https://mylostgnu.altervista.org/guide-linux/bash/primo-script-bash/?doing_wp_cron=1729679980.8859040737152099609375

BASH è un interprete di comandi localizzato in /bin/bash
Per creare un file bash bisogna usare un editor di testo come nano o vim per creare il file:

nano nome_file.sh

(In nano, premi Ctrl+O, poi Invio per salvare e Ctrl+X per uscire)
Per eseguirla bisogna:

./script.sh

Struttura di uno script:

- La prima riga deve essere #!/bin/bash, che specifica l'interprete
- Lo script deve essere reso eseguibile con il comando chmod:

  chmod +x nome_file.sh

Per eseguire lo script bash senza dover scrivere tutto il percorso la directory corrente (indicata da ./) deve essere aggiunta alla variabile PATH con export PATH="$PATH:.".

Variabili posizionali:

- Gli script possono accettare argomenti tramite variabili posizionali ($1, $2, ecc.)
- Esempi di script:
  - cal $1: Mostra il calendario del mese specificato
  - cal $1 > $2: Salva il calendario in un file
  - cal $1 | tee $2: Visualizza e salva il calendario
- **Creazione di comandi personalizzati**:
  - Gli script permettono di ampliare i comandi della shell.
  - Possono essere eseguiti come i comandi predefiniti.
- **Debugging**:
  - Per eseguire il debugging, si usa bash -v nome_script, che mostra ogni riga eseguita per individuare eventuali errori.

Gli script bash consentono di automatizzare operazioni, personalizzare l'ambiente e ampliare le funzionalità della shell.

Per dichiarare una variabile bisogna scrivere nome_variabile=valore (non si deve mettere nessun spazio tra il simbolo di =)

- intero: a=3

- stringa: nome='Marco'

e per visualizzarli si usa echo $a (output: 3)

Per leggere in input si usa read:

```
read name
echo $name
```

Ci sono simboli speciali per gestire le variabili

$ è usato per accedere al valore di una variabile:
bash

```
nome="Alice"

echo $nome
```
(Output: Alice)

- Usato con variabili speciali:
  - $0: Nome dello script
  - $1, $2: Argomenti dello script
  - $?: Stato di uscita del comando precedente
  - $$: PID del processo corrente

$() espande il risultato di un comando:

```
data=$(date)

echo $data
```
(Output: La data e ora corrente)

Può eseguire calcoli:
bash

```
somma=$((3 + 5))

echo $somma
```
(Output: 8)

${}

- **Gestione avanzata delle variabili:**
  - Visualizzazione: ${variabile}
  - Default: ${variabile:-valore} (usa "valore" se la variabile non è definita).
  - Assegnazione: ${variabile:=valore} (assegna "valore" se non definita).
  - Lunghezza: ${#variabile}.

    - Sottostringhe: ${variabile:inizio:lunghezza}.
    - Sostituzione: ${variabile/vecchio/nuovo}.

{}

Isolamento:
bash

prefisso="file"

echo ${prefisso}_nome (Output: file_nome)

Espansione di sequenze:
bash

echo {1..5} (Output: 1 2 3 4 5)

echo {a,b,c}_test (Output: a_test b_test c_test)

Gli array contengono più valori indicizzati da un indice (che parte da 0)

Creazione: array=(valore1 valore2 valore3)

- Esempio: numeri=(10 21 32)
    - numeri[0] → 10
    - numeri[1] → 21
    - numeri[2] → 32

Aggiunta/modifica:

- array[indice]=valore
- Esempio: numeri[1]=22 modifica il secondo valore

Stampa:

- Tutti gli elementi: echo ${array[*]}
- Numero di elementi: echo ${#array[*]}

Eliminazione:

- Tutto l'array: unset array
- Un elemento specifico: unset array[indice]
- Esempio: unset numeri[1] elimina il secondo valore

Argomenti e parametri posizionali

- Quando si esegue uno script, l'utente può fornire degli argomenti insieme al comando
- Questi argomenti vengono associati a **variabili posizionali**: $1, $2, $3, ecc.

Esempio:

```
# Esegui lo script con: ./script.sh 11 2015

echo $1  (Output: 11)

echo $2  (Output: 2015)
```

- La variabile speciale **$#** rappresenta il **numero di argomenti** passati allo script.

Esempio:

```
# Esegui: ./script.sh 11 2015

echo $#  # Output: 2
```

2. Variabili di Sistema

Le variabili di sistema sono predefinite dalla shell e forniscono informazioni sull'ambiente di lavoro:

- **$PATH**: Elenco delle directory dove la shell cerca i comandi.

Esempio:

```
echo $PATH
```

- **$HOME**: Il percorso della home directory dell'utente.

Esempio:

```
echo $HOME
```

- **$PWD**: La directory di lavoro corrente.

Esempio:

```
echo $PWD
```

Una struttura di selezione è realizzata per mezzo dei comandi if, then, else, fi. La sintassi è la seguente:

```
if [ condizione ]
  then
        blocco_comandi
fi
```

L'espressione condizionale deve essere scritta lasciando uno spazio dopo la parentesi quadra aperta [e uno spazio prima della parentesi quadra chiusa].

La struttura di selezione a due vie si rappresenta con:

```
if [ condizione ]
  then
        blocco_comandi_di_then
  else
        blocco_comandi_di_else
fi
```

```
#!/bin/bash
#
# This script creates a new user on the local sytem.
# You will be prompted to enter the username (login), the
person name and a password.
# The username, password, and host for the account will be
displayed.

# Make sure the script is being executed with superuser
privileges.
if [[ "${UID}" -ne 0 ]]
then
   echo 'please run with sudo or as root'
   exit 1
fi
```

```
# Get the username (login)
read -p 'Enter the username to create: ' USER_NAME

# Get the real name (content for the description field)
read -p 'Enter the name of the person or application that will
be using this account: ' COMMENT

# Get the password
read -p 'Enter the password to use for the account: ' PASSWORD

# Create the account
useradd -c "${COMMENT}" -m ${USER_NAME}

# Check to see if the useradd command succeeded
# We don't want to tell the user that an account was created
when it hasn't been
if [[ "${?}" -ne 0 ]]
then
   echo 'The account could ot be created'
   exit 1
fi

# Set the password
echo ${PASSWORD} | passwd --stdin ${USER_NAME}
if [[ "${?}" -ne 0 ]]
then
   echo 'The password for the account could not be sent'
   exit 1
fi

# Force password change on first login
password -e ${USER_NAME}

# Display the username, password, and the host where the user
vas created
echo
echo 'username:'
echo "${USER_NAME}"
echo
echo 'password:'
echo "${PASSWORD}"
echo
echo 'host:'
echo "${HOSTNAME}"
```

```
exit 0
```

---

questo fa il backup di una directory

```
#!/bin/bash
# Script per fare il backup di una directory

origine="/home/utente/documenti"
destinazione="/home/utente/backup"

if [ ! -d "$destinazione" ]; then
    mkdir -p "$destinazione"
fi

cp -r "$origine"/* "$destinazione"
echo "Backup completato!"
```

```
#!/bin/bash
#PASSWORD=${RANDOM}
#echo "${PASSWORD}${PASSWORD}${PASSWORD}"

#PASSWORD=$(date +%s%N)
#echo "${PASSWORD}"

PASSWORD=$(date +%s%N | sha256sum | head -c10)
#echo "${PASSWORD}"

S_C1=$(echo '!@$%&^*()_-+=' | fold -w1 | shuf |head -c1)
S_C2=$(echo '!@$%&^*()_-+=' | fold -w1 | shuf |head -c1)
echo "${S_C1}${S_C2}${PASSWORD}${S_C2}${S_C1}"
```

Il ciclo while esegue un blocco di comandi finché una condizione è vera. La sintassi di base è:

```
while [condizione]
do
   comando
done
```

Esempio: conta da 1 a 5

```
count=1
while [ $count -le 5 ]
do
  echo $count
  ((count++))  # Incrementa count
done
```

Output:
1
2
3
4
5

Il ciclo for permette di iterare su una sequenza di valori (numerica o di stringhe). Le sintassi più comuni sono:

Iterazione su una sequenza numerica

Puoi usare la sintassi {start..end} per iterare su una sequenza di numeri.

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

Output:
1
2
3
4
5

Generare un numero casuale con $RANDOM

$RANDOM è una variabile integrata di Bash che genera un numero casuale compreso tra 0 e 32767.

codice:
```
echo $RANDOM
```

Se vuoi limitare l'intervallo del numero casuale, puoi usare il modulo (%):

codice:
```
echo $((RANDOM % 100))  # Genera un numero casuale tra 0 e 99
```

Si può usare anche date:

```
timestamp=$(date +%s) # Ottieni il timestamp corrente in secondi
random_number=$((timestamp % 100)) # Ottieni un numero casuale tra 0 e 99
echo $random_number
```