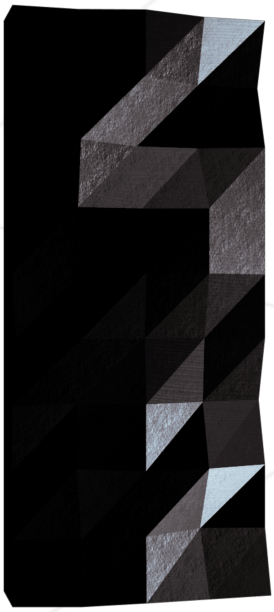


Projet à tendance UNIX

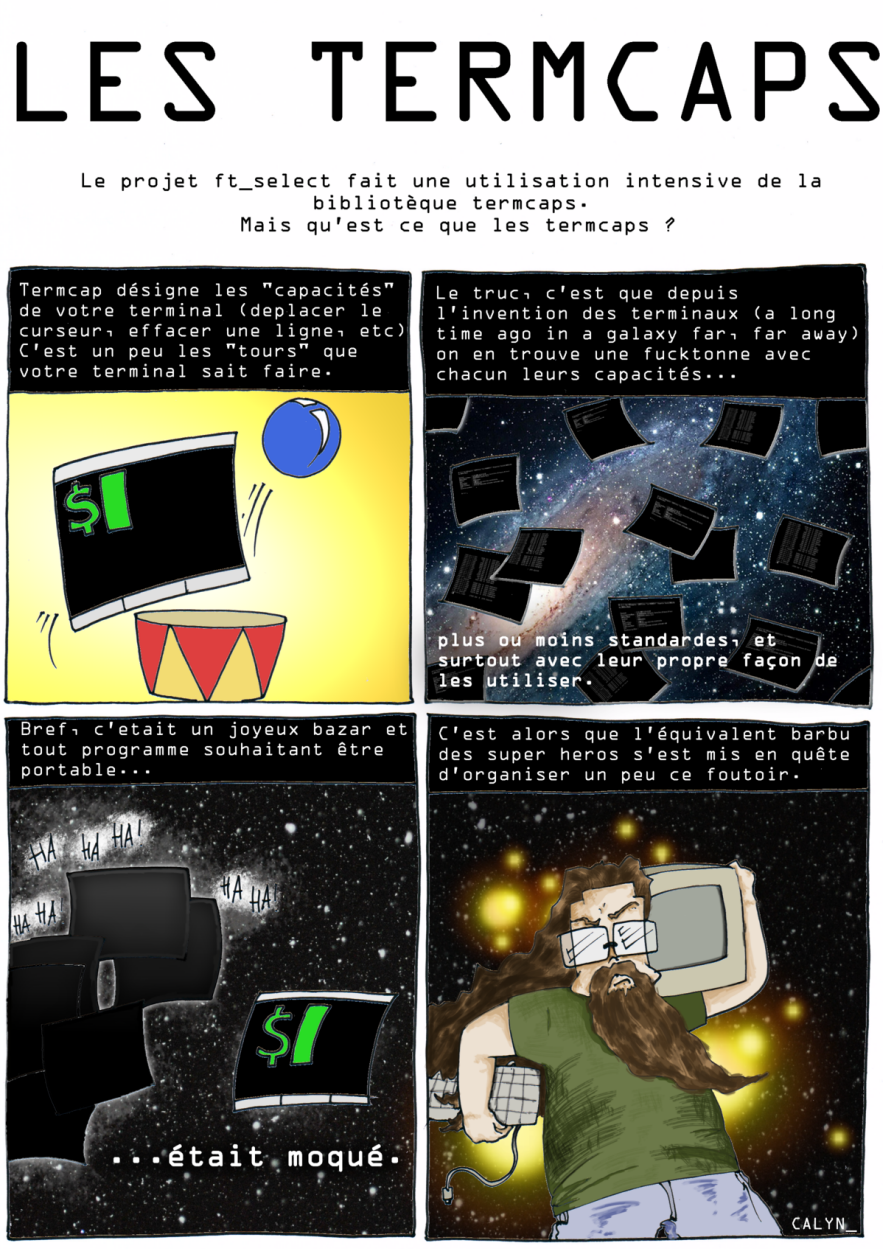
ft_select

Résumé: Ce projet a pour but de vous faire coder un petit programme en termcaps qui permettra de selectionner un ensemble de choix parmi une liste et de le retourner à votre shell.

Table des matières

Chapitre I

Préambule




Tous les terminaux ont été inventoriés avec la liste de leur capacités afin d'avoir une base de données fiable des capacités proposées par l'ensemble des terminaux. Et ils ne se sont pas arrêtés la !
Les barbus ont pris sur eux d'écrire une bibliothèque...
...permettant de connaître les capacités disponibles pour un terminal donné et de les utiliser de manière UNIFIÉE !
much awesome!
so work!
TERMCAP.H
C'est ainsi qu'est née la bibliothèque termcaps que vous allez utiliser dans plusieurs projets.
calyn_

Chapitre II

Introduction

Des programmes comme votre shell, `aptitude`, `top`, `tig`, `mcabber`, `dwarf fortress`, `zangband` ou `herrie` ont tous au moins deux points communs : ces programmes s'exécutent dans un terminal et proposent une interface utilisateur avancée malgré l'absence d'interface graphique fenêtrée à laquelle `Microsoft Windows` et `OSX` (dans son utilisation `Starbucks`) vous ont habitué.

Chapitre III

Objectifs

Réaliser une interface utilisateur pour un programme qui s'execute dans un terminal est possible et demande un travail de programmation qu'il est important de maitriser car dans son mode "brut", un terminal ne fait pas grand chose. Pour vous en convaincre, lancez la commande `cat` sans argument et appuyez sur des touches ou des combinaisons de touches de votre clavier...

Dans cette manipulation, tant que vous appuyez sur des touches alphanumériques, rien de particulier n'est à constater. Par contre, si vous appuyez sur les flèches, la touche `echap` ou les touches `f`*`n`* par exemple, des caractères arbitraires s'affichent... Refaites la manipulation dans votre shell maintenant. Vous êtes habitués à ce que la flèche gauche recule le curseur d'une colone. Pourquoi ces différences entre `cat` et votre shell ?

Votre terminal gère beaucoup de choses pour vous sans que vous y prêtiez la moindre attention, comme par exemple l'affichage des caractères au fur et à mesure que vous les tapez ou encore la bufferisation par ligne.

Si maintenant vous vous demandez comment vous pouvez prendre le contrôle de votre terminal, ce projet est fait pour vous. Vous y apprendrez à configurer votre termial via la structure "`struct termios`" et à utiliser ses capacités, les fameux "`termcaps`".

Chapitre IV

Consignes générales

- Interdiction d'utiliser des variables globales, hormis celles qui pourraient déja être définies pour vous. On tolerera une exception à cette règle pour gerer les signaux, même s'il est possible de faire autrement.
- Un fonction locale à un fichier `C` (qui n'est donc pas prototypée dans fichier `.h`) doit être définie `static` dans le respect de la Norme.
- Vous devez prêter attention à vos types et utiliser judicieusement les casts quand c'est nécéssaire, en particulier lorsqu'un type `void *` est impliqué. Dans l'absolu, évitez les casts implicites, quels que soient les types concernés.
- Vous pouvez utiliser autant de fonctions et de fichiers que vous le souhaitez.
- Vous devez rendre un `Makefile` qui compilera vos sources en C vers un éxécutable nommé `ft_select`.
- Votre `Makefile` doit au moins proposer les règles `$(NAME)`, `all`, `clean`, `fclean` et `re` dans l'ordre qui vous paraîtra le plus adapté.
- Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant votre login suivi d'un '\n' :

```
$>cat -e auteur
xlogin$
```

- Votre `Makefile` doit compiler votre travail avec les flags de compilation `-Wall`, `-Wextra` et `-Werror`.
- Si vous êtes malin et que vous utilisez votre biliothèque `libft` pour votre `ft_select`, vous devez en copier les sources et le `Makefile` associé dans un dossier nommé `libft` qui devra être à la racine de votre dépot de rendu.
- Votre executable doit linker avec la bibliothque `termcap`.
- Seul le contenu présent sur votre dépot sera évalué en soutenance.
- Votre projet doit être à la Norme. La Norminette sera utilisée pour vérifier la Norme lors de la soutenance. Une faute de norme invalide immédiatement l'évaluation.

- En aucun cas votre programme ne doit quitter de façon inattendue (Segmentation fault, bus error, double free, etc) ou diverger (boucle infinie). Votre projet serait alors considéré comme non fonctionnel.
- Un projet qui affiche de l'art abstrait (caractères étranges et/ou inadéquats) est considéré comme non fonctionnel.
- Dans le cas où votre programme est tué en cours d'éxécution, votre terminal doit bien évidement fonctionner normalement ensuite...
- Toute mémoire allouée sur le tas doit être libérée proprement.
- Vous ne devez jamais rendre de code que vous n'avez pas écrit vous-même. En cas de doute, vous serez invités à une séance de recode au bocal pour juger de votre bonne foi.
- La bibliothèque `termcap` est obligatoire pour ce projet.
- Ce projet réclame une bonne réflexion. Il est absolument vital que vous discutiez entre-vous et que vous partagiez vos informations !
- Avant que vous ne posiez la question, la bibliothèque `ncurse` est interdite pour ce projet. Sinon il n'y aurait pas de challenge...
- Les fonctions autorisées sont :
  - isatty, ttyname, ttyslot
  - ioctl
  - getenv
  - tcsetattr, tcgetattr
  - tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs
  - open, close, write
  - malloc, free
  - read, exit
  - signal
- Vous pouvez poser vos questions sur le Forum, Slack...

Chapitre V

Partie obligatoire

- Ecrire un programme "ft_select" qui prend en paramètre une série d'arguments. La liste d'arguments s'affiche.
- L'utilisateur peut alors se déplacer dans la liste des arguments à l'aide des flèches (la liste est circulaire).
- Un ou plusieurs choix peuvent être sélectionnés ou désélectionnés à l'aide de la touche `espace`. A chaque sélection effectuée, le curseur doit automatiquement se positionner sur l'élément suivant.
- Dès que l'utilisateur valide la sélection à l'aide de la touche `return`, la liste des choix doit être renvoyée au shell. Les choix renvoyés devront être separés par le caractère espace. Ceci permettra alors d'utiliser votre programme ft_select à l'intérieur d'un script shell (pour faire un "set", par exemple).
- On doit pouvoir écrire les commandes suivantes :

```
$> set reponse = `ft_select choix1 choix2 choix3 choix4`

$> more `ft_select *.c`

$> rm `ft_select ~/*`   NB: N'utilisez cette commande QUE si
                            vous etes SUR que votre ft_select
                            fonctionne. Nous degageons  toute
                            responsabilite  en  cas de bug de
                            votre ft_select...
```

- Vous devrez également gérer le redimensionnement de la fenêtre par l'utilisateur. La liste doit s'afficher sur plusieurs colonnes si la taille de la fenêtre ne comporte pas assez de lignes pour tout afficher en une seule colonne. Si l'utilisateur redimensionne la fenêtre en cours d'utilisation, l'ensemble des choix doit se repositionner automatiquement et les choix selectionnés doivent le rester. Le curseur de selection doit être positionné de manière raisonnable après un redimensionnement.
- Si la fenêtre est trop petite pour tout afficher (pas assez de lignes et/ou de colonnes, alors le programme doit réagir raisonnablement tant que la dimension de la fenêtre n'est pas suffisante. En aucun cas il ne doit quitter. Lorsque la fenêtre est de nouveau assez grande, le programme doit fonctionner de nouveau normalement.

- Si l'utilisateur appuie sur la touche `echap`, le programme ne doit rien renvoyer au shell et terminer normalement.
- Si l'utilisateur appuie sur la touche `delete` ou `backspace`, l'élément sur lequel pointe le curseur doit être effacé de la liste. S'il n'y a plus d'élément dans la liste, le comportement est le même que si l'utilisateur avait appuyé sur la touche `echap`.
- choix non selectionné : texte normal.
- choix selectionné : texte video inversé.
- position curseur : texte souligné.
- choix selectionné + position curseur : texte video inversé souligné.
- Quelque soit le moyen par lequel votre programme se termine, la configuration par defaut de votre terminal **DOIT** être restaurée. Oui, même après avoir reçu un signal (sauf signaux qu'on ne peut pas intercepter, mais ca veut dire que votre programme ne fonctionne pas de toute façon).
- On doit pouvoir interrompre votre programme avec un `ctrl+z` et le restaurer avec `fg` sans que cela influe sur son comportement.
- Si le programme est lancé avec un environnement vide, vous devez adopter un comportement raisonnable.

Chapitre VI

Partie bonus

Si vous avez réussi parfaitement la partie obligatoire, cette section propose quelques pistes pour aller plus loin. Les bonus seront comptabilisés si vous validez la partie obligatoire.

- Les colonnes défilent de gauche à droite en fonction de la position du curseur quand la fenêtre est trop petite.
- Après la terminaison du programme, ce qui doit l'être est effacé et le prompt et le curseur apparaissent sur la ligne suivant l'appel du programme. Lancez la commande `tig` pour voir ce que je veux dire. Pensez aux signaux !
- Une belle interface (c'est au correcteur de juger, pas à vous).
- Si les choix sont des fichiers, colorer selon l'extension (un peu comme `ls -G` sous `OSX`).
- Positionnement du curseur lorsque l'on tape une séquence de caractères qui correspond à un élément de la liste (recherche dynamique).

Chapitre VII

Rendu et peer-évaluation

Rendez-votre travail sur votre dépot GiT comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance.