OS SHIT

SHELL COMMANDS

```
cut -d: -f1 /etc/passwd
#The -d option specifies the delimiter to use, which is colon in this case,
#and the -f option specifies the field(s) to extract, which is the first field (username)
#in this case.
# The output will be a list of all the user IDs (UIDs) on the system.
```

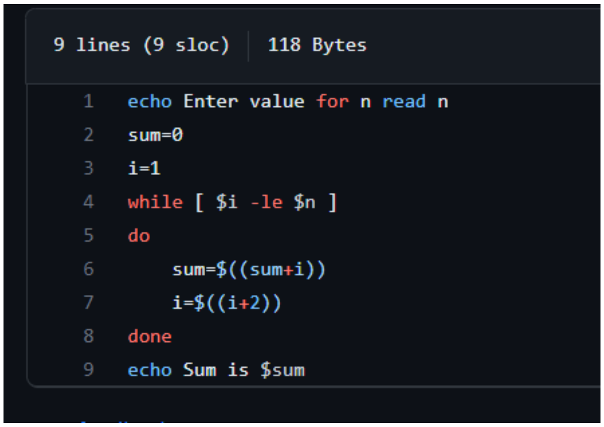
```
grep "^SRM" f1
# a command to display the lines of the file f1 starts with SRM
```

```
#"count print word character and lines in a file"
cat abc.txt | wc
cat abc.txt | wc -c #char
cat abc.txt | wc -l #lines
cat abc.txt | wc -w #words
```

```
sort -r /etc/passwd
# You can use the "sort" command to sort the "/etc/passwd" file in descending order based on the usernames
```

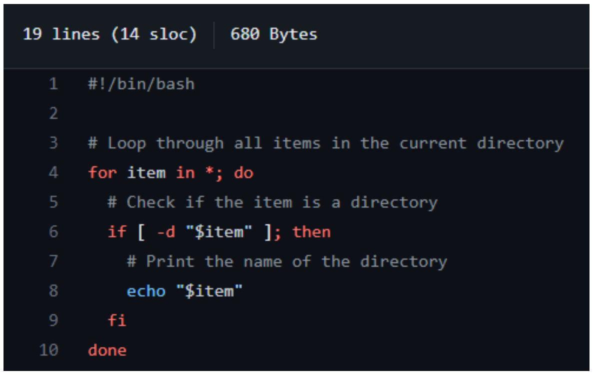
```
cut -c 5-8 f1
# write a command to cut 5 to 8 characters of the file f1
```

```
comm -12 f1 f2
# In this command, the "-12" option is used to suppress the output of lines that are
#unique to the first file ("f1") and lines that are unique to the second file ("f2").
#This leaves only the lines that are common to both files. When you execute this command,
```



9 lines (9 sloc) | 118 Bytes

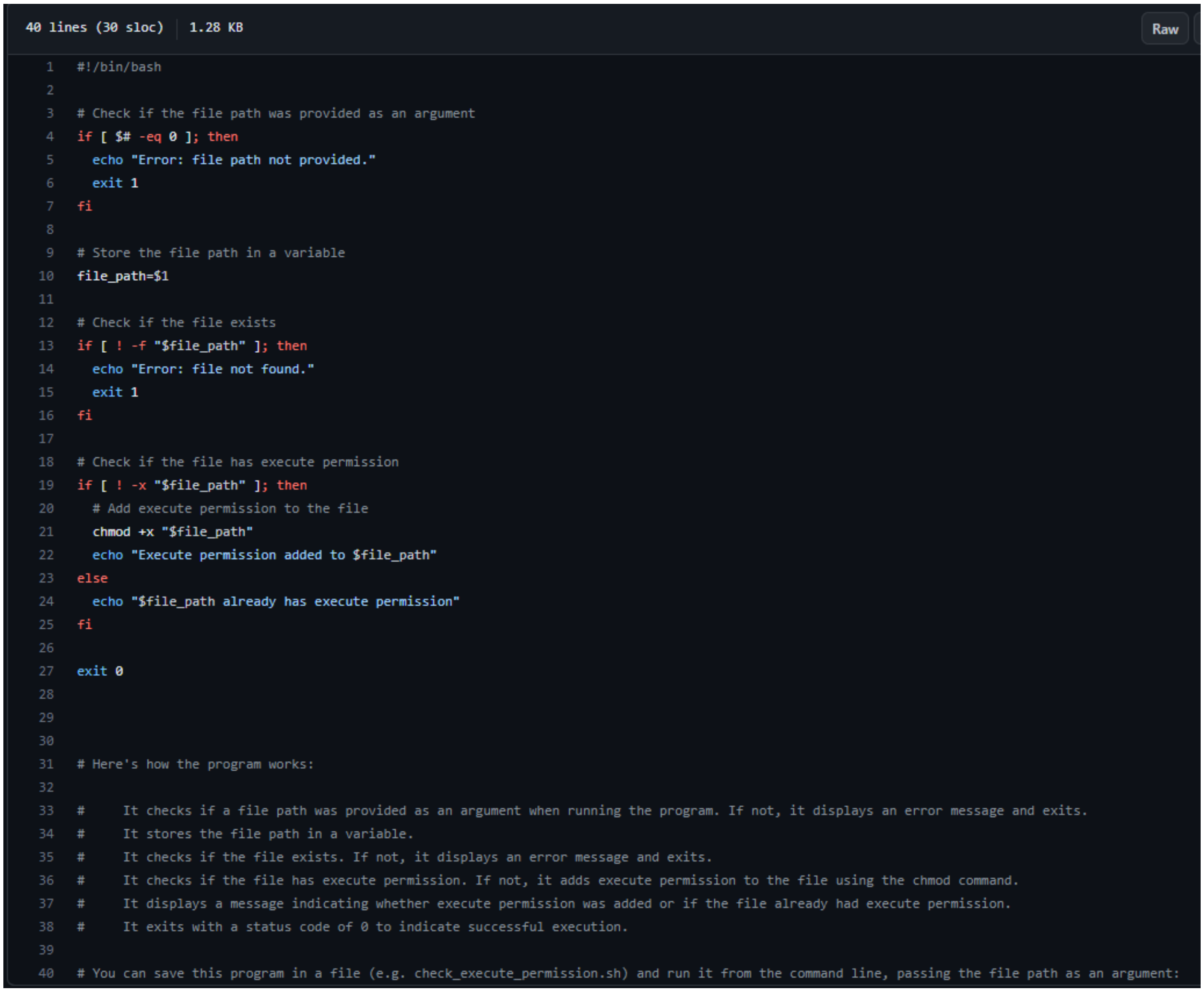
```
echo Enter value for n read n
sum=0
i=1
while [ $i -le $n ]
do
      sum=$((sum+i))
      i=$((i+2))
done
echo Sum is $sum
```



19 lines (14 sloc) | 680 Bytes

```
#!/bin/bash

# Loop through all items in the current directory
for item in *; do
  # Check if the item is a directory
  if [ -d "$item" ]; then
    # Print the name of the directory
    echo "$item"
  fi
done
```



40 lines (30 sloc) | 1.28 KB

```
#!/bin/bash

# Check if the file path was provided as an argument
if [ $# -eq 0 ]; then
  echo "Error: file path not provided."
  exit 1
fi

# Store the file path in a variable
file_path=$1

# Check if the file exists
if [ ! -f "$file_path" ]; then
  echo "Error: file not found."
  exit 1
fi

# Check if the file has execute permission
if [ ! -x "$file_path" ]; then
  # Add execute permission to the file
  chmod +x "$file_path"
  echo "Execute permission added to $file_path"
else
  echo "$file_path already has execute permission"
fi

exit 0



# Here's how the program works:

#     It checks if a file path was provided as an argument when running the program. If not, it displays an error message and exits.
#     It stores the file path in a variable.
#     It checks if the file exists. If not, it displays an error message and exits.
#     It checks if the file has execute permission. If not, it adds execute permission to the file using the chmod command.
#     It displays a message indicating whether execute permission was added or if the file already had execute permission.
#     It exits with a status code of 0 to indicate successful execution.

# You can save this program in a file (e.g. check_execute_permission.sh) and run it from the command line, passing the file path as an argument:
```

Process scheduling fcfs rr

FCFS

```
#include<stdio.h>

int main()
{
    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
    printf("Enter total number of processes(maximum 20):");
    scanf("%d",&n);

    printf("\nEnter Process Burst Time\n");
    for(i=0;i<n;i++)
    {
        printf("P[%d]:",i+1);
        scanf("%d",&bt[i]);
    }

    wt[0]=0;    //waiting time for first process will be zero

    //calculating waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
    }

    printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

    //calculating turnaround time
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        avwt+=wt[i];
        avtat+=tat[i];
        printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
    }

    avwt/=i;
    avtat/=i;
    printf("\n\nAverage Waiting Time:%d",avwt);
    printf("\nAverage Turnaround Time:%d",avtat);

    return 0;
}
```

ROUND ROBBIN

```
#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[])
{
    wt[0] = 0;

    for (int i = 1; i < n; i++)
    {
        wt[i] = bt[i-1] + wt[i-1];
    }
}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])
{
    // bt[i] + wt[i]
    for (int i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i];
    }
}

void findavgTime(int processes[], int n, int bt[])
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    findWaitingTime(processes, n, bt, wt);

    findTurnAroundTime(processes, n, bt, wt, tat);

    printf("Processes   Burst time   Waiting time   Turn around time\n");

    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        printf("   %d ",(i+1));
        printf("       %d ", bt[i]);
        printf("       %d",wt[i]);
        printf("       %d\n",tat[i]);
    }

    float s=(float)total_wt / (float)n;
    float t=(float)total_tat / (float)n;

    printf("Average waiting time = %f",s);
    printf("\n");
    printf("Average turn around time = %f ",t);
}

int main()
{
    int processes[] = { 1, 2, 3};
    int n = sizeof processes / sizeof processes[0];

    int burst_time[] = {10, 5, 8};

    findavgTime(processes, n, burst_time);
    return 0;
}
```

SHARED MEMORY

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/shm.h>
int main()
{
```

```
int shmid;
char *str;
shmid=shmget((key_t)6, 1024, IPC_CREAT |0666);
str=(char *)shmat(shmid, (char *)0,0);
printf("Enter data :");
fgets(str ,sizeof(str),stdin);
printf("Data successfully written into memory \n");
shmdt(str);
return 0;
}

// gcc ex9a.c
// ./a.out   srmist
```

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int main()
{
    int shmid;
    char *str;
    shmid = shmget((key_t)6, 1024, IPC_CREAT | 0666);
    str = (char *)shmat(shmid, (char *)0, 0);
    printf("Data read from memory : %s \n", str);
    shmdt(str);
    shmctl(shmid, IPC_RMID, NULL);
    return 0;
}

// gcc ex9b.c
// ./a.out
```

PROCESS CREATION

w4q1

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    int a = 5, b = 10, pid;
    printf("Before fork a=%d b=%d \n", a, b);
    pid = fork();
    if (pid == 0)
    {
        a = a + 1;
        b = b + 1;
        printf("In child a=%d b=%d \n", a, b);
    }
    else
    {
        sleep(1);
        a = a - 1;
        b = b - 1;
        printf("In Parent a=%d b=%d \n", a, b);
    }
    return 0;
}
```

w4q2

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    int a = 5, b = 10, pid;
    printf("Before fork a=%d b=%d \n", a, b);
    pid = vfork();
    if (pid == 0)
    {
        a = a + 1;
        b = b + 1;
        printf("In child a=%d b=%d \n", a, b);
    }
    else
    {
        sleep(1);
        a = a - 1;
        b = b - 1;
        printf("In Parent a=%d b=%d \n", a, b);
    }
    return 0;
}
```

w4q3

```
#include <stdio.h>
#include<unistd.h>
int main()
{
fork();
fork();
fork();
printf("SRMIST\n");
return 0;
}
```

w4q4 odd even using threads

```
#include <stdio.h>
#include<unistd.h>
int main()
{
int pid,n,i,oddsum=0,evensum=0;
printf("Enter the value of n : ");
scanf("%d",&n);
pid=fork();
if(pid == 0){
  for(i=0;i<=n;i++){
    if(i%2==0){evensum+=i;}
    else{oddsum+=i;}
    }
  }
printf("in parent");
printf("%d",evensum);
printf("%d",oddsum);
return 0;
}
```

w4q5 get print pid of child and parent

```
#include<stdio.h>
#include<unistd.h>
int main(){
int pid = fork();
if(pid==0){
  printf(" in child %d %d",getpid(),getppid());
}
else{
  sleep(1);
  printf(" in parent %d %d",getpid(),getppid());
}
return 0;
}
```

PIPES AND IPC

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
  int pipes[2];
  char buff[25];

  pipe(pipes);

  if (fork() == 0) {
    printf("Child: Writing to pipe\n");
    write(pipes[1], "Hello World!", 13);
    printf("Child Exiting\n");
  } else {
    wait(NULL);
    printf("Parent: Reading from pipe\n");
    read(pipes[0], buff, 13);
    printf("Pipe content is: %s\n", buff);
  }

  return 0;
}
```

FIFO OR NAMED PIPE

```
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<stdio.h>
int main()
{
char buff[25];
int rfd,wfd;
mkfifo("fif1",O_CREAT|0644);
if (fork()==0)
{
printf("Child writing into FIFO\n");
wfd=open("fif1",O_WRONLY);
write(wfd,"Hello",6);
}
else
{
rfd=open("fif1",O_RDONLY);
read(rfd,buff,6);
printf("Parent reads from FIFO : %s\n",buff);
}
return 0;
}
```

Message queue writer and reader

```
#include<stdio.h> //writer to message queue
#include<string.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/msg.h>
int main(int argc,char *argv[]){
    int len,mid,i=1;
    struct buffer {
        long mtype;
        char buf[50];
    } x;
    mid=msgget((key_t)6,IPC_CREAT|0666);
    x.mtype=atol(argv[1]);  //message type number
    strcpy(x.buf,argv[2]); // message text
    len=strlen(x.buf);
    msgsnd(mid,&x,len,0);
    printf("message of size %d sent successfully \n",len);
    return 0;
}

// $ ipcs -q
// gedit ex8a.c
// gcc ex8a.c
// /.a.out  1  welcome
// /.a.out  2  hello
// /.a.out  3  srmist
// $ ipcs -q
```

```
#include <stdio.h> // reader to message queue
#include<string.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/msg.h>
int main(int arg, char *argv[])
{
    int len, mid, i = 1;
    struct buffer
    {
        long mtype;
        char buf[50];
    } x;
    mid = msgget((key_t)6, 0666);
    x.mtype = atoi(argv[1]); //message type number
    len = atoi(argv[2]); //length of the message
    msgrcv(mid, &x, len,x.mtype, 0);
    printf("The message is : %s\n",x.buf);
    return 0;
}

// gedit ex8b.c
// gcc ex8b.c
// a.out  2  10
// $ ipcs -q
```

OVERLAY

```
// First create f1 with some contents (cat>f1)
// Hi hello ashok
// How r u
// Good morning

#include <stdio.h>
#include<unistd.h>
int main()
{
printf("Transfer to execlp function \n");
execlp("head", "head","-2","f1",NULL);
printf("This line will not execute \n");
return 0;
}

// output

// Transfer to execlp function
// Hi hello ashok
// How r u
```

SYSTEM V SEMAPHORE MUTUAL EXCLUSION

```
// Execute and write the output of the following program for mutual exclusion using
// system V semaphore
#include<sys/ipc.h>
#include<sys/sem.h>
#include<stdio.h>
#include <unistd.h>
int main()
{
int pid,semid,val;
struct sembuf sop;
semid=semget((key_t)6,1,IPC_CREAT|0666);
pid=fork();
sop.sem_num=0;
sop.sem_op=0;
sop.sem_flg=0;
if (pid!=0)
{
sleep(1);
printf("The Parent waits for WAIT signal\n");
semop(semid,&sop,1);
printf("The Parent WAKED UP & doing her job\n");
sleep(10);
printf("Parent Over\n");
}
else
{
printf("The Child sets WAIT signal & doing her job\n");
semctl(semid,0,SETVAL,1);
sleep(10);
printf("The Child sets WAKE signal & finished her job\n");
semctl(semid,0,SETVAL,0);
printf("Child Over\n");
}
return 0;
}
```

POSIX SEMAPHORE MUTUAL EXCLUSION

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex;
void* thread(void *arg)
{
    // wait
    sem_wait(&mutex);
    printf("\nEntered..\n");
    // critical section
    sleep(4);
    // signal
    printf("\nJust Exiting...\n");
    sem_post(&mutex);
}
int main()
{
    sem_init(&mutex, 0, 1);
    pthread_t t1, t2;
    pthread_create(&t1, NULL, thread, NULL);
    sleep(2);
    pthread_create(&t2, NULL, thread, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    sem_destroy(&mutex);
    return 0;
}
```

Dining philosopher

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };

sem_t mutex;
sem_t S[N];

void test(int phnum)
{
  if (state[phnum] == HUNGRY
    && state[LEFT] != EATING
    && state[RIGHT] != EATING) {
    // state that eating
    state[phnum] = EATING;

    sleep(2);

    printf("Philosopher %d takes fork %d and %d\n",
          phnum + 1, LEFT + 1, phnum + 1);

    printf("Philosopher %d is Eating\n", phnum + 1);

    // sem_post(&S[phnum]) has no effect
    // during takefork
    // used to wake up hungry philosophers
    // during putfork
    sem_post(&S[phnum]);
  }
}

// take up chopsticks
void take_fork(int phnum)
{
  sem_wait(&mutex);

  // state that hungry
  state[phnum] = HUNGRY;

  printf("Philosopher %d is Hungry\n", phnum + 1);

  // eat if neighbours are not eating
  test(phnum);

  sem_post(&mutex);

  // if unable to eat wait to be signalled
  sem_wait(&S[phnum]);

  sleep(1);
}

// put down chopsticks
void put_fork(int phnum)
{

  sem_wait(&mutex);

  // state that thinking
  state[phnum] = THINKING;

  printf("Philosopher %d putting fork %d and %d down\n",
    phnum + 1, LEFT + 1, phnum + 1);
  printf("Philosopher %d is thinking\n", phnum + 1);

  test(LEFT);
  test(RIGHT);

  sem_post(&mutex);
}

void* philosopher(void* num)
{

  while (1) {

    int* i = num;

    sleep(1);

    take_fork(*i);

    sleep(0);

    put_fork(*i);
  }
}

int main()
{

  int i;
  pthread_t thread_id[N];

  // initialize the semaphores
  sem_init(&mutex, 0, 1);

  for (i = 0; i < N; i++)

    sem_init(&S[i], 0, 0);

  for (i = 0; i < N; i++) {

    // create philosopher processes
    pthread_create(&thread_id[i], NULL,
          philosopher, &phil[i]);

    printf("Philosopher %d is thinking\n", i + 1);
  }

  for (i = 0; i < N; i++)

    pthread_join(thread_id[i], NULL);
}
```

READER WRITER PROBLEM

```
#include<semaphore.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
sem_t x,y;
pthread_t tid;
pthread_t writerthreads[100],readerthreads[100];
int readercount = 0;

void *reader(void* param)
{
    sem_wait(&x);
    readercount++;
    if(readercount==1)
        sem_wait(&y);
    sem_post(&x);
    printf("%d reader is inside\n",readercount);
    usleep(3);
    sem_wait(&x);
    readercount--;
    if(readercount==0)
    {
        sem_post(&y);
    }
    sem_post(&x);
    printf("%d Reader is leaving\n",readercount+1);
    return NULL;
}

void *writer(void* param)
{
    printf("Writer is trying to enter\n");
    sem_wait(&y);
    printf("Writer has entered\n");
    sem_post(&y);
    printf("Writer is leaving\n");
    return NULL;
}

int main()
{
    int n2,i;
    printf("Enter the number of readers:");
    scanf("%d",&n2);
    printf("\n");
    int n1[n2];
    sem_init(&x,0,1);
    sem_init(&y,0,1);
    for(i=0;i<n2;i++)
    {
        pthread_create(&writerthreads[i],NULL,reader,NULL);
        pthread_create(&readerthreads[i],NULL,writer,NULL);
    }
    for(i=0;i<n2;i++)
    {
        pthread_join(writerthreads[i],NULL);
        pthread_join(readerthreads[i],NULL);
    }

}


// gcc xyz.c -o xyz -lpthread
```