Système

Aomar Maddi – L3 Info Rennes 1

2015 – 2016, S1

1 Shell UNIX

1.1 Commandes

- `cd <dir>` se déplacer
- `ls <nom>` afficher
- `more <nom>`
- `less <nom>` idem avec scrolling
- `ln (-s) <nom1> <nom2>` créer un lien (symbolique)
- `mkdir <nom>` créer un répertoire
- `rm -rf <nom>` supprimer
- `cp -r <nom1> <nom2>` copier nom1 vers nom2
- `cat <nom1> ... <nomn>` concaténer et afficher les fichiers
- `echo <message>`
- `chmod (ugo )(+-=)(rwx) <nom>` changer les droits d'un fichier
- `basename <nom>` nom sans chemin
- `dirname <nom>` chemin sans le nom
- `pwd` afficher le répertoire courant
- `exit` quitter le programme
- `read <var1> ... <varn>` lire des variables (0 si OK)
- Redirections : `>` sortie standard, `>>` concatène, `>` entrée standard, `|` redirection ES-SS
- Délimiteurs : `'` pas d'interprétation, `"` idem sauf `$` et `‘`, `‘` interprète le contenu
- Caractères spéciaux : `*` joker, `?` un caractère, `\` échappement

1.2 grep et find

`grep (-r) <regex> <nom>` sort toutes les lignes de `nom` (fichier ou répertoire) contenant l'expression régulière donnée (où `.` joker, `*` étoile de Kleene, `[x-y]` classe de caractères).

`find <dir> (-name <nom> -usr <user> -size <n(okM)> -type (dfl) ...)` affiche les fichers contenus dans le répertoire spécifié en filtrant selon les critères donnés ;
`-not -a -o \( \)` permettent de combiner les filtres, `-exec cmd {} \;` exécute la commande donnée sur chaque fichier trouvé.

1.3 Programmation

- `chmod +x script` (le script commence par `#!/bin/sh`) puis `./script arg1 ... argn`
- Identificateurs : `$#` nombre de paramètres, `$*` ou `$@` tous, `$0..n` nom du programme puis paramètres, `$HOME`, `$PATH`, ...
- Commentaires : `# commentaire`
- Variables : `var=(valeur,var2,‘commande‘)`, peuvent être initialisées par `read`

— Tests : `! -a -o` logique, `-d -f <nom>` existence, `-s <nom>` non vide, `-r -w -x <nom>` droits, `-z -n <chaine>` longueur nulle/non nulle, `<ch1> (!)= <ch2>`, `<n1> -eq -neq -gt -lt -ge -le <n2>` arithmétique

— Conditions :

```
if [ commandes ] # si dernière commande renvoie 0
then             # alors
    commandes
else             # sinon
    commandes
fi
```

— Itérations :

```
while [ commandes ] # tant que der = 0
do
    commandes
done
for var
in <liste> # explicite $foo $bar $so ou liste telle que $*
do
    commandes
done
```

2 Programmation C

2.1 Mémo syntaxique

```c
/* Préprocesseur */
#include <stdio.h> // chargement d'une lib c
#include "libperso.h"
#define CONSTANTE "valeur" // constante de préprocesseur

/* Variables et pointeurs */
int i, j = 0; // types : (unsigned ou signed) char short int long float double
char machin = (char) truc; // cast de types
int* pi = &i; // pi est un pointeur d'int sur i
// le type pointeur neutre est void*, le pointeur nul est NULL
*pi += 1; // déréférencement

/* Tableaux */
float stats[64]; // taille statique
int p[] = {2, 3, 5, 7, 11, 13}; // initialisation
p == &p[0]; // tableaux et pointeurs c'est pareil
p[i] == *(p+i); // et oui
// mais on ne peut pas copier ou affecter n'importe comment (strcpy...)

/* Fonctions */
type fonction(type par1, type par2, ...) {
    instructions;
    return resultat; // de type correct, rien si void
}
// localité de bon sens
// paramètres passés par valeur (pour référence on utilise les pointeurs)
char* exemple(int n, char* str); // prototype
// permet d'utiliser une fonction non déclarée
// àrassembler dans un fichier .h
```

```
30
31 /* Point d'entrée */
32 int main(int argc, char* argv[]) { // nb d'arguments commande, leur liste
33     ...
34     return 0; // valeur de retour si pas d'erreurs
35 }
36
37 /* Chaînes et E/S */
38 #include <string.h> // une chaîne est un char[] finissant par \0
39 int strln(char* s); // |s|
40 char* strcpy(char *s1, char *s2); // s1 <- s2
41 int strcmp(char *s1, char *s2); // 0 si s1 == s2
42 #include <stdio.h> // fonctions d'entrée / sortie
43 int getchar(); // lit le prochain caractère sur stdin
44 void putchar(char c); // écrit un caractère sur stdout
45 int printf(char* format, e1, ..., en); // écrit selon le formatage
46 int scanf(char* format, a1, ..., an); // lit les variables selon le formatage
47 // format : %d %c %s %lf ...
48
49 /* Types énumérés et structurés */
50 enum couleur {coeur, pique, carreau, trefle};
51 enum couleur macouleur = pique;
52 struct carte {couleur c, int val}; // référence récursive possible
53 struct carte macarte = {pique, 9};
54 typedef struct {couleur c, int val} carte; // type alias (marche aussi avec enum)
55 macarte.val = 1;
56 pcarte->c = trefle; // pour un pointeur vers une structure
57 size_t sizeof(type t); // donne la taille en octets d'un type
58
59 /* Gestion de la mémoire */
60 void* malloc(size_t s); // retourne un pointeur neutre (cast) vers un espace mémoire
61 void free(void* p); // libère la mémoire
62
63 /* Gestion de fichiers */
64 #include <stdio.h>
65 FILE* fopen(char* name, char* mode); // mode r, w, a (append)
66 int fclose(FILE* f); // 0 si fermé, EOF sinon
67 int putc(int c, FILE* file); // EOF en cas d'erreur
68 int getc(FILE* file);
69 // écrit (lit) nb blocs de taille size de buf vers file
70 int fwrite(void* buf, int size, int nb, FILE* file);
71 int fread(void* buf, int size, int nb, FILE* file);
72 // idem printf et scanf
73 int fprintf(FILE* file, char* format, e1, ..., en);
74 int fscanf(FILE* file, char* format, a1, ..., an);
```

2.2 Makefile

Si un fichier a été modifié plus récemment que la cible, les commandes s'exécutent :

```
target: file1 file2 ...
    commands
```

On rajoute des cibles génériques (`clean`, `all`...) par confort d'utilisation ; on peut aussi définir des variables Shell et tout ce qu'il faut pour avoir un Makefile universel.