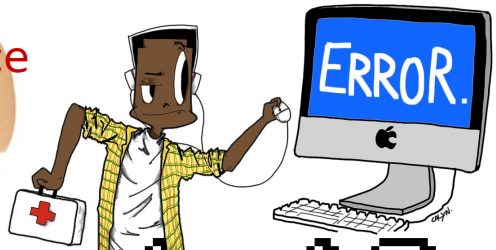
6 pages sur tout ce que vous devez savoir à 42!



42's Survival Toolkit

LE 42 Toolkit est maintenant un Wiki !

Les outils pour programmer depuis chez vous:
Empathy (Linux) pour vous connecterà votre XMPP (Jabber)
Stypi pour récupérer vos codes en ligne
Code::Blocks (avec l'include <unistd.h>) éventuellement Vim ou Emacs sur Linux
Et se guide bien sûr! :D

→ Terminal Commands

Note: les tirets – suivit d'une option sont à ajouter après la commande. Vous pouvez en combiner plusieurs exemples ls -la (lister les fichiers + détails et droits + fichiers cachés).

Aide et raccourcies

man manual permet d'obtenir de l'aide sur une commande ainsi que ses options ex: (**q** pour quitter)
flèche du haut taper une ancienne commande, **flèche du ba**s taper une commande plus récente après avoir fait flèche du haut.
tabulation autocompletion, utile pour éviter de taper un nom long de fichier ou dossier.
Ctrl + c kill le processus (souvent après boucle infinie)
ldapsearch 'uid = (login)' retrouve les informations d'un élève à partir du login
ifconfig donne des informations sur le réseau
sh fichier.sh permet d'executer un fichier .sh contenant des scripts Unix

Déplacements et affichage

pwd affiche le dossier dans lequel on est **ls** lister les fichiers **-*l*** (lettre L minuscule) détails et droits **-*a*** fichiers cachés
cd changer de dossier, par défaut le lien est relatif ex: cd Rendu **..** dossier parent cd seul retourne au home
cat affiche le contenu d'un fichier, à utiliser pour les textes ASCII (!= binaires) **-e** détails sur les espaces, etc

Manipulation des fichiers

mv déplacer un fichier ex: monFichier mondossier/ renommer un fichier ex: monFichier fichierRenommé
cp copier un fichier ex: cp monFichier fichierCopie **-R** copier un dossier ex: monDossier1 monDossier2
mkdir créér un fichier ex: créér un dossier ex: mkdir monDossier
rm supprime le fichier **-r** supprime le dossier + contenu **-f** force la suppression
ln créé un lien **-s** symbolique le plus utilisé comme sur Windows ex: ln fichier1 fichier2
chmod gestion des droits visible avec **ls -l** constitué de d rwx rwx rwx (dossier utilisateur groupe autres)
r(valeur 4): read **w**(2): write/supprimer **x**(1): executer il faut additionner les valeurs pour donner les 3 droits possible **chmod 777 monFichier.txt** donne tout les droits à tout le monde

→ Emacs (emacs est conseillé si vous souhaitez vous lancer rapidement)

Commandes

Ctrl + c -> Ctrl + h créé le header
Ctrl + x -> Ctrl + s sauvegarder mettre en fond
Ctrl + s puis espace affiche les espaces et tabulations très pratique pour respecter la norme **Ctrl + s** puis tab voir les tabulations
Ctrl + - undo (trait d'union et non underscore)
Esc + x et taper linum voir les numéros de lignes

fg dans le shell pour revenir sur emacs (fg pour foreground l'inverse de background)

→ Vim (lire “Vi Aille ème” pas vime)

Vim en tapant le nom depuis le shell le programme se lancera ou **Vim monFichier.c** pour créer en même temps

Note: il y a 3 modes, le mode intéractif permet d'utiliser des raccourcies comme le copier/coller mais c'est le point central pour accéder aux deux autres modes qui sont “Intéractifs” pour taper du texte et “Commande” pour enregistrer, quitter, etc.

-Mode intéractif avec le raccourcie **Echap**
-Puis mode insertion avec raccourcie **I**, le mode insertion se lance si la touche n'est pas un raccourcie
- Ou le raccourcie : pour entrer en mode commandes

Text by Bekhoucha Danyl (dbekhouc) - linko_3D@hotmail.com - Linko_3D on Twitter
Graphics by Calyn (cpickens) - calynpickens@gmail.com - www.42comic.tumblr.com

Commandes
:w enregistrer
:q quitter
:x enregistrer puis quitter
fn + F1 ajoute le header
dd couper
p coller (sous la ligne en cours)

-> Les types de fichiers

Les extensions de fichier
.sh contient des scripts Unix
.c fichier source en langage C
.cpp fichier source en C++
.h fichier contenant les prototypes (voir plus bas leur role)
.o fichier binaire compilé à partir de plusieurs .c, permet de gagner du temps à la compilation et de généraer des .a
.a c'est un librairie contenant souvent plusieurs fonctions. Voir comment compiler une librairie avec GCC.

Intérêt et fonctionnement des prototypes .h
//note auteur: je bloque sur les .h je ne peux pas rediger cette section, une aide est bienvenues

→ Git (se pronnonce guite et pas jite)
Vous devez créer un dossier Rendu sur votre home contenant des sous dossier nommé jxx (xx correspond au numéro du jour).

Créer un dépot git ou récupérer un dépot
git clone l'adresse nomDuDossier //chaque jour à une adresse différente, vérifiez que ça correspondent bien Permet aussi de récupérer un dépot pour la correction

Mettre les fichiers et dossiers dans le dépot créé
ga nomFichier
ga nomDossier
*/*ga revient à écrire git add */*
gcmsg "message de la mise à jour"
*/*gcmsg revient à écrire git commit -m */*
git push origin master //première fois puis sans origin master

Voir contenu d'un dépôt et supprimer un fichier/dossier
git ls-files permet de voir le contenu du dépot **git rm nomFichier**
git rm -r nomDossier (**-r** pour suppression récursive, ajouter **-f** pour forcer si ça je fonctionne pas)
puis confirmer:
git commit -m "suppression de nomFichier"s

→ Compilation avec GCC
Compilation basique
gcc nomFichier1.c nomFichier2.c -o nomExecutable ou *gcc -o nomExecutable nomFichier1.c nomFichier2.c*

Exécution
./nomFichier

-o créé l'executable de sortie (mot écrit après l'option) //attention à ne pas confondre -o ne créé pas de .o !
-c permet de générer un .o (mot écrit après l'option)
-L permet de spécifier où est la libraire
. dans le répertoire courant, vous pouvez donc utiliser -L.
-lstr inclu lla librairie libSTR à la compilation
-lft inclu lla librairie libft à la compilation
ranlib myLibrary.a index les fonctions de la librairie et optimise le temps de compilation

Les flags
-W affiche des avertissements sur des choses pouvant être améliorer
-Wall affiche d'avantages d'avertissements
-Werror chaque avertissement devient une erreur (pris en compte par la moulinette)
-Wextra affiche des warning supplémentaire

Création d'une librairie
gcc -c ft_putchar.c // creation .o
ar rc ft_putchar.a ft_putchar.o // création de la librairie .a à partir du .o
gcc test.c ft_putchar.a -o test //création de l'executable test à partir d'un fichier .c et de la librairie créé (pas d'include nécessaire)

Les Makefiles
Les MakeFiles automatisent la compilation, plutôt que de réécrire gcc suivi des 50 fichiers à compiler, de flags et autres options une simple commande permettra de compiler tout le contenu du MakeFile.

Text by Bekhoucha Danyl (dbekhouc) - linko_3D@hotmail.com - Linko_3D on Twitter
Graphics by Calyn (cpickens) - calynpickens@gmail.com - www.42comic.tumblr.com

Pour compiler le Makefile taper **make**. Le makefile ne compile que si les tabulations sont respectées ! Ajoutez un backslash \ pour indiquer une nouvelle ligne.

```
.PHONY: clean, all, re, fclean
NAME = nomDeVotreLibrairie.a
CFLAGS = -Wall -Wextra -Werror //vos flags
SOURCES = main.c ft_putchar.c \
                        ft_putstr ft_strlen \
                        ft_strstr.c ft_strdup.c

OBJS = $(SOURCES:.c=.o)
all: $(NAME)
$(NAME):
        gcc $(CFLAGS) -c $(SOURCES) -I. ar rc $(NAME) $(OBJS) ranlib $(NAME)
clean:
        rm -f $(OBJS) nomDeVotreLibrairie.h.gch
fclean: clean
        rm -f $(NAME)
re: fclean all
```

→ C

Formatage et norme

/* commentaire multi ligne */
#include <systemLibrary.h> importer une librairie système fourni avec le langage (biblothèque) **#include "myLibrary"** importer sa librairie
{ } contenu d'une fonction
( ) paramètres

NORME: les commentaire en double slash // sont interdit. En rouge sont représenté les tabulations (Ctrl + s puis tab pour vérifier les tabulations sous Emacs). Il faut une tabulation après le type d'une fonction et variable et sauter une ligne pour les variables.

```
#include <librarie.h>

type  fonction (paramètres)
{
        type  variable1; /*comment in english*/
        type  variable 2;
/*Toujours sauter une ligne après la déclaration d'une variable sinon 0 !*/
        variable 1 = 5;
        variable 2 = 0;
        if (variable1 > variable2)
        {
                contenu;
        }
        return (0);
}
```

Libraries

include <unistd.h> (activé par défaut dans le terminal permet de manipuler des commandes comme write). On utilise < > pour les headers système, qui fonctionnent avec le C.
stdlib.h et **stdio.h** permet de commandes basiques tel que le printf pour afficher du texte, le scanf pour récupérer une saisie
include "maLibrairie.h" inclu sa propre libraire, mettre des guillemets. Pour la créer voir la compilation avec GCC.

Un define avec une Macro

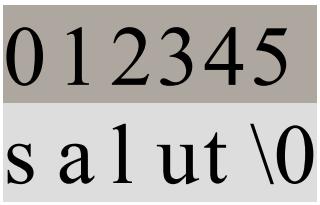
```
#define ECRIRE_QUARANTE_DEUX() printf("42"); /*c'est une macro car il y a des parenthèses et une commande qui s'execute */
int     main () {
ECRIRE_QUARANTE_DEUX();
}
```

Les types de variables

NORME: après chaque type mettre une tabulation y compris dans les noms des fonctions ! Les octets vous permettront de mieux comprendre l'allocation dynamique (voir plus bas).
void (4 octets) à utiliser souvent dans une fonction quand celle-ci ne renvoit pas de résultat ou utilse des pointeurs
int et **long** (4 octets) pour un nombre entié
double et **float** (8 octets)pour un nombre décimal, float mieux géré par la carte graphique.
char pour une lettre, ne gère pas les accents et respecte la casse. La valeur doit être en apostrophes ex: 'a

Lire et créer une chaine de charactères

Une chaine de charactère est un tableau permettant de former mots et phrases. Le tableau commence par la case 0 et il faut toujours compter une case supplémentaire pour la case du tableau qui se termine par \0. char chaine [5];




```
char  chaine[] = "salut";
int i = 0;

while (chaine[i] != '\0')
 {
        printf("%c", chaine[i]);
        i++;
}
```

Structures conditionnels

if (valeur comparaison valeur et/ou valeur comparaison valeur) { contenu; }
== égal à
!= différent de
>= supérieur ou égal à
&& et
|| ou

Les listes chaînées

Définition, déclaration et affectation d'un pointeur (un pointeur dans une même fonction à peu d'interets)

Une pointeur ne fonctionne que dans les accolades dans lequel il est créé. Pour l'utiliser en dehors il faut soit créer une variable qui copiera sa valeur puis utilisera un return pour copier la nouvelle valeur dans la variable d'origine, soit pointer sur son adresse pour la modifier directement (donc fonction de type void possible).

```
int     variable;
int     *pointeurSurVariable;

variable = 10; pointeurSurVariable = &variable; //& pour adresse, on pointe vers l'adresse pour récupérer la valeur

Fonction qui pointe sur une variable d'une autre fonction void      fonctionPointeur                (int
*pointeurSurVariable) //pointeur qui reçoit l'adresse, fonction type void suffit {
*pointeurSurVariable += 2; //toujours utiliser l'étoile
}
int     main () {
int variable;

variable = 5;
fonctionPointeur (&variable);
}
```

Le déréférencement

La fonction write

write(sortie, “contenu”, bits);
sortie 1 équivalent à mettre stdout pour standard output, correspond à l'écran entre guillemet le contenu ensuite le nombre de bits ou plutot le nombre de lettres affichées
write(1, “abc”, 3);

La fonction printf

%d affiche nombre entié ou valeur ASCII de la lettre
%f affiche le contenu d'un décimal (float et double)
%c affiche caractère
%ld affiche un long (nombre plus grand que int)

Lire et écrire dans des fichiers

NE FONCTIONNE PAS

```
#include <stdlib.h>
#include <stdio.h>

FILE* fichier = NULL;

fichier = fopen(“fichierTexte.txt”, “w”);  /*r: écrire, w: lecture , r+: lecture + écriture, w+: pareil mais avec supression avant, a: ajoute du texte à la fin du fichier*/
```

La récursivité

Text by Bekhoucha Danyl (dbekhouc) - linko_3D@hotmail.com - Linko_3D on Twitter
Graphics by Calyn (cpickens) - calynpickens@gmail.com - www.42comic.tumblr.com

Une fonction récursive est une fonction qui se rappel elle même. Elle peut se rappeler même si toutes les instructions n'ont pas été lu, ainsi arrivé à la fin de l'appel en boucle elle liera la suite. Voici l'exemple donné durant la Piscine:

```
int ft (int i)
{/*i n'est pas initialisé à 0 car sinon ça valeur revient à 0 à chaque rappel se qui produit une boucle infini, on reçoit 0 en paramètre*/
	if (i < 5)
	{
		i++;
		write (1, “D”, 1);
		fn(i);/*lis le programme jusqu'à cette partie, une fois arrivé au bout lis seulement la suite, la valeur actuel de i est renvoyé */
		write (1, “F”, 1);
		return (0);
	}
	else
	{
		return (0);
	}
}

int main ()
{
	fn (0);
	return (0);
}
```

L'allocation dynamique de mémoire

Fonction **<stdlib.h>** obligatoire pour utiliser malloc et free
malloc permet d'allouer de la mémoire
free libère ensuite de la mémoire quand la variable n'est plus utilisé

L'allocation dynamique est utile lorsque l'on veut travailler avec des plages mémoires tel que des tableaux ou autre dont on ne connait pas la taille à la compilation (au moement où on écrit le code). La taille du tableau est déterminée par une variable mais il est interdit d'écrire int tableau [taille]; car les compilateurs ne comprennent pas toujours se code (seulement la version gcc c99). Il est extrèmement rare d'utiliser une simple variable (ou matrice) avec l'allocation dynamique. Le tableau peut s'ecrire comme ceci sur une ligne (ne respecte pas la norme):
int* memory= malloc(sizeof(int)*10) /* le 10 peut être remplacé par une variable qui peut prendre une valeur entré par l'utilisateur */
En sachant que int vaut 4 octects on peut aussi écrire directement 4*10 ou même 40 le résultat:
int* memory = malloc(40);

L'allocation dynamique selon la norme et vérification si ça marche dans un if:

```
int	nbDeCases;
int	i;
int* monTableauDynamique;

nbDeCases = 0;
i = 0;
monTableauDynamique = NULL;

scanf("%d", &nbDeCases);

if (nbDeCases > 0)
{
	monTableauDynamique = malloc(nbDeCases * sizeof(int));
if (monTableauDynamique == NULL) /*si le tableau est toujours égale à NULL c'est qu'il n'y a pas de cases..*/
{
	exit(0); /*..donc le programme se quitte, l'allocation à échoué ou la valeur nbDeCases inférieur ou égal à 0*/
}

...

while (i < nbDeCases) {
}
	printf("Valeur case numéro %d: ", i + 1);  /*Valeur case 0 sera affiché au départ, pour comencer à 1 écrire i + 1 */
	scanf("%d", &monTableauDynamique[i]);  /*pour lire le contenu du tableau voir “Lire et créer une chaine de charactères” plus haut */
	i++;
}
	free(monTableauDynamique);
}
```

Text by Bekhoucha Danyl (dbekhouc) - linko_3D@hotmail.com - Linko_3D on Twitter
Graphics by Calyn (cpickens) - calynpickens@gmail.com - www.42comic.tumblr.com

```
return 0;
```

Passer un argument

```
int	main (int argc, char **argv) /*argc nb d'arguments, argv[0] tableau de chaine de charactères*/
{
	if (argc == 2) /*si argc contien deux arguments, le premier étant le nom de l'executable à la case argv[0],
			l'argument est en argv[1]*/
	{
		ft_putstr (argv[1]);
	}
	else
	{
		ft_putstr(“aucun argument entré”);
	}
	ft_putchar(“\n”);
	return (0);
}
```

Il faut écrire l'argument après le nom de l'executable exemple: ./test argument1 Les espaces permettent de passer à l'argument suivant sauf si vous utilisez les guillemts: ./test “argu 1” “argu 2”

Listes chaînées

...

Besoin d'aide pour rédiger cette partie, login: dbekhouc

→ C++

Vector et Deques

Classes et constructeurs/destructeurs

```
#include <iostream>
using namespace std;
class Personnage {
	public;
	in health ()
	{
	}
};

int main()
{
	return 0;
}
```

Polymorphisme

...