**System Logging and Simple scripting (Hand-Out)**

1. **Start with the course objectives:**

**System logging**  
 Importance of the logs

Location of the logs in Linux  
 Analyzing the file logs

Extracting and Filtering  
 Logs timestamp

Log rotation

journalctl usage and how to make it persistent.

Remote Logging

**Simple scripting**

Script Basics

Scripts and command execution

Variables

Numeric and string comparisms

File operators and conditional statement

Functions and loops

1. **In the Introduction part, Ask question to audience why logging is needed**

Troubleshoot issues

Debug application or system behavior

Security auditing

Compliance

Learning

1. **Types of logging:**

**Kernel logging**: related to errors, warning or information entries that your kernel may write. As an example of a Kernel logging [**dmesg**]

**User logging**: linked to the user space, those log entries are related to processes or services that may run on the host machine.

1. **Location of the logs:**

Where I can find the logs:

* Direct Write:

The location of these logs are being set in the application configuration files.

* Journald:

All services are writing to journald

* rsyslogd:

This is writing into the /var/log.

1. Analyzing the logs:

What do I need for the analysis of a log file?

Type of log file and location

Time stamp or time frame to look for. Otherwise you are looking at information not related to the incident

A log file can be quite extensive with multiple MB not being unusual. Knowing the time stamps or a specific keyword or error to look for is very helpful

A tool to read the text file. You should all be familiar with multiple of them by now.

Depends on the preference.

tail with the option –f is very useful for keeping track of changes in a log file (can do a short demo of this)

1. **Extracting and Filtering:**

You will want to focus on the important information in a log file.

Also if you are sending on log files to other people, they do not value your 1GB log file from the last 4 weeks.

So how do we filter and extract information.

(pipe has already been covered in previous session)

grep searches for string or also regular expressions and is often used in conjunction with | to redirect the STDOUT to another commands STDIN

An example here to extract all information regarding the dhcpclient from /var/log/messages and displaying them in less

1. **Time Stamps:**

(Explain importance of time stamps)

Different time stamps in some log files

most of them will show a known time format of month/day/time.

Dmesg shows hardware and boot information and time stamp is in seconds after reboot

(do a little whiteboard calculation)

How do you know how long linux has been running?

Uptime -sp

Some time stamps might also be in so called EPOCH, which is the time in seconds since 01/01/1970

There are EPOCH converters on the internet so I don’t encourage you todo the maths yourself there

Time zone very important, otherwise you don’t know where to search for entries on log file

1. **syslogd and logrotate:**

logrotate is responsible for rotating the logs after a configurable amount of time, compress them and discard older log files alltogether.

This is done so that one log file does not grow too large over time, which makes it hard to open and navigate

provide support for system logging and kernel message trapping. Support of both internet and unix domain sockets enables this utility package to support both local and remote logging.

This is basically the interface between the application which is trying to send a message and the log file where this message is stored.

syslogd does receive the information from the kernel or application and then processes them in a standardized format to either store them locally or to send them on to a remote logging server.

You will sometimes see syslogd showing up as rsyslog, which stands for rocket fast syslog server, but has the same functionality

Configuration under rsyslog.conf and in the directory rsyslog.d, but it is very rarely touched these daus

logrotate configuration files are under /etc/logorotate.conf

You can see an example entry here, to give you an idea

1. the log files are rotated weekly, kept for 4 weeks
2. new log files are created and old ones get an added suffix of the date

3) log files are not being compressed  
You can configure this to your liking

1. **Become a logging adept (exercise):**

(consider this optional depending on your time management, can be postponed until end of session))

First activity:

Become a logging adept

This is for getting used to the configuration, how to read log files, translate different timestamps, explore the different log files in /var/log, filtering and monitoring

1. **Journalctl:**

journalctl is present in systemd and technically replaced the syslogd/logrotate and /var/log text based log files

You should now all be familiar with systemd and SysVinit.

So what does this mean for?

Instead of using different log files and reading tools like cat,tail,head,more,less there is only journalctl

It is like systemctl for logging

If the command is present, the system uses journalctl

All files are stored under /var/log/journal, but they are all binary, means you can’t read them without journalctl

It is faster and more efficient when reading and writing large log files

Quite controversial, some would say shitstorm, since admins are used to text files and a lot of tools depend on it

That’s why still nowadays in all big distributions, the is a rsyslog compatibility integrated

So you can ignore it if you really want to, but journalctl is the way forward

So let’s have a look how this works…

1. **journalctl “continued”:**

You can do that as an exercise, and ask how to make the logs of the journalctl persistent.

All boot operations that are written into the journald are not persistent.

Sometimes, you might need to debug service startup issues during the boot. In that case you can use the persistent mode:

Change the "/etc/systemd/journald.conf" Storage=persistent

This will allow the boot logs to be saved on a disk like /var/log/journal. However, the directory must already exist and have the proper permissions set. If it does not exist, then journal data is stored in the volatile /run/log/journal/ directory, and the data is erased when the system shuts down.

1. **Examples of journalctl:**

Everything (almost) is done with the journalctl command

(explain examples or yet better do a demo)

1. works like tail –f
2. time based search
3. show only entries for sshd service
4. show only error of severity error to emergency
5. send a message to the new journalctl (if you want to)
6. **Logging Exercise:**

Ask the following question, make them check online, and know how to use the rsyslog to log the commands executed by the users:

[How to log all Bash commands by all users on a server?](https://askubuntu.com/questions/93566/how-to-log-all-bash-commands-by-all-users-on-a-server)

* For BASH shells, edit the system-wide BASH runtime config file:

sudo -e /etc/bashrc

Append to the end of that file:

export PROMPT\_COMMAND='RETRN\_VAL=$?;logger -p local6.debug "$(whoami) [$$]: $(history 1 | sed "s/^[ ]\*[0-9]\+[ ]\*//" ) [$RETRN\_VAL]”’

* Set up logging for "local6" with a new file:

sudo -e /etc/rsyslog.d/bash.conf

And the contents below:

local6.\* /var/log/commands.log

* Restart rsyslog:

sudo service rsyslog restart

* Log out then Log in

1. **Remote logging / CloudWatch:**

When you have hundreds or thousands or hundreds of thousands of servers, you don’t log into them individually for the logs

So you want them in a central location

Can be done via rsyslogd and journalctl, but really nobody does that and everybody uses dedicated monitoring and logging tools

Since we are at AWS, cloudwatch is a good example for a well used tool to centralize logging

(ask audience if they are aware of cloudwatch)

(optional exercise here to push logs from EC2 linux to cloudwatch)

https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/QuickStartEC2Instance.html

Steps:

1) Use or create an amazon linux 2 EC2 instance

2) Open the IAM console at <https://console.aws.amazon.com/iam/>.

3) In the navigation pane, choose **Roles**.

4) Click “create role”

5) Under “**Choose the service that will use this role” choose EC2.**

6)On the next page select [“AmazonEC2RoleforSSM](https://console.aws.amazon.com/iam/home) “, tick the box next to it and select next twice

7) Choose a name and click “create role”

8) In the EC2 console, right click your instance, choose “instance settings” and “add/replace IAM role”

9) Choose the role created previously and attach it

10) Log into the instance via ssh

11) sudo yum update –y

12) sudo yum install -y awslogs

13) (optional) Edit the /etc/awslogs/awslogs.conf to include more log files

14) (optional) Edit /etc/awslogs/awscli.conf to change the region from us-east-1

15) sudo systemctl start awslogsd

16) Check your cloud watch console and choose logs in the configured region (might take a minute or two)

Check /var/log/awslogs.log for any errors.

**Simple Scripting:**

1. **simple scripting:**

Ask audience what are the advantages of scripting?

There are many, but one of the most important are:

1. You can automate tasks with it, meaning you don’t have to type the same command into the command line every time
2. You can make changes easily and creates different versions of scripts
3. You can schedule to run a script at predefined times or based on conditions
4. very flexible, so you can customize your script to your needs

They can range from very simple and a few lines to very complex including variables, loops, branching, functions, comparism and much more.

While we do have a vast amount of possible languages available to us, we will be focusing on bash scripting here for the basics.

python has also become very popular in the last years to use on linux and other operating systems

(examples to follow were adapted from https://linuxconfig.org/bash-scripting-tutorial-for-beginners)

1. **simple command execution:**

#!/bin/bash

cp -r /var/www/html /mnt/backup/

(this is the little script being used in crontab exercise in previous session)

in it’s simplest form, you just put a command into a file.

We can see that the default web server directory has now been copied to /mnt/backup, basically making a backup of of our web server files

Now there is some potential for improvement, since we do not get any feedback, backups simply overwrite, we don’t have a time stamp.

Also we did not include config or log files and have a static path. Lets try to fix this as we go along and introduce new elements

next up is variables

1. **Variables:**

You have already learning about variables in one of the first sessions.

So let’s see how we can use them to our advantage.

In this case we want to make the script also print out the time when the backup was being created along with some more information

here we do add a comment in line 2 to state what the script is actually doing

we then define the folder in a variable, as well as the day, which we get from the date command

we then echo out some information to STDOUT using the variables defined beforehand

we then use the variables again in the actual copy command

Variables are usually in all capital letters as a convention. Technically, it does not matter

Since we want to keep backups easily manageable and also compress them where possible, let’s do a little detour here

#!/bin/bash

# this script is to backup the web server files

WEBFOLDER=/var/www/html

WEBFOLDER=/mnt/backup

DAY=$(date)

echo "Executing script to backup $WEBFOLDER to $BACKUPDIR"

echo "date of execution is $DAY"

cp -r $WEBFOLDER $BACKUPDIR

1. **backup via tar:**

GNU `tar' saves many files together into a single tape or disk archive, and can restore individual files from the archive.

It’s like a zip file in Windows

Especially useful if working with a lot of files.

Has additional options to enable compression.

tar option used:

c for create

z for compression

f to indicate file name

This way we have a single compressed file which now also has a date in a format that we can order it easily in ls to get the latest one

(add 2> /dev/null to tar command to get rid of message about leading `/’)

#!/bin/bash

# this script is to backup the web server files

WEBFOLDER=/var/www/html

BACKUPDIR=/mnt/backup/web/$(date +%Y-%m-%d\_%H%M%S).tar.gz

DAY=$(date)

echo "Executing script to backup $WEBFOLDER to $BACKUPDIR"

echo "date of execution is $DAY"

tar -czf $BACKUPDIR $WEBFOLDER

1. **numeric and string comparisms:**

We can use numeric and string comparism to determine if a string or number is for example equal/not equal or lesser or greater

The example on the left has two numbers in a variable, num\_a and num\_b

Both are a numeric value of 100

We then need to use brackets in order to execute the comparism in bash.

Here we are asking if the numbers are equal

The echo $! does return the exit code and is 0 for true and 1 for false

using this with conditional statements will make much more sense. So lets have a look at them

1. **conditional statements:**

conditional statements or sometimes called branching will execute code based on certain conditions

This is widely used with if/else/fi loops

So lets use an example with a numeric comparism and a conditional statement

We have two number stored in the variables, 100 and 200

Then we start the conditional statement with “if”

So if 100 is less than 200 then execute then continue execution.

Otherwise jump to the the next part of the statement, which in this case is fi and indicates then end of the statement.

The use if an else is optional.

#!/bin/bash

NUM\_A=100

NUM\_B=200

if [ $NUM\_A -lt $NUM\_B ]; then

echo "$NUM\_A is less than $NUM\_B!"

fi

How can we use this now in our backup script?

Lets include some sanity checks and harden the script

1. **file operators:**

We can also test if a certain file does exist or even check if it’s an expected type

The ones most commonly used are

-d

-e

-f

How does this look in application?

Before we are interacting with any type of files, we want to check they are actually present.

This is also useful if we are not sure of the name or directory and have to look in multiple places

In this example we are checking if /etc/resolv.conf is a regular file, which it is indeed

There is a number of file operators:

https://www.tldp.org/LDP/abs/html/fto.html

So how can we apply all of this new knowledge to our script now ?

1. **in application:**

Let us include some sanity checks to harden the script

#!/bin/bash

# this script is to backup the web server files

WEBFOLDER=/var/www/html

BACKUPDIR=/mnt/backup/web/$(date +%Y-%m-%d\_%H%M%S).tar.gz

DAY=$(date)

# check if both directories are present

if [ -d $WEBFOLDER ]; then

        echo "Executing script to backup $WEBFOLDER to $BACKUPDIR"

        echo "date of execution is $DAY"

        tar -czf $BACKUPDIR $WEBFOLDER

else

        echo "$WEBFOLDER is not a directory"

fi

1. **functions:**

Functions allow a programmer to organize and reuse code, hence increasing the efficiency, execution speed as well as readability of the entire script.   
  
It is possible to avoid using functions and write any script without including a single function in it. However, you are likely to end up with a chunky, inefficient and hard to troubleshoot code.

Functions only really become useful on larger scripts, but try to incorporate them from early on, since you never know if your script might stay at 5 lines or will get continuously worked on and expanded.

We can see an example here using the loops we just covered

1. **loops:**

Loops generally help to execute pieces of the code multiple times. There are a number of different loops suited for different purposes.

Some examples are:

for

while

until

An example is for do done

Let’s have a look at them

(go through them and have student explain the outpout)

1. **You should now be familiar with…:**

Where to go from here?

I encourage you to work on a backup script yourself and to try to automate other system administration tasks

All the scripts shown here are also in the comment of the slides available.

The examples here are by no means perfect but should give you a good starting point in scripting and ideas to come back to if needed.

Try to make the script run every day automatically, include multiple folders, give more verbose information about the files and folders and size of them backed up.

You can create a backup logfile which you then automatically uploads to cloud watch logs, which you can then analyze and connect to an SNS topic.

More on AWS services in the coming weeks

more stringent version of the exercise

make interactive script to ask use which directory to backup

backup the folder or file to S3

create a log file to record any actions done

implement error logic to make sure directory is valid

In general, if you’re having issues with your script, you can include ”set –x” at the start of the script after the shebang.

I adds verbosity and basically prints the tracing of the commands inside the script. Set is a bash builtin function like echo.

Scripts are also often used here in AWS for the AWS CLI or the AWS SDK.

(if not done already, the log labs can be used now in slide 8 and 12)

1. **scripting exercise:**

skeleton

#!/bin/bash

aws=`which aws`

Read -p “enter dir to backup” dir

Read -p “S3 bucket name” bucket

Aws s3 cp $dir s3://$bucket >> /home/ec2-user/upload.log 2>$1

https://docs.aws.amazon.com/cli/latest/userguide/cli-services-s3-commands.html

https://docs.aws.amazon.com/cli/latest/userguide/install-cliv1.html#install-tool-pip