set -euo pipefail

- e → non zero exit code
- u → unset variable
- -o pipefail → any failure in pipes

Title: Build an Application Process Manager Script

Sam needs a reusable Bash script to manage a named application process. The script should:

1. Print its own PID and PPID when starting.
2. Start, stop, and restart a named application.
3. Check status of the application (running/not running).
4. Monitor CPU and memory usage at regular intervals.
5. Gracefully stop the process, with a fallback to force-kill if needed.
6. Handle PID file tracking to avoid duplicate starts.
7. Use clean logging for all operations.

Skeleton

Expected usage:

```
./process_manager.sh webapp start
./process_manager.sh webapp status
./process_manager.sh webapp monitor
```

BREAK UP:

1) Print PID & PPID

$$

$PPID

2) Start nohup &

Stop Using name of process → stop. wait → sleep start _

Restart

3) Check status of process

ps -p PID

4) Monitor CPU & Mem

ps -p -o %cpu,%mem

5) Gracefully kill the process, ← kill 15

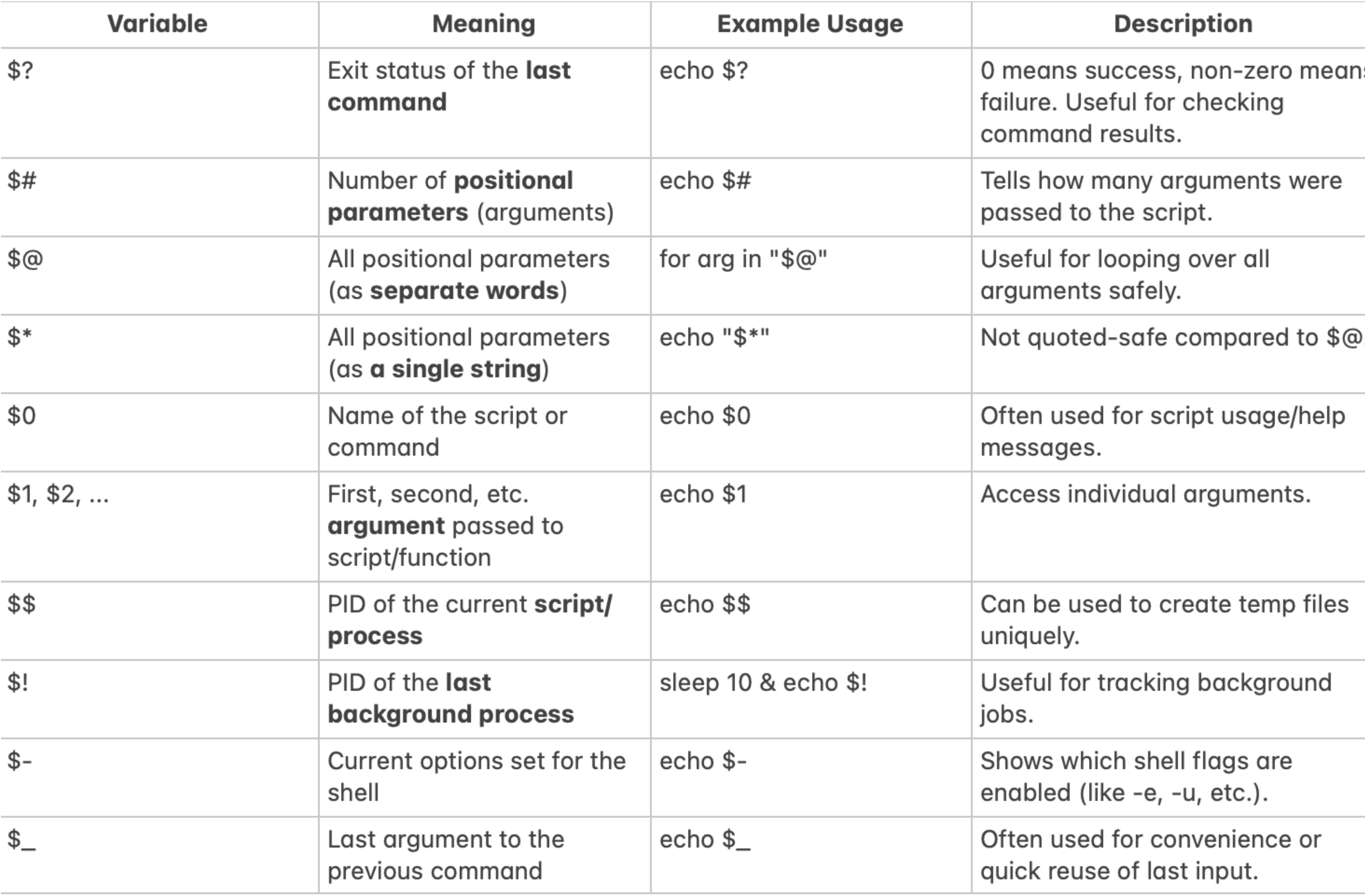
forcefully kill the process – kill 9

$?

$#

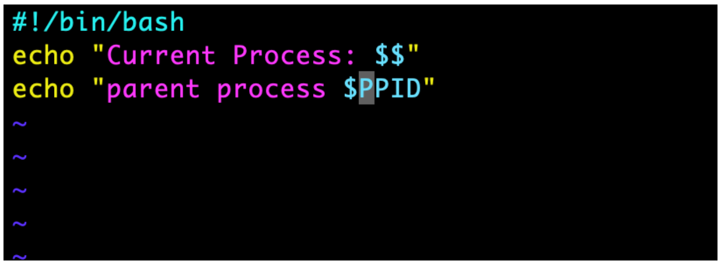
$@

$!



Variable	Meaning	Example Usage	Description
$?	Exit status of the **last command**	echo $?	0 means success, non-zero mean failure. Useful for checking command results.
$#	Number of **positional parameters** (arguments)	echo $#	Tells how many arguments were passed to the script.
$@	All positional parameters (as **separate words**)	for arg in "$@"	Useful for looping over all arguments safely.
$*	All positional parameters (as **a single string**)	echo "$*"	Not quoted-safe compared to $@
$0	Name of the script or command	echo $0	Often used for script usage/help messages.
$1, $2, ...	First, second, etc. **argument** passed to script/function	echo $1	Access individual arguments.
$$	PID of the current **script/ process**	echo $$	Can be used to create temp files uniquely.
$!	PID of the **last background process**	sleep 10 & echo $!	Useful for tracking background jobs.
$-	Current options set for the shell	echo $-	Shows which shell flags are enabled (like -e, -u, etc.).
$_	Last argument to the previous command	echo $_	Often used for convenience or quick reuse of last input.

☆ PID & PPID of current script



```
#!/bin/bash
echo "Current Process: $$"
echo "parent process $PPID"
~
~
~
~
~
```

→ single threaded
single process

./script.sh → spawns a process

Browser → chrome.exe

( )

(date -%Y%m%d) — Subshell

↓

Child proces

★ nohup

- → no hang up
- → process keeps running even if terminal is closed

&

- → background proces
- → proces gets killed if you close the terminal

SKELETON

```
#! /bin/bash
set -euo pipefail
trap 'echo "error at line $LINENO";exit 1'
ERR
APP_NAME="${1:-}"
ACTION="${2:-status}"
PID_FILE="/var/run/$APP_NAME.pid"
print_ids()
start_process()
{
}

stop_process()
{
}

restart()
{     stop
      sleep 1
}     start
```

```
status()
{
}

monitor()
{
}

main ()
{     print_ids

        case "$ACTION" in
          start)  start_process
          stop)  stop_process
          status)  status
         monitor)  monitor
         restart)  restart
         *) echo '              '
      esac
```

```
}

main $@
```

☆ ZOMBIE PROCESSES

```
#!/bin/bash
set -euo pipefail
trap 'echo "Error at line $LINENO"; exit 1 ' ERR

APP_NAME="${1:-}"
ACTION="${2:-}"
PID_FILE="/var/run/$APP_NAME.pid"

print_ids(){
  echo "PID: $$"
  echo "PPID: $PPID"
}

is_running(){
  [[ -f "$PID_FILE" ]] && pid=$(cat $PID_FILE) && ps -p "$pid" > /dev/null 2>&1
}

start_process(){
  if is_running ; then
          echo "Already running $(cat $PID_FILE)"
          return
  fi
  nohup sleep infinity > /dev/null 2>&1 &
  echo $! > "$PID_FILE"
}

status(){
  if is_running; then
          echo "Process is running"
  else
          echo "Process is not running"
  fi

}

stop_process(){
 if ! is_running; then
  echo "Processs not running"
 else
         pid="$(cat $PID_FILE)"
         kill -15 $pid
         count=0
         while is_running; do
           sleep 1
           ((count++))
           if (( count >= 10 )); then
                   echo "Force killing process "
                   kill -9 $pid
                   break
           fi
         done
  rm $PID_FILE
fi
}

restart(){
  stop_process
  echo "Stopping process"
  sleep 1
  echo "Starting process"
  start_process
}

monitor(){
  pid="$(cat $PID_FILE)"
  cpu=$(ps -p "$pid" -o %cpu --no-headers)
  mem=$(ps -p "$pid" -o %mem --no-headers)
  echo "CPU: $cpu"
  echo "MEM: $mem"
}

main(){
  print_ids
  case "$ACTION" in
          start) start_process ;;
          stop) stop_process ;;
          status) status;;
          monitor) monitor;;
          restart) restart ;;
          *) echo 'Wrong usage, correct usage is $0 [start|stop|status|monitor|restart]'
  esac
}
main "$@"
```