Scripting Shell : configurer sa tablette Wacom en quelques heures

Alban Mancheron

SCRIPTING SHELL : CONFIGURER SA TABLETTE WACOM EN QUELQUES HEURES

Alban MANCHERON
[Enseignant-chercheur en bioinformatique à l'Université de Montpellier, linuxien depuis 1997 (convaincu depuis 1998)]

Cet article s'adresse principalement à celles et ceux qui auraient une tablette de la marque Wacom à configurer. Pour les autres, c'est l'occasion de lire quelques lignes de Bash et de (re)découvrir quelques commandes comme xbindkeys ou encore notify-send (donc un petit plaisir pour les yeux).

Une tablette graphique ! Voici le dernier outil de travail mis à ma disposition. Notre direction a fait les choses bien. Ils ont choisi la marque **Wacom**, qui est complètement compatible avec **GNU/Linux**. Sauf si vous avez un modèle récent et que votre distribution est un peu vieillissante, ces tablettes fonctionnent « *out of the box* » (expression assez amusante au demeurant, car une tablette qui reste dans son emballage, ça manque cruellement d'intérêt).

Oups, ma distribution est vieillissante (comme moi). Pire que cela, mon gestionnaire de fenêtre est à l'opposé de tous les standards. Vais-je pouvoir profiter de cet outil ? La réponse est oui. Après quelques heures, j'ai une tablette digne de la **DeLorean** du Dr Emmett Brown à la fin du premier opus. Elle ressemble au reste de mon système, une configuration qu'on ne retrouve pas chez tout le monde, un prolongement de mes doigts...

Les codes complets afférents à cet article sont disponibles sur : https://gite.lirmm.fr/doccy/wacom_config. Certains extraits de code ne sont pas fournis dans cet article pour économiser un peu de place.

1. RETOUR VERS LA TABLETTE

Tout d'abord, revenons sur le modèle. La tablette est une **Intuos BT M**. Ce nom de code cache quelques informations. Tout d'abord la série Intuos correspond aux tablettes à stylet de cette marque. Les lettres BT signifient que ce modèle est « connectable » par *bluetooth* (en plus de l'USB) et la lettre M qu'il s'agit de la taille *Medium*.

Vous aviez le projet d'investir dans une tablette (ou c'est déjà fait), mais ce n'est pas le même modèle... Et bien, passez à l'article d'après.

Non, c'était une (mauvaise) blague. En réalité, l'essentiel de l'article devrait s'appliquer à tous les modèles de la marque, et en essence à n'importe quelle autre marque qui fournirait un utilitaire de configuration en ligne de commande.

FIGURE 1

La tablette est une Intuos BT M.

Description du modèle

Pour revenir à ce modèle, il est donc connectable par USB ou *bluetooth*. La tablette a une diagonale de 7 pouces, ce qui fait 0,58 pied pour ceux qui ne savent pas compter en pouces ou encore 0,17 verge pour ceux qui ne savent pas compter en pieds (on me chuchote à l'oreillette que ce système métrique n'est pas universel et que par conséquent il faut dire que la diagonale fait environ 18 cm). En réalité, c'est entre du A5 et du A4. Il y a 5 boutons sur la partie supérieure. Le bouton central sert à allumer ou éteindre la tablette lorsqu'elle

est connectée en *bluetooth* et est équipée d'une LED pour indiquer le statut (allumée ou éteinte, en charge ou chargée, en cours de connexion...). Je vous renvoie à la photo de la figure 1 pour vous représenter la bête.

Ce modèle est également fourni avec un stylet disposant d'une pointe (qui en réalité est un bouton) capable de distinguer 4096 niveaux de pression et de deux boutons latéraux.

Enfin, ce modèle est fourni avec un câble USB, 3 pointes de rechange, ainsi qu'un mode d'emploi *suécophile* (NDLA : l'explication vient dans les lignes qui suivent).

Le mode d'emploi dit quoi ?

Là, j'avoue avoir été déstabilisé. J'attendais un mode d'emploi en 47 langues, de 12 pages par langue indiquant clairement qu'il ne faut pas mettre la prise USB dans ses trous de nez ni passer la tablette au lave-vaisselle... eh bien non ! Un simple schéma en trois dessins indiquant qu'il faut brancher la tablette en USB à l'ordinateur, qu'il faut aller télécharger quelque chose sur le site de Wacom et enfin qu'il faut débrancher la tablette, puis appuyer sur le bouton central pour se connecter en *bluetooth*.

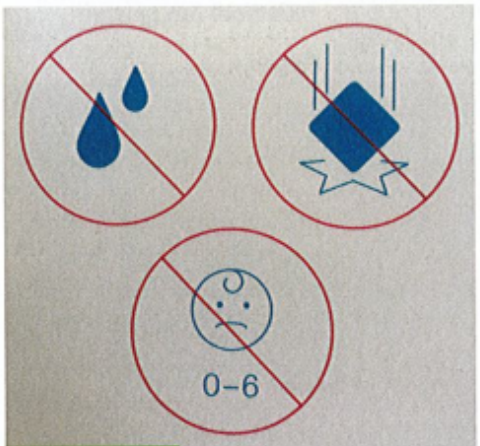


FIGURE 2 *Précautions d'emploi.*

Me voici rassuré, au dos du mode d'emploi il y a également trois dessins (cf. figure 2) : des gouttes dans un panneau barré pour dire qu'il ne faut pas pleurer, une tablette qui tombe par terre dans un panneau barré pour dire qu'il ne faut pas jouer au basket avec, et une tête de bambin stylisée toute triste sous-titrée par « 0-6 » dans un panneau barré pour dire qu'il ne faut pas faire des enfants de moins de 6 ans (là, j'avoue ne pas voir le rapport).

Bon, je me lance, je branche le câble. J'avance avec précaution le stylet de la tablette, et magie : le curseur bouge et se déplace. J'effleure la tablette avec la pointe du stylet et mon menu spécial clic gauche de souris sur le bureau apparaît. Je relève la pointe et j'appuie sur le bouton latéral du bas du stylet et là, c'est mon menu spécial clic droit sur le bureau qui apparaît. J'appuie sur le bouton latéral haut et j'ai le droit à mon menu spécial clic-milieu. Je continue de découvrir et j'appuie sur le premier bouton en haut de la tablette. C'est un clic-gauche, le second bouton, c'est un clic-droit. J'ignore le troisième bouton (on/off) pour voir ce que fait le quatrième et j'ai mon clic-milieu. Le dernier bouton ne fait rien.

Je débranche la tablette, j'appuie sur le bouton on/off, je vois que la LED clignote rapidement. Je me dis que c'est le mode découverte du *bluetooth*. Je vais dans mon gestionnaire de dent bleue et je vois la tablette. Je l'appaire (et non pas apparie), elle est détectée comme un dispositif de pointage. C'est génial. Par acquit de conscience, j'envisage de refaire les tests précédents et là, rien ne se produit. Pas même un déplacement de curseur.

Je retourne vers le mode d'emploi (par habitude) et je constate que le logotype « ne pas pleurer » prend tout son sens. Je tente les manipulations logiques, les manipulations justifiables, quoique surprenantes, les manipulations ésotériques. Rien.

Principe de base, avant d'incriminer le matériel, commençons par nous remettre en cause. Mon gestionnaire de fenêtre est **Enlightenment** en version 0.16. Je passe sur un compte `guest` sous **Unity** pour voir si ceci explique cela. Non, cela ne vient pas de ça. Cependant, le passage sous Unity me permet d'accéder au menu des paramètres, qui dispose d'une icône « Tablette Wacom ». Je vais sur cette icône et en mode USB, la tablette est bien détectée. Par contre, les paramétrages que je peux faire via cette interface sont limités et surtout ne fonctionnent pas.

Ma distribution est une **Ubuntu 16.10** (que je n'ai pas envie/besoin de mettre à jour). Je lance une machine virtuelle avec **Ubuntu 19.10**, je transfère le port USB et là, je peux paramétrer la tablette correctement. Mon diagnostic me laisse perplexe. Il me faudrait mettre ma distribution à jour.

2. EXPLORONS LES ALTERNATIVES

Je ne céderai pas devant l'adversité. Je ne ferai pas une mise à jour juste pour une tablette, non mais !!!

Je vais sur les forums, et très rapidement j'arrive sur les mêmes liens et les mêmes tutoriels et surtout sur la page qui contient tous les programmes et toutes les instructions : **https://linuxwacom.github.io/**.

Premièrement, regarder quelle version des *drivers* Wacom est installée et quels sont les prérequis pour faire fonctionner la tablette sur le site **https://github.com/linuxwacom/input-wacom/wiki/Device-IDs**.

Il me faut visiblement au moins un noyau Linux en version 4.17, le driver **input-wacom** en version 3.17 et la librairie **libwacom** en version 0.30. Je regarde ce qui est installé par défaut et j'ai un noyau en version 4.15, le *driver* **input-wacom** en version 0.32 et la librairie **libwacom** en version 0.22... Ne pas pleurer (c'est sur la notice) ! Bon en réalité, la version du *driver* n'est pas corrélée à sa *release* (le véritable numéro de version du *driver* installé sur ma distribution est 3.13), mais je suis quand même sur des versions trop anciennes.

Ce n'est pas grave, je persévère... et je prends un verre.

Installation des pilotes, outils et librairies

Je ne cherche pas à faire dans le détail, je commence par cloner le dépôt Git du *driver* **input-wacom** en suivant bêtement les instructions qui sont fort bien décrites dans la documentation (**https://github.com/linuxwacom/input-wacom/wiki/Installing-input-wacom-from-source**).

Pas de soucis, tout se passe bien, j'ai mes nouveaux *drivers* `wacom` et `wacom_w8001` correctement installés.

Je passe ensuite à l'installation des *drivers* pour le serveur graphique en suivant la documentation (**https://github.com/linuxwacom/xf86-input-wacom/wiki/Building-The-Driver**). La seule chose que je modifie, c'est que je n'exécute pas :

```
$ set -- --prefix="/usr" --libdir="$(readlink -e $(ls -d /usr/lib*/
xorg/modules/input/../../../ | head -n1))"
$ if test -x ./autogen.sh; then ./autogen.sh "$@"; else ./configure
"$@"; fi && make && sudo make install || echo "Build Failed"
```

Mais j'exécute directement :

```
$ ./autogen.sh --prefix="/usr" --libdir="$(readlink -e $(ls -d /usr/
lib*/xorg/modules/input/../../../ | head -n1))"
$ make
$ sudo make install
```

Même topo, **xsetwacom** est correctement installé. Je ne prends même pas la peine de redémarrer ma session comme préconisé (tellement je suis un fou dans ma tête !!!).

Enfin, j'installe la librairie permettant de reconnaître les différents modèles de tablettes (**https://github.com/linuxwacom/libwacom/wiki**).

L'outil **libwacom-list-local-devices** est correctement créé et installé.

Comme dirait l'autre : *yapuka* !

Premiers tests

Je rebranche ma tablette en USB, elle semble bien fonctionner. Je relance **unity-control-center**, vais sur l'icône de configuration « Tablette Wacom » et...

```
$ unity-control-center

(unity-control-center:10666): wacom-cc-panel-WARNING **: Could not set the
current stylus ID 0x0 for tablet 'Wacom Intuos BT M', no general pen found
**
wacom-cc-panel:ERROR:gsd-wacom-device.c:1850:gsd_wacom_device_set_current_
stylus: assertion failed: (device->priv->styli)
Abandon (core dumped)
```

C'est le moment de vérifier deux/trois choses dans notre installation. Commençons par lister les périphériques Wacom reconnus :

```
$ xsetwacom --list devices
Wacom Intuos BT M Pad pad               id: 16  type: PAD
Wacom Intuos BT M Pen stylus            id: 17  type: STYLUS
Wacom Intuos BT M Pen eraser            id: 18  type: ERASER
Wacom Intuos BT M Pen cursor            id: 19  type: CURSOR
```

Bon, les périphériques semblent reconnus. Voyons quelles sont les informations supplémentaires dont on dispose sur ce modèle :

```
$ libwacom-list-local-devices
# Device node: /dev/input/event19
[Device]
Name=Wacom Intuos BT M
ModelName=CTL-6100WL
DeviceMatch=usb:056a:0378;bluetooth:056a:0379;
...

[Features]
```

```
...

[Buttons]
...

------------------------------------------------------------------
# Device node: /dev/input/event18
[Device]
Name=Wacom Intuos BT M
ModelName=CTL-6100WL
DeviceMatch=usb:056a:0378;bluetooth:056a:0379;
...

[Features]
...

[Buttons]
...

------------------------------------------------------------------
```

Ça semble relativement fidèle à la description de la tablette... On notera au passage comme information intéressante que la connexion USB correspond au périphérique du vendeur **056a** (au sens **udev**) et au produit **0378** tandis que la connexion *bluetooth* correspond au périphérique du vendeur **056a** et au produit **0379**.

Bon, avant de chercher plus avant comment configurer la tablette, voyons si la connexion *bluetooth* est opérationnelle. Je débranche la tablette, j'appuie sur son bouton central, et en quelques secondes, la tablette est appairée. Moment de vérité, j'approche le stylet et ça fonctionne.

Configuration

Deux choix s'offrent désormais à moi. Soit trouver pourquoi l'interface graphique de configuration ne trouve pas de stylet, soit tenter de passer par l'outil **xsetwacom**, que les forums encensent (un peu trop à mon goût, car j'ai une confiance modérée dans sa robustesse).

Je choisis de tester la ligne de commande. Soit l'outil est robuste et j'aurai fait le bon choix, soit l'outil n'est pas à la hauteur de sa réputation et j'en aurai ainsi le cœur net.

Pour appréhender ses fonctionnalités, il suffit d'exécuter la commande sans arguments qui affiche un message d'usage sobre et précis (faites-le, vous verrez).

Nous avons déjà pris un peu d'avance, car nous avons déjà vu la sous-commande permettant de lister les périphériques Wacom détectés (la tablette était branchée en USB). En mode *bluetooth*, on obtient :

```
$ xsetwacom --list devices
Wacom Intuos BT M Pad pad               id: 16  type: PAD
Wacom Intuos BT M Pen stylus            id: 17  type: STYLUS
```

La commande permet de récupérer la liste des différents paramètres contrôlables avec cet outil (la liste offre 39 possibilités, dont certaines attirent davantage que d'autres mon attention).

```
$ xsetwacom --list parameters
Area              - Valid tablet area in device coordinates.
Button            - X11 event to which the given button should be mapped.
...
Mode              - Switches cursor movement mode (default is absolute).
...
Rotate            - Sets the rotation of the tablet. Values = none, cw, ccw,
                    half (default is none).
...
Threshold         - Sets tip/eraser pressure threshold (default is 27).
...
all               - Get value for all parameters.
```

Je tente le premier paramètre au hasard sur chacun des deux périphériques accessibles en *bluetooth* :

```
$ xsetwacom --get "Wacom Intuos BT M Pad pad" Area
Property 'Wacom Tablet Area' does not exist on device.
$ xsetwacom --get "Wacom Intuos BT M Pen stylus" Area
10 20 100 200
```

Étrange. Je m'attendais à ce que ce soit le périphérique de type **PAD** qui ait cette propriété plutôt que celui de type **PEN**.

Je regarde ce qui se passe si je fournis l'option **--xconf** ou **--shell** :

```
$ xsetwacom --get "Wacom Intuos BT M Pad pad" Area
Property 'Wacom Tablet Area' does not exist on device.
$ xsetwacom --get "Wacom Intuos BT M Pen stylus" Area
10 20 100 200
$ xsetwacom --xconf --get "Wacom Intuos BT M Pen stylus" Area
Option "Area" "10 20 100 200"
$ xsetwacom --shell --get "Wacom Intuos BT M Pen stylus" Area
xsetwacom set "Wacom Intuos BT M Pen stylus" "Area" "10 20 100 200"
```

Ça, ça me plaît bien (surtout l'option **--shell**).

Juste pour voir ce qui se passe si je modifie une valeur :

```
$ xsetwacom set "Wacom Intuos BT M Pen stylus" "Area" "10 20 100 200"
```

Eh bien, il ne me reste que quelques millimètres du coin haut-gauche qui sont encore actifs... Je déconnecte la tablette, puis la reconnecte. Le paramétrage est perdu. Il faut donc que tout changement de paramètre soit effectué à chaque connexion de la tablette.

Je vois tout de suite l'intérêt du paramètre **all**. Je vais créer un script pour gérer finement mon paramétrage.

3. SCRIPTONS, JOYEUX COMPAGNONS

Je liste la configuration par défaut de la tablette (**pad** + **pen**) pour voir ce qui est paramétrable :

```
$ for t in "Pen stylus" "Pad pad"; do
>   xsetwacom -s get "Wacom Intuos BT M ${t}" all 2> /dev/null;
> done
xsetwacom set "Wacom Intuos BT M Pen stylus" "Area" "0 0 21600 13500"
xsetwacom set "Wacom Intuos BT M Pen stylus" "Button" "1" "button +1 "
xsetwacom set "Wacom Intuos BT M Pen stylus" "Button" "2" "button +2 "
...
xsetwacom set "Wacom Intuos BT M Pen stylus" "PanScrollThreshold" "1300"
xsetwacom set "Wacom Intuos BT M Pad pad" "Button" "1" "button +1 "
xsetwacom set "Wacom Intuos BT M Pad pad" "Button" "2" "button +2 "
...
xsetwacom set "Wacom Intuos BT M Pad pad" "PanScrollThreshold" "13"
```

Je comprends mieux pourquoi les commentaires sur les forums encensent cet outil. Encore un « couteau suisse » qui fait ce pour quoi il est conçu, tout en restant très simple d'utilisation.

Visiblement, il est même possible de définir des boutons virtuels (avec des codes fantaisistes) et de capturer les signaux envoyés par ces boutons pour les associer à des combinaisons de touches *via* la commande **xbindkeys** (cf. note à ce sujet).

À la lecture de la documentation de **xsetwacom**, il est donc possible d'associer à un bouton une action consistant à émuler l'appui sur un bouton de souris (ce qui est fait par défaut). Il est également possible d'associer à un bouton une action consistant à simuler une succession d'événements clavier (ce qui permet par exemple d'ouvrir un menu d'une application, de s'y déplacer et de sélectionner un item). La syntaxe est alors d'utiliser le mot clé **key** suivi de la séquence de touches à simuler, au lieu de **button** suivi de l'identifiant. Il y a également d'autres possibilités, mais j'ai choisi de les ignorer.

Un script comme les autres

Tout d'abord, je crée un répertoire dédié :

```
$ cd /usr/local/src
$ sudo mkdir wacom_config
$ sudo chown $(id -u):$(id -g) wacom_config
$ cd wacom_config
```

Je m'empresse de le versionner avec **git**, d'ajouter un fichier **README.md** et de créer mon fichier de filtre des fichiers à ignorer dans le suivi (ce qui est fait n'est plus à faire).

Attaquons par le commencement, le script de configuration de la tablette que je baptise avec une audace mêlée d'originalité : **wacom_config**. Clairement, les premières lignes n'ont rien de spécifique au projet et je les récupère d'un autre script. Ces lignes permettent simplement de récupérer le nom du script (donc ici **wacom_config**) et le répertoire absolu dans lequel il réside (donc ici **/usr/local/src/wacom_config/**). Pourquoi le calculer alors que je le connais ? Eh bien tout simplement parce que j'ai l'espoir immense qu'au moins une personne récupère un jour ce script et que si cette personne l'installe ailleurs ou bien le renomme, eh bien cette personne ne sera pas obligée de mettre les mains dans le cambouis pour pouvoir l'utiliser.

L'outil XBindKeys (fourni par le paquet éponyme sur Ubuntu/Debian) permet de capturer un événement – ou une combinaison d'événements – clavier ou souris envoyés au serveur X et d'y associer une action particulière (typiquement, un programme à exécuter).

La page de manuel est très bien documentée, donc je ne m'étendrai pas sur tout ce que peut faire cette commande pour vous, mais seulement sur ce dont j'ai besoin ici.

Tout d'abord, il est possible de lancer la commande afin d'obtenir la description d'un événement capturé grâce à l'option `-m` ou mieux si l'on veut faire plusieurs tests, avec l'option `-mk` :

```
$ xbindkeys -mk
Press combination of keys or/and click under the window.
You can use one of the two lines after "NoCommand"
in $HOME/.xbindkeysrc to bind a key.

--- Press "q" to stop. ---
"NoCommand"
    m:0x11 + c:24
    Shift+Mod2 + a
"NoCommand"
    m:0x11 + c:50
    Shift+Mod2 + Shift_L
"NoCommand"
    m:0x414 + b:3   (mouse)
"NoCommand"
    m:0x14 + c:37
    Control+Mod2 + Control_L
"NoCommand"
    m:0x10 + c:38
    Mod2 + q
$
```

Dans cet exemple, j'ai pressé la touche <Shift> (de gauche) et la touche <A>, puis j'ai relâché la touche <Shift>. La valeur `Mod2` qui apparaît partout correspond au fait que mon pavé numérique est verrouillé. Ensuite, j'ai appuyé sur la touche <Ctrl> (de gauche), puis ai cliqué sur le bouton droit de la souris. Enfin, j'ai appuyé sur la touche <Q> ce qui m'a permis de quitter le programme (comme indiqué au démarrage).

La sortie produite correspond ni plus ni moins aux lignes à ajouter au fichier de configuration `${HOME}/.xbindkeysrc` pour associer une action (ici, l'action `NoCommand`) à la combinaison d'événements capturés.

```
 1: #!/bin/bash
 ...
74: set -e
75:
76: ## Configuration variables
77: DIRNAME=$(dirname "${BASH_SOURCE[0]}")
78: DIRNAME=$(readlink -m "${DIRNAME}")
79: PROGNAME=$(basename "${BASH_SOURCE[0]}")
```

Toujours dans l'optique que quelqu'un utilise cet outil, ce quelqu'un pourrait vouloir affecter des actions différentes aux boutons de la tablette. Je prévois donc de passer par un bon vieux fichier de configuration. Je prévois même le cas d'une installation multi-utilisateur avec un fichier de configuration global (par défaut) et un fichier de configuration spécifique à un utilisateur (dans son répertoire maison). Je définis donc trois variables `WACOM_CONFIG_DIR`, `WACOM_DEFAULT_CONFIG` et `WACOM_USER_CONFIG` (lignes 81 à 83 du fichier).

Faisons une première pause (on n'est pas des bêtes) et réfléchissons à ce(s) fichier(s) de configuration. Tout d'abord, j'aime que les choses soient bien rangées, donc le fichier de configuration par défaut se nommera `default_config` et sera installé dans un sous-répertoire `config`.

Je vais considérer que toutes les variables qui seront déclarées dans ce fichier seront en *screaming snake case* (je vous renvoie à la lecture d'un précédent article dans votre revue favorite **[1]** pour les différents standards de notation). De plus, elles auront toutes pour préfixe `WACOM_`. Toutes les autres variables seront en minuscules. Je m'occuperai du contenu par défaut de ces variables ultérieurement.

De la même manière, tous les noms de fonctions seront écrits en *snake case*. Les fonctions « privées » seront préfixées par `_wacom_` tandis que les fonctions « publiques » seront préfixées par `wacom_` (la notion de privée/publique dans ce contexte sera explicitée un peu plus tard).

Fin de la pause, reprenons le script principal. Comme je vais permettre d'afficher des messages d'aide, je vais commencer par définir le début de ce message dans une variable `WACOM_HELP_MESSAGE` (qui pourra bien entendu être écrasée dans les fichiers de configuration).

Pour afficher des notifications, je vais utiliser l'outil **notify-send** (fourni par le paquet **libnotify-bin** sous Ubuntu 16.04). Si cet outil n'est pas installé (et donc la commande `which` échoue), alors il n'y aura pas de notification. C'est triste, mais cela n'est en rien fondamental. De même, si la variable (privée) `wacom_dont_notify` est égale à la chaîne `true`, alors on n'affichera aucune notification. Dans le cas contraire, une notification sera affichée avec comme titre le premier argument de la fonction précédé du contenu de la variable `WACOM_NAME` et comme corps de message le second argument (techniquement, on passe tous les arguments de la fonction comme arguments de `notify-send`). Bien que dans l'implémentation courante sur ma distribution la durée d'affichage de la notification ne soit pas prise en compte, je pars du principe qu'un jour ce sera le cas, donc je spécifie cette durée comme étant égale au contenu de la variable `WACOM_NOTIFICATION_DURATION` ; je choisis d'afficher l'icône spécifiée dans la variable `WACOM_ICON` ; et surtout, si deux notifications interviennent dans un faible intervalle de temps, la seconde notification remplace la première (par défaut, elles se succèdent, ce qui est assez pénible lorsqu'elles se sont accumulées), c'est le rôle du *hint* (option `-h "string:x-canonical-private-synchronous:${PROGNAME// /_}"`).

```
 91: # usage: _wacom notify <title suffix> <message>
 92: # display a notification if notify-send program is installed
 93: # and wacom_dont_notify is not set to "true"
 94: function _wacom_notify {
 95:   which notify-send > /dev/null || return
 96:   if test "${wacom_dont_notify}" != "true"; then
 97:     local t=${1}
 98:     shift
 99:     notify-send \
100:       -t "${WACOM_NOTIFICATION_DURATION}" \
101:       -i "${WACOM_ICON}" \
102:       -h "string:x-canonical-private-synchronous:${PROGNAME// /_}" \
103:       "${WACOM_NAME} ${t}" \
104:       "${@}"
105:   fi
106: }
```

Pour modifier la configuration de la tablette (**pad** + **pen**), je vais définir deux fonctions dédiées qui, au passage, afficheront une notification précisant la modification effectuée. Ces fonctions ne sont *in fine* qu'une surcouche de la commande **xsetwacom** (seule la fonction **_wacom_set_pen** est détaillée ici, la fonction **_wacom_set_pad** suit le même principe ; cf. fichier ligne 116 à 122).

```
108: # usage: _wacom_set_pen <args>
109: # wrapper to xsetwacom set for the stylus attributes
110: # that displays a notification.
111: function _wacom_set_pen {
112:   _wacom_notify "Pen" "${*}"
113:   xsetwacom set "${WACOM_PEN_STYLUS}" "${@}"
114: }
```

Chargement de la configuration

La configuration doit permettre notamment d'associer à un bouton de la tablette une action. Sur le modèle à ma disposition, j'ai 4 boutons dont les identifiants sont respectivement **1**, **2**, **3** et **8**. Ces identifiants correspondent également aux boutons gauche (1), milieu (2), droit (3) et précédent (8) (le cas échéant) d'une souris, comme l'illustre l'image de la figure 3 extraite de **https://regisestuncool.wordpress.com/2010/11/12/configuration-d'une-tablette-wacom-sur-debian/**.

Plutôt que de coder ces identifiants « en dur » dans le programme, autant passer par une variable de configuration **WACOM_PAD_BUTTON_IDS** de type vecteur, ce qui offre l'avantage de pouvoir adapter le script à un modèle disposant d'un nombre de boutons différents avec des identifiants quelconques.

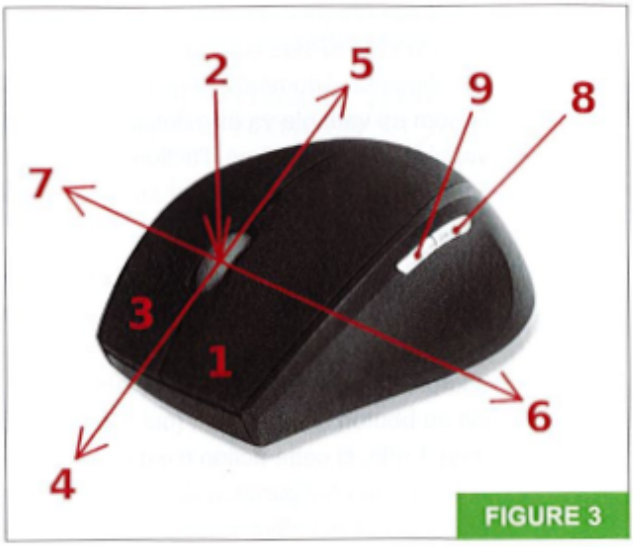


FIGURE 3

Identifiants associés aux boutons de la souris.

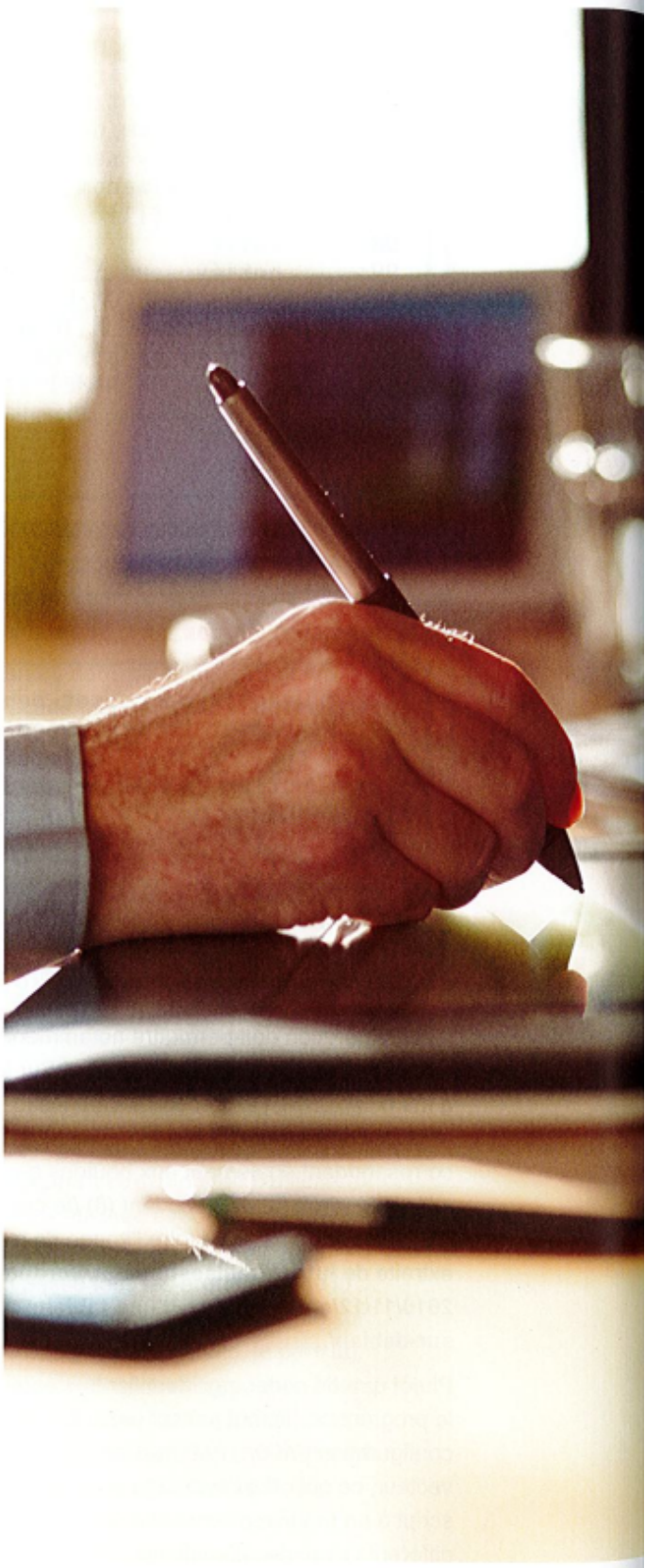
En jouant avec `xbindkeys`, vous aurez remarqué que l'on peut également gérer des combinaisons de touches. Ainsi il va être possible d'étendre le nombre d'actions possibles en gérant la présence éventuelle de modificateurs standard (**<Ctrl>**, **<Alt>** et **<Shift>**).

Ceci va permettre d'associer par exemple la combinaison **<Ctrl>** + **<Alt>** + **<Bouton 2>** à une fonction différente de la combinaison **<Shift>** + **<Bouton 2>**. Les fonctions associées seront alors définies dans le fichier de configuration sous la forme des variables `WACOM_PAD_BUTTON_ALT_CTRL_2` et `WACOM_PAD_BUTTON_SHIFT_2`. Pour simplifier l'écriture du script, les modificateurs seront toujours écrits dans l'ordre alphabétique dans les noms des variables. Si une variable n'est pas définie, cela signifie qu'aucune action n'est associée à la combinaison qu'elle décrit.

La fonction `_wacom_init_buttons` va commencer par désactiver les notifications (elles seront rétablies à la fin de la fonction).

Pour chaque identifiant de bouton (défini dans le tableau `WACOM_PAD_BUTTON_IDS`), nous allons récupérer le contenu de la variable correspondante si celle-ci est définie. Si l'action associée commence par le mot clé `key`, alors c'est l'action native de `xsetwacom` qui est associée à ce bouton et aucun modificateur ne sera pris en compte. Dans le cas contraire, alors pour chaque combinaison de modificateurs (le tableau `modifiers` inclut également la chaîne vide correspondant à l'absence de modificateur) énumérée dans l'ordre alphabétique, le nom de variable va être défini et stocké dans la variable `var`. De même, l'action à associer dans le fichier de configuration de `xbindkeys` va être stockée dans la variable `cfg`.

Afin que les actions associées aux boutons de la tablette n'impactent pas les boutons de souris, nous ajoutons arbitrairement `20` à l'identifiant du bouton. Ainsi, le bouton `2` de la tablette sera associé au bouton de souris `22` (qui *a priori* n'existe pas). Enfin, si cette action n'est pas trouvée dans le fichier de configuration de `xbindkeys`, alors un message est affiché pour suggérer les lignes à ajouter à ce fichier.



```
124: # usage: _wacom_init_buttons
125: # Initialize the pad buttons to (a priori) non existant codes.
126: function _wacom_init_buttons {
127:   wacom_dont_notify=true
128:   # Set Pad buttons to not_assigned if they are not assigned...
129:   # Any combination
130:   for button in "${WACOM_PAD_BUTTON_IDS[@]}"; do
131:     eval val="\${WACOM_PAD_BUTTON_${button}}"
132:     if [ "${val#key }" != "${val}" ]; then
133:       _wacom_set_pad "button" "${button}" "${val}"
134:       WACOM_HELP_MESSAGE+="\n- Button ${button} sends keys '${val#key }'"
135:     else
136:       local modifiers=("" "SHIFT" "ALT" "CTRL")
137:       local xbrc="${HOME}/.xbindkeysrc"
138:       for modifier1 in "${modifiers[@]}"; do
139:         for modifier2 in "${modifiers[@]}"; do
140:           [[ "${modifier1}" < "${modifier2}" ]] || [ -z "${modifier1}" ] ||
continue
141:           for modifier3 in "${modifiers[@]}"; do
142:             [[ "${modifier2}" < "${modifier3}" ]] || [ -z "${modifier2}" ] ||
continue
143:             local var="WACOM_PAD_BUTTON_"
144:             [ -z "${modifier1}" ] || var+="${modifier1}_"
145:             [ -z "${modifier2}" ] || var+="${modifier2}_"
146:             [ -z "${modifier3}" ] || var+="${modifier3}_"
147:             var+=${button}
148:             local cfg=""
149:             [ -z "${modifier1}" ] || cfg+="${modifier1,,} + "
150:             [ -z "${modifier2}" ] || cfg+="${modifier2,,} + "
151:             [ -z "${modifier3}" ] || cfg+="${modifier3,,} + "
152:             local button_id="$((20+button))"
153:             cfg+="b:${button_id}   (mouse)"
154:             eval "${var}=\${${var}:-not_assigned}"
155:             eval "val=\${${var}}"
156:             if [ "${val}" != "not_assigned" ]; then
157:               _wacom_set_pad "button" "${button}" "button +${button_id}"
158:               WACOM_HELP_MESSAGE+="\n- Button ${button} runs '${val}'"
159:               local line="${DIRNAME}/${PROGNAME} pad_button ${var#WACOM_PAD_
BUTTON_}"
160:               if [ ! -f "${xbrc}" ] || ! grep -q "${line}" "${xbrc}"; then
161:                 echo "# Please consider adding the following in your
'${xbrc}' file:"
162:                 cat <<EOF
163: "${line}"
164:      ${cfg/ctrl/control}
165: EOF
166:               fi
167:             fi
168:           done
169:         done
170:       done
171:     fi
172:   done
173:   wacom_dont_notify=false
174: }
```

Comme dans toute application de configuration qui se respecte, il faut permettre à l'utilisateur de remettre la configuration par défaut. C'est donc tout naturellement qu'est définie la fonction `_wacom_reset_values` (voir fichier de la ligne 176 à 185).

De même, comme dans tout système respectable, il faut un élément neutre, donc ici l'action qui ne fait rien (mais qui fait quand même quelque chose, puisqu'elle va envoyer une notification pour dire qu'il n'y a pas d'action, voir fichier de la ligne 188 à 197).

Il reste à définir les fonctions que l'on souhaite pouvoir associer aux boutons. Commençons par l'action permettant de basculer entre une mode de pointage relatif (le déplacement du stylet sur la tablette est appliqué à la position courante du curseur sur l'écran) et le mode absolu (la surface de la tablette représente l'écran et la position du stylet sur la tablette positionne le curseur sur l'écran). Pour cela, il suffit de récupérer le `Mode` du stylet, puis selon que le mode est `Absolute` ou non, il suffit de le basculer vers le mode alternatif.

```
199: # usage: wacom_toggle_mode
200: # Toggles the pointing mode between Absolute or Relative
201: function wacom_toggle_mode {
202:   v=$(xsetwacom get "${WACOM_PEN_STYLUS}" "Mode")
203:   test "${v}" = "Absolute" && v="Relative" || v="Absolute"
204:   _wacom_set_pen "Mode" "${v}"
205: }
```

L'autre action qui m'a semblé intéressante (notamment lorsque l'on dispose d'un écran pivotant à 90°) est précisément d'appliquer une rotation du système de coordonnées de la tablette. Il se trouve que celle-ci dispose de 4 orientations :

- l'orientation « naturelle » (paysage, boutons vers le haut) de la tablette est associée à la valeur `none` ;
- l'orientation d'un quart de tour vers la droite (portrait, boutons sur le côté droit) est associée à la valeur `cw` ;
- l'orientation d'un quart de tour vers la gauche (portrait, boutons sur le côté gauche) est associée à la valeur `ccw` ;
- l'orientation « à l'envers » (portrait, boutons vers le bas) est associée à la valeur `half`.

Il est donc aisé d'implémenter une fonction permettant de récupérer l'orientation actuelle de la tablette et de calculer l'orientation résultant d'un quart de tour vers la droite (dans le sens des aiguilles d'une montre pour celles et ceux qui ont encore une montre à aiguille...) :

```
207: # usage: wacom_rotate
208: # Rotates the pad by 90° clockwise
209: function wacom_rotate {
210:   v=$(xsetwacom get "${WACOM_PEN_STYLUS}" "Rotate")
211:   case "${v,,}" in
212:     "none") v="cw";;
213:     "cw") v="half";;
214:     "half") v="ccw";;
215:     *) v="none"
216:   esac
217:   _wacom_set_pen "Rotate" "${v}"
218: }
```

Parmi les options qui m'ont également semblé judicieuses, il est possible de modifier la sensibilité du stylet. Cette sensibilité correspond en réalité à une valeur seuil qu'il faudra augmenter pour diminuer la sensibilité et inversement, qu'il faudra abaisser pour augmenter la sensibilité. La valeur à ajouter/soustraire au seuil sera donnée par la variable **WACOM_STYLUS_THRESHOLD_STEP**. Les valeurs minimales et maximales pour ce seuil seront quant à elles paramétrées par les variables **WACOM_STYLUS_THRESHOLD_MIN** et **WACOM_STYLUS_THRESHOLD_MAX**.

```
220: # usage: wacom_decrease_sensitivity
221: # Decreases the stylus sensitivity
222: function wacom_decrease_sensitivity {
223:   v=$(xsetwacom get "${WACOM_PEN_STYLUS}" "Threshold")
224:   if [ "${v}" -le $((WACOM_STYLUS_THRESHOLD_MAX - WACOM_STYLUS_
THRESHOLD_STEP)) ]; then
225:     v=$((v+WACOM_STYLUS_THRESHOLD_STEP))
226:   else
227:     v=${WACOM_STYLUS_THRESHOLD_MAX}
228:   fi
229:   _wacom_set_pen "Threshold" "${v}"
230: }
```

La fonction **wacom_increase_sensitivity** n'est pas détaillée ici, mais est construite sur le même principe que **wacom_decrease_sensitivity** (voir fichier ligne 232 à 242).

La tablette dispose d'une batterie (sinon, il n'y aurait pas grand intérêt à pouvoir la connecter en *bluetooth*). Cette batterie se charge automatiquement lorsque la tablette est connectée en USB. Je suis parti du postulat (raisonnable dans mon cas, mais pas nécessairement juste dans le cas général) qu'il n'y aura qu'une tablette connectée à la fois. Le cas échéant lorsque la tablette est connectée en *bluetooth*, il devrait exister un répertoire **/sys/class/power_supply/**. Dans ce répertoire, on doit typiquement trouver un sous-répertoire **wacom_battery_0**.

Dans mes tests, je me suis retrouvé (une fois) à avoir un sous-répertoire **wacom_battery_0** vide et un répertoire **wacom_battery_1** contenant les informations de la batterie. Donc, sous l'hypothèse discutable qu'il y a plus de chances d'avoir une erreur de comptage de la batterie par le système que d'avoir deux batteries (pour la même tablette ou bien plusieurs tablettes Wacom connectées en *bluetooth* au système), je parcours tous les sous-répertoires correspondant au motif **wacom_battery_*** à la recherche d'un fichier intitulé **capacity**. Si un tel fichier est trouvé, alors son contenu semble indiquer le pourcentage de charge restant de la batterie. Si aucun fichier n'est trouvé, c'est soit qu'il n'y a pas de tablette connectée (mais alors il n'y a pas non plus d'action associée à un bouton à gérer), soit que la tablette est connectée en USB (donc en charge ou ne disposant pas de batterie).

```
244: # usage: wacom_power_status
245: # Displays a notification about the power status
246: function wacom_power_status {
247:   v=
248:   for f in /sys/class/power_supply/wacom_battery_*/capacity; do
249:     test -f "${f}" && v="-h int:value:$(cat "${f}")" || true
250:   done
251:   test -n "${v}" || v="Charging"
252:   _wacom_notify "Pad" ${v}
253: }
```

La fonction **wacom_pad_button** va simplifier l'appel des actions associées aux combinaisons de touches et de boutons en préfixant son argument par la chaîne **WACOM_PAD_BUTTON_** (qui est commune à toutes les combinaisons de touches que nous déclarons, comme nous l'avons vu lors de la définition de la fonction **_wacom_init_buttons**). Ainsi en évaluant le contenu de cette variable nouvellement créée, on récupère l'action qui est associée. En préfixant maintenant cette action par la chaîne **wacom_**, nous obtenons le nom de la fonction à appeler.

Ainsi, l'appel de la fonction **wacom_pad_button** avec la valeur **CTRL_8** va reconstruire le nom de variable **WACOM_PAD_BUTTON_CTRL_8**. En supposant que celui-ci contienne la chaîne **"power_status"**, alors la fonction **wacom_power_status** sera appelée, renvoyant une notification de l'indication de charge de la tablette.

```
255: # usage: wacom_pad_button <number>
256: # Generic wrapper for calling the action associated to
257: # the pad button <number>
258: function wacom_pad_button {
259:   eval "wacom_\${WACOM_PAD_BUTTON_${1}}"
260: }
```

Il reste une action (probablement la plus importante) à définir. L'action **wacom_help** qui affiche l'aide de ce script sur le terminal si le script est invoqué en ligne de commande, ainsi que sous forme de notification (voir fichier lignes 262 à 267).

Les fonctions suivantes n'ont pas lieu d'être associées à des boutons, mais sont plutôt à exploiter lors de la connexion/déconnexion de la tablette. Lors de la connexion, il faut initialiser la configuration, restaurer l'état de la tablette aux valeurs par défaut (du fichier de configuration), puis avertir l'utilisateur que la tablette est fonctionnelle. Lors de la déconnexion, il n'y a rien de spécial à faire, mais j'aime bien l'idée d'être notifié (notamment en cas de panne de batterie ou de perte de la connexion *bluetooth*). Ces deux fonctions (baptisées dans un élan d'inspiration **wacom_init** et **wacom_exit**) ne présentant pas d'intérêt technique ne sont pas présentées ici (voir fichier lignes 269 à 284).

À l'instar de nombreux scripts, lorsque celui-ci est invoqué, il va charger (s'ils existent) en premier lieu le fichier de configuration globale du système, puis le fichier de configuration de l'utilisateur. Pour rappel, les noms de ces fichiers sont donnés respectivement par les variables **WACOM_DEFAULT_CONFIG** et **WACOM_USER_CONFIG** déclarées au tout début du script.

```
287: ###############
288: # Main tasks #
289: ###############
290:
291:
292: # Load default configuration then user configuration if found.
293: for cfg in "${WACOM_DEFAULT_CONFIG}" "${WACOM_USER_CONFIG}"; do
294:   [ ! -f "${cfg}" ] || source "${cfg}"
295: done
```

Ensuite, les valeurs par défaut de l'icône à utiliser et du délai des notifications (pour le jour où cette dernière fonctionnalité sera implémentée par les auteurs de **libnotify**) sont utilisées si elles n'apparaissent pas dans un des fichiers de configuration.

```
297: # Set default notification icon if not already set.
298: WACOM_ICON=${WACOM_ICON:-icon-tablet}
299: # Set default notification duration
300: WACOM_NOTIFICATION_DURATION=${WACOM_NOTIFICATION_DURATION:-2}
```

Parce que parfois, il est agréable de visualiser ou de « journaliser » l'état d'un système, j'affiche le contenu des variables qui m'intéressent dans **/dev/null** (donc nulle part, mais si je le souhaite je commente la première ligne et dé-commente la seconde ou la troisième pour afficher respectivement dans un fichier de log ou sur la sortie standard).

```
302: cat <<EOF > /dev/null
303: #cat <<EOF >> /tmp/$PROGNAME.log
304: #cat <<EOF
305: WACOM_NAME=${WACOM_NAME}
...
322: WACOM_NOTIFICATION_DURATION=${WACOM_NOTIFICATION_DURATION}
323: args=$@
324: EOF
325:
326: #echo "[${PROGNAME}]$(date --iso-8601=seconds)" "Running with
args '${*}'." >> "/tmp/${PROGNAME}.log"
```

Enfin, si le script est invoqué sans argument, un message d'usage est affiché sur la sortie standard. Si le script est invoqué avec un argument, la fonction correspondante est appelée sans véritable contrôle. Ainsi je peux par exemple appeler le script avec l'argument **power_status** pour afficher le niveau de charge de la batterie, ou encore avec les arguments **pad_button** et **CTRL_8** pour appeler l'action associée à la combinaison de la touche **Control** avec le quatrième bouton de la tablette (d'après les fichiers de configuration).

```
327: if test -z "${1}"; then
328:   cat <<EOF
329: usage: $(basename "${0}") <command>
...
343: EOF
344: else
345:   _wacom_init_buttons
346:   eval wacom_${1} ${2}
347: fi
348: #echo "[${PROGNAME}]$(date --iso-8601=seconds)" "End of script
with args '${*}'." >> "/tmp/${PROGNAME}.log"
```

Pour que l'état de charge de la batterie s'affiche lorsque l'on appuie sur la touche **<Ctrl>** en même temps que le quatrième bouton de la tablette, il faut éditer le fichier **${HOME}/.xbindkeysrc** pour y ajouter :

```
"/usr/local/src/wacom_config/wacom_config pad_button CTRL_8"
     control + b:28   (mouse)
```

Détection automatique

Bon, là on se dit que c'est pas mal et qu'on a presque fini. Eh bien, pas tout à fait...

Il reste à automatiser le chargement de la configuration lorsque la tablette se connecte au système.

Je pourrai détailler longuement la démarche que j'ai effectuée pour y arriver, mais elle dénote essentiellement d'un manque d'habitude. Pour résumer, et comme vous le savez probablement, les périphériques sont gérés comme des fichiers spéciaux qui apparaissent dans le répertoire `/dev/`. Vous le savez probablement également, il y a un programme qui s'appelle `udev` qui gère la création/suppression de ces fichiers dynamiquement. Pour y parvenir, `udev` dispose d'un grand nombre de règles qui permettent entre autres d'exécuter des programmes spécifiques lorsque certaines conditions sont réunies. Donc pour répondre à mon problème d'initialisation automatique, il faut simplement intercepter un maillon de la chaîne des règles exécutées afin d'appeler la commande `wacom_config` avec l'argument `init`. Les règles à installer (typiquement dans le répertoire `/etc/udev/rules.d` sur ma machine) sont disponibles dans le fichier `wacom_config.udev_rules` dans le sous-répertoire `config` du projet (pour ma part, j'ai créé des liens symboliques plutôt que de copier les fichiers).

Il reste encore une ultime subtilité qui m'a bien fait tourner en bourrique, c'est que pour pouvoir afficher les notifications, le script a besoin d'une part que la variable d'environnement `DISPLAY` soit définie, ainsi que la variable `XAUTHORITY` contenant le chemin vers le fichier d'autorisation d'affichage sur l'écran. Pour contourner ce problème, il suffit d'ajouter en en-tête de votre fichier de configuration les valeurs appropriées (si quelqu'un sait comment faire plus propre, qu'il n'hésite pas à me le faire savoir).

```
export DISPLAY=:0.0
export XAUTHORITY=/home/doccy/.Xauthority
```

CONCLUSION

À travers cet exemple, j'espère avoir fourni aux possesseurs de tablette suffisamment d'éléments pour leur permettre de la paramétrer efficacement.

J'espère également avoir montré (à l'instar des nombreux auteurs qui publient dans ce magazine) la puissance et l'expressivité du Bash qui (à défaut d'être performant en termes de vitesse de calcul) offre la possibilité de maîtriser son système d'exploitation préféré. ▌

RÉFÉRENCE

[1] A. MANCHERON, « *Conservez l'historique de vos commandes pour chaque projet, le retour* », *GNU/Linux Magazine France, n° 241*, octobre 2020 : **https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-241/Conservez-l-historique-de-vos-commandes-pour-chaque-projet-le-retour**.