Bash Tutorial

ASL – Fall 2017

Week 2

Comments from previous years

- *“I had to stay up all night to run experiments!”*
- *“I cannot work on this from home..”*
- *“It took me more than 40hrs./week to work on this.*
- Solution is simple

Automate the experiments!

Bash Basics

- Bash is the command line interface used by most linux systems
- On Windows: Use putty to connect to machines in Azure
- Use bash or other scripting languages (e.g. python) to automate your experiments

Running bash commands (I)

#starts a command prompt, returns as soon as the command finished

```
command arg1 arg2
```

#runs the command in the background

```
command arg1 arg2 &
```

#runs the command, writes the standard output to a file called mw.log

```
command arg1 arg2 > mw.log
```

#runs the command, appends the standard output to a file called mw.log

```
command arg1 arg2 >> mw.log
```

#runs the command, writes the standard error to a file called mw-err.log

```
command arg1 arg2 2> mw-err.log
```

Running bash commands (II)

#runs the command, writes the standard output and error to a file called mw.log

```
command arg1 arg2 &> mw.log
```

#runs the command and “pipes” the output into command2

```
command arg1 arg2 | command2
```

Conditionals in bash

```
#set variable
CLIENTS=5
#check if clients smaller 1 or greater 10
if [ $CLIENTS -lt 1 ]; then
      echo “Not enough clients specified!”
elif [ $CLIENTS -gt 10 ]; then
      echo “Too many clients!”
else
      echo “all good!”
fi
```

Loops in bash (I)

```
#for loop variation 1
for i in {1..10}
do
      echo $i
done
#for loop variation 2
for i in `seq 1 10`; do
      echo $i
done
```

Loops in bash (II)

```
#while loop
COUNTER=0
while [ $COUNTER -lt 10 ]
do
      echo $COUNTER
      let COUNTER=COUNTER+1
done
```

Example script

```
#!/bin/bash

cmdpart="./memtier_benchmark --port=11211 --protocol=memcache_text --ratio=0:1 --expiry-range=9999-10000
--key-maximum=1000 --hide-histogram"
server="localhost"
time=20

#Check if there is an argument to the script
if [[ $# > 0 ]]; then
    server="$1"
fi
#Check if there is a 2nd argument to the script
if [[ $# > 1 ]]; then
    time="$2"
fi

#define parameter ranges
clients=(1 2 4)
threads=(2 4 8 16)

for c in "${clients[@]}"; do
    for th in "${threads[@]}"; do
        #add parameters to the command
        cmd="${cmdpart} --server=${server} --test-time=${time} --clients=${c} --threads=${th}"
        #run the command
        $cmd
    done
done

echo "done."
```

Ping example script

```
#!/bin/bash

nodeid=0
server="bach0"
interval=5

#Check if there is an argument to the script
if [[ $# > 0 ]]; then
          nodeid="$1"
fi

#Check if there is a 2nd argument to the script
if [[ $# > 1 ]]; then
          interval="$2"
fi

#cleanup old files
rm ping*.log

for i in {1..8}; do
          if [ $nodeid != $i ]; then
                    cmd="ping -i ${interval} ${server}${i} > ping-${i}.log"
                    eval $cmd &
          fi
done

echo "pings are running."
```

Hint: use something like `pkill -9 ping` **to stop the ping processes**

Example scripts are on the website

Make sure they are executable after you download them:

```
$ chmod +x memtier_bash.sh
$ ./memtier_bash.sh
```

Working on remote machines

- Use SSH and SCP (linux)

login to machine called “mymachine.azure.com” as user “myazureusername”

```
ssh myazureusername@mymachine.azure.com
```

Execute command “ls” on “mymachine.azure.com” as user “myazureusername”

```
ssh myazureusername@mymachine.azure.com “ls”
```

Copy project.jar to home on “mymachine.azure.com”

```
scp project.jar myazureusername@mymachine.azure.com:~/
```

Hint: you can simply clone your git project to the azure machines

Passwordless login on Azure

- Add your public key to the json template
- Generate public key (if it doesn't exist):

```
mkdir ~/.ssh
chmod 700 ~/.ssh
ssh-keygen –t rsa
```

Hint: Don't enter a password for the key

Key can be found in ~/.ssh/id_rsa.pub

Hint: Don't share the secret part!

Running experiments while away

What if my connection fails while running experiments
-> Use `screen` [1]

Bad:

```
user@localmachine: ~$ ssh remotemachine
user@remotemachine:~$ ./runExperiments.sh
```

- Connection lost -> experiment fails

Good:

```
user@localmachine:  ~$ ssh remotemachine
user@remotemachine: ~$ screen –S experiment
user@remotemachine: ~$ ./runExperiments.sh
```

- Connection lost -> experiment continues

```
user@localmachine:  ~$ ssh remotemachine
user@remotemachine: ~$ screen –dR experiment
```

[1] http://www.gnu.org/software/screen/manual/screen.html#Overview

Caution!

Before you start all your experiments and walk away for hours..

- If your experiments fail and you run all machines for a few hours you wasted a **lot of money**!
- Test **all possible configurations** with short runs (no repetitions) to be confident that your long running experiments will succeed.
- While running experiments check your log/result files from time to time to make sure that your experiment hasn’t failed.

Controlling multiple machines

- clusterssh (cssh)
- ssh to machine 1-8
  `cssh myazuername@myazuerdns{1..8}.azure.com`

- Alternatively enter your clusters in config file
  open `~/.clusterssh/clusters` and add
  `clustername machinename1 machinename2`

  `clients myazuerdns{1..3}.azure.com`

  `servers myazuerdns{6..8}.azure.com`