# Linux Operations

## Linux Environment Variables and Shell Variables

- In Linux and Unix based systems environment variables are a set of dynamic named values, stored within the system that are used by applications launched in shells or subshells. In simple words, an environment variable is a variable with a name and an associated value.
- Environment variables allow you to customize how the system works and the behavior of the applications on the system. For example, the environment variable can store information about the default text editor or browser, the path to executable files, or the system locale and keyboard layout settings.

In this guide, we will explain to read and set environment and shell variables.

- Variables format:

```
KEY=value
KEY="Some other value"
KEY=value1:value2
```

> Note:

- Names of the variables are case-sensitive. By Naming convention, environment variables should be UPPER CASE.
- There is no space around the equals = symbol.

Variables can be classified into two main categories, `environment variables, and shell variables`.

- Environment variables are variables that are available system-wide and are inherited by all spawned child processes and shells.

- Shell variables are variables that apply only to the current shell instance.
- Each shell such as `bash`, has its own set of internal shell variables.

Below commands that allows us to list and set environment variables in Linux:

- `printenv` – The command prints all or the specified environment variables.
- `export` – The command sets environment variables.
- `env` – The command allows you to run another program in a custom environment without modifying the current one. When used without an argument it will print a list of the current environment variables.

```
[cloudshell-user@ip-10-0-76-152 ~]$ whereis pwd
pwd: /usr/bin/pwd /usr/share/man/man1/pwd.1.gz
# Binaries present in /usr/bin/pwd do not require sudo permissions to execute.

[cloudshell-user@ip-10-0-76-152 ~]$ whereis useradd
useradd: /usr/sbin/useradd /usr/share/man/man8/useradd.8.gz

# Binaries present in /usr/sbin/useradd requires sudo permissions to execute.
```

- `set` – The command sets or unsets shell variables. When used without an argument it will print a list of all variables including environment and shell variables, and shell functions.
- `unset` – The command deletes shell and environment variables.

```
set
a=10
echo $a
unset a
echo $a
```

## List Environment Variables

```
printenv
printenv HOME
```

- Commonly used environment variables:
  - `USER` - The current logged in user.
  - `HOME` - The home directory of the current user.
  - `SHELL` - The path of the current user’s shell, such as bash or zsh.
  - `PATH` - A list of directories to be searched when executing commands. When you run a command the system will search those directories in this order and use the first found executable.
    - `echo $PATH`
      - `/sbin:/bin:/usr/sbin:/usr/bin`
        - These are directory path that are separted by colon `:`
  - `TERM` - The current terminal emulation.

### Setting Environment Variables

```
export MYVAR="MyVar"
export MYPATH="/tmp"
export AWS_REGION="us-east-1"
```

### Persistent Environment Variables

- To make the `Environment variables` persistent, we can those variables in the bash configuration files.
- Per-user shell specific configuration files, that gets executed whenever a user login happens. For example, if you are using Bash, you can declare the variables in the `~/.bashrc`:

```
export PATH="$HOME/bin:$PATH"
export MYUSERVAR="$USER"
export AWS_REGION="us-east-1"
```

- Execute the `~/.bashrc` using `source ~/.bashrc`.
- Or just exit the bash session and login again.
- In most Linux distributions when you start a new session/login terminal shell, environment variables are read from the following files:
- `/etc/environment` - Use this file to set up system-wide environment variables. Variables in this file are set in the following format:
- `/etc/profile` - Variables set in this file are loaded whenever a bash login shell is entered. When declaring environment variables in this file you need to use the export command:

```
export JAVA_HOME="/path/to/java/home"
export PATH=$PATH:$JAVA_HOME/bin
```

- To load the new environment variables into the current shell session use the source command:

```
source ~/.bashrc
```

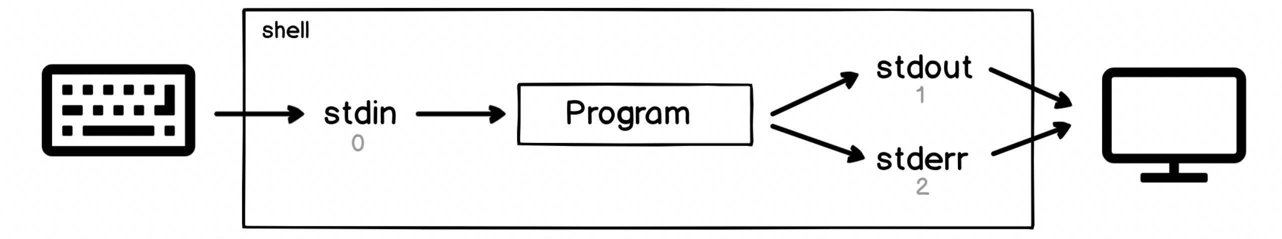
## Linux Text Processing

- Download a Sample Text File.
  - **curl**
    - The curl tool can fetch a given URL from the command-line. This can be useful when we want a web file to be saved on computer
  - `curl https://filesamples.com/samples/document/txt/sample3.txt --output sample3.txt`
  - Use `-s, --silent` to execute the command in Silent or quiet mode.

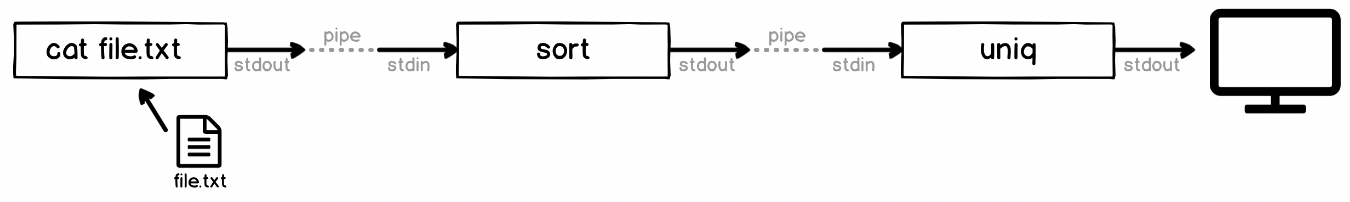
- **wget**
  - `Wget` is the non-interactive downloader which is used to download files from the server/remote path.
- `wget https://filesamples.com/samples/document/txt/sample3.txt -o sample3.txt`

## Piping in Linux

- A `pipe` is a form of **redirection** in Linux that is used to send the output of one command to another command for further processing. The Linux systems allows **stdout** of a command to be connected to **stdin** of another command. You can make it do so by using the pipe character **|**.
- Syntax :
  - `command_1 | command_2 | command_3 | .... | command_N`
- There are three file descriptors that we know i.e `stdin` - 0, `stdout` - 1 and `stderr` - 2



- A Pipe Operator is just a shell convenience which attaches the stdout of one command/process directly to the stdin of the next.



```
# Print number of lines in a file
cat sample3.txt | wc -l
cat sample3.txt | sort | uniq -l
```

## Pattern Matching

- **grep**
  - In Case, you want to search a particular information from a text file.

```
# Print all lines having a specific keyword
cat sample3.txt | grep "ipsum"
# Count all lines having a specific keyword
cat sample3.txt | grep "ipsum" | wc -l
```

## Stream Editor - Replacing a String - `sed`

- **sed** is a stream editor, we can use it to replacing a string with another either in-place or to a new file.
- The substitute flag `/g` (global replacement) specifies the `sed` command to replace all the occurrences of the string in the line.
    - Use the combination of `/1, /2 etc and /g` to replace all the patterns from the nth occurrence of a pattern in a line.
- Create a random file with some text and replace the new content in a new file using `sed`

```
echo "www:google:com" > test.txt

sed 's/:/./g' test.txt > newtest.txt

cat newtest.txt

# To edit the file with inplace changes
sed 's/:/./g' test.txt -i

echo "this is aws" >> test.txt
echo "this is aws" >> test.txt

# Replacing string on a specific line number
sed '3 s/aws/cli/' test.txt -i

# Deleting lines from a particular file, where n is the line number
sed 'nd' test.txt
sed '2d' test.txt -i

# Delete last line
sed '$d' test.txt -i
```

## Awk Fundamentals

- Filtering content from files
- Formatting output
- Displaying non-system users from /etc/passwd
- Using awk control files
- Syntax : `awk options 'selection _criteria {action }' input-file > output-file`
- Built In Variables In Awk
- Field variables - `$1, $2, $3` ... (`$0` is the entire line), that breaks a line of text into individual words or pieces called fields with a specified delimiter.
- `NR`: NR command keeps a current count of the number of input records.
- `NF`: NF command keeps a count of the number of fields within the current input record.

- `FS`: FS command contains the field separator character which is used to divide fields on the input line. The default is `white space`.
- `OFS`: OFS command stores the output field separator, which separates the fields when Awk prints them. The default is a blank space. Whenever print has several parameters separated with commas, it will print the value of OFS in between each parameter.

```
awk ' { print } ' /etc/passwd

#To print only the first column with delimiter as `:`
awk -F":" '{ print $1 }' /etc/passwd

#To print first and last column with delimiter as `:`
awk -F":" '{ print $1,$NF }' /etc/passwd

#number of lines
cat /etc/passwd | wc -l

#To print first lines
awk ' NR < 6 ' /etc/passwd

#To print lines from 5 to 10
awk ' NR==5,NR==10 ' /etc/passwd

#To print only 5th line
awk -F":" 'NR==5{ print $1 }' /etc/passwd

#To print range of lines
awk -F":" 'NR>0 && NR<6{ print $1 }' /etc/passwd

# Display Line Number
awk '{print NR,$0}' /etc/passwd
```

**Searching Files & Directory in Linux**

- There are two different programs that enable us to search the Linux file system.
- **Find**
    - This command searches directories and sub-directories for files matching the criteria.
    - `find /var -name "*.conf"`
- **Locate**
    - This program works by building a daily database of files and their respective locations.When a search is performed, it looks in the DB for the results, the only problem is that the file may have changed in the last 24 hours and so the results returned will be wrong.
    - `locate "*.conf"`
- **Find the Largest Top 10 Files and Directories On a Linux**

```
sudo du -a /var | sort -n -r | head -n 10
sudo du -h /var | sort -rh | head -10
```

- Above command does the following:
  - **du command -h** : Display sizes in human readable format (e.g., 1K, 234M, 2G).
  - **sort command -r** : Reverse the result of comparisons
  - **head command -10 OR -n 10** : It shows the first 10 lines.
- **Find Files Older Than 30 Days**

```
# stat of file
stat /tmp/file1.txt

# Find files older than 30 days
find /opt/backup -type f -mtime +30

# Use '.' as folder name to execute the find command in current directory and
delete them.
find / -type f -mtime +30 -exec stat {} \;
sudo find . -name "*.txt" -type f -mtime +10 -exec stat {} \;
```

- **Find All Csv files and create a zip**

```
# Create empty csv file
touch test{1..5}.csv
# Write some content in above create file
for f in test1.csv test2.csv test3.csv test4.csv test5.csv; do echo -e
"one,two,three,four,five" >> $f; done
# Find all csv files and zip into a single file
find . -name '*.csv' -print | zip csvZipFile.zip -@
```

## Linux CPU and Hardware Information

- **/proc/cpuinfo**
  - The CPU information detected by the OS can be found by displaying the contents of the file `/proc/cpuinfo`
  - `cat /proc/cpuinfo | grep processor | wc -l`
- **lscpu**
  - display information about the CPU architecture
- **lshw**
  - display limited information about the cpu.
- **/proc/loadavg**
  - `cat /proc/loadavg` - The first 3 figures displayed show the load average of the system taken over the last `1, 5, and 15 minutes`.

- **uptime**
    - amount of time the system has been running
- **top**
    - display the top processes currently running on the machine as well as give you load averages and information about the computers memory.

## Security Group Modification using Shell Script

- Find a Security Group by a Name and add ingress/inbound rules to this security group from a csv file.

1. Find a particular Security Group ID when Name is given.
2. Read a csv file and add the inbound rule in the security group.
3. Command to add one security group ingress rule.

- The above use case can be done using `AWS CLI` or using Python(boto3 library) as well. We will see it as below using aws cli.
- Generic command to add a inbound security group will look like.

```
aws ec2 authorize-security-group-ingress --protocol tcp --port $port --cidr ${ip}
--group-id $group_id

aws ec2 authorize-security-group-ingress --protocol tcp --port 22 --cidr
192.168.1.1 --group-id sg-03aa0c3fe2a061293

aws ec2 authorize-security-group-ingress --group-name jenkins-sg --protocol tcp --
port 22 --cidr 203.0.113.0/24
```

- Find security group options in the aws cli command

```
aws ec2 help | grep security
```

```
aws ec2 describe-security-groups --region us-east-1
```

- Replace the Group Name in the below command as per your AWS Account.

```
aws ec2 describe-security-groups --query "SecurityGroups[?GroupName=='no-ingress-
sg']"
```

- Below is the sample Output for above command

```
[
    {
        "Description": "launch-wizard-1 created 2018-12-03T21:11:54.345+05:30",
        "GroupName": "launch-wizard-1",
        "IpPermissions": [
            {
                "IpProtocol": "-1",
                "IpRanges": [
                    {
                        "CidrIp": "49.35.206.245/32"
                    }
                ],
                "Ipv6Ranges": [],
                "PrefixListIds": [],
                "UserIdGroupPairs": []
            },
            {
                "FromPort": 22,
                "IpProtocol": "tcp",
                "IpRanges": [
                    {
                        "CidrIp": "49.35.206.245/32"
                    }
                ],
                "Ipv6Ranges": [],
                "PrefixListIds": [],
                "ToPort": 22,
                "UserIdGroupPairs": []
            }
        ],
        "OwnerId": "0829123458139",
        "GroupId": "sg-0a97252c1661bf218",
        "IpPermissionsEgress": [
            {
                "IpProtocol": "-1",
                "IpRanges": [
                    {
                        "CidrIp": "0.0.0.0/0"
                    }
                ],
                "Ipv6Ranges": [],
                "PrefixListIds": [],
                "UserIdGroupPairs": []
            }
        ],
        "Tags": [
            {
                "Key": "Name",
                "Value": "Test-SG"
            }
        ],
        "VpcId": "vpc-7ad93503"
    }
]
```

- Shell Script individual commands

```
aws ec2 describe-security-groups --query "SecurityGroups[?GroupName=='Demo-ALB-
SG'].[GroupId]"
```

```
[
    [
        "sg-0a97252c1661bf218"
    ]
]
```

```
aws ec2 describe-security-groups --query "SecurityGroups[?GroupName=='Demo-ALB-
SG'].[GroupId]" --output text

sg-0a97252c1661bf218
```

```
aws ec2 describe-security-groups --query "SecurityGroups[?GroupName=='Demo-ALB-
SG']"
```

- Add a single inbound rule using aws cli command:

```
aws ec2 authorize-security-group-ingress --protocol tcp --port 80 --cidr
10.0.0.0/24 --group-id sg-0a97252c1661bf218
```

- `echo` the content of the csv file using shell for loop command

```
for i in $(cat inbound_rules.csv); do echo "This is i: $i"; done
```

- Now we need to get individual value from each line that is separated using `,` and execute the `aws cli` command to add the ip in the security group.
- Get the individual value using `awk` utility.

```
cat ./inbound_rules.csv | awk -F, '{ print $1 }'
cat ./inbound_rules.csv | awk -F, '{ print $2 }'
```

```
#!/bin/bash
for i in $(cat inbound_rules.csv);
```

```
do
    INBOUND_IP=$(echo $i | awk -F, '{ print $1}')
    INBOUND_PORT=$(echo $i | awk -F, '{ print $2}')
    echo "Inbound ip is $INBOUND_IP"
    echo "Inbound port is $INBOUND_PORT"
done
```

```
for i in $(cat inbound_rules.csv); do echo $i | awk -F, '{ print $1 $2 }'; done
```

- In the above shell script we have individual variable value for IP and Port, we can include the command to add security group inbound rule to run for each rule to be added to specific group.

```
#!/bin/bash
SECURITY_GROUP_NAME="Demo-ALB-SG"
SECURITY_GROUP_ID=$(aws ec2 describe-security-groups --query "SecurityGroups[?
GroupName=='$SECURITY_GROUP_NAME'].[GroupId]" --output text)
echo "SECURITY_GROUP_ID value is $SECURITY_GROUP_ID"
for i in $(cat inbound_rules.csv);
do
    INBOUND_IP=$(echo $i | awk -F, '{ print $1}')
    INBOUND_PORT=$(echo $i | awk -F, '{ print $2}')
    echo "Inbound ip is $INBOUND_IP"
    echo "Inbound port is $INBOUND_PORT"
    aws ec2 authorize-security-group-ingress --protocol tcp --port $INBOUND_PORT -
-cidr $INBOUND_IP --group-id $SECURITY_GROUP_ID
done
```

- Execute the above shell script

**Shell Script Enhancements**

- Pass the `SECURITY_GROUP_NAME` value as the command line parameter
- Take ibnound csv file path as Command Line Argument
- Add path for input csv file, also check if the path is correct
- Add exception handling for `SECURITY_GROUP_ID` variable
- Check if CIDR is valid

```
#!/bin/bash
SECURITY_GROUP_NAME=$1
INPUT_FILE_NAME=$2
REGION_NAME=$3
SECURITY_GROUP_ID=$(aws ec2 describe-security-groups --region $REGION_NAME --query
"SecurityGroups[?GroupName=='$SECURITY_GROUP_NAME'].[GroupId]" --output text)
echo "SECURITY_GROUP_ID value is $SECURITY_GROUP_ID --region $REGION_NAME"
if [ $SECURITY_GROUP_ID != "" ]; then
    if [ -f $INPUT_FILE_NAME ]; then
```

```
        for i in $(cat $INPUT_FILE_NAME);
        do
            INBOUND_IP=$(echo $i | awk -F, '{ print $1}')
            INBOUND_PORT=$(echo $i | awk -F, '{ print $2}')
            echo "Inbound ip is $INBOUND_IP"
            echo "Inbound port is $INBOUND_PORT"
            aws ec2 authorize-security-group-ingress --region $REGION_NAME --
protocol tcp --port $INBOUND_PORT --cidr $INBOUND_IP --group-id $SECURITY_GROUP_ID
        done
    else
        echo "File $INPUT_FILE_NAME does not exists"
    fi
else
    echo "$SECURITY_GROUP_ID is blank, cannot execute"
fi

# Execute shell script with below cli parameters
# bash add_inbound.sh defaut inbound_rules.csv us-west-2
# bash add_inbound.sh ElasticMapReduce-slave inbound_rules.csv us-east-2
```