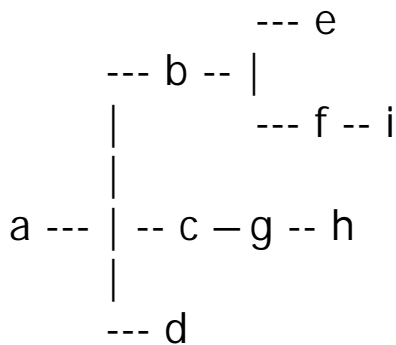


Esercitazioni di Sistemi Operativi

I modulo – 2000/2001

Esercitazione 1: Manipolazione di File e comandi di Shell

Esercizio1: costruire il seguente albero dei direttori:



modificare i diritti di accesso ai direttori in modo che:

1. chiunque possa leggere e scrivere in a;
2. nessuno possa creare sottodirettori di d;
3. *others* non può vedere il contenuto di c, ma può creare fratelli di g;
4. *others* può entrare in g ma non in h;
5. *group* non può vedere g (visualizzare il contenuto di g), ma può entrare in h;
6. *others* non ha nessun diritto su b, *group* li ha tutti;
7. *owner* può leggere i, ma non ha altri diritti su i.

Creare un file xyz in a (usare il comando touch) e sistemare i diritti in modo che solo l'utente possa leggere e scrivere il file. Cosa succede se un altro utente prova ad accedervi? E cosa succede se questo utente prova a rimuovere il file?

Esercizio 2 – Umask

Verificare con quali permessi vengono, di norma, creati i file e le directory. Quindi, tramite il comando umask, provate a fare in modo che questi vengano creati in modo che siano leggibili e scrivibili da tutti. Provate, inoltre, a fare in modo che i file siano leggibili e scrivibili solo dall'utente. Cosa succede per le directory?

Esercitazioni di Sistemi Operativi

I modulo – 2000/2001

Esercitazione 4: Script di shell.

Esercizio 1: Scrivere uno script in grado di leggere due parametri a linea di comandi. I due parametri rappresentano i nomi di due directory. Copiare tutti i file dalla prima directory nella seconda, modificando il contenuto di ogni file in maniera da sostituire tutte le occorrenze della stringa "DAY" con la stringa "GIORNO".

Esercizio 2: Verificare che la variabile PATH contenga la directory /usr/local/bin. Stampare un opportuno messaggio.

Esercizio 3: Leggere da stdin una serie di stringhe fino a quando la stringa letta è diversa da "FINE". Copiare ogni riga su stdout, preceduta da un numero progressivo.

Esercizio 4: Dati due numeri come parametri a linea di comando, visualizzare un rettangolo avente dimensioni pari ai valori letti.

CreaRettangolo 5 10

```
→ +-----+
   |         |
   |         |
   |         |
   |         |
   +-----+
```

Esercizio 5: Siano dati due parametri a linea di comando. Il primo è il nome di una directory, il secondo è la dimensione in byte di un file. Visualizzare tutti i file ordinari nella directory specificata che abbiano permesso di lettura e dimensione superiore a quella specificata. Verificare che i parametri passati siano corretti.

Esercitazioni di Sistemi Operativi

I modulo – 2000/2001

Esercitazione 3: Tools e comandi avanzati.

Esercizio 1: Mostrare una lista di tutti i file chiamati “core” in tutto il File System. Non visualizzare i messaggi di errore.

Esercizio 2: Visualizzare una lista di tutti i processi dell'utente root.
Il formato della lista deve essere:

PID	%CPU	Comando

La lista deve essere visualizzata con l'intestazione come nella tabella precedente.

Esercizio 3: Creare 15 file vuoti denominati [xxx1, xxx2, xxx3 , , xxx15]. Mostrare la lista dei nomi dei file precedentemente creati in ordine ascendente e discendente. Salvare la lista in un file chiamato “lista.txt”

Esercizio 4: Trovare tutte le directory della vostra home il cui nome termina con la stringa “backup” e spostarle in un'unica directory di backup. (Se non avete directory che terminano con backup createne un po').

Esercizio 5: Trovare tutti i file presenti nel sistema che:
hanno dimensione maggiore di 2Mbyte oppure hanno il nome che inizia con la lettera “a” e termina con la lettera “o”.

Esercizio 6: Utilizzando il comando ypcat passwd per visualizzare il file delle password trovare tutti gli utenti con matricola 80XXX. Fare in modo che i vari campi vengano visualizzati incolonnati con caratteri di tabulazione.
Ordinare la lista ottenuta per UID decrescenti.

Esercizio 7: Scrivere un file di bookmark inventando un proprio formato in maniera che sia possibile inserire due livelli gerarchici di categorie. Scrivere uno script awk in grado di generare una pagina HTML partendo dal bookmark file.

Esercizio 8: Scrivere uno script awk in grado di indentare un file C.

Esercitazioni di Sistemi Operativi

I modulo – 2000/2001

Esercitazione 2: L'editor VI e comandi per il confronto di file.

Esercizio 1: Scrivere un file chiamato prova.txt utilizzando VI contenente almeno 30 righe tra testo e numeri.

Esercizio 2: Identificare i comandi di VI che siano in grado di eseguire ognuna delle seguenti azioni sul file prova.txt:

1. Trovare il numero della linea in cui è posizionato il cursore all'interno di un file.
2. Cancellare la linea corrente e le tre righe seguenti.
3. Cercare all'indietro dalla posizione corrente del cursore la più vicina occorrenza della stringa 'xx' (una qualsiasi stringa del file scelta da voi).
4. Cancellare le ultime dieci linee del file.
5. Annullare l'ultima operazione eseguita.
6. Cercare in avanti dalla linea corrente il primo numero contenente tre o più cifre.
7. Cancellare tutti i caratteri della linea corrente tra la parola xxx e la parola yyy incluse (dove xxx e yyy sono due parole scelte da voi in una linea del file).
8. Rimpiazzare tutte le occorrenze del carattere 'a' con il carattere '@'.

Esercizio 3: Salvare il file prova.txt con il nome prova1.txt. Rimpiazzare in prova1.txt tutte le occorrenze di 'i' nelle ultime 10 righe con '!'. Utilizzare il comando diff per visualizzare le differenze dei due file.

Esercizio 4: Applicare i comandi comm, cmp e diff ai file prova.txt e prova1.txt. Quali sono le differenze tra i tre comandi ?

Esercizio 5: Provare a usare i comandi tail ad head per visualizzare solo le prime nove o le ultime nove righe del file prova.txt.

Esercizio 3 – Ricerca File

- Trovare tutti i file core presenti nel sistema;
- Trovare tutti i file il cui nome contiene la stringa conf e visualizzarne le dimensioni;
- Trovare tutti i file dell'utente bin, contenuti nella directory /usr (e nelle sottodirectory), i cui permessi siano -r-xr-xr-x;
- Trovare tutti i vostri file cui avete acceduto da meno di una settimana

Esercitazioni di Sistemi Operativi

Il modulo – 2000/2001

Esercitazione 5: Gestione File e Processi.

Esercizio 1: Scrivere una versione semplificata del comando ls che riceva come parametro il nome di una directory e visualizzi i nomi di tutti i files in essa contenuti.

Esercizio 2: Scrivere un programma che copi il contenuto di un file passato come parametro in un secondo file il cui nome e' anche passato come parametro.

Esercizio 3: Scrivere un programma in grado di memorizzare su disco i dati anagrafici di un elenco di persone nel seguente formato:

Nome – Cognome – Indirizzo – Cap – Telefono.

Il file deve essere organizzato sotto forma di una lista ordinata in ordine alfabetico.

Il programma deve permettere la stampa in ordine alfabetico di tutti i nominativi e l'inserimento di un nuovo nome.

Esercizio 4: Scrivere un semplice programma che faccia una fork e che stampi dei messaggi sia dal processo padre che dal processo figlio.

Esercizio 5: Fare in modo nel programma precedente che il processo padre inizi a stampare i messaggi solo quando il processo figlio ha terminato la sua esecuzione.

Esercizio 6: Scrivere un programma in grado di eseguire i seguenti comandi Linux:

```
cp /etc/fstab .  
sort fstab -o myfstab  
cat myfstab
```

Esercitazioni di Sistemi Operativi

Il modulo – 2000/2001

Esercitazione 6: Gestione File e Processi.

Esercizio 1: Scrivere una funzione chiamata `mydup2()` che fornisca le stesse funzionalità della system call `dup2()` ma sia scritta usando solo la `dup()`.

Esercizio 2: Scrivere un programma che crei una pipe ed effettui una `fork()` in modo che due processi indipendenti siano in esecuzione. I due processi devono poi usare la pipe per passare il contenuto di un grosso file da un processo all'altro.

Esercizio 3: Realizzare il programma precedente facendo sì che i due processi che scambiano dati non abbiano un processo padre in comune ma siano due programmi eseguibili distinti.

Esercizio 4: Scrivere un programma in grado eseguire il seguente comando di shell: `ls | wc -c`.

Esercizio 5: Scrivere un semplice programma di tipo Client/Server per fornire al client alcune informazioni circa il sistema sul quale il server è in esecuzione (Il numero di utenti correntemente loggiati, o il numero di user ID nel file delle password).

Esercizio 6: Scrivere un semplice chat server che permetta a più client di connettersi e mandare messaggi di testo al server. Il server quando riceve un messaggio dal client lo invia in broadcast a tutti i client correntemente connessi. Gli utenti devono utilizzare il telnet come programma client.

Esercitazioni di Sistemi Operativi

Il modulo – 2000/2001

Esercitazione 7: Semafori.

Esercizio 1: Si implementi in linguaggio C un programma che realizzi un processo che richieda in input il nome di un file nel quale siano contenuti 100 interi compresi tra 0 e 4 generati in modo casuale fuori linea.

Il processo deve generare 5 figli, poi entrare in un loop in cui legge da file il prossimo intero e segnalare al figlio corrispondente a quel numero.

Il figlio deve attendere la segnalazione, aspettare un numero random di secondi (compreso tra 0 e 5) e poi comunicare al padre attraverso una pipe il suo pid e la data attuale. Il padre produrrà queste informazioni sullo standard output.

Il processo padre deve restare sospeso nel caso in cui il prossimo processo a cui deve segnalare non abbia ancora risposto alla precedente segnalazione.

Esercizio 2: Si implementi in linguaggio C un programma che realizzi 10 processi "treno".

Ogni treno t percorre una sua linea ferroviaria $LE(t)$ ed entra in una stazione S con un solo binario.

Dalla stazione escono le linee $LU(1)$, $LU(2)$, $LU(3)$ ed $LU(4)$.

Un treno non può entrare in stazione provenendo dalla linea $LE(k)$ se la stazione è occupata oppure se la linea di uscita che il treno richiede per proseguire il suo viaggio non è libera.

Un treno arriva alla stazione ad intervalli di tempo casuali non inferiori a K secondi, decide di proseguire su una linea LU scelta a caso, e la occupa per un numero casuale di secondi O .

Quando la stazione viene liberata da un treno e più treni sono in attesa di entrare viene fatto procedere quello che arriva dalla linea di entrata con indice più basso.

Esercizio 3: Si implementi in linguaggio C un programma che generi due processi comunicanti tramite una pipe. Un processo scrive 4 caratteri nella pipe, il secondo effettua una prima select sulla pipe, quando esce legge due caratteri e successivamente effettua una seconda select. Verificare se la seconda select è bloccante oppure no.

Esercizio 1A (28-11-00)

- Scrivere uno script che accetti una serie di parametri:
 - nth 3 ":" file1.txt file2.txt file3.c
 e che stampi l'i-esimo campo (indicato dal primo parametro) di ciascun file, usando il secondo parametro come field separator.

```
#!/bin/bash
if [ $# -ge 3 ]
then
    n=1
    for i in $*
    do
        if [ $n -ge 3 ]
        then
            cat $i | awk -F$2 -v field=$1 '{print $field;}'
        fi
        n=$((n+1))
    done
fi
```

Esercizio 2A (28-11-00)

- Scrivere uno script che accetti una serie di parametri:
 - nth 3 6 9
 e che stampi i campi \$1, \$2, \$3, ... (indicato dai parametri) del file proveniente da stdin.

```
#!/bin/bash
if [ $# -ge 1 ]
then
    nawk -v var="$*" \
        'BEGIN {t=split(var,field," ");} \
        { \
            for (i=1;i<=t;i++) printf ("%s ",$field[i]); \
            printf ("\n");}'
fi
```

Esercizio 1B (28-11-00)

- Trovare tutti i file .c o .h nella directory /usr/src (e sottodirectory) e sostituire tutte le occorrenze della stringa writemessage con la stringa WriteMessage.

Soluzione 1

```
find /usr/src -name '*.[ch]' \
    -exec sed 's/writemessage/WriteMessage/g' \{} > \
    /var/tmp/myFile \; \
    -exec mv /var/tmp/myFile \{} \;
```

Soluzione 2

```
find /usr/src -name '*.ch' -exec modifica \{} \;
```

Script modifica:

```
#!/bin/bash
sed 's/writemessage/WriteMessage/g' $1 > /var/tmp/myFile
mv /var/myFile $1
```

Esercizio 2B (28-11-00)

- Scrivere uno script che riceva come parametro il nome di un file. Sostituire in tale file tutte le occorrenze della stringa write (case insensitive) con la stringa Write.

Soluzione 1

```
#!/bin/bash
sed 's/write/Write/ig' $1 > /var/myFile
mv /var/myFile $1
```

Soluzione 2

```
#!/bin/bash
sed 's/[wW][rR][iI][tT][eE]/Write/g' $1 > /var/myFile
mv /var/myFile $1
```

Il sistema operativo LINUX

Esercitazione 3

Giorgio Di Natale <dinatale@polito.it>
Stefano Di Carlo <dicarlo@polito.it>

Politecnico di Torino
Dip. Automatica e Informatica

Esercizio 1

- **Mostrare una lista di tutti i file chiamati "core" in tutto il file system. Non visualizzare i messaggi di errore**
- **Con bash:**
 - find / -name core 2> /dev/null

Esercizio 2

- **Visualizzare una lista di tutti i processi dell'utente root. Il formato della lista deve essere:**

PID	%CPU	Comando
<pre>ps aux awk '\$1 ~ "^(root PID)" { print \$3 "\t" \$5 "\t" \$7; };</pre>		

Esercizio 5 - Soluzione

- **Trovare i file che hanno dimensione maggiore di 2000 oppure un nome che inizi con "a" e termini con "o"**
 - find . \(-size +2000 -o -name 'a*o' \)

Il sistema operativo LINUX

Esercitazione 4

Giorgio Di Natale <dinatale@polito.it>
Stefano Di Carlo <dicarlo@polito.it>

Politecnico di Torino
Dip. Automatica e Informatica

Esercizio 1

- **Date due directory a linea di comando, copiare tutti i file dalla prima directory nella seconda, modificando il contenuto di ogni file in maniera da sostituire tutte le occorrenze della stringa DAY con la stringa GIORNO**

Esercizio 1 - Soluzione 1

```
for i in `ls $1/*`
do
    sed -e 's/DAY/GIORNO/g' $i > $2/$i
done
exit 0
```

Note: non copia i file nelle sottodirectory

Esercizio 1 - Soluzione 2

```
cp -R $1/* $2
for i in `find $2 -type f`
do
    sed -e 's/DAY/GIORNO/g' $2/$i > /var/tmp/myFile
    mv /var/tmp/myFile $2/$i
done
exit 0
```

Note: il sed ha effetto anche su i file che erano già presenti nella directory \$2

Esercizio 1 - Soluzione 3

```
cp -R $1/* temp
for i in `find temp -type f`
do
    sed -e 's/DAY/GIORNO/g' temp/$i > myFile
    mv myFile temp/$i
done
cp -R temp/* $2
rm -rf temp/*
```

Esercizio 2

- Verificare che la variabile PATH contenga la directory /usr/local/bin

```
echo $PATH | grep "/usr/local/bin" 2> /dev/null
if [ $? -eq 0 ]
then
    echo "Trovata"
else
    echo "Non trovata"
fi
```

Esercizio 2 - Soluzione 2

- Lo script trova la stringa anche se è presente la directory /usr/local/bin/prova
- Affinchè il risultato sia corretto, grep deve trovare la stringa intera
- La variabile PATH separa le directory con il simbolo :
- Es:


```
PATH = /usr/bin:/usr/local/bin:/bin
```

Esercizio 2 - Soluzione 2 (cont)

- PATH = /usr/bin:/usr/local/bin:/bin
- Tutte le directory (tranne la prima e l'ultima) sono precedute e terminate dal simbolo :
- Il comando grep potrebbe essere eseguito sulla stringa :/usr/local/bin:

Esercizio 2 - Soluzione 2 (cont)

- Affinchè il comando funzioni correttamente anche sulla prima e sull'ultima directory:
echo :\$PATH: | grep ":/usr/local/bin:" 2> /dev/null

Esercizio 3

- Leggere da stdin una serie di stringhe fino a quando la stringa letta è diversa da "FINE". Copiare ogni riga su stdout, preceduta da un numero progressivo

Esercizio 3 - Soluzione 1

```
count=0
read newLine
while [ newLine != "FINE" ]
do
    echo $count: $newLine
    count=$((count+1))
    read newLine
done
```

Note: ogni riga è stampata subito dopo averla letta

Esercizio 3 - Soluzione 2

```
count=0
echo > tempFile
read newLine
while [ newLine != "FINE" ]
do
    echo $count: $newLine >> tempFile
    count=$((count+1))
    read newLine
done
cat tempFile
```

Il sistema operativo LINUX Temi d'esame (modulo I)

Giorgio Di Natale <dinatale@polito.it>
Stefano Di Carlo <dicarlo@polito.it>

Politecnico di Torino
Dip. Automatica e Informatica

Esercizio 1

- Quali operazioni occorre effettuare su un sistema Unix per inserire un nuovo utente?

Esercizio 2

- Descrivere cosa produce l'esecuzione di ciascun comando contenuto nel seguente script file

```
mount pippo:/home/os /mnt
cd
echo pippo > pluto
ln pluto /mnt/pluto
ln -s pluto /mnt/pippo
rm /mnt/pippo
more pluto
more /mnt/p????
```

Mount

- Il mount può avvenire anche per un file system su un computer remoto. In questo caso si utilizza il Network File System (NFS)
- Per montare un disco da un altro computer, il computer remoto deve avere dato il permesso.

Mount (cont)

- File /etc/exports

/directory hostname(permessi) [, hostname (permessi)]

- Esempi:

```
/home cclix*.polito.it(rw), ccultra.polito.it(ro)
/home/users cclinf*.polito.it(rw)
/cadtools cclix1.polito.it(ro)
```

Mount (cont)

- Il comando mount viene eseguito nel seguente modo:
 - mount host:/dir /mnt_point
 - mount giove:/home /home

Esercizio 2 (cont)

- Descrivere cosa produce l'esecuzione di ciascun comando contenuto nel seguente script file

```
mount pippo:/home/os /mnt
cd
echo pippo > pluto
ln pluto /mnt/pluto
ln -s pluto /mnt/pippo
rm /mnt/pippo
more pluto
more /mnt/p????
```

Passa alla home directory

Esercizio 2 (cont)

- Descrivere cosa produce l'esecuzione di ciascun comando contenuto nel seguente script file

```
mount pippo:/home/os /mnt
cd
echo pippo > pluto
ln pluto /mnt/pluto
ln -s pluto /mnt/pippo
rm /mnt/pippo
more pluto
more /mnt/p????
```

Crea il file pluto e scrive al suo interno la stringa "pippo"

Esercizio 2 (cont)

- Descrivere cosa produce l'esecuzione di ciascun comando contenuto nel seguente script file

```
mount pippo:/home/os /mnt
cd
echo pippo > pluto
ln pluto /mnt/pluto
ln -s pluto /mnt/pippo
rm /mnt/pippo
more pluto
more /mnt/p????
```

Crea un link fisico
(può essere eseguito
solo da root)

Esercizio 2 (cont)

- Descrivere cosa produce l'esecuzione di ciascun comando contenuto nel seguente script file

```
mount pippo:/home/os /mnt
cd
echo pippo > pluto
ln pluto /mnt/pippo
ln -s pluto /mnt/pippo
rm /mnt/pippo
more pluto
more /mnt/p????
```

Crea un link simbolico

Esercizio 2 (cont)

- Descrivere cosa produce l'esecuzione di ciascun comando contenuto nel seguente script file

```
mount pippo:/home/os /mnt
cd
echo pippo > pluto
ln pluto /mnt/pippo
ln -s pluto /mnt/pippo
rm /mnt/pippo
more pluto
more /mnt/p????
```

Elimina il file /mnt/pippo
se è un file ordinario

Esercizio 2 (cont)

- Descrivere cosa produce l'esecuzione di ciascun comando contenuto nel seguente script file

```
mount pippo:/home/os /mnt
cd
echo pippo > pluto
ln pluto /mnt/pippo
ln -s pluto /mnt/pippo
rm /mnt/pippo
more pluto
more /mnt/p????
```

Visualizza (a pagine) il
contenuto del file pluto

Esercizio 2 (cont)

- Descrivere cosa produce l'esecuzione di ciascun comando contenuto nel seguente script file

```
mount pippo:/home/os /mnt
cd
echo pippo > pluto
ln pluto /mnt/pippo
ln -s pluto /mnt/pippo
rm /mnt/pippo
more pluto
more /mnt/p????
```

Visualizza (a pagine) tutti i
file di 5 lettere che iniziano
con la lettera p

Esercizio 3

- Scrivere uno script che accetti, come parametro a linea di comando, l'uid di un utente e lo cancelli dall'insieme degli utenti del sistema. Si provveda a creare un file compresso contenente tutti i file della relativa home directory.

*Esercizio 3 - Soluzione***Script rmUser:**

```
#!/bin/bash
awk -f rmUser.awk -v uid=$1 < /etc/passwd > /etc/newPasswd
if [ $? -eq 0 ]
then
    mv /etc/newPasswd /etc/passwd
else
    echo UID $1 non trovato.
fi
exit 0
```

*Esercizio 3 - Soluzione (cont)***Script rmUser.awk:**

```
BEGIN {
    FS = ":";
    flag = 0;
}

$3 != uid {
    print $0;
}
```

Esercizio 3 - Soluzione (cont)

```
$3 == uid {
    system ("tar cf /zhome/" $1 ".tar" "$6");
    system ("gzip /zhome/" $1 ".tar");
    system ("rm -rf " $6);
    flag = 1;
}
```

Esercizio 3 - Soluzione (cont)

```
END {
    if (flag == 1) {
        exit 0;
    } else {
        exit 1;
    }
}
```

Esercizio 4

- Eliminare le sezioni
#ifdef XXX
- di un file C per cui non esiste
#define XXX

Esercizio 1A (28-11-00)

- Scrivere uno script che accetti una serie di parametri:
- nth 3 ":" file1.txt file2.txt file3.c
e che stampi l'i-esimo campo (indicato dal primo parametro) di ciascun file, usando il secondo parametro come field separator.


```
#!/bin/bash

if [ $# -ge 3 ]
then
    n=1
    for i in $*
    do
        if [ $n -ge 3 ]
        then
            cat $i | awk -F$2 -v field=$1 '{print $field;}'
        fi
        n=$((n+1))
    done
fi
```

Esercizio 2A (28-11-00)

- Scrivere uno script che accetti una serie di parametri:
- nth 3 6 9
e che stampi i campi \$1, \$2, \$3, ... (indicato dai parametri) del file proveniente da stdin.

```
#!/bin/bash

if [ $# -ge 1 ]
then
    nawk -v var="$*" \
    'BEGIN {t=split(var,field," ");} \
    { \
        for (i=1;i<=t;i++) printf ("%s ",field[i]); \
        printf ("\n");}'
fi
```

Esercizio 1B (28-11-00)

- Trovare tutti i file .c o .h nella directory /usr/src (e sottodirectory) e sostituire tutte le occorrenze della stringa writemessage con la stringa WriteMessage.

Soluzione 1

```
find /usr/src -name '*.ch' \
-exec sed 's/writemessage/WriteMessage/g' {} > \
/var/tmp/myFile \; \
-exec mv /var/tmp/myFile {} \;
```

Soluzione 2

```
find /usr/src -name '*.ch' -exec modifica {} \;
```

Script modifica:

```
#!/bin/bash
sed 's/writemessage/WriteMessage/g' $1 > /var/tmp/myFile
mv /var/tmp/myFile $1
```

Esercizio 2B (28-11-00)

- Scrivere uno script che riceva come parametro il nome di un file. Sostituire in tale file tutte le occorrenze della stringa write (case insensitive) con la stringa Write.

Soluzione 1

```
#!/bin/bash  
sed 's/write/Write/ig' $1 > /var/myFile  
mv /var/myFile $1
```

Soluzione 2

```
#!/bin/bash  
sed 's/[wW][rR][iI][tT][eE]/Write/g' $1 > /var/myFile  
mv /var/myFile $1
```