



Cours de C

Commandes Linux de base & introduction au C

Sébastien Paumier



Objectif du cours de C

- apprendre le langage C
- analyser un problème pour écrire le programme qui le résout, de façon élégante et modulaire
- coder proprement
- savoir respecter un cahier des charges lors de projets
- être au niveau pour le cours de système en IR2



Planning

- Slot 0: stage de rentrée de 3 jours
- Slot 1:
 - 5 cours, 5 TD sur machine
 - 1 examen
- Slot 2:
 - 5 cours, 5 TD sur machine
 - 1 examen



Objectif du stage de rentrée

- mise à niveau sur:
 - les commandes système de base
 - l'écriture et la compilation propre d'un programme C simple
- 3 journées intensives:
 - 2h de cours, 2h de TD, 4h de TP (1 par poste)
- dernière journée:
 - TP noté pour évaluation



Linux

- système d'exploitation libre
 - open source
 - gratuit
- multi-utilisateurs
- support natif du réseau
- interface graphique indépendante du système



Différences principales avec Windows

- Pas de C:\... D:\...
 - racine unique /
 - périphériques vus comme des sous-répertoires. Ex: `/mnt/floppy` pour A:\
- Casse des noms de fichiers
 - `Hello.txt` \neq `hello.txt` \neq `HELLO.TXT`
- Droits d'accès sur les fichiers



Compte utilisateur

- identité = login+mot de passe
- espace disque privé
 - `/home/ens/paumier` ou `~paumier`
 - on y est par défaut quand on se loge
- appartenance à un ou des groupe(s)



Lister un répertoire

- commande **ls**

```
cogmu/  plp/  projet.html
```

- option **-l** pour les détails

```
total 16
```

```
drwxr-xr-x  3 paumier igm 4096 sep 18  2006 cogmu/
```

```
drwxr-xr-x  2 paumier igm 4096 jui 26  2006 plp/
```

```
-rw-r--r--  1 paumier igm 4764 nov 20  2006 projet.html
```

- jokers: **ls *.html file?.bin**



Fichiers spéciaux et cachés

- . = répertoire courant
- .. = répertoire parent
- .**xxxxxx** = fichier caché
 - utiles pour des fichiers de configuration

Fichiers visibles avec **ls -a**

```
./ ../ .bashrc cogmu/ plp/ projet.html
```



Répertoires

- `pwd` affiche le répertoire courant
- on change de répertoire avec `cd`
 - relatif:
 - `cd toto`
 - `cd ..`
 - `cd ../W3/prog`
 - absolu:
 - `cd /tmp/test`
 - retour à la racine du compte:
 - `cd` ou `cd ~` ou `cd ~paumier`



Répertoires

- créer un répertoire

```
mkdir toto
```

- détruire un répertoire

```
rmdir toto si toto est vide
```

```
rm -ri toto sinon
```

```
rm -rf toto pour éviter de confirmer pour  
chaque fichier
```

- **rm** marche aussi pour les fichiers



Copie et renommage

- copie de fichier:

```
cp toto ../titi
```

- copie de répertoire:

```
cp -r toto/ tutu
```

- renommage ou déplacement:

```
mv toto tutu
```

```
mv toto ../../test/toto.txt
```




Droits d'accès

	fichier F	répertoire R
r	on peut lire F	on peut voir le contenu de R
w	on peut écrire dans F ou le détruire	on peut créer des fichiers ou des répertoires dans R
x	on peut exécuter F si F est un exécutable	on peut rentrer dans R avec cd

Pour pouvoir créer un fichier **titi/tutu**, on doit avoir les droits **w** et **x** sur **titi**



Changer les droits

- `chmod u=wx toto` : fixe à `wx` les droits de l'utilisateur (`u`ser) pour le fichier `toto`, et laisse les autres inchangés:
 - avant: `-rw-r--r--`
 - après: `--wxr--r--`
- `g` modifie les droits du `g`roupe
- `o` modifie ceux des autres (`o`thers)
- `a` modifie tous les droits (`a`ll)



Changer les droits

- + ajoute les droits indiqués

```
chmod a+x toto
```

- avant: `-rw-r--r--`

- après: `-rwxr-xr-x`

- - enlève les droits indiqués

```
chmod g-r,o-r toto
```

- avant: `-rw-r--r--`

- après: `-rw-----`



Lancer une tâche de fond

- **anjuta** dans un terminal lance un éditeur, mais la console n'a plus la main
- pour garder la main:

```
$>anjuta&
```

```
[1] 2715
```

– lance un autre processus



Voir les processus en cours

- **ps** : affiche les processus du terminal courant

```
PID    TTY          TIME CMD
1962   pts5        00:00:00 bash
3517   pts5        00:00:00 emacs
3518   pts5        00:00:00 ps
```

- **ps -u paumier** : affiche tous les processus de l'utilisateur **paumier**



Tuer un programme

- `ctrl+c` : tue le programme
- `ctrl+z` : stoppe le programme
 - on pourra y revenir avec `fg`
- `kill -3 PID` : tue le programme
- `kill -9 PID` : tue le programme sauvagement (dernier recours)
- `xkill` : permet de sélectionner une fenêtre



Afficher un fichier

- `cat toto`: affiche le contenu de `toto`
- `more toto`: idem, avec défilement si `toto` ne tient pas sur l'écran
- `less toto`: mieux que `more` (on peut remonter)
- `head -n 17 toto`: affiche les 17 premières lignes de `toto`
- `tail -n 17 toto` : affiche les 17 dernières lignes de `toto`



Rediriger les sorties

- `ls > toto` : redirige la sortie standard de `ls` dans le fichier `toto`
- `ls >> toto` : idem, mais ajoute les données à la fin de `toto`
- `ls 2> toto` : redirige la sortie d'erreur dans le fichier `toto`
- `ls 2>> toto` : idem, avec ajout en fin



Rediriger les sorties

```
>$ ls -l
total 8
d----- 2 paumier igm 4096 jun 22 17:13 titi/
-rw-r--r-- 1 paumier igm  11 jun 22 17:13 tutu
```

```
>$ ls * > list 2> err
```

```
>$ cat list
tutu
```

```
>$ cat err
ls: titi: Permission denied
```



Fichier spécial /dev/null

- trou noir
- si on veut ne voir que les sorties d'erreur de `ls`:

```
ls > /dev/null
```



Chercher dans des fichiers

- `grep xxx *.txt` : affiche toutes les lignes des fichiers `*.txt` qui contiennent `xxx`

```
>$ grep passwd ind*  
index:chpasswd (8) - update password file in batch  
index2:passwd [ssl-passwd] (1) - compute password hashes  
ind:rpc.yppassdd [rpc] (8) - NIS password update daemon
```



Obtenir de l'aide

- `man xxx` : donne la page de manuel de la commande `xxx`
- `man -k xxx` : donne les pages de manuel dont le titre contient `xxx`
- `xxx --help` : convention pour toutes les commandes du système



Introduction au C ou comment compiler proprement son premier programme



Le langage C

- C, parce qu'après le langage B :)
- 1972, Laboratoires Bell, Dennis Ritchie & Kenneth Thompson
- 1989: norme ANSI C
- langage impératif
- à la fois haut et bas niveau
- programmation système (noyau Linux)



Quelques références

- B.W.Kernighan D.M.Ritchie, Le langage C
- http://fr.wikibooks.org/wiki/Programmation_C

+ tous les livres qui prennent la poussière à la bibliothèque...

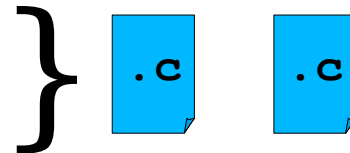
- rendu: charte des projets

http://igm.univ-mlv.fr/~paumier/charte_des_projets.html

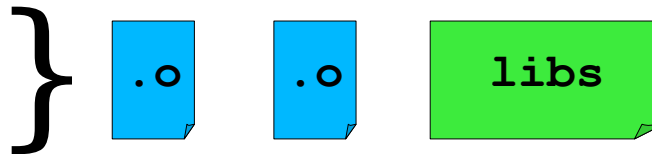


Qu'est-ce qu'un programme?

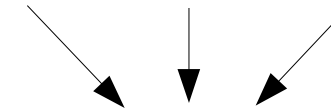
Instructions dans un langage lisible par un humain



Fichiers objets

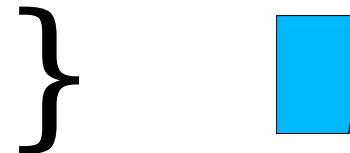


Compilation



Édition de liens

Programme exécutable





Les fichiers sources

- fichiers textes contenant les instructions
- extension spéciale .c
- syntaxe très précise à respecter
- séparateurs = espace + tab + retour à la ligne
- programmation modulaire: un fichier par thème:
 - bibliothèque mathématique, graphique, ...



Le premier programme

```
/* HelloWorld.c
 * Le classique des classiques */

#include <stdio.h>

int main(int argc, char* argv[]) {
    printf("Hello world!\n");
    return 0;
}
```

Des commentaires

Inclusion de la bibliothèque
contenant la fonction **printf**



Le premier programme

main est toujours la fonction principale d'un programme C

```
/* HelloWorld.c
 * Le classique des classiques */

#include <stdio.h>

int main(int argc, char* argv[]) {
    printf("Hello world!\n");
    return 0;
}
```

Retourne un entier

argc : nombre de paramètres
argv : les paramètres



Le premier programme

Début de bloc d'instructions

```
/* HelloWorld.c
 * Le classique des classiques */
#include <stdio.h>

int main(int argc, char* argv[]) {
    printf("Hello world!\n");
    return 0;
}
```

La fonction se termine et renvoie 0

Fin de bloc d'instructions

Fonction affichant à l'écran une chaîne de caractères



Le premier programme

- compilation:

```
$>gcc HelloWorld.c -c
```

```
$>gcc HelloWorld.o -o HelloWorld
```

ou, en une étape:

```
$>gcc HelloWorld.c -o HelloWorld
```

- exécution:

```
$>./HelloWorld
```

```
Hello world!
```



Un autre exemple

```
#include <stdio.h>
#define PI 3.14

float a,b;

float sum(float x,float y) {
    return x+y;
}

int main(int argc,char* argv[]) {
    float c,d;
    a=PI;
    b=1;
    c=sum(a,b);
    printf("%f + %f = %f\n",a,b,c);
    printf("d=%f\n",d);
    return 0;
}
```

Constante

Variables globales

Une fonction

Le C permet l'utilisation
de variables non
initialisées !!! 



Le même en deux fichiers

f1.c

```
#include <stdio.h>
#define PI 3.14

float a,b;

extern float sum(float x, float y);

int main(int argc, char* argv[]) {
    float c,d;
    a=PI;
    b=1;
    c=sum(a,b);
    printf("%f + %f = %f\n", a,b,c);
    printf("d=%f\n", d);
    return 0;
}
```

f2.c

```
float sum(float x, float y) {
    return x+y;
}
```

→ Déclaration d'une fonction définie dans un autre fichier



Le même en trois fichiers

f1.c

```
#include <stdio.h>
#include "f2.h"
#define PI 3.14

float a,b;

int main(int argc, char* argv[]) {
    float c,d;
    a=PI;
    b=1;
    c=sum(a,b);
    printf("%f + %f = %f\n",a,b,c);
    printf("d=%f\n",d);
    return 0;
}
```

f2.c

```
float sum(float x, float y) {
    return x+y;
}
```

f2.h

```
float sum(float x, float y);
```

Inclusion du fichier *header* qui contient la définition de **sum**



Sources et headers

- `.c`: code
 - on n'inclut pas les `.c` ! ⚡
- `.h`:
 - définitions des fonctions visibles de l'extérieur du `.c`
 - définitions de constantes
 - description de la bibliothèque
 - licence du code



Fichiers .h

- ne doivent pas être compilés avec gcc ⚡
- problème de doubles définitions
⇒ protection avec des macros

f2.h

```
#ifndef f2_H
#define f2_H

float sum(float x, float y);

#endif
```



Un beau .h

```
/*
 * Copyright (C) 2007 Sébastien Paumier <paumier@univ-mlv>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * ...
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
 */

#ifndef ANGLES_H_
#define ANGLES_H_

/**
 * This library provides implementations of trigonometric functions.
 */

#define PI 3.14

float cosinus(float angle);
float sinus(float angle);

#endif
```

Inutile de commenter
les choses évidentes



Options de compilation

- **-ansi** : norme qui garantie la portabilité
- **-Wall** : affiche des avertissements

warnings.c

```
main(int argc, char* argv[]) {  
    int a,b;  
    printf("a=%d\n",a);  
}
```

(se méfier des traductions
très mauvaises)

```
$>gcc warnings.c
```

```
$>gcc warnings.c -Wall -ansi
```

```
Warnings.c:1: AVERTISSEMENT: le défaut choisi du type retourné est « int »
```

```
Warnings.c: Dans la fonction « main »:
```

```
Warnings.c:3: AVERTISSEMENT: déclaration implicite de la fonction « printf »
```

```
Warnings.c:2: AVERTISSEMENT: variable inutilisée « b »
```

```
Warnings.c:4: AVERTISSEMENT: contrôle a atteint la fin non void de la  
fonction
```



Messages d'erreurs

- Possibilité d'erreurs en cascade
 - toujours commencer par la première erreur

`cascade.c`

```
int sum(int a,int b) {  
    return a+b;  
}
```

```
$>gcc -Wall -ansi cascade.c  
cascade.c:1: erreur d'analyse syntaxique avant « b »  
cascade.c: Dans la fonction « sum »:  
cascade.c:2: « a » non déclaré (première utilisation dans cette fonction)  
cascade.c:2: (Chaque identificateur non déclaré est rapporté une seule fois  
cascade.c:2: pour chaque fonction dans laquelle il apparaît.)  
cascade.c:2: « b » non déclaré (première utilisation dans cette fonction)
```



Les fichiers Makefile

- instructions pour faciliter la compilation
- utilisables avec **make**
- nom = **Makefile** ou **makefile**

cible: dépendances

[TAB] action1

[TAB] action2

...

```
all: trigo
```

```
trigo: main.o angles.o
```

```
gcc main.o angles.o -o trigo
```

```
@echo Compilation finie.
```

```
clean:
```

```
rm -f trigo main.o angles.o
```

```
%.o: %.c
```

```
gcc -c -Wall -ansi $<
```



Les fichiers Makefile

Cible par défaut

Cible usuelle pour effacer les fichiers produits par le Makefile

Règle pour créer les fichiers objets

```
all: trigo

trigo: main.o angles.o
    gcc main.o angles.o -o trigo
    @echo Compilation finie.

clean:
    rm -f trigo main.o angles.o

%.o: %.c
    gcc -c -Wall -ansi $<
```



Les fichiers Makefile

- détection des modifications basée sur les dates de fichiers
- **make** ne reconstruit que le nécessaire

```
$> make
gcc -c -Wall -ansi main.c
gcc -c -Wall -ansi angles.c
gcc main.o angles.o -o trigo
Compilation finie.
$> make
make: Rien à faire pour « all ».
```




Propreté du code

1) indenté proprement (utiliser **indent** si nécessaire)


```
int foo( void)

{
    int i,
    j=0;

int u;
    for (i=0; i < 10 ;i ++ ) { j=j+i;
u=u+j;
    }
    return u;
}
```



```
int foo(void) {
    int i,j=0,u;
    for (i=0;i<10;i++) {
        j=j+i;
        u=u+j;
    }
    return u;
}
```





Propreté du code

2) commenté judicieusement

on ne commente pas la syntaxe! ⚡

```
/* Affiche un tableau passé en paramètre */  
void affiche_tableau(int tableau[],int taille) {  
int i; /* déclaration d'une variable i */  
for (i=0;i<taille;i++) { /* on parcourt le tableau */  
    printf("%d ",tableau[i]); /* en affichant chaque case */  
}  
}
```



```
/* Alloue et retourne un tableau de 'taille' entiers, ou NULL en  
 * cas d'erreur. Chaque case d'indice i du tableau est initialisée  
 * avec f(i), où f est une fonction passée en argument de f_alloc. */  
int* f_alloc(int taille,int (*f)(int)) {  
int* t=(int*)malloc(taille*sizeof(int));  
if (t==NULL) return NULL;  
int i;  
for (i=0;i<taille;i++)  
    t[i]=f(i);  
return t;  
}
```





Propreté du code

3) des identificateurs judicieux

```
a f(a b,c d) {  
    if (b==NULL) return NULL;  
    if (b->e==d) return b;  
    return f(b->s,d);  
}
```



```
ColorList find_color(ColorList l,Color c) {  
    if (l==NULL) return NULL;  
    if (l->color==c) return l;  
    return find_color(l->next,c);  
}
```





Propreté du code

4) du code lisible par tous (mondialisation oblige...)

```
listacores achacor(listacores l, cor c) {  
    if (l==NULL) return NULL;  
    if (l->cor==c) return l;  
    return achacor(l->proxima,d);  
}
```



```
ColorList find_color(ColorList l, Color c) {  
    if (l==NULL) return NULL;  
    if (l->color==c) return l;  
    return find_color(l->next,c);  
}
```





Propreté du code

5) éviter le style gourou

- bon pour l'ego, peut-être
- mauvais pour le code, sûrement

```
int* pstrand(int* p) {  
    while (!*p++);  
    return (*p++&=0xE8)?p:0;  
}
```



```
i, j; main(n) { for (; i < 1E4; j || printf("%d ", n, ++i)) for (j = n++; n % j == 0; j--); }
```



Propreté du code

6) cohérence de présentation avec:

- le reste du code
- celui du binôme
- celui de l'équipe

```
/* This function looks for the given
   color in the given color list. */
ColorList
findColor(ColorList l, Color c) {
if (l==NULL) {
    return NULL;
}
if (l->color==c) {
    return l;
}
return findColor(l->next,c);
}
```

```
/**
 * This function looks for the given
 * color in the given color list.
 */
ColorList find_color(ColorList l,Color c) {
    if (l==NULL) {return NULL;}
    if (l->color == c) {return l;}
    return find_color(l->next,c);
}
```

```
/* This function looks for the given color */
/* in the given color list. */
ColorList find_Color(ColorList l , Color c)
{
if (l == NULL) return NULL ;
if (l->color == c) return l ;
return find_Color(l->next , c) ;
}
```