

CRaSH guide

CRaSH

Julien Viet (eXo Platform)

Copyright © 2010

Preface	iii
1. Interacting with the shell	1
1.1. Running CRaSH	1
1.1.1. Web archive deployment	1
1.1.2. Standalone mode	1
1.2. Shell usage	1
1.2.1. Connection	1
1.2.2. Features	2
1.3. Command usage	2
1.3.1. Getting basic help	2
1.3.2. Command line usage	2
1.4. Base commands	4
1.4.1. sleep command	4
1.4.2. man command	4
1.4.3. log command	4
1.4.4. thread command	5
1.5. JCR	6
1.5.1. JCR commands	6
2. Configuring the shell	9
2.1. Change the SSH server key	9
2.2. Change the ports of the telnet or SSH server	9
2.3. Remove the telnet or SSH access	9
2.4. Configure the shell default message	10
3. Extending the shell	11
3.1. Groovy command system	11
3.1.1. Groovy file	11
3.1.2. Groovy execution	11
3.1.3. Shell context	11
3.1.4. You want to contribute a command?	12

Preface

The Common Reusable SHell (CRaSH) deploys in a Java runtime and provides interactions with the JVM. Commands are written in Groovy and can be developed at runtime making the extension of the shell very easy with fast development cycle.

The `bye` command disconnect from the shell.

1.2.1.2. SSH access

SSH connection is done on port 2000 with the password **crash** :

```
juliens-macbook-pro:~ julien$ ssh -p 2000 -l root localhost
root@localhost's password:
CRaSH 1.0.0-beta18-SNAPSHOT (http://crsh.googlecode.com)
Welcome to juliens-macbook-pro.local!
It is Fri Jan 08 21:12:53 CET 2010 now.
%
```

The `bye` command disconnect from the shell.

1.2.1.3. Native access

A third mode is available for standalone CRaSH usage because it uses the JVM native input and output. When you are using it, CRaSh will be available just after the JVM is launched.

1.2.2. Features

- Line edition: the current line can be edited via left and right arrow keys
- History: the key up and key down enable history browsing
- Quoting: simple quotes or double quotes allow to insert blanks in command options and arguments, for instance *"old boy"* or *'old boy'*. One quote style can quote another, like *"ol' boy"*.
- Completion: an advanced completion system is available

1.3. Command usage

1.3.1. Getting basic help

The `help` command will display the list of known commands by the shell.

```
[/]% help
Try one of these commands with the -h or --help switch [addmixin, cd, checkin, checkout, commit, connect, disconnect, ...]
```

1.3.2. Command line usage

The basic CRaSH usage is like any shell, you just type a command with its options and arguments. However it is possible to compose commands and create powerful combinations.

1.3.2.1. Basic command usage

Typing the command followed by options and arguments will do the job

```
% ls -d /
...
```

1.3.2.2. Command help display

Any command help can be displayed by using the `-h` argument:

```
% ls -h
List the content of a node
VAL      : Path of the node content to list
-d (--depth) N : Print depth
```

1.3.2.3. Advanced command usage

A CRaSH command is able to consume and produce a stream of object, allowing complex interactions between commands where they can exchange stream of compatible objects. Most of the time, JCR nodes are the objects exchanged by the commands but any command is free to produce or consume any type.

By default a command that does not support this feature does not consumer or produce anything. Such commands usually inherits from the `org.crsh.command.ClassCommand` class that does not care about it. If you look at this class you will see it extends the `org.crsh.command.BaseCommand`.

More advanced commands inherits from `org.crsh.command.BaseCommand` class that specifies two generic types `<C>` and `<P>`:

- `<C>` is the type of the object that the command consumes
- `<P>` is the type of the object that the command produces

The command composition provides two operators:

- The pipe operator `|` allows to stream a command output stream to a command input stream
- The distribution operator `+` allows to distribute an input stream to several commands and to combine the output stream of several commands into a single stream.

1.3.2.4. Connecting a `<Void,Node>` command to a `<Node,Void>` command through a pipe

Example 1.1. Remove all `nt:unstructured` nodes

```
% select * from nt:unstructured | rm
```

1.3.2.5. Connecting a `<Void,Node>` command to two `<Node,Void>` commands through a pipe

Example 1.2. Update the security of all `nt:unstructured` nodes

```
% select * from nt:unstructured | setperm -i any -a read + setperm -i any -a write
```

1.3.2.6. Connecting two `<Void,Node>` command to a `<Node,Void>` commands through a pipe

Example 1.3. Add the mixin `mix:referenceable` to any node of type `nt:file` or `nt:folder`

```
% select * from nt:file + select * from nt:folder | addmixin mix:referenceable
```

1.3.2.7. Mixed cases

When a command does not consume a stream but is involved in a distribution it will not receive any stream but will be nevertheless invoked.

Likewise when a command does not produce a stream but is involved in a distribution, it will not produce anything but will be nevertheless invoked.

1.4. Base commands**1.4.1. `sleep` command**

```
NAME
    sleep main - Sleep for some time

SYNOPSIS
    sleep [-h | --help] [-h | --help] main time

PARAMETERS
    [-h | --help]
        Provides command usage

    [-h | --help]
        Provides command usage

    time
        Sleep time in seconds
```

1.4.2. `man` command

```
NAME
    man main - format and display the on-line manual pages

SYNOPSIS
    man [-h | --help] [-h | --help] main command

PARAMETERS
    [-h | --help]
        Provides command usage

    [-h | --help]
        Provides command usage

    command
        the command
```

1.4.3. `log` command

```
NAME
    log
```

SYNOPSIS

```
log [-h | --help] [-h | --help] COMMAND [ARGS]
```

PARAMETERS

```
[-h | --help]
Provides command usage
```

```
[-h | --help]
Provides command usage
```

COMMANDS

```
add
create one or several loggers
```

set

The set command sets the level of a logger. One or several logger names can be specified as arguments and the -l option specify the level among the trace, debug, info, warn and error levels. When no level specified, the level is cleared and the level will be inherited from its ancestors.

```
% logset -l trace foo
% logset foo
```

The logger name can be omitted and instead stream of logger can be consumed as it is a <Logger,Void> command. The following set the level warn on all the available loggers:

```
% log ls | log set -l warn
```

send

The send command log one or several loggers with a specified message. For instance the following imp the javax.management.mbeanserver class and send a message on its own logger.

```
## log send -m hello javax.management.mbeanserver
```

Send is a <Logger, Void> command, it can log messages to consumed log objects:

```
% log ls | log send -m hello -l warn
```

info

The loginfo command displays information about one or several loggers.

```
% loginfo javax.management.modelmbean
javax.management.modelmbean<INFO>
```

The loginfo command is a <Logger,Void> command and it can consumed logger produced by the logls command.

```
% logls -f javax.* | loginfo
javax.management.mbeanserver<INFO>
javax.management.modelmbean<INFO>
```

ls

The logls command list all the available loggers., for instance:

```
% logls
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/].[default]
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/eXoGadgetServer].[concat]
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/dashboard].[jsp]
...
```

The -f switch provides filtering with a Java regular expression

```
% logls -f javax.*
javax.management.mbeanserver
javax.management.modelmbean
```

The logls command is a <Void,Logger> command, therefore any logger produced can be consumed.

1.4.4. thread command

NAME

```
thread
```

SYNOPSIS

```
thread [-h | --help] [-h | --help] COMMAND [ARGS]
```



```

PARAMETERS
    [-h | --help]
        Provides command usage

    [-h | --help]
        Provides command usage

COMMANDS
    stop
        stop vm threads

    ls
        List the vm threads

```

1.5. JCR

1.5.1. JCR commands

1.5.1.1. `ws` command

```

NAME
    ws

SYNOPSIS
    ws [-h | --help] [-h | --help] COMMAND [ARGS]

DESCRIPTION
    The ws command provides a set of commands interacting with JCR workspace.

PARAMETERS
    [-h | --help]
        Provides command usage

    [-h | --help]
        Provides command usage

COMMANDS
    login

        This command login to a JCR workspace and establish a session with the repository.
        When you are connected the shell maintain a JCR session and allows you to interact with the session in
        oriented fashion. The repository name must be specified and optionally you can specify a user name and
        have more privileges.

        % connect -c portal portal-system
        Connected to workspace portal-system

        % connect -c portal -u root -p gtn portal-system
        Connected to workspace portal-system

    logout
        This command logout from the currently connected JCR workspace

```

1.5.1.2. `cd` command

```

NAME
    cd main - changes the current node

SYNOPSIS
    cd [-h | --help] [-h | --help] main path

DESCRIPTION
    The cd command changes the current node path. The command used with no argument changes to the root
    node. A relative or absolute path argument can be provided to specify a new current node path.

```


The addnode command creates one or several nodes. The command takes at least one node as argument, but can take more. Each path can be either absolute or relative, relative path creates nodes relative to the workspace. By default the node type is the default repository node type, but the option -t can be used to specify a different node type.

```
[/registry]% addnode foo
Node /foo created
```

```
[/registry]% addnode -t nt:file bar juu
Node /bar /juu created
```

The addnode command is a <Void,Node> command that produces all the nodes that were created.

```
import
Imports a node from an nt:file node located in the workspace:
```

```
[/]% importnode /gadgets.xml /
Node imported
```

```
export
Exports a node as an nt file in the same workspace:
```

```
[/]% node export gadgets /gadgets.xml
The node has been exported
```

1.5.1.5. mixin command

NAME

mixin

SYNOPSIS

mixin [-h | --help] [-h | --help] COMMAND [ARGS]

DESCRIPTION

The mixin command manipulates JCR node mixins. Mixins can be added to or removed from nodes.

PARAMETERS

[-h | --help]
Provides command usage

[-h | --help]
Provides command usage

COMMANDS

add

The add command adds a mixin to one or several nodes, this command is a <Node,Void> command, and can also add a mixin from an incoming node stream, for instance:

```
[/]% select * from mynode | mixin add mix:versionable
```

remove

The remove command removes a mixin from one or several nodes, this command is a <Node,Void> command, and can also remove a mixin from an incoming node stream, for instance:

```
[/]% select * from mynode | mixin remove mix:versionable
```

Chapter 2. Configuring the shell

CRaSH can be configured by tweaking various files of the CRaSH web archive

- *WEB-INF/web.xml*
- *WEB-INF/groovy/login.groovy*

Note that to fully secure the server, you should remove the unauthenticated telnet access as describe below.

2.1. Change the SSH server key

The key can be changed by replacing the file *WEB-INF/sshd/hostkey.pem*. Alternatively you can configure the server to use an external file by using the *ssh.keypath* parameter. Uncomment the XML section and change the path to the key file.

```
<!--  
<context-param>  
  <param-name>ssh.keypath</param-name>  
  <param-value>/path/to/the/key/file</param-value>  
  <description>The path to the key file</description>  
</context-param>  
-->
```

2.2. Change the ports of the telnet or SSH server

The ports of the server are parameterized by the *ssh.port* and *telnet.port* parameters in the *web.xml* file

```
<context-param>  
  <param-name>ssh.port</param-name>  
  <param-value>2000</param-value>  
  <description>The SSH port</description>  
</context-param>
```

```
<context-param>  
  <param-name>telnet.port</param-name>  
  <param-value>5000</param-value>  
  <description>The telnet port</description>  
</context-param>
```

2.3. Remove the telnet or SSH access

To remove the telnet access, remove or comment the following XML from the *web.xml* file

```
<listener>  
  <listener-class>org.crsh.term.spi.telnet.TelnetLifeCycle</listener-class>  
</listener>
```

To remove the SSH access, remove or comment the following XML from the *web.xml* file

```
<listener>  
  <listener-class>org.crsh.term.spi.sshd.SSHLifeCycle</listener-class>  
</listener>
```

2.4. Configure the shell default message

The *login.groovy* file contains two closures that are evaluated each time a message is required

- The `prompt` closure returns the prompt message
- The `welcome` closure returns the welcome message

Those closure can be customized to return different messages.

Chapter 3. Extending the shell

3.1. Groovy command system

The shell command system is based on the [Groovy](#) language and can easily be extended.

3.1.1. Groovy file

Each command has a corresponding Groovy file that contains a command class that will be invoked by the shell. The files are located in the `/WEB-INF/groovy/commands` directory and new files can be added here.

In addition of that there are two special files called *login.groovy* and *logout.groovy* in the `/WEB-INF/groovy` directory that are executed upon login and logout of a user. Those files can be studied to understand better how the shell works.

3.1.2. Groovy execution

When the user types a command in the sell, the command line is parsed by the [args4j](#) framework and injected in the command class. A simple example, the command `connect -c portal -u root -p gtn portal-system` creates the connect command instance and args4j injects the options and arguments on the class:

```
@Description("Connect to a workspace")
class connect extends org.crsh.command.ClassCommand
{
    @Option(name="-u",aliases=["--username"],usage="user name")
    def String userName;

    @Option(name="-p",aliases=["--password"],usage="password")
    def String password;

    @Option(name="-c",aliases=["--container"],usage="portal container name (eXo portal specific)")
    def String containerName;

    @Argument(required=true,index=0,usage="workspace name")
    def String workspaceName;

    public Object execute() throws ScriptException {
        ...
    }
}
```

3.1.3. Shell context

A command is a Groovy object and it can access or use the contextual variables. A few variables are maintained by the shell and should be considered with caution. The shell also provides a few functions that can be used, those functions defined in *login.groovy*

3.1.3.1. Existing context variables

- The `session` variable is managed by the `connect` and `disconnect` commands. Commands should be able to use it for accessing JCR session but not update this variable.
- The `currentPath` variable contains the current path of the shell and it should not be used directly. Instead

one should use the function `getCurrentNode()` and `setCurrentNode(Node node)` to update the underlying path.

3.1.3.2. Existing functions

- The `assertConnected()` checks that the user is connected. It should be used at the beginning of a command that interacts with the session
- The `getCurrentNode()` returns the current node
- The `setCurrentNode(Node node)` updates the current node
- The `findNodeByPath()` functions returns a node based on the provided path. If the provided path is null then the "/" root path is considered. The path can be either relative or absolute. If the path is relative the current node will be used to find the node.
- The `formatValue(Value value)` formats a JCR value into a suitable text value.
- The `formatPropertyValue(Property property)` formats a JCR property value into a suitable text value. If the property is multiple then it will return a comma separated list surrounded by square brackets.

3.1.4. You want to contribute a command?

Drop me an email (see my @ on www.julienviet.com).