# Just something to share with you

Generating JUNOS config with UNIX shell scripts

Platform: Whatever Junos device you need

OS: As long as it's junos you're fine

---

I ran into a lot of repeating config lines lately. This is a problem to me, because I really don't like typing (don't even think about counting all the typo's I posted here). Since I come from a UNIX background I wrote some scripts to generate the needed code for me. I know you can do a lot with Junos scripting to of course, but I tend to use what I know instead of what I have to learn when there's pressure on a project. I was pleased with the results so maybe I can help some of you to start thinking about scripts for your own needs.

I'll give two examples. The first is for assigning large numbers of ports to a vlan in a EX4200 Virtual chassis. I always divide vlans over "columns" not rows. This way you're dividing the vlan over as many members of your VC as possible. This way losing one single member has the lowest impact. Furthermore it's clear for an end-user. You can document like "al ports 0 and 1 are for wireless access points. All ports 3 to 7 are for voip telephones. 8 to 15 for servers and so on. Of course I know about interface-ranges, but sometime you prefer to configure per port instead of per range.

Now about the scripting: Let's say I want in my 5 member VC interface ge-x/0/0 and ge-x/0/1 assigned to vlan wireless. I enter edit mode and go to the UNIX prompt with run start sh. Cd to /var/tmp. There I use the shell script generate-port-config:

```
#! /bin/sh

if [  $# -ne 5 ]
then
        echo "Usage: $0 [vlan] [start member] [end member] [start port] [end
port]"        exit 1

fi

vlan=$1
first_member=$2
last_member=$3
first_port=$4
last_port=$5

i=$first_member
while [ $i -le $last_member ]
do
        j=$first_port
        while [ $j -le $last_port ]
        do
                echo "set interfaces ge-$i/0/$j unit 0 family ethernet-switching
vlan members $vlan"
                 j=`expr $j + 1`
```

```
        done
        i=`expr $i + 1`
done
```

The output from running this scipt looks like this

```
root@cluster-node0% sh generate-port-config wireless 0 4 0 1

set interfaces ge-0/0/0 unit 0 family ethernet-switching vlan members wireless
set interfaces ge-0/0/1 unit 0 family ethernet-switching vlan members wireless
set interfaces ge-1/0/0 unit 0 family ethernet-switching vlan members wireless
set interfaces ge-1/0/1 unit 0 family ethernet-switching vlan members wireless
set interfaces ge-2/0/0 unit 0 family ethernet-switching vlan members wireless
set interfaces ge-2/0/1 unit 0 family ethernet-switching vlan members wireless
set interfaces ge-3/0/0 unit 0 family ethernet-switching vlan members wireless
set interfaces ge-3/0/1 unit 0 family ethernet-switching vlan members wireless
set interfaces ge-4/0/0 unit 0 family ethernet-switching vlan members wireless
set interfaces ge-4/0/1 unit 0 family ethernet-switching vlan members wireless
root@cluster-node0%
```

Now copy the generated text in to the copy/paste buffer, return to the junos prompt with exit and paste the text. As an alternative you can use load merge set terminal. Or redirect (>) the output to a file and use load merge set /var/tmp/<filename>

Of course when you configure bigger vlans the time saved using scripting increases.

The second example is for more complex code, spanning more than one line of configuration. In a situation like this a “here document” come to help. That’s some text within a shell script. The trick with this type of text is you can refer to variables within it.

The problem I had to solve here was configuring vpn’s to a cisco at the othere side. I configured one vpn to test the connection. After that I needed 18 vpn’s with all their own proxy-id’s to match the cisco’s cryptomap at the other side.  This time I went into the security ipsec hierarchy in the config and once again run start sh to go to the UNIX prompt. After some try an error with awk (some people say it stands for awful) I remembered the syntax. Awk is initiated (with the begin statement) to use the dot as field separator. So $1 here is the first part of the network address, $ 2 the second etc. I repeat this for both subnet.

A little explanation about the structure of this script: It starts with the definition of a function within this function $1, $2 etc refers to the parameters the function is called upon with. The last part in the script calls the function 18 times with all the combination of local and remote subnets:

```
create_vpn_config()

{
remote=$1
local=$2
vpn_name="vpn-`echo $local | awk  'BEGIN        { FS = "."} \
                                        { printf("%s_%s_%s", $1, $2, $3) }'
              `"

vpn_name="${vpn_name}-`echo $remote | awk  'BEGIN       { FS = "."} \
                                        { printf("%s_%s_%s", $1, $2, $3) }'
         `"
```

```
cat <<here
vpn $vpn_name {
    ike {
        gateway ike_gate-to-cisco;
        proxy-identity {
            local $local
            remote $remote
            service any;
        }
        ipsec-policy vpn-pol-to-cisco;
    }
}
here



}

# End of function now call it with the subnets

create_vpn_config 172.29.184.0/24 10.100.30.0/24
create_vpn_config 172.29.185.0/24 10.100.30.0/24
create_vpn_config 172.29.186.0/24 10.100.30.0/24
create_vpn_config 172.29.187.0/24 10.100.30.0/24
create_vpn_config 172.29.171.0/24 10.100.30.0/24
create_vpn_config 172.29.172.0/24 10.100.30.0/24
create_vpn_config 172.29.173.0/24 10.100.30.0/24
create_vpn_config 172.29.174.0/24 10.100.30.0/24
create_vpn_config 172.29.188.0/23 10.100.30.0/24
create_vpn_config 172.29.184.0/24 10.100.20.0/24
create_vpn_config 172.29.185.0/24 10.100.20.0/24
create_vpn_config 172.29.186.0/24 10.100.20.0/24
create_vpn_config 172.29.187.0/24 10.100.20.0/24
create_vpn_config 172.29.171.0/24 10.100.20.0/24
create_vpn_config 172.29.172.0/24 10.100.20.0/24
create_vpn_config 172.29.173.0/24 10.100.20.0/24
create_vpn_config 172.29.174.0/24 10.100.20.0/24
create_vpn_config 172.29.188.0/23 10.100.20.0/24
```

This outputs:

```
vpn vpn-10_100_30-172_29_184 {
    ike {
        gateway ike_gate-to-cisco;
        proxy-identity {
            local 10.100.30.0/24
            remote 172.29.184.0/24
            service any;
        }
        ipsec-policy vpn-pol-to-cisco;
    }
}
vpn vpn-10_100_30-172_29_185 {
    ike {
        gateway ike_gate-to-cisco;
        proxy-identity {
            local 10.100.30.0/24
            remote 172.29.185.0/24
            service any;
        }
```

```
        ipsec-policy vpn-pol-to-cisco;
    }
}
vpn vpn-10_100_30-172_29_186 {

…… lots of code more here
```

The vpn proposal and policies are already there of cource, as is the single phase I. I redirected the output from this script to a file. Then loaded file this after exiting UNIX with “load merge relative /var/tmp/<filename> and all my vpn’s where there.