Development Tools

From Matchi Wiki

Contents

Choice of Linux Distro for use on your BYO Desktop/Laptop

Choose one of the many available Linux distributions for use on your BYO machine. Here are some guidelines:

Sabayon Linux

This is the best Linux Distribution one for serious development work on an X86-based machine. Based on Gentoo. This is an Italian build and is a good as the finest pasta you will ever find.

- Installation command: equo install [package]

Mint Linux

This is very nice and is based on Ubuntu but does not integrate well with Komodo IDE when using the KDE desktop - probably because it is Irish.

- Installation command: apt-get install [package]

Also consider...

- Rosa Linux - Features exquisite KDE desktop integration, all the way from Mother Russia
- SuSe Linux - Finest, detailed German attention to detail
- Mandriva Linux - It's French. Meh.
- Gentoo Linux - Excellent and well-documented learning curve about the underlying mysteries of Linux. It is highly recommended that everyone technical attempts to perform an end-to-end comprehensive Gentoo installation. The installation is a time-consuming process since everything is built from source code, but the result is that it will squeeze value out of every CPU-cycle on your particular machine's CPU.

Development Tools

Perl Coding Tools

Great command-line tools to code Perl are vi, nano, emacs. Some of the great GUI development tools available are Komodo from http://activestate.com and Visual Studio from you-know-who. The GUI tools also offer code correction and built-in debugging as you would expect to find in any other fully-fledged IDE.

Command-line Perl Debugger

This allows you to step through code, interrogate and set variables.

Installation

This is installed by default on any environment that uses Perl and is available for Linux, Windows and Mac.

Usage

Type `man perldebug` for a complete list of debugging commands. Here is a list of the (mostly single-letter) debugging commands:

```
List/search source lines:
  l [ln|sub]  List source code
  - or .      List previous/current line
  v [line]    View around line
  f filename  View source in file
  /pattern/ ?patt?   Search forw/backw
  M           Show module versions

Control script execution:
  T           Stack trace
  s [expr]    Single step [in expr]
  n [expr]    Next, steps over subs
  <CR/Enter>  Repeat last n or s
  r           Return from subroutine
  c [ln|sub]  Continue until position

Debugger controls:
  o [...]     Set debugger options
  <[<]|{[{]|>[>] [cmd] Do pre/post-prompt
  ! [N|pat]   Redo a previous command
  H [-num]    Display last num commands
  = [a val]   Define/list an alias
  h [db_cmd]  Get help on command
  h h         Complete help page
  |[|]db_cmd  Send output to pager
  q or ^D     Quit
  L           List break/watch/actions
  t [n] [expr] Toggle trace [max depth] ][trace expr]
  b [ln|event|sub] [cnd] Set breakpoint
  B ln|*      Delete a/all breakpoints
  a [ln] cmd  Do cmd before line
  A ln|*      Delete a/all actions
  w expr      Add a watch expression
  W expr|*    Delete a/all watch exprs
  ![!] syscmd Run cmd in a subprocess
  R           Attempt a restart

Data Examination:
  expr           Execute perl code, also see: s,n,t expr
  x|m expr       Evals expr in list context, dumps the result or lists methods.
  p expr         Print expression (uses script current package).
  S [[!]pat]     List subroutine names [not] matching pattern
  V [Pk [Vars]]  List Variables in Package.  Vars can be ~pattern or !pattern.
  X [Vars]       Same as "V current_package [Vars]".  i class inheritance tree.
  y [n [Vars]]   List lexicals in higher scope <n>.  Vars same as V.
  e Display thread id
  E Display all thread ids.
```

BASH Development Tool

Coding Tools

Great command-line tools to code BASH are vi, nano, emacs. One of the best GUI development tools available for BASH coding is Komodo from http://activestate.com since it offers code correction, although it does not do built-in debugging yet. However, there is a very effective command-line debugger available for BASH, bashdb.

Lint Tool for BASH

It is good practice to run all BASH code through a code linter at least once before deployment to ensure robustness. This linting process highlights any potential coding errors, bad practices and other warnings, such as data type mismatches.

ShellCheck via a browser

Go to http://www.shellcheck.net and paste your BASH scripts in the editing window. It will immediately analyse the code and show any potential shortcomings.

ShellCheck via the command line

- Installation on Sabayon:

```
$ sudo equo install shellcheck
```

- Installation on Red Hat:

There does not seem to be a reliable way to install *shellcheck* on Red Hat yet.

</source>

Usage

```
$ shellcheck mybashscript.sh
...
In mybashscript.sh line 222:
  printf $retval
         ^-- SC2059: Don't use variables in the printf format string. Use printf "..%s.." "$foo".
         ^-- SC2086: Double quote to prevent globbing and word splitting.
etc...
```

Command-line BASH Debugger

This debugger allows you to step through a BASH script and interrogate variables. It is available for UNIX-like systems only, unless you are running CYGWIN on a Windows machine.

Installation

The only Linux distribution that seems to have an installation package for this utility is Gentoo and all its derivatives (Sabayon, Pentoo, BigNose). If you are running one of these Linux's, install it like this:

```
$ sudo echo "app-shells/bashdb" >> /etc/portage/package.unmask/01-developmenttools.package.unmask
$ sudo emerge app-shells/bashdb
... lots of cool commands ...
>>> Installing (1 of 1) app-shells/bashdb-4.3.0.91-r1::gentoo
...
```

If there is no package for your particular Linux, it needs to be manually downloaded, built and installed: Download the installation from here: https://sourceforge.net/projects/bashdb/files/bashdb/4.3-0.91/bashdb-4.3-0.91.tar.bz2/download. The go to your Downloads directory, unpack the file `bashdb-4.3-0.91.tar.bz2` and configure, build and install it:

```
$ cd ~/Downloads
$ tar -xjvf bashdb-4.3-0.91.tar.bz2
$ cd bashdb-4.3-0.91 directory
$ ./configure --prefix=/usr/local
$ make
$ sudo make install
```

This should now have installed the executable `bashdb` in the `/usr/local/bin` directory, and since this directory is on your executable path (you can check, type: `echo $PATH`), you should now be able to run the `bashdb` program from anywhere on this install-environment.

Note
It is good practise to not install **bashdb** on a production or performance-testing environment, as it can be computationally very expensive to run. It has no business being on such an environment as it is a development

tool, come to think of it...

Usage

The commands and behaviour of bashdb are similar to that of the Perl debugger:

```
Available commands:
  action     condition  edit     frame    load     run     source  unalias
  alias      continue   enable   handle   next     search  step    undisplay
  backtrace  debug      eval     help     print    set     step-   untrace
  break      delete     examine  history  pwd      shell   step+   up
  clear      disable    export   info     quit     show    tbreak  watch
  commands   display    file     kill     return   signal  trace   watche
  complete   down       finish   list     reverse  skip    tty
```

Apart from the usual documentation that you can find from the usual `man bashdb` command, you can get details of each command when in the debugger shell by typing `help [command]`, for example:

```
help next
next [COUNT]

Single step a statement, skipping functions.

If COUNT is given, stepping occurs that many times before stopping. Otherwise COUNT is one. COUNT can be an arithmetic exp

Functions and source-ed files are not traced. This is in contrast to "step". See also "skip".

Aliases for next: n
```

Note that many of these commands have aliases, like "next" (or "step over", which does the same as Visual Studio's `F10` key) that has an alias `n`. Similarly, "step" (or "step into", which does the same as Visual Studio's `F11` key) has an alias of `s`.

You can repeat the previous command by hitting the `Return`/`Enter` key.

Starting the Debugging Session

Some scripts need to run under a different user.

You can either *sudo su* - to this user:

```
$ sudo su -
# bashdb ~/matchi/30Development/usr/local/bin/export_data_snapshots.sh -h localhost -u root -p xxxxxxx -d testdatabase -e
```

Or you can run the entire session under *sudo*, which must include the bashdb command:

```
$ sudo bashdb ~/matchi/30Development/usr/local/bin/export_data_snapshots.sh -h localhost -u root -p xxxxxxx -d testdatabas
```

The BASH Debugger always displays the line that it is about to execute, rather than the one it has just executed. Also, for multi-line commands, only the last line of that command is displayed.

List the next 10 lines of code

Use the **l** command:

```
bashdb<127> l
 96:        host_ips="${host_ips} ${host_item}"
 97:      fi
 98:    done
 99:
100:    # Remove duplicate IPs
101:    host_ips=$(echo $host_ips | xargs -n1 | sort -u | xargs )
102:
103:
104:    blacklist_incidents=0
105:    blacklist_hosts=''
```

Continue to a line of code

Use the **c** command and the desired line number to run the script straight through to that line number.

```
bashdb<128> c 101
One-time breakpoint 1 set in file 03blacklist_checker.sh, line 101.
(03blacklist_checker.sh:101):
101:    host_ips=$(echo $host_ips | xargs -n1 | sort -u | xargs )
bashdb<129>
```

View the value of a BASH variable

Use the **print** command (not 'p') and the BASH variable:

```
bashdb<130> print $host_ips
104.24.5.17 104.24.4.17 66.102.1.26 64.233.162.26 74.125.68.27 64.233.162.27 212.227.15.179 212.227.15.163 212.227.15.183
```

Execute the next line of code

Use the **n** command :

```
bashdb<134> n
105:    blacklist_hosts=''
bashdb<135> n
106:    for host_ip in ${host_ips}; do
bashdb<136> n
108:      reverse_dns=$(dig +short -x ${host_ip})
bashdb<137> n
109:      reverse_ip=$(IsIPAddress $host_ip)
bashdb<138> and so on...
```

You can also hit `Enter` to repeat the last command **n** command.

Code Coverage Tool

All BASH scripts should be walked through using this tool as part of the code coverage test.

PHP Development Tools

Xdebug Debugger

Installation

Install it using the PEAR Installer, PECL:

```
$ sudo pecl install xdebug
...
Build process completed successfully
Installing '/usr/lib64/php/modules/xdebug.so'
```

Configuration

Add this line to /etc/php.ini:

```
zend_extension="/usr/lib64/php/modules/xdebug.so"
```

Restart the Apache webservice. On RedHat 6.x

```
$ sudo /etc/init.d/https restart
Stopping httpd:                                            [  OK  ]
Starting httpd:                                            [  OK  ]
```

Testing Installation of Xdebug

Create a file in /var/www/html/info.php:

```
<?php
phpinfo();
?>
```

Browse to http://[server]/info.php and verify that the xDebug module is now sucessfully installed.

Usage

A detailed user guide for Xdebug can be found here: https://xdebug.org/docs

Tools for Windows Users

Scripting using Komodo

Komodo works equally well under Windows as it does under Linux.

Terminal Access with PuTTY

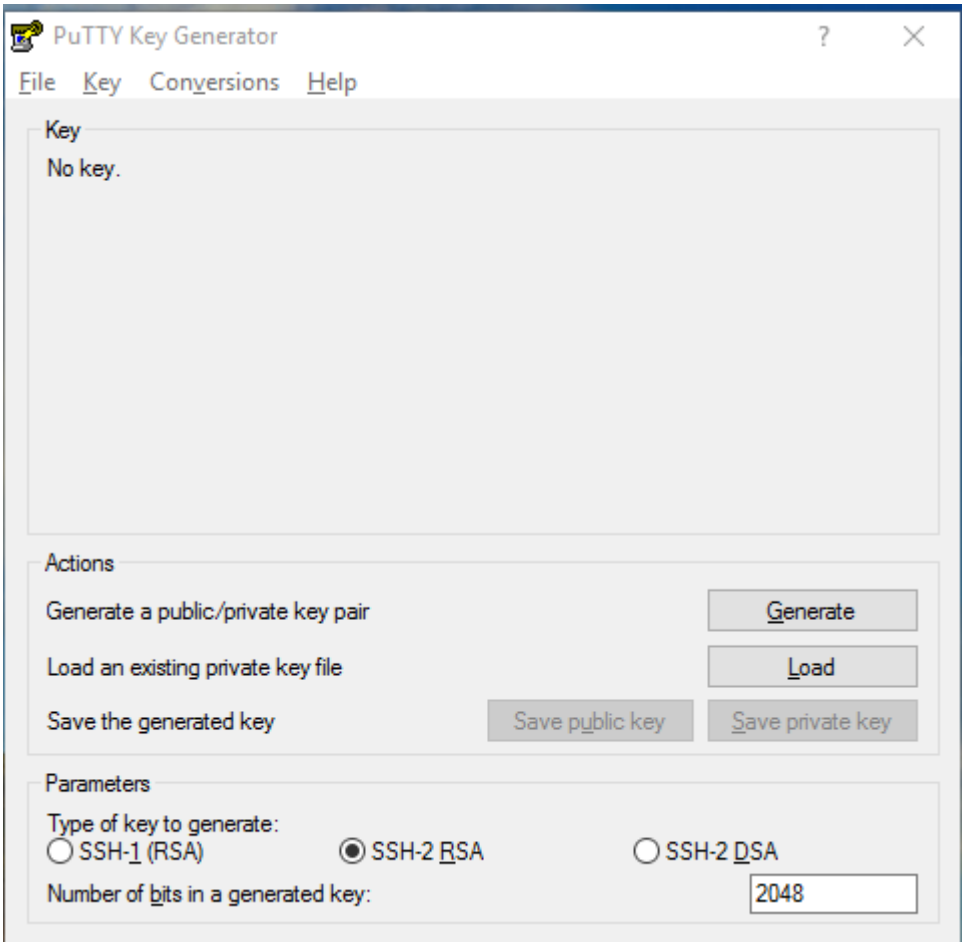
Download the entire PuTTY MSI install-file from https://the.earth.li/~sgtatham/putty/latest/x86/putty-0.67-installer.msi and install it.

Setting up password-less authentication

We only use public-private key pairs to remote access servers and database on them. For this, you should create a 2048-bit RSA key-pair and store the results in your directory `C:\Users\[my_account]\.ssh` as files `id_rsa.pub` (the public key file) and `id_rsa` (the private key file).

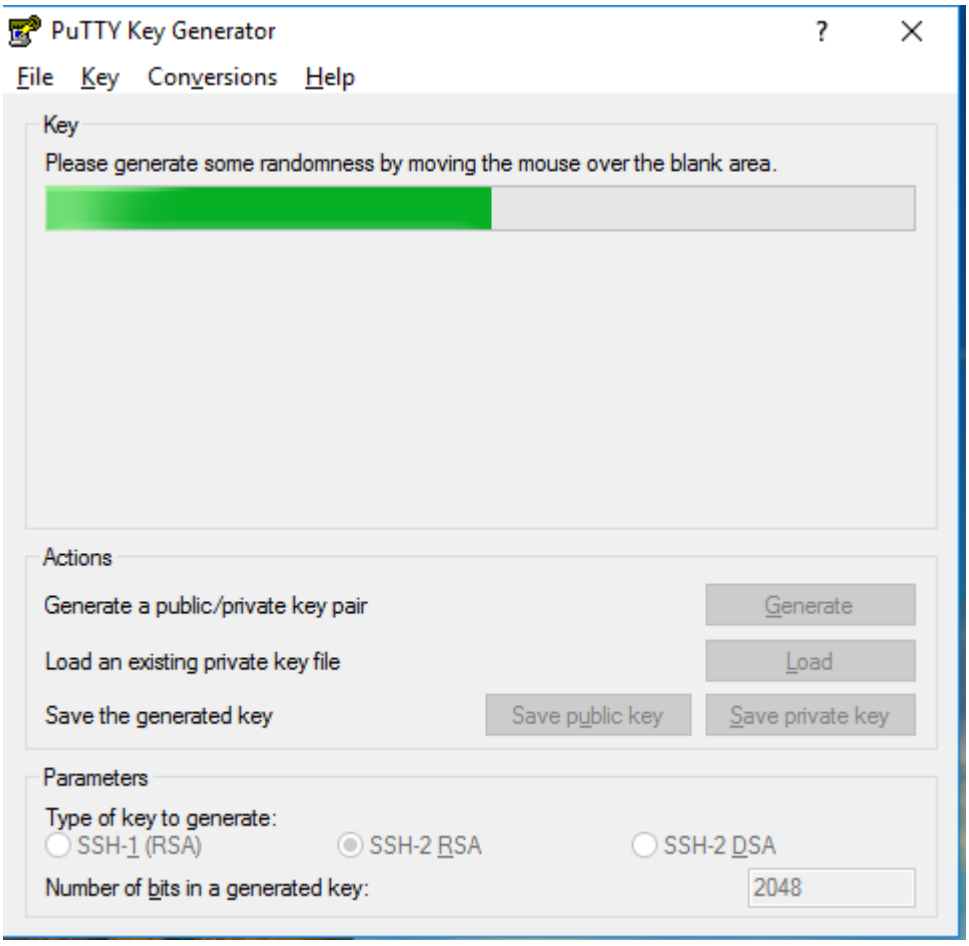
1. Run PuTTYgen.exe

This is the tool that will create your magic key pair:



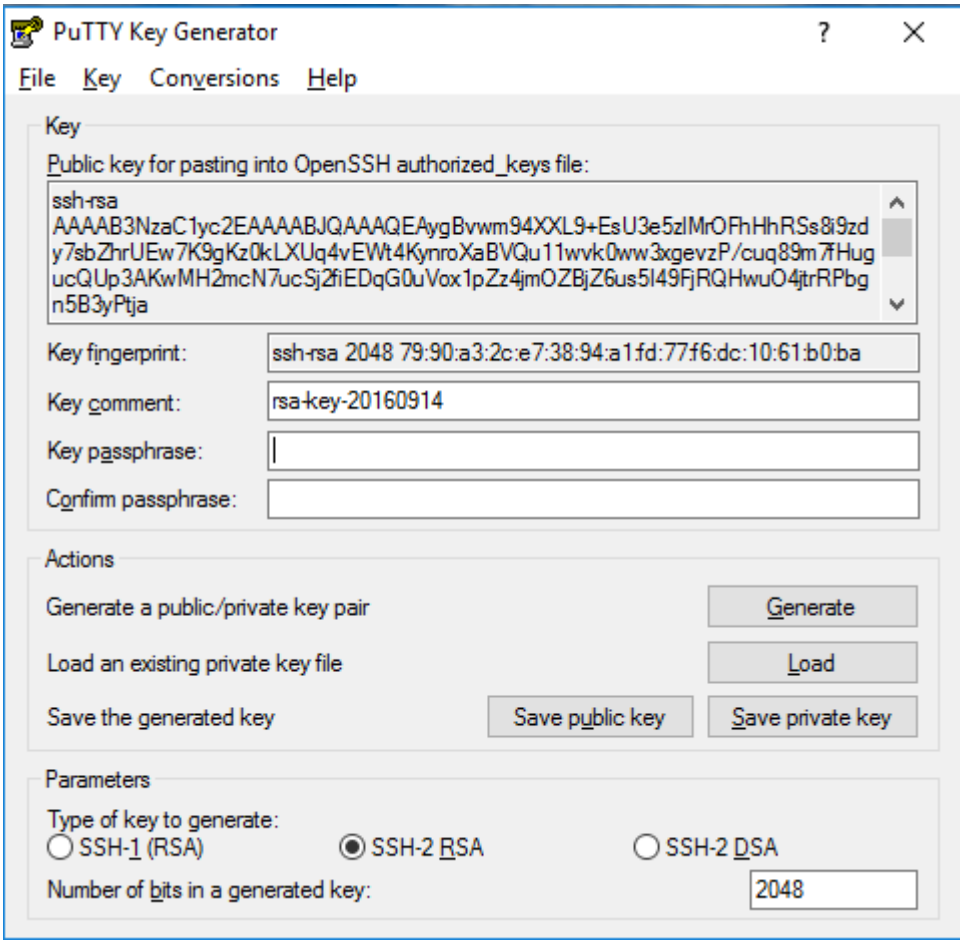
2. Generate your Key Pair

Hit the *Generate* button and wiggle your mouse about for some added entropy to generate your key:



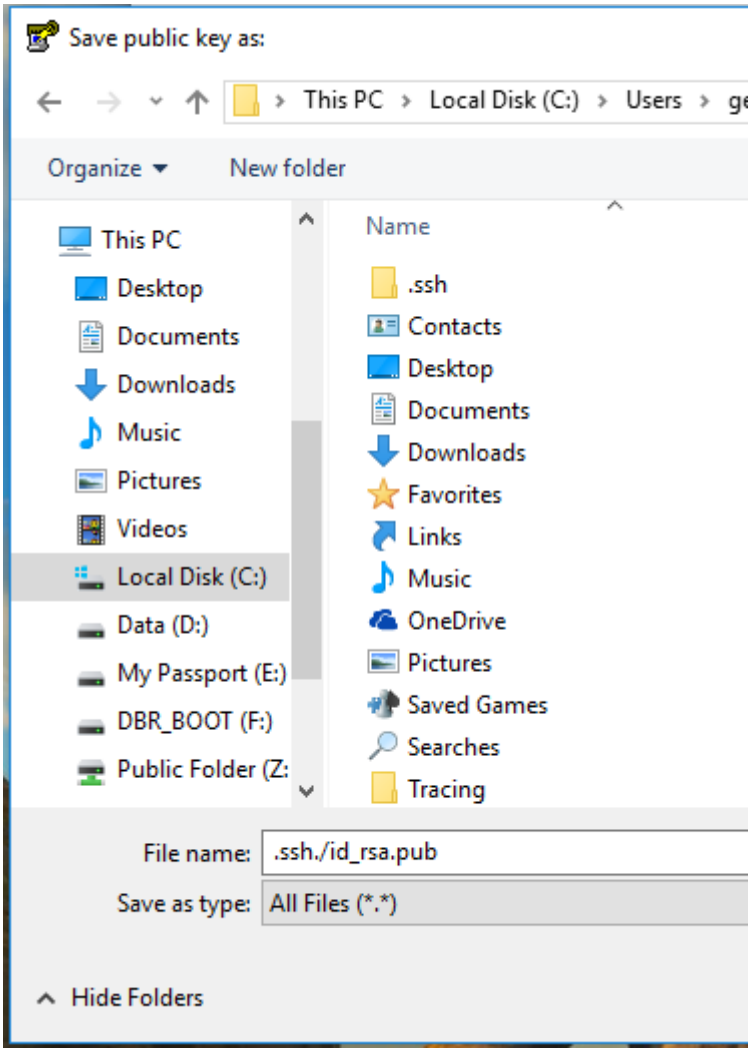
3. Behold, a key is made and lo, it was good. Verily, there was much rejoicing.

This is what the public part of the key pair looks like. You may also optionally add a further passphrase protection to your private key at this stage. The next step is to save it in a standard location.



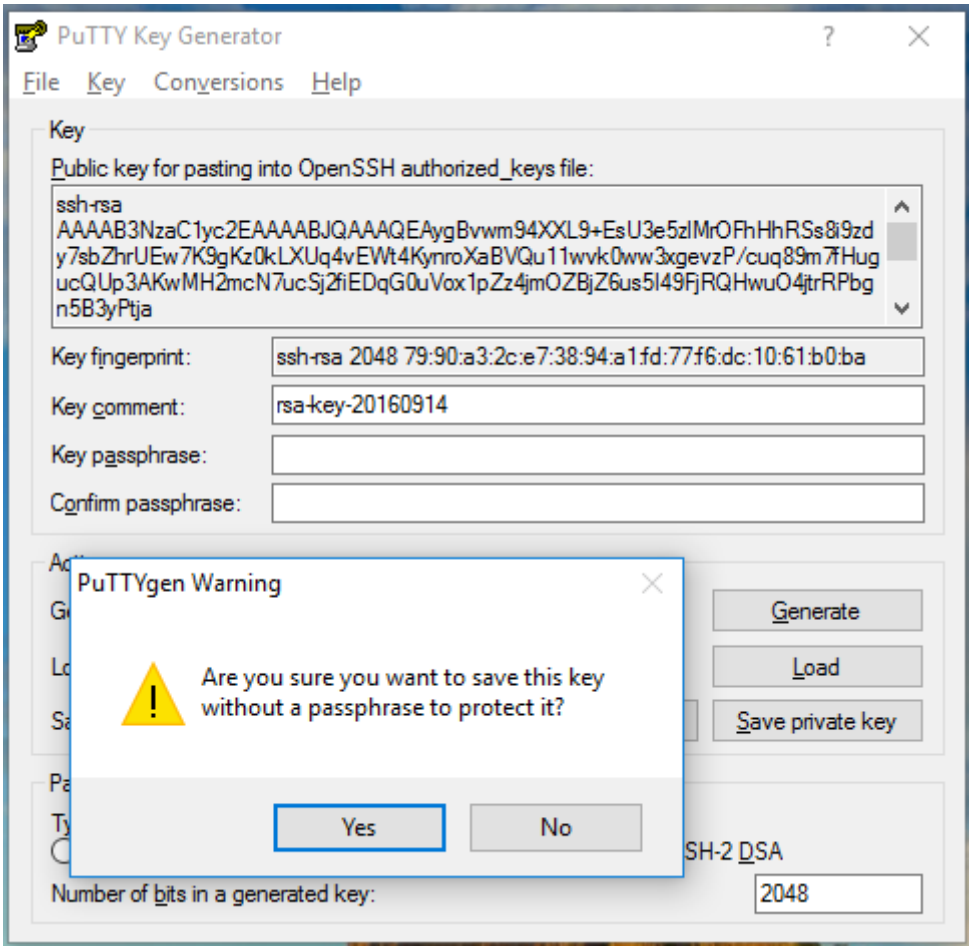
4. Save the public key file

Click the `Save public key` button and save the public key file to `C:\Users\[my_account]\.ssh\id_rsa.pub`

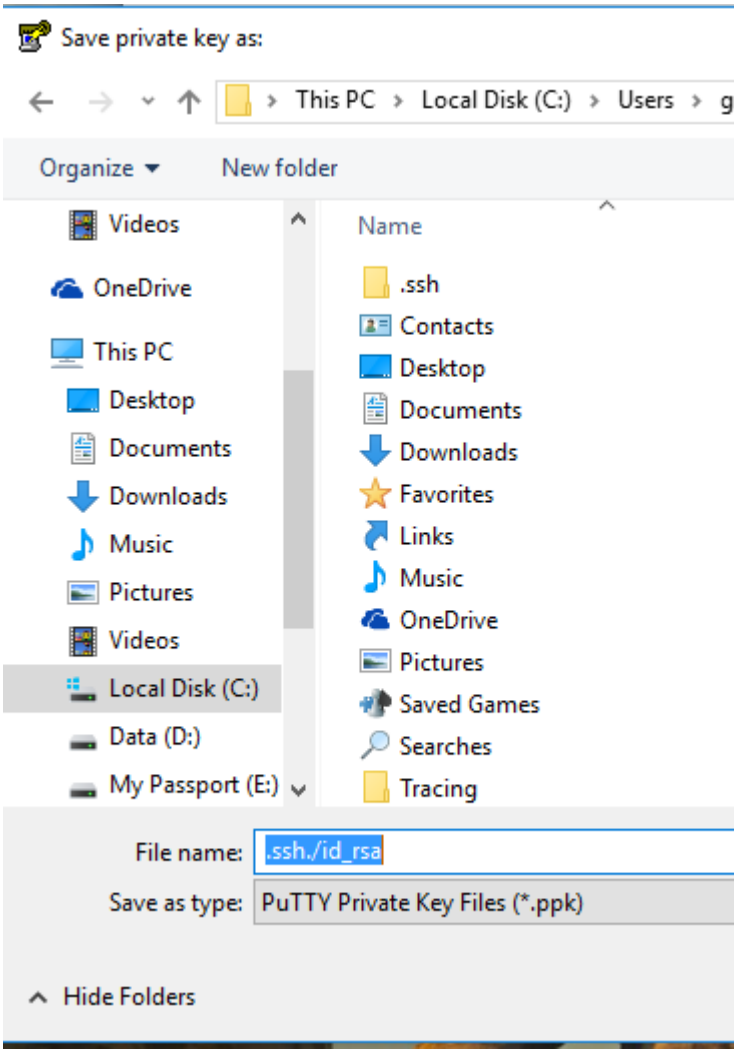


5. Save the private key file

Click the `Save private key` button and save the private key file to `C:\Users\[my_account]\.ssh\id_rsa`. If you did not provide a passphrase when your generated the key pair, you will be reminded about it. Just hit `Yes` and proceed to save the key file.



It is not necessary to save the private key file with the `.ppk` extension.



6. Done!

You have managed to create a key code that is uncrackable. All you need to do to safeguard this key code safe is to keep the private key file secure. The public key file, of course, is the only file that you should expose to the public.

Using your key pair

The content of the *public* key file needs to be added to the remote server in file `/home/madman/.ssh/authorized_keys`. Only someone with existing access to the remote server can do this for you. You can email the content of `C:\Users\[my_account]\.ssh\id_rsa.pub` to sysadmin@matchi.biz.

Once this is in place, you can connect to the remote server, assuming you have configured your hosts file in `C:\Windows\System32\Drivers\etc\hosts` correctly: This applies to when using using PuTTY and connecting with a database development tool such as Navicat to a database server.

SQL Development Tools

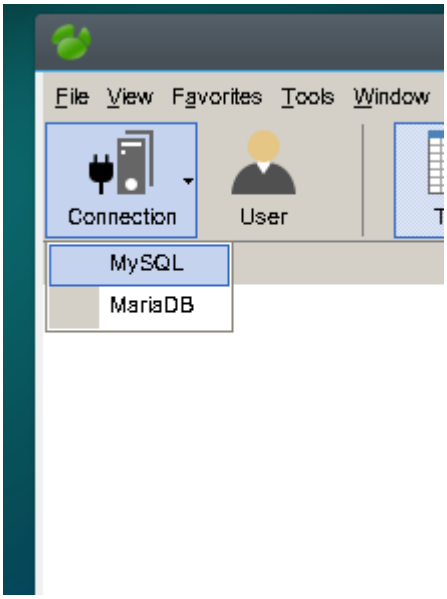
Navicat

Install Navicat from https://www.navicat.com. When you run it, choose to run the Trial version.

Connect to a remote database

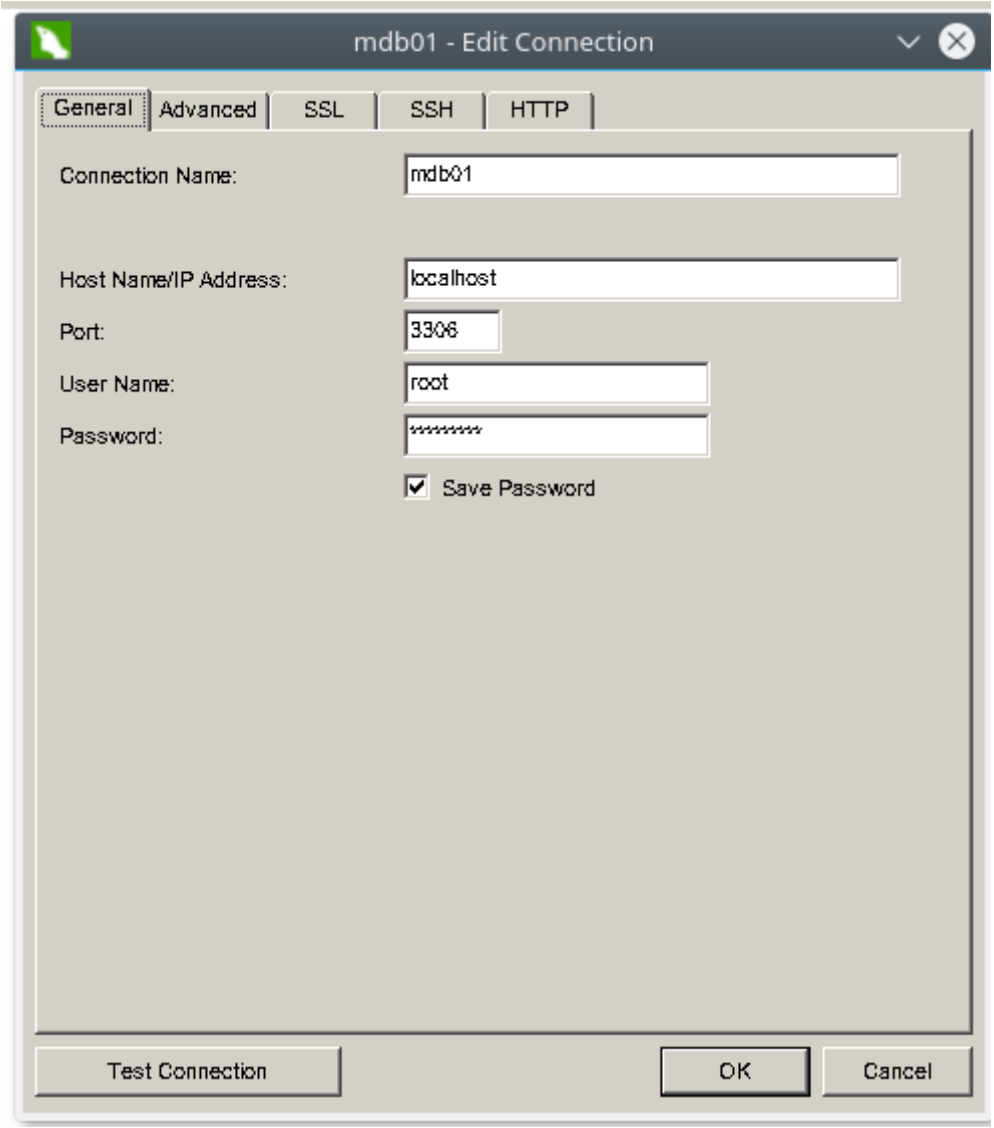
You can only connect to a remote Matchi database over an SSH tunnel. Set up your connection as follows:

1. Create a new connection



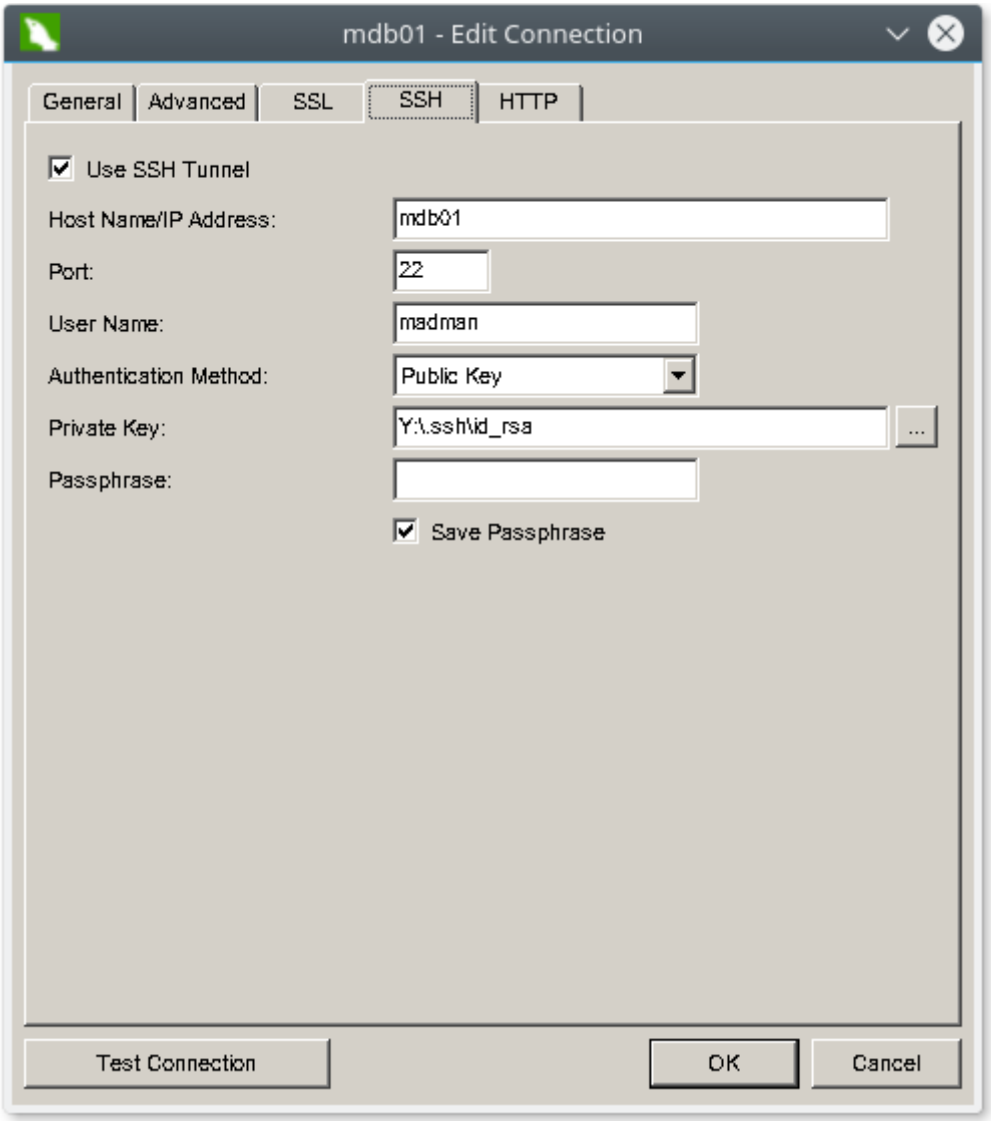
2. Set the basic connection details to the database

These are the basic connection details to the database itself, as it you were already on the database server. The user name is always *root* and the password is currently *Merlin100*.

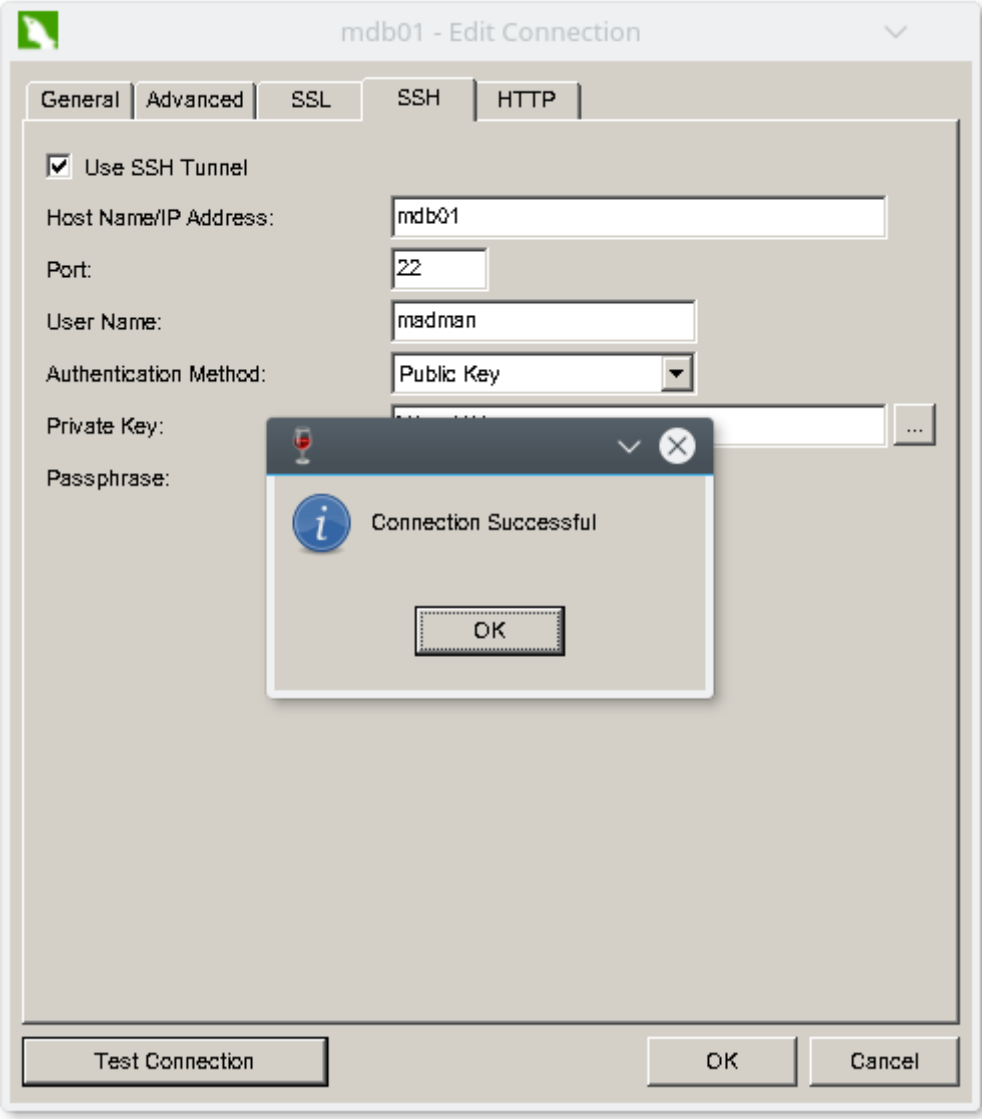


3. Set up the SSH tunnel

Select the SSH tab and provide the full path of your *private* key file. If you created your key-pair with a passphrase, enter that too, and select the *Save Passphrase* option. If you do not need a passphrase, click the *Save Passphrase* option regardless.

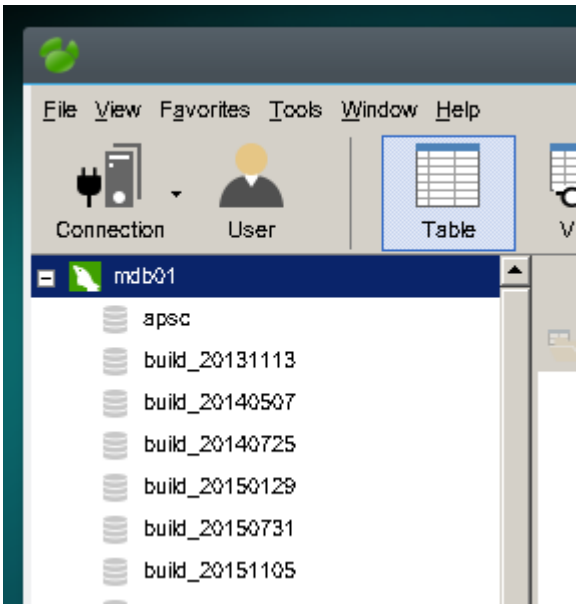


4. Test the connection



5. Open the new connection

Double-click the database connection icon. All databases on that server should be displayed.



Can't connect to the remote database?

Talk to the sysadmin dude/doedie. There may be a problem with the SSH tunneling. The Server should have the following settings set in `/etc/ssh/sshd_config`:

```
...
AllowTcpForwarding yes
GatewayPorts yes
...
```

Connect to a local database

If you have MySQL running on your local machine, then the connection process is similar, except that you do not need to specify the details for the SSH-tunnel. Your username and password may also be different.

MySQL Shell

TODO

MySQL Workbench

TODO

SQLDeveloper

TODO