Quick guide to matool

Version 0.31

Lucas Merckelbach
lucas.merckelbach@hzg.de

February 3, 2012

Contents

1 Introduction

MATOOL is a simple tool aimed at facilitating the preparation, uploading and versioning of glider ma and mi files. Its use allows that if a glider is piloted by several people, they can get an overview of the status of a glider (what files are active, or being set for uploading), comments and issues for that specific glider. In addition the MATOOL provides an interface for external programs to query the contents of the activated mission and ma-files.

1.1 Structure

The structure of the MATOOL is a server-client model. One instance of the server is running on the same machine where the dockserver is running. The server has access to the *to-glider* directories and maintains a repository for

each glider. The client, of which many instances may be running on different computers, contacts the server with requests, such as show the contents of a given ma-file for a given glider, or open such a file for editing. The server will do what is necessary. For example, if a request for editing is received, the server will return the contents of the file to be edited. If the client returns a modified version, the server will put the mafile in the correct directory for uploading, and add the file to the repository. Once the glider picked up the file, the server will notice this and mark the file as active.

As always with communications between server and clients, a firewall may complicate matters in that a firewall may prevent communications between server and client. A common trick is circumvent such problems is the use of tunnels. In this case, the program ssh is used to setup securely tunnels that forward the server ports to the local machine. This is simple and incorporated in MATOOL when running under linux. On windows machines, things are more complicated, and a VPN may provide the solution, or using the windows client putty to set up the tunnels manually.

2 Installation

2.1 Linux

MATOOL is written in python, using only standard modules (no further dependencies). Installation is straight-forward and conform the standard for installing python modules.

1. Unpack the tgz file: tar xvzf matool-xxx.tar.gz
2. Cd into the newly created directory
3. run python setup.py build
4. run as root python setup.py install

This requires root privileges, and installs MATOOL systemwide. MATOOL can be installed for the current user only. Then, the option "–prefix=/some/non-standard/directory/ needs to be appended to step 3, and step 4 can be run as normal user.

The installation process will install both the client and the server, as well some common modules.

2.2 Windows

To be done.

3 Matool, the client

The MATOOL client is called MATOOL. Currently it has only a text interface. It can be run with a command, which is then exectuted and the program exits, or it can be run without a command. When run without a command, a menu and a prompt are provided, allowing the user to formulate one or more commands. At this stage the user interface is fairly basic and lacks features as auto-completion and history.

3.1 matool run without commands

If MATOOL is run without a command, the following menu is displayed:

```
edit [rev <revision_number>] <mafile> <glider> [author]
show <current|queue> <mafile> <glider>
show rev <revision_number> <mafile> <glider>
show log <glider>
queue [glider]
list <glider>
delete <mafile> <glider>
log <glider> [author]
diff <mafile> <glider> <rev_x> [rev_y]
import [[[filename] [filename] ...] <glider>
export <current|queue> <mafile> <glider>

ls
help [command]

m | menu to display this menu
q | quit to quit
```

This list shows all available commands such as edit, show etc. and the arguments that each command can take. The convenction used is that each argument between `< >` is required, and that arguments between `[ ]` are optional. Order in which arguments are supplied matters.

edit [rev <revision_number>] <mafile> <glider> [author]

The edit command is used to modify an mafile. The required arguments are the name of the mafile and the name of the glider. Optionally a revision number can be supplied. In that case, the mafile with the corresponding revision number will be used as basis, instead of the current one.

The author name is optional. If none is provided the login name of the user is used. The choice of editor can be made user dependent, see also ?????

If this command is run, the user is presented with an editor and the contents of the mafile to be modified. If all modifications are finished, the file is saved and the editor closed. The program will ask the user to propagate the changes.

If answered yes, then the changes return to the server, which creates a file in the to-glider directory, and adds the file to its repository (with a new version number). The file is now ready for being picked up by the glider. Note that the user is responsible for instructing the dockserver to upload the mafile to the glider (by running the appropriate xml-script).

show <current—queue> <mafile> <glider>

Shows the requested ma-file for given glider. Depending on the argument current or queue, the contents of the last uploaded file (current), or the contents of the file in the to-glider directory (queue) is returned.

show rev <revision_number> <mafile> <glider>

Shows the requested ma-file for given glider for given revision number. The file is retrieved from the repository.

show log <glider>

Returns the log entries for specified glider. See also the log command.

queue [glider]

Returns the filenames of all files in the to-glider directory. If glider is not specified, all glider directories are searched.

list <glider>

Lists all files currently known in the repository. If a file is marked with a *, then it is not known what the contents is of the corresponding file on the glider. I.e., the file has not been uploaded yet.

delete <mafile> <glider>

Removes <filename> from the 'to-glider' directory of glider <glider>. This effectively cancels the 'edit' action.

log <glider> [author]

Opens an editor with the log file for given glider. This log file can be edited with remarks and comments. Upon saving and quitting the editor, the log file on the server is updated.

diff <mafile> <glider> <rev_x> [rev_y]

Shows the difference between two revisions or one revision and current file.

import [[[filename] [filename] ...] <glider>

Imports given filenames for <glider>. If no filename(s) are given, all filenames ending with ma or mi found in the current directory are imported. Note that this imports any ma files in the current directory on the computer from which the client is running.

export <current—queue> <mafile> <glider>

current: Saves the contents of the latest file that has been uploaded to the glider, i.e. the contents of the file that should be active on the glider, to the local directory.
queue: Saves the contents of the file that is queued for uploading to the glider to the local directory.

ls

This command is a for convenience; it lists all files in the current direction with extension .ma or .mi.

help [command]

The help command is the built-in help system. Without any commands, it displays general help. If a valid command is supplied, the detailed help of that command is displayed.

menu

The menu command (or the short-hand m) gives a list of all commands and their invocation. This command can be requested any time.

quit

The quit command has the expected result. A short-hand is just the q.

3.2 matool run with commands

If MATOOL is run with a command, no menu is displayed, but the command is executed immediately, and upon completion, the command line is returned. Only the commands "edit", "show", "queue", "list", "delete", "log", "diff", "import" and "export" are vaild commands.

3.3 Firewall issues

Normally, the client would connect directly to port to which the MA_EDIT_SERVER listens. When invoked from the same machine, or within the intranet, this should work as intended. If, on the other hand, the client is run outside the intranet, a firewall has to be by-passed. For linux, the most appropriate way to

do this is to use ssh to setup tunnels. On Windows a similar mechanism can be done, using the ssh client putty. For linux, MATOOL can be configured such that the tunnels are setup automatically.

The basic idea behind tunnels is that the ssh-client acts as a program that listens to a port on the local machine (that is, the machine where the ssh-client and thus MATOOL is run). If a program connects to this port, the ssh client forwards this port to a machine on the remote end. This can be the machine which ssh logs in to, but also a machine that is only visible by the remote machine (as it is behind the firewall) and not from the local machine. For this method to work, the user should have a valid user account on the server bastion.hzg.de. Without such an account, no connection can be made from outside the hzg network.

3.4 Configuration file

The MATOOL can be configured using a configuration file .matoolrc, stored in the user's home directory. If it doesn't exist on startup, a configuration file is created using a default setting.

A typical configuration file looks like:

```
[Network]
host = 141.4.0.159
port = 9000

[Ssh]
sshserver = bastion.hzg.de
username = merckelb
gracetime = 300

[lucas]
editor = /usr/bin/gedit
```

The configuration file consists of a section Network (mandatory) and an optional section Ssh. The section Network has two parameters, host, and port. The host has as argument the name or ip address of the machine that is running the MA_EDIT_SERVER. If MATOOL is taking care of the ssh tunneling, then this must be the real address of that machine, even if it is not visible or connectable from the local machine. The port number refers to the port number the MA_EDIT_SERVER is listening too, and must be 9000.

The Ssh section is optional, but usually required if MATOOL is run outside the hzg network. The sshserver argument takes the name or ip address of the ssh server to which the connection from outside the hzg network is made (bastion.hzg.de). If the username on that server is different from the user name on the local machine, then this can be specified by the argument username. The argument gracetime sets the time in seconds how long a connection should be kept alive. (If a connection times out, a new attempt will be made.)

Then any number of sections can be defined where the name of the section corresponds to a username. Only one argument can be specified: the path to the preferred editor. The default editor is gedit, but if any other editor is preferred (emacs, vi, nano, etc.) then this can be specified on a per user basis.

If MATOOL is not supposed to take care of any tunneling, then, sshserver in section Ssh should be "none" or, the whole section Ssh should not exist. In such case, host and port should take the values that match the tunnel definition used.

An example of such a situation is that when putty is used to setup the tunnels. Assuming putty is configured to log into bastion.hzg.de abd forwards the local port 8000 to 141.4.0.159's port 9000, then the configuration file should have no section Ssh and the host and port entries in the section Network should read host = localhost, and port = 8000.

4 Matool, the server

The server is intended to be run on the dockserver. It probably makes most sense to have it run by the user account localuser.

When run, it will listen to port 9000 incomming connections. Currently, there is no configuration file or command line options to change this. There should be only one instance running of the server. If run from the command line, it will first check whether there are files already present in the to-glider directories and ask whether they should be added to the queue. Basically that means that the user tells the server whether or not the server should keep track of the files in the to-glider directories.

The server is typically run from the command line by running the command MA_EDIT_SERVER. A log file ma_edit_server.log is maintained in the home directory. As the current version of the server is still in alpha state, it is best to run the program in the foreground so that any error messages generated can be used for debugging. If, accidentally, a seconds instance of the server is run, it will abort complaining the the address (port) is already in use. If no server is running, the client will complain that there is no server available.