Scripting and Automation

•••

What is the command line?

- The “scary terminal” (if you’re new to it, it’s not that scary, don’t worry)
- In computing, a shell is a user interface for access to an operating system's services. (Wikipedia)
- Gives you access to more powerful things that are behind the scenes if you spend most of your time in a graphical environment

What can I do with my command line?

- Pretty much anything! There are command line web browsers, chat clients, text editors, and even image manipulation utilities.

- You can customize it with custom commands, prompts, even different shells

- It lets you access and manipulate your system in ways that graphical environments often don’t support

Basic (Linux) shell commands

- File management: touch, mv, chmod, chown, rm, cp, ln, ls
- Software (package) management: apt-get, pacman, yum, emerge, aptitude
- Accessing files: cat, tac, expand, vim, nano, emacs

Shell Scripting

What is shell scripting?

- Writing simple programs that run in your shell
    - Your shell has its own programming language

- Many shell scripts are built by chaining utilities together

Basic Operators

- for, while, conditionals, etc
- Backticks (“`”) run a command and gives you back the output
  - there are more ways of doing this that can be more appropriate for what you’re doing
- Pipes (“|”) take the output of something, and send it to something else.
- Redirection (“>”) takes the output and dumps it into the file you specify
- Queueing (“&&”) lets you run another command after the first if the first ran successfully
- Queueing (“||”) lets you run another command after the first, regardless of whether the first ran successfully
- Some of these have more advanced options, but we’ll stay simple for now

Simple shell script

```
#!/bin/bash

logFile=$HOME/dyndns.log

lastIP=`awk '/SUCCESS/ { save=$NF }END{ print save }' $logFile 2>/dev/null`

currIP=`curl -s --connect-timeout 5 icanhazip.com`

if [ ! "$lastIP" = "$currIP" ] || [ ! -e $logFile ]; then

    curl -s --connect-timeout 5 $URL && Success || Error

else

    echo "`date`: OK: IP address ($currIP), has not changed since last check." >> $logFile

fi
```

from https://github.com/thenaterhood/shellzilla/blob/master/dynamic_ip_update.sh

Let’s break it down

```
#!/bin/bash

logFile=$HOME/dyndns.log

lastIP=`awk '/SUCCESS/ { save=$NF }END{ print save }' $logFile 2>/dev/null`

currIP=`curl -s --connect-timeout 5 icanhazip.com`

if [ ! "$lastIP" = "$currIP" ] || [ ! -e $logFile ]; then

    curl -s --connect-timeout 5 $URL && Success || Error

else

    echo "`date`: OK: IP address ($currIP), has not … > $logFile

fi
```

“Shebang.” This says what to use to run our script.

Set a variable with the path to our log.

Open our log with awk to find the last IP we had. Redirect errors into /dev/null

Get our current IP using curl

If our IP has changed or the log doesn’t exist, use curl to tell DNS and report the success or failure

What might you use this for?

- It's a programming language. You can use it for anything!
- Scripts I use frequently:
    - Update my dynamic IP address in DNS
    - Fix a permissions problem with my printer
    - Adjust my system volume
    - Configure multiple monitors
    - Back up my system
    - Print a folder of PDF files from the Internet
    - ...and many more (github.com/thenaterhood/shellzilla)

Automation

For our purposes...

- ...Automation is having your system run tasks for you automatically

Our Task

- Update our IP address in DNS so our domain name points to the right place
- We’ll use the script we looked at before, placed in /bin/update_my_ip.sh:

```
# Figures out the previous IP address update by looking at the log file, and exits if it is
# the same as the current one, reflecting this in the log
lastIP=`awk '/SUCCESS/ { save=$NF }END{ print save }' $logFile 2>/dev/null`
currIP=`curl -s --connect-timeout 5 icanhazip.com`

# Checks if a logfile exists and if the last and current IP's are the same,
# and pulls the url if either happens to not be the case.
if [ ! "$lastIP" = "$currIP" ] || [ ! -e $logFile ]; then
    curl -s --connect-timeout 5 $URL && Success || Error
else
    echo "`date`: OK: IP address ($currIP), has not changed since last check." >> $logFile
fi
```

- You can write scripts and tools in any language you want

Cron (and relatives)

- Schedule things to run at certain times in a recurring fashion
    - Simple enough, right?

- Use Anacron (a similar utility that comes with some versions of cron) to make sure tasks aren’t missed when your system is off

- Optionally, have it email you the results

Setting up Cron (the easy way)

- Make sure cron is installed and enabled
  - `sudo apt-get install cron`
  - `sudo systemctl enable cron --now`

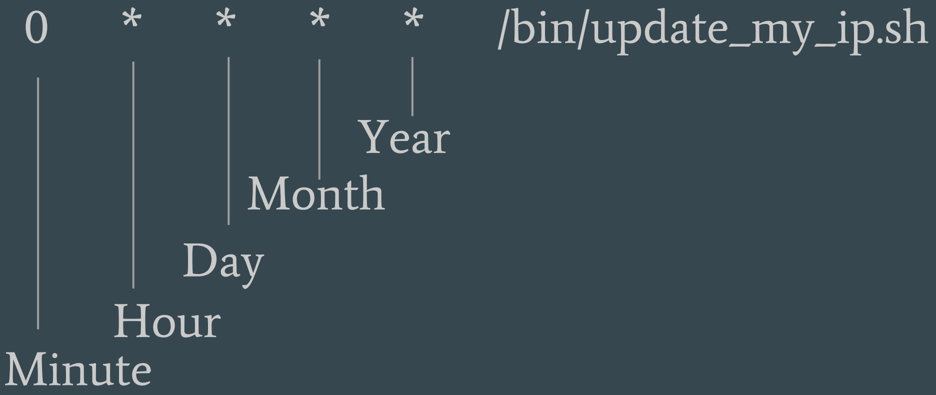
- Set up the task
  - Put the script in /etc/cron.hourly/ to run it hourly
    - `sudo cp /bin/update_my_ip.sh /etc/cron.hourly/`

Setting up Cron (the detailed way)

- Make sure cron is installed and enabled
  - `sudo apt-get install cron`
  - `sudo systemctl enable cron --now`

- Set up the task
  - Run `crontab -e` to edit your crontab
  - Add `0 * * * * /bin/update_my_ip.sh`

Detailed Cron



```
0    *    *    *    *    /bin/update_my_ip.sh
```

- You can break these down further, e.g. for every 5 minutes put */5 for minutes
- There are special values you can use instead, such as @reboot, @midnight, etc
- You can also have tasks run as a specific user, email the output, and other things
    - Good reference: https://wiki.archlinux.org/index.php/Cron

Systemd Timers

- A little more powerful than cron, but harder to set up and no email notifications
  - Requires you to create a service file and a timer file
- Already installed on most modern distros as part of systemd
- Manage your timers like other services on your system

Systemd Services (quick aside)

- How most (likely all) of your system services are handled
- Service or unit files describe services
  - And, they’re pretty simple to write!
- These are what systemd looks at when you tell it to do things with services
  - Your systemctl commands

Create the service file for our timer

```
[Unit]
Description=MyIPUpdateService

[Service]
Type=simple
ExecStart=/bin/update_my_ip.sh

[Install]
After=network.target
```

This section is where you describe metadata about your service. Ours is super simple.

This section is where you describe how to run your service. We're tell it what to run and that it will exit once run

This section is where you describe how your service behaves. Here, we say not to start it until the network comes up.

Systemd Services

- You can go into far more detail about your service if need be
  - https://www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files has a great breakdown of service files

So now, timers!

- We’ll take the service file we created a couple slides back…

```
[Unit]
Description=MyIPUpdateService

...
```

  - …and put it in /etc/systemd/system/update_my_ip.service

- Next, we’ll create the timer for it

Our Timer (similar to the systemd service)

```
[Unit]
Description=Runs our IP update every hour

[Timer]
OnBootSec=10min
OnUnitActiveSec=1h
Unit=update_my_ip.service

[Install]
WantedBy=multi-user.target
```

Timer definition. Wait 10 seconds after we boot and run our service every hour

Save and enable your timer

- We’ll save that file into /etc/systemd/system/update_my_ip.**timer**

- Enable and start our timer
    - sudo systemctl enable update_my_ip.timer --now

At and Batch

- `at` is a utility that will run a task exactly once when you tell it to, and `batch` will run a task when the system load is below a threshold
- If you want to schedule a task more than once, `at` is not the best choice
  - We'll use our same IP script here, but `at` is likely not the best option for handling it
- Not usually pre-installed, but should be in your package manager
  - You will need to start and enable its service (atd)

Create an “at” task

```
at 1am tomorrow
```

```
/bin/update_my_ip.sh
```

(hit enter then ctrl+d - “at”)

“at” will give you back a message which includes a job number. Cancel your job using `` `atrm $jobnumber` ``

Create a “batch” task

```
batch 0.5
```

```
/bin/update_my_ip.sh
```

(hit enter then ctrl+d)

“batch” will give you back a message which includes a job number. Cancel your job using `atrm $jobnumber`.

Other basic scheduling options

- Run the script when you log in to a shell - likely a pretty useless way of handling IP updates:
  - Add `/bin/update_my_ip.sh` to your shell config or ~/.profile

- Run the script when you log into X11 (your graphical desktop) - likely even less useful of a way to handle IP updates:
  - Add `exec /bin/update_my_ip.sh` to your ~/.xinitrc