Écrire un script shell

Un script shell est un fichier texte contenant un ensemble de commandes, habituellement saisies sur la ligne de commande. Elles sont regroupées sous la forme d'un fichier dont l'exécution peut être lancée comme une commande Linux habituelle.

Ce paragraphe ne prétend pas fournir en quelques lignes une formation à l'écriture de scripts shell. L'objectif plus modeste est de montrer le mécanisme depuis l'écriture d'un script jusqu'à son exécution. Une fois l'intérêt de ce type de programmation découvert, un ouvrage comme *Scripts Shell* de J.M Barabger et T. Schomaker - Éditions ENI sera un complément indispensable.

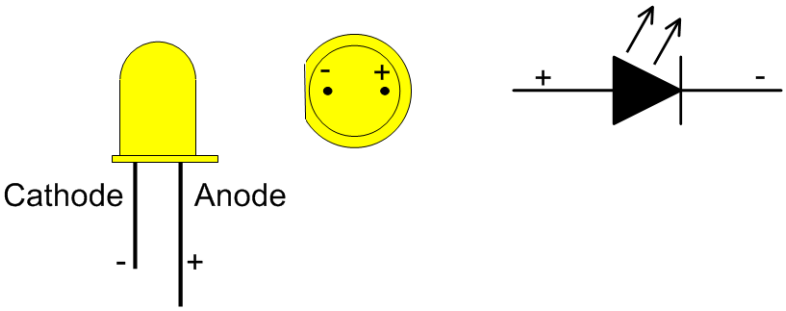
À quoi sert un script shell ?

Un script shell sert à automatiser des tâches qu'un administrateur doit faire de façon répétitive. Cela peut être par exemple une sauvegarde régulière ou une action déclenchée par la présence d'un fichier particulier.

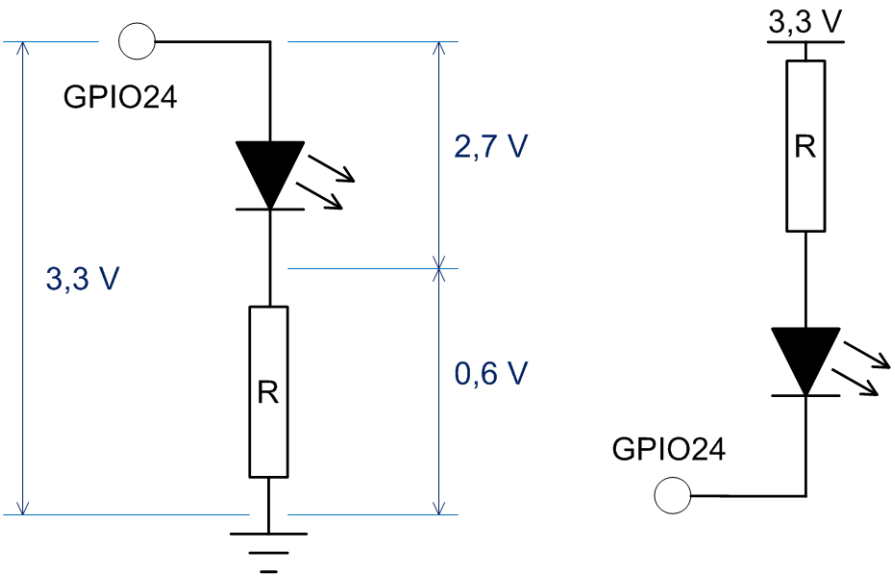
Allumer une LED en ligne de commande

Allumer une diode LED

Les programmeurs ont une coutume : le premier programme qu'ils réalisent dans un langage de programmation affiche "Hello world" ou "Bonjour monde". Cette coutume a été transposée dans le monde des systèmes embarqués : le premier programme consiste à faire clignoter une LED.



La LED utilisée est une diode à lumière blanche qui émet de la lumière lorsqu'elle est parcourue par un courant dans le sens direct. Le repérage des pattes d'une LED est représenté sur le schéma ci-dessus. La cathode est la patte la plus courte du composant. Elle correspond également au méplat situé sur la collerette de la LED.



La diode LED blanche peut être reliée à la broche GPIO 24 de deux manières.

- La première méthode (à gauche sur le schéma ci-dessus) allume la LED lorsque le niveau de sortie de la broche GPIO 24 est haut (3,3 volts). L'anode est reliée à la broche GPIO 24 et la cathode rejoint la masse via une résistance destinée à limiter le courant.
- La seconde méthode (à droite sur le schéma ci-dessus) allume la LED lorsque le niveau de sortie de la broche GPIO 24 est bas (0 volt). La cathode est reliée à la broche GPIO 24. L'anode reçoit les 3,3 volts au travers d'une résistance.

Dans les deux cas, la tension présente aux bornes de l'ensemble diode-résistance se répartit entre les deux composants. La courbe caractéristique de la LED impose une tension de 2,7 volts à ses bornes. Le reste de la tension disponible 3,3 volts - 2,7 volts = 0,6 volt apparaît aux bornes de la résistance.
Les caractéristiques de la LED utilisée indiquent un courant de 20 mA, au-delà des possibilités de la sortie GPIO (16 mA). La LED émet de la lumière à partir d'un courant inférieur à 1 mA avec une luminosité suffisante pour les essais. Cette valeur de courant ne fait courir aucun risque à la sortie GPIO 24 sur laquelle la LED est connectée.
Pour des applications dans lesquelles la luminosité maximale de la LED doit être obtenue, il faudra prévoir un buffer capable de fournir l'intensité requise (transistor, ULN2803...).

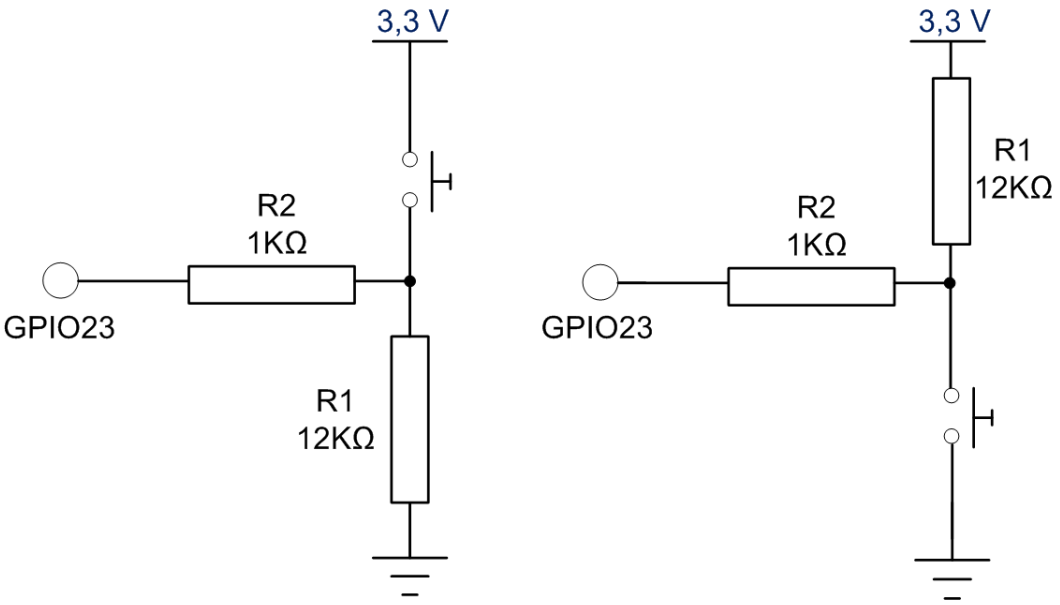
L'ULN2803 est un circuit intégré contenant huit transistors capables de commander une charge de 500 mA sous 50 volts.

La valeur de la résistance est déterminée par R = U/I, avec U = 0,6 volt et I = 1 mA. Une valeur proche de 600 Ω fera l'affaire (560 ou 680 Ω).
Dans les exemples qui suivent, la LED est connectée au GPIO 24 conformément au schéma de gauche : anode reliée au GPIO 24 et cathode reliée à la résistance qui rejoint la masse.

Lire un niveau d'entrée

Les broches du GPIO du Raspberry Pi peuvent également être configurées en entrée. Elles peuvent alors détecter si le niveau présent sur l'entrée est haut ou bas. Configurée en entrée, la broche GPIO est flottante, c'est-à-dire qu'elle n'a pas de niveau de tension défini. Pour garantir le bon fonctionnement de l'entrée, il faudra fixer son niveau avec une résistance de tirage.

RAPPEL IMPORTANT : les broches GPIO du Raspberry Pi ne sont pas protégées. Elles ne doivent pas être soumises à une tension inverse, ni à une tension supérieure à 3,3 volts sous peine de destruction. Il est impératif de vérifier soigneusement les circuits avant de les mettre sous tension.
Lorsqu'il est nécessaire de déclencher une action sur le Raspberry Pi, un interrupteur connecté sur une entrée du GPIO est souvent la solution retenue.
Selon le besoin, un bouton poussoir peut être connecté au rail 3,3 volts ou à la masse. La résistance R1 de 12 kΩ limite le courant à 0,3 mA lors de l'appui sur le poussoir. La résistance R2 de 1 kΩ n'a pas d'effet si GPIO23 est configurée en entrée. Du fait de la haute impédance de l'entrée, le courant qui circule dans R2 ne provoque qu'une chute de tension minime, sans influence sur le fonctionnement de l'entrée.



Par contre, sur le schéma de gauche, dans le cas où le port GPIO23 est configuré en sortie et passe à 0, en l'absence de résistance R2 la sortie est reliée directement au rail +3,3 volts si on appuie sur le poussoir. La sortie est détruite par cette connexion directe à l'alimentation. La résistance R2 sert de protection en limitant le courant dans la sortie à 3,3 volts / 1 kΩ = 3 mA environ. Le GPIO est protégé.
Le rôle de la résistance R2 est identique sur le schéma de droite : lorsque le GPIO23 est configuré en sortie avec un niveau haut et que le poussoir est actionné, R2 limite le courant à 3 mA environ.
Pour les scripts de lecture du port GPIO23, c'est le schéma de gauche comprenant une résistance de tirage reliée à la masse qui sera utilisé.

Gérer le GPIO en ligne de commande

Sur les systèmes Linux, tout est fichier. Par exemple le dossier */dev* mentionné dans le chapitre Utiliser la ligne de commande contient des fichiers qui sont le moyen d'accès du système aux périphériques, aussi bien en écriture qu'en lecture (disques durs, terminaux, imprimantes...). Cette approche standardise les accès et simplifie la tâche des développeurs.
Le GPIO est accessible en ligne de commande à travers un système de fichiers disponible dans l'espace utilisateur (*userspace*). Dans cet espace, les commandes ou applications ne peuvent pas accéder, même accidentellement, à une zone mémoire ne leur appartenant pas.

Accès au GPIO

Les broches du GPIO sont accessibles via */sys/class/gpio*. Ce dossier contient :

```
pi@raspberrypi:/sys/class/gpio $ ls -al
total 0
drwxrwx---  2 root gpio    0 juil.  8 18:51 .
drwxr-xr-x 51 root root    0 janv.  1  1970 ..
-rwxrwx---  1 root gpio 4096 juil.  8 18:51 export
lrwxrwxrwx  1 root gpio    0 juil.  8 18:51 gpiochip0 ->
../../devices/platform/soc/3f200000.gpio/gpio/gpiochip0
lrwxrwxrwx  1 root gpio    0 juil.  8 18:51 gpiochip100 ->
../../devices/platform/soc/soc:virtgpio/gpio/gpiochip100
-rwxrwx---  1 root gpio 4096 juil.  8 18:51 unexport
```

Les fichiers *export* et *unexport* sont utilisés pour créer un accès à une broche du GPIO. *gpiochip0* contient des informations sur le GPIO.

- *export* : fichier en écriture seule, permet de demander au noyau d'exporter le contrôle d'une broche du GPIO dans l'espace utilisateur. Il suffit d'écrire le numéro de la broche du GPIO dans export avec une redirection pour réaliser cette opération. Attention, il s'agit bien ici du numéro de la broche du GPIO sur le SoC et pas le numéro de la broche du connecteur de sortie GPIO situé sur la carte de circuit imprimé du Raspberry Pi !
- *gpiochip0* : le contrôleur du GPIO est accessible par *gpiochip0*. Dans ce dossier figurent en particulier des fichiers en lecture seule permettant de lire les informations sur le GPIO :

```
pi@raspberrypi:/sys/class/gpio/gpiochip0 $ ls -al
total 0
drwxrwx--- 3 root gpio    0 juil.  8 18:51 .
drwxr-xr-x 3 root root    0 juil.  8 18:51 ..
-rwxrwx--- 1 root gpio 4096 juil.  8 18:51 base
lrwxrwxrwx 1 root gpio    0 juil.  8 18:51 device -> ../../../3f200000.gpio
-rwxrwx--- 1 root gpio 4096 juil.  8 18:51 label
-rwxrwx--- 1 root gpio 4096 juil.  8 18:51 ngpio
drwxrwx--- 2 root gpio    0 juil.  8 18:51 power
lrwxrwxrwx 1 root gpio    0 juil.  8 18:51 subsystem ->
../../../../../../class/gpio
-rwxrwx--- 1 root gpio 4096 juil.  8 18:51 uevent
```

Les fichiers les plus utiles de ce dossier sont :

- *base* qui renvoie le numéro du contrôleur de GPIO.
- *label* qui retourne le nom attribué au GPIO.
- *ngpio* qui indique le nombre de broches du GPIO sur le SoC. Il est intéressant de noter que *ngpio* renvoie 54, alors que seules 26 broches du GPIO sont accessibles. Toutes les broches ne sont pas forcément accessibles, ou sont utilisées pour commander d'autres parties de Raspberry Pi.
- *unexport* : fichier en écriture seule, permet d'informer le noyau qu'il doit supprimer l'accès à une broche du GPIO. Il faut écrire le numéro de la broche dont il faut supprimer l'accès dans *unexport* avec une redirection.

Création d'un accès à une broche GPIO

Par défaut aucun accès à une broche du GPIO n'existe. Pour pouvoir utiliser une broche il faut dans un premier temps demander que l'accès soit créé :

```
pi@raspberrypi:/sys/class/gpio $ echo 24 > export
```

Un nouveau dossier *gpio24* apparaît dans le dossier */sys/class/gpio*. C'est le point d'entrée pour configurer la broche GPIO 24 du SoC :

```
pi@raspberrypi:/sys/class/gpio $ ls -al gpio24
```

```
lrwxrwxrwx 1 root gpio 0 juil.  8 19:14 gpio24 ->
../../devices/platform/soc/3f200000.gpio/gpio/gpio24
```
Pour créer d'autres accès à d'autres broches du GPIO, il faut recommencer l'opération en indiquant à chaque fois le numéro de la broche à laquelle on souhaite accéder.

Suppression d'un accès à une broche GPIO

De la même manière que le fichier *export* informe le noyau qu'il doit créer un accès à une broche du GPIO, le fichier *unexport* va lui indiquer qu'il doit supprimer un accès. Pour supprimer l'accès au GPIO 24 :

```
pi@raspberrypi:/sys/class/gpio $ ls
export  gpio24  gpiochip0  gpiochip100  unexport
pi@raspberrypi:/sys/class/gpio $ echo 24 > unexport
```

Après écriture du numéro de broche (ici 24) dans le fichier *unexport*, le dossier *gpio24* a disparu :

```
pi@raspberrypi:/sys/class/gpio $ ls
export  gpiochip0  gpiochip100  unexport
```

Récupérer les informations d'une broche du GPIO

Il est possible de récupérer les informations de direction (broche en entrée ou en sortie) et la valeur existant actuellement sur la broche du GPIO :

```
pi@raspberrypi:/sys/class/gpio $ echo 24 > export
pi@raspberrypi:/sys/class/gpio $ ls
export  gpio24  gpiochip0  gpiochip100  unexport
pi@raspberrypi:/sys/class/gpio $ cd gpio24
pi@raspberrypi:/sys/class/gpio/gpio24 $ ls
active_low  device  direction  edge  power  subsystem  uevent  value
pi@raspberrypi:/sys/class/gpio/gpio24 $ cat direction
in
pi@raspberrypi:/sys/class/gpio/gpio24 $ echo out > direction
pi@raspberrypi:/sys/class/gpio/gpio24 $ echo 1 > value
pi@raspberrypi:/sys/class/gpio/gpio24 $ cat value
1
pi@raspberrypi:/sys/class/gpio/gpio24 $ echo 0 > value
```

Ici la broche GPIO 24 est paramétrée en sortie (out) et la valeur en sortie est 0 (0 volt). L'écriture d'un 1 dans le fichier *value* fait passer la sortie à 1 (3,3 volts), ce qui est confirmé lors de la nouvelle lecture de ce fichier. La LED branchée sur le GPIO 24 est allumée. Lorsque la broche GPIO 24 est paramétrée en entrée, la lecture de *value* donne le niveau présent sur la broche.

Clignotement d'une LED en shell

En utilisant les commandes précédentes, allumez et éteignez successivement une LED connectée au GPIO 24.
Faites d'abord la manipulation manuellement.

Ecrivez ensuite un script *led_clignote.sh* pour automatiser le clignotement.

Planifier des tâches

S'il est intéressant de disposer d'un script automatisant certaines opérations, il serait également pratique que ce script se lance automatiquement à certaines heures. Linux

dispose de cette fonction, c'est `crontab` qui permet l'exécution d'un script ou d'une commande à une heure donnée ou selon un cycle défini par l'utilisateur. `crontab` est le diminutif de chrono table (table de planification). Cette table spécifie les tâches à exécuter ainsi que l'heure à laquelle l'exécution doit être déclenchée. Une périodicité peut également être indiquée pour chacune des tâches.

Pour créer une table ou l'éditer, la commande est `crontab -e`.

Cette commande ouvre l'éditeur de texte par défaut (*nano* sur Raspbian) sur la *crontab* personnelle de l'utilisateur. Lors de la première ouverture, ce fichier ne contient que des lignes d'explications commençant par un #.

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
```

À la fin du fichier, sous les commentaires, l'utilisateur peut programmer des tâches et le moment de leur exécution.

Syntaxe des lignes dans la crontab utilisateur

`m h dom mon dow commande`

`m`	Minutes de 0 à 59.
`h`	Heure de 0 à 23.
`dom`	*day of month* = jour du mois de 1 à 31.
`mon`	*month* = mois de 1 à 12 ou l'abréviation du mois (en anglais soit : jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov et dec).
`dow`	*day of week* = jour de la semaine avec 0 et 7 pour dimanche, 1 pour lundi…
`commande`	Le script doit au moins avoir le droit exécution pour l'utilisateur qui lance la tâche.

Pour chaque unité de temps, plusieurs notations existent :

`*`
: la tâche est exécutée à chaque unité de temps.

`3-6` la tâche est exécutée aux unités de temps 3, 4, 5 et 6.

`*/2` la tâche est exécutée toutes les deux unités de temps 0, 2, 4, 6, 8…

`2,7` la tâche est exécutée aux unités de temps 2 et 7.

Exemples de programmation de la tâche sauve.sh

La sauvegarde se lance à la demi de toutes les heures :

```
30    *     *     *     *     /home/pi/sauve.sh
```

La sauvegarde se lance tous les jours à 18 H 00 :

```
00    18    *     *     *     /home/pi/sauve.sh
```

La sauvegarde se lance toutes les 10 minutes :

```
10,20,30,40,50,0    *     *     *     *     /home/pi/sauve.sh
*/10    *     *     *     *     /home/pi/sauve.sh
```

La sauvegarde se lance le jeudi à 12 H 30 :

```
30    12    *     *     4     /home/pi/sauve.sh
```

La vérification des tâches programmées se fait avec la commande `crontab -l` qui affiche la liste des tâches présentes dans la *crontab* utilisateur.

Il est également possible de programmer une tâche pour qu'elle s'exécute au démarrage de la machine :

Syntaxe

```
@reboot commande
```

La directive `@reboot` remplace les cinq champs de date et d'heure

Mise en pratique

1- Ecrire un script Shell faisant clignoter la LED connectée au GPIO 1 fois par seconde
2- Ecrire un script Shell lancé en cron qui produit 2 flash de la LED toutes les 10 secondes