarios. The only problem is that sometimes, depending of the host machine, could be a little bit slower.

We used these host machines during this work:

- 1- Personal computer: Conventional laptop i5 dual core processor 8GB RAM Memory and SATA2 SSD Solid Hard Disk.

  Mainly used for the development of the scripts for Debian, Fedora and Cygwin.

  The platforms used are:

  - Ubuntu 12.04.4 i386 (Oracle VB)
  - Ubuntu 12.04.4 amd64 (Oracle VB)
  - Fedora 20 i386 (Oracle VB)
  - Fedora 20 amd64 (Oracle VB)
  - Windows 8 64 bits (Native)

- 2- jim computeserver at CTM Mathematik Department UNI-KL

This computer is the host of the ftp server and also our "private file server" where we store all the additional files needed by the scripts.

Also, used for the testing of the scripts on clean install of different OS and Architectures.

This host machine provides an exceptional performance, making the tests easier and fast.

The platforms used are:

- Ubuntu 12.04.4 i386 (Oracle VB)
- Ubuntu 12.04.4 amd64 (Oracle VB)
- Fedora 20 i386 (Oracle VB)
- Fedora 20 amd64 (Oracle VB)
- Windows XP 32 bits (Oracle VB)
- Windows XP 64 bits (Oracle VB)

Chapter 5

Results

The results of this thesis are:

- This document: Contains the explanation of the problem, and all the steps we followed to reach the solution. Also, the contents of the scripts and other related files are present on its Appendix.
- Scripts:

  We use the same concept for the studied operating systems:

  - setup script: This script only needs to be run once.

    It will let the system ready for the correct operation of the main script.

    This script contains everything explained in Chapter 3.1 - How to prepare the build hosts.
  - main script: This script is meant to be run multiple times on the same machine. For this reason, all the steps will be done in a temporal working directory, `/tmp/wd_singular` which will be created at the start of the script.

    This script is meant to produce/update the ftp repository with the respective binary packages, obtained by compiling the source code.

    Contains everything explained in the chapters 3.2 - How to build the packages and 3.3 - How to setup the repository.

- Installation instructions:

  Set of steps which end users have to follow in order to use our repositories. Available on the directory `/extra` of the ftp server, under the `instructions` file name and also in the chapter 3.4 - Installation instructions.

- Test repository:

  Copy of the ftp://jim.mathematik.uni-kl.de/repo/ server. Is the result of testing our scripts.

  Contain ubuntu, fedora and cygwin functional repositories, with both architectures, 32 and 64 bits supported.

- Other files:

  Stored on the scripts folder of our file server and in the extra folder of our ftp server.

  This files have been mentioned before in the Chapter 3 and are used by the scripts or downloaded following the installation instructions.

  This files include third party scripts, configuration files and gpg keys.

This document will be printed, signed and submitted along with a CD-ROM which contain a copy of the documentation in PDF format and two folders, one containing a copy of the current repository, hosted on ftp://jim.mathematik.uni-kl.de/ and the other containing the scripts and "needed_files" folder used by scripts:

```
jim.mathematik.uni-kl.de/repo
├── ubuntu
│   └── trusty
│       ├── i386
│       └── amd64
├── fedora
│   └── 20
│       ├── i386
│       └── x86_64
├── cygwin
│   ├── x86
│   └── x86_64
└── extra
    ├── ubuntu
    │   ├── singular.list
    │   └── instructions
    ├── fedora
    │   ├── singular.repo
    │   └── instructions
    ├── cygwin
    │   └── instructions
    └── gpg
```

```
scripts
├── debian
│   ├── setup.sh
│   └── main.sh
├── fedora
│   ├── setup.sh
│   └── main.sh
├── cygwin
│   ├── setup.sh
│   └── main.sh
└── needed_files
    ├── cygwin
    │   ├── genini
    │   ├── setup.hint
    │   └── singular.cygport
    └── gpg
        ├── public.key
        └── secret.key
```

Chapter 6

Future work

This thesis is based on a real scenario. This work supposed to be used in the future to distribute the version 4.0 of SINGULAR.

There are many ways to use this scripts, and the way its going to be used is not specified at 100

All the steps are explained as best as possible, so SINGULAR team can use the whole scripts, which are ready to produce the final repositories or just take some parts like the binaries generation, in order to create a custom script if needed.

Before finishing, some useful tips:

- 1- First, we have to know that most of the scripts have some parameters on top. This parameters are already filled with our "test" gpg passphrase and ssh username.

  The first thing to do should be to generate again a gpg keypair as explained in the Appendix A.

  Then, create a ssh username on the file server without sessions limit for avoid errors and before running the scripts follow the Appendix B, Set Up SSH Keysm for using ssh on the virtual machines without enter the password.

  Obviously, some files like the gpg keys should be replaced in the file server with the final ones.

- 2- The virtual machines are probably the best way to get the repositories and packages. We can run setup scripts only one time in our VMs and then we can run the main scripts all the times we want.

Also, by using these VMs exclusively for this proposal, we will prevent getting external errors due to the environment.

- 3- This scripts have been tested on Ubuntu trusty, Fedora 20 and Cygwin under Windows XP with satisfactory results.

  We cant guarantee this scripts will work with every version of Ubuntu, Fedora and Windows, but should. Also they should work in Debian, RedHat and more similar Linux distributions, because we tried to use relative variables on our scripts.

- 4- If finally the decision of using the scripts is made, should be implemented a backup system of the ftp server. This is because some error can be produced which can let the repository damaged.

Based on this tips, this is just one example of how the implementation of this concepts can look in a real scenario:

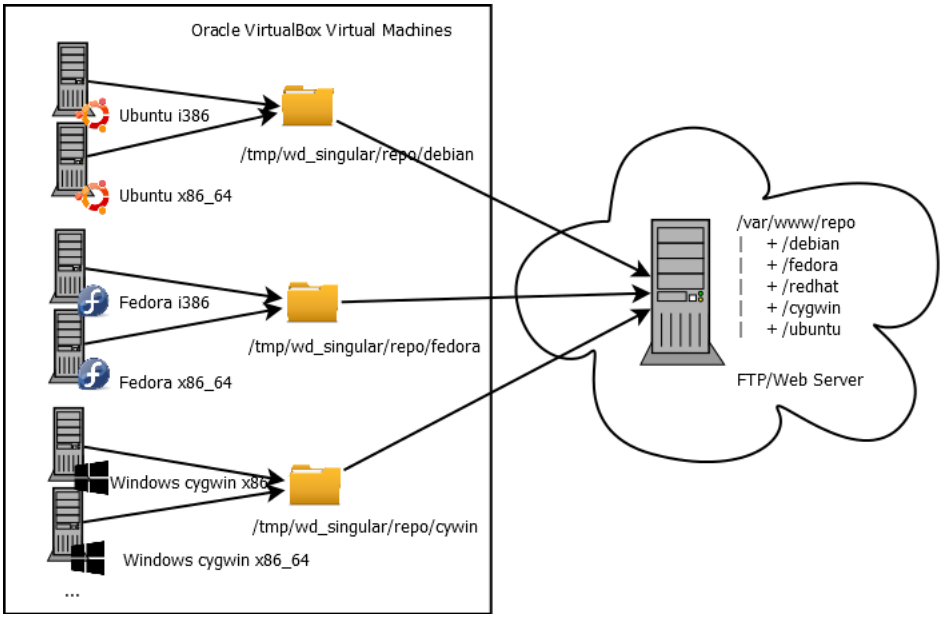

Figure 6.1: Real Scenario Implementation Concept

Appendix A

GPG keypair generation

The following instructions will show how to generate a gpg keypair and how to export and import them.

Type the next command:

```
gpg --gen-key
```

It will ask for chose between 4 options:

```
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
```

We choose option number one (1)

Now, it will ask for the length of the key:

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
```

Just press enter for the default value (2048)

Next. The validity of the key:

```
Please specify how long the key should be valid.
         0 = key does not expire
```

```
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
```

Press enter for default (never expire)

```
Is this correct? (y/N)
```

Type y and press enter

Now, it will ask for some "personal" values (Real Name, Comment and email address of the signer).

For example:

```
- Real name: Jose Antonio Martin
- E-Mail: joseantonomh1992@gmail.com
- Comment: Student of the Mathematic Department

Next press (O)kay for continue
```

Now, it will ask for a create a passphrase. A complicated one is recommended. We used "pass" for this example.

Now, maybe will ask for create some workload in order to generate random bytes. Just open programs, transfer files, whatever you want.

A small trick is to execute this command in a new terminal:

```
dd if=/dev/zero of=tmp/5gb.tmp bs=1G count=5
```

This will create a 5gb file in /tmp. Just delete after the key generation.

We will export public and private keys and store it in the private file server, in order to be downloaded and imported by the scripts.

First, we list the our keys:

```
gpg --list-keys
```

Now we can get the ID of our keypair, in this case, 91DB2771.

```
jose@jose-VirtualBox:/tmp$ gpg --list-keys
```

```
/home/jose/.gnupg/pubring.gpg
------------------------------
pub   2048R/1E05ACF7 2014-05-22
uid                  Jose Antonio Martin (Student of Mathematic Department)
sub   2048R/91DB2771 2014-05-22
```

Export the public and private keys by running the following commands:

```
gpg --output public.gpg -armor --export 91DB2771
gpg --output secret.gpg -armor --export-secret-key 91DB2771
```

Now we can store it in our private file server. The import of the keys is already done on the scripts explanation.

Appendix B

Set Up SSH Keys

The following instructions will show how to set up ssh keys. This procedure is MANDATORY before running the scripts.

This is because the scripts often use ssh or scp to copy files or to run some command in the file server, like file in the case of modifying the ftp directory.

SSH is a cryptographic network protocol for secure data communication, remote command-line login, remote command execution, and other secure network services between two networked computers.

Its easy to understand how it works. User A in computer A want to connect to B as the user B.

This connection has to be secure and in order to grant it, we can just enter the user b password in every command or we can just create a ssh keys in computer A as user A (in our case, as root user) and then add them in the file authorized_keys on the user B folder on the computer B.

By doing this, we will prevent password prompts and we can run ssh and scp commands without entering the password. First of all, make you are using root user:

```
su root
```

This is because the keys will be valid only for the user who generate them. We need root user for running the scripts.

Then, we generate our ssh keys:

```
ssh-keygen -t dsa
```

Next, we import the keys to the authorized_keys file of the user X, in our case, jamartin:

```
cat ~.ssh/id_dsa.pub | ssh jamartin@jim 'cat >> .ssh/authorized_keys'
```

This will be the last time we will be asked for the password.

Finally, just lets refresh ssh by running:

```
ssh-add
```

Now, we can try to login in the remote computer by running:

```
ssh jamartin@jim
```

If doesn't ask for a password, seems that everything is O.K. and the scripts, ready to be executed.

Appendix C

Scripts sources

Listing C.1: Setup.sh Debian Script

```
#!/bin/sh

################## DEBIAN SETUP ##########################

# Test if the script is running under superuser permissions
if [ "$(id -u)" != "0" ]; then
  echo "Sorry, you must run this script under superuser permissions"
  exit 1
fi

apt-get update -y
apt-get upgrade -y

apt-get install devscripts -y
apt-get install build-essential -y
apt-get install autoconf -y
apt-get install autogen -y
apt-get install libtool -y
apt-get install libreadline6-dev -y
apt-get install libglpk-dev -y
apt-get install libgmp-dev -y
apt-get install libmpfr-dev -y
apt-get install libcdd-dev -y
apt-get install libntl-dev -y
apt-get install graphviz -y
apt-get install debhelper -y

apt-get install gpg -y
apt-get install reprepro -y
```

```
apt-get install wget -y
apt-get install expect -y


cd /tmp
rm -rf wd_singular
mkdir wd_singular
cd wd_singular

mkdir flint
cd flint
wget ftp://www.mathematik.uni-kl.de/pub/Math/Singular/src/4-0-0/flint
    -2.4.4.tar.gz
tar xvfz flint-2.4.4.tar.gz
cd flint-2.4.4
./configure --with-gmp=/usr --prefix=/usr --with-mpfr=/usr
make -j 2
make install
cd ../../

mkdir 4ti2
cd 4ti2
wget http://www.4ti2.de/version_1.6/4ti2-1.6.tar.gz
tar xvfz 4ti2-1.6.tar.gz
cd 4ti2-1.6
./configure
make
make install
cd ../../


echo "Everything installed"
```

Listing C.2: Main.sh Debian Script

```
#!/bin/sh

################## DEBIAN MAIN ##########################

# Test if the script is running under superuser permissions
if [ "$(id -u)" != "0" ]; then
  echo "Sorry, you must run this script under superuser permissions"
  exit 1
fi

#-------------------------------------------------------------------------
#                  PARAMETERS

# ssh jim user
sshuser='jamartin'

# gpg passphrase
password='pass'

# system architecture
machinetype=`uname -i`

if [ ${machinetype} == 'x86_64' ]; then
architecture='amd64'
else
architecture='i386'
fi

# system codename
codename=$(lsb_release -c | cut -f2)

# system name
systemname=$(. /etc/os-release; echo $ID)

#-------------------------------------------------------------------------
#             BUILDING *.DEB PACKAGES

# Create working directory
cd /tmp
rm -rf wd_singular
mkdir wd_singular
cd wd_singular

# Generating Singular binaries
mkdir debSingular
cd debSingular
```

```
wget ftp://www.mathematik.uni-kl.de/pub/Math/Singular/src/4-0-0/singular
    -4.0.0.tar.gz
mv singular-4.0.0.tar.gz singular_4.0.0.orig.tar.gz
tar xzvf singular_4.0.0.orig.tar.gz
cd singular-4.0.0
debuild -us -uc


#---------------------------------------------------------------------
#             DEBIAN REPOSITORY SETUP

# Creating repository basis and importing gpg keys

cd /tmp/wd_singular

scp $sshuser@jim:/tmpbig/jamartin/scripts/needed_files/gpg/{public.key,
    secret.key} ./

gpg --import public.key
gpg --allow-secret-key-import --import secret.key
rm *.key
mkdir -p repo/$systemname/conf
scp -r $sshuser@jim:/home/ftp/repo/$systemname ./repo/

options='./repo/'"$systemname"'/conf/options'

/bin/cat <<EOM >$options
verbose
basedir /tmp/wd_singular/repo/$systemname
ask-passphrase
EOM

distributions='./repo/'"$systemname"'/conf/distributions'

if [ ! -f $distributions ]; then
    /bin/cat <<EOM >$distributions
Origin: Singular
Label: Singular
Architectures: i386 amd64
Components: main
Description: APT repository for Singular
SignWith: yes
Codename: $codename
EOM
else

if ! grep -q "Codename: $codename" "$distributions"; then
    /bin/cat <<EOM >>$distributions
```

```
Origin: Singular
Label: Singular
Architectures: i386 amd64
Components: main
Description: APT repository for Singular
SignWith: yes
Codename: $codename
EOM

fi

fi

# Deb files to be imported to the repository
deblist=(ls /tmp/wd_singular/debSingular/${prefix}*.deb)
list=${deblist[*]}
list2=${list/ls /}

# Naming the present distribution and modification of the "distributions"
    file
cd repo/$systemname

# Prevent reprepro open password dialog
unset GPG_AGENT_INFO

# Automated reprepro execution with self password send using expect
# Download actual repository, remove only the packages related with the
    present architecture and distribution, if exists.
expect -c '
set timeout 5
spawn '"reprepro -A $architecture removematched $codename singular*"'
expect {
  "Please enter passphrase:" {
    send '"$password\r"'
    exp_continue
  }
}
sleep 1
exit
'

expect -c '
set timeout 5
spawn '"reprepro includedeb $codename $list2"'
expect {
  "Please enter passphrase:" {
    send '"$password\r"'
```

```
    exp_continue
  }
}
sleep 1
exit
'

#Update the repository to the ftp server
ssh $sshuser@jim 'rm -rf /home/ftp/repo/'"$systemname"''
scp -r /tmp/wd_singular/repo/$systemname $sshuser@jim:/home/ftp/repo/
```

Listing C.3: Setup.sh Fedora Script

```
#!/bin/sh

################## FEDORA SETUP ############################

# Test if the script is running under superuser permissions
if [ "$(id -u)" != "0" ]; then
  echo "Sorry, you must run this script under superuser permissions"
  exit 1
fi


yum update -y

yum install gcc -y
yum install gcc-c++ -y
yum install autoconf -y
yum install autogen -y
yum install libtool -y
yum install readline-devel -y
yum install gmp-devel -y
yum install mpfr-devel -y
yum install cddlib-devel -y
yum install ntl-devel -y
yum install graphviz -y
yum install 4ti2 -y
yum install byacc -y
yum install rpm-build -y

yum install createrepo -y
yum install gpg -y
yum install rpm-sign -y

yum install wget -y
yum install expect -y


cd /tmp
rm -rf wd_singular
mkdir wd_singular
cd wd_singular

MACHINE_TYPE=`uname -i`
if [ ${MACHINE_TYPE} != 'i386' ]; then

mkdir flint
cd flint
```

```
wget ftp://www.mathematik.uni-kl.de/pub/Math/Singular/src/4-0-0/flint
    -2.4.4.tar.gz
tar xvfz flint-2.4.4.tar.gz
cd flint-2.4.4
./configure --with-gmp=/usr --prefix=/usr --with-mpfr=/usr
make -j 2
make install
cd ../../

fi


mkdir 4ti2
cd 4ti2
wget http://www.4ti2.de/version_1.6/4ti2-1.6.tar.gz
tar xvfz 4ti2-1.6.tar.gz
cd 4ti2-1.6
./configure
make
make install
cd ../../


echo "Everything installed"
```

Listing C.4: Main.sh Fedora Script

```
#!/bin/sh

################## FEDORA MAIN ##########################

# Test if the script is running under superuser permissions
if [ "$(id -u)" != "0" ]; then
  echo "Sorry, you must run this script under superuser permissions"
  exit 1
fi

#-------------------------------------------------------------------------
#                  PARAMETERS

# ssh jim user
sshuser='jamartin'

# gpg passphrase
password='pass'

# system architecture
architecture=`uname -i`

# system name
systemname=$(. /etc/os-release; echo $ID)

# system version
systemversion=$(. /etc/os-release; echo $VERSION_ID)

#-------------------------------------------------------------------------
#           BUILDING *.RPM PACKAGES

# Create working directory
cd /tmp
rm -rf wd_singular
mkdir wd_singular
cd wd_singular

# Modify .rpmmacros file to avoid debug libraries dependency error
rm -f ~/.rpmmacros
echo "%debug_package %{nil}" > ~/.rpmmacros

# Generating Singular binaries
wget ftp://www.mathematik.uni-kl.de/pub/Math/Singular/src/4-0-0/singular
    -4.0.0.tar.gz
rpmbuild -tb singular-4.0.0.tar.gz
```

```
#-----------------------------------------------------------------------
#             FEDORA REPOSITORY SETUP

# Importing gpg keys
cd /tmp/wd_singular
scp $sshuser@jim:/tmpbig/jamartin/scripts/needed_files/gpg/{public.key,
    secret.key} ./
gpg --import public.key
gpg --allow-secret-key-import --import secret.key
mv public.key RPM-GPG-KEY-singular
rpm --import RPM-GPG-KEY-singular
mv RPM-GPG-KEY-singular /etc/pki/rpm-gpg/
rm -f *.key

# Modify .rpmmacros file to enable gpg rpm signature.
echo "%_signature gpg" >> ~/.rpmmacros
echo "%_gpg_name singular" >> ~/.rpmmacros

# Creating repository
mkdir -p repo/$systemname/$systemversion/$architecture
cp -ar $HOME/rpmbuild/RPMS/$architecture/*rpm /tmp/wd_singular/repo/
    $systemname/$systemversion/$architecture
cp -ar $HOME/rpmbuild/RPMS/noarch/*rpm /tmp/wd_singular/repo/$systemname/
    $systemversion/$architecture

# Sign the rpms using expect to prevent password asking
rpmlist=(ls /tmp/wd_singular/repo/$systemname/$systemversion/$architecture
    /${prefix}*.rpm)
list=${rpmlist[*]}
list2=${list/ls /}

expect -c '
set timeout 5
spawn '"rpm --resign $list2"'
expect {
  "Enter pass phrase:" {
    send '"$password\r"'
    exp_continue
  }
}
sleep 1
exit
'
createrepo /tmp/wd_singular/repo/$systemname/$systemversion/$architecture

#Update the repository to the ftp server
ssh $sshuser@jim 'rm -rf /home/ftp/repo/'"$systemname"'/'"$systemversion
    "'/'"$architecture"''
```

```
scp -r /tmp/wd_singular/repo/$systemname $sshuser@jim:/home/ftp/repo
```

Listing C.5: Setup.sh Cygwin Script

```
#!/bin/sh

################## CYGWIN SETUP ##########################

############################################################
############################################################
#################### ATENTION ##############################
# RUN EXCLUIVLY THIS SCRIPT IF YOU HAVE INSTALLED WGET AND
# SUBVERSION IN YOUR CYGWIN INSTALLATION!
############################################################


svn --force export http://apt-cyg.googlecode.com/svn/trunk/ /bin/
chmod +x /bin/apt-cyg
apt-cyg install openssh
apt-cyg install libarchive-tar-perl

apt-cyg install gcc-core
apt-cyg install gcc-g++
apt-cyg install automake
apt-cyg install make
apt-cyg install libtool
apt-cyg install libreadline7
apt-cyg install libreadline-devel
apt-cyg install libgmp-devel
apt-cyg install libmpfr-devel
apt-cyg install mpfr
apt-cyg install cygport

cd /tmp
rm -rf wd_singular
mkdir wd_singular
cd wd_singular

cd /tmp/wd_singular
mkdir ntl
cd ntl
wget http://www.shoup.net/ntl/ntl-6.0.0.tar.gz
tar -xvf ntl-6.0.0.tar.gz
cd ntl-6.0.0
cd src
./configure DEF_PREFIX=/usr NTL_GMP_LIP=on
make
make install

cd /tmp/wd_singular
```

```
mkdir cddlib
cd cddlib
wget ftp://ftp.ifor.math.ethz.ch/pub/fukuda/cdd/cddlib-094g.tar.gz
tar -xvf cddlib-094g.tar.gz
cd cddlib-094g
./configure --prefix=/usr
make -j 2
make install

cd /tmp/wd_singular
mkdir flint
cd flint
wget http://www.flintlib.org/flint-2.4.3.tar.gz
tar xvfz flint-2.4.3.tar.gz
cd flint-2.4.3
./configure --with-gmp=/usr --prefix=/usr --disable-shared --with-mpfr=/
    usr
make -j 2
make install

cd /tmp/wd_singular
mkdir 4ti2
cd 4ti2
wget http://www.4ti2.de/version_1.6/4ti2-1.6.tar.gz
tar xvfz 4ti2-1.6.tar.gz
cd 4ti2-1.6
./configure
make
make install
```

Listing C.6: Main.sh Cygwin Script

```
#!/bin/sh

################## CYGWIN MAIN ##########################

#-------------------------------------------------------------------------
#                   PARAMETERS

# ssh jim user
sshuser='jamartin'

# system architecture
machinetype=`uname -m`

if [ ${machinetype} == 'x86_64' ]; then
architecture='x86_64'
else
architecture='x86'
fi


#-------------------------------------------------------------------------
#            BUILDING CYGWIN PACKAGES

# Create working directory
cd /tmp
rm -rf wd_singular
mkdir -p wd_singular/cygport
cd wd_singular/cygport

# Generating Singular binaries
wget ftp://www.mathematik.uni-kl.de/pub/Math/Singular/src/4-0-0/singular
    -4.0.0.tar.gz
scp $sshuser@jim:/tmpbig/jamartin/scripts/needed_files/cygwin/singular.
    cygport ./
cygport singular.cygport all

#-------------------------------------------------------------------------
#              CYGWIN REPOSITORY SETUP

# Creating repository basis
cd /tmp/wd_singular
mkdir -p repo/cygwin/$architecture/release
cd repo/cygwin/$architecture/release
cp -r /tmp/wd_singular/cygport/singular*/dist/* ./
mv ./singular/singular-singular-singular.tar.xz ./singular/singular-4.0.0.
    tar.xz
```

```
mv ./singular/singular-src.tar.xz ./singular/singular-4.0.0-src.tar.xz
scp $sshuser@jim:/tmpbig/jamartin/scripts/needed_files/cygwin/setup.hint
    ./singular/

# Download genini script for create the repository
cd /tmp/wd_singular/repo/cygwin
scp $sshuser@jim:/tmpbig/jamartin/scripts/needed_files/cygwin/genini ./
perl genini --arch=$architecture $architecture/release/* > setup.ini
mv ./setup.ini ./$architecture

#Update the repository to the ftp server
scp -r /tmp/wd_singular/repo/cygwin $sshuser@jim:/home/ftp/repo/
```

Appendix D

Instructions sets

Listing D.1: Instructions set file Ubuntu

```
STEPS to install SINGULAR:

1: Download and import gpg key, run:

wget -O - ftp://jim.mathematik.uni-kl.de/repo/extra/gpg|apt-key add -

2: Add Singular repository to apt's repo list, run:

wget ftp://jim.mathematik.uni-kl.de/repo/extra/debian/singular.list -P /
    etc/apt/sources.list.d/

*please note that you should modify this file, by changing trusty for the
    codename of your ubuntu version. But first, check if our repository
    support your version.

3: Update apt, run:

apt-get update

4: Install Singular, run:

apt-get install singular

5: Run Singular, run:

Singular
```

Listing D.2: Instructions set file Fedora

```
STEPS to install SINGULAR:

1: Add Singular repository to yum's repo list, run:

wget ftp://jim.mathematik.uni-kl.de/repo/extra/fedora/singular.repo -P /
    etc/yum.repos.d/

*please note that if you get an error in the next step, maybe is because
    our repository sill dont support your version. In this case, edit the
    downloaded /etc/yum.repos.d/singular.repo file and change the
    $releasever and $basearch for ones contained in our repository and
    which could support your system, like /fedora/20/i386 for 32bits or /
    fedora/20/x86_64 for 64 bits.

2: Install Singular, run:

yum install singular

3: Run Singular, run:

Singular
```

Listing D.3: Instructions set file Cygwin

```
STEPS to install SINGULAR:

Actually, we provide instructions for install SINGULAR in two different
    ways:

-------------------------------------------------------
Using the default cygwin setup.exe:

1: Start cygwin setup.exe with the -X option:

This could be done by running setup-x86.exe -X or setup-x86_64.exe -X in
    the location of the file, in a cmd console.

2: Add our repository to the repositories list:

ftp://jim.mathematik.uni-kl.de/repo/cygwin

3: Select it before pressing "next button"

4: Select the "singular" package in the "Math" section.


-------------------------------------------------------
Using the apt-cyg tool:

1: If you have already installed apt-cyg, just jump to the step 4, if not,
     continue with step 2:

2: Install wget (Devel)  and subversion (Web) using the cygwin setup.exe

3: Install apt-cyg by running the following commands:

svn --force export http://apt-cyg.googlecode.com/svn/trunk/ /bin/

chmod +x /bin/apt-cyg

4: Install SINGULAR:

apt-cyg install singular -m ftp://jim.mathematik.uni-kl.de/repo/cygwin

-------------------------------------------------------

Now, you can start SINGULAR by running:

Singular
```

Bibliography

[1] Singular Team. Singular sources git wiki, . URL https://github.com/Singular/Sources/wiki.

[2] Steven Gunn and Sunil Patel. Latex thesis template. URL http://www.sunilpatel.co.uk/thesis-template/.

[3] Debian Wiki. Intro to debian packaging, . URL https://wiki.debian.org/es/IntroDebianPackaging.

[4] Debian Wiki. How to setup a debian repository, . URL https://wiki.debian.org/HowToSetupADebianRepository.

[5] Debian Wiki. Setting up signed apt repository with reprepro, . URL https://wiki.debian.org/SettingUpSignedAptRepositoryWithReprepro.

[6] Apt cyg team. A command-line software installer for cygwin. URL https://code.google.com/p/apt-cyg/.

[7] Fedora Project Wiki. How to create a rpm package, . URL https://fedoraproject.org/wiki/How_to_create_an_RPM_package/es.

[8] linuxsysconfig. Create a yum repository with custom gpg-signed rpm packages. URL http://linuxsysconfig.com/2013/04/create-a-yum-repository-with-custom-gpg-signed-packages/.

[9] Cygwin. Cygwin package contributor's guide, . URL https://cygwin.com/setup.html.

[10] Cygwin. The cygwin setup.exe package server, . URL https://sourceware.org/cygwin-apps/package-server.html.

[11] Cygwin. Cygport readme, . URL https://www.cygwin.com/cygport/README.

[12] Expect. Expect manual. URL http://www.tcl.tk/man/expect5.31/expect.1.html.

[13] Charles Lockhart. Quick'n easy gpg cheatsheet. URL http://irtfweb.ifa.hawaii.edu/~lockhart/gpg/gpg-cs.html.

[14] Singular. Singular official website. URL http://www.singular.uni-kl.de/.

[15] Cygwin Team. Cygwin security changes, . URL https://www.cygwin.com/ml/cygwin-announce/2008-08/msg00001.html.

[16] The Dumb Terminal. Generating random bytes for gpg. URL http://www.thedumbterminal.co.uk/?action=showArticle&articleId=150.