

set -eux o pipefail

- `set -e` (instructs bash to immediately exit if any command [1] has a non-zero exit status. If one line in a script fails, but the last line succeeds, the whole script has a successful exit code. That makes it very easy to miss the error.)
- `set -u` unbound variable" to stderr.
- `set -o pipefail` This setting prevents errors in a pipeline from being masked. If any command in a pipeline fails, that return code will be used as the return code of the whole pipeline.\
- `set -x` (Clear Debugging, Enables a mode of the shell where all executed commands are printed to the terminal. In your case it's clearly used for debugging, which is a typical use case for `set -x` : printing every command as it is executed may help you to visualize the control flow of the script if it is not functioning as expected.)

Bash scripting

Example

```
#!/usr/bin/env bash
```

```
NAME="John"
echo "Hello $NAME!"
```

Variables

```
NAME="John"
echo $NAME
echo "$NAME"
echo "${NAME}!"
```

String quotes

```
NAME="John"
echo "Hi $NAME"           #=> Hi John
echo 'Hi $NAME'           #=> Hi $NAME
```

Shell execution

```
echo "I'm in $(pwd)"
echo "I'm in `pwd`"      # Same
```

Conditional execution

```
git commit && git push
git commit || echo "Commit failed"
```

Functions

```
get_name() {  
    echo "John"  
}  
  
echo "You are $(get_name)"
```

Conditionals

```
if [[ -z "$string" ]]; then  
    echo "String is empty"  
elif [[ -n "$string" ]]; then  
    echo "String is not empty"  
fi
```

Strict mode

```
set -euo pipefail  
IFS=$'\n\t'
```

Brace expansion

```
echo {A,B}.js  
{A,B}                Same as A B  
  
{A,B}.js            Same as A.js B.js  
  
{1..5}              Same as 1 2 3 4 5
```

Parameter expansions

Basics

```
name="John"  
echo ${name}  
echo ${name/J/j}    #=> "john" (substitution)  
echo ${name:0:2}     #=> "Jo" (slicing)  
echo ${name::2}      #=> "Jo" (slicing)  
echo ${name::-1}     #=> "Joh" (slicing)  
echo ${name: (-1)}   #=> "n" (slicing from right)  
echo ${name: (-2):1} #=> "h" (slicing from right)
```

```

echo ${food:-Cake}  #=> $food or "Cake"
length=2
echo ${name:0:length}  #=> "Jo"

STR="/path/to/foo.cpp"
echo ${STR%.cpp}      # /path/to/foo
echo ${STR%.cpp}.o    # /path/to/foo.o
echo ${STR%/*}        # /path/to

echo ${STR##*.}       # cpp (extension)
echo ${STR##*/}       # foo.cpp (basepath)

echo ${STR#*/}        # path/to/foo.cpp
echo ${STR##*/}       # foo.cpp

echo ${STR/foo/bar}   # /path/to/bar.cpp
STR="Hello world"
echo ${STR:6:5}       # "world"
echo ${STR: -5:5}     # "world"
SRC="/path/to/foo.cpp"
BASE=${SRC##*/}       #=> "foo.cpp" (basepath)
DIR=${SRC%$BASE}      #=> "/path/to/" (dirpath)

```

Substitution

<code>\${FOO%suffix}</code>	Remove suffix
<code>\${FOO#prefix}</code>	Remove prefix
<code>\${FOO%%suffix}</code>	Remove long suffix
<code>\${FOO##prefix}</code>	Remove long prefix
<code>\${FOO/from/to}</code>	Replace first match
<code>\${FOO//from/to}</code>	Replace all
<code>\${FOO/%from/to}</code>	Replace suffix
<code>\${FOO/#from/to}</code>	Replace prefix

Comments

```

# Single line comment
: '
This is a
multi line
comment
'

```

Substrings

`${FOO:0:3}` Substring (position, length)

`${FOO:(-3):3}` Substring from the right

Length

`${#FOO}` Length of \$FOO

Manipulation

```
STR="HELLO WORLD!"  
echo ${STR,}           #=> "hello WORLD!" (lowercase 1st letter)  
echo ${STR,,}          #=> "hello world!" (all lowercase)
```

```
STR="hello world!"  
echo ${STR^}           #=> "Hello world!" (uppercase 1st letter)  
echo ${STR^^}          #=> "HELLO WORLD!" (all uppercase)
```

Default values

`${FOO:-val}` \$FOO, or val if unset (or null)

`${FOO:=val}` Set \$FOO to val if unset (or null)

`${FOO:+val}` val if \$FOO is set (and not null)

`${FOO:?message}` Show error message and exit if \$FOO is
unset (or null)

Loops

Basic for loop

```
for i in /etc/rc.*; do  
    echo $i  
done
```

C-like for loop

```
for ((i = 0 ; i < 100 ; i++)); do  
    echo $i  
done
```

Ranges

```
for i in {1..5}; do
    echo "Welcome $i"
done
```

With step size

```
for i in {5..50..5}; do
    echo "Welcome $i"
done
```

Reading lines

```
cat file.txt | while read line; do
    echo $line
done
```

Forever

```
while true; do
    ...
done
```

Functions

Defining functions

```
myfunc() {
    echo "hello $1"
}
# Same as above (alternate syntax)
function myfunc() {
    echo "hello $1"
}
myfunc "John"
```

Returning values

```
myfunc() {
    local myresult='some value'
    echo $myresult
}
result="$(myfunc)"
```

Raising errors

```

myfunc() {
    return 1
}
if myfunc; then
    echo "success"
else
    echo "failure"
fi

```

Arguments

<code>##</code>	Number of arguments
<code>\$*</code>	All positional arguments (as a single word)
<code>\$@</code>	All positional arguments (as separate strings)
<code>\$1</code>	First argument
<code>\$_</code>	Last argument of the previous command
<code>\$?</code>	(Exit Status for the Last command) 0 (success) or 1(Failed or wrong or error)

Note: `$@` and `$*` must be quoted in order to perform as described. Otherwise, they do exactly the same thing (arguments as separate strings).

Conditionals

Conditions

Note that `[]` is actually a command/program that returns either 0 (true) or 1 (false). Any program that obeys the same logic (like all base utils, such as `grep(1)` or `ping(1)`) can be used as condition, see examples.

<code>[[-z STRING]]</code>	Empty string
<code>[[-n STRING]]</code>	Not empty string
<code>[[STRING == STRING]]</code>	Equal
<code>[[STRING1 \<= STRING2]]</code>	String 1 is smaller than String2
<code>[[STRING != STRING]]</code>	Not Equal

<code>[[NUM -eq NUM]]</code>	Equal
<code>[[NUM -ne NUM]]</code>	Not equal
<code>[[NUM -lt NUM]]</code>	Less than
<code>[[NUM -le NUM]]</code>	Less than or equal
<code>[[NUM -gt NUM]]</code>	Greater than
<code>[[NUM -ge NUM]]</code>	Greater than or equal
<code>[[STRING =~ STRING]]</code>	Regex
<code>((NUM < NUM))</code>	Numeric conditions

<code>[[-o noclobber]]</code>	If OPTIONNAME is enabled
<code>[[! EXPR]]</code>	Not
<code>[[X && Y]]</code>	And
<code>[[X Y]]</code>	Or

File conditions

<code>[[-e FILE]]</code>	Exists
<code>[[-r FILE]]</code>	Readable
<code>[[-h FILE]]</code>	Symlink
<code>[[-d FILE]]</code>	Directory
<code>[[-w FILE]]</code>	Writable
<code>[[-s FILE]]</code>	Size is > 0 bytes
<code>[[-f FILE]]</code>	File
<code>[[-x FILE]]</code>	Executable
<code>[[FILE1 -nt FILE2]]</code>	1 is more recent than 2
<code>[[FILE1 -ot FILE2]]</code>	2 is more recent than 1
<code>[[FILE1 -ef FILE2]]</code>	Same files

Example

```
# String
if [[ -z "$string" ]]; then
    echo "String is empty"
elif [[ -n "$string" ]]; then
    echo "String is not empty"
else
    echo "This never happens"
fi

# Combinations

if [[ X && Y ]]; then
    ...
fi

# Equal
if [[ "$A" == "$B" ]]

# Regex
if [[ "A" =~ . ]]
if (( $a < $b )); then
    echo "$a is smaller than $b"
fi
if [[ -e "file.txt" ]]; then
    echo "file exists"
fi
```

Arrays

Defining arrays

```
Fruits=('Apple' 'Banana' 'Orange')
Fruits[0]="Apple"
Fruits[1]="Banana"
Fruits[2]="Orange"
```

Working with arrays

echo \${Fruits[0]}	# Element #0
echo \${Fruits[-1]}	# Last element
echo \${Fruits[@]}	# All elements, space-separated
echo \${#Fruits[@]}	# Number of elements
echo \${#Fruits}	# String length of the 1st element
echo \${#Fruits[3]}	# String length of the Nth element
echo \${Fruits[@]:3:2}	# Range (from position 3, length 2)
echo \${!Fruits[@]}	# Keys of all elements, space-separated

Operations

```
Fruits=("${Fruits[@]}" "Watermelon")    # Push
Fruits+=('Watermelon')                  # Also Push
Fruits=( ${Fruits[@]/Ap*/} )            # Remove by regex match
unset Fruits[2]                          # Remove one item
Fruits=("${Fruits[@]}")                  # Duplicate
Fruits=("${Fruits[@]}" "${Veggies[@]}") # Concatenate
lines=(`cat "logfile"`)                 # Read from file
```

Iteration

```
for i in "${arrayName[@]"; do
    echo $i
done
```

Dictionaries

Defining

```
declare -A sounds
sounds[dog]="bark"
sounds[cow]="moo"
sounds[bird]="tweet"
sounds[wolf]="howl"
```

Working with dictionaries

```
echo ${sounds[dog]} # Dog's sound
echo ${sounds[@]}   # All values
echo ${!sounds[@]}  # All keys
echo ${#sounds[@]}  # Number of elements
unset sounds[dog]   # Delete dog
```

Iteration

```
for val in "${sounds[@]"; do
    echo $val
done
```

```
for key in "${!sounds[@]"; do
    echo $key
done
```

Options

Options

```
set -o noclobber # Avoid overlay files (echo "hi" > foo)
set -o errexit   # Used to exit upon error, avoiding cascading errors
set -o pipefail  # Unveils hidden failures
set -o nounset   # Exposes unset variables
```

Glob options

```
shopt -s nullglob # Non-matching globs are removed ('*.foo' => '')
shopt -s failglob # Non-matching globs throw errors
shopt -s nocaseglob # Case insensitive globs
shopt -s dotglob   # Wildcards match dotfiles ("*.sh" => ".foo.sh")
shopt -s globstar  # Allow ** for recursive matches ('lib/**/*.rb' =>
'lib/a/b/c.rb')
```

History

Commands

history	Show history
shopt -s histverify	Don't execute expanded result immediately

Expansions

!\$	Expand last parameter of most recent command
!*	Expand all parameters of most recent command
!-n	Expand nth most recent command
!n	Expand nth command in history
!<command>	Expand most recent invocation of command <command>

Operations

!!	Execute last command again
!!:s/<FROM>/<TO> /	Replace first occurrence of <FROM> to <TO> in most recent command

<code>!!:gs/<FROM>/<TO>/</code>	Replace all occurrences of <FROM> to <TO> in most recent command
<code>!\$:t</code>	Expand only basename from last parameter of most recent command
<code>!\$:h</code>	Expand only directory from last parameter of most recent command

!! and !\$ can be replaced with any valid expansion.

Slices

<code>!!:n</code>	Expand only nth token from most recent command (command is 0; first argument is 1)
<code>!^</code>	Expand first argument from most recent command
<code>!\$</code>	Expand last token from most recent command
<code>!!:n-m</code>	Expand range of tokens from most recent command
<code>!!:n-\$</code>	Expand nth token to last from most recent command

!! can be replaced with any valid expansion i.e. !cat, !-2, !42, etc.

Miscellaneous

Numeric calculations

```
$(a + 200)    # Add 200 to $a
$($RANDOM%200) # Random number 0..199
```

Subshells

```
(cd somedir; echo "I'm now in $PWD")
pwd # still in first directory
```

Redirection

```
python hello.py > output.txt    # stdout to (file)
python hello.py >> output.txt    # stdout to (file), append
python hello.py 2> error.log     # stderr to (file)
python hello.py 2>&1              # stderr to stdout
python hello.py 2>/dev/null      # stderr to (null)
python hello.py &>/dev/null      # stdout and stderr to (null)
python hello.py < foo.txt        # feed foo.txt to stdin for python
diff <(ls -r) <(ls)             # Compare two stdout without files
```

Inspecting commands

```
command -V cd           #=> "cd is a function/alias/whatever"
```

Trap errors

```
trap 'echo Error at about $LINENO' ERR
or
traperr() {
    echo "ERROR: ${BASH_SOURCE[1]} at about ${BASH_LINENO[0]}"
}

set -o erretrace
trap traperr ERR
```

Case/switch

```
case "$1" in
    start | up)
        vagrant up
        ;;

    *)
        echo "Usage: $0 {start|stop|ssh}"
        ;;
esac
```

Source relative

```
source "${0%/*}/../share/foo.sh"
```

printf

```
printf "Hello %s, I'm %s" Sven Olga
#=> "Hello Sven, I'm Olga"
```

```
printf "1 + 1 = %d" 2
#=> "1 + 1 = 2"
```

```
printf "This is how you print a float: %f" 2
#=> "This is how you print a float: 2.000000"
```

Transform strings

- | | |
|----|---|
| -c | Operations apply to characters not in the given set |
| -d | Delete characters |
| -s | Replaces repeated characters with single occurrence |

<code>-t</code>	Truncates
<code>[:upper:]</code>	All upper case letters
<code>[:lower:]</code>	All lower case letters
<code>[:digit:]</code>	All digits
<code>[:space:]</code>	All whitespace
<code>[:alpha:]</code>	All letters
<code>[:alnum:]</code>	All letters and digits

```
echo "Welcome To Devhints" | tr [:lower:] [:upper:]
WELCOME TO DEVHINTS
```

Directory of script

```
DIR="${0%/*}"
```

Getting options

```
while [[ "$1" =~ ^- && ! "$1" == "--" ]]; do case $1 in
  -V | --version )
    echo $version
    exit
    ;;
  -s | --string )
    shift; string=$1
    ;;
  -f | --flag )
    flag=1
    ;;
esac; shift; done
if [[ "$1" == '--' ]]; then shift; fi
```

Heredoc

```
cat <<END
hello world
END
```

Reading input

```
echo -n "Proceed? [y/n]: "
read ans
echo $ans
read -n 1 ans      # Just one character
```

Special variables

<code>\$?</code>	Exit status of last task
<code>\$!</code>	PID of last background task
<code>\$\$</code>	PID of current shell
<code>\$0</code>	Filename of the shell script
<code>\$_</code>	Last argument of the previous command

Go to previous directory

```
pwd # /home/user/foo
cd bar/
pwd # /home/user/foo/bar
cd -
pwd # /home/user/foo
```

Check for command's result

```
if ping -c 1 google.com; then
    echo "It appears you have a working internet connection"
fi
```

Grep check

```
if grep -q 'foo' ~/.bash_history; then
    echo "You appear to have typed 'foo' in the past"
fi
```

`$*` prints all the argument in single line

`${#mytext}` = prints no. of all the characters passed

`$@` will print all the argument in different lines

```
string="hello world"
```

```
echo "The length of the string is: ${#string}"
```

```
The length of the string is: 11
```

Echo `$?` ----- previous command 0--- successfully executed

1--- Unsuccessful (output not found)

Echo `$ARG{10}`----- prints 11th argument passed.

Arguments

`$#`

Number of arguments

`$*`

All positional arguments (as a single word)

`$@`

All positional arguments (as separate strings)

`$1`

First argument

`$_`

Last argument of the previous command

`$?`

Exit status of last task

`$!`

PID of last background task

`$$`

PID of shell

`$0`

Filename of the shell script

```
GNU nano 4.8                                backup.sh
#!/bin/bash

src_dir=/home/ubuntu/scripts
tgt_dir=/home/ubuntu/backups

curr_timestamp=$(date "+%Y-%m-%d-%H-%M-%S")
backup_file=$tgt_dir/$curr_timestamp.tgz

echo "Taking backup on $curr_timestamp"
#echo "$backup_file"

tar czf $backup_file --absolute-names $src_dir

echo "Backup complete"
```

```
#!/bin/bash
alert=90
df -H | awk '{print $5 " " $1}' | while read output;
do
    #echo "Disk Detail: $output"
    usage=$(echo $output | awk '{print $1}' | cut -d'%' -f1)
    file_sys=$(echo $output | awk '{print $2}')
    #echo $usage
    if [ $usage -ge $alert ]
    then
        echo "CRITICAL for $file_sys"
    fi
done
~
```

Digital Clock

Bash:

```
while :  
do echo -en "$(date +%T)\r"  
sleep 1  
done
```

Python:

```
from datetime import datetime  
import time  
  
while True:  
    now = datetime.now()  
    print (f"{now.hour}:{now.minute}:{now.second}", end = '\r')  
    time.sleep(1)
```


Filesystem Alert

Bash:

Method: 1

```
df -H | grep -Po "^/dev/(?!.*snap).*" | while read i
do
percent=$(echo $i | awk 'gsub("%", ""){print $5}')
fs=$(echo $i | awk '{print $1}')

if (( $percent ≥ 80 )); then
echo "Running out of space $fs $percent% on $(hostname)."
fi
done
```

Bash:

Method: 2

```
df -H --output=source,pcent | grep -Po "^/dev/(?!.*
(snap|loop)).*" \
| sed 's/%//g' | while read -r fs pc
do
if (( $pc > 80 ))
then
echo "Running out of space $fs $pc% on $HOSTNAME."
fi
done
```

Python:

```
from os import statvfs as svfs
from re import compile, search
from pathlib import Path

comp = compile(r'^/dev/(?!.*snap)')
mline = Path('/proc/mounts').read_text().splitlines()
fs = [i.split()[1] for i in mline if comp.search(i)]

for x in fs:
    def usep(x, y):
        return f"{{(x - y)/x:.0%}}"
    percent = usep(svfs(x).f_blocks,svfs(x).f_bfree)
    if int(percent.replace('%','')) > 80:
        print(f'Running out of space {x} abvoe {percent:80}.'
```

Output

Running out of space / abvoe 80% .

How To Validate Mount Points?

Bash:

```
fstab_mount=( $(awk '/^UUID/{print $2}' /etc/fstab) )
current_mount=( $(grep -P '^/dev/(?!.*snap)' /proc/mounts | \
awk '{print $2}'))
for i in ${fstab_mount[@]}
do
if [[ ${current_mount[@]} != *"$i"* ]]
then
echo "These Mount Point Not Mounted:"
grep "$i" /etc/fstab | awk '{print $2}'
out='fail'
fi
done
if [ -z $out ]
then
echo "All Disks Are Mounted Correctly."
fi
```

Python:

```
from pathlib import Path
from re import search

fstab_read = Path('/etc/fstab').read_text().splitlines()
fstab = {x.split()[1] for x in fstab_read
         if x.startswith('UUID')}
mounts_read = Path('/proc/mounts').read_text().splitlines()
mounts = {y.split()[1] for y in mounts_read
          if search(r'^/dev/(?!.*snap)', y)}
out = fstab - mounts

if not out:
    print("All Disks Are Mounted Correctly.")
else:
    print('These Mount Point Not Mounted:', *out, sep = '\n')
```

Output:

All Disks Are Mounted Correctly.

Last Modification Time Of A File

Bash:

```
stat -c "%y" /home/mana/Work/fruit.txt
```

Output

```
stat -c "%y" /home/mana/Work/fruit.txt
```

Python:

```
from time import ctime
import os.path
```

```
file = '/home/mana/Work/fruit.txt'
mseconds = os.path.getmtime(file)
print(ctime(mseconds))
```

Output

```
Tue Jun 23 18:53:54 2020
```

How to view Logs Specific Range Of Time?

Python:

```
from datetime import datetime, timedelta
from pathlib import Path
from re import search

start_time = 'Apr 19 02:00'
end_time = 'Apr 19 02:15'
log_lines =
Path('/var/log/syslog').read_text().splitlines()

for x in log_lines:
    log_time =
search(r'^\w{3}\s+\d+\s\d{2}:\d{2}',x).group()
    if start_time ≤ log_time ≤ end_time:
        print(x)
```

Check Status Of Services And Store Into CSV File

Bash:

```
systemctl --type=service | sed -n '/service/p' | \
sed -rn 's/^(.*)loaded.*(running|exited|failed)(.+)/\1,
\2/p' | sort -rk3 > ~/Work/status.csv
```

Python:

```
from subprocess import check_output
from re import search
import csv

cmd = 'systemctl --type=service'
services = check_output(cmd.split()).decode().splitlines()[1:]
status = []

for i in services:
    match = search(r'^(.*)loaded.*(running|exited|failed)', i)
    if match:
        status.append(list(match.groups()))

status = sorted(status, key = lambda x: x[1], reverse = True)

with open('/home/mana/Work/status.csv', 'w') as file:
    writer = csv.writer(file)
    writer.writerows(status)
```

Starting and Stopping AWS EC2 Instance By BOT037

Starting EC2 Instance

```
import boto3
AWS_REGION = "ap-south-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-XXXXXXXXXXXX'
instance = EC2_RESOURCE.Instance(INSTANCE_ID)
instance.start()
print(f'Starting EC2 instance: {instance.id}')
instance.wait_until_running()
print(f'EC2 instance "{instance.id}" has been started')
```

Stopping EC2 Instance

```
import boto3
AWS_REGION = "ap-south-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-XXXXXXXXXXXX'
instance = EC2_RESOURCE.Instance(INSTANCE_ID)
instance.stop()
print(f'Stopping EC2 instance: {instance.id}')
instance.wait_until_stopped()
print(f'EC2 instance "{instance.id}" has been stopped')
```


How to Reboot and Terminate AWS EC2 Instance?

How to Reboot and Terminate AWS EC2 Instance?

Rebooting EC2 Instance

```
import boto3
AWS_REGION = "ap-north-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-XXXXXXXXXXXXXX'
instance = EC2_RESOURCE.Instance(INSTANCE_ID)
instance.reboot()
print(f'EC2 instance "{instance.id}" has been rebooted')
```

Terminating EC2 Instance

```
import boto3
AWS_REGION = "ap-north-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-XXXXXXXXXXXXXX'
instance = EC2_RESOURCE.Instance(INSTANCE_ID)
instance.terminate()
print(f'Terminating EC2 instance: {instance.id}')
instance.wait_until_terminated()
print(f'EC2 instance "{instance.id}" has been terminated')
```

Listing Amazon All S3 Buckets?

How To List S3 Buckets Using Boto3 Client?

```
import boto3
AWS_REGION = "ap-north-2"
client = boto3.client("s3", region_name=AWS_REGION)
response = client.list_buckets()
print("Listing Amazon S3 Buckets:")
for bucket in response['Buckets']:
    print(f"{bucket['Name']}")
```

How To List S3 Buckets Using Boto3 Resource?

```
import boto3
AWS_REGION = "ap-north-2"
resource = boto3.resource("s3", region_name=AWS_REGION)
iterator = resource.buckets.all()
print("Listing Amazon S3 Buckets:")
for bucket in iterator:
    print(f"{bucket.name}")
```


How To Increase The Size Of The EBS Volume Using Boto3?

```
import boto3
import time

AWS_REGION = "ap-north-2"
EC2_CLIENT = boto3.client('ec2', region_name=AWS_REGION)
VOLUME_ID = 'vol-xxxxxxxxxx'

def get_modification_state(volume_id):
    response = EC2_CLIENT.describe_volumes_modifications(
        VolumeIds=[
            VOLUME_ID
        ]
    )
    return response['VolumesModifications'][0]
['ModificationState']
modify_volume_response = EC2_CLIENT.modify_volume(
    VolumeId=VOLUME_ID,
    Size=30
)
while True:
    state = get_modification_state(VOLUME_ID)
    if state == 'completed' or state == None:
        break
    elif state == 'failed':
        raise Exception('Failed to modify volume size')
    else:
        time.sleep(60)
print(f'Volume {VOLUME_ID} successfully resized')
```

Creating AWS RDS DB Instance By Boto3

How To Create AWS RDS DB Instance Using Boto3?

```
import boto3
client = boto3.client('rds')
response = client.create_db_instance(
    AllocatedStorage=4,
    DBInstanceClass='db.t2.micro',
    DBInstanceIdentifier='database-instance-01',
    Engine='MySQL',
    MasterUserPassword='xxxxxxx',
    MasterUsername='test',
)
print(response)
```

Creating S3 Bucket in AWS

How to Create S3 Bucket Using Boto3 Client?

```
import boto3
AWS_REGION = "ap-north-2"
client = boto3.client("s3", region_name=AWS_REGION)
bucket_name = "Axxa-bucket"
location = {'LocationConstraint': AWS_REGION}
response = client.create_bucket(Bucket=bucket_name,
CreateBucketConfiguration=location)
print(f"Amazon {bucket_name} S3 bucket has been created")
```

How to Create S3 Bucket Using Boto3 Resource?

```
import boto3
AWS_REGION = "ap-north-2"
resource = boto3.resource("s3", region_name=AWS_REGION)
bucket_name = "Axxa-bucket"
location = {'LocationConstraint': AWS_REGION}
bucket = resource.create_bucket(
    Bucket=bucket_name,
    CreateBucketConfiguration=location)
print(f"Amazon {bucket_name} S3 bucket has been created")
```

Creating A AWS EC2 Security Group

```
import boto3
AWS_REGION = "ap-north-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
VPC_ID = 'vpc-xxxxxx'
security_group = EC2_RESOURCE.create_security_group(
    Description='Allow inbound SSH traffic',
    GroupName='allow-inbound-ssh',
    VpcId=VPC_ID,
    TagSpecifications=[
        {
            'ResourceType': 'security-group',
            'Tags': [
                {
                    'Key': 'Name',
                    'Value': 'allow-inbound-ssh'
                }
            ]
        }
    ],
)
security_group.authorize_ingress(
    CidrIp='0.0.0.0/0',
    FromPort=22,
    ToPort=22,
    IpProtocol='tcp',
)
print(f'Security Group {security_group.id} has been created')
```

Server Health Report

```
#!/bin/bash
###Author: Manavalan Michael###
top_process () {
tp=$(top -bn2 -d1 | awk '/PID/{i+=1}i==2 {print $1,$12,$9}' | head -11)
readarray -t records < <(echo "$tp")
for i in `seq 0 10`; do echo -e "${records[i]}"; done
}

###Health Check Report (CPU,Process,Disk Usage, Memory)###
echo "#####
Health Check Report (CPU,Process,Disk Usage, Memory)
#####
Hostname : `hostname`
Kernel Version : `uname -mrs`
Uptime : `uptime | awk '{gsub (",","");print $3}`
Last Reboot Time : `who -b | awk '{print $3,$4}`
Load Average : `uptime | awk -F'[!^a-z,]*' '{ gsub("[:]", "");print $6 }' | tr -d " "`
CPU Usage : `top -bn2 -d1 | awk '/^top/{i+=1}i==2' | awk -F '[!a-z,]*' 'NR==3 { gsub ("[%]", "");print 100-$6}%'`
IO Wait : `top -bn2 -d1 | awk '/^top/{i+=1}i==2' | awk -F '[!a-z,]*' 'NR==3 { gsub ("[%]", "");print $7}%' | tr -d " "`
echo ""
echo ""
"*****
Health Status
*****"
ncpu=$(grep "model name" /proc/cpuinfo | wc -l)
echo "Number of cpu : $ncpu"
echo "Health Status : `uptime | awk -F'[!^a-z,]*' '{ gsub("[:]", "");print $6 }' | awk '{ if ($1>$ncpu+1) print "Unhealthy";else print "Normal"}'` "
echo ""
echo ""
"*****
Process
*****"
⇒ Top CPU using process/application
`top_process | column -t`
⇒ Top memory using process/application
`ps -A --sort -rss -o pid,comm,pmem | head | column -t`
echo ""
echo ""
"*****
Disk Usage && Disk Status
*****"
echo "`df -h | egrep -iv 'tmpfs|filesystem|none|udev' | awk '{print $1,$6,$4\"free\",$3\"used\"}' | gsub (\"[%]\", \"\");if ($5>95)print \"Unheathy\";else if ($5>90) print \"Caution\"; else print \"Normal\"}' | sed 'N;s/\n/ /' | column -t`"
echo ""
echo ""
"*****
Memory
*****"
⇒ Physical Memory
totmem=`free -m | awk 'NR==2 {printf "%2.2f\n", $2/1024}`
usemem=`free -m | awk 'NR==3 {printf "%2.2f\n", $3/1024}`
freemem=`free -m | awk 'NR==3 {printf "%2.2f\n", $4/1024}`
freeper=$(echo "$freemem * 100/$totmem" | bc)
echo -e "Total\tUsed\tFree\tFree
${totmem}GB\t${usemem}GB \t${freemem}GB\t${freeper}%"
echo ""
echo "⇒ Swap Memory"
totswap=`free -m | awk '/Swap/{printf "%2.2f\n", $2/1024}`
useswap=`free -m | awk '/Swap/{printf "%2.2f\n", $3/1024}`
freeswap=`free -m | awk '/Swap/{printf "%2.2f\n", $4/1024}`
freeper=$(echo "$freeswap * 100/$totswap" | bc)
echo -e "Total\tUsed\tFree\tFree
${totswap}GB\t${useswap}GB \t${freeswap}GB\t${freeper}%"
```


How To Search Directory Recursively?

Bash:

```
find /home/mana/Work -type d -iname "Test" -printf  
"%f" Directory exists."
```

Python:

```
from pathlib import Path  
  
mydir = input('Enter Search Directory:').casefold()  
dir_list = Path('/home/mana/').rglob('Work/*')  
only_dir = [i.name for i in dir_list if i.is_dir()]  
if mydir in only_dir:  
    print(f'"{mydir}" Directory exists.')  
else:  
    print(f'"{mydir}" does not Directory exists.')
```

Output:

```
Enter Search Directory:Test  
"test" Directory exists.
```

Copying files between S3 buckets

```
"""
This program copies objects from source to destination S3
bucket
"""

import boto3
REGION = 'ap-north-2'
SRC_BUCKET = 'Axxa-src-bucket'
DST_BUCKET = 'Axxa-dst-bucket'
S3_RESOURCE = boto3.resource('s3', region_name=REGION)

def copy_objects():
    src_bucket = S3_RESOURCE.Bucket(SRC_BUCKET)
    dst_bucket = S3_RESOURCE.Bucket(DST_BUCKET)
    print(f'Source S3 bucket: {SRC_BUCKET}')
    print(f'Destination S3 bucket: {DST_BUCKET}')

    for s3_object in src_bucket.objects.all():
        print(f'Copying object: {s3_object.key}')
        source_file_data = {
            'Bucket': SRC_BUCKET,
            'Key': s3_object.key
        }
        dst_bucket.copy(source_file_data, s3_object.key)

copy_objects()
```

Calculate Your Age

Bash:

```
read -p "Enter Your Birth Year: " yr
printf "%2d\n" $(date -d "-$(date +%yr) year" +%Y)
```

Python:

```
from datetime import date
birth_yr = int(input('Enter Your Birth Year: '))
print(date.today().year - birth_yr)
```

Output:

```
Enter Your Birth Year: 1985
35
```


Creating AWS CloudWatch Log Group Using Boto3

```
import boto3
import json
AWS_REGION = "ap-north-2"
client = boto3.client('logs', region_name=AWS_REGION)
retention_period_in_days = 5
# Back end Log Group
log_group = 'CRMBackendLogs'
response = client.create_log_group(
    logGroupName=log_group,
    tags={
        'Type': 'Back end',
        'Frequency': '30 seconds',
        'Environment': 'Production',
        'RetentionPeriod': str(retention_period_in_days)
    }
)
print(json.dumps(response, indent=4))
response = client.put_retention_policy(
    logGroupName=log_group,
    retentionInDays=retention_period_in_days
)
print(json.dumps(response, indent=4))
# Front end Log Group
log_group = 'CRMFrontendLogs'
response = client.create_log_group(
    logGroupName=log_group,
    tags={
        'Type': 'Front end',
        'Frequency': '30 seconds',
        'Environment': 'Production',
        'RetentionPeriod': str(retention_period_in_days)
    }
)
response = client.put_retention_policy(
    logGroupName=log_group,
    retentionInDays=retention_period_in_days
)
```

Useful Git Commands Cheatsheet

✓ git init	→	Initializes a new Git repository
✓ git add <files>	→	Adds files to the staging area.
✓ git status	→	Used to check the state of the staging area, as well as the working directory
✓ git log	→	Used to view the entire commit history.
✓ git commit -m "message"	→	Used to commit files (locally) on the repository.
✓ git clone	→	Used to download existing code from a remote repository.
✓ git branch	→	Used to list all the local branches on the machine.
✓ git merge <branch-name>	→	Merges the provided branch with the current working branch.
✓ git branch <branch-name>	→	Used to create a new branch locally.
✓ git branch -d <branch-name>	→	Used to delete a branch.
✓ git branch -m <new-name>	→	Used to rename the current working branch.
✓ git checkout <branch-name>	→	Used to switch from the current branch to another one.
✓ git push <remote> <branch-name>	→	Used to save all commits to the remote repository.
✓ git checkout -b <branch-name>	→	Creates a new branch and switches to the new one.
✓ git pull <remote>	→	Used to pull down all the updates from the remote repository.
✓ git rm <file-name>	→	Used to remove a file from the working directory.
✓ git stash	→	Used to temporarily remove uncommitted changes.
✓ git reset	→	Undo the changes to the local files, and restore to the last commit.
✓ git diff	→	Displays the difference between files in two commits or between a commit and your current repository.

Attaching An Elastic IP To An EC2 Instance

```
import boto3
AWS_REGION = "ap-north-2"
EC2_CLIENT = boto3.client('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-xxxxxxxxxxxxx'
response = EC2_CLIENT.describe_addresses(
    Filters=[
        {
            'Name': 'tag:Name',
            'Values': ['my-elastic-ip']
        }
    ]
)
public_ip = response['Addresses'][0]['PublicIp']
allocation_id = response['Addresses'][0]['AllocationId']
response = EC2_CLIENT.associate_address(
    InstanceId=INSTANCE_ID,
    AllocationId=allocation_id
)
print(f'EIP {public_ip} associated with the instance {INSTANCE_ID}')
```

Creating A AWS EC2 Security Group

```
import boto3
AWS_REGION = "ap-north-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
VPC_ID = 'vpc-xxxxxx'
security_group = EC2_RESOURCE.create_security_group(
    Description='Allow inbound SSH traffic',
    GroupName='allow-inbound-ssh',
    VpcId=VPC_ID,
    TagSpecifications=[
        {
            'ResourceType': 'security-group',
            'Tags': [
                {
                    'Key': 'Name',
                    'Value': 'allow-inbound-ssh'
                }
            ]
        }
    ],
)
security_group.authorize_ingress(
    CidrIp='0.0.0.0/0',
    FromPort=22,
    ToPort=22,
    IpProtocol='tcp',
)
print(f'Security Group {security_group.id} has been created')
```

How google map works?

What is Latitude and Longitude?

Latitude and Longitude are the units that represent the coordinates at geographic coordinate system.

Latitudes and longitudes are horizontal and vertical lines respectively.

Latitudes are parallel to equator and equator itself is also a latitude i.e. 0° Latitude.

Longitude intersects with equator at 90°

Python Code:

```
import bisect

lati_long = [[], [20.5937, 78.9629], [37.0902, 95.7129],
              [52.1326, 5.2913], [55.3781, 3.4360]]
country = ['', 'India', 'USA', 'Netherlands', 'UK']

def map(arg):
    idx = bisect.bisect(lati_long, arg) - 1
    return f'Latitude {arg[0]}, Longitude {arg[1]} is {country[idx]}'

map([55.3781, 3.4360])
```


How to Reboot and Terminate AWS EC2 Instance?

How to Reboot and Terminate AWS EC2 Instance?

Rebooting EC2 Instance

```
import boto3
AWS_REGION = "ap-north-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-XXXXXXXXXXXXXXX'
instance = EC2_RESOURCE.Instance(INSTANCE_ID)
instance.reboot()
print(f'EC2 instance "{instance.id}" has been rebooted')
```

Terminating EC2 Instance

```
import boto3
AWS_REGION = "ap-north-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-XXXXXXXXXXXXXXX'
instance = EC2_RESOURCE.Instance(INSTANCE_ID)
instance.terminate()
print(f'Terminating EC2 instance: {instance.id}')
instance.wait_until_terminated()
print(f'EC2 instance "{instance.id}" has been terminated')
```

Listing Amazon All S3 Buckets?

How To List S3 Buckets Using Boto3 Client?

```
import boto3
AWS_REGION = "ap-north-2"
client = boto3.client("s3", region_name=AWS_REGION)
response = client.list_buckets()
print("Listing Amazon S3 Buckets:")
for bucket in response['Buckets']:
    print(f"{bucket['Name']}")
```

How To List S3 Buckets Using Boto3 Resource?

```
import boto3
AWS_REGION = "ap-north-2"
resource = boto3.resource("s3", region_name=AWS_REGION)
iterator = resource.buckets.all()
print("Listing Amazon S3 Buckets:")
for bucket in iterator:
    print(f"{bucket.name}")
```

Creating S3 Bucket in AWS

How to Create S3 Bucket Using Boto3 Client?

```
import boto3
AWS_REGION = "ap-north-2"
client = boto3.client("s3", region_name=AWS_REGION)
bucket_name = "Axxa-bucket"
location = {'LocationConstraint': AWS_REGION}
response = client.create_bucket(Bucket=bucket_name,
CreateBucketConfiguration=location)
print(f"Amazon {bucket_name} S3 bucket has been created")
```

How to Create S3 Bucket Using Boto3 Resource?

```
import boto3
AWS_REGION = "ap-north-2"
resource = boto3.resource("s3", region_name=AWS_REGION)
bucket_name = "Axxa-bucket"
location = {'LocationConstraint': AWS_REGION}
bucket = resource.create_bucket(
    Bucket=bucket_name,
    CreateBucketConfiguration=location)
print(f"Amazon {bucket_name} S3 bucket has been created")
```