Mastering Shell Scripting: Essential Scripts for Daily Use, Automation, and Interviews

Shell scripting is an indispensable skill for anyone working in IT, DevOps, or system administration. Whether you're automating repetitive tasks, streamlining processes, or preparing for interviews, shell scripting offers immense value. In this blog, we'll cover essential shell scripts ranging from basic to advanced levels, including scripts frequently used in daily tasks and commonly discussed in interviews. Let's dive in!

---

1. File Backup Script

Automating file backups is crucial for maintaining data integrity. Here's a simple script to perform both full and incremental backups:

Full Backup Script:

```
#!/bin/bash
src_dir="/path/to/source"
backup_dir="/path/to/backup"
timestamp=$(date +%Y%m%d_%H%M%S)
backup_file="$backup_dir/backup_full_$timestamp.tar.gz"

tar -czf $backup_file $src_dir
echo "Full backup completed: $backup_file"
```

Incremental Backup Script:

```
#!/bin/bash
src_dir="/path/to/source"
backup_dir="/path/to/backup"
timestamp=$(date +%Y%m%d_%H%M%S)
backup_file="$backup_dir/backup_incremental_$timestamp.tar.gz"
last_backup="$backup_dir/last_backup.txt"

if [ -f "$last_backup" ]; then
  find $src_dir -type f -newer $last_backup > changed_files.txt
else
  find $src_dir -type f > changed_files.txt
fi

tar -czf $backup_file -T changed_files.txt
touch $last_backup
echo "Incremental backup completed: $backup_file"
```

---

2. System Monitoring Script

System health checks are vital for maintaining operational efficiency. This script monitors CPU and memory usage:

```
#!/bin/bash
cpu_usage=$(top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | awk '{print 100 - $1}')
memory_usage=$(free | grep Mem | awk '{print $3/$2 * 100.0}')
```

```
echo "CPU Usage: $cpu_usage%"
echo "Memory Usage: $memory_usage%"
```

3. User Account Management Script

Managing user accounts is a common administrative task. This script adds or removes users:

```
#!/bin/bash
manage_user() {
  username=$1
  action=$2

  if [ "$action" == "add" ]; then
    sudo useradd -m $username
    echo "User $username added."
  elif [ "$action" == "remove" ]; then
    sudo userdel -r $username
    echo "User $username removed."
  else
    echo "Invalid action. Use 'add' or 'remove'."
  fi
}

if [ -z "$1" ] || [ -z "$2" ]; then
  echo "Usage: $0 <username> <add|remove>"
else
  manage_user $1 $2
fi
```

---

4. Log Analyzer Script

Analyzing logs is essential for troubleshooting. This script extracts critical errors from a log file:

```
#!/bin/bash
log_file="/path/to/logfile.log"
output_file="critical_errors.log"

grep -i "error" $log_file > $output_file
echo "Critical errors saved to $output_file"
```

Rotating Logs:

```
#!/bin/bash
log_dir="/path/to/logs"
max_size=1000000  # 1 MB

for file in $log_dir/*.log; do
  log_size=$(stat -c %s "$file")
  if [ $log_size -gt $max_size ]; then
    mv "$file" "$file.old"
    echo "Log file $file rotated."
  fi
done
```

---

5. Password Generator Script

Creating strong passwords is essential for security. This script generates random passwords:

```
#!/bin/bash
length=${1:-12}
password=$(tr -dc 'A-Za-z0-9!@#$%^&*()_+=' < /dev/urandom | head -c $length)
echo "Generated Password: $password"
```

---

6. Automated Software Installation Script

This script simplifies software installation, especially on new systems:

```
#!/bin/bash
software_list=(git vim curl wget)

for software in "${software_list[@]}"; do
  if ! dpkg -l | grep -q $software; then
    sudo apt-get install -y $software
    echo "$software installed."
  else
    echo "$software is already installed."
  fi
done
```

Explanation: This script checks if a package is already installed before attempting installation. Modify the software_list array for your requirements.

---

7. Network Connectivity Checker Script

This script checks network connectivity:

```
#!/bin/bash
host="google.com"

if ping -c 1 $host > /dev/null; then
  echo "$host is reachable."
else
  echo "$host is unreachable."
fi
```

---

8. File Encryption/Decryption Script

Secure your files with encryption and decryption:

```
#!/bin/bash
action=$1
file=$2
output=$3

if [ "$action" == "encrypt" ]; then
  openssl enc -aes-256-cbc -salt -in $file -out $output
  echo "File encrypted: $output"
elif [ "$action" == "decrypt" ]; then
  openssl enc -aes-256-cbc -d -in $file -out $output
  echo "File decrypted: $output"
else
  echo "Usage: $0 <encrypt|decrypt> <file> <output>"
fi
```

9. Data Cleanup Script

This script cleans up temporary files and unnecessary logs:

```
#!/bin/bash
cleanup_dirs=("/tmp" "/var/tmp")

for dir in "${cleanup_dirs[@]}"; do
  find $dir -type f -atime +7 -exec rm -f {} \;
  echo "Cleaned up $dir."
done
```

---

10. Website Uptime Checker Script

Monitor website uptime with this script:

```
#!/bin/bash
url="https://example.com"

if curl -s --head $url | grep "200 OK" > /dev/null; then
  echo "$url is up."
else
  echo "$url is down."
fi
```

---

11. System Information Script

Retrieve detailed system information:

```
#!/bin/bash
echo "System Information:"
uname -a
echo "CPU Information:"
lscpu
echo "Memory Information:"
free -h
echo "Disk Information:"
df -h
```

---

12. Task Scheduler Script

Automate tasks using cron:

```
#!/bin/bash
backup_script="/path/to/backup/script.sh"
cron_time="0 3 * * *"  # Runs at 3 AM daily

(crontab -l; echo "$cron_time $backup_script") | crontab -
echo "Task scheduled to run daily at 3 AM."
```

---

13. Disk Space Monitoring Script

Monitor and report low disk space:

```
#!/bin/bash
threshold=80

for partition in $(df -h | grep "^/dev" | awk '{print $5 " " $1}'); do
  usage=$(echo $partition | awk '{print $1}' | sed 's/%//')
  mount=$(echo $partition | awk '{print $2}')
  if [ $usage -ge $threshold ]; then
    echo "Warning: $mount is $usage% full."
  fi
done
```

Here's a collection of **cloud automation and DevOps-oriented shell scripts** that are highly practical in real-world scenarios. These scripts cover areas like cloud resource management, configuration, CI/CD tasks, monitoring, and troubleshooting.

---

1. AWS Cloud Automation Scripts

1.1 Create an S3 Bucket

bash
Copy code

```
#!/bin/bash
bucket_name="my-unique-bucket-$(date +%s)"
aws s3 mb s3://$bucket_name --region us-east-1
echo "S3 bucket $bucket_name created."
```

1.2 Upload Files to an S3 Bucket

bash
Copy code

```
#!/bin/bash
bucket_name="my-bucket-name"
file_path="/path/to/your/file"
aws s3 cp $file_path s3://$bucket_name/
echo "File uploaded to $bucket_name."
```

1.3 Launch an EC2 Instance

bash
Copy code

```
#!/bin/bash
aws ec2 run-instances \
    --image-id ami-12345678 \
    --count 1 \
    --instance-type t2.micro \
    --key-name my-key-pair \
    --security-group-ids sg-12345678 \
    --subnet-id subnet-12345678
echo "EC2 instance launched."
```

1.4 Stop All Running EC2 Instances

bash
Copy code

```
#!/bin/bash
instances=$(aws ec2 describe-instances \
    --filters "Name=instance-state-name,Values=running" \
    --query "Reservations[*].Instances[*].InstanceId" \
    --output text)

if [ -n "$instances" ]; then
    aws ec2 stop-instances --instance-ids $instances
    echo "Stopped instances: $instances"
else
    echo "No running instances found."
fi
```

---

2. Kubernetes (k8s) Automation Scripts

2.1 List All Pods in a Namespace

bash
Copy code
#!/bin/bash
namespace="default"
kubectl get pods -n $namespace

2.2 Deploy a Kubernetes Manifest

bash
Copy code
#!/bin/bash
manifest_path="/path/to/manifest.yaml"
kubectl apply -f $manifest_path
echo "Manifest applied: $manifest_path"

2.3 Check Pod Logs

bash
Copy code
#!/bin/bash
pod_name="my-pod"
kubectl logs $pod_name

2.4 Scale a Deployment

bash
Copy code
#!/bin/bash
deployment_name="my-deployment"
replicas=3
kubectl scale deployment $deployment_name --replicas=$replicas
echo "Scaled $deployment_name to $replicas replicas."

---

3. CI/CD Automation Scripts

3.1 Build and Push Docker Images

bash
Copy code
#!/bin/bash
image_name="my-app"
tag="latest"
docker build -t $image_name:$tag .
docker tag $image_name:$tag my-docker-repo/$image_name:$tag
docker push my-docker-repo/$image_name:$tag
echo "Docker image pushed: my-docker-repo/$image_name:$tag"

3.2 Trigger Jenkins Job

bash
Copy code
#!/bin/bash
jenkins_url="http://jenkins-server/job/my-job/build"
curl -X POST $jenkins_url --user "username:token"
echo "Jenkins job triggered."

3.3 Rollback to a Previous Kubernetes Deployment

bash
Copy code
```
#!/bin/bash
deployment_name="my-deployment"
kubectl rollout undo deployment/$deployment_name
echo "Rollback performed for deployment: $deployment_name."
```

---

4. Monitoring and Troubleshooting Scripts

4.1 Monitor CPU and Memory Usage

bash
Copy code
```
#!/bin/bash
top -b -n1 | head -n 20
```

4.2 Check Disk Space Usage

bash
Copy code
```
#!/bin/bash
df -h
```

4.3 Monitor a Website's Uptime

bash
Copy code
```
#!/bin/bash
url="https://example.com"
if curl -s --head $url | grep "200 OK" > /dev/null; then
   echo "$url is up."
else
   echo "$url is down."
fi
```

---

5. Infrastructure as Code (IaC) Automation

5.1 Deploy Terraform Infrastructure

bash
Copy code
```
#!/bin/bash
cd /path/to/terraform/code
terraform init
terraform apply -auto-approve
echo "Terraform infrastructure deployed."
```

5.2 Destroy Terraform Infrastructure

bash
Copy code
```
#!/bin/bash
cd /path/to/terraform/code
terraform destroy -auto-approve
echo "Terraform infrastructure destroyed."
```

---

6. Cloud Networking Automation

6.1 Create a Security Group

bash
Copy code
```
#!/bin/bash
aws ec2 create-security-group \
    --group-name my-sg \
    --description "My Security Group" \
    --vpc-id vpc-12345678
echo "Security group created."
```

6.2 Allow SSH Access to Security Group

bash
Copy code
```
#!/bin/bash
aws ec2 authorize-security-group-ingress \
    --group-name my-sg \
    --protocol tcp \
    --port 22 \
    --cidr 0.0.0.0/0
echo "SSH access enabled for security group."
```

---

7. Backup and Recovery Automation

7.1 Automated Directory Backup

bash
Copy code
```
#!/bin/bash
src_dir="/path/to/source"
dest_dir="/path/to/backup"
timestamp=$(date +%Y%m%d_%H%M%S)
backup_file="$dest_dir/backup_$timestamp.tar.gz"

tar -czf $backup_file $src_dir
echo "Backup completed: $backup_file"
```

7.2 Restore from Backup

bash
Copy code
```
#!/bin/bash
backup_file="/path/to/backup.tar.gz"
restore_dir="/path/to/restore"
tar -xzf $backup_file -C $restore_dir
echo "Restore completed."
```

---

8. Logging and Auditing Scripts

8.1 System Logs Analysis

bash
Copy code
```
#!/bin/bash
log_file="/var/log/syslog"
grep "ERROR" $log_file | tail -n 20
```

8.2 Rotate Logs

bash
Copy code
```
#!/bin/bash
log_file="application.log"
mv $log_file $(date +%Y%m%d)_$log_file
touch $log_file
echo "Log rotated."
```

---

9. Miscellaneous

9.1 Clean Up Old Files

bash
Copy code
```
#!/bin/bash
find /path/to/directory -type f -mtime +30 -exec rm {} \;
echo "Old files cleaned up."
```

9.2 Check Service Status

bash
Copy code
```
#!/bin/bash
service_name="nginx"
if systemctl is-active --quiet $service_name; then
    echo "$service_name is running."
else
    echo "$service_name is not running."
fi
```

Scripts

1.1 Provision an AWS EC2 Instance with a Specific Configuration

bash
Copy code
```
#!/bin/bash
instance_type="t2.micro"
key_name="my-key"
security_group="sg-12345678"
subnet_id="subnet-12345678"
ami_id="ami-12345678"

aws ec2 run-instances \
    --image-id $ami_id \
    --count 1 \
    --instance-type $instance_type \
    --key-name $key_name \
    --security-group-ids $security_group \
    --subnet-id $subnet_id \
    --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=MyInstance}]'

echo "EC2 instance provisioned with type $instance_type."
```

1.2 Dynamic Scaling of AWS EC2 Instances Based on Load

bash
Copy code

```
#!/bin/bash
cpu_threshold=70
scale_up_count=2

avg_cpu=$(aws cloudwatch get-metric-statistics \
   --namespace AWS/EC2 \
   --metric-name CPUUtilization \
   --statistics Average \
   --period 300 \
   --start-time $(date -u -d '5 minutes ago' +%Y-%m-%dT%H:%M:%SZ) \
   --end-time $(date -u +%Y-%m-%dT%H:%M:%SZ) \
   --query 'Datapoints[0].Average' --output text)

if (( $(echo "$avg_cpu > $cpu_threshold" | bc -l) )); then
   echo "Scaling up by $scale_up_count instances."
   aws autoscaling set-desired-capacity \
      --auto-scaling-group-name my-auto-scaling-group \
      --desired-capacity $((current_capacity + scale_up_count))
else
   echo "No scaling needed. CPU usage is $avg_cpu%."
fi
```

1.3 Automatically Delete Unused S3 Buckets

bash
Copy code

```
#!/bin/bash
buckets=$(aws s3api list-buckets --query 'Buckets[*].Name' --output text)

for bucket in $buckets; do
   if ! aws s3 ls s3://$bucket; then
      echo "Deleting unused bucket: $bucket"
      aws s3 rb s3://$bucket --force
   fi
done
```

---

2. DevOps Automation Scripts

2.1 Automate CI/CD Pipeline Trigger

bash
Copy code

```
#!/bin/bash
branch="main"
git_url="https://github.com/user/repo.git"
jenkins_url="http://jenkins-server/job/my-job/build"
jenkins_user="user"
jenkins_token="token"

# Clone repository and pull latest changes
git clone $git_url repo
cd repo
git checkout $branch
git pull

# Trigger Jenkins pipeline
```

```
curl -X POST $jenkins_url \
   --user "$jenkins_user:$jenkins_token"

echo "CI/CD pipeline triggered for branch $branch."
```

2.2 Kubernetes Namespace Cleaner

bash
Copy code

```
#!/bin/bash
namespace="test-environment"

kubectl get namespaces | grep $namespace && \
kubectl delete namespace $namespace && \
echo "Namespace $namespace deleted."

# Recreate namespace
kubectl create namespace $namespace
echo "Namespace $namespace recreated."
```

2.3 Docker Image Cleanup

bash
Copy code

```
#!/bin/bash
threshold_days=30

docker images --format '{{.Repository}} {{.Tag}} {{.CreatedSince}}' | while read repo tag created; do
   if [[ $created == *"$threshold_days days ago"* ]]; then
      echo "Deleting image: $repo:$tag"
      docker rmi "$repo:$tag"
   fi
done
```

---

3. Complex and Advanced Scripts

3.1 Full Backup and Restore with Verification

Backup Script:

bash
Copy code

```
#!/bin/bash
src_dir="/data"
backup_dir="/backups"
timestamp=$(date +%Y%m%d_%H%M%S)
backup_file="$backup_dir/backup_$timestamp.tar.gz"

tar -czf $backup_file $src_dir && \
md5sum $backup_file > "$backup_file.md5"

echo "Backup completed: $backup_file"
```

Restore Script:

bash
Copy code

```
#!/bin/bash
backup_file="/backups/backup_latest.tar.gz"
restore_dir="/restore"
```

```
# Verify integrity
if md5sum -c "$backup_file.md5"; then
    tar -xzf $backup_file -C $restore_dir
    echo "Restore successful to $restore_dir."
else
    echo "Backup integrity verification failed!"
    exit 1
fi
```

3.2 Automated Incident Monitoring and Reporting

bash
Copy code

```
#!/bin/bash
log_file="/var/log/syslog"
incident_log="incident_report.log"
email="admin@example.com"

tail -Fn0 $log_file | while read line; do
    if echo "$line" | grep -q "ERROR"; then
        echo "$line" >> $incident_log
        echo "Incident reported: $line"
        echo "$line" | mail -s "Incident Alert" $email
    fi
done
```

---

4. Monitoring and Debugging Scripts

4.1 Network Latency Monitor

bash
Copy code

```
#!/bin/bash
hosts=("8.8.8.8" "1.1.1.1" "google.com")

for host in "${hosts[@]}"; do
    if ! ping -c 1 $host &> /dev/null; then
        echo "Host $host is unreachable!"
    else
        latency=$(ping -c 1 $host | grep time= | awk -F 'time=' '{print $2}' | cut -d' ' -f1)
        echo "Host $host latency: $latency ms"
    fi
done
```

4.2 Disk I/O Monitoring

bash
Copy code

```
#!/bin/bash
while true; do
    iostat -dx 1 5 | awk '/sda/ {print "Device: " $1 ", I/O Utilization: " $14 "%"}'
    sleep 10
done
```

---

5. Utility Scripts for Cloud Professionals

5.1 Extracting AWS Cost Reports

bash
Copy code

```
#!/bin/bash
start_date="2023-01-01"
end_date="2023-01-31"
output_file="aws_cost_report.csv"

aws ce get-cost-and-usage \
    --time-period Start=$start_date,End=$end_date \
    --granularity MONTHLY \
    --metrics "UnblendedCost" \
    --output text > $output_file

echo "AWS cost report saved to $output_file."
```

5.2 Automated Cleanup of Expired IAM Keys

bash
Copy code

```
#!/bin/bash
days=90

aws iam list-users | jq -r '.Users[].UserName' | while read user; do
    aws iam list-access-keys --user-name $user | \
    jq -r '.AccessKeyMetadata[] | select(.Status=="Active") | "\(.UserName) \(.AccessKeyId) \(.CreateDate)"' | while read
uname key date; do
        if [ $(($(date +%s) - $(date -d "$date" +%s))) -gt $((days * 86400)) ]; then
            aws iam delete-access-key --user-name $uname --access-key-id $key
            echo "Deleted expired key $key for user $uname."
        fi
    done
done
```

How These Scripts Help in the Real World

1. **Time Savings**: Automate repetitive tasks like backups, monitoring, and cleanup.
2. **Error Reduction**: Predefined scripts reduce human errors in critical operations.
3. **Scalability**: Dynamic scaling and provisioning make it easier to handle changing demands.
4. **Compliance**: Automated cleanup ensures security compliance.
5. **Proactive Monitoring**: Incident detection scripts prevent downtime and enable faster response.