batch script language manual

by hank (or yupeng)*

Version 1.0

December 5, 2011

Contents

*If you have any question or advice, you can send email to one of these address: pyu@redhat.com or yupeng0921@gmail.com

1 summary

"batch" is a crash extension.

You can find crash here:

`http://people.redhat.com/anderson/crash_whitepaper`

"batch" is an interpreter. It give crash the ability to run script. It support a C like style script language. Generally you can call a crash command, and store the command output to a variable. Then get a part of the command output, transfer it to another crash command.For example, you know a super_block address is "ffff81087d829400", and you want to know the gendisk address corresponding to the super block.First, you should find the block_device address:

```
crash> struct super_block.s_bdev ffff81087d829400
  s_bdev = 0xffff81087e1fbb80,
```

Then, you find the gendisk from the block_device:

```
crash> struct block_device.bd_disk 0xffff81087e1fbb80
  bd_disk = 0xffff81087f70e400,
```

Use batch, you can write it as a script:

```
cmd = "struct super_block.s_bdev " + "ffff81087d829400";
callret = call(cmd);
getret = get(callret, 0, 11, 18);
cmd = "struct block_device.bd_disk " + getret;
callret = call(cmd); getret = get(callret, 0, 0, -1);
print getret;
```

"call" and "get" are buildin functions. "cmd", "callret", and "getret" are variables, and their type are string. Transfer a string to "call" function, it will exectute the string as a crash command, and return the crash command output as a string. In our example, we set the cmd varaible to a string:

```
cmd = "struct super_block.s_bdev " + "ffff81087d829400";
```

It will set the "cmd" varaible to string:

```
"struct super_block.s_bdev ffff81087d829400"
```

Then run the "call" function:

```
callret = call(cmd);
```

It will run the crash command:

```
struct super_block.s_bdev ffff81087d829400
```

And store the output to variable "callret". So after run that line, the value of "callret" will be:

```
"  s_bdev = 0xffff81087e1fbb80, "
```

"get" function can get a substring of a string. It has 4 parameters, the first one is the input string, the second one specify which lines of the string, as the string may has many lines. The third parameter specify we get the substring from which characters on that line, and the fourth paramter specify how many characters we want to get.

So this line:

```
getret = get(callret, 0, 11, 18);
```

means we get the first line (0 means first line, 1 means second line, and so on), from the 11th character, and get total 18 characters.

The "callret" varaible only has one line, that is line 0. Its 11th character is "0", from that character, we get 18 characters, the result will be:

```
"0xffff81087e1fbb80"
```

It will be store to "getret" variable.

This line will print the string in variable getret:

```
print getret;
```

The "print" function can print a string or number to screen.

2 how to install

2.1 Download the source code

`https://github.com/yupeng820921/batch--a-crash-extention-`

2.2 Build from source code.

Here is an article explain how to build a crash extention:

`http://people.redhat.com/anderson/crash_whitepaper/#SHARED_LIBRARY`

Generall, if you run crash on a x86_64 system, you can run:

```
gcc -nostartfiles -shared -rdynamic -o batch.so batchcmd.c
-fPIC -DX86_64 -D_FILE_OFFSET_BITS=64
```

2.3 Run a crash session

```
crash vmcore vmlinux
```

2.4 Use the "extend" command load batch.so file to crash

copy the "batch.so" file to the current directory of crash session, then use the "extend" command load the "batch.so" file.

```
crash> extend batch.so
./batch.so: shared object loaded
```

2.5 Write a script to test whether it can work

For example, you create a file named "test.cr", and write one line to that file:

```
print "hello world";
```

Then call "batch" command, and pass the script file name as parameter:

```
crash> batch test.cr
hello world
```

3 The syntax of batch script

The batch script is a very simple C-like language, each statement stop with a ';'.

3.1 variable

Identifiers for variables are alphanumeric sequences, and may include the underscore (_) characters. They may not start with a plain digit. The batch script support two kind of variable, the "long" type and the "string" type.

- Create a "long" type variable:

  ```
  a_long_variable = 10;
  ```

- Create a "string" type variable:

  ```
  a_string_variable = "hello world";
  ```

3.2 loop

general syntax:

```
while (exp) stmt
```

example:

```
count = 0;
total = 0;
while (count < 10) {
     total = total + count;
     count = count + 1;
}
```

Note: until now, the loop don't support "break" or "continue".

3.3 if

general syntax

```
if (exp) stmt1 [ else stmt2 ]
```

example:

```
a = 1;
if (a == 0)
print "a is zero";
else
    print "a is not zero";
```

3.4 comment

Both of "#" and "//" are OK.

3.5 command line parameters

Use $1, $2, ... $<NN> obtain the long type parameters, Use @1, @2, ... @<NN> obtain the string type parameters. And $# indicate the total parameters number. For example, if you run a batch script named "test.cr":

```
crash> batch test.cr a_string 0xff
```

The parameters number $# is 3, the first parameter is the file name "test.cr", it is store in @0, and you can get the second parameter "a_string" by @1, get the third parameter 0xff by "#1", You can also get a string "0xff" by @1.

4 build in function

4.1 call

prototype

```
call:string(command_name:string)
```

The "call" function need a string type parameter, then run this string as a crash command. The return value of "call" function is a string, it is the output of the crash command.

example:

```
a = call("sys");
```

After run the "call" function, the output of the crash "sys" will be store in variable "a". It may be something like this:

```
      KERNEL: vmlinux
    DUMPFILE: localhost_vmcore  [PARTIAL DUMP]
        CPUS: 16
        DATE: Tue Nov  1 19:22:45 2011
      UPTIME: 28 days, 00:48:26
LOAD AVERAGE: 1.11, 1.11, 1.12
       TASKS: 886
    NODENAME: localhost
     RELEASE: 2.6.18-194.26.1.el5
     VERSION: #1 SMP Fri Oct 29 14:21:16 EDT 2010
     MACHINE: x86_64  (2266 Mhz)
      MEMORY: 63.1 GB
       PANIC: "SysRq : Trigger a crashdump"
         PID: 24446      COMMAND: "bash"
        TASK: ffff810cf4b33820  [THREAD_INFO: ffff810da301e000]
         CPU: 8
       STATE: TASK_RUNNING (SYSRQ)
```

4.2 get

prototype

```
get:string(command_output:string, lines:long, start:long, len:long)
```

The "get" function can return a substring of an input string. Generally, it is used to get a part of a crash command's output. "command_output" is a string, "lines" specific which line of the string. "start" is specific we get the substring from which characters on the specific line, "len" specific we get how many characters. If "len" is -1, means get until the end of this line.

example:

```
a = call("sys");
b = get(a, 4, 14, 23);
```

Thus, the string "Tue Nov 1 19:22:45 2011" will be stored in variable b.

4.3 lines

prototype

```
lines:long(command_output:string)
```

Input a string, then return how many lines the string contains.

4.4 len

prototype

```
len:long(command_output:string)
```

Return the length of the input string.

4.5 str2long

prototype

```
str2long:long(inputstring:string)
```

Convert a string to its corresponding number, such as convert the string "0xff" to number 255, convert the string "128" to number 128.

4.6 long2decstr

prototype

```
long2decstr:string(inputnumber:long)
```

Convert a number to dec format string, such as convert the number 128 to string "128".

4.7 long2hexstr

prototype

```
long2hexstr:string(inputnumber:long)
```

Convert a number to hex format string, such as convert the number 255 to string "0xff".

4.8 generic math calculate and logic function

+ , - , * , / , >= , <= , != , && , ||