

1.FCFS

```
#!/bin/bash
```

```
echo "INPUT:"  
echo "Enter the number of processes--"  
read n
```

```
declare -a pid  
declare -a bt  
declare -a wt  
declare -a tat
```

```
for ((i = 0; i < n; i++)); do  
    pid[$i]=$(($i))  
    echo "Enter burst time for Process ${pid[$i]}--"  
    read bt[$i]  
done
```

```
wt[0]=0
```

```
for ((i = 1; i < n; i++)); do  
    wt[$i]=$(($wt[$i-1] + bt[$i-1]))  
done
```

```
for ((i = 0; i < n; i++)); do  
    tat[$i]=$(($wt[$i] + bt[$i]))  
done
```

```
echo "OUTPUT:"  
echo -e "\nProcess\tBurst Time\tWaiting Time\tTurnaround Time"  
total_wt=0  
total_tat=0
```

```
for ((i = 0; i < n; i++)); do  
    echo -e "P${pid[$i]}\t${bt[$i]}\t${wt[$i]}\t${tat[$i]}"  
    total_wt=$((total_wt + wt[$i]))  
    total_tat=$((total_tat + tat[$i]))  
done
```

```
avg_wt=$(echo "scale=6; $total_wt / $n" | bc)  
avg_tat=$(echo "scale=6; $total_tat / $n" | bc)
```

```
echo -e "\nAverage Waiting Time-- $avg_wt"
echo "Average Turnaround Time-- $avg_tat"
```

2.sjf(non preemptive)

```
#!/bin/bash
```

```
echo "INPUT"
echo -n "Enter number of processes: "
read n

for (( i=0; i<n; i++ ))
do
    echo -n "Enter burst time for process $((i))-- "
    read bt[$i]
    pid[$i]=$i
done

# Sort burst times using simple bubble sort
for (( i=0; i<n-1; i++ ))
do
    for (( j=0; j<n-i-1; j++ ))
    do
        if [ ${bt[j]} -gt ${bt[j+1]} ]
        then
            # Swap burst time
            temp=${bt[j]}
            bt[j]=${bt[j+1]}
            bt[j+1]=$temp

            # Swap process ID
            temp=${pid[j]}
            pid[j]=${pid[j+1]}
            pid[j+1]=$temp
        fi
    done
done

# Calculate waiting and turnaround times
wt[0]=0
tat[0]=$(bt[0])
total_wt=0
total_tat=0

for (( i=1; i<n; i++ ))
do
    wt[$i]==$((wt[i-1] + bt[i-1]))

```

```

tat[$i]=$(($wt[i] + bt[i]))
done

echo "OUTPUT"
echo -e "\nProcess\tBurst Time\tWaiting Time\tTurnaround Time"
for (( i=0; i<n; i++ ))
do
    echo -e "p${pid[$i]}\t${bt[i]}\t${wt[i]}\t${tat[i]}"
    total_wt=$((total_wt + wt[i]))
    total_tat=$((total_tat + tat[i]))
done

# Average calculations
avg_wt=$(echo "scale=6; $total_wt / $n" | bc)
avg_tat=$(echo "scale=6; $total_tat / $n" | bc)

echo -e "\nAverage Waiting Time-- $avg_wt"
echo -e "Average Turnaround Time-- $avg_tat"

```

3.sjf(preemptive)

```

#!/bin/bash

echo -n "Enter number of processes: "
read n

for (( i=0; i<n; i++ ))
do
    echo -n "Enter arrival time for process $((i+1)): "
    read at[$i]
    echo -n "Enter burst time for process $((i+1)): "
    read bt[$i]
    rem_bt[$i]="${bt[$i]}" # Remaining burst time
    pid[$i]=$i
done

completed=0
time=0
min=9999
shortest=-1
finish_time=0
total_wt=0
total_tat=0

for (( i=0; i<n; i++ ))
do
    wt[$i]=0

```

```

tat[$i]=0
done_flag[$i]=0
done

echo -e "\nGantt Chart:"

while (( completed != n ))
do
    shortest=-1
    min=9999

    for (( i=0; i<n; i++ ))
    do
        if (( at[i] <= time && rem_bt[i] > 0 && rem_bt[i] < min ))
        then
            min=${rem_bt[i]}
            shortest=$i
        fi
    done

    if (( shortest == -1 ))
    then
        ((time++))
        continue
    fi

    echo -n "|P$((shortest+1)) "

    (( rem_bt[shortest]-- ))
    (( time++ ))

    if (( rem_bt[shortest] == 0 ))
    then
        completed=$((completed + 1))
        finish_time=$time
        tat[shortest]=$(( finish_time - at[shortest] ))
        wt[shortest]=$(( tat[shortest] - bt[shortest] ))

        total_wt=$(( total_wt + wt[shortest] ))
        total_tat=$(( total_tat + tat[shortest] ))
    fi
done

echo "|"

echo -e "\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnAroundTime"
for (( i=0; i<n; i++ ))

```

```

do
echo -e "$((i+1))\t${at[i]}\t${bt[i]}\t${wt[i]}\t${tat[i]}"
done

avg_wt=$(echo "scale=2; $total_wt / $n" | bc)
avg_tat=$(echo "scale=2; $total_tat / $n" | bc)

echo -e "\nAverage Waiting Time: $avg_wt"
echo -e "Average Turnaround Time: $avg_tat"

```

4.priority scheduling

```

#!/bin/bash
echo "INPUT"
echo "Enter the number of processes -- "
read n

for (( i=0; i<n; i++ ))
do
    echo -n "Enter the Burst Time & Priority of Process $i --- "
    read bt[$i] pr[$i]
    pid[$i]=$i
done

for (( i=0; i<n-1; i++ ))
do
    for (( j=0; j<n-i-1; j++ ))
    do
        if [ ${pr[j]} -gt ${pr[j+1]} ]
        then
            temp=${pr[j]}
            pr[j]=${pr[j+1]}
            pr[j+1]=$temp

            temp=${bt[j]}
            bt[j]=${bt[j+1]}
            bt[j+1]=$temp

            temp=${pid[j]}
            pid[j]=${pid[j+1]}
            pid[j+1]=$temp
        fi
    done
done

```

```

wt[0]=0
tat[0]=$(bt[0])
total_wt=0
total_tat=0

for (( i=1; i<n; i++ ))
do
    wt[$i]=$(($wt[i-1] + bt[i-1]))
    tat[$i]=$(($wt[i] + bt[i]))
done

# Output
echo -e "\nOUTPUT"
echo -e "PROCESS\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME"

for (( i=0; i<n; i++ ))
do
    echo -e "${pid[i]}\t${pr[i]}\t${bt[i]}\t${wt[i]}\t${tat[i]}"
    total_wt=$((total_wt + wt[i]))
    total_tat=$((total_tat + tat[i]))
done

avg_wt=$(echo "scale=6; $total_wt / $n" | bc)
avg_tat=$(echo "scale=6; $total_tat / $n" | bc)

echo -e "\nAverage Waiting Time is --- $avg_wt"
echo -e "Average Turnaround Time is --- $avg_tat"

```

5.Round Robin

```

#!/bin/bash
echo "INPUT:"
echo "Enter the number of processes-- "
read n

declare -a pid bt rt wt tat completed

# Input burst times
for ((i=0; i<n; i++))
do
    pid[i]=$((i+1))
    echo -n "Enter Burst Time for process $((i+1)) -- "
    read bt[i]
    rt[i]="${bt[i]}"
    wt[i]=0

```

```

tat[i]=0
completed[i]=0
done

# Input time quantum
echo -n "Enter the size of time slice -- "
read tq

# Round Robin Logic
time=0
remain=$n

while (( remain > 0 ))
do
  for ((i=0; i<n; i++))
  do
    if (( rt[i] > 0 ))
    then
      if (( rt[i] > tq ))
      then
        time=$((time + tq))
        rt[i]=$((rt[i] - tq))
      else
        time=$((time + rt[i]))
        wt[i]=$((time - bt[i]))
        rt[i]=0
        tat[i]=$((bt[i] + wt[i]))
        completed[i]=1
        remain=$((remain - 1))
      fi
    fi
  done
done

# Output
echo -e "\nOUTPUT:"
echo -e "PROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME"

total_wt=0
total_tat=0
for ((i=0; i<n; i++))
do
  echo -e "${pid[i]}\t${bt[i]}\t${wt[i]}\t${tat[i]}"
  total_wt=$((total_wt + wt[i]))
  total_tat=$((total_tat + tat[i]))
done

```

```

# Averages with decimals
avg_wt=$(echo "scale=6; $total_wt / $n" | bc)
avg_tat=$(echo "scale=6; $total_tat / $n" | bc)

echo -e "\nThe Average Turnaround time is-- $avg_tat"
echo -e "Average Waiting time is----- $avg_wt"

```

5.Producer consumer

```
#!/bin/bash
```

```
buffer=0
```

```
full=0
```

```

produce() {
    if [ $full -eq 1 ]
    then
        echo "Buffer is Full"
    else
        echo -n "Enter the value: "
        read buffer
        full=1
    fi
}

```

```

consume() {
    if [ $full -eq 0 ]
    then
        echo "Buffer is Empty"
    else
        echo "The consumed value is $buffer"
        buffer=0
        full=0
    fi
}

```

```

while true
do
    echo
    echo "1. Produce  2. Consume  3. Exit"
    echo -n "Enter your choice: "
    read choice

```

```

case $choice in
    1) produce ;;
    2) consume ;;
    3) exit 0 ;;

```

```

*) echo "Invalid choice" ;;
esac
done

6.Dinning Philosoper
#!/bin/bash

echo "DINING PHILOSOPHER PROBLEM"

echo -n "Enter total number of philosophers: "
read n

echo -n "How many are hungry: "
read h

declare -a hungry

for ((i=0; i<h; i++))
do
    echo -n "Enter philosopher position $((i+1)): "
    read hungry[$i]
done

while true
do
    echo
    echo "OUTPUT"
    echo "1. One philosopher can eat at a time"
    echo "2. Two philosophers can eat at a time"
    echo "3. Exit"
    echo -n "Enter your choice: "
    read choice

    if [ $choice -eq 1 ]
    then
        echo
        echo "Allow one philosopher to eat at a time"
        for ((i=0; i<h; i++))
        do
            echo "P ${hungry[$i]} is eating"
            for ((j=0; j<h; j++))
            do
                if [ $i -ne $j ]
                then
                    echo "P ${hungry[$j]} is waiting"
                fi
            done
        done
    fi
done

```

```

done
done

elif [ $choice -eq 2 ]
then
echo
echo "Allow two philosophers to eat at a time"
found=0

for ((i=0; i<h; i++))
do
for ((j=i+1; j<h; j++))
do
diff=$(( hungry[$i] - hungry[$j] ))
abs=${diff#-}

if [ $abs -ne 1 ] && [ $abs -ne $((n-1)) ]
then
echo "P ${hungry[$i]} and P ${hungry[$j]} are eating"
for ((k=0; k<h; k++))
do
if [ $k -ne $i ] && [ $k -ne $j ]
then
echo "P ${hungry[$k]} is waiting"
fi
done
found=1
break
fi
done
[ $found -eq 1 ] && break
done

if [ $found -eq 0 ]
then
echo "No two philosophers can eat together (all are neighbours)"
fi

elif [ $choice -eq 3 ]
then
echo "Exiting..."
exit 0

else
echo "Invalid choice"
fi
done

```

7.MFT

```
#!/bin/bash
```

```
read -p "Enter the total memory available (in Bytes) -- " total_mem  
read -p "Enter the block size (in Bytes) -- " block_size  
read -p "Enter the number of processes -- " num_proc
```

```
declare -a processes  
for ((i=1; i<=num_proc; i++))  
do  
    read -p "Enter memory required for process $i (in Bytes) -- " p  
    processes[$i]=$p  
done
```

```
num_blocks=$((total_mem / block_size))  
echo -e "\nNo. of Blocks available in memory -- $num_blocks\n"
```

```
allocated=0  
internal_frag=0  
external_frag=0
```

```
echo -e "OUTPUT"  
echo -e "PROCESS\tMEMORY REQUIRED\tALLOCATED\tINTERNAL FRAGMENTATION"  
  
for ((i=1; i<=num_proc; i++))  
do  
    p=${processes[$i]}  
    if [ $allocated -lt $num_blocks ]; then  
        if [ $p -le $block_size ]; then  
            allocated=$((allocated + 1))  
            frag=$((block_size - p))  
            internal_frag=$((internal_frag + frag))  
            echo -e "$i\t$p\tYES\t$frag"  
        else  
            echo -e "$i\t$p\tNO\t----"  
        fi  
    else  
        echo -e "$i\t$p\tNO\t----"  
        external_frag=$((external_frag + p))  
    fi  
done
```

```
echo -e "\nMemory is Full, Remaining Processes cannot be accommodated"
echo "Total Internal Fragmentation is $internal_frag"
echo "Total External Fragmentation is $external_frag"
```

9.MVT

```
#!/bin/bash
# MVT (Multiprogramming with Variable Tasks) Algorithm Simulation

echo "Enter total memory available (in Bytes)--"
read ms

total=$ms
allocated=0

declare -a process
declare -a mem

i=0

while [ $ms -gt 0 ]
do
    echo "Enter memory required for process $((i+1)) (in Bytes)--"
    read req

    if [ $req -le $ms ]
    then
        process[$i]=$((i+1))
        mem[$i]=$req
        ms=$((ms - req))
        allocated=$((allocated + req))
        echo "Memory is allocated for Process $((i+1))"
    else
        echo "Memory is Full"
        break
    fi

    echo "Do you want to continue (y/n)--"
    read choice
    if [ "$choice" = "n" ]
    then
        break
    fi
    i=$((i+1))
done

echo
```

```

echo "Total Memory Available = $total"
echo
echo "PROCESS  MEMORY ALLOCATED"
for j in $(seq 0 $i)
do
    echo " ${process[$j]}      ${mem[$j]}"
done

echo
echo "Total Memory Allocated = $allocated"
extfrag=$((total - allocated))
echo "Total External Fragmentation = $extfrag"

```

10.Worst

```
#!/bin/bash
```

```

echo -n "Enter number of blocks: "
read nb
echo -n "Enter number of files: "
read nf

declare -a block file allocation fragment used_block

echo "Enter block sizes:"
for ((i=0;i<nb;i++)); do
    read -p "Block $((i+1)): " block[$i]
done

echo "Enter file sizes:"
for ((i=0;i<nf;i++)); do
    read -p "File $((i+1)): " file[$i]
done

for ((i=0;i<nf;i++)); do
    worst=-1
    for ((j=0;j<nb;j++)); do
        if [ ${block[$j]} -ge ${file[$i]} ]; then
            if [ $worst -eq -1 ] || [ ${block[$j]} -gt ${block[$worst]} ]; then
                worst=$j
            fi
        fi
    done
    if [ $worst -ne -1 ]; then
        allocation[$i]=$worst
        fragment[$i]=${block[$worst]}-${file[$i]}
        used_block[$i]=${block[$worst]}
    fi
done

```

```

block[$worst]=-1
else
    allocation[$i]=-1
fi
done

echo -e "\nFileNo\tFileSize\tBlockNo\tBlockSize\tFragment"
for ((i=0;i<nf;i++)); do
    if [ ${allocation[$i]} -ne -1 ]; then
        echo -e "$((i+1))\t${file[$i]}\t$((allocation[$i]+1))\t${used_block[$i]}\t${fragment[$i]}"
    else
        echo -e "$((i+1))\t${file[$i]}\tNot Allocated"
    fi
done

```

11.Best_fit

```
#!/bin/bash
```

```

echo -n "Enter number of blocks: "
read nb
echo -n "Enter number of files: "
read nf

declare -a block file allocation fragment used_block

echo "Enter block sizes:"
for ((i=0;i<nb;i++)); do
    read -p "Block $((i+1)): " block[$i]
done

echo "Enter file sizes:"
for ((i=0;i<nf;i++)); do
    read -p "File $((i+1)): " file[$i]
done

for ((i=0;i<nf;i++)); do
    best=-1
    for ((j=0;j<nb;j++)); do
        if [ ${block[$j]} -ge ${file[$i]} ]; then
            if [ $best -eq -1 ] || [ ${block[$j]} -lt ${block[$best]} ]; then
                best=$j
            fi
        fi
    done
    if [ $best -ne -1 ]; then
        allocation[$i]=$best
    fi
done

```

```

fragment[$i]=$(($block[$best]-file[$i]))
used_block[$i]=${block[$best]}
block[$best]=-1
else
    allocation[$i]=-1
fi
done

echo -e "\nFileNo\tFileSize\tBlockNo\tBlockSize\tFragment"
for ((i=0;i<nf;i++)); do
    if [ ${allocation[$i]} -ne -1 ]; then
        echo -e "$((i+1))\t${file[$i]}\t$((allocation[$i]+1))\t${used_block[$i]}\t${fragment[$i]}"
    else
        echo -e "$((i+1))\t${file[$i]}\tNot Allocated"
    fi
done

```

11. First Fit

```
#!/bin/bash
```

```

echo -n "Enter number of blocks: "
read nb
echo -n "Enter number of files: "
read nf

declare -a block file allocation fragment used_block

echo "Enter block sizes:"
for ((i=0;i<nb;i++)); do
    read -p "Block $((i+1)): " block[$i]
done

echo "Enter file sizes:"
for ((i=0;i<nf;i++)); do
    read -p "File $((i+1)): " file[$i]
done

for ((i=0;i<nf;i++)); do
    allocation[$i]=-1
    for ((j=0;j<nb;j++)); do
        if [ ${block[$j]} -ge ${file[$i]} ]; then
            allocation[$i]=$j
            fragment[$i]=$(($block[$j]-file[$i]))
            used_block[$i]=${block[$j]}
            block[$j]=-1
            break
        fi
    done
done

```

```

    fi
done
done

echo -e "\nFileNo\tFileSize\tBlockNo\tBlockSize\tFragment"
for ((i=0;i<nf;i++)); do
    if [ ${allocation[$i]} -ne -1 ]; then
        echo -e "$((i+1))\t${file[$i]}\t$((allocation[$i]+1))\t${used_block[$i]}\t${fragment[$i]}"
    else
        echo -e "$((i+1))\t${file[$i]}\tNot Allocated"
    fi
done

```

12.fifo_page replacement

```
#!/bin/bash
```

```
# FIFO Page Replacement
```

```

read -p "Enter number of frames: " f
read -p "Enter number of pages: " n
echo "Enter the reference string (space separated): "
read -a pages

```

```

frame=()
page_faults=0
front=0

```

```

for ((i=0; i<n; i++))
do
    found=0
    for val in "${frame[@]}"; do
        if [ "$val" -eq "${pages[$i]}" ]; then
            found=1
            break
        fi
    done

    if [ $found -eq 0 ]; then
        if [ ${#frame[@]} -lt $f ]; then
            frame+=("${pages[$i]}")
        else
            frame[$front]=${pages[$i]}
            front=$(( (front+1) % f ))
        fi
        page_faults=$((page_faults+1))
    fi
echo "Frame: ${frame[@]}"

```

```
done
```

```
echo "Total Page Faults = $page_faults"
```

13.LRU

```
#!/bin/bash
```

```
# LRU Page Replacement
```

```
read -p "Enter number of frames: " f
read -p "Enter number of pages: " n
echo "Enter the reference string (space separated): "
read -a pages
```

```
frame=()
page_faults=0
declare -A recent
```

```
for ((i=0; i<n; i++))
do
    found=0
    for val in "${frame[@]}"; do
        if [ "$val" -eq "${pages[$i]}" ]; then
            found=1
            break
        fi
    done

    if [ $found -eq 0 ]; then
        if [ ${#frame[@]} -lt $f ]; then
            frame+=("${pages[$i]}")
        else
            lru_index=0
            min=${recent[$frame[0]]}
            for ((k=0; k<${#frame[@]}; k++)); do
                if [ ${recent[$frame[$k]]} -lt $min ]; then
                    min=${recent[$frame[$k]]}
                    lru_index=$k
                fi
            done
            frame[$lru_index]=${pages[$i]}
        fi
        page_faults=$((page_faults+1))
    fi
    recent[${pages[$i]}]=$i
    echo "Frame: ${frame[@]}"
done
```

```
echo "Total Page Faults = $page_faults"
```

14.Optimal Page Repalcement

```
#!/bin/bash
```

```
# Optimal Page Replacement
```

```
read -p "Enter number of frames: " f
read -p "Enter number of pages: " n
echo "Enter the reference string (space separated): "
read -a pages

frame=()
page_faults=0

for ((i=0; i<n; i++))
do
    found=0
    for val in "${frame[@]}"; do
        if [ "$val" -eq "${pages[$i]}" ]; then
            found=1
            break
        fi
    done

    if [ $found -eq 0 ]; then
        if [ ${#frame[@]} -lt $f ]; then
            frame+=("${pages[$i]}")
        else
            farthest=-1
            replace_index=0
            for ((k=0; k<${#frame[@]}; k++)); do
                next=-1
                for ((j=i+1; j<n; j++)); do
                    if [ ${frame[$k]} -eq ${pages[$j]} ]; then
                        next=$j
                        break
                    fi
                done
                if [ $next -eq -1 ]; then
                    replace_index=$k
                    break
                elif [ $next -gt $farthest ]; then
                    farthest=$next
                    replace_index=$k
                fi
            done
        fi
    fi
done
```

```

done
frame[$replace_index]="${pages[$i]}"
fi
page_faults=$((page_faults+1))
fi
echo "Frame: ${frame[@]}"
done

echo "Total Page Faults = $page_faults"

```

14.Bankers

```
#!/bin/bash
```

```
echo "BANKER'S ALGORITHM "
```

```
read -p "Enter number of processes: " n
read -p "Enter number of resources: " m
```

```
declare -a avail finish safe
declare -A alloc max need
```

```
echo "Enter available resources (space separated):"
read -a avail
```

```
echo "Enter allocation matrix (row-wise):"
```

```
for ((i=0; i<n; i++)); do
```

```
    echo -n "P$i: "
```

```
    read -a row
```

```
    for ((j=0; j<m; j++)); do
```

```
        alloc[$i,$j]="${row[$j]}"
    done
done
```

```
done
```

```
echo "Enter max matrix (row-wise):"
```

```
for ((i=0; i<n; i++)); do
```

```
    echo -n "P$i: "
```

```
    read -a row
```

```
    for ((j=0; j<m; j++)); do
```

```
        max[$i,$j]="${row[$j]}"
    done
done
```

```
done
```

```
# Need calculation
```

```
for ((i=0; i<n; i++)); do
```

```
    for ((j=0; j<m; j++)); do
```

```
        need[$i,$j]="$(( max[$i,$j] - alloc[$i,$j] ))"
    done
done
```

```

done
done

# Print table ONCE
echo
echo "Process | Allocation | Max      | Need"
echo "-----"
for ((i=0; i<n; i++)); do
    printf "P%-3d | $i
    for ((j=0; j<m; j++)); do printf "%-2d ${alloc[$i,$j]}"; done
    printf " | "
    for ((j=0; j<m; j++)); do printf "%-2d ${max[$i,$j]}"; done
    printf " | "
    for ((j=0; j<m; j++)); do printf "%-2d ${need[$i,$j]}"; done
    echo
done
echo "-----"

# Print Available separately (dynamic)
echo -n "Available: "
for ((j=0; j<m; j++)); do echo -n "${avail[$j]} "; done
echo

# Banker logic
for ((i=0; i<n; i++)); do finish[$i]=0; done
count=0

while [ $count -lt $n ]; do
    found=0
    for ((i=0; i<n; i++)); do
        if [ ${finish[$i]} -eq 0 ]; then
            can_run=1
            for ((j=0; j<m; j++)); do
                if [ ${need[$i,$j]} -gt ${avail[$j]} ]; then
                    can_run=0
                    break
                fi
            done

            if [ $can_run -eq 1 ]; then
                echo
                echo ">>> P$i executed"

                for ((j=0; j<m; j++)); do
                    avail[$j]=$( avail[$j] + alloc[$i,$j] )
                done
            fi
        fi
    done
done

```

```

        finish[$i]=1
        safe[$count]=$i
        count=$((count+1))
        found=1

        echo -n "Available: "
        for ((j=0; j<m; j++)); do echo -n "${avail[$j]} "; done
        echo
    fi
fi
done
[ $found -eq 0 ] && break
done

echo
if [ $count -eq $n ]; then
    echo "System is in SAFE STATE"
    echo -n "Safe Sequence: "
    for ((i=0; i<n; i++)); do echo -n "P${safe[$i]} "; done
    echo
else
    echo "System is NOT in Safe State(Deadlock Possible)."
fi

```

15.Single_Level

```

#!/bin/bash
# Single Level Directory Simulation

declare -A directory

while true
do
    echo -e "\n1. Create File\n2. Delete File\n3. Search File\n4. Display Files\n5. Exit"
    read -p "Enter choice: " ch
    case $ch in
        1)
            read -p "Enter file name: " name
            if [ "${directory[$name]}" ]; then
                echo "File already exists!"
            else
                directory[$name]=1
                echo "File '$name' created."
            fi
            ;;
        2)
            read -p "Enter file name to delete: " name

```

```

if [ "${directory[$name]}" ]; then
    unset directory[$name]
    echo "File '$name' deleted."
else
    echo "File not found."
fi
;;
3)
read -p "Enter file name to search: " name
if [ "${directory[$name]}" ]; then
    echo "File '$name' found."
else
    echo "File not found."
fi
;;
4)
echo "Files: ${!directory[@]}"
;;
5)
exit ;;
*)
echo "Invalid choice" ;;
esac
done

```

16.Two level

```

#!/bin/bash
# Two Level Directory Simulation

declare -A users

while true
do
    echo -e "\n1. Create User\n2. Create File\n3. Delete File\n4. Search File\n5. Display User Files\n6. Exit"
    read -p "Enter choice: " ch

    case $ch in
        1)
            read -p "Enter username: " user
            if [ "${users[$user]}" ]; then
                echo "User already exists!"
            else
                users[$user]=""
                echo "User '$user' created."
            fi
        ;;
        2)
            read -p "Enter file name: " file
            if [ "${directory[$file]}" ]; then
                echo "File '$file' already exists."
            else
                directory[$file]=${!user}
                echo "File '$file' created." > ${!user}
            fi
        ;;
        3)
            read -p "Enter file name: " file
            if [ "${directory[$file]}" ]; then
                echo "File '$file' deleted."
                unset directory[$file]
            else
                echo "File '$file' not found."
            fi
        ;;
        4)
            read -p "Enter file name to search: " name
            if [ "${directory[$name]}" ]; then
                echo "File '$name' found."
            else
                echo "File not found."
            fi
        ;;
        5)
            echo "Files: ${!directory[@]}"
        ;;
        6)
            exit ;;
        *)
            echo "Invalid choice" ;;
    esac
done

```

```

;;
2)
read -p "Enter username: " user
if [ "${users[$user]}"; then
  read -p "Enter filename: " file
  users[$user]="${users[$user]} $file"
  echo "File '$file' created under user '$user'."
else
  echo "User not found!"
fi
;;
3)
read -p "Enter username: " user
read -p "Enter filename to delete: " file
files=${users[$user]}
users[$user]=$(echo $files | sed "s/\b$file\b//g")
echo "File '$file' deleted (if existed)."
;;
4)
read -p "Enter username: " user
read -p "Enter filename to search: " file
if echo "${users[$user]}" | grep -wq "$file"; then
  echo "File '$file' found under user '$user'."
else
  echo "File not found."
fi
;;
5)
for user in "${!users[@]}"; do
  echo "User: $user → Files: ${users[$user]}"
done
;;
6)
exit ;;
*)
  echo "Invalid choice" ;;
esac
done

```