The WebSphere MQ Toolbox

20 Scripts, 1-liners, & Utilities for UNIX & Windows

Certified for IBM. WebSphere. software

Certified for IBM. Power Systems

T.Rob Wyatt
Managing Partner, IoPT Consulting
704-443-TROB (8762)
t.rob@ioptconsulting.com
https://ioptconsulting.com

Speed dating

- We have less than an hour to go over 20 tools so this will necessarily be very high level.
- We'll start with some Windows/UNIX compatibility, then go into the utilities.
- The intent is to provide an introduction in the session & provide sufficient detail to use the deck as reference.
- Look for a download on the MQTC site and/or my site at https://t-rob.net/links

UNIX/Windows equivalents

UNIX	Windows
grep	findstr
# Comment	:: Comment
cp	copy
rm	del
mv	move
&	start
cat	type
xargs	for / in / do
wc	find -c

20 Tools:

#	Win	*NIX	Description
1/2	X	X	Parse A Trigger Message
3/4	X	X	Check for running QMgr
5/6	X	X	Check for admin privs
7/8	X	X	Find queues with depth
9/10	X	X	Enhanced MQSC scripts
11/12	X	X	Dump the cert from a remote QMgr
13/14	X	X	Age messages off of all queues on a QMgr
15/16	X	X	Stop all channels on a QMgr
17	XML		Enhanced FTE XML files
18	N/A	X	Who's in the mqm group, anyway?
19	OpenSSL		Dump the cert from a remote QMgr
20	Perl		Recover stashed password

Parse A Trigger Message - Structure

How do we get this from a single command-line parameter?

```
Trigger Message    struct tagMQTMC2 {
  MQCHAR4    StrucId;      /* Structure identifier */
  MQCHAR4    Version;      /* Structure version number */
  MQCHAR48   QName;        /* Name of triggered queue */
  MQCHAR48   ProcessName;  /* Name of process object */
  MQCHAR64   TriggerData;  /* Trigger data */
  MQCHAR4    ApplType;     /* Application type */
  MQCHAR256  ApplId;       /* Application identifier */
  MQCHAR128  EnvData;      /* Environment data */
  MQCHAR128  UserData;     /* User data */
  MQCHAR48   QMgrName;     /* Queue manager name */
};
```

1) Trigger Message - Windows

Assuming that the TMC2 is in %0, use positional substring extraction:

```
set TMC2StrucID=%0:~0,4%
set TMC2Version=%0:~4,4%
set TMC2Queue=%0:~8,48%
set TMC2Process=%0:~56,48%
set TMC2TrigData=%0:~104,64%
set TMC2ApplType=%0:~168,4%
set TMC2ApplID=%0:~172,256%
set TMC2EnvData=%0:~428,128%
set TMC2UserData=%0:~556,128%
set TMC2QMgr=%0:~684,48%
```

The parms may not be in the order you expect so test for the desired TMC2StrucID and TMC2Version before using the values.

2) Trigger Message – UNIX/Linux

Assuming that the TMC2 is in $1, use positional substring extraction:

```
TMC2StrucId=${1:0:4}
TMC2Version=${1:4:4}
TMC2QName=${1:8:48}
TMC2ProcessName=${1:56:48}
TMC2TriggerData=${1:104:64}
TMC2ApplType=${1:168:4}
TMC2ApplId=${1:172:256}
TMC2EnvData=${1:428:128}
TMC2UserData=${1:556:128}
TMC2QMgrName=${1:684:48}
```

The parms may not be in the order you expect so test for the desired TMC2StrucID and `TMC2`Version before using the values.

Check for running QMgr

- Prior to running a script, we want to make sure the QMgr is running.
- Checking processes isn’t sufficient. What if the QMgr is quiescing? We need to know what MQ thinks the status is.
- Prereq: Script has already validated that dspmq is on PATH. Alternatively, trap errors on the dspmq command.
- If you use valid QMgr status values such as "Running" for a QMgr name, there’s a 10th ring of Hell reserved just for you.

3) Check for running QMgr: Windows

Assumes that the target QMgr name is in "`$QMgr`

```
for /f %%i in ('dspmq ^| findstr "($QMgr)" ^| find /i /c
"Running"') do set isrunning=%%i
if %isrunning% NEQ 1 (
      echo %~nx0 $QMgr not found or not running
      exit /B 2
)
```

- Filters entries from a dspmq on the QMgr name in parenthesis. This avoids getting a hit on TREES searching for TREE, for example.
- Exclude any lines that do not have (RUNNING) as the status.
- Set %isrunning% to the number of lines found. This will be zero or one.
- Test the value of %isrunning% before continuing.
- If you care whether %isrunning%==0 means "not here" or "not found" add a nested IF to test the output of dspmq looking for the QMgr name.

4) Check for running QMgr: UNIX/Linux

Assumes that the target QMgr name is in "`$QMgr`

```
[[ $(dspmq | grep '(Running)' | grep "$QMgr" | wc -l | tr -d "
") != 1 ]] && print "\n${0##*/}: Queue Manager '$QMgr' not
found or not running.\n" && exit 1
```

- **Filters running QMgr entries from a dspmq**
- **Looks for a line with the QMgr name in parenthesis. This avoids searching for a QMgr named TREE and getting a hit on TREES, for example.**
- **Count the number of lines found. This will be zero or one.**
- **If the result of the above does not equal 1, then the QMgr is either not running or not here.**
- **If you care which it is, go back and grep for the QMgr name and use the same technique to display its status if there’s a line or a "not found" error.**

Check for admin privileges

- **Prior to running a script, we want to make sure the user has MQ admin.**
- **Does this by attempting to execute a command that**
  - Requires admin privileges.
  - Is display-only.
  - Does not need to know the QMgr name in advance.
  - Minimal effect on monitoring programs.
  - Returns the same error code across platforms and versions.
- **Intentionally run dspmqtrn with invalid syntax.**
- **AMQ7028 says the command ran OK but you didn't give a QMgr name.**
- **Assumes any other value means the command failed for authorization.**
  This may change over time so validate for each new release!

5) Check for admin privs – Windows

```
:: Test to see if user has WMQ admin privileges
set Response=
set isAdmin=
for /f "tokens=*" %%a in ('dspmqtrn 2^>^&1') do @set
Response=%%a

set isAdmin=!Response:~0,7!

if NOT "%isAdmin%"=="AMQ7028" (
      echo.
      echo %0: This script must be run from an account with
WeSphere MQ adminstrator privileges.
      exit /B 2
      )
```

6) Check for admin privs – UNIX/Linux

- Most scripts must be run as mqm:

```
[[ $(whoami) != "mqm" ]] &&  print "\n${0##*/}: Script must be
executed as the mqm user.\n" && exit 1
```

- Sometimes it’s OK just to be an admin:

```
[[ $(dspmqtrn 2>&1 | grep ‘AMQ7028' | wc -l | tr -d " ") != 1
]] && print "\n${0##*/}: User `whoami` does not appear to have
MQ admin privileges.\n" && exit 1
```

Find queues with depth

- **Need to find queues with depth > 0 (or some arbitrary number)**
- **Possibly need to exclude SYSTEM.***
- **Possibly need to include/exclude Xmit Queues.**
- **This is ridiculously easy with MQSCX from MQGem.**
  - Unsolicited, unpaid endorsement but true.
  - The difference in productivity on any of my scripting engagements would pay for the program several times over.
  - Several people in attendance wish their company had bought it.
- **We'll make do with the built-in utils instead.**

7) Find queues with depth - Win

```
for /f "tokens=2 delims=()" %a in ('echo dis q^(*^)
where^(curdepth gt 0^) ^| runmqsc $QMgr ^| findstr " QUEUE("')
do { :: Something interesting here. }
```

- Assumes the target QMgr name is in $QMgr
- We just want the names, not the actual depths.
- Passes the output to a ‘do’ block
- Adjust accordingly to exclude SYSTEM.* queues:

```
for /f "tokens=2 delims=()" %a in ('echo dis q^(*^)
where^(curdepth gt 0^) ^| runmqsc $QMgr ^| findstr " QUEUE("
^| findstr /V "(SYSTEM." ') do { :: Something interesting
here. }
```

8) Find queues with depth – UNIX/Linux

```
echo 'dis q(*) where(CURDEPTH gt 0)' | runmqsc $QMgr | tr ')'
'\n' | grep "QUEUE(" | tr "(" "\n" | grep -v "\sQUEUE$"
```

- Assumes the target QMgr name is in $QMgr.
- Arguably easier with awk, but not as universally compatible as tr.
- We just want the names, not the actual depths.
- Wrap the output in a ‘for’ or an ‘xargs’
- Adjust accordingly to exclude SYSTEM.* queues:

Enhanced MQSC scripts

- Want to make sure output is logged on each run.
- Would be great to allow substitutions inside the MQSC scripts.
- Can make scripts dynamically adjust to platform, version, etc.

9) Enhanced MQSC scripts - Win

```
for /f "tokens=2-4 delims=.:/ " %a in ("%date%") do set
filename=$~n0.%c-%a-%b.out
(date /t & echo.) > %filename%
runmqsc %QMgr% < commands.mqsc > %filename% 2>&1
```

- Saves the output in [filename].[date].out
- Captures STDERR to file as well

10) Enhanced MQSC scripts – UNIX/Linux

```
$filename=$0.`date "+%y%m%d"`.out
(date;echo)   >$filename
runmqsc $1   >>$filename  2>&1 << EOF
* ------------------------------------------------------------
* DLQ.ksh - Define DLQ on $1
* ------------------------------------------------------------
dis qmgr qmname
DEFINE QLOCAL ('$1.DLQ') REPLACE
ALTER QMGR DLQNAME('$1.DLQ')

* ------------------------------------------------------------
* E N D   O F   S C R I P T
* ------------------------------------------------------------
EOF
```

- Same as Windows but MQSC commands are in the same file and can take substitutions!

Age messages off of all queues

- Need to delete all messages older than 30 days from all queues on the QMgr.
- Skip SYSTEM.* and AMQ.EXPLORER.* queues
- Uses the QLoad program from SupportPac MO03
  http://ibm.co/SupptPacMO03

13) Age msgs from a QMgr - Windows

- Assumes the QMgr name is in %QMgr%

```
FOR /F "delims=^(^); tokens=2" %%y IN ('echo dis q^(^*^)
TYPE^(QLOCAL^) ^| runmqsc %QMgr% ^| findstr "   QUEUE(" ^|
findstr /V "   QUEUE(SYSTEM" ^| findstr /V "
QUEUE(AMQ.EXPLORER"') DO qload -m %QMgr% -I%%y -T30:00:00 -
F%%x_%%y_%%c_%%HH%%M%%%S.txt
```

14) Age msgs from a QMgr – UNIX/Linux

- Assumes the QMgr name is in $QMgr

```
echo "DIS Q(*) TYPE(QLOCAL)" | runmqsc $QMgr | tr ")" "\n" |
grep "^   QUEUE" | awk -F'(' '{print $2}' | grep -v ^SYSTEM. |
grep -v ^AMQ.EXPLORER. | {
    while read QName;do
        Cmd="qload -m $QMgr -I$QName -T30:00:00 \
                 -F${QMgr}_${QName}_%c_%HH%M%S.txt"
        print "\n$Cmd"
        eval $Cmd
    done
    echo
}
```

Stop all channels on a QMgr

- **Prior to shutting QMgr down.**
- **Prior to issuing REFRESH SECURITY TYPE(SSL) command.**
- **Can be executed as a service.**
- **Modify command to start channels.**
  - May not want to issue START channel commands as a service.

15) Stop all Channels - Win

```
for /f "tokens=2 delims=()" %%a in ('echo dis chl^(*^)
where^(chltype ne clntconn^) ^| runmqsc JMSDEMO ^| findstr "
CHANNEL("') DO echo stop chl(%%a) status^(inactive^) | runmqsc
JMSDEMO
```

- **Lists all channels that are not of type CLNTCONN.**
- **Issues STOP command with status of INACTIVE**

16) Stop all Channels – UNIX/Linux

```
echo 'DIS CHL(*) where(CHLTYPE NE CLNTCONN)' | runmqsc $QMgr |
tr ')' "\n" | grep ' CHANNEL(' | tr '(' '\n' | grep -v '
CHANNEL$' | {
    while read ChlName;do
        echo "stop chl($ChlName) status(inactive)" | runmqsc
$QMgr
    done
}
```

- Lists all channels that are not of type CLNTCONN.
- Issues STOP command with status of INACTIVE

Enhanced FTE XML files

- **Metadata values normally spread throughout the XML file.**
- **Some of these are repeated in different places.**
- **Changes rely on the maintainer to find and update all the instances.**
- **Can be difficult to track down.**
- **Solution? Move all the metadata to the top!**

17) Enhanced FTE XML files

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE request [
    <!ENTITY jobname       "multicast">
    <!ENTITY userID        "fteadmin">
    <!ENTITY Cost_Center   "Unknown">
    <!ENTITY sourceAgent   "FTEHUBD1_0001">
    <!ENTITY sourceQM      "FTEAGT01">
    <!ENTITY hostName      "fteagents.example.com">
]>
<request version="4.00"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="FileTransfer.xsd">
 <managedCall>
…
```

- All the metadata fields are listed at the top of the XML file.

17) Enhanced FTE XML files

```
<managedCall>
  <originator>
      <hostName>&hostName;</hostName>
      <userID>&userID;</userID>
    </originator>
    <agent QMgr="&sourceQM;" agent="&sourceAgent;"/>
    <transferSet priority="5">
      <metaDataSet>
        <metaData key="Cost_Center">&Cost_Center;</metaData>
        <metaData key="BusinessUnit">&BusinessUnit;</metaData>
      </metaDataSet>
…
```

- Use the entities in the XML and the values are substituted automatically at execution time.
- Use the same variable in multiple places, it is reliably updated everywhere at once.

Who’s in the mqm group, anyway?

- **We are really good at provisioning access.**
- **We rather stink at decommissioning access.**
- **Good to periodically look at membership of the different groups.**
  - Mostly relevant to UNIX & Linux.
  - Efficacy may vary depending on PAM configuration, NIS+, AD, etc.

18) Get the mqm group membership

- Uses a Perl 1-liner to iterate over all the groups

```
# Get the /etc/passwd entries that have mqm as the primary group
awk -v gid=$(cat /etc/group | grep ^mqm: | cut -d: -f 3) 'BEGIN { FS = ":" } ;
$3==gid { print $1 }' /etc/passwd

# Get the mqm members from the /etc/group entry
perl -e 'while (($name,$members) = (getgrent)[0,3]) {print join("\n", split(" ",
$members)) if $name eq 'mqm';}'

# Bonus points – combine, sort &  undupe
(awk -v gid=$(cat /etc/group | grep ^mqm: | cut -d: -f 3) 'BEGIN { FS = ":" } ;
$3==gid { print $1 }' /etc/passwd && perl -e 'while (($name,$members) =
(getgrent)[0,3]) {print join("\n", split(" ", $members)) if $name eq 'mqm';}' ) |
sort | uniq
```

Dump the cert of a remote QMgr

- **Easy to get the certs over the network from a central location.**
- **Uses the features of the SSL protocol.**
- **Requires preparation of a key file with a self-signed cert that the QMgr does *not* know about.**
- **Even works for your business partners.**
  - When you call to tell them their cert is about to expire, they think you are a genius!
- **Requires OpenSSL to be installed.**
  - IBM installs it with the JRE for just about everything.
  - Standard for nearly all UNIX/Linux distros.

19) Dump remote QMgr cert

```
openssl s_client -connect %CONNAME% -cert Dummy.pem -prexit
2>&1 | openssl x509 -enddate -issuer -subject -noout 2>&1
```

- **CONNAME is host or IP followed by :port, for example 127.0.0.1:1414**
- **Dummy.pem contains a self-signed cert in PEM format**

Recover stashed password

- You know you want to!
- Mostly not needed with new -stashed option on runmqakm/runmqckm.
- The people who you need to worry about already know this.
- Understand the threat that obfuscating the stash file mitigates.

20) Recover stashed password

```
#!/usr/bin/perl -w
use strict;

die "Usage: $0 stashfile\n" if $#ARGV != 0;
open(F,$ARGV[0]) || die "Can't open $ARGV[0]: $!";

my $stash;
my $passwd = '';
read F,$stash,1024;
my @unstash=map { $_^0xf5 } unpack("C*",$stash);
foreach my $c (@unstash) {
     last if $c eq 0;
     $passwd =sprintf "$passwd%c",$c;
}
print "$passwd\n";
```

Questions & Answers