Lab Exercises

Lab 4

Objective

You will use regular expression to analyse logs.

---

1. **Logs and Regular Expression (Regex)**

- Load up your wiki (DokuWiki) in the Firefox web browser on the *desktop* VM.

  We will be creating a new page.
  - add an internal link to an “as yet non-existent” page called “`Lab 4`”
  - save the existing page with the new link in it
  - now follow that link to find a “This topic does not exist yet” page. Click the appropriate link on the right to create the new page/topic,
  - write a single word on the new page: “`Logs`”, and save.

- Now on the server.

  ```
  cd /var/log
  ll
  ```

  to see a few of the logs that you can read.

  ```
  cd apache2
  ll -t
  ```

  and you should see that the first log file listed there is `access.log` (see the timestamp from a moment ago?). I want you to count how many times you have made a 'GET' call to that website.

  ```
  grep "GET /dokuwiki" access.log | wc -l
  ```

  The word "`grep`" came from line editors many years ago, referring to "*global regular expression print*". What you need to know is that `grep` uses regular expressions to allow for pattern matching in text files, printing out the lines that match the pattern that you supply to the command.

  "`wc`" = word count, and the "`-l`" flag indicates that it is lines should be counted, not words. man `wc` for details.

  See the pipeline character: `|`

  That takes the output of the first command and passes it to the second command (which counts how many lines of output there are). Note that loading a single web page makes multiple GET calls for each of that page's elements (like CSS, images, HTML, ...) How many GET entries are in your log?

How easy would it be for your teacher to check if you have been writing regularly in your wiki? Simply query the `POST` entries.

```
grep "POST /dokuwiki" access.log | wc -l
```

(Blunt reminder: you should be adding content to your wiki! So there should be quite a few POST entries in addition to today's records.)

How about if I just wanted to know how many GET requests have been made after you have set the IP of your desktop VM to 10.0.2.200?

From that subset above, we can filter by the timestamp too... and count those alone that are from this month:

```
grep "GET /dokuwiki" access.log | grep "10.0.2.200" | wc -l
```

(Note the result for your teacher.)

As any IT professional will tell you, at some point you will need to search through logs as part of your troubleshooting, whether hired as a systems administrator or even as a developer debugging your own code. Knowledge of regexes is very, very powerful, and well worth your investment of time. You will find them useful in a much wider range of use cases than you probably suspect at this point.

2. **Syslog**

- `Syslog` provides a well-trodden path for sending log messages to a centralised logging server (why is that good idea?), but the protocol is so widely adopted that most logs are formatted in this manner even if there is no centralised logging server present. Defined here: https://tools.ietf.org/html/rfc5424

But that is a fairly dry read. Let's just dive in and experiment:

```
cd /var/log
less syslog
```

There is plenty of information in syslog, much of which may lack any meaning for you (particularly the kernel messages). So how about we exclude kernel messages from the output:

```
cat syslog | grep -v kernel | less
```

I expect that you will see a bit of background traffic like `systemd`, and other non-kernel services.

(`grep -v` shows any lines that do NOT match the supplied pattern string.)

Time service on fresh Ubuntu 19.04 install

In recent Ubuntu releases `timedatectl` replaces `ntpdate`, which is considered

deprecated. By default `timedatectl` syncs the time once on *boot* and later on uses socket activation to recheck once *network connections become active*.

Ubuntu's default install now uses `system-timesyncd` instead of `ntpd`. `system-timesyncd` connects to the same time servers and works in roughly the same way, but is more lightweight and more integrated with `systemd` and the low level workings of Ubuntu.

`ntpd` daemon connects to a pool of NTP servers that provide it with *constant and accurate* time updates using more sophisticated techniques to constantly and gradually keep the system time on track. Though `system-timesyncd` is fine for most purposes, some applications that are very sensitive to even the slightest perturbations in time may be better served by `ntpd`.

- Install NTP daemon and start NTP service
  - Before installing `ntpd`, you need to turn off `systemd-timedat` to void the system visiting two different time servers in parallel to cause system clock being periodically jumpy.

```
timedatectl status
sudo timedatectl set-ntp no
timedatectl status
```

Look for `system-tiemsyncd.service active: no` in the output. This means `system-tiemsyncd` has been stopped.

  - Now install and start NTP

```
sudo apt install ntp
sudo service ntp start   (or sudo systemctl start ntp.service)
```

  - Confirm NTP is running and syncing with the time servers

```
sudo ntpq -p
```

You should see the first a few lines of the output as:

```
     remote           refid      st t when poll reach   delay   offset
jitter
==============================================================================
 0.ubuntu.pool.n  .POOL.          16 p    -   64    0    0.000    0.000
0.000
 1.ubuntu.pool.n  .POOL.          16 p    -   64    0    0.000    0.000
0.000
 2.ubuntu.pool.n  .POOL.          16 p    -   64    0    0.000    0.000
0.000
 3.ubuntu.pool.n  .POOL.          16 p    -   64    0    0.000    0.000
0.000
 ntp.ubuntu.com   .POOL.          16 p    -   64    0    0.000    0.000
0.000
```

- Determine to which NTP server your VM is synchronised?

  (Note that the IP address may change, but the host name in DNS shouldn't change too often.)

  ```
  grep "pool server" /var/log/syslog | grep ^Dec
  ```

  (Note the result for your teacher.)

  The ^ character in this context says "`Start of the line`".

  (Be aware that in other contexts the ^ character may have different meaning, eg. "`[^ABC]`" means "`NOT letters A, B or C.`")

- Once you have obtained a line of output, how would you break it up into just the relevant pieces?

  I mostly use the commands `awk`, `sed`, and `cut` - have a look at some examples online and read their man pages for more details.

  For example, I want to know about all of those test accounts we created earlier. Let's extract them, and report their `uids`.

  ```
  grep ^test[0-9][0-9]*: /etc/passwd | awk -F: '{print $1, $3}'
  ```

  The regex says look for the lines in `/etc/passwd` that contain usernames that start with "`test`", having at least 1 digit and zero or more further digits. Then take each of those lines, separate them at the ":" boundaries, and then print the first and third tokens (which corresponds to the `username` and `uid`).

  Your turn: Do the same again but instead record usernames and uids for any account that starts with a vowel. Hint: `[aeiou]` I find it useful to sort that output too, so now pipe your output to `sort -u`

  (Note the result for your teacher.)

- `logger` command

  - What if you needed to write to `syslog`? All useful languages have a way to do this. From the Linux command line and shell scripts, you can call upon the '`logger`' command. (In Python, use the `logging` or `syslog` libraries. In Perl, use `Sys::Syslog`)

    On the server VM,

    ```
    logger -t "MyLabel" "Writing to syslog in Lab 4"
    tail /var/log/syslog
    ```

  - Now let's write something to the log based on a condition: add a user to a group only if they have an odd uid.

    ```
    sudo groupadd odds
    ```

    ```
    sudo apt-get install openssh-server
    ```

    And then create a shell script called `ifoddthenlog.sh`

```
cd
pico ifoddthenlog.sh
```

Modify the file as follows:

```
#!/bin/bash

# Adds an account to a special group 'odds' if the passed username has
# a uid that is odd, and logs to syslog directly.
#

THE_GROUP="odds"
usage() {
echo "USAGE: $0 username"
exit 2
}

#MAIN

# check for argument.
if [ $# -ne 1 ] ; then
echo "One argument expected."
usage
fi

# This script must be run as root user to execute the group modification.
if [ $EUID -ne 0 ]; then
echo "Must run $0 as root."
exit 2
fi

# obtain the uid correponding to this username
# (note the `backticks`)
UIDIN=`/bin/grep ^$1: /etc/passwd | awk -F: '{print $3}' `

# check if that uid is odd and if so, add user to the odds group
if [ -n $UIDIN ] && [ $((UIDIN%2)) -eq 1 ]; then
logger -t $0 "$1 has uid $UIDIN which is odd. Adding to group."
sudo /usr/sbin/usermod -a -G $THE_GROUP $1
else
logger -t $0 "$1 was not added to the $THE_GROUP group."
fi

exit 0
```

Run some arguments:

```
sudo ./ifoddthenlog.sh test1
sudo ./ifoddthenlog.sh test2
sudo ./ifoddthenlog.sh test3
sudo ./ifoddthenlog.sh test4
sudo ./ifoddthenlog.sh test5
```

```
tail /var/log/syslog
grep ^odds: /etc/group
```

(Note the result for your teacher.)

Submission and mark

For full marks today, show your teacher

- 1.5 for finding out the count of `GET /dokuwiki` requests from 10.0.2.200;
- 1.5 for finding out NTP servers your server VM has found;
- 1.5 for your command and the sorted username entries starting with `[aeiou]`;
- 1.5 for finding out which accounts are in the odds group;

You should be ready to answer any questions to demonstrate that all work is done by yourself otherwise you may receive 0 mark.

IMPORTANT NOTE: You will need to document all of your lab work in your wiki.